

DOE FILE COPY

MCR-80-1377A

SOFTWARE/FIRMWARE DESIGN SPECIFICATION

for

10 MWe SOLAR THERMAL

CENTRAL RECEIVER PILOT PLANT

January 1982

Prepared Under Contract No. DE-AC03-80SF10539

by

Martin Marietta Corporation

for

Department of Energy

10 MW_e COLLECTOR SUBSYSTEM
CENTRAL RECEIVER SYSTEM
SOFTWARE/FIRMWARE DESIGN SPECIFICATION

January 1982

Prepared by: *Timothy D. Ladewig*
Timothy D. Ladewig
HAC Lead Engineer

CONCURRENCE: *D. Powell*
Daniel A. Powell
HFC/HC Lead Engineer

Approved by: *W. G. Hart*
Walter G. Hart
Collector Subsystem Controls Manager

Approved by: *P. R. Brown*
Paul R. Brown
Program Manager
Collector Systems

~~AMS~~ → File 8.1

MARTIN MARIETTA AEROSPACE

DENVER AEROSPACE
POST OFFICE BOX 179
DENVER, COLORADO 80201
TELEPHONE (303) 977-3000

February 8, 1982

Refer to: SOL-82-150

To: U. S. Department of Energy
San Francisco Operations Office
Solar Ten Megawatt Project Office
5301 Bolsa Avenue, MS 14-1
Huntington Beach, California 92647

Attn: Mr. Richard Schweinberg

Subj: Contract No. DE-AC03-80SF10539, Transmittal of Data

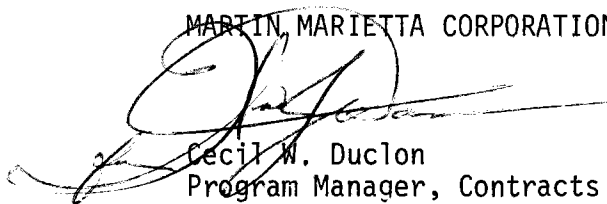
Ref: (a) Task B.3 of Exhibit II, SOW of
Contract No. DE-AC03-80SF10539

Encl: (1) Software/Firmware Design Specification
MCR-82-1377A, dated January 1982

1. Enclosure (1) documentation is transmitted in accordance with Reference (a) Contract Requirements.
2. Questions regarding this documentation should be directed to Mr. Melvin Frohardt, Telephone No. (303) 977-0527, at the letterhead address, Mail No. C4000.

Very truly yours,

MARTIN MARIETTA CORPORATION



Cecil W. Duclon
Program Manager, Contracts
Solar Energy Systems
Martin Marietta Aerospace
Denver Division

RECEIVED
FEB 10 1982
STMPO

CWD/dc
COPY TO:
SNLL/Mr. W. Rorke (2 Copies Encl.)
STMPO/Mr. M. Slaminski (1 Copy Encl.)

Enclosure

TABLE OF CONTENTS

<u>Section</u>	<u>Title</u>	<u>Page</u>
1.0	INTRODUCTION	1
1.1	Scope	1
1.2	Problem Statement	1
1.3	Design Approach	2
2.0	APPLICABLE DOCUMENTS	3
3.0	DESIGN REQUIREMENTS	4
3.1	System Functional Description	4
3.1.1	Hardware System Description	4
3.1.1.1	HAC Computer System	4
3.1.1.2	HFC Microprocessor	9
3.1.1.3	HC Microprocessor	9
3.1.1.4	Chromatics Intelligent Terminals	10
3.1.2	Software/Firmware System Description	10
3.1.2.1	HAC Software Description	10
3.1.2.1.1	Terminology	10
3.1.2.1.2	Operating Modes	11
3.1.2.1.3	Functional Overview	12
3.1.2.2	HFC Firmware Description	16
3.1.2.3	HC Firmware Description	16-A
3.1.2.4	Chromatics Intelligent Terminal (GDC) Software Description	17
3.2	HAC Software Design	17
3.2.1	Man-Machine Interface Module - (MANMIF)	18
3.2.1.1	Purpose	18
3.2.1.2	Requirements	18
3.2.1.2.1	Design Requirements	18
3.2.1.2.2	Derived Requirements	19
3.2.1.3	Design Approach	20
3.2.1.3.1	Functional Allocations	28
3.2.1.3.2	Resource Budgets	30
3.2.1.4	Design Description	30
3.2.1.4.1	Module Structure	30
3.2.1.4.1.1	Submodule I - MMI	30

TABLE OF CONTENTS (con't.)

<u>Section</u>	<u>Title</u>	<u>Page</u>
3.2.1.4.1.2	Submodule II - MMIDEQ	51
3.2.1.4.1.3	Submodule III - MMIWRD	53
3.2.1.4.1.4	Submodule IV - MMICLK	53
3.2.1.4.1.5	Submodule V - MMICMD	59
3.2.1.4.1.6	Submodule VI - MMINUM	60
3.2.1.4.1.7	Submodule VII - MMIHXD	62
3.2.1.4.1.8	Submodule VIII - MMIMAP	64
3.2.1.4.1.9	Submodule IX - MMINON	66
3.2.1.4.1.10	Submodule X - MMISTR	74
3.2.1.4.1.11	Submodule XI - MMIST2	76
3.2.1.4.1.12	Submodule XII - MMIST3	78
3.2.1.4.1.13	Submodule XIII - MMIREF	78
3.2.1.4.1.14	Submodule XIV - MMIRSP	80
3.2.1.4.1.15	Submodule XV - MMICRT	82
3.2.1.4.1.16	Submodule XVI - MMIERR	84
3.2.1.4.1.17	Submodule XVII - MMITST	88
3.2.1.4.1.18	Submodule XVIII - MMIAID	89
3.2.1.4.1.19	Submodule XIX - MMIBCS	91
3.2.1.4.1.20	Submodule XX - DSK	94
3.2.1.4.1.21	Submodule XXI - DSKAIM	96
3.2.1.4.1.22	Submodule XXII - DSKINK	98
3.2.1.4.1.23	Submodule XXIII - DSKTRF	101
3.2.1.4.1.24	Submodule XXIV - DSKBIA	102
3.2.1.4.1.25	Submodule XXV - DSKERR	104
3.2.1.4.1.26	Submodule XXVI - DSKBAK	106
3.2.1.4.1.27	Submodule XXVII - DSKUPD	108
3.2.1.4.1.28	Submodule XXVIII - CFO	112
3.2.1.4.1.29	Submodule XXIX - CFOPOS	115
3.2.1.4.1.30	Submodule XXX - CFOGET	115
3.2.1.4.1.31	Submodule XXXI - CFOMSG	117
3.2.1.5	Interface Description	122
3.2.1.6	Test Requirements	122
3.2.2	Command Processor Module (CMDPRC)	124
3.2.2.1	Purpose	124

TABLE OF CONTENTS (con't.)

<u>Section</u>	<u>Title</u>	<u>Page</u>
3.2.2.2	Requirements	124
3.2.2.2.1	Design Requirements	124
3.2.2.2.2	Derived Requirements	124
3.2.2.3	Design Approach	125
3.2.2.3.1	Functional Allocations	127
3.2.2.3.1.1	Task Activations	128
3.2.2.3.1.2	Task Satisfaction of Derived Requirements	131
3.2.2.3.2	Resource Budgets	132
3.2.2.4	Design Description	133
3.2.2.4.1	Module Structure	133
3.2.2.4.1.1	Submodule I - CMD	135
3.2.2.4.1.2	Submodule II - CMDSCK	141
3.2.2.4.1.3	Submodule III - CMDFAD	148
3.2.2.4.1.4	Submodule IV - CMDBAD	157
3.2.2.4.1.5	Submodule V - CMDINO	163
3.2.2.4.1.6	Submodule VI - CMDARA	168
3.2.2.4.1.7	Submodule VII - CMDSAL	172
3.2.2.4.1.8	Submodule VIII - CMDTRK	180
3.2.2.4.1.9	Submodule IX - CMDDSK	186
3.2.2.4.1.10	Submodule X - CMDPOS	188
3.2.2.4.1.11	Submodule XI - CMDSBY	192
3.2.2.4.1.12	Submodule XII - CMDS ^D DN	194
3.2.2.4.1.13	Submodule XIII - CMDBCS	197
3.2.2.4.1.14	Submodule XIV - CMDSUP	199
3.2.2.4.1.15	Submodule XV - GET	202
3.2.2.4.1.16	Submodule XVI - GETINI	204
3.2.2.4.1.17	Submodule XVII - GETAIM	208
3.2.2.4.1.18	Submodule XVIII - GETAL1	214
3.2.2.4.1.19	Submodule XIX - GETAL2	217
3.2.2.4.1.20	Submodule XX - GETWSH	219
3.2.2.4.1.21	Submodule XXI - GETSTO	222
3.2.2.4.1.22	Submodule XXII - GETSAR	227
3.2.2.4.1.23	Submodule XXIII - SEQ	231
3.2.2.4.1.24	Submodule XXIV - SEQGAT	238
3.2.2.4.1.25	Submodule XXV - SEQBPT	243

TABLE OF CONTENTS (con't.)

<u>Section</u>	<u>Title</u>	<u>Page</u>
3.2.2.4.1.26	Submodule XXVI - SEQCCK	246
3.2.2.4.1.27	Submodule XXVII - SEQCOR	250
3.2.2.4.1.28	Submodule XXVIII - SEQADD	253
3.2.2.4.1.29	Submodule XXIX - SEQLK	260
3.2.2.4.1.30	Submodule XXX - SEQDEL	262
3.2.2.4.1.31	Submodule XXXI - BHI	266
3.2.2.4.1.32	Submodule XXXII - BHISH5	269
3.2.2.4.1.33	Submodule XXXIII - BHC	269
3.2.2.5	Interface Description	273
3.2.2.6	Test Requirements	273
3.2.3	Sun Vector Module - SUNVEC	278
3.2.3.1	Purpose	278
3.2.3.2	Requirements	278
3.2.3.2.1	Design Requirements	278
3.2.3.2.2	Derived Requirements	278
3.2.3.3	Design Approach	278
3.2.3.3.1	Functional Allocations	278
3.2.3.3.2	Resource Budgets	280
3.2.3.4	Design Description	280
3.2.3.4.1	Module Structure	280
3.2.3.4.1.1	Submodule I - SUN (Main Routine)	281
3.2.3.4.1.2	Submodule II - SUNPOS	282
3.2.3.4.1.3	Submodule III - SUNDJ	286
3.2.3.4.1.4	Submodule IV - SUNCON	286
3.2.3.4.1.5	Submodule V - SUNPOL	290
3.2.3.4.1.6	Submodule VI - SUNDMA	292
3.2.3.5	Interface Description	294
3.2.3.6	Test Requirements	294
3.2.4	Field Communications Processor Module - FLDCOM	296
3.2.4.1	Purpose	296
3.2.4.2	Requirements	296
3.2.4.2.1	Design Requirements	296
3.2.4.2.2	Derived Requirements	297
3.2.4.3	Design Approach	297

TABLE OF CONTENTS (con't.)

<u>Section</u>	<u>Title</u>	<u>Page</u>
3.2.4.3.1	Functional Allocations	301
3.2.4.3.2	Resource Budgets	302
3.2.4.4	Design Description	307
3.2.4.4.1	Module Structure	307
3.2.4.4.1.1	Submodule I - FCP	307
3.2.4.4.1.2	Submodule II - FCPOUT	310
3.2.4.4.1.3	Submodule III - FCPIN	314
3.2.4.4.1.4	Submodule IV - FCPCHO	317
3.2.4.4.1.5	Submodule V - FCPCHI	320
3.2.4.4.1.6	Submodule VI - FCPUPD	322
3.2.4.4.1.7	Submodule VII - FCPSWH	323
3.2.4.4.1.8	Submodule VIII - FCPFLP	325
3.2.4.5	Interface Description	326
3.2.4.6	Test Requirements	329
3.2.5	Alarm Processor Module - ALARMS	330
3.2.5.1	Purpose	330
3.2.5.2	Requirements	330
3.2.5.2.1	Design Requirements	330
3.2.5.2.2	Derived Requirements	330
3.2.5.3	Design Approach	330
3.2.5.3.1	Functional Allocations	330
3.2.5.3.2	Resource Budgets	335
3.2.5.4	Design Description	336
3.2.5.4.1	Module Structure	336
3.2.5.4.1.1	Submodule I - ALM	338
3.2.5.4.1.2	Submodule II - ALMDTC	340
3.2.5.4.1.3	Submodule III - ALMFLD	342
3.2.5.4.1.4	Submodule IV - ALMLNE	344
3.2.5.4.1.5	Submodule V - ALMHFC	346
3.2.5.4.1.6	Submodule VI - ALMHC	348
3.2.5.4.1.7	Submodule VII - ALMGET	350
3.2.5.4.1.8	Submodule VIII - ALMCLT	356
3.2.5.4.1.9	Submodule IX - ALMRPT	357
3.2.5.4.1.10	Submodule X - ALMBLK	359

TABLE OF CONTENTS (con't.)

<u>Section</u>	<u>Title</u>	<u>Page</u>
3.2.5.4.1.11	Submodule XI - ALMQUE	363
3.2.5.4.1.12	Submodule XII - ALO	366
3.2.5.4.1.13	Submodule XIII - ALOBLD	367
3.2.5.4.1.14	Submodule XIV - ALODQU	372
3.2.5.4.1.15	Submodule XV - ALOCVT	375
3.2.6	Status Display Module - STATUS	378
3.2.6.1	Purpose	378
3.2.6.2	Requirements	378
3.2.6.2.1	Design Requirements	378
3.2.6.2.2	Derived REquirements	378
3.2.6.3	Design Approach	380
3.2.6.3.1	Functional Allocations	380
3.2.6.3.2	Resource Budgets	384
3.2.6.4	Design Description	396
3.2.6.4.1	Module Structure	396
3.2.6.4.1.1	Submodule I - STS	396
3.2.6.4.1.2	Submodule II - STSGET	397
3.2.6.4.1.3	Submodule III - STA	400
3.2.6.4.1.4	Submodule IV - STAFLD	402
3.2.6.4.1.5	Submodule V - STAIND	402
3.2.6.4.1.6	Submodule VI - STAMOD	405
3.2.6.4.1.7	Submodule VII - STARNG	407
3.2.6.4.1.8	Submodule VIII - STAGET	409
3.2.6.4.1.9	Submodule IX - STAMCH	411
3.2.6.4.1.10	Submodule X - STACRT	413
3.2.6.5	Interface Description	415
3.2.6.6	Test Requirements	415
3.2.7	Data Base Initialization Module - DBINIT	417
3.2.7.1	Purpose	417
3.2.7.2	Requirements	417
3.2.7.2.1	Design Requirements	417
3.2.7.2.2	Derived Requirements	417
3.2.7.3	Design Approach	418

TABLE OF CONTENTS (con't.)

<u>Section</u>	<u>Title</u>	<u>Page</u>
3.2.7.3.1	Functional Allocations	425
3.2.7.3.2	Resource Budgets	437
3.2.7.4	Design Description	438
3.2.7.4.1	Module Structure	438
3.2.7.4.1.1	Submodule I - Task DIN	438
3.2.7.4.1.2	Submodule II - DINLOC	445
3.2.7.4.1.3	Submodule III - DINTRF	460
3.2.7.4.1.4	Submodule IV - DININC	461
3.2.7.4.1.5	Submodule V - DINSRT	466
3.2.7.4.1.6	Submodule VI - DINTTL	467
3.2.7.4.1.7	Submodule VII - DINANG	470
3.2.7.4.1.8	Submodule VIII - DINPAC	474
3.2.7.4.1.9	Submodule IX - DINFLG	478
3.2.7.4.1.10	Submodule X - DINLAC	481
3.2.7.4.1.11	Submodule XI - DINCKI	485
3.2.7.4.1.12	Submodule XII - DINCKR	487
3.2.7.4.1.13	Submodule XIII - DINAEA	489
3.2.7.4.1.14	Submodule XIV - DINGRF	491
3.2.7.4.1.15	Submodule XV - DINUPC	494
3.2.7.4.1.16	Submodule XVI - DBI (Task)	496
3.2.7.4.1.17	Submodule XVII - DBIFAC	497
3.2.7.4.1.18	Submodule XVIII - DBICPU	499
3.2.7.4.1.19	Submodule XIX - DBICRT	501
3.2.7.4.1.20	Submodule XX - DBIPCS	504
3.2.7.4.1.21	Submodule XXI - DBITRP	508
3.2.7.4.1.22	Submodule XXII - DBIDTA	509
3.2.7.4.1.23	Submodule XXIII - DBIBCK	512
3.2.7.4.1.24	Submodule XXIV - DBICOR	514
3.2.7.4.1.25	Submodule XXV - DBIAIM	518
3.2.7.4.1.26	Submodule XXVI - DBIBCK	522
3.2.7.4.1.27	Submodule XXVII - DBIERR	526
3.2.7.4.1.28	Submodule XXVIII - CLK (Task)	531
3.2.7.4.1.29	Submodule XXIX - CLKEST	532

TABLE OF CONTENTS (con't.)

<u>Section</u>	<u>Title</u>	<u>Page</u>
3.2.7.4.1.30	Submodule XXX - CLKONL	534
3.2.7.4.1.31	Submodule XXXI - CLKACT	538
3.2.7.5	Interface Description	542
3.2.7.6	Test Requirements	542
3.2.8	Operating System Modifications Module - MAXIVM	544
3.2.8.1	Buffer Management Function	544
3.2.8.1.1	Purpose	544
3.2.8.1.2	Requirements	544
3.2.8.1.2.1	Design Requirements	544
3.2.8.1.2.2	Derived Requirements	545
3.2.8.1.3	Deisgn Approach	545
3.2.8.1.3.1	Functional Allocations	545
3.2.8.1.3.2	Resource Budgets	545
3.2.8.1.4	Design Description	545
3.2.8.1.4.1	Module Structure	545
3.2.8.1.4.1.1	Submodule I - QINIT	545
3.2.8.1.4.1.2	Submodule II - LEASE	546
3.2.8.1.4.1.3	Submodule III - FREE	549
3.2.8.1.4.1.4	Submodule IV - ENQUE	549
3.2.8.1.4.1.5	Submodule V - DEQUE	552
3.2.8.1.4.1.6	Submodule VI - QRST	555
3.2.8.1.4.1.7	Submodule VII - BMFIF	557
3.2.8.1.5	Interface Description	557
3.2.8.1.6	Test Requirements	561
3.2.8.2	"Clock" Function	568
3.2.8.2.1	Purpose	568
3.2.8.2.2	Requirements	568
3.2.8.2.2.1	Design Requirements	568
3.2.8.2.2.2	Derived Requirements	568
3.2.8.2.3	Design Approach	569
3.2.8.2.3.1	Functional Allocations	569
3.2.8.2.3.2	Resource Budgets	569
3.2.8.2.4	Design Description	569
3.2.8.2.4.1	Module Structure	569

TABLE OF CONTENTS (con't.)

<u>Section</u>	<u>Title</u>	<u>Page</u>
3.2.8.2.4.1.1	TOK - Major Interval Timer	569
3.2.8.2.4.1.2	TIK - Minor Interval Timer	573
3.2.8.2.5	Interface Description	573
3.2.8.2.6	Test Requirements	577
3.2.8.3	Receiver Trip	577
3.2.8.3.1	Purpose	577
3.2.8.3.2	Requirements	577
3.2.8.3.2.1	Design Requirements	577
3.2.8.3.2.2	Derived Requirements	577
3.2.8.3.3	Design Approach	577
3.2.8.3.3.1	Functional Allocations	578
3.2.8.3.3.2	Resource Budgets	578
3.2.8.3.4	Design Description	578
3.2.8.3.4.1	Module Structure	578
3.2.8.3.4.1.1	Submodule I - RTH	578
3.2.8.3.4.1.2	Submodule II - RTL	579
3.2.8.3.5	Interface Description	581
3.2.8.3.6	Test Requirements	583
3.2.8.4	Switching Function	584
3.2.8.4.1	Purpose	584
3.2.8.4.2	Requirements	584
3.2.8.4.2.1	Design Requirements	584
3.2.8.4.2.2	Derived Requirements	584
3.2.8.4.3	Design Approach	584
3.2.8.4.3.1	Functional Allocations	584
3.2.8.4.3.2	Resource Budgets	586
3.2.8.4.4	Design Description	586
3.2.8.4.4.1	Module Structure	586
3.2.8.4.4.1.1	Submodule I - SWI (Task)	586
3.2.8.4.4.1.2	Submodule II - SWIINT	590
3.2.8.4.4.1.3	Submodule III - SWIPCS	592
3.2.8.4.4.1.4	Submodule IV - SWIHFC	597
3.2.8.4.4.1.5	Submodule V - SWIRG1	598
3.2.8.4.4.1.6	Submodule VI - SWIRG8	600

TABLE OF CONTENTS (con't.)

<u>Section</u>	<u>Title</u>	<u>Page</u>
3.2.8.4.4.1.7	Failover Interrupt Handler	602
3.2.8.4.5	Interface Description	605
3.2.8.4.6	Test Requirements	605
3.2.8.5	Field Power Loss	606
3.2.8.5.1	Purpose	606
3.2.8.5.2	Requirements	606
3.2.8.5.2.1	Design Requirements	606
3.2.8.5.2.2	Derived Requirements	606
3.2.8.5.3	Design Approach	606
3.2.8.5.3.1	Functional Allocations	606
3.2.8.5.3.2	Resource Budgets	607
3.2.8.5.4	Design Description	607
3.2.8.5.4.1	Module Structure	607
3.2.8.5.4.1.1	Submodule I - PLH	607
3.2.8.5.4.1.2	Submodule II - PLL	608
3.2.8.5.5	Interface Description	610
3.2.8.5.6	Test Requirements	610
3.2.9	Graphics Display Processor Module - (GRAPHIC)	612
3.2.9.1	Purpose	612
3.2.9.2	Requirements	612
3.2.9.2.1	Design Requirements	612
3.2.9.2.2	Derived Requirements	612
3.2.9.3	Design Approach	613
3.2.9.3.1	Functional Allocations	613
3.2.9.3.2	Resource Budgets	624
3.2.9.4.1.1	Submodule I - GRF (Task)	624
3.2.9.4.1.2	Submodule II - GRFSEG	628
3.2.9.4.1.3	Submodule III - GRFFLD	633
3.2.9.5	Interface Description	635
3.2.9.6	Test Requirements	635
3.2.10	External Interface Module - EXTINF	637
3.2.10.1	Purpose	637
3.2.10.2	Requirements	637
3.2.10.2.1	Design Requirements	637

TABLE OF CONTENTS (con't.)

<u>Section</u>	<u>Title</u>	<u>Page</u>
3.2.10.2.2	Derived Requirements	637
3.2.10.3	Design Approach	638
3.2.10.3.1	Functional Allocations	639
3.2.10.3.2	Resource Budgets	639
3.2.10.4	Design Description	639
3.2.10.4.1	Module Structure	641
3.2.10.4.1.1	Submodule I - CSI Task (Main Routine)	641
3.2.10.4.1.2	Submodule II - CSO Task	642
3.2.10.4.1.3	Submodule III - EXI Task	646
3.2.10.4.1.4	Submodule IV - EXO Task	650
3.2.10.5	Interface Description	650
3.2.10.6	Test Requirements	656
3.2.11	BCS Automatic Processing Module - BCSMOD	658
3.2.11.1	Purpose	658
3.2.11.2	Requirements	658
3.2.11.2.1	Design Requirements	658
3.2.11.2.2	Derived Requirements	659
3.2.11.3	Design Approach	660
3.2.11.3.1	Functional Allocations	674
3.2.11.3.2	Resource Budgets	680
3.2.11.4	Design Description	682
3.2.11.4.1	Module Structure	682
3.2.11.4.1.1	Submodule I - BCS Task	682
3.2.11.4.1.2	Submodule II - BCSCMD	696
3.2.11.4.1.3	Submodule III - BCSCCTL	697
3.2.11.4.1.4	Submodule IV - BCSCMMC	702
3.2.11.4.1.5	Submodule V - BCSCMEA	703
3.2.11.4.1.6	Submodule VI - BCSCMSG	707
3.2.11.4.1.7	Submodule VII - BCSCOUT	708
3.2.11.4.1.8	Submodule XVIII - BCSCMES	711
3.2.11.4.1.9	Submodule IX - BCSCON	719
3.2.11.4.1.10	Submodule X - BCSCRX	721
3.2.11.4.1.11	Submodule XI - BCSDS1	725
3.2.11.4.1.12	Submodule XII - BCSDS2	727

TABLE OF CONTENTS (con't.)

<u>Section</u>	<u>Title</u>	<u>Page</u>
3.2.11.4.1.13	Submodule XIII - BCSP12	729
3.2.11.4.1.14	Submodule XIV - BCSMT	733
3.2.11.4.1.15	Submodule XV - BCSFIL	736
3.2.11.4.1.16	Submodule XVI - BCSRNG	738
3.2.11.4.1.17	Submodule XVII - BCSP31	740
3.2.11.4.1.18	Submodule XVIII - BCSP32	744
3.2.11.4.1.19	Submodule XIX - BCSP33	748
3.2.11.4.1.20	Submodule XX - BCSCAL	752
3.2.11.4.1.21	Submodule XXI - BCSLOG	754
3.2.11.4.1.22	Submodule XXII - BCSBIA	756
3.2.11.4.1.23	Submodule XXIII - BCSCOR	758
3.2.11.4.1.24	Submodule XXIV - BCSSLQ	760
3.2.11.4.1.25	Submodule XXV - BCSSUN	762
3.2.11.4.1.26	Submodule XXVI - BCSGIM	764
3.2.11.4.1.27	Submodule XXVII - BCSNOR	765
3.2.11.4.1.28	Submodule XXVIII - BCSLSC	767
3.2.11.4.1.29	Submodule XXIX - BCSEPD	769
3.2.11.4.1.30	Submodule XXX - BCSVEC	771
3.2.11.4.1.31	Submodule XXXI - BCSLIN	773
3.2.11.4.1.32	Submodule XXXII - BCSMPA	775
3.2.11.4.1.33	Submodule XXXIII - BCSBOS	779
3.2.11.4.1.34	Submodule XXXIV - BCSAL2	781
3.2.11.4.1.35	Submodule XXXV - BCSMPB	783
3.2.11.4.1.36	Submodule XXXVI - BCSMPC	784
3.2.11.4.1.37	Submodule XXXVII - BCSGET	787
3.2.11.4.1.38	Submodule XXXVIII - BCSCLR	791
3.2.11.4.1.39	Submodule XXXIX - BCSRTN	793
3.2.11.4.1.40	Submodule XL - BCSSTB	794
3.2.11.4.1.41	Submodule XLI - BCSSTS	797
3.2.11.4.1.42	Submodule XLII - BCSNET	799
3.2.11.4.1.43	Submodule XLIII - BCSNWT	801
3.2.11.5	Interface Description	803
3.2.11.6	Test Requirements	803

TABLE OF CONTENTS (con't.)

<u>Section</u>	<u>Title</u>	<u>Page</u>
3.2.12	Heliostat Data Storage Capability - HDSMOD	804
3.2.12.1	Purpose	804
3.2.12.2	Requirements	804
3.2.12.2.1	Design Requirements	804
3.2.12.2.2	Derived Requirements	804
3.2.12.3	Design Approach	805
3.2.12.3.1	Functional Allocations	805
3.2.12.3.2	Resource Budgets	806
3.2.12.4	Design Description	808
3.2.12.4.1	Submodule I - HDS	808
3.2.12.4.2	Submodule II - HDSRDT	811
3.2.12.4.3	Submodule III - HDSCNV	816
3.2.12.4.4	Submodule IV - HDSINP	818
3.2.12.4.5	Submodule V - HDSGET	820
3.2.12.4.6	Submodule VI - HDSLOG	822
3.2.12.4.7	Submodule VII - HDSWRI	824
3.2.12.4.8	Submodule VIII - HDSDEL	827
3.3.1	Data Base Design	830
3.3.1.1	Purpose	830
3.3.1.2	Requirements	830
3.3.1.2.1	Design Requirements	831
3.3.1.2.2	Derived Requirements	832
3.3.1.3	Design Approach	837
3.3.1.3.1	Functional Allocations	837
3.3.1.3.2	Resource Budgets	838
3.3.1.4	Design Description	838
3.3.1.4.1	Structure	838
3.3.1.4.1.1	Global Common Area COM1S1	838
3.3.1.4.1.2	Global Common Area COM1S2	839
3.3.1.4.1.3	Global Common Area COM1S3	840
3.3.1.4.1.4	Global Common Area COM1S4	841
3.3.1.4.1.5	Global Common Area COM1S5	844
3.3.1.4.1.6	Global Common Area COM1S6	850
3.3.1.4.1.7	Global Common Area COM8S1	851

TABLE OF CONTENTS (con't.)

<u>Section</u>	<u>Title</u>	<u>Page</u>
3.3.1.4.1.8	Global Common Area COM8S2	852
3.3.1.4.1.9	Global Common Area COM8S3	853
3.3.1.4.1.10	Global Common Area COM8S4	854
3.3.1.4.1.11	Global Common Area COMIN1	854
3.3.1.4.1.12	Global Common Area COMIN2	855
3.3.1.4.1.13	Global Common Area COMIN3	855
3.3.1.4.1.14	Global Common Area COMIN4	858
3.3.1.4.1.15	Free Storage (COMQUE)	862
3.3.1.4.1.16	Free Storage (COMQU2)	863
3.3.1.4.1.17	Disk Data Base	863
3.3.1.5	Interface Description	866
3.3.1.6	Test Requirements	876
3.4	HFC Firmware Design	885

TABLE OF CONTENTS (con't.)

<u>Section</u>	<u>Title</u>	<u>Page</u>
3.4.1	HFC Firmware Module - HFCMOD	885
3.4.1.1	Purpose	885
3.4.1.2	Requirements	885
3.4.1.2.1	Design Requirements	885
3.4.1.2.2	Derived Requirements	886
3.4.1.3	Design Approach	886
3.4.1.3.1	Functional Allocations	886
3.4.1.3.2	Resource Budgets	887
3.4.1.4	Design Description	887
3.4.1.4.1	Module Structure	887
3.4.1.4.2	Submodule I - Command Interpreter (CMDI)	889
3.4.1.4.2.2	CMDI Subroutine I - CMDCMD	890
3.4.1.4.2.3	CMDI Subroutine II - CMDINI	895
3.4.1.4.2.4	CMDI Subroutine III - CMDS4	895
3.4.1.4.2.5	CMDI Subroutine IV - POLIHC	898
3.4.1.4.3	Submodule II - Corridor-Walk Calculation (CWCALC)	898
3.4.1.4.3.1	Main Routine	898
3.4.1.4.4	Submodule III - HC Operations (HCOPS)	901
3.4.1.4.4.1	Main Routine	901
3.4.1.4.4.2	HCOPS Subroutine I - HCOSSC	904
3.4.1.4.4.3	HCOPS Subroutine II - HCOCRR	904
3.4.1.4.4.4	HCOPS Subroutine III - HCOPOL	904
3.4.1.4.5	Submodule IV - Emergency Corridor-Walk Operations (ECWOPS)	908
3.4.1.4.5.1	Main Routine	908
3.4.1.4.5.2	ECWOPS Subroutine I - ECWMG	910

TABLE OF CONTENTS (con't.)

<u>Section</u>	<u>Title</u>	<u>Page</u>
3.4.1.4.5.3	ECWOPS Subroutine II - ECWSEQ	912
3.4.1.4.6	Submodule V - HFC Timer Interrupt Handlers (HFCTMR)	916
3.4.1.4.6.1	Main Routine	916
3.4.1.4.6.2	HFCTMR Subroutine I - HFCTOC	918
3.4.1.4.6.3	HFCTMR Subroutine II - HFCTOF	918
3.4.1.4.7	Submodule VI - HAC Input Interrupt Handler (HACIN)	920
3.4.1.4.7.1	Main Routine	920
3.4.1.4.8	Submodule VII - HAC Output Interrupt Handler (HACOUT)	924
3.4.1.4.8.1	Main Routine	924
3.4.1.4.9	Submodule VIII - HC Input Interrupt Handler (HCIN)	924
3.4.1.4.9.1	Main Routine	924
3.4.1.4.10	Submodule IX - HC Output Interrupt Handler (HCOUT)	926
3.4.1.4.10.1	Main Routine	926
3.5	HC Firmware Design	930
3.5.1	HC Firmware Module - HCMOD	930
3.5.1.1	Purpose	930
3.5.1.2	Requirements	930
3.5.1.2.1	Design Requirements	930
3.5.1.2.2	Derived Requirements	930
3.5.1.3	Design Approach	931
3.5.1.3.1	Functional Allocations	931
3.5.1.3.2	Resource Budgets	939
3.5.1.3.2.1	Submodule Priority	939
3.5.1.3.3	Operating System Variables	939
3.5.1.4	Design Description	939
3.5.1.4.1	Module Structure	948
3.5.1.5	Interface Description	951
3.5.1.6	Test Requirements	951
3.6	Graphic Display Console (GDC) Software	952
3.6.1	Purpose	952
3.6.2	Requirements	952
3.6.2.1	Design Requirements	952

TABLE OF CONTENTS (con't.)

<u>Section</u>	<u>Title</u>	<u>Page</u>
3.6.2.2.	Derived Requirements	957
3.6.3	Design Approach	957
3.6.3.1	Functional Allocations	957
3.6.3.2	Resource Budgets	958
3.6.4	Design Description	959
3.6.4.1	Module Structure	959
3.6.4.1.1	Submodule I - EXEC (Main Routine)	959
3.6.4.1.2	Submodule II - RCV	961
3.6.4.1.2.1	Subroutine 1 - RCVISR	961
3.6.4.1.2.2	Subroutine 2 - HACAVAIL	963
3.6.4.1.2.3	Subroutine 3 - HACDATA	963
3.6.4.1.3	Submodule III - KBD	963
3.6.4.1.3.1	Subroutine 1 - KBDISR	967
3.6.4.1.3.2	Subroutine 2 - KBDVAIL	967
3.6.4.1.3.3	Subroutine 3 - KBDDATA	969
3.6.4.1.4	Submodule IV - SEND (Main Routine)	969
3.6.4.1.5	Submodule V - DIALOG	973
3.6.4.1.6	Submodule VI - FFDISP	974
3.6.4.1.6.1	Subroutine 1 - FFDISPI (Full-Field Display Initialization)	974
3.6.4.1.6.2	Subroutine 2 - FFDISPU (Full-Field Display Update)	976
3.6.4.1.6.3	Subroutine 3 - FFDISPD (Full-Field Display Delete)	979
3.6.4.1.7	Submodule VII - SEGDISP	979
3.6.4.1.7.1	Subroutine 1 - SEGDISPI (Segment Display Initialization)	979
3.6.4.1.7.2	Subroutine 2 - SEGIDSPU (Segment Display Update)	981
3.6.4.1.7.3	Subroutine 3 - SEGDISPD (Segment Display Delete)	984
3.6.4.1.8	Submodule VIII - HELP	984
3.6.4.1.8.1	Subroutine 1 - HELPI (Initialize HELP Display)	984
3.6.4.1.8.2	Subroutine 2 - HELPD (Delete HELP Display)	987
3.6.4.1.9	Submodule IX - PROHAC	987
3.6.4.1.10	Submodule X - FFPP	987
3.6.4.1.11	Submodule XI - SEGPP	992

TABLE OF CONTENTS (con't.)

<u>Section</u>	<u>Title</u>	<u>Page</u>
3.7	System Interfaces	996
3.7.1	HAC/HFC Interface Description	996
3.7.2	HFC/HC Interface Description	996
3.7.3	HAC/OCS-DAS Interface Description	996
3.7.4	HAC/GDC Interface Description	996
3.7.5	HAC/Receiver Interface Description	997
3.7.6	Man-Machine Interface Description	997
3.7.6.1	Command Input Mode	997
3.7.6.2	Command File Execute Mode	997
3.7.6.3	Command Logging Mode	997
3.7.6.4	Alarm Response Mode	998
3.7.6.5	Status Display Mode	998
4.0	SOFTWARE/FIRMWARE SYSTEM VALIDATION	1062
4.1	Test Phases	1062
4.2	Functional Testing	1062
4.3	Integration Testing	1062
4.4	Breadboard Integration Testing	1063
4.5	System Level Testing	1063
5.0	ACRONYMS	1064

LIST OF FIGURES

<u>Figure</u>	<u>Title</u>	<u>Page</u>
3.1-1	"Prime"	5
3.1-2	"Backup"	6
3.1-3	HAC Computer System Hardware Configuration	7
3.1-4	HAC Task Activation Flow	13
3.2.1-1	MANMIF Hierarchical Overview	31
3.2.1-2	MMI Functional Overview	32
3.2.1-3	DSK and CFO Functional Overview	33
3.2.1-4	MMI Submodule	37
3.2.1-5	MMI Global Common Area	38
3.2.1-6	MMI Global Common Input/Output	39
3.2.1-7	MMICOM Input/Output	40
3.2.1-8	Flowchart - MMI	47
3.2.1-9	Flowchart - MMIDEQ	52
3.2.1-10	Flowchart - MMIWRD	54
3.2.1-11	Flowchart - MMICLK	57
3.2.1-12	Flowchart - MMICMD	61
3.2.1-13	Flowchart - MMINUM	63
3.2.1-14	Flowchart - MMIHXD	65
3.2.1-15	Flowchart - MMIMAP	68
3.2.1-16	Flowchart - MMINON	72
3.2.1-17	Flowchart - MMISTR	75
3.2.1-18	Flowchart - MMIST2	77
3.2.1-19	Flowchart - MMIST3	79
3.2.1-20	Flowchart - MMIREF	81
3.2.1-21	Flowchart - MMIRSP	83
3.2.1-22	Flowchart - MMICRT	85
3.2.1-23	Flowchart - MMIERR	87
3.2.1-24	Flowchart - MMITST	90
3.2.1-25	Flowchart - MMIAID	92
3.2.1-26	Flowchart - MMIBCS	95
3.2.1-27	Flowchart - DSK	97
3.2.1-28	Flowchart - DSKAIM	99
3.2.1-29	Flowchart - DSKINK	100

LIST OF FIGURES (con't.)

<u>Figure</u>	<u>Title</u>	<u>Page</u>
3.2.1-30	Flowchart - DSKTRF	103
3.2.1-31	Flowchart - DSKBIA	105
3.2.1-32	Flowchart - DSKERR	107
3.2.1-33	Flowchart - DSKBAK	109
3.2.1-34	Flowchart - DSKUPD	111
3.2.1-35	Flowchart - CFO	114
3.2.1-36	Flowchart - CFOPOS	116
3.2.1-37	Flowchart - CFOGET	118
3.2.1-38	Flowchart - CFOMSG	120
3.2.2-1	Hierarchy Diagram for the Command Processor Module (CMDPRC)	129
3.2.2-2	Module CMDPRC Overview	130
3.2.2-3	CMD Hierarchy	134
3.2.2-4	Task GET Hierarchy	136
3.2.2-5	Task SEQ Hierarchy	137
3.2.2-6	Tasks BHI, BHC Hierarchy	138
3.2.2-7	Flowchart - CMD	142
3.2.2-8	Flowchart - CMDSCK	149
3.2.2-9	CMDSCK Input/Output Buffers	154
3.2.2-10	Flowchart - CMDFAD	155
3.2.2-11	Flowchart - CMDBAD	164
3.2.2-12	Flowchart - CMDINO	169
3.2.2-13	Flowchart - CMDARA	173
3.2.2-14	Flowchart - CMDSAL	181
3.2.2-15	Flowchart - CMDTRK	187
3.2.2-16	Flowchart - CMDDSK	190
3.2.2-17	Flowchart - CMDPOS	193
3.2.2-18	Flowchart - CMDSBY	195
3.2.2-19	Flowchart - CMDSDN	198
3.2.2-20	Flowchart - CMDBCS	200
3.2.2-21	Flowchart - CMDSUP	203
3.2.2-22	Flowchart - GET	205
3.2.2-23	HC Bias File (HCB)	209

LIST OF FIGURES (con't.)

<u>Figure</u>	<u>Title</u>	<u>Page</u>
3.2.2-24	HC Coordinates (HCC)	210
3.2.2-25	Flowchart - GETINI	211
3.2.2-26	Aim-Point Arrays File (AIM)	213
3.2.2-27	Flowchart - GETAIM	215
3.2.2-28	Format and Structure of HC ALT1STOW Angles Disk File	216
3.2.2-29	Flowchart - GETAL1	218
3.2.2-30	Format and Structure of ALT2STOW Angle Disk File	220
3.2.2-31	Flowchart - GETAL2	221
3.2.2-32	Format and Structure of Wash Angles File (WSH)	223
3.2.2-33	Flowchart - GETWSH	224
3.2.2-34	Format and Structure of HC Stow Angles Disk File	226
3.2.2-35	Flowchart - GETSTO	228
3.2.2-36	Tracking Configuration Save File (AIM)	232
3.2.2-37	Flowchart - GETSAR	233
3.2.2-38	Wait-Sequence List (WAITSQ)	236
3.2.2-39	Flowchart - SEQ	239
3.2.2-40	Flowchart - SEQGAT	244
3.2.2-41	Flowchart - SEQBPT	247
3.2.2-42	Flowchart - SEQCCK	251
3.2.2-43	Flowchart - SEQCOR	254
3.2.2-44	Example of ACTLIS Management	257
3.2.2-45	Flowchart - SEQADD	261
3.2.2-46	Flowchart - SEQRLK	263
3.2.2-47	Flowchart - SEQDEL	267
3.2.2-48	Flowchart - BHI	270
3.2.2-49	Flowchart - BHISH5	271
3.2.2-50	Flowchart - BHC	274
3.2.3-1	SUNVEC Module	279
3.2.3-2	Flowchart - SUN	283
3.2.3-3	Flowchart - SUNPOS	287
3.2.3-4	Flowchart - SUNDJ	289
3.2.3-5	Flowchart - SUNCON	291
3.2.3-6	Flowchart - SUNPOL	293
3.2.3-7	Flowchart - SUNDMA	295

LIST OF FIGURES (con't.)

<u>Figure</u>	<u>Title</u>	<u>Page</u>
3.2.4-1	Flowchart - FCP Functional Overview	303
3.2.4-2	FLDCOM One-Second Communications Time Frame	304
3.2.4-3	FLDCOM Module	308
3.2.4-4	Flowchart - FCP Task	311
3.2.4-5	Flowchart - FCPOUT	315
3.2.4-6	Flowchart - FCPIN	318
3.2.4-7	Flowchart - FCPCHO	321
3.2.4-8	Flowchart FCPCHI	324
3.2.4-9	Flowchart - FCPUPD	324
3.2.4-10	Flowchart - FCPSWH	327
3.2.5-1	Alarms Processing Overview	331
3.2.5-2	Alarms Status Levels and Related Tables	333
3.2.5-3	Alarms Module Structure	337
3.2.5-4	Flowchart - ALM	341
3.2.5-5	Flowchart - ALMDTC	343
3.2.5-6	Flowchart - ALMFLD	345
3.2.5-7	Flowchart - ALMLNE	347
3.2.5-8	Flowchart - ALMHFC	349
3.2.5-9	Flowchart - ALMHC	351
3.2.5-10	Flowchart - ALMGET	355
3.2.5-11	Flowchart - ALMCLT	358
3.2.5-12	Flowchart - ALMRPT	360
3.2.5-13	Flowchart - ALMBLK	362
3.2.5-14	Flowchart - ALMQUE	365
3.2.5-15	Flowchart - ALO	368
3.2.5-16	Flowchart - ALOBLD	373
3.2.5-17	Flowchart - ALODQU	376
3.2.5-18	Flowchart - ALOCVT	377
3.2.6-1	STATUS Module Structure	381
3.2.6-2	HAC Operator's Console Synchronous Status Display	382
3.2.6-3	STAFLD - Field Status Display	386
3.2.6-4	STAIND Display Format	389
3.2.6-5	STAMOD Display Format	393

LIST OF FIGURES (con't.)

<u>Figure</u>	<u>Title</u>	<u>Page</u>
3.2.6-6	STARNG Segment Track Status	394
3.2.6-7	Flowchart - STS	398
3.2.6-8	Flowchart - STSGET	401
3.2.6-9	Flowchart - STA	403
3.2.6-10	Flowchart - STAFLD	404
3.2.6-11	Flowchart - STAIND	406
3.2.6-12	Flowchart - STAMOD	408
3.2.6-13	Flowchart - STARNG	410
3.2.6-14	Flowchart - STAGET	412
3.2.6-15	Flowchart - STAMCH	414
3.2.6-16	Flowchart - STACRT	416
3.2.7-1	Hierarchy Diagram of the Data Base Initialization Module (DBINIT)	419
3.2.7-2	Functional Diagram of the Disk Initialization Task	420
3.2.7-3	Functional Allocation of the HAC Initialization Task	421
3.2.7-4	Functional Diagram of the System Interface Initialization Submodule	422
3.2.7-5	Functional Diagram of the Data Initialization Control Submodule	423
3.2.7-6	Functional Diagram of the HAC Start-Up Task	424
3.2.7-7	HC Numbering Scheme Mappings	442
3.2.7-8	Segment Map and Segment Pointer Interaction	444
3.2.7-9	Flowchart - DIN	446
3.2.7-10	Flowchart - DINLOC	453
3.2.7-11	Flowchart - DINTRF	462
3.2.7-12	Flowchart - DININC	465
3.2.7-13	Flowchart - DINSRT	468
3.2.7-14	Flowchart - DINTTL	471
3.2.7-15	Flowchart - DINANG	475
3.2.7-16	Flowchart - DINPAC	479
3.2.7-17	Flowchart - DINFLG	482
3.2.7-18	Flowchart - DINLAC	484
3.2.7-19	Flowchart - DINCKI	486
3.2.7-20	Flowchart - DINCKR	488
3.2.7-21	Flowchart - DINAEA	490

LIST OF FIGURES (con't.)

<u>Figure</u>	<u>Title</u>	<u>Page</u>
3.2.7-22	Flowchart - DINGRF	493
3.2.7-23	Flowchart - DINUPC	495
3.2.7-24	Flowchart - DBI	498
3.2.7-25	Flowchart - DBIFAC	500
3.2.7-26	Flowchart - DBICPU	502
3.2.7-27	Flowchart - DBICRT	505
3.2.7-28	Flowchart - DBIPCS	507
3.2.7-29	Flowchart - DBITRP	510
3.2.7-30	Flowchart - DBIDTA	513
3.2.7-31	Flowchart - DBIBCK	515
3.2.7-32	Flowchart - DBICOR	519
3.2.7-33	Flowchart - DBIAIM	523
3.2.7-34	Flowchart - DBIBCK	526
3.2.7-35	Flowchart - DBIERR	530
3.2.7-36	Flowchart - CLK	533
3.2.7-37	Flowchart - CLKEST	535
3.2.7-38	Flowchart - Computation of Local Time	539
3.2.7-39	Flowchart - Computation of GMT Parameters	540
3.2.7-40	Flowchart - CLKONL	541
3.2.7-41	Flowchart - CLKACT	543
3.2.8.1-1	Internal Buffer Structure	547
3.2.8.1-2	Flowchart - QINIT	548
3.2.8.1-3	Flowchart - LEASE	550
3.2.8.1-4	Flowchart - FREE	551
3.2.8.1-5	Message Structure	553
3.2.8.1-6	Flowchart - ENQUE	554
3.2.8.1-7	Flowchart - DEQUE	556
3.2.8.1-8	Flowchart - QRST	558
3.2.8.1-9	Flowchart - BMFIF	559
3.2.8.1-10	Interface Specification for QINIT	560
3.2.8.1-11	Interface Specification for LEASE	562
3.2.8.1-12	Interface Specification for FREE	563
3.2.8.1-13	Interface Specification for ENQUE	564

LIST OF FIGURES (con't.)

<u>Figure</u>	<u>Title</u>	<u>Page</u>
3.2.8.1-14	Interface Specification for DEQUE	565
3.2.8.1-15	Interface Specification for QRST	566
3.2.8.1-16	Interface Specification for BMFIF	567
3.2.8.2-1	Clock Module, Component Task Interaction	570
3.2.8.2-2	Flowchart - TOK	574
3.2.8.2-3	Flowchart - TIK	575
3.2.8.3-1	Flowchart - RTH	580
3.2.8.3-2	Flowchart - RTL	582
3.2.8.4-1	Flowchart - SWI	591
3.2.8.4-2	Flowchart - SWIINT	593
3.2.8.4-3	Flowchart - SWIPCS	596
3.2.8.4-4	Flowchart - SWIHFC	599
3.2.8.4-5	Flowchart - SWIRG1	601
3.2.8.4-6	Flowchart - SWIRG8	603
3.2.8.4-7	Flowchart - Failover Interrupt Handler	604
3.2.8.5-1	Flowchart - PLH	609
3.2.8.5-2	Flowchart - PLL	611
3.2.9-1	GRAPHIC Functional Interface Diagram	614
3.2.9-2	Functional Allocation of the GRF Task	619
3.2.9-3	Flowchart - GRF	629
3.2.9-4	Flowchart - GRFSEG	634
3.2.9-5	Flowchart - GRFFLD	636
3.2.10-1	External Interface Module	640
3.2.10-2	Flowchart - CSI	643
3.2.10-3	Flowchart - CSO	647
3.2.10-4	Flowchart - EXI	651
3.2.10-5	Flowchart - EXO	655
3.2.11-1	BCS Submodule Diagram - Group 1	675
3.2.11-2	BCS Submodule Diagram - Group 2	677
3.2.11-3	BCS Submodule Diagram - Group 3	679
3.2.11-4	BCS Submodule Diagram - Group 4	681
3.2.11-5	Flowchart - BCS	694
3.2.11-6	Flowchart - BCSCMD	698
3.2.11-7	Flowchart - BCCTL	701

LIST OF FIGURES (con't.)

<u>Figure</u>	<u>Title</u>	<u>Page</u>
3.2.11-8	Flowchart - BCSDMM	704
3.2.11-9	Flowchart - BCSDMA	706
3.2.11-10	Flowchart - BCSDMG	709
3.2.11-11	Flowchart - BCSDOT	712
3.2.11-12	Flowchart - BCSDMS	718
3.2.11-13	Flowchart - BCSDON	720
3.2.11-14	Flowchart - BCS Displays	722
3.2.11-15	Flowchart - BCSDRX	724
3.2.11-16	Flowchart - BCSDS1	726
3.2.11-17	Flowchart - BCSDS2	728
3.2.11-18	Flowchart - BCSP12	731
3.2.11-19	Flowchart - BCSDMT	735
3.2.11-20	Flowchart - BCSDFIL	737
3.2.11-21	Flowchart - BCSDRNG	739
3.2.11-22	Flowchart - BCSP31	742
3.2.11-23	Flowchart - BCSP32	746
3.2.11-24	Flowchart - BCSP33	750
3.2.11-25	Flowchart - BCSDCAL	753
3.2.11-26	Flowchart - BCSDLOG	755
3.2.11-27	Flowchart - BCSDBIA	757
3.2.11-28	Flowchart - BCSDCOR	759
3.2.11-29	Flowchart - BCSDLSQ	761
3.2.11-30	Flowchart - BCSDSUN	763
3.2.11-31	Flowchart - BCSDGIM	766
3.2.11-32	Flowchart - BCSDNOR	768
3.2.11-33	Flowchart - BCDSLSC	770
3.2.11-34	Flowchart - BCSDSEP	772
3.2.11-35	Flowchart - BCSDVEC	774
3.2.11-36	Flowchart - BCDSLIN	776
3.2.11-37	Flowchart - BCSDMPA	778
3.2.11-38	Flowchart - BCSDBOS	780
3.2.11-39	Flowchart - BCSDAL2	782
3.2.11-40	Flowchart - BCSDMPB	785
3.2.11-41	Flowchart - BCSDMPC	788

LIST OF FIGURES (con't.)

<u>Figure</u>	<u>Title</u>	<u>Page</u>
3.2.11-42	Flowchart - BCSGET	790
3.2.11-43	Flowchart - BCSCLR	792
3.2.11-44	Flowchart - BCSRTN	795
3.2.11-45	Flowchart - BCSSTB	796
3.2.11-46	Flowchart - BCSSTS	798
3.2.11-47	Flowchart - BCSNET	800
3.2.11-48	Flowchart - BCSNWT	802
3.2.12-1	HMSMOD Function Diagram	807
3.2.12-2	Flowchart - HDS	810
3.2.12-3	Flowchart - HDSRDT	814
3.2.12-4	Flowchart - HDSCNV	817
3.2.12-5	Flowchart - HDSINP	819
3.2.12-6	Flowchart - HDSGET	821
3.2.12-7	Flowchart - HDSLOG	823
3.2.12-8	Flowchart - HDSWRI	826
3.2.12-9	Flowchart - HDSDEL	829
3.3.1.4-1	Format and Structure of the Aim-Point File (AIM)	864
3.3.1.4-2	Format and Structure of the HC ALT1STOW Angles Disk File (AL1)	865
3.3.1.4-3	Format and Structure of ALT2STOW Angles Disk File (AL2)	867
3.3.1.4-4	Format and Structure of File DIN	868
3.3.1.4-5	Format and Structure of the HC Locations Disk File (HCC)	870
3.3.1.4-6	Format and Structure of the Bias Disk File (HCB)	871
3.3.1.4-7	Alarm Messages Stored in File SAM/DBI	872
3.3.1.4-8	Tracking Configuration Save File (AIM)	873
3.3.1.4-9	Format and Structure of HC Stow Angles Disk File	874
3.3.1.4-10	Format and Structure of the Wash Angles Disk File (WSH)	875
3.4.1-1	HFC Module Structure	888
3.4.1-2	Flowchart - CMDI	892
3.4.1-3	Flowchart - NRMHI	894
3.4.1-4	Flowchart - CMDCMD	896
3.4.1-5	Flowchart - CMDINI	896
3.4.1-6	Flowchart - CMDS4	899

LIST OF FIGURES (con't.)

<u>Figure</u>	<u>Title</u>	<u>Page</u>
3.4.1-7	Flowchart - POL1HC	900
3.4.1-8	Flowchart - CWCALC	902
3.4.1-9	Flowchart - CKDELTA	903
3.4.1-10	Flowchart - HCOPS	905
3.4.1-11	Flowchart - HCOSSC	906
3.4.1-12	Flowchart - HCOARR	907
3.4.1-13	Flowchart - HCOPL	909
3.4.1-14	Flowchart - ECWOPS	911
3.4.1-15	Flowchart - ECWGM	913
3.4.1-16	Flowchart - MGIHC	914
3.4.1-17	Flowchart - ECWSEQ	917
3.4.1-18	Flowchart - HFCTOC	919
3.4.1-19	Flowchart - HFCTOF	921
3.4.1-20	Flowchart - HACIN	922
3.4.1-21	Flowchart - HACIND	923
3.4.1-22	Flowchart - HACOUT	925
3.4.1-23	Flowchart - HCIN	927
3.4.1-24	Flowchart - HCIND	928
3.4.1-25	Flowchart - HCOU	929
3.5.1-1	HC System Memory Map	932
3.5.1-2	Flowchart - INIT	933
3.5.1-3	Flowchart - SYSCLK	934
3.5.1-4	Flowchart - DECODE	935
3.5.1-5	Flowchart - POINT	936
3.5.1-6	Flowchart - READE	937
3.5.1-7	Flowchart - AZMOTR, ELMOTR, POWER	938
3.5.1-8	Flowchart - SCIO	940
3.5.1-9	Flowchart - CALC	941
3.5.1-10	Flowchart - System Block Diagram	949
3.5.1-11	System State Diagram	950
3.6.2-1	Full Field Display	953
3.6.2-2	3 x 5 Symbol/Color Encoding for Full-Field Display	954
3.6.2-3	Segment Display	955
3.6.2-4	5 x 10 Symbol/Color Encoding For Segment Displays	956

LIST OF FIGURES (con't.)

<u>Figure</u>	<u>Title</u>	<u>Page</u>
3.6.4-1	GDC Module Structure	960
3.6.4-2	Flowchart - EXEC	962
3.6.4-3	Flowchart - RCVISR	964
3.6.4-4	Flowchart - HACAVAIL	965
3.6.4-5	Flowchart - HACDATA	966
3.6.4-6	Flowchart - KBDISR	968
3.6.4-7	Flowchart - KBDAVAIL	970
3.6.4-8	Flowchart - KBDDATA	971
3.6.4-9	Flowchart - SEND	972
3.6.4-10	Flowchart - DIALOG	975
3.6.4-11	Flowchart - FFDISPI	977
3.6.4-12	Flowchart - FFDISPU	978
3.6.4-13	Flowchart - FFDISPD	980
3.6.4-14	Flowchart - SEGDISPI	982
3.6.4-15	Flowchart - SEGDISPU	983
3.6.4-16	Flowchart - SEGDISPD	985
3.6.4-17	Flowchart - HELPI	986
3.6.4-18	Flowchart - HELPD	988
3.6.4-19	Flowchart - PROHAC	989
3.6.4-20	Flowchart - FFPP	991
3.6.4-21	SEGPP Internal Data Structures	994
3.6.4-22	Flowchart - SEGPP	995
3.7-1	FLDCOM One-Second Communications Time Frame	999

LIST OF TABLES

<u>Table</u>	<u>Title</u>	<u>Page</u>
3.1-I	HAC Computer Hardware Acronyms and Model Numbers	8
3.2.1-I	Collector Subsystem Command List	21
3.2.1-II	Collector Subsystem Command Addressing Format	27
3.2.1-III	MANMIF Errors	36
3.2.1-IV	MMI Local Common Areas	41
3.2.1-V	MMICOM Description	42
3.2.1-VI	STRCOM Description	45
3.2.1-VII	Description of ARGADD	56
3.2.2-I	CMD Error Return Codes and REsulting MMI Display Messages	140
3.2.2-II	Operational Commands and Their Valid Initial Orientations	144
3.2.2-III	MMI/CMD HC Block Buffer	158
3.2.2-IV	MMI/CMD Command Buffer	161
3.2.2-V	Sequence Command Packet	175
3.2.2-VI	Disk Data Command Packet Format	189
3.2.2-VII	Typical Cell in Active-Sequence List (ACTLIS)	255
3.2.5-I	ALARM Tables in Memory	353
3.2.5-II	ALARM Detection Tables	354
3.2.5-III	Compressed Message Format	364
3.2.5-IV	ASCII Alarm Formats	370
3.2.6-I	Heliostat Mode Definitions	379
3.2.6-II	10 MWe Status (STSGET)	385
3.2.6-III	HAC-OCS/DAS Environment Status (Field) Format	387
3.2.6-IV	HAC-OCS/DAS Environment Status (HC) Format	390
3.2.6-V	HAC-OCS/DAS Environment Status (Mode) Format	392
3.2.6-VI	HAC-OCS/DAS Environment Status (Ring) Format	395
3.2.7-I	Data Base Card Format - Card Type 1	426
3.2.7-II	Data Base Card Format - Card Type 2	427
3.2.7-III	Data Base Card Format - Card Type 3	428
3.2.7-IV	Data Base Card Format - Card Type 4	429
3.2.7-V	Data Base Card Format - Card Type 5	430
3.2.7-VI	Data Base Card Format - Card Type 6	431
3.2.7-VII	Source Data Error Handling Procedure	432

LIST OF TABLES (con't.)

<u>Table</u>	<u>Title</u>	<u>Page</u>
3.2.7-VIII	Aim Point Limits	464
3.2.7-IX	Error Messages	529
3.2.9-I	GDC-HAC GDC Command	615
3.2.9-II	GDC-HAC GDC Command	616
3.2.9-III	GDC-HAC GDC Command	617
3.2.9-IV	GDC-HAC GDC Command	618
3.2.9-V	HAC-GDC Text Message to GDC	620
3.2.9-VI	HAC-GDC Graphics Initialization	621
3.2.9-VII	HAC-GDC Full Field Status	623
3.2.9-VIII	HAC-GDC HAC Segment Status	625
3.2.11-I	BCS Command Summary	661
3.2.11-II	Control Processing Phase and State Description	666
3.2.11-III	CSTATE Condition Table	667
3.2.11-IV	State Transition Table for Control Processing	668
3.2.11-V	BCS Measurement Processing States (MSTATE(T))	669
3.2.11-VI	MSTATE Condition Table	670
3.2.11-VII	State Transition Table for Measurement Processing	671
3.2.11-VIII	BCS Target Numbers	672
3.2.11-IX	Heliostat Status	673
3.2.11-X	BCS Target Status	673
3.2.11-XI	BCS Global Common Reference Table	685
3.2.11-XII	Timing Variable	687
3.2.11-XIII	BCSCM1 - Control & Measurement Variables	688
3.2.11-XIV	BCSCM2 - Input Data by BCS Operator Commands	689
3.2.11-XV	BCSCM3 - Input Data by OCS Messages	690
3.2.11-XVI	BCSCM4 - Bias Calculation Variables	691
3.2.11-XVII	BCSCM5 - Miscellaneous Data	692
3.2.11-XVIII	BCSCM6 - Other Variables	693
3.2.11-XIX	Expected Command Messages from MMI	700
3.2.11-XX	BCS Messages to CSO	715
3.2.11-XXI	Unexpected BCS Messages to Console	716
3.2.11-XXII	BCS Alarms	717
3.2.11-XXIII	BCS Processing File Format	734

LIST OF TABLES (con't.)

<u>Table</u>	<u>Title</u>	<u>Page</u>
3.2.12-I	HDS Input Tape Format	813
3.2.12-II	HDS Output Tape Format	825
3.3.1-I	Task Utilization of Global Common COMDAT	877
3.3.1-II	Global Common to Task Assignments	882
3.3.1-III	Task Utilization of Message Queue Global Common COMQUE	883
3.3.1-IV	Task Utilization of Data Base Disk Files	884
3.4.1-I	HFC State Transition Matrix	891
3.5.1-I	HC System Operating Variables	942
3.7-I	HC Initialization Command Format	1000
3.7-II	Beam Pointing Command Format	1001
3.7-III	Corridor-Walk Start-Up Command Format	1002
3.7-IV	Azimuth/Elevation Pointing Command Format	1003
3.7-V	Status Poll Command Format	1004
3.7-VI	Four Heliostat Status Response Format	1005
3.7-VII	Heliostat Controller Status Bit Breakdown	1007
3.7-VIII	HFC Status Breakdown	1008
3.7-IX	Sun Position Command Format	1009
3.7-X	HFC Initialization Command Format (Subtypes 0,1,2)	1010
3.7-XI	HFC Initialization Command Format (Subtypes 3,4,5)	1011
3.7-XII	HFC Initialization Command Format (Subtype 6)	1012
3.7-XIII	HC Sun/Synchronization Command Format (Beam Pointing)	1013
3.7-XIV	HC Sun/Synchronization Command Format (Azimuth/ Elevation Pointing)	1016
3.7-XV	Sun/Synchronization Command Format (HC Initialization)	1019
3.7-XVI	HC Command Response Format	1022
3.7-XVII	HC Status Poll Command Format	1023
3.7-XVIII	HC Status REsponse Format	1024
3.7-XIX	OCS/DAS-HAC Commands Format	1025
3.7-XX	HAC-OCS/DAS Environment Format	1026
3.7-XXI	HAC-OCS/DAS Environment Status (Field) Format	1027
3.7-XXII	HAC-OCS/DAS Environment Status (Mode) Format	1029
3.7-XXIII	HAC-OCS/DAS Environment STATUS (HC) Format	1030
3.7-XXIV	HAC/OCS-DAS Environment Status (Ring) Format	1032

LIST OF TABLES (con't.)

<u>Table</u>	<u>Title</u>	<u>Page</u>
3.7-XXV	HAC/BCS BCS Initiation Request Message Format	1033
3.7-XXVI	BCS/HAC BCS Initiation Response Message Format	1034
3.7-XXVII	HAC/BCS BCS Measurement Initiation Request Message Format	1035
3.7-XXVIII	BCS/HAC BCS Measurement Initiation Response Message Format	1036
3.7-XXIX	HAC/BCS Heliostat on BCS Target Message Format	1037
3.7-XXX	BCS/HAC Heliostat BCS Removal REquest Message Format	1039
3.7-XXXI	Heliostat BCS REmoval REsponse Message Format	1040
3.7-XXXII	BCS/HAC BCS Measurement REsults	1041
3.7-XXXIII	HAC/BCS BCS Measurements REsults Response	1043
3.7-XXXIV	HAC/BCS BCS Termination Message Format	1045
3.7-XXXV	HAC/BCS Heliostat Measurement Historical Data Request Message Format	1046
3.7-XXXVI	HAC/BCS Heliostat Measurement Historical Data Response Format	1047
3.7-XXXVII	HAC/BCS Heliostat Bias Results	1049
3.7-XXXVIII	HAC-GDC Field Position Initialization	1051
3.7-XXXIX	HAC-GDC Full Field Status	1053
3.7-XL	HAC-GDC Segment Status	1054
3.7-XLI	HAC-GDC TExt Message to GDC	1055
3.7-XLII	HAC=GDC Graphics Initialization	1056
3.7-XLIII	GDC-HAC GDC Command	1057
3.7-XLIV	GDC-HAC GDC Command	1058
3.7-XLV	GDC-HAC Command	1059
3.7-XLVI	GDC-HAC GDC Command	1060
3.7-XLVII	Receiver-to-HAC Trip Signal Logic	1061

1.0

INTRODUCTION

1.1

Scope

This Collector Subsystem Software/Firmware Design Specification exists as a stand-alone document to provide a complete description of the software and firmware employed for the operation of the 10 MWe Solar Thermal Central Receiver Pilot Plant Collector Subsystem.

1.2

Problem Statement

The software/firmware systems have the capability to allow operator control of up to 2048 heliostats in the operation of the 10 MWe Solar Thermal Central Receiver Pilot Plant at Barstow, California. This function includes the capability of operator-commanded mode control, graphic displays, status displays, alarm generation, system redundancy and interfaces to the Operational Control System (OCS), the Data Acquisition System (DAS), and the Beam Characterization System (BCS) through the OCS. The operational commands will provide for the following:

- a. Safe beam movement whenever automatic beam movement is required;
- b. Single and multiple heliostat addressing;
- c. Emergency heliostat movement for high-wind conditions and receiver problems; and
- d. Recovery for full or partial power-loss conditions.

The control hardware consists of a host computer, the Heliostat Array Controller (HAC), interfaced to a group of communication controllers, the Heliostat Field Controllers (HFCs), communicating with individual processors, the Heliostat Controllers (HCs), which monitor and command a single heliostat. The system consists of two HACs and 64 HFCs with up to 32 HCs per HFC.

The solar position data must be calculated accurately and often enough so that the reflected beams remain on target throughout the day. The status interface must include general field status updated automatically, while specific heliostat status is available upon operator request. Alarms must be generated and displayed for abnormal conditions requiring operator attention.

An interface must be available to the OCS for command input, status requests and alarms allowing the operator a single Master Control System for plant control. In addition, an interface must be maintained with the DAS to provide data collection capability for plant study and analysis. The Collector Subsystem (CS) is one

of the most critical components of the pilot plant thus requiring redundant HACs and dual communications to the field. This system redundancy is necessary to minimize interruptions to plant operations due to computer failures.

1.3

Design Approach

The method used to design this software/firmware system was to functionally decompose the requirements into eleven software modules for execution in the HAC computer, one firmware module for execution in the HFC microprocessor, and one firmware module for execution in the HC microprocessor. First the interfaces between the HAC computer/HFC microprocessor and HFC microprocessor/HC microprocessor were thoroughly defined. Next, the interfaces between the individual HAC software modules were defined. Further decomposition of the modules was performed until each submodule (independent tasks or subroutines) was defined.

The design of these modules is presented in paragraphs 3.2, 3.4, and 3.5. The data packets have been defined for the interface with the OCS, BCS, DAS, and between the Chromatics terminals and the HAC.

APPLICABLE DOCUMENTS

The following are the documents applicable to the development of the software for the project:

- a. Department of Energy Request for Proposal No. DE-RP03-79SF10539 for "Phase II, Collector Subsystem for the 10 MWe Solar Thermal Central Receiver Pilot Plant" received 7 June 1979;
- b. Amendment 2 to RFP DE-RP03-79SF10539, received 16 July, 1979;
- c. Amendment 3 to RFP DE-RP03-79SF10539, received 20 July 1979;
- d. CS-MCS and CS-Plant Interface Requirements document, (MDC G7852) dated 18 April 1980; RE changed by items j and k;
- e. Martin Marietta Proposal for Phase II, Collector Subsystem for 10 MWe Solar Central Receiver Pilot Plant, dated 6 August 1979
 1. Volume 1 - Technical Proposal P79-48372-1
 2. Volume 2 - Price/Business Proposal P78-48372-2;
- f. Martin Marietta Denver Division Software Development Procedure (Standard Procedure 1.4 dated 19 May 1978);
- g. Martin Marietta Corporation Aerospace Division Software Standards (Denver Division Engineering Practices Manual G13 dated 2 July 1979);
- h. 10 MWe Collector Subsystem Software/Firmware Development Plan;
- i. 10 MWe Collector System Contract Change Order #4, dated 19 February 1980;
- j. STMPO letter dated 16 April 1980, Collector Control System Changes;
- k. STMPO letter dated 5 May 1980, Collector Interface Meeting on 28 April 1980;
- l. 10 MWe Collector System Contract Change Order #8, dated 30 May 1980;
- m. Martin Marietta Denver Division Policy TO-8-D1, Software Development Management and Control.
- n. 10 MWe Collector Subsystem Software/Firmware Functional Requirements Specification, MCR-80-1341 dated 12 June 1980.
- o. 10 MWe Collector System Contract Change Order #16, dated 2 June 1981.
- p. 10 MWe Collector System Contract Change Order #17, dated 9 June 1981.

3.0

DESIGN REQUIREMENTS

The requirements satisfied by each module of the Software/firmware system are taken from the 10 MWe Collector Subsystem Software/Firmware Functional Requirements Specification. In addition, this section details the design of the modules to satisfy the requirements, the interfaces between the computers, the software system structure, and the computers in which the software and firmware will execute.

3.1

System Functional Description

The top-level flowchart diagrams of the HAC "Prime" and "Backup" software system are shown in Figures 3.1-1 and 3.1-2 and show the data flow of the system. The structure of the software system relies heavily on the Request Executive Services feature of the MAX IV Operating System which allows tasks to suspend themselves until a specified time has elapsed, and to activate other tasks which have been suspended. These tasks remain memory resident, but use no processing time until activated. The structure of the software system in the HAC computer can be described as a synchronous sequence of tasks and an asynchronous sequence of tasks.

The synchronous sequence tasks execute in one-second intervals or "frames." The purpose of these "frames" is field synchronization and communications. During the one-second frame: the sun position vector is transmitted to the entire field; a new sun position is calculated to be used in the next "frame;" a status poll is generated and status is received from one-eighth of the field; status is updated and reported based on the status data; alarms are generated automatically for irregular heliostat operation; global data required for failover is transferred to the "Backup" computer; status data is made available for transfer to the Graphics terminals; and processed commands are transmitted to the field.

The asynchronous sequence of tasks is initiated by command input from the CS Control Console, data received from the Chromatics Terminals, command input from the OCS or DAS, or activation by another task within the system. Inter-task communication is handled via the global common data base.

3.1.1

Hardware System Description

The computer systems of the Control Subsystem are the HAC computer, the HFC microprocessor, the HC microprocessor, and the Chromatics Intelligent Terminals. Each of these computers is described in the following paragraphs.

3.1.1.1

HAC Computer System

The HAC computer system hardware configuration is shown in Figure 3.1-3 and the HAC computer hardware Acronyms and Model Numbers are listed in Table 3.1-I. This consists of

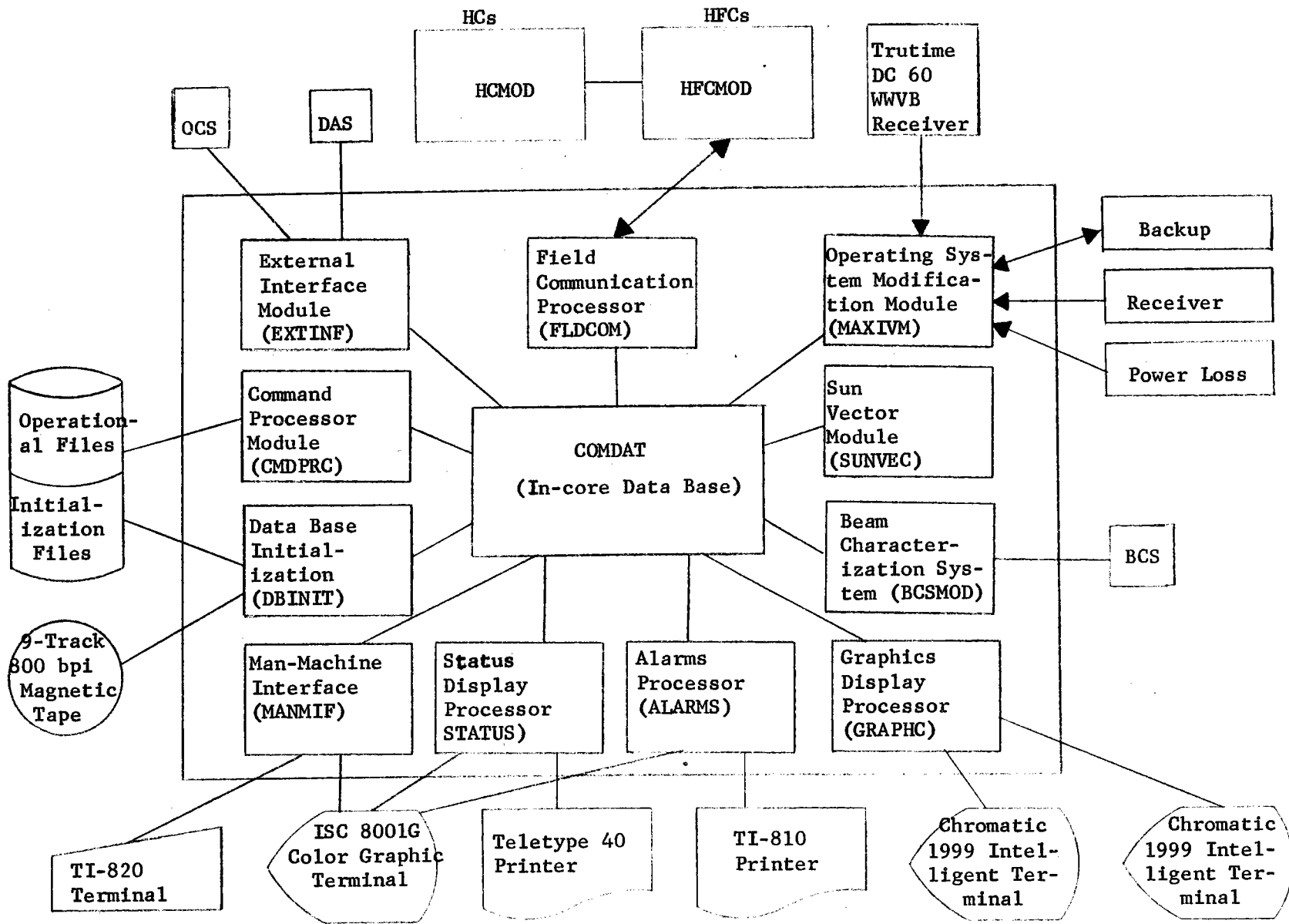


Figure 3.1-1 "Prime"

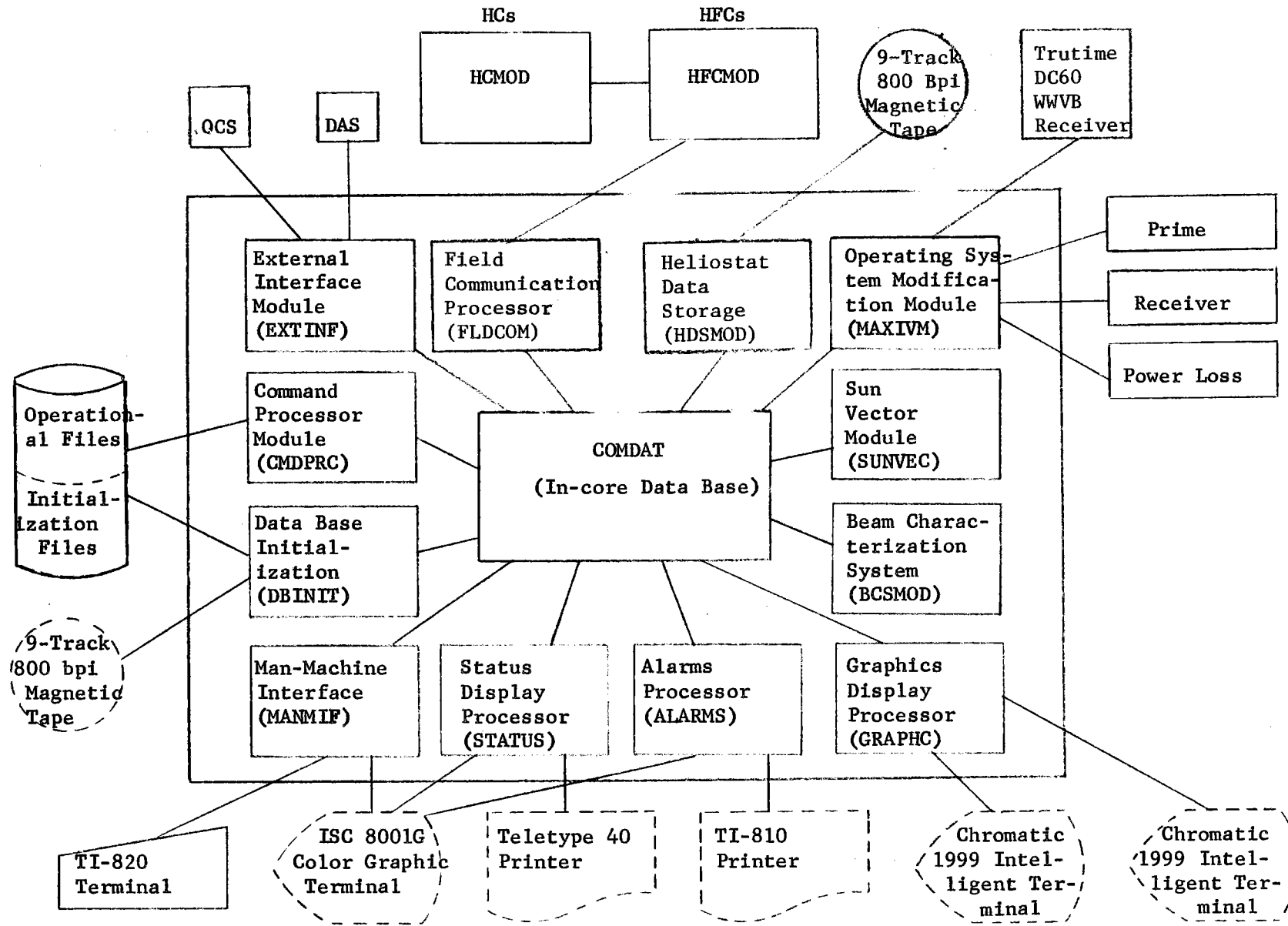


Figure 3.1-2 "Backup"

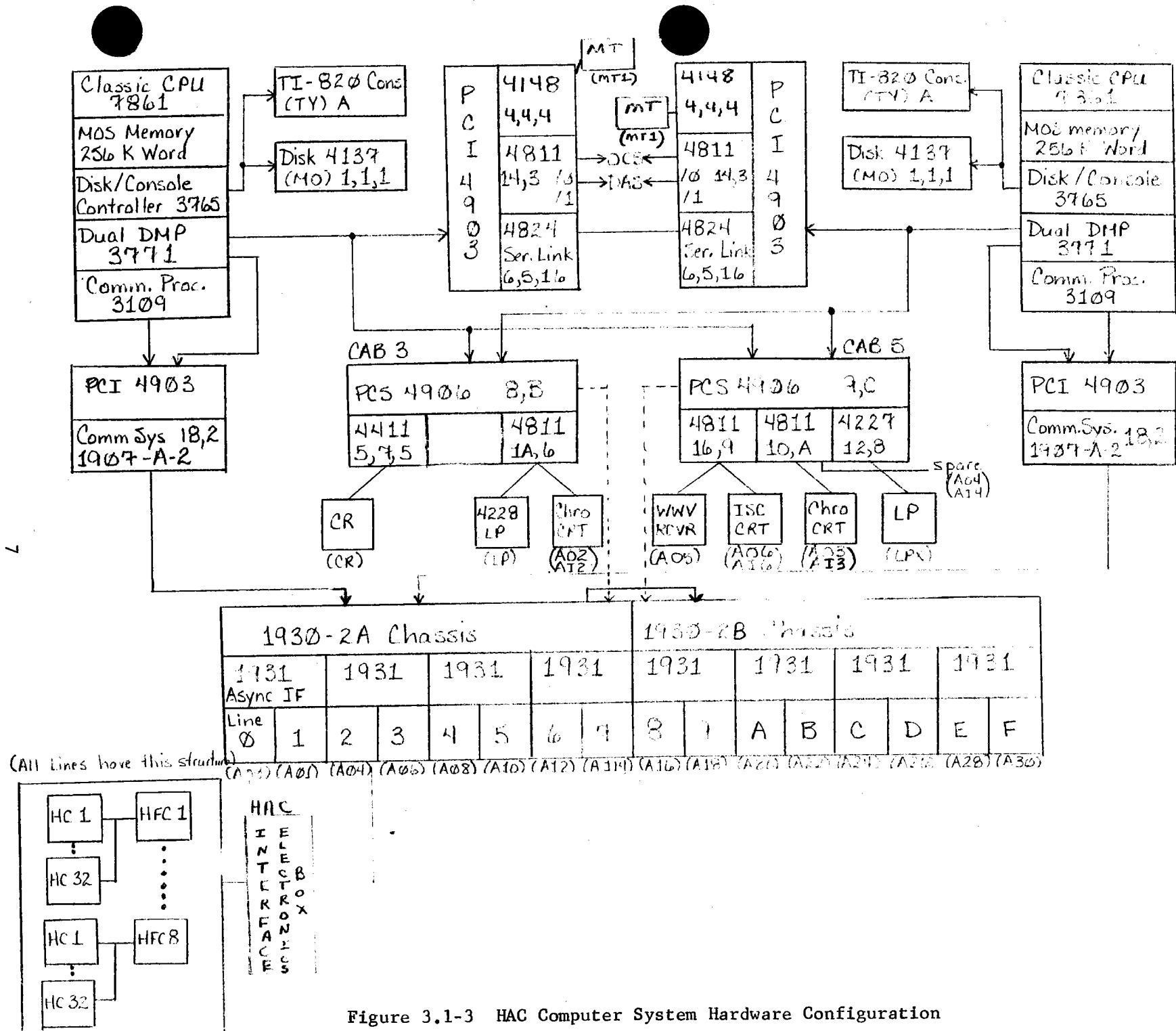


Figure 3.1-3 HAC Computer System Hardware Configuration

HAC COMPUTER HARDWARE ACRONYMS AND MODEL NUMBERS

7861	MODCOMP Classic Central Processing Unit.
CPU	See "7861"
3765	Combination 4137 disk and operator's console controller board
4137	Disk pack drive, with one removeable and one fixed disk. Each disk has 816 tracks, 24 sectors per track, 256 bytes per sector (total 10 megabytes per drive). These drives are also known loosely as "200 TPI" drives.
TI-820	Hard-copy operator's console, manufactured by Texas Instruments, capable of 120 characters/second.
3771	Dual port direct memory access (DMP) processor.
4903	Chassis box and power supply for containing peripheral controllers. Also known as a Peripheral Controller Interface (PCI).
4906	Same as 4903, but containing electronics allowing one of two CPUs to connect itself to the enclosed peripherals either manually or via automatic control. Also known as a Peripheral Control Switch (PCS).
4824	One half of a full duplex CPU-CPU serial link, operating at 125K baud.
4811	General purpose asynchronous controller, with two independent full duplex channels.
4411	DMP card reader with controller.
4227	Teletype Model 40 printer with controller.
4228	TI Model 810 RO terminal, used as low-speed printer.
3109	Communications Processor (CP), an independent microprocessor-controlled communications controller.
1907-A	Communications line multiplexor.
1930	Chassis and power supply for communications handlers, such as 1931s.
1931	A two channel, full duplex asynchronous controller.
PCI	See 4903.
PCS	See 4906.
CP	See 3109.
4148	Nine-track 800 bpi magnetic tape

Table 3.1-I

two MODCOMP CLASSIC computers (#7861) each with 256K words of MOS memory, combination disk and operator's console controller (#3765), dual-port direct memory access (DMP) processor (#3765), Communication Processor (CP) option (#3109), and two Peripheral Controller Interfaces (PCIs) (#4903). One of the PCIs contains a communication line multiplexer (#1907-A-2) for communications with the field, the other contains a general purpose asynchronous controller (#4811) for communications with the OCS and DAS and a CPU-CPU serial link (#4824) for communications with the other HAC computer. The field communications is handled via the 1907-A-2 which addresses either the normal or alternate two-channel asynchronous controllers (#1931) within the Universal Communications Chassis (#1930).

For the peripheral equipment, each HAC computer contains a TI-820 terminal for a systems console and a 10 Megabyte Disk unit (#4137). The other peripheral equipment is referenced by one of two Peripheral Control Switches (PCS) (#4906). One PCS contains a 300 cards-per-minute card reader (#4411), a nine-track 800 bits-per-inch magnetic tape unit (#4148), a TI-810 low-speed printer (#4228) and a Chromatics Terminal (#1999). The other PCS contains a WWVB receiver, an ISC 8001 Color CRT, a Teletype 40 printer (#4227) and another Chromatics Terminal (#1999).

3.1.1.2

HFC Microprocessor

The HFC microprocessor is a Motorola #6803 microprocessor with 4096 8-bit bytes of read only memory and 1152 8-bit bytes of random access memory. Communications with the HAC computer is maintained by the microprocessor's two external ACIA's, with accompanying line drivers and receivers to provide redundancy and switchable communications lines. Communications with the HC microprocessor is through the HFC microprocessor's serial input/output port. In addition, the HFC microprocessor requires a dead-man timer, as well as a dip switch, to tell the HFC which one he is. These two devices are accessed through the microprocessor's parallel input/output port.

3.1.1.3

HC Microprocessor

The HC microprocessor is a Motorola #6803 microprocessor with 2048 8-bit bytes of read only memory and 256 8-bit bytes of random access memory. Communications with the HFC microprocessor is maintained through the microprocessor's serial input/output port. Like the HFC microprocessor, the HC microprocessor uses a dead-man timer and dip switches, accessed through the microprocessor's parallel input/output

port. In addition, the HC uses an external parallel interface adapter (PIA), through which the HC interfaces with the azimuth and elevation incremental encoders. The motor drive signals also use this PIA.

3.1.1.4 Chromatics Intelligent Terminals

The graphics display systems consists of two identical Chromatics CG1999 intelligent color graphic terminals and their associated peripherals. These systems include user-programmable Z-80 microprocessors with 33K bytes program random-access memory (RAM), 18K bytes system program read-only memory (ROM), 128K bytes of screen refresh memory, 512x512 resolution, 19-inch diagonal, 8-color display CRT, dual 256K bytes floppy-disk drives, RS232C serial input/output Port, a light pen, and an alphanumeric keyboard with user-definable function keys.

3.1.2 Software/Firmware System Description

The HAC Software Design is located in Sections 3.2 and 3.3. The HFC Firmware design is located in Section 3.4, and HC Firmware design is located in Section 3.5, and the Chromatics Intelligent Terminal software design is located in Section 3.6.

3.1.2.1 HAC Software Description

3.1.2.1.1 Terminology

The software for the HAC computer consists of modules, tasks and submodules. The definition of each and the scheme used to identify them are as follows:

- a. Module - Six unique alpha characters that identify a specific function (i.e. SUNVEC for the sun vector generation function). The module name does not identify a piece of code, it represents a function of related tasks and submodules.
- b. Task - Three unique alpha characters that identify a specific task within a module. The task has code associated

with it either as a stand-alone program or a main submodule and a series of submodules. If the task calls a series of submodules, it will be considered the main submodule.

- c. Submodule - Six alpha characters, the first three equal to the task name which calls the submodule, and the second three are a unique description of the submodule. Submodules called by more than one task will be identified by the most frequent calling task.

3.1.2.1.2

Operating Modes

All tasks within the HAC will be executing under the MODCOMP MAXNET IV operating system. These tasks are subdivided into one of three operating modes: system initialization mode; synchronous mode; and the asynchronous mode.

The system initialization mode is composed of both an offline and a real-time portion. The offline portion is designed to initialize the disk resident data base files and provides initialization data to the Chromatics Terminals. An offline activity was elected because of time considerations and added flexibility for the operator. This portion of the initialization process may be done at the operator's convenience. The real-time system initialization mode consists of those tasks within the DBINIT module that execute in response to system boot-up. These tasks initialize the global common data base, CS control console and backup system, establishes tasks and system interfaces, and provides initialization and synchronization of the universal and local system time base. The system initialization tasks will no longer exist in main memory when the initialization function is complete.

The synchronous mode is initiated upon activation by the timing tasks within the MAXIVM module. This mode consists of those tasks which automatically operate every time frame and they may be activated by other tasks within the synchronous loop or self-activated by time delays. Synchronous processing is required by the FLDCOM, SUNVEC, MAXIVM, CMDPRC, ALARMS, and STATUS modules. The tasks required for synchronous processing in general will be given higher priorities than the asynchronous tasks in order to ensure that the synchronous processing is accomplished each time frame.

The asynchronous mode consists of those tasks which are activated only when processing is necessary. These tasks may be activated by any other task within the system, specifically when commands or status requests are entered or when the graphics software requires data for displays. The asynchronous tasks may emulate the synchronous tasks as long as there is an outstanding request for a specific task's function. For example, the sequencing task of the CMDPRC module will operate periodically as long as there is

an active command sequence within the system. As soon as all command sequences have been terminated, the command sequence task will be inactivated and only reactivated when required by another sequence command. Asynchronous processing is required by the MANMIF, MAXIVM, CMDPRC, ALARMS, STATUS, GRAPHC, BCSMOD and EXTINF modules.

3.1.2.1.3

Functional Overview

The functional overview of the system is presented in the following paragraphs (see figure 3.1-4).

The HAC software is readied for system operation through the initialization process. This process is broken into the offline initialization phase and the real-time initialization phase.

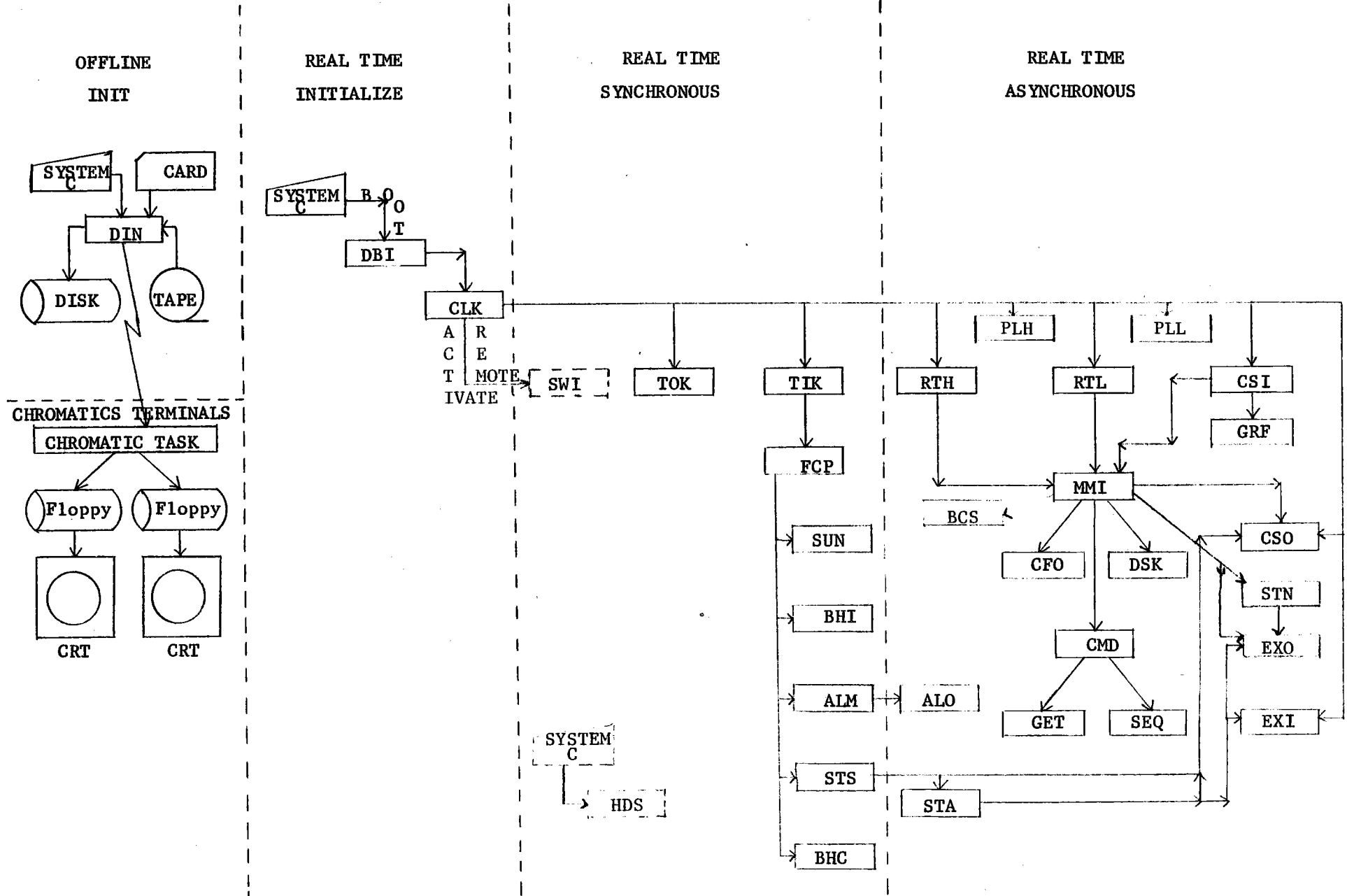
DIN, the offline initialization task of the Data Base Initialization module (DBINIT), is executed at the operator's convenience well in advance of anticipated field control. DIN edits source data from cards or magnetic tape to create the disk-resident data base and transmits required initialization data to the Chromatics graphic terminals. This Chromatics initialization function is handled offline due to the amount of time required for transmission.

Following the offline initialization of the disk-resident data base and graphics terminals, the real-time system is booted. The prescheduled task, DBI (DBINIT module), is automatically activated to initialize the real-time system including global common and system interfaces. Communications with the backup system is established at this time via the activation of the switching task, SWI, of the Operating Systems Modification module (MAXIVM).

In order to release memory for subsequent tasks, DBI activates a smaller task, CLK (DBINIT module), for real-time task establishment and activation, and then terminates. After all resident tasks have been established, CLK activates TOK and TIK, the timekeeping tasks of the MAXIVM module. Task TIK is subsequently self-resumed once per second to maintain the time in the global common data base and to initiate the synchronous loop for each time frame by activating the Field Communications Processor module (FLDCOM) task, FCP.

The task FCP transmits the sun position vector to the entire field. It then activates the sun position calculation task, SUN, of the Sun Vector module (SUNVEC). The task FCP then suspends itself, to be resumed at approximately 310 milliseconds into the frame for status polling. During this time, the SUN task calculates the sun position to be used in the next frame. It then terminates itself until activated in the next frame. At the time for status poll of the field, FCP is resumed by the operating system. The task polls the heliostat field, receives status information for up to one-eighth of the field, and stores the status data in the data base. FCP then performs preliminary checking of the status data to determine communications status. If alarm conditions are found, this data

HAC TASK ACTIVATION FLOW



13

Figure 3.1-4 HAC Task Activation Flow

is passed to the alarm task for alarm display. FCP then activates the HFC initialization task, BHI, of the Command Processor module (CMDPRC), the alarms monitoring task, ALM, of the Alarms Processor module (ALARMS), and the synchronous status task, STS, of the Status Display Processor module (STATUS), and suspends itself until time to transmit the operational commands to the field (approximately 725 milliseconds into the "frame").

BHI monitors the HFC status determining which HFCs, if any, require initialization data. It then begins the initialization process for the indicated HFCs and terminates itself until activated in the next frame.

ALM executes to monitor the status data and generates alarms for irregularly operating heliostats. If alarms are generated, ALM activates the alarms output task, ALO (ALARMS module), to display the alarms and terminates itself until activated next frame. If ALO is activated, it displays all successfully queued alarms and terminates itself.

STS executes to process the latest received status and report the field status on the CS Control console. It checks to determine if there is an operator status request pending, and if so, STS activates the asynchronous status task, STA (STATUS module), and then terminates itself until activated next frame. Task STA will determine the type of status requested, scan the status data array, format the status message, and transmit the status via the console output task, CSO, or the external interface task, EXO, of the External Interface Module (EXTINF), depending on the requesting source. It then terminates itself.

When the time to transmit operational commands arrives, FCP resumes itself and scans its buffer of commands ready for transmission and transmits any commands waiting. After command transmission, FCP checks the data base buffer to see if CMDPRC has additional commands to be transmitted. If so, FCP transfers these commands into its buffer of commands waiting for transmission. Task FCP then activates task BHC (CMDPRC module), and terminates itself until started by the timekeeping task at the start of the next "frame." Task BHC checks the array of 2048 possible heliostat commands. For corridor commands, BHC finds the physical corridor number assigned to an HC (range 1-4) and uses it as an index into the appropriate HFCS3G word to find the logical corridor number. If the commands are ready for transmission, and the HFC buffer of commands in the data base is available for a new command, the task will compress all similar HC commands into an HFC command (with the HC bit mask) and load the new command into the HFC command buffer. The task will then terminate itself.

The asynchronous sequence of tasks is also initiated by the start-up control task, CLK (DBINIT module). The communications tasks within EXTINF that establish and maintain communications with the OCS, DAS, CS Control Console, and Chromatics terminals are activated by CLK. EXI establishes and maintains communications with the OCS and DAS. EXI accepts commands and status requests.

from the OCS and status requests from the DAS, and passes these inputs to the MMI task (MANMIF module) for syntax checking and processing. EXO will transmit the resultant data from these requests back to the respective source, OCS or DAS. The console interface task, CSI (EXTINF module), accepts inputs from the CS Control Console and Chromatics terminals and routes the input to either the MANMIF module for commands, or to the Graphics Display Processor Module (GRAPHC) for graphic display status requests. CSO will transmit responses back to the CS control console. When the graphics processing task, GRF (GRAPHC module), receives a request for data, it determines the type of request and scans for the appropriate data in the status array created by the synchronous STS task. The status packet is formatted and held for transmission pending a request to send message from the CSI task. Once the data has been transmitted, GRF terminates itself.

The Central Receiver requires constant monitoring in order to prevent receiver damage. Consequently, CLK activates the receiver trip high task; RTH, and receiver trip low task, RTL (MAXIVM module), to monitor receiver status. This status consists of two signals, one the inverse of the other, which when reversed from normal, cause a DEFOCUS command to be transmitted to MMI.

When MMI receives a command from any source through a series of priority queues (the emergency command queue is highest priority), it checks each command for valid syntax and source. If an error occurs, the source of the command will be notified, and MMI will terminate itself until activated by another command. If the command is valid and requires disk updating, task DSK (MANMIF module) will be activated to process the command, and MMI will suspend itself. DSK processes all commands that only require updating of the disk data base. If the valid command requires a command file from disk to be started, MMI will activate task CFO (MANMIF module) to process the command file and then suspend itself. If the command is a status request, MMI will notify STA via the data base that status has been requested and then suspend itself. If the valid command requires communication with the field, MMI will build a data packet, activate task CMD (CMDPRC module) and then suspend itself. If the command is STHIWIND or RESTORE, MMI activates the STN task to notify the OCS of any HCs that could not respond. CMD will check the command for reasonableness and report via the data base an error code or a success ratio. MMI will report either the error or the success ratio to the operator. If an error occurred, CMD will resume task MMI and terminate. CMD will build the HC commands array and mark the commands ready for BHC to map into HFC commands. If the command requires a sequence for implementation, task SEQ (CMDPRC module) is activated to perform this processing. If the command requires data from disk to complete processing, CMD will activate task GET (CMDPRC module) to read the disk data. After CMD builds the necessary commands and activates the appropriate tasks, it resumes task MMI and terminates.

SEQ, once activated, will run periodically by setting timers for self-activation as long as there are sequences to process. The system allows up to 16 concurrent sequences, and SEQ is reactivated on an optimal basis to monitor one particular sequence per "visit" in a modified round-robin fashion. When there is no longer an active sequence, SEQ terminates itself.

The Backup HAC varies from the Prime HAC in the following ways:

- 1) The global common in the Backup HAC is initialized by task DBIBCK from the Prime. Prior to the completion of the data base initialization function in the Prime, task DBI activates DBIBCK, which transfers all initialized global common regions to the Backup. This is the only time some of these global regions get transferred.
- 2) After the data base initialization is complete, DBI activates task CLK, which remotely activates task SWI in the Backup. SWI is the controlling task for HAC failover. It activates SWIRG1 once each second to read the global common regions from the Prime into the Backup that are required to be current every second. SWI also activates SWIRG8 once every eight seconds to read the global common regions from the Prime into the Backup that are required to be current every eight seconds. SWI also monitors the Prime status and, if a failover condition exists, commands the Peripheral Control Switches (PCS) to the Backup; and then all SWI tasks are removed from the system. When the PCSs are commanded to the Backup, a signal from each PCS is sent to the corresponding 1930, switching the 1930's from the Prime's 1907 to the Backup's 1907.
- 3) The Heliostat Data Storage (HDS) task is activated by the operator in the Backup HAC when he wants to record HC data on tape. This task automatically exits the system upon HAC failover.

3.1.2.2

HFC Firmware Description (See Figure 3.4.1-1)

The HFC firmware interposes between the HAC and HCs. The HFC firmware performs command translation from HAC to HC, status collection and formatting from HC to HAC, corridor-walk sequencing, and emergency corridor-walk sequencing. HFC firmware consists of a set of foreground (interrupt-level) and background (noninterrupt) tasks.

HAC input and HAC output are two foreground tasks which do byte to/from message packet translation to allow the HFC background tasks to communicate on a message basis. HC input and HC output are the two foreground tasks which perform the same function for HC I/O. HFC timers is a foreground task which provides delay and dead-man timers of different granularities to the background tasks.

Three background tasks operate in a serial fashion under control of the main background task, CMDI. CMDI controls the operational mode of the HFC (restart, normal, or emergency corridor walk) according to the current mode and input stimulus, and processes all commands from the HAC. However, once each second, whether triggered by a sun/sync message from the HAC or by an internal timer during emergency corridor walk; CMDI activates the three other background tasks in order, to perform the main HFC functions. HC operations task is called first. HC operations commands the HCs, collects and formats command/response information, polls four HCs each second for status, and collects and formats the returned status. Second is the corridor walk calculation. The corridor walk calculation task updates the current corridor target points and handles corridor end conditions. The final task is emergency corridor walk sequencing. Emergency corridor walk sequencing normally does nothing; but during emergency corridor walk modes, it generates commands, in sequence, and checks the operation of the HCs in order to accomplish the emergency corridor walk. After these three tasks complete, CMDI is ready to accept any HAC commands again.

3.1.2.3

HC Firmware Description (See Figure 3.5.1-1)

HC firmware positions the heliostat to commanded or calculated positions based on direct position commands or target position coordinates. HC firmware consists of a set of foreground tasks (initiated by system interrupts) and background tasks. INIT is a background task activated by power up or in the event of communications loss. The primary function of INIT is to set the system operating devices to the proper state (i.e., motor power, line driver, and receiver enables and disables) and initialize system memory and status flags.

CALC is the second background task activated by the communications handler (SCIO) upon acceptance of a command requiring position calculation. The function of CALC is to determine actual azimuth and elevation positions based on current target, heliostat, and sun vectors. This involves vector addition, subtraction, and normalization routines, as well as cartesian-to-polar conversions utilizing the "cordic algorithm."

SYSCCLK is a foreground task activated by a hardware timer. The primary function of SYSCCLK is to assure activation of time critical events such as command response or motor control.

SCIO is a foreground task activated by the internal hardware transmitter or receiver. The function of SCIO is to handle communications between the HFC and the HC.

POINT is a foreground task activated by SYSCCLK. The function of POINT is to perform closed-loop control of the heliostat. This is accomplished by activating azimuth and elevation motors in either high or low speeds while acquiring data from incremental position encoders.

3.1.2.4

Chromatics Intelligent Terminal (GDC) Software Description

The GDC software module consists of foreground (interrupt driven) submodules, background (real time) processing submodules and offline (non-real time) preprocessing submodules.

The offline preprocessors (FFPREP and SEGPREP) are run whenever the field configuration data base on the HAC is updated. They manage the formatted data transferred from the HAC and conversion of that data into run-time efficient, unformatted, disk-resident files.

GDC online software includes foreground submodules which handle interrupts from the SIO link (RCV) and keyboard (KBD). These submodules provide data buffering and signal availability of data to the EXEC, which invokes appropriate background tasks to process the data. These background tasks include the interactive user/GDC dialogue controller (DIALOG) and real-time display updating tasks (FFDISP and SEGDISP). A collection of utility functions (UTIL) provides application-oriented numeric and character-handling capabilities to both offline and online processes.

3.2

HAC Software Design

The detail design of the modules and submodules of the HAC system software are presented in the following paragraphs.

3.2.1 Man-Machine Interface Module - (MANMIF)

3.2.1.1 Purpose

The Man-Machine Interface Module serves as an interface between the operator(s) and the HAC, through which an operator can implement commands for heliostat field control and monitoring and data base updating. MANMIF is comprised of a main module, MMI, and 29 submodules: MMIDEQ, MMIWRD, MMICLK, MMICMD, MMINUM, MMIHXD, MMIMAP, MMINON, MMISTR, MMIST2, MMIST3, MMIREF, MMIRSP, MMICRT, MMIERR, MMITST, MMLAID, MMIBCS, CFO, CFOPOS, CFOGET, CFOMSG, DSK, DSKBIA, DSKTRF, DSKINK, DSKERR, DSKBAK, DSKAIM, DSKUPD.

3.2.1.2 Requirements

3.2.1.2.1 Design Requirements

Section 3.1 of the Software/Firmware Functional Requirements Specification (12 June 1980) lists the following requirements which concern MANMIF:

- a. Control of up to 2048 heliostats in all modes required to operate the heliostats.

This requirement indicates the need for operator-entered commands to be interpreted, processed, and forwarded to the CMDPRC module for implementation.

- b. Monitor and display the operational status of all heliostats.

This requirement indicates the need for an operator status request to be interpreted, processed, and forwarded to the STATUS module for implementation.

- c. Detect, report, and respond to failures and irregularities.

This requirement indicates the need for operator-entered response to be interpreted, processed, and forwarded to the ALARMS module for further processing.

- d. Maintain a "Prime" and "Backup" system, as well as redundant processing in the "Backup" system in order to keep disk files up to date.

- e. Maintain safe beam control.

This requirement indicates a need to perform inclusion area checking of new aim-point data.

- f. Respond to receiver trip for emergency defocus.

This requirement indicates a need to format a DEFOCUS command to be forwarded to the CMDPRC module upon receiving a receiver trip.

- g. Maintain command/response protocol and data transfer with the OCS.

This requirement indicates a need to determine the source of a given command, build response messages appropriate to the command, and send the responses to the source of the command.

- h. Provide status data to the DAS.

This requirement indicates a need for operator-entered status requests from the DAS to be interpreted, processed, and forwarded to the STATUS module.

- i. Provide automatic control of beam characterization within the HAC.

This requirement indicates the need for operator-entered commands to be interpreted, processed, and forwarded to the BCSMOD module for implementation.

3.2.1.2.2 Derived Requirements

Section 3.2.1.1 of the Software/Firmware Functional Requirements Specification (12 June 1980) lists the following derived requirements for the MANMIF module:

- a. Accept commands from the CS control console, OCS, DAS, and CS graphics console, and read command files from disk;
- b. Decode command addresses for the Command Processor Module and convert commands to internal computer format;
- c. Check all commands for valid syntax;
- d. Generate error messages and route them to the appropriate output devices;
- e. Pass valid operational commands to the Command Processor Module;
- g. Log all commands on the console/printer with appropriate time stamps;
- h. Update a heliostat's bias disk file upon operator request; and
- i. Update aim-point array and perform inclusion-area checking.

Additionally, the MANMIF module must be able to:

- a. Generate operator-aid menus of various formats upon request;
- b. Control operation of execution of command files;
- c. Refresh the CS console CRT screen upon request;
- d. Generate messages responding to each command, indicating successful completion of the command, or syntax or other errors; and
- e. Exclude certain commands during emergency processing, or those from invalid sources.

A list of valid commands and their effects is given in Table 3.2.1-I, and the addressing formats are described in Table 3.2.1-II. It should be noted that at least the first four, and up to eight, of the letters of each command are necessary.

3.2.1.3 Design Approach

MANMIF is comprised of 31 submodules, including three tasks: MMI, DSK, and CFO. MMI is the main routine, and may activate DSK for any disk data updating or CFO for command file execution. MMI and its submodules perform syntax checking of the command string and build data packets for the CMDPRC, STATUS, and BCSSMOD modules. DSK and CFO handle all disk operations separately from the syntax checking and operational command processing.

VALUE OF KMAND	COMMAND	INPUT SOURCE	ADDRESSING	DESCRIPTION
1	<u>TRACK</u>	CS OCS	Heliostat Segment Wedge Ring Field Controller Arc	Directs heliostat(s) tracking the STANDBY position to track the target.
2	<u>INCREASE</u>	CS OCS	Segment Wedge Ring	Directs heliostat(s) tracking the STANDBY position to track the target. **The optional number address may be used with this command (see Table 3.2.1-II).
3	<u>LOAD</u>	CS	Heliostat Field Controller Field Arc	This command will cause download of initialization data to the heliostat controller.
4	<u>MARK</u>	CS	Heliostat Field Controller Segment Wedge Ring Field Arc	Directs heliostat(s) to move in both axes such that shafts pass reference marks on encoders. Used to determine accurate heliostat orientation prior to use.
5	<u>STANDBY</u>	CS OCS	Heliostat Segment Wedge Ring Field Controller Arc	Directs heliostat(s) tracking the target to track the STANDBY position.
6	<u>STOW</u>	CS OCS	Heliostat Segment Field Arc	Directs heliostat(s) to respective STOW positions
7	<u>POSITION</u>	CS	Heliostat Segment Wedge Ring Field Controller Arc	Directs heliostat(s) to commanded orientation. Beam safety is operator responsibility.
8	<u>ONLINE</u>		Heliostat Segment Wedge Ring Field Controller Arc	No heliostat movement results from this command. Allows heliostat(s) previously commanded OFFLINE to be commanded.

Table 3.2.1-I. Collector Subsystem Command List

VALUE OF KMAND	COMMAND	INPUT SOURCE	ADDRESSING	DESCRIPTION
9	<u>OFFLINE</u>	CS OCS	Heliostat Segment Wedge Ring Field Controller Arc	Directs heliostat(s) to remain in current position and accepts no further commands to these heliostats until ONLINE command entered.
10	<u>DECREASE</u>	CS	Segment Wedge Ring	Directs heliostat(s) tracking the target to track the STANDBY position. **The optional number address may be used with this command (see Table 3.2.1-II).
11	<u>BCSTRACK</u>	CS	Heliostat	Directs heliostat to track BCS target to which it is assigned.
12	<u>RETURN</u>	CS	Heliostat	Directs heliostat(s) in BCS mode to track the STANDBY point.
13	<u>WASH</u>	CS	Heliostat Arc	Directs heliostat(s) to go to their respective WASH orientation. This command may be unsafe if used during the period when the sun is present.
14	<u>ALT1STOW</u>	CS	Heliostat Segment Wedge Ring Field Controller Arc Field	Directs heliostat(s) to respective ALT1STOW positions
15	<u>ALT2STOW</u>	CS	Heliostat Segment Field Arc	Directs heliostat(s) to respective ALT2STOW elevations while maintaining last reported azimuth.
16	<u>RESTORE</u>	CS	Field	Restores aimpoint array assignment from saved configuration to track target and heliostat(s) that were tracking STANDBY points. Only heliostat(s) in TRACK or STANDBY modes will respond to this command.

Table 3.2.1-I. Collector Subsystem Command List (cont.)

VALUE OF KMAND	COMMAND	INPUT SOURCE	ADDRESSING	DESCRIPTION
17	<u>UNSTOW</u>	CS OCS	Heliostat Segment Wedge Ring Field Controller Field Arc	Directs heliostat(s) in one of STOW positions to track the STANDBY position.
18	<u>STHIWIND</u>	CS OCS GRAPHIC	Field	Directs all heliostats in the allowable modes to their respective STOW positions. No further commands allowed until RLHIWIND command is entered. Those heliostats doing corridor walk will be allowed to complete the corridor before responding.
19	<u>DEFOCUS</u>	CS OCS RS GRAPHIC	Field	Directs all heliostat(s) tracking the target or going to the target to track the STANDBY position. No further target tracking commands accepted until DEFRLSE command.
20	<u>AIMPOINT</u>	CS OCS	(Segment Field), NN (Array Number)	This command will assign aimpoint arrays for the segment(s) commanded. If any heliostats in commanded segment(s) are tracking the target, they will be redirected to the newly assigned aimpoint.
21	<u>HOLD</u>	CS	Heliostat	Directs a transitioning heliostat to maintain current orientation. Includes those heliostats doing corridor walk.
22	<u>SAVE</u>	CS	Field	No heliostat movement results from this command. Tracking configuration saved for use with restore command. Aimpoint array assignment also saved.
23	<u>RELWASH</u>	CS	Heliostat Arc	This command is the only way a heliostat may be released from WASH mode. No movement occurs, and the heliostat is placed in the DIRECTED POSITION mode.

Table 3.2.1-I. Collector Subsystem Command List (cont.)

VALUE OF KMAND	COMMAND	INPUT SOURCE	ADDRESSING	DESCRIPTION
24	<u>ESTANDBY</u>	CS OCS	Heliostat Field Controller Segment Wedge Ring Field Arc	This command allows an operator to command heliostats tracking the target to the STANDBY mode after power loss and initialization.
25	<u>ESTOW</u>	CS OCS	Heliostat Field Controller Segment Wedge Ring Field Arc	This command allows an operator to command heliostats on corridors directly to STOW with no beam safety after power loss and initialization.
26	<u>RLHIWIND</u>	CS	Field	No heliostat movement results from this command. This command allows heliostat(s) affected by STHIWIND command to be commanded again.
27	<u>DEFRLSE</u>	CS	Field	No heliostat movement results from this command. This command allows heliostats which were put in STANDBY mode due to DEFOCUS command to be commanded to track again. This command will not be accepted if the Receiver Trip signal has not returned to normal.
28	<u>UPAIM</u>	CS	NN (Array number to be updated)	This command will replace the array specified with a completely new array from magnetic tape.
29	<u>UPBIAS</u>	CS	H/NNNN, AZ, EL NNNN = heliostat AZ - Azimuth bias in HEX EL - Elevation bias in HEX	This command updates the encoder bias file on disc.
30	<u>STATUS</u>	CS OCS DAS	Heliostat Mode Ring Field	Allows request for STATUS in one of four forms: 1) individual heliostat STATUS including mode, orientation, commanded mode, and position compare flag; 2) list of the heliostats in requested mode; 3) number of heliostats within ring which are in the TRACK and STANDBY modes, and 4) number of heliostats in all modes.

Table 3.2.1-I Collector Subsystem Command List (cont.)

VALUE OF KMAND	COMMAND	INPUT SOURCE	ADDRESSING	DESCRIPTION
31	<u>HELP</u>	CS	Blank, Command or Address	No address will give the operator a list of all commands available. By using an individual command, the information will contain the format and valid addressing.
32	<u>CFSTART</u>	CS OCS	FFF (File name)	This command will cause the execution of a particular command file to begin.
33	<u>CFABORT</u>	CS	N/A	This command will stop the execution of a particular command file.
34	<u>CFWAIT</u>	CS	SSSS (seconds)	This command will hold the execution of a particular command file for the specified number of seconds.
35	<u>REFRESH</u>	CS	N/A	Allows the CS control console to be completely refreshed.
36	<u>AIMLIST</u>	N/A	N/A	Not implemented.
37	<u>SAVDOS</u>	N/A	N/A	Not implemented.
38	<u>TEST</u>	CS OCS DAS	Test N/Z	Provides access to "Test" commands.
39	<u>BCSSTART</u>	CS	None	Start BCS task.
40	<u>BCSXTAPE</u>	CS	None	Read data tape.
41	<u>BCSFILE</u>	CS	File Number (1 to 3)	Choose BCS file number.
42	<u>BCSDISPL</u>	CS	ON/OFF	Display ON or OFF
43	<u>BCSRANGE</u>	CS	N1, N2 (N1, N2: One to 60)	Choose range of HCS in file.
44	<u>BCSMEAS</u>	CS	None	Initiate measurement processing.
45	<u>BCSINPUT</u>	CS	HC number	Initiate manual input.
46	<u>BCSOFFS</u>	CS	Date, Time, Offset Values	Input offset values.

TABLE 3.2.1-I. Collector Subsystem Command List (cont.)

VALUE OF KMAND	COMMAND	INPUT SOURCE	ADDRESSING	DESCRIPTION
47	<u>BCSCALC</u>	CS	"INP", "ALL" or HC Number	Calculate new biases
48	<u>BCSUPD</u>	CS	Y/N	Update data base.
49	<u>BCSEND</u>	CS	None	Stop BCS task normally.
50	<u>BCSABORT</u>	CS	None	Abort BCS task (abnormally).

Table 3.2.1-I. Collector Subsystem Command List (cont.)

COLLECTOR SUBSYSTEM COMMAND ADDRESSING FORMAT

ADDRESS	FORMAT
HELIOSTAT (H)	H/NNNN,NNNN...NNNN WHERE: NNNN IS THE HELIOSTAT NUMBER EXAMPLE: STOW H/2901
HELIOSTAT FIELD CONTROLLER (F)	F/NN,NN...NN WHERE: NN IS THE FIELD CONTROLLER NUMBER EXAMPLE: UNSTOW F/02
SEGMENT (S)	(XX)/S/NNN,NNN...NNN WHERE: XX IS THE (OPTIONAL) NUMBER OF HELIOSTATS PER SEGMENT. IF NOT INCLUDED THE WHOLE SEGMENT IS ASSUMED. NNN IS THE SEGMENT NUMBER EXAMPLE: INCREASE 10/S/302
WEDGE (W)	(SS)/W/NN,NN...NN WHERE: XX IS THE (OPTIONAL) NUMBER OF HELIOSTATS REQUIRED OUT OF EACH SEGMENT IN THE WEDGE. IF NOT INCLUDED THE WHOLE WEDGE IS ASSUMED. NN IS THE WEDGE NUMBER EXAMPLE: DECREASE 05/W/01
RING (R)	(XX)/R/N,N...N WHERE: XX IS THE (OPTIONAL) NUMBER OF HELIOSTATS REQUIRED OUT OF EACH SEGMENT IN THE RING. IF NOT INCLUDED THE WHOLE RING IS ASSUMED. N IS THE RING NUMBER EXAMPLE: INCREASE 10/R/5
ARC (A)	A/NNNN,NNNN/NNNN,NNNN/.../NNNN,NNNN WHERE: THE FIRST NNNN IS THE BEGINNING HELIOSTAT NUMBER. THE SECOND NNNN IS THE ENDING HELIOSTAT NUMBER. NOTE: THESE NUMBERS ARE EITHER EVEN INCLUSIVE OR ODD INCLUSIVE. EXAMPLE: WASH A/2901,2909
FIELD (ALL)	ALL THE WHOLE FIELD.
MODE (M)	TRK - TRACK TRN - TRANSITION STB - STANDBY WSH - WASH BCS - BCS DPO - DIRECTED POSITION STO - STOW OFF - OFFLINE AL1 - ALT1STOW MRK - MARK AL2 - ALT2STOW INI - INIT EXAMPLE: STAT M/TRK

TABLE 3.2.1-II. Collector Subsystem Command Addressing Format

3.2.1.3.1 Functional Allocations

The MANMIF modules have the following functions:

- a. MMI - Controlling task. Activates CMD and ALM.
- b. MMIDEQ - Dequeues ASCII command string, determines source, writes message to printer.
- c. MMIWRD - Divides the ASCII command string into "words", (groups of ASCII characters separated by delimiters: comma, blank, or slash), sets pointer array, blanks unused portion of string.
- d. MMICLK - Checks for presence of valid commands and addresses. Checks for allowability of command with its source and lock flags, checks address allowability with the commands. Optionally performs the checking functions for MMIAID.
- e. MMICMD - Checks the ASCII string for a valid command.
- f. MMINUM - Decodes ASCII numbers to binary format. Optionally range checks the following parameters: HC, HFC, Segment, Wedge, and Ring numbers.

Output from this routine can be mapped to HFC-HC numbers by MMIMAP without further range checking.
- g. MMIHXD - Decodes ASCII representations of hexadecimal or floating point numbers.
- h. MMIMAP - Maps addressing parameters (Segment numbers, etc.) into HFC-HC numbers for CMDPRC and others.
- i. MMINON - A controlling subroutine for nonoperational commands (KMAND greater than 25). Activates DSK and CFO.
- j. MMISTR - Processes the HC, Ring, Mode and Field Status request command string. Sets ISTATG and activates STA.
- k. MMIST2 - Processes the "H1", "H2", and "F2" Test/Debug Status requests.
- l. MMIST3 - Processes the "ER" Test/Debug Status requests.
- m. MMIREF - Clears the CS console and redraws the Alarms and Status areas.
- n. MMIRSP - Interprets data passed back by the CMDPRC module to format response messages.

3.2.1.3.1 Functional Allocations (Continued)

- o. MMICRT - Enqueues a message to the CS console and/or writes to logger and/or writes to OCS and/or DAS core devices.
- p. MMIERR - Formats error messages.
- q. MMITST - Processes TEST commands.
- r. MMIAID - Writes a list of commands and addresses to the logger, or optionally provides detailed information on any given command or address. Uses same data base as MMICHTK so that the messages must reflect the true checking.
- s. MMIBCS - Processes command string for the BCSMOD module and enqueues them to BCS.

The disk operation submodules are:

- a. DSK - Controlling task.
- b. DSKAIM - Reads card type four from mag tape, checks values.
- c. DSKINK - Performs inclusion area check of aimpoint data.
- d. DSKTRF - Transforms aimpoint data from the Lambert grid system to internal format.
- e. DSKBIA - Updates the HCB (Biases) file.
- f. DSKERR - Outputs error messages to CS console and logger.
- g. DSKBAK - Updates the AIM file from Scratch file data.
- h. DSKUPD - Updates the Scratch file as data is checked.

The Command File operation submodules are:

- a. CFO - Controlling task.
- b. CFOPOS - Positions on the desired SED file.
- c. CFOGET - Reads a Command File record, decode wait time; enqueues message to MANMIF.
- d. CFOMSG - Outputs messages to CS console and logger.

See Figure 3.2.1-1 for the MANMIF hierarchical overview.
See Figure 3.2.1-2 for the MMI functional overview and
3.2.1-3 for the DSK and CFO functional overview.

3.2.1.3.2 Resource Budgets

- a. MANMIF's three tasks have the following memory requirements (estimated):
 1. MMI - 10K 16-bit words.
 2. DSK - 6K 16-bit words.
 3. CFO - 4K 16-bit words.
- b. Disk and magnetic tape utilization:
 1. MMI - None.
 2. DSK - Access to the heliostat aim point array file, the heliostat bias file, a scratch disk file, and access to magnetic tape.
 3. CFO - Access to the command file disk file.
- c. Other:
 1. MMI - Core devices, logging printer.
 2. DSK - Logging printer.
 3. CFO - Logging printer.

3.2.1.4 Design Description

3.2.1.4.1 Module Structure

The MANMIF module is composed of the submodules briefly described in sections 3.2.1.3 and 3.2.1.3.1, above, and described in more detail in sections 3.2.1.4.1.1 through 3.2.1.4.1.30.4, below.

3.2.1.4.1.1 Submodule I - MMI

3.2.1.4.1.1.1 - Description

- a. Language used - FORTRAN.
- b. How invoked - Activated by EXTINF and CFO via enqueue.
- c. Constraints and limitations - The operator-entered ASCII command string must follow the syntax rules described below. Additionally, scanning of the string ends when two delimiters in a row are encountered.
- d. Processing

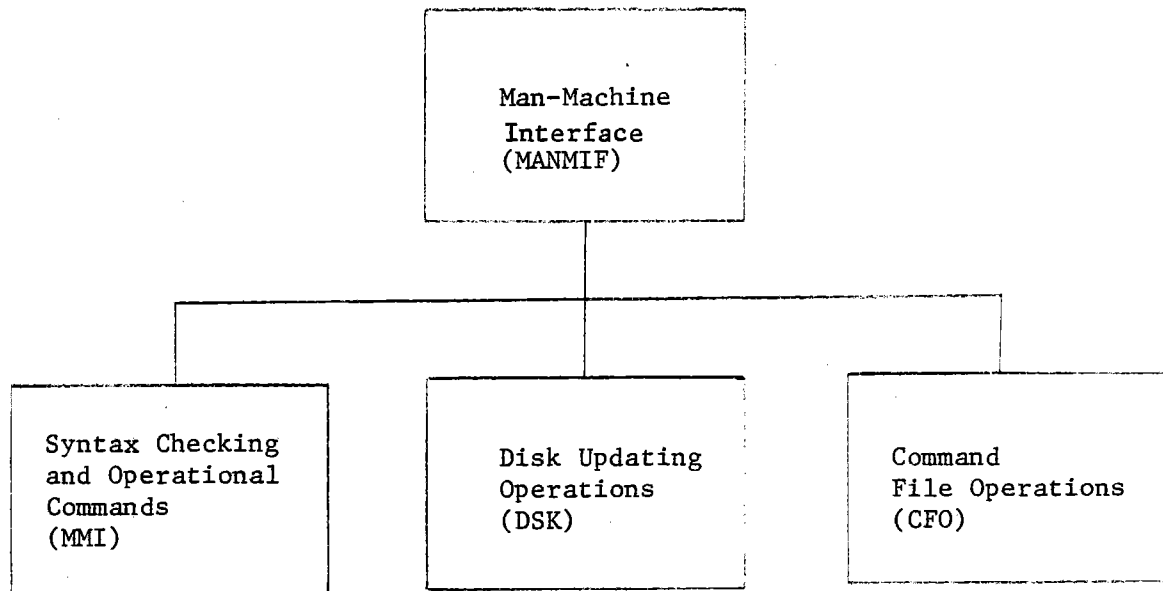


Figure 3.2.1-1 MANMIF Hierarchical Overview

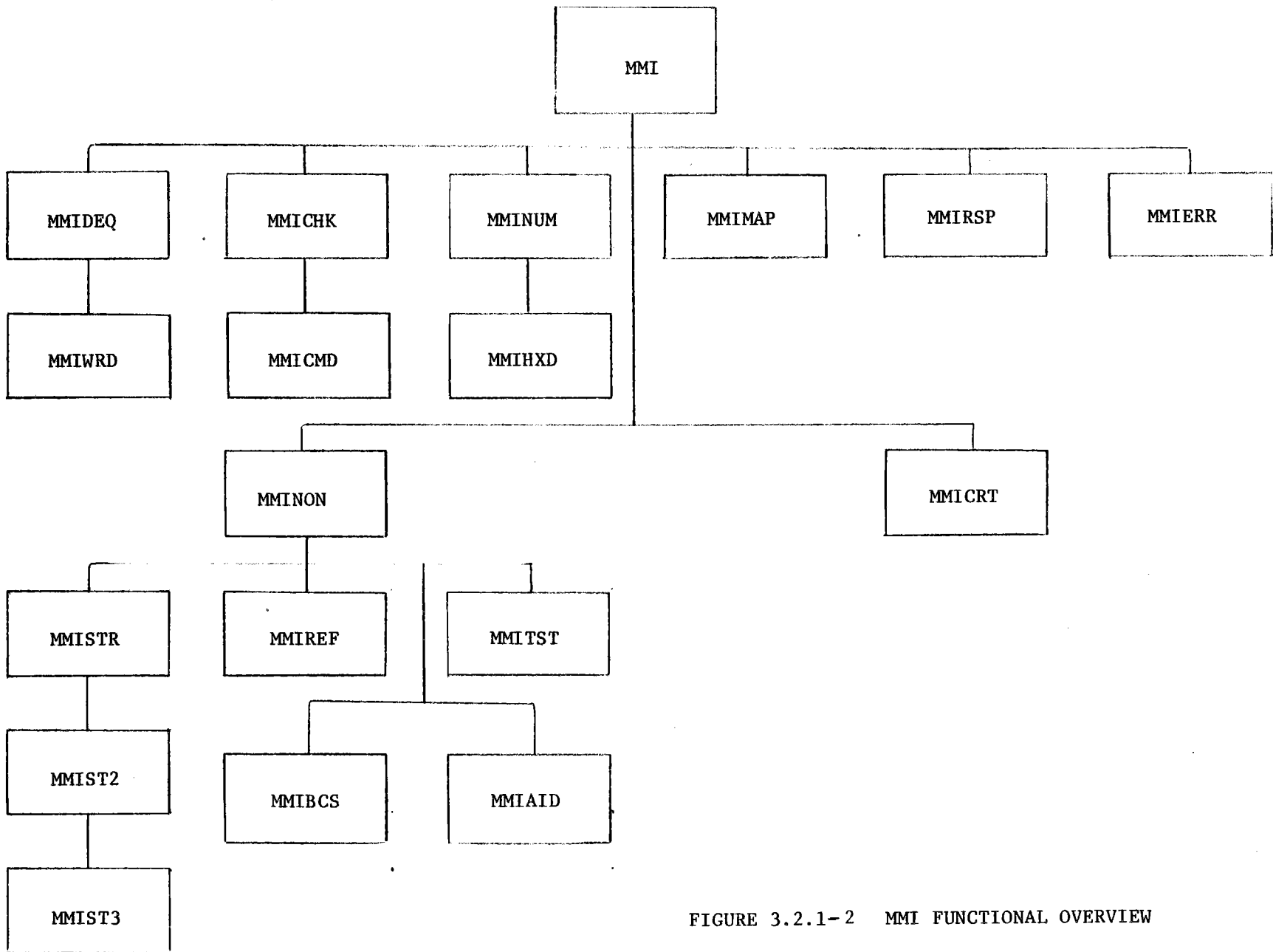


FIGURE 3.2.1-2 MMI FUNCTIONAL OVERVIEW

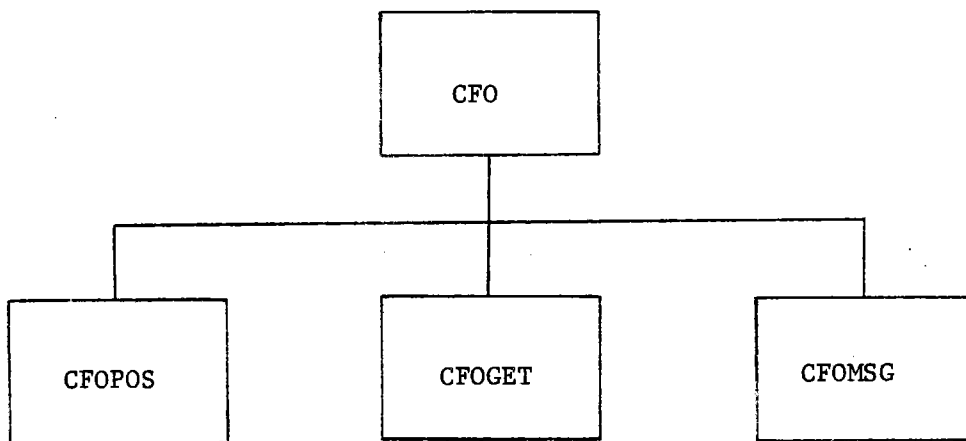
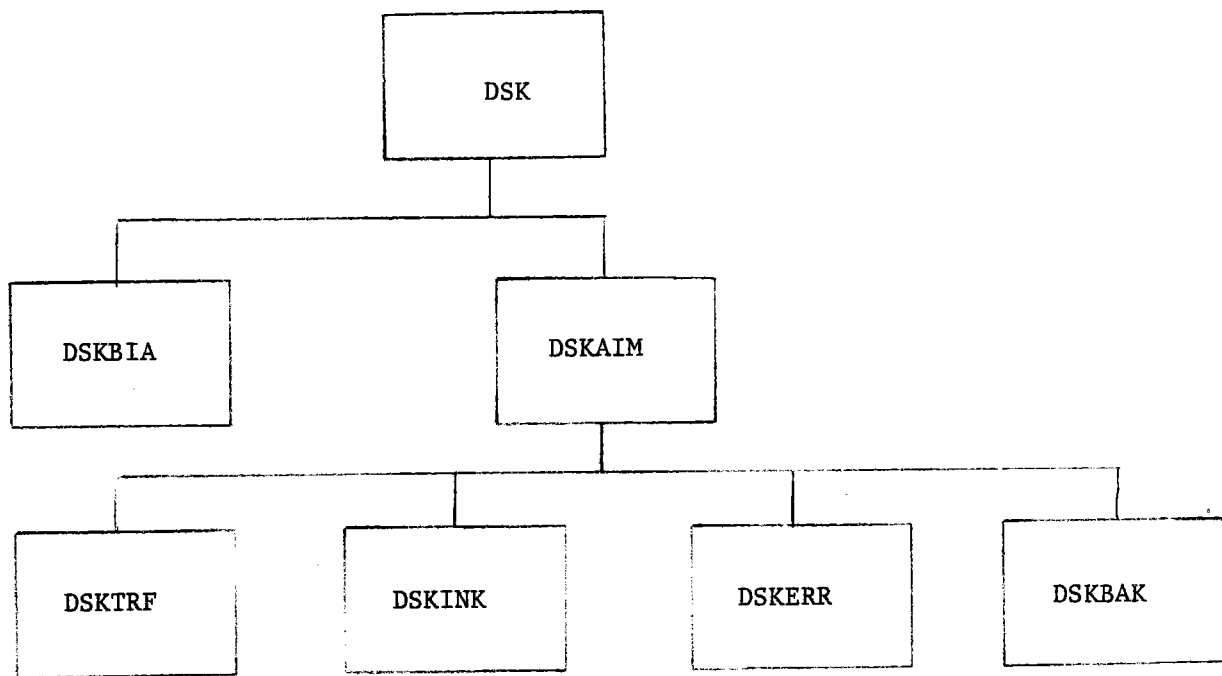


FIGURE 3.2.1-3 DSK AND CFO FUNCTIONAL OVERVIEWS

d. Processing (cont.)

1. Upon activation by CSI, CFO, or EXI initializes some local + global common.
2. MMIDEQ is called to dequeue the message.
3. Check for alarm response: "!!". Activate ALO if present.
4. MMICLK is called to determine which command was entered.
5. Check for "TEST" command; if so, go to 24.
6. Check for command error.
7. Check for "nonoperational" commands (KMAND>25). If so, go to 24.
8. Check for STHI or DEFO commands. If not, go to 12.
9. Check for presence of address. If not, go to 11.
10. MMICLK is called to check for "ALL".
11. If STHI, set LOKSTG=1. If DEFO, set LOKDEG=1. Set CPPG for CMD, branch to step 26.
12. Call MMICLK to process the addressing parameter.
13. Check for possible optional number; if not, go to 15.
14. Decode the number set CPPG(2), and go to step 12.
15. Set CPPG(2) for CMD.
16. If not "POSITION", go to 18.
17. Decode the AZ/EL words, store in CPPG(3,4).
18. If not "AIMPOINT", go to 20.
19. Call MMINUM to decode the Aimpoint number. Check AIMOKG, load Aimpoint number into CPPG(3).
20. If "ALL" address, go to step 26.

d. Processing (cont.)

21. Call MMINUM to decode the addressing parameters.
22. Call MMIMAP to map the addressing numbers to HFC-HC numbers for CMD.
23. Go to step 26.
24. Call MMINON to process "nonoperational" commands.
25. Go to step 29.
26. Activate STN if SAVE or STHI command.
27. Activate CMD, hold until resumed by CMD.
28. Call MMIRSP to report success rates.
29. Write EOF to logger, EXIT.

- e. Error Messages and Recovery - Syntax checking of the command string is done from left to right (except for POSITION or AIMPOINT). If a syntax error is encountered, IERR is set appropriately and control passed to MMIERR for reporting. Other errors (invalid source, emergency locks set, etc.) are also reported by MMIERR.

CMDPRC will set global word CPPRTG to a non-zero value if it is unable to implement a command, and these errors are also reported by MMIERR.

See Table 3.2.1-III for a list of reported errors.

3.2.1.4.1.1.2

Data, Logic, and Command Paths

Input/Output Data -- See Figures 3.2.1-4 through 3.2.1-7 and Tables 3.2.1-IV through 3.2.1-VI.

3.2.1.4.1.1.3

Internal Data Description

Miscellaneous local pointers and counters.

3.2.1.4.1.1.4

Flowchart

See Figure 3.2.1-8 for the MMI flowchart.

SYNTAX ERRORS (Message contains the part of the
string found to be invalid)

1. DUPLICATE NUMBERS
2. PARAMETER MISSING
3. INVALID COMMAND
4. INVALID ADDRESS
5. ADDRESS NOT ALLOWED
6. INVALID NUMERIC
7. NUMBER OUT OF RANGE
8. NUMBER OF CHARACTERS
9. INVALID MODE
10. INVALID FILE NAME

NON-SYNTAX ERRORS

1. INVALID SOURCE OF COMMAND
2. NOT ALLOWED (DEFOCUS)
3. NOT ALLOWED (STHIWIND)
4. NOT ALLOWED (EMERG SEQUENCE)
5. COMMAND FILE NOT ACTIVE
6. DEFOCUS NOT GIVEN
7. STHIWIND NOT GIVEN
8. BCS NOT ACTIVE
9. BCS ALREADY ACTIVE
10. COMMAND FILE ALREADY ACTIVE
11. COMMAND NOT AVAILABLE
12. NO HCS IN CORRECT MODE
13. ALL HCS OFFLINE
14. NO HCS INSTALLED
15. ONLY 1 HC/BCS TARGET
16. ONLY 16 SEQUENCES POSSIBLE
17. EMERGENCY SEQUENCE ACTIVE
18. AIMPOINT ARRAY NO GOOD
19. RECEIVER NOT READY
20. POWER HAS NOT BEEN RESTORED

Table 3.2.1-III MANMIF Errors

MMI DEQ WRD CHK CMD NUM HXD MAP NON STR ST2 ST3 REF RSP CRT ERR TST AID BCS

MMI	X																		
MMIDEQ	c	X																	
MMIWRD		C	X																
MMICLK	C			X			C	C										C	
MMICMD				C	X													C	
MMINUM	C				X		C	C											C
MMIHXD	C				C	X	C		C										
MMIMAP	C						X	C	C									C	
MMINON	C							X											
MMISTR							C	X											
MMIST2								C	X										
MMIST3									C	X									
MMIREF								C			X								
MMIRSP	C										X								
MMICRT		C					C			C		C	X	C	C	C	C	?	
MMIERR	C													X					?
MMITST							C								X				
MMIAID							C											X	
MMIBCS							C												X
DEQU/ENQ		C	Deq										Enq						C
Activates:																			
CMD	A																		
DSK							A												
STA								A	A	A									
CFO							A												
BCS																			A

37

Figure 3.2.1-4

MMI SUBMODULE REFERENCING CHART

NOT
USED

MMI DEG WRD CHK CMD NUM HXD MAP NON STR ST2 ST3 REF RSP CRT ERR TST AID BCS?

	MMI	DEG	WRD	CHK	CMD	NUM	HXD	MAP	NON	STR	ST2	ST3	REF	RSP	CRT	ERR	TST	AID	BCS?
COMIN1								X											X
COMIN2								X	X	X									
COMIN3	X					X		X	X	X				X					X
COMIN4	X			X					X	X	X	X		X					
COM1S1																			
COM1S2																			
COM1S3																			
COM1S4																			
COM1S5	X	X		X					X					X					X
COM8S1																			
COM8S2																			
COMQUE	X	X													X				X
MMICOM	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X	X	X
STRCOM										X	X	X							
COMAIZ					X														X
COMAID				X															X
MMICOL														X	X	X	X	X	X

38

Figure 3.2.1-5

MMI GLOBAL COMMON AREA REFERENCE CHART

MMICOM	MMI	MMI	MMI	MMI	MMI	MMI	MMI	MMI	MMI	MMI	MMI	MMI	MMI	MMI	MMI	MMI	MMI	MMI
	DEG	WRD	CHK	CMD	NUM	HXD	MAP	NON	STR	ST2	ST3	REF	RSP	CRT	ERR	TST	AID	BCS
NUMARY	I				0		I	I	I				0			I		
NPBLOK							0						I/O					
CHRPNT		0			I	I			I	I	I		0		I	I		
A2	I	I/O	I/O										0					
A1		0	0	I	I	I	I		I	I/O	I				I/O		I	
BUFFY		0						0			0		0	I/O	0	0	0	
NWORD	I		0	I				I	I	I					I	I	I	
KMAND	I/O			0				I								I		
ADDRES	I/O		0					I		I						I/O		
IERR	I	0		0	0	0		I/O	I/O	I/O	0		0		I/O	I/O		
ONWORD	0			I	0	I		0	0	0	0				I/O	0		
SOURCE	I	0		I														
NADCHK				I												0	I	
NSOCHK				I												0	I	
LP															I			
BLN																		
CS		I																
TEST	I															I/O		
JODY		0						0			0		0	I/O	0	0	0	
WER2GO		0							I					I				
MISFAG				I														0
OP2X1F	0												0	I	0			

Figure 3.2.1-7

MMICOM INPUT/OUTPUT REFERENCE CHART

Description of MMI Local Common COMAI2

KMANDS(3,50) - CAN code representation of allowable commands
(See table 3.2.1-I).
Each (8 character) command used 3 elements.

Description of MMI Local Common COMAID

ALL - CAN coded "ALL".

ARGU(6) - ASGII addresses: "H", "A", "F", "S", "W", "R"

OKSTUF(11) - Filled by MMICLK and used by MMIAID as follows:

OKSTUF(I) ≠ ∅

I MEANING (for command number = MISFAG)

1	"H"	Address	Allowed
2	"A"	"	"
3	"F"	"	"
4	"S"	"	"
5	"W"	"	"
6	"R"	"	"
7	"ALL"	"	"

8	COMMAND	ALLOWED	FROM	FILE
9	"	"	"	DAS
10	"	"	"	OCS
11	"	"	"	CS

Description of MMI Local Common MMICOL

Contains ASCII color codes for ISC 8001
(Foreground RED, GREEN, BLUE, and WHITE:
Background on GREEN or RED).

TABLE 3.2.1-IV

MMI LOCAL COMMON AREAS

DESCRIPTION OF MMICOM: (All 16-Bit Integer)

- NUMARY(40) - Array of binary numbers decoded from I/P ASCII string.
Set By: MMINUM, MMIRSP
Used By: MMI, MMIMAP, MMINON, MMISTR, and MMITST
- NPBLOK(40) - Array indicating the number of "blocks of HCMNPG to use for each addressing parameter.
Set By: MMIMAP, MMIRSP
Used By: MMIRSP
- CHRPNT(40) - Byte count (index to A1) of first character of the "word" used as the index.
Set By: MMIWRD, MMIRSP
Used By: MMINUM, MMIHXD, MMISTR, MMIST2, MMIST3, MMIERR, MMITST, MMIBCS.
- A2(40) - A2 formatted (2 characters per 16-bit word) input command string.
Set By: MMIDEQ, MMIWRD, MMIRSP
Used By: MMI
- A1(80) - A1 formatted (1 character per 16-bit word) input command string.
Set By: MMIDEQ, MMIWRD, MMIST2, MMIERR
Used By: MMICLK, MMICMD, MMINUM, MMIHXD, MMISTR, MMIST2, MMIST3, MMIERR, MMIAID.

TABLE 3.2.1-V

MMICOM DESCRIPTION

MMICOM

Buffy(40) -	O/P message buffer, A2 format. Set By: MMIDEQ, MMINON, MMIST3, MMIRSP, MMICRT, MMIERR, MMITST, MMIAID Used By: MMICRT
NWORD -	Number of "words" in command string. Set By: MMIWRD Used By: MMI, MMICHK, MMIERR, MMITST, MMIAID, MMISTR, MMINON, MMIST2.
KMAND -	Command number being processed. Set By: MMICHK, MMI, MMIBCS Used By: MMICHK, MMI, MMINON, MMITST, MMIBCS.
ADDRES -	Address type (H, A, F, S, W, R, ALL; 1 7) Set By: MMICHK, MMI, MMITST Used By: MMICHK, MMI, MMIMAP, MMISTR, MMITST
IERR -	Error Number/Flag Set By: MMIDEQ, MMICHK, MMINUM, MMIHXD, MMIMAP, MMINON, MMISTR, MMIST2, MMIST3, MMIRSP, MMIERR, MMITST Used By: MMI, MMINON, MMISTR, MMIST2, MMIERR, MMITST
ONWORD -	ASCII "word" in command string being processed. Set By: MMI, MMINUM, MMINON, MMISTR, MMIST2, MMIST3, MMIERR, MMIAID Used By: MMICHK, MMIHXD, MMIERR

TABLE 3.2.1-V (Cont'd.)

MMICOM DESCRIPTION

MMICOM

SOURCE -	Numeric code for source of command:	1 Command File 2 DAS 3 OCS 4 CS 5 Graphics 6 Receiver trip
	Set By:	MMIDEQ
	Used By:	MMI, MMICHK
NADCHK -	Address check flag:	Ø = check 1 = no check
	Set By:	MMITST
	Used By:	MMICHK, MMIAID
NSOCHK -	Source check flag:	Ø = check = source, no check from that source
	Set By:	MMITST
	Used By:	MMICHK, MMIAID
LP -	Line printer unit number (6ØØ)	
	Used By:	MMICRT
TEST -	Test command indicator:	1 = test commands allowed 0 = test commands not allowed
	Set By:	MMITST
	Used By:	MMI, MMITST
JODY -	Byte count of message encoded into BUFFY	
	Set By:	(See BUFFY)
	Used By:	MMICRT
WER2GO -	Where to send message output.	
	Set By:	MMIDEQ
	Used By:	MMISTR, MMICRT
MISFAG -	Indicates call by MMIAID.	
	Set By:	MMIAID
	Used By:	MMICHK
OP2XIF -	Indicates to MMICRT that this message should go to WER2GO.	
	Set By:	MMI, MMIRSP, MMIERR
	Used By:	MMICRT

TABLE 3.2.1-V (Cont'd.)

MMICOM DESCRIPTION

STRCOM

CANNERR(30) -- Can coded error MNEMONICS in "STATUS ER AAA" where

CANNERR = (@AAA)	HC ERROR	ER
MOT	No motion	1
CM1	Comm error 1.	2
NCE	? ? Elevation	3
NCA	? ? AZ	4
MCD		5
XCD		6
TIM	Time-Out	7
STU	Not Avail "STH2"	8
NSV	Not Avail "SAVE"	9
UMK	Unmarked	10
OPO		11
SAL		12
CM2	Comm error 2	13
NIN	Not installed	14
B00	But number in HFCS2G to check	15
B01		16
B02		17
B03		18
B04		19
B05		20
B06		21
B07		22
B08		23
B09		24
B10		25
B11		26
B12		27
B13		28
B14		29
B15		30

MASKER(30) -- MASK to use with HCST1G, HCST2G, HCST3G:
 HCST(HFCHC) and MASKER(ER) = MMC HFCHC has
 characteristic ER

TABLE 3.2.1-VI

STRCOM DESCRIPTION

NUMCAN - Dimension of CANERR + MASKER = 30.
 HEL - Can-coded: @ HEL - help memnomic:
 "STAT_ΔER_ΔHEL".
 STAR - ASCII "*".
 AITEMP(4) - Temporary AI formmated ASCII buffer.
 ERRNUM - Index to CANNERR & MASKER.
 ICHAR - Character pointer to word.
 NCHAR - Number of characters in word.
 MASK - Value for STAMCH to logically AND
 with HCST1G, HCST2G or ????. See MATCH.
 MATCH - Matched with result of above ANDing. A Match
 means that it has the desired characteristic.
 IP, IC, IB - Misc. pointers.
 CANDID - Can coded command input.
 Matched against CANERR.
 NUMOUT - Result of DECODE.
 MASK1(4) - ASCII 1 to decode for MASK.
 MATH1(4) - ASCII " " " MATCH.
 CANMOD - Can-coded three character mode.
 RET -
 FILCAN - Can-coded: @ LMbbbb.
 STAPRT -
 STACAN - Can-coded: @ STAbbbb.
 MODES(12) - Can codes for "STAT M AAA".
 See Table 3.2.1-II "MODE".

TABLE 3.2.1-VI (Cont'd.)

STRCOM DESCRIPTION

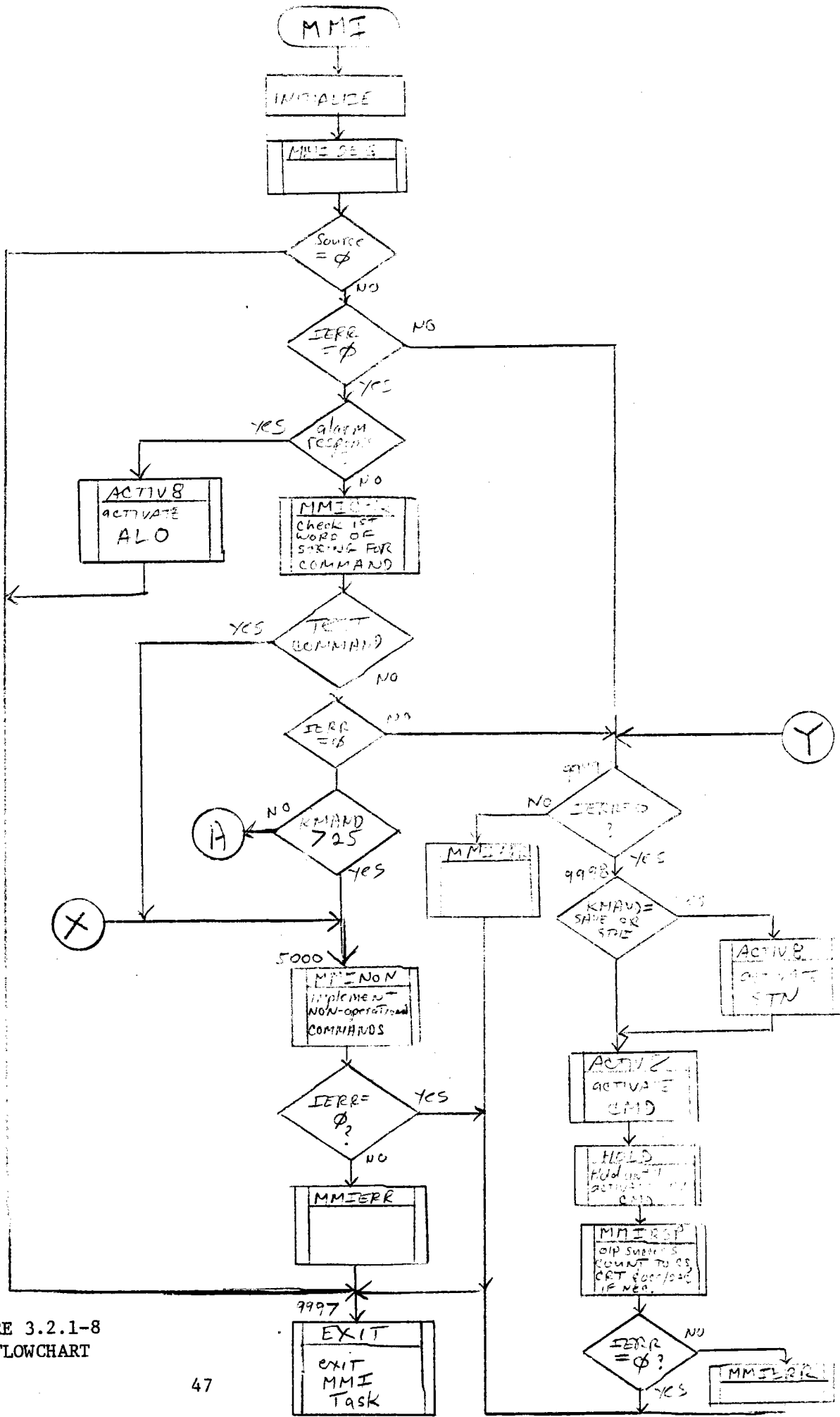


FIGURE 3.2.1-8
MMI FLOWCHART

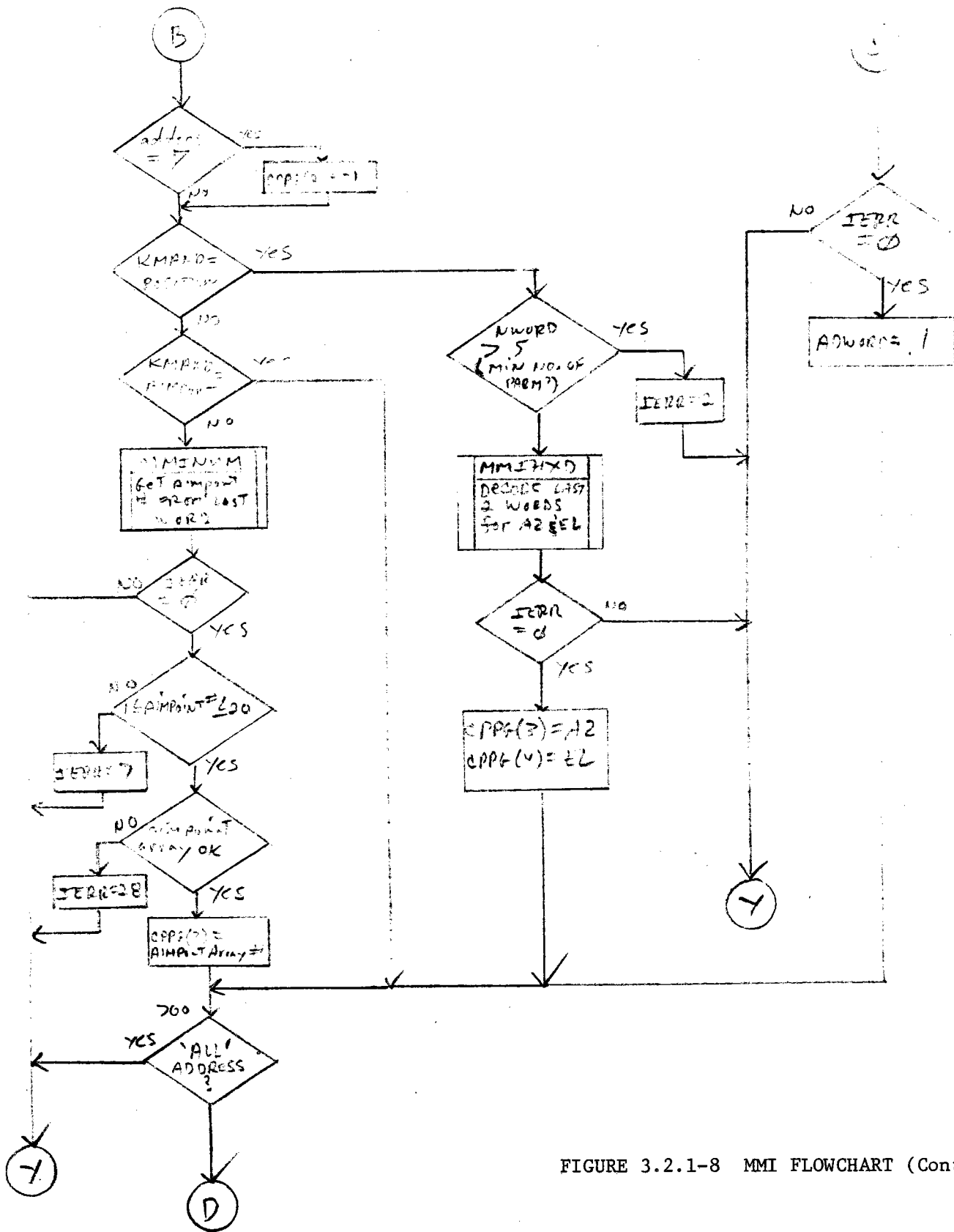


FIGURE 3.2.1-8 MMI FLOWCHART (Continued)

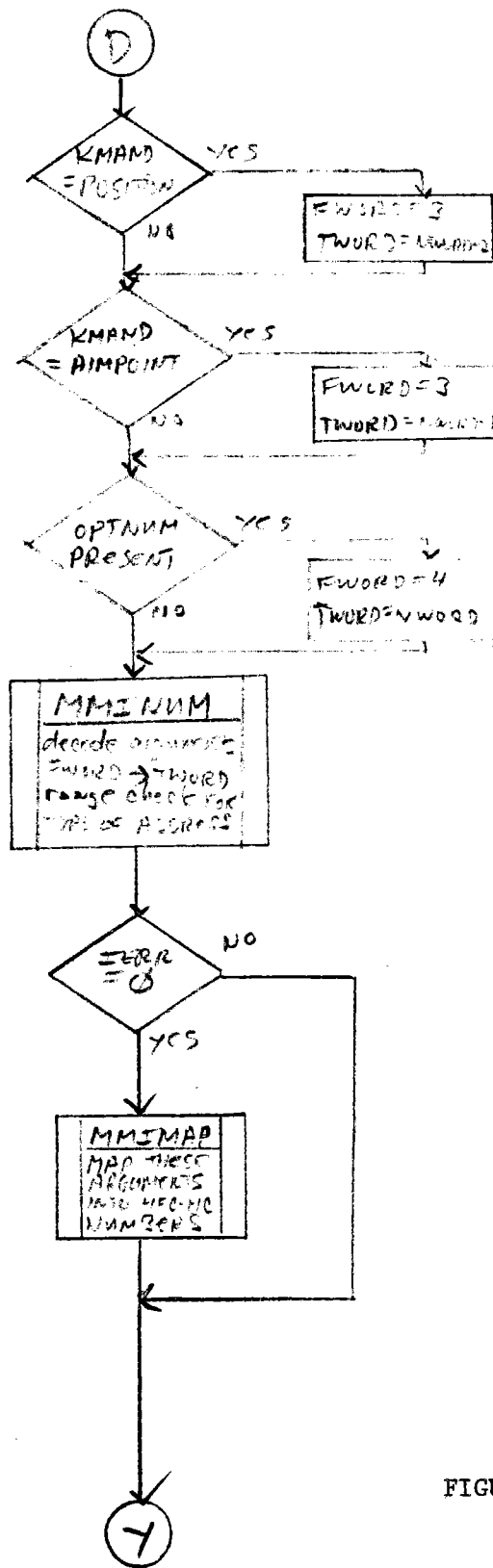


FIGURE 3.2.1-8 MMI FLOWCHART
(CONTINUED)

3.2.1.4.1.2 Submodule II - MMIDEQ

3.2.1.4.1.2.1 - Description

- a. Language Used - FORTRAN
- b. How Invoked - called by MMI
- c. Constraints and Limitations - message must be in Q numbers $\emptyset \rightarrow 4$.
- d. Processing.
 1. Local common buffer A2 is set to ASCII blanks.
 2. Call DEQU with Q# = \emptyset through 4, until a message found.
 3. Set SOURCE = S - Q#.
 4. If not from the emergency Q, go to step 6.
 5. Get source code from first word of string. If not a good code, default source to CS.
 6. Set local common word WER2GO.
 7. Call MMIWRD to get "word" pointers (CHRPNT).
 8. If message is from CS console, go to 11.
 9. Output the "COMMAND FROM" message to CS console.
 10. Output the command string to CS console.
 11. Write TIME and command to logger.
 12. Return to calling routine.

3.2.1.4.1.2.2 - Data Logic, and Command Paths

See figures 3.2.1-5 and 3.2.1-6 and Table 3.2.1-V.

3.2.1.4.1.2.3 - Internal Data Description

SORNAM - 32 bit integer ASCII

Source names: "FILE", "DAS", "OCS", "CS", "GRF", and "RTP" used for logger message.

EMGSOR - 16 bit integer ASCII

Source codes: "FI", "DA", "OC", "CS", "GR", and "RT" used to check first word of strings in Q# \emptyset .

3.2.1.4.1.2.4 - Flow

See figure - 3.2.1-9.

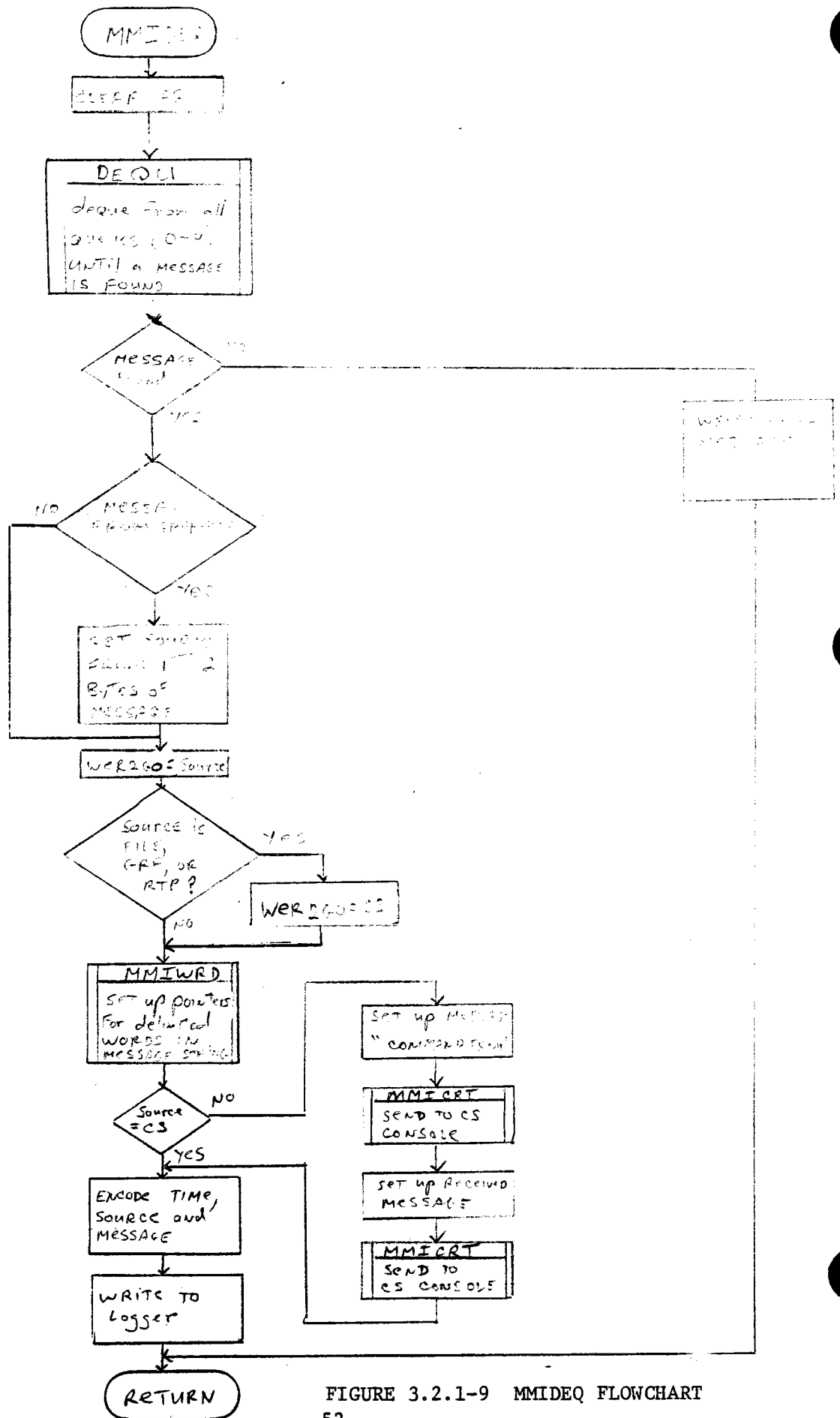


FIGURE 3.2.1-9 MMIDEQ FLOWCHART

3.2.1.4.1.3 Submodule III - MMIWRD

3.2.1.4.1.3.1 - Description

- a. Language Used - FORTRAN
- b. How Invoked - called by MMI
- c. Constraints and Limitations - Processing of the command string stops when two delimiters in a row are encountered.
- d. Processing
 - 1. Convert ASCII string from A2 format to A1 format.
 - 2. Set unused part of string to blanks.
 - 3. Build CHRPNP - pointers to beginning of each "word" in the string. Change commas and slashes to blanks.
 - 4. Encode the A1 data back to A2 format.
 - 5. Return to calling routine.
- e. Error Messages and Recovery - None.

3.2.1.4.1.3.2 - Data, Logic, and Command Paths

See figure 3.2.1-5 and Table 3.2.1-V.

3.2.1.4.1.3.3 - Internal Data Description

Miscellaneous local counters and pointers.

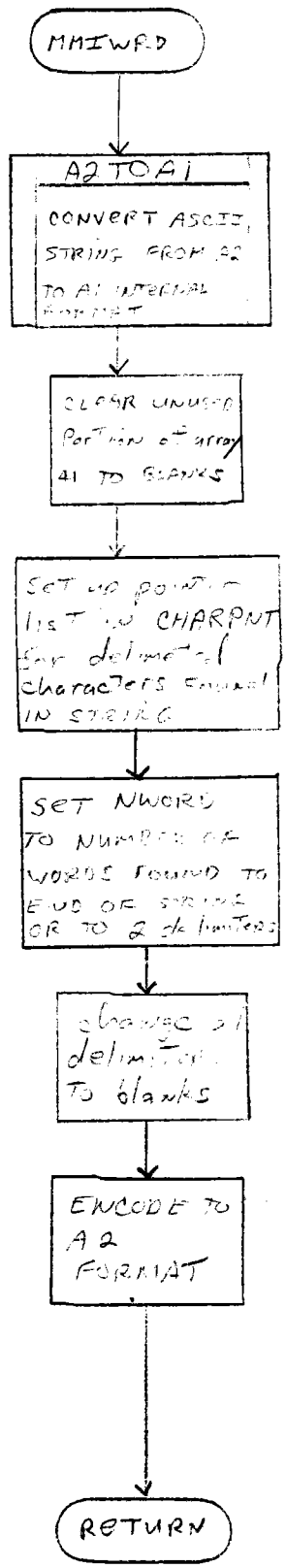
3.2.1.4.1.3.4 - Flowchart

See figure 3.2.1-10 for the MMIWRD flowchart.

3.2.1.4.1.4 Submodule IV - MMICLK

3.2.1.4.1.4.1 - Description

- a. Language Used - FORTRAN
- b. How Invoked - Called by MMI
- c. Constraints and Limitations - None



Fixed delimited word in string in array A2 - FROM - starting byte TO - last byte

FIGURE 3.2.1-10 MMIWRD FLOWCHART

d. Processing

1. Check local common MISFAG to see if it was called by MMIAID. If not, go to step 3.
2. Fill array OKSTUF as defined for command MISFAG and ARGADD. Go to step 12.
3. If this is a command check (KMAND \neq \emptyset), go to step 7.
4. Check for one-character addresses; check for "ALL".
5. Check that this address is allowed for command (KMAND).
6. Go to step 12.
7. Call MMICMD to check for a valid command (KMAND).
8. If source check is in effect, make sure KMAND is allowed from SOURCE.
9. If LOKDEG or LOKSTG are set, make sure KMAND is allowed.
10. Check EMSEQG - emergency sequence allowability.
11. Check Receiver trip bit in CPUSG if DEFR.
12. Return to calling routine.

- e. Error Messages and Recovery - Returns values for IERR and ONWORD if syntax or other errors are found.

3.2.1.4.1.4.2 - Data, Logic, and Command Paths

See Figures 3.2.1-5 and 3.2.1-6 and Table 3.2.1-V.

3.2.1.4.1.4.3 - Internal Data Description

ARGADD - A forty-word array whose values indicate which addressing and sources are valid for a command, and whether the command is allowable during emergencies. See Table 3.2.1-VII.

3.2.1.4.1.4.4 - Flowchart

See figure 3.2.1-11 for the MMICCHK flowchart.

INTERNAL DATA LAYOUT

APPLICATION NAME - MMICR

VARIABLE - ARGADD

PROGRAMMER	DATE	
		100 Address Allowed
		ARC
		HEC
		SECRET
		WIND
		WIND
		FILE
		Allocation (STWIND)
		De
		Disable (STWIND)
		Disable (DFOCUS)
		Disable (CMDRRG)

The data in this table is for informational purposes only. It is not intended to be used as a source of data for any other purpose. All data is subject to change without notice.

Table 3.2.1-VII Description of ARGADD

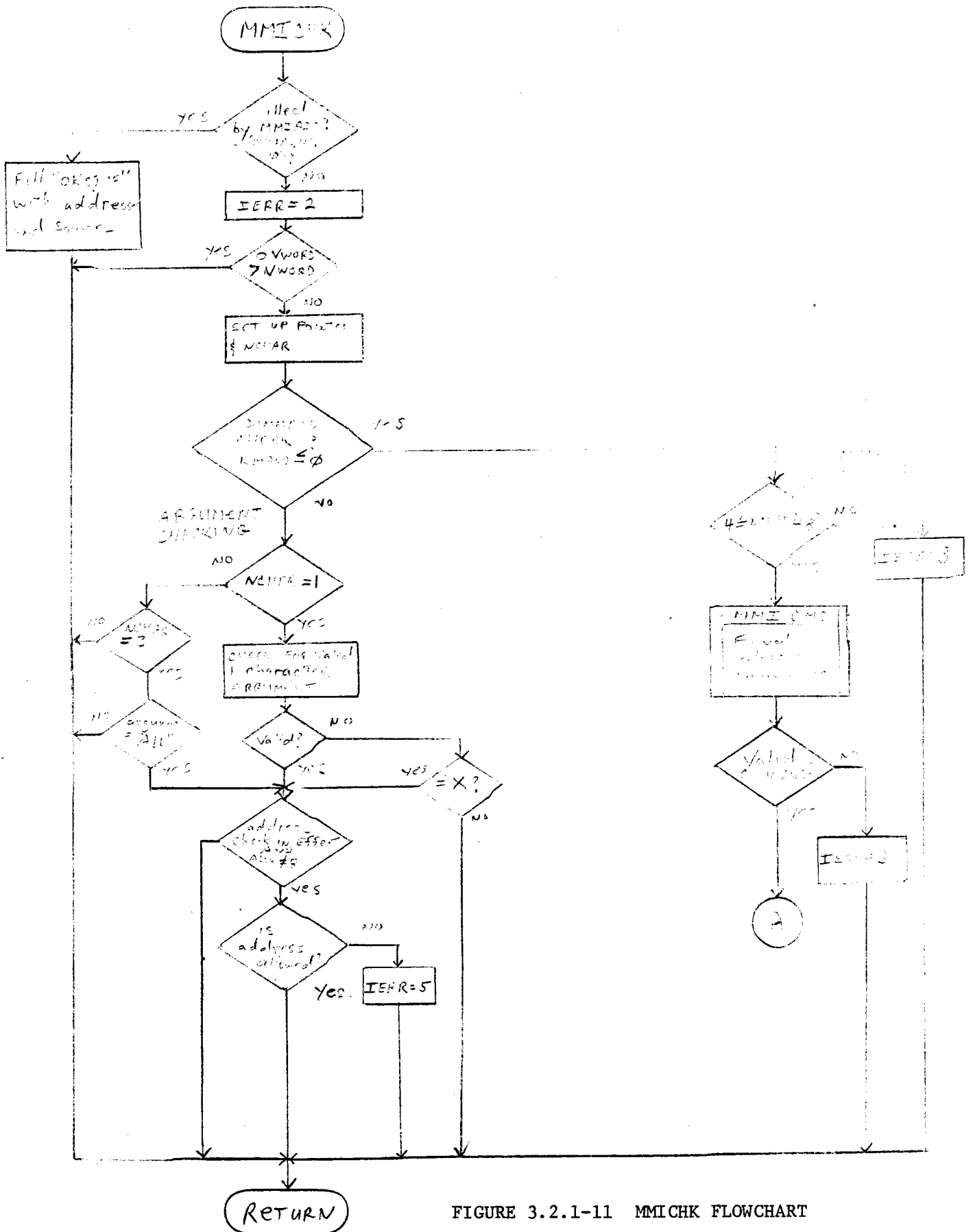


FIGURE 3.2.1-11 MMICK FLOWCHART

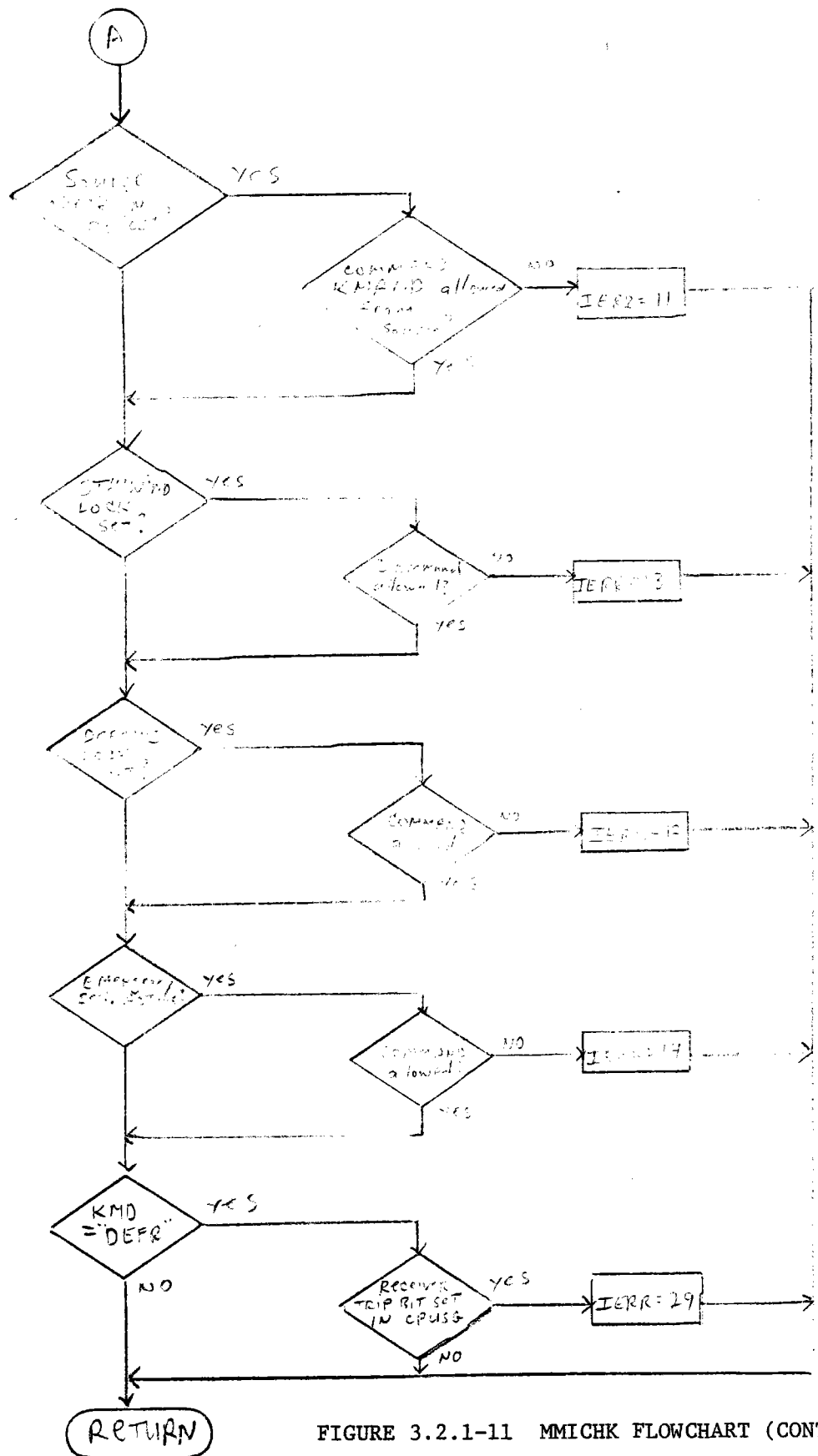


FIGURE 3.2.1-11 MMICK FLOWCHART (CONTINUED)

3.2.1.4.1.5 Submodule V - MMICMD

3.2.1.4.1.5.1 Description

- a. Language Used - FORTRAN
- b. How Invoked - Called by MMICLK, MMIAID
- c. Constraints and Limitations - None
- d. Processing
 - 1. Can-Code the input ASCII string A1. Blank fill for total of three words of three characters each.
 - 2. Calculate number of elements of KMANDS to check: $N2CHEK = (NCHAR + 2)/3$
 - 3. Calculate number of characters in last element to ignore (blank out): $N2BLNK = NCHAR - N2CHEK$.
 - 4. Check N2CHEK-1 elements of KMANDS against the input command string.
 - 5. Convert element N2CHEK to ASCII, blank fill N2BLNK elements, and can-code result.
 - 6. Example: KMANDS = @ INC @ REA @ SE
I/P string = @ AAA @ AAb @ bbb
N2CHEK = 2, N2BLNK = 1 : check @ AAA @ AAb against @ INC @ REb
 - 7. Go to step 4 and repeat until command found.
 - 8. Return to calling routine.
- e. Error Messages and Recovery - Output KMD; zero if no command match found.

3.2.1.4.1.5.2 - Data, Logic, and Command Paths

Data passed in subroutine call:

KMD - Output command number (as KMANDS).

ICHAR - Byte position in command string of "word" to check. Index to A1.

NCHAR - number of characters in word

See also figure 3.2.1-7 and Table 3.2.1-V.

3.2.1.4.1.5.3 - Internal Data Description

Temporary buffers for ASCII and can-coded data.

3.2.1.4.1.5.4 - Flowchart

Figure 3.2.1-12.

3.2.1.4.1.6 Submodule VI - MMINUM

3.2.1.4.1.6.1 - Description

- a. Language Used - FORTRAN
- b. How Invoked - Called by MMI, MMINON, MMISTR, MMIST3.
- c. Constraints and Limitations - None.
- d. Processing

1. If hexadecimal conversion (convert = 1), (results → NUMARY) call MMIHXD and go to step 6.
2. Check for correct number of digits.
3. Decode the ASCII to binary (NUMARY).
4. Check for duplicate numbers.
5. Perform range check for addresses:

Heliostat, Arc: "RRNN"
Constraint: $1 < RR < 29$ and $1 < NN < (MDNPRG(RR+1) - MDNPRG(RR))$

HFC: $1 < HFC < 64$

SEGMENT: RWW
Constraint: $1 < R < 4$ and $1 < WW < 12$
or $R = 5$ and $4 < WW < 9$

WEDGE: $1 < WW < 12$

RING: $1 < R < 5$

6. Load -9999 into NUMARY to signify end of data, and return to calling routine.

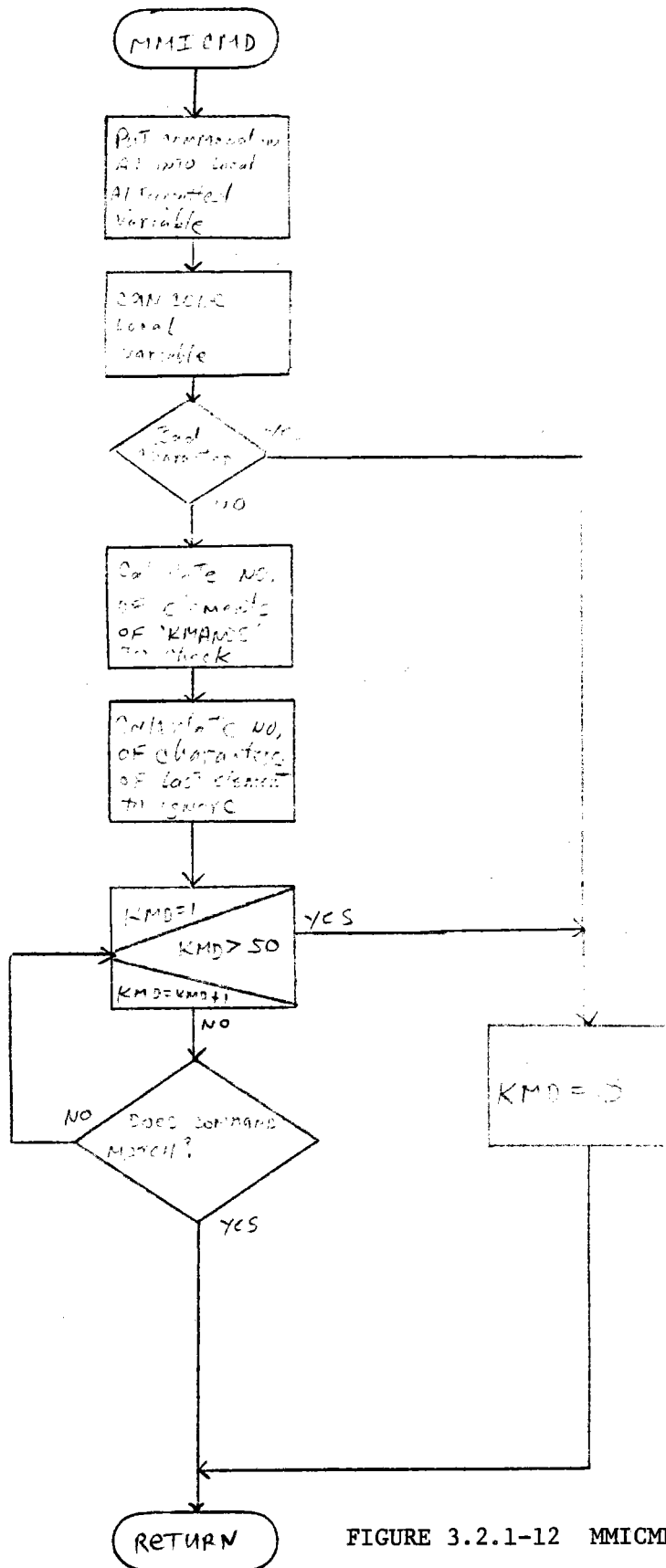


FIGURE 3.2.1-12 MMICMD FLOWCHART

3.2.1.4.1.6.2 - Data, Logic, Command Paths

Data passed in subroutine call

- FWORD - First "word" in ACSII string to decode.
- TWORD - Last "word" in ASCII string to decode.
- CONVERT - 1 = Hex input; other = integer input.
- ADDTYP - Address type for range check, \emptyset = no check.
Like ADDRESS in MMICOM.

See also Figures 3.2.1-5 through 3.2.1-7 and Table 3.2.1-V.

3.2.1.4.1.6.3 - Internal Data Description

- a. Number of digits and value ranges for each type of address.

3.2.1.4.1.6.4 - Flowchart

See figure 3.2.1-13.

3.2.1.4.1.7 Submodule VII - MMIHXD

3.2.1.4.1.7.1 - Description

- a. Language - FORTRAN.
- b. How Invoked - Called by MMI, MMINUM, MMINON, and MMIST2.
- c. Error Messages and Recovery - IERR is set if a decoding error is encountered.
- d. Processing
 1. Check for the presence of five characters, first = pound (#) sign. If so, go to step 4.
 2. Check for presence of decimal point; decode into Real* 4 variable.
 3. Multiply result of 2 by 91.022 to get integer hex value. Go to step 5.
 4. Decode using Z4 format.
 5. Return to calling routine.

- e. Error Messages and Recovery - Only detected errors may be in Arc mapping. HC numbers in different rows or opposite sides of the field will cause IERR to be set.

3.2.1.4.1.7.2 - Data, Logic, Command Paths

See figures 3.2.1-5 and Table 3.2.1-V.

3.2.1.4.1.7.3 - Internal Data

ASCII values for "#" and "." (decimal point).

3.2.1.4.1.7.4 - Flowchart

Figure 3.2.1-14.

3.2.1.4.1.8 Submodule VIII - MMIMAP

3.2.1.4.1.8.1 - Description

- a. Language - FORTRAN
- b. How Invoked - Called by MMI, MMINON, MMISTR, and MMITST.
- c. Constraints and Limitations - Input data (NUMARY) must have been validity-checked and correspond to the value of ADDRES.
- d. Processing
 1. Check for Segment, Wedge, or Ring input. If not, go to step 5.
 2. For each input Segment, Wedge, or Ring number, loop through the segments defined by:

	R1	R2	W1	W2		
W = W1 to W2	Where:	SEG=RWW	R	R	WW	WW
R = R1 to R2		WEDG=WW	1	5	WW	WW
		RING=R	R	R	1	12

3. Set local common NPBLOK, load global common HCMAPG with data from SEGMPG.
4. Go to step 8.

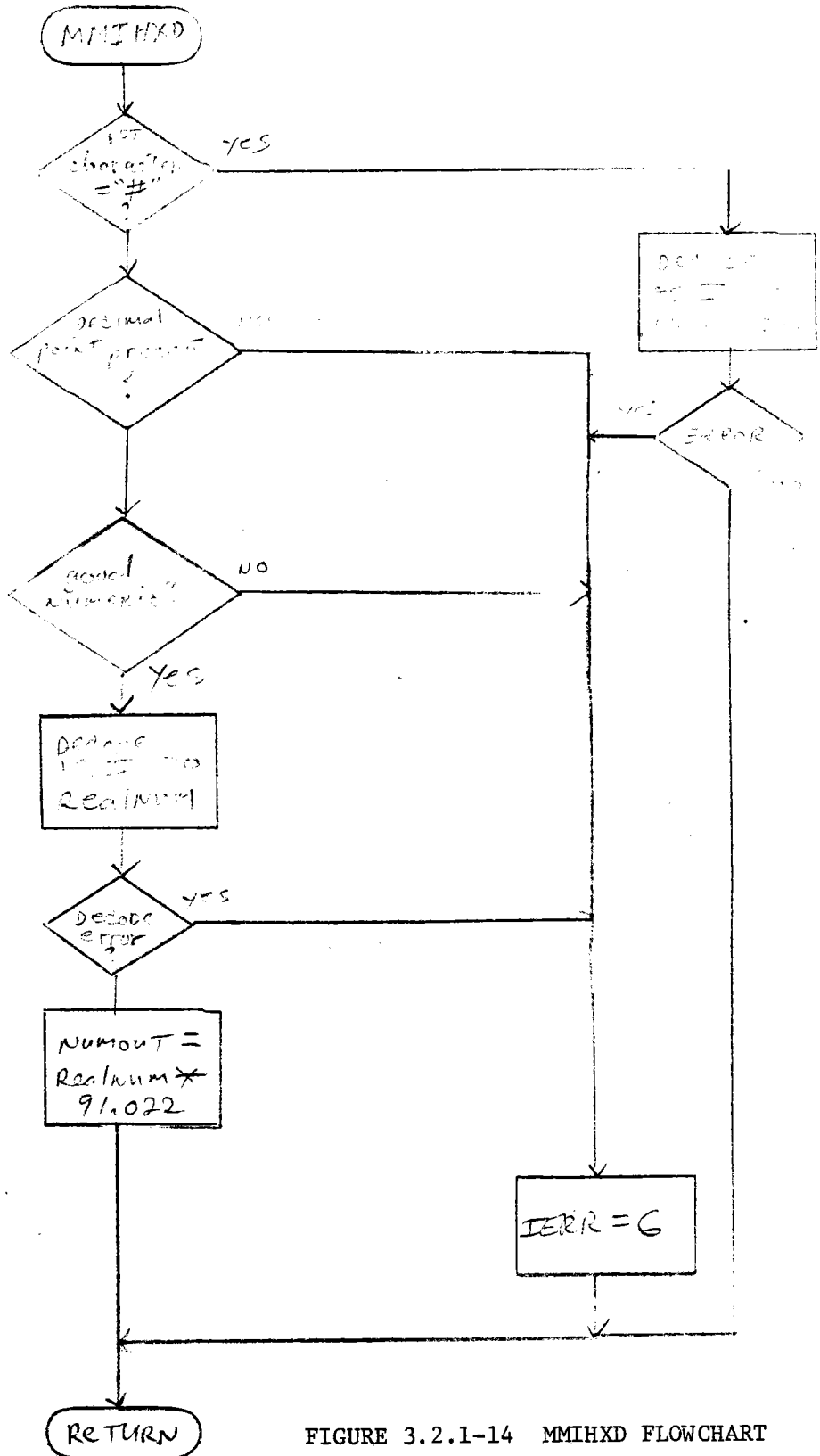


FIGURE 3.2.1-14 MMHXD FLOWCHART

5. For HFC address, calculate 32 HFC-HC numbers by: $HFCHC = (HFC-1)*32 + IHC$, $IHC = 1,32$. Load into HCMAPG.
 6. For HC addressing, load from MD2HCG to HCMAPG.
 7. For ARC addressing, check to insure that each HC pair is in the same row, and the same side of field. Loop through the HC pairs, loading HFCHC numbers from MD2HCG to HCMAPG.
 8. Set last element of HCMAPG to signify end of data, and return to calling routine.
- e. Error Messages and Recovery - IERR is set if two HC numbers defining an Arc are not in the same row and the same side of the field.

3.2.1.4.1.8.2 - Data, Logic. and Command Paths

See figures 3.2.1-5 through 3.2.1-7.

3.2.1.4.1.8.3 - Internal Data Description

Miscellaneous array pointers.

3.2.1.4.1.8.4 - Flowchart

Figure 3.2.1-15.

3.2.1.4.1.9 Submodule IX - MMINON

3.2.1.4.1.9.1 - Description

- a. Language Used - FORTRAN.
- b. How Invoked - Called by MMI.
- c. Constraints and Limitations - None.
- d. Processing - MMINON takes these actions for the following commands:
 1. RLHIWIND - Clear LOKSTG is set, error message if not set.
 2. DEFRLSE - Clear LOKDEG if set, error message if not set.
 3. UPAIM - Call MMINUM to get Aimpoint Array number. Set DISKRG and activate DSK.

4. UPBIAS - Call MMICLK for "H" address.
Call MMINUM for HC number.
Call MMIMAP for HFCHC number.
Call MMIHXD for AZ/EL values.
Set DISKRG and activate DSK.
 5. STATUS - Call MMISTR.
 6. HELP - Call MMIAID.
 7. CFSTART - Check if CFO is already active.
Get CFNAMG from ASCII input, activate CFO.
 8. CFABORT - Set CFABOG if CFO is active.
 9. CFWAIT - Decode wait time from ASCII string
and set CFWATG if CFO is active.
 10. REFRESH - Call MMIREF.
 11. AIMLIST, SAVEPOS - Set IERR = 21 (command not
available).
 12. TEST - Call MMITST.
 13. BCS Commands - Call MMIBCS.
 14. All Commands - Return to calling routines.
- e. Error Messages and Recovery - Any errors encountered
during the processing described in d. will cause
IERR to be set.

3.2.1.4.1.9.2 - Data, Logic, Command Paths

See figures 3.2.1-5 through 3.2.1-7.

3.2.1.4.1.9.3 - Internal Data Description

Miscellaneous array pointers.

3.2.1.4.1.9.4 - Flowchart

Figure 3.2.1-16.

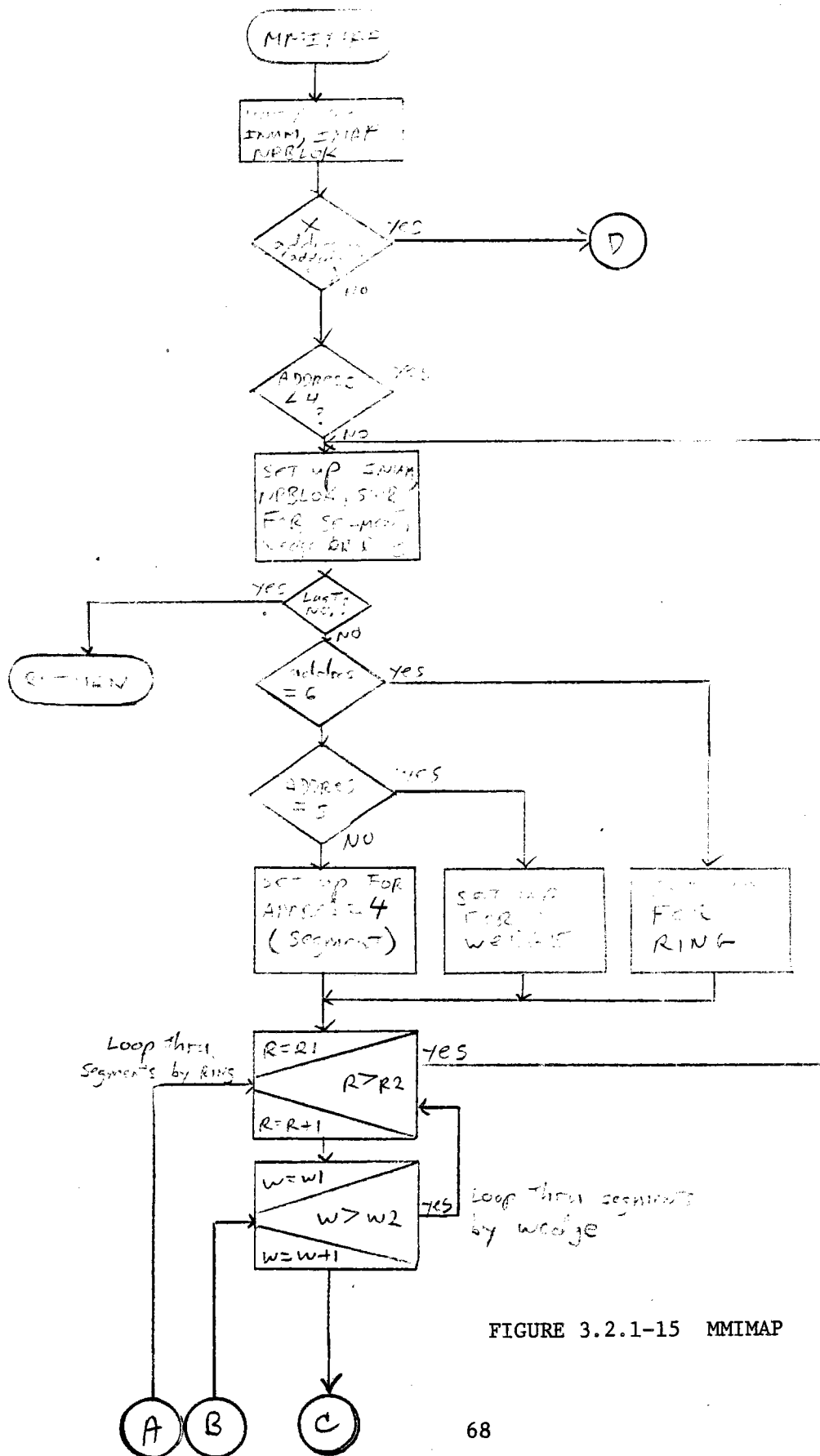


FIGURE 3.2.1-15 MMIMAP

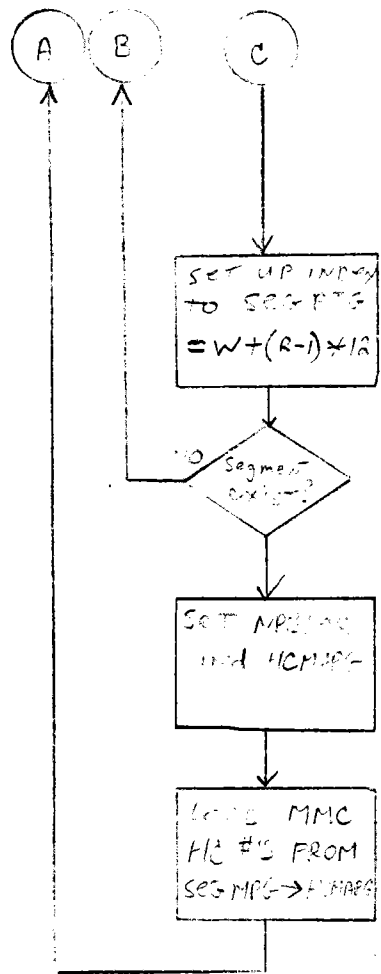


FIGURE 3.2.1-15 MMIMAP (Continued)

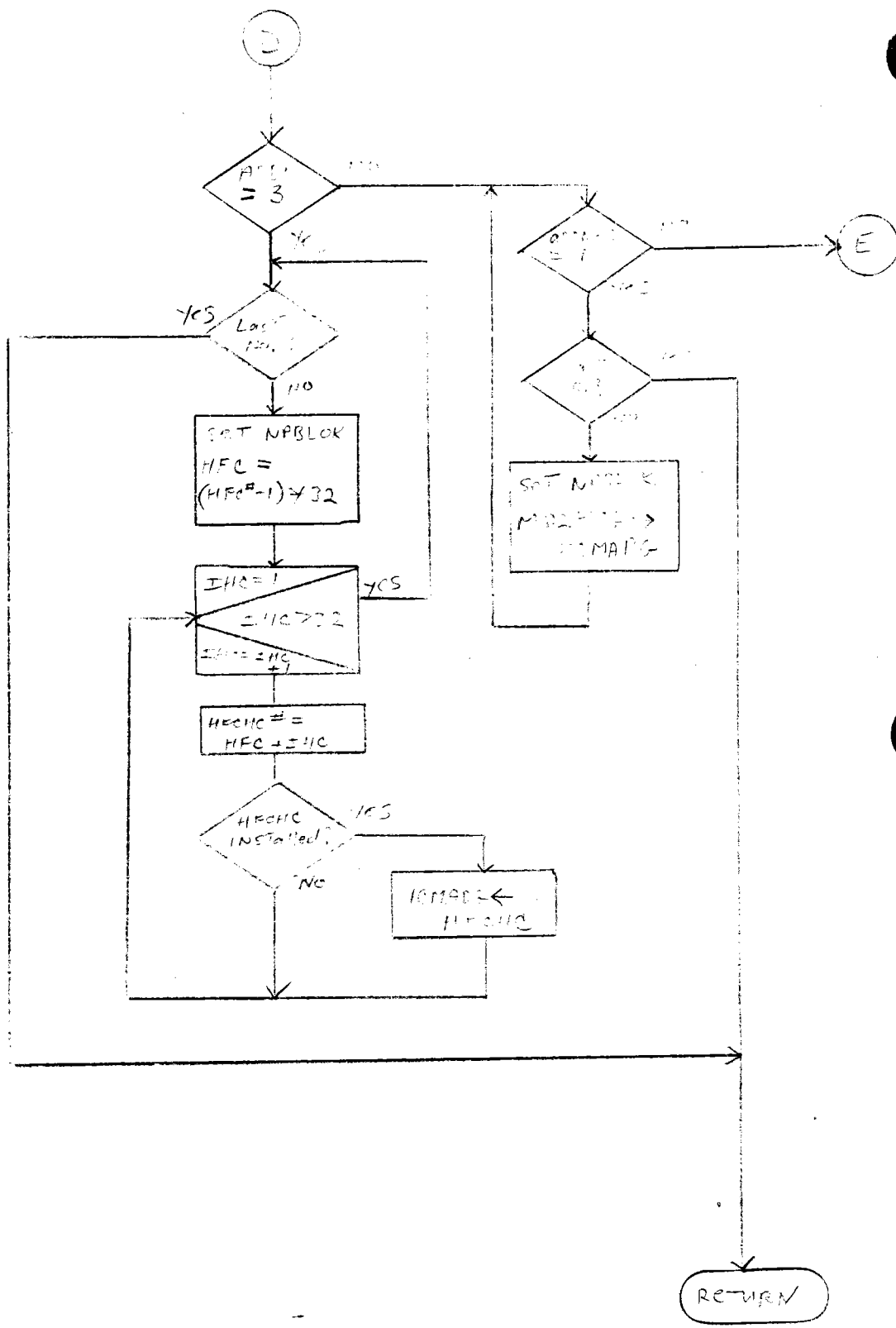


FIGURE 3.2.1-15 MMIMAP (Continued)

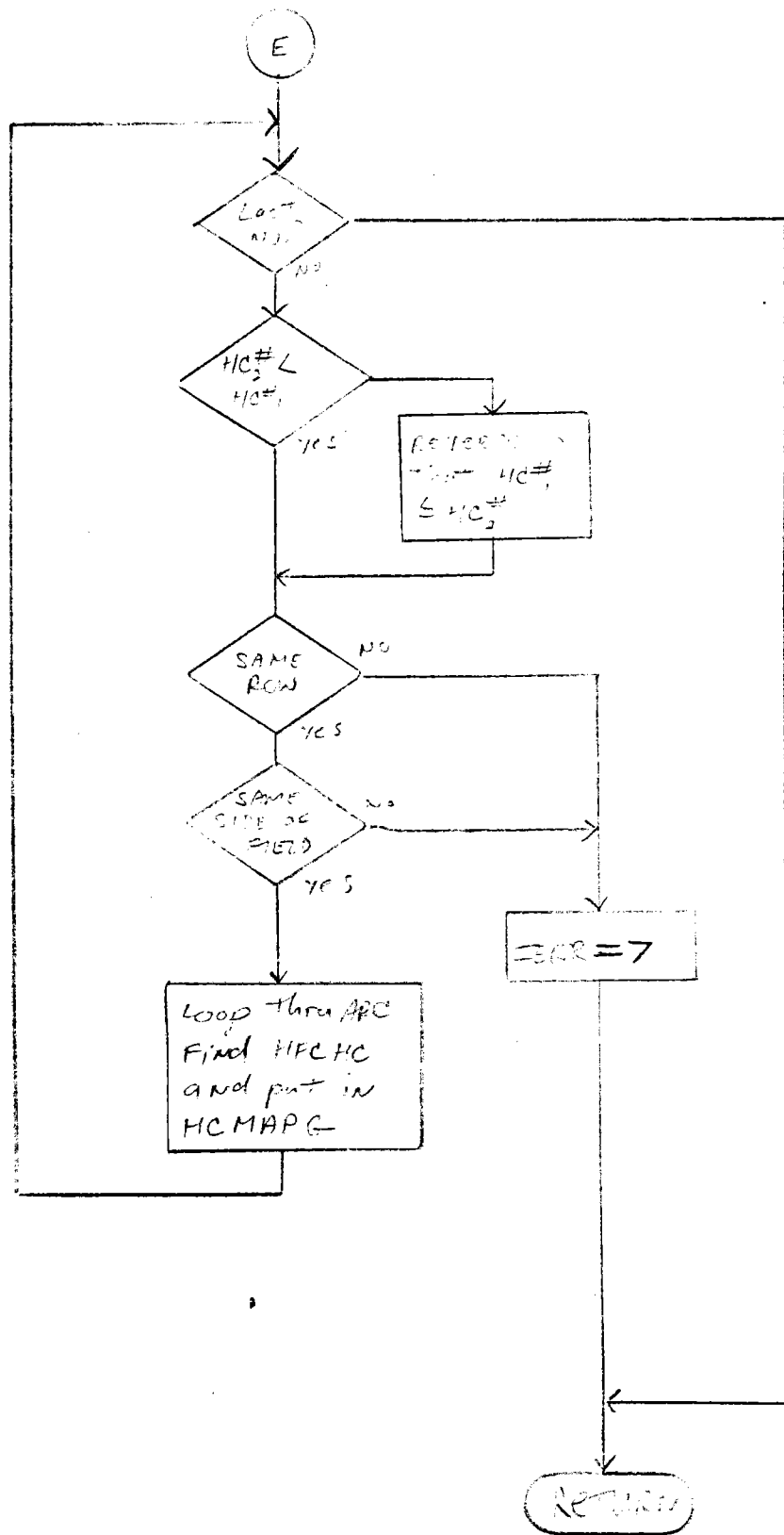


FIGURE 3.2.1-15 MMIMAP (Continued)

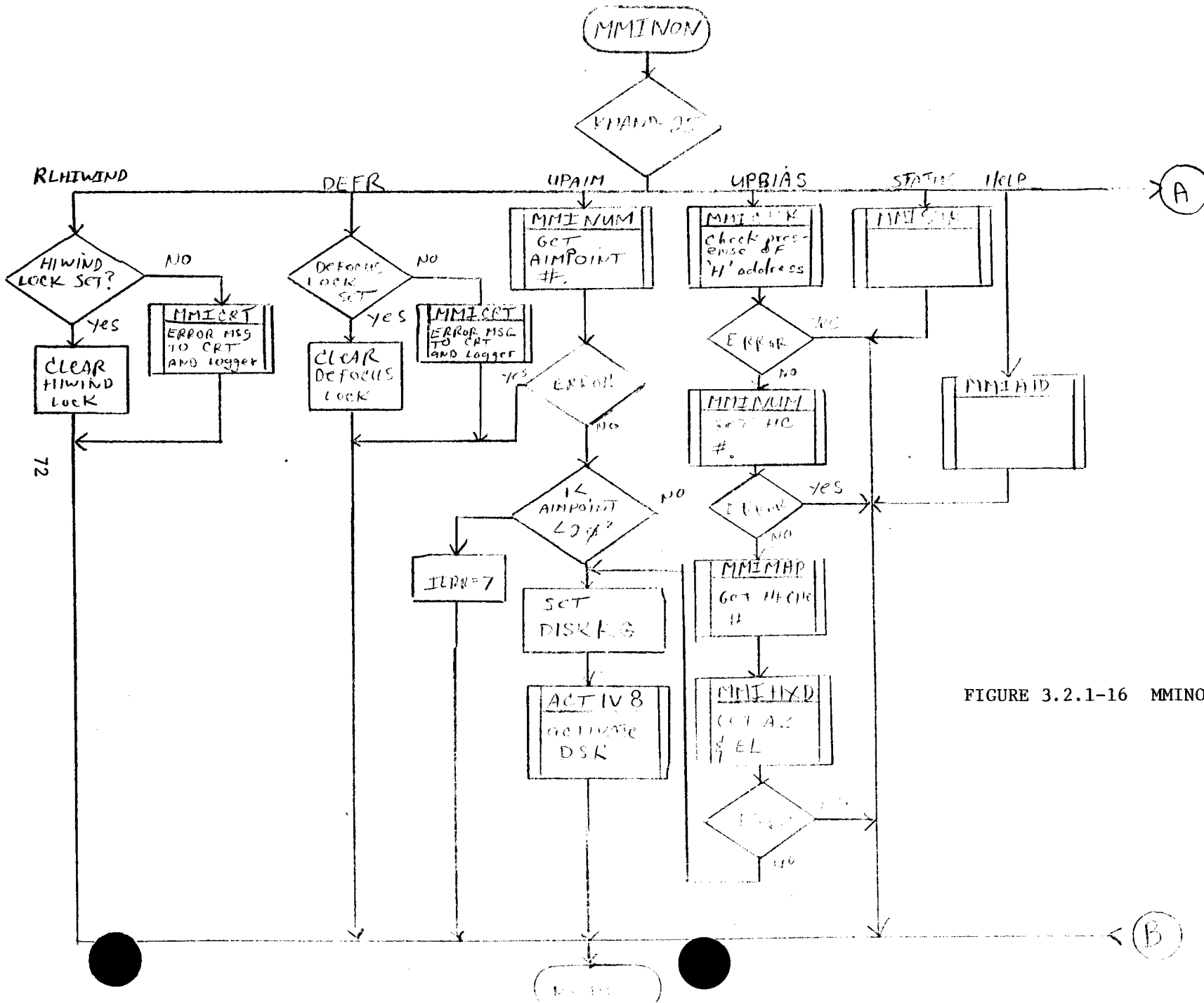


FIGURE 3.2.1-16 MMINON FLOWCHART

3.2.1.4.1.10 Submodule X: MMISTR

3.2.1.4.1.10.1 - Description

- a. Language Used - FORTRAN
- b. How Invoked - Called by MMINON
- c. Constraints and Limitations - None
- d. Processing
 1. Clear ISTATG. If two characters in second word, call MMIST2 and go to 7.
 2. Call MMICLK for "H", "R", or "ALL" status.
 3. If check 2. shows an error, check for "M".
 4. If "H", call MMINUM and MMIMAP to get HFCHC number.
 5. If "R", call MMINUM for Ring number.
 6. Set ISTATG.
 7. Activate STA.
 8. Return to calling routine.
- e. Error Messages and Recovery - Errors encountered during processing in d. will cause IERR to be set.

3.2.1.4.1.10.2 - Data, Logic, Command Paths

See figures 3.2.1-5 through 3.2.1-7 and Tables 3.2.1-IV through 3.2.1-VI.

3.2.1.4.1.10.3 - Internal Data Description

ASCII value for "M".

3.2.1.4.1.10.4 - Flowchart

Figure 3.2.1-17.

3.2.1.4.11 Submodule XI: MMIST2

3.2.1.4.11.1 - Description

- a. Language - FORTRAN
- b. How Invoked - Called by MMISTR
- c. Constraints and Limitations - None
- d. Processing

1. Check for one of the following two-character codes in second "word" of command string, and set ISTATG as follows:

Code	ISTATG(1)
"H1"	16
"H2"	17
"F2"	18
"ER"	Call MMIST3; go to step 5.

2. Set the MASK and MATCH values as follows:

Input Character	Mask Character	Match Character
*	Ø	Ø
Other	F	Same as Input

(Example: INPUT = "*A1*"; MASK = ØFFØ
MATCH = ØA1Ø).

3. Set ISTATG(2) = MASK
(3) = MATCH.
4. Activate STA.
5. Return to calling routine.

- e. Error Messages and Recovery - Errors detected in any processing steps (d) cause IERR to be set.

3.2.1.4.1.11.2 - Data, Logic, Command Paths

See figures 3.2.1-5 through 3.2.1-7 and Tables 3.2.1-IV through 3.2.1-VI.

3.2.1.4.1.11.3 - Internal Data Description

ASCII codes for "H", "F", "E", "R", "Ø", "1", and "2".

3.2.1.4.1.11.4 - Flowchart

Figure 3.2.1-18.

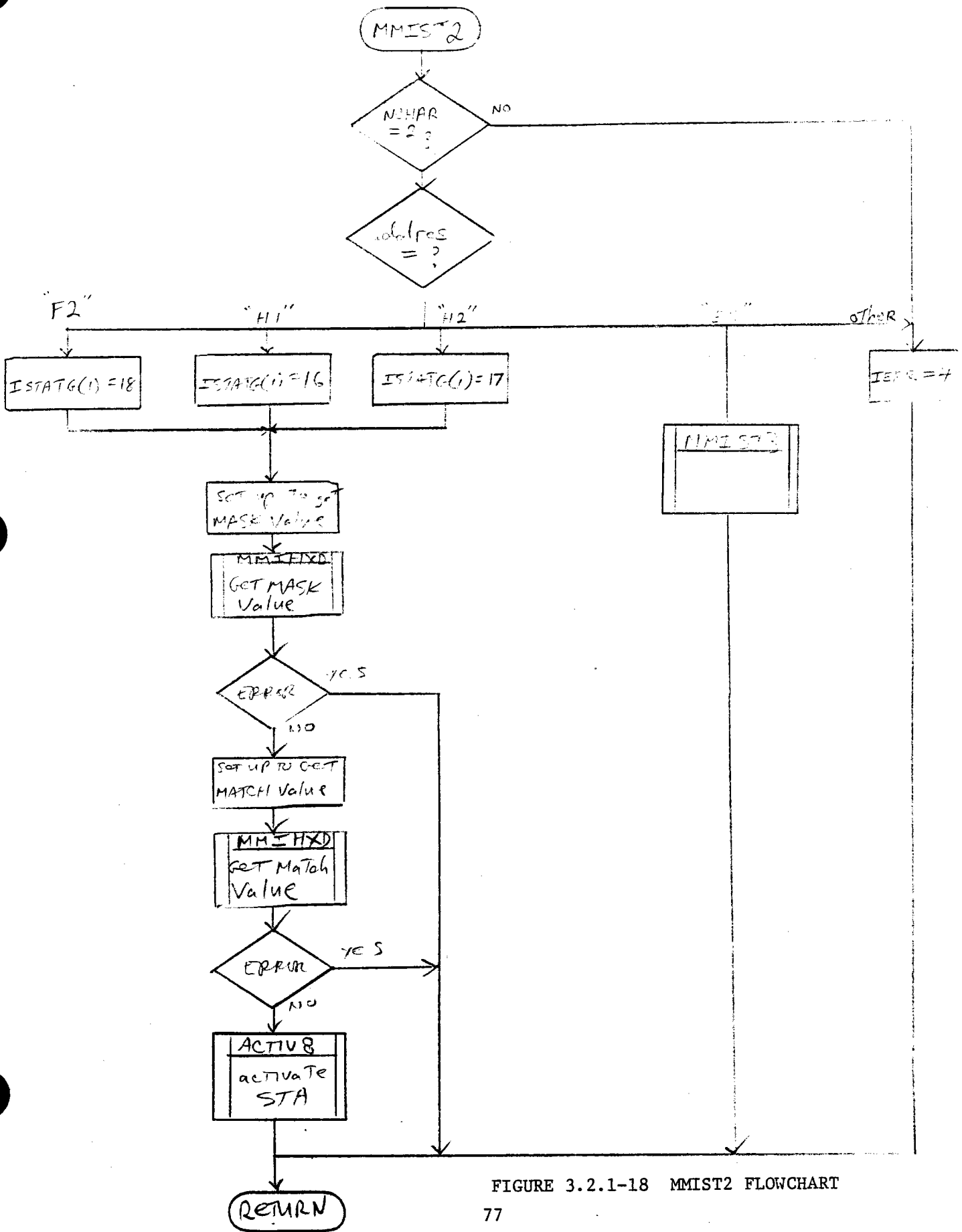


FIGURE 3.2.1-18 MMIST2 FLOWCHART

3.2.1.4.1.12 Submodule XII - MMIST3

3.2.1.4.1.12.1 - Description

- a. Language - FORTRAN.
- b. How Invoked - Called by MMIST2.
- c. Constraints and Limitations - Second "word" of command string must be "ER".
- d. Processing
 1. Can-Code the third "word" in the command string.
 2. If result is @ HEL (help), output a message with a list of memnomics and go to 7.
 3. Check result of 1. against CANERR array.
 4. Set ISTATG(2) and ISTATG(3) to the corresponding MASKER elements.
 5. Set ISTATG(1).
 6. Activate STA.
 7. Return to calling routine.
- e. Error Messages and Recovery - Errors found in any of the above processing steps (d) cause IERR to be set.

3.2.1.4.1.12.2 - Data, Logic, and Command Paths

See figures 3.2.1-5 through 3.2.1-7 and Tables 3.2.1-IV through 3.2.1-VI.

3.2.1.4.1.12.3 - Internal Data Description

ASCII array names "HCST1G", "HCST2G", and "HFC22G" for messages.

3.2.1.4.1.12.4 - Flowchart

Figure 3.2.1-19.

3.2.1.4.1.13 Submodule XIII - MMIREF

3.2.1.4.1.13.1 - Description

- a. Language - FORTRAN.
- b. How Invoked - Called by MMINON.

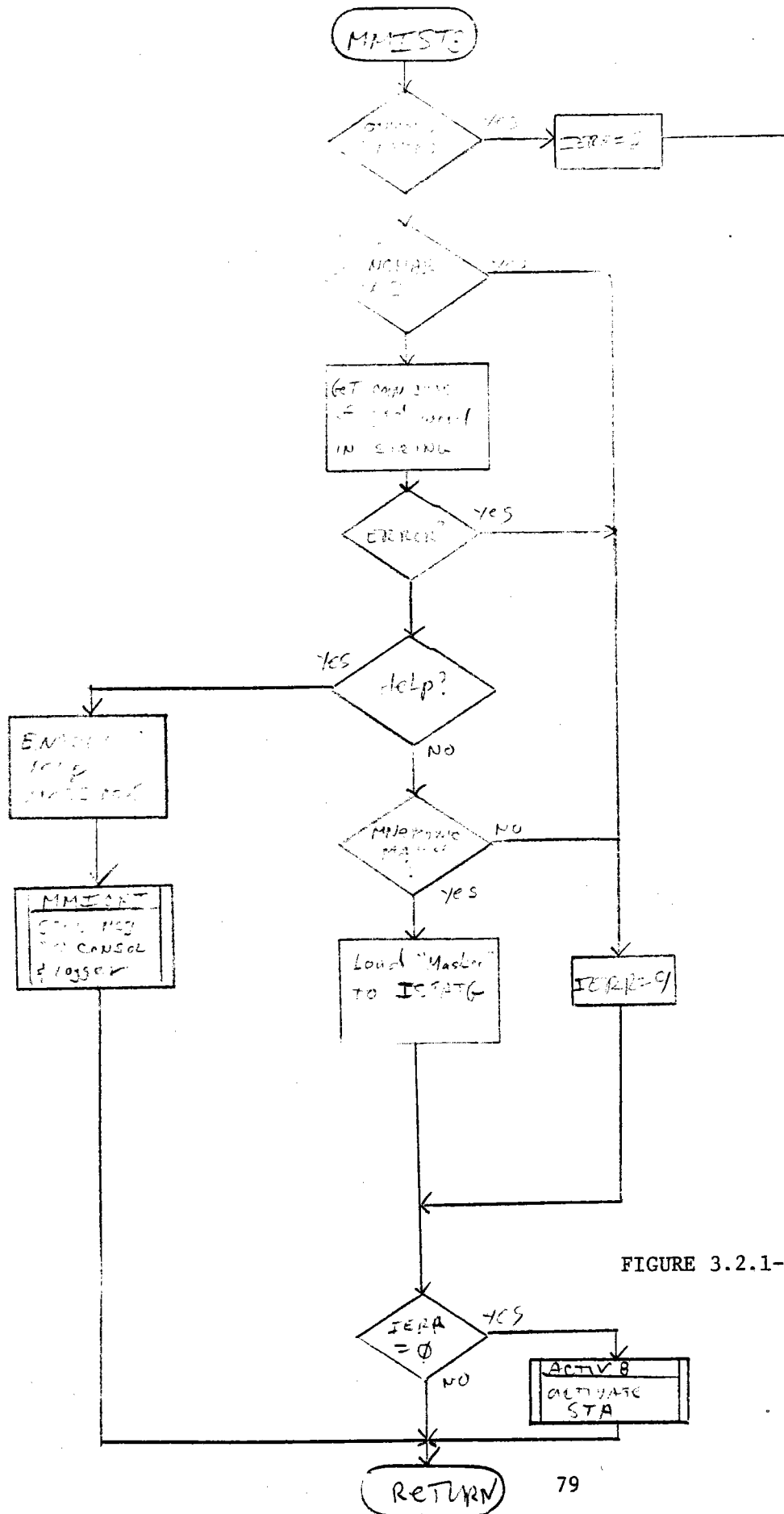


FIGURE 3.2.1-19 MMIST3 FLOWCHART

c. Constraints and Limitations - Input to the CS console is not possible while MMIREF redraws the displays.

d. Processing

1. Clear CS colsole screen.
2. Write out red and yellow alarms areas.
3. Write out blue status area.
4. Write out status titles.
5. Set SPARE5(1) for STS.
6. Return to calling routine.

e. Error Messages and Recovery - None.

3.2.1.4.1.13.2 - Data, Logic, and Command Paths

See Figures 3.2.1-5 through 3.2.1-7 and Tables 3.2.1-IV through 3.2.1-VI.

3.2.1.4.1.13.3 - Internal Data Description

Contains data arrays for ISC 8001 position and color codes. Contains arrays of ittle names.

3.2.1.4.1.13.4 - Flowchart

Figure 3.2.1-20.

3.2.1.4.1.14 Submodule XIV - MMIRSP

3.2.1.4.1.14.1 - Description

a. Language - FORTRAN

b. How Invoked - Called by MMI.

c. Constraints and Limitations - HMAPG must be properly set by CMDPRC.

d. Processing

1. If the CMDPRC module has set CPPRTG to indicate an error, set IERR accordingly.
2. If the command address was "ALL", get the total of HCS accepting the command.

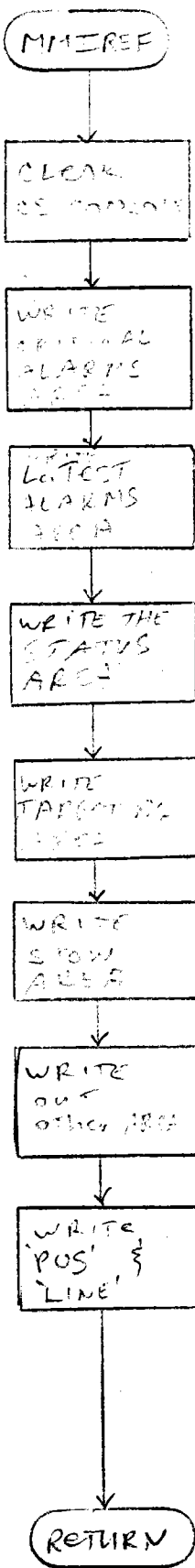


FIGURE 3.2.1-20 MMIREF FLOWCHART

3. For other addresses, get totals for the HC accepting (OK), wrong mode (WM), offline (OF), and not installed (NI).
4. If one "block" of address in HCMAPG, format output for one line. If more than one "block", format for four or eight lines of output.
5. Call MMICRT for each line output.
6. Return to calling routine.

e. Error Messages and Recovery - None.

3.2.1.4.1.14.2 - Data, Logic, and Command Paths

See figures 3.2.1-5 through 3.2.1-7 and Tables 3.2.1-IV through 3.2.1-VI.

3.2.1.4.1.14.3 - Internal Data Description

Miscellaneous array pointers, ASCII string for titles, bell.

3.2.1.4.1.14.4 - Flowchart

Figure 3.2.1-21.

3.2.1.4.1.15 Submodule XV - MMICRT

3.2.1.4.1.15.1 - Description

- a. Language - FORTRAN
- b. How Invoked - Called by MMIDEG, MMINON, MMIST3, MMIRSP, MMIERR, MMITST, and MMIAID.
- c. Constraints and Limitations - None.
- d. Processing
 1. Calculate number of words to output from (local common) number of bytes.
 2. If number of words less than one or greater than 54, output an error message and go to 6.
 3. If output goes to CS console, enqueue the message to CS0 (Q#7).
 4. If output goes to logger, remove the color codes and write message to logger.

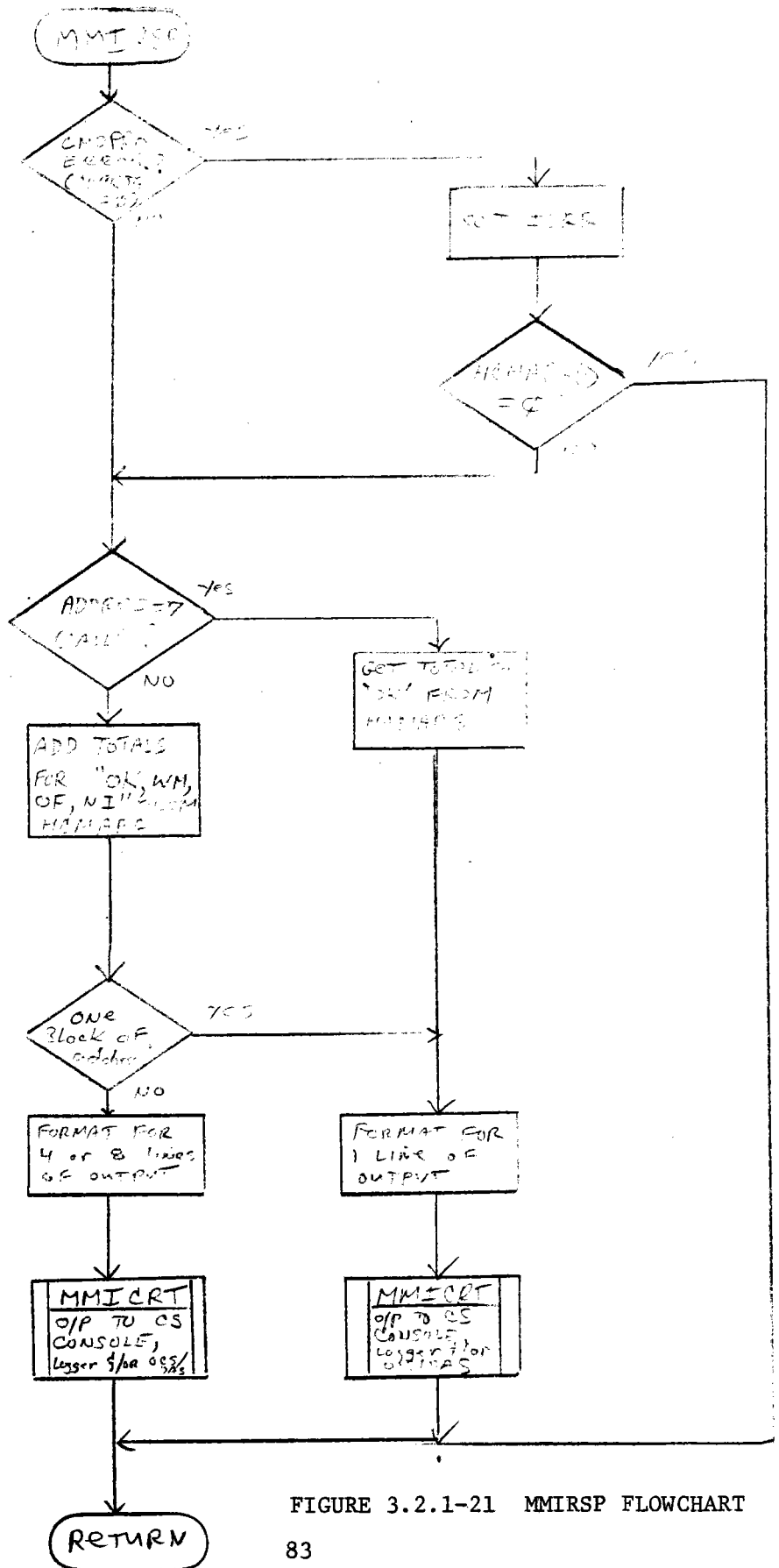


FIGURE 3.2.1-21 MMIRSP FLOWCHART

5. If message goes to OCS or DAS (SOURCE set appropriately and OP2X1F=1), encode header and message into buffer; write buffer to OCS/DAS core device.
 6. Return to calling routine.
- e. Error Messages and Recovery - Write message to system console for an enqueue failure or if passed bad parameters.

3.2.1.4.1.15.2 - Data, Logic, Command Paths

Data passed through subroutine call--

```
WHAT = 0   CRT ONLY
      = 1   Both
      = 2   Logger only.
```

Additionally, if OP2X1F \neq 0 and SOURCE indicates OCS or DAS, the ASCII message is written to a core device.

3.2.1.4.1.15.3 - Internal Data Description

Contains an ASCII header to preface the output message for OCS or DAS use.

3.2.1.4.1.15.4 - Flowchart

Figure 3.2.1-22.

3.2.1.4.1.16 Submodule XVI - MMIERR

3.2.1.4.1.16.1 - Description

- a. Language - FORTRAN
- b. How Invoked - Called by MMI.
- c. Constraints and Limitations - Input value for IERR must be $0 < IERR < 30$, and if $IERR < 11$, ONWORD must be set.
- d. Processing
 1. Set OP2X1F for MMICRT.
 2. If not a syntax error, encode error message, then go to 6.
 3. If parameter missing, set IERR=2.

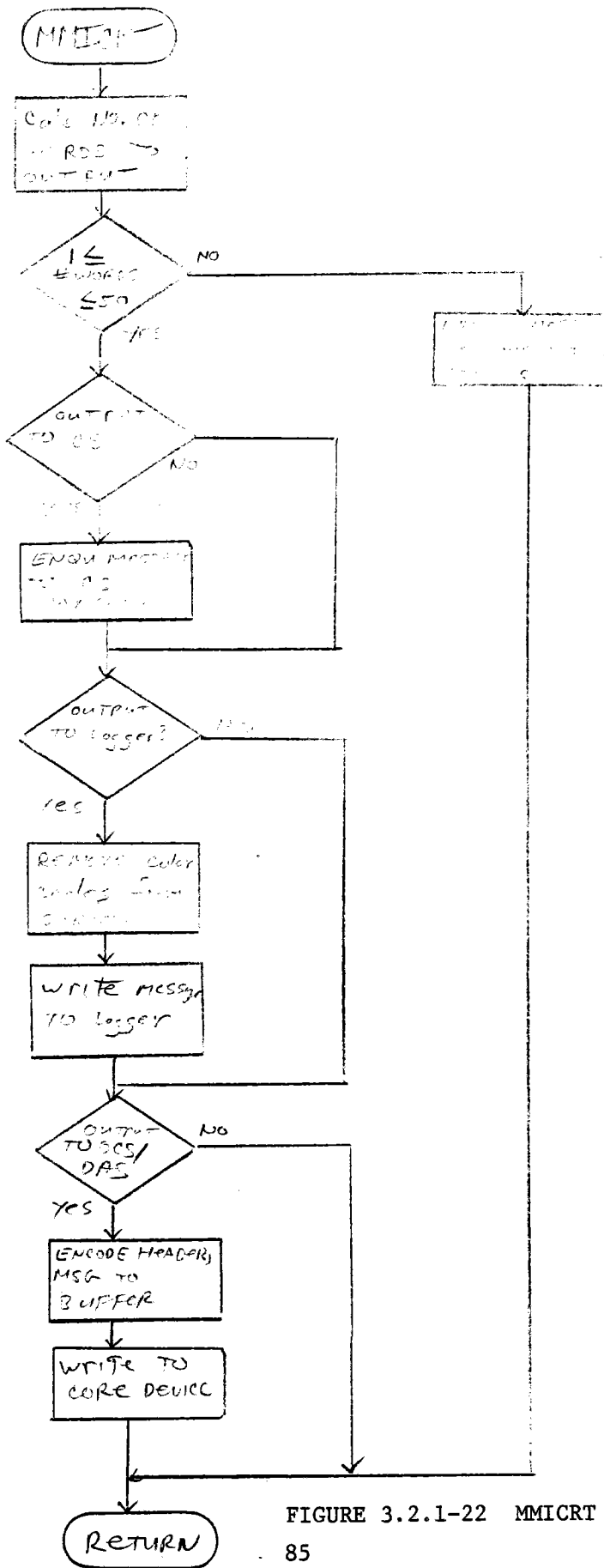


FIGURE 3.2.1-22 MMICRT FLOWCHART

4. If message on left of echoed part of command string, format message with LEFTFT format.
 5. If message on right side of echoed part of command string, format message with RITEFT format.
 6. Call MMICRT to output message to CS console, logger, and OCS or DAS.
 7. Return to calling routine.
- e. Error Messages and Recovery - If ONWORD is greater than NWORD, IERR is reset to 2 (parameter missing).

3.2.1.4.1.16.2 - Data, Logic, Command Paths

The output message can have two syntax error formats (a and b) and one nonsyntax error format (c). (/ = color change):

- a. Echoed "word" / error message.
- b. Error message / echoed "word".
- c. Error message.

If IERR > 10, the format is C.

If the byte position (CHRPNT(ONWORD)) of the "word" to be echoed is greater than 22, the format is (6), otherwise it is a.

3.2.1.4.1.16.3 - Internal Data Description

In addition to ASCII buffers containing the error messages, MMIERR uses two format buffers for formats a. and b. (syntax errors). For messages to the right of the echoed "word", the error message is encoded with this format:

"(AZ, NX, MA1, 1X, 12A2)".

For messages to the left:

"(A2, 10A2, NX, A2, MA1)".

Values for M and N are calculated, and their ASCII representation is put in phase in the format buffer.

3.2.1.4.1.16.4 - Flowchart

Figure 3.2.1-23.

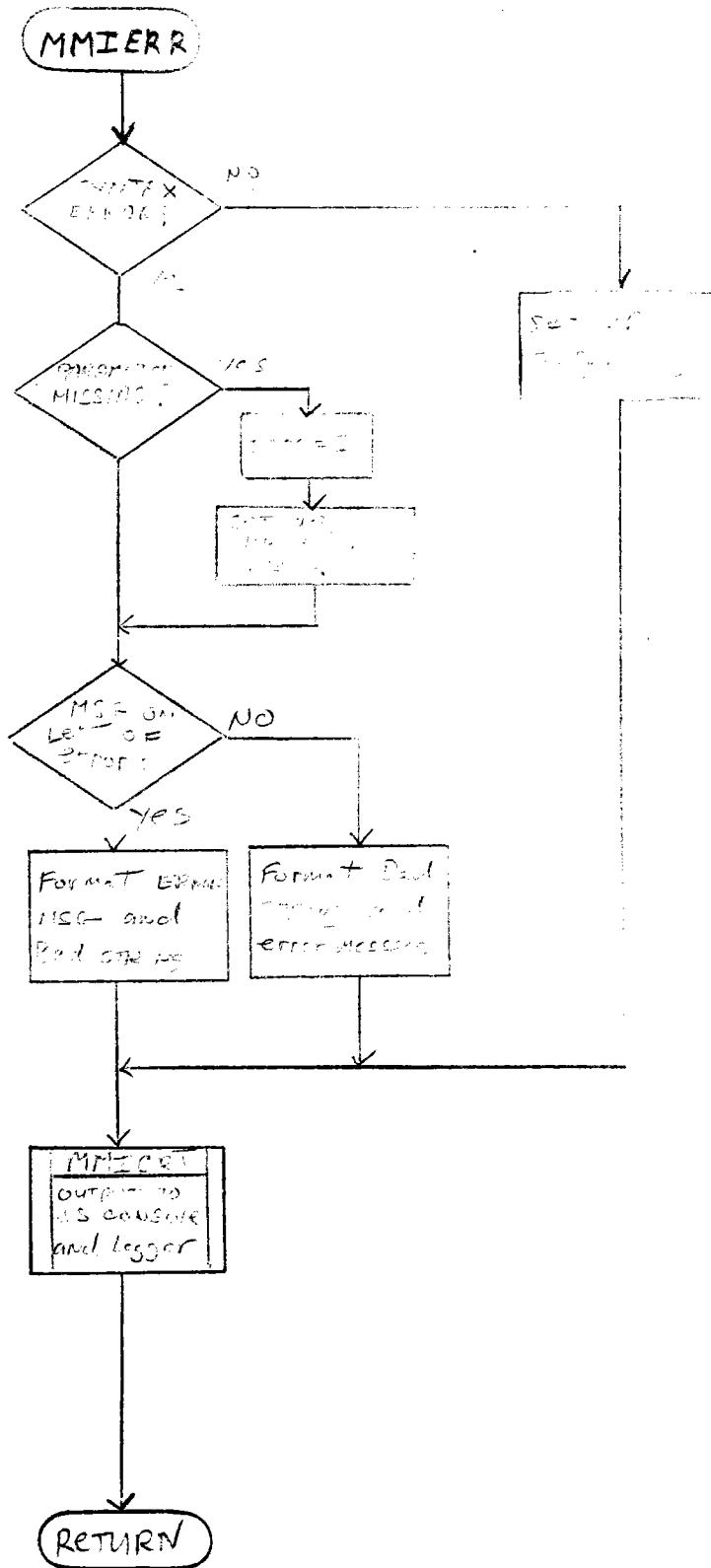


FIGURE 3.2.1-23 MMIERR FLOWCHART

3.2.1.4.1.17 Submodule XVII - MMITST

3.2.1.4.1.17.1 - Description

- a. Language - FORTRAN.
- b. How Invoked - Called by MMINON.
- c. Constraints and Limitations - None.
- d. Processing
 1. If the command is "TEST", reset the test flag, output message, and go to step 13.
 2. Check first "word" in command string against internal ASCII array.
 3. For "MDC", decode MDAC number from second "word", convert to HFC-HC number, and call MMICRT.
 4. For "HCS", decode HFC-HC number from second "word", convert to MDAC number, and call MMICRT.
 5. For "SEG", format and output a list of MDAC HC numbers in the segment indicated by the second "word" in the command string.
 6. For "MAP", set IERR = 3 (invalid command).
 7. For "ALW", if NWORD = 1, set source check to default; if NWORD = 2, check second word in string for "OCS" or "DAS", and set source check to their source numbers (for MMICLK).
 8. For "NAD", reset the address check flag for MMICLK.
 9. For "AIM", format and output message about AIMOKG.
 10. For "DIS", decode HFC-HC number and set SPARE5 (1) for STS.
 11. For "HFC", decode HFC number and create format and output a list of MDAC HC numbers on that HFC.
 12. For "EMS", if EMSEQG \neq \emptyset , set EMSEQG = \emptyset ; if EMSEQG = \emptyset , output a message.
 13. Return to calling routine.
- e. Error Messages and Recovery - Errors encountered in any of the above processing steps cause IERR to be set appropriately.

3.2.1.4.1.17.2 - Data, Logic, and Command Paths

See Figures 3.2.1-5 through 3.2.1-7 and Tables 3.2.1-IV through 3.2.1-VI.

3.2.1.4.1.16.3 - Internal Data Description

Miscellaneous array pointers, ASCII string for test commands, source names, and temporary values.

3.2.1.4.1.16.4 - Flowchart

Figure 3.2.1-29.

3.2.1.4.1.18 Submodule XVIII - MMIAID

3.2.1.4.1.18.1 - Description

a. Language - FORTRAN

b. How Invoked - Called by MMINON.

c. Constraints and Limitations - None.

d. Processing

1. If general help (NWORD = 1), write a list of commands to the logger, then go to Step 15.
2. If specific help (NWORD >1), loop through the words as below (steps 3 through 14). When done, go to step 15.
3. Check for possible command; if not a command, go to step 14.
4. Un-Can the full command name from KMANDS and load into O/P buffer.
5. Call MMICLK to fill OKSTUF.
6. Use OKSTUF to fill O/P buffer with a list of valid addresses and sources for this command.
7. Call MMICRT to O/P the buffer.
8. If this command needs extra help, encode the message and call MMICRT.
9. Go to step 3.

10. If not an address, go to step 13.
11. Encode the address message and call MMICRT.
12. Go to step 3.
13. Encode an error message and call MMICRT.
14. Go to step 3.
15. Return to calling routine.

- e. Error Messages and Recovery - "Words" in the command string which are neither valid commands nor addresses produce an error message "XXXX NOT A VALID COMMAND/ ADDRESS".

3.2.1.4.1.18.2 - Data, Logic, Command Paths

Input data: See figures 3.2.1-5 through 3.2.1-7 and Tables 3.2.1-IV through 3.2.1-VI.

3.2.1.4.1.18.3 - Internal Data Description

Contains ASCII message formats for errors and valid help responses.

3.2.1.4.1.18.4 - Flowchart

See figure 3.2.1-25.

3.2.1.4.1.19 Submodule XIX- MMIBCS

3.2.1.4.1.19.1 - Description

- a. Language - FORTRAN.
- b. How Invoked - Called by MMINON.
- c. Constraints and Limitations - None.
- d. Processing
 1. Set BCSBUF to -1's, BCSBUF(1) = BCS command number - 1.
 2. If command is "BCSSTART", check TBUSYG to see if the BCS task is already active; if not, activate BCS and return.
 3. For other commands, check TBUSYG to make sure BCS is active.

4. For "BCSXTAPE", "BCSMEAS", and "BCSEND", check to make sure no extraneous parameters are present, then go to step 14.

Other BCS commands are handled as follows:

5. "BCSFILE N" call MMINUM to decode the file number, check to make sure the file number is 1, 2, or 3, set BCSBUF(2) = file number, go to step 14.
6. "BCSDISPL ON/OFF" - check for 2 or 3 characters in second argument, CAN code second word; if "ON", set BCSBUF(2) = 1, if "OFF", set BCSBUF(2) = 0. Go to step 14.
7. "BCSRANGE R1, R2" - call MMINUM to decode the second and third words (range numbers). Check to make sure both numbers are between 1 and 60, then set BCSBUF(2) = lower range number, BCSBUF(3) = higher range number. Go to step 14.
8. "BCSINPUT HHHH" - call MMINUM to decode the HC number, and set BCSBUF(2) = HC number (MDAC).
9. "BCSOFFS MM/DD/YY, HH/MM/SS,OH,OV" - Call MMINUM to decode the date and time and set:

BCSBUF(2) = month
BCSBUF(3) = day
BCSBUF(4) = year
BCSBUF(5) = hours
BCSBUF(6) = minutes
BCSBUF(7) = seconds.

Call MMIHXD to decode the offset values, go to step 14.

10. "BCSCALC HHHH/INP/ALL" - call MMINUM to decode the (possible) HC number. If successful, set BCSBUF(2) = HC, and go to step 14. If not, check for "INP" or "ALL". If "INP", set BCSBUF(2) = 10,000; if "ALL", set BCSBUF(2) = 10,001 and go to step 14.
11. "BCSUPD Y/N" - If second word is "Y", set BCSBUF(2) = 1. If "N", set BCSBUF(2) = 0; go to step 14.
12. "BCSABORT" - call KILL4 to abort BCS. Go to step 15.
13. Engu BCSBUF to BCS (Queue 13).
14. Return to calling routine.

- e. Error Messages and Recovery - If any errors are encountered in the above processing, IERR and ONWORD are set appropriately.

3.2.1.4.1.19.2 - Data, Logic, Command Paths

Input data is A1 ASCII string. Output is IERR and ONWORD, or BCSBUF enqueued to the BCS task.

3.2.1.4.1.19.3 - Internal Data Description

MMIBCS contains CAN code representations of "ON ", "OFF", "INP", "ALL", "BCS", and "CMD", and ASCII representations of "Y" and "N".

3.2.1.4.1.19.4 - Flowchart

See figure 3.2.1-26.

3.2.1.4.1.20 Submodule XX - DSK

3.2.1.4.1.20.1 - Description

- a. Language - FORTRAN
- b. How Invoked - Activated by MMINON
- c. Constraints and Limitations - Data passed in DISKRG must be valid.
- d. Processing
 1. Upon activation, DSK checks DISKRG(1);
 2. If DISKRG(1) = 28, DSKAIM is called. Go to step 5.
 3. If DISKRG(1) = 29, DSKBIA is called. Go to step 5.
 4. If DISKRG(1) = other values, an error message is written to the system console.
 5. Call Exit.
- e. Error Messages and Recovery - See 3.2.1.4.1.20.1-d.4.

3.2.1.4.1.20.2 - Data, Logic, Command Paths

Input data is passed by global common DISKRG:

	UPAIM Command	UPBIAS Command
DISKRG(1):	28	29
DISKRG(2):	AIMPOINT Number	HFCHC Number
DISKRG(3):	Not Used	Bias AZ
DISKRG(4):		Bias EL

3.2.1.4.1.20.3 - Internal Data Description

None.

3.2.1.4.1.20.4 - Flowchart

See figure 3.2.1-27.

3.2.1.4.1.21 Submodule XXI' - DSKAIM

3.2.1.4.21.1 - Description

- a. Language - FORTRAN.
- b. How Invoked - Called by DSK.
- c. Constraints and Limitations - same as DSK.
- d. Processing
 1. "AIM" and scratch files rewound.
 2. Read card image record from mag tape: card type, MDAC HC number, Aimpoint number, Aimpoint coordinates (Lambert feet). If end of input, go to step 7.
 3. Check card type, HC number, Aimpoint number.
 4. Call DSKTRF to convert coordinates from Lambert feet to MMC meters (packed to 5 words).
 5. Call DSKUPD to write data to scratch file.
 6. Go to step 2.
 7. When input is finished, call DSKUPD to dump partial buffer.
 8. Call DSKBAK to update the aimpoint file (AIM) from scratch file data.
 9. Call DSKERR.
 10. Return to calling routine.

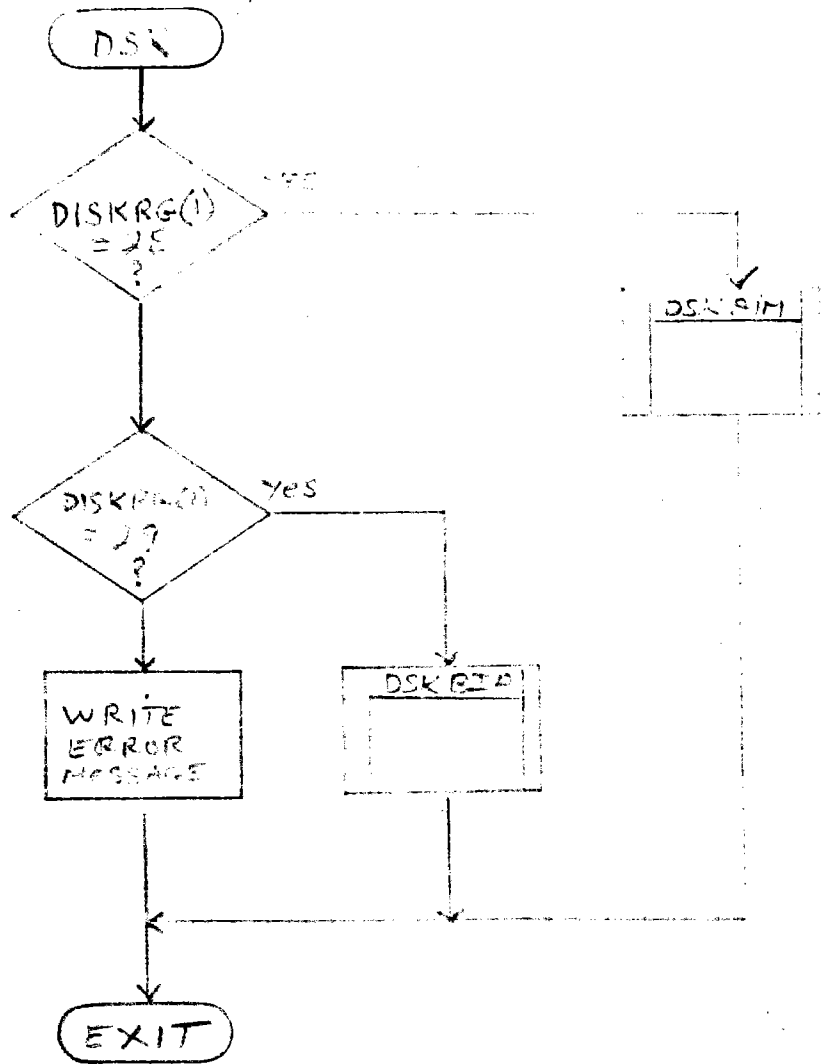


FIGURE 3.2.1-27 DSK FLOWCHART

- e. Error Messages and Recovery - Any error encountered during processing causes IERR to be set, and processing jumps to step 9, above.

3.2.1.4.1.21.2 - Data, Logic, Command Paths

Input data is DISKRG(2) = Aimpoint array number, and data from card type 4. Output data is updated AIM file.

3.2.1.4.1.21.3 - Internal Data Description

None.

3.2.1.4.1.21.4 - Flowchart

See figure 3.2.1-28.

3.2.1.4.1.22 Submodule XXII - DSKINK

3.2.1.4.1.22.1 - Description

- a. Language - FORTRAN.
- b. How Invoked - Called by DSKTRF.
- c. Constraints and Limitations - Row number must be 1 to 29.
- d. Processing - If $(\text{Aimpoint} - X)^2$ plus $(\text{Aimpoint} - y)^2$ is greater than .001, or Aimpoint Z is greater than ZHIGHT(ROW) or Aimpoint Z is less than ZLOWT(ROW), set IERR = 4; otherwise, set IERR = 0.
- e. Error Messages and Recovery - None.

3.2.1.4.1.22.2 - Data, Logic, and Command Paths

Row number of the HC is a passed parameter.

3.2.1.4.22.3 - Internal Data Description

- ZLOWT(29) - Acceptable lower limit for Aimpoint Z values in MMC meters.
- ZHIGHT(29) - Acceptable upper limit for Aimpoint Z values in MMC meters.

3.2.1.4.22.4 - Flowchart

See figure 3.2.1-29.

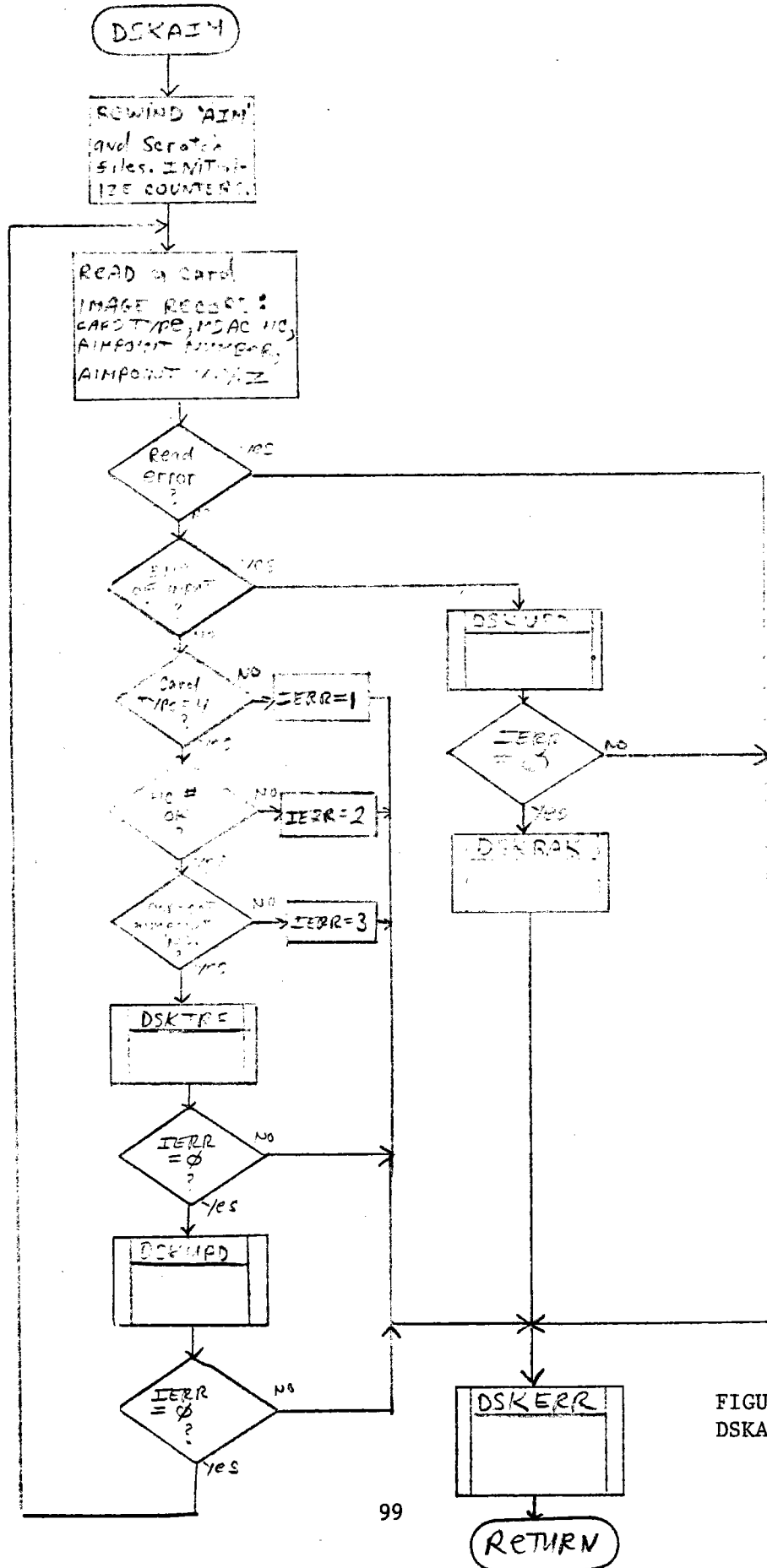


FIGURE 3.2.1-28
DSKAIM FLOWCHART

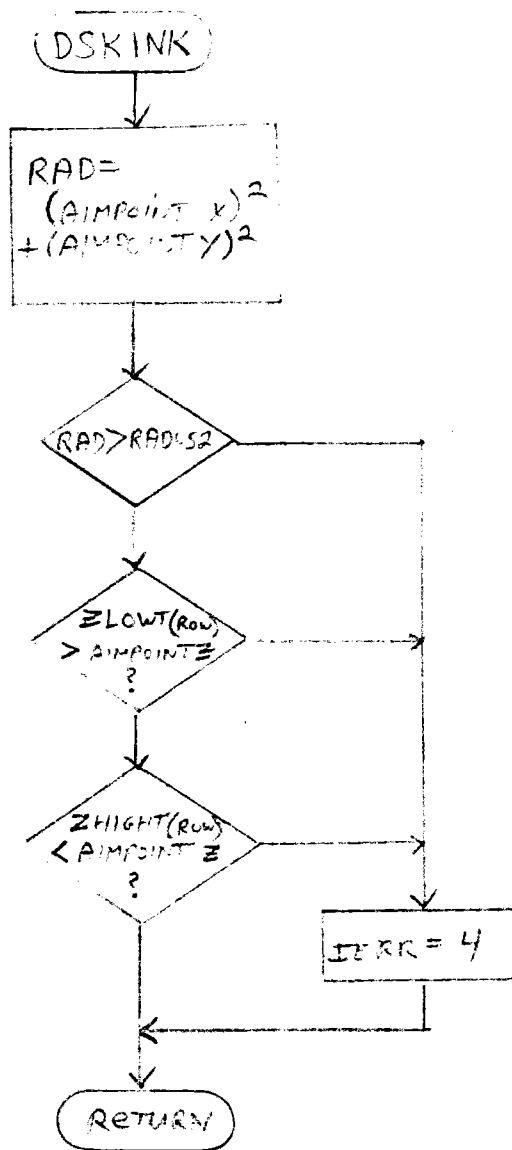


FIGURE 3.2.1-29 DSKINK FLOWCHART

3.2.1.4.1.23 Submodule XXIII - DSKTRF

3.2.1.4.1.23.1 - Description

- a. Language - FORTRAN.
- b. How Invoked - Called by DSKAIM.
- c. Constraints and Limitations - None.
- d. Processing
 1. Convert aimpoint X, Y, Z from Lambert feet to MMC meters.
 2. Pack converted values into 5-16 bit words.
 3. Add aimpoint array number to the last of the five words.
 4. Call DSKINK for inclusion area check.
 5. Return to calling routine.
- e. Error Messages and Recovery - None.

3.2.1.4.1.23.2 - Data, Logic, and Command Paths

To convert coordinates from MMC meters to 5-16 bit words:

1. Multiply by $2 \text{ EE } 18$.

```
IX = TEMP 1 * TWO18 + POINT5
IY = TEMP 2 * TWO18 + POINT5
IZ = TEMP 3 * TWO18 + POINT5
```

2. Extract left most 24 bits.

```
IX = IAND (IX, MASK)
IY = IAND (IY, MASK)
IZ = IAND (IZ, MASK)
```

3. Shift IY 24 bits right. Add to X to get IXYZ4(1).
Shift Y left 8 bits. Z right 16 bits.
And add to get IXYZ4 2).

```
IYSHFI = ISHFT (IY, -I24)
IXYZ4(1) = IX + IYSHFT
```

```
IYSHFT = ISHFT (IY . I8 )
IZSHFT = ISHFT (IZ . -I16)
IXYZ4(2) = IYSHFT + IZSHFT
```

4. Shift Z left 24 bits. Get IXYZ4(3).

IXYZ4(3) = ISHFT (IZ , I16)

All values are 32 bit integers, except TEMP, TWO18 and POINTS, which are 64 bit floating point:

TEMP = X, Y, Z in MMC meters.
TWO18 = 262144.
POINTS = .5

The 16-bit, 6-element array IP is equivalenced to IXYZ4, and the first 5 elements contain the packed coordinates.

3.2.1.4.1.23.3 - Internal Data Description

Contains values for coordinate (system) rotation (and translation), conversion from feet to meters, and those used for 5-word packing.

3.2.1.4.1.23.4 - Flowchart

See figure 3.2.1-30.

3.2.1.4.1.24 Submodule XXIV - DSKBIA

3.2.1.4.1.24.1 - Description

- a. Language - FORTRAN.
- b. How Invoked - Called by DSK.
- c. Constraints and Limitations - Same as DSK.
- d. Processing
 1. Calculate HCB record number from HFCHC number:
$$\text{Record \#} = (\text{HFCHC} - 1) / 64 + 1.$$
 2. Calculate word index to replace:
$$\text{WORD} = (\text{HFCHC} - (\text{Record \#} - 1) * 64) * 2 - 1.$$
 3. Read the HCB record.
 4. Replace WORD and WORD + 1 of the buffer with the AZ and EL values.
 5. Write buffer over old record.

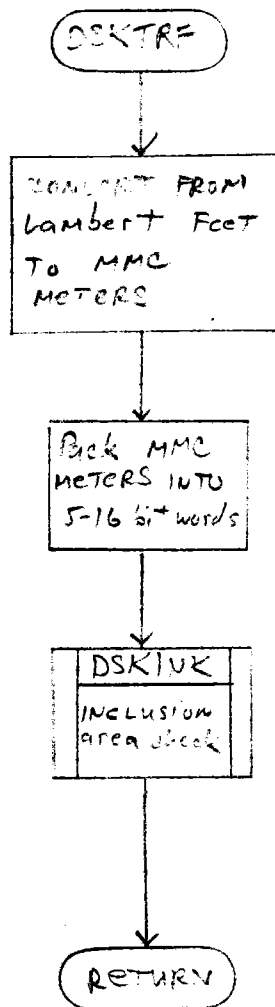


FIGURE 3.2.1-30 DSKTRF FLOWCHART

6. Encode and Enqueue message to CS console; write to logger.
 7. Call DSKERR.
 8. Return to calling routine.
- e. Error Messages and Recovery - Errors encountered cause IERR to be set, and processing jumps to step 7. Disk I/O errors also produce an error message on the system console.

3.2.1.4.1.24.2 - Data, Logic, Command Paths

Input data is via DISKRG. See DSK.

3.2.1.4.1.24.3 - Internal Data Description

None.

3.2.1.4.1.24.4 - Flowchart

See figure 3.2.1-31.

3.2.1.4.1.25 Submodule XXV - DSKERR

3.2.1.4.1.25.1 - Description

- a. Language - FORTRAN.
- b. How Invoked - Called by DSKBIA, DSKAIM.
- c. Constraints and Limitations - Same as DSK.
- d. Processing
 1. Encode an error message using (IERR +1) as an index.
 2. If there is a card input error, encode the input record number.
 3. Enqueue message to CS console.
 4. Return to calling routine.
- e. Error Messages and Recovery - Error messages produced are:

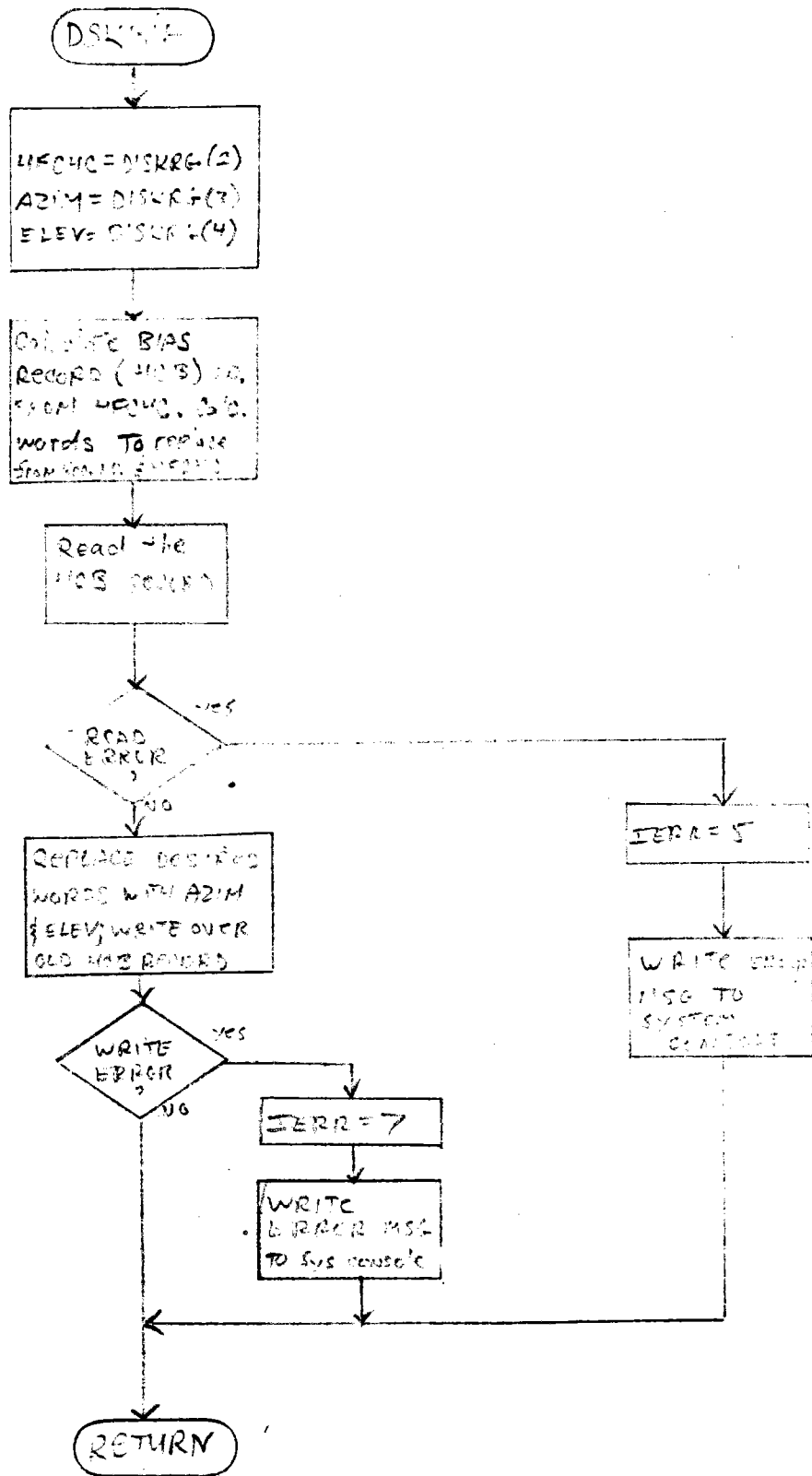


FIGURE 3.2.1-31 DSKBIA FLOWCHART

<u>IERR</u>	<u>MESSAGE</u>
0	"UPBIAS OKAY" (no errors)
0	"UPAIM OKAY" " "
1	"UPAIM WRONG CARD TYPE, RECORD N"
2	"UPAIM BAD HC NUMBER, RECORD N"
3	"UPAIM WRONG AIMPT #, RECORD N"
4	"UPAIM BAD AIMPT COORDS, RECORD N"
5	"UPAIM(BIAS) DISK READ ERROR"
6	"UPAIM(BIAS) DISK WRITE ERROR"
7	"UPAIM TAPE READ ERROR"

3.2.1.4.1.25.2 - Data, Logic, Command Paths

Input data is IERR.

3.2.1.4.1.25.3 - Internal Data Description

ASCII representation of the error messages.

3.2.1.4.1.25.4 - Flowchart

See figure 3.2.1-32.

3.2.1.4.1.26 Submodule XXVI - DSKBAK

3.2.1.4.1.25.1 - Description

- a. Language - FORTRAN.
- b. How Invoked - Called by DSKAIM.
- c. Constraints and Limitations - Must have valid data on scratch file.
- d. Processing
 1. Perform steps 2 through 7 for each scratch record written by DSKUPD.
 2. Read a scratch record. Perform steps 3 through 6 for each HC in the data buffer (up to 18).
 3. Extract the HFCHC number, aimpoint number, and 5 words of packed X, Y, Z data. See DSKUPD.
 4. Use HFCHC number as record number for AIM file, aimpoint number to calculate beginning of data on this record.

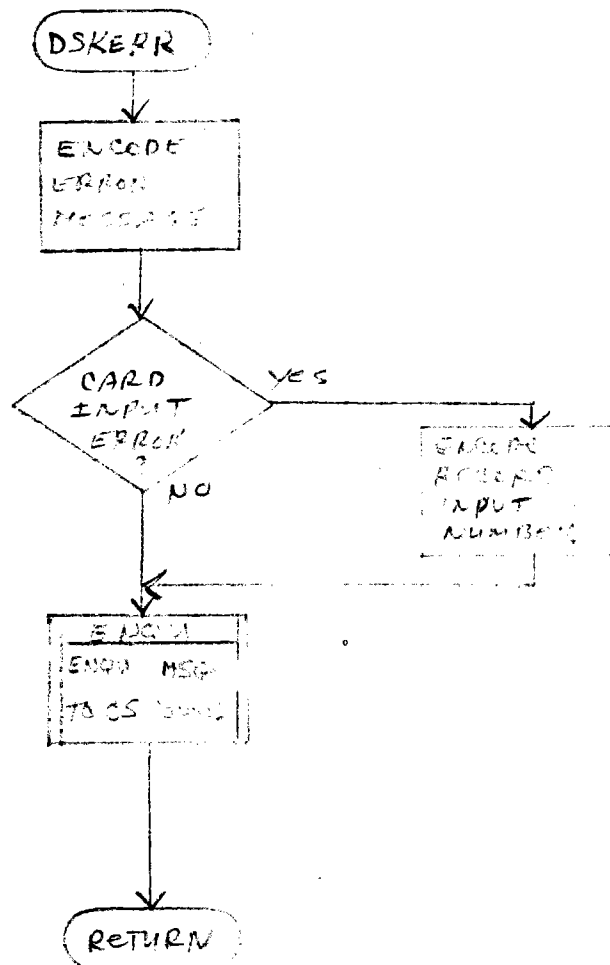


FIGURE 3.2.1-32 DSKERR FLOWCHART

5. Read this AIM record, replace aimpoint data, and write over old record.
6. Increment scratch buffer pointer.
7. Increment scratch record number.
8. Return to calling routine.

e. Error Messages and Recovery - Disk I/O errors cause IERR to be set, and processing jumps to step 8.

3.2.1.4.1.26.2 - Data, Logic, Command Paths.

See DSKUPD for the layout of a scratch file record.

3.2.1.4.1.26.3 - Internal Data

None

3.2.1.4.1.26.4 - Flowchart

See figure 3.2.1-33.

3.2.1.4.1.27 Submodule XXVII - DSKUPD

3.2.1.4.1.27.1 - Description

- a. Language - FORTRAN.
- b. How Invoked - Called by DSKAIM.
- c. Constraints and Limitations - None.
- d. Processing
 1. Load HFCHC number and aimpoint number. Save buffer.
 2. Load 5 packed words of X, Y, Z data.
 3. Increment save buffer pointer.
 4. If buffer not full, go to step 6.
 5. Write SAVE buffer to scratch and increment record number.
 6. Return to calling routine.

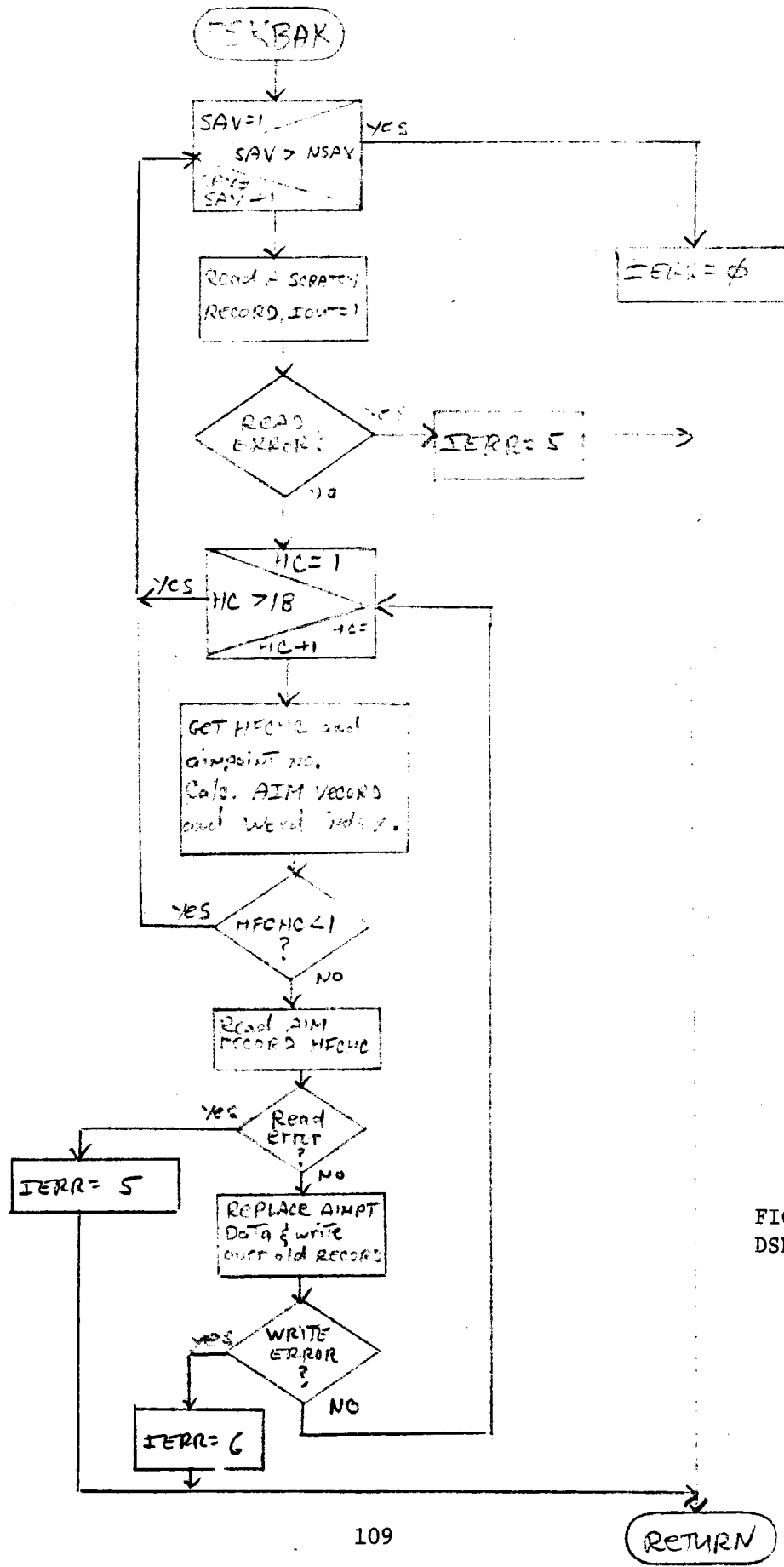
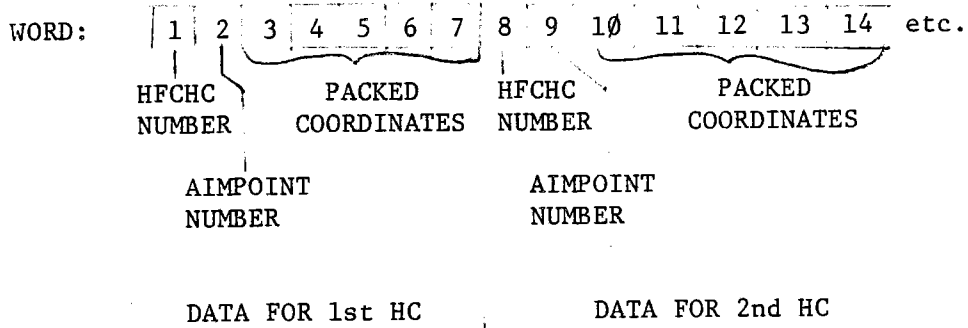


FIGURE 3.2.1-33
DSKBAK FLOWCHART

e. Error Messages and Recovery - Disk I/O errors cause IERR to be set.

3.2.1.4.1.27.2 - Data, Logic, and Command Paths

The 128 word SAVE buffer is used as follows:



Seven words are used for each HC, so the 128 words can contain data for up to 18 HCs. HFCHC number = -1 indicates end of data.

3.2.1.4.1.27.3 - Internal Data Description

None.

3.2.1.4.1.27.4 - Flowchart

See figure 3.2.1-34.

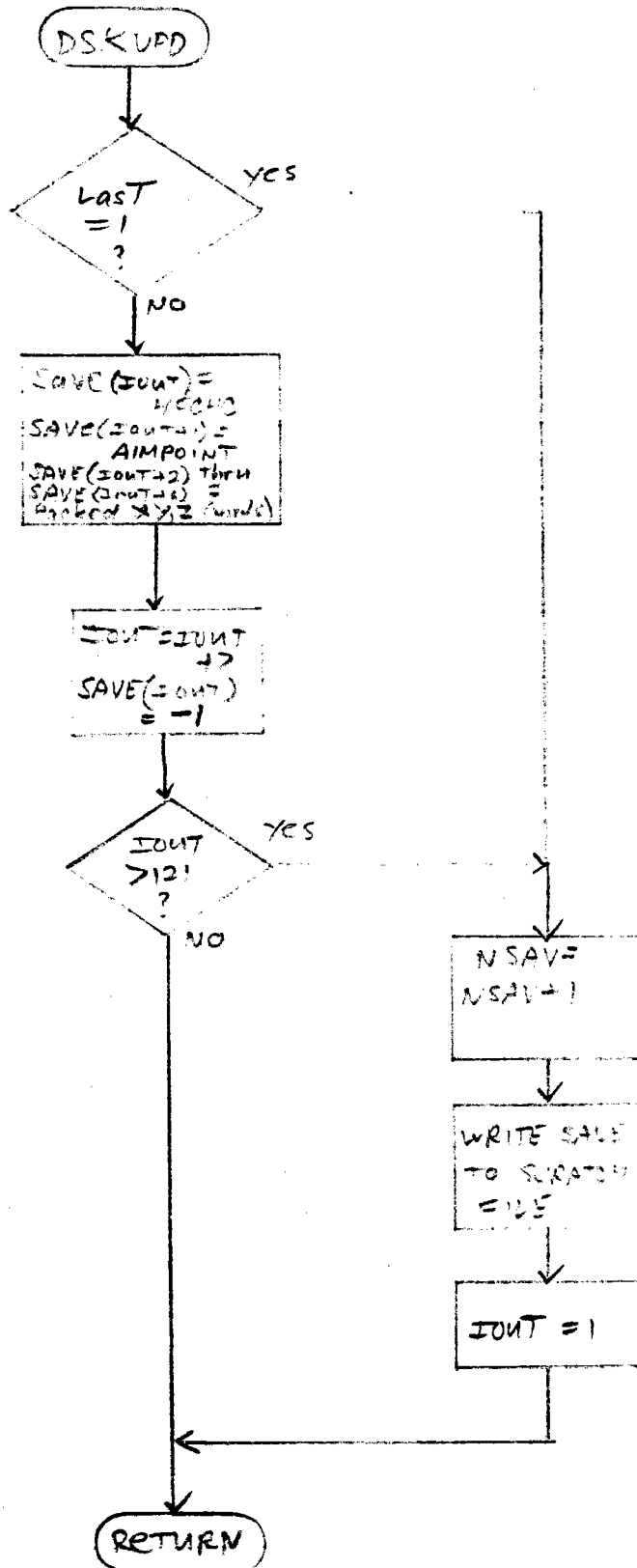


FIGURE 3.2.1-34 DSKUPD FLOWCHART

3.2.1.4.1.28 Submodule XXVIII - CFO

3.2.1.4.1.28.1 Description

- a. Language used - FORTRAN
- b. How invoked - activated by MMI
- c. Constraints and limitations - processing stops if an emergency sequence is in operation.
- d. Processing - CFO is the controlling program for the submodules used to process command files.
 1. CFO calls CFOPOS to position on the command file specified in CFNAMG.
 2. If no errors are encountered while positioning on the file, CFO calls CFOGET to read the record and enqueue the command string to MMI.
 3. CFO calls CFOMSG to report any error conditions or results to the CS console and then exits.
- e. Error messages and recovery - None (see CFOMSG)

3.2.1.4.1.28.2 Data, Logic and Command Paths

Input data:

CFWATG - global common, additional wait time
CFABOG - global common, abort flag
CFNAMG - global common, command file name
IERR - local common, condition codes

Output data:

None

3.2.1.4.1.28.3 Internal Data Description

The command strings will reside in the directoried SED source partition CFM. This partition will be sized at 100k bytes and contain a maximum of 100 source files.

Each record will have the following format:

SSSSCCCC....

where SSSS is the time delay in seconds,

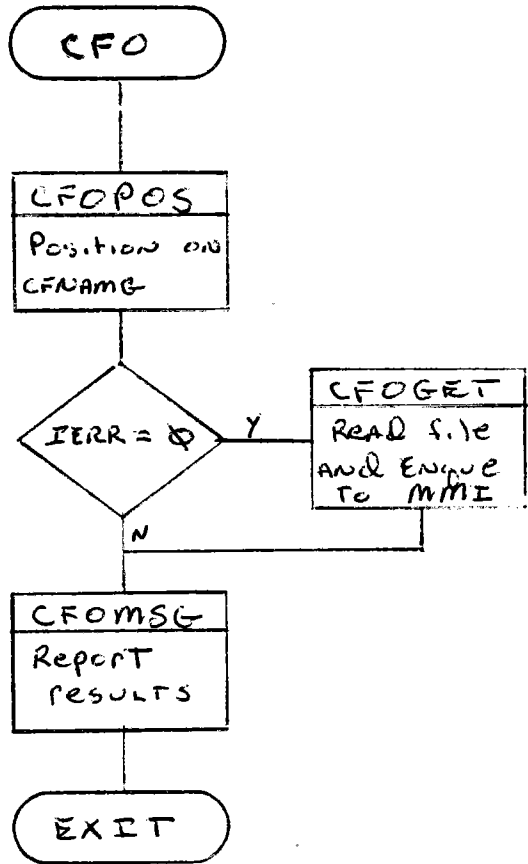
and CCCCC.... is the command string.

The right justified delay is the time to wait before implementing the command string CCCCC....

Each record may contain up to 80 bytes (characters): four bytes of time delay, and up to 76 for the command string.

3.2.1.4.1.28.4 Flowchart

See Figure 3.2.1-35 for the CFO flowchart.



3.2.1.4.1.29 Submodule XXIX - CFOPOS

3.2.1.4.1.29.1 Description

- a. Language used - FORTRAN and MODCOMP Assembler
- b. How invoked - Called by CFO
- c. Constraints and limitations - None
- d. Processing - CFOPOS is the submodule that positions on the file name specified in CFNAMG.
 1. CFOPOS reads the SED directory to find the file name. If the file name is not in the directory an error flag is set.
 2. If there is no directory in this partition another error flag is set.
 3. If the file name is found the starting record number is placed in the compressed read routine and CFOPOS returns to the calling program.
- e. Error messages and recovery - None (see CFOMSG)

3.2.1.4.1.29.2 Data, Logic and Command Paths

Input data:

CFNAMG - global
CFM - command file disk partition

Output data:

IERR - local common
WRECK - record number of file

3.2.1.4.1.29.3 Internal Data Description - None

3.2.1.4.1.29.4 Flowchart

See Figure 3.2.1-36 for the CFOPOS flowchart.

3.2.1.4.1.30 Submodule XXX - CFOGET

3.2.1.4.1.30.1 Description

- a. Language used - FORTRAN
- b. How invoked - called by CFO
- c. Constraints and limitations - None

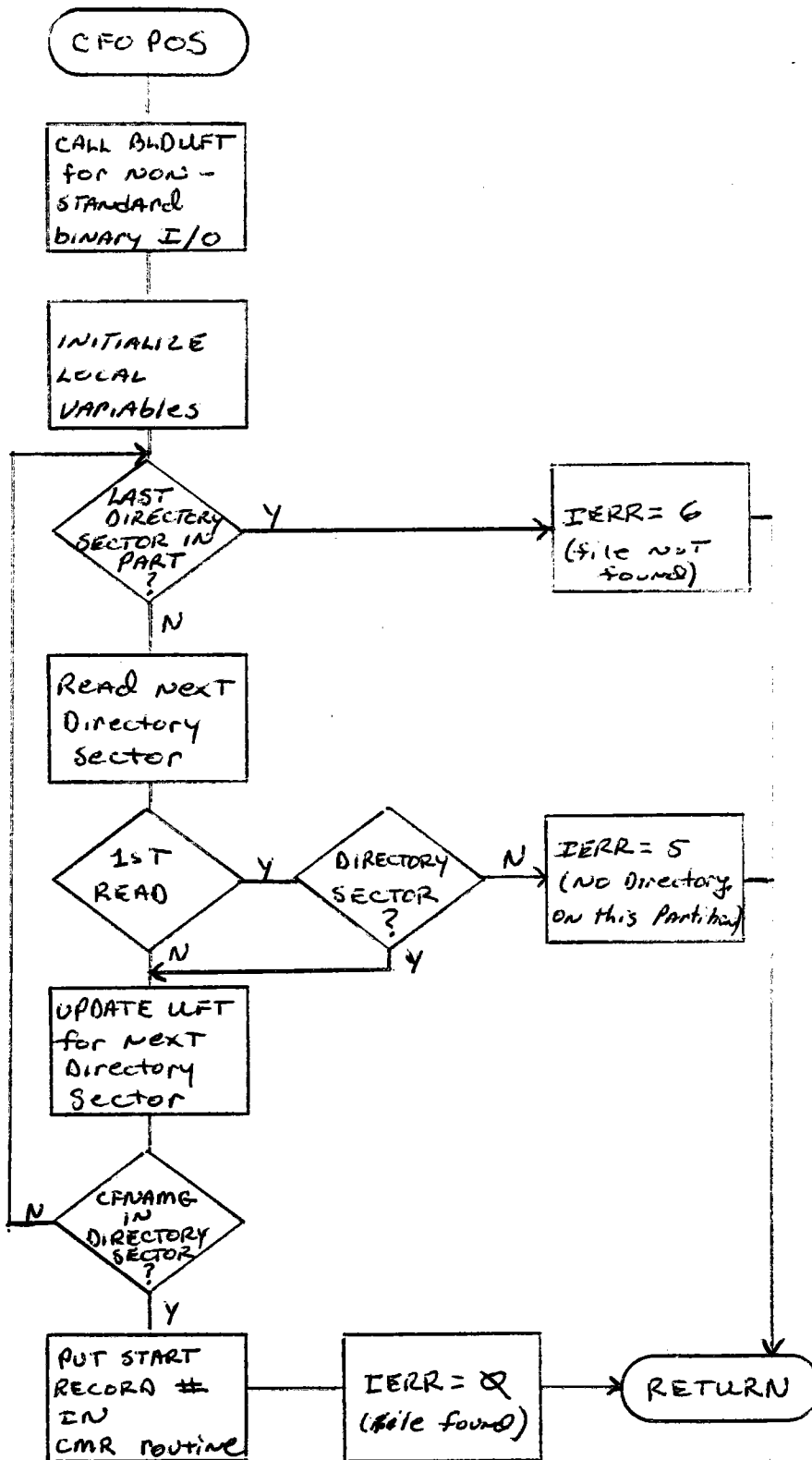


Figure 3.2.1-36 Flowchart - CFOPOS

- d. Processing - CFOGET reads the records from the file that has been positioned by CFOPOS. The reads are done using the compressed read routines.
 - 1. CFOGET decodes the delay time and if good computes a new total delay time before this message is to be sent to MMI.
 - 2. A message is sent to the CS console indicating the delay time before this command is to be executed.
 - 3. CFOGET then calls the wait routine to delay until the time has expired and then enques the command to MMI.
 - 4. Flags are reset and error flags set if necessary and CFOGET returns.

e. Error messages and recovery - None (see CFOMSG)

3.2.1.4.1.30.2 Data, Logic and Command Paths

Input data:

CFABOG - global common
 CFWATG - global common
 Records from disk file

Output data:

CFABOG - global common
 CFWATG - global common
 IERR - local common
 Enque to MMI

3.2.1.4.1.30.3 Internal Data Description

Total delay time computed from CFWATG and decoded delay time.

3.2.1.4.1.30.4 Flowchart

See Figure 3.2.1-37 for the CFOGET flowchart.

3.2.1.4.1.31 Submodule XXXI - CFOMSG

3.2.1.4.1.31.1 Description

- a. Language used - FORTRAN
- b. How invoked - Called by CFO and CFOGET
- c. Constraints and limitations - None

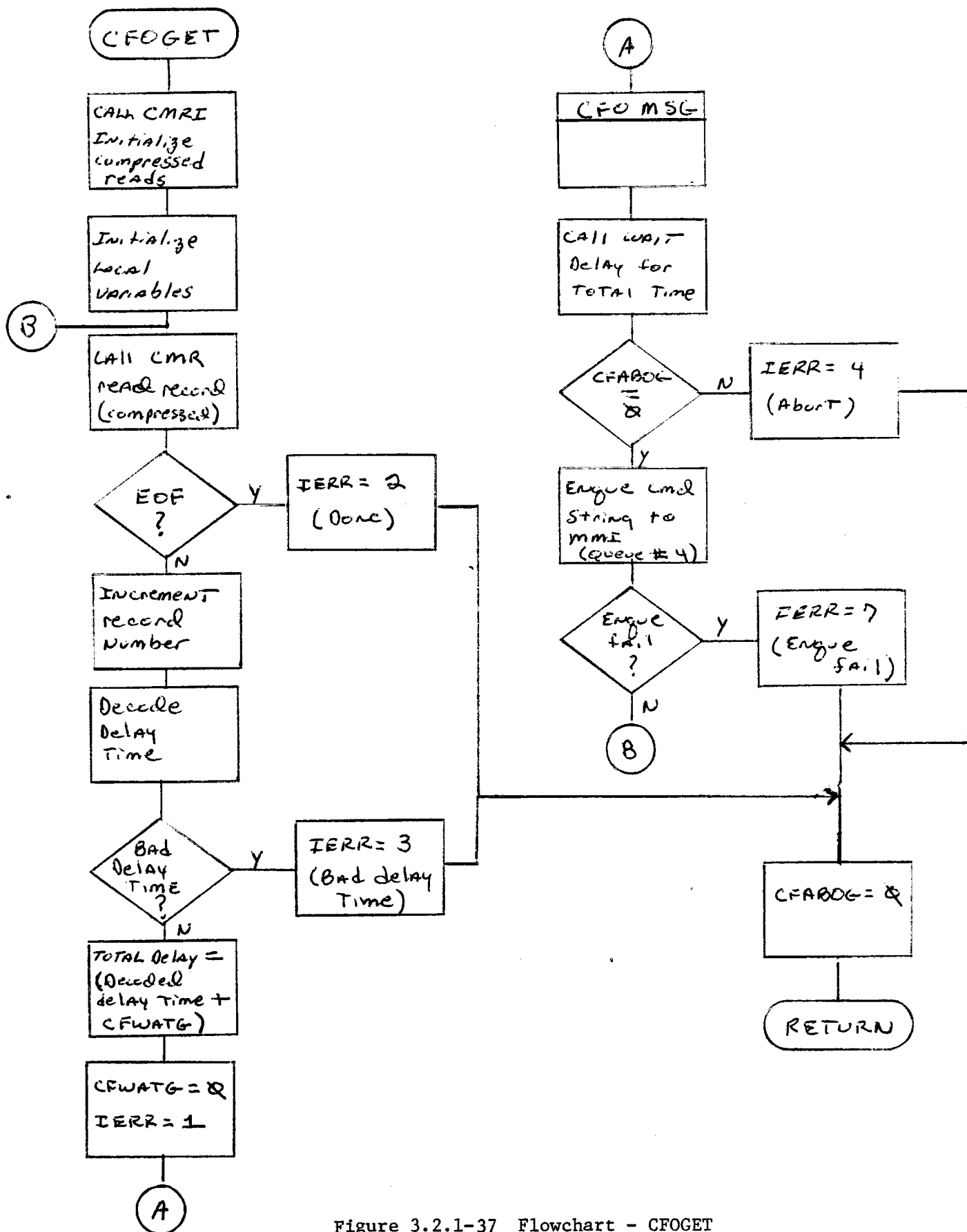


Figure 3.2.1-37 Flowchart - CFOGET

- d. Processing - CFOMSG is called to send informational and error messages to the CS console and logger.
1. CFOMSG determines if the message is informational or an error and sets the appropriate color.
 2. It then encodes the messages with the appropriate format text based on the data in IERR.
 3. The message is written to the CS console and logger. If the CS console enqueue fails, a message is written to the system console and CFOMSG returns to the caller.

e. Error messages and recovery -

IERR = 1 command ok display new wait time
 IERR = 2 command file done display records processed
 IERR = 3 bad delay time
 IERR = 4 CFABORT entered
 IERR = 5 no directory in the CFM partition
 IERR = 6 file entered not found
 IERR = 7 Enqueue to MMI failed

3.2.1.4.1.31.2 Data, Logic and Command Paths

Input data:

CFNAMG - global common
 IERR - local common
 TICKS - local common
 WRECK - local common

Output data:

IERR - global common
 Messages to CS console, logger, TTY

3.2.1.4.1.31.3 Internal Data Description

Format statements corresponding to the IERR indicators.

3.2.1.4.1.31.4 Flowchart

See Figure 3.2.1-38 for the CFOMSG flowchart.

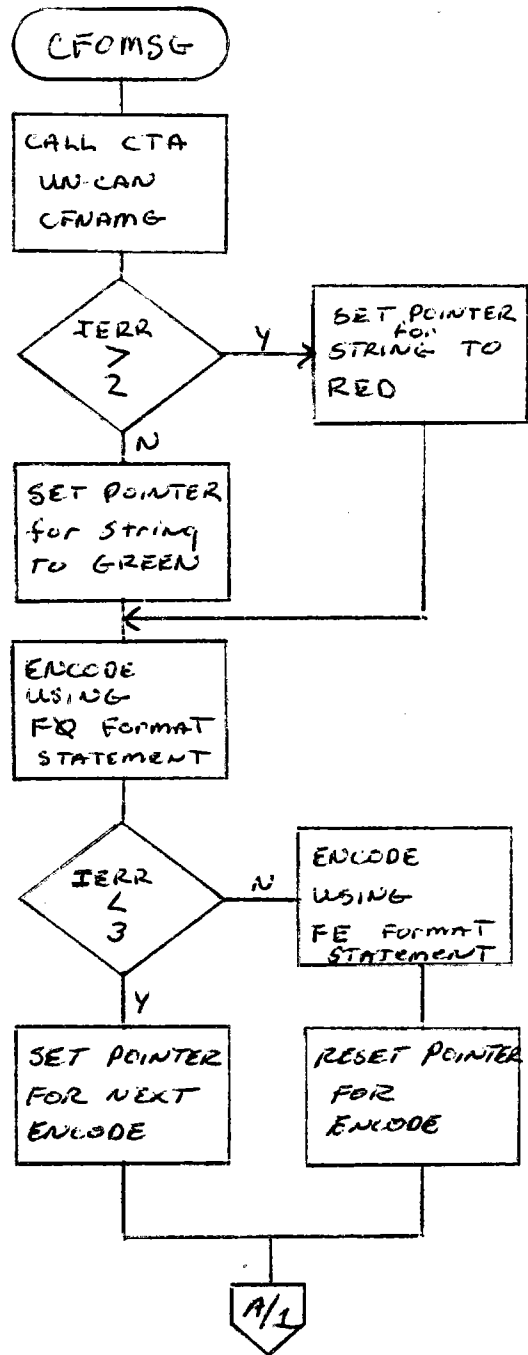


Figure 3.2.1-38 Flowchart - CFMSG

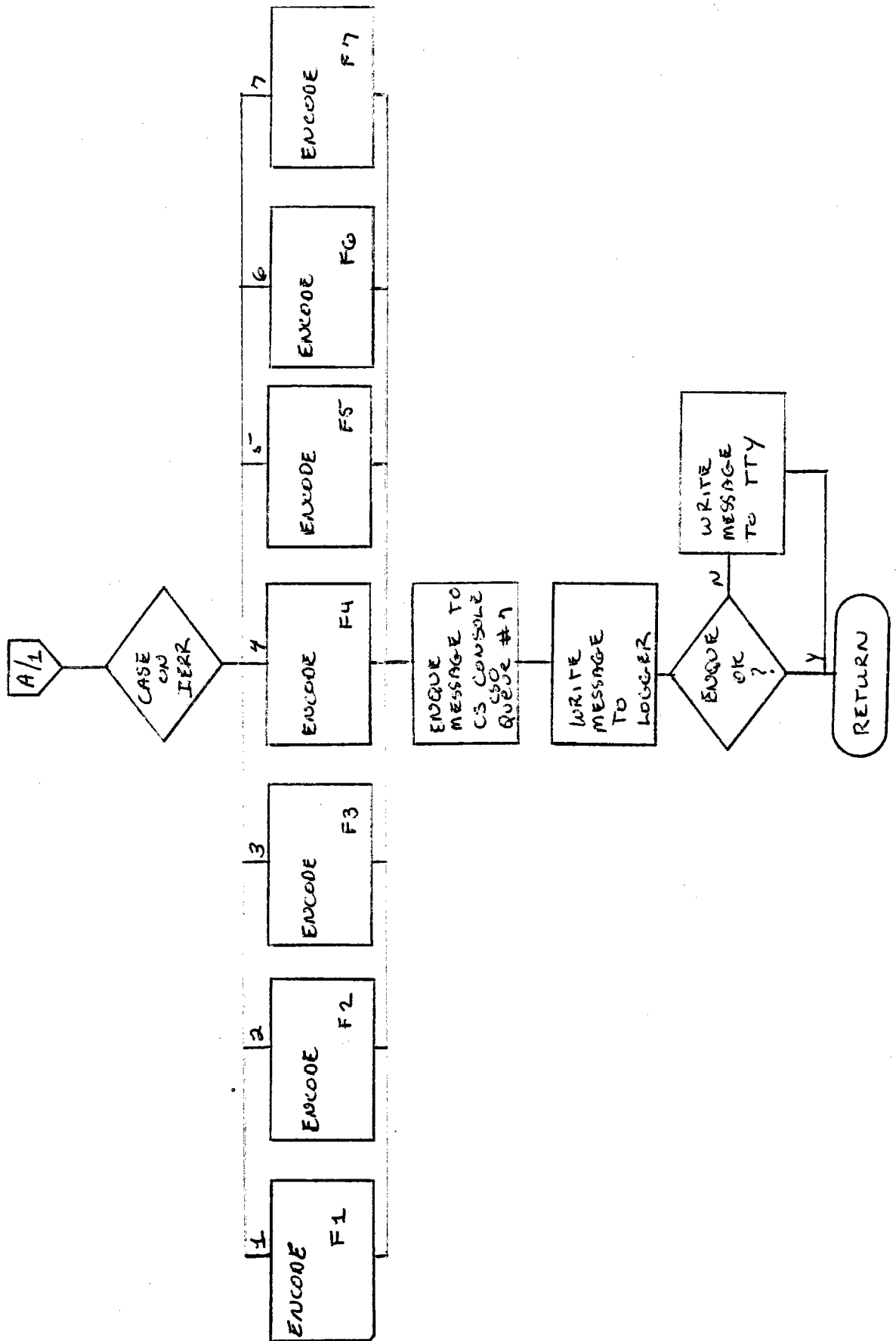


Figure 3.2.1-38 Flowchart - CFOMSG (Continued)

3.2.1.5

Interface Description

Input ASCII commands are enqueued to MANMIF through the five queues in MMI. Operator-entered commands are enqueued by the EXTINF module, command file commands by MANMIF submode CFO, and the Receiver Trip function may enqueue a DEFOCUS command. Responses to commands are enqueued to EXTINF by MMIRSP or MMIERR. Responses to commands are sent to the source of the command (except for emergency commands, where the response is sent to the source of the CFSTART command).

MANMIF communicates with the CMDPRC module via global words CPPG, HCMAPG, AND CPPRTG, and with the STATUS module through global words ISTATG. Communication with submodule DSK is via global words DISKRG, and with submodule CFO via global words CFABOG, CFWATG, and via initial enqueue.

3.2.1.6

Test Requirements

The MANMIF module's main routine, MMI, must be tested to be sure that each type of command is being properly processed by its submodules. The submodules should be tested as follows:

- MMIDEQ - Insure that proper values of NWORD and CHRPOS result for all types of character strings.
- MMICLK - Insure that commands and addresses are properly identified and that addressing and source allowability checks work properly.
- MMICMD - Insure that the all commands processed are valid commands.
- MMINUM - Insure that the validity checks for the numerical data work properly, and that the ASCII input data is correctly decoded.
- MMIHXD - Insure that all hex values are translated into proper decimal values.
- MMIMAP - Insure that HCMAPG has the proper format and the calculated heliostat numbers are correct.
- MMINON - Insure that the proper subroutines are called for the operational commands.
- MMISTR - Insure that global words ISTATG are properly set for all STATUS command formats.
- MMIST2 - Insure that special coded status requests are processed correctly.
- MMIST3 - Insure that special status "ER" is processed correctly.

MMIREF - Insure that the CS console is properly refreshed and that the status display is correct.

MMIRSP - Insure that the values returned in HCMAPG are properly displayed.

MMICRT - Insure that all output goes to the proper place.

MMIERR - Insure proper error message for a given error code number.

MMITST - Insure that all test commands are processed properly.

MMIAID - Insure that the information provided is correct for the request.

MMIBCS - Insure that all BCS commands are properly error-checked and passed to the BCS task properly.

DSK - Insure that the proper submodule is called.

DSKAIM - Insure that error detection and file updating is correct.

DSKINK - Insure that the inclusion area checks are done correctly.

DSKTRF - Insure that the transformation from Lambert to Martin Marietta Corporation internal is done correctly.

DSKBIA - Insure that the error detection and file updating is correct.

DSKERR - Insure that all errors detected are reported properly.

DSKBAK - Insure that the AIM file is updated correctly from the scratch file.

DSKUPD - Insure that the proper data and proper format is placed on the scratch file.

CFO - Insure that the command strings are properly enqueued to MMI and that the delay times are properly decoded.

CFOPOS - Position the file specified by scanning the SED Directory.

CFOGET - Read the positioned command file.

CFOMSG - Display informational and error messages.

3.2.2 Command Processor Module (CMDPRC)

3.2.2.1 Purpose

The purpose of the Command Processor Module is to convert high-level operator commands from the Man-Machine Interface Module (MANMIF) into explicit commands for the field of heliostats. These commands are output by the Field Communications Module (FLDCOM) to the field of heliostat controllers (HCs).

Via the flags and buffers that CMD, GET, and SEQ set for BHC, the Command Processor Module becomes the interface between certain asynchronous man-machine activities and the field-oriented synchronous ones.

3.2.2.2 Requirements

3.2.2.2.1 Design Requirements

The software requirements listed in Section 3.1 of the 10 MWe Collector Subsystem Software/Firmware Functional Requirements Specification, 12 June 1980 that apply to the CMDPRC module are:

- a. Control operational phase sequences as required for integrated control of a field of up to 2048 heliostats;
- b. Generate and transmit heliostat mode commands as required by the phase sequences during either operational or maintenance phases while maintaining safe beam control;
- c. Maintain safety through controlled beam movement and inclusion-area processing; (inclusion-area processing not applicable to this module);
- d. Transmit emergency DEFOCUS command upon receiver trip;
- e. Provide automatic HFC initialization and the capability to command heliostats away from the receiver following initialization within 60 seconds after power restoration.

3.2.2.2.2 Derived Requirements

The software requirements listed in Section 3.2.1.2 of the 10 MWe Collector Subsystem Software/Firmware Functional Requirements Specification, 12 June 1980, that apply to the CMDPRC module are:

- a. Check command arguments for reasonableness;
- b. Check heliostat's state for ability to execute commands;
- c. Translate operational commands and command sequences into individual HC commands;
- d. Maintain commanded heliostat mode table;
- e. Allow up to 16 command sequences at any one time;

- f. Obtain individual target or HC initialization information from disk; and
- g. Combine HC commands into HFC groups for faster throughput, whenever possible.

Further software requirements which are an expansion of those above and others, not stated, are:

- h. Allow the STHIWIND and HOLD command processing to rob heliostats from existing sequences;
- i. Ignore processing of heliostats which are involved in automatic operations except for STHIWIND;
- j. Allow only one emergency sequence (STHIWIND) at any given time;
- k. Provide feedback to MMI on the "degree of success" (how many HCs are in proper orientation) for each operator input command;
- l. Provide appropriate error-message feedback to MMI if no HCs can respond to the operator input command;
- m. Obtain orientation angles for Stow, Alt1stow, Alt2stow, and Wash positions from disk;
- n. Ability to stow and retrieve the field tracking configuration on disk;
- o. Terminate all sequences in the HAC upon detection of full-field power loss;
- p. Allow only one sequence to walk a given corridor at any particular time due to beam safety constraints;
- q. Reestablish management of current sequences in the backup HAC upon HAC failover;
- r. Provide a priority scheme for HC output commands;
- s. Determine eligible HCs in pecking-order-dependent operator commands; and
- t. Automatically reinitialize any HFC which reports a power loss.

3.2.2.3

Design Approach

The following is a list of assumptions used in the design of the CMDPRC module:

- a. Any data fetched from memory-resident or disk-resident files

has been checked for beam safety and/or reasonableness during off-line operations;

- b. All files referenced for HC data have been created and initialized;
- c. If task GET experiences any disk error, an alarm message is sent to Alarms Output task (ALO), the entire disk is considered unusable, and HAC failover flag is set.
- d. If a full-field power loss occurs and is then restored, it is assumed all sequences have been terminated prior to subsequent HC initialization.

Terms which are used in the following design approach are defined below:

- a. Orientation - the current mode/submode and position compare information which establishes a heliostat/s status;
- b. Command phase - generation of an HC output command based upon the input command and the HC's orientation;
- c. Sequence - the actions and/or necessary data required to move a set of HCs through one or more corridors to the desired destination;
- d. Sequence command - an input command possibly requiring more than one HC command (multiple command phases) to be sent to the field over a predetermined time interval (Example: track to stow via corridors);
- e. Critical (emergency) commands - those input commands which require quick movement (or cessation of movement) of HCs for safety purposes; these commands will have the "critical command" bit set in the HC's HCCMDG word; type 1 critical commands are DEFOCUS and STHIWIND; type 2 critical commands are HOLD and LOAD; a count representing the number of type 1 and type 2 critical commands outstanding is contained in EMCC1G(1) and EMCC1G(2), respectively;
- f. Software offline - the status condition produced when the operator wishes to isolate a normally-operating HC from further positioning (until an ONLINE command is entered for the HC);
- g. Field error offline - the status condition produced when the software detects an abnormal field status for an HC; the abnormal status bits in the HC's HCST2G word are: communication error, serious alarm, unmarked, and time-out (Note: the ONLINE command will reset all except the unmarked bit);
- h. Block - a set of HC numbers passed to task CMD from MMI

which represent an operator-addressable group of the entire field (i.e., arc, segment, wedge, ring); the degree of success in commanding each input block is reported back to MMI for each input command;

- i. Sequence state - sequences can exist in one of three states: non-corridor gather, corridor(s) management, and waiting for corridor(s);
- j. Non-corridor gather - the command phase of a sequence which requires collecting a group of HCs at the upper or lower-limit points of one or more corridors; the HCs are not moving within any corridors;
- k. Corridor management - the command phases of a sequence which require corridor-walk commands, HC timeout checks, and issuance of the appropriate beam-pointing commands to those HCs completing the corridor walks;
- l. Waiting for corridor(s) - the command phase of a sequence which examines the corridor status words (CORRSG) for all corridor(s) required; when all corridor(s) are free, the sequence state is changed to corridor management;
- m. Enque - operating system service call which allows one task to activate another and simultaneously pass a data buffer;
- n. Deque - operating system service call which allows one task to retrieve the contents of a data buffer passed by the task (using Enque);
- o. Internal HC numbers - a contiguous set of integers in the range: one is less than or equal to N which is less than or equal to 2048 used to index into data arrays; MDAC numbers input by operator(s) are translated into the internal numbering system by the MANMIF module.

3.2.2.3.1

Functional Allocations

The Command Processor Module is decomposed into the following functional areas:

- a. Check input command reasonableness and translate into individual HC commands;
- b. Fetch disk-resident data for certain input commands;
- c. Manage sequence commands;
- d. Monitor HFC status and reinitialize if necessary; and

- e. Build and combine HC commands on a priority basis for output to the field of HCs.

These functions (tasks) are shown in Figure 3.2.2-1.

The requirements for module CMDPRC are met by the five tasks shown in Figure 3.2.2-1. Their interrelationship is shown on the module overview, Figure 3.2.2-2. Only the most pertinent global common arrays are shown.

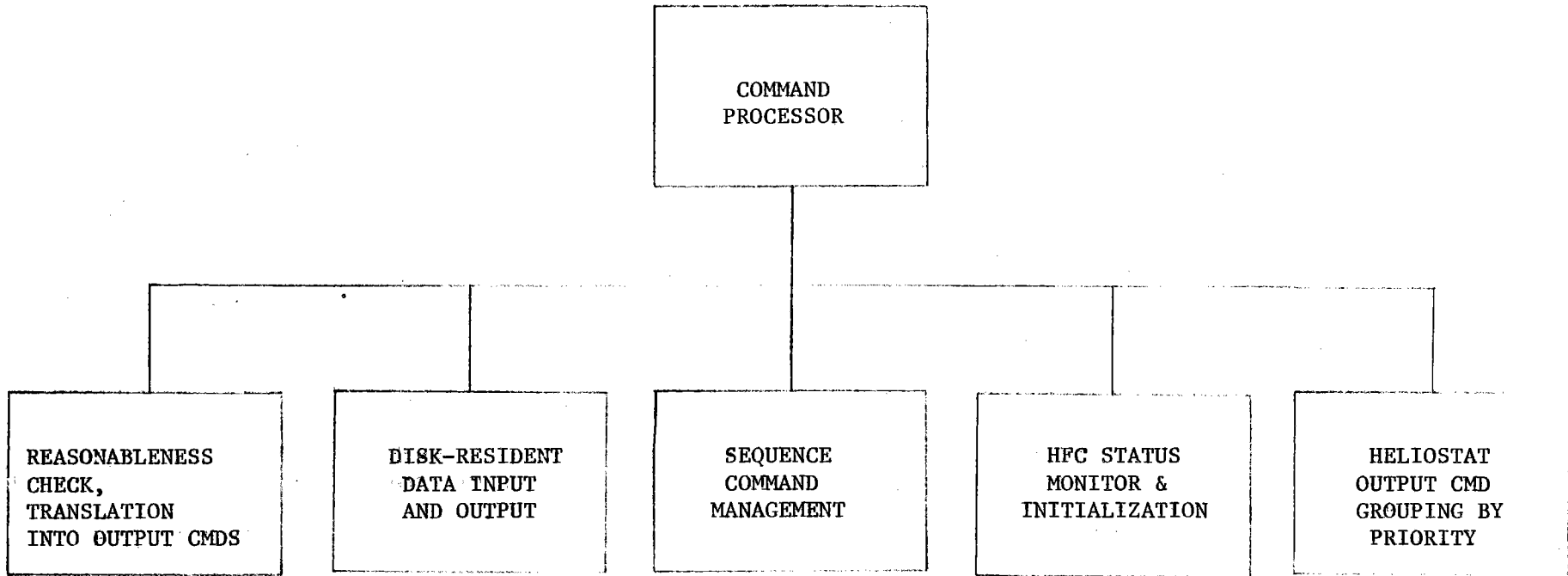
3.2.2.3.1.1

Task Activations

There are ten activations into the CMDPRC module:

- a. Module MANMIF activates task CMD whenever a new operator-input command needs to be processed; the MANMIF task (MMI) suspends itself until resumed by CMD;
- b. CMD activates task SEQ whenever an input command requires multiple HC commands (sequence); CMD builds the appropriate HC commands if some HCs involved can be moved immediately (not requiring corridor walks); CMD will not initiate any corridor walks itself;
- c. CMD activates task GET when an input command needs disk data to build HC commands or requires data to be saved on disk;
- d. SEQ reactivates itself periodically depending on sequence load and timing decisions;
- e. SEQ activates GET whenever a sequence has completed a corridor walk and now requires disk data for the final phase of HC commands;
- f. GET activates SEQ whenever the final HC commands have been prepared for a sequence and the sequence is to be deleted;
- g. Module FLDCOM activates task BHI every frame to determine if any HFC needs to be reinitialized;
- h. Module FLDCOM activates task BHC every frame to detect, build, and group (by HFC) all types of HC commands except sun-position data;
- i. Module MAXIVM (task FPL) activates SEQ to terminate all sequences when it detects a full-field power loss; and
- j. Module MAXIVM (task SWI) activates SEQ to reestablish sequence management whenever a HAC failover occurs.

Data transfer takes place via global common or disk files. Pertinent areas of global common are shown on the overview figure (Figure 3.2.2-2) as well as the specific disk files accessed by task GET.



129

Figure 3.2.2.-1 Hierarchy Diagram for the Command Processor Module (CMDPRC)

The functions of the five tasks in the CMDPRC module are:

- a. CMD - Evaluate operator input commands for reasonableness and executability; determine current orientation for all HCs involved in the input command; determine whether a sequence is required by examining the input command and current orientations; invoke the error exit if 16 sequences active and input command requires sequencing; rob HCs in an active sequence for the HOLD, LOAD and STHIWIND emergency input commands; respond to emergency DEFOCUS command; provide task BHC with a count of type 1 and 2 critical commands; build HC commands for those HCs which may be commanded immediately; prepare buffers to be used by tasks GET and SEQ; activate GET if HC command data is not memory-resident; allocate new sequences; activate SEQ if a new sequence is required; report back to the MANMIF module how many HCs were able to participate in the input command; report an error number to the MANMIF module if no HCs were able to participate in an input command. This task meets derived requirements "a" thru "e," and "h" thru "l," and "s."
- b. GET - Determine disk data request type and build appropriate HC commands after disk access; fetch HC Coordinates data and HC Bias data for LOAD commands; fetch Stow azimuth/elevation angles for final phase of STOW commands; fetch Alt1stow azimuth/elevation angles for final phase of ALT1STOW commands; fetch Alt2 stow elevation angle data from disk for final phase of ALT2STOW commands; fetch Wash azimuth/elevation angles for WASH commands; fetch aim-point data by aim-point array number for AIMPOINT commands; write field tracking configuration for SAVE commands; fetch field tracking configuration data for RESTORE commands; report any disk errors to the ALARMS module via global common; activate SEQ after building HC commands for final phase of sequence commands. This task meets derived requirements "c," "d," "f," "m," and "n."
- c. SEQ - Accept (Deque) new sequences from CMD; determine whether a sequence must wait for corridors or join the corridor management group of sequences which are actively moving and monitoring their set of HCs; recompute the minimum expected time and maximum expected time associated with monitoring an active sequence; determine which non-waiting sequence must be analyzed during "visit"; gather HCs at their respective CULPs or CLLPs; tag any HC which does not report position-compare by the sequence phase maximum time (set HC's timeout bit); build HC commands for initiating corridor walks; reactivate itself depending on sequence loading and timing decisions; activate GET

whenever disk data is needed to complete the sequence; reorder the active-sequence list whenever a sequence has been completed and deleted; examine waiting sequences to detect one or more candidates eligible for insertion into the active-sequence list; manage lists of HCs belonging to the individual sequences; allow deletion of all sequences upon detection of a full-field power loss; allow assumption of control of existing sequences upon HAC failover. This task meets derived requirements "b" thru "e," and "o" thru "q."

- d. BHC-Scan HC-Command array for new HC commands; act on any type 1 or type 2 commands first; build and combine HC commands for the appropriate HFCs; format these similar commands for output transmission by task FCP. This task meets derived requirements "g" and "r."
- e. BHI-Scan HFC status array to detect any HFCs requesting an initialization series of commands; relieve the operator of the HFC reinitialization duties. This task meets derived requirement "t."

3.2.2.3.2

Resource Budgets

This section represents currently available module information.

a. Memory requirements:

- 1. CMD - 8K
- 2. GET - 7.75K
- 3. SEQ - 8K
- 4. BHC - 2.5K
- 5. BHI - 2K

b. Timing:

- 1. CMD - asynchronous upon operator input
- 2. GET - asynchronous from CMD or SEQ
- 3. SEQ - asynchronous from CMD or GET; periodic when active sequences dictate
- 4. BHC - synchronous after FCP outputs commands available in global common buffer CMDBFG (55 msec steady state)
- 5. BHI - synchronous after FCP processes last status response (5 msec steady state)

c. Disk files:

- 1. HC Coordinates (HCC) - 20,480 bytes, 64 records
- 2. HC Biases (HCB) - 8,192 bytes, 32 records
- 3. Aim-point file (AIM) - 524,288 bytes total, 2,048 records (words 1-100 of each record)
- 4. Alt1stow AZ/EL angles (AL1) - 8,192 bytes, 32 records

5. Alt2stow EL angle (AL2): 8,192 bytes, 32 records
6. Wash AZ/EL angles (WSH): 8,192 bytes, 32 records
7. Tracking Configuration Snapshot (AIM): 524,288 bytes, 2048 records (words 123-128 of each record) same file as item 3 above.
8. Stow AZ/EL angles (STO): 8,192 bytes, 32 records

d. Disk accesses:

GET - maximum is 4096 for field addressing LOAD command

e. Magnetic tape access - none

f. Task priorities:

The tasks within the Command Processor module are responsible for qualifying and translating operator commands into HFC-compatible commands. They need a priority number just below that of the synchronous timing and field input/output tasks due to command through-put requirements. CMD has higher priority than GET or SEQ because the MMI task is suspended until CMD's completion.

1. CMD - 115
2. GET - 120
3. SEQ - 125
4. BHC - 65
5. BHI - 60

3.2.2.4 Design Description

3.2.2.4.1 Module Structure

The structure of module CMDPRC is depicted in the overview figure (Figure 3.2.2-2). The five tasks communicate via scheduling (MAX IV REX Activate or Enque or its FORTRAN equivalent), global common, and various operational disk files. These tasks and their associated submodules are briefly described below:

a. CMD - Command processor task (see Figure 3.2.2-3)

1. CMDSCK - orientation test control
2. CMDSAL - sequence allocation
3. CMDTRK - track-the-target processor
4. CMDDSK - disk packet build
5. CMDPOS - AZ/EL-pointing processor
6. CMDSBY - track-the-CULP processor
7. CMDSDN - down-corridor sequence preprocessor

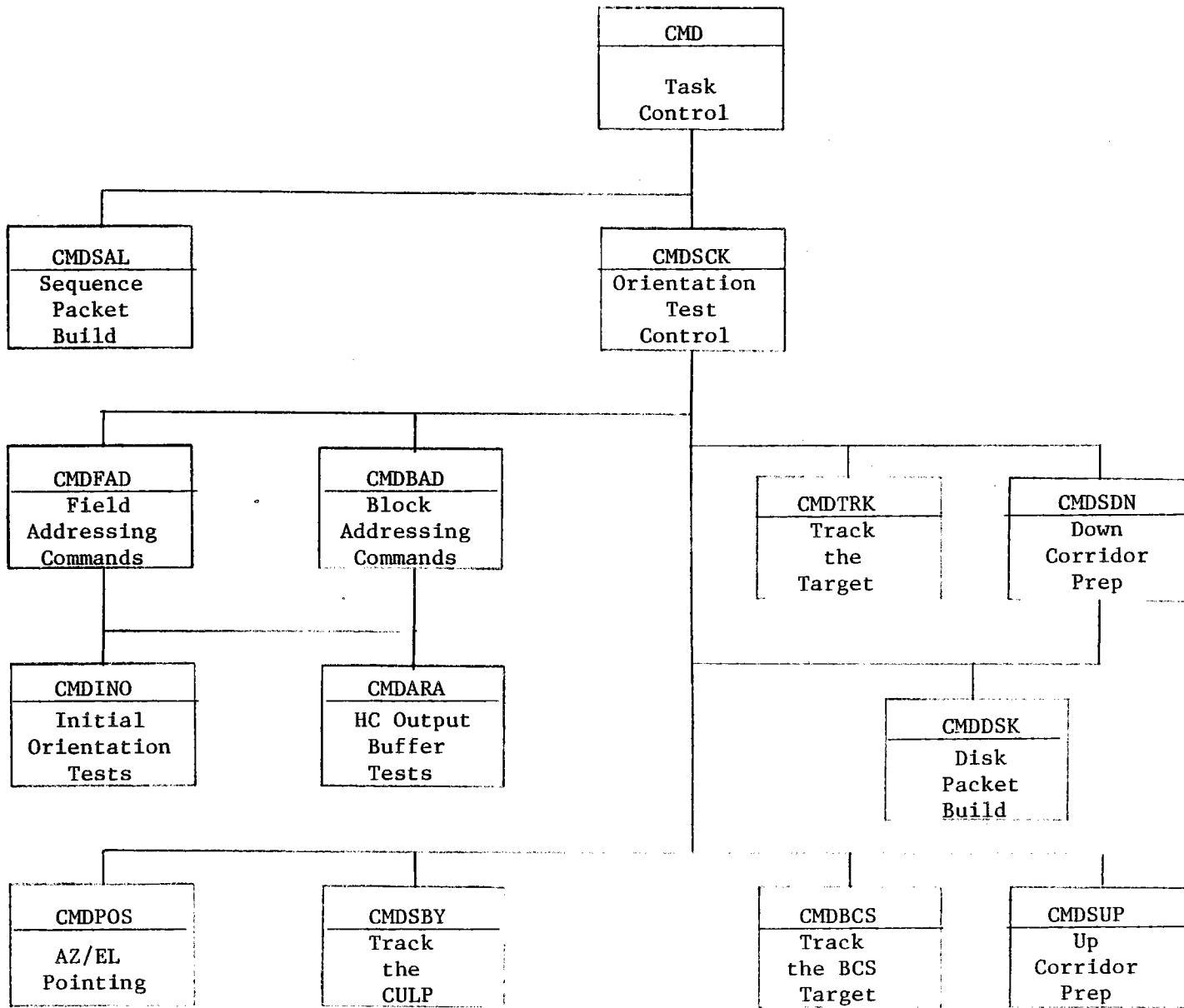


Figure 3.2.2-3 CMD Hierarchy

8. CMDBCS - track the BCS-target processor
9. CMDSUP - up-corridor sequence preprocessor
10. CMDFAD - field addressing commands
11. CMDBAD - block addressing commands
12. CMDINO - initial orientation tests
13. CMDARA - HC output buffer tests
14. CMDSPB - sequence packet build
15. BLKCMD - block data initialization file for LOCCMD
16. LOCCMD - local common for task CMD

b. GET - Disk data acquisition task (see Figure 3.2.2-4):

1. GETAZL - azimuth/elevation angle processor
2. GETXYZ - coordinates processor
3. GETSAR - tracking configuration save and restore
4. GETDSK - disk I/O processor
5. LOCGET - local common for task GET

c. SEQ - Command sequence control task (see Figure 3.2.2-5):

1. SEQGAT - HC-gathering processor
2. SEQBPT - Beam-pointing processor
3. SEQCCK - Corridor availability test
4. SEQCOR - Corridor-walk processor
5. SEQADD - Add sequence to active list
6. SEQRLK - Relink sequence in active list
7. SEQDEL - Delete sequence from active list
8. BLKSEQ - Block data initialization file for LOCSEQ
9. LOCSEQ - local common for task SEQ

d. BHC - Task to build and combine HC commands into HFC packets by priority levels (see Figure 3.2.2-6)

e. BHI - Automatic HFC initialization task (see Figure 3.2.2-6):

BHISH5 - Bit-shift processor for building HFC initialization commands

3.2.2.4.1.1 Submodule I - CMD

3.2.2.4.1.1.1 Description

CMD is the submodule (task) which gains control when an operator input command is to be processed for possible HC output commands. CMD accepts operator input commands in binary format from task MMI of the MANMIF module. All addressing parameters have been translated into their respective internal HC numbers (1-2048). Commands specifying "field" addressing have no HC numbers passed to CMD.

- a. Language used - FORTRAN IV
- b. How invoked - activated by MMI to process all operational commands (MMI is suspended until CMD completes).
- c. Constraints and limitations - None

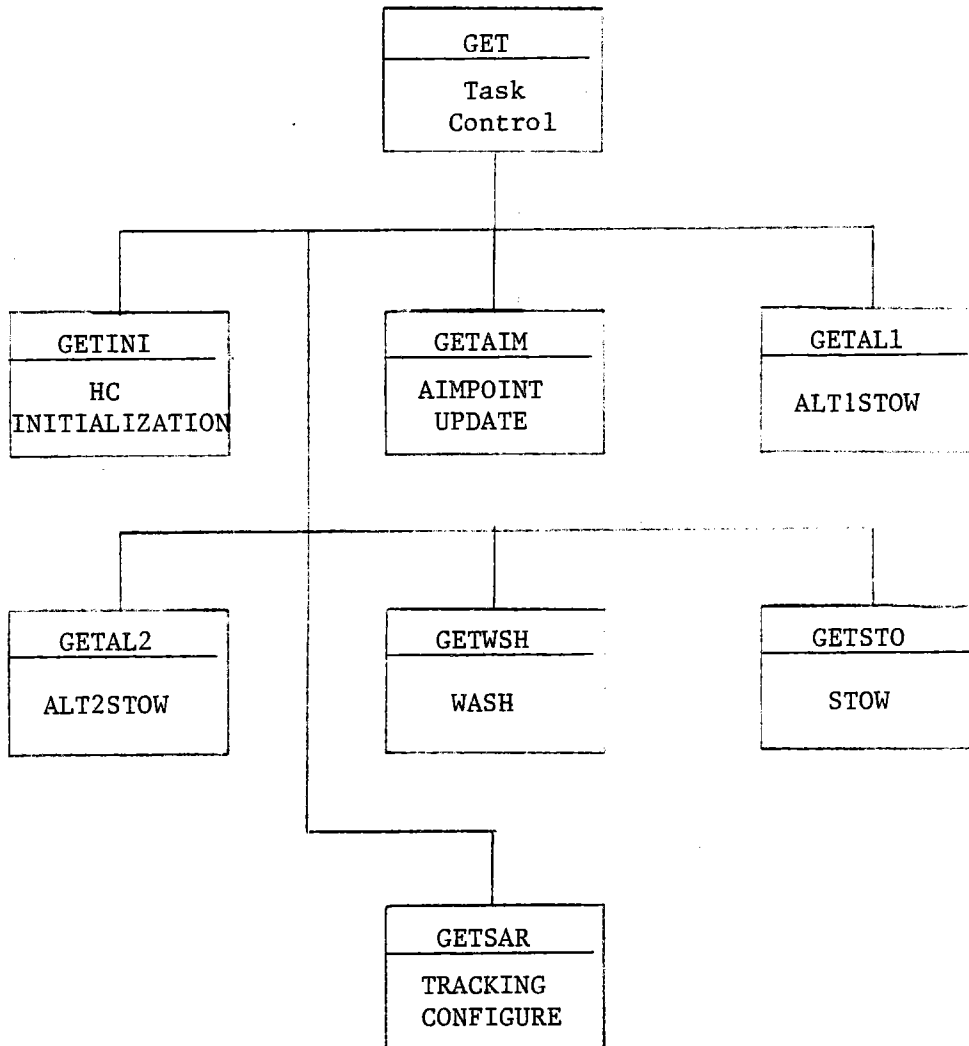


Figure 3.2.2-4

Task GET Hierarchy

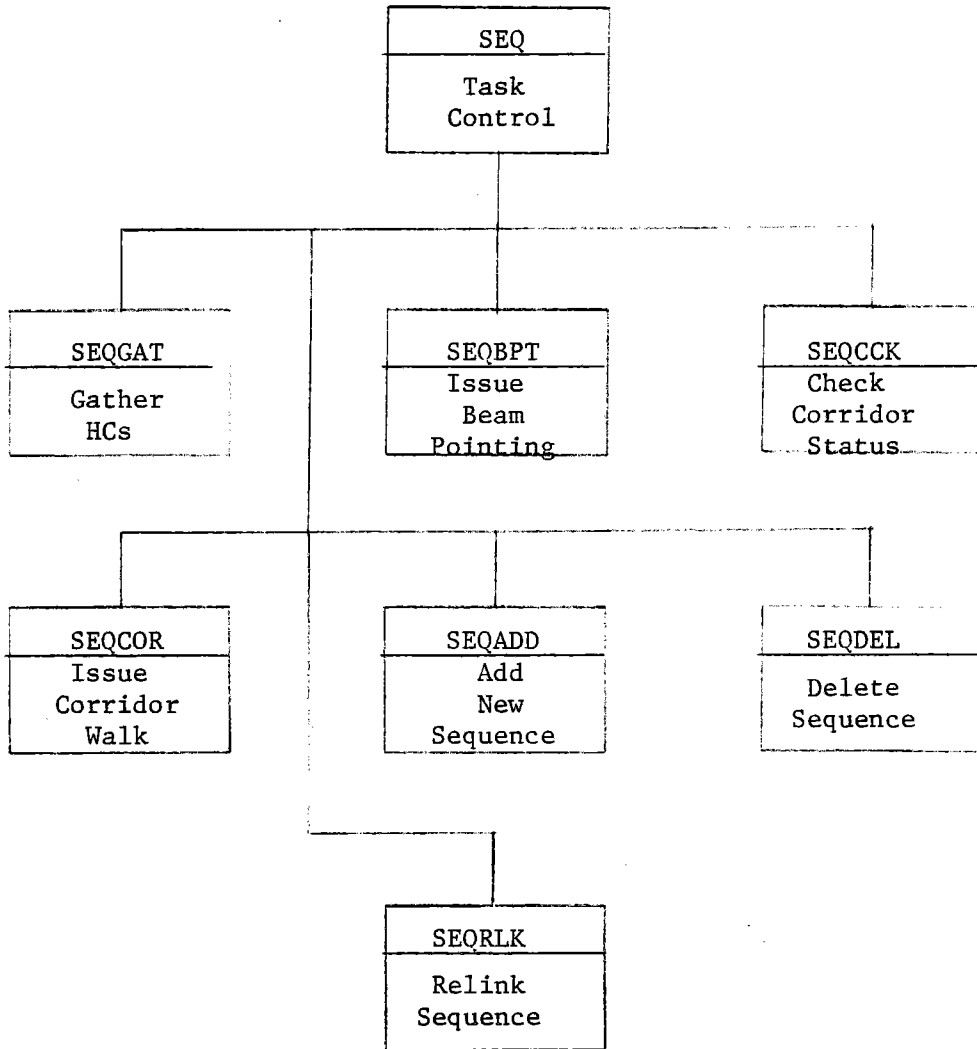


Figure 3.2.2-5 Task SEQ Hierarchy

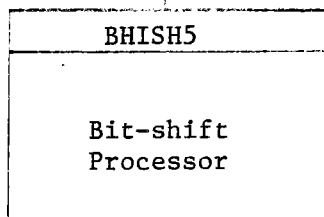
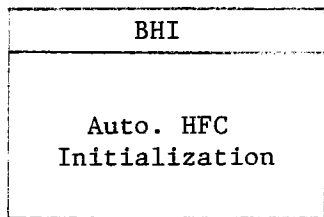
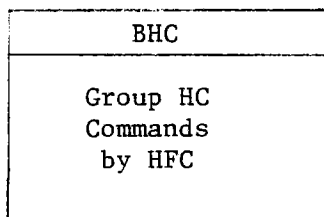


Figure 3.2.2-6 Tasks BHI, BHC Hierarchy

d. Processing -

1. Call SETBF to set CMD's task busy bit. Initialize high-level parameters by zeroing the MMI error flag (CPPRTG), the disk data required flag (DISDAT), and the sequence required flag (SEQREQ).
2. Check if a field power fail situation is present. If not, go to step (3). If the command is LOAD, ESTANDBY or ESTOW (allowable commands during field power failure), go to step (3). Otherwise, zero the overall success count for MMI, set the MMI error flag = 8, and go to step (7).
3. Call CMDSCK to determine which HCs qualify for the input command, whether a sequence is required, and whether any disk data is required.
4. Upon return, check if the MMI error flag is nonzero; if so, go to step (6). Check if a sequence is required; if not, go to step (6). Call CMDSAL to allocate a sequence and to build the appropriate sequence command and data buffers.
5. Upon return, check if a sequence is still required; if not, go to step (6). Check if the MMI error flag is nonzero; if so, go to step (6). Call ENQU to enqueue a sequence command message to task SEQ using queue 10 of the COMQUE global area (see Figure 3.2.2-7). If the enqueue is successful, go to step (6). Call MSSG4 to issue following message to the system console:

ENQUE FAILURE: TASK CMD

and continue to step (6).
6. If disk data is not required, go to step (7). Call ENQU to enqueue a disk data command message to task GET using queue 9 of the COMQUE global area (see Figure 3.2.2-7). If the enqueue is successful, go to step (7). Call MSSG4 to issue following message to the system console:

ENQUE FAILURE: TASK CMD

and continue to step (7).
7. Call RESUME to resume task MMI. Call SETBT to clear CMD's task busy bit. Call EXIT to terminate task.

CMD Error Code & Location

1	(See note, offset = 10000)	HC not in correct orientation
2	(See note, offset = 20000)	HC offline
3	(See note, offset = 30000)	HC not installed
4	CPPRTG	BCSTRACK allows only 1 HC per BCS target at a time
5	CPPRTG	Sequence disallowed; only 16 sequences possible at a time
8	CPPRTG	Sequence disallowed; power fail flag set

NOTE: HCMAPG contains applicable internal HC #s for non-field addressing. Add an error-indicative offset # to the HC # if it cannot respond to the input command.

Table 3.2.2-I CMD Error Return Codes

- e. Error messages and recovery - MMI error code of 8 is set if command is not allowed during power failure. A message is sent to the system console if an enqueue to either SEQ and/or GET is unsuccessful.

3.2.2.4.1.1.2

Data, Logic and Command Paths

See glossary for the meaning of data names.

Input data:

a. Global common:

- 1. CPPG(1)
- 2. CPPRTG
- 3. CPUSG
- 4. IQUEUE

b. Local common:

- 1. SEQREQ
- 2. DISDAT
- 3. SEQBFR
- 4. SEQSIZ
- 5. DSKBFR
- 6. DSKSIZ

Output data:

a. Global common:

- 1. CPPRTG
- 2. TBUSYG
- 3. HCMAPG(1)

b. Enqueue of task SEQ (see Figure 3.2.2-7)

c. Enqueue of task GET (see Figure 3.2.2-7)

d. Local common:

- 1. SEQREQ
- 2. DISDAT

3.2.2.4.1.1.3

Internal Data Description

None

3.2.2.4.1.1.4

Flowchart

See Figure 3.2.2-7 for the CMD flowchart.

3.2.2.4.1.2

Submodule II - CMDSCK

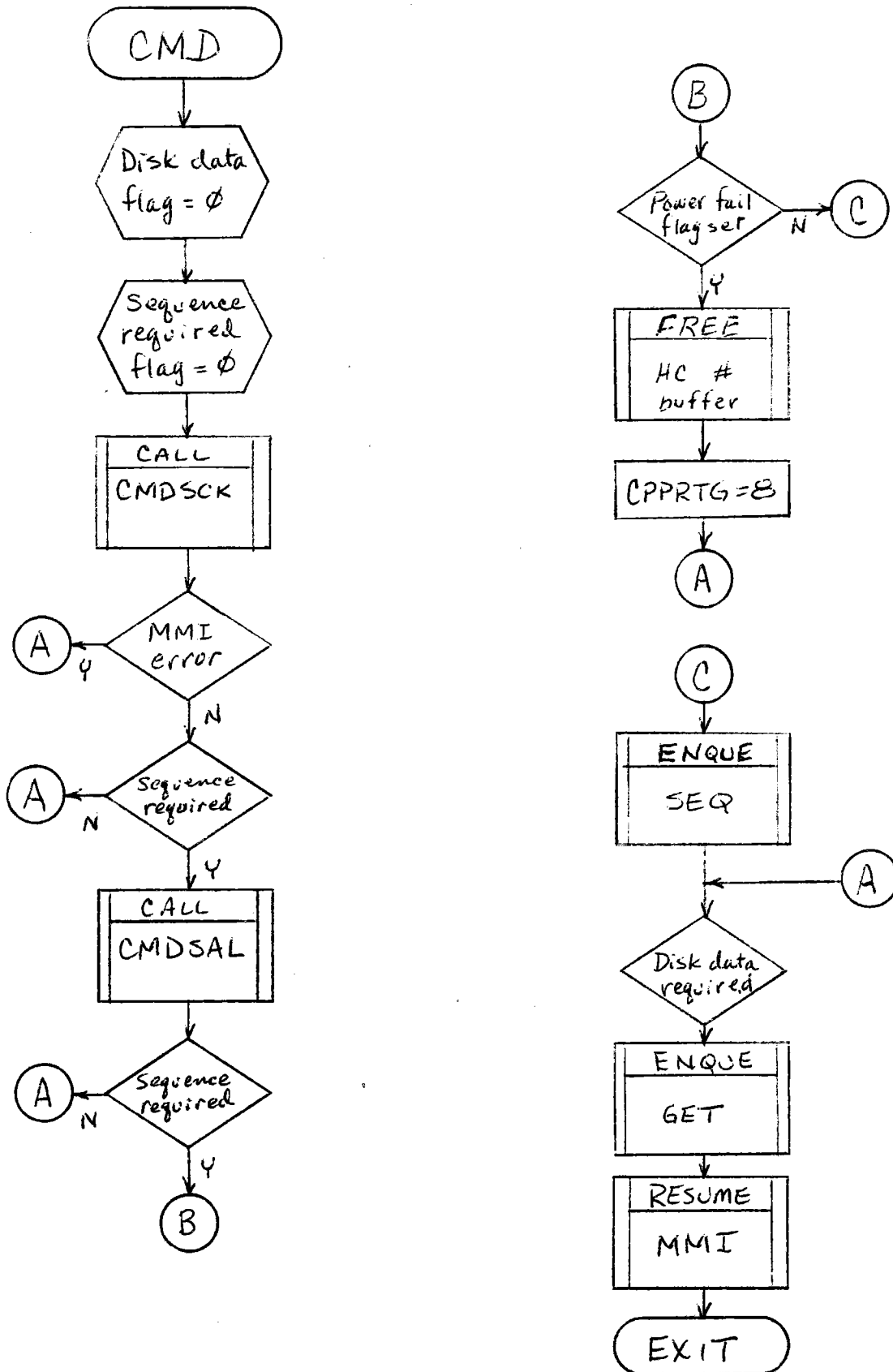


Figure 3.2.2-7 Flowchart - CMD

3.2.2.4.1.2.1 Description

CMDSCK is responsible for determining type of input command to be processed, how to size the HC number buffers (IMARAY, SQARAY, SBARAY, COARAY), which type of addressing is specified, how many HCs are able to respond, whether a sequence is required, which output command processing submodule to call, and setting the MMI error return word CPPRTG.

- a. Language used - FORTRAN IV
- b. How invoked - called by CMD
- c. Constraints and limitations -
 1. Accepts only those operational commands listed in Table 3.2.2-II.
 2. Allows up to 16 sequences simultaneously.
 3. Allows only STHIWIND, LOAD, and HOLD to rob HCs from sequences.
- d. Processing -
 1. Initialize all appropriate flags and counters.
 2. Determine the offset into the initial orientation array (INORIE) based upon the input command number. Determine how many valid orientations will not require a sequence and how many will require a sequence.
 3. There are up to four HC number buffers which must be sized for an input command. Commands which do not require a sequence only use the first buffer (immediate array). Commands which require a non-emergency sequence (STOW, ALT1STOW, and UNSTOW) will possibly need to use the first three buffers (immediate, pre-corridor movement, and the HCs-at-CULP arrays). STHIWIND requires an emergency sequence and will possibly need to use all four buffers (immediate, pre-corridor movement, HCs-at-CULP, and HCs-in-corridors arrays). Use the LMODE array to find which MODEG words to access. Each buffer size calculation requires a unique subset of MODEG values to be added together. When a buffer size is available, it is used to calculate the pointer to the next buffer area, if applicable.
 4. Determine the input addressing type in CPPG(2). See Table 3.2.2-___. If field addressing, call submodule CMDFAD. For block addressing, call submodule CMDBAD.

<u>Command No.</u>	<u>Command</u>	<u>Acceptable Abbreviation</u>	<u>Valid Initial Orientations</u>
1	TRACK	(TRAC)	Standby
2	INCREASE	(INCR)	Standby
3	LOAD	(LOAD)	All
4	MARK	(MARK)	Stow
5	STANDBY	(STAN)	Track
6	STOW	(STOW)	Initialization Alt1stow Position Mark Standby Track
7	POSITION	(POSI)	Initialization Stow Position Standby Alt1stow Mark Alt2stow Transition
8	ONLINE	(ONLI)	Offline
9	OFFLINE	(OFFL)	Stow Alt1stow Alt2stow Position
10	DECREASE	(DECR)	Track
11	BCSTRACK	(BCST)	Standby
12	RETURN	(RETU)	BCS
13	WASH	(WASH)	All except offline and auto-BCS BCSTRACK, TRANS to BCSTRACK
14	ALT1STOW	(ALT1)	Stow Position Standby Track
15	ALT2STOW	(ALT2)	Position Standby
16	RESTORE	(REST)	All

Table 3.2.2-II Operational Commands and Their Valid Initial Orientations

<u>Command No.</u>	<u>Command</u>	<u>Acceptable Abbreviation</u>	<u>Valid Initial Orientations</u>
17	UNSTOW	(UNST)	Stow Alt1stow Alt2stow
18	STHIWIND	(STHI)	Initialization Alt1stow Position Mark CLLP Standby Track BCS Corridor walk In transition to Wash } Position compare or in transition
19	DEFOCUS	(DEFO)	Track: at or approaching the target
20	AIMPOINT	(AIMP)	All
21	HOLD	(HOLD)	Any transition: position compare false
22	SAVE	(SAVE)	Track Standby
23	RELWASH	(RELW)	Wash
24	ESTANDBY	(ESTA)	Initialized (sent HC a 'LOAD') and orientation at power fail time was: Track: at or approaching the target, or Standby: at or approaching the CULP
25	ESTOW	(ESTO)	Initialized (sent HC a 'LOAD') and orientation at power fail time was: In corridors, or at or approaching the CLLP

Table 3.2.2-II Operational Commands and Their Valid Initial Orientations (cont'd)

5. A success count is returned from these submodules. If count is zero, go to step (10).
6. Check if a sequence is required; if not, go to step (9). Check if there are less than 16 sequences currently active; if so, go to step (8). Check for STHIWIND command; if not, go to step (7). If so, call MSSG4 to issue following message to the system console:

STHIWIND AND 16 SEQUENCES

and go to step (7).
7. Zero the sequence required flag, set the internal error flag to five, and go to step (9).
8. Check for STHIWIND command; if not, go to step (9). Set the emergency sequence flag and zero the BCS busy word.
9. Rewind the dynamic HC number buffer pointers to the values calculated in step (3). Submit the input command number to a CASE statement to determine the proper submodule for generating output commands to the field. The submodule branch decisions are:
 - a. Call CMDTRK for TRACK or INCREASE (command numbers 1 or 2);
 - b. Call CMDDSK for LOAD, WASH, ALT2STOW, SAVE, AIMPOINT, RESTORE, or ESTOW (command numbers 3, 13, 15, 16, 20, 22, or 25);
 - c. Call CMDPOS for MARK, POSITION, ONLINE, OFFLINE, HOLD, or RELWASH (command numbers 4, 7, 8, 9, 21, or 23);
 - d. Call CMDSBY for STANDBY, DECREASE, RETURN, DEFOCUS, or ESTANDBY (command numbers 5, 10, 12, 19, or 24);
 - e. Call CMDSDN for STOW, ALT1STOW, or STHIWIND (command numbers 6, 14, or 18);
 - f. Call CMDBCS for BCSTRACK (command number 11); and
 - g. Call CMDSUP for UNSTOW (command number 17).
10. Set the MMI error return flag to the value of the internal error flag and return to CMD.

- e. Error messages and recovery - Only error detection is for a sequence required but 16 sequences are already active. See steps (6) and (7) for details.

3.2.2.4.1.2.2 Data, Logic and Command Paths

See Glossary for the meaning of data names.

Input data:

a. Global common:

1. MODEG
2. CPPG(1)
3. HCMAPG
4. HCST1G
5. SEQNMG

b. Local common:

1. IOPTR
2. INORIE
3. MODPTR
4. LMODE
5. IMCNT
6. KIN
7. LIN
8. SQSPTR
9. SBSPTR
10. COSPTR
11. CNT
12. FLDORI
13. IMSIZE
14. SQSIZE
15. SBSIZE
16. SEQREQ
17. EFLAGI

Output data:

a. Global common:

1. EMSEQG
2. BCSBYG
3. CPPRTG

b. Local common:

1. EFLAGI
2. IMCNT

3. SQCNT
4. IMSIZE
5. SQSIZE
6. SBSIZE
7. COSIZE
8. CNT
9. FLDORI
10. KIN
11. LIN
12. SEQREQ
13. IMDPTR
14. SQSPTR
15. SQDPTR
16. SBSPTR
17. SBDPTR
18. COSPTR
19. CODPTR

3.2.2.4.1.2.3 Internal Data Description

There is no data internal to this submodule.

3.2.2.4.1.2.4 Flowchart

See Figure 3.2.2-8 for the CMDSCK flowchart.

3.2.2.4.1.3 Submodule III - CMDFAD

3.2.2.4.1.3.1 Description

CMDFAD is responsible for initial processing of input commands using field addressing. Because the HCMAPG array is not used for this type of addressing, every installed HC is initially considered as involved. Those HCs which are able to respond to the input command are placed in the appropriate HC-number array by submodule CMDARA. Only one word in HCMAPG is set for the input command's success count.

- a. Language used - FORTRAN IV
- b. How invoked - called by CMDSCK
- c. Constraints and limitations - None
- d. Processing -
 1. Perform DO-UNTIL processing where HC equals

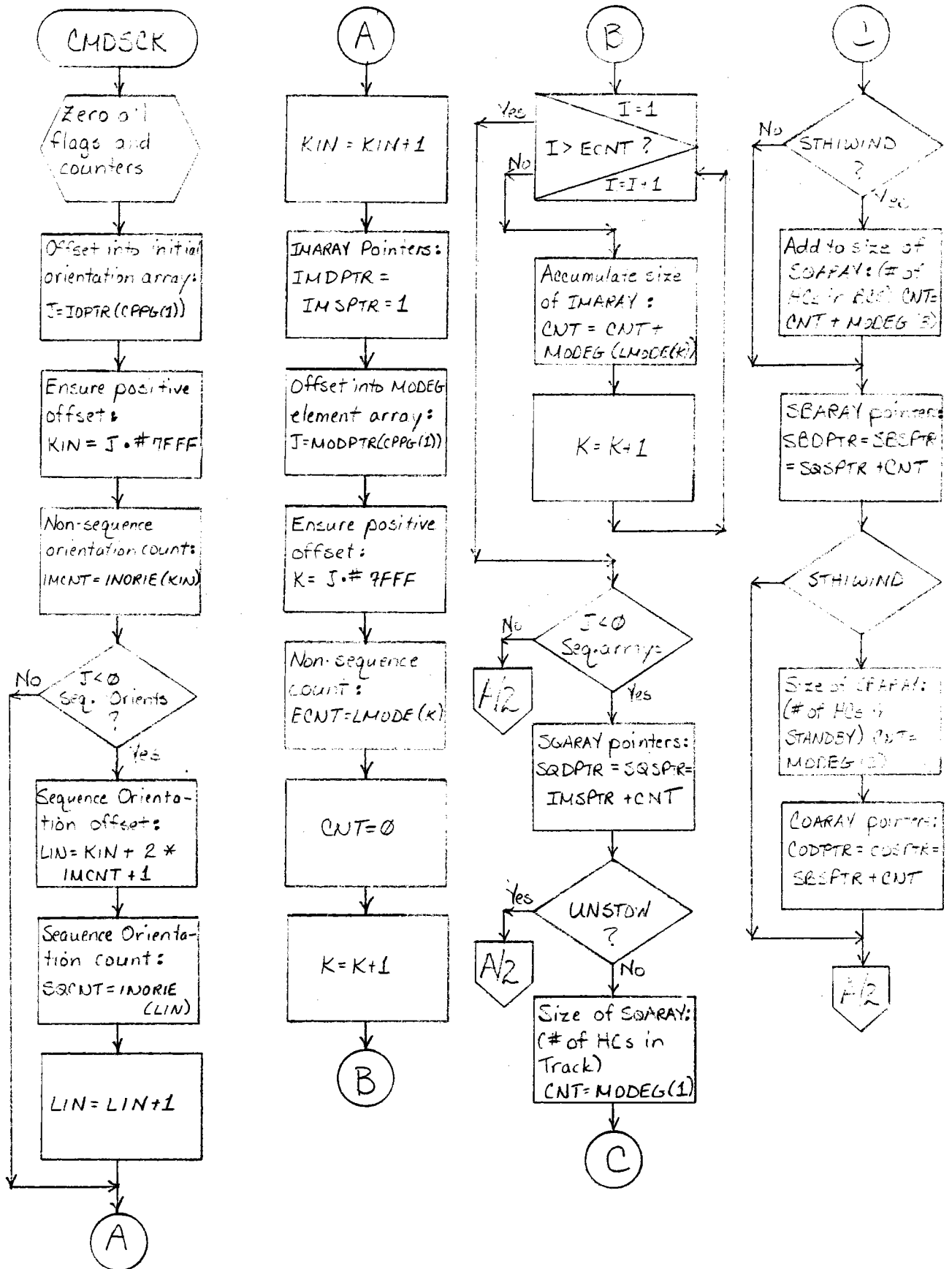


Figure 3.2.2-8 Flowchart - CMDSCK

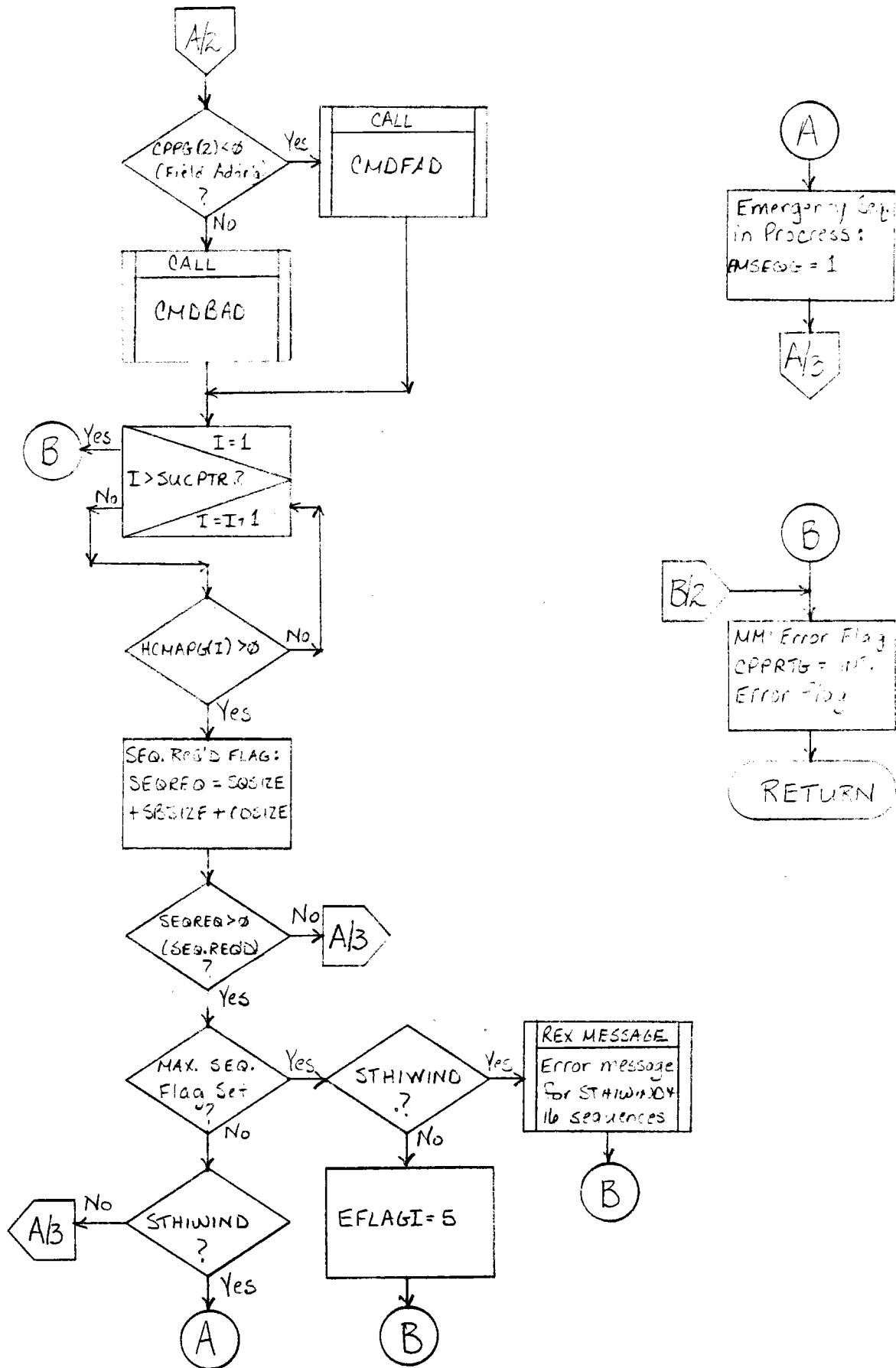


Figure 3.2.2-8 Flowchart - CMDSCK (continued)

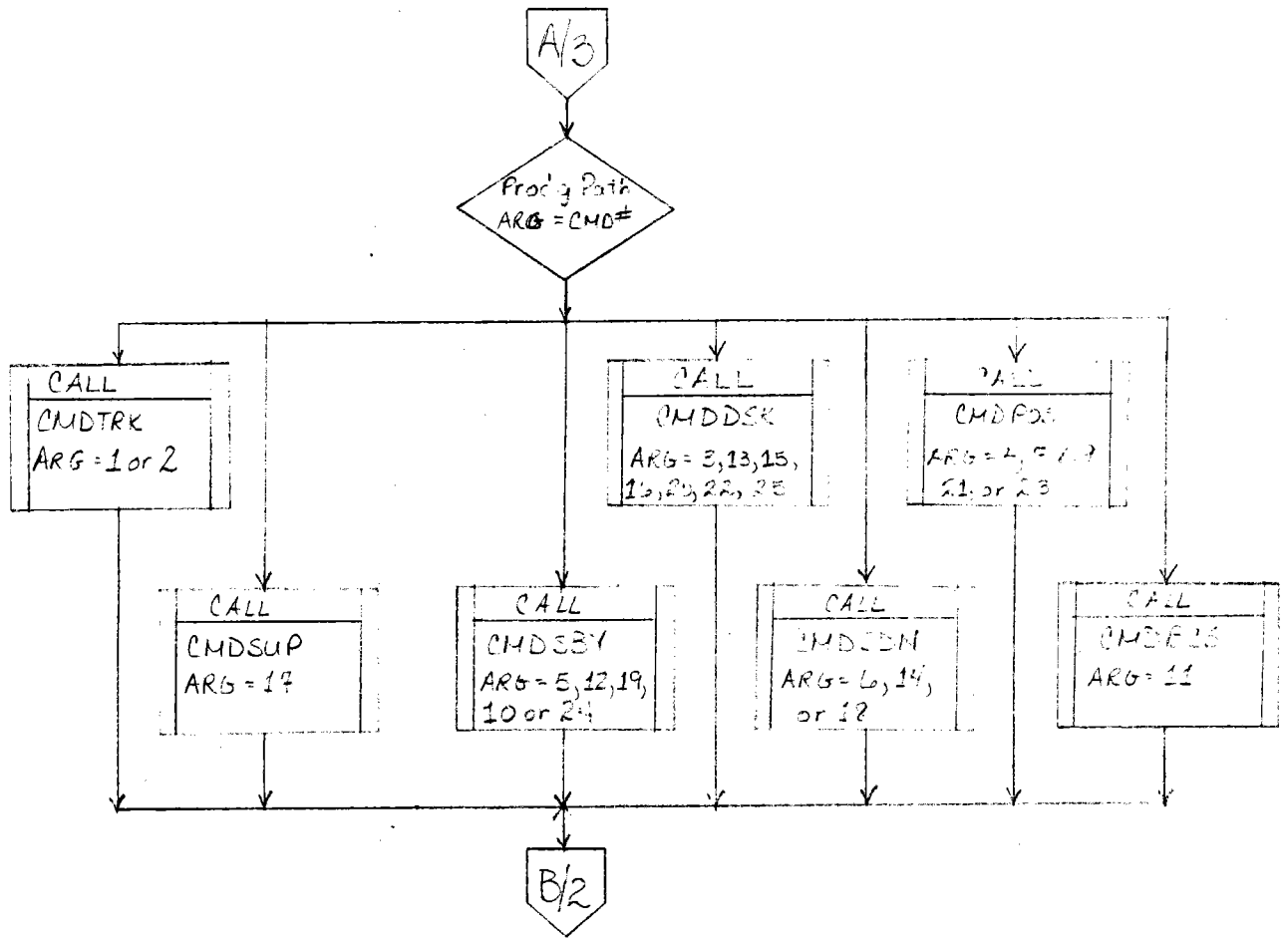


Figure 3.2.2-8 Flowchart - CMDSCK (continued)

one to 2048. If HC is greater than 2048, set the final success count by:

HCMAPG(SUCPTR) = SUCCNT

and return to CMDSCK.

2. Check if HC is installed by examining HCST2G(HC). If value is less than zero (not installed), set EFLAGI to equal three and go to step (1). Otherwise, call the initial orientation processor submodule CMDINO for this HC.
3. Upon return, check the initial orientation-okay flag(PASS). If greater than zero, HC has passed normal initial orientation tests for this input command; go to step (4). Otherwise, HC did not pass normal tests; check if command is STHIWIND. If not, set EFLAGI to equal one, and go to step (1). If STHIWIND, set bit 7 of HCST2G(HC) to inform the ALARMS module that this HC could not respond to STHIWIND; go to step (1).
4. If command is not ESTANDBY, go to step (5). For ESTANDBY commands, determine the orientation when full-field power loss was detected:

FLDORI = PWHC1G(HC).AND.#7C.

If FLDORI equals #20(Track), go to step (6). Otherwise, if FLDORI equals #30(Standby), go to step (6). If neither, set PASS to equal zero and EFLAGI to equal one; go to step (1).

5. If the command is not ESTOW, go to step (6). For ESTOW commands, determine the orientation when full-field power loss was detected:

FLDORI = PWHC1G(HC).AND.#70

If FLDORI equals #40(Corridor), go to step (6). Otherwise, if FLDORI equals #50(Corridor), go to step (6). If neither, set up other test by:

FLDORI = PWHC1G(HC).AND.#7C

If FLDORI equals #34(CLLP), go to step (6). Otherwise, set PASS to equal zero and EFLAGI to equal one; go to step (1).

6. Determine if the HC is in a sequence by examining HCST3G(HC), bits 11 through 15. If zero, go to step (7). If nonzero, check if command is STHIWIND. If not, go to step (1). If STHIWIND, determine if HC is in a Stow sequence by examining HCST3G(HC), bit 7. If set, ignore this HC because STHIWIND also sends HCs to the Stow orientation; go to step (1). If not set, delete (rob) the HC from its sequence by zeroing bits 11 through 15 of HCST3G(HC).

7. Call submodule CMDARA to place the HC number into one of four HC number buffers (see Figure 3.2.2-9).
 8. Upon return, increment the success count (SUCCNT) by one, and go to step (1).
- e. Error messages and recovery - EFLAGI is set to one if an HC is not in the correct orientation and set to three if an HC is not installed.

3.2.2.4.1.3.2 Data, Logic and Command Paths

Input data:

- a. Global common:
 1. HCST2G - HC installed status
 2. CPPG(1) - command number
 3. PWHC1G - HC orientation at power fail
 4. HCST3G - HC sequence assignment
- b. Local common:
 1. SUCCNT - command's success count
 2. PASS - initial-orientation okay flag
 3. HC - current HC number

Output data:

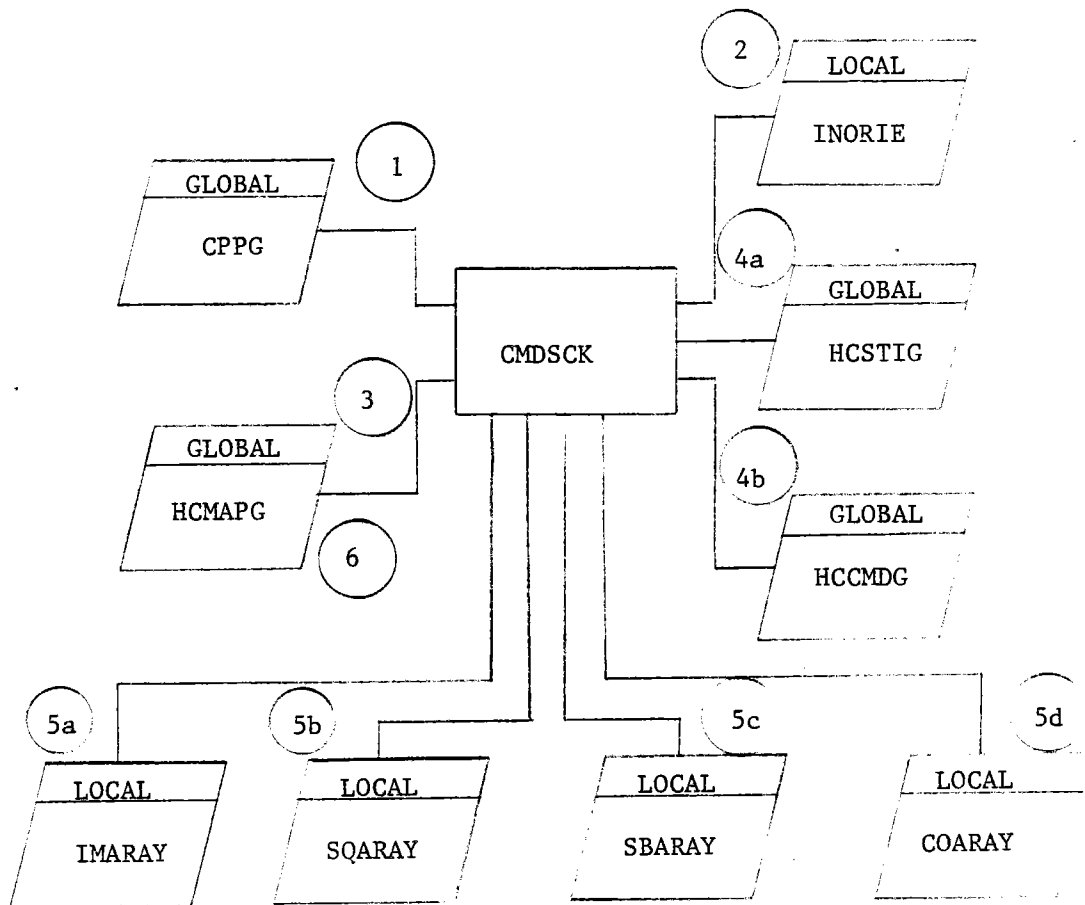
- a. Global common:
 1. HCMAPG - command's success array
 2. HCST2G - STHIWIND alarm bit
 3. HCST3G - HC sequence assignment
- b. Local common:
 1. EFLAGI - internal error flag
 2. FLDORI - field orientation
 3. PASS - initial-orientation okay flag
 4. SUCCNT - command's success count
 5. SUCPTR - HCMAPG output-success pointer
 6. HC - current HC number

3.2.2.4.1.3.3 Internal Data Description

There is no data internal to this submodule.

3.2.2.4.1.3.4 Flowchart

See Figure 3.2.2-10 for the CMDFAD flowchart.



<u>Event</u>	<u>Description</u>
1	Get command #, addressing type
2	Get all valid initial orientations for command
3	Get addressed HC #s (if non-field addressing)
4a	Get last reported orientation for HC
4b	Get last commanded orientation
5a	Store HC #s for immediate, non-sequence movement
5b	Store HC #s for non-corridor, sequence movement
5c	Store HC #s for corridor, sequence movement
5d	Store HC #s for corridor walk gather (STHI only)
6	Store command's degree of success in buffer

Figure 3.2.2-9 CMD SCK Input/Output Buffers

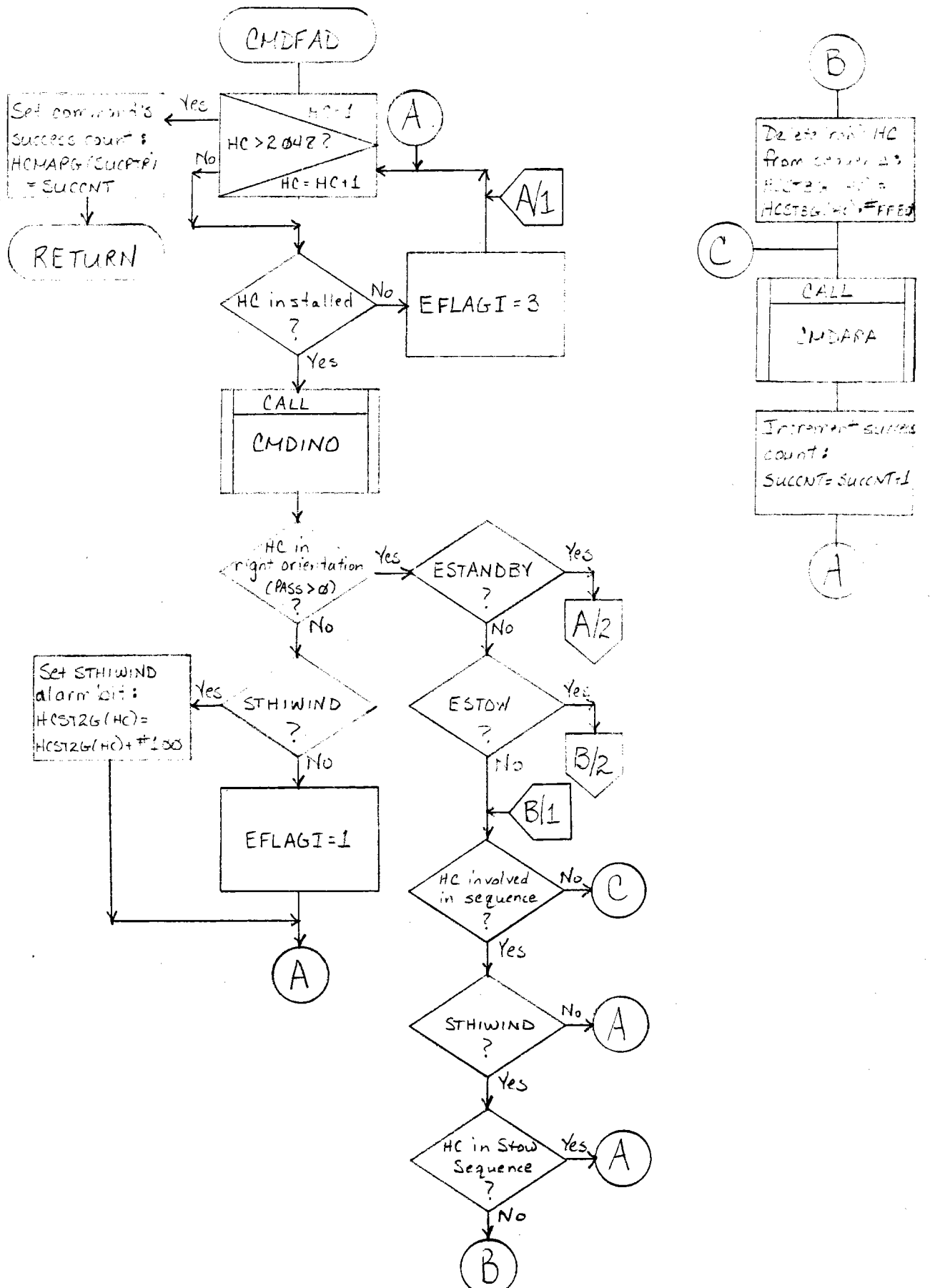


Figure 3.2.2-10 Flowchart - CMDFAD

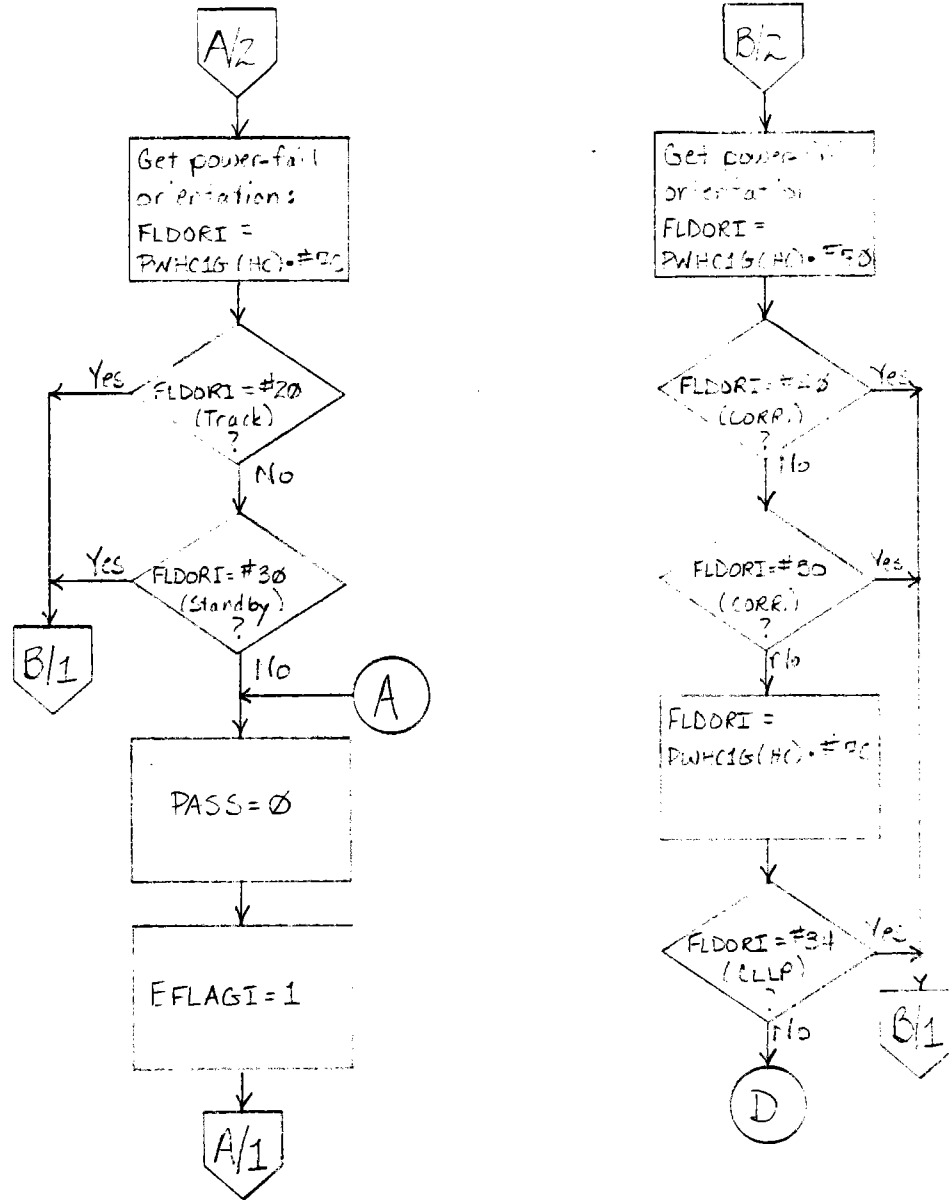


Figure 3.2.2-10 Flowchart - CMDFAD (continued)

3.2.2.4.1.4

Submodule IV - CMDBAD

3.2.2.4.1.4.1

Description

CMDBAD is responsible for initial processing of input commands using block addressing (see Table 3.2.2-III). Only those HC numbers contained in the blocks of HCMAPG are initially considered as involved. Those HCs which are able to respond to the input command are placed in the appropriate HC number buffer by submodule CMDARA. A success-count value for each HCMAPG block is inserted into HCMAPG for MMI feedback to the command input source.

- a. Language used - FORTRAN IV
- b. How invoked - called by CMDSCK
- c. Constraints and limitations - None
- d. Processing -

1. Set the INPTR augment value (INPTRA) to one. If command is not INCREASE or DECREASE, go to step (2). Otherwise, check for the optional number allowed with INCREASE or DECREASE commands, CPPG(2). If zero (use entire block), go to step (2). Otherwise, add the negative block length (see Table 3.2.2-III) to CPPG(2). If the result is greater than zero, go to step (2). If less than zero, set the INPTR augment value:

$$\text{INPTRA} = -\text{HCMAPG}(\text{INPTR}) - \text{CPPG}(2) + 1.$$

This value is used to increment INPTR to point to the next negative block count whenever the current block has been completed. Set LOOP to equal CPPG(2) to only process the optional number of HCs for the INCREASE or DECREASE command. Go to step (3).

2. Set LOOP equal to -HCMAPG(INPTR) to process all the HCs in the block.
3. Perform DO-UNTIL processing where I equals one to LOOP. If I is greater than LOOP, go to step (13).
4. Increment the HCMAPG input pointer (INPTR) by one to point to the next HC number in the block. Extract the HC number by:

$$\text{HC} = \text{HCMAPG}(\text{INPTR}).$$

Check if the HC is installed by examining HCST2G(HC). If less than zero (not installed), set EFLAGI equal to three, and go to step (3). Otherwise, call the initial orientation processor or submodule CMDINO for this HC.

5. Upon return, check the initial-orientation okay

I. MMI-to-CMD (Single block of HC #s)

<u>Array (Element)</u>	<u>Value</u>	<u>Description</u>
HCMAPG (1)	$-N_t$	Total # of HCs in block
(2)	HC #	} N_t HCs
.	.	
.	.	
.	.	
(N_t+1)	HC #	
(N_t+2)	-9999_{10}	End of data

II. MMI-to-CMD (Multiple blocks of HC #s)

HCMAPG (1)	$-N_1$	# of HCs in 1st block	} 1st block
(2)	HC #	} N_1 HCs in pecking order	
.	.		
.	.		
.	.		
(N_1+1)	HC #		
(N_1+2)	$-N_2$	# of HCs in 2nd block	} 2nd block
(N_1+3)	HC #	} N_2 HCs in pecking order	
.	.		
.	.		
.	.		
(N_1+N_2+2)	HC #		
.	.		
.	.		
.	.		
NB			
$(\sum_{i=1}^{NB} N_i+NB+1)$	-9999_{10}	End of data	

Where NB = # of blocks inherent in command addressing

Table 3.2.2-III MMI/CMD HC Block Buffer

III. CMD-to-MMI Statistics (degree of success)

<u>Array (Element)</u>	<u>Value</u>	<u>Description</u>
HCMAPG (1)		# of HCs successful from 1st block or for field addressing
(2)		# of HCs successful for 2nd block
.		
.		
.		
(N+1)	-9999 ₁₀	(End of data)

NOTE: These statistics reflect initial success rate; any subsequent HC timeout problems are reported via alarms.

HCMAPG(1) will be set to 0 for non-field addressing if no HCs within any block can respond.



flag (PASS). If greater than zero, HC has passed the normal initial orientation tests for this input command; go to step (7).

6. HC did not pass all initial orientation tests. Set EFLAGI to equal one, and go to step (3).
7. Check if command is HOLD. If not, go to step (8). For HOLD, do two additional tests. Get HC's field orientation:

FLDORI = HCST1G(HC).AND.#7C.

If FLDORI equals #18 (initialization), ignore this HC by going to step (3). Otherwise, get HC's last commanded orientation:

FLDORI = HCCMDG(HC).AND.#D07C.

If FLDORI equals #9030, HC is participating in a DEFOCUS action; ignore this HC by going to step (3). If HC is not being commanded via a DEFOCUS command, go to step (8).

8. If command is not ESTANDBY, go to step (9). For ESTANDBY commands, determine orientation when full-field power loss was detected:

FLDORI = PWHC1G(HC).AND.#7C.

If FLDORI equals #20 (Track), go to step (10). Otherwise, if FLDORI equals #30 (Standby), go to step (10). If neither, set PASS equal to zero and EFLAGI equal to one; go to step (3).

9. If command is not ESTOW, go to step (10). For ESTOW commands, determine orientation when full-field power loss was detected:

FLDORI = PWHC1G(HC).AND.#70.

If FLDORI equals #40 (Corridor), go to step (10). Otherwise, if FLDORI equals #50 (Corridor), go to step (10). If neither, set up other test by:

FLDORI = PWHC1G(HC).AND.#7C.

If FLDORI equals #34 (CLLP), go to step (10). Otherwise, set PASS to equal zero and EFLAGI to equal one; go to step (3).

10. Determine if HC is in a sequence by examining HCST3G(HC), bits 11 through 15. If zero, go to step (11). If non-zero, check if command is HOLD. If not, go to step (6). If HOLD, delete (rob) the HC from its sequence by zeroing bit 7 (Stow sequence status) and bits 11 through 15 (sequence number) of HCST3G(HC).
11. Call submodule CMDARA to place the HC number into one of four HC number buffers (see Table 3.2.2-IV).



<u>Array (Element)</u>	<u>Description</u>
CPPG(1)	Command # (range: 1 to 25)
CPPG(2)	<p>>0: # of HC's/block (used only for pecking order related commands; e.g., INCREASE 10/R/5 would cause the number 10 to appear in this word)</p> <p>=0: Use entire block</p> <p><0: Use entire field</p>
CPPG(3)	Azimuth (for POSITION commands) or Aim-point Array # (for AIMPOINT commands) or 0
CPPG(4)	Elevation (for POSITION commands) or 0

For "segment", "wedge", or "ring" addressing, a block is a segment. For "field controller" addressing, a block is a HFC. For "heliostat" or "arc" addressing, a block is the set of addressed HCs. For "field" addressing, CPPG(2) < 0 and no blocks of HC numbers are passed in the HCMAPG array (Figure 3.2.2-8). HC numbers involved in a non-field command are transmitted in the HCMAPG array.

Table 3.2.2-IV MMI/CMD Command Buffer



12. Upon return, increment the block's success count (SUCCNT) by one, and go to step (3).

Block processing has been completed. Section to look for end of data or next block:

13. Set block's final success count by:

HCMAPG (SUCPTR) = SUCCNT.

Set INPTR to next block count by:

INPTR = INPTR + INPTRA.

Zero the block's success count (SUCCNT).

14. Examine the next block count word in HCMAPG to check for end of data (-9999). If -9999, return to CMDSCK. Otherwise, another block is present. Increment the success-count pointer (SUCPTR) by one, and go to step (1).

- e. Error messages and recovery - EFLAGI is set to one if an HC is not in the correct orientation and set to three if an HC is not installed.

3.2.2.4.1.4.2

Data, Logic and Command Paths

Input data:

a. Global common:

1. HCST2G - installed bit
2. CPPAG(1) - command number
3. FWHC1G - HC orientation at power fail
4. HCST3G - HC sequence assignment

b. Local common:

1. SUCCNT - block's success count
2. PASS - initial-orientation okay flag
3. HC - current HC number

Output data:

a. Global common:

1. HCMAPG - command's success array
2. HCST3G - HC sequence assignment

b. Local common:

1. EFLAGI - internal error value
2. FLDORI - field orientation
3. PASS - initial-orientation okay flag
4. SUCCNT - block's success count
5. SUCPTR - HCMAPG output-success pointer
6. HC - current HC number

3.2.2.4.1.4.3 Internal Data Description

There is no data internal to this submodule.

3.2.2.4.1.4.4 Flowchart

See Figure 3.2.2-11 for the CMDBAD flowchart.

3.2.2.4.1.5 Submodule V - CMDINO

3.2.2.4.1.5.1 Description

CMDINO is responsible for determining whether the input HC is in a valid initial orientation for the input command. It also tests for non-software offline (field problem) and participation in the automatic BCS operations. The initial-orientation okay flag (PASS) is non-zero for an HC in a valid orientation.

- a. Language used - FORTRAN IV
- b. How invoked - called by CMDFAD or CMDBAD
- c. Constraints and limitations - only STHWIND can rob an HC from automatic BCS operations.
- d. Processing -
 1. Zero the initial-orientation okay flag (PASS) and the HC-in-sequence orientation flag (SEQOR). Check the non-sequence orientation count (IMCNT). If zero, command is ONLINE or LOAD and all orientations are valid; set PASS to equal one and go to step (9). Otherwise, check HC to be offline due to a field problem by examining HCST2G(HC), bits 1 through 4 and 8. If non-zero (field problem), go to step (10). Otherwise, set the non-sequence, initial-orientation array (INORIE) index K to equal KIN. KIN was set in CMDSCK to point to the input command's first non-sequence orientation value in INORIE.

Test HC to be in a non-sequence orientation. This means a sequence will not be required to move the HC to the final destination specified in the input command.

2. Perform DO-UNTIL processing where M equals one to IMCNT. If M is greater than IMCNT, check the input command's sequence-orientation count (SQCNT). If greater than zero (sequence orientations exist in INORIE), go to step (5). If zero, go to step (10).
3. Get the next valid non-sequence orientation for the input command:
ORIENT = INORIE(K).

Increment K by one and get the corresponding mask

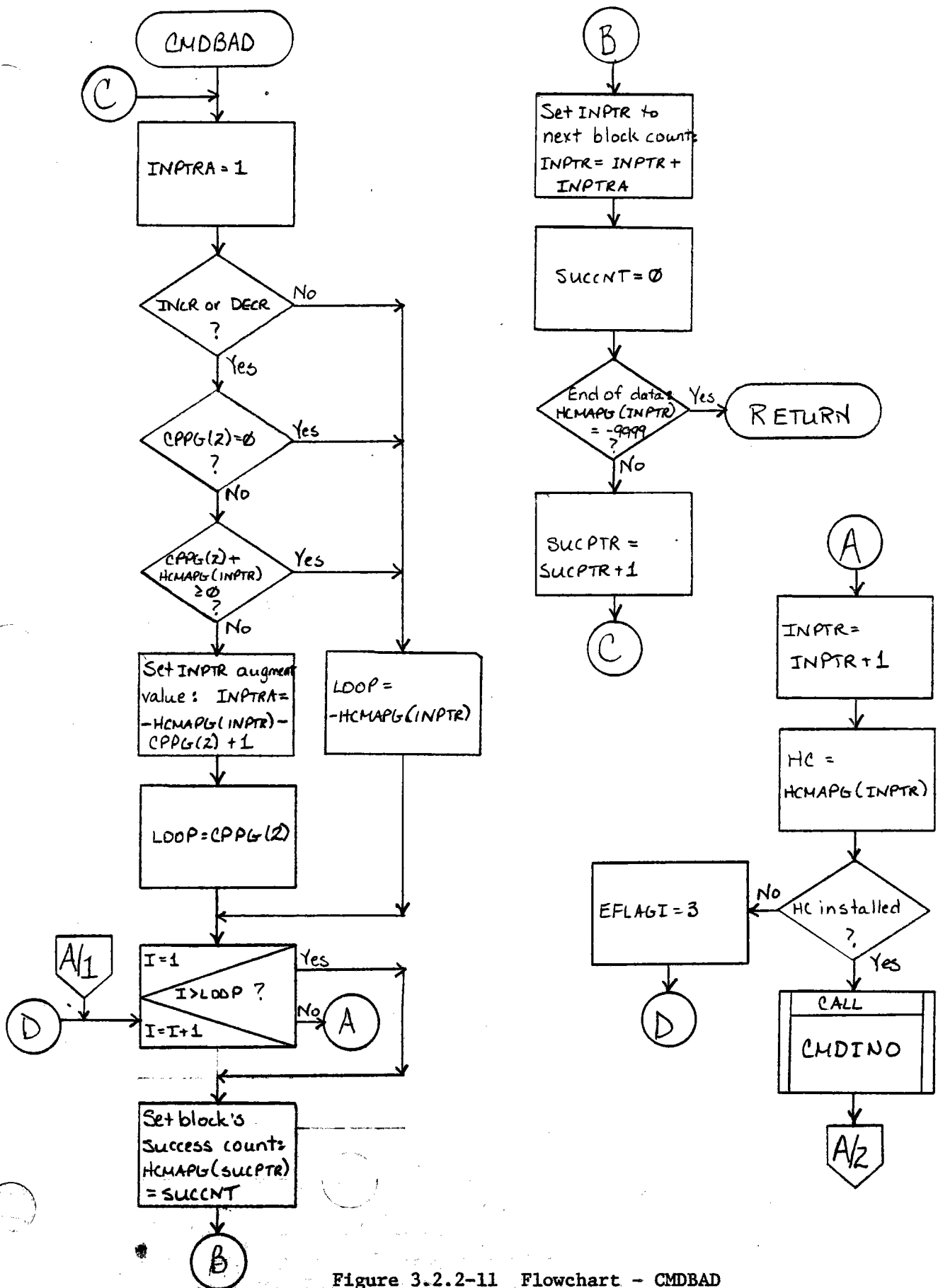


Figure 3.2.2-11 Flowchart - CMDBAD

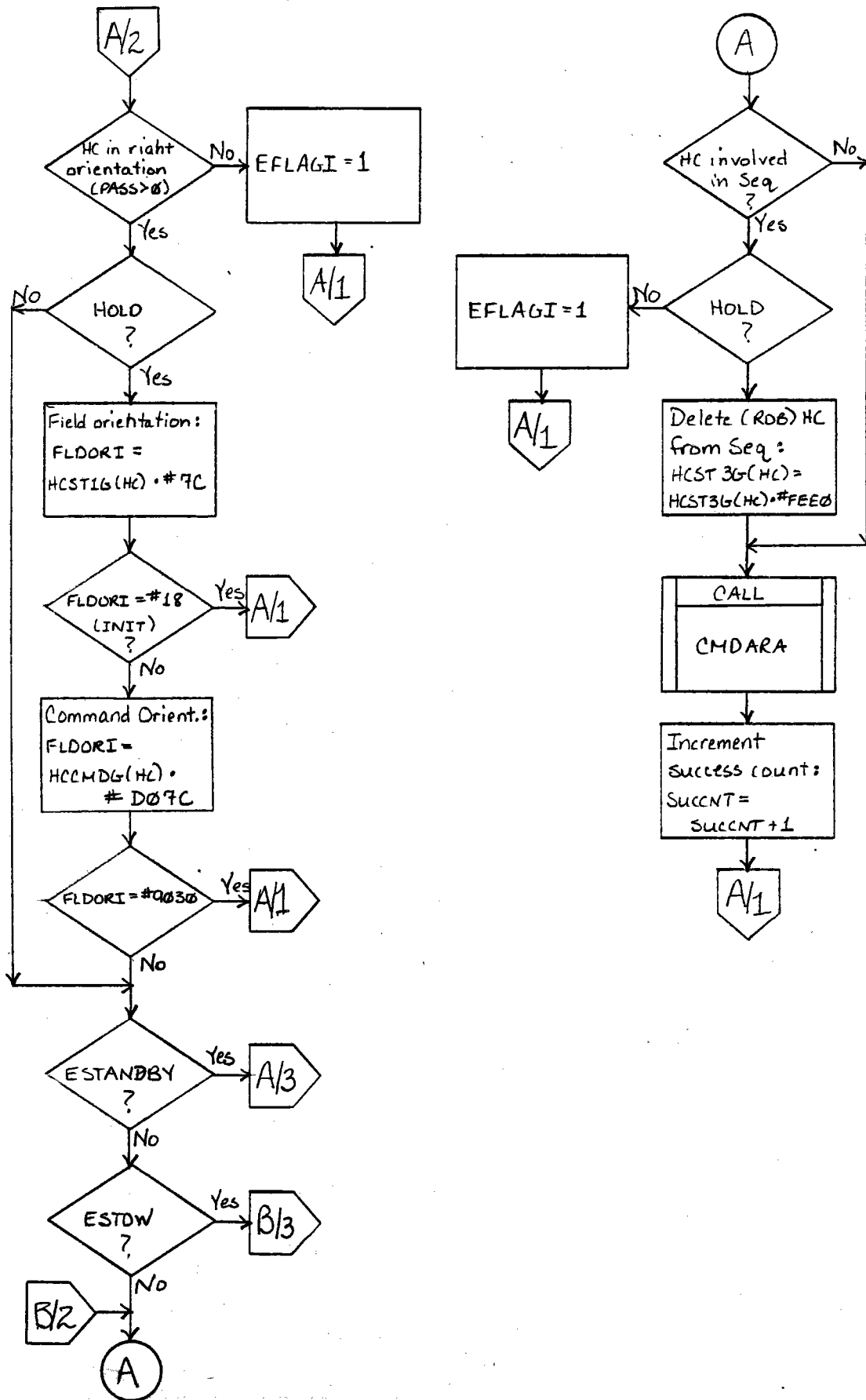


Figure 3.2.2-11 Flowchart - CMDBAD (continued)

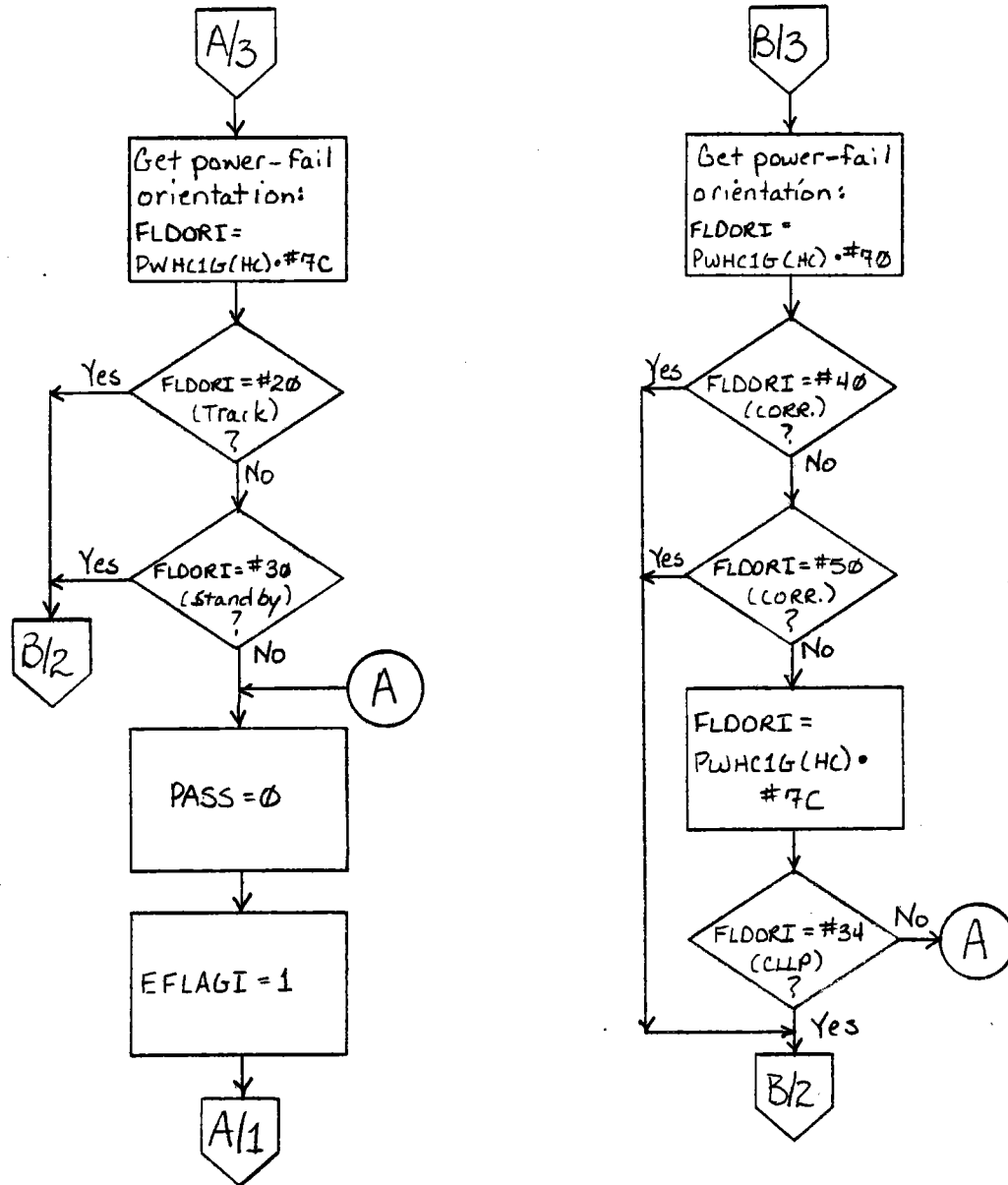


Figure 3.2.2-11 Flowchart - CMDBAD (continued)

for HC's field orientation:

MASK = INORIE(K).

Get the current field orientation by:

FLDORI = HCST1g(HC).AND.MASK.

See Figure 3.2.2-9 for overview.

4. Compare the valid orientation value against the field orientation value. If they match, set PASS to equal one and go to step (9). If not, increment K by one, and go to step (2).

Test HC to be in a sequence orientation. This means a sequence will be required to move HC to final destination specified in the input command.

5. Set the HC-in-sequence orientation flag (SEQOR). Set the sequence, initial-orientation array (INORIE) index L to equal LIN. LIN was set in CMDSCK to point to input command's first sequence orientation value in INORIE.
6. Perform DO-UNTIL processing where M equals one to SQCNT. If M is greater than SQCNT, HC is not in a proper orientation; go to step (10).
7. Get the next valid sequence initial orientation for the input command, ORIENT equals INORIE(L). Increment L by one and get the corresponding mask for HC's field orientation, MASK equals INORIE(L). Get the current field orientation by:

FLDORI = HCST1G(HC).AND.MASK.

See Figure 3.2.2-9 for overview.

8. Compare the valid orientation value against the field orientation value. If they match, set PASS to equal one and go to step (9). If not, increment K by one, and go to step (6).

Test successful HC to be in automatic BCS operations:

9. Get the automatic BCS status by:

BCSFLG = HCST3G(HC).AND.#80.

If zero, go to step (10). Otherwise, check command to be STHIWIND. If not, ignore HC by setting PASS to equal zero, and go to step (10). If STHIWIND, delete(rob) the HC from automatic BCS by:

HCST3G(HC) = HCST3G(HC).AND.#FFEF.

10. Return to caller.

e. Error messages and recovery - None

3.2.2.4.1.5.2 Data, Logic and Command Paths

Input data:

a. Global common:

1. HCST2G - HC offline status
2. HCST1G - HC field status
3. HCST3G - HC automatic BCS status
4. CPPG(1) - command number

b. Local common:

1. IMCNT - non-sequence orientation count
2. SQCNT - sequence orientation count
3. KIN - non-sequence orientation pointer
4. LIN - sequence orientation pointer
5. INORIE - initial-orientation value and mask array

Output data:

a. Global common:

HCST3G - HC automatic BCS status

b. Local common:

1. PASS - initial-orientation okay flag
2. SEQOR - sequence orientation flag
3. K - non-sequence orientation into INORIE
4. L - sequence orientation into INORIE
5. ORIENT - valid orientation value from INORIE
6. MASK - field-orientation mask value from INORIE
7. FLDORI - masked field orientation
8. BCSFLG - automatic BCS status for HC

3.2.2.4.1.5.3 Internal Data Description

There is no data internal to this submodule.

3.2.2.4.1.5.4 Flowchart

See Figure 3.2.2-12 for the CMDINO flowchart.

3.2.2.4.1.6 Submodule VI - CMDARA

3.2.2.4.1.6.1 Description

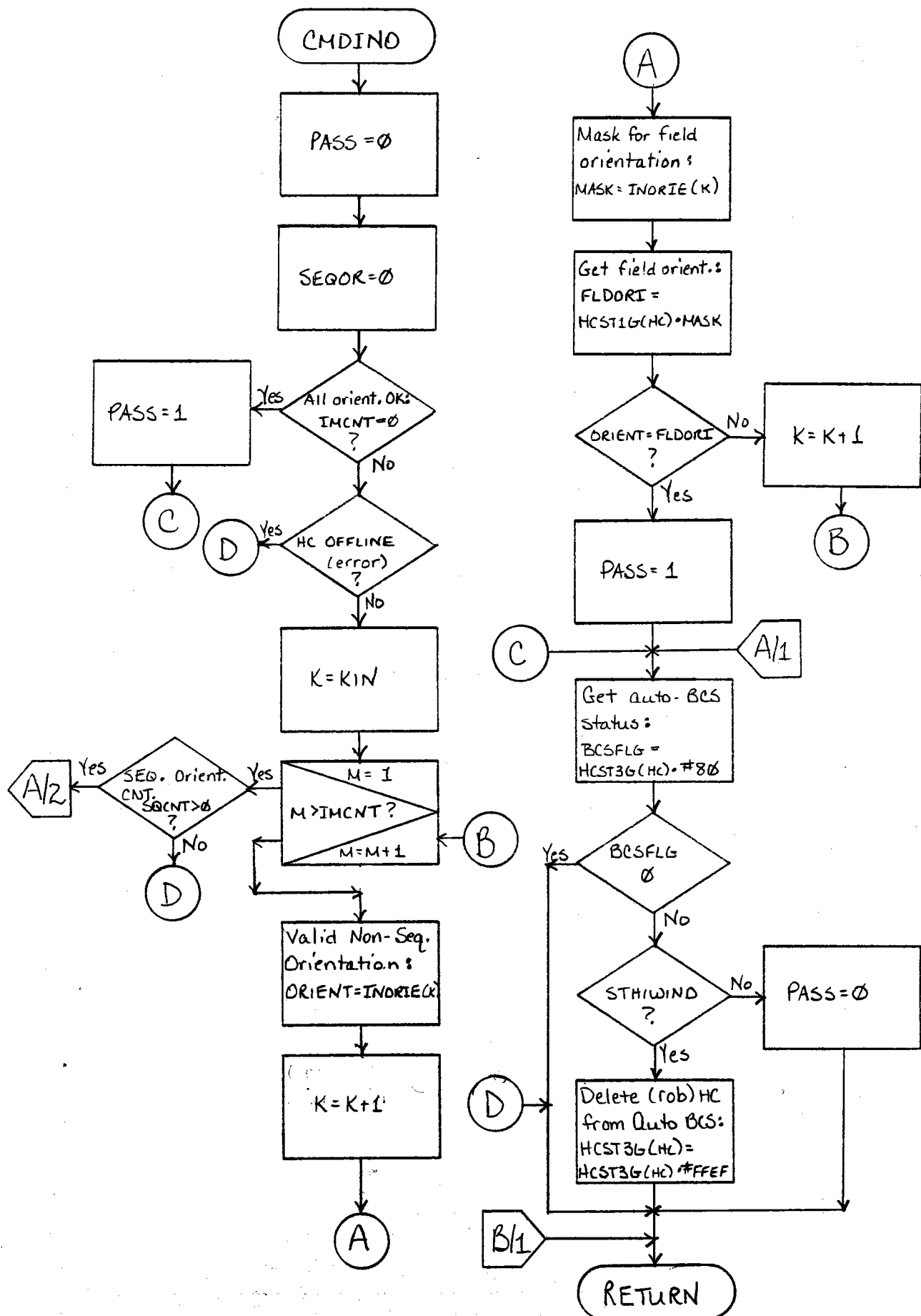


Figure 3.2.2-12 Flowchart - CMDINO

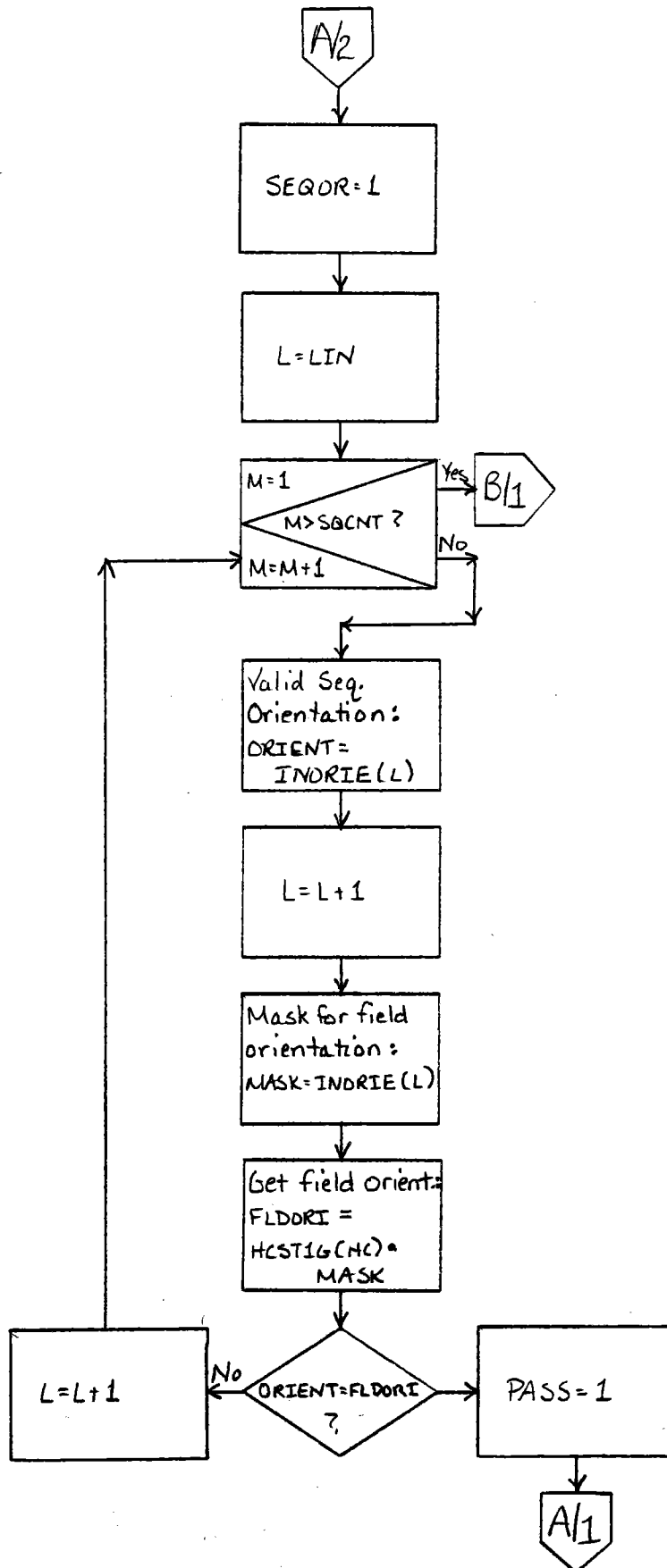


Figure 3.2.2-12 Flowchart - CMDINO (continued)

CMDARA is responsible for inserting the HC number specified in local common word "HC" into the appropriate HC number buffer for subsequent processing. The value of flag SEQOR dictates whether the HC number goes into the non-sequence buffer (IMARAY) or one of the sequence buffers (SQARAY, SBARAY, COARAY). The non-sequence buffer is used to build a disk packet for those input commands requiring disk data. The sequence buffers are used to build a sequence packet for those input commands requiring sequence processing.

- a. Language used - FORTRAN IV
- b. How invoked - called by CMDFAD or CMDBAD
- c. Constraints and limitations - None
- d. Processing -
 1. Determine the type of orientation found for the current HC by examining flag SEQOR. IF greater than zero, HC must be command with sequence processing; go to step (2). If zero, put the HC number in IMARAY by setting HCARAY(IMDPTR) to equal HC. Increment the dynamic IMARAY pointer (IMDPTR) by one and increment the IMARAY size (IMSIZE) by one; go to step (6).
 2. Ignore the position-compare bit when getting HC's field orientation:
$$FLDORI = HCST1G(HC).AND.\#7C.$$
If FLDORI equals #20 (Track), go to step (3). If FLDORI equals #2C(BCS), go to step (3). If FLDORI equals #60 (Stow), go to step (3). If FLDORI equals #70 (Alt1stow), go to step (3). Otherwise, go to step (4).
 3. Put the HC number in SQARAY by setting HCARAY (SQDPTR) to equal HC. Increment the dynamic SQARAY pointer (SQDPTR) by one and increment the SQARAY size (SQSIZE) by one; go to step (6).
 4. If FLDORI is not #30 (Standby), go to step (5). Otherwise, put the HC number in SBARAY by setting HCARAY(SBDPTR) to equal HC. Increment the dynamic SBARAY pointer (SBDPTR) by one and increment the SBARAY size (SBSIZE) by one; go to step (6).
 5. Orientation is corridor walk and only used by STHIWIND command. Put the HC number in COARAY by setting HCARAY(CODPTR) to equal HC. Increment the dynamic COARAY pointer (CODPTR) by one and increment the COARAY size (COSIZE) by one; go to step (6).
 6. Return to caller.
- e. Error messages and recovery - None



3.2.2.4.1.6.2 Data, Logic and Command Paths

Input data:

a. Global common:

HCSTIG - HC field status

b. Local common:

1. SEQOR - HC's sequence-orientation flag
2. FLDORI - field orientation
3. IMDPTR - dynamic IMARAY pointer
4. IMSIZE - IMARAY size
5. SQDPTR - dynamic SQARAY pointer
6. SQSIZE - SQARAY size
7. SBDPTR - dynamic SBARAY pointer
8. SBSIZE - SBARAY size
9. CODPTR - dynamic COARAY pointer
10. COSIZE - COARAY size
11. HC - current HC number

Output data:

Local common:

- a. HCARAY - overall HC-number buffer area
- b. FLDORI - field orientation
- c. IMDPTR - dynamic IMARAY pointer
- d. IMSIZE - IMARAY size
- e. SQDPTR - dynamic SQARAY pointer
- f. SQSIZE - SQARAY size
- g. SBDPTR - dynamic SBARAY pointer
- h. SBSIZE - SBARAY size
- i. CODPTR - dynamic COARAY pointer
- j. COSIZE - COARAY size

3.2.2.4.1.6.3 Internal Data Description

There is no data internal to this submodule.

3.2.2.4.1.6.4 Flowchart

See Figure 3.2.2-13 for the CMDARA flowchart.

3.2.2.4.1.7 Submodule VII - CMDSAL

3.2.2.4.1.7.1 Description

CMDSAL is responsible for determining a sequence number when

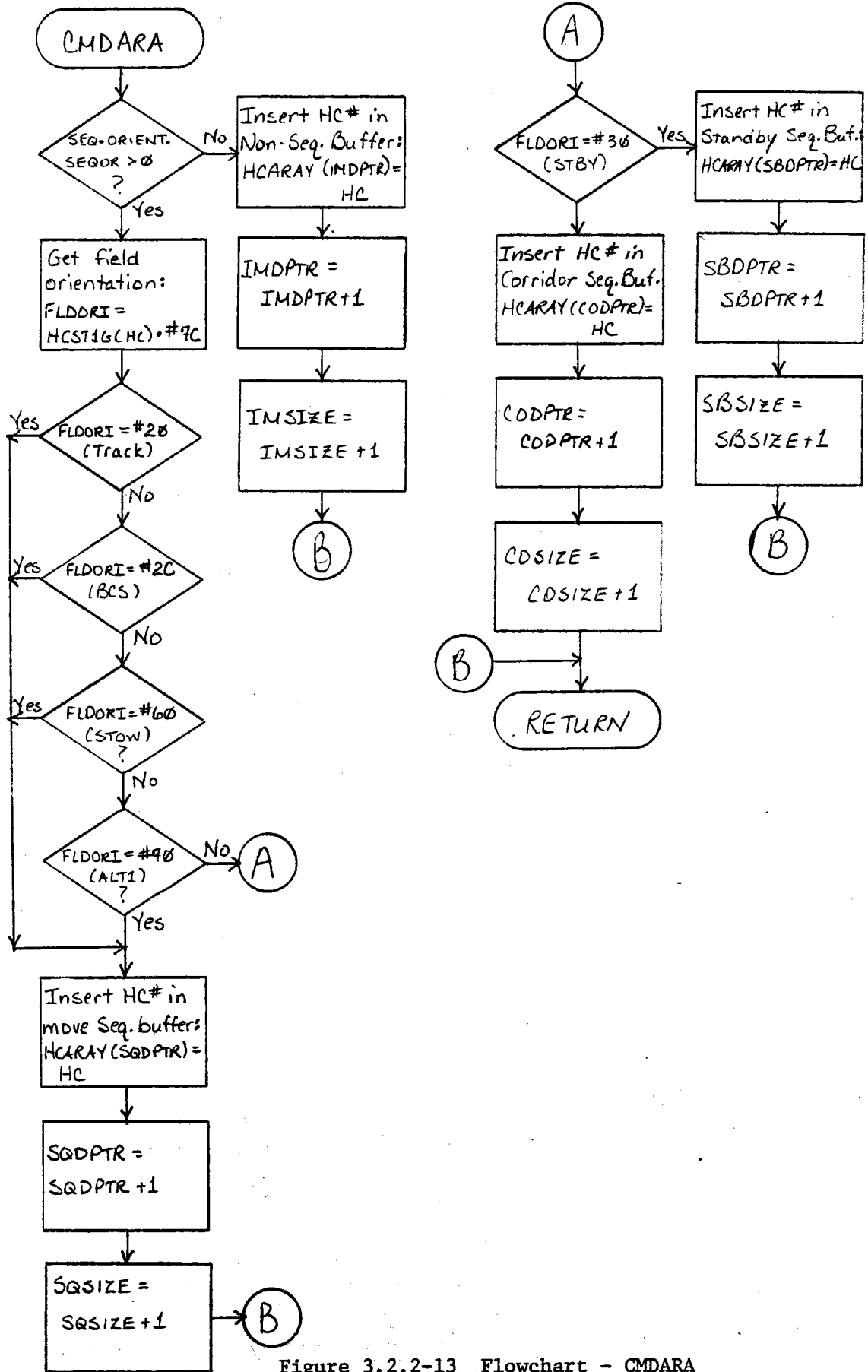


Figure 3.2.2-13 Flowchart - CMDARA

the input command requires sequence processing. It then determines all of the corridors needed for the sequence. Then it builds the sequence packet (see Table 3.2.2-V) required for CMD to Enque task SEQ. Possible sequence commands are UNSTOW, STOW, STHWIND, and ALT1STOW.

- a. Language used - FORTRAN IV
- b. How invoked - called by CMD to allocate a sequence for the input command and build a sequence packet.
- c. Constraints and limitations - assume maximum number of corridors is eight.
- d. Processing -

Allocation of sequence number:

- 1a. Set loop count (LOOP) to equal 16 for the maximum number of sequences.
- 1b. Perform DO-UNTIL processing where I equals one to LOOP. If I is greater than LOOP, set CPPRTG to equal five and go to step (31).
- 1c. If SEQLSG(I) is greater than zero, go to step (1b).
2. When a negative number is found in SEQLSG(I), take the two's-complement and update this array element. Set the allocated sequence number (SEQNO) to this positive value. Increment the active-sequences count (SEQNMG) by one, and if SEQNMG equals 16, set the maximum-sequences flag (MAXSEQ). Otherwise, clear MAXSEQ.

To determine all the corridors required by this sequence:

3. Clear the corridors-required array (CORARY), zero the corridors-required count (CORCNT), zero the HC-buffers count (BUFCNT), and zero the command-phase word (CPHASE).
4. If the size word (SBSIZE) of the HCs-in-standby array (SBARAY) is not greater than zero, go to step (7). If greater than zero, set LOOP to equal SBSIZE, set CPHASE to equal five, and set CALLER to equal one.
5. Perform DO-UNTIL processing where I equals one to LOOP. If I is greater than LOOP, go to step (7).
6. Set N to equal SBARAY(I) and call internal search routine (steps (32) to (36)) to find the corridor associated with this HC in SBARAY. The routine increments CORCNT when inserting a unique corridor number in CORARY. Upon return, check CORCNT to equal eight (maximum number of corridors). If so, go to step (13). Otherwise, go to step (5).
7. If the size word (SQSIZE) of the HCs-not-at-standby array (SQARAY) is not greater than zero,

<u>WORD</u>	<u>DESCRIPTION</u>
1	COMMAND # = 0: FIELD POWER LOSS or HAC FAILOVER 6: STOW 14: ALT1STOW 17: UNSTOW 18: STHIWIND
2	COMMAND PHASE = 4: GATHER HCs COMMANDED BY TASK CMD 5: WAIT FOR ALL REQUIRED CORRIDORS TO BE FREE 10: END SEQUENCE 20: FIELD POWER LOSS 30: HAC FAILOVER
3	SEQUENCE #: RANGE 1 to 16
4	# of CORRIDORS REQUIRED: RANGE 1 to 8
5 to N	CORRIDORS #(s) REQUIRED
N+1	# of BUFFER DESCRIPTIONS TO FOLLOW: UNSTOW: 1 STOW: 1 or 2 ALT1STOW: 1 or 2 STHIWIND: 1, 2, or 3
N+2	OFFSET (INTO COMQU2) FOR LEASED HC# BUFFER
N+3	ORIENTATION FOR BUFFER #1 ORIENTATION CODES ARE: 1 (HCs MOVING to CULP) 2 (HCs at CULP) 3 (HCs MOVING in CORRIDORS) 4 (HCs MOVING to CLLP) UNSTOW: 4 ALT1STOW or STOW: 1 or 2 STHIWIND: 1, 2, or 3
N+4	SIZE FOR BUFFER #1: RANGE 1 to 2048
N+5	OFFSET (INTO COMQU2) for BUFFER #1
FOLLOWING WORDS ARE OPTIONAL:	
N+6	ORIENTATION FOR BUFFER #2 ALT1STOW or STOW: 2 STHIWIND: 2 or 3
N+7	SIZE FOR BUFFER #2: RANGE 1 to 2047
N+8	OFFSET (INTO COMQU2) for BUFFER #2
N+9	ORIENTATION for BUFFER #3 STHIWIND: 2
N+10	SIZE for BUFFER #3: RANGE 1 to 2046
N+11	OFFSET (INTO COMQU2) for BUFFER #3

go to step (10). If greater than zero, set LOOP to equal SQSIZE, increment BUFCNT by one, set CPHASE to equal four, and set CALLER to equal two.

8. Perform DO-UNTIL processing where I equals one to LOOP. If I is greater than LOOP, go to step (10).
9. Set N to equal SQARRAY(I) and call internal search routine (steps (32) to (36)) to find the corridor associated with this HC in SQARRAY. The routine increments CORCNT when inserting a unique corridor number in CORARY. Upon return, check CORCNT to equal eight (maximum number of corridors). If so, go to step (13). Otherwise, go to step (8).
10. If the size word (COSIZE) of the HCs-in-corridor-walks array (COARAY) is not greater than zero, go to step (13). If greater than zero, set LOOP to equal COSIZE, increment BUFCNT by one, set CPHASE to equal four, and set CALLER to equal three.
11. Perform DO-UNTIL processing where I equals one to LOOP. If I is greater than LOOP, go to step (13).
12. Set N to equal COARAY(I) and call internal search routine (steps (32) to (36)) to find the corridor associated with this HC in COARAY. The routine increments CORCNT when inserting a unique corridor number in CORARY. Upon return, check CORCNT to equal eight. If so, go to step (13). Otherwise, go to step (11).

Determine size of sequence packet:

13. Determine the sequence packet size using the equation:

$$\begin{aligned} \text{PACSIZ} = & 5 \text{ (words 1 to 4 and N+1 in Table} \\ & \text{3.2.2-V)} + \\ & \text{CORCNT (size for words 5 to N in Table} \\ & \text{3.2.2-V)} + \\ & \text{BUFCNT*2 (total number of buffer} \\ & \text{orientation and size words} \\ & \text{in Table 3.2.2-V)} + \\ & \text{SQSIZE (size of HCs-not-at-standby} \\ & \text{array)} + \\ & \text{SBSIZE (size of HCs-in-standby} \\ & \text{array)} + \\ & \text{COSIZE (size of HCs-in-corridor-} \\ & \text{walks array)}. \end{aligned}$$

14. The local common area used for building the packet is BUFR.

Set up the packet header:

15. Set BUFR(1) to equal CPPG(1), the command number. Set BUFR(2) equal to CPHASE, the command phase. Set BUFR(3) to equal SEQNO, the allocated sequence number. Set BUFR(4) to equal CORCNT, the corridor count. Set LOOP to equal CORCNT and set BUFR index J to equal five, the first location for corridor numbers in the packet.
16. Perform DO-UNTIL processing where I equals one to LOOP. If I is greater than LOOP, go to step (18).
17. Set BUFR(J) to equal CORARY(I), which transfers corridor numbers to the packet area. Increase J by one, and go to step (16).
18. Set BUFR(J) to equal BUFCNT, the number of HC buffers to be passed in the packet. Increment J by one.

Move HC numbers into the packet:

19. If SQSIZE is not greater than zero, go to step (23). If more than zero, check command to be UNSTOW. If so, set the first buffer orientation word BUFR(J) to equal four to describe this set of HC numbers as moving towards their CLLPs. If not UNSTOW, set BUFR(J) to equal one to describe this set of HC numbers as moving towards their CULPs.
20. Increment J by one and set BUFR(J) to equal SQSIZE to specify size of first buffer. Set LOOP to equal SQSIZE and increment J by one.
21. Perform DO-UNTIL processing where I equals one to LOOP. If I is greater than LOOP, go to step (23).
22. Set BUFR(J) to equal SQARAY(I) to move the HC number into the packet area. Increment J by one, and go to step (21).
23. If COSIZE is not greater than zero, go to step (27). If greater than zero, set BUFR(J) to equal three to describe this set of HC numbers as moving in their corridor(s). This buffer will only be used for the STHIWND command.
24. Increment J by one and set BUFR(J) to equal COSIZE to specify size of this buffer. Set LOOP equal to COSIZE and increment J by one.
25. Perform DO-UNTIL processing where I equals one to LOOP. If I is greater than LOOP, go to step (27).
26. Set BUFR(J) equal to COARAY(I) to move the HC number into the packet area. Increment J by one,

and go to step (25).

27. If SBSIZE is not greater than zero, go to step (31). If greater than zero, set BUFR(J) equal to two to describe this set of HC numbers as being at their CULPs.
28. Increment J by one and set BUFR(J) equal to SBSIZE to specify size of this buffer. Set LOOP to SBSIZE and increment J by one.
29. Perform DO-UNTIL processing where I equals one to LOOP. If I is greater than LOOP, the sequence packet is completely built and ready for the REX Enque call in CMD; go to step (31).
30. Set BUFR(J) equal to SBARAY(I) to move the HC number into the packet area. Increment J by one, and go to step (29).
31. Return to CMD.

Search routine to add corridor number to array of required corridors:

32. Determine HC's corridor number by setting
CORR = HCST3G(N).AND.#FOOO
and shift right 12 bits. Set LOOP equal to eight for the maximum number of corridors.
33. Perform DO-UNTIL processing where I equals one to LOOP. If I is greater than LOOP, go to step (35).
34. If CORARY(I) equals CORR, go to step (36). If not, go to step (33).
35. Add the corridor number found to CORARY array by setting:
CORARY(CORCNT+1) equals CORR.
Increment the corridor count (CORCNT) by one.
36. Return to calling section by using flag CALLER. If CALLER equals one, go to step (6). Otherwise, go to step (9) for CALLER to equal two. If CALLER equals three, go to step (12).

- e. Error messages and recovery - If an unused sequence number is not found in SEQLSG, the MMI-error return value (CPPRTG) is set to five and control is returned to CMD.

3.2.2.4.1.7.2 Data, Logic and Command Paths

Input data:

- a. Global common:

1. SEQLSG - sequence-number list array

2. SEQNMG - active-sequences count
3. HCST3G - HC corridor assignment

b. Local common:

1. SBSIZE - SBARRAY size
2. SBARRAY - HCs-in-standby array
3. CORCNT - required-corridors count
4. CORARY - required-corridors array
5. CALLER - search-routine caller flag
6. SQSIZE - SQARRAY size
7. SQARRAY - HCs-not-at-standby array
8. COSIZE - COARRAY size
9. COARRAY - HCs-in-corridor-walk array
10. BUFCNT - HC buffers-in-packet count
11. CPHASE - command phase for packet
12. SEQNO - allocated sequence number

Output data:

a. Global common:

1. SEQLSG - sequence number list array
2. SEQNMG - active sequences count

b. Local common:

1. SEQNO - allocated sequence number
2. MAXSEQ - maximum-sequences flag
3. CORARY - required-corridors array
4. CORCNT - required-corridors count
5. BUFCNT - HC buffers-in-packet count
6. CPHASE - command phase for packet
7. CALLER - search-routine caller flag
8. CORR - right-justified HC corridor assignment
9. LEASAD - leased buffer address
10. PACSIZ - sequence-packet size

c. Call to Lease for packet buffer.

3.2.2.4.1.7.3

Internal Data Description

a. Required-corridors count:

CORCNT has range of one through eight

b. Required-corridors array:

CORARY is a 1x8 array used to collect up to eight

unique corridor numbers necessary to control the allocated sequence.

c. HC buffers count:

BUFCNT has a range of one through three to indicate the differing orientations of the HCs involved in this sequence command.

d. HC corridor assignment:

CORR has a range of one through eight.

e. Packet buffer:

BUFR is a 2067-word area used for building a sequence packet.

3.2.2.4.1.7.4 Flowchart

See Figure 3.2.2-14 for the CMDSAL flowchart.

3.2.2.4.1.8 Submodule VIII - CMDTRK

3.2.2.4.1.8.1 Description

CMDTRK is responsible for building beam pointing/target HC commands for the TRACK and increase input commands. It uses the HC numbers placed in the IMARAY array by CMDARA.

a. Language used - FORTRAN IV

b. How invoked - called by CMDSCK

c. Constraints and limitations - None

d. Processing -

1. Set LOOP equal to IMSIZE.
2. Perform DO-UNTIL processing where I equals one to LOOP. If I is greater than LOOP, return to caller. Let N equal IMARAY(I).
3. Set HCDATG(N) equal to AIMPTG(N) for all five words of target X,Y,Z coordinates.
4. Set HCCMDG(N) equal to #8020. Go to step (2).

e. Error messages and recovery - None

3.2.2.4.1.8.2 Data, Logic and Command Paths

Input data:

a. Global common:

AIMPTG - memory-resident aim-point array

b. Local common:

1. IMARAY - non-sequence HC number array
2. IMSIZE - IMARAY size

Output data:

a. Global common:

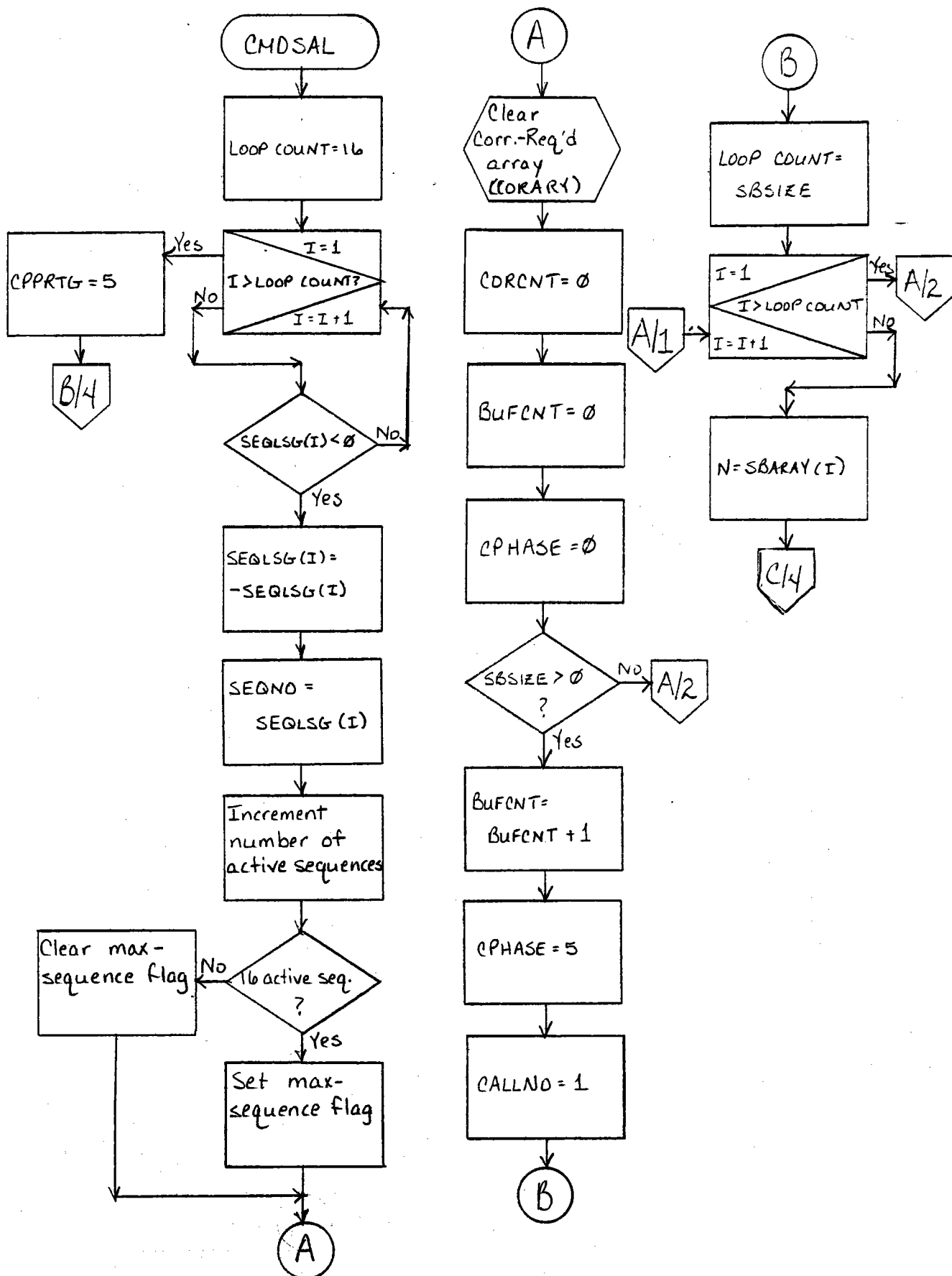


Figure 3.2.2-14 Flowchart - CMD5AL
181

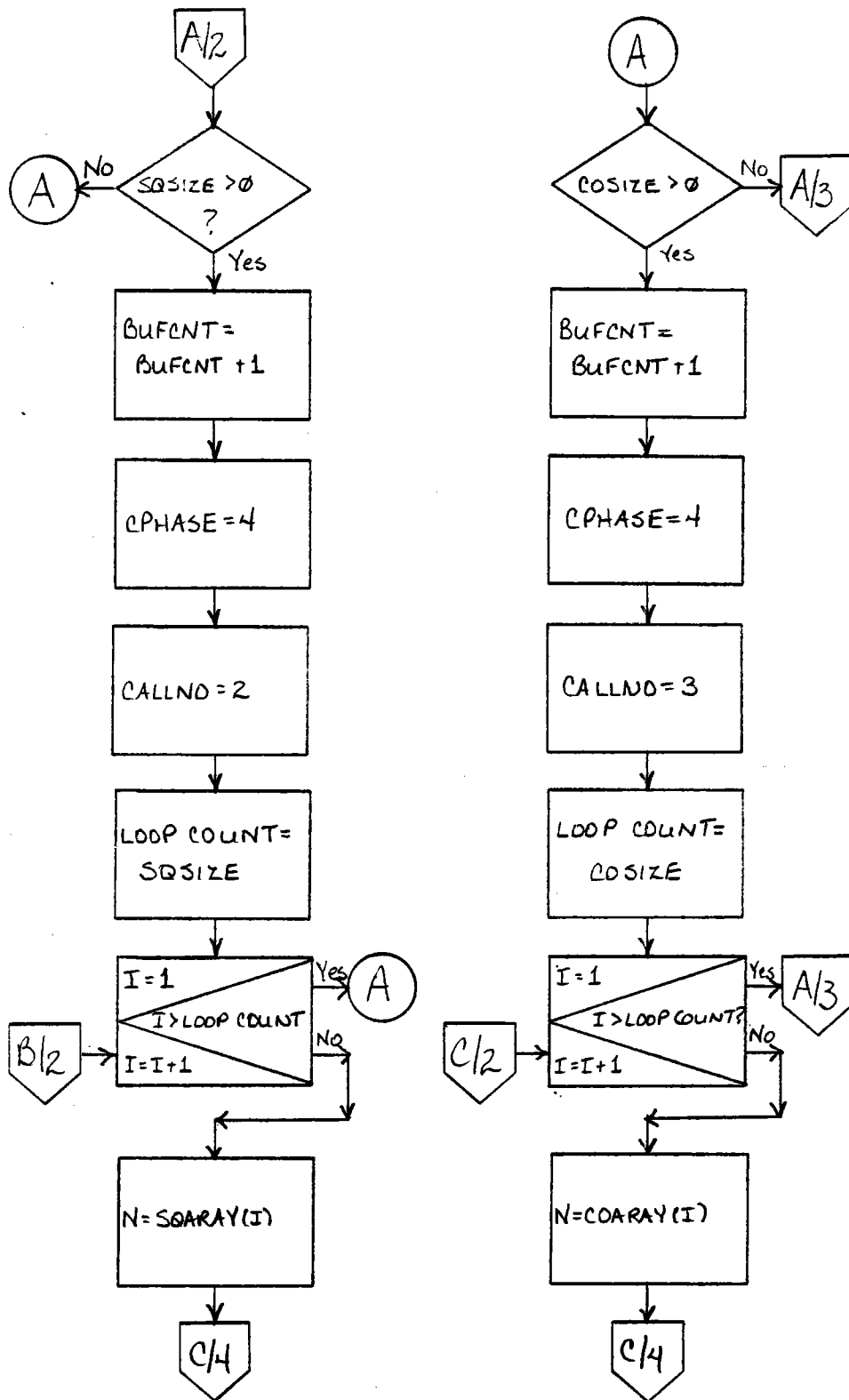


Figure 3.2.2-14 Flowchart - CMDSAL (continued)

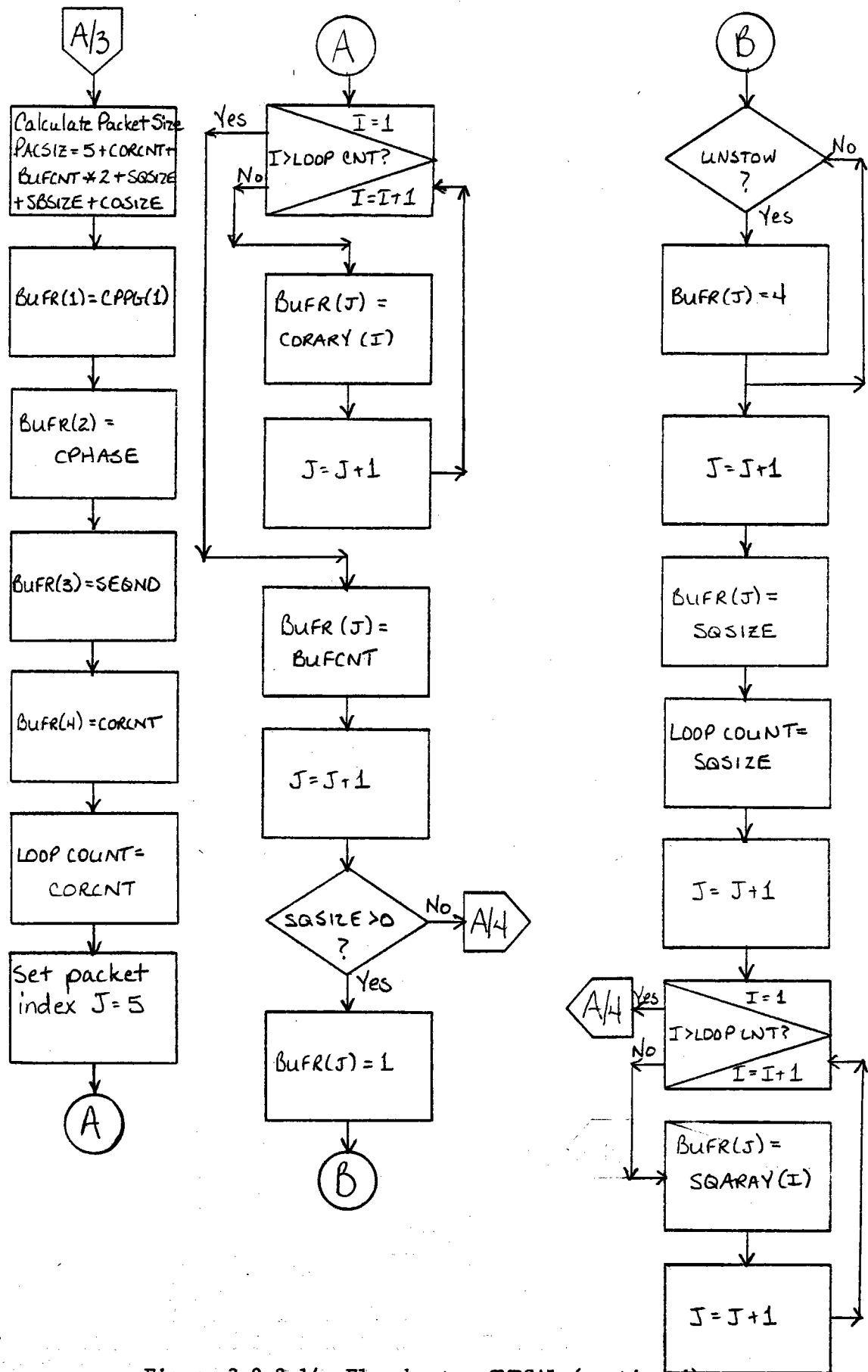


Figure 3.2.2-14 Flowchart - CMDSAL (continued)

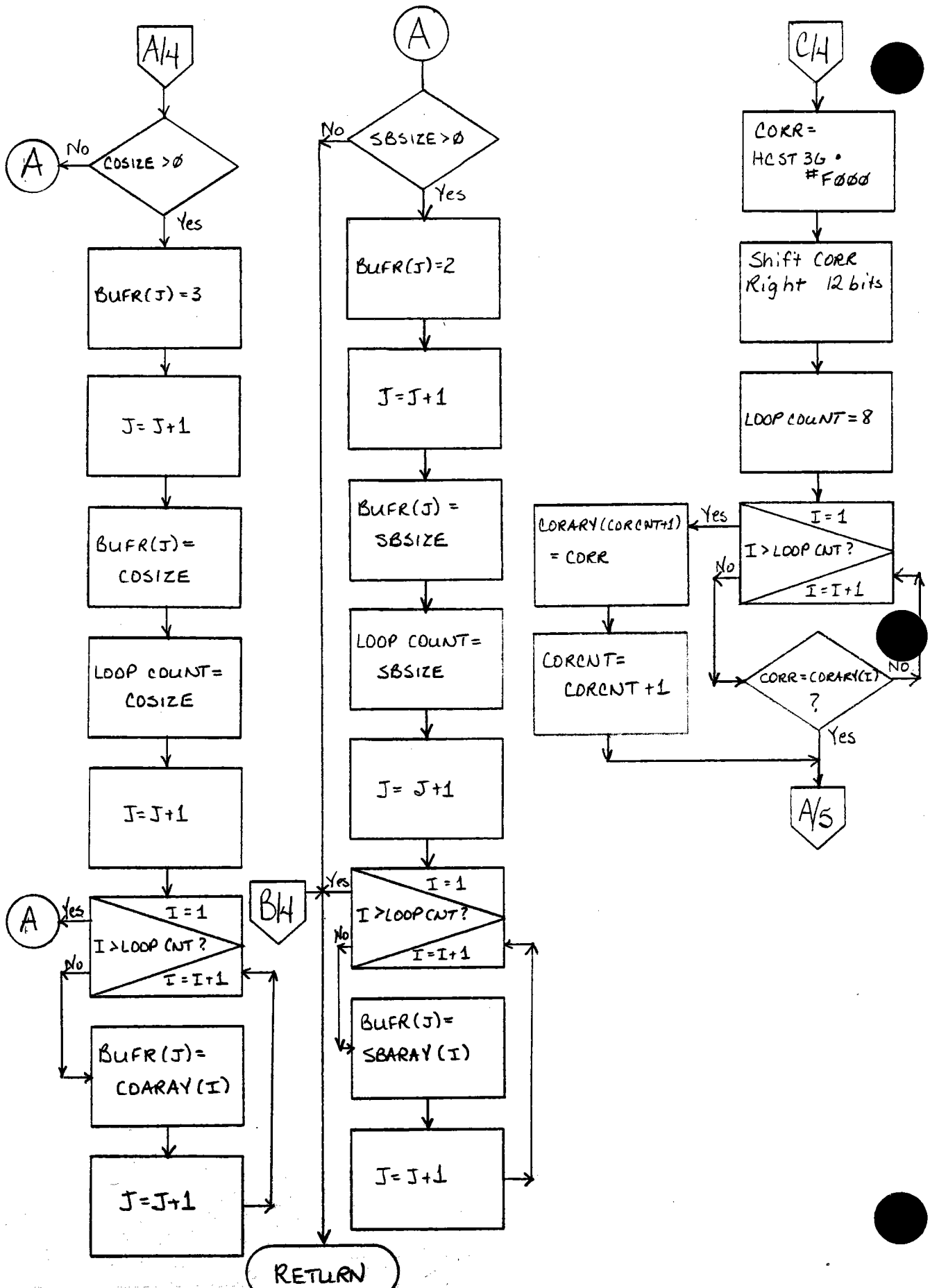


Figure 3.2.2-14 Flowchart - CMDSAL (continued)

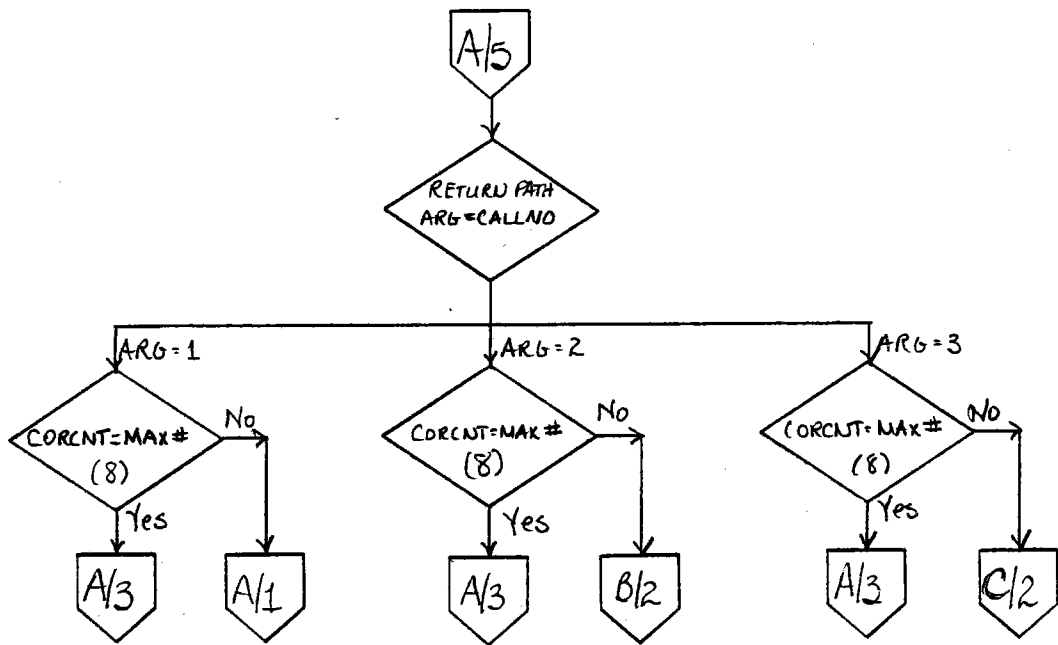


Figure 3.2.2-14 Flowchart - CMD5AL (continued)

1. HCDATG - HC data output array
2. HCCMDG - HC command output array

b. Local common:

LOOP - loop count

3.2.2.4.1.8.3 Internal Data Description

There is no data internal to this submodule.

3.2.2.4.1.8.4 Flowchart

See Figure 3.2.2-15 for the CMDTRK flowchart.

3.2.2.4.1.9 Submodule IX - CMDDSK

3.2.2.4.1.9.1 Description

CMDDSK is responsible for building the disk packet of HC numbers to be sent to task GET for those commands which require disk data to determine the output commands. This submodule is either called by CMDSCK for the LOAD, WASH, SAVE, RESTORE, ALT2STOW, AIMPOINT, and ESTOW input commands, or may also be called by CMDSDN submodule when certain addressed HCs can move to the Stow or Alt1stow positions without beam-safety requirements.

a. Language used - FORTRAN IV

b. How invoked - called by CMDSCK or CMDSDN

c. Constraints or limitations - None

d. Processing -

1. A local buffer area is used to build the disk packet for task GET. Set up the header as follows:
 - a) BUF(1) = CPPG(1), the command number
 - b) BUF(2) = command phase equals three
 - c) BUF(3) = 0, sequence number
 - d) BUF(4) = CPPG(3) if command number equals 20 (AIMPOINT) this is the aim-point number specified by operator
= 1 otherwise to indicate one buffer being passed
2. If command is RESTORE, set BUF(5) equal to zero and go to step (5). Otherwise, set BUF(5) equal to IMSIZE to record the buffer size and set LOOP equal to IMSIZE. Set the buffer pointer J equal to six.
3. Perform DO-UNTIL processing where I equals one to LOOP. If I is greater than LOOP, go to step (5).

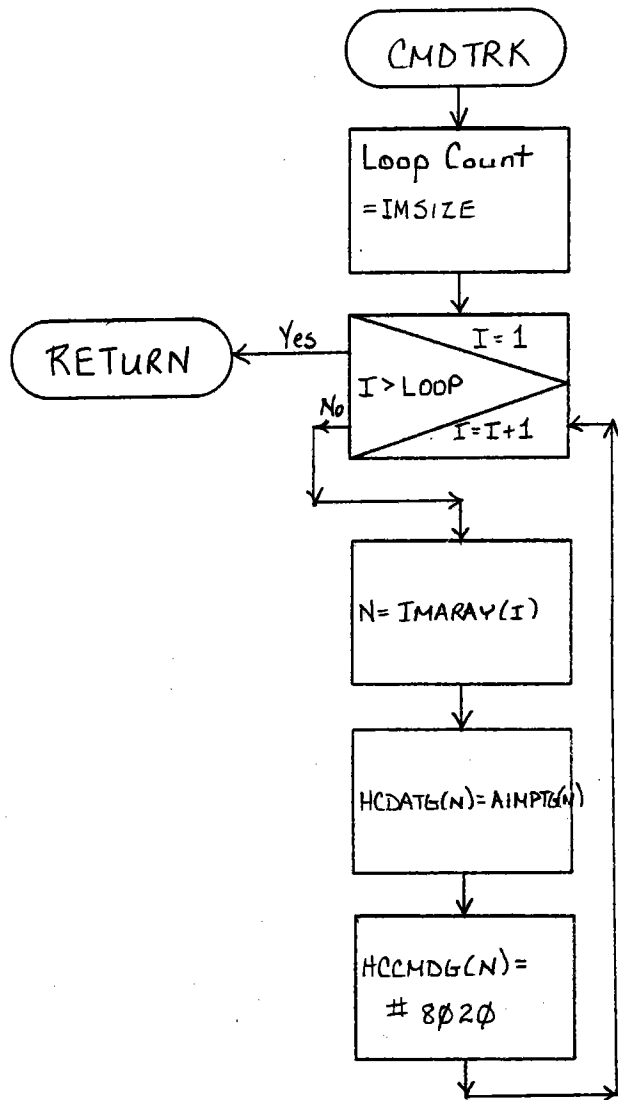


Figure 3.2.2-15 Flowchart - CMDTRK

4. Move the HC numbers in the IMARAY array to the GET packet. Increment J and go to step (3)
5. Set the disk-data flag (DISDAT) and return to caller.

e. Error messages and recovery - None

3.2.2.4.1.9.2 Data, Logic and Command Paths

Input data:

- a. Global common:
 1. CPPG(3) - aim-point array number for command number equals 20.
 2. CPPG(1) - command number
- b. Local common:
 1. IMARAY - non-sequence HC number array
 2. IMSIZE - IMARAY size

Output data:

Local common:

- a. Local common buffer BUF used to build disk packet (see Table 3.2.2-VI for format)
- b. DISDAT - disk-data flag
- c. LOOP - loop count

3.2.2.4.1.9.3 Internal Data Description

There is no data internal to this submodule.

3.2.2.4.1.9.4 Flowchart

See Figure 3.2.2-16 for the CMDDSK flowchart.

3.2.2.4.1.10 Submodule X - CMDPOS

3.2.2.4.1.10.1 Description

CMDPOS is responsible for building all AZ/EL pointing output commands (MARK, POSITION, INLINE, OFFLINE, HOLD, and RELWASH) which do not require disk data. It also processes the ONLINE command which only resets global common bits in the HC's HCST2G word.

- a. Language used - FORTRAN IV
- b. How invoked - called by CMDSCK.
- c. Constraints and limitations - ONLINE processing will not reset HCs which are unmarked.
- d. Processing -
 1. Set LOOP equal to IMSIZE.
 2. Perform DO-UNTIL processing where I equals one

<u>WORD</u>	<u>DESCRIPTION</u>
1	COMMAND #
2	COMMAND PHASE: = 3: WHEN CMD ENQUES GET = 9: WHEN SEQ ENQUES GET
3	SEQUENCE #: = 0: WHEN CMD ENQUES GET ≠ 0: WHEN SEQ ENQUES GET
4	AIMPOINT # (IF COMMAND # = 20) # OF HC BUFFERS (FOR ALL OTHER COMMANDS)
5	BUFFER SIZE
6	HC # BUFFER LEASE OFFSET: = 0: RESTORE COMMAND = 0: ALL OTHER COMMANDS REQUIRING DISK DATA

Table 3.2.2-VI Disk Data Command Packet

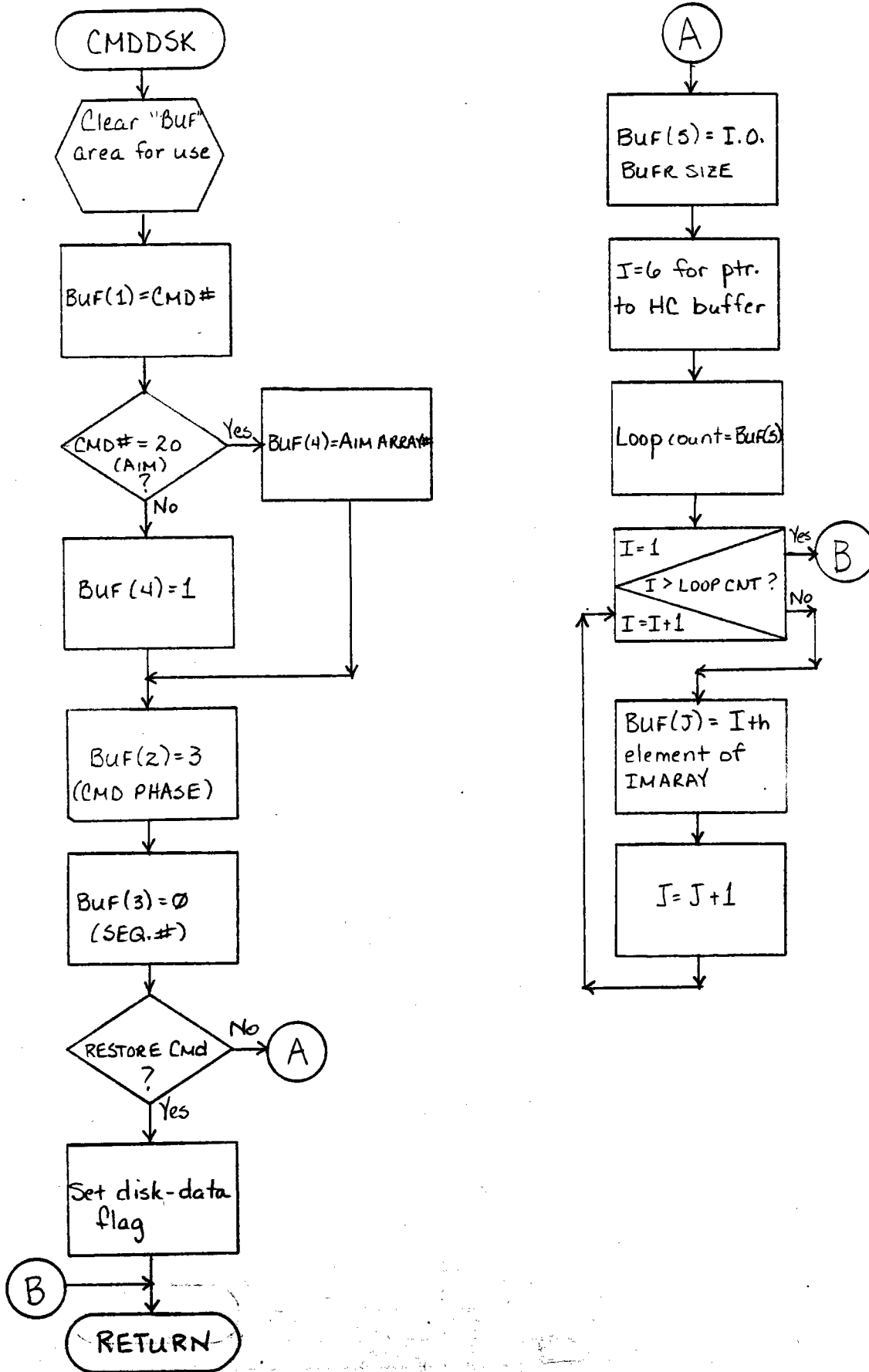


Figure 3.2.2-16 Flowchart - CMDDSK

to LOOP. If I is greater than LOOP, return to caller. Let N equal IMARAY(I).

3. If command is ONLINE, reset HCST2G(N), bits 1 through 3 and 8. Go to step (2).
4. If command is MARK, insert the local Mark mode values into HCDATG(N). Set HCCMDG(N) equal to #8064, and go to step (2).
5. If command is POSITION, insert the operator-specified AZ/EL contained in CPPG(3) and CPPG(4) into HCDATG(N). Set HCCMDG(N) equal to #806C, and go to step (2).
6. If command is OFFLINE, set HCST2G(N), bit three and set HCDATG(N) to AZIMG(N) and ELEVG(N), the last reported AZ/EL values from the field. Set HCCMDG(N) equal to #806C, and go to step (2).
7. If command is HOLD, set HCDATG(N) equal to AZIMG(N) and ELEVG(N). Set HCCMDG(N) equal to #906C. Increment EMCC1G(2) by one to tell the BHC task how many type-2 critical commands exist in the HCCMDG array. Go to step (2).

e. Error messages and recovery - None

3.2.2.4.1.10.2 Data, Logic and Command Paths

Input data:

a. Global common:

1. CPPG(1) - command number
2. CPPG(3) - operator-specified azimuth for POSITION
3. CPPG(4) - operator-specified elevation for POSITION
4. AZIMG - last reported azimuth array
5. ELEVG - last reported elevation array

b. Local common:

1. IMARAY - non-sequence HC number array
2. IMSIZE - IMARAY size
3. MARKAZ - Mark mode azimuth
4. MARKEL - Mark mode elevation

Output data:

Global common:

- a. HCDATG - HC-data output array
- b. HCCMDG - HC-command output array
- c. HCST2G - HC derived-status array
- d. EMCC1G(2) - type-2 critical command count

3.2.2.4.1.10.3 Internal Data Description

There is no data internal to this submodule.

3.2.2.4.1.10.4 Flowchart

See Figure 3.2.2-17 for the CMDPOS flowchart.

3.2.2.4.1.11 Submodule XI - CMDSBY

3.2.2.4.1.11.1 Description

CMDSBY is responsible for building non-sequence, beam pointing/ track-the-CULP output commands for the STANDBY, RETURN, DEFOCUS, DECREASE, and ESTANDBY input commands. It uses the HC numbers placed in the IMARAY array by CMDARA.

- a. Language used - FORTRAN IV
- b. How invoked - called by CMDSCK
- c. Constraints and limitations - None
- d. Processing -
 1. Set LOOP equal to IMSIZE.
 2. Perform DO-UNTIL processing where I equals one to LOOP. If I is greater than LOOP, return to caller. Let N equal IMARAY(I).
 3. Determine the HC's corridor assignment, J equals HCST3G(N), bits zero through three. Set HCDATG(N) equal to CORRCG(1-5,J) for all five words of CULP X,Y,Z coordinates. If command is not DEFOCUS, go to step (5).
 4. Set HCCMDG(N) equal to #9030. Increment EMCC1G(1) to tell task BHC how many type-1 critical commands exist in the HCCMDG array. Go to step (2).
 5. Extract the BCS target assignment for the HC if the command is RETURN. If not RETURN, go to step (6). Right-justify the BCS target
 $(TGT) = HCST3G(N).AND.#60.$
Compute the mask for BCSBYG by:
 $BCSMSK = 2**TGT.$
Reset the BCS-busy status bit in BCSBYG by:
 $BCSBYG = BCSBYG.XOR.BCSMSK.$
 6. Set HCCMDG(N) equal to #8030, and go to step (2).
- e. Error messages and recovery - None

3.2.2.4.1.11.2 Data, Logic and Command Paths

Input data:

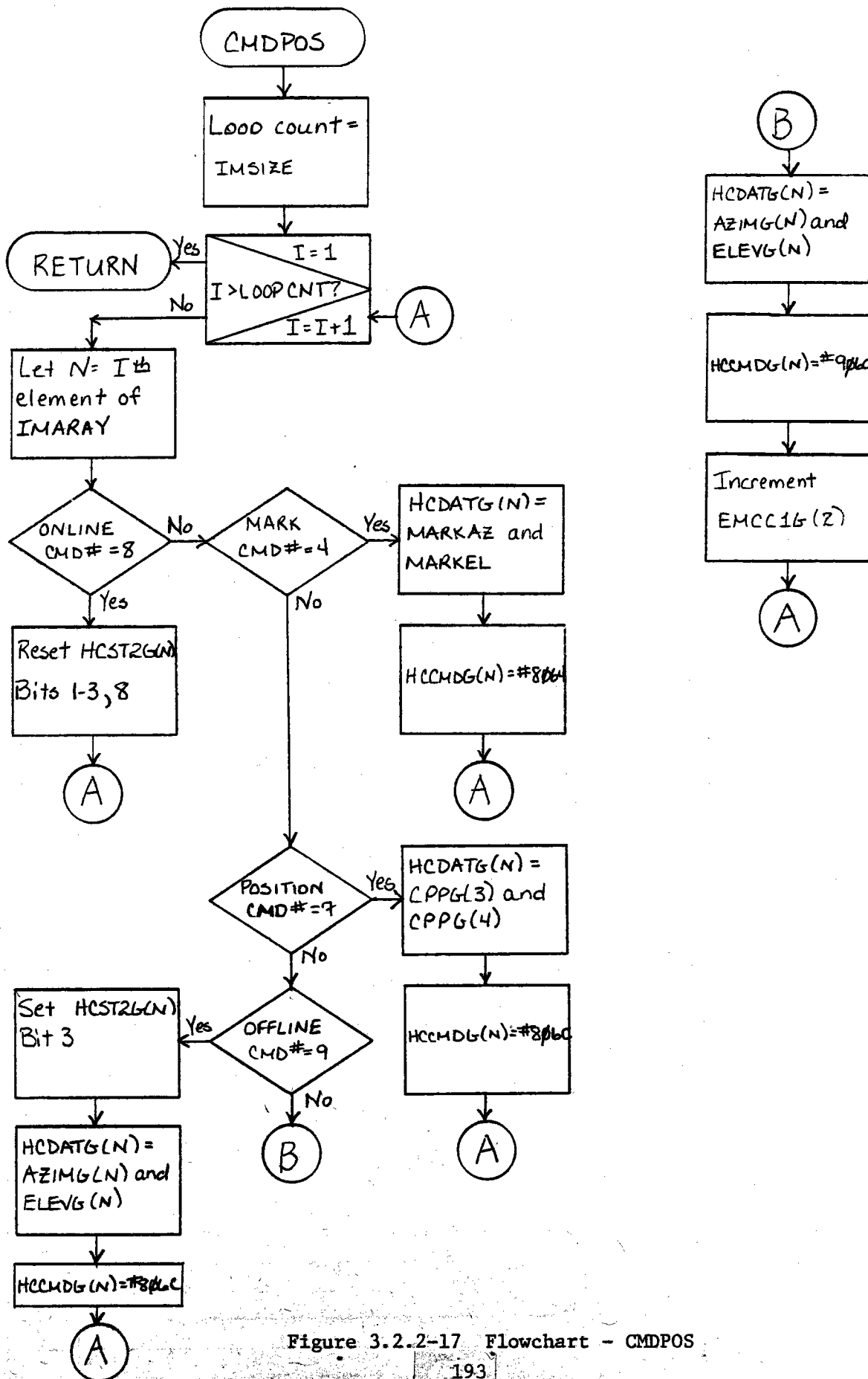


Figure 3.2.2-17 Flowchart - CMDPOS

a. Global common:

1. CPPG(1) - command number
2. HCST3G - HC corridor and BCS assignment
3. CORRCG - corridor coordinates
4. BCSBYG - BCS-busy status

b. Local common:

1. IMARAY - non-sequence HC number array
2. IMSIZE - IMARAY size

Output data:

a. Global common:

1. HCDATG - HC data output array
2. HCCMDG - HC command output array
3. EMCC1G(1) - type-1 critical command count
4. BCSBYG - BCS-busy status

b. Local common:

1. LOOP - loop count
2. TGT - BCS assignment
3. BCMSK - BCS status mask

3.2.2.4.1.11.3 Internal Data Description

There is no data internal to this submodule.

3.2.2.4.1.11.4 Flowchart

See Figure 3.2.2-18 for the CMDSBY flowchart.

3.2.2.4.1.12 Submodule XII - CMDSDN

3.2.2.4.1.12.1 Description

CMDSDN is responsible for building output commands, if necessary, for the STOW, STHIWIND, and the ALT1STOW input commands. If any involved HCs are in a non-sequence orientation, they are assembled into a disk packet in submodule CMDDSK. Sequence orientations are checked and HCs are moved towards their respective CULPs if required.

a. Language used - FORTRAN IV

b. How invoked - called by CMDSCK for down-corridor initial processing

c. Constraints and limitations - None

d. Processing -

1. Check IMSIZE to be non-zero. If so, there are some involved HCs which can be moved without

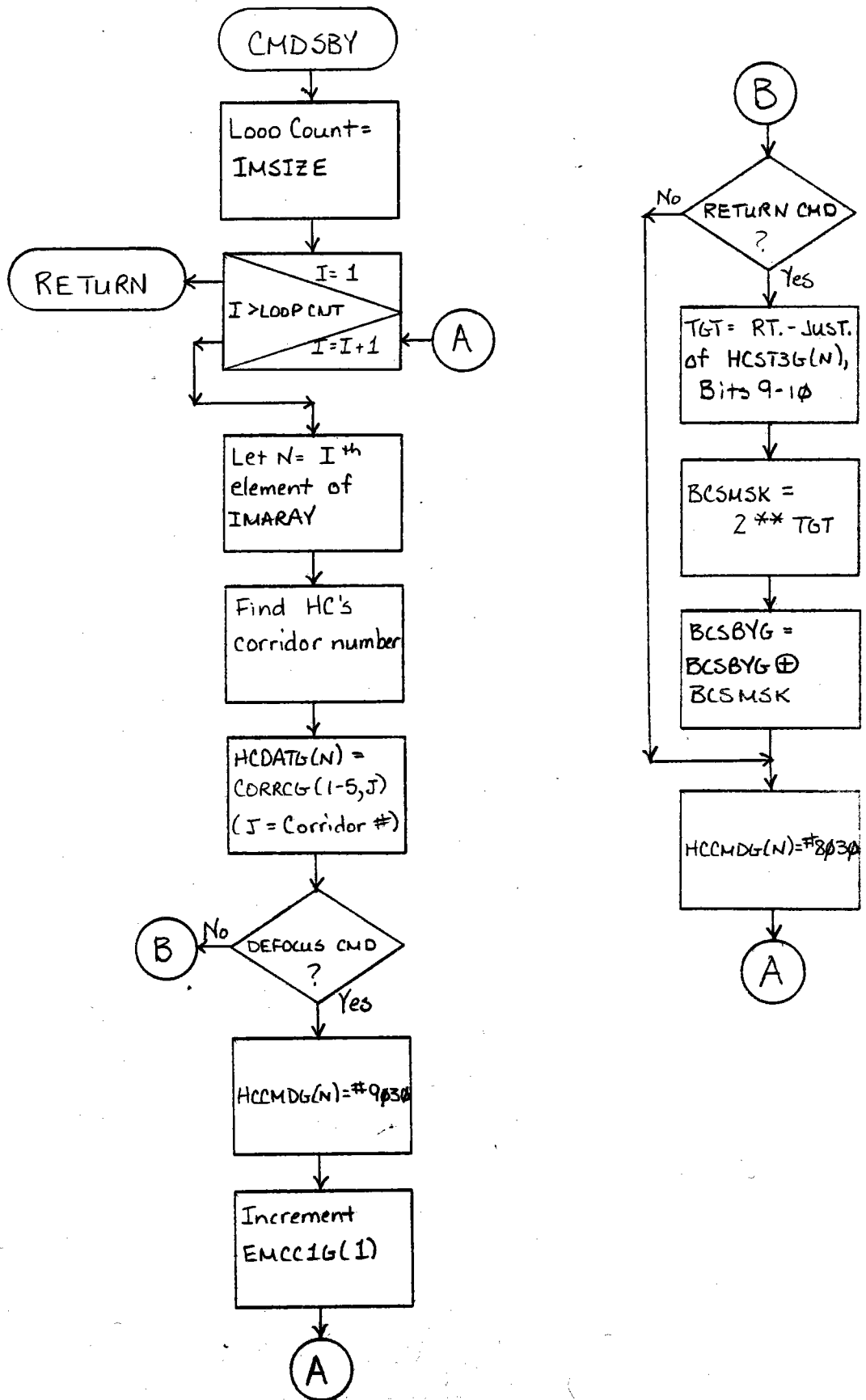


Figure 3.2.2-18 Flowchart - CMDSBY

beam-safety requirements. Call CMDDSK to handle this orientation. If IMSIZE equals zero, go directly to step (2).

2. Check the sequence-required flag (SEQREQ), and if not set, go to step (9). If set, check if the command is STHIWIND. If so, set BIT3 equal to #1000; otherwise, set BIT3 to zero. This word is used later in building the HCCMDG word (bit 3 is the critical-command bit).
3. Set LOOP equal to SQSIZE.
4. Perform DO-UNTIL processing where I equals one to LOOP. If I is greater than LOOP, go to step (9).
5. Set N equal to SQARRAY(I) to determine HC number.
6. Determine the HC's corridor assignment J equal to HCST3G(N), bits zero through three. Set HCDATG(N) equal to CORRCG(1-5,J) for all five words of CULP X,Y,Z coordinates.
7. Set HCCMDG(N) equal to #8030 plus BIT3.
8. If BIT3 is non-zero, increment EMCC1G(1) to tell task BHC how many type-1 critical commands exist in the HCCMDG array. If not ALT1STOW, set stow-sequence bit in HCST3G, and go to step (4)
9. Return to caller.

e. Error messages and recovery - None

3.2.2.4.1.12.2 Data, Logic and Command Paths

Input data:

- a. Global common:
 1. HCST3G - HC corridor assignment
 2. CORRCG - corridor coordinates
- b. Local common:
 1. IMARRAY - non-sequence HC number array
 2. IMSIZE - IMARRAY size
 3. SQSIZE - SQARRAY size
 4. SQARRAY - HCs-not-at-standby array
 5. SEQREQ - sequence-required flag

Output data:

- a. Global common:
 1. HCDATG - HC-data output array
 2. HCCMDG - HC-command output array
 3. HCST3G - HC-in-Stow-sequence status bit

b. Local common:

LOOP - loop count

3.2.2.4.1.12.3 Internal Data Description

Critical-command bit value:

BIT3 = 0: non-critical command

= #1000: critical command (STHIWIND)

3.2.2.4.1.12.4 Flowchart

See Figure 3.2.2-19 for the CMDSDN flowchart.

3.2.2.4.1.13 Submodule XIII - CMDBCS

3.2.2.4.1.13.1 Description

CMDBCS is responsible for building manual BCS output commands whenever the BCSTRACK input command is entered. Only one HC may be tracking a BCS target at one time.

a. Language used - FORTRAN IV

b. How invoked - called by CMDSCK

c. Constraints and limitations - only one HC may be commanded for each entry of the BCSTRACK command.

d. Processing -

1. Set HC equal to IMARAY(1) as this will be the only HC processed.
2. Extract and right-justify the HC's BCS target number (TGT) from HCST3G(HC), bits nine through ten. It will have a range of zero through three.
3. Determine BCS-busy mask by setting:
 $BCSMSK = 2^{**}TGT.$
4. Determine the BCS-target busy status by setting:
 $BCSSTA = BCSMSK.AND.BCSBYG.$
5. If result is non-zero, set the internal-error flag (EFLAGI) to four, and go to step (9). If result is zero, set the BCS-target busy by setting:
 $BCSBYG = BCSBYG + BCSMSK.$
6. Set TGT equal to TGT plus one for index purposes.
7. Set HCDATG(HC) equal to BCSTGG(1-5,TGT).
8. Set HCCMDG(HC) equal to #802C.
9. Return to caller.

e. Error messages and recovery - If the BCS target is already busy, EFLAGI is set to four.

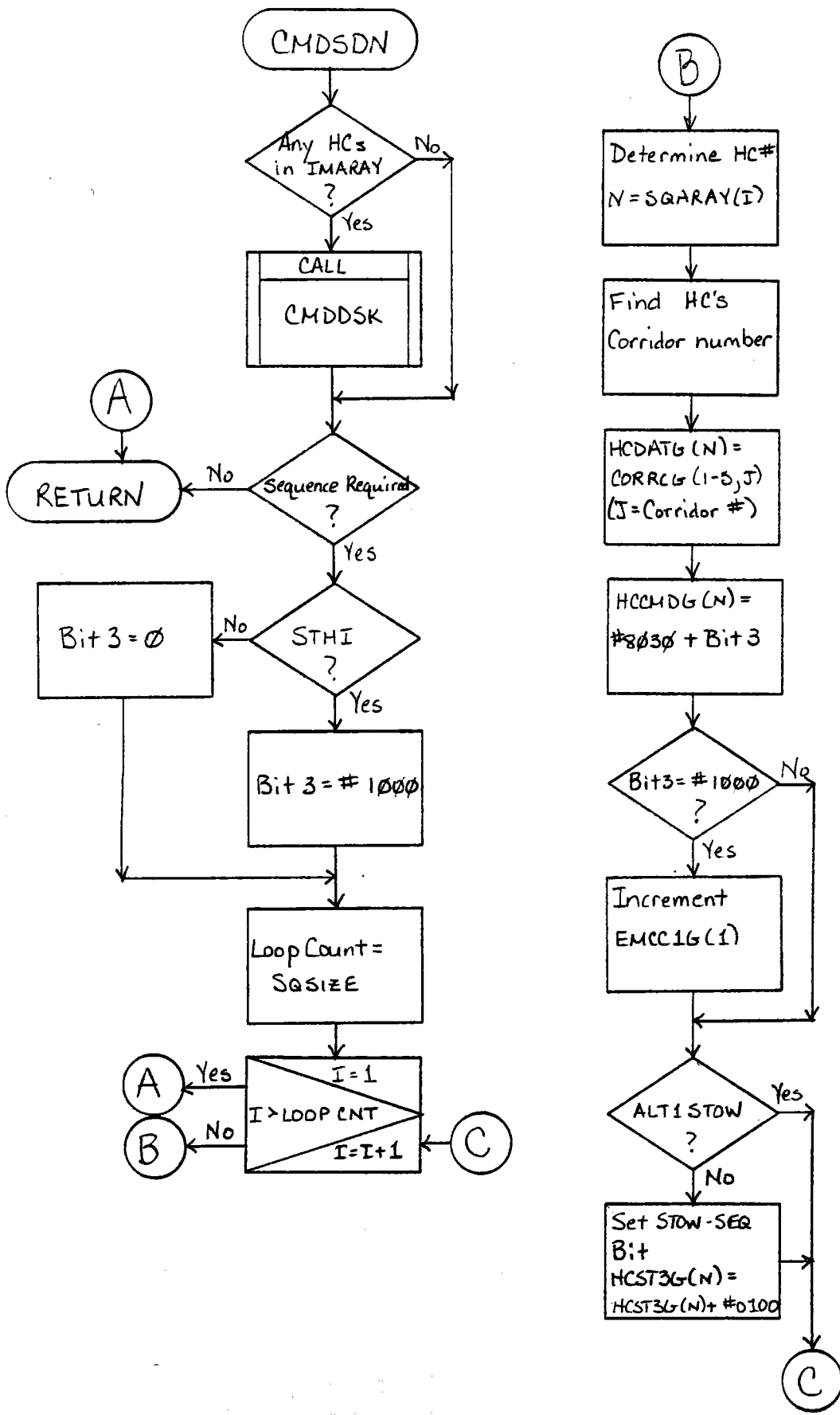


Figure 3.2.2-19 Flowchart - CMDSDN

3.2.2.4.1.13.2 Data, Logic and Command Paths

Input data:

Global common:

- a. HCST3G - HC's BCS target number
- b. BCSBYG - BCS-target busy status
- c. BCSTGG - BCS target coordinates

Output data:

a. Global common:

1. HCDATG - HC-data output array
2. HCCMDG - HC-command output array
3. BCSBYG - BCS-target busy status

b. Local common:

1. TGT - BCS target index
2. BCMSK - bit mask for BCSBYG
3. EFLAGI - CMD-error flag
4. BCSTA - BCS-target busy status

3.2.2.4.1.13.3 Internal Data Description

a. BCS target index:

TGT has value of zero through three when right-justified from HCST3G(HC).

b. Bit mask for BCSBYG:

BCMSK has value of one, two, four, or eight.

3.2.2.4.1.13.4 Flowchart

See Figure 3.2.2-20 for the CMDBCS flowchart.

3.2.2.4.1.14 Submodule XIV - CMDSUP

3.2.2.4.1.14.1 Description

DMDSUP is responsible for building output commands for the UNSTOW input command. If any involved HCs are in the Alt2stow orientation, they are immediately commanded to their CULPs without beam-safety requirements. HCs in the Stow and Alt1stow orientations are commanded to their CLLPs as the first step of the up-corridor sequence.

- a. Language used - FORTRAN IV
- b. How invoked - called by CMDSCK
- c. Constraints and limitations - None
- d. Processing -

1. Check IMSIZE to determine if any HCs can be

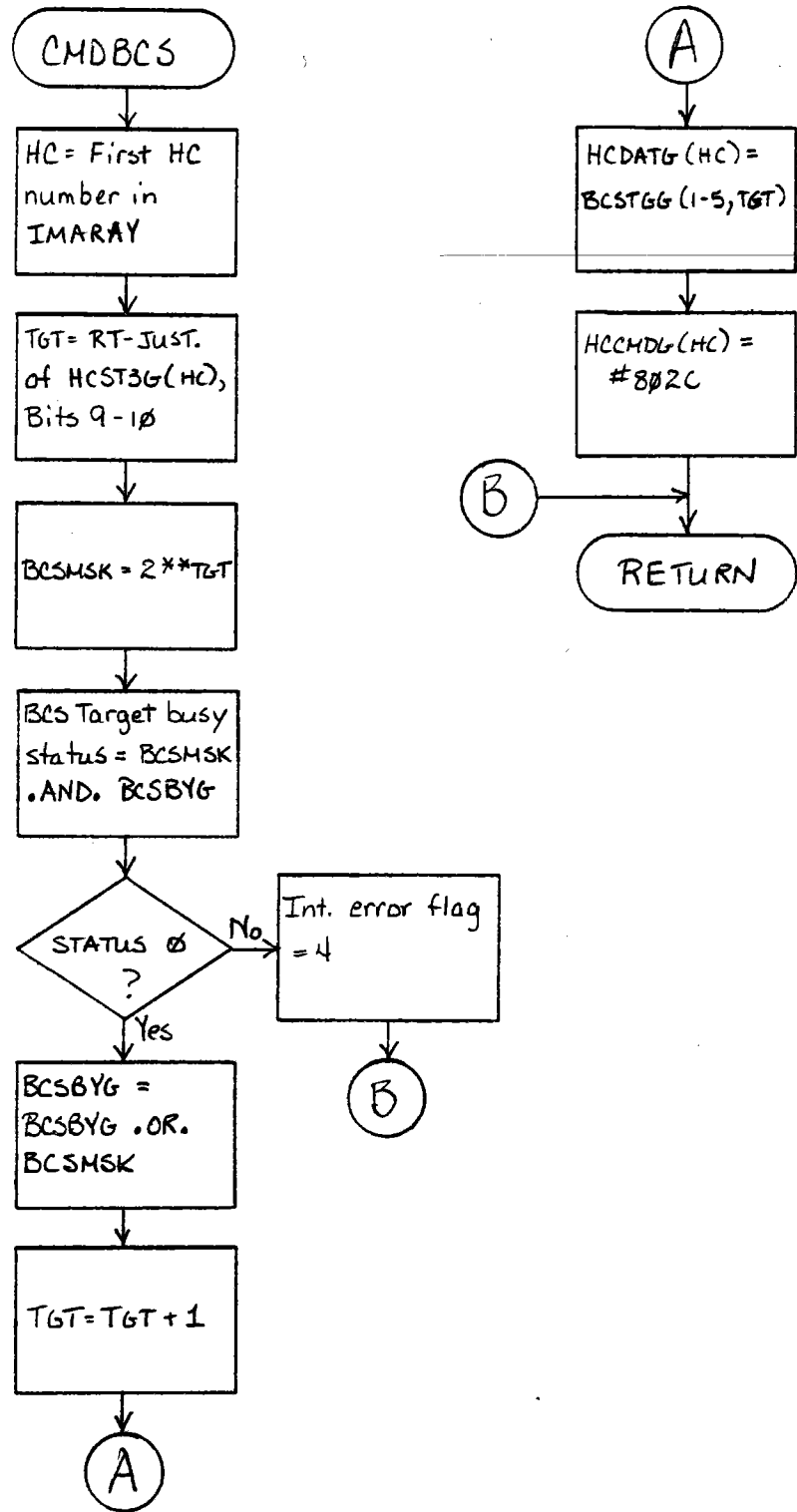


Figure 3.2.2-20 Flowchart - CMDBCS

moved without beam-safety considerations. If zero, go to step (6). Otherwise, set LOOP equal to IMSIZE.

2. Perform DO-UNTIL processing where I equals one to LOOP. If I is greater than LOOP, go to step (6).
3. Set N equal to IMARAY(I) to determine HC number.
4. Determine the HC's corridor assignment, J equals HCST3G(N), bits zero through three. Set HCDATG(N) equal to CORRCG(1-5,J) for all five words of CULP X,Y,Z coordinates.
5. Set HCCMDG(N) equal to #8030, and go to step (2).
6. Check the sequence-required flag (SEQREQ) to determine if any involved HCs are in a Stow or Alt1stow orientation. If not, go to step (11). If so, set LOOP equal to SQSIZE.
7. Perform DO-UNTIL processing where I equals one to LOOP. If I is greater than LOOP, go to step (11).
8. Set N equal to SQARAY(I) to determine HC number.
9. Determine HC's corridor assignment J equals HCST3G(N), bits zero through three. Set HCDATG(N) equal to CORRCG(6-10,J) for all five words of CLLP X,Y,Z coordinates.
10. Set HCCMDG(N) equal to #8034, and go to step (7).
11. Return to caller.

e. Error messages and recovery - None

3.2.2.4.1.14.2 Data, Logic and Command Paths

Input data:

- a. Global common:
 1. HCST3G - HC corridor assignment
 2. CORRCG - corridor coordinates
- b. Local common:
 1. IMARAY - non-sequence HC number array
 2. IMSIZE - IMARAY size
 3. SQARAY - HC-not-at-Standby HC number array
 4. SQSIZE - SQARAY size
 5. SEQREQ - sequence-required flag

Output data:

- a. Global common:
 1. HCDATG - HC-data output array
 2. HCCMDG - HC-command output array
- b. Local common:

LOOP - loop count

3.2.2.4.1.14.3 Internal Data Description

There is no data internal to this submodule.

3.2.2.4.1.14.4 Flowchart

See Figure 3.2.2-21 for the CMDSUP flowchart.

3.2.2.4.1.15 Submodule XV - GET

3.2.2.4.1.15.1 Description

GET is responsible for disk-data accesses for operational commands. HC numbers are passed in a disk packet. GET Deques the entire packet into a local buffer DBUF and preserves the format in case it needs to send it back to the task SEQ for sequence deletion.

- a. Language used - FORTRAN IV
- b. How invoked - Enqued by CMD or SEQ
- c. Constraints and limitations - DBUF buffer is 2048 words long, assuming no more than 2048 HC numbers may be passed in a packet.
- d. Processing -

1. Initialize:

Zero the SEQ-call flag (SEQCAL), the disk-error flag (DKERR), the record numbers in memory (MREC1, MREC2), and set the DBUF Index J to five.

2. Call Deque REX service to obtain the disk data packet and have it transferred to local buffer DBUF.
3. Examine DBUF(2). If set to nine, set SEQCAL to one. Set LOOP equal to DBUF(5) to establish the loop count.
4. Submit the command number in DBUF(1) to a CASE statement to determine appropriate processing submodule to call:

- a) Call GETINI if command number = 3(Load);
- b) Call GETAIM if command number = 20(AIMPOINT);
- c) Call GETAL1 if command number = 14(ALT1STOW);
- d) Call GETAL2 if command number = 15(ALT2STOW);
- e) Call GETWSH if command number = 13(WASH);
- f) Call GETSTO if command number = 6, 18 or 25(STOW, STHI, or ESTOW)

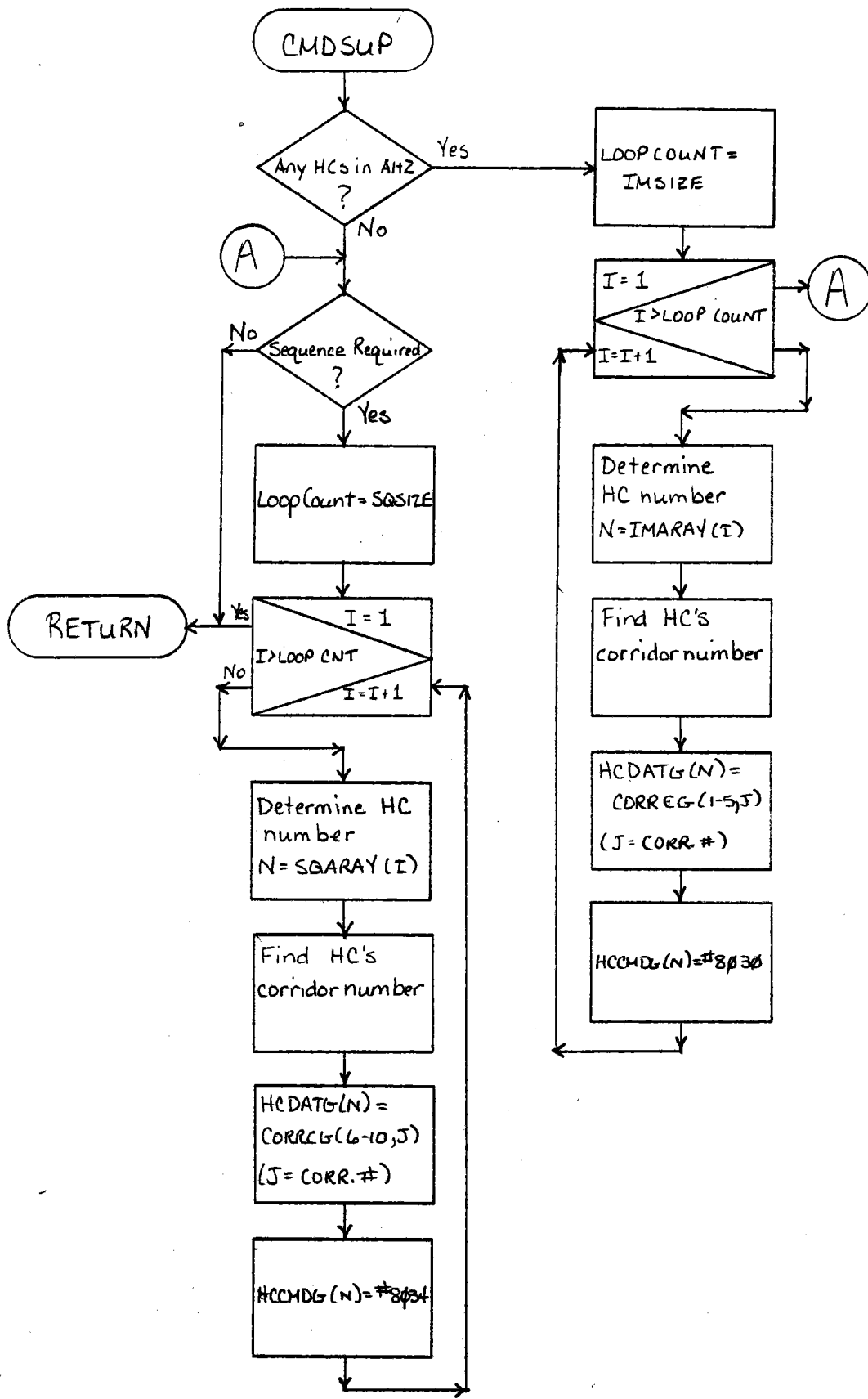


Figure 3.2.2-21 Flowchart - CMLSUP

g) Call GETSAR if command number = 16 or 22
(SAVE
RESTORE)

5. All submodules return to this step. Examine DKERR. If zero, go to step (6). If non-zero, Enque the alarms output task ALO with a disk-read error critical-alarm message. Go to step (7).
6. Examine the SEQ-call flag (SEQCAL). If zero, go to step (7). Otherwise, revise the command phase word in DBUF by setting it to ten (delete sequence-command phase). Enque SEQ using the DBUF address.
7. Relinquish control of task GET.

e. Error messages and recovery - a disk-read error message is sent to task ALO for any disk error.

3.2.2.4.1.15.2 Data, Logic and Command Paths

Input data:

Local common:

- a. Buffer DBUF contains all packet information
- b. DKERR - disk-error flag
- c. SEQCAL - SEQ-call flag

Output data:

- a. Enque of SEQ
- b. Enque of ALO
- c. DBUF(3) - command phase

3.2.2.4.1.15.3 Internal Data Description

See Table 3.2.2-VI for disk data-packet format.

3.2.2.4.1.15.4 Flowchart

See Figure 3.2.2-22 for the GET flowchart.

3.2.2.4.1.16 Submodule XVI - GETINI

3.2.2.4.1.16.1 Description

GETINI is responsible for fetching the HC bias AZ/EL angles from file HCB, the HC X,Y,Z coordinates from disk file HCC, and the last reported AZ/EL angles for the LOAD input command. HC numbers come from the local buffer DBUF.

- a. Language used - FORTRAN IV
- b. How invoked - called by GET

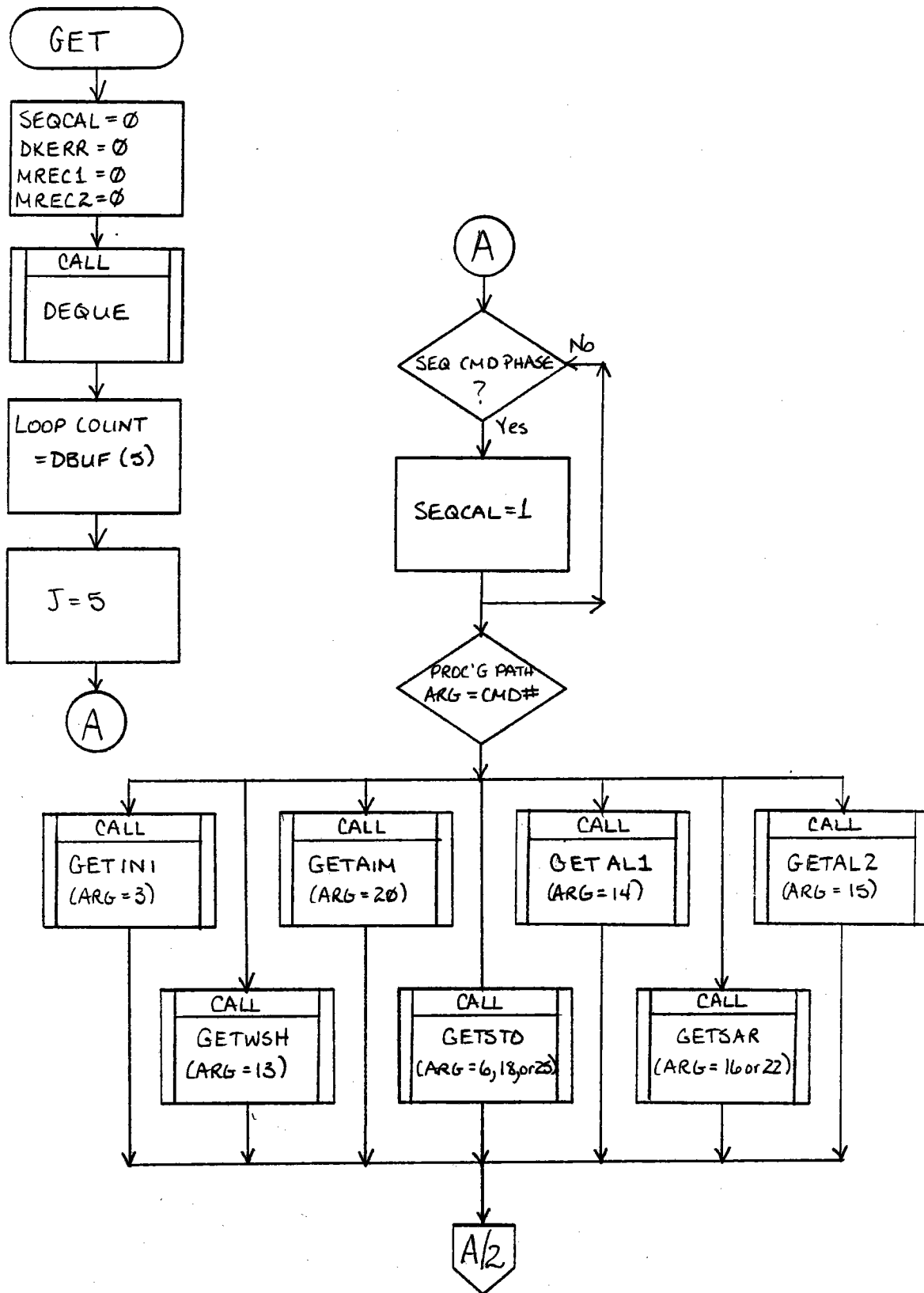


Figure 3.2.2-22 Flowchart - GET

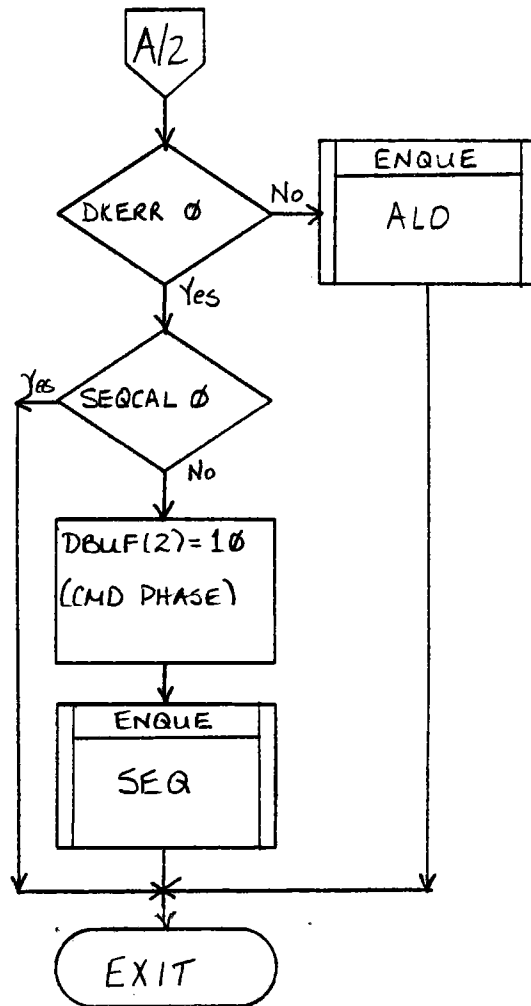


Figure 3.2.2-22 Flowchart - GET (continued)

c. Constraints and limitations - None

d. Processing -

1. Perform DO-UNTIL processing where I equals one to LOOP. If I is greater than LOOP, go to step (16).
2. Increment J by one.
3. Set HC equal to DBUF(J).
4. Zero out all bits in HCST2G(HC) except the not-installed (bit 0) and the error-reporting (bits 11-15).
5. Set the unmarked bit (bit 4) in HCST2G(HC).
6. Compute the HC bias record number:
$$BREC=((HC-1)/64)+1.$$
7. Test if the bias record is already in memory by checking BREC equals MREC1. If so, go to step (8). If not, read the bias record into memory and set MREC1 equals BREC. If disk error occurs, set DKERR, and go to step (16).
8. Compute the HC's offset into the bias record.
9. Insert the bias AZ/EL angles into the first two words of HCDATG(HC).
10. Compute the HC coordinates record number:
$$CREC=((HC-1)/32)+1.$$
11. Test if the coordinates record is already in memory by checking CREC equals MREC2. If so, go to step (12). If not, read the coordinates record into memory and set MREC2 equals CREC. If disk error, set DKERR, and go to step (16).
12. Compute the HC's offset into the coordinates record.
13. Insert the five word HC X,Y,Z coordinates into words three through seven of HCDATG(HC).
14. Set HCCMDG(HC) equal to #9818.
15. This is a type-2 critical command. Increment EMCC1G(2) to tell task BHC how many type-2 commands exists in the HCCMDG array. Go to step (1).
16. Return to caller.

e. Error messages and recovery - for disk error, set DKERR and immediately return to caller.

3.2.2.4.1.16.2 Data, Logic and Command Paths

Input data:



- a. Local buffer DBUF contains HC number, size and HC numbers
- b. Disk file HCB (see Figure 3.2.2-23)
- c. Disk file HCC (see Figure 3.2.2-24)

Output data:

- a. Global common:
 - 1. HCST2G - derived HC status
 - 2. HCDATG - HC-data output array
 - 3. HCCMDG - HC-command output array
 - 4. EMCC1G(2) - type-2 critical command count
- b. Local common:
 - 1. MREC1 - bias record number in memory
 - 2. MREC2 - coordinates record number in memory
 - 3. BREC - fetched record number for bias
 - 4. CREC - fetched record number for coordinates
 - 5. DKERR - disk-error flag

3.2.2.4.1.16.3 Internal Data Description

There is no data internal to this submodule.

3.2.2.4.1.16.4 Flowchart

See Figure 3.2.2-25 for the GETINI flowchart.

3.2.2.4.1.17 Submodule XVII - GETAIM

3.2.2.4.1.17.1 Description

GETAIM is responsible for fetching the target aim-point X,Y,Z coordinates from the disk file AIM for the AIMPOINT input command. This data updates the memory-resident aim-point array AIMPTG. It also must test each HC whose number was passed in the data packet to determine if that HC is at, or approaching, the target. If so, the new target coordinates are sent to the HC. HC numbers come from the local buffer DBUF.

- a. Language used - FORTRAN IV
- b. How invoked - called by GET
- c. Constraints and limitations - None
- d. Processing -
 - 1. Set the aim-point record offset (OFSET) equal to DBUF(2) which is the aim-point array number.
 - 2. Perform DO-UNTIL processing where I equals one to LOOP. If I is greater than LOOP, go to step (9).

REC 1	REC 2	REC 3	REC 4		REC i		REC 31	REC 32
----------	----------	----------	----------	--	----------	--	-----------	-----------

HFC $i*2-2$					HFC $i*2-1$								
HC 0	HC 1	HC 2		HC j		HC 31	HC 0	HC 1	HC 2		HC j		HC 31

HC j			
WORD 0		WORD 1	
BYTE 0	BYTE 1	BYTE 2	BYTE 3
AZIMUTH		ELEVATION	

Figure 3.2.2-23 HC Bias File (HCB)

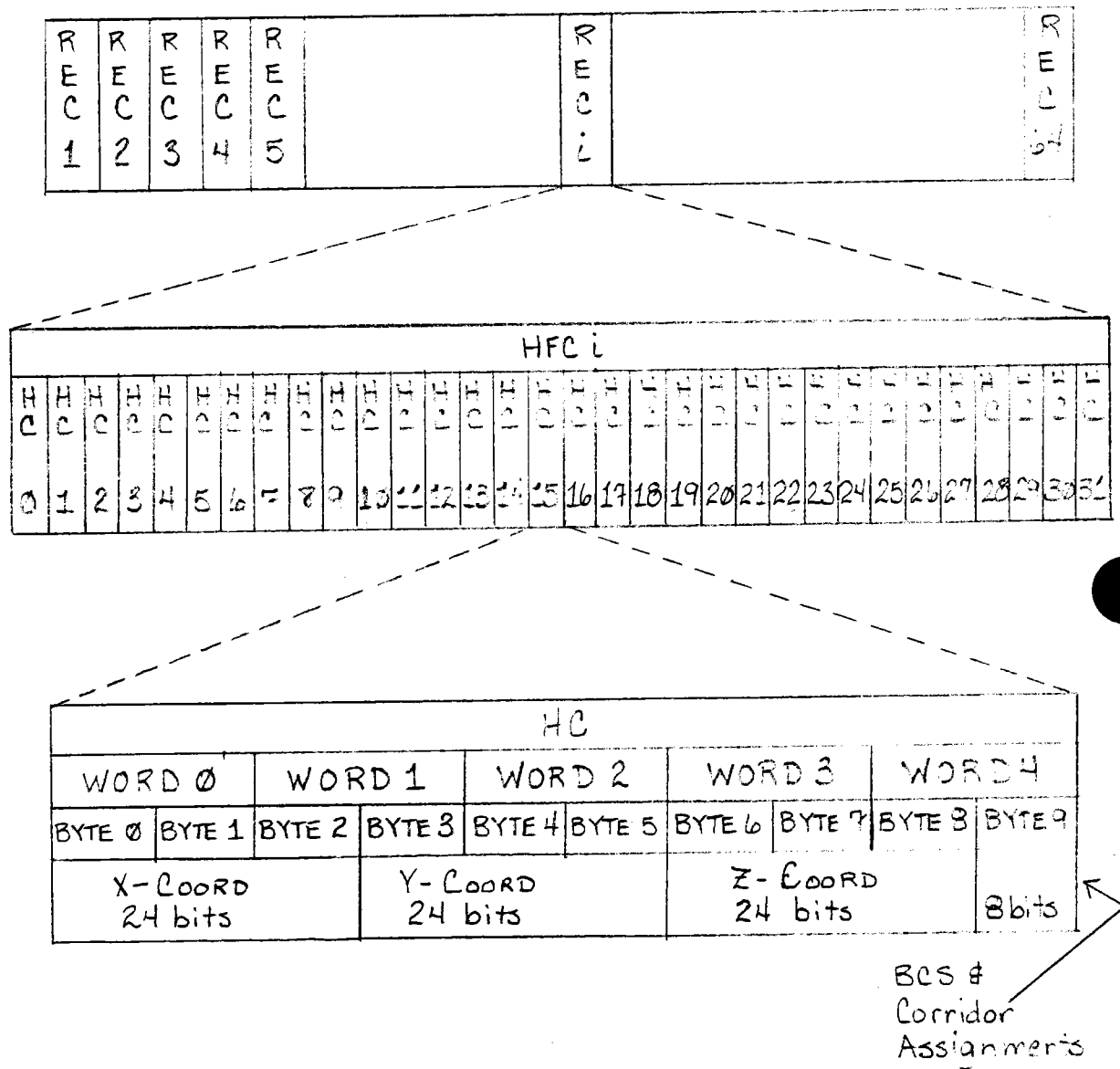


Figure 3.2.2-24 HC Coordinates (HCC)

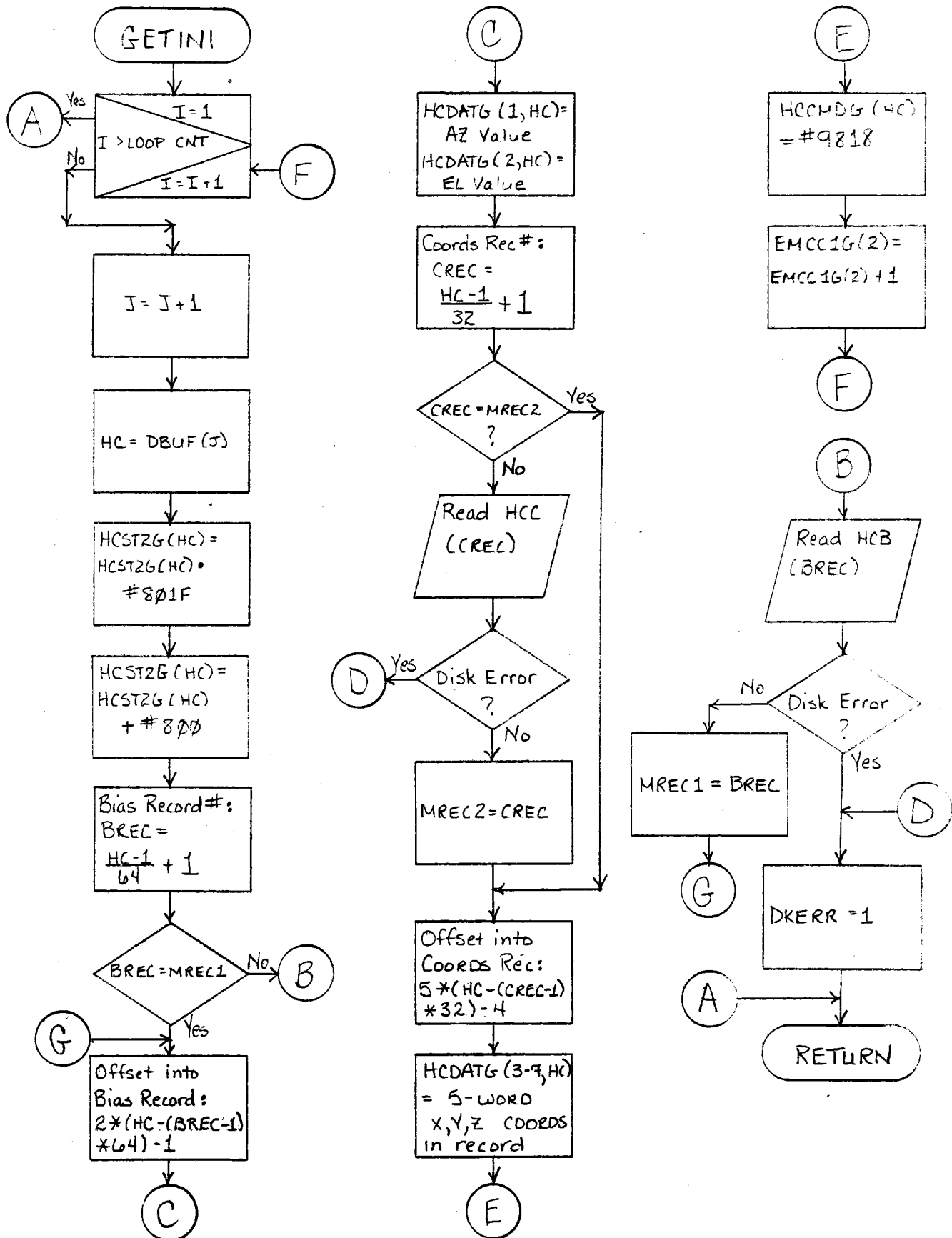


Figure 3.2.2-25 Flowchart - GETINI

3. Increment J by one.
 4. Set HC equal to DBUF(J).
 5. Set the aim-point record number equal to HC and read the aim-point record into memory. Compute the offset to the HC data:

$$\text{HOFSET} = 5 * (\text{OFSET} - 1).$$
 If disk error, set DKERR, and go to step (9).
 6. Update the memory-resident aim-point array, AIMPTG(HC) equals HC aim-point X,Y,Z coordinates in record.
 7. Test the HC to be at, or approaching, its target. If:

$$\text{HCST1G}(\text{HC}) . \text{AND} . \#007\text{C} = \#0020,$$
 go to step (8). Otherwise, go to step (2) for next HC.
 8. Set HCDATG(HC) to the five words of aim-point X,Y,Z coordinates. Set HCMDG(HC) equal to #8020, and go to step (2).
 9. Return to caller.
- e. Error messages and recovery - for disk error, set DKERR and immediately return to caller.

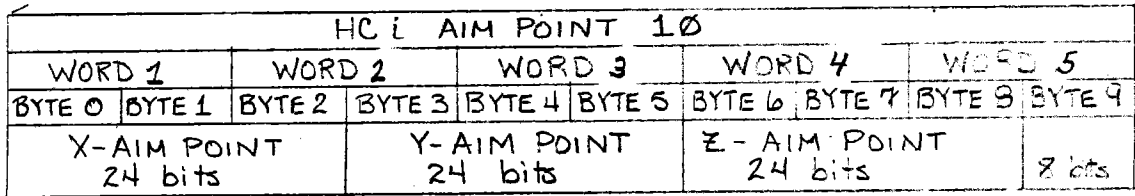
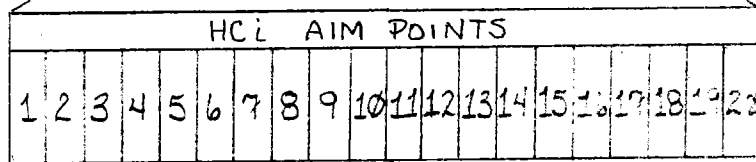
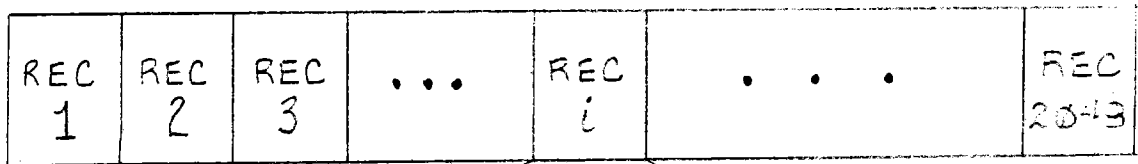
3.2.2.4.1.17.2 Data, Logic and Command Paths

Input data:

- a. Global common:
 - HCCMDG - last commanded status
- b. Local common:
 1. OFSET - offset into aim-point record
 2. Local buffer DBUF for HC numbers
- c. Disk file AIM (see Figure 3.2.2-26)

Output data:

- a. Global common:
 1. HCDATG - HC-data output array
 2. HCCMDG - HC-command output array
 3. AIMPTG - memory-resident aim-point array
- b. Local common:
 1. LOOP - loop count
 2. OFSET - offset into aim-point record
 3. HOFSET - aim-point offset
 4. DKERR - disk-error flag



AIM-POINT
ARRAY #
(1-20)

NOTE: Aim-point data is contained in words 1-100 of each record, while words 123-128 are used by the SAVE and RESTORE commands.

Figure 3.2.2-26 Aim-Point Arrays File (AIM)

3.2.2.4.1.17.3 Internal Data Description

There is no data internal to this submodule.

3.2.2.4.1.17.4 Flowchart

See Figure 3.2.2-27 for the GETAIM flowchart.

3.2.2.4.1.18 Submodule XVIII - GETAL1

3.2.2.4.1.18.1 Description

GETAL1 is responsible for fetching the Alt1stow AZ/EL angles from disk file AL1 for the input command ALT1STOW. HC numbers come from the local buffer DBUF.

- a. Language used - FORTRAN IV
- b. How invoked - called by GET
- c. Constraints and limitations - None
- d. Processing -
 1. Perform DO-UNTIL processing where I equals one to LOOP. If I is greater than LOOP, go to step (8).
 2. Increment J by one.
 3. Set HC equal to DBUF(J).
 4. Compute the record number:
$$BREC = ((HC-1)/64)+1.$$
 5. Test if record number already in memory by checking BREC equals MREC1. If so, go to step (6). If not, read the Alt1stow record into memory and set MREC1 equal to BREC. If disk error, set DKERR, and to to step (8).
 6. Compute the HC's offset into the Alt1stow record.
 7. Insert the Alt1stow AZ/EL angles into words one through two of HCDATG(HC). Set HCCMDG(HC) equal to #8070, and go to step (1).
 8. Return to caller.
- e. Error messages and recovery - for disk error, set DKERR, and immediately return to caller.

3.2.2.4.1.18.2 Data, Logic and Command Paths

Input data:

- a. BREC - fetched record number
- b. Local file DBUF
- c. Disk file AL1 (see Figure 3.2.2-28)

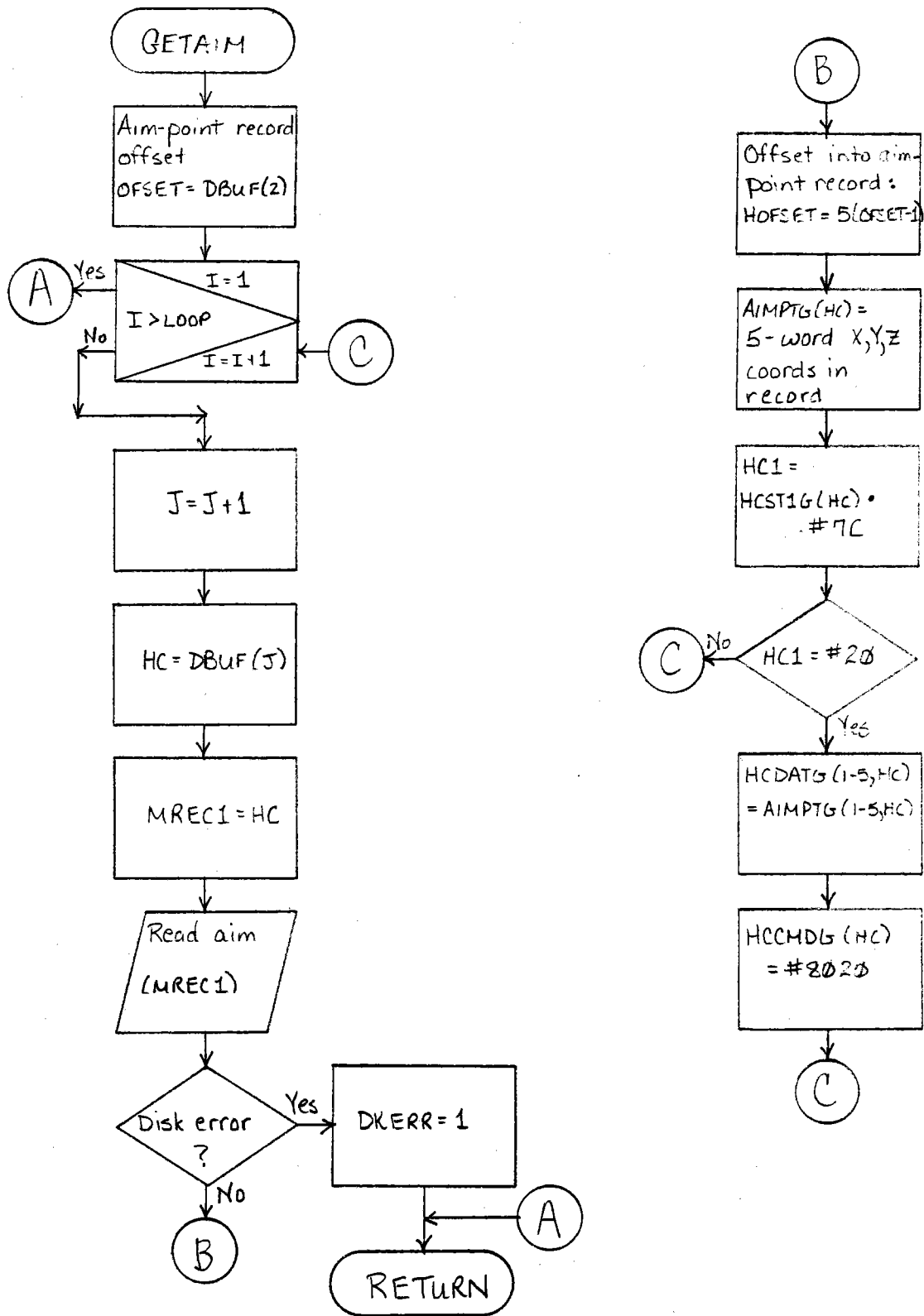


Figure 3.2.2-27 Flowchart - GETAIM

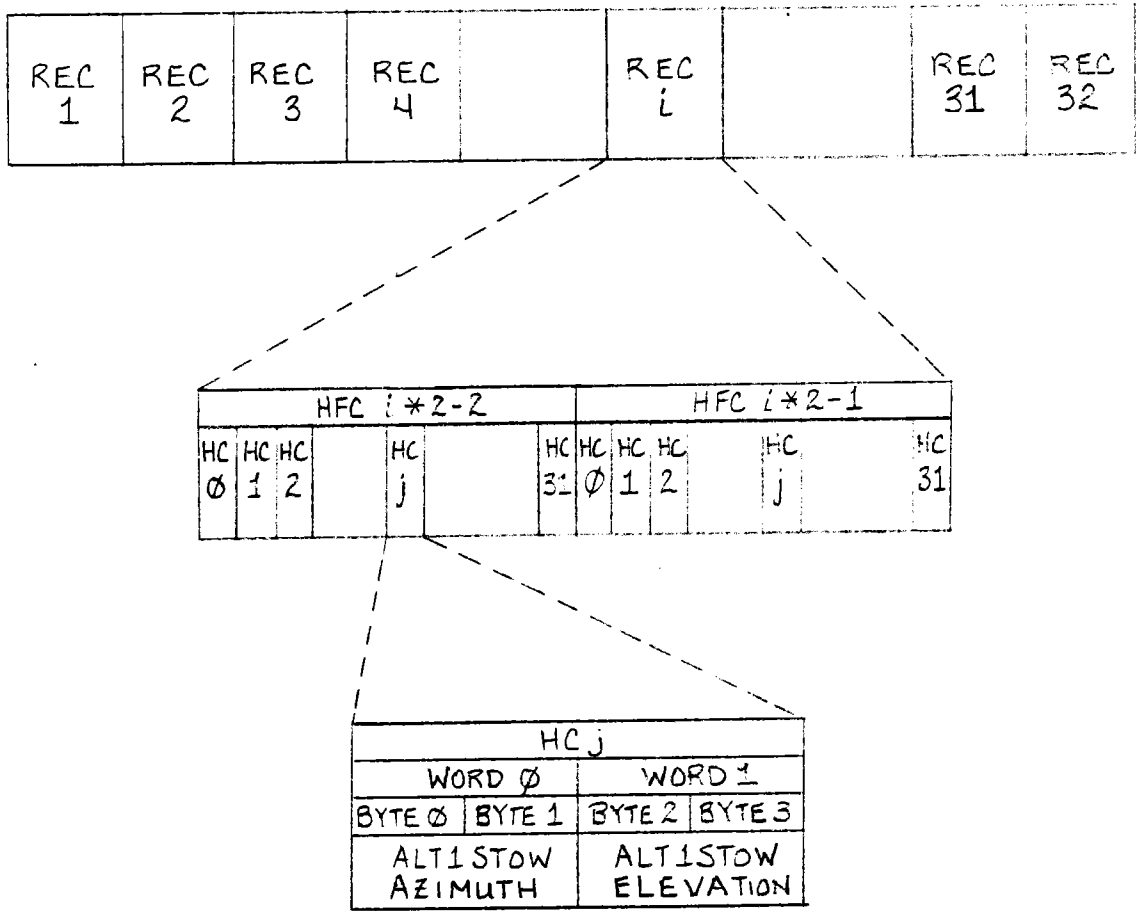


Figure 3.2.2-28 Format and Structure of HC ALT1STOW Angles Disk File

Output data:

a. Global common:

1. HCDATG - HC-data output array
2. HCCMDG - HC-command output array

b. Local common:

1. LOOP - loop count
2. MREC1 - record number in memory
3. BREC - fetched record number

3.2.2.4.1.18.3 Internal Data Description

There is no data internal to this submodule.

3.2.2.4.1.18.4 Flowchart

See Figure 3.2.2-29 for the GETAL1 flowchart.

3.2.2.4.1.19 Submodule XIX - GETAL2

3.2.2.4.1.19.1 Description

GETAL2 is responsible for fetching the Alt2stow elevation angle from the disk file AL2 for the input command ALT2STOW. HC numbers come from the local buffer DBUF. The azimuth angle for the output command comes from the AZIMG array.

a. Language used - FORTRAN IV

b. How invoked - called by GET

c. Constraints and limitations - None

d. Processing -

1. Perform DO-UNTIL processing where I equals one to LOOP. If I is greater than LOOP, go to step (8).
2. Increment J by one.
3. Set HC equal to DBUF(J).
4. Compute the record number:
$$BREC = ((HC-1)/64)+1.$$
5. Test if record is in memory by checking BREC equals MREC1. If so, go to step (6). If not, read the Alt2stow record into memory and set MREC1 equal to BREC. If disk error, set DKERR and go to step (8).
6. Compute HC's offset into the record.
7. Insert the Alt2stow elevation into word 2 of HCDATG(HC) and insert AZIMG(HC) into word 1 of HCDATG(HC). Set HCCMDG(HC) equal to #8074, and go to step (1).

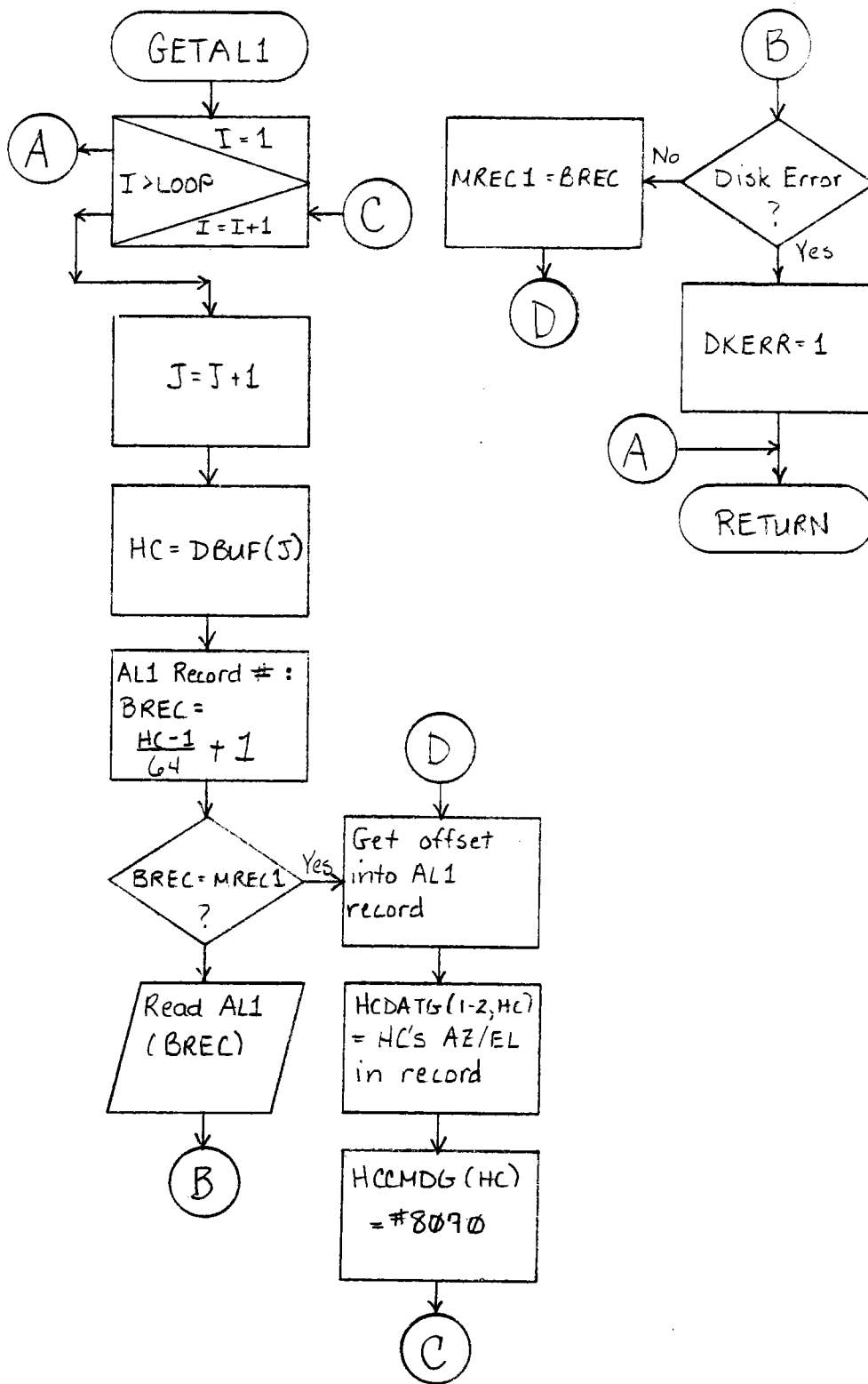


Figure 3.2.2-29 Flowchart - GETAL1

8. Return to caller.

- e. Error messages and recovery - for disk error, set DKERR and immediately return to caller.

3.2.2.4.1.19.2 Data, Logic and Command Paths

Input data:

- a. Global common:
AZIMG - last reported azimuth angle for HC
- b. Local common:
1. Local buffer DBUF
2. BREC - fetched record number
- c. Disk file AL2 (see Figure 3.2.2-30)

Output data:

- a. Global common:
1. HCDATG - HC-data output array
2. HCCMDG - HC-command output array
- b. Local common:
1. LOOP - loop count
2. MREC1 - record number in memory
3. BREC - fetched record number

3.2.2.4.1.19.3 Internal Data Description

There is no data internal to this submodule.

3.2.2.4.1.19.4 Flowchart

See Figure 3.2.2-31 for the GETAL2 flowchart.

3.2.2.4.1.20 Submodule XX - GETWSH

3.2.2.4.1.20.1 Description

GETWSH is responsible for fetching the Wash AZ/EL angles from the disk file WSH for the input command WASH. HC numbers come from the local buffer DBUF.

- a. Language used - FORTRAN IV
- b. How invoked - called by GET
- c. Constraints and limitations - None
- d. Processing -
1. Perform DO-UNTIL processing where I equals one to LOOP. If I is greater than LOOP, go to step (8).
 2. Increment J by one.

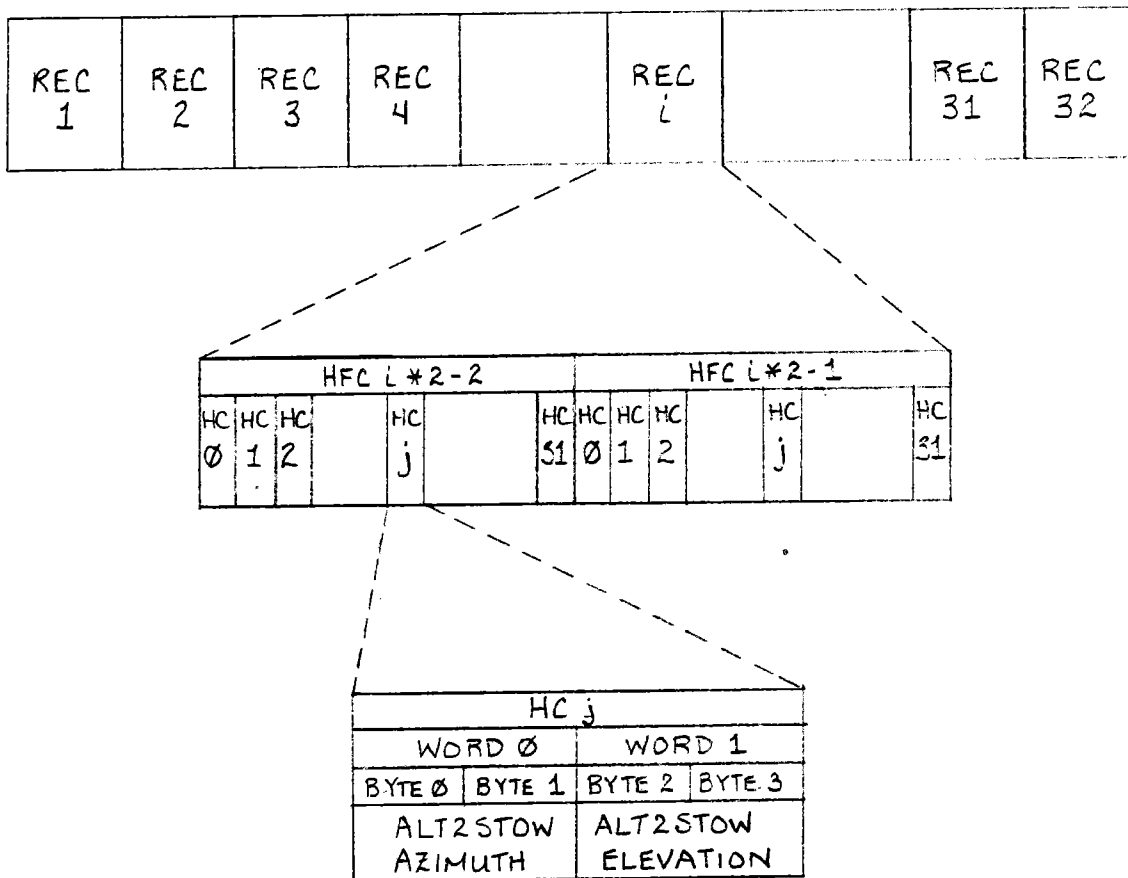


Figure 3.2.2-30 Format and Structure of ALT2STOW Angle Disk File

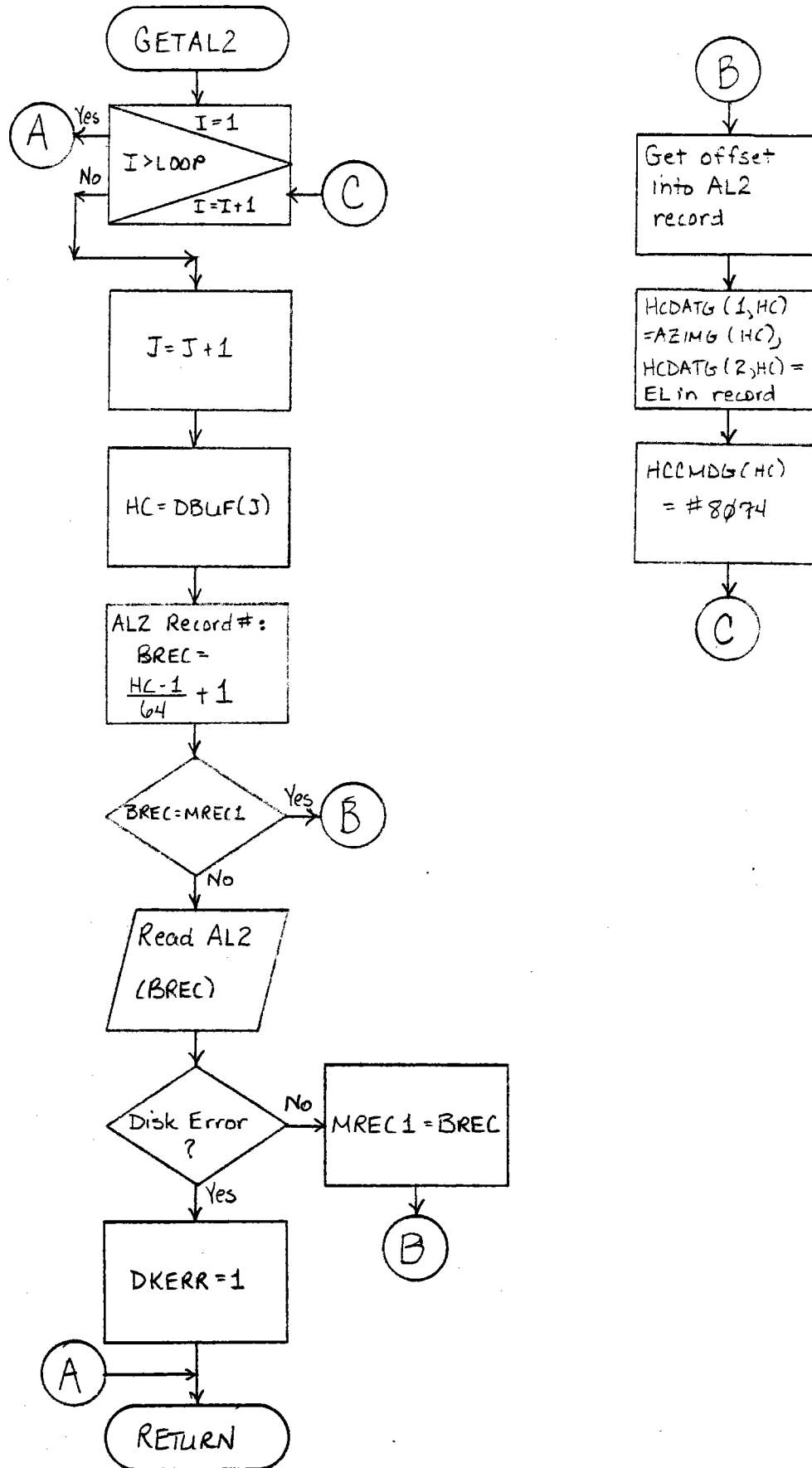


Figure 3.2.2-31 Flowchart - GETAL2

3. Set HC equal to DBUF(J).
 4. Compute the record number:

$$BREC = ((HC-1)/64)+1.$$
 5. Test if record number already in memory by checking BREC equals MREC1. If so, go to step (6). If not, read the Wash record into memory and set MREC1 equal to BREC. If disk error, set DKERR and go to step (8).
 6. Compute HC's offset into the Wash record.
 7. Insert the Wash AZ/EL into words one through two of HCDATG(HC). Set HCCMDG(HC) equal to #8068 and go to step (1).
 8. Return to caller.
- e. Error messages and recovery - for disk error, set DKERR and immediately return to caller.

3.2.2.4.1.20.2 Data, Logic and Command Paths

Input data:

- a. BREC - fetched record number
- b. Local buffer DBUF
- d. Disk file WSH (see Figure 3.2.2-32)

Output data:

- a. Global common:
 1. HCDATG - HC-data output array
 2. HCCMDG - HC-command output array
- b. Local common:
 1. LOOP - loop count
 2. MREC1 - record number in memory
 3. BREC - fetched record number

3.2.2.4.1.20.3 Internal Data Description

There is no data internal to this submodule.

3.2.2.4.1.20.4 Flowchart

See Figure 3.2.2-33 for the GETWSH flowchart.

3.2.2.4.1.21 Submodule XXI - GETSTO

3.2.2.4.1.21.1 Description

GETSTO is responsible for fetching the Stow AZ/EL angles from disk file STO for the input commands STOW, STHIWIND, or ESTOW.

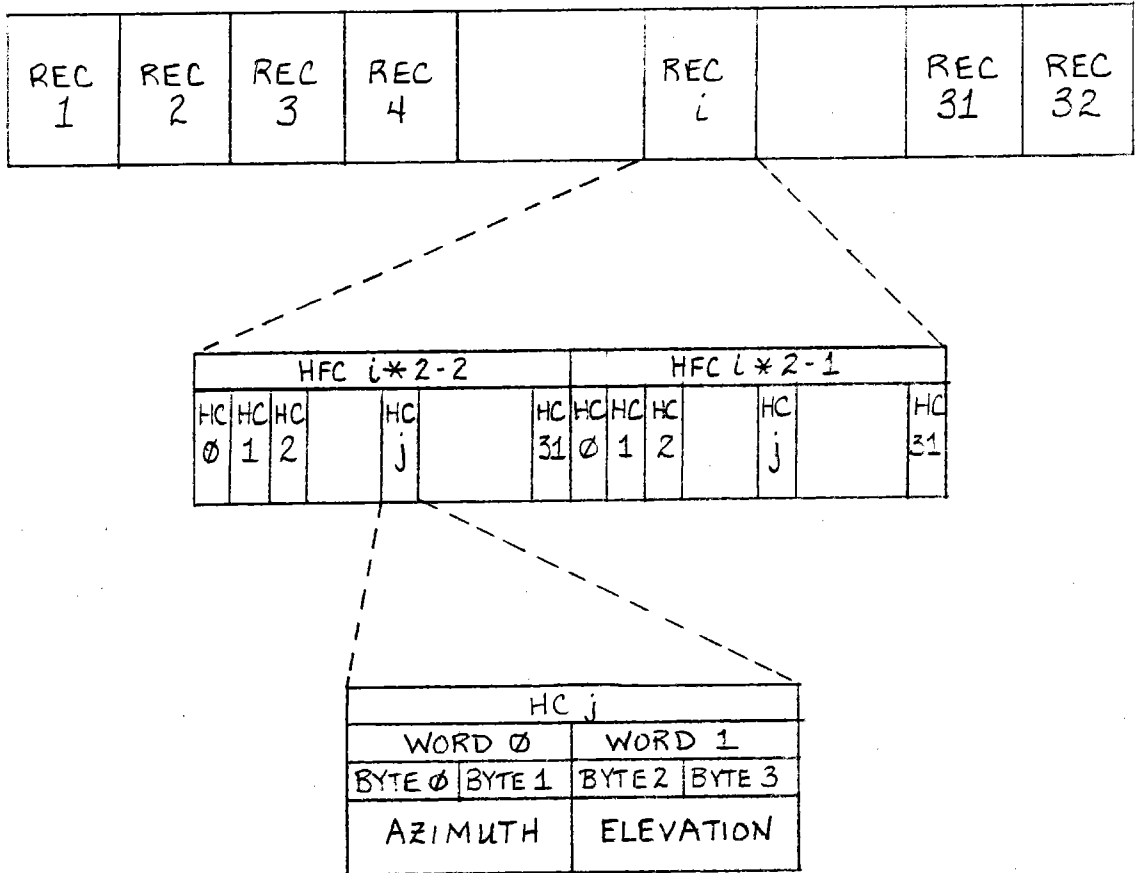


Figure 3.2.2-32 Format and Structure of Wash Angles File (WSH)

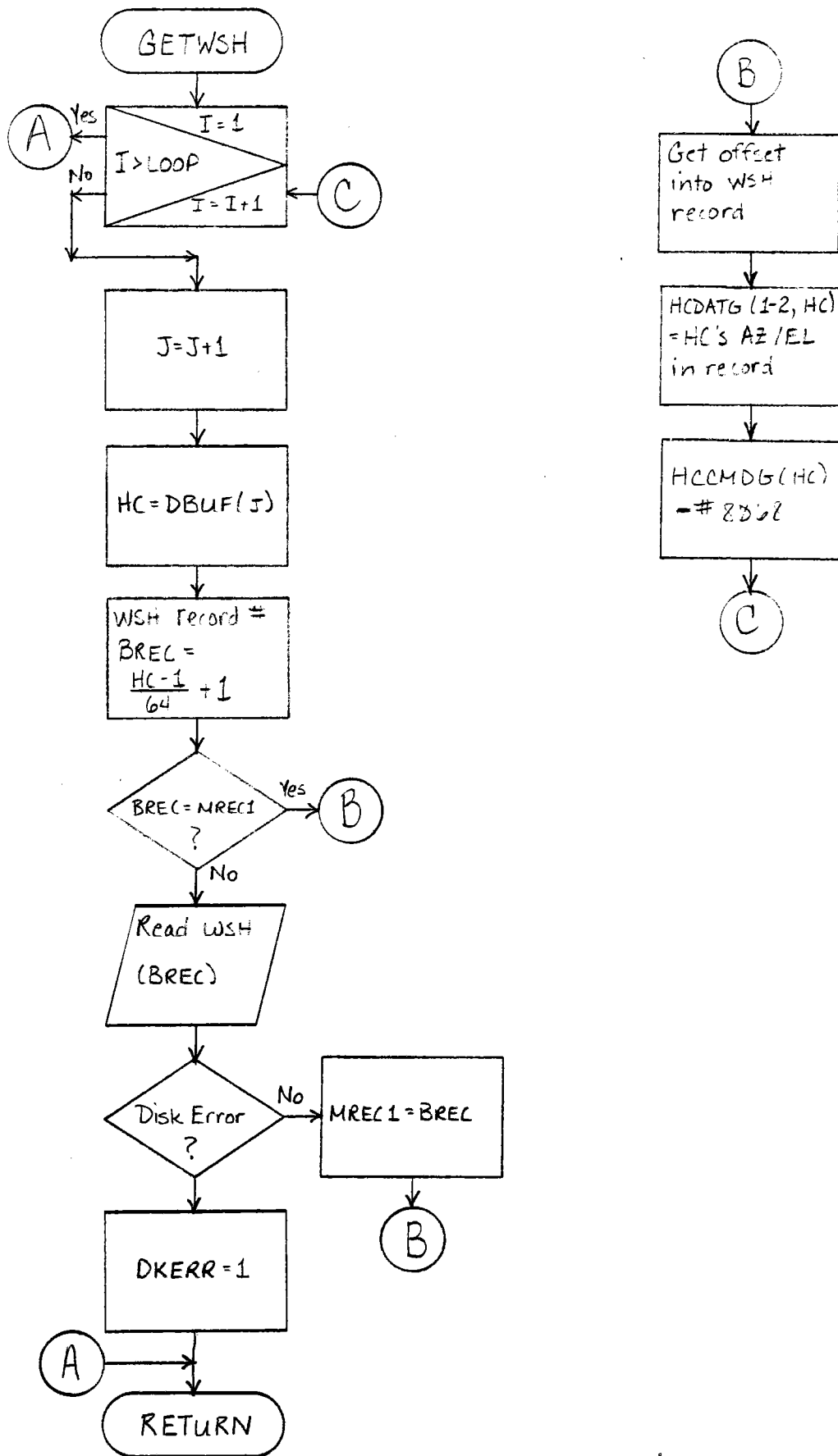


Figure 3.2.2-33 Flowchart - GETWSH

HC numbers come from the local buffer DBUF.

- a. Language used - FORTRAN IV
- b. How invoked - called by GET
- c. Constraints and limitations - None
- d. Processing -
 1. Zero, the critical-command bit value (BIT3).
 2. If command is STHIWIND, set BIT3 equal to #1000.
 3. Perform DO-UNTIL processing where I equals one to LOOP. If I is greater than LOOP, go to step (10).
 4. Increment J by one.
 5. Set HC equal to DBUF(J).
 6. Compute the record number:
$$\text{BREC} = ((\text{HC}-1)/64)+1.$$
 7. Test if the record number is in memory by checking BREC equals MREC1. If so, go to step (8). If not, read the Stow record into memory and set MREC1 equal to BREC. If disk error, set DKERR and go to step (10).
 8. Compute the HC's offset into the Stow record.
 9. Insert the Stow AZ/EL into words one through two of HCDATG(HC). Set HCCMDG(HC) equal to #8060 plus BIT3. If DBUF(3) equals nine, reset HC-in-Stow-sequence:
$$\text{HCST3G}(\text{HC}) = \text{HCST3G}(\text{HC}) . \text{AND} . \# \text{FEFF}.$$

Go to step (3).
 10. Return to caller.
- e. Error messages and recovery - for disk error, set DKERR and immediately return to caller.

3.2.2.4.1.21.2 Data, Logic and Command Paths

Input data:

- a. Local buffer DBUF
- b. BREC - fetched record number
- c. Disk file STO (see Figure 3.2.2-34)

Output data:

- a. Global common:
 1. HCDATG - HC-data output array
 2. HCCMDG - HC-command output array
- b. Local common:
 1. LOOP - loop count
 2. MREC1 - record number in memory
 3. BREC - fetched record number

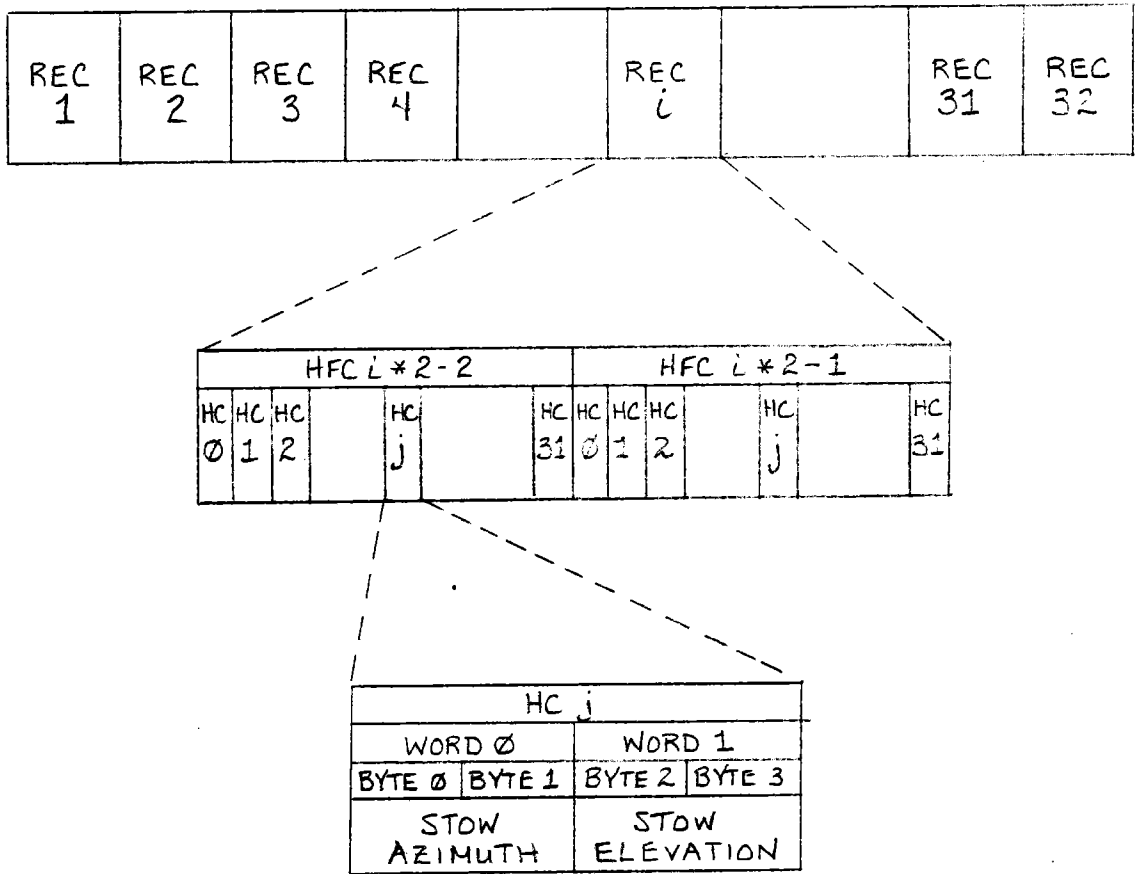


Figure 3.2.2-34 Format and Structure of HC Stow Angles Disk File

3.2.2.4.1.21.3 Internal Data Description

Critical command bit value:

- BIT3 = 0: command is not STHIWIND
- = #1000: command is critical (STHIWIND)

3.2.2.4.1.21.4 Flowchart

See Figure 3.2.2-35 for the GETSTO flowchart.

3.2.2.4.1.22 Submodule XXII - GETSAR

3.2.2.4.1.22.1 Description

GETSAR is responsible for two command actions:

- a. Write the field's Tracking configuration (those HCs which are in the Standby and Track orientations) to disk file SAV; and
- b. Fetch the saved Tracking configuration from disk file SAV and try to restore this configuration to the field.

The local buffer DBUF contains the HCs which are in either Standby or Track orientation for the SAVE command. RESTORE processing does not get HC numbers in the DBUF buffer and must read every record of file SAV and input the current HC status from the HCSTIG array. Alarms bits are set for those HCs which are unable to return to their saved orientation.

- a. Language used - FORTRAN IV
- b. How invoked - called by GET
- c. Constraints and limitations - None
- d. Processing -

- 1. If the command is not SAVE, go to step (12).

SAVE processing:

- 2. Perform DO-UNTIL processing where I equals one to LOOP. If I is greater than LOOP, go to step (4).
- 3. Insert the HC number in DBUF(I) into the sort buffer SOBUF(I) in ascending order, and go to step (2).
- 4. Set the SOBUF buffer index J equal to 1.
- 5. Perform DO-UNTIL processing where I equals one to 2048. If I is greater than 2048, go to step (11).
- 6. Set record number MRECl equal to I. If J is greater than LOOP, set the SAV-file HC number, SAVHC equal to zero, and go to step (7). Otherwise, set the SAV-file HC number, SAVHC equal to SOBUF(J).

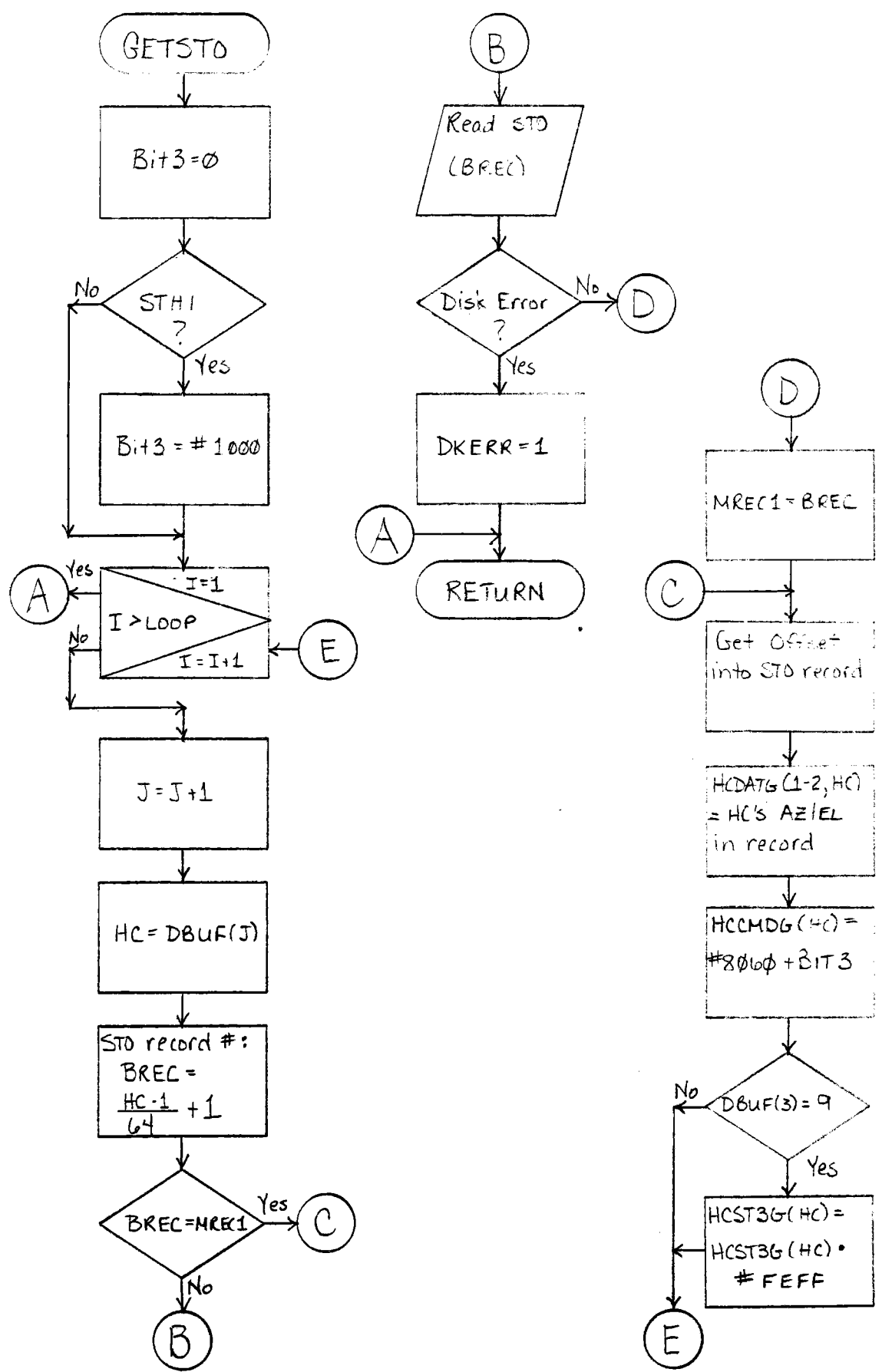


Figure 3.2.2-35 Flowchart - GETSTO

7. Clear the SAV-file record buffer: SOBUF(1-6) equals zero.
8. If I is not equal to SAVHC, go to step (9) to write a blank record in the SAV file for this HC number. Otherwise, increment J by one and get the HC status:

HC1 = HCST1G(SAVHC).AND.#7C.

If HC1 equals #30 (Standby), set SBUF(1) equal to minus one, and go to step (9). Otherwise, set SBUF(1) equal to one (save orientation is Track) and place the HC's aim-point in the record, SBUF(2-6) equals ADMPTG(1-5,SAVHC).

9. Write a record to disk file SAV using MREC1 as the record number. If no disk error occurs, go to step (5).
10. For disk errors, set disk-error flag DKERR equal to one.
11. Return to caller.

RESTORE processing:

12. Perform DO-UNTIL processing where I equals one to 2048. If I is greater than 2048, go to step (11).
13. Set record number MREC1 equal to I. Read a record from the disk file SAV using MREC1 as record number. If disk error occurs, go to step (10).
14. Get the HC status:

HC1 = HCST1G(I).AND.#7C.

If the first word of the SAV record, SBUF(1) equals zero, this HC was not in Standby or Track at save time; go to step (12). Otherwise, if SBUF(1) is greater than zero (save orientation was Track), go to step (19). Otherwise, SBUF(1) is less than zero (save orientation was Standby).

15. If HC1 equals #30 (Standby), HC is in proper orientation and requires no movement; go to step (12). Otherwise, if HC1 is not equal to #20, go to step (18) to set HC's alarm bit. If HC1 equals #20 (at or approaching Track), it can be commanded back to its save orientation (Standby).
16. Get the HC's corridor assignment by:

L=HCST3G(I).AND.#FOOO.

Right-justify by shifting right 12 bits.

17. Set HCDATG(1-5,I) equal to CORRGG(1-5,I) to pick

up the five-word CULP X,Y,Z coordinates. Set HCCMDG(I) equal to #8030, and go to step (12).

Alarms processor for RESTORE, one step only:

18. Set the RESTORE alarms bit (bit 6) in HCST1G(I) to tell Alarms-dection task ALM that this HC was not in the proper orientation to return to its save orientation. Go to step (12).
 19. If HC1 equals #20 (at or approaching Track), HC is in proper orientation but could be assigned a different aim-point than the one saved at save time; go to step (21). Otherwise, if HC1 is not equal to #30, go to step (18) to set HC's alarm bit. If HC1 equals #30 (Standby), fall through to step (20).
 20. Set HCDATG(1-5,I) equal to SBUF(2-6) to pick up the five-word save-time aim-point X,Y,Z coordinates. Set HCCMDG(I) equal to #8020. Update the memory-resident aim-point array by setting AIMPTG(1-5,I) equal to SBUF(2-6). Go to step (12).
 21. If the save-time aim-point located in SBUF(2-6) is equal to the latest memory-resident aim-point, AIMPTG(1-5,I), no adjustment movement to a new aim-point is required; go to step (12). Otherwise, the save-time aim-point must be sent to the HC in the field; go to step (20).
- e. Error messages and recovery - for disk errors, set DKERR and immediately return to caller.

3.2.2.4.1.22.2. Data, Logic and Command Paths

Input data:

- a. Global common:
 1. HCST1G - HC status
 2. AIMPTG - memory-resident aim-point array
 3. CORRCG - corridor coordinates
 4. HCST3G - HC corridor assignment
- b. Local common:
 1. DBUF - local buffer of HC numbers
 2. SBUF - 6-word SAV record buffer

Output data:

- a. Global common:
 1. HCDATG - HC-data output array
 2. HCCMDG - HC-command output array
 3. HCST2G - HC-derived status, restore-alarm bit

4. AIMPTG - memory-resident aim-point array

b. Local common:

1. MRECl - file record number
2. HC - involved HC number
3. HCl - current HC field status
4. SBUF - 6-word SAV record buffer

c. Disk file SAV (see Figure 3.2.2-36)

3.2.2.4.1.22.3 Internal Data Description

SAV record buffer:

SBUF - 6-word record buffer for disk file SAV processing

3.2.2.4.1.22.4 Flowchart

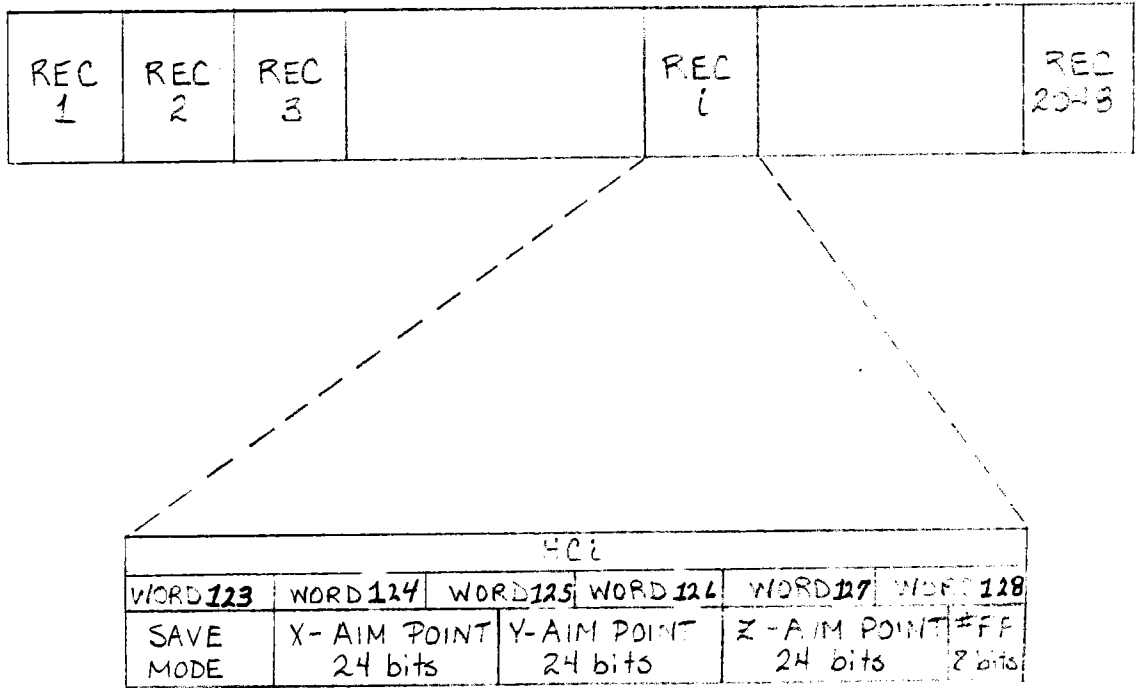
See Figure 3.2.2-37 for the GETSAR flowchart.

3.2.2.4.1.23 Submodule XXIII - SEQ

3.2.2.4.1.23.1 Description

SEQ is responsible for overall control of sequence processing. It determines the correct processing path for command phases 4, 5, 10, 20, and 30. The dequeued sequence command packet follows the format of Table 3.2.2-V.

- a. Language used - FORTRAN IV
- b. How invoked - activated via Enque by tasks CMD or GET or by timer expiration connected and thawed by this task.
- c. Constraints and limitations - None
- d. Processing -
 1. Call TIMDT4 to acquire current time: minutes since midnight and ticks in minutes.
 2. Call DEQU to attempt to acquire a sequence command packet. If the message length returned by DEQU is 0 (queue empty), activation was activated by a timer expiration; go to Step (9). If the length is nonzero, the sequence command packet is located in buffer SBUF.



Where Save Mode = \emptyset : other
 > \emptyset : HC was at Track and its Aim point is in words 1-5
 < \emptyset : HC was at Stand by

NOTE: Aim-point data is contained in words 1-100 of each record, while words 123-128 are used by the SAVE and RESTORE commands.

Figure 3.2.2-36 Tracking Configuration Save File (AIM)

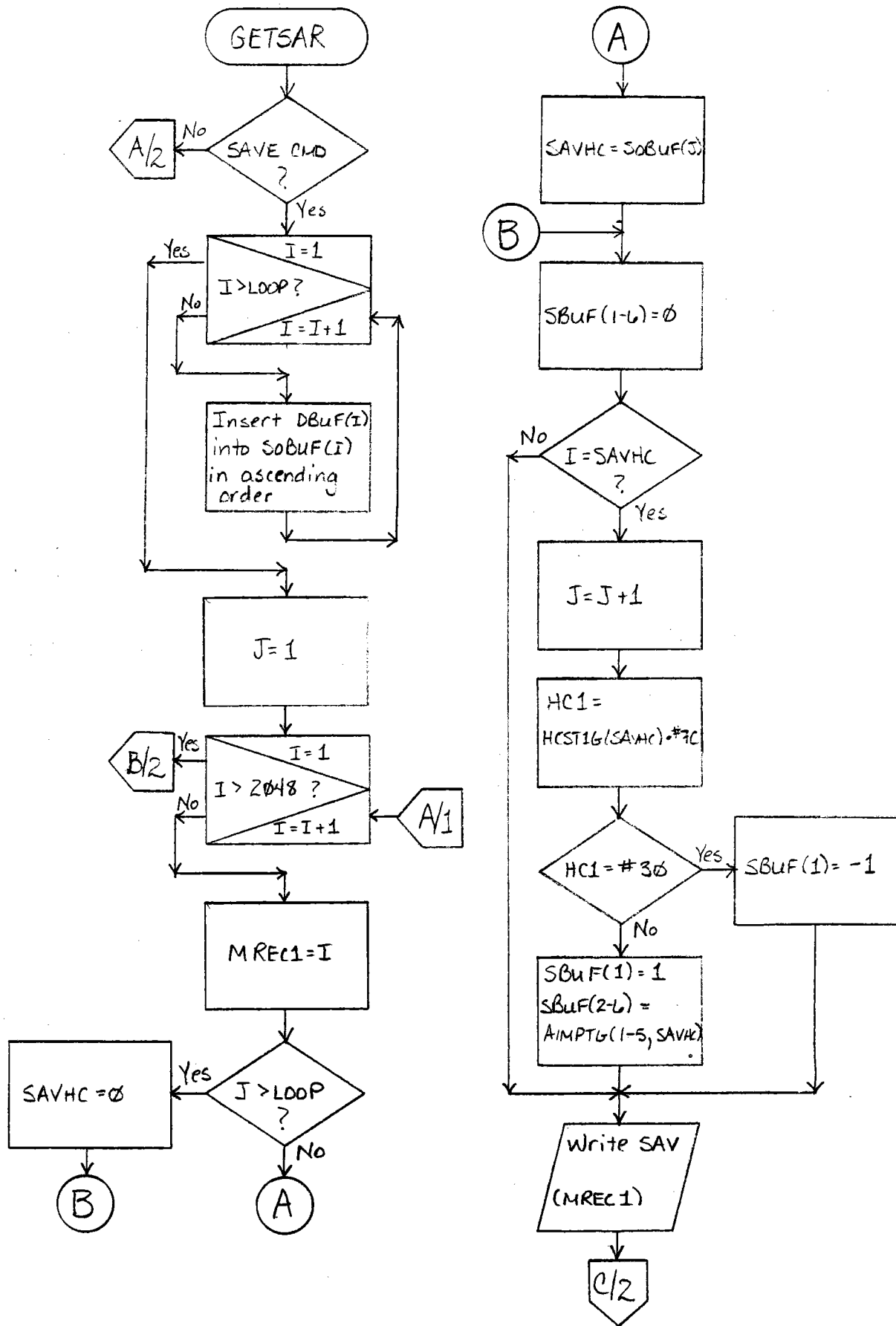


Figure 3.2.2-37 Flowchart - GETSAR

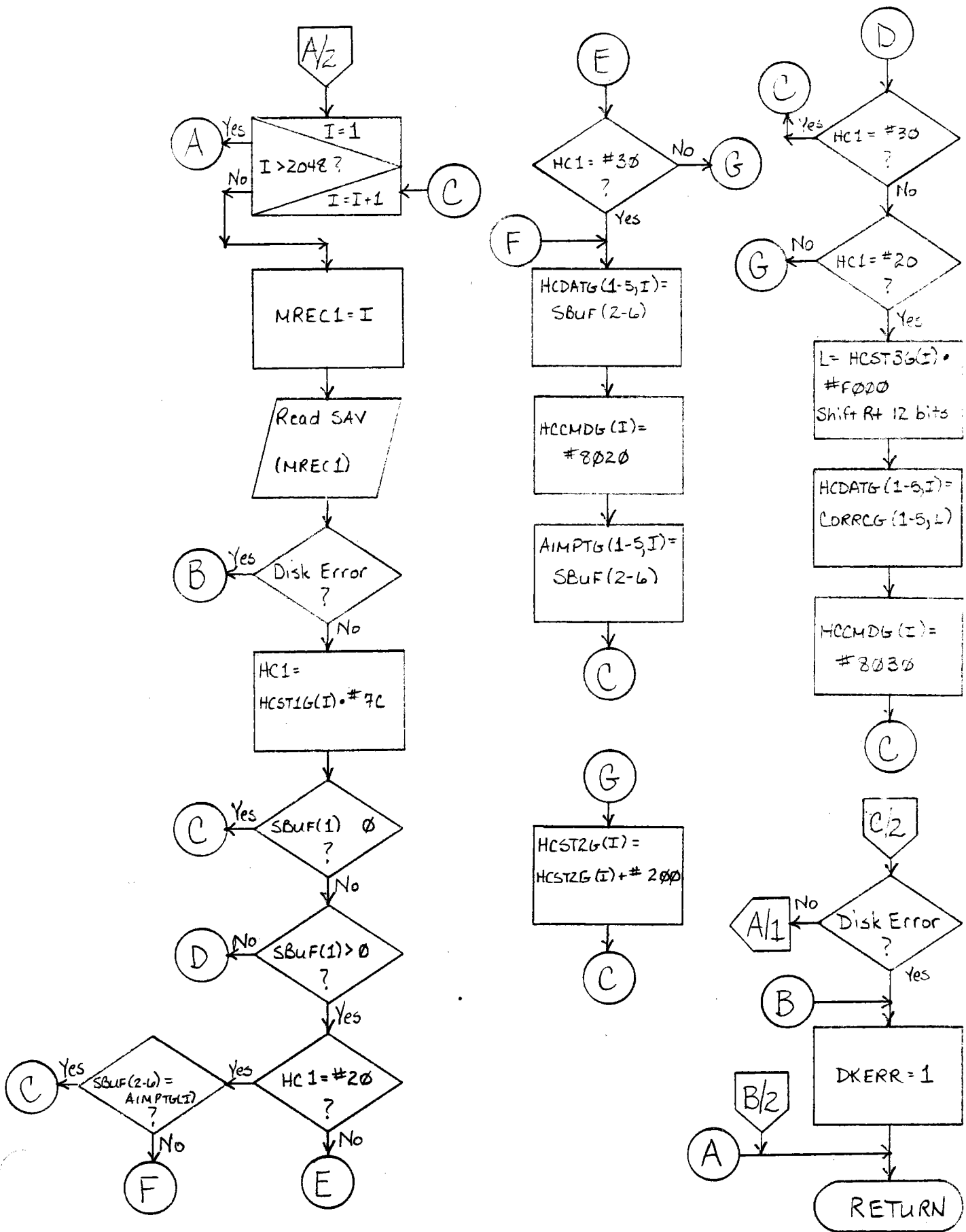


Figure 3.2.2-37 Flowchart -GETSAR (continued)

Decode the dequeue message:

3. If the command phase in SQBUF(2) is not four, go to Step (4). At least some of the heliostats involved in the sequence command are moving towards either their CLLPs or CULPs. Zero the relink flag (RELINK) and call submodule SEQADD to put sequence in the active-sequence list (ACTLIS). Upon return, go to Step (12).
4. If the command phase is five, go to Step (7). If the command phase is not ten, go to Step (5). Call submodule SEQDEL to delete sequence. Upon return, go to Step (8).
5. The activation of this task was caused by an enqueue by either the field power fail task (PLH) or the HAC failover task (SWI). If the command phase is not 20 (field power loss), go to Step (6). Set the delete level = -2 (delete all sequences) and call submodule SEQDEL. Unconnect timer if one was assigned to this task and go to Step (13).
6. If the command phase is not 30 (HAC failover), go to Step (8). Set the relink flag (RELINK) and go to Step (12) to set up timer.
7. Add the sequence to the wait-sequence list (WAITSQ). See Figure 3.2.2-38 for format. If list is full, call MSSG4 to print following message on system console:

"CANNOT ADD SEQUENCE TO WAIT LIST"

and go to Step (12).

8. Call submodule SEQCCK to add and/or search the wait-sequence list and determine whether any sequence has all of its required corridors free. If not, go to Step (12). Otherwise, zero the relink flag and call submodule SEQADD to put sequence into the active-sequence list (ACTLIS). Upon return, call submodule SEQCOR to move the heliostats into their respective corridor(s). Upon return, go to Step (12).

Activation caused by timer expiration:

9. Determine visit sequence by examining the time-link pointer (DUMTLP). Call submodule SEQGAT to gather the sequence's heliostats

WAIT SEQ. CELL #	WAIT SEQUENCE CELLS
1	(See Table 3.2.2-V for format)
.	
.	
.	
.	
15	

NOTES: Entries are up to 23 words long.

First word = 0 for unused entries.

All data comes from the deque sequence command packet buffer SQBUF.

Figure 3.2.2-38 Wait-Sequence List (WAITSQ)

arriving at either their CLLPs or CULPs. Upon return, go to Step (10) if gather process done. Otherwise, check if the sequence was completely robbed (the commands STHIWIND, LOAD, or HOLD may rob some or all heliostats from an established sequence). If so, go to Step (4) to delete sequence. Otherwise, set the relink flag and call submodule SEQRLK to relink visit time-links in the active-sequence list. Upon return, go to Step (12).

10. When the gather process is done, check the command phase to be eight (corridor gather). If so, go to Step (11). If command phase is four, all of the involved HCs are either at their CULPs or CLLPs. The sequence must be taken out of the active-sequence list and added to the corridor-wait list to check if all required corridors are free. Therefore, transfer sequence data into SQBUF, set the delete level = 0 (delete sequence only from active-sequence list) and call SEQDEL. Upon return, go to Step (7).
11. If command phase is eight, call submodule SEQBPT to command the involved HCs to track either their CULPs or CLLPs. Upon return, check the command phase to be ten and if so, go to Step (4) to delete sequence. If updated command phase is nine, call ENQU to send a disk data command packet to task GET in order to send HCs to their final destination. Go to Step (8).

Exit processing:

12. If relink flag is zero, go to Step (13). Fetch the next visit time using the active-sequence time-link pointer (DUMTLP). If zero, call UNCON4 to unconnect any timer allocated to this task. For a valid revisit time, call CONNCT to obtain a timer. Call THAW4 to unfreeze the timer.
 13. Call RETIRE to exit and save timer resource.
- e. Error messages and recovery - if wait-sequence list has no unoccupied entries for a new wait sequence, output text message to system console and go to exit processing.

3.2.2.4.1.23.2 Data, Logic and Command Paths

Input data:

Local common:

- a. SQBUF - deque buffer for sequence packet
- b. WAITSQ - wait sequence list (see Figure 3.2.2-38)
- c. ACTLIS - dummy-cell time link in active-sequence list

Output data:

Local Common:

- a. RELINK - relink ACTLIS flag
- b. STUB - future-definition event flag
- c. WAITSQ - wait-sequence list

3.2.2.4.1.23.3 Internal Data Description

There is no data internal to this submodule.

3.2.2.4.1.23.4 Flowchart

See Figure 3.2.2-39 for the SEQ flowchart.

3.2.2.4.1.24 Submodule XXIV - SEQGAT

3.2.2.4.1.24.1 Description

SEQGAT is responsible for gathering HCs at either their CLLPs or CULPs for the sequence being visited. As HCs report position compare, they are deleted from the gather buffer and inserted into a leased buffer. When the gather buffer is empty or the maximum time is reached, the gather process is done. Any HC not responding by the maximum time will be set offline and out of the sequence by setting the HC's timeout bit in HCST2G(HC).

- a. Language used - FORTRAN IV
- b. How invoked - called by SEQ for command phases four and eight
- c. Constraints and limitations - None
- d. Processing -
 1. Set gather loop count (LOOP) equal to the gather buffer size.
 2. If the sequences-gather lease flag is set, go to Step (3). Otherwise, a buffer must be leased. Call routine LEASE to obtain an area in which to put the HCs reporting position compare. Save the lease-area address in the sequence's ACTLIS entry. Zero the lease buffer size and set the gather-lease flag.
 3. Determine the lease buffer pointer K equals lease size plus one.
 4. Perform DO-UNTIL processing where I equals one to LOOP. If I is greater than LOOP, go to Step (9).

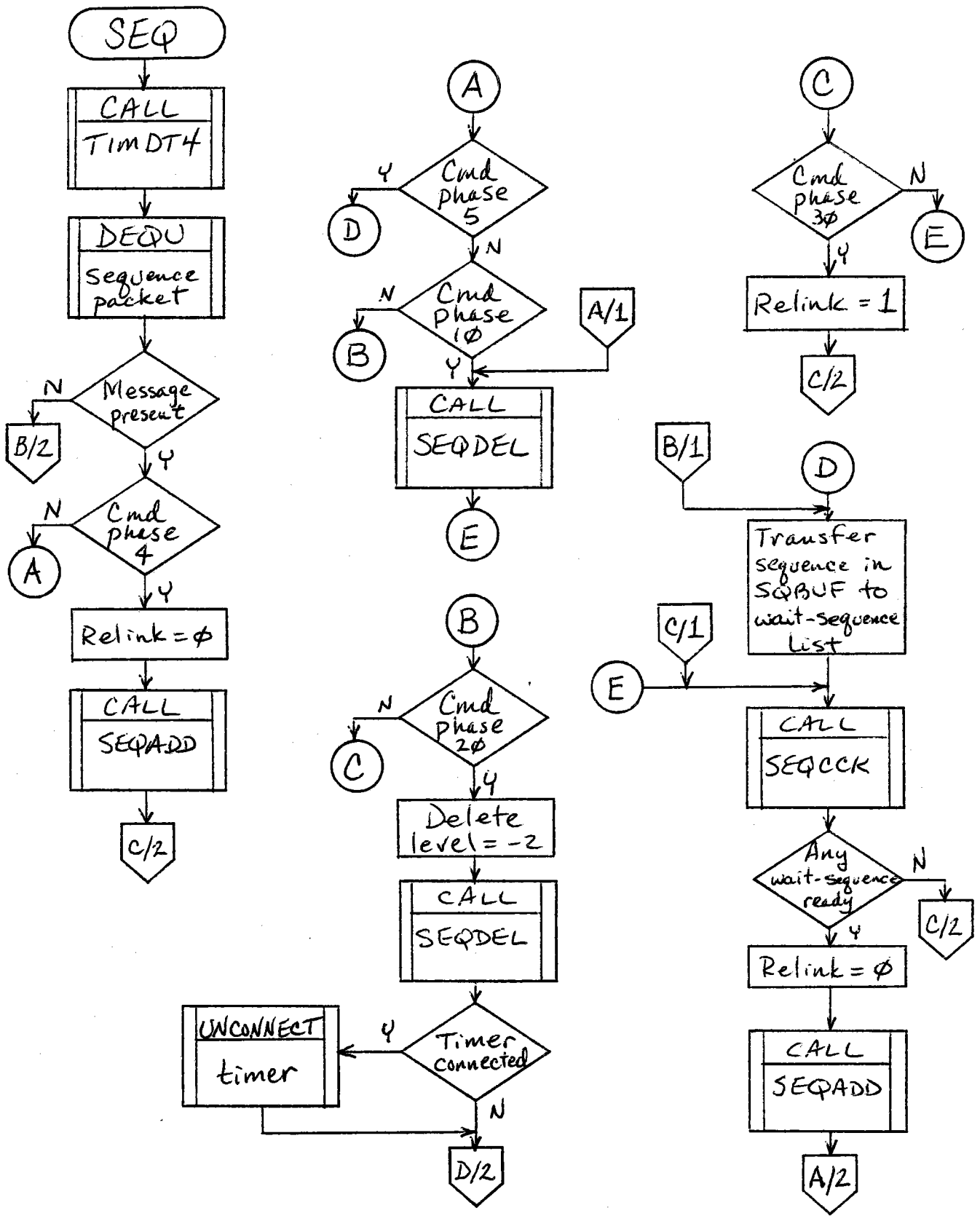


Figure 3.2.2-39 Flowchart - SEQ

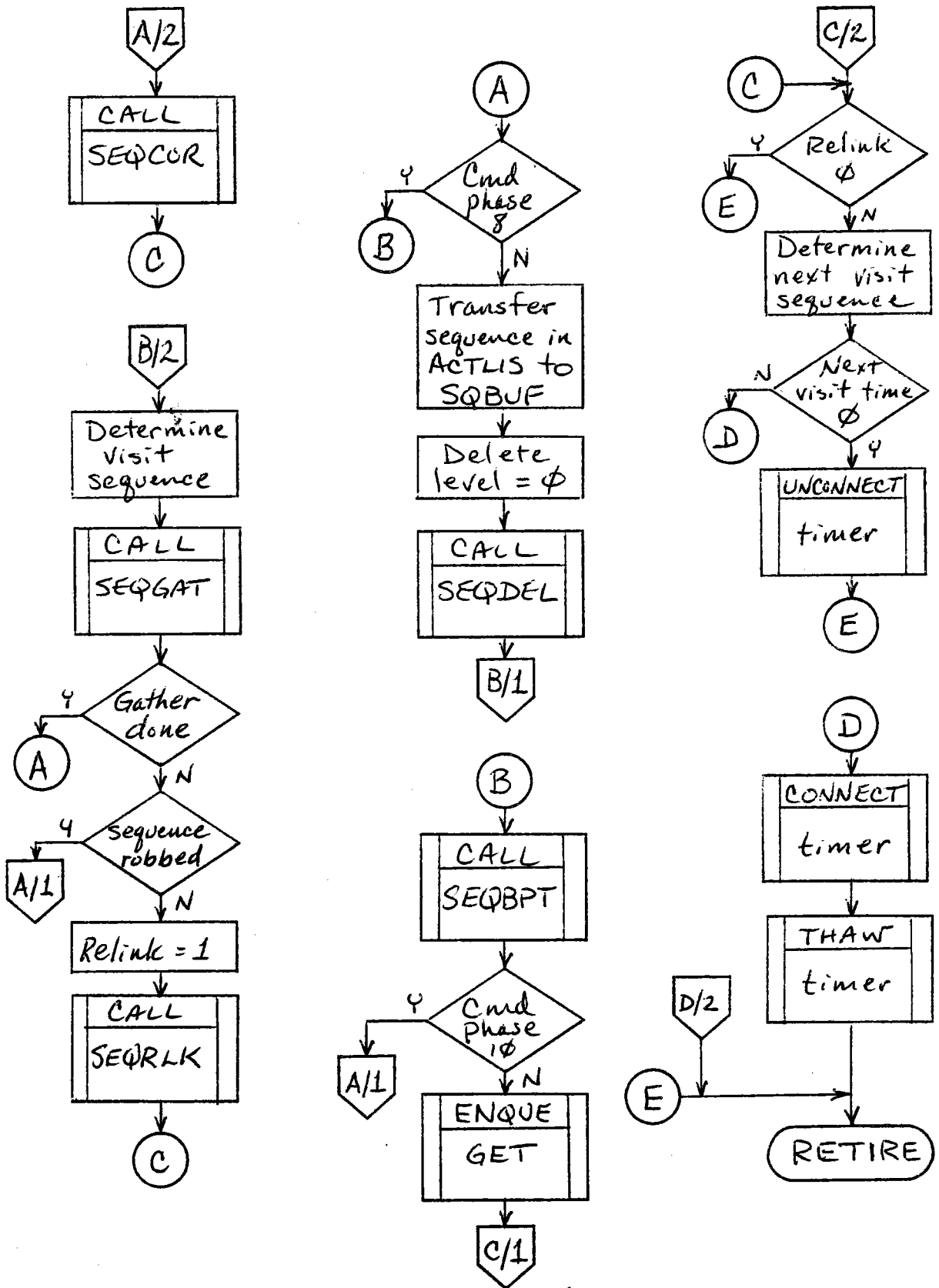


Figure 3.2.2-39 Flowchart - SEQ (Continued)
240

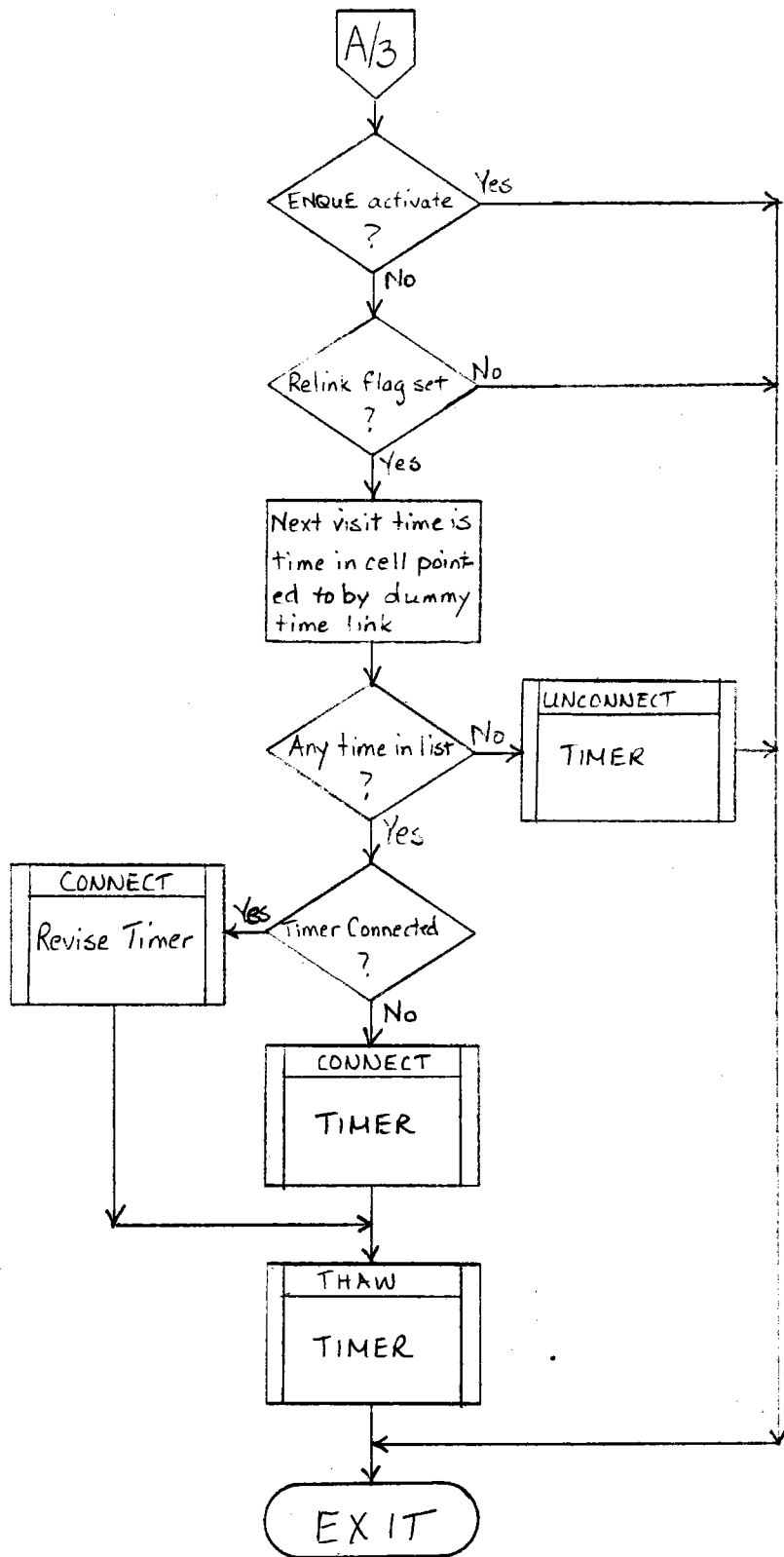


Figure 3.2.2-39 Flowchart - SEQ (continued)

5. Set N equal to element I of the gather buffer. Check if HC is still in the sequence. If so, go to Step (7).
6. Decrement the gather buffer size by one and delete the HC number from the gather buffer. Go to Step (4).
7. Check if the HC reports position compare. If so, go to Step (8). Otherwise, check if the timeout flag is set. If not, go to Step (4). If so, set the timeout flag in the HCST2G word (bit 8), and go to Step (6).
8. Transfer the HC number from the gather buffer to the lease buffer. Increment the lease buffer-pointer K and the lease buffer size. Go to Step (6).
9. If the timeout flag is set, go to Step (10). Otherwise, check the gather buffer size. If zero, gather is done; set return flag equal to one, and go to Step (11). If not zero, gather is not done; set return flag to zero, and go to Step (11).
10. Check if the lease buffer size is zero. If so, the sequence was robbed by an emergency command; set return flag equal to minus one, and go to Step (11). If not zero, gather is done; set return flag equal to one, and go to Step (11).
11. Return to caller.

e. Error messages and recovery - None

3.2.2.4.1.24.2 Data, Logic and Command Paths

Input data:

- a. Global common:
 1. HCST3G - HC sequence assignment
 2. HCST1G - HC position-compare bit
- b. Local common:
 1. Gather buffer
 2. Gather-lease flag
 3. TMOUT - sequence maximum-time flag

Output data:

- a. Global common:
 - HCST2G - HC timeout bit
- b. Local common:
 1. Lease buffer - those HCs reporting position compare for gather process
 2. Lease buffer address
 3. Lease buffer size
 4. Lease buffer pointer
 5. Gather buffer size
 6. Gather buffer
 7. Gather return flag

3.2.2.4.1.24.3 Internal Data Description

There is no data internal to this submodule.

3.2.2.4.1.24.4 Flowchart

See Figure 3.2.2-40 for the SEQGAT flowchart.

3.2.2.4.1.25 Submodule XXV - SEQBPT

3.2.2.4.1.25.1 Description

SEQBPT is responsible for issuing beam-pointing output commands for tracking either the CULP or the CLLP. When all involved HCs have been so commanded, release the corridor resources and free the buffer leased in submodule SEQGAT if the command is UNSTOW.

- a. Language used - FORTRAN IV
- b. How invoked - called by SEQ
- c. Constraints and limitations - none
- d. Processing -
 1. Set LOOP to the HC buffer size.
 2. Perform DO-UNTIL processing where I equals one to LOOP. If I is greater than LOOP, go to step (6).
 3. Set N equal to element I in the HC buffer to get HC number. If HC is in the sequence, go to step (4). If not, decrement the buffer size and delete the HC number from the buffer. Go to step (2).
 4. If the command is not UNSTOW, go to step (5). Otherwise, find the HC's corridor number J and right-justify. Set HCDATG(1-5,N) equal to CORRCG(1-5,J) to pick up the five-word CULP X,Y,Z coordinates. Set HCCMDG(N) equal to #8030, and go to step (2).
 5. If the command is UNSTOW, find HC's corridor number J and right-justify. Set HCDATG(1-5,N) equal to CORRCG(6-10,J) to pick up the five-word CLLP X,Y,Z coordinates. Set HCCMDG(N) equal to #8034, and go to step (2).
 6. Initialize for releasing corridor resources by setting sequence-packet corridor index J equal to four. Set LOOP equal to number of corridors in packet, SQBUF(4).
 7. Perform DO-UNTIL processing where I equals one to LOOP. If I is greater than LOOP, go to step (9).
 8. Increment J by one and set corridor number COR equal to SQBUF(J). Zero the corridor/sequence

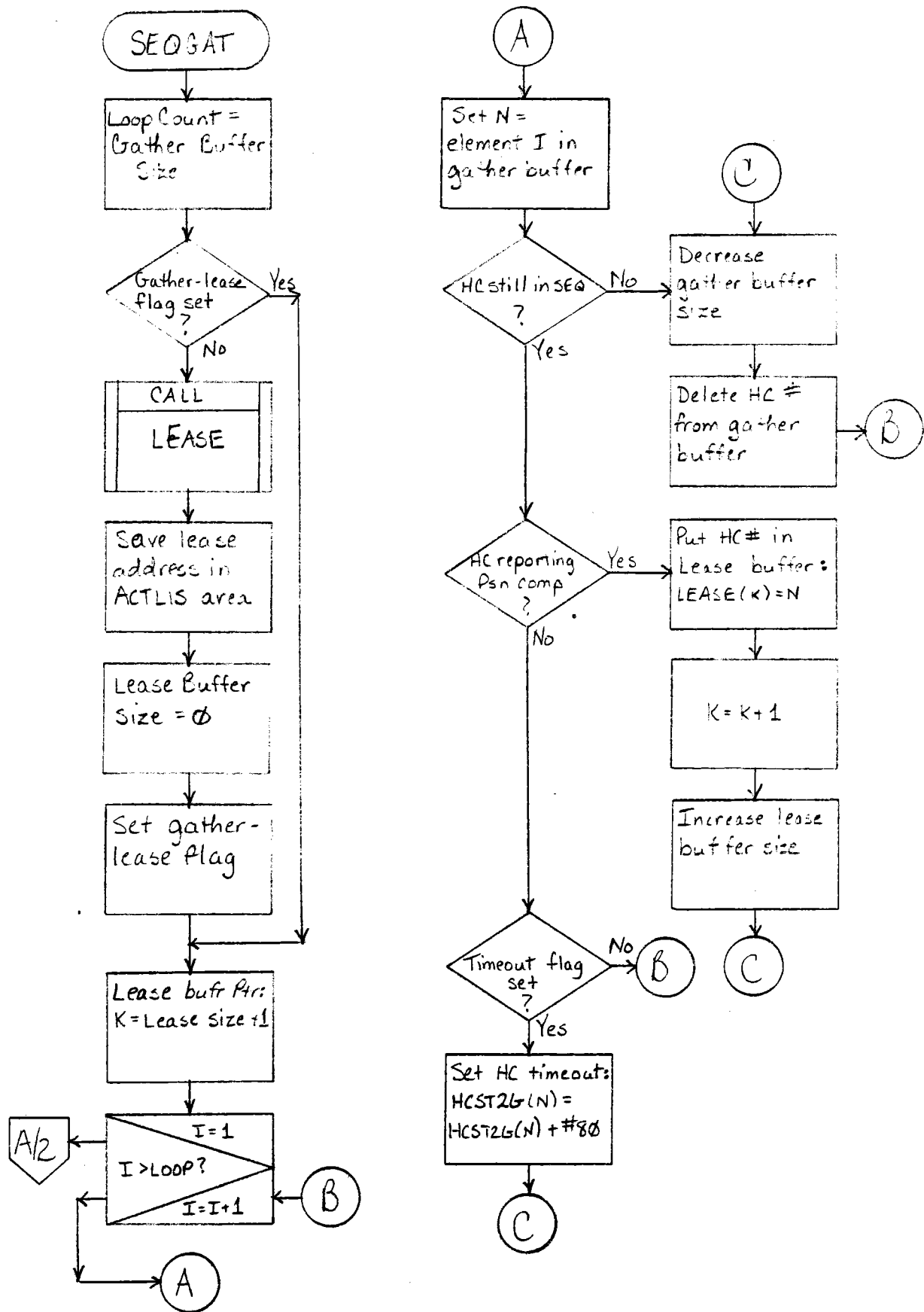


Figure 3.2.2-40 Flowchart - SEQGAT

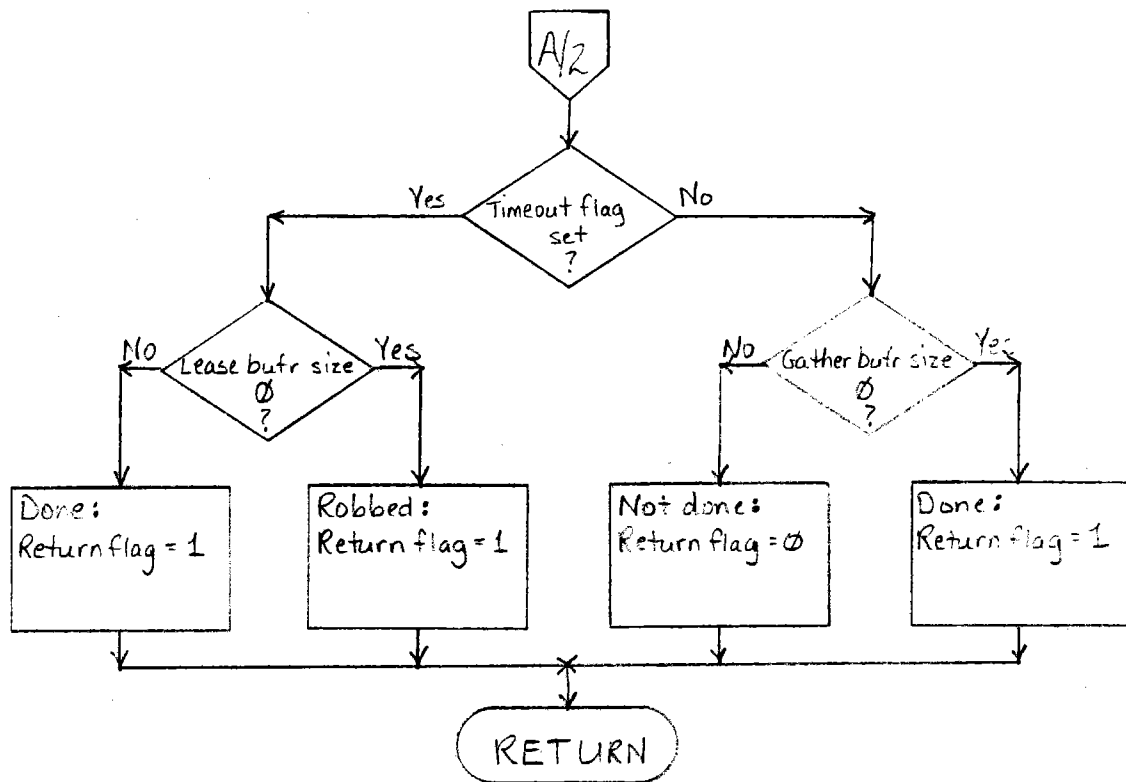


Figure 3.2.2-40 Flowchart - SEQGAT (continued)

number element: CORSQG(COR) equals zero.
Zero the corridor status element CORRSG(COR)
equal to zero. Go to step (7).

9. If the command is UNSTOW(corridor-up sequence), go to step (10). Otherwise, set the command phase to nine, and go to step (11).
10. For UNSTOW commands, call routine FREE to release the leased buffer obtained in SEQGAT. Set the command phase to ten.
11. Return to caller.

e. Error messages and recovery - None

3.2.2.4.1.25.2 Data, Logic and Command Paths

Input data:

- a. Global common:
 - HCST3G - HC corridor and sequence assignment
- b. Local common:
 - SQBUF - HC sequence packet

Output data:

- a. Global common:
 1. HCDATG - HC-data output array
 2. HCCMDG - HC-command output array
 3. CORSQG - corridor/sequence number array
 4. CORRSG - corridor-status array
- b. Local common:
 - LOOP - loop counter

3.2.2.4.1.25.3 Internal Data Description

There is no data internal to this submodule.

3.2.2.4.1.25.4 Flowchart

See Figure 3.2.2-41 for the SEQBPT flowchart.

3.2.2.4.1.26 Submodule XXVI - SEQCCK

3.2.2.4.1.26.1 Description

SEQCCK is responsible for searching the corridor-wait list (WAITSQ) and determining if any sequence can be added to the active-sequence list. This is allowed only if all the corridors required by the sequence are free. When this occurs, the sequence is deleted from the wait list and the sequence-ready flag is set.

- a. Language used - FORTRAN IV
- b. How invoked - called by SEQ

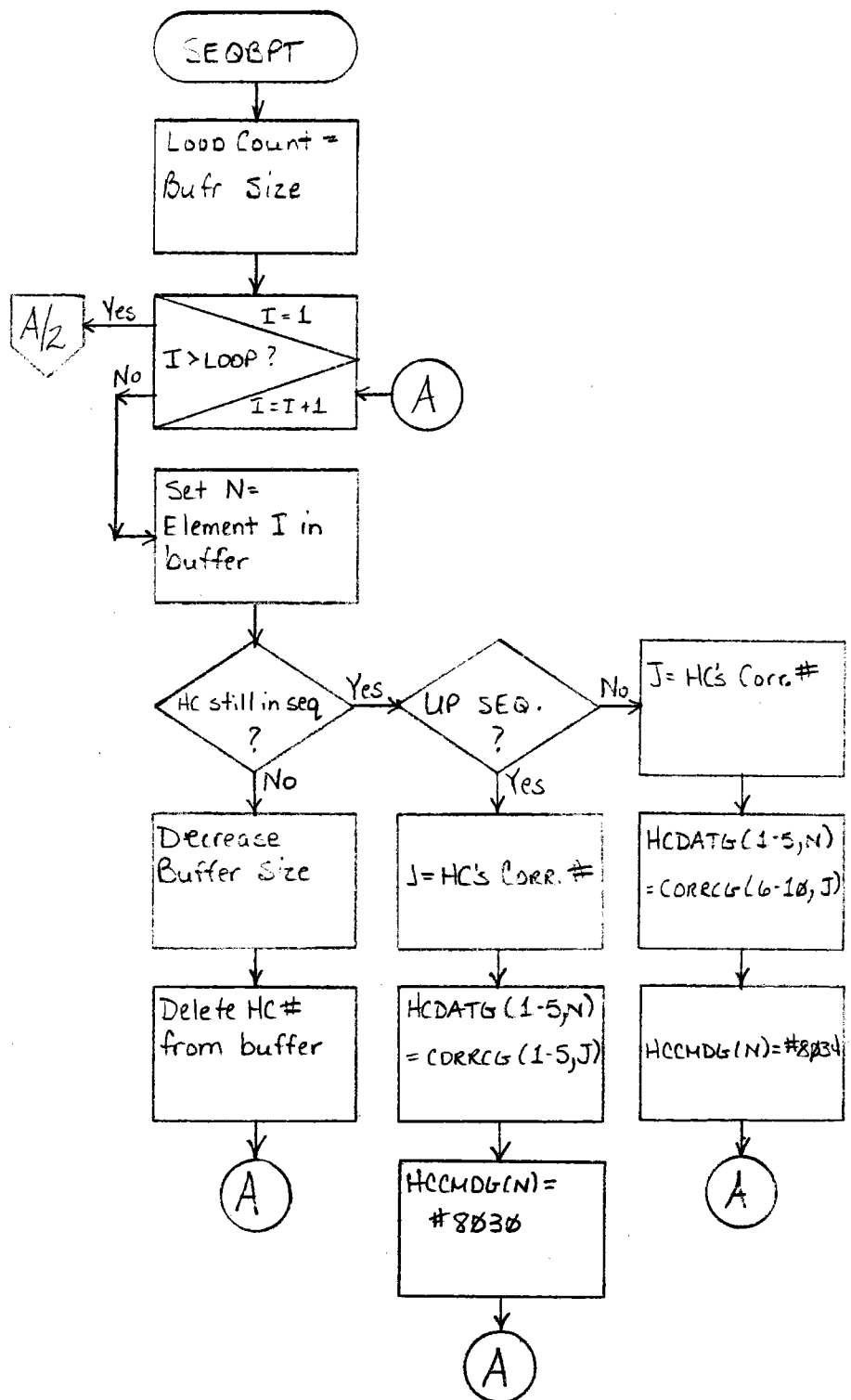


Figure 3.2.2-41 Flowchart - SEQBPT

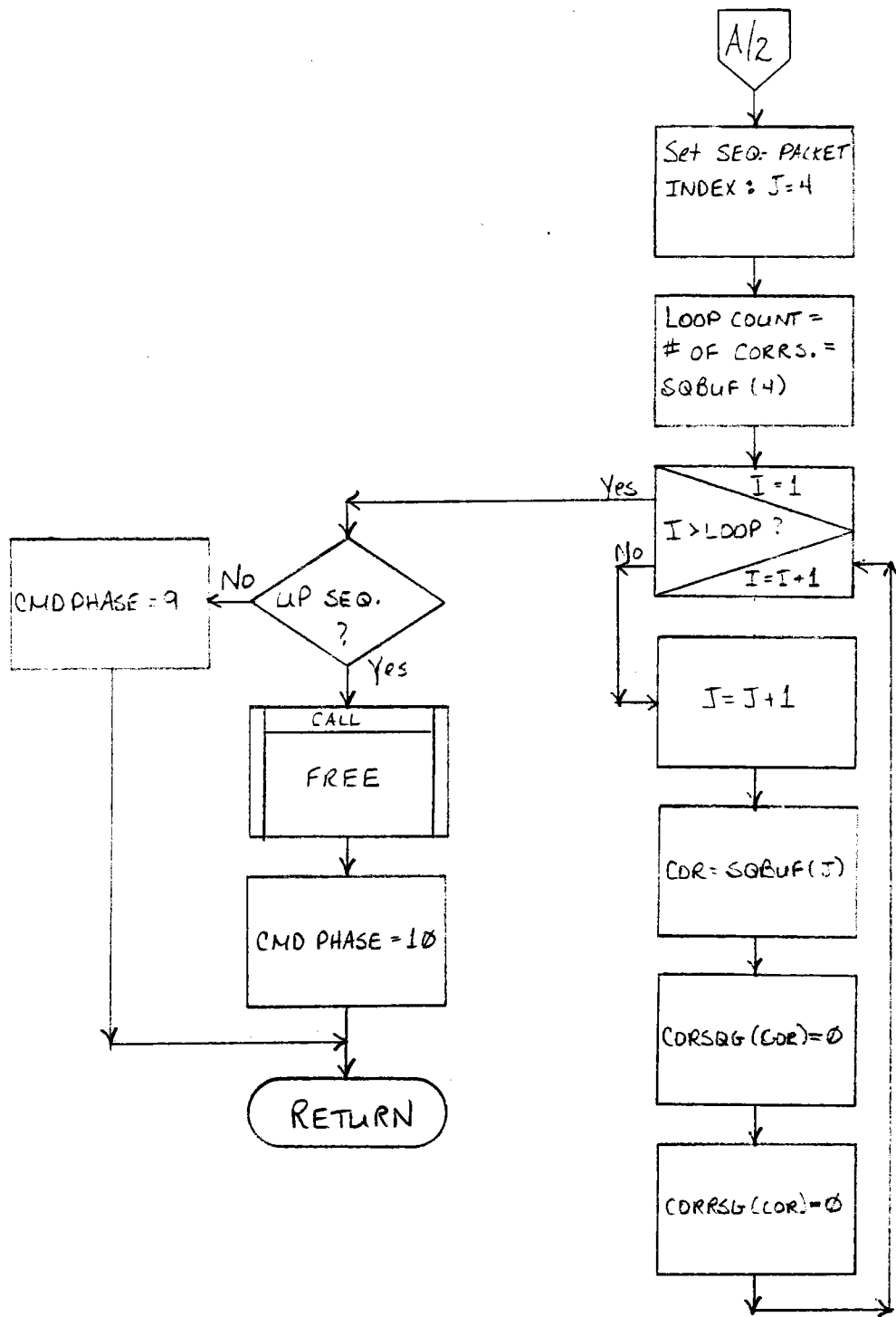


Figure 3.2.2-41 Flowchart - SEQBPT (continued)

c. Constraints and limitations - None

d. Processing -

1. Perform DO-UNTIL processing where J equals one to 15. If J is greater than 15, go to step (5).
2. See Figure 3.2.2-38 for the WAITSQ array format. Check WAITSQ(J,1) for a negative number. If so, this is the end of list; go to step (5). If not, determine if a sequence is present (WAITSQ(J,1) is non-zero). If not, go to step (1). If so, set the loop count LOOP equal to WAITSQ(J,2), the number of corridors required for the sequence. Set the WAITSQ corridor index ICOR equal to three to point at the first corridor number in the entry.
3. Perform DO-UNTIL processing where K equals one to LOOP. If K is greater than LOOP, go to step (6).
4. Set the corridor number COR equal to WAITSQ(J,ICOR). If CORRSG(COR) equals zero (corridor is free), go step (3). If not, go to step (1) to get another wait-sequence entry.
5. Clear the sequence-ready flag, and go to step (11).
6. Set corridor index ICOR equal to three.
7. Perform DO-UNTIL processing where K equals one to LOOP. If K is greater than LOOP, go to step (9).
8. Get corridor number COR equal to WAITSQ(J,ICOR). If command is UNSTOW, set command phase equal to six and set the corridor's status word for ready to go up, CORRSG(COR) equals one; go to step (7). If not UNSTOW, set the command phase equal to seven and set the corridor's status word for ready to go down, CORRSG(COR) equals two; go to step (7).
9. Set SQAD equal to address of the sequence packet which is contained in WAITSQ(J,11). Delete the sequence from the wait-sequence list by WAITSQ(J,1) equal to zero.
10. Set the sequence-ready flag.
11. Return to caller.

e. Error messages and recovery - None

3.2.2.4.1.26.2 Data, Logic and Command Paths

Input data:

a. Global common:

CORRSG - corridor-status array

b. Local common:

1. WAITSQ - wait-sequence list
2. SQBUF(1) - command number

Output data:

a. Global common:

CORRSG - corridor-status array

b. Local common:

1. LOOP - loop count
2. Sequence-ready flag
3. WAITSQ - wait-sequence list
4. COR - corridor assigned to sequence
5. ICOR - WAITSQ-entry corridor index
6. SQBUF(2) - command phase for sequence
7. SQAD - address of sequence packet

3.2.2.4.1.26.3 Internal Data Description

There is no data internal to this submodule.

3.2.2.4.1.26.4 Flowchart

See Figure 3.2.2-42 for the SEQCCK flowchart.

3.2.2.4.1.27 Submodule XXVII - SEQCOR

3.2.2.4.1.27.1 Description

SEQCOR is responsible for commanding the involved HCs into their respective corridors. If an HC has been robbed from the sequence, the HC is deleted from the buffer.

a. Language used - FORTRAN IV

b. How invoked - called by SEQ

c. Constraints and limitations - None

d. Processing -

1. Set LOOP equal to buffer size.
2. If command is UNSTOW (up corridor), set HFCCOI (HFC corridor-increment word) to zero. If not UNSTOW, set HFCCOI equal to four.
3. Perform DO-UNTIL processing where I equals one to LOOP. If I is greater than LOOP, go to step (7).
4. Set N equal to element I in the buffer to get the HC number.
5. Determine if HC is still in the sequence. If so, go to step (6). If not, decrement the buffer

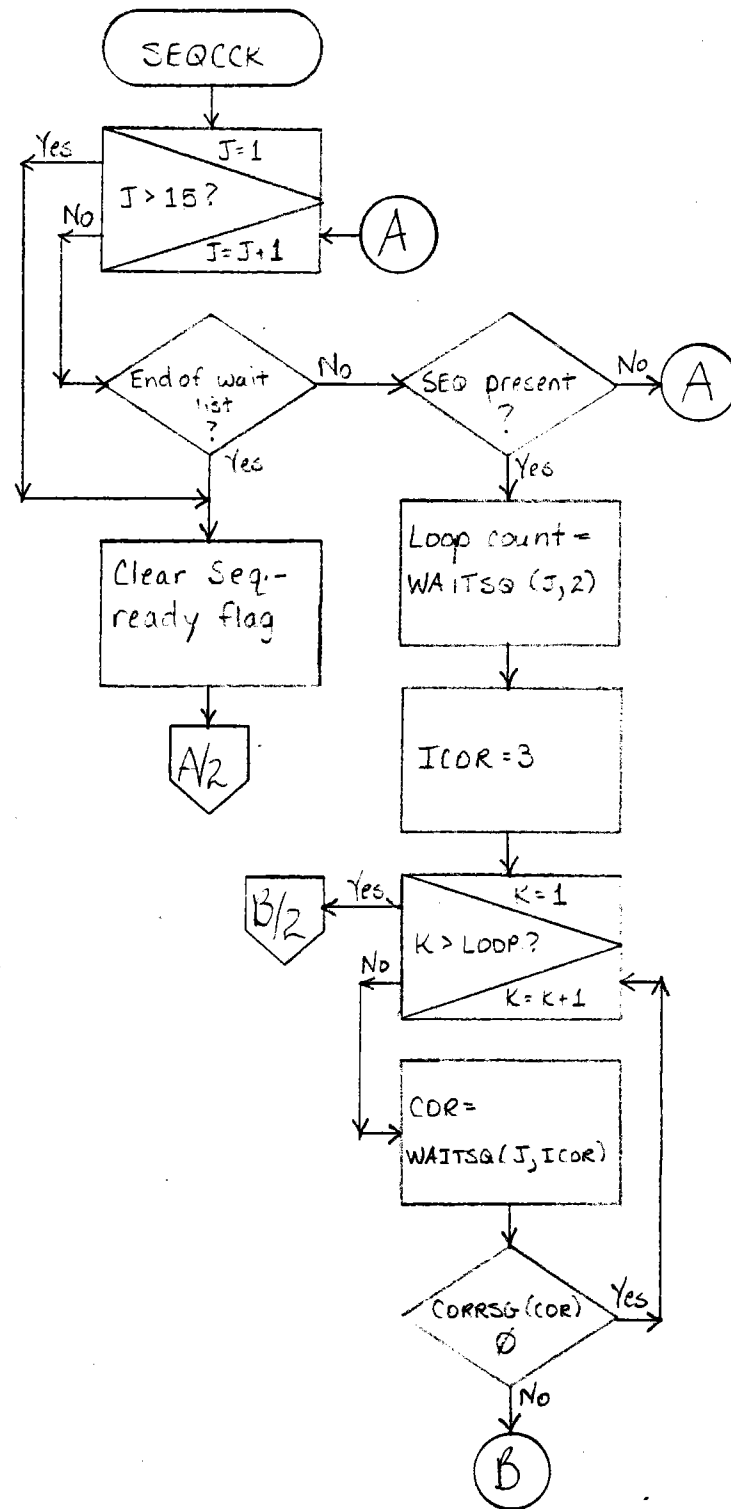


Figure 3.2.2-42 Flowchart - SEQCCK

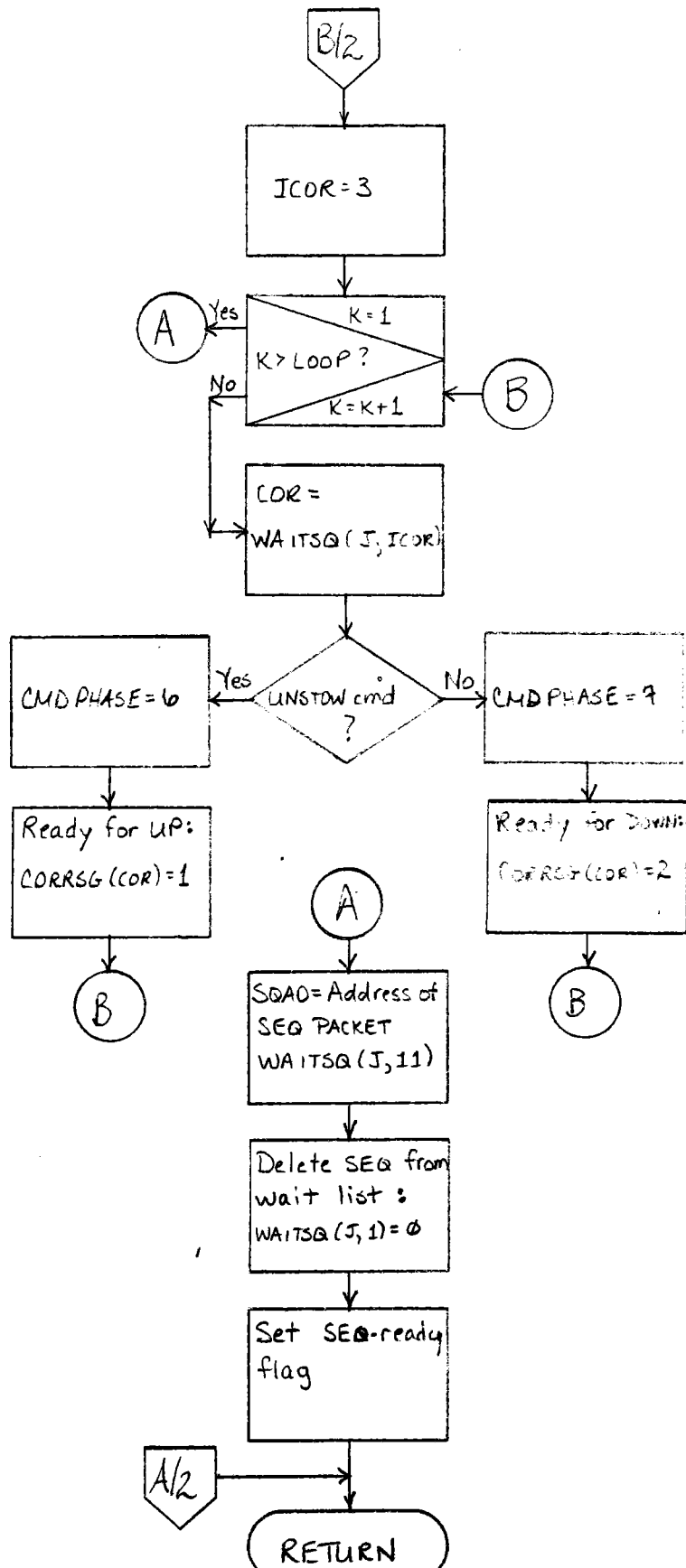


Figure 3.2.2-42 Flowchart - SEQCCK (continued)

size by one and delete the HC number from the buffer. Go to step (3).

6. Set HCCMDG(N) equal to #8040 plus HFCCOI, and go to step (3).

7. Set command phase equal to eight (gather HCs in corridor) and return to caller.

e. Error messages and recovery - None

3.2.2.4.1.27.2 Data, Logic and Command Paths

Input data:

a. Global common:

HCST3G - HC sequence-assignment array

b. Local common:

1. Gather buffer

2. HFCCOI - HFC corridor-increment word

3. Gather buffer size

Output data:

a. Global common:

HCCMDG - HC-command output array

b. Local common:

1. LOOP - loop count

2. HFCCOI - HFC corridor-increment word

3. SQBUF(2) - command phase

4. Gather buffer

5. Gather buffer size

3.2.2.4.1.27.3 Internal Data Description

There is no data internal to this submodule.

3.2.2.4.1.27.4 Flowchart

See Figure 3.2.2-43 for the SEQCOR flowchart.

3.2.2.4.1.28 Submodule XXVIII - SEQADD

3.2.2.4.1.28.1 Description

SEQADD is responsible for either adding a sequence to the active-sequence list (ACTLIS) or merely providing a call to the relink submodule (SEQRLK). See Table 3.2.2-VII for ACTLIS format.

a. Language used - FORTRAN IV

b. How invoked - called by SEQ

c. Constraints and limitations - None

d. Processing -

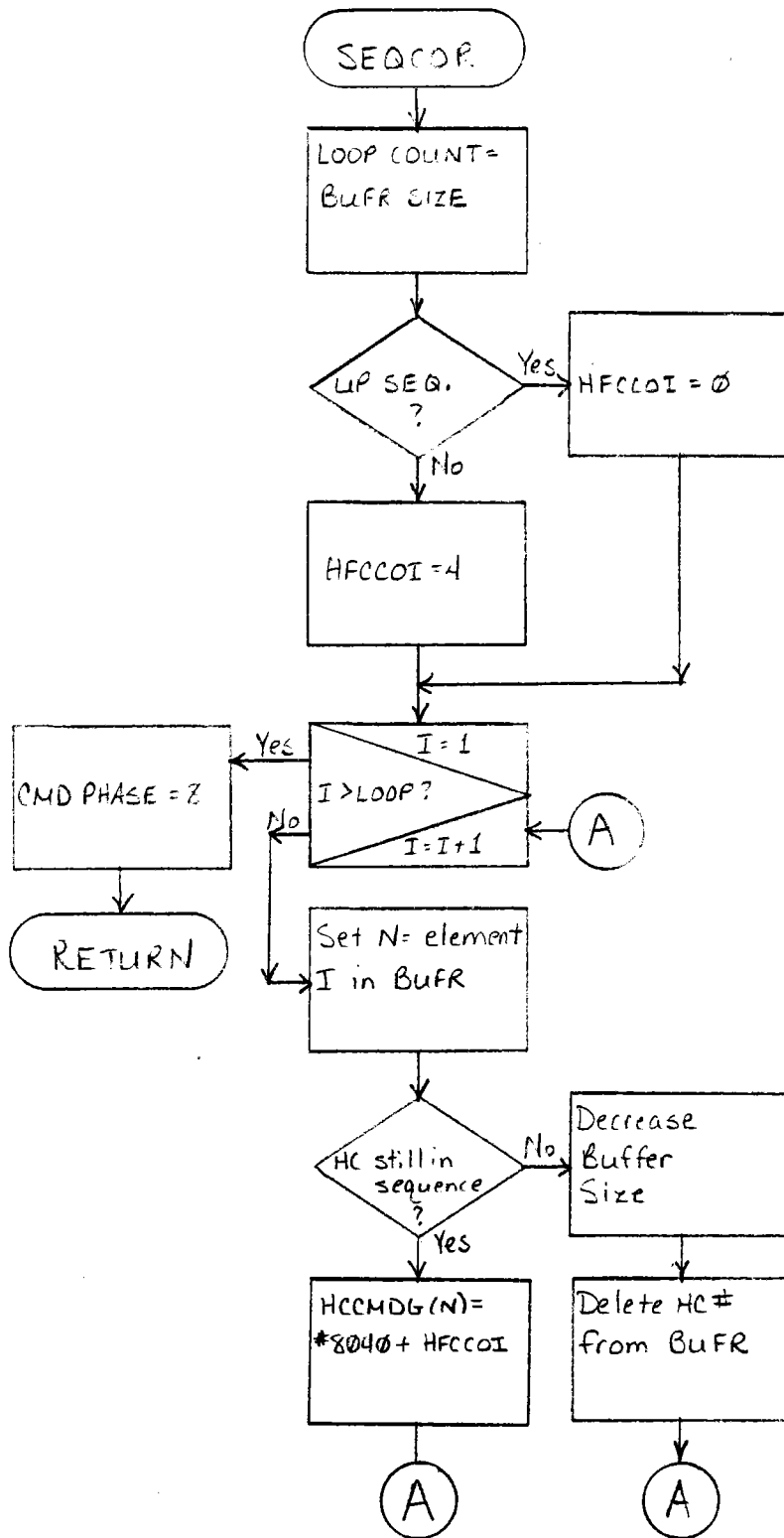


Figure 3.2.2-43 Flowchart - SEQCOR

<u>WORD</u>	<u>DESCRIPTION</u>
1,2	NEXT VISIT TIME FOR THIS SEQUENCE (MINUTES & TICKS SINCE MIDNIGHT)
3	TIME-LINK CELL POINTER; CONTAINS THE CELL # HAVING THE NEXT CHRONOLOGICAL VISIT TIME AS COMPARED TO THIS SEQUENCE'S VISIT TIME
4	NEXT AVAILABLE MEMORY CELL; WHEN THIS CELL IS EMPTY, VALUE IS NEXT EMPTY CELL #; WHEN A SEQUENCE OCCUPIES THIS CELL, VALUE IS THIS CELL #
5,6	MAXIMUM (FINAL) GATHER TIME FOR THIS SEQUENCE (MINUTES & TICKS SINCE MIDNIGHT)
7	COMMAND # (VALUES: 6, 14, 17 or 18)
8	COMMAND PHASE (VALUES: 4 or 8)
9	SEQUENCE # (RANGE: 1 to 16)
10	# OF CORRIDORS REQUIRED (RANGE: 1 to 8)
11-N	CORRIDOR #(s)
N+1	# OF BUFFER DESCRIPTIONS TO FOLLOW
N+2	OFFSET (INTO COMQU2) FOR LEASED HC # BUFFER
N+3	SEQUENCE GATHER COUNT
N+4	SIZE FOR BUFFER #1 (RANGE: 1 to 2048)
N+5	OFFSET (INTO COMQU2) FOR BUFFER #1
N+6 thru N+11	Optional - see Table 3.2.2-V

NOTE: Format of words 7 thru N+11 follow Table 3.2.2-V except word N+3 which is assigned as a counter. List is 16 cells of 29 words per cell. It is a time-linked list of active sequences (HCs moving).

Table 3.2.2-VII Typical Cell in Active - Sequence List (ACTLIS)

1. If the command phase is four (gather), go to step (2). Otherwise, set the corridor-walk gather minimum and maximum times to current time plus 360 and 424 seconds, respectively. Go to step (3).
2. Set the non-corridor gather minimum and maximum times to current time plus 400 and 464 seconds, respectively.
3. Insert the sequence number and minimum time into the cell (add cell) referred to as available in the dummy cell in ACTLIS. Insert the maximum time, the gather buffer address, and size. Zero the lease-flag word in the cell. Swap the available memory numbers between the add cell and the dummy cell. If the dummy time-link cell number is not equal to zero, go to step (4). If zero, no other sequences are active; set the dummy time-link cell number equal to add cell number. Set the relink flag, and go to step (9). Reorder the time-linked list (see Figure 3.2.2-44 for example).
4. Set the old cell equal to zero for dummy reference. Set the compare cell equal to dummy time-link cell.
5. Check if the add-cell's time is less than compare-cell's time. If yes, go to step (7). Reorder the time-linked list (see Figure 3.2.2-44 for example). Otherwise, check if add-cell's time equals compare-cell's time. If yes, increment the add-cell's time by 500 milliseconds and check if the compare-cell's time link is zero. If not, go to step (6). If so, set compare-cell's time link equal to add-cell number. Set the add-cell's time link equal to zero (add cell has largest visit-time value in ACTLIS). Go to step (9).
6. Set old cell equal to compare cell. Set compare cell equal to compare-cell's time link. Go to step (5).
7. Check if the old cell equals zero (dummy). If so, set dummy time link equal to add cell. If not, set old-cell's time link equal to add cell. Set the add-cell's time link equal to compare cell. If old cell equals zero, set the relink flag, and go to step (8). Otherwise, go to step (9).
8. Call submodule SEQRLK to relink all time-link cell pointers according to visit times.
9. Return to caller.

e. Error messages and recovery - None

$\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \text{ DUMMY CELL}$ $\begin{pmatrix} 0 & 0 \\ 0 & 2 \end{pmatrix} \text{ CELL \#1}$ $\begin{pmatrix} 0 & 0 \\ 0 & 3 \end{pmatrix} \text{ CELL \#2}$ $\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \text{ CELL \#3}$	$\begin{pmatrix} 1 & 0 \\ \emptyset & 2 \end{pmatrix}$ $\begin{pmatrix} 10:06:40 & 1 \\ \emptyset & \emptyset \\ 0 & 1 \\ & 2 \end{pmatrix}$ $\begin{pmatrix} 0 & 0 \\ 0 & 3 \end{pmatrix}$ $\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 1 & 3 \end{pmatrix}$ $\begin{pmatrix} 10:06:40 & 1 \\ 2 & 1 \\ \emptyset & \emptyset \end{pmatrix}$ $\begin{pmatrix} 10:07 & 2 \\ \emptyset & \emptyset \\ 0 & 2 \\ & 3 \end{pmatrix}$ $\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
<p>Assume ACTLIS to hold 3 seqs. for this example. Only first 5 words of an entry are used here. Dummy cell is at top and contains no visit time value.</p>	<p>Action: Add seq. 1, cmd phase = 4</p>	<p>Action: Add seq. 2, cmd phase = 8</p>
<p>Legend:</p> <p>V.T. SEQ</p> <p>T.L. A.M.</p> <p>V.T. - Visit time (minutes, ticks)</p> <p>A.M. - Next available memory cell when this cell cleared</p> <p>T.L. - Time link pointing to cell with next visit time</p> <p>SEQ - Sequence #</p>	<ul style="list-style-type: none"> - Current time = 10:00 - Seq min = visit time = 10:06:40 - Seq max = 10:07:44 - Av. mem cell in dummy cell is 1 (add cell) - Insert visit time, seq # in cell 1 - Swap av. mem. #s between add cell and dummy cell - Dummy T.L. = 0 ? Yes - Dummy T.L. = add cell - Set relink flag 	<ul style="list-style-type: none"> - Current time = 10:01 - Seq min = visit time = 10:07 - Seq max = 10:08:04 - Av. mem cell in dummy cell is 2 (add cell) - Insert visit time, seq # in cell 2 - Swap av. mem. #s between add cell and dummy cell - Dummy T.L. = 0 ? No - Old cell = 0 (Dummy) - Comp cell = dummy T.L. = 1 - Add cell V.T. < comp cell V.T. ? No - Comp cell T.L. = 0 ? Yes - Comp cell T.L. = add cell = 2 - Add cell T.L. = 0

Figure 3.2.2-44 Example of ACTLIS Management

$\begin{pmatrix} & 0 \\ 1 & 3 \\ & 56 \\ 10:06:40 & 1 \\ 2 & 1 \\ & 10:07 \\ 0 & 2 \\ & 0 \\ 0 & 0 \\ & 0 \end{pmatrix}$ <p>Action: Visit seq. 1</p>	$\begin{pmatrix} & 0 \\ 2 & 3 \\ & 1 \\ 10:07:12 & 1 \\ 10:06:50 & 1 \\ 0 & 1 \\ & 2 \\ & 10:07 \\ 1 & 2 \\ & 0 \\ 0 & 0 \\ & 0 \end{pmatrix}$ <p>Action: Visit seq. 1</p>	$\begin{pmatrix} & 0 \\ 1 & 3 \\ & 2 \\ 10:07:12 & 1 \\ 2 & 1 \\ & 10:07:16 \\ 0 & 2 \\ & 0 \\ 0 & 0 \\ & 0 \end{pmatrix}$ <p>Action: Visit seq. 2</p>
<ul style="list-style-type: none"> - Current time = 10:06:41 - Old cell = 0 - Visit cell visit time = 10:06:56 - Comp cell = visit cell T.L. = 2 - Comp cell = 0 ? No - Visit time < comp. cell V.T. ? Yes - Dummy T.L. = visit cell = 1 	<ul style="list-style-type: none"> - Current time = 10:06:57 - Old cell = 0 - Visit cell V.T. = 10:07:12 - Comp cell = visit cell T.L. = 2 - Comp cell = 0 ? No - Visit time < comp cell V.T. ? No - Visit time = comp cell V.T. ? No - Comp cell T.L. = 0 ? Yes - Comp cell T.L. = visit cell = 1 - Visit cell T.L. = 0 - Old cell = 0 ? Yes - Dummy T.L. = comp cell = 2 	<ul style="list-style-type: none"> - Current time = 10:07 - Old cell = 0 - Visit cell V.T. = 10:07:16 - Comp cell = visit cell T.L. = 1 - Comp cell = 0 ? No - Visit time < comp cell V.T. ? No - Visit time = comp cell V.T. ? No - Comp cell T.L. = 0 ? Yes - Comp cell T.L. = visit cell = 2 - Visit cell T.L. = 0 - Old cell = 0 ? Yes - Dummy T.L. = comp cell = 1

Figure 3.2.2-44 Example of ACTLIS Management (cont'd)

$\begin{pmatrix} & 0 \\ 1 & 0 \\ & 0 \\ 10:07:12 & 1 \\ 2 & 1 \\ 10:07:16 & 2 \\ 3 & 2 \\ 10:14:20 & 3 \\ \emptyset & \emptyset \\ 0 & 3 \\ & \emptyset \end{pmatrix}$ <p>Action: Add seq. 3, cmd phase = 4</p>		$\begin{pmatrix} & 0 \\ 2 & 1 \\ 1 & \emptyset \\ 0 & 0 \\ 10:07:14 & 1 \\ 2 & 0 \\ & 1 \\ 10:07:32 & 2 \\ 3 & 2 \\ 10:14:20 & 3 \\ 0 & 3 \end{pmatrix}$ <p>Action: Delete Seq. 1</p>
<ul style="list-style-type: none"> - Current time = 10:07:40 - Seq. min = visit time = 10:14:20 - Seq. max = 10:15:24 - Av. mem. cell in dummy cell = 3 (add cell) - Insert visit time, seq. # in cell 3 - Swap av. mem. #s between add cell and dummy cell - Dummy T.L. = 0 ? No - Old cell = 0 (Dummy) - Comp cell = Dummy T.L. = 1 - Add cell V.T. < comp cell V.T. ? No - Add cell V.T. = comp cell V.T. ? No - Comp cell T.L. = 0 ? No 	<ul style="list-style-type: none"> - Old cell = comp cell = 1 - Comp cell = comp cell T.L. = 2 - Add cell V.T. < comp cell T.L. ? No - Add cell V.T. = comp cell T.L. ? No - Comp cell T.L. = 0 ? Yes - Comp cell T.L. = add cell = 3 - Add cell T.L. = 0 	<ul style="list-style-type: none"> - Current time = 10:07:45 - Old cell = 0 (Dummy) - Comp cell = dummy T.L. = 1 - Comp cell = 0 ? No - Comp cell seq # = del seq. # ? Yes - Del cell seq # = 0 - Del cell V.T. = 0 - Swap av. mem. #s between del cell and dummy cell - Comp cell T.L. = 0 ? No - Old cell = 0 ? Yes - Dummy T.L. = comp cell T.L. = 2 - Step 6 of SEQDEL

Figure 3.2.2-44 Example of ACTLIS Management (cont'd)

3.2.2.4.1.28.2 Data, Logic and Command Paths

Input data:

Local common:

- a. RELINK - relink flag
- b. SQBUF(2) - command phase

Output data:

Local common:

ACTLIS - active-sequence list

3.2.2.4.1.28.3 Internal Data Description

ACTLIS (see Table 3.2.2-VII)

3.2.2.4.1.28.4 Flowchart

See Figure 3.2.2-45 for the SEQADD flowchart.

3.2.2.4.1.29 Submodule XXIX - SEQRLK

3.2.2.4.1.29.1 Description

SEQRLK is responsible for reordering the active-sequences list (ACTLIS) for ascending-order visit times. The visit sequence is that sequence which has just been processed by submodule SEQGAT.

- a. Language used - FORTRAN IV
- b. How invoked - called by SEQ
- c. Constraints and limitations - None
- d. Processing -

Reorder the time-linked list (see Figure 3.2.2-44 for example):

1. Initialize by: zero the equal flag (EQUAL). Set old cell equal to zero (dummy). Update the visit-sequence's time for next visit by adding 16 seconds to its value in the ACTLIS entry. Set the compare cell equal to visit-cell time link. Check if the compare cell equals zero. If yes, go to step (7).
2. Check if visit time is less than compare-cell time. If so, go to step (6). If not, check visit time to equal compare-cell time. If so, go to step (5). Check the compare-cell's time link (t.l.) to be zero. If not, go to step (3). Set the compare-cell's time link equal to visit cell and set the visit-cell time link equal to zero to put the visit sequence at the bottom of the linked-list temporarily. Go to step (8).

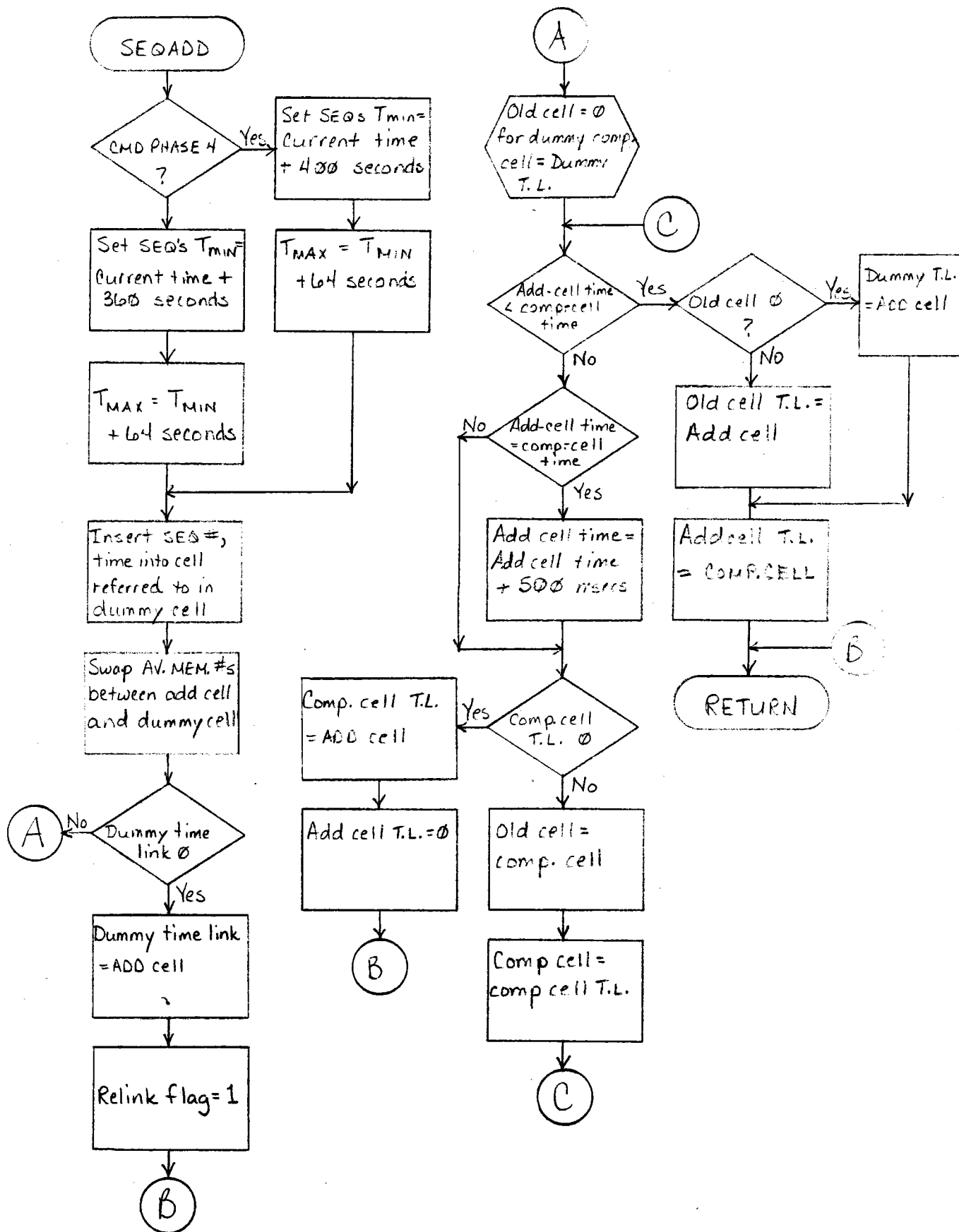


Figure 3.2.2-45 Flowchart - SEQADD

3. Set dummy-cell time link equal to visit-cell time link and set the visit-cell time link equal to compare-cell time link.
4. Set old cell equal to compare cell and set compare cell equal to compare-cell's time link. Go to step (2).
5. Set visit time equal to visit time plus 500 milliseconds to break conflict when two sequences have same revisit time. Go to step (4).
6. Check if equal flag (EQUAL) set. If so, set the visit-cell time link equal to old-cell's time link.
7. Check if old cell equals zero. If so, set dummy time link equal to visit cell and return to caller. Otherwise, set the old-cell's time link equal to visit cell and return to caller.
8. Check if old cell equals zero. If so, set dummy time link equal to compare cell and return to caller. Otherwise, set old cell equal to compare cell and return to caller.

e. Error messages and recovery - None

3.2.2.4.1.29.2 Data, Logic and Command Paths

Input data:

Local common:

ACTLIS - active-sequence list

Output data:

Local common:

a. ASTLIS - active-sequence list

b. EQUAL - visit-sequence's time matched another sequence's time during processing

3.2.2.4.1.29.3 Internal Data Description

ACTLIS (see Table 3.2.2-VII for format)

3.2.2.4.1.29.4 Flowchart

See Figure 3.2.2-46 for the SEQRLK flowchart.

3.2.2.4.1.30 Submodule XXX - SEQDEL

3.2.2.4.1.30.1 Description

SEQDEL is responsible for deleting a sequence from the active-sequence list (ACTLIS).

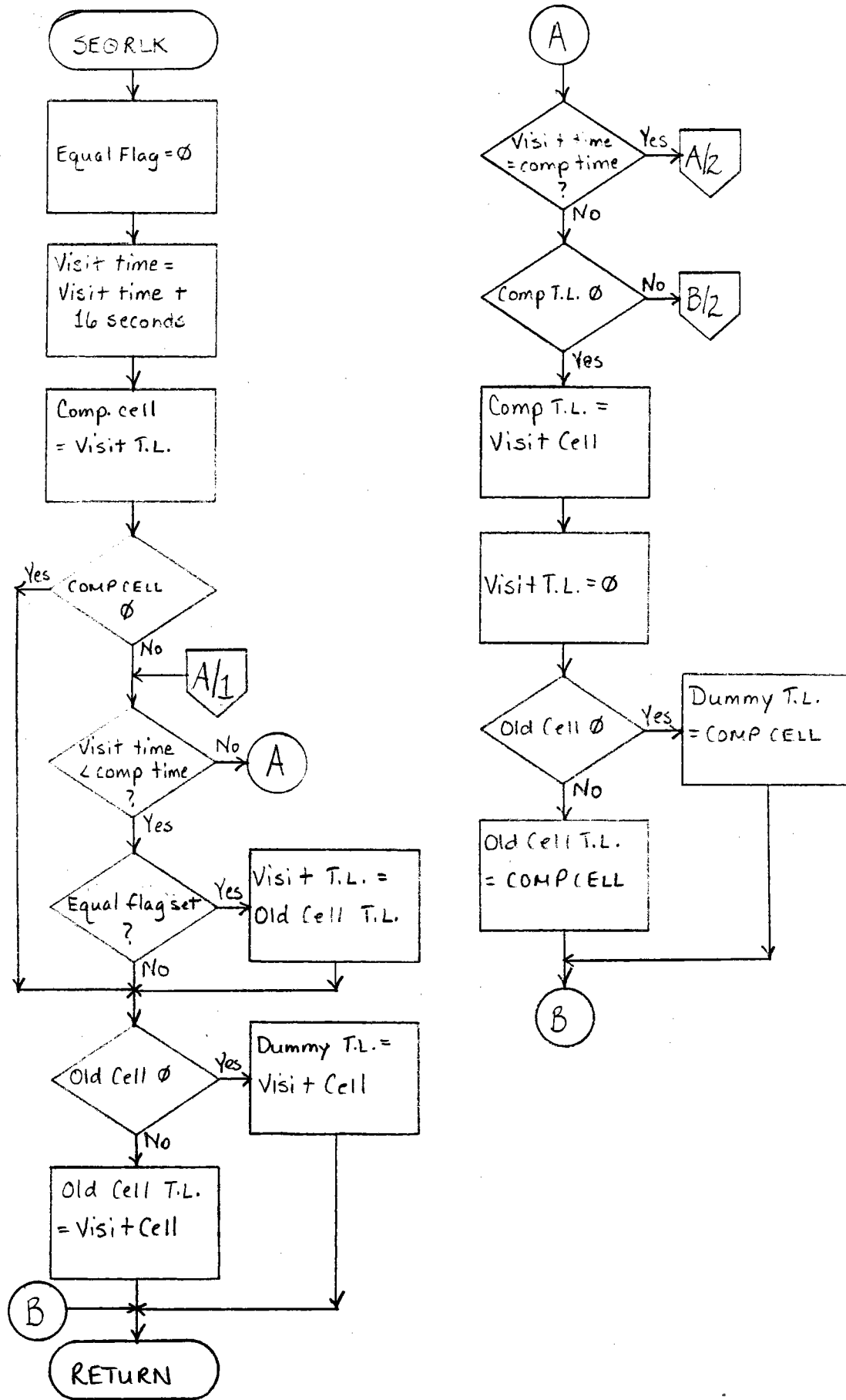


Figure 3.2.2-46 Flowchart - SEORLK

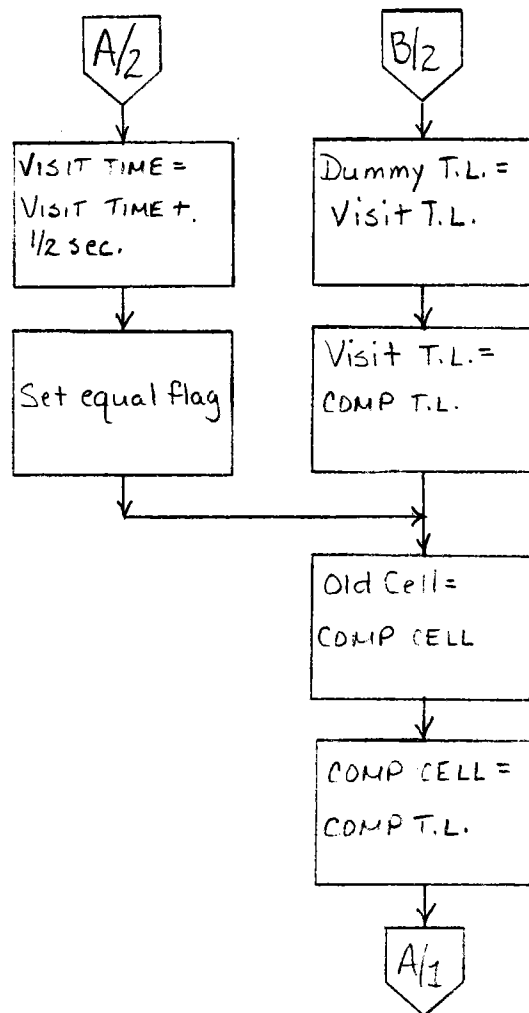


Figure 3.2.2-46 Flowchart - SEQRLK (continued)

- a. Language used - FORTRAN IV
- b. How invoked - called by SEQ
- c. Constraints and limitations - None
- d. Processing -

See Figure 3.2.2-44 for example.

1. Set old cell equal to zero (dummy) and set compare cell equal to dummy time link. If compare cell equals zero, go to step (7).
2. Check compare cell's sequence number to delete cell's sequence number. If equal, go to step (4).
3. Set old cell equal to compare cell and set compare cell equal to compare-cell time link. Go to step (2).
4. Clear the delete cell by zeroing sequence number and visit time. Swap the available memory cell numbers between the delete cell and the dummy cell. Check if the compare-cell's time link equals zero (no sequence has later visit time). If not, go to step (5). If so, zero the relink flag (RELINK) and check if old cell equals zero. If so, the delete cell was the only active cell; set dummy time link equal to zero and go to step (6). If old cell is non-zero, set old-cell's time link equal to zero, and go to step (6).
5. Check if old cell equals zero. If so, set dummy time link equal to compare-cell's time link. Set the relink flag (RELINK), and go to step (6). If old cell non-zero, set old-cell's time link equal to compare-cell's time link. Set the relink flag, and go to step (6).
6. Reset the sequence status in the SEQLSG array to the negative sequence number by SEQLSG (SQBUF(3)) equal to minus SEQLSG (SQBUF(3)). Set SEQNMG equal to SEQNMG minus one to decrement the number of sequences in progress. If the command is STHIWIND, clear the emergency-sequence flag EMSEQG equal to zero.
7. Return to caller.

- e. Error messages and recovery - None

3.2.2.4.1.30.2 Data, Logic and Command Paths

Input data:

Local common:

- a. ACTLIS - active-sequence list
- b. SQBUF(1) - command number

Output data:

Local common:

- a. ACTLIS - active-sequence list
- b. RELINK - relink flag

3.2.2.4.1.30.3 Internal Data Description

There is no data internal to this submodule.

3.2.2.4.1.30.4 Flowchart

See Figure 3.2.2-47 for the SEQDEL flowchart.

3.2.2.4.1.31 Submodule XXXI - BHI (Task)

3.2.2.4.1.31.1 Description

- a. Language used - FORTRAN IV
- b. How invoked - It is invoked by FCP task once each second after the status response messages have been processed and before commands are output to the HFCs.
- c. Constraints and limitations - The task BHI is constrained to send one subtype of HFC initialization command each second. Task BHI is allowed 40 msec to execute and shall have a high-priority just below the FCP task.
- d. Processing - When activated by FCP, the BHI task shall check the HFC status to determine if any require initialization. Checks shall be made to ensure the HFC is installed, requires initialization, and does not already have a command ready to be output. If all these conditions are true, the BHI task will construct the appropriate subtype initialization message and place it into the command buffer.

Subroutine BHISH5 is called to manipulate a five-word command input buffer and place the command byte into the command output buffer. After checking all HFCs, the BHI task will release control.
- e. Error messages and recovery - None

3.2.2.4.1.31.2 Data, Logic and Command Paths

Input data:

- a. HFC status words (HFCS1G, HFCS2G, and HFCS3G).
- b. Corridor coordinates (CORRCG).

Output data:

HFC initialization commands in command buffer (CMDBFG).

Algorithms:

Logic to calculate corridor assignment masks. BHISH5 formats the command buffer.

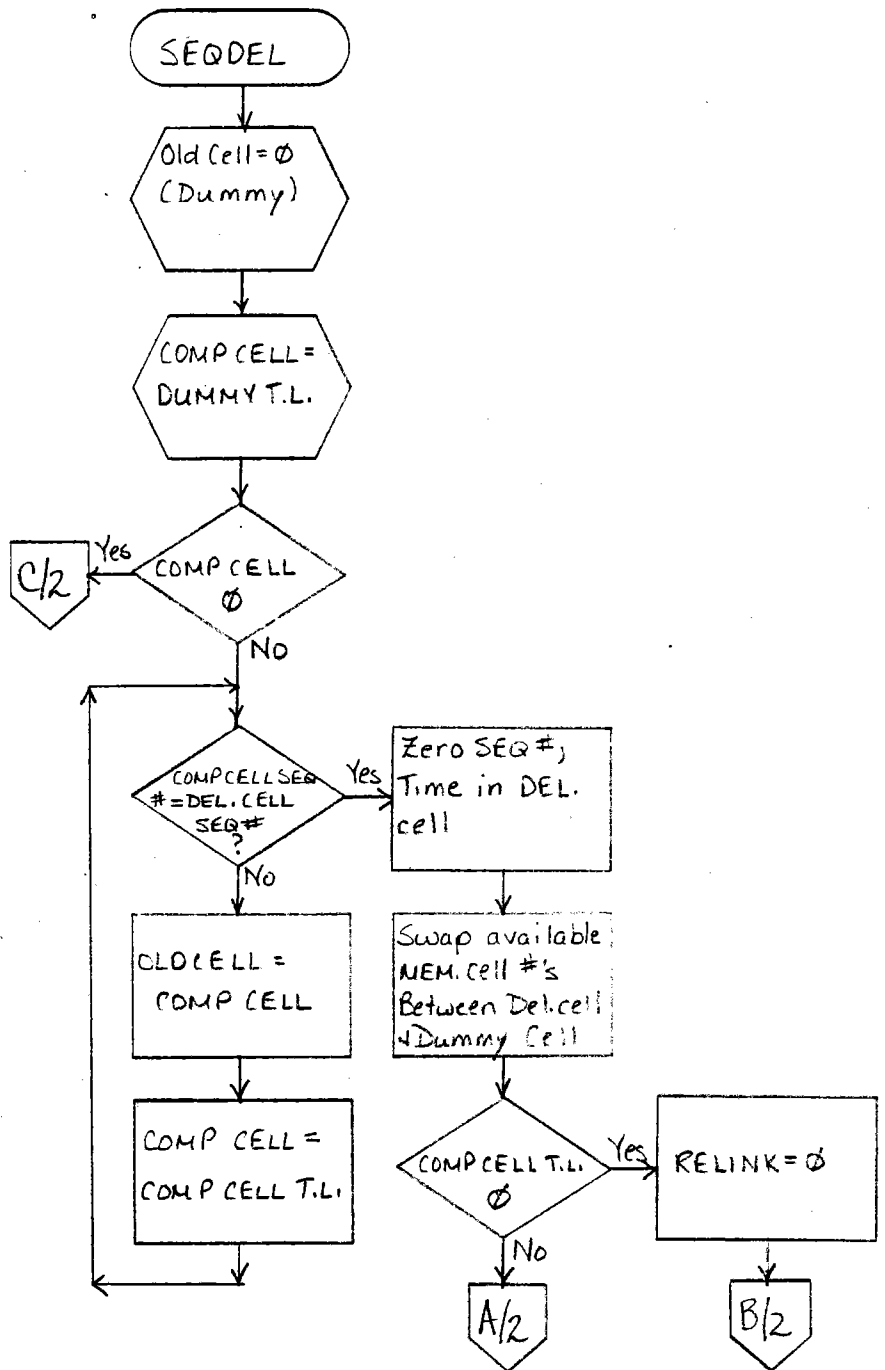


Figure 3.2.2-47 Flowchart - SEQDEL

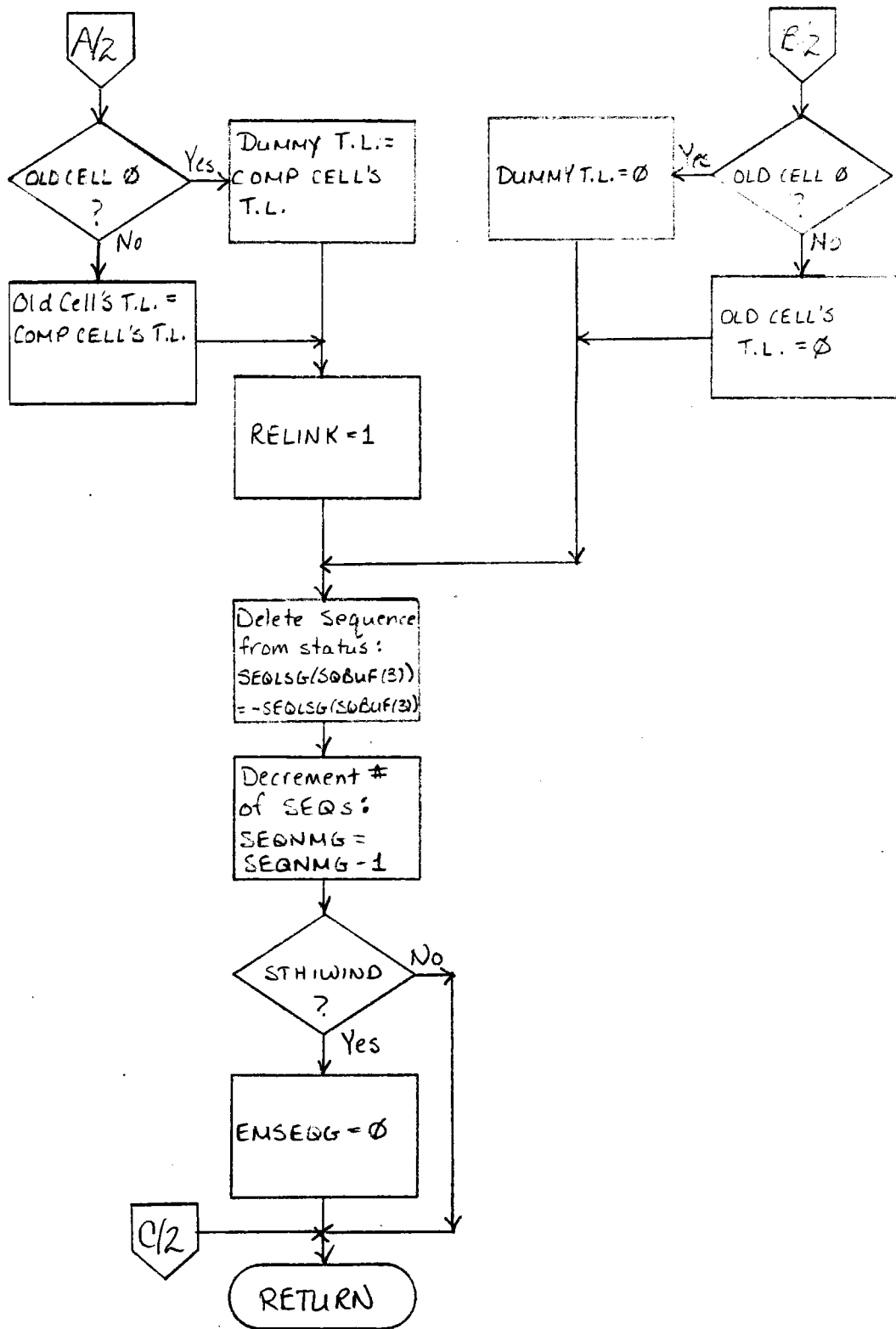


Figure 3.2.2-47 Flowchart - SEQDEL (continued)

- 3.2.2.4.1.31.3 Internal Data Description
Miscellaneous variables for temporary storage.
- 3.2.2.4.1.31.4 Flowcharts
See Figure 3.2.2-48 for the BHI flowchart.
- 3.2.2.4.1.32 Submodule XXXII - BHISH5
- 3.2.2.4.1.32.1 Description
- a. Language used - MODCOMP Classic Assembler
 - b. How invoked - called by the BHI task.
 - c. Constraints and limitations - input and output arrays are five-word buffers.
 - d. Processing - the BHISH5 routine takes a five-word input array (i.e. corridor coordinates) and shifts all ten bytes right, losing the original last byte. Then it adds the command byte at the first byte location and places the result in a five-word output array.
 - e. Error messages and recovery - None
- 3.2.2.4.1.32.2 Data, Logic and Command Paths
- Input data:
- a. FORTRAN-compatible input parameter five-word array (corridor coordinates).
 - b. Command to be inserted into array.
- Output data:
- FORTRAN-compatible output parameter is the five-word array consisting of the shifted combination of the two input parameters.
- 3.2.2.4.1.32.3 Internal Data Description
Seven-word register save area.
- 3.2.2.4.1.32.4 Flowchart
See Figure 3.2.2-49 for the BHISH5 flowchart.
- 3.2.2.4.1.33 Submodule XXXIII - BHC (Task)
- 3.2.2.4.1.33.1 Description
- a. Language used - FORTRAN IV
 - b. How invoked - invoked by FCP following output of commands to the HFCs.
 - c. Constraints and limitations - The task BHC is constrained to send one type of command to each HFC per second. BHC

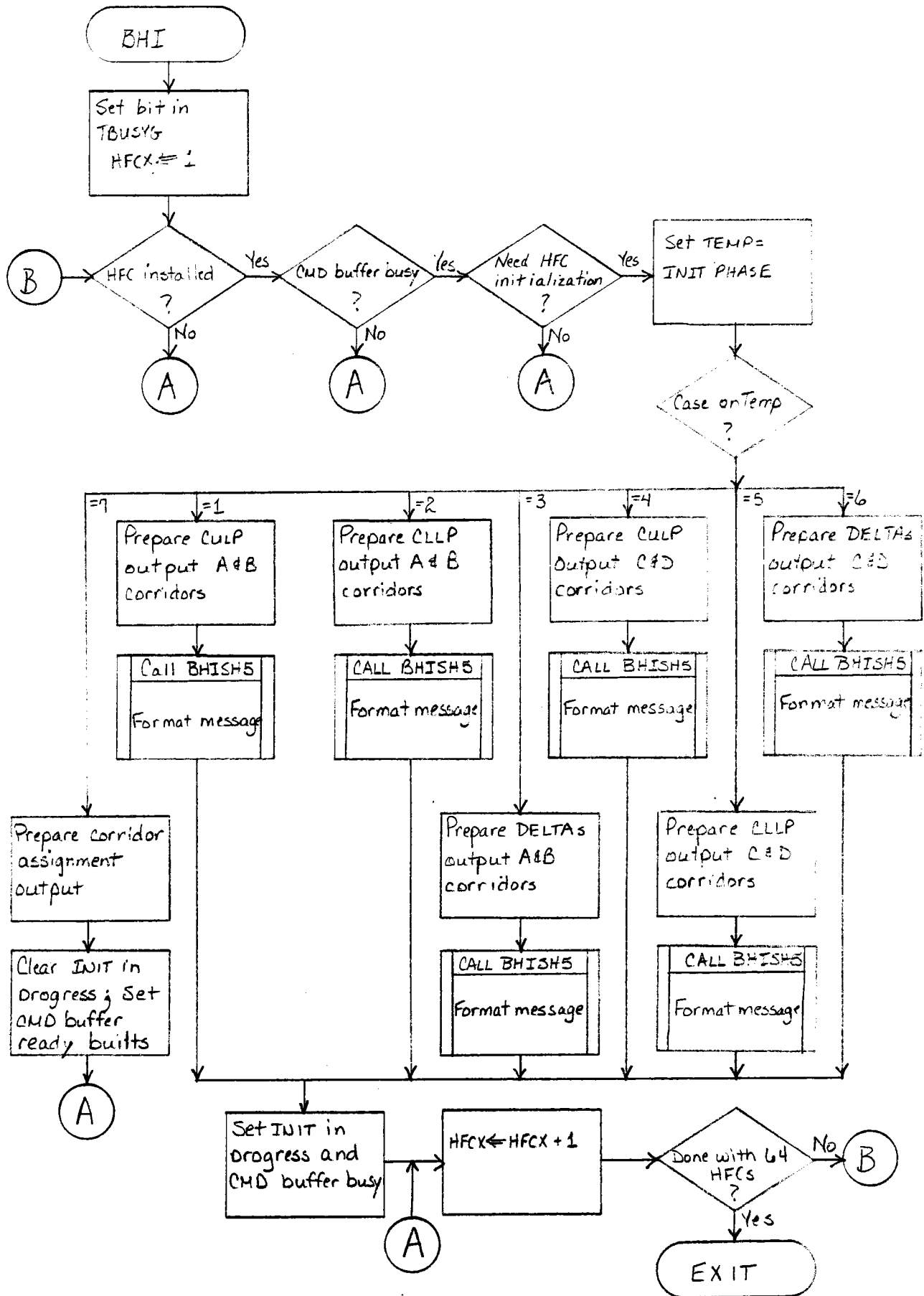


Figure 3.2.2-48 Flowchart - BHI

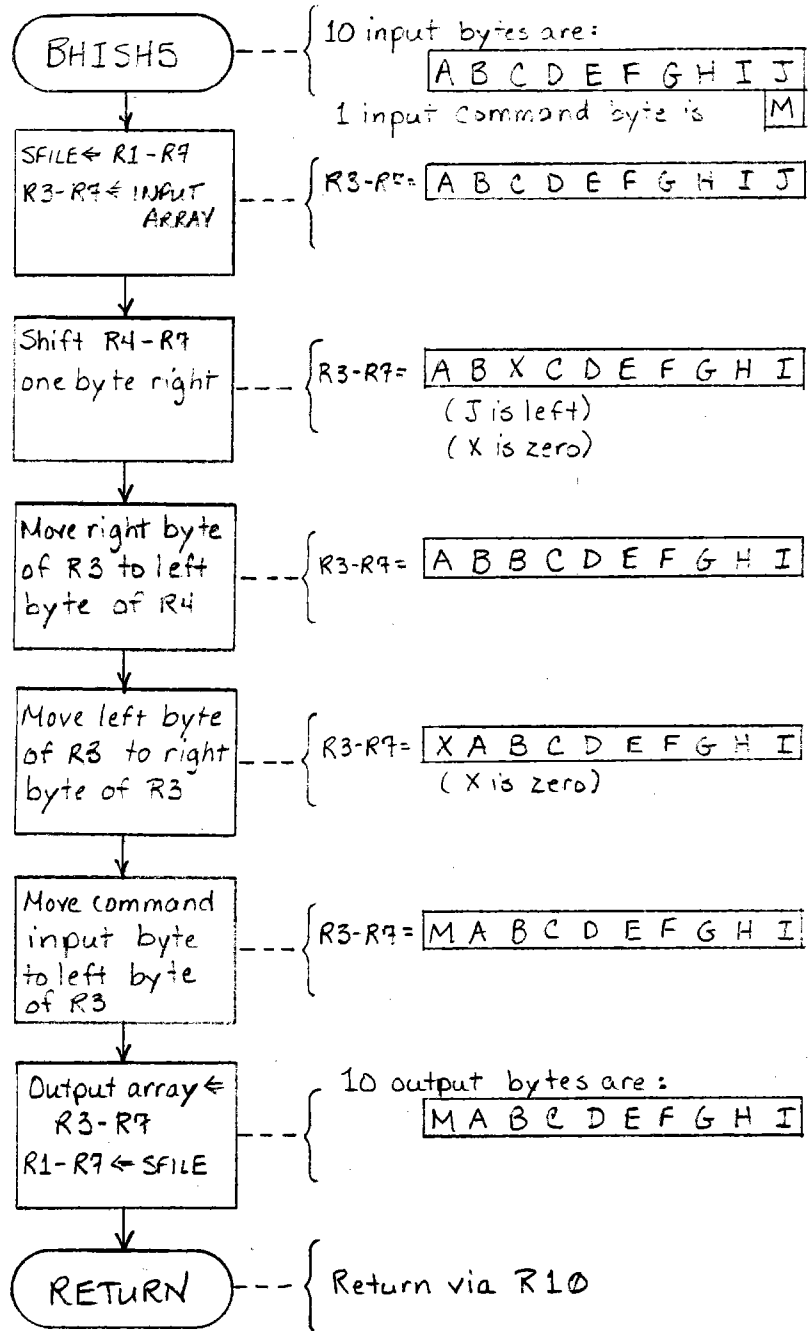


Figure 3.2.2-49 Flowchart - BHISH5

will not interrupt an HFC initialization sequence with HC commands until the sequence is finished.

- d. Processing - The BHC task, when activated, shall check to see if any HC commands are ready to be placed in the command buffer for FCP to output. Since only one command can be sent to an HFC, a priority scheme will be used by the BHC task when searching for a HC command to send out. The priority scheme used from high to low priority is as follows:

1. Critical-command bit set with beam-pointing command (Track CULP) or AZ/EL-pointing command (STOW).
2. Critical-command bit set with AZ/EL-pointing command (Directed Position) or HC initialization command.
3. Corridor-walk startup commands.
4. All other HC commands.

For each priority level the BHC task will search the HC command array in global common to determine if a command at that level is present. If a command is present the command buffer is filled with the appropriate command data.

To avoid loop processing when no critical commands are present, there will be two counters, one for each level of critical commands, in global common which are incremented when a critical command is to be output.

When BHC processes a critical command the appropriate counter will be decremented. If the counters are zero, then BHC will not search the HC command array for any critical commands. The BHC task also compresses any HC commands having the same coordinates into an HFC command (see Software/Firmware Functional Requirements Specification). (i.e. If identical HC commands are required for two or more HCs connected to the same HFC, the HC command mask shall be set to reflect this condition. After checking the HC command array, the BHC task will exit.

- e. Error messages and recovery - no error messages are produced by task BHC.

3.2.2.4.1.33.2 Data, Logic and Command Paths

Input data:

- a. HC-command status array - (HCCMDG)
- b. HC status words - (HCST1G, HCST2G, and HCST3G)
- c. Corridor-status flags - (CORRSG)
- d. HC command data (HCDATG)
- e. Current HC elevation - (ELEV G)
- f. Current HC azimuth - (AZIM G)

- g. HFC status - (HFCS2G)
- h. Emergency command counter - (EMCC1G)

Output data:

- a. HC-command status array - (HCCMDG)
- b. Command buffer - (CMDBFG)
- c. HFC status - (HFCS2G)
- d. Corridor-status flags - (CORRSG)
- e. Emergency command counter - (EMCC1G)

Algorithms:

Logic to calculate HC command masks and routine to move specific bytes.

3.2.2.4.1.33.3 Internal Data Description

Miscellaneous variables for temporary storage.

3.2.2.4.1.33.4 Flowchart

See Figure 3.2.2-50 for the BHC flowchart.

3.2.2.5 Interface Description

The software interfaces for module CMDPRC are depicted in the overview diagram (Figure 3.2.2-2). The message format for MMI-to-CMD input communication is contained in Tables 3.22-III and 3.22-IV (HCMAPG and CPPG arrays). The message format for BHC-to-FCP output communication is contained in Tables 3.7-I thru 3.7-IV and Tables 3.7-X thru 3.7-XII.

The hardware interface consists of the set of operational disk files specified in Section 3.2.2.3.2c.

3.2.2.6 Test Requirements

Due to the accelerated development of the MMI task, input commands can be entered at the HAC operator's console. Diagnostic tests of each task within this module are to be verified with output messages to the system console. A utility dump task is to be used to display global common values.

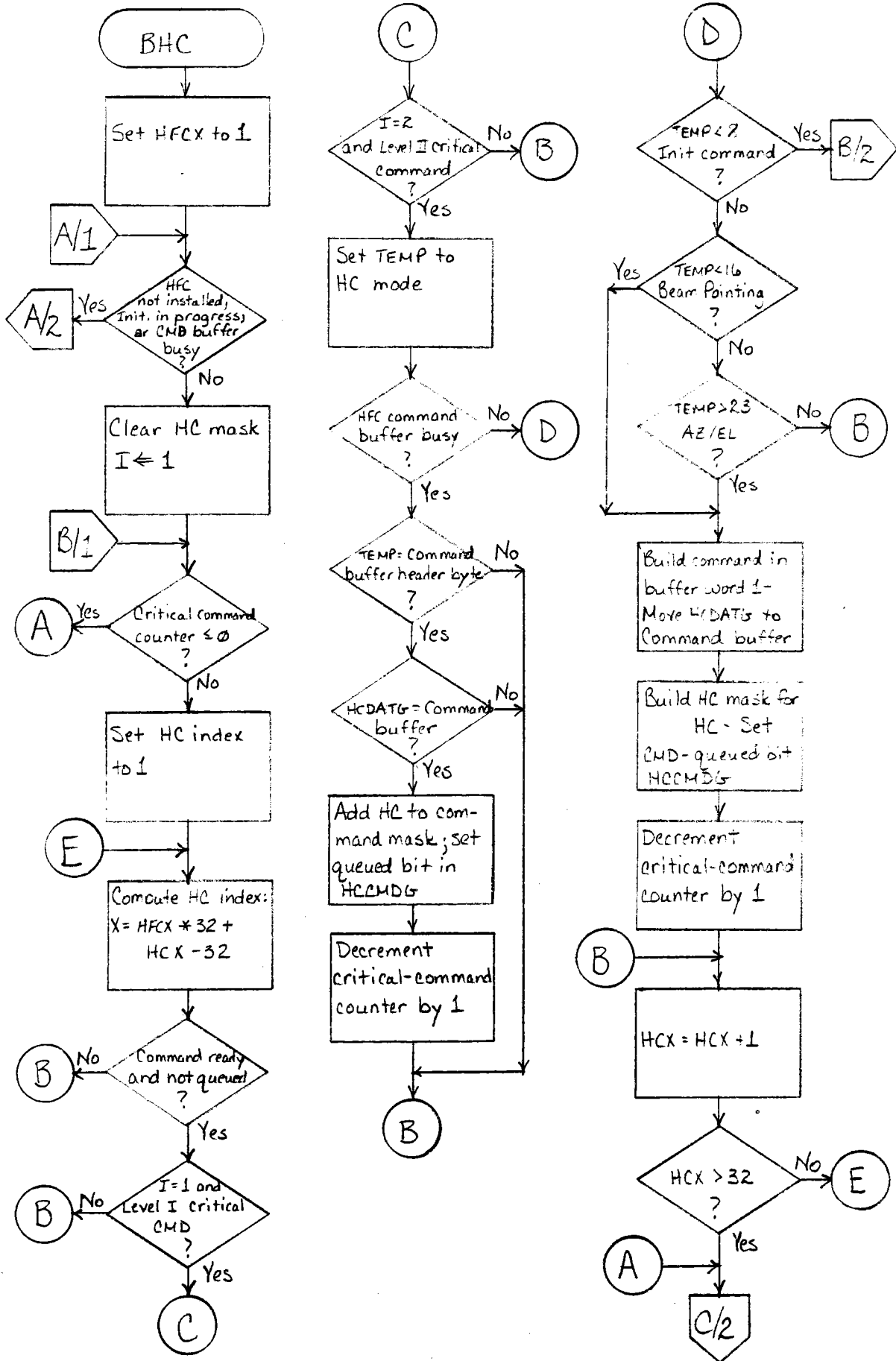


Figure 3.2.2-50 Flowchart - BHC

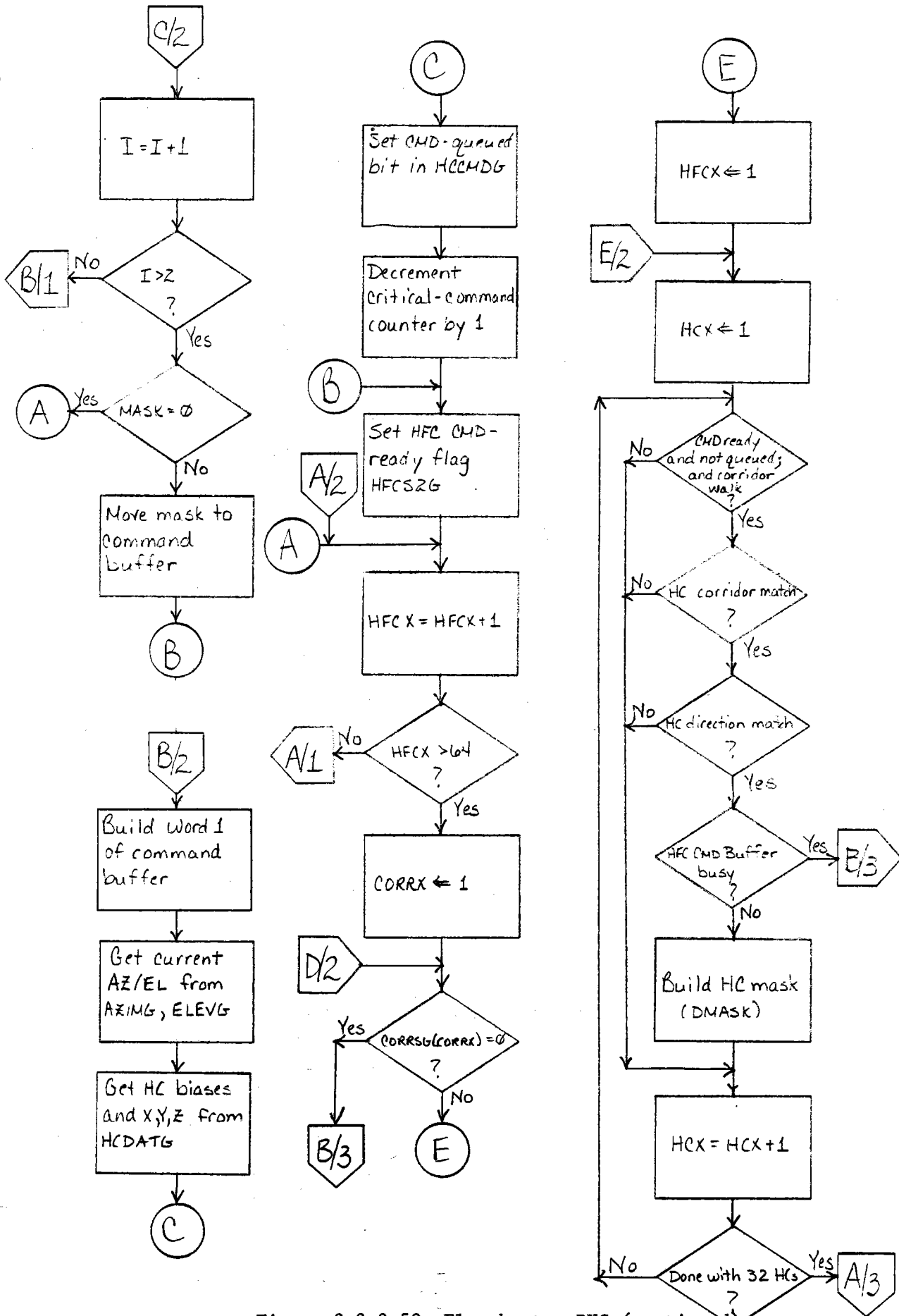


Figure 3.2.2-50 Flowchart - BHC (continued)

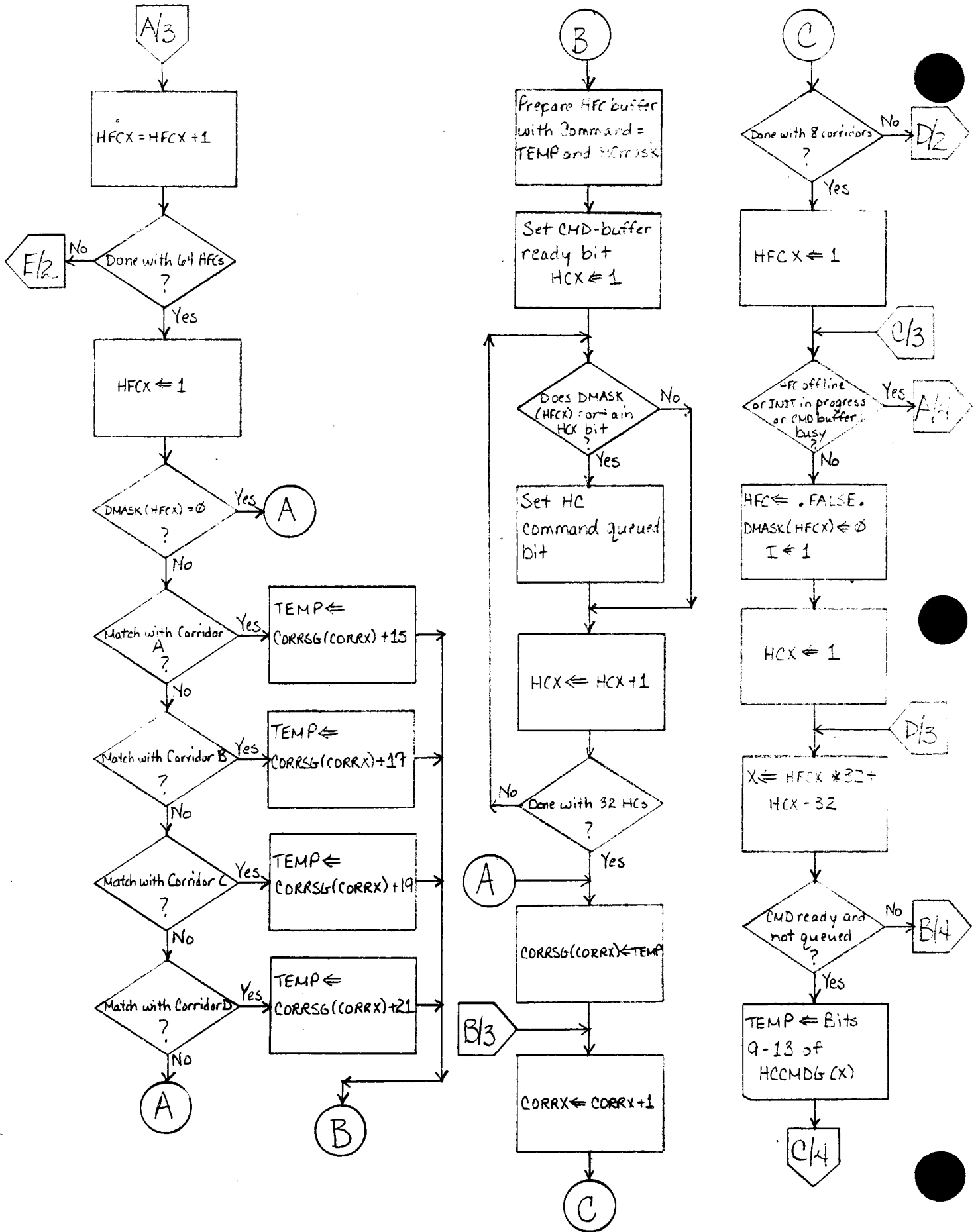


Figure 3.2.2-50 Flowchart - BHC (continued)

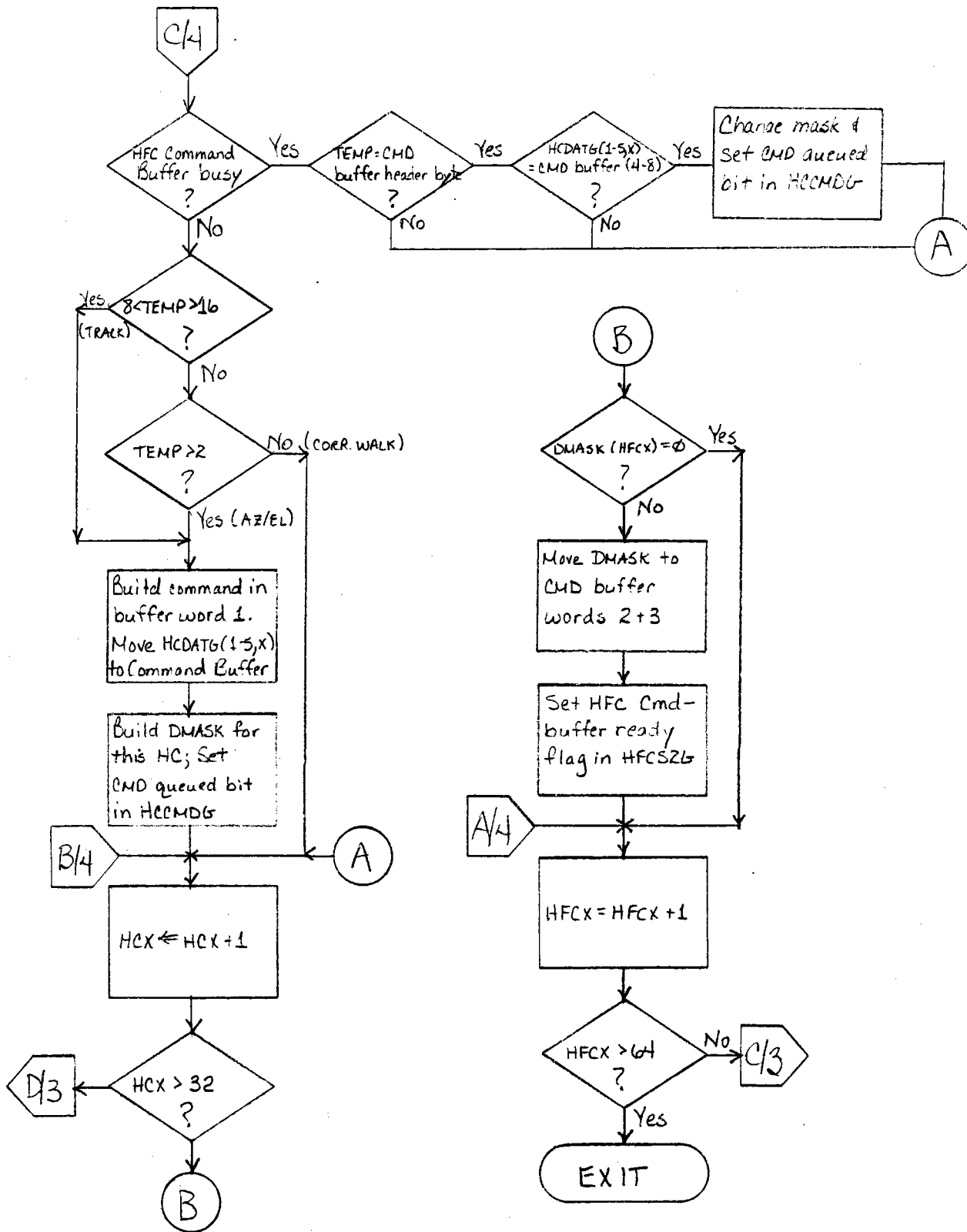


Figure 3.2.2-50 Flowchart - BHC (continued)

3.2.3 Sun Vector Module - SUNVEC

3.2.3.1 Purpose

The purpose of this module is to calculate a sun-position unit vector once per second, for transmission to the field of heliostats. The sun-position vector is used to calculate heliostat pointing angles to allow tracking of desired targets with the reflected beam.

3.2.3.2 Requirements

3.2.3.2.1 Design Requirements

The requirements for this software module are given in section 3.1 of the 10 MWe Collector Subsystem Software/Firmware Functional Requirements Specification. The requirements applicable to this module are:

- a. Control of up to 2048 heliostats in all modes required to operate the heliostats. The software system requires the calculation of sun position to be able to operate the heliostats in the Track mode; and
- b. Generate and transmit sun position information (once per second) to the entire field.

3.2.3.2.2 Derived Requirements

The derived requirements for this software module are given in section 3.2.1.3 of the 10 MWe Collector Subsystem Software/Firmware Functional Requirements Specification. These requirements are:

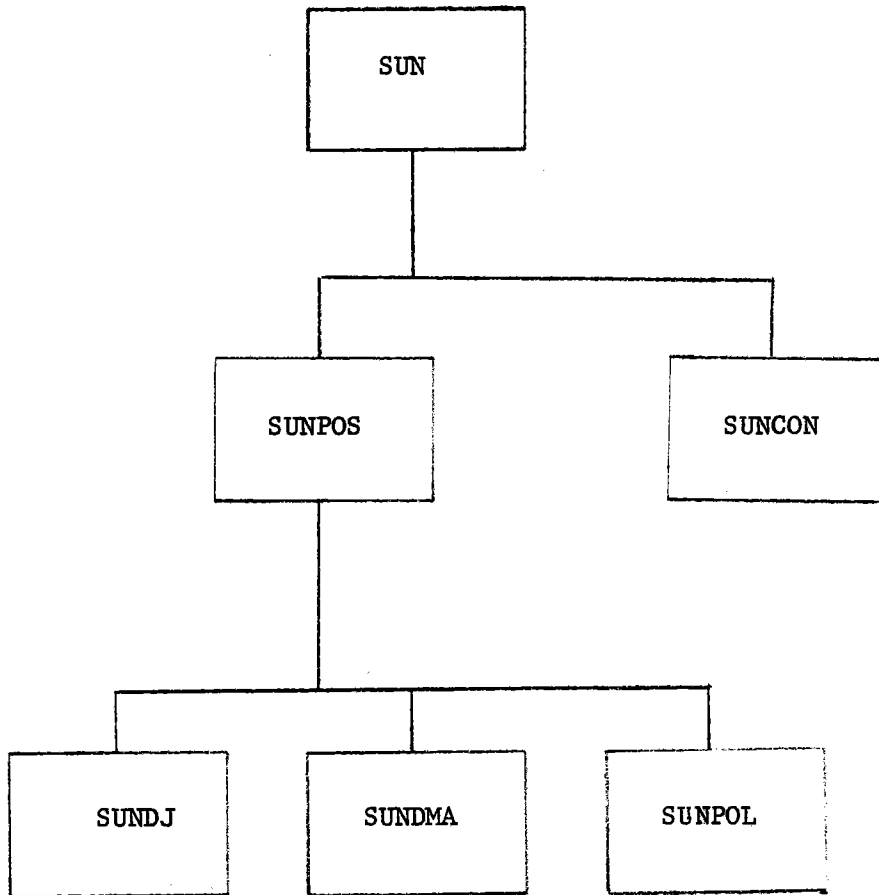
- a. Calculate a sun unit vector as a function of universal time, on a once-per-second basis; and
- b. Format the sun-position unit vector into the format required for message packets and store in the data base.

3.2.3.3 Design Approach

The SUNVEC module consists of one task (SUN) which in turn has five subroutines which it calls to perform required calculations. The various subroutines are functionally separated out for ease of coding, checkout, testing, and maintenance.

3.2.3.3.1 Functional Allocations

This module consists of six submodules. The names of these submodules and a brief description of the function performed by each is given below. (See figure 3.2.3-1)



SUNVEC MODULE

Figure 3.2.3-1

- a. SUN - the task submodule which initializes certain common variables used by other routines, maintains the internal month-day-year array, IDATE, converts coordinated-universal time from hour-minute-second format to seconds past midnight and biases time to calculate sun position, and calls routine SUNPOS to calculate sun-position vector. This task converts the double-precision, floating-point, sun vector into a double-precision, integer, sun vector, scaled at binary point four. The task then exits and awaits activation in the next one-second frame.
- b. SUNPOS - a submodule which calculates the sun-position unit vector in facility coordinates, corrected for atmospheric refraction, mean temperature, and mean pressure. The resulting sun vector is a double-precision, floating-point, three-element array.
- c. SUNDJ - a submodule whose function is to calculate the Julian days elapsed since January 1900, using the month-day-year array IDATE.
- d. SUNCON - a submodule whose function is to maintain the month-day-year array, IDATE, from the year since 1900 value (GTMEG(1)) and day-of-year (GTMEG(2)) value.
- e. SUNPOL - a submodule whose function is to calculate the value of a third-order polynomial.
- f. SUNDMA - a submodule which performs a specialized matrix multiplication.

3.2.3.3.2

Resource Budgets

This task will execute as a memory-resident task. The estimated resources are as follows:

- a. Memory required - 4000 words;
- b. Timing - less than one-half of one percent CPU;
- c. Disk files required - none;
- d. Disk accesses required - none;
- e. Magnetic tape access - none ; and
- f. Task priority - 110 (medium-high priority).

3.2.3.4

Design Description

3.2.3.4.1

Module Structure

This module consists of six submodules. The submodule SUN is a stand-alone task, consisting of a main routine which uses the other

five submodules as subroutines. Each of these submodules is described in detail in the following paragraphs.

3.2.3.4.1.1 Submodule I - SUN (Main Routine)

3.2.3.4.1.1.1 Description

- a. Language used - FORTRAN IV
- b. How invoked - activated by the task FCP once each time frame.
- c. Constraints and limitations - none
- d. Processing - The first section of code initializes constants, computes geocentric latitude, and calls SUNCON to update the IDATE array. This section is executed conditionally where the parameter IDAY has changed and, by default, when the system is initialized. The geodetic latitude is obtained from global common and converted to geocentric latitude with an equation that accounts for oblateness effects up to the third order. Time is updated by setting the parameter IDAY to the current day of the year and the third element of the IDATE array to the year value. Submodule SUNCON is called to calculate the month and date from these values, which are stored in the IDATE array. After this update is complete, the universal time parameter (UT) is calculated (seconds since midnight) from the hours-minutes-seconds values of coordinated-universal time maintained in the data base. Submodule SUNPOS is called to calculate the double-precision, floating-point, sun-position, unit vector, stored in the one by three SVEC array. This main routine then converts the array to a double-precision, integer array scaled at binary point four. The main routine then releases processor control until activation in the next "frame" by task FCP.
- e. Error messages and recovery - none.

3.2.3.4.1.1.2 Data, Logic and Command Paths

The input data to this submodule is in the data base. The same is true of the output data.

Input Data:

- GTIMEG(1) - Current year, since 1900 (e.g. 1980: 80)
- GTIMEG(2) - Current day of year
- GTIMEG(3) - Current coordinated-universal time since midnight in hours
- GTIMEG(4) - Current coordinated-universal time in minutes past the hour
- GTIMEG(5) - Current coordinated-universal time in seconds past the minute

Output Data:

- SUNPOG(1) - X-component of sun-position unit vector, in facility coordinates
- SUNPOG(2) - Y-component of sun-position unit vector, in facility coordinates
- SUNPOG(3) - Z-component of sun-position unit vector, in facility coordinates

The only algorithm used in this routine is the conversion from double-precision, floating-point values to double-precision, integer values scaled at binary point four. Since integer values normally have the binary point after the least significant bit, the way to scale the number (a 32-bit number) is to shift the binary point left 28 places. Thus, the floating point value is multiplied by two raised to the 28th power, shifting the binary point, and the result is integerized.

3.2.3.4.1.1.3

Internal Data Description

The following is a description of the data internal to this routine:

- IDATE(1) - Current month of year in coordinated-universal time
- IDATE(2) - Current day of month in coordinated-universal time
- IDATE(3) - Current number of years since 1900 in coordinated-universal time
- IDAY - Current day of year in coordinated-universal time
- DTOR - constant, degree-to-radians conversion
- PI - constant, value of number of radians in circle
- RTOD - constant, radians-to-degrees conversion
- SLAT - site latitude, radians
- SLONG - site longitude, radians
- TO28 - Constant, 2 to the 28th power value
- UT - time since midnight, seconds, coordinated-universal time

3.2.3.4.1.1.4

Flowcharts

See figure 3.2.3-2.

3.2.3.4.1.2

Submodule II - SUNPOS

3.2.3.4.1.2.1

Description

- a. Language used - FORTRAN IV
- b. How invoked - called by task SUN
- c. Constraints and limitations - none

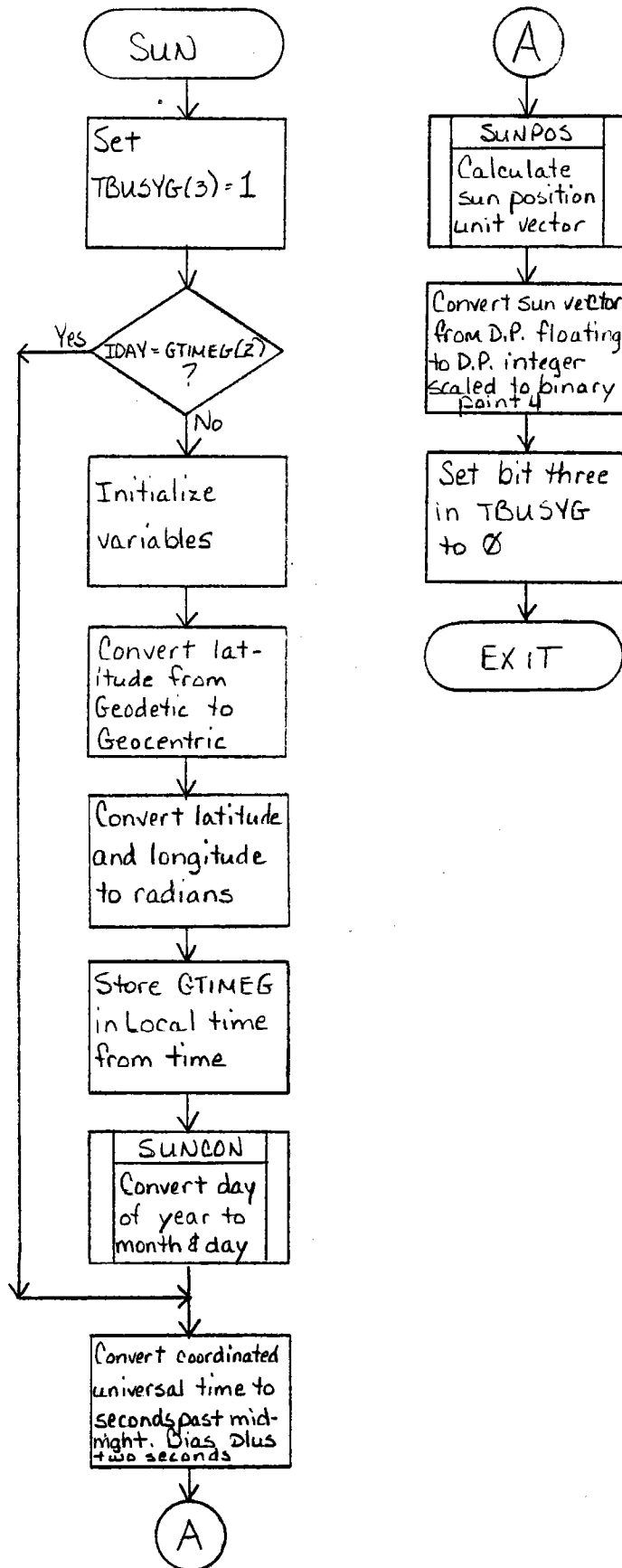


Figure 3.2.3-2 Flowchart - SUN

d. Processing - When called by SUN, the first step in the submodule is to call submodule SUNDJ to calculate the number of days elapsed since January 1900, using the month-day-year array IDATE. From this, the subroutine calculates the Julian centuries elapsed since January 1900 (DJC). The correction factor due to mean temperature and pressure is calculated as a function of the current month. Next, the ephemeris time (ET) is calculated. This is then calculated as Julian centuries (TOD). Next, a factor is subtracted to account for the light-time from sun to earth. If the current date has changed, the equatorial-to-facility transformation (TEF) is recalculated. Otherwise, the calculation is skipped. The mean obliquity (OBL2) is calculated as a function of time since 1900. The mean longitude of perihelion is calculated as a function of time since 1900 (GAMA). The earth orbit eccentricity is calculated as a function of time since 1900 (ECEN). Then, the mean anomaly of the earth in orbit about the sun (ANOM) is calculated as a function of time since 1900. Using the mean anomaly as a first guess, the eccentric anomaly is calculated, using the Newton-Raphson iterative process. Then the true anomaly (TANOM) is calculated as a function of eccentricity and eccentric anomaly. The sun-position vector in earth-centered, inertial coordinate system is calculated as a function of obliquity, mean longitude of perihelion and true anomaly. Next, the right ascension of the Greenwich meridian (RAM) is calculated. This is used to convert the sun position from earth-centered, inertial coordinates to equatorial meridian coordinates (SVC). In order to correct for atmospheric refraction, the true azimuth (TZA) is calculated as a function of sun position. Note that if the sun elevation angle is less than zero, this is the apparent azimuth angle (AZA). Two separate methods are used to calculate the apparent azimuth angle, if the elevation angle is positive. One method is used for angles between zero and five degrees, and another is used for angles greater than five degrees. The correction factor for apparent azimuth angle is calculated and combined with the previously calculated correction for mean temperature and humidity. Using this apparent azimuth angle, the sun position vector in facility coordinates is calculated and normalized (SVE).

e. Error messages and recovery - none

3.2.3.4.1.2.2

Data, Logic and Command Paths

This submodule uses input data described below:

UT - seconds since midnight, coordinated-universal time
 IDATE(1) - month of year, coordinated-universal time

IDATE(2) - day of month, coordinated-universal time
IDATE(3) - elapsed years since 1900, coordinated-universal time

This submodule generates the following double-precision, floating-point, three-element array as output:

SVEC(1) - X-component of sun position in facility coordinates
SVEC(2) - Y-component of sun position in facility coordinates
SVEC(3) - Z-component of sun position in facility coordinates

The algorithms used in this submodule are a second or third-order polynomial curve fit of certain parameters as a function of time and the Newton-Raphson iterative process for calculation of the implicit function eccentric anomaly.

3.2.3.4.1.2.3

Internal Data Description

Internal data used in this submodule is as follows:

ALTFAC - Correction factor for mean temperature and pressure
ANOM - Mean anomaly of earth in orbit around sun, radians
AZA - Complement of sun's apparent elevation (with refraction), radians
CAT - Normalization factor for final sun vector (with refraction)
DJC - Julian centuries since January 1, 1900 epoch
DPC - constant, days per Julian century
DTOR - constant, degrees-to-radians conversion
EANOM - Eccentric anomaly of earth orbit, radians
ECEN - Earth orbit eccentricity
ET - Ephemeris time, seconds since midnight
GAMA - Mean longitude of perihelion of earth's orbit, radians
DBLQ - Mean obliquity of ecliptic, radians
R - Atmospheric refraction correction factor
RA - R corrected with ALTFAC
RAD - Distance from sun to earth, astronomical units
RTOD - Constant, radians-to-degrees conversion
SLAT - Site latitude, radians
SLONG - Site longitude, radians
SOL - Speed of light, astronomical units per second
SPD - Constant, seconds per day
SVC - Sun vector in facility coordinates - no refraction correction
SVE - Sun vector in equatorial Greenwich meridian coordinates
SVI - Sun vector in earth centered inertial coordinates
TANOM - True anomaly of earth in orbit, radians
TEF - Coordinate conversion matrix from equatorial to facility coordinates
TOD - Julian centuries elapsed since midnight
TZA - 90 degree minus sun true elevation (without refraction), radians

3.2.3.4.1.2.4

Flowcharts

See figure 3.2.3-3.

3.2.3.4.1.3

Submodule III - SUNDJ

3.2.3.4.1.3.1

Description

- a. Language used - FORTRAN IV
- b. How invoked - called by SUNPOS
- c. Constraints and limitations - none
- d. Processing - The submodule, when called by SUNPOS, calculates the day of the year from current month and date. It then checks for current year being a leap year, and if so, and the month is not January or February, a day is added to correct for the leap day. Then the integer number of Julian days is calculated for years elapsed since January 1900 epoch to this year, and the days elapsed in this year are added to get total Julian days elapsed since the January 1900 epoch.
- e. Error messages and recovery - none

3.2.3.4.1.3.2

Data, Logic and Command Paths

The input data to this routine are as follows:

- IDAYT(1) - Month of current year
- IDAYT(2) - day of current month
- IDAYT(3) - elapsed years since January 1900 epoch

The output data generated by this submodule is:

- DJ1900 - number of Julian days elapsed since January 1900 epoch

3.2.3.4.1.3.3

Internal Data Description

A twelve element array is used in this submodule, initialized to values equal to numbers of days in each of twelve months.

3.2.3.4.1.3.4

Flowcharts

See figure 3.2.3-4.

3.2.3.4.1.4

Submodule IV - SUNCON

3.2.3.4.1.4.1

Description

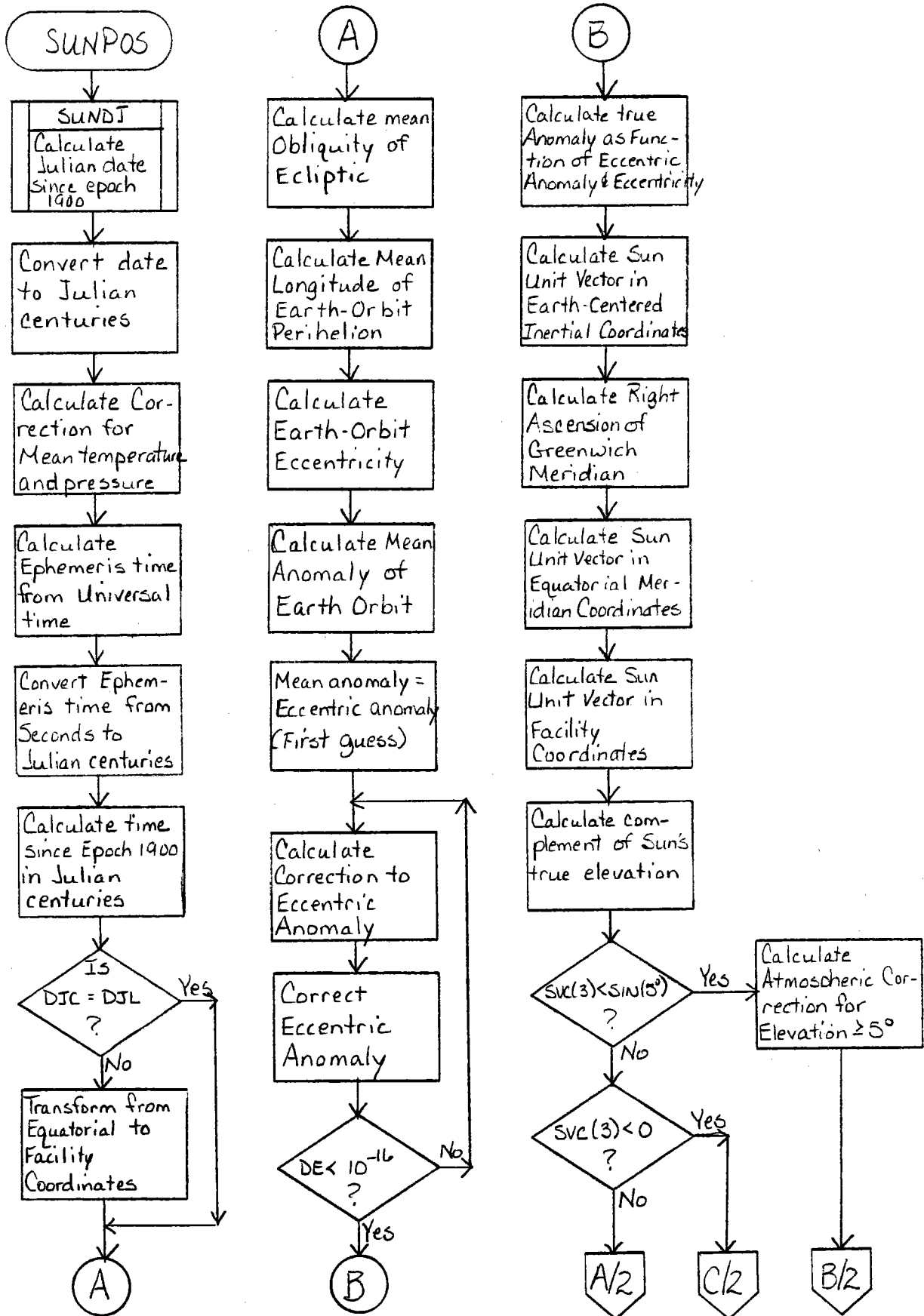


Figure 3.2.3-3 Flowchart - SUNPOS

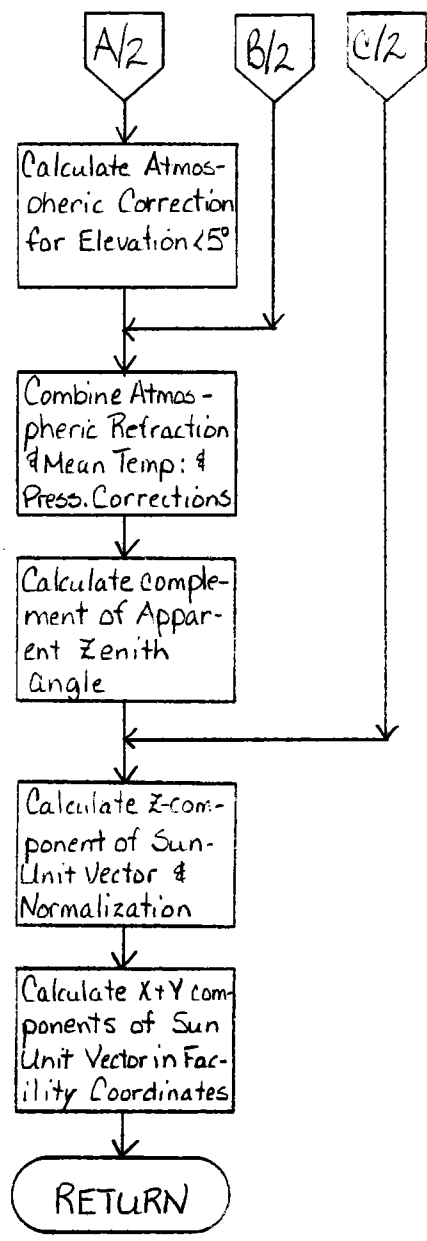


Figure 3.2.3-3 (cont.) Flowchart - SUNPOS

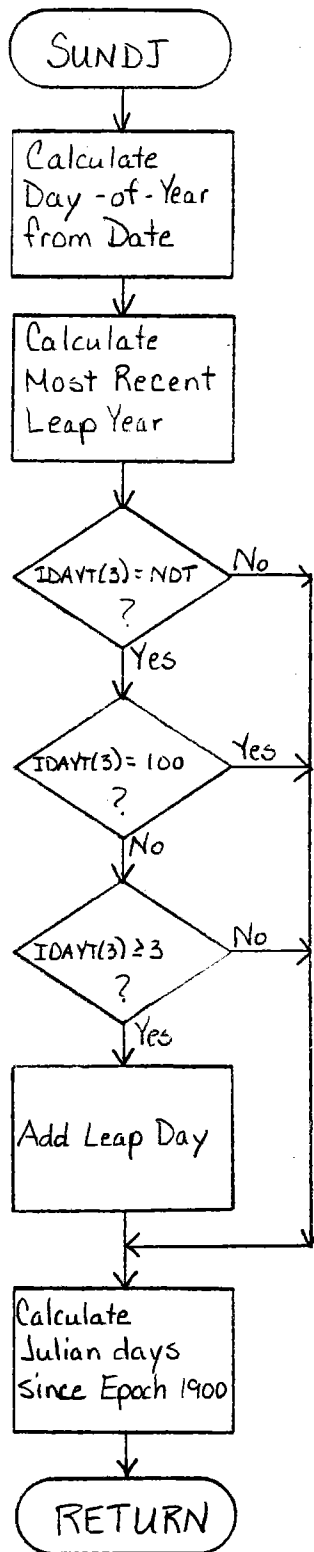


Figure 3.2.3-4 Flowchart - SUNDJ

- a. Language used - FORTRAN IV
- b. How invoked - called by task SUN
- c. Constraints and limitations - none
- d. Processing - This submodule determines the month and date from a day-of-year input. To do this, an internal, twelve-element, array is used with the number of days in each month as elements. The subroutine first updates the February value, depending on whether the current year is leap year. Then it loops through, determining whether pointer points to current month, and subtracting days of month from day-of-year value until month is found. The remainder is the day of month.
- e. Error messages and recovery - none

3.2.3.4.1.4.2 Data, Logic and Command Paths

The input parameters to this routine are as follows:

IDAY - day-of-year, coordinated-universal time
 IDATE(3) - elapsed years since January 1900 epoch

The output parameters generated by this routine are:

IDATE(1) - current month of year
 IDATE(2) - current day of month

3.2.3.4.1.4.3 Internal Data Description

The internal data used in this submodule is a twelve element array (NDATE), with values of days contained in each month.

3.2.3.4.1.4.4 Flowcharts

See figure 3.2.3-5

3.2.3.4.1.5 Submodule V - SUNPOL

3.2.3.4.1.5.1 Description

- a. Language used - FORTRAN IV
- b. How invoked - called by the SUNPOS
- c. Constraints and limitations - none
- d. Processing - SUNPOL is a submodule which calculates the value of a third-order polynomial.
- e. Error messages and recovery - none

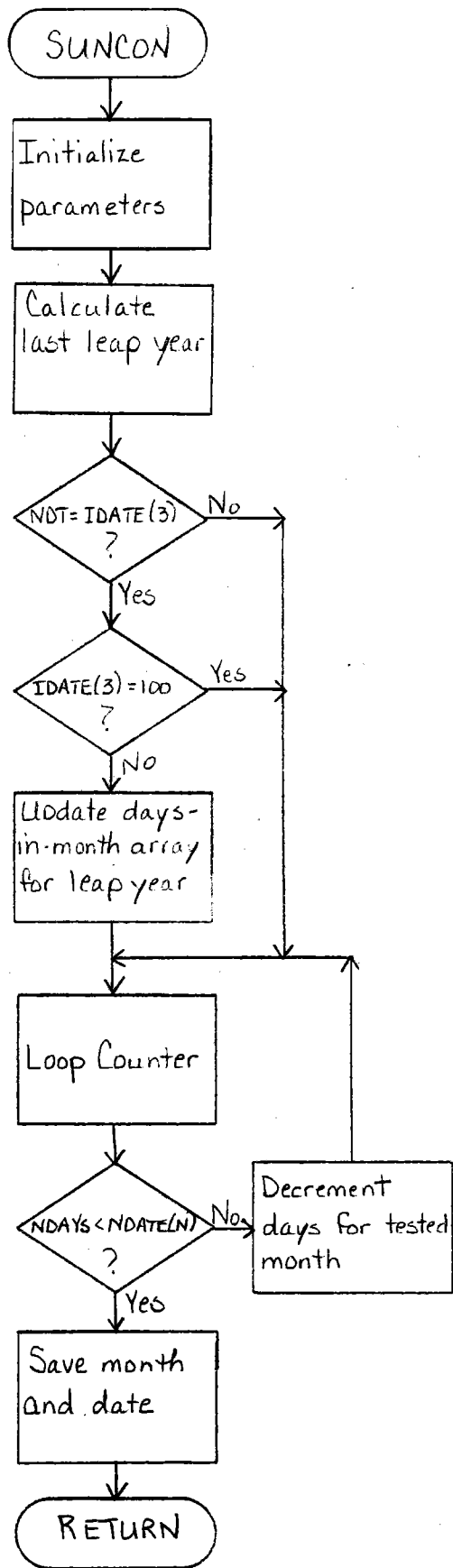


Figure 3.2.3-5 Flowchart - SUNCON

3.2.3.4.1.5.2 Data, Logic and Command Paths

The input parameters to this routine are as follows:

TU - Independent variable
A(1) - A(4) - Polynomial coefficients

The output parameter generated by this routine is:

POLY3 - Resultant value of polynomial expression

3.2.3.4.1.5.3 Internal Data Description

There is no data internal to this submodule.

3.2.3.4.1.5.4 Flowcharts

See figure 3.2.3-6.

3.2.3.4.1.6 Submodule VI - SUNDMA

3.2.3.4.1.6.1 Description

- a. Language used - FORTRAN IV
- b. How invoked - called by subroutine SUNPOS
- c. Constraints and limitations - none
- d. Processing - SUNDMA is a submodule which is used to perform a specialized, double-precision, matrix multiplication. The submodule is called from SUNPOS and performs a double-precision, matrix multiplication of the type:

$$A * B = C$$

Where A is a three-by-three matrix and C is the resultant three-by-one matrix. SUNPOS uses SUNDMA to transform inertial coordinates to facility coordinates.

- e. Error messages and recovery - none

3.2.3.4.1.6.2 Data, Logic and Command Paths

The input parameters are as follows:

A - generalized three-by-three matrix
B - generalized three-by-one matrix

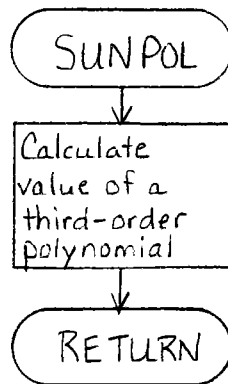


Figure 3.2.3-6 Flowchart - SUNPOL

The output parameter is:

C - Generalized three-by-one matrix

3.2.3.4.1.6.3 Internal Data Description

There is no data internal to this submodule.

3.2.3.4.1.6.4 Flowcharts

See figure 3.2.3-7.

3.2.3.5 Interface Description

This module interfaces with the other modules of the Collector Subsystem software through its input data (coordinated-universal time in the data base), its output data (sun position vector in the data base), and its calling sequency (activated by task FCP).

3.2.3.6 Test Requirements

This module can be tested in a stand-alone manner by inserting known, fixed values of universal time in the data base, activating the task, dumping the resulting values of sun position from the data base after the program is complete, and comparing the results to precalculated values.

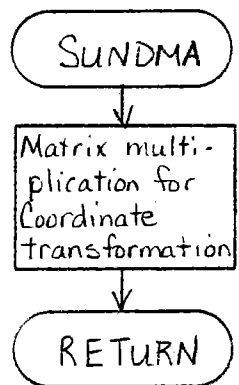


Figure 3.2.3-7 Flowchart - SUNDMA

3.2.4 Field Communications Processor Module - FLDCOM

3.2.4.1 Purpose

The Field Communications Processor provides communications to the field of 2048 heliostats. As such it provides the interface for the Command Processor Module to the field via eight communication lines each with eight HFCs and the interfaces for the Alarm, Backup, and Status Display Modules. All interfaces are via the global data base.

The Field Communications Processor Module consists of one task (FCP) operating at the third highest priority in the system which, in turn, utilizes seven submodules (FCPOUT for output checking, FCPIN for input checking and HFC status updates, FCPUPD for updating HC status, FCPCHO for byte checksum calculations with checksum storage into output buffers, FCPCHI for checksum calculations without storage and FCPSWH for recording I/O errors, detecting communications failure, and FCPFLP for performing line communications switchover).

3.2.4.2 Requirements

3.2.4.2.1 Design Requirements

Section 3.2.1.4 of the 10 MWe Software/Firmware Functional Requirements Specification states the following requirements for the FLDCOM module:

- a. Synchronize field operations by transmitting to the HFC computer a sun vector command once per frame;
- b. Generate, at the proper time in the time frame, the polling command for the HFC;
- c. Receive the HC and HFC status in response to each polling command and save same in the data base;
- d. Transmit to the HFC, the operational commands generated by the Command Processor Module;
- e. Detect communications errors and automatically switch over to the redundant communications lines or the "Backup" computer, and report these conditions to the Alarm Processor Module;
- f. Mark (calibrate) heliostats in both azimuth and elevation;
- g. Generate proper timing to accomplish all of the above in conjunction with HFC/HC firmware;
- h. Implement retransmission of HC commands (tracking and AZ/EL only); and
- i. Prevent communications failover upon detection of field power-loss signal.

3.2.4.2.2

Derived Requirements

Based upon the design requirements stated in paragraph 3.2.4.2.1, the following are derived requirements for the FLDCOM module:

- a. The FLDCOM module to insure proper timing within the one-second time frame, will use a delay timer;
- b. To detect the occurrence of a communications error, the FLDCOM module will examine the user file tables (UFTs) and the reported HFC and HC status messages. Communication errors will be indicated to the ALARM module via global common;
- c. In order to receive the HC and HFC Status, the FLDCOM module will accept input from the HFC and update the HFC/HC Status in global common;
- d. Automatically switching to the redundant communications lines implies the changing of the logical device names present in the user file tables;
- e. Requirement "a" of Section 3.2.4.2.1 implies an interface with the SUNVEC module;
- f. Requirements "a," "b," and "d" of Section 3.2.4.2.1 imply that the FLDCOM module has the ability to output data and commands to the HFCs; and
- g. Requirement "e" of Section 3.2.4.2.1 implies that the FLDCOM module has an interface with the BACKUP module.

3.2.4.3

Design Approach

The following is a list of assumptions used in the design of the FLDCOM module:

- a. It is assumed that the equipment configuration defined in the phase II proposal, Section 3, will be used as the operational equipment. This implies a primary and backup HAC each with dual communication lines to the field of heliostats;
- b. There is no requirement to provide the operator with the capability to manually switch the communication lines from normal to alternate or from alternate to normal;
- c. The field of heliostats can be powered up in either sections or in its entirety. The field can be powered up either before or after the HAC is operational;

- d. If all of the field of heliostats loses power, an interrupt will be supplied to both the primary and backup HAC to indicate the loss of power;
- e. The current communication line and field status in the primary HAC shall be transmitted to the backup HAC on a periodic basis once each time frame;
- f. Errors can occur anywhere along the communication lines to the HFCs, and error indications received at the HAC are of minimal value for diagnostic purposes. Therefore, I/O errors indicated by the MAX IV (MAXNET) operating system, in the user file tables (UFTs) which the FLDCOM module uses to do I/O, shall be considered line errors; and
- g. There is no requirement to recover from a failure to the alternate or secondary communications lines.

For purposes of this design discussion various terminology shall be used and is defined below.

- a. Communication errors - Errors can occur at various levels in the communication hierarchy: field, line, HFC, and HC. The definition of these errors follows:
 - 1. Field error - The only error recognized at the field level shall be loss of communication to entire field. This error will occur when all communications to the field is lost and no power loss interrupt has been detected.
 - 2. Line error - Line errors shall be detected by examination of the input and output UFTs which FLDCOM uses to perform I/O to the field.
 - 3. HFC, HC error - These errors are indicated by the status messages and UFT status received by FLDCOM from the field. The UFT status shall indicate input/output time-out conditions. HFC/HAC communications failure and HC command errors shall be detected by examination of the status response message. These HFC/HC errors do not differ from phase I and will be flagged for the ALARM module for operator display.
- b. Communication status - Since communication with the field is of a hierarchical nature and I/O errors can occur at the various levels, status shall be kept at the field, line, HFC, and HC levels. Definitions of the status values follow below:

1. Field status - The status of the field shall be set to one of the following values:
 - a) ACTIVE - initial state prior to establishing communications with the field; sun/synchs and status poll messages are being sent, but the field has not yet responded;
 - b) ENABLED - communications have been established with the HFCs in the field, (i.e. a status response message has been received from an HFC); and
 - c) There shall also be bits in the field status which indicate if a communication failure or power loss interrupt has occurred.

 2. Line status - The status of the lines shall be one of the following values:
 - a) IDLE - not currently being used to communicate with the field (initial value for the alternate communication lines);
 - b) ACTIVE - line being used to attempt establishment of communications with HFCs;
 - c) ENABLED - I/O to the field is currently active on this line; and
 - d) FAILED - failure criterion for this line has been met and the line has been declared failed and is not to be used for communications with the field.

 3. HFC, HC status - Is described in the data base section (HFCSIG, HFC2G, HCST1G, HCST2G).
- c. Normal/Alternate line - For the purposes of this design discussion, the term "normal line" shall refer to the line initially used to perform I/O to the field. The eight normal lines shall all reside in the same MODCOMP 1930 Universal Communication Chassis. There are four MODCOMP 1931 dual asynchronous interfaces in each 1930, each with two communication lines. The term "alternate line" will refer to the communication lines resident in the second MODCOMP 1930 Universal Communication Chassis and shall serve as the backup communication line to the field.

The following is a discussion of the design present in the FLDCOM module.

When the HAC is initialized, the field and eight normal lines shall be in the active state. The FLDCOM module will send sun/synch and status poll messages to the field using the eight normal communication lines. If the FLDCOM module is executing in the backup HAC, a check shall be made to determine why the backup HAC is being used. If a communication failure occurred to the entire field, FCP shall switch to alternate lines. Otherwise the lines that were in use by the primary HAC will be used by the backup HAC.

Until the HFCs respond with a status response message, the field and lines will be considered in the active state.

Once a status response message is received, the status of the field shall be set to enabled.

As HFCs respond, the lines they are connected to will be set to the enabled state. Communications will then take place with the field normally in the synchronous loop until the occurrence of an I/O error. As I/O errors occur, they may cause FLDCOM to initiate a switch to the alternate communication lines or to the backup HAC. A discussion of these conditions follows:

- a. Line Switchover - A switch to the alternate lines shall be made if one of the following conditions is true:
 1. Three consecutive input errors on a line.
(Error present in the input UFT for that line.)
 2. Three consecutive output errors on the line.
(Error present in output UFT.)
 3. Three consecutive errors (either input or output or both) on the line.
 4. If communication is lost with two or more HFCs on a line (HFC input time-out error or output time-out error).

When one of the above conditions is true the FLDCOM module shall mark the normal line as failed and initiate a switchover to the alternate line. Since a MODCOMP 1931 has two lines, both lines residing in the 1931, both will be switched to the alternate lines. The alternate line status shall be changed from idle to active and the sun/synch and status poll messages shall be transmitted on the alternate lines. When a status response is received by the HAC, the alternate lines shall be set to the enabled state. A line failure on an alternate line shall result in an alarm to the operator, however, I/O to the field shall remain active on the alternate line.

- b. Backup HAC Switchover - When the FLDCOM module detects a loss of communications with all enabled lines in the field, it shall initiate a switch to the backup HAC. The backup HAC shall initiate communications with the field on the alternate lines. If the backup HAC is not active, an alarm shall be raised and the FLDCOM module shall attempt to reestablish communications with the field on the alternate lines.

The switch to the backup HAC shall be indicated by the FLDCOM module setting a bit in global common which the SWI module shall detect. Upon detection, the SWI module shall initiate an orderly transition to the backup computer.

If a loss of communication is detected and a power loss interrupt has occurred, no switchover to the backup HAC shall be initiated.

3.2.4.3.1 Functional Allocations

To accomplish the required processing the FLDCOM module consists of a single task called FCP. The task FCP consists of four functional parts and utilizes seven submodules. The four functional parts of the FCP task perform the following functions:

- a. Sun/synch preparation, output, output check, and activation of the SUNVEC task, which calculates the sun vector for the next second during FCP's idle time;
- b. Status request loop (DO-UNTIL eight lines of eight HFCs each are done) for one-eighth of the field, with output checks, input checks, status updates, and heliostat "marking" (calibrating);
- c. After all the current HC statuses are updated, ALM (ALARM) and STS (STATUS) tasks are activated to determine any new alarms or field status changes. Task BHI of the Command Processor Module is activated to build any HFC initialization commands to be output. FCP suspends itself for 55 msec to give BHI enough time to execute; and
- d. Command output loop (DO-WHILE commands to be sent) with command buffer copying to "private" command buffers, command output retries, and output checking. Command returns (feedback from HCs whether or not commands were received) are monitored the next second along with HFC and HC status. If no commands are to be output or when command output is done, FCP will activate task BHC to build any new commands to be output during the next time frame and terminates itself via REX, EXIT.

NOTE: The timing functions necessitated by the field communications lines and the firmware in the HFCs are distributed over "a" through "d" of the above parts.

Called submodules:

- a. Output check routine (FCPOUT);
 - b. Input check and HFC status update routine (FCPIN);
 - c. HC status update routine (FCPUPD);
 - d. Byte checksum routine with storage (FCPCHO);
 - e. Byte checksum routine without storage (FCPCHI); and
 - f. I/O error recording routine (FCPSWH).
 - g. Line switching routine (FCPFLP).
- Utilized Executive Requests (REX calls) in FCP and its submodules are:

- a. DELAY - used for timing;
- b. OUTPUT - used to send data on eight HFC communication lines;
- c. ACTIVATE - used to trigger other tasks;
- d. SUSPEND - used to "put FCP to sleep." (Expiration of DELAY timer will wake FCP);
- e. INPUT - used to receive data on eight HFC communication lines;
- f. TERMINATE - used to terminate I/O following error detection; and
- g. EXIT - used to terminate FCP until reactivated by TIK.

All REX calls are documented in the MODCOMP Reference Manual for MAX IV (MAXNET) General Operating System. See the FCP functional overview Figure 3.2.4-1 for an overview.

3.2.4.3.2

Resource Budgets

- a. Memory required: 3K of resident memory plus access to global common;
- b. Timing: See HAC/HFC/HC timing diagram (Figure 3.2.4-2);
- c. Mass storage (disk, tape) required: None; and

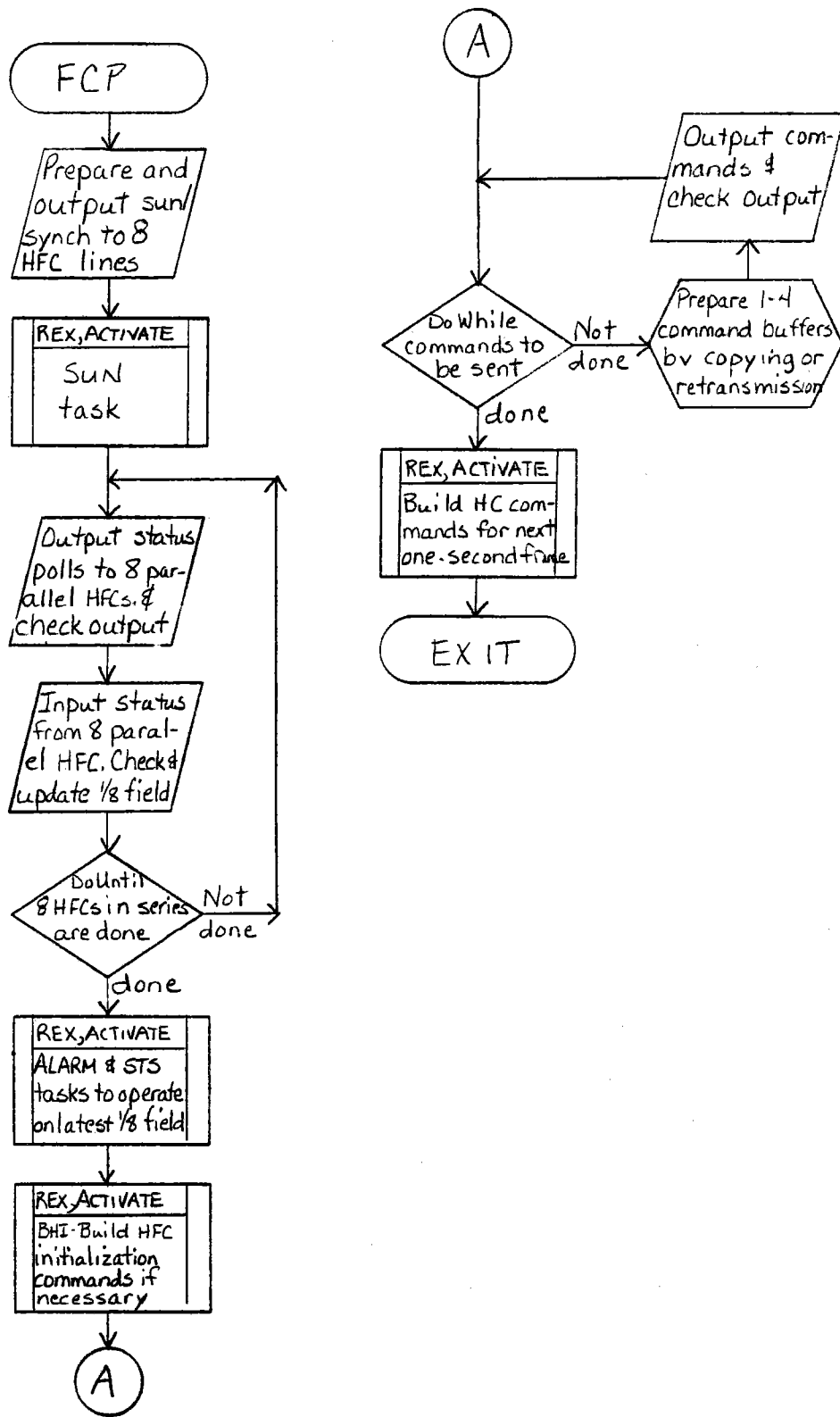


Figure 3.2.4-1 Flowchart - FCP Functional Overview

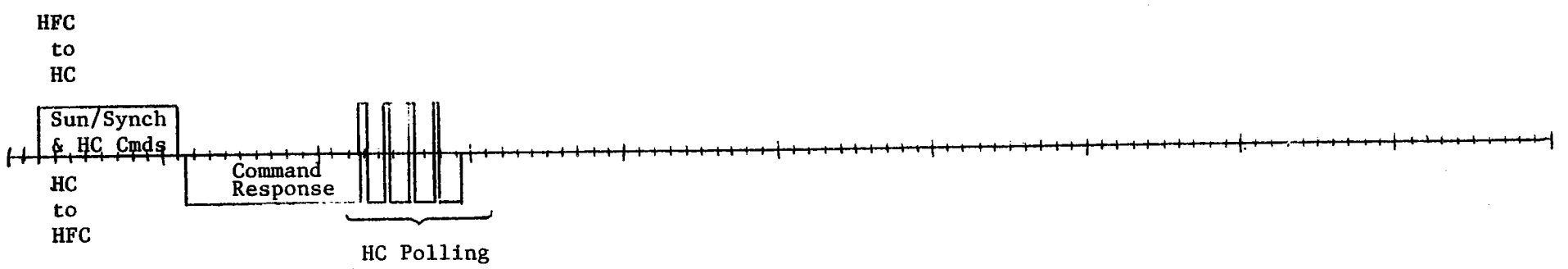
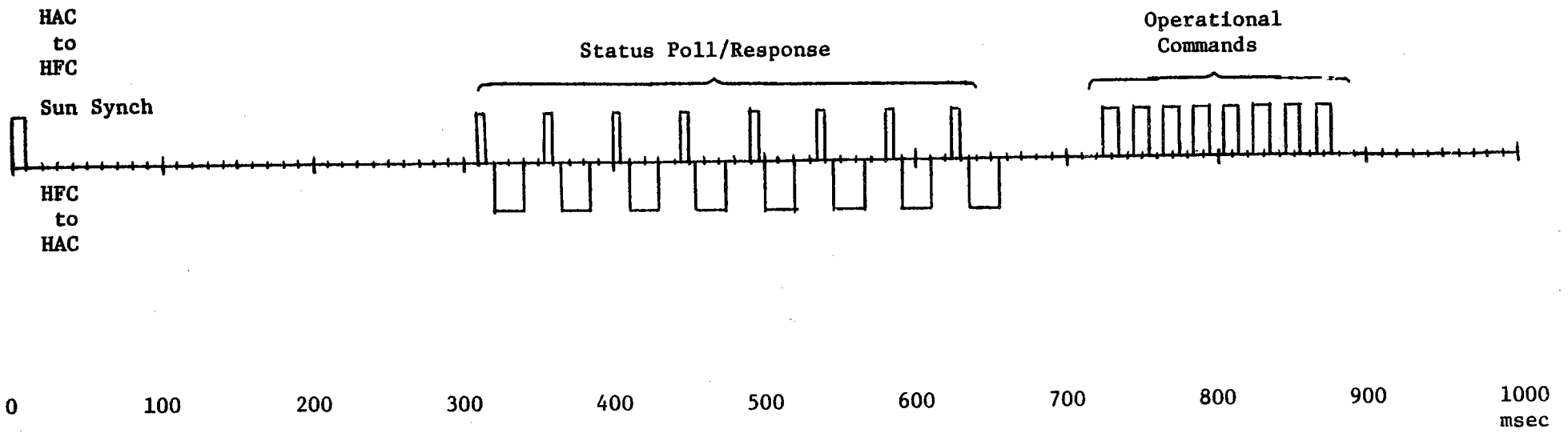


Figure 3.2.4-2 FLDCOM One-Second Communications Time Frame

Time From Start of Data Frame, msec	Duration, msec	Data	Path
0	10	Sun position data.	HAC to HFC
20	90	Sun position plus operational commands received in last frame, if any.	HFC to HC
115	113	Response from each HC, in order, to indicate correct/incorrect receipt of commands.	HCs to HFC
228	3	Status request to first of four HCs being polled this frame.	HFC to HC
231	13	Status response.	HC to HFC
244	3	Status request to second of four HCs being polled this frame.	HFC to HC
247	13	Status response.	HC to HFC
260	3	Status requests to third of four HCs being polled this frame.	HFC to HC
263	13	Status response.	HC to HFC
276	3	Status request to fourth of four HCs being polled this frame.	HFC to HC
279	13	Status response.	HC to HFC
310	5	Status request to first HFC on data bus.	HAC to HFC
320	20	Status response, including status from four HCs.	HFC to HAC
355	5	Status request to second HFC on data bus.	HAC to HFC
365	20	Status response, including status from four HCs.	HFC to HAC
400	5	Status request to third HFC on data bus.	HAC to HFC
410	20	Status response, including status from four HCs.	HFC to HAC
445	5	Status request to fourth HFC on data bus.	HAC to HFC
455	20	Status response, including status from four HCs.	HFC to HAC
490	5	Status request to fifth HFC on data bus.	HAC to HFC
500	20	Status response, including status from four HCs.	HFC to HAC
535	5	Status request to sixth HFC on data bus.	HAC to HFC
545	20	Status response, including status from four HCs.	HFC to HAC

Figure 3.2.4-2 FLDCOM One-Second Communications Time Frame (con't.)

Time From Start of Data Frame, msec	Duration, msec	Data	Path
580	5	Status request to seventh HFC on data bus.	HAC to HFC
590	20	Status response, including status from four HCs.	HFC to HAC
625	5	Status request to eighth HFC on data bus.	HAC to HFC
635	20	Status response, including status from four HCs.	HFC to HAC
725	10	Operational commands, if any, to first HFC on data bus.	HAC to HFC
745	10	Operational commands, if any, to second HFC on data bus.	HAC to HFC
765	10	Operational commands, if any, to third HFC on data bus.	HAC to HFC
785	10	Operational commands, if any, to fourth HFC on data bus.	HAC to HFC
805	10	Operational commands, if any, to fifth HFC on data bus.	HAC to HFC
825	10	Operational commands, if any, to sixth HFC on data bus.	HAC to HFC
845	10	Operational commands, if any, to seventh HFC on data bus.	HAC to HFC
865	10	Operational commands, if any, to eighth HFC on data bus.	HAC to HFC

Figure 3.2.4-2 FLDCOM One-Second Communications Time Frame (con't.)

- d. Task priority: Highest priority task in the HAC system after TOK and TIK (timer tasks).

3.2.4.4 Design Description

3.2.4.4.1 Module Structure (See Figure 3.2.4-3)

FCP is a stand-alone task which is memory resident. As described under 3.2.4.3.1, there are four major parts to the task and six submodules to do checksum calculations, output checking, input checking, HC status updates, and communications system switching.

3.2.4.4.1.1 Submodule I - FCP

3.2.4.4.1.1.1 Description

- a. Language used - MODCOMP Classic Assembler
- b. How invoked - Activated by the timer task TIK on a once-per-second basis.
- c. Constraints and limitations - In order to allow the most precise timing possible, and thereby make optimum use of the HAC CPU and I/O System, the FCP task will run on the third-highest priority of the system. Other constraints include being able to do all output and inputs within one second and not doing excessive processing between time delay periods.
- d. Processing

The major parts of processing consist of:

1. When first activated FCP shall check to see if it resides in the primary or backup HAC. If FCP is in the backup HAC and a communication failure caused the switch, then FCP will switch the UFTs to the alternate communication lines. Otherwise, the lines in use by the primary HAC will also be used by the backup.
2. Then each one second frame, FCP will clear reported line errors in global common array LINESG, build the sun/synch packet, outputting it to all eight HFC lines, reporting any I/O errors to the data base, and switching field communications if necessary. Also, the SUN task is activated to calculate the next time frame's sun vector. The FCP task will also check the status of each line. If the line status does not go enabled in thirty minutes, the FCP task will switch to the alternate line automatically to establish communications with the field.
3. Polling each of eight HFCs on all of eight lines for status of four successive HCs, such that

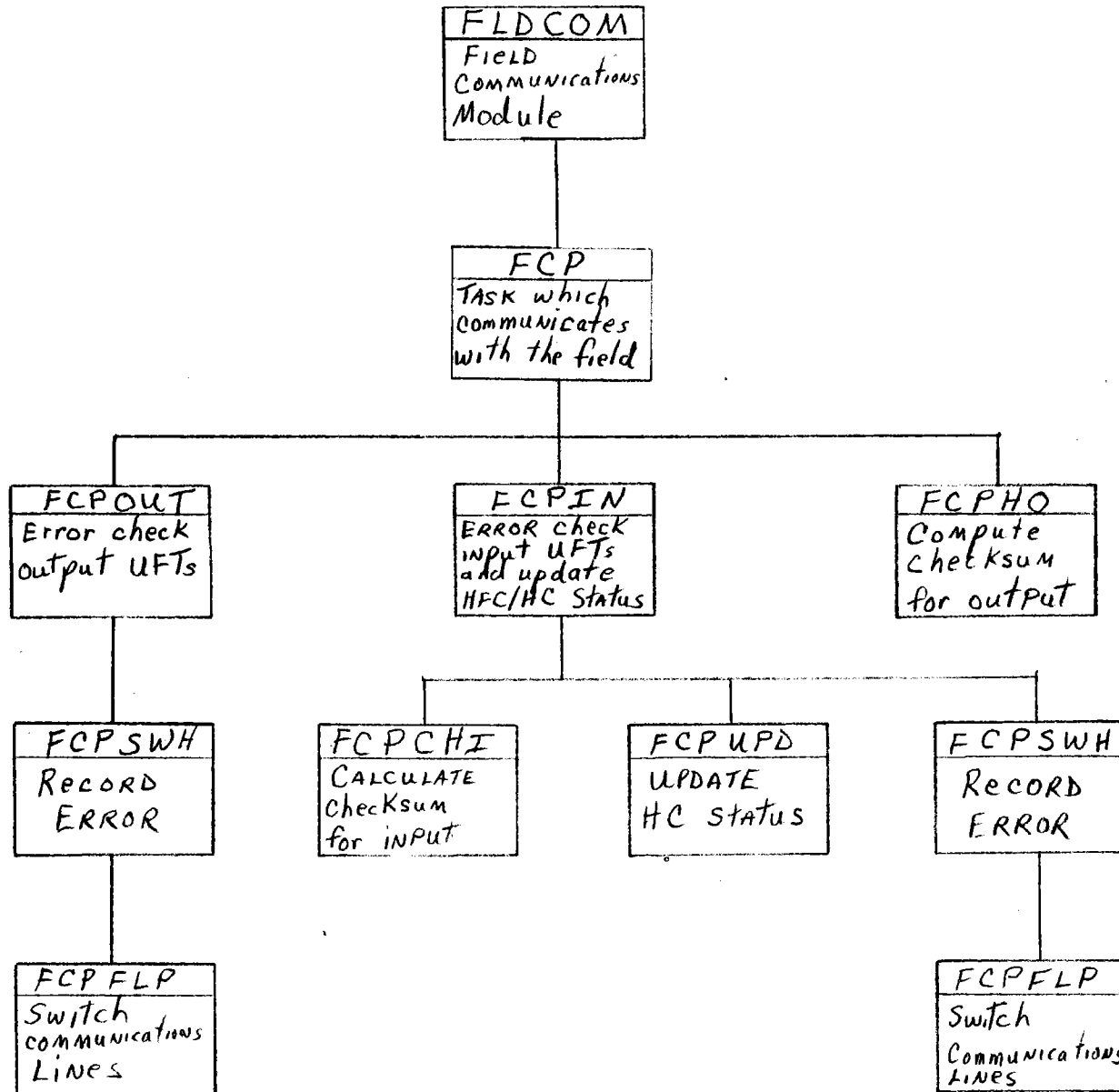


Figure 3.2.4-3 FLDCOM Module

one-eighth of the field is updated every second and the whole field every eight seconds. Again, I/O errors are reported to the data base and field communications switched if necessary. DO-UNTIL implementation is utilized.

4. The ALM and STS tasks are activated to operate on the newly arrived status for one-eighth of the field. Task BHI of the Command Processor Module is triggered to fill the command buffer with any HFC initialization commands required to be sent to the field. A 55 msec delay is involved to give BHI enough time.
5. As long as commands to individual HFCs need to be output (as determined by the local retry counter or by the command buffer ready bit in HFCS2G), they are output to the corresponding HFCs. If for a particular HFC, the local command buffer is not tied up with retries and a new buffer is ready to be output, it is copied from the global CMDBFG to the local command buffer, and the command buffer ready bit is reset so that new commands can be filled in the next time FCP triggers the BHI and the BHC tasks. One to eight HFC lines are commanded quasi-simultaneously (in parallel) and one to eight HFCs on each line are commanded in 20 msec steps (in series). Output errors are reported to the data base immediately; failure of HCs to report command returns are reported after three retries. FCP then initiates the BHC task to build HC commands required for the next one-second time frame and suspends itself until the beginning of the next frame.

- e. Error messages and recovery - The FCP task has no error messages.

3.2.4.4.1.1.2 Data, Logic, and Command Paths

Input data for FCP:

- a. Sun position input from data base location SUNPOG;
- b. Input from global common is the command buffer CMDBFG, and the associated bit flag HFCS2G (bit 10) indicating the command buffer is ready to output; and
- c. Input from each HFC into local holding buffer.
- d. Line status, LINESG; field status, FLDSTG.

Output data for FCP:

- a. Output to each HFC for sun/synch , Status requests, and for commands from local holding buffers;
- b. Output of HC statuses to data base for STS and ALM tasks;
- c. Output of I/O errors to data base for ALM task;
- d. Algorithm for calculating a byte checksum called FCPCHO; and
- e. Command Paths: Activate **SUN, ALM, STA, BHI, and BHC** tasks.

3.2.4.4.1.1.3 Internal Data Description

- a. 5-word Register Save area;
- b. 2 10-word UFT address tables;
- c. 16 10-word UFTs (input and output);
- d. 5 1-word task names;
- e. 64 1-word retry counters;
- f. 1 8-word buffer for keeping track of command outputs;
- g. 1 8-word buffer for sun/synch;
- h. 8 2-word buffers for polling;
- i. 64 10-word local HFC command buffers;
- j. 8 17-word status input buffers; and
- k. 8 1-word logical device names for alternate lines.

3.2.4.4.1.1.4 Flowchart(s)

See attached Figure 3.2.4-4.

3.2.4.4.1.2 Submodule II - Subroutine FCPOUT

3.2.4.4.1.2.1 Description

- a. Language used - MODCOMP Classic Assembler
- b. How invoked - It is invoked by FCP after each output completion.

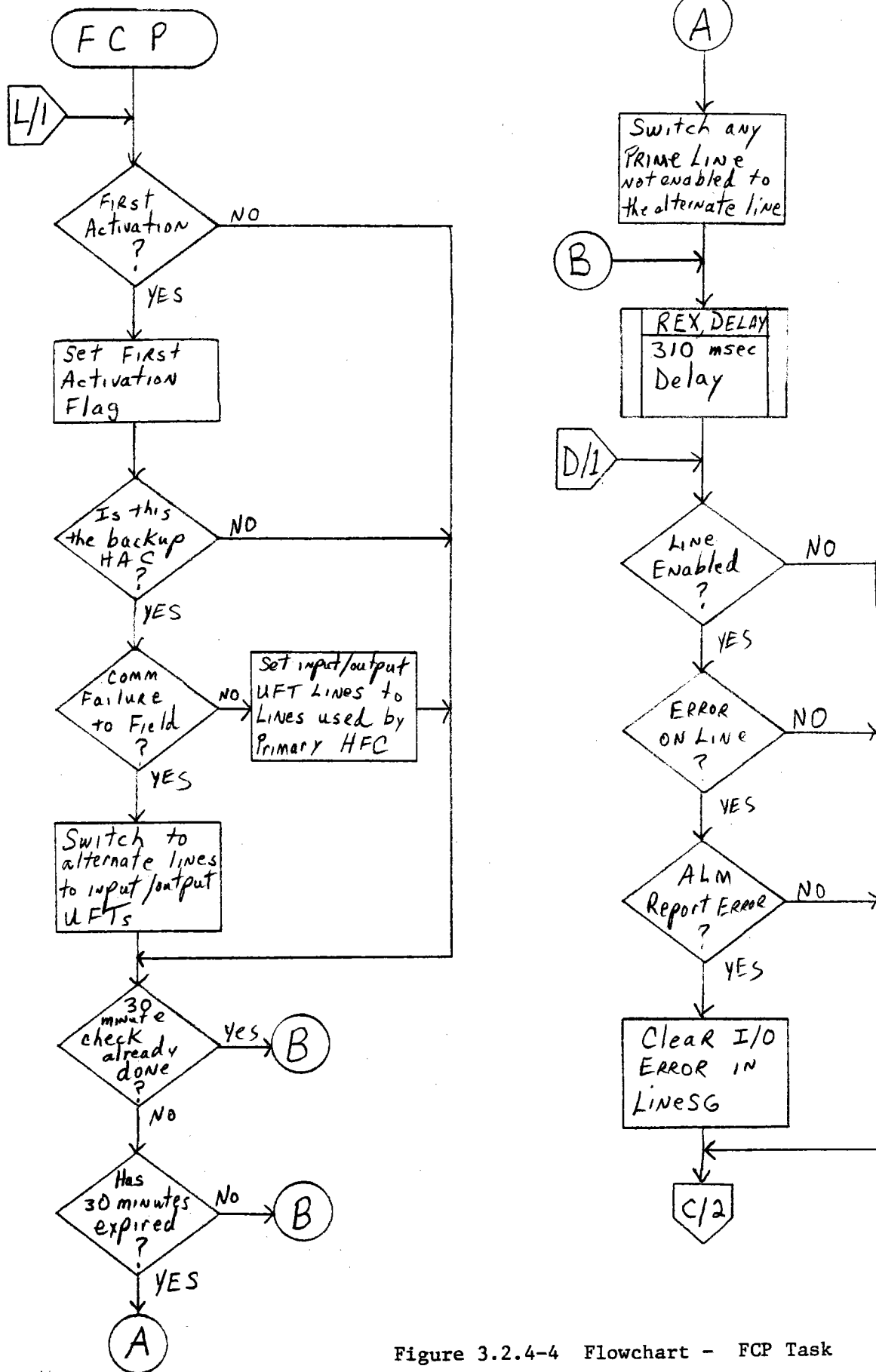


Figure 3.2.4-4 Flowchart - FCP Task

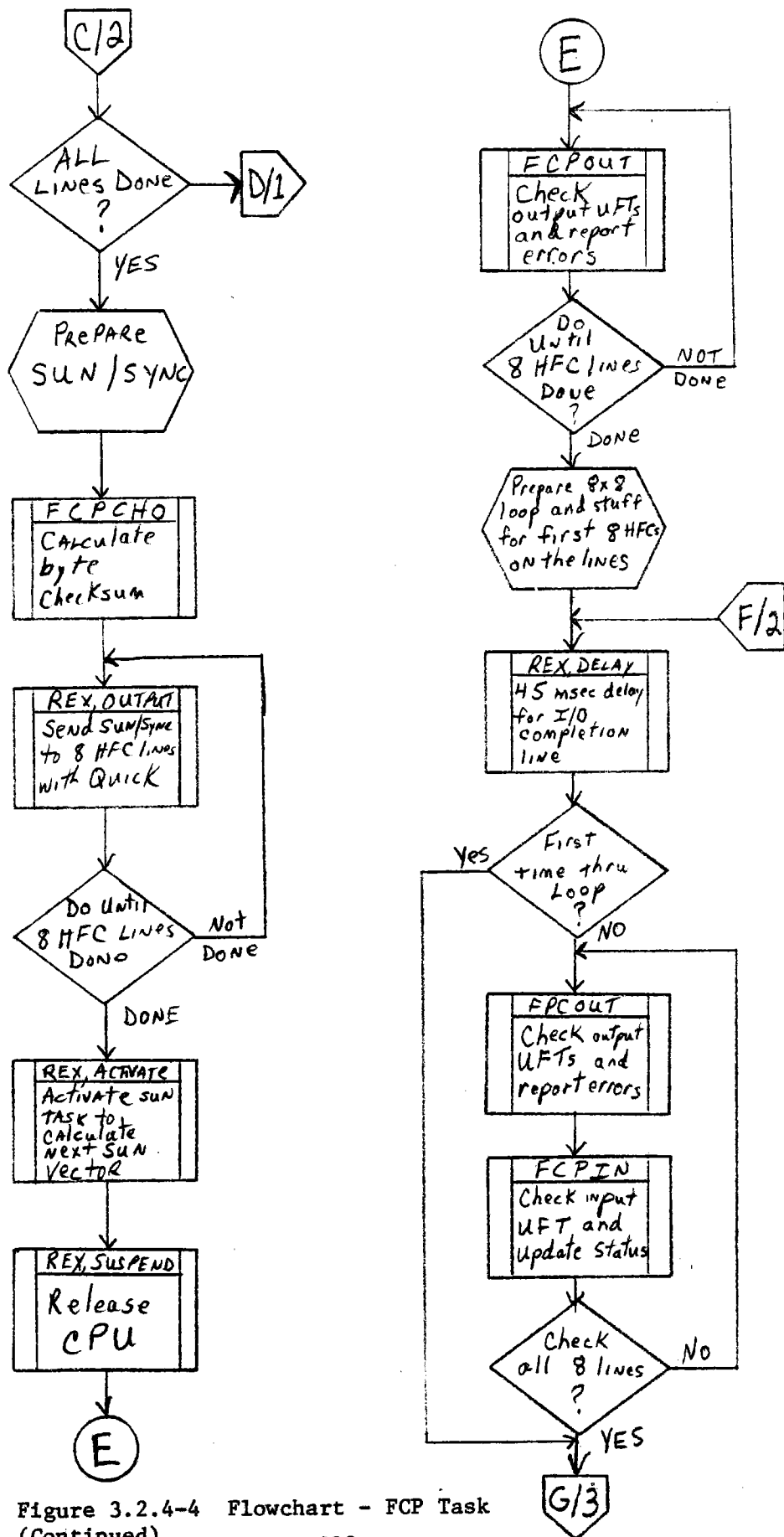


Figure 3.2.4-4 Flowchart - FCP Task (Continued)

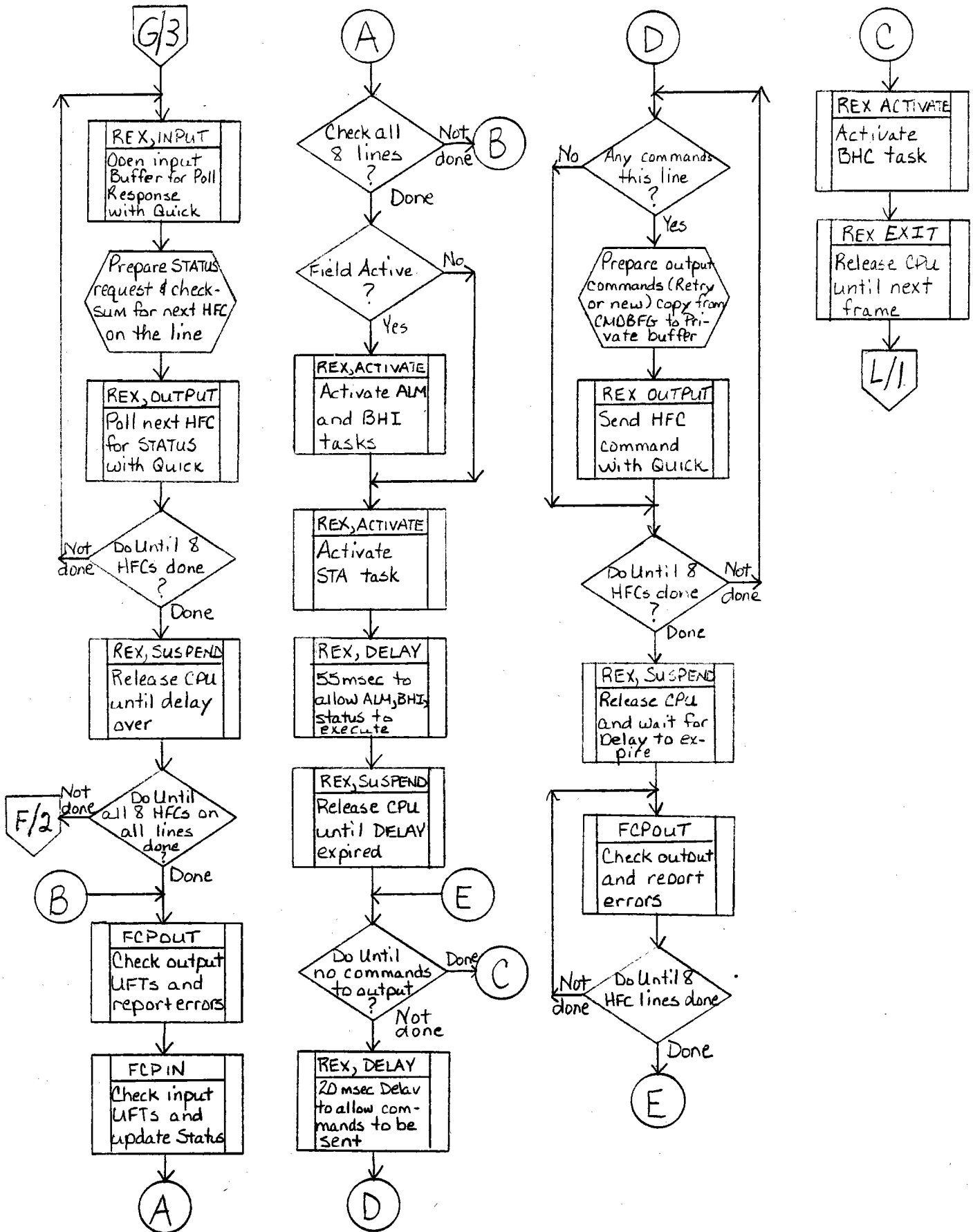


Figure 3.2.4-4 Flowchart - FCP Task (continued)

- c. Constraints and limitations - Each call to FCPOUT will check one UFT for errors being present.
- d. Processing - This routine when called shall check the UFT for output errors and mark the data base with communication errors on an output channel basis.
 - 1. A UFT which did not complete will imply an HFC output time-out condition. All other errors denote a line error.
 - 2. If an error is found, the routine FCPSWH will be called to determine if the failure criterion for the line having the error has been met.
- e. Error messages and recovery - There are no error messages arising from FCPOUT processing. Output errors found are indicated in the global data base and output by the ALARM module.

3.2.4.4.1.2.2 Data, Logic and Command Paths

Input data for FCPOUT:

- a. UFTs (user file tables);
- b. A flag for retries; and
- c. LINE/HFC/HC index.

Output data for FCPOUT:

- a. Global data base bits;
- b. The local retry table; and
- c. Calling of routine FCPSWH.

3.2.4.4.1.2.3 Internal Data Description

15-word register save area.

3.2.4.4.1.2.4 Flowchart

See Figure 3.2.4-5

3.2.4.4.1.3 Submodule III - Subroutine FCPIN

3.2.4.4.1.3.1 Description

- a. Language used - MODCOMP Classic Assembler
- b. How invoked - It is invoked by FCP after each input completion or time-out of such input.

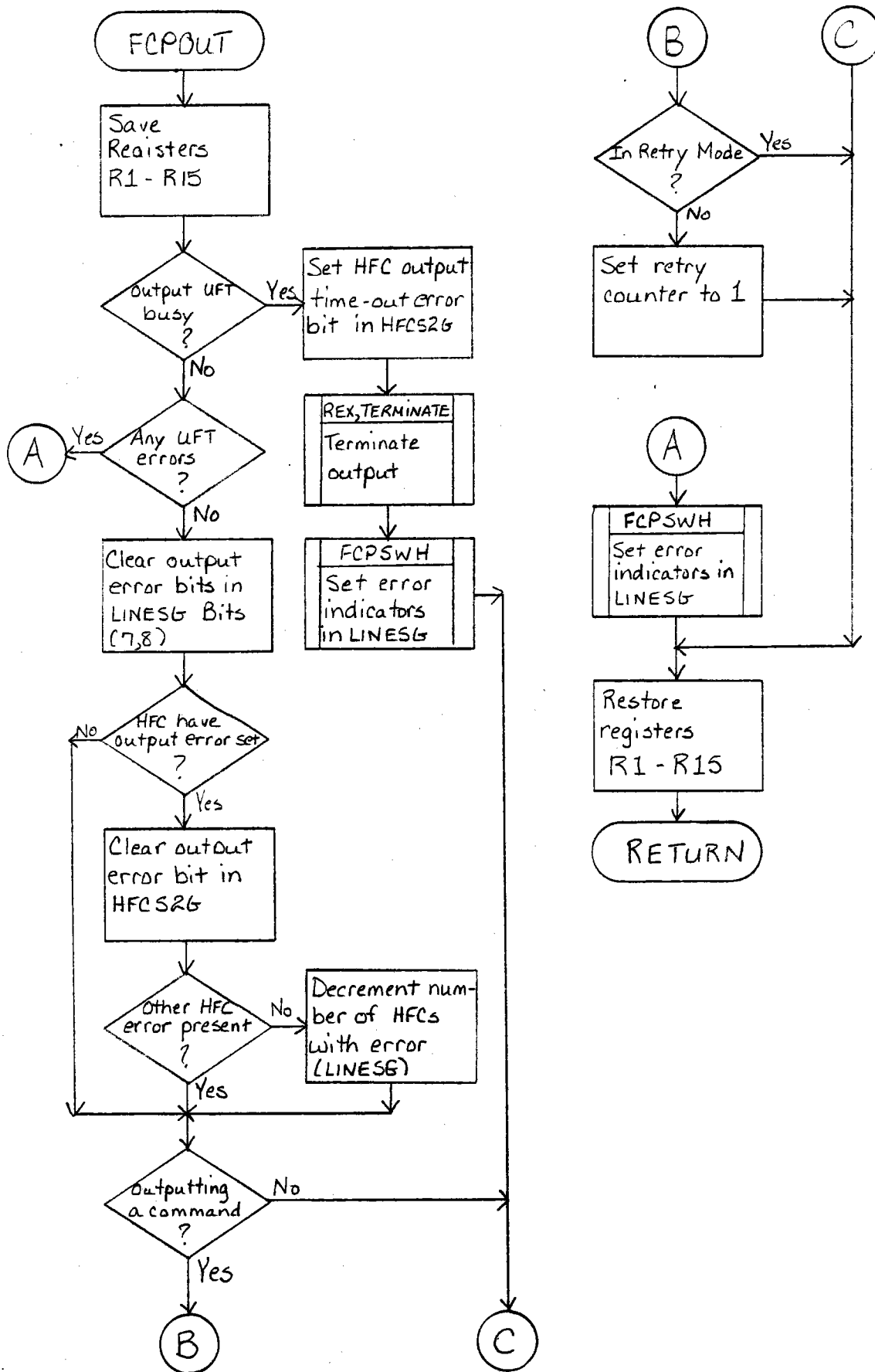


Figure 3.2.4-5 Flowchart - FCPOUT

c. Constraints and limitations - FCPIN will check one input UFT per invocation.

d. Processing

1. This routine will check the input UFT and the HFC status response message for input errors. The various input errors checked for are the following:

- a) If a UFT did not complete (i.e. the input buffer was not full) it is considered an HFC input time-out error;
- b) All other errors indicated in the UFT are considered line errors;
- c) Not enough bytes transferred, a bad checksum, or if the input header byte was not a normal status indicates an HFC/HAC communication error;
- d) The last command sent didn't match the last command received; and
- e) Missing and extra HC command returns are checked for and marked in HC global states.

2. If one of these input errors is detected for an installed HFC with a good status during the last second, the routine FCPSWH is called to determine if the failure criterion for the line has been reached.

3. If no errors were found on the line, the line status will be set to enabled (if not already enabled), the number of enabled lines in the field shall be incremented, and the field status set to enabled.

4. If no HFC or line errors are present or command output retries are in progress, then FCPIN will update HC status by calling routine FCPUPD. Before returning, the HFC status will be updated in global common array HFCS1G.

e. Error messages and recovery - None.

3.2.4.4.1.3.2 Data, Logic, and Command Paths

a. Input data are UFTS (user file tables), input buffers, and the LINE/HFC/HC index; and

- b. Output data are data base bits and HC status words (AZIMG, ELEVG, HCST1G - all indirectly via subroutine FCPUPD) as well as HFC status words (HFCS1G), line status words (LINESG), and field status (FLDSTG).

3.2.4.4.1.3.3 Internal Data Description

- a. 15-word save area for registers 1-15;
- b. Save location for last command;
- c. Various bit masks; and
- d. Various constants for byte-counts and commands.

3.2.4.4.1.3.4 Flowchart

See attached Figure 3.2.4-6.

3.2.4.4.1.4 Submodule IV - Subroutine FCPCHO

3.2.4.4.1.4.1 Description

- a. Language used - MODCOMP Classic Assembler
- b. How invoked - Called by FCP task to compute checksum for output buffer.
- c. Constraints and limitations - None
- d. Processing - This routine calculates an n-byte checksum and stores the result in byte n+1. It is used for each output to the field.
- e. Error messages and recovery - None

3.2.4.4.1.4.2 Data, Logic, and Command Paths

Input data for FCPCHO are:

- a. Return link address;
- b. 5-word save area;
- c. Byte count;
- d. Buffer address; and
- e. Actual buffer to be checksummed.

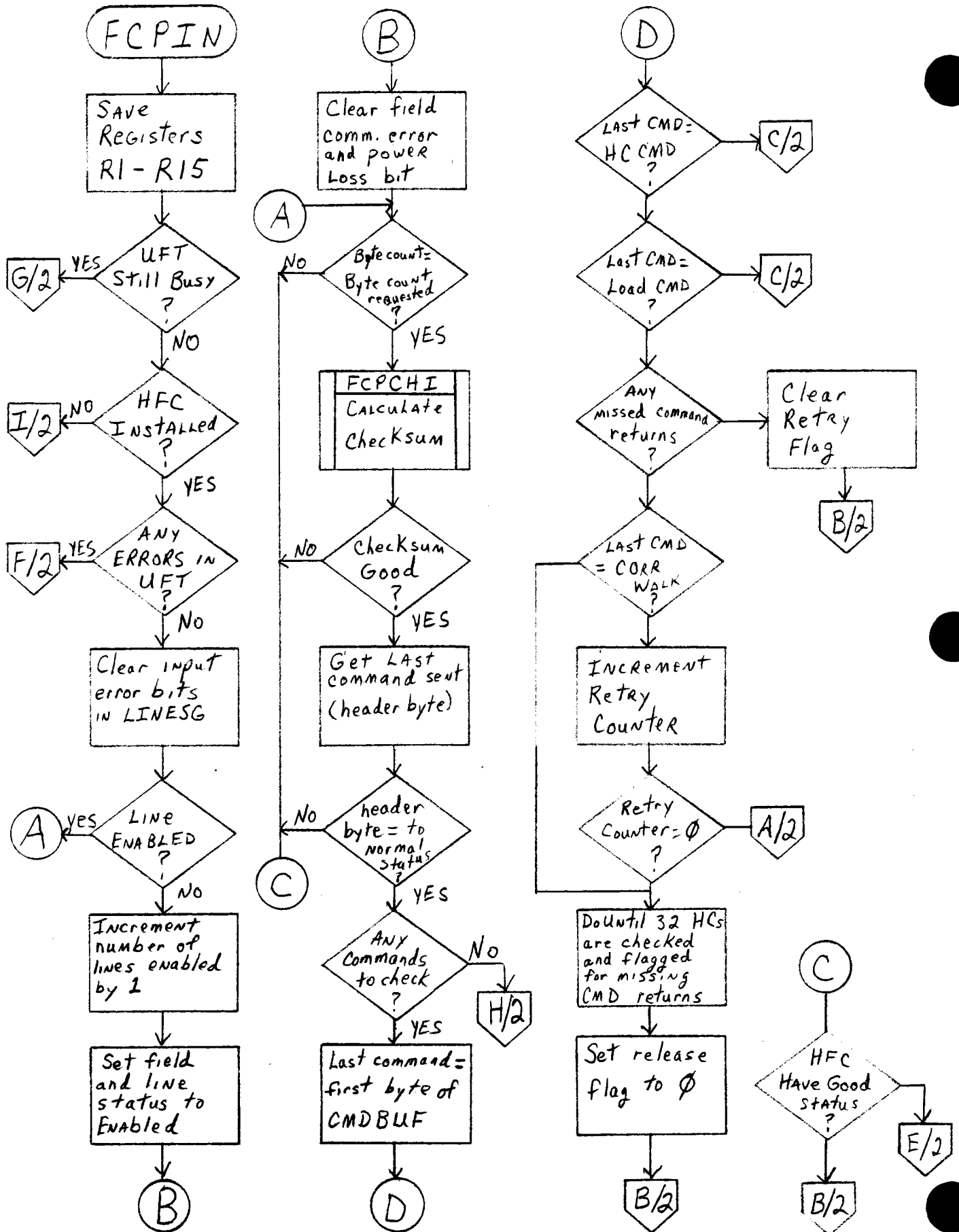


Figure 3.2.4-6 Flowchart - FCPIN

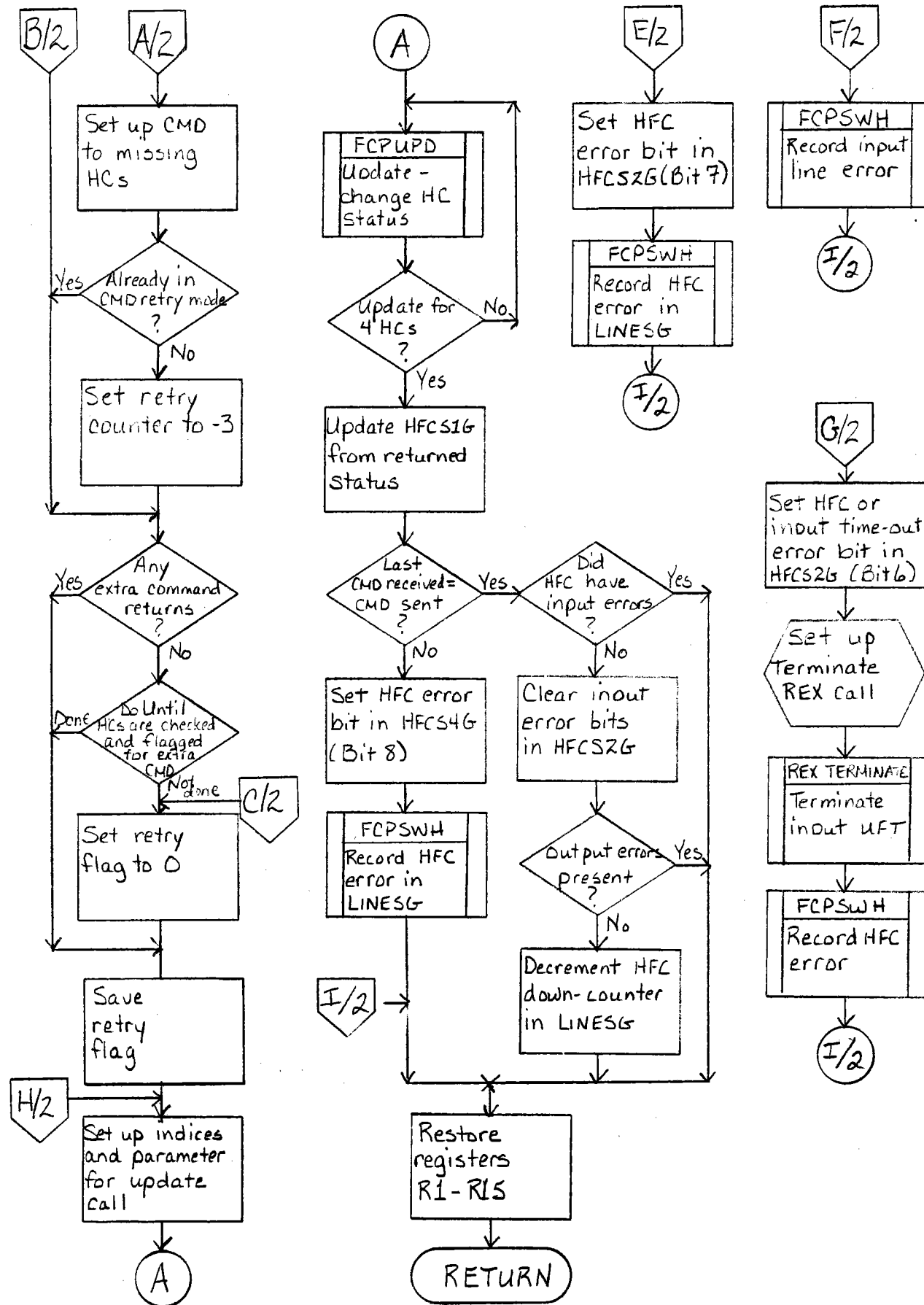


Figure 3.2.4-6 Flowchart - FCPIN (continued)

Output data for FCPCHO is the (n+1) byte of the buffer which contains the actual checksum upon exit.

3.2.4.4.1.4.3 Internal Data Description

None

3.2.4.4.1.4.4 Flowchart

See attached Figure 3.2.4-7.

3.2.4.4.1.5 Submodule V - Subroutine FCPCHI

3.2.4.4.1.5.1 Description

- a. Language used - MODCOMP Classic Assembler
- b. How invoked - It is invoked by FCPIN to compute the checksum.
- c. Constraints and limitations - None
- d. Processing - This routine calculates an n-byte checksum and returns it in R11. It is used to compare against actual checksums after input completions.
- e. Error messages and recovery - None

3.2.4.4.1.5.2 Data, Logic, and Command Paths

Input data for FCPCHI are:

- a. Return link address;
- b. 3-word save area address;
- c. Byte count;
- d. Buffer address; and
- e. Actual buffer to be checksummed.

Output data for FCPCHI is R11 which contains the calculated checksum upon exit.

3.2.4.4.1.5.3 Internal Data Description

None

3.2.4.4.1.5.4 Flowchart

See attached Figure 3.2.4-8.

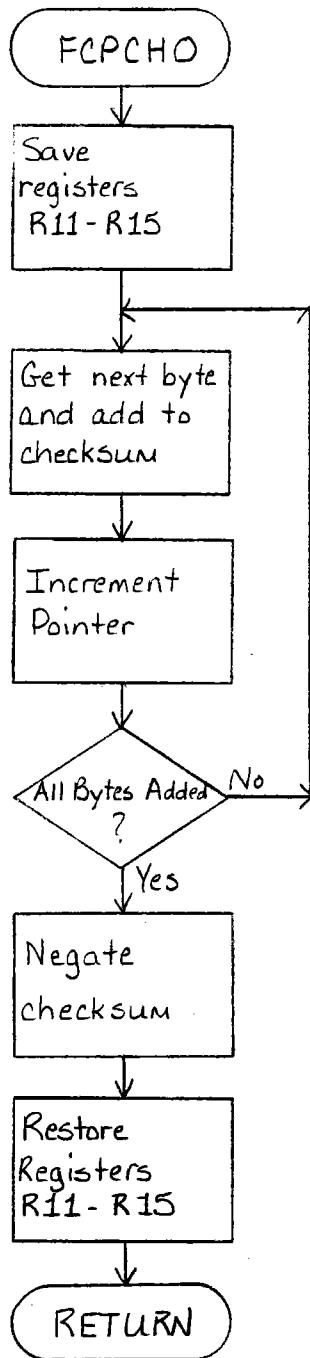


Figure 3.2.4-7 Flowchart - FCPCHO

3.2.4.4.1.6 Submodule VI - Subroutine FCPUPD

3.2.4.4.1.6.1 Description

- a. Language used - MODCOMP Classic Assembler
- b. How invoked - Called by FCPIN to update HC status following input of a status response message.
- c. Constraints and limitations - Update status for one HC per invocation.
- d. Processing

This routine updates HC status (HCST1G), Azimuth encoder position (AZIMG), and Elevation encoder position (ELEVG) for a single heliostat as reported by an HC via the HFC to the HAC.

In addition to status updates, FCPUPD shall save the mark encountered bits if the HC is in the "SEEK MARK" mode/submode such that all such bits are available for final evaluation and subsequent mode changes and/or alarms. FCPUPD does not update AZ and EL if mode/submode equal zero.

- e. Error messages and recovery - None

3.2.4.4.1.6.2 Data, Logic, and Command Paths

Input parameters for FCPUPD are:

- a. Return link address;
- b. Buffer pointer; and
- c. HC index (0-2047).

Output parameters for FCPUPD are:

- a. (HCST1G) - HC status array;
- b. (AZIMG) - Heliostat Azimuth Encoder Position array; and
- c. (ELEVG) - Heliostat Elevation Encoder Position array.

3.2.4.4.1.6.3 Internal Data Description

Internal data used by FCPUPD are three masks:

- a. One to define the mode/submode field (#007C-bits 9-13);

- b. A second to define SEEK MARK mode/submode (#0064); and
- c. A third to define the two mark-encountered bits (#0000).

3.2.4.4.1.6.4 Flowchart

See Figure 3.2.4-9.

3.2.4.4.1.7 Submodule VII - Subroutine FCPSWH

3.2.4.4.1.7.1 Description

- a. Language used - MODCOMP Classic Assembler
- b. How invoked - Called by either FCPIN or FCPOUT upon detection of a line or HFC error.
- c. Constraints and limitations - None
- d. Processing - The FCPSWH routine shall be responsible for recording the error in global common (LINESEG) and determining if a communications switchover condition is present (see paragraph 3.2.4.3).
 1. The FCPSWH routine begins processing by checking if a power loss condition is present. If a power loss is present, then no error recording is performed, and control is returned to the calling routine. Otherwise, the FCPSWH routine will continue processing by checking if all enabled lines have had an I/O error. If so, then either a communication failure to the entire field has occurred, or the field has suffered a power loss.
 2. If the backup HAC is operational, the FCPSWH routine will set a bit in global common to initiate a switchover. Otherwise a switch to the alternate lines will be made. No switching will be performed for a power loss condition.
 3. If a communication failure was not detected, the FCPSWH routine shall record an error in global common and perform checks to determine if a switchover condition is present for the indicated line. If so, the FCPFLP routine will be called to switch communication lines.
- e. Error messages and recovery - No error messages are produced by FCPSWH.

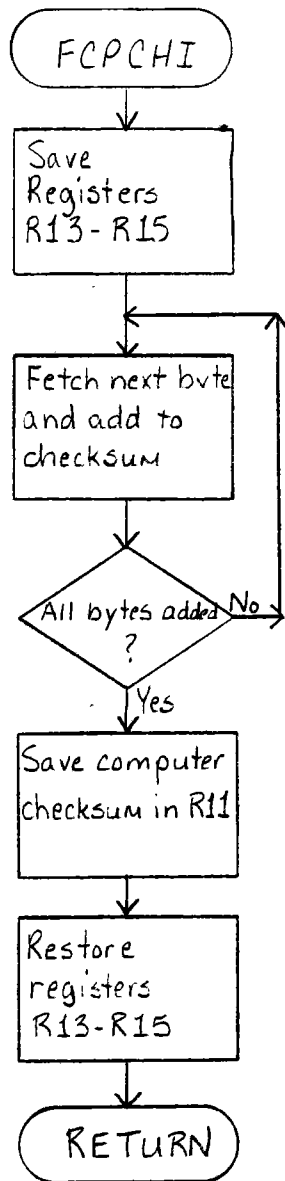


Figure 3.2.4-8 Flowchart - FCPCHI

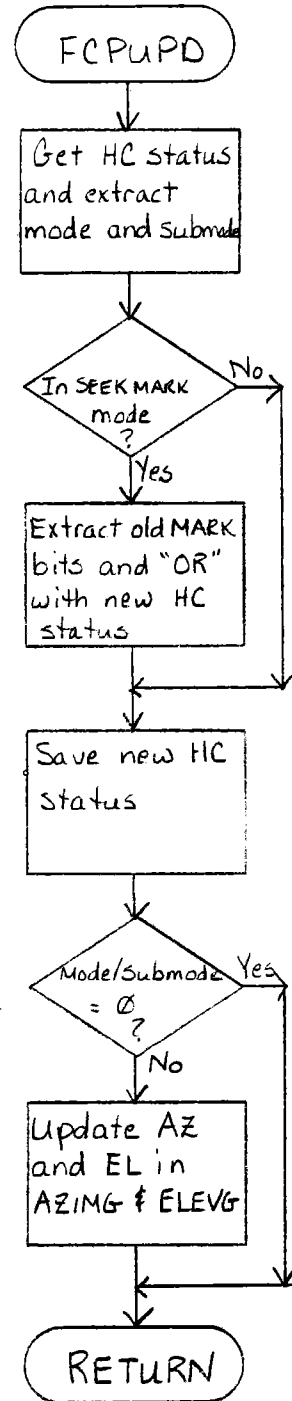


Figure 3.2.4-9 Flowchart - FCPUPD

3.2.4.4.1.7.2 Data, Logic, and Command Paths

Input data from FCPSWH are:

- a. Line number which had the error;
- b. HFC number which had the error:
- c. Type of error (input or output); and
- d. Global common variables FLDSTG, LINESEG, HFCS2G, status bit for backup HAC in SWOVG.

Output data from FCPSWH are:

- a. Modified global common words LINESEG and FLDSTG; and
- b. Global common bit to indicate switch to backup HAC.

3.2.4.4.1.7.3 Internal Data Description

The following are internal data used by FCPSWH:

- a. 15-word save area for registers R1 - R15; and
- b. Miscellaneous local variables.

3.2.4.4.1.7.4 Flowchart

See Figure 3.2.4-10.

3.2.4.4.1.8 Submodule VIII - Subroutine FCPFLP

3.2.4.4.1.8.1 Description

- a. Language used - MODCOMP Classic Assembler
- b. How invoked - Called by either the FCP task or by subroutine FCPSWH when a line switchover condition is detected.
- c. Constraints and limitations - Two lines will be switched to the backup lines (if not already switched), and register R4 must have the line index of the line to be switched. (R4 must point to the lower-numbered line number of the pair to be switched; i.e., if lines 1 and 2 are to be switched, then R4 must have a 1 in it when FCPFLP is called.)
- d. Processing - Upon being called, the FCPFLP routine will document the number of lines enabled in global common word FLDSTG and then switch the line status in global common area LINESEG. The FCPFLP routine will then switch the user file table addresses to the alternate lines.

- e. Error messages and recovery - No error messages are produced by FCPFLP.

3.2.4.4.1.8.2 Data, Logic, and Command Paths

Input data from FCP and FCPSWH are:

- a. R4 - Pointer to the lesser number of the lines to be switched;
- b. LINESG - Global common line status;
- c. FLOSTG - Field status;
- d. FCP tasks UFT addresses.

Output data from FCPLLP:

- a. LINESG - Modified line status;
- b. FLOSTG - Modified field status;
- c. FCP tasks UFT addresses will be modified to indicate the alternate lines.

3.2.4.4.1.8.3 Internal Data Description

- a. 15-word save area for registers R1 - R15.

3.2.4.4.1.8.4 Flowchart

See Figure 3.2.4-11.

3.2.4.5 Interface Description

- a. Data Base Input:

SUNPOG - Sun Position Vector

CMDBFG - Command Buffer

HCST1G - HC Status 1

HCST2G - HC Status 2

HFCS2G - HFC Status 2

- b. Data Base Output:

HCST1G - HC Status 1

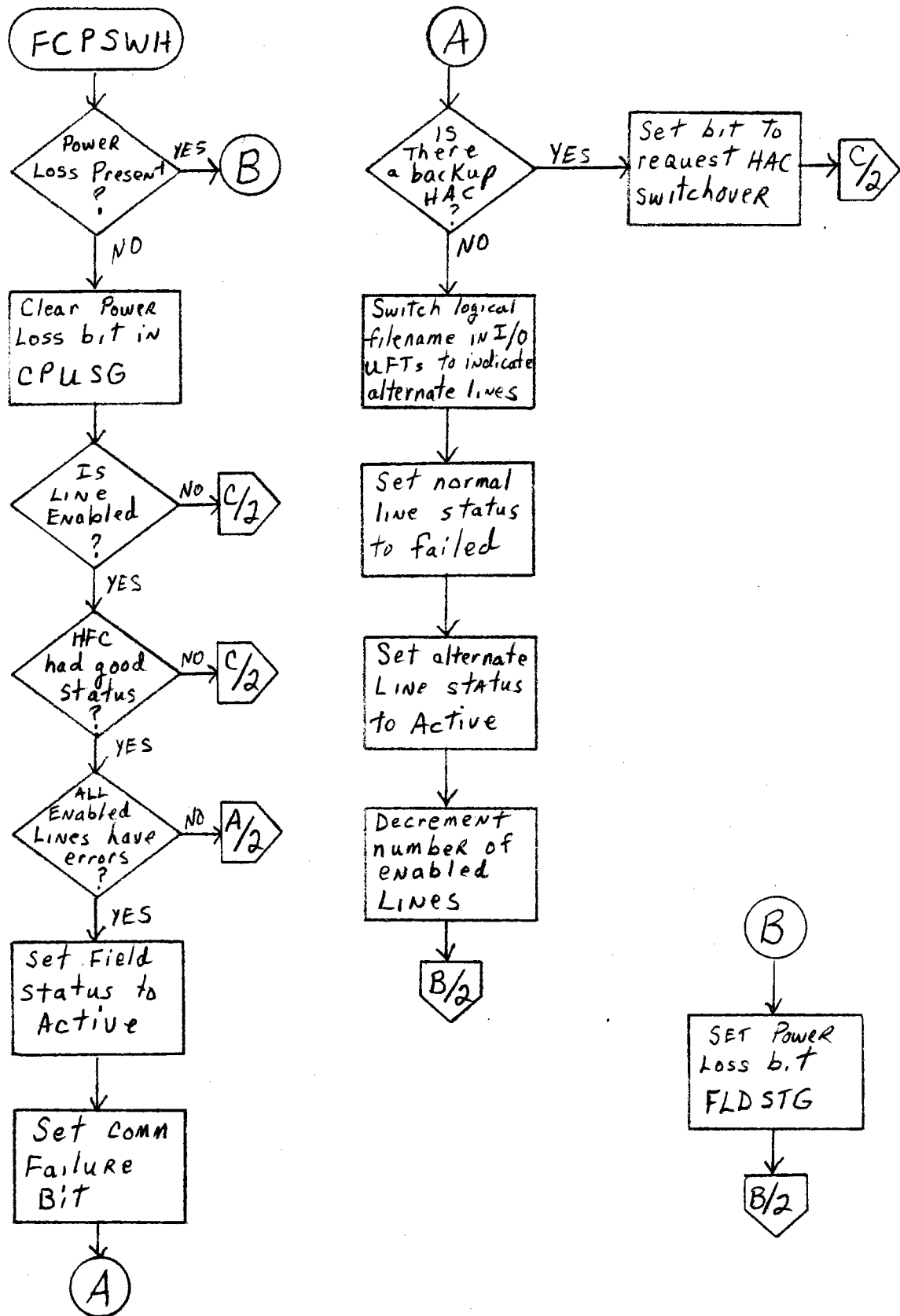


Figure 3.2.4-10 Flowchart - FCPSWH

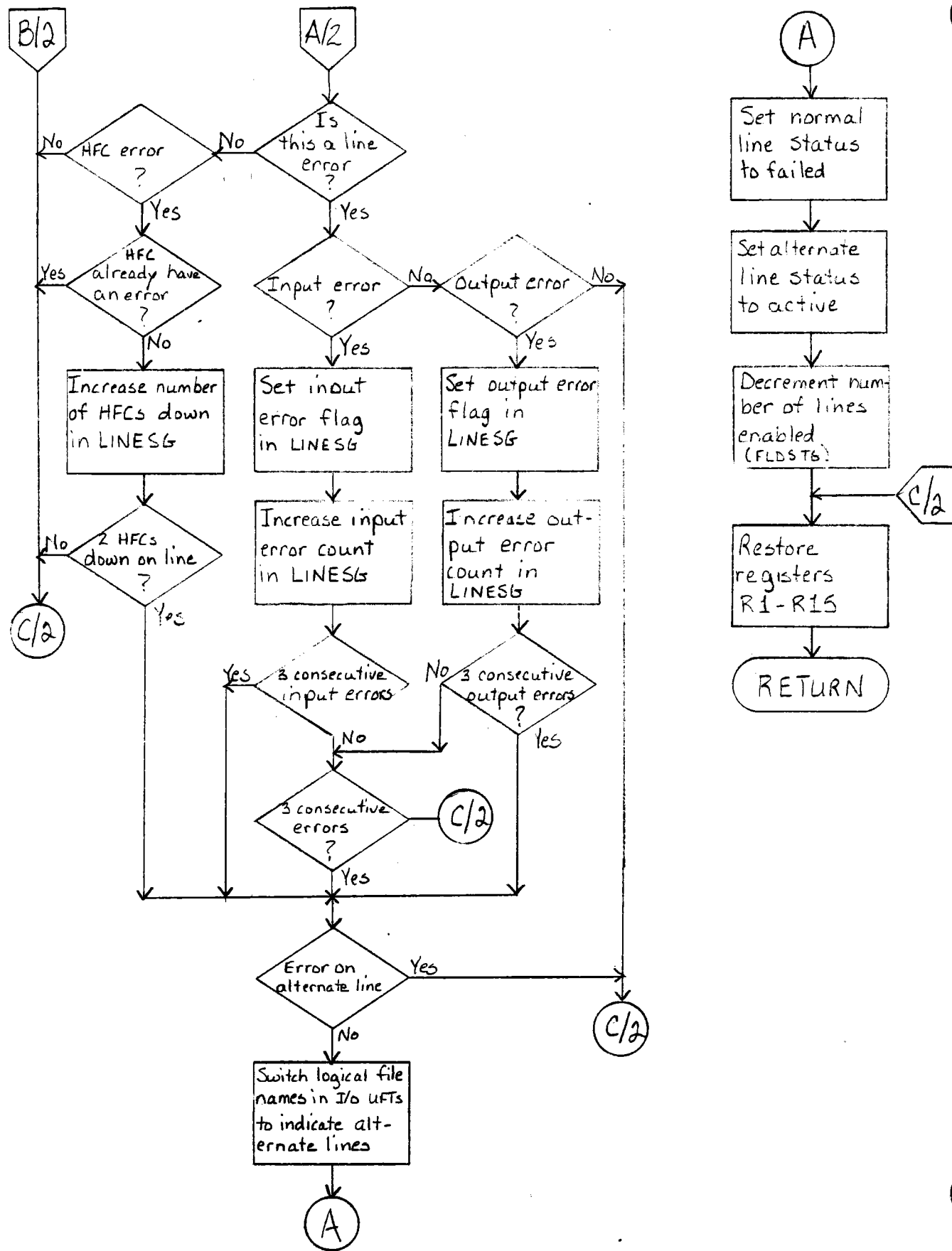


Figure 3.2.4-10 Flowchart - FCPSWH (continued)

HCST2G - HC Status 2
 HFCS1G - HFC Status 1
 HFCS2G - HFC Status 2
 AZ1MG - Azimuth Position
 ELEVG - Elevation Position
 CURHSG - Current HC group being statused
 TBUSYG - Task Busy Flag
 LINESG - Communication Line Status*
 FLDSTG - Field Status*
 HACSTG - HAC Status*

*These data items are required by the FLDCOM module to be transferred to the backup HAC on a once-per-second basis.

c. Physical Output:

SUN/SYNCH - Active or enabled to eight HFC lines
 STATUS POLL (MODCOMP 1931) Packets as described
 COMMANDS in 3.2.1.2 of S/W Functional Require-
 ments Specification.

d. Physical Input:

STATUS - From eight HFC lines (MODCOMP 1931)
 Packets as described in 3.2.1.2 of
 S/W Functional Requirements Specification.

e. Activation by TIK; and

f. Activation of SUN, ALM, BHI, BHC, and STS tasks.

3.2.4.6 Test Requirements

- a. Verification of FCP's logic will be done using MAX IV DEBUG. This system utility allows selective execution of portions of code, tracing, and modification of memory. The memory modification feature will be used to simulate HFC and HC behavior;
- b. Verification of timing will be done by repeated execution as well as oscilloscope inspection of communication lines; and
- c. Verification of communication performance to a field of 2048 heliostats is implied by "a" and "b" because FCP is designed to communicate with 64 HFCs and 2048 heliostats at all times. If any HFCs or HCs do not exist, error handling logic replaces status updates, but the timing does not change.

3.2.5 Alarm Processor Module - ALARMS

3.2.5.1 Purpose

The purpose of this module is to detect and display error conditions reported by the HCs, HFCs, and other software modules.

3.2.5.2 Requirements

3.2.5.2.1 Design Requirements

Section 3.1 of the 10 MWe Software/Firmware Functional Requirements Specifications states a requirement for control and monitoring of heliostats in all modes. This implies a need for real-time detection and display of abnormal heliostat responses.

3.2.5.2.2 Derived Requirements

Section 3.2.1.5 of the 10 MWe Software/Firmware Functional Requirements Specification imposes the following requirements on the alarms module:

- a. Monitor the heliostat status being returned from the field;
- b. Detect error conditions reported by the HC in that status;
- c. Report the alarms to the HAC operator using the alarms printer and an alarms area of the CS control console;
- d. Send alarm messages to the OCS through the OCS interface;
- e. Display alarms detected by other software modules; and
- f. Maintain ONLINE/OFFLINE status for the heliostats.

3.2.5.3 Design Approach

3.2.5.3.1 Functional Allocations

ALARMS processing decomposes logically into three major functions: alarms monitoring, alarms queueing, and alarms output. Figure 3.2.5-1 provides a system overview relating these three functions to each other and to certain relevant system elements.

The alarms monitoring function satisfies those requirements to periodically examine all heliostat status locations for conditions that are abnormal and demand operator attention. Additionally, the alarms monitoring function includes the capability to set individual heliostats to an offline mode as required by item (f) above.

ALARMS PROCESSING

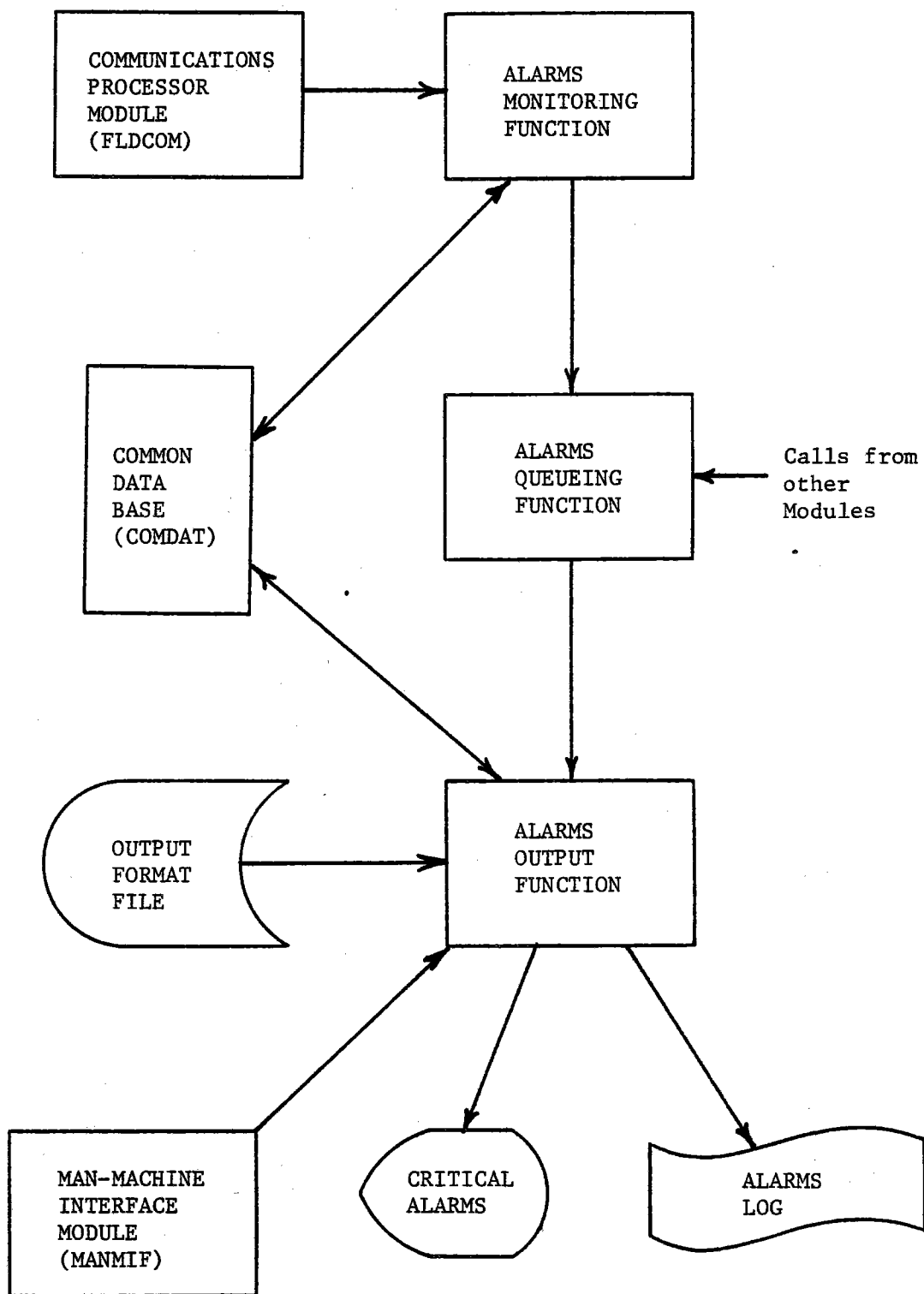


Figure 3.2.5-1 Alarms Processing Overview

The alarms output function exists to provide the visible result of the alarms monitoring activities. For specific HC problems, it normally provides detailed alarms output but, when necessary, concentrates multiple HC alarms into higher level messages that display essential information in an abbreviated format.

The alarms queueing function provides for interfacing the high-speed activities of the alarms output function. It is designed to minimize the possibility of any alarm data being lost while accommodating the response time of an operator interface via the CRT. Additionally, the stand-alone nature of the alarms queueing function provides for an easy interface to accept alarm data from other system modules as required by item (e) above.

A brief description of the functions of Alarms Detection, Alarms Collection, Alarms Reporting, Alarms Queueing, and Alarms Output is presented in the remainder of this subsection. Figure 3.2.5-2 shows the four "levels" of status tables referred to in the text, as well as related tables which are discussed later in this ALARMS module section.

Alarms Detection - Consists of a main submodule (ALMDTC) that integrates alarms detection between status levels, four specialized submodules (ALMFLD, ALMLNE, ALMHFC, and ALMHC) that perform the alarms detection at each level on an individual block basis, and a support submodule (ALMGET) that determines the table-derived alarm number. The detection process proceeds from the lowest level status blocks to the highest level block, and existence of an unreported alarm anywhere in the system results in a flag eventually being set at the highest level. Thus, HC alarms propagate to HFC alarms, HFC alarms to line alarms, and line alarms to field alarms. The field alarm flag then triggers the alarms collection function for analysis of the situation.

The general logic flow of the four specialized submodules is similar in nature, but the specific error and offline testing is directly related to the structure of the corresponding status words in the common data base. Once the existence of an alarm has been determined, the alarm number is computed by ALMGET using the proper alarm detection table. The error testing is performed in order of alarm significance such that less important alarms never replace more important alarms. The alarm number maintained in the status word is always the most critical alarm currently in existence regardless of whether lesser unreported alarms exist. The detection of any alarm at any level results in a Type One alarm being propagated to next higher level, unless a different alarm already exists at the higher level. The lower-level status block is set with the actual alarm condition number and the reported alarm flag is cleared for all new alarms. The alarms collection function eventually executes and keys on the Type One alarm flags in its collection process.

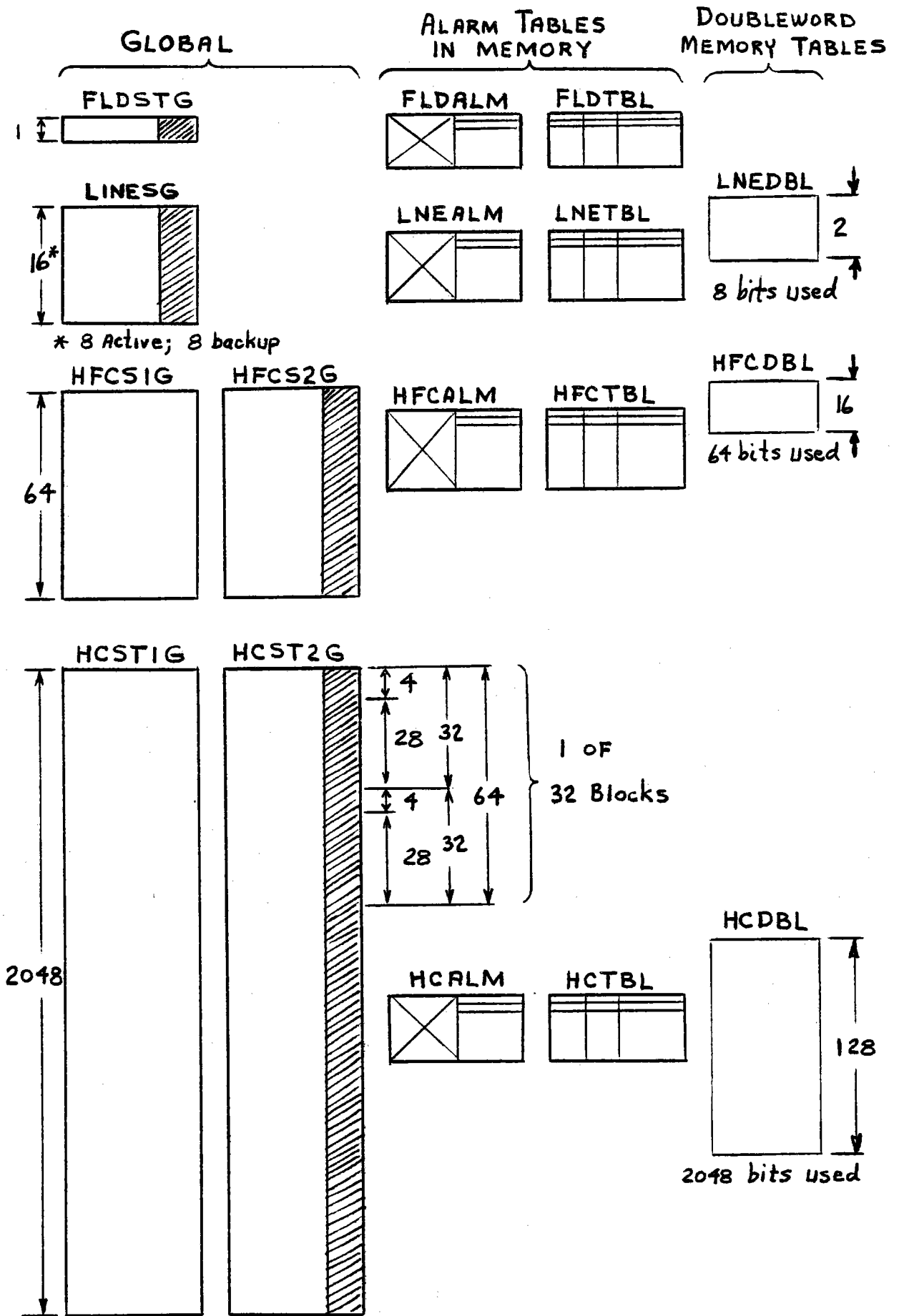


Figure 3.2.5-2 Alarms Status Levels and Related Tables

Alarms Collection - This function is invoked only if alarms detection has indicated the existence of a new alarm condition. Alarms collection is accomplished by a single submodule (ALMCLT) that collects alarms at a given level, converts them into an alarm at the next higher level, and maintains doubleword memories for multiple error conditions. Alarms collection proceeds in a bottom-up manner through three successive calls to ALMCLT to collect HC alarms to HFC alarms, HFC alarms to line alarms, and line alarms to field alarms. When the collection process terminates, the data base status words are set with all appropriate alarm indicators to be processed by the alarms reporting function.

ALMCLT uses alarm Types One and Two as the key for its processing. Type One means that at least one lower-level alarm exists and a search must be made to determine if multiple alarms exist. Multiple alarms are then flagged with a Type Two alarm. If a Type Two alarm has been previously detected, the lower-level blocks are searched to insure that the condition still exists. The Type Two alarm is cleared if the multi-alarm condition no longer exists. All new alarms are flagged as unreported.

Alarms Reporting - This function executes only if the alarms collection functions have been performed. The alarms reporting function consists of a general submodule (ALMRPT) that executes once and a support submodule (ALMBLK) that is called once for each alarm condition that must be reported. Alarms reporting proceeds in a top-down direction so that low-level alarms reporting is suppressed when higher-level alarms are in existence. The alarms reporting function interfaces with the alarms queueing function and updates the common status flags as alarms are reported.

ALMRPT sets up the processing loops for each level of status and tests the alarm reported bit for each level. If required, a call is made to ALMBLK for processing of the actual alarm. The alarm number is used as an index into the proper alarm definition table, and the input for subroutine ALMQUE is constructed. The reported flag is set only after the ALMQUE submodule returns a flag indicating successful queueing of the alarm. Thus, if queueing is unsuccessful, the alarm report attempt will be retried after succeeding status updates until the queueing is successful or the alarm condition disappears.

Alarms Queueing - This function provides the interface between alarms monitoring and alarms output. It also provides an interface for other modules in the system that wish to output alarm messages. This function consists of the ALMQUE submodule which builds and queues compressed message blocks into a chain of free storage blocks. To prevent storage overflow, the chain is not allowed to exceed an arbitrary length. When the chain is full, the oldest alarm

message block is discarded and the newest message block added. This would normally occur only when one of the output devices has failed or the console operator is ignoring critical alarm messages. A queue-full message (unsuccessful queue) is returned only when the free storage system cannot provide a block for the new alarm message, or the ALMQUE work area is full.

Alarms Output - This function consists of the alarms output task (ALO) and support submodules to build ASCII messages from data blocks (ALOBLD and ALOCVT) and to dequeue expended message blocks (ALODQU). The ALO task is activated by the alarms queueing function when new alarm messages are added to the queue. It can also be activated by the MMI task when the console operator acknowledges the presence of a critical alarm on the screen. ALO executes at a lower priority level than either the alarms monitor task or the MMI task. All alarm formats are stored on disk by an off-line initialization program that is described in the initialization module (DBINIT) documentation. Four formats are stored on each sector, and the records are stored in the order of the associated alarm code numbers. The message formats themselves are defined in the same manner as FORTRAN format statements and are capable of handling H, X, F, E, A, and Z data conversions.

Normally, submodule ALOBLD reads message prototypes from disk, inserts the user parameters into the prototype, appends the time to the message, and writes it to the specified device. However, special processing is provided for the first five message types. Messages one, two, and three include parameters that define which lines, HFCs or HCs have errors in a multiple alarm message. The particular units that are defective are designated by the appropriate bits being set in a doubleword parameter so that the amount of storage space is minimized. ALOBLD calls ALOCVT to decode the compressed format and creates the final message output format. Message Types Four and Five relate to Mark conditions and require a disk read of the Mark biases to complete the message. All alarms are output to the general alarms portion of the operator's screen while only critical alarms are sent to the area of the screen that requires an operator response before the next critical alarm will be displayed.

3.2.5.3.2

Resource Budgets

15,000 bytes of core to keep alarms processing software resident.

8,192 bytes of core for free-storage buffer area.

6,400 bytes of disk space for alarm message formats.

1 percent of CPU time for monitoring function-no alarms processing.

CRT with two 80-character alarms display areas.

Printer logging device.

Priorities:

ALM - relatively high (synchronous)

ALO - relatively low (asynchronous)

3.2.5.4 Design Description

3.2.5.4.1 Module Structure

The software organization of the ALARMS processing module consists of two tasks and thirteen subroutines ("tasks" and "subroutines" take on the meaning defined by MODCOMP). Refer to Figure 3.2.5-3 for an overview of the ALARMS module structure.

Tasks:

- a. ALM - Alarms Monitoring - main routine which initiates all Alarms Processor Module actions; and
- b. ALO - Alarms Output - handles the output of all system alarms to the various output devices.

Subroutines:

- a. ALMDTC - Alarm Detection - Integrates the alarm status for two contiguous levels;
- b. ALMFLD - Field Test - Performs alarms detection at the field level;
- c. ALMLNE - Line Test - Performs alarms detection at the line level;
- d. ALMHFC - Heliostat Field Controllers Test - Performs alarm detection at the HFC level.
- e. ALMHC - Heliostat Controllers Test - Performs alarms detection at the HC level;
- f. ALMGET - Get Alarm - Does the status bit testing to determine which alarm bit is set, and returns an alarm number for that status bit;
- g. ALMCLT - Alarm Collect - Collects alarms reported at a lower status level, sets a higher-level multiple alarm flag if necessary, and maintains memory doublewords of all multiple alarms;
- h. ALMRPT - Alarm Report - Searches all status from the top level to the HC level and reports the highest-level alarms that exist;

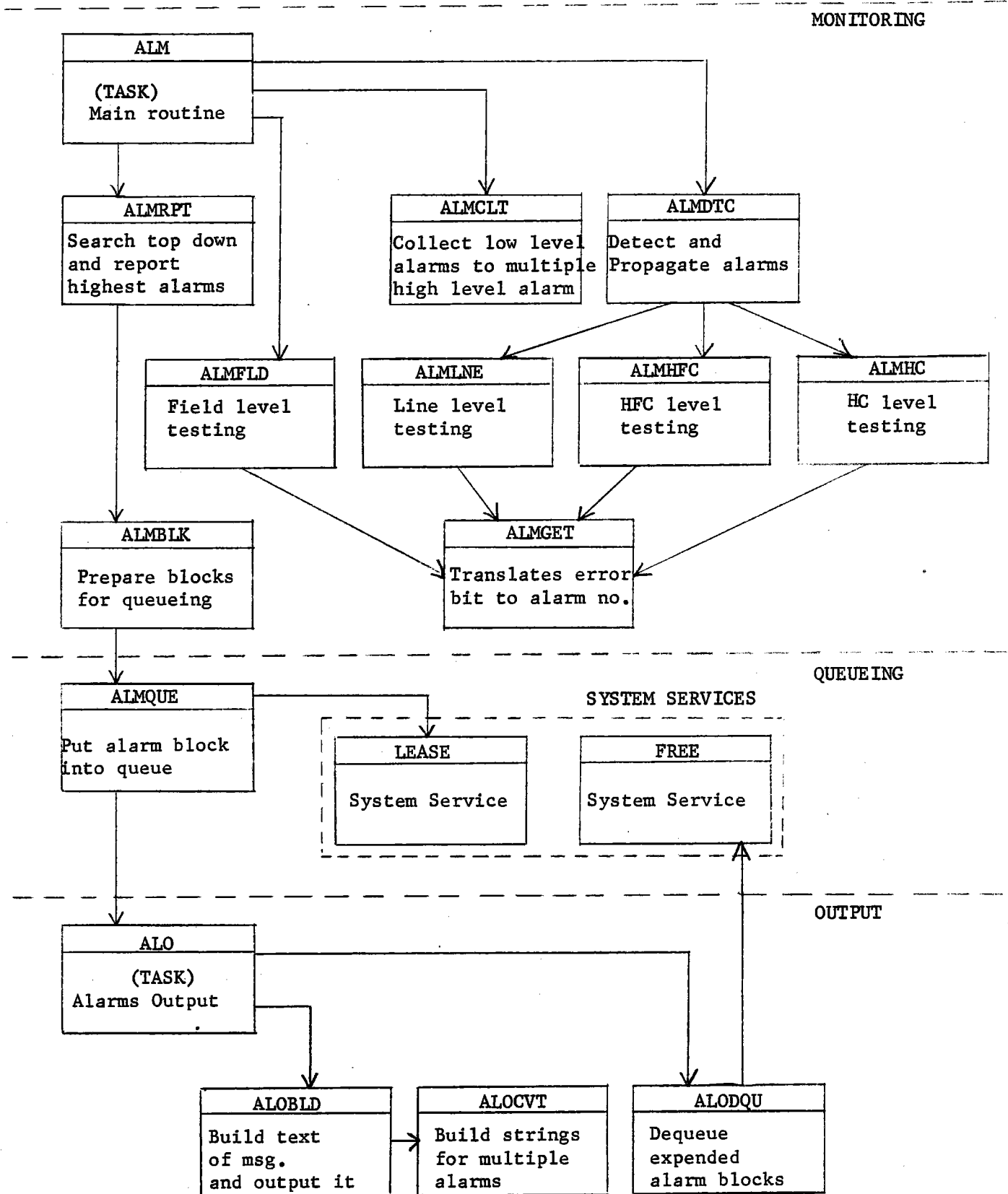


Figure 3.2.5-3 Alarms Module Structure

- i. ALMBLK - Alarm Block - Handles the interface between the Alarms Monitor Task (ALM) and the alarms queueing system by organizing the parameters for a call to ALMQUE;
- j. ALMQUE - Alarm Queue - Can be called by Alarms Processor module (or any other system program) to build a compressed message block from passed parameters and to queue up the message block;
- k. ALOBLD - Message Build - Reads the disk to build alarm messages from queued message blocks;
- l. ALODQU - Deque - Handles the alarm message chain and FREES message blocks when all required output has been performed; and
- m. ALOCVT - Convert - Creates a list of items for multiple alarms formats.

3.2.5.4.1.1 Submodule I - ALM

3.2.5.4.1.1.1 Description

- a. Language used - MODCOMP Assembly
- b. How invoked - REX, ACTIVATE issued once a second by FLDCOM
- c. Constraints and limitations - None
- d. Processing - The Alarms Monitor submodule (ALM) is the central element in the alarms processing system and executes at a relatively high-priority level in the system task structure. It provides all of the field, line, HFC and HC alarm detection and reporting in the system. It also assumes responsibility for setting individual heliostats offline after errors occur and clears alarm flags when conditions return to normal.
 - 1. ALM accesses CURHSG to determine which one-eighth of the 2048-word HC status arrays is to be monitored for alarm conditions.
 - 2. ALM calls ALMDTC, which calls ALMHC four times, checking four online HC status words and maintaining the highest alarm for each. ALMDTC loops 64 times till 256 HCs are processed.
 - 3. ALM calls ALMDTC which calls ALMHFC eight times, checking eight HFC status words (only if installed) and maintaining the highest alarm for each. ALMDTC loops 8 times till 64 HFCs are processed.

4. ALM calls ALMDTC which calls ALMLNE eight times, checking eight line status words and maintaining the highest alarm for each.
5. ALM calls ALMFLD to test the field status word.
6. If an alarm change has not been detected, ALM terminates; otherwise, ALM continues as follows.
7. ALM calls ALMCLT which steps through 32 words of HC status if required. The alarms are counted to see if a multiple alarm situation exists. The HC alarm bit table, HCDBL, is updated. ALMCLT loops 64 times until all 2048 HCs are processed.
8. ALM calls ALMCLT which steps through eight words of HFC status if required. The alarms are counted to see if a multiple alarm situation exists. The HFC alarm bit table, HFCDBL, is updated. ALMCLT loops eight times until all 64 HFCs are processed.
9. ALM calls ALMCLT which steps through eight words of line status if required. The alarms are counted to see if a multiple alarm situation exists. The line alarm bit table, LNEDBL, is updated.
10. ALM calls ALMRPT to report alarms to the queuing system. ALM then terminates.

To summarize, ALM is activated once each second by the communications module after one-eighth of the heliostat field status has been updated. ALM then scans this updated HC status as well as all HFC, line, and field status and sets any necessary error conditions into global common memory. Alarm detection is the only function performed by ALM when no errors exist, so the task execution time requirement is minimal. Detection of alarm conditions forces execution of the collection and reporting subfunctions, and the execution time increases in proportion to the number of alarms. If an abnormally large number of alarms occur, the ALM task is designed to delay detection processing and catch up as time becomes available.

e. Error messages and recovery - None

3.2.5.4.1.1.2 Data, Logic and Command Paths

- a. Input Description - All input is through global common;
- b. Output Description - No direct output;
- c. Submodules called - ALMDTC, ALMFLD, ALMCLT, and ALMRPT; and

d. Global common usage - CURHSG.

3.2.5.4.1.1.3 Internal Data Description

The doubleword memory tables HCDBL, HFCDBL, and LNEDBL (reference Figure 3.2.5-2) are physically located here. The use of these tables is described in Submodule VIII, ALMCLT (see Section 3.2.5.4.1.8).

3.2.5.4.1.1.4 Flowcharts

See Figure 3.2.5-4.

3.2.5.4.1.2 Submodule II - ALMDTC

3.2.5.4.1.2.1 Description

a. Language used - MODCOMP Assembly

b. How invoked - Calling Sequence:

BLM,9	ALMDTC
DFC	HIPTR
DFC	HICNT
DFC	LOPTR
DFC	LOCNT
DFC	SKPCNT
DFC	TSTSUB

c. Constraints and limitations - None

d. Processing - ALMDTC integrates the alarm status for two continuous status levels. It calls the specified test subroutine for each block at the lower level and sets alarm flags at the higher level when necessary.

1. The passed parameters are accessed.
2. A high (outer) loop is set up to start at HIPTR and loop HICNT times, incrementing the LOPTR by SKPCNT each time through the loop.
3. A low (inner) loop is set up to start at LOPTR and loop LOCNT times, calling submodule TSTSUB each time through the loop.
4. Execution of the loops begins. LOPTR and HIPTR point to global common status arrays, a

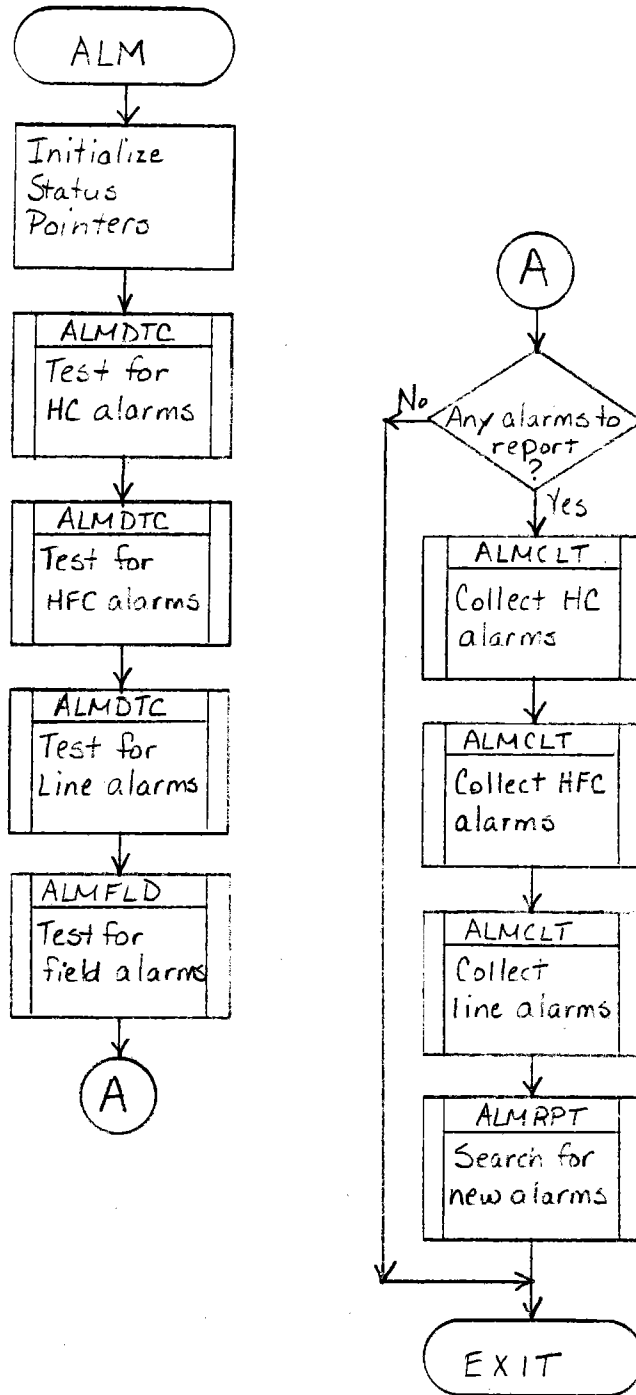


Figure 3.2.5-4. Flowchart - ALM

low level and a higher level. If the calls to TSTSUB detect an Alarm change stepping through the low loop, and the corresponding high loop status word has no alarm of its own, the high-level status word is set to alarm one, thus propagating low-level alarms upward.

5. When the inner and outer loops both complete execution, the submodule exits.

e. Error messages and recovery - None

3.2.5.4.1.2.2 Data, Logic and Command Paths

a. Input description

HIPTTR - Pointer to high-level status array

HICNT - Number of high-level blocks

LOPTR - Pointer to low-level status array

LOCNT - Number of low-level blocks tested per high-level block

SKPCNT - Number of low-level blocks not tested per high-level block

TSTSUB - Pointer to low-level status testing submodule

b. Output description - There is no output in this submodule.

c. Submodules called - TSTSUB can be ALMLNE, ALMHFC, or ALMHC.

d. Global common usage

FLDSTG, LINESG, HFCS2G, HCST1G, and HCST2G.

3.2.5.4.1.2.3 Internal Data Description

There is no data internal to this submodule.

3.2.5.4.1.2.4 Flowcharts

See Figure 3.2.5-5.

3.2.5.4.1.3 Submodule III - ALMFLD

3.2.5.4.1.3.1 Description

a. Language used - MODCOMP Assembly

b. How invoked - FLM,8 ALMFLD

c. Constraints and limitations - None

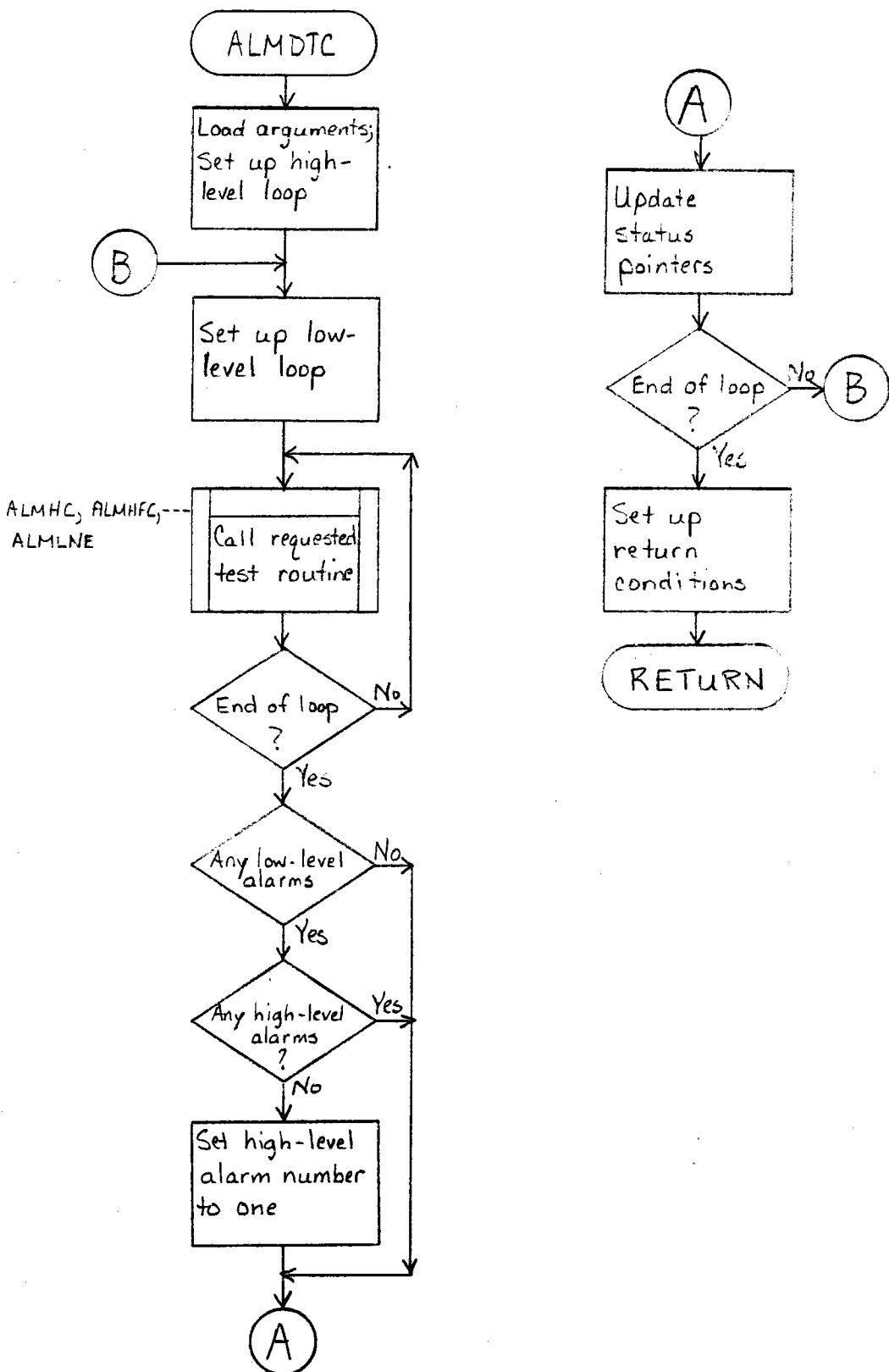


Figure 3.2.5-5 Flowcharts - ALMDTC

d. Processing - This submodule performs alarms detection at the field level. It sets an alarm flag for the highest-order existing error and clears the alarm flag when the error condition disappears.

1. The submodule first checks for error bits in FLDSTG. If none are found and the old alarm was greater than three, it is cleared. If the alarm was one (a propagated alarm) it will be propagated higher. The submodule exits.
2. If an alarm was found, it is resolved with a call to ALMGET. The alarm number is stored in FLDSTG but will not be reported if it has already been reported. The submodule exits.

e. Error messages and recovery - None

3.2.5.4.1.3.2 Data, Logic and Command Paths

- a. Input description
This call must pass registers set up in ALM.
- b. Output description
Register 10 is alarm indicator (1=No Alarm; 0=Alarm)
- c. Submodules called
ALMGET
- d. Global common usage
FLDSTG

3.2.5.4.1.3.3 Internal Data Description

FLDALM table - Field alarm detection table; and
FLDTBL table - Field alarm definition table.

3.2.5.4.1.3.4 Flowcharts
See Figure 3.2.5-6.

3.2.5.4.1.4 Submodule IV - ALMLNE

3.2.5.4.1.4.1 Description

- a. Language used - MODCOMP Assembly
- b. How invoked - BLM,8 ALMLNE
- c. Constraints and limitations - None
- d. Processing - This submodule performs alarms detection at the line level. It sets an alarm flag for the

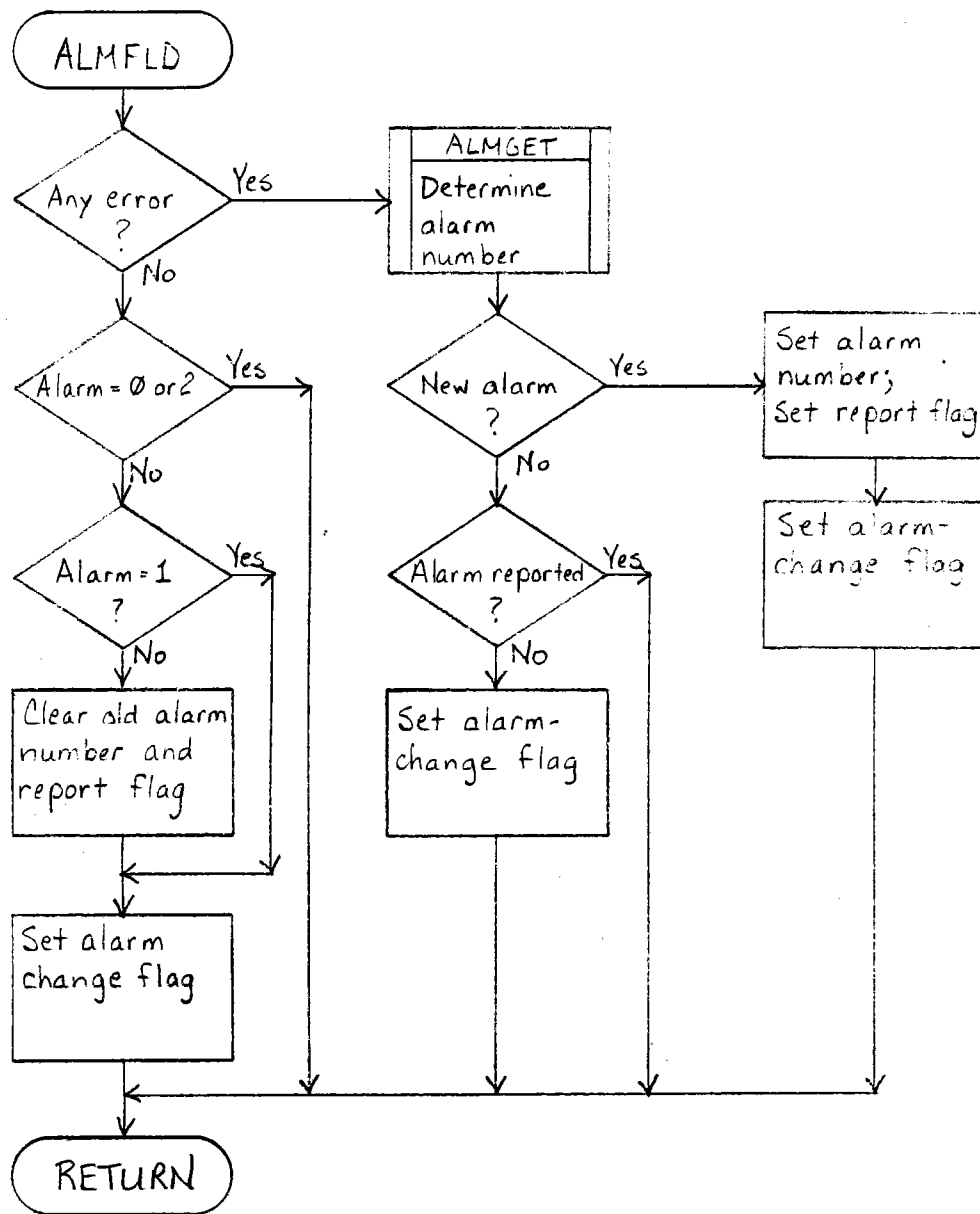


Figure 3.2.5-6 Flowcharts - ALMFLD

highest-order existing error and clears the alarm flag when the error condition disappears.

1. The submodule first checks for error bits in LINESG. If none are found and the old alarm was greater than three, it is cleared. If the alarm was one (a propagated alarm), it will be propagated higher. The submodule exits.
2. If an alarm was found, it is resolved with a call to ALMGET. The alarm number is stored in LINESG but will not be reported if it has already been reported. The submodule exits.

e. Error messages and recovery - None

3.2.5.4.1.4.2 Data, Logic and Command Paths

- a. Input description
This call must pass registers set up in ALM and ALMDTC.
- b. Output description
Register 10 is alarm indicator (1=No alarm; 0=Alarm)
- c. Submodules called
ALMGET
- d. Global common usage
LINESG

3.2.5.4.1.4.3 Internal Data Description

LNEALM - Line alarm detection table; and
LNETBL - Line alarm definition table.

3.2.5.4.1.4.4 Flowcharts

See Figure 3.2.5-7.

3.2.5.4.1.5 Submodule V - ALMHFC

3.2.5.4.1.5.1 Description

- a. Language used - MODCOMP Assembly
- b. How invoked - BLM,8 ALMHFC
- c. Constraints and limitations - None
- d. Processing - This submodule performs alarms detection at the HFC level. It sets an alarm flag for the highest-order existing error and clears the alarm flag when the error condition disappears. When an

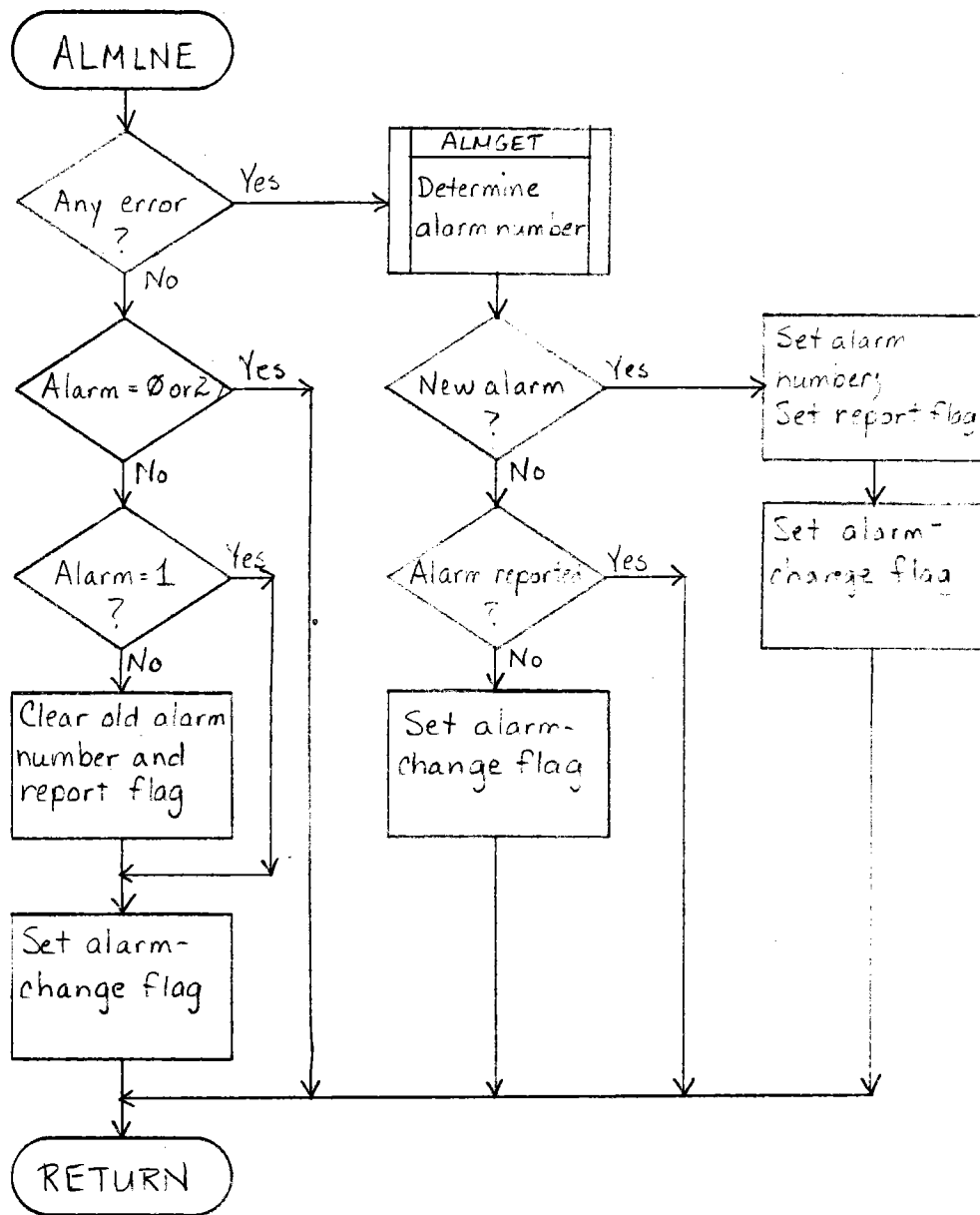


Figure 3.2.5-7 Flowcharts - ALMLNE

HFC communications error is detected, all affected HCs are set offline.

1. This submodule first checks the not-installed bit in HFCS2G and exits if true.
2. The next check is for error bits in HFCS1G. If none are found and the old alarm was greater than three, it is cleared. If the alarm was one (a propagated alarm), it will be propagated higher. The submodule exits.
3. The alarm number is resolved with a call to ALMGET and stored in HFCS2G, but will not be reported if it already has been reported. The submodule exits.

e. Error messages and recovery - None

3.2.5.4.1.5.2 Data, Logic and Command Paths

- a. Input description
This call must pass registers set up in ALM and ALMDTC.
- b. Output description
Register 10 is alarm indicator (1=No alarm; 0=Alarm)
- c. Submodules called
ALMGET
- d. Global common usage
HFCS1G, HFCS2G

3.2.5.4.1.5.3 Internal Data Description

HFCALM - HFC alarm detection table
HFCTBL - HFC alarm definition table

3.2.5.4.1.5.4 Flowcharts

See Figure 3.2.5-8.

3.2.5.4.1.6 Submodule VI - ALMHC

3.2.5.4.1.6.1 Description

- a. Language used - MODCOMP Assembly
- b. How invoked - BLM,8 ALMHC
- c. Constraints and limitations - None

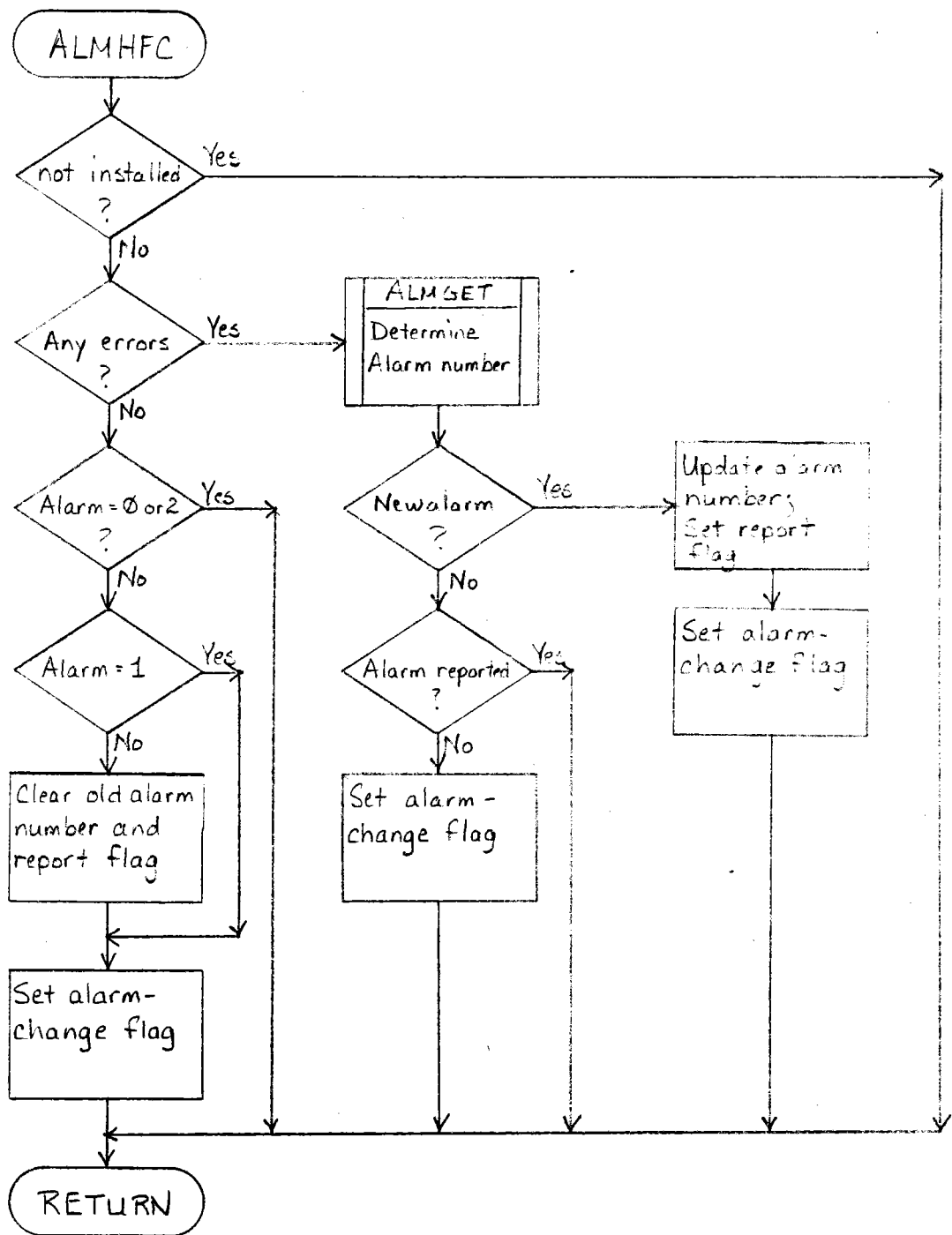


Figure 3.2.5-8 Flowchart - ALMHFC

d. Processing - This submodule performs alarms detection at the HC level. It sets an alarm flag for the highest-order existing error and clears the alarm flag when the error condition disappears. It sets the HC offline bits as necessary following alarms detection.

1. This submodule first checks the offline bits in HCST2G and exits if true.
2. The next checks are for error bits in HCST1G (including communication error), then HCST2G, then a special test for UNMARKED status. If no errors are found and the old alarm was greater than three, it is cleared and the submodule exits.
3. If an alarm was found, the alarm number is resolved with a call to ALMGET and stored in HCST2G, but will not be reported if it already has been reported. The submodule exits.

e. Error messages and recovery - None

3.2.5.4.1.6.2 Data, Logic and Command Paths

- a. Input description
This call must pass registers set up in ALM and ALMDTC.
- b. Output description
Register 10 is alarm indicator (1=No alarm; 0=Alarm)
- c. Submodules called
ALMGET
- d. Global common usage
HCST1G, HCST2G

3.2.5.4.1.6.3 Internal Data Description

HCALM - HC alarm detection table
HCTBL - HC alarm definition table

3.2.5.4.1.6.4 Flowcharts

See Figure 3.2.5-9.

3.2.5.4.1.7 Submodule VII - ALMGET

3.2.5.4.1.7.1 Description

a. Language used - MODCOMP Assembly

b. How invoked -

BLM,14	ALMGET
DFC	ALMTBL
DFC	TBLEN

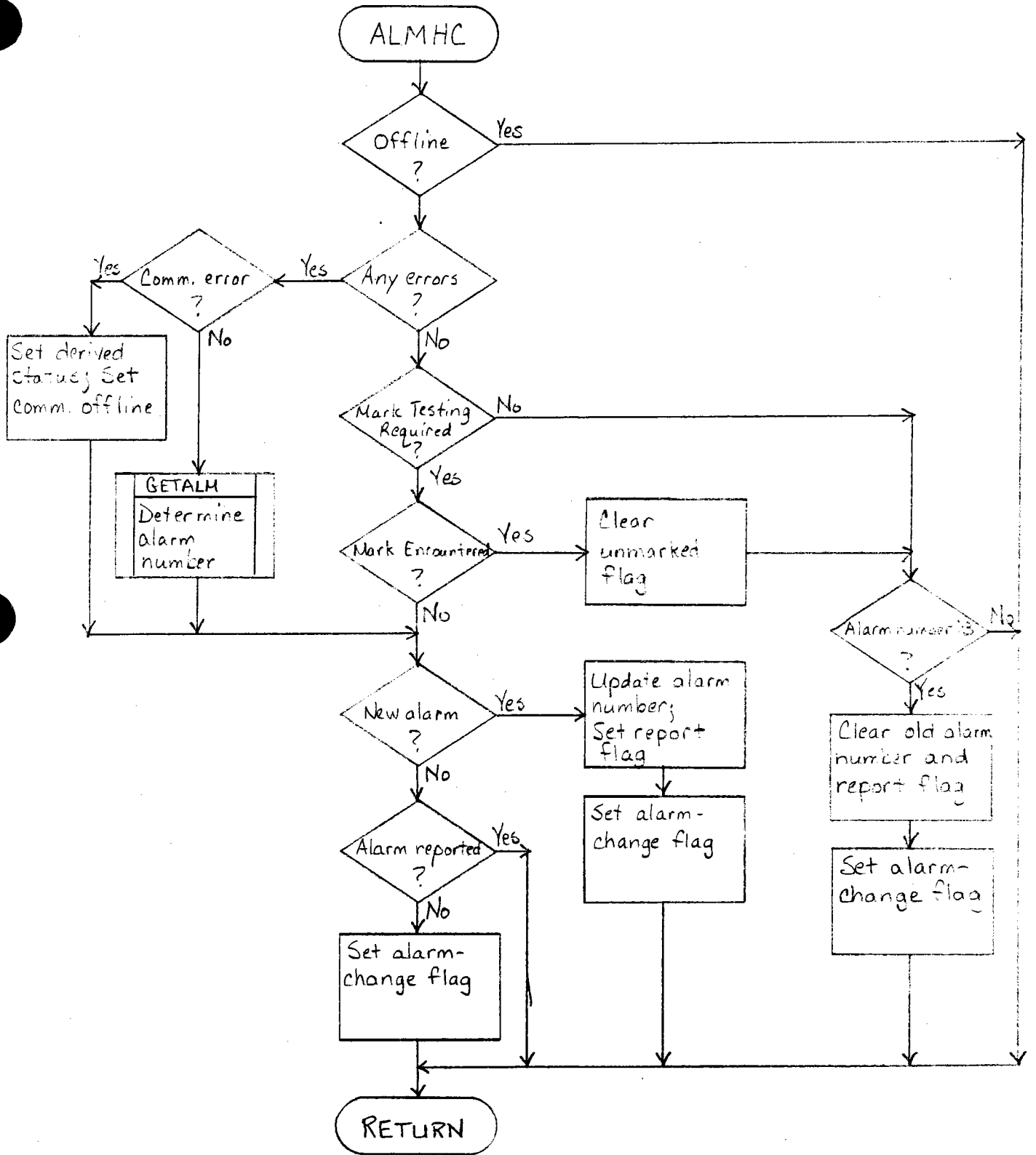


Figure 3.2.5-9 Flowcharts - ALMHC

- c. Constraints and limitations - This submodule is called with a generic parameter, ALMTBL, which can be FLDALM, LNEALM, HFCALM, or HCALM as shown in Figure 3.2.5-2. The contents of these alarm detection tables must be arranged in descending priority order, and a parallel table (FLDTBL, LNETBL, HFCTBL, or HCTBL) must exist. Refer to Table 3.2.5-I.
- d. Processing - This submodule does the status bit testing and translates the highest priority status bit into an alarm number (four through fifteen) which will fit into the four-bit field of the status word. The process is table driven by the contents of ALMTBL, and the output is correlated to the position within the table where the match occurs.
- e. Error messages and recovery - None

3.2.5.4.1.7.2 Data, Logic and Command Paths

a. Input description

ALMTBL - Pointer to table specifying the register and bit to test for each alarm.

TBLN - Length of ALMTBL

R12 - Current actual status

R13 - Current derived status

NOTE: Each ALMTBL table entry is a two-digit hexadecimal number that specifies the location and bit number for each alarm. If the first digit is "C", the actual status location is to be interrogated, while a "D" means the derived status location is to be used. The second digit specifies the bit number to be tested in either the actual or derived status. Refer to Table 3.2.5-II.

b. Output description

R14 - Alarm number for four-bit field in status word

c. Submodules called

None

d. Global common usage

None

3.2.5.4.1.7.3 Internal Data Description

There is no data internal to this submodule.

3.2.5.4.1.7.4 Flowcharts

See Figure 3.2.5-10.

¹ ALARM DETECTION TABLE NAME	² ALARM DEFINITION TABLE NAME	LOCATION	PROVIDES ACTUAL ALARM NOS. FOR
HCALM	HCTBL	ALMHC	HC
HFCALM	HFCTBL	ALMHFC	HFC
LNEALM	LNETBL	ALMLNE	LINE
FLDALM	FLDTBL	ALMFLD	FIELD

1. Alarm detection tables

- a. A table of DFCs arranged in descending priority
- b. DFC format: # X Y, X = #C or #D
 register status bit
- c. ALMGET is entered with a status word(s) in register(s) X. ALMGET steps through the Alarm Detection Table doing a Test Bit in Register instruction for each table entry until the status error bit is found; ALMGET returns the index value (0, 1, 2, . . .) of the table entry +4 (i.e. 4 through 15 max.).

2. Alarm definition tables

- a. A table of DFCs arranged in the same order as 1; the table effectively starts at 2 (2, 3, 4, . . . 15 max.) relative to the value returned by ALMGET in c. above. Thus, a value of 4 from ALMGET points to the 3rd entry in the Alarm definition table.
- b. DFC format: # A B C D
 | | | |
 Level No. parameters MSG No. actual (range 0-255)
 0 = non-critical
 1 = critical

HC ALARMS (HCALM)

#C5	HC Communications Error
#DA	Missing Command Return
#C6	No Motion Error (AZ or EL)
#D7	HC Unavailable (WASH/OFFLINE) to STHIWIND
#D6	HC Cannot Return to SAVE Position
#C4	EL MARK Encountered
#C3	AZ MARK Encountered
#D8	Sequence Command Timed Out
#D9	Extra Command Returned

HFC ALARMS (HFCALM)

#C1	HFC Detected HAC/HFC Communications Error
#C2	Firmware Error
#C5	HAC to HFC Communications Output Timed Out
#C6	HAC to HFC Communications Input Timed Out
#C7	Invalid Data Received from HFC
#C8	HFC Did Not Receive Last Command

LINE ALARMS (LNEALM)

#C0	Line Communications Output Error
#C1	Line Communications Input Error

FIELD ALARMS (FLDALM)

#C3	Field Power Loss
#C4	Communication Failure With Field

Table 3.2.5 - II ALARM Detection Tables

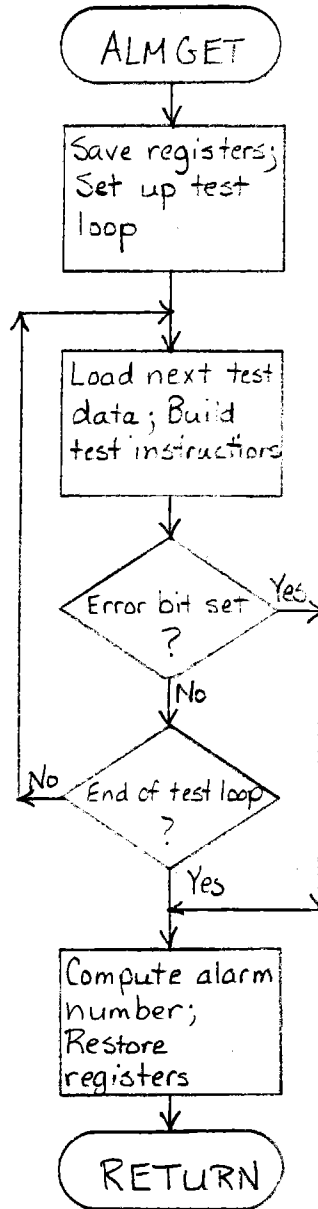


Figure 3.2.5-10 Flowcharts - ALMGET

3.2.5.4.1.8 Submodule VIII: ALMCLT

3.2.5.4.1.8.1 Description

a. Language used - MODCOMP Assembly

b. How invoked - Calling Sequence:

BLM,8	ALMCLT
DFC	FLGPTR
DFC	HIPTR
DFC	HICNT
DFC	LOPTR
DFC	LOCNT

c. Constraints and limitations - None

d. Processing - ALMCLT collects all alarms reported at a lower status level and sets a higher-level multiple-alarm flag if necessary.

1. The passed parameters are accessed.
2. A high (outer) loop is set up to start at HIPTR and loop HICNT times.
3. A low (inner) loop is set up to start at LOPTR and loop LOCNT times.
4. Execution of the loops begin. HIPTR and LOPTR point to global common status arrays, a high level and a low level. If the high-level alarm is greater than three, the high-level loop proceeds to the next high-level status word. If the high-level alarm is one or two, the low-level status words (LOCNT of them) are checked for any alarm, and if one is found, it is counted and a corresponding bit is set into the doubleword flag table pointed to by FLGPTR. If the total count exceeds one at the end of the low-level loop, the high-level error field is set to two to indicate a multiple alarm condition; otherwise it is cleared. If the doubleword contents has changed from the last time this subroutine was executed, the high-level status is marked as unreported.
5. When the inner and outer loops both complete execution, the submodule exits.

e. Error messages and recovery - None

3.2.5.4.1.8.2 Data, Logic and Command Paths

a. Input Description:

FLGPTR - Pointer to alarm flag array
HIPTR - Pointer to higher-level status array
HICNT - Number of higher-level status words
LOPTR - Pointer to lower-level status array
LOCNT - Number of lower-level status words

b. Output Description - Output is to the global common status tables;

c. Submodules Called - None; and

d. Global Common Usage -
FLDSTG, LINESEG, HFCS2G, HCST2G.

3.2.5.4.1.8.3 Internal Data Description

There is no data internal to this submodule.

3.2.5.4.1.8.4 Flowcharts

See Figure 3.2.5-11.

3.2.5.4.1.9 Submodule IX: ALMRPT

3.2.5.4.1.9.1 Description

a. Language used - MODCOMP Assembly

b. How invoked - BLM,1 ALMRPT

c. Constraints and limitations - None

d. Processing - ALMRPT searches through all status from the top level to the HC level and reports the highest-level alarms that exist.

1. ALMRPT starts at the highest level and works down (FLDSTG, LINESEG, HFCS2G, HCST2G), checking for alarms.
2. If a FLDSTG alarm exists and is not reported, ALMBLK is called; ALMRPT then exits.
3. If no FLDSTG alarm exists, the eight words of LINESEG are checked for alarms. If an alarm is found and not reported, ALMBLD is called. Then all eight HFC and all (8 * 32) HC alarms (if any) are skipped for that line, and the next line is checked.

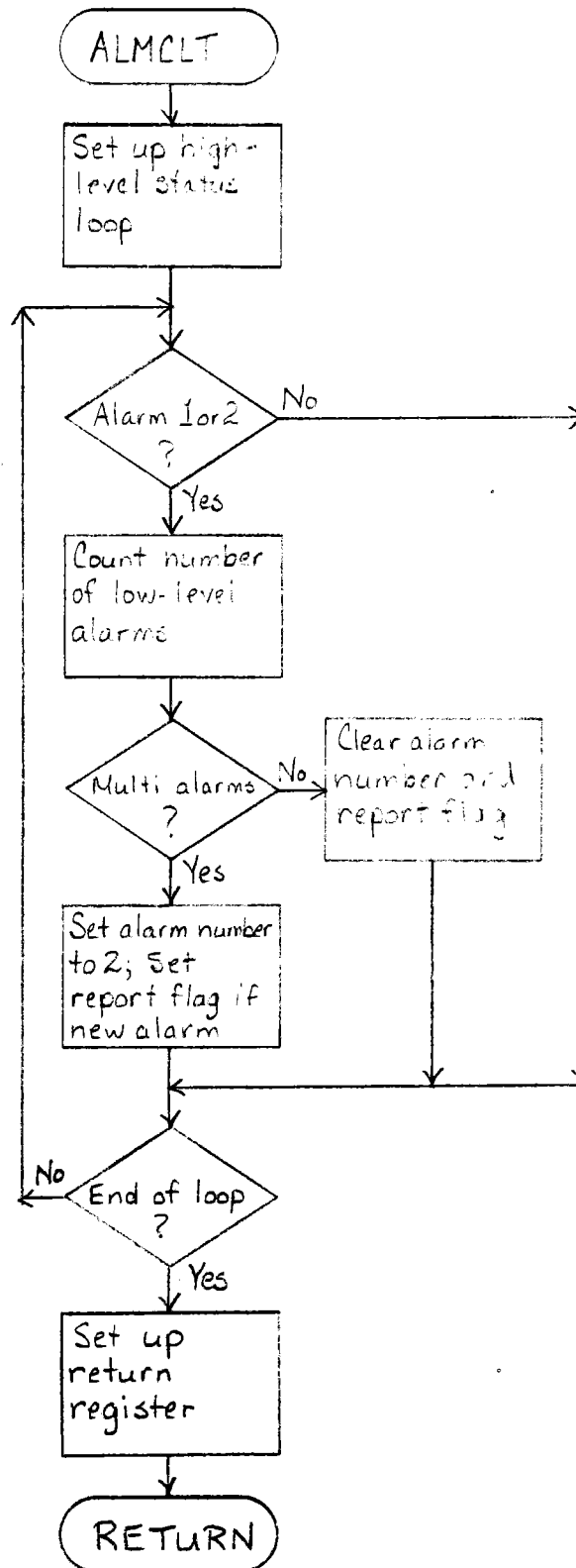


Figure 3.2.5-11 Flowcharts - ALMCLT

5. If the HFCS2G word has no alarm, the 32 HCs associated with the HFC are checked. If an alarm is found in HCST2G and not reported, ALMBLK is called. The next HC is then checked.

6. The submodule exits.

e. Error messages and recovery - None

3.2.5.4.1.9.2 Data, Logic and Command Paths

- a. Input Description
Global common status arrays
- b. Output Description
Global common status arrays
- c. Submodules Called
ALMBLK
- d. Global Common Usage
LINESG, FLDSTG, HFCS2G, and HCST2G.

3.2.5.4.1.9.3 Internal Data Description

There is no data internal to this submodule.

3.2.5.4.1.9.4 Flowcharts

See Figure 3.2.5-12.

3.2.5.4.1.10 Submodule X: ALMBLK

3.2.5.4.1.10.1 Description

- a. Language used - MODCOMP Assembly
- b. How invoked - Calling Sequence:

BLM,8	ALMBLK
DFC	TABLE
DFC	UNIT

- c. Constraints and limitations - None
- d. Processing - This submodule handles the interface between the alarms monitor function and alarms queueing. It sets up the alarm parameters for each type of alarm and calls ALMQUE. It sets the alarm reported flag once the alarm has been successfully queued.

ALMBLK isolates the four-bit error field (error numbers one to fifteen) from the status word and transforms it,

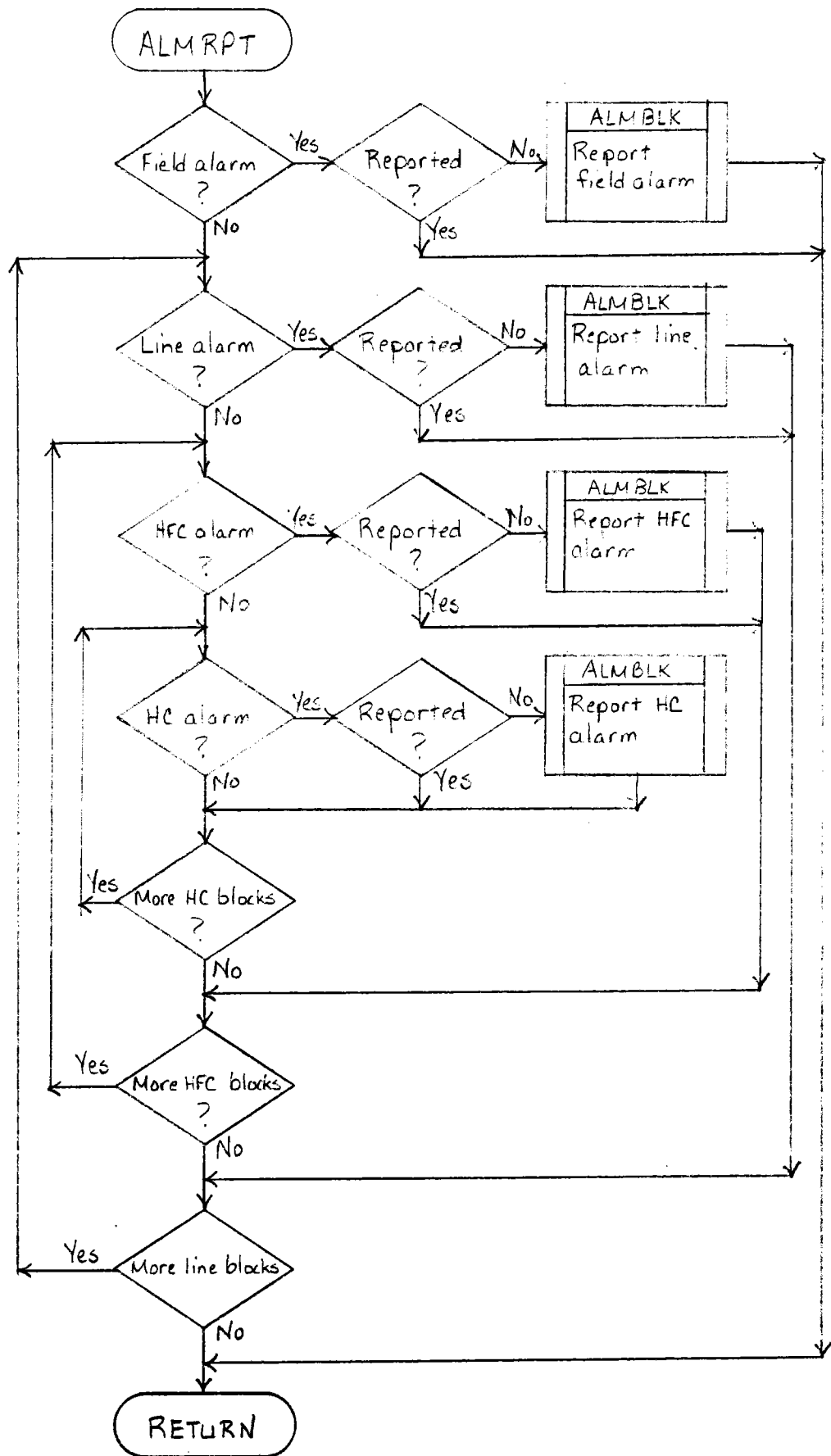


Figure 3.2.5-12 Flowcharts - ALMRPT

using the appropriate Alarm Definition Table, into an actual message code, a parameter count, and level flag. If AZ and EL are required parameters, they are pulled from global common AZIMG and ELEVG. The 32 flag bits of the next lower level are pulled (for multiple alarm reporting). These data values become parameters for a call to ALMQUE.

e. Error messages and recovery

3.2.5.4.1.10.2 Data, Logic and Command Paths

a. Input Description

TABLE	-	Pointer to Alarm Definition Table
UNIT	-	Number of Field, Line, or HFC
Register 2	-	Status word pointer
Register 5	-	HC index
Register 6	-	Multiple alarm doubleword
Register 7	-	

b. Output Description

None

c. Submodules Called

ALMQUE

d. Global Common Usage

FLDSTG, LINESG, HFCS2G, and HCST2G.

3.2.5.4.1.10.3 Internal Data Description

This is the data for the call to ALMQUE:

PARMS	+0	UNIT
	+1	
	+2	
	+3	32 bit doubleword
	+4	HFC No.
	+5	HC No.
	+6	AZ
	+7	EL
MSGCODE		MSG #
NPARMS		# Parameters
LEVEL		Level
QFUL		Flag to be tested (0 = not full; 1 = queue full)

3.2.5.4.1.10.4 Flowcharts

See Figure 3.2.5-13.

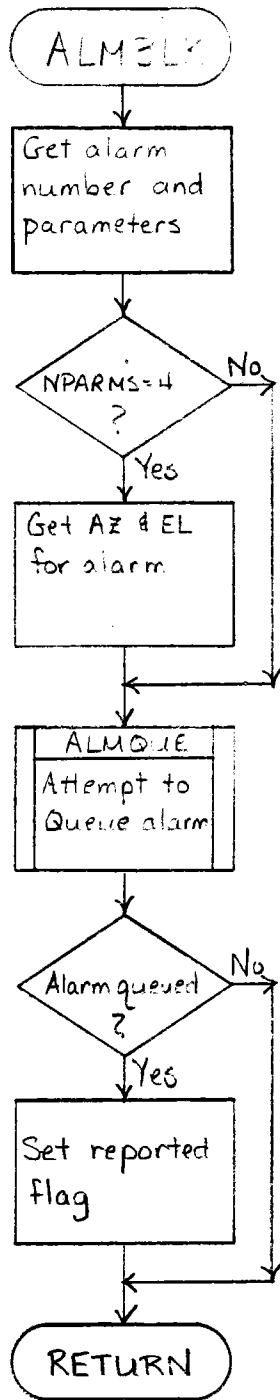


Figure 3.2.5-13 Flowcharts - ALMBLK

3.2.5.4.1.11 Submodule XI - ALMQUE

3.2.5.4.1.11.1 Description

a. Language Used - FORTRAN

b. How Invoked - Calling Sequence

Call ALMQUE (MSGNUM, NPARMS, PARS, LEVEL, INDIC)

c. Constraints and Limitations - Message formats must be defined on disk. The caller must test INDIC to determine if queueing was successful.

d. Processing - The ALMQUE submodule is the interface program for all system alarm reporting. It can be called by any system module to queue a predefined alarm message to the ALARMS output system.

1. ALMQUE starts building a compressed message block (see Table 3.2.5-III).

2. The caller's parameters, if any, are moved into the block.

3. The ENQUE service is called to put the block into the ALO queue.

4. The caller's indicator (INDIC) is set to reflect the success/failure of the ENQUE.

e. Error messages and recovery - None

3.2.5.4.1.11.2 Data, Logic and Command Paths

a. Input Description

MSGNUM - Message Code Number (INTEGER*2)

NPARMS - Number of Parameters in Message (INTEGER*2)

PARMS - Parameter Array (REAL*4)

LEVEL - Alarm Priority Level (0 = Normal; 1 = Critical) (INTEGER*2)

b. Output Description

INDIC - Success/fail indicator (0 = Success; 1 = Failure) (INTEGER*2)

c. Submodules Called

ENQUE Service

d. Global Common Usage

COMQUE - Common Queue

3.2.5.4.1.11.3 Flowcharts

See Figure 3.2.5-14.

0* 1* 2* 3* 4* 5* 6* 7* 8* 9*10*11*12*13*14*15*			
NEXT			
C*	COUNT*	*NPARMS	MESSAGE CODE NUMBER
TIME			
PARAMETER NUMBER 1			
		.	.
		.	.
		.	.
PARAMETER NUMBER NPARMS			

NEXT = Pointer to Next Alarm (FSB Index)
NPARMS = Number of Parameters
C = Critical Alarm Flag
COUNT = Number of Output Devices
TIME = Integer Seconds Since Midnight

Table 3.2.5-III Compressed Message Format

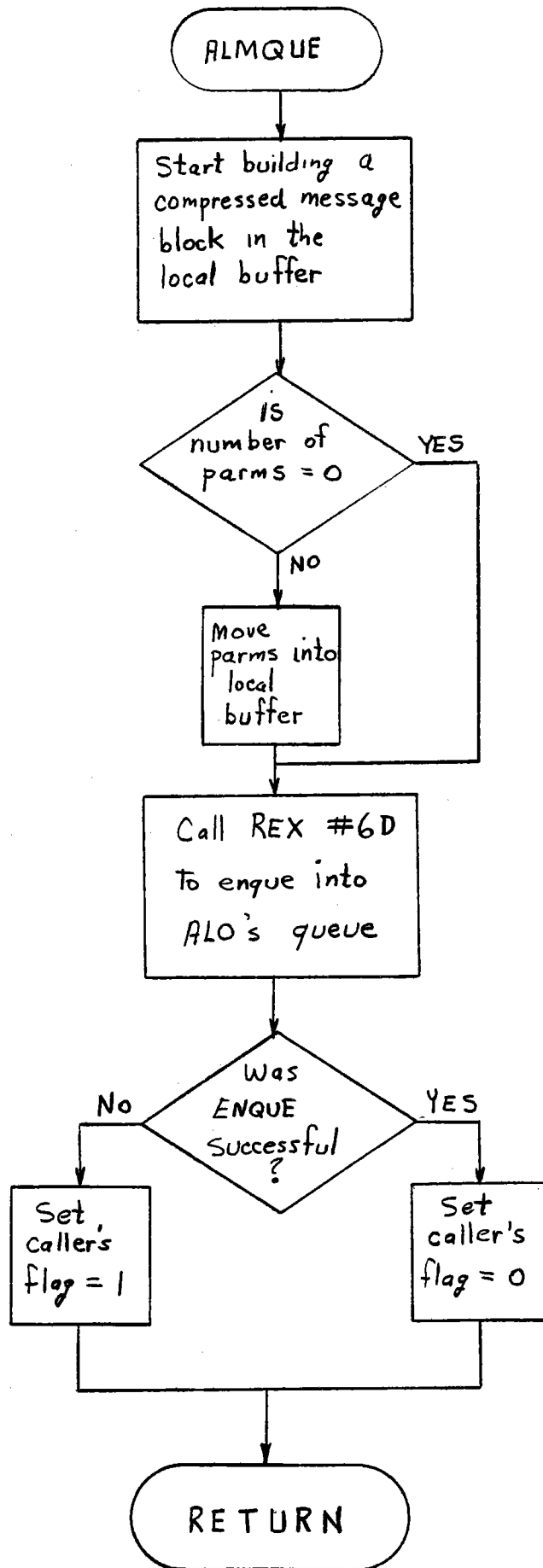


Figure 3.2.5-14 Flowchart - ALMQUE

3.2.5.4.1.12 Submodule XII - ALO

3.2.5.4.1.12.1 Description

- a. Language Used - FORTRAN
- b. How Invoked - Invoked by ALMQUE or MANMIF
- c. Constraints and Limitations - None
- d. Processing - ALO is a submodule that handles the outputting of all system alarms to the various output devices.
 1. ALO and ALMQUE utilize the same queue; ALMQUE enqueues compressed message blocks and ALO dequeues them for further processing.
 2. When the ALO task is first activated, it will check if it is in the backup HAC. If so, then ALO will initialize to zero its alarm queue pointers. These include the following pointers: FIRSTG, COUNTG, LASTG, PTRSG(1), PTRSG(2), PTRSG(3).
 3. ALO clears the critical alarms screen line after the operator command (!!), then dequeues, builds, and outputs the next critical alarm (if any).
 4. ALO always dequeues, builds, and outputs non-critical screen lines until all enqueued blocks have been processed. This is also tone of logger/printer alarms.
 5. All screen messages are outputted via an ENQUE to the CSO queue; printer messages are outputted via a WRITE statement.
- e. Error Messages and Recovery - None

3.2.5.4.1.12.2 Data, Logic and Command Paths

a. Input Description

Input to ALO are the compressed message blocks enqueued by ALMQUE.

b. Output Description

Output of ALO consists of messages to the output devices.

c. Submodules Called

DEQU
LEAS
FRE
ENQU
ALODQU
ALOBLD

d. Global Common Usage

COMQUE
GTIMEG
PTRSG
FIRSTG
LASTG
COUNTG
ALRMSG
TBUSYG

3.2.5.4.1.12.3 Internal Data Description

There is no data internal to this submodule.

3.2.5.4.1.12.4 Flowcharts

See Figure 3.2.5-15.

3.2.5.4.1.13 Submodule XIII - ALOBLD

3.2.5.4.1.13.1 Description

- a. Language Used - FORTRAN
- b. How Invoked - Call ALOBLD (IDEST)
- c. Constraints and Limitations - Must be preceded by a call to ALODQU.
- d. Processing - ALOBLD is a support submodule for ALO that reads the disk to build alarm messages from the compressed format generated by the alarms queueing system (refer to Table 3.2.5-IV). The ASCII alarm is then output to the specified device.
 1. Alarm types one through five require special processing. Messages one, two, and three include parameters that define which LINES, HFCs, or HCs have errors in a multiple alarm message. A doubleword (32 bits) specifies which units have errors; subroutine ALOCVT creates the final message output format from information in the doubleword.

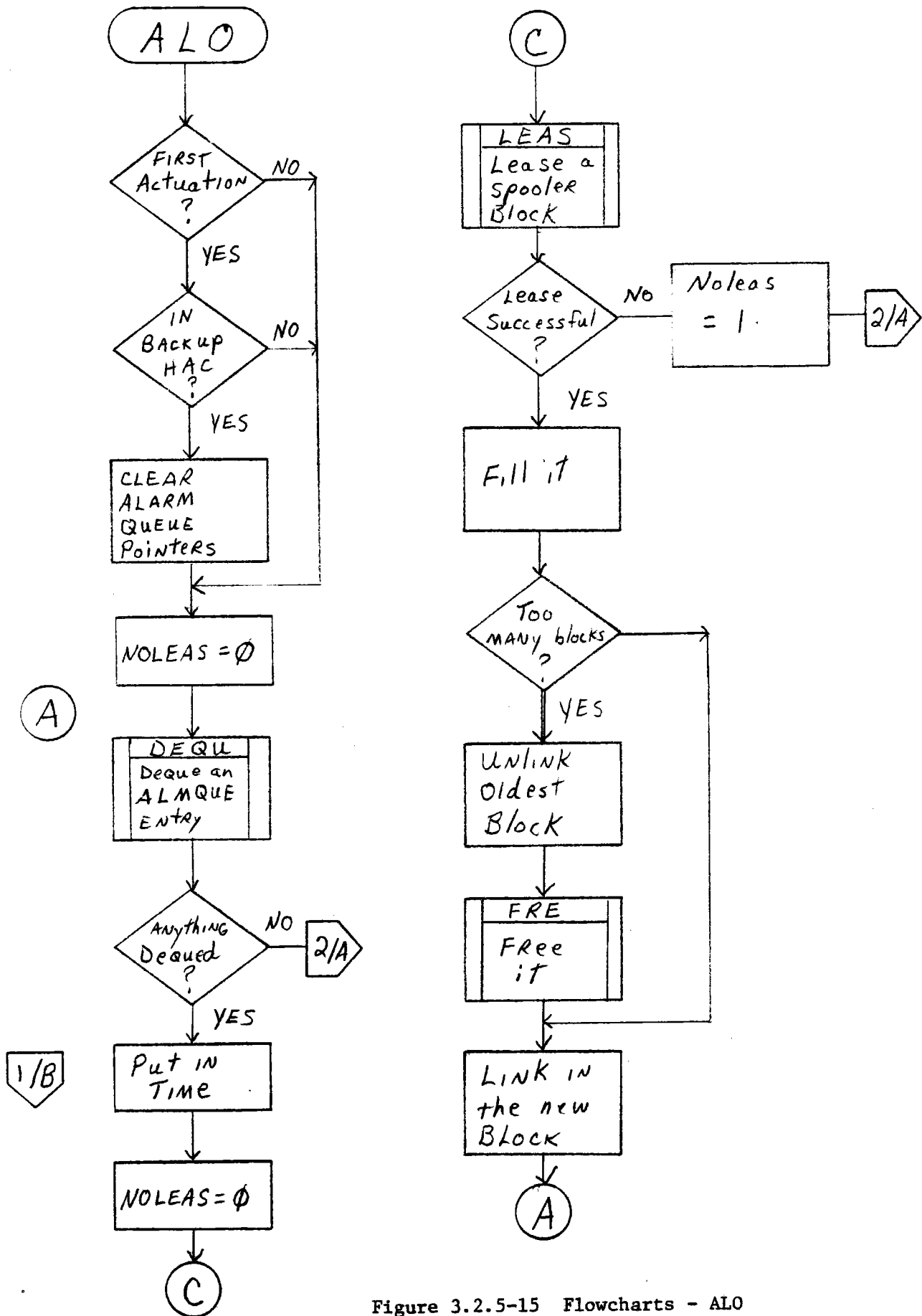


Figure 3.2.5-15 Flowcharts - ALO

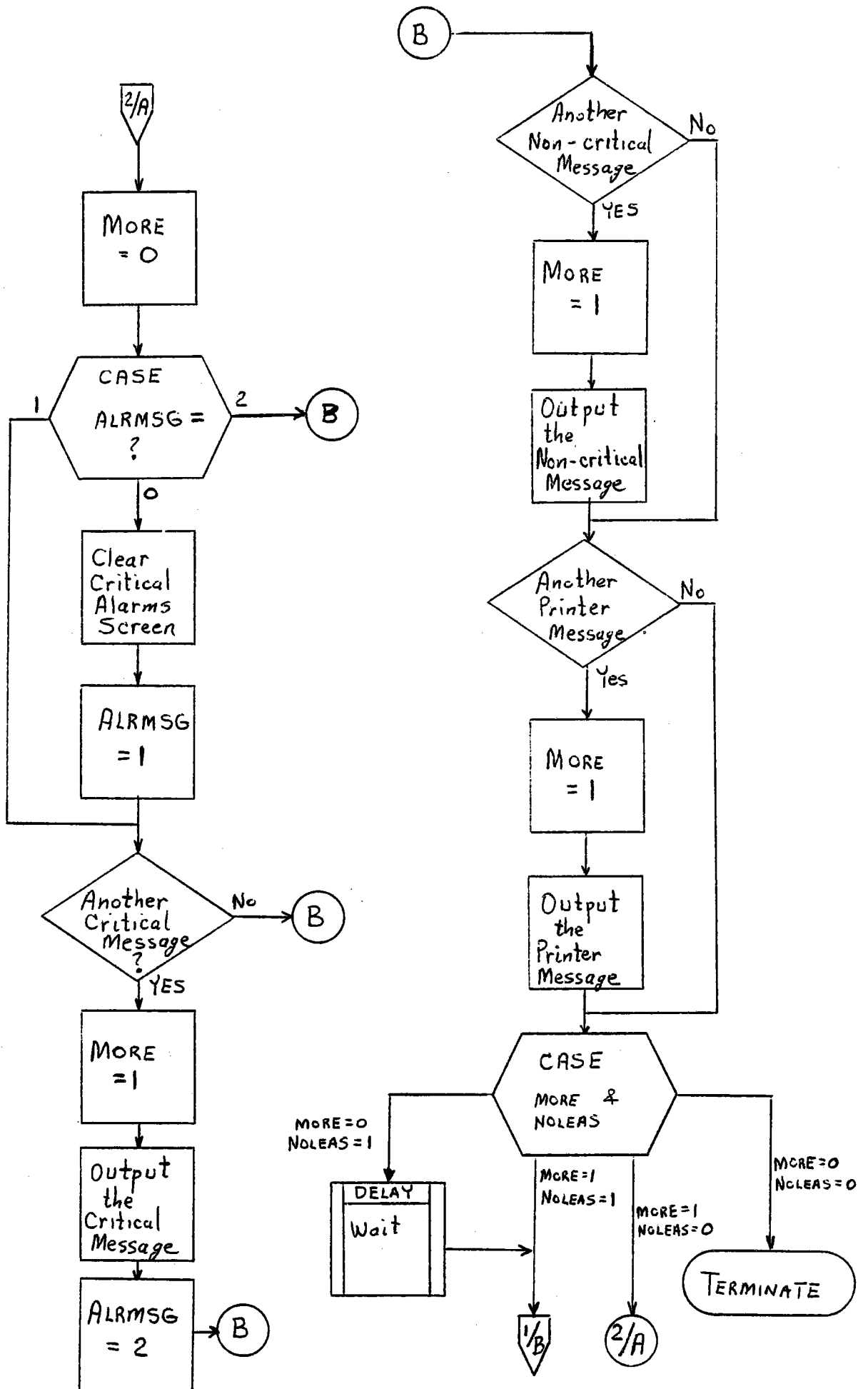


Figure 3.2.5-15 Flowcharts - ALO (Continued) 369

CRITICALFIELD ALARMS

Y HH:MM:SS FIELD POWER LOSS
 Y HH:MM:SS COMMUNICATION FAILURE WITH FIELD
 Y HH:MM:SS MULTIPLE LINE ERRORS; LINE 1,2,3,4,5,6,7,8

LINE ALARMS

Y HH:MM:SS LINE FAILURE AND SWITCHOVER; LINE N
 Y HH:MM:SS LINE COMMUNICATIONS OUTPUT ERROR; LINE N
 Y HH:MM:SS LINE COMMUNICATIONS INPUT ERROR; LINE N
 Y HH:MM:SS MULTIPLE HFC ERRORS; LINE N; HFC 1,2,3,4,5,6,7,8

HFC ALARMS

Y HH:MM:SS HFC DETECTED HAC/HFC COMMUNICATIONS ERROR; HFC NN
 Y HH:MM:SS FIRMWARE ERROR - HFC NN STATUS = XXXX
 Y HH:MM:SS HAC TO HFC COMMUNICATIONS OUTPUT TIMED OUT; HFC NN
 Y HH:MM:SS HAC TO HFC COMMUNICATIONS INPUT TIMED OUT; HFC NN
 Y HH:MM:SS INVALID DATA RECEIVED FROM HFC NN
 Y HH:MM:SS HFC DID NOT RECEIVE LAST COMMAND; HFC NN
 Y HH:MM:SS HC ERRORS; HFC NN; HC 123456789/0123456789/0123456789/012

HC ALARMS

N HH:MM:SS HC COMMUNICATIONS ERROR; HFC NN; HC NNNN
 N HH:MM:SS MISSING COMMAND RETURN; HFC NN; HC NNNN
 N HH:MM:SS NO MOTION ERROR (AZ OR EL); HFC NN; HC NNNN
 N HH:MM:SS HC UNAVAILABLE (WASH/OFFLINE) TO STHIWIND:HFC NN; HC NNNN
 N HH:MM:SS HC CANNOT RETURN TO SAVE POSITION: HFC NN; HC NNNN
 N HH:MM:SS EL MARK ENCOUNTERED: HFC NN; HC NNNN; EL=#XXXX; BIAS=#XXXX
 N HH:MM:SS AZ MARK ENCOUNTERED: HFC NN; HC NNNN; EL=#XXXX; BIAS=#XXXX
 N HH:MM:SS SEQUENCE COMMAND TIMED OUT; HFC NN; HC NNNN
 N HH:MM:SS EXTRA COMMAND RETURNED; HFC NN; HC NNNN
 N HH:MM:SS HC MARK POSITION OUTSIDE TOLERANCE; HFC NN; HC NNNN

SYSTEM ALARMS

HH:MM:SS OPERATOR CONSOLE MALFUNCTION
 HH:MM:SS CRT MALFUNCTION
 HH:MM:SS DISK REAK ERROR: TASK AAA FILE AAA
 HH:MM:SS DISK WRITE ERROR: TASK AAA FILE AAA
 HH:MM:SS RESTORE COMMAND ENCOUNTERED A HELIOSTAT IN THE WRONG MODE
 HH:MM:SS CHANGE IN CLOCK ACCURACY STATUS FROM AAAAAAAA TO AAAAAAAA

Message types four and five relate to MARK conditions and require a disk read of the MARK biases to complete the message.

2. ALOBLD decodes time into HH:MM:SS format.
3. ALOBLD reads the message format from the disk and inserts the caller's parameters into the ASCII string. A check is made for disk read errors.
4. The completed ASCII message is output to one of three devices per call: the critical alarms line, the non-critical alarms line, or the printer. A check is made for output errors.
5. Errors encountered in the execution of this submodule are queued, like any other alarm, via a call to ALMQUE. A flag is maintained to prevent repetitive queueing of a non-changing error condition. Messages destined for the printer are transferred to the system console when the printer is returning error indicators.
6. The submodule exits.

- e. Error messages and recovery - Error messages are queued to ALARMS.

3.2.5.4.1.13.2 Data, Logic and Command Paths

- a. Input Description

IDEST - Output destination: 1 = CR; 2 = OC; and
3 = TY

- b. Output Description

None

- c. Submodules Called

ALMQUE
ALOCVT

- d. Common Usage

MSGCDE
OUTPUT

3.2.5.4.1.13.3 Internal Data Description

Local buffers and arrays.

3.2.5.4.1.13.4 Flowcharts

See Figure 3.2.5-16

3.2.5.4.1.14 Submodule XIV - ALODQU

3.2.5.4.1.14.1 Description

- a. Language used - FORTRAN
- b. How invoked - Call ALODQU (IDEST)
- c. Constraints and limitations - None
- d. Processing - ALODQU is a support submodule for ALO that handles the alarm message chain and frees message blocks when all required output has been performed.
 1. ALODQU copies the compressed message block from free storage to a local buffer.
 2. The FSB queue pointer is updated for DEVICE in preparation for the next call.
 3. The FSB of current interest is dequeued and FREE'd if it has been sent to all devices; otherwise it is retained in FSB with a decremented count field.
 4. The submodule exits.
- e. Error messages and recovery - None

3.2.5.4.1.14.2 Data, Logic and Command Paths

a. Input Description

IDEST - Output destination: 1 = CR; 2 = OC; and
3 = TY

b. Output Description

None

c. Submodules Called

FREE (System Service)

d. Global Common Usage

FIRSTG, LASTG, COUNTG, and PTRSG

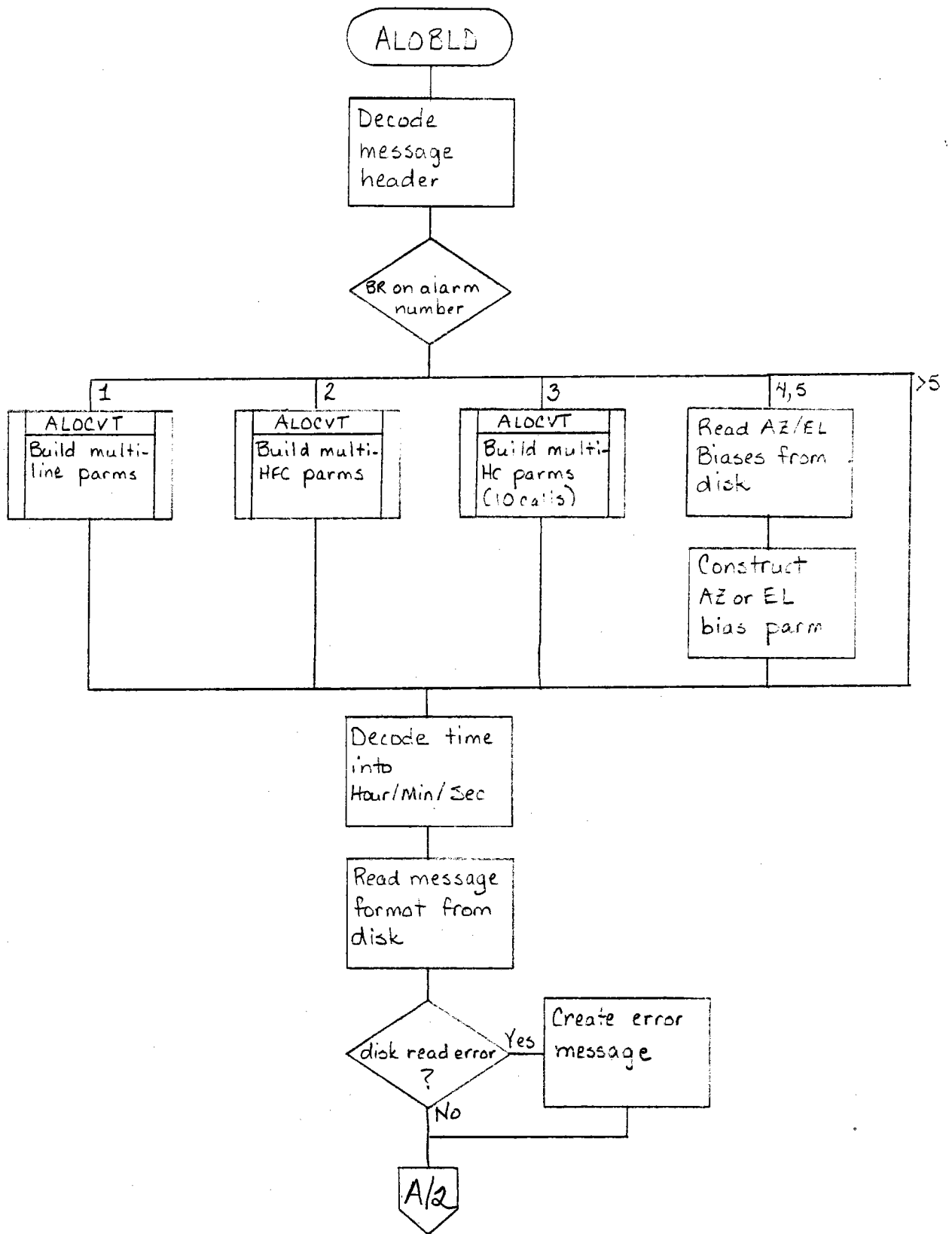


Figure 3.2.5-16 Flowcharts - ALOBLD

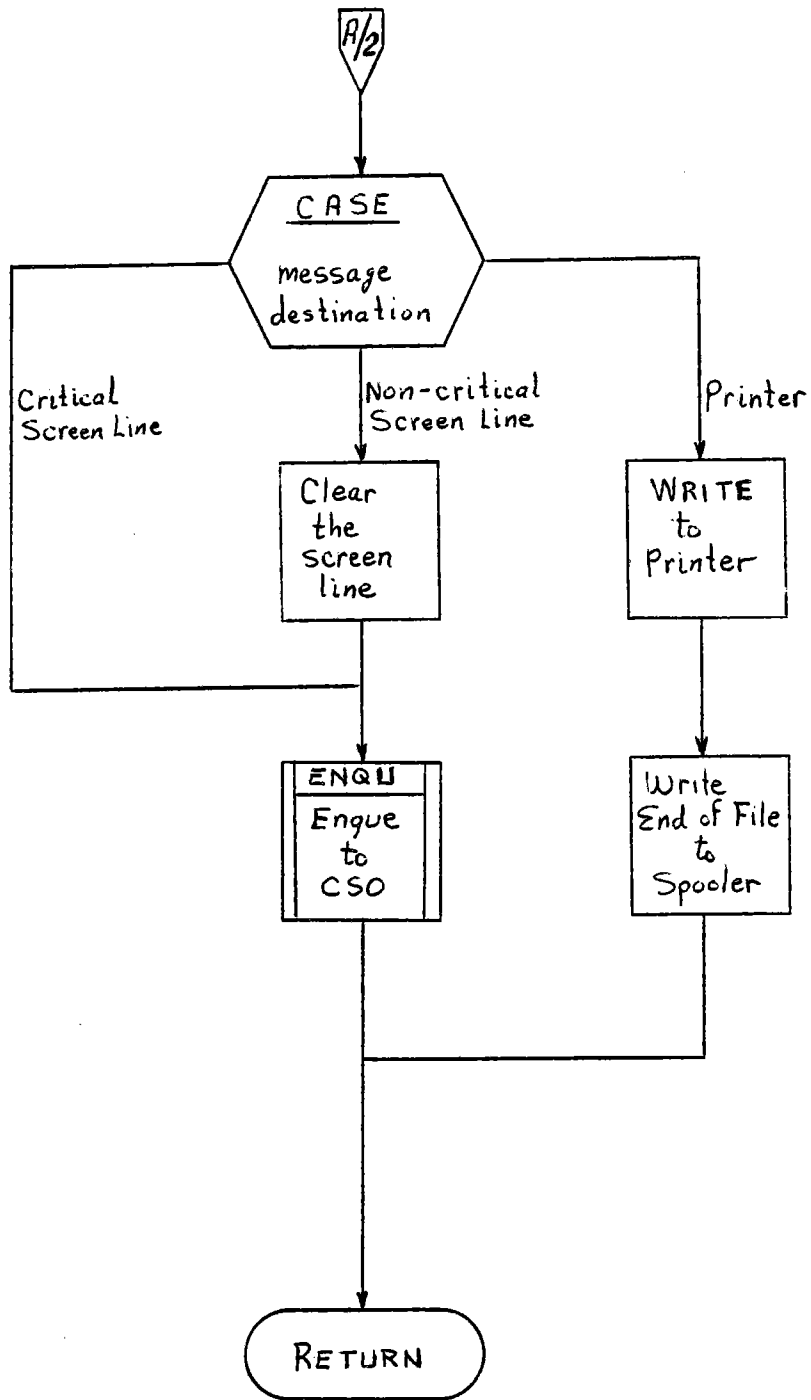


Figure 3.2.5-16 Flowcharts - ALOBLD (Continued)

e. Common Usage

MSGCDE and OUTPUT

3.2.5.4.1.14.3 Internal Data Description

There is no data internal to this submodule.

3.2.5.4.1.14.4 Flowcharts

See Figure 3.2.5-17.

3.2.5.4.1.15 Submodule XV - ALOCVT

3.2.5.4.1.15.1 Description

- a. Language used - FORTRAN
- b. How invoked - Call ALOCVT (FLAG, IB, IC, NC, ICOM, ASC)
- c. Constraints and limitations - None
- d. Processing - This submodule converts a doubleword of bits into ASCII integers for each bit that is set and blanks for each zero bit.
- e. Error messages and recovery - None

3.2.5.4.1.15.2 Data, Logic and Command Paths

a. Input Description

FLAG - Array of bits (1-32)
IB - Initial bit to test
IC - Initial character to compute (0-9)
NC - Number of characters (even integer)
ICOM - Flag for inserting commas between each character
1 = insert commas; 2 = don't insert commas

b. Output Description

ASC - Array of output ASCII

c. Submodules Called

TESTB (FORTRAN Library)

d. Global Common Requirements

None

3.2.5.4.1.15.3 Internal Data Description

ASC - Local buffer for ASCII string.

3.2.5.4.1.15.4 Flowchart

See Figure 3.2.5-18.

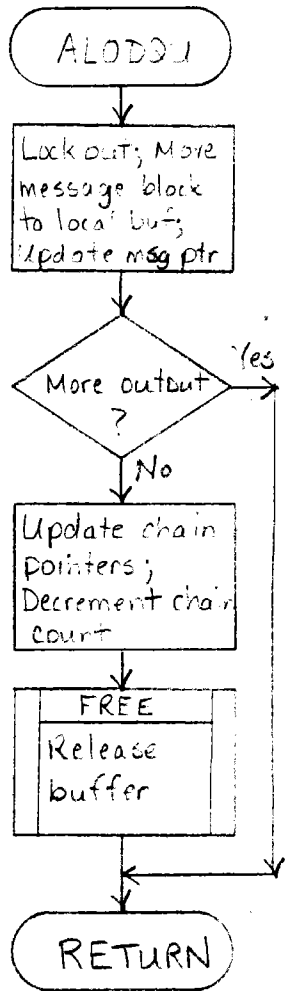


Figure 3.2.5-17 Flowcharts - ALODQU

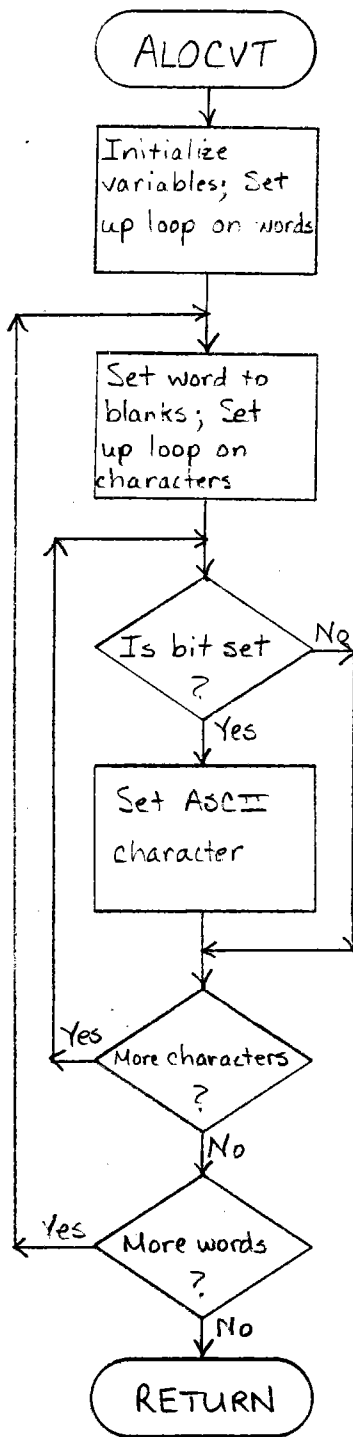


Figure 3.2.5-18 Flowcharts - ALOCVT

3.2.6 Status Display Module - STATUS

3.2.6.1 Purpose

The STATUS module decodes global status words to report on the status of the heliostat field. The STATUS module consists of two tasks: STS, the Synchronous Status Processor; and STA, the Asynchronous Status Processor. STS, in conjunction with sub-module STSGET, synchronously reports on the status of the entire heliostat field. STA, in conjunction with submodules STAFLD, STAIND, STAMOD, STARNG, STAMCH, and STACRT, gives more detailed status information of varying formats upon request. See Table 3.2.6-I for the heliostat mode definitions.

3.2.6.2 Requirements

3.2.6.2.1 Design Requirements

Section 3.1 of the 10 MWe Software/Firmware Functional Requirements Specification (12 June 1980) lists three requirements involving the STATUS module:

- a. Monitor and display the operational status of all heliostats.

This requirement indicates a need to decode each heliostat's reported status data and format and display this data.

- b. Maintain command/response protocol and data transfer with the OCS.

This requirement indicates a need to format requested status information into the HAC to OCS message formats and send this data to the OCS.

- c. Provide status data to the DAS.

This requirement indicates a need to format requested status information into the HAC to DAS message formats and send this data to the DAS.

3.2.6.2.2 Derived Requirements

Section 3.2.1.6 of the 10 MWe Software/Firmware Functional Requirements Specification lists the following requirements of the STATUS module:

- a. Monitor heliostat status from status data reported by the HCs.

This requirement indicates a need to decode each heliostat's reported status information.

<u>MODE NO.</u>	<u>MODE NAME</u>	<u>DESCRIPTION</u>
1	TRACK	Heliostat tracking the primary target
2	STDBY	(Standby) Heliostat tracking corridor upper limit point
3	BCS	Heliostat tracking the Beam Characterization System Target
4	TRANS	Heliostat moving from one mode to another
5	STOW	Heliostat stowed face down
6	ALT1	Heliostat in mirror vertical STOW position
7	ALT2	Heliostat in mirror face up STOW position
8	MARK	Heliostat successfully positioned by MARK command
9	DIR.POS.	Heliostat in directed position mode for maintenance
10	WASH	Heliostat commanded to WASH mode
11	INIT	Heliostat being powered up, initialized, or communication error
12	OFFLINE	Heliostat set offline by OFFLINE command, timeout or communication error

Table 3.2.6-I. Heliostat Mode Definitions

b. Format field status.

This requirement indicates a need to format the field status into a displayable format.

c. Display field status on the STATUS area of the CS control console, updating display as required.

This requirement indicates a need to synchronously monitor the heliostat field, and detect changes in the data to be displayed.

d. Respond to operator-entered commands for display status as reported from the MANMIF module, and format the status as requested.

This requirement indicates a need for communication with the MANMIF module and format the requested data.

e. Output the formatted status to the status printer.

This requirement indicates a need to write to the status printer when required.

f. Respond to OCS and DAS generated status requests by transmitting the requested status to the requestor.

This requirement indicates a need to communicate with the EXTINF module for data transmission, and a communication with the MANMIF module to indicate the source of the request.

3.2.6.3 Design Approach

3.2.6.3.1 Functional Allocations

The STATUS module is comprised of ten submodules. (See Figure 3.2.6-1 for the STATUS module structure.) The basic purpose of each is described below:

- a. STS (Synchronous Status Processor) - This submodule monitors, on a synchronous basis, the number of heliostats in each operational mode. STS scans one-eighth of the field each second. Any changes found in the data to be displayed cause an update in the data and display. STS also builds data packets for the graphics display units, and synchronously outputs date and time to the HAC operator's console. See Figure 3.2.6-2 for the HAC operator's console synchronous status display;
- b. STSGET (Synchronous Status Decoder) - This submodule is called by STS to decode the HC status words (HCST2G and HCST1G) to determine the mode of a given

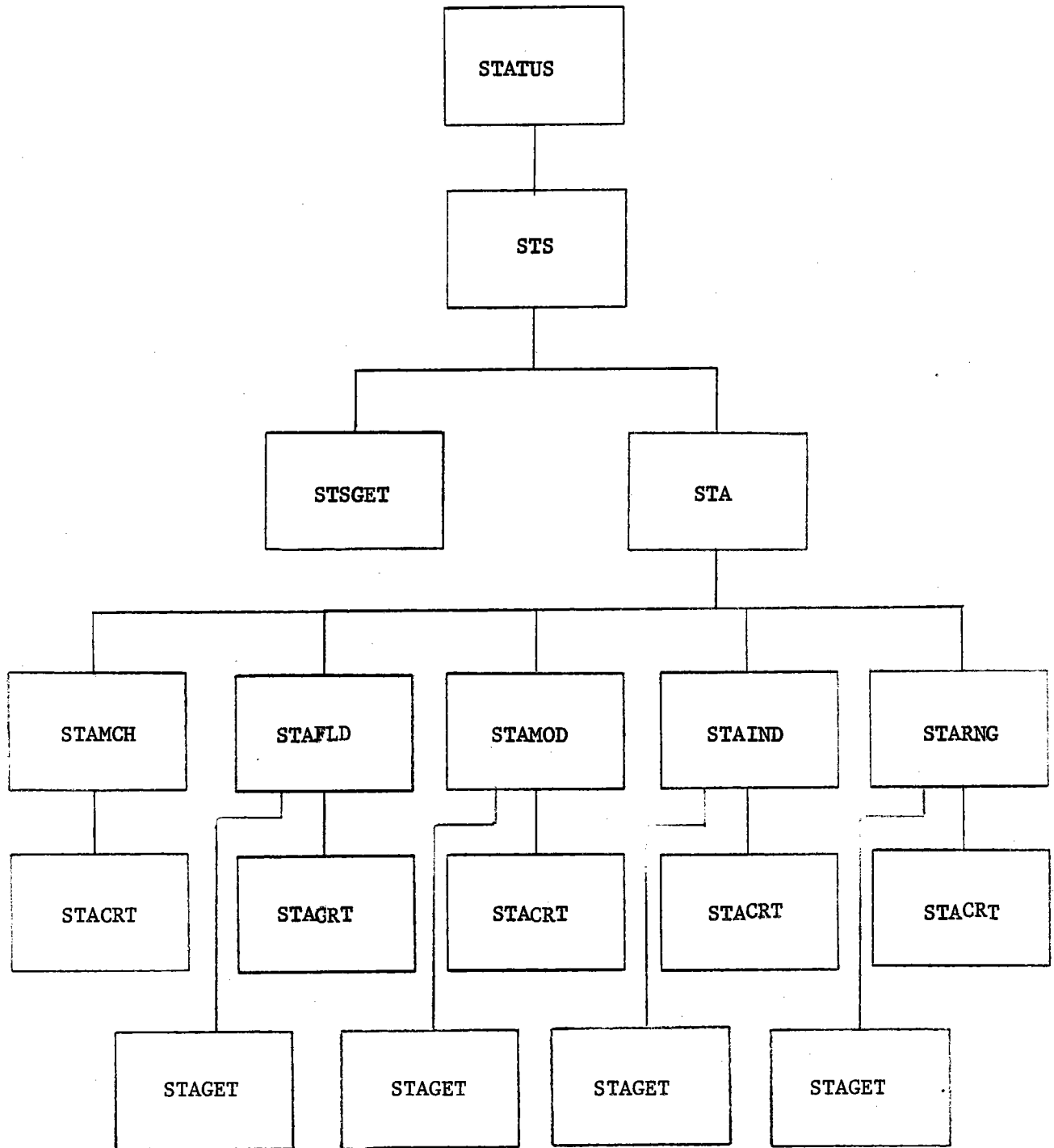


Figure 3.2.6-1 STATUS

Module Structure

HAC SYNCHRONOUS STATUS DISPLAY AREA

COLUMN

1 5 0 5 0 5 0 5 0 5 0 5 0 5 0 5 0

ROW
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

CRITICAL ALARMS AREA											
LATEST ALARM AREA											
MM/DD/YY HH:MM:SS											
TARGETING MODES			TRAN	STOW MODES				DIR. POS.	WASH	INIT	OFF-LINE
TRACK	STDBY	BCS		STOW	ALT1	ALT2	MARK				
nnnn	nnnn	nnnn	nnnn	nnnn	nnnn	nnnn	nnnn	nnnn	nnnn	nnnn	nnnn

382

CONVERSATIONAL AREA



Figure 3.2.6-2 HAC Operator's Console Synchronous Status Display

HC. See Table 3.2.6-II for the mapping from actual mode to reported mode;

- c. STA (Asynchronous Status Processor) - This submodule is activated by STS if the Man-Machine Interface module (MANMIF) has indicated an operator-initiated status request. Depending on the type of request ("Field," "Mode," "Individual," or "Ring"), STA calls one of the following four submodules: STAFLD, STAIND, STAMOD, and STARNG;
- D. STAFLD (Field Status Processor) - STAFLD is called by STA to process "Field" status requests. It formats the latest status data created by STS (MODEG) and sends the results to the OCS or DAS consoles, HAC line printer, or, optionally, the HAC operator's console. See Figure 3.2.6-3 for the STAFLD output format, and Table 3.2.6-III for the STAFLD OCS and DAS message format;
- e. STAIND (Individual Status Processor) - STAIND is called by STA to process individual status requests. For the given HC it determines the actual mode, commanded mode, segment, azimuth, and elevation. See Figure 3.2.6-4 for the CS console display and Table 3.2.6-IV for the OCS and DAS message format;
- f. STAMOD (Mode Status Processor) - STAMOD is called by STA to process "Mode" status requests. It determines all the HCs which are in the requested mode, formats this list (see Table 3.2.6-V), and sends it to the DAS or OCS consoles, CS line printer, or optionally, the CS console (see Figure 3.2.6-5);
- g. STARNG (Ring Status Processor) - STARNG is called by STA to process a "Ring" status request. It determines all the segment numbers in the ring, the total number of HCs in each of the segments, and the numbers of HCs in TRACK and STANDBY for each segment in the ring. See Figure 3.2.6-6 for the STARNG CS console display, and Table 3.2.6-VI for the OCS and DAS Segment Track Status Message format;
- h. STAGET (Asynchronous Status Decoder) - STAGET is called by STAFLD, STAIND, STAMOD, and STARNG to decode the HC status words (HCST2G and HCST1G) to determine the actual mode and the commanded mode of a given HC.

i. STAMCH - Processes "MATCH" status requests:

```
"STATUS ER/XXX"  
"STATUS H1, MMMM, NNNN"  
"STATUS H2, MMMM, NNNN"  
"STATUS F2, MMMM, NNNN"
```

Where XXX = Memnomic
MMMM = Match Value
NNNN = Mask Value; and

j. STACRT - Enques output messages to CS console and logger, or OCS, DAS core device(s).

3.2.6.3.2 Resource Budgets

STATUS requires approximately (5K) words of memory. It is the lowest priority foreground module in the heliostat system and is synchronously activated. STATUS requires no disk or magnetic tape access.

10 MWe STATUS (STSGET)

MAPPING FROM ACTUAL STATUS TO REPORTED MODE

<u>ACTUAL MODE</u>	<u>STATUS MODE</u>	<u>NO.</u>
Offline - Not installed	Not reported	-1
Offline - Communication Error	OFFLINE	12
Offline - Serious Alarm	OFFLINE	12
Offline - Operator	OFFLINE	12
Offline - Unmarked	OFFLINE	12
Position, No Compare	TRANSITION	4
Restarted - Power-up or time-out	INIT	11
Restarted - Successive Comm. Errors	INIT	11
Beam Pointing - Track	TRACK	1
Beam Pointing - BCS	BCS	3
Beam Pointing - Standby or CULP	STANDBY	2
Beam Pointing - GLLP	TRANSITION	4
Corridor Walk - Up Corridor A, B, C, or D	TRANSITION	4
Corridor Walk - Down Corridor A, B, C, or D	TRANSITION	4
Az-E1 Pointing - Stow	STOW	5
Az-E1 Pointing - Seek Marks	MARK	8
Az-E1 Pointing - Wash	WASH	10
Az-E1 Pointing - Maintenance	DIR.POS.	9
Az-E1 Pointing - Alt1	ALT1	6
Az-E1 Pointing - Alt2	ALT2	7

Table 3.2.6-II. 10 MWe Status (STSGET)

FIELD STATUS

<u>MODE</u>	<u>TOTAL</u>
TRACK	nnnn
STDBY	nnnn
BCS	.
TRANS	.
STOW	.
ALT1	.
ALT2	.
MARK	.
DIR. POS.	.
WASH	.
INIT	.
OFFLINE	nnnn
TOTAL	tttt

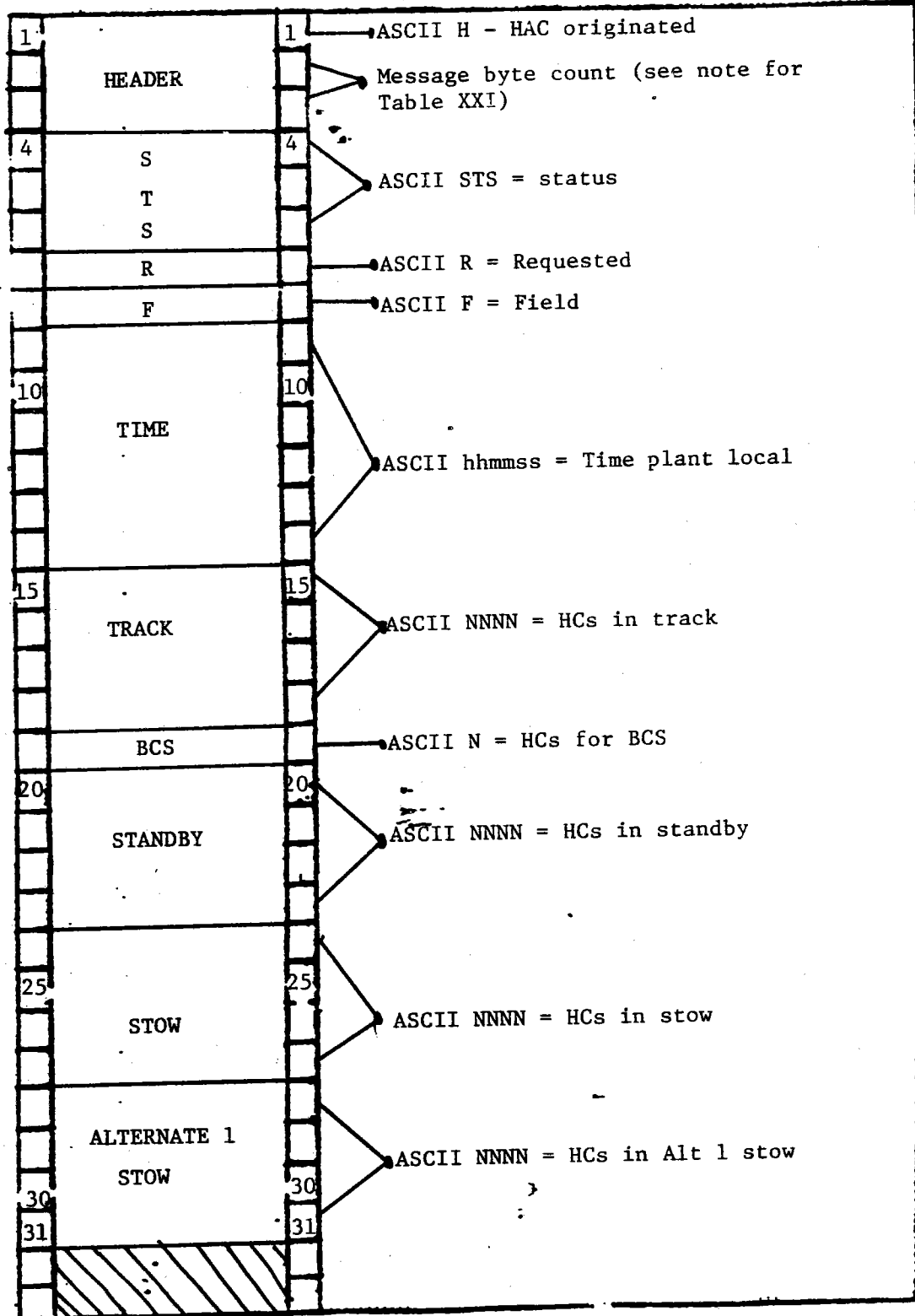
Figure 3.2.6-3 STAFLD Field Status Display

TABLE 3.2.6-III HAC-OCS/DAS ENVIRONMENT STATUS (FIELD) FORMAT

MESSAGE LAYOUT

APPLICATION HAC - OCS/DAS MESSAGE TYPE STATUS - FIELD

PROGRAMMER T. Ladewig DATE 7/8/80



MESSAGE LAYOUT

APPLICATION HAC OCS/DAS STATUS-FIELD
MESSAGE TYPE (CONTINUED)

PROGRAMMER T. Ladewig

DATE 2/8/80

32	ALTERNATE ? STOW	32	ASCII NNNN = HCs in ALT?STOW
35	TRANSITION	35	ASCII NNNN = HCs in Transition
40	WASH	40	ASCII NNNN= HCs In Wash
45	DIRECTED POSITION	45	ASCII NNNN = HCs in directed position
50	OFFLINE	50	ASCII NNNN = HCs in offline
55	MARK	55	ASCII NNNN= HCs in Mark
	INIT		ASCII NNNN = HCs in INIT
60	UNUSED	60	
77		77	
80	CHECKSUM	80	8 bit checksum such that all bytes sum to zero.

INDIVIDUAL STATUS:

HELIOSTAT NUMBER: NNNN

SEGMENT: NNN

AZIMUTH: SNNN.NN

ELEVATION: SNNN.NN

COMMANDED MODE: MMM

ACTUAL MODE: MMM

Figure 3.2.6-4 STAIN Display Format

TABLE 3.2.6-IV HAC-OCS/DAS ENVIRONMENT STATUS (HC) FORMAT

MESSAGE LAYOUT

APPLICATION HAC OCS /DAS MESSAGE TYPE STATUS
 INDIVIDUAL HC

PROGRAMMER T. Ladewig DATE 7/8/80

1	HEADER	1	• ASCII H = HAC Originated
			• Message byte count (See note for Table XXI)
5	STS	5	• ASCII STS = Status
	R		• ASCII R = Requested
	H		• ASCII H = Individual Status
10	TIME	10	
15	HC#	15	• ASCII 4 Digit HC Number
20	AZIMUTH	20	
25	ELEVATION	25	
30		30	• ASCII 7 Digit SNNN.NN Elevation
32		32	

TABLE 3.2.6-IV HAC-OCS/DAS ENVIRONMENT STATUS (HC) FORMAT (CONTINUED)

MESSAGE LAYOUT

APPLICATION HAC-OCS/DAS STATUS
 MESSAGE TYPE INDIVIDUAL HC

PROGRAMMER T. Ladewig DATE 5/12/80

33		33	ASCII 3 Characters. See Table XXIV for acceptable 3 character codes.
	MODE		
35		35	ASCII T = TRANSITION P = POSITION COMPARE
36	TRANSITION FLAG	36	
			<p>NOTE: The heliostat will be in the mode identified if the transition flag is a <u>P</u>. The heliostat will be in transition to the mode identified if the transition flag is a <u>T</u>.</p> <p>If the mode identified is TRN then the heliostat is within a sequence and has not yet received the final command of the sequence.</p>
	UNUSED		
80	CHECKSUM	80	

TABLE 3.2.6-V HAC-OCS/DAS ENVIRONMENT STATUS (MODE) FORMAT
MESSAGE LAYOUT

APPLICATION HAC-OCS/DAS MESSAGE TYPE STATUS--MODE

PROGRAMMER T. Ladewig DATE 2/8/80

1	HEADER	1	ASCII E = HAC originated
			Message byte count (see Note for Table XXI)
5	STS	5	ASCII STS - STATUS
	CONDITION		ASCII R = Requested A = Alarmed*
	M		ASCII M = Mode
10	MODE	10	ASCII TRK = Track BCS = BCS STB = Standby STO = Stow AL1 = ALT1STOW AL2 = ALT2STOW TRN = Transition WSH = Wash DPO = Directed Position OFF = Offline MRK = Mark INI = Init UST = Unavail. for STHIWIND * ¹ URS = Unavail. for RESTORE * ¹
15	TIME	15	ASCII hhhhmmss - Time
20	HC#	20	ASCII 4 digit HC number. Additional messages will be used to transmit additional HCs.
	HC#		
25	⋮	25	
	UNUSED		*NOTE: Alarmed will be used to report HCs unable to respond to a STHIWIND command. HCs in WASH, OFFLINE, and DIRECTED POSITION will be transmitted
80	CHECKSUM	80	* ¹ NOTE: UST and URS are alarmed only and cannot be requested. Checksum

MODE STATUS

HELIOSTATS IN MODE MMM:

NNNN, NNNN, NNNN, --- NNNN

NNNN,

.

.

.

NNNN, NNNN

Figure 3.2.6-5 STAMOD Display Format

SEGMENT TRACK STATUS

RING X

SEGMENT NUMBER	HCs IN SEGMENT	HCs IN TRACK	HCs IN STANDBY
nnn	nnn	nnn	nnn
nnn	nnn	nnn	nnn
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
nnn	nnn	nnn	nnn
TOTALS:	ttt	ttt	ttt

Figure 3.2.6-6 STARNG Segment Track Status

TABLE 3.2.6-VI HAC-OCS/DAS ENVIRONMENT STATUS (RING) FORMAT
MESSAGE LAYOUT

APPLICATION HAC-OCS/DAS STATUS RING
MESSAGE TYPE RING

PROGRAMMER T. Ladewig DATE 5/12/80

1	HEADER	1	ASCII H = HAC Originated
			Message byte count (see note for Table XXI.)
5	STS	5	ASCII STS = STATUS
	R		ASCII R = Requested
	R		ASCII R = Ring
10	TIME	10	ASCII Time Plant Local
15	RING	15	ASCII 1-digit Ring number requested
	SEGMENT		ASCII 3-digit Segment number
20	HCS in SEGMENT	20	ASCII 2-digit number of HCS in Segment
	TRACK		ASCII 2-digit number of HCS in TRACK in Segment
	STANDBY		ASCII 2-digit number of HCS in STANDBY in Segment
	REPEAT		REPEAT as many times as necessary to cover complete ring - multiple messages may be necessary.
80	CHECKSUM	80	

3.2.6.4 Design Description

3.2.6.4.1 Module Structure

STATUS is divided into ten submodules: STS, STSGET, STA, STAFLD, STAIND, STAMOD, STARNG, STAGET, STAMCH, and STACRT. STS and STSGET are synchronously activated by FCP, and STA is activated by MMI if there is a pending operator status request. See Figure 3.2.6-1 for the STATUS module structure.

3.2.6.4.1.1 Submodule I - STS

3.2.6.4.1.1.1 Description

- a. Language used - FORTRAN
- b. How invoked - Synchronously activated by FCP
- c. Constraints and limitations - None
- d. Processing -
 1. Upon being activated by FCP, STS determines which one-eighth of the field is being monitored and clears the corresponding row of matrix EMODE.
 2. Next a loop is made through the heliostats in the appropriate one-eighth of the field. The mode of each heliostat is determined using submodule STSGET. If the heliostat is installed (mode returned by STSGET is positive), the mode number serves as an index to the columns of EMODE, and the contents of the appropriate row and column of EMODE is incremented by one. Additionally, STS stores the GRSTSG mode numbers in global array for use by the Graphics Module (GRAPHIC). After this loop is complete, this row of EMODE will contain the number of heliostats in each mode for the given one-eighth of the field.
 3. The time and date are output to the STATUS area of the CS console.
 4. After the above loop is complete, the one-eighth of field subtotals in EMODE are summed and compared to the total field sum for each mode and stored in MODEG. If any of the totals are different, the MODEG array is updated from the information in EMODE, and this information is formatted and transmitted to the CS console via the EXTINF module.

5. Next, global words ISTATG (set by MMI) are checked to see if there is a pending operator request for more status information. If so, submode STA is activated.

6. STS then exits.

e. Error messages and recovery - None

3.2.6.4.1.1.2 Data, Logic and Command Paths

Input data:

MODEG - MODEG is a 12-word array, each word containing the total number of heliostats in the corresponding mode (1 to 12). MODEG is updated by STS by calculating the new subtotals in EMODE and comparing; and

ISTATG - ISTATG is a three-word array, set by MMI to indicate a pending operator status request if ISTATG(1) is non-zero.

Output data:

GRSTSG - GRSTSG is a 2048-word array containing the mode number (1 to 12) of the corresponding heliostat. GRSTSG is updated by the corresponding one-eighth of the field.

3.2.6.4.1.1.3 Internal Data Description

EMODE - EMODE is an eight-by-twelve word array, the eight rows corresponding to each one-eighth of the field, and the 12 columns corresponding to the 12 possible modes. Each element contains the number of heliostats in a given one-eighth of the field in a given mode. This, EMODE (2,5) contains the number of heliostats in STOW (mode number five) in the second one-eighth of the field.

3.2.6.4.1.1.4 Flowchart

See Figure 3.2.6-7.

3.2.6.4.1.2 Submodule II - STSGET

3.2.6.4.1.2.1 Description

- a. Language used - FORTRAN
- b. How invoked - Called by STS
- c. Constraints and limitations - None

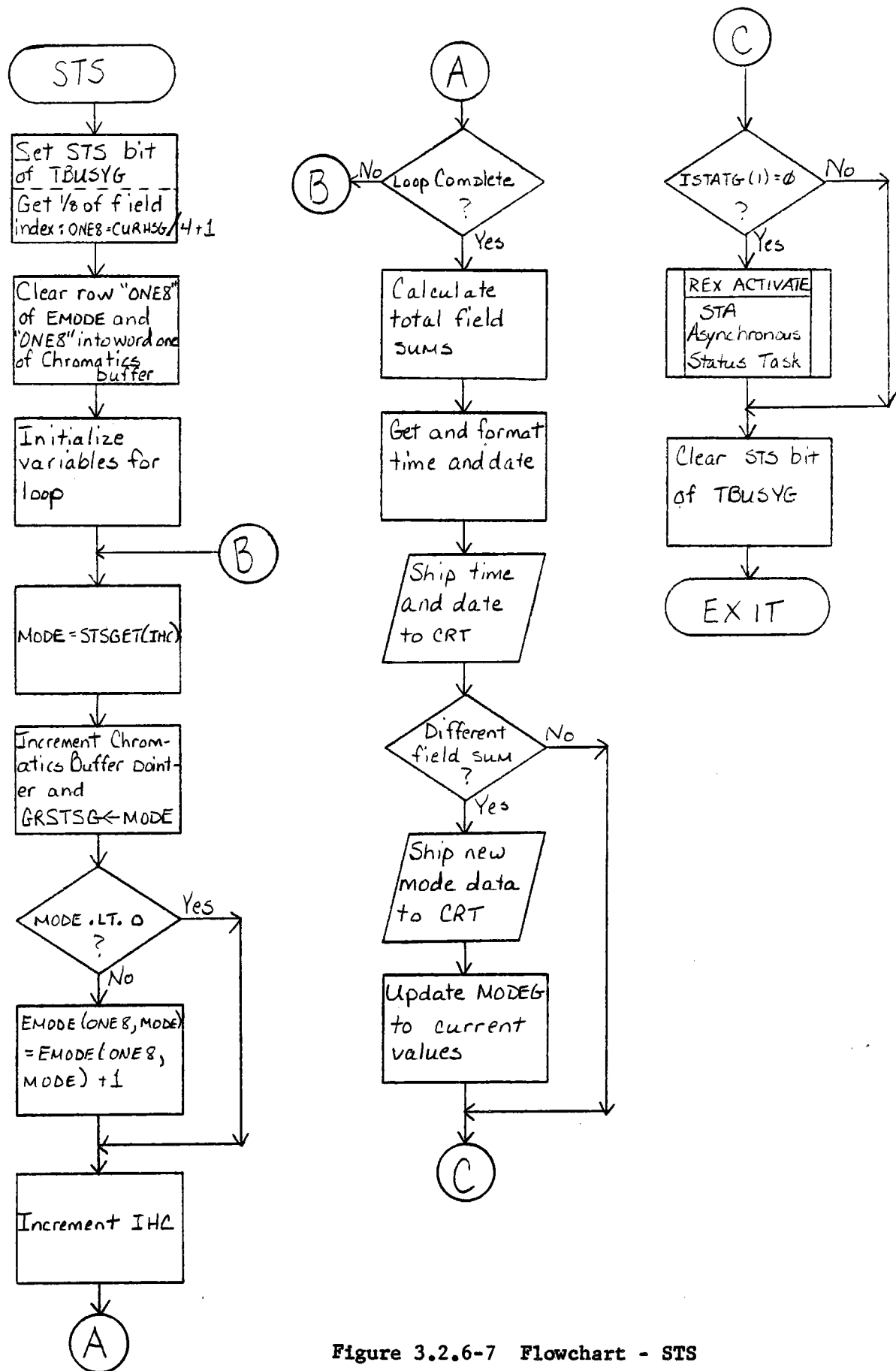


Figure 3.2.6-7 Flowchart - STS

d. Processing - STSGET decodes the global HC status words HCST1G and HCST2G to determine the mode of a given heliostat. See data base section 3.3.1 for a description of HCST1G and HCST2G.

1. Upon being called, STSGET checks the highest-order bit of HCST2G to determine if the heliostat is actually installed (if it is not installed, this bit is set). If this bit is set, the mode number is set to minus one.
2. If the HC is installed, the next three high-order bits of HCST2G are checked, indicating that the HC is set offline, in which case the mode number is returned as 12.
3. If the HC is installed and online, as indicated by HCST2G, HCST1G is logically ANDed with a mask (207C hex) to isolate bits 13 (position compare) and six through two (mode-submode). The result of the ANDing is checked against an array containing values which will match the result if the heliostat is in the mode which is the index to this array. See Table 3.2.6-II for the mapping from mode-submode in HCST1G to the reported mode.

If the result of the ANDing indicates that the mode-submode is "Seek Mark", another ANDing is made to determine if the "Mark Encountered" bits of HCST1G are set, in which case the mode is "Transition."

4. If the result of the ANDing(s) does not indicate a definite mode, the mode is set to "Transition."

e. Error messages and recovery - None.

3.2.6.4.1.2.2 Data, Logic and Command Paths

Input data:

HFCHC - the HFC-HC number (1 to 2148) whose mode is desired

HCST1G - HC status word

HCST2G - HC status word

Output data:

MODE - the resultant mode number (-1 or values from 1 to 12)

3.2.6.4.1.2.3 Internal Data Description

MASK - Value of 207C hex ANDED with HCST1G

MASKR - An array of values, whose index is the heliostat mode if the result of masking HCST1G is equal to one of the array elements.

MASKM - Value of C000 hex ANDED with HCST1G to determine "Mark Encountered."

3.2.6.4.1.2.4 Flowchart

See Figure 3.2.6-8

3.2.6.4.1.3 Submodule III - STA

3.2.6.4.1.3.1 Description

- a. Language used - FORTRAN
- b. How invoked - Activated by STS if there is a pending operator status request.
- c. Constraints and limitations - None
- d. Processing - Upon activation, STA checks the value of ISTATG(1) to determine which type of status request is pending, and calls one of the following subroutines as follows:

If ISTATG(1) = 1 thru 12, STAMOD is called to process a "MODE" status request.

If ISTATG(1) = 13, STAFD is called to process a "FIELD" status request.

If ISTATG(1) = 14, STAIND is called to process an "INDIVIDUAL" status request.

If ISTATG(1) = 15, STARNG is called to process a "RING-SEGMENT" status request.

If ISTATG(1) = 16, 17, or 18, STAMCH is called to process a match request.

- e. Error messages and recovery - None

3.2.6.4.1.3.2 Data, Logic and Command Paths

Input data:

ISTATG is used as described above.

3.2.6.4.1.3.3 Internal Data Description

STA has no pertinent internal data.

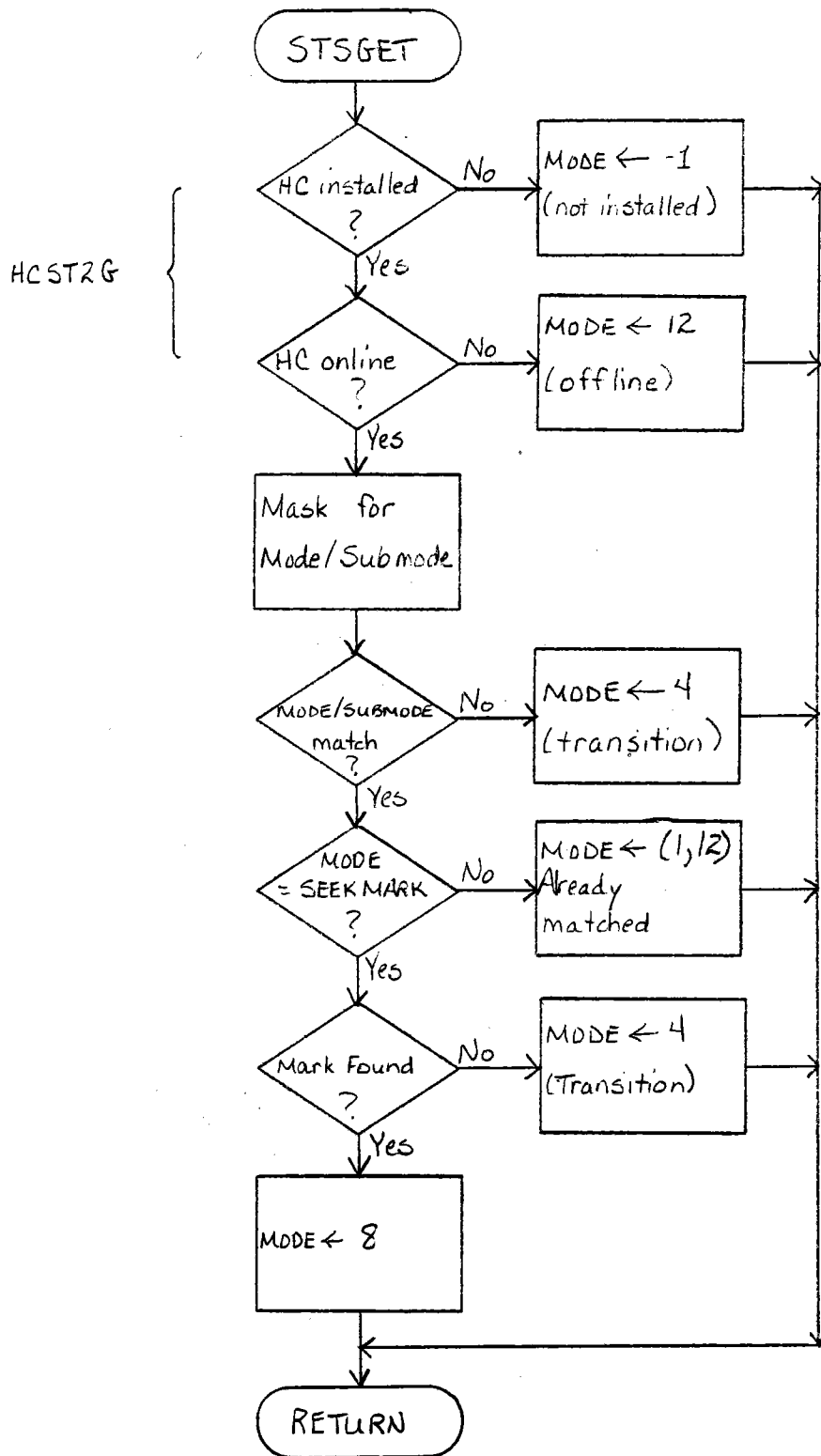


Figure 3.2.6-8 Flowchart - STSGET

3.2.6.4.1.3.4

Flowchart

See Figure 3.2.6-9.

3.2.6.4.1.4

Submodule IV - STAFLD

3.2.6.4.1.4.1

Description

- a. Language used - FORTRAN
- b. How invoked - Called by STA
- c. Constraints and limitations - None
- d. Processing - STAFLD formats global common words MODEG and the local time and date to process a field status request. See Table 3.2.6-III for the OCS and DAS message formats and Figure 3.2.6-2 for the CS console display.
- e. Error messages and recovery - None

3.2.6.4.1.4.2

Data, Logic, and Command Paths

Input data:

Global words MODEG.

3.2.6.4.1.4.3

Internal Data Description

STAFLD has an output buffer to store the output data in.

3.2.6.4.1.4.4

Flowchart

See Figure 3.2.6-10

3.2.6.4.1.5

Submodule V - STAIND

3.2.6.4.1.5.1

Description

- a. Language used - FORTRAN
- b. How invoked - called by STA
- c. Constraints and limitations - None
- d. Processing - STAIND calls STAGET to determine the mode and commanded mode of the desired heliostat and formats this data with the date and time (see Table 3.2.6-VI for the message format and Figure 3.2.6-3 for the display format).
- e. Error messages and recovery - None

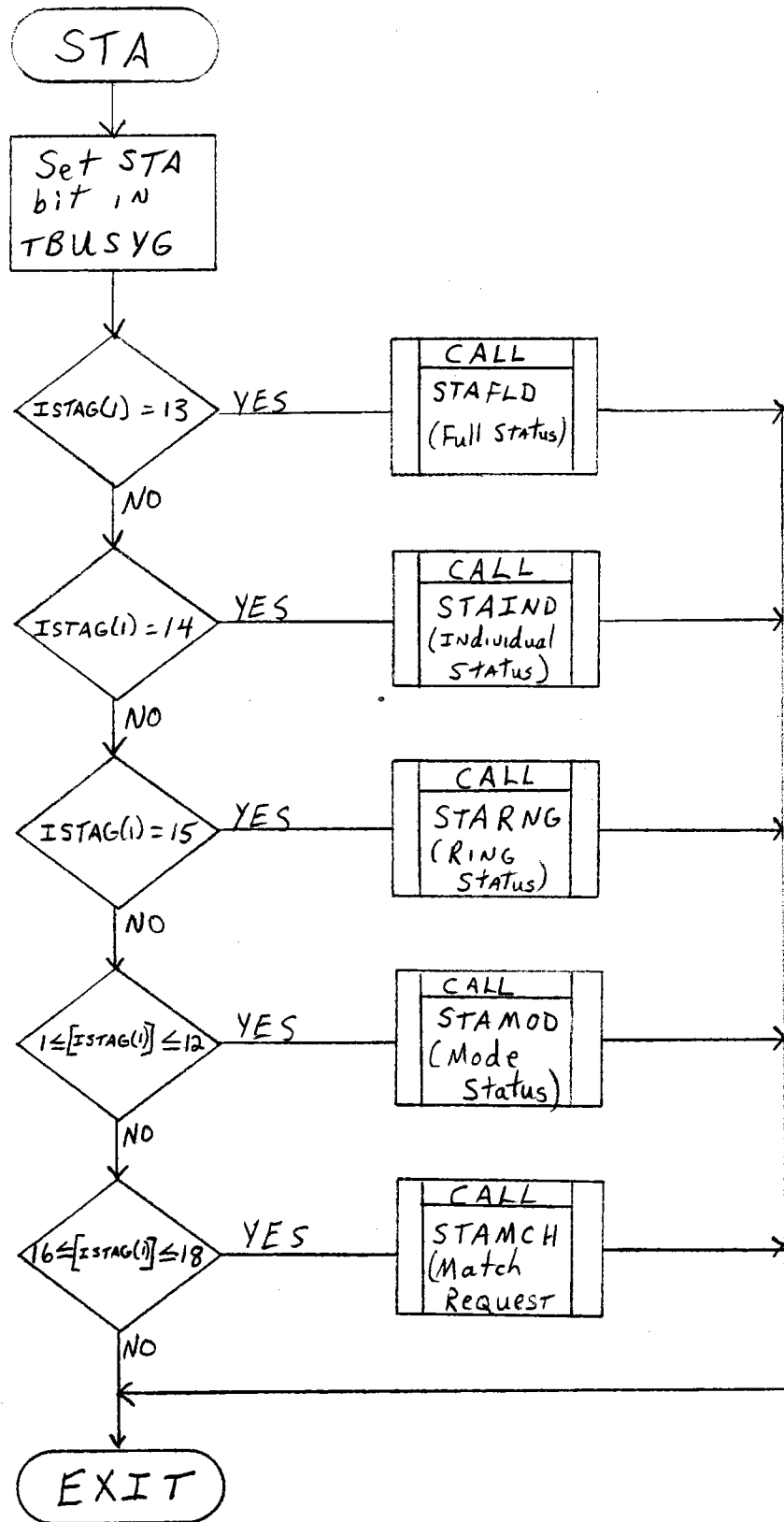


Figure 3.2.6-9 Flowchart - STA

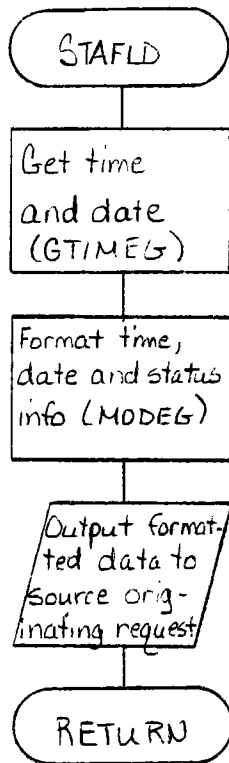


Figure 3.2.6-10 Flowchart - STAFD

3.2.6.4.1.5.2 Data, Logic, and Command Paths

Input data:

ISTATG(2) - contains the desired heliostat number.

ISTATG(3) - contains code number for source of request.

Output data:

Enqued to EXTINF.

3.2.6.4.1.5.3 Internal Data Description

STAIND has an output buffer.

3.2.6.4.1.5.4 Flowchart

See Figure 3.2.6-11

3.2.6.4.1.6 Submodule VI - STAMOD

3.2.6.4.1.6.1 Description

a. Language used - FORTRAN

b. How invoked - Called by STA

c. Constraints and limitations - None

d. Processing -

1. Local time and date are obtained and local variables initialized.
2. A loop is made through all the heliostats in the field; STAGET is called for each one to determine the mode, and each heliostat in the desired mode has its MDAC heliostat number (obtained via HC2MDG) loaded into the output buffer (see Table 3.2.6-V.) The output buffer is enqued to the EXTINF module each time it is filled.
3. When the loop through the heliostats is completed, the output buffer is sent if partially filled.

e. Error messages and recovery - None

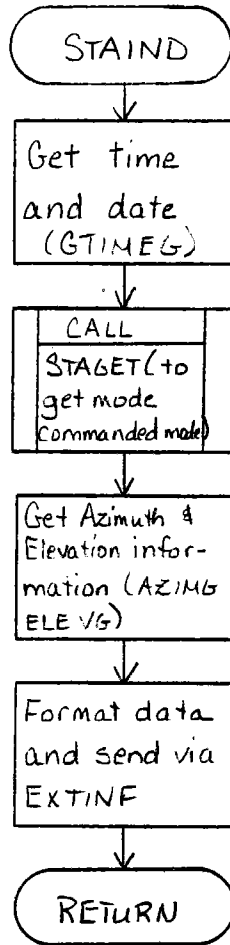


Figure 3.2.6-11 Flowchart - STAIND

3.2.6.4.1.6.2 Data, Logic, and Command Paths

Input data:

ISTATG(1) - contains the desired mode number

GTIMEG - contains local time and date

Output data:

Enqueued through EXTINF.

3.2.6.4.1.6.3 Internal Data Description

STAMOD contains an output buffer.

3.2.6.4.1.6.4 Flowchart

See Figure 3.2.6-12

3.2.6.4.1.7 Submodule VII - STARNG

3.2.6.4.1.7.1 Description

- a. Language used - FORTRAN
- b. How invoked - Called by STA
- c. Constraints and limitations - None
- d. Processing -
 1. STARNG uses global data SEGPTG and SEGMPG to obtain the heliostat numbers in the desired ring.
 2. A loop is made through these heliostat numbers, and the mode of these heliostats is determined by STAGET.
 3. If the mode is Track or Standby, the Track or Standby counter for this segment is incremented.
 4. After looping through all the heliostats in the ring, the response message (see Table 3.2.6-VI) is formatted from the data and enqueued to EXTINF.
- e. Error messages and recovery - None

3.2.6.4.1.7.2 Data, Logic, and Command Paths

Input data:

ISTATG(2) - Ring number.

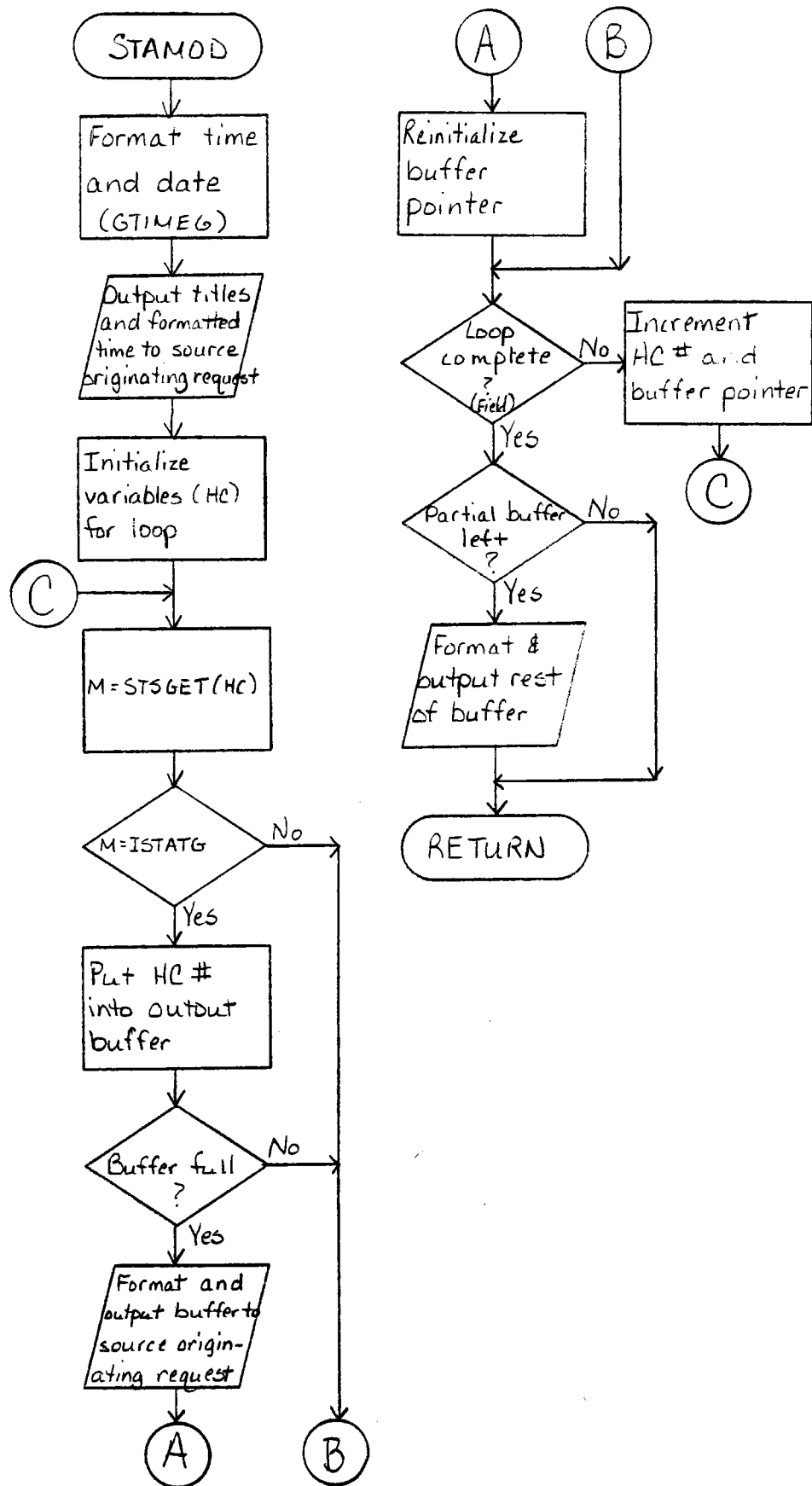


Figure 3.2.6-12 Flowchart - STAMOD

ISTATG(3) - Code indicating source of request.

SEGPTG - Global array used to obtain HC numbers from ring number, in conjunction with SEGMPG.

SEGMPG - Global array used to obtain HC numbers, in conjunction with SEGPTG.

See the DBINIT write-up for a more complete description of SEGPTG and SEGMPG.

The indices of SEGPTG are obtained using the formula:

$$\text{INDEX}(I) = (\text{RING} - 1) * 12 + I$$

$$I = 1, 12$$

and the segment numbers obtained via

$$\text{SEG} * (I) = \text{RING} * 100 + I$$

$$I = 1, 12$$

Output data:

Enqued to EXTINF

3.2.6.4.1.7.3 Internal Data Description

STARNG has two 12-word arrays to contain the number of heliostats in Track and Standby in up to 12 segments in the desired ring.

3.2.6.4.1.7.4 Flowchart

See Figure 3.2.6-13

3.2.6.4.1.8 Submodule VIII - STAGET

3.2.6.4.1.8.1 Description

- a. Language used - FORTRAN
- b. How invoked - Called by STAIND, STAMOD, STAFLD, and STARNG.
- c. Constraints and limitations - None
- d. Processing - Processing is similar to that of STSGET with the exception that STAGET determines the command mode of a given HC if its actual mode is Transition.
- e. Error messages and recovery - None

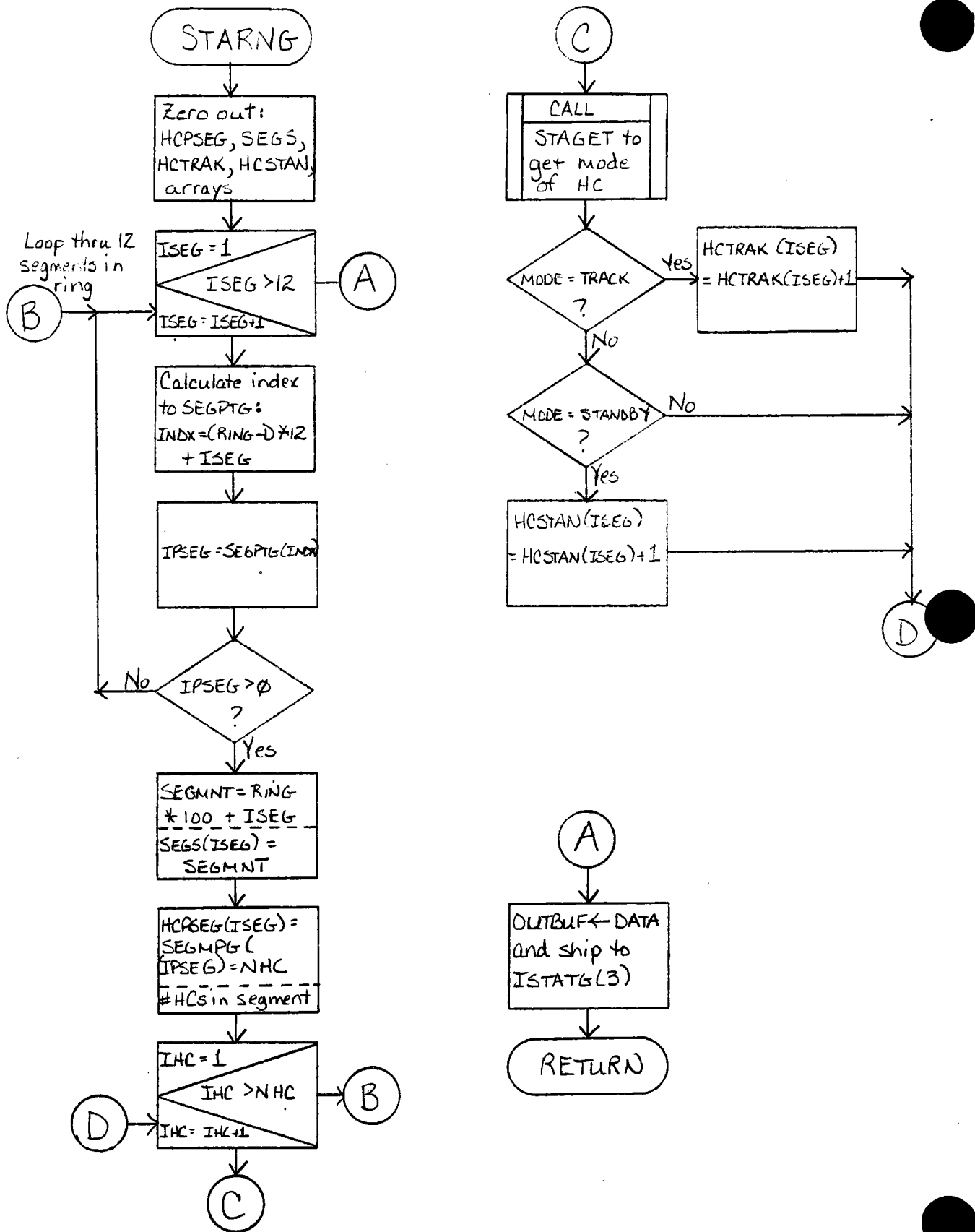


Figure 3.2.6-13 Flowchart - STARNG

3.2.6.4.1.8.2 Data, Logic and Command Paths

Input data:

HFCHC - the HFC-HC number (1 to 2048) whose actual and commanded modes are desired.

HCST1G - HC status word.

HCST2G - HC status word.

Output data:

MODE - the resultant mode number (-1 or 1 to 12).

CMODE - the command mode if MODE is transition.

3.2.6.4.1.8.3 Internal Data Description

The internal data used by STAGET is the same as for STSGET, but with the addition of:

a. MASKR - a mask used to find the commanded mode: isolates bits six through two of HCST1G (mode-submode only). MASKR = 007C Hex

b. MASK - similar to MASKR but with the "position compare" bit position set. MASK = 207C Hex

3.2.6.4.1.8.4 Flowchart

See Figure 3.2.6-14

3.2.6.4.1.9 Submodule IX - STAMCH

3.2.6.4.1.9.1 Description

a. Language used - FORTRAN

b. How invoked - Called by STA

c. Constraints and limitations - None

d. Processing -

1. Encode a message with the MASK and MATCH values, and call STACRT to output the message to the CS console.
2. Find the HC or HFCs which have
MATCH = 3 AND (VALUE, MATCH)

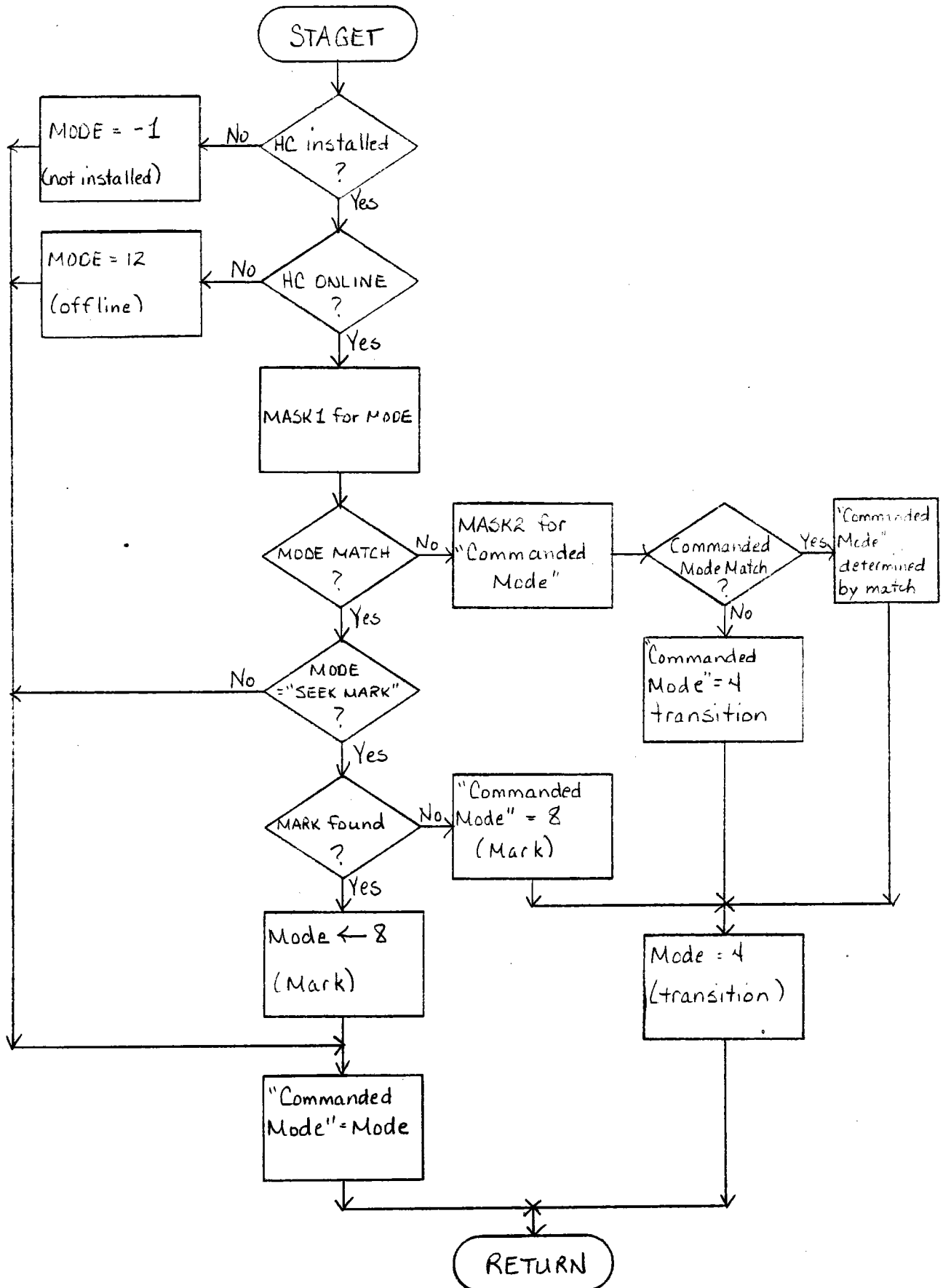


Figure 3.2.6-14 Flowchart - STAGET

Where:

<u>ISTATG(1)</u>	<u>VALUE</u>	<u>TYPE</u>
16	HCST1G	HC
17	HCST2G	HC
18	HFCS2G	HFC,

and load the MDAC HC numbers into an output buffer (holds one output line of data).

3. When buffer full, encode the HC numbers and call STACRT. Check for a partial buffer when done.
 4. Return to calling routine.
- e. Error messages and recovery - If no HCS or HFCs meet the above criteria, encode a message: "NO HCS WITH DESIRED MATCH." An enqueue failure message will also be written to the systems console.

3.2.6.4.1.9.2 Data, Logic, and Command Paths

Input data is local common.

3.2.6.4.1.9.3 Internal Data Description - None.

3.2.6.4.1.9.4 Flowchart

See Figure 3.2.6-15.

3.2.6.4.1.10 Submodule X - STACRT

3.2.6.4.1.10.1 Description

- a. Language used - FORTRAN
- b. How invoked - Called by STAMOD, STARNG, STAFLD, STAIND, STAMCH
- c. Constraints and limitations - None
- d. Processing -
 1. Get word count from byte count. Determine where command output goes.
 2. If output to OCS or DAS, go to Step 6.

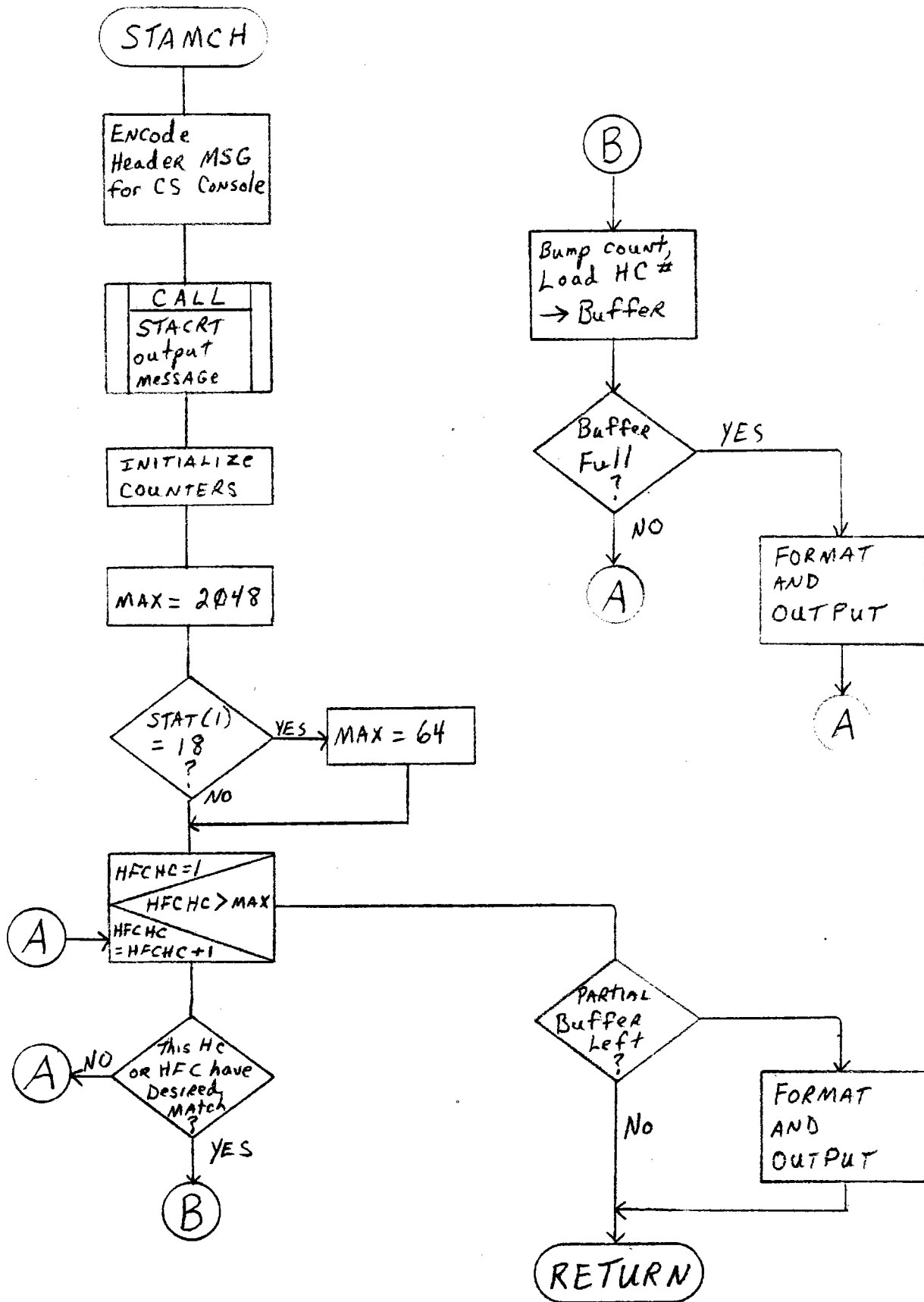


Figure 3.2.6-15
Flowchart - STAMCH

3. Enqueue message buffer to CS console. Write error message to system console if enqueue failure.
4. Remove color codes from message, and write to logging printer.
5. Go to Step 7.
6. Write message to OCS or DAS core device.
7. Return to calling routine.

e. Error message and recovery - Enqueue failure message written to system console.

3.2.6.4.1.10.2 Data, Logic, and Command Paths

Input data is local common.

3.2.6.4.1.10.2 Internal Data Description

3.2.6.4.1.10.4 Flowchart

See Figure 3.2.6-16.

3.2.6.5 Interface Description

STS stores the number of HCs in each mode in global array MODEG and the mode number of each heliostat in global array GRSTSG for use by the graphics display. Communication between MANMIF and STATUS is accomplished via global array ISTATG.

3.2.6.6 Test Requirements

STSGET and STAGET should first be tested to make sure that they perform the correct mapping from the mode representations in HCST1G and HCST2G to the reported mode. The other modules basically format the data decoded by STSGET and STAGET, and the formatted data can be checked by dumping the ASCII message buffer.

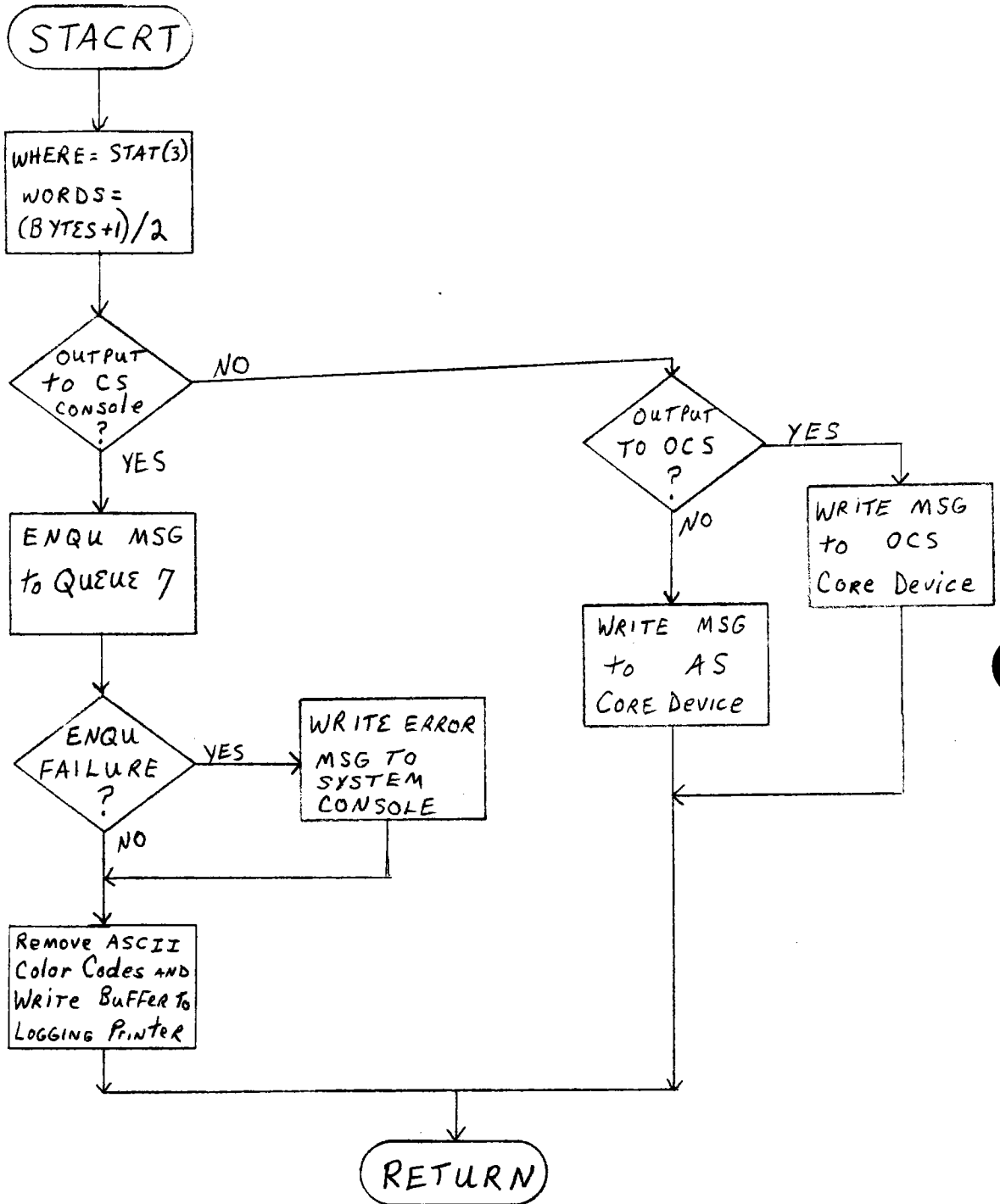


Figure 3.2.6-16
Flowchart - STACRT

3.2.7 Data Base Initialization Module - DBINIT

3.2.7.1 Purpose

The purpose of the Data Base Initialization Module (DBINIT) is to initialize the global data base, as well as the disk-resident data base. Additionally, DBINIT initializes the operator's CRT console, backup system and system interfaces, and provides initialization and synchronization of the universal and local system time base. Upon establishment of the time base reference, the timekeeping task is established and activated which, through subsequent action, will establish and activate the various tasks which are components of the heliostat system.

3.2.7.2 Requirements

3.2.7.2.1 Design Requirements

Software Requirements listed in Section 3.1 of the 10 MWe Collector Subsystem Software/Firmware Functional Requirements Specification, 12 June 1980, that apply to the DBINIT module are:

- a. Detect, report, and respond to failures and irregularities;
- b. Maintain a "Prime" and "Backup" system, as well as redundant field communications;
- c. Maintain a stable time base;
- d. Provide graphic displays of the heliostat field or field segments; and
- e. Provide a stable time base utilizing the WWV Trutime input for the "Prime" HAC time and internal time for the "Backup" HAC time. When "Backup" HAC becomes "Prime," the WWV time will automatically be used, if available.

3.2.7.2.2 Derived Requirements

Section 3.2.1.7 of the 10 MWe Collector Subsystem Software/Firmware Functional Requirements Specification, 12 June 1980, states the following requirements for the Data Base Initialization task:

- a. Define the global-common area, accessible to all applications tasks;
- b. Provide an initialization task to initialize the values in the global data base;
- c. Build appropriate data-base files for alarm messages, HC initialization coordinates, HC biases, control group mapping, BCS targets, Stow positions, Wash positions,

corridor coordinates, initialization azimuth and elevation positions, and multiple aim-points;

- d. Read WWV time values from the WWV device, if it is present and operating; and activate the "clock" task to perform time-base maintenance; and
- e. Initiate task execution sequences to the operating system, and allow for specification of "Prime" or "Backup" computer; and if "Prime" computer, make data available for transfer to "Backup," and if "Backup" computer, accept data from "Prime."

Additional derived requirements for DBINIT are:

- a. Perform reasonableness checking of the card-image source data before storing it onto the disk data base;
- b. Write error messages on the operator's console with respect to source data errors; and
- c. Provide a hard-copy listing (offline) of the card-image source upon operator request.

3.2.7.3

Design Approach

The design of the Data Base Initialization Module (DBINIT) is accomplished with a top-down functional allocation approach. This top-down approach facilitates both module and submodule development, implementation, and testing. Figure 3.2.7-1 exhibits the hierarchical allocation of the DBINIT required functions. Figure 3.2.7-1 shows that DBINIT is decomposed into three functional areas. The design structure of each functional area is shown in Figures 3.2.7-2 through 3.2.7-6. This design approach allows for maximum flexibility in requirement fulfillment and software maintainability.

The disk-resident data base is initialized with the offline task DIN utilizing Phase I developed software as much as possible. The card-image data source is error checked and converted from ASCII to internal binary representation and stored on its appropriate file. In total, there are 6 disk files created and filled offline. These files are the HC coordinates, the Wash angles, Stow angles, Alternate 1 Stow angles, the Alternate 2 Stow elevation angle, and the aim-point arrays. The file structures format and types are exhibited in Section 3.3.1. This offline initialization of the above files supports system flexibility, resource management, and timeliness of the initialization process. System flexibility is enhanced by the offline feature which allows the files to be updated in a batch or background process. Resource management of the disk space is improved by not storing the card-image source on the disk, and initialization timeliness is improved by eliminating operator handling of card decks and/or magnetic tapes during real-time initialization.

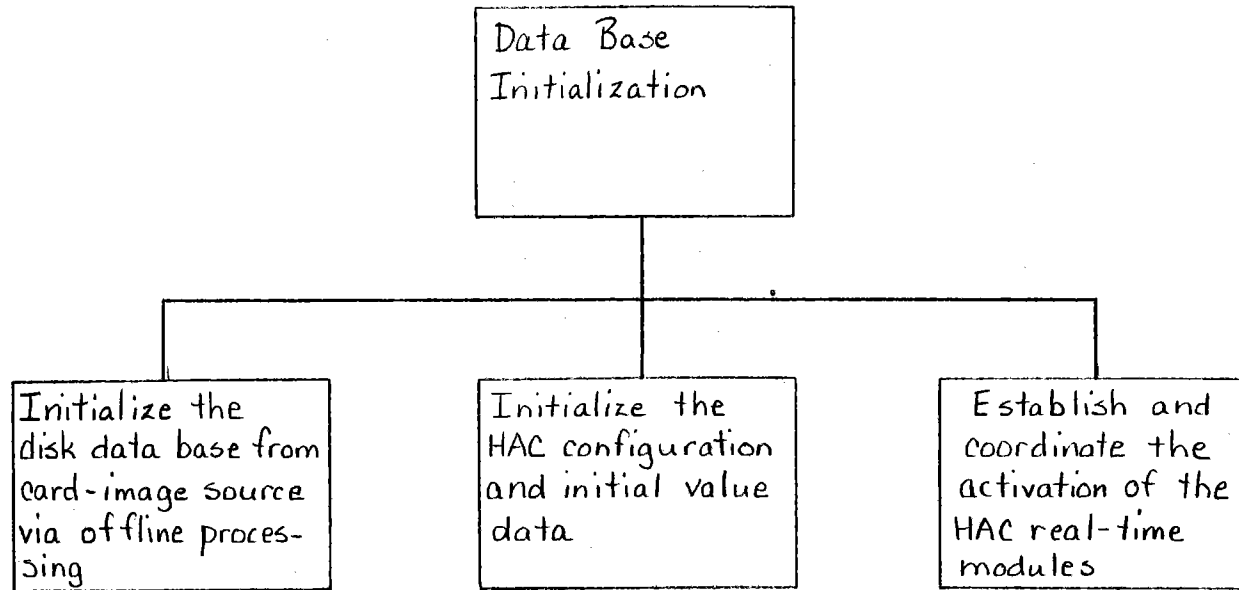


Figure 3.2.7-1 Hierarchy Diagram of the Data Base Initialization Module (DBINIT)

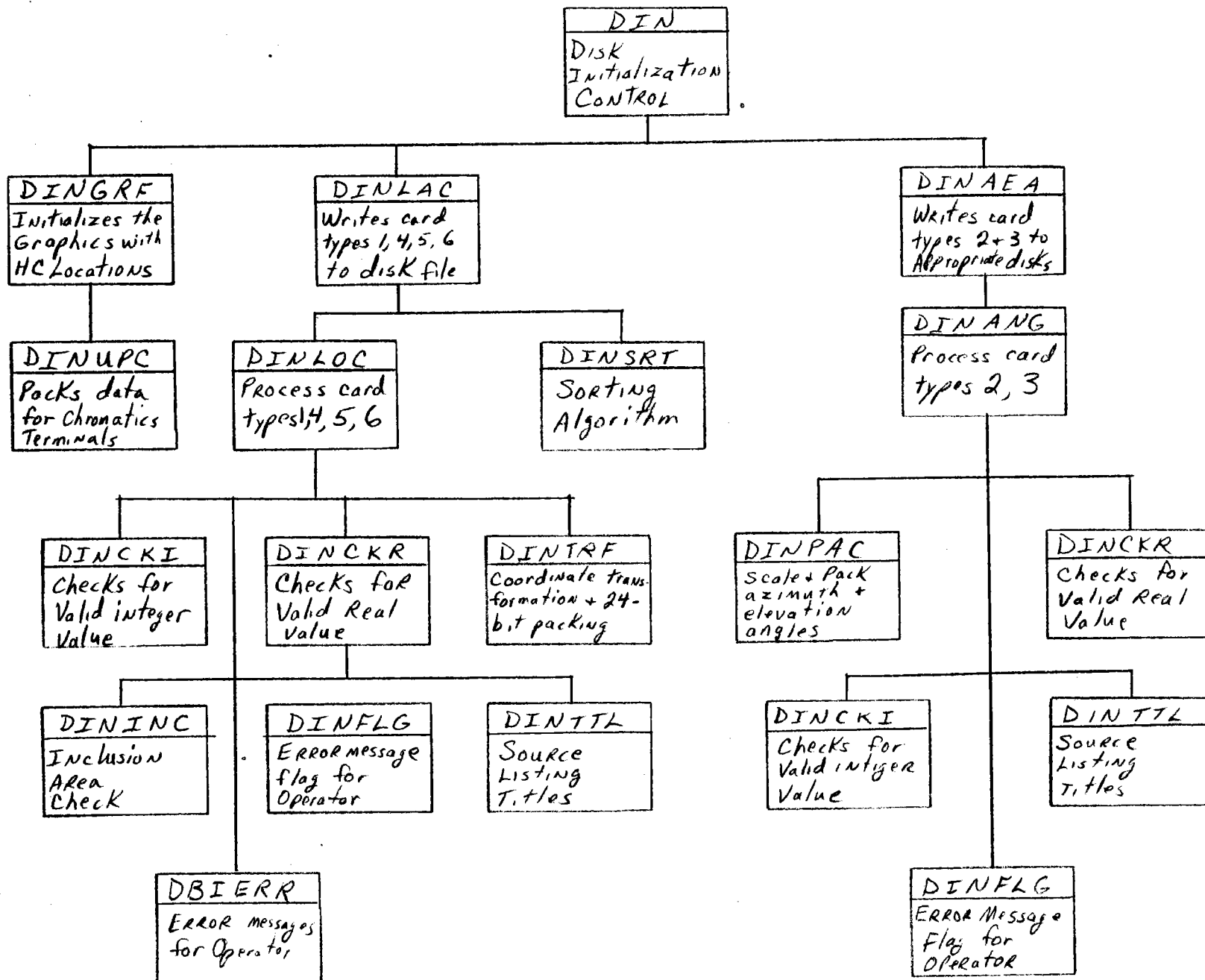


Figure 3.2.7-2 Functional Diagram of the Disk Initialization Task

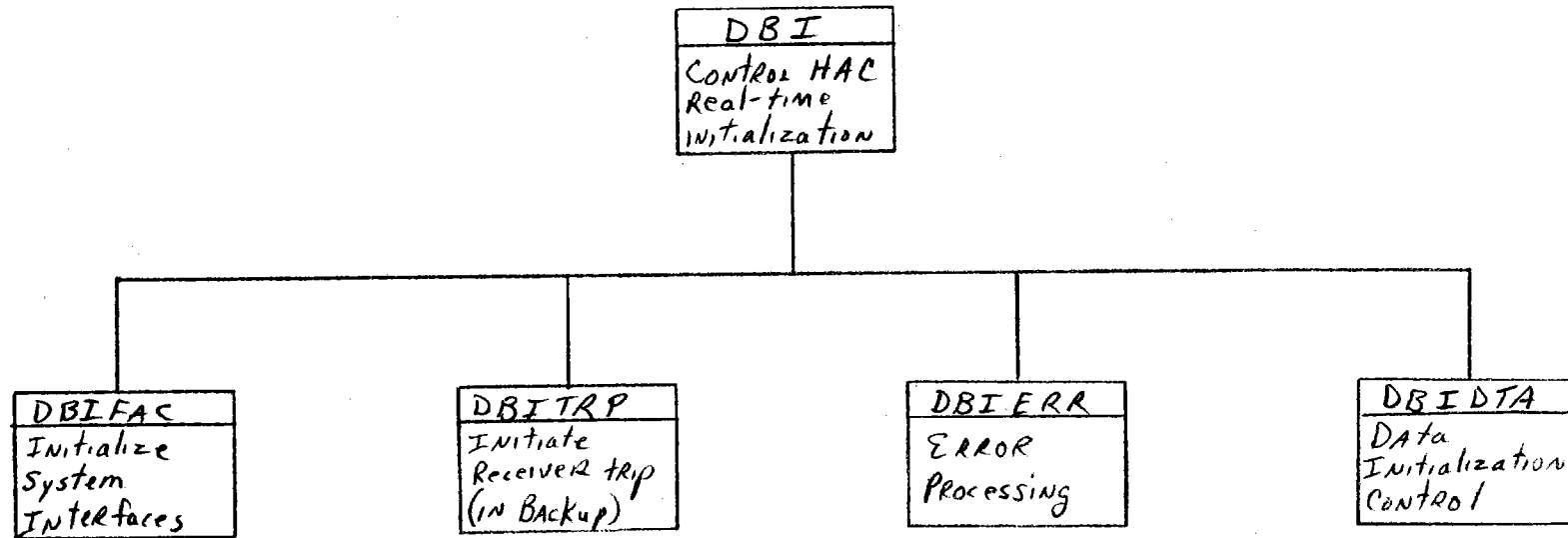


Figure 3.2.7-3 Functional Allocation of the HAC Initialization Task

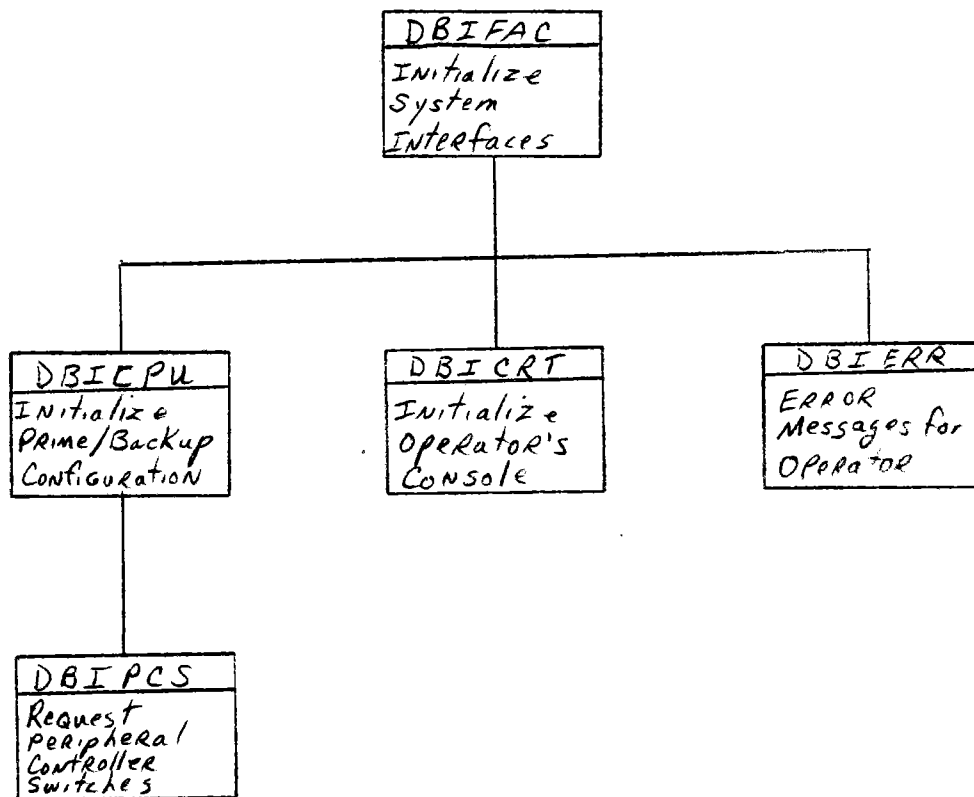


Figure 3.2.7-4 Functional Diagram of the System Interface Initialization Submodule

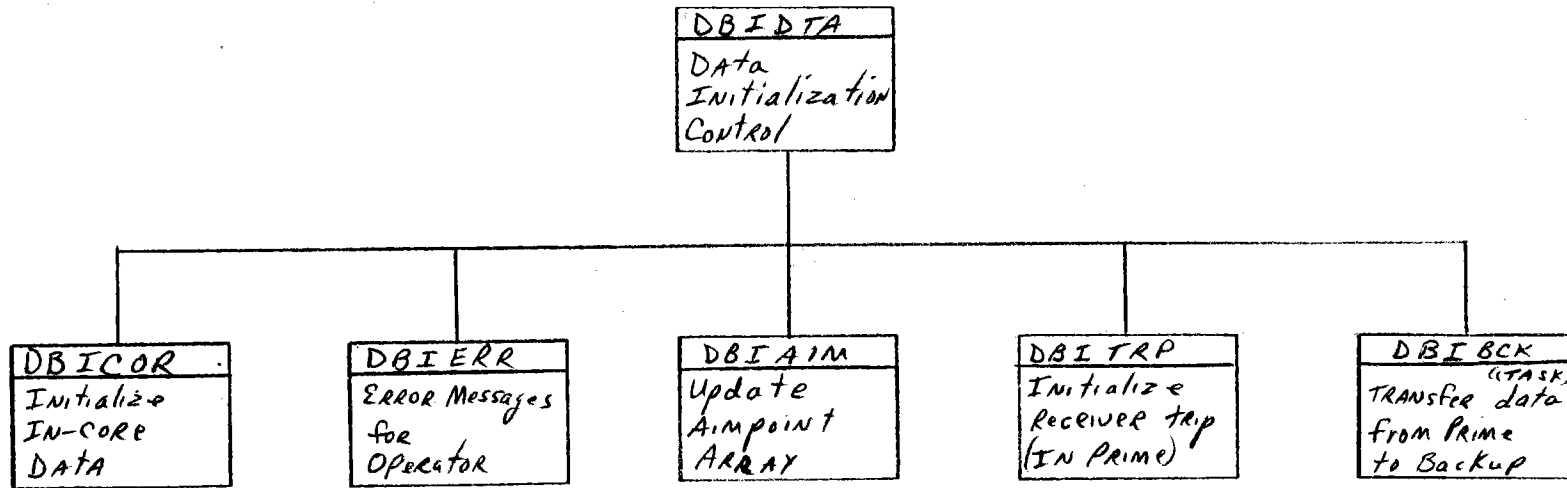


Figure 3.2.7-5 Functional Diagram of the Data Initialization Control Submodule

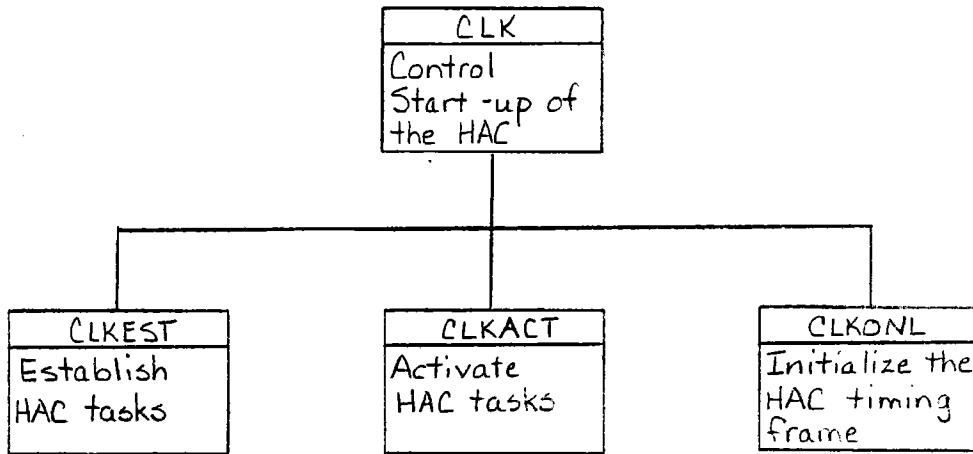


Figure 3.2.7-6 Functional Diagram of the HAC Start-up Task

The in-core data base and peripheral initialization is performed by a prescheduled task (DBI) which is activated at system boot via Sysgen. This initial task (DBI) performs the data initialization required then activates a second task (CLK) to initialize the real-time operations tasks. This design feature releases main memory storage required for DBI for establishment of the real-time operations tasks in core.

3.2.7.3.1

Functional Allocations

The offline disk data base initialization task (DIN) consists of 15 submodules and utilizes one submodule from the DBI task. The functional allocation of the submodules is shown in Figure 3.2.7-2. A brief description of each submodule follows:

- a. DIN - performs overall control of operator interfacing and processing of the card-image source data for the Heliostat assignments and locations, heliostat Wash and Stow positions, Alternate 1 and Alternate 2 Stow positions, heliostat aim points, BCS target coordinates, and the corridor definition coordinates. The format of these card-image source data are shown in Tables 3.2.7-I through 3.2.7-VI. This submodule presents a menu for operator selection to input the source data and whether a source listing is desired.
- b. DINLOC - processes source data from card-image types 1, 4, 5, and 6 (Tables 3.2.7-I, 3.2.7-IV, 3.2.7-V, and 3.2.7-VI). This submodule verifies the format, limit checks the values, performs coordinate transformation and data packing, inclusion area processing, data base structuring, and disk storage. As each card-image is processed, the card type is compared against the operator-requested data type for error processing. The values of the source data are processed in comparison to the values exhibited in Tables 3.2.7-I, 3.2.7-IV, 3.2.7-V, and 3.2.7-VI. If during the processing of an input source record an error is encountered, the error handling procedure shown in Table 3.2.7-VII is followed, and the source record is listed on the hard-copy device.
- c. DINTRF - performs the coordinate transformation from the California Lambert grid system to the right-handed orthogonal site reference system. Additionally, the X, Y, and Z transformed values are scaled and packed into 24-bit words. Both the transformed and packed values are returned to the calling submodule.
- d. DINSRT - is a sorting algorithm which sorts a subject array in ascending order. The subject array is not disturbed during the sorting procedure, but rather an array of pointers is established that point to the subject array in an ordered manner.

Table 3.2.7-I Data Base Card Format

Card Type 1 - Heliostat Assignments and Locations

X-Y-Z Coordinates in Lambert California Grid System with Units of Feet

CT HC HFC X-COORD. Y-COORD. Z-COORD. SEG P B W GEN.TIME
 XX XXXX XX †XXXX.XX †XXXX.XX XXXX.XX XXX XX X X XXXXXXXX

Card columns	Format	Description	Units
1 - 2	Integer	Card type designation, 01 for HAA cards.	N.D.
6 - 9	Integer	Heliostat identification number (HC). (row 1-29, position 1-99)	N.D.
12 - 13	Integer	Identification number of heliostat field controller assigned to HC (1-64)	N.D.
15 - 22	F.P.	HC X-coordinate. X is positive north. Zero value is at receiver tower vertical center line. (0 to +/- 1400)	Feet
24 - 31	F.P.	HC Y-coordinate. Y is positive east. Zero value is at receiver tower vertical center line. (0 to +/- 1400)	Feet
33 - 39	F.P.	HC Z-coordinate. Z is positive up. Zero value is at 1849 feet above mean sea level. (0 to + 500)	Feet
42 - 44	Integer	Segment number. First integer designates ring number, second two integers designate the wedge number. (ring 1 to 5, wedge 1 to 12)	N.D.
46 - 47	Integer	Pecking order number assigned to HC (1 - 50)	N.D.
49	Integer	BCS target number assigned to HC (1 - 4)	N.D.
51	Alpha	Corridor-walk letter assigned to HC (A - H)	N.D.
73 - 80	Integer	Generation time of this card.	N.D.
73 - 74	Integer	Month (01 - 12)	N.D.
75 - 76	Integer	Day (01 - 31)	N.D.
77 - 78	Integer	Year of 20th century (79 - 99)	N.D.
79 - 80	Integer	Hour on 24-hour clock. 13 hour at 1 PM. (00 - 23)	N.D.

426

Table 3.2.7-II Data Base Card Formats

Card type 2 - Heliostat Wash and Stow Positions

CT	HC	WASH		STOW		GEN.TIME
		AZ	EL	AZ	EL	
XX	XXXX	±XXX.XXX	±XXX.XXX	±XXX.XXX	±XXX.XXX	XXXXXXXX

Punched card format description for heliostat wash and stow positions cards

Card Columns	Format	Description	Units
1 - 2	Integer	Card type designation, (02)	N.D.
6 - 9	Integer	Heliostat identification number (row 1 - 29; position 1 - 99)	N.D.
16 - 23	F.P.	Azimuth angle (AZ). AZ is zero east and positive counter-clockwise. AZ specified 0 to +/- 180	Degrees
25 - 32	F.P.	Elevation angle (EL). EL is zero when mirror is vertical and positive with glass up. (0 to +/- 180).	Degrees
39 - 46	F.P.	Azimuth angle. (0 to +/- 180)	Degrees
48 - 55	F.P.	Elevation angle. (0 to +/- 180)	Degrees
73 - 80	Integer	Generation time of this card.	N.D.
73 - 74	Integer	Month (01 - 12)	N.D.
75 - 76	Integer	Day (01 - 31)	N.D.
77 - 78	Integer	Year of 20th century (79 - 99)	N.D.
79 - 80	Integer	Hour on 24-hour clock. 13th hour is 1 PM. (00 - 23)	N.D.

427

Table 3.2.7-III Data Base Card Format

Card Type 3 - Stow - Alternate 1 and 2 Positions

CT	HC	Stow - Alt 1		Stow - Alt 2		GEN.TIME
XX	XXXX	AZ	EL	AZ	EL	XXXXXXXX
		±XXX.XXX	±XXX.XXX	±XXX.XXX	±XXX.XXX	

Punched card format description for heliostat alternate positions cards.

Card Columns	Format	Description	Units
1 - 2	Integer	Card type designation, (03)	N.D.
6 - 9	Integer	Heliostat identification number (row 1 - 29; position 1 - 99)	N.D.
16 - 23	F.P.	Azimuth angle (AZ). AZ is zero east and positive counter-clockwise. AZ specified 0 to +/- 180.	Degrees
25 - 32	F.P.	Elevation angle (EL). EL is zero when mirror is vertical and positive with glass up. (0 to +/- 180)	Degrees
39 - 46	F.P.	Azimuth angle. (0 to +/- 180)	Degrees
48 - 55	F.P.	Elevation angle. (0 to +/- 180)	Degrees
73 - 80	Integer	Generation time of this card.	N.D.
73 - 74	Integer	Month (01 - 12)	N.D.
75 - 76	Integer	Day (01 - 31)	N.D.
77 - 78	Integer	Year of 20th century (79 - 99)	N.D.
79 - 80	Integer	Hour on 24-hour clock. 13th hour at 1 PM. (00 - 23)	N.D.

428

Table 3.2.7-IV Data Base Card Formats

Card type 4 - Heliostat Aim Points

CT	HC	PT	LOCATIONS (FEET)			AIM POINT IDENTIFIER	GEN.TIME
XX	XXXX	XX	X-COORD	Y-COORD	Z-COORD	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXX
			±XXXX.XX	±XXXX.XX	XXXX.XX		

Card Columns	Format	Descriptions	Units
1 - 2	Integer	Card type designation, 04 for HAP cards.	N.D.
6 - 9	Integer	Heliostat identification number (HC) (row 1 - 29; position 1 - 99)	N.D.
12 - 13	Integer	Aim Point number (1 - 20)	N.D.
15 - 22	F.P.	HC X-coordinate. X is positive north. Zero value is at receiver tower vertical center line.	Feet
24 - 31	F.P.	HC Y-coordinate. Y is positive east. Zero value is at receiver tower vertical center line.	Feet
33 - 39	F.P.	HC Z-coordinate. Z is positive up. Zero value is at 1849 feet above mean sea level.	Feet
44 - 70	Alpha-Num	Aim point identifier. Indicates time of year for which aim points were generated and generation data.	N.D.
73 - 80	Integer	Generation time of this card	N.D.
73 - 74	Integer	Month (01 - 12)	N.D.
75 - 76	Integer	Day (01 - 31)	N.D.
77 - 78	Integer	Year of 20th century. (79 - 99)	N.D.
79 - 80	Integer	Hour on 24-hour clock. 13th hour at 1 PM. (00 - 23)	N.D.

429

Table 3.2.7-V Data Base Card Format

Card Type 5 - BCS Target Coordinates

	BCS	LOCATION (feet)		
CT	NUM	X-COORD.	Y-COORD.	Z-COORD.
xx	x	+xxxx.xx	+xxxx.xx	xxxx.xx

Card Columns	Format	Description	Units
1-2	Integer	Card type designation, 05 for BCS targets	N.D.
9	Integer	BCS target number (1 to 4)	N.D.
15-22	F.P.	BCS target X coordinate. X is positive north. Zero value is at receiver tower vertical center line. (0 to +/- 30)	Feet
24-31	F.P.	BCS target Y coordinate. Y is positive east. Zero value is at receiver tower vertical center line. (0 to +/- 30)	Feet
33-39	F.P.	BCS target Z coordinate. Z is positive up. Zero value is at 1849 feet above mean sea level. (0 to +300)	Feet

430

SOURCE DATA ERROR HANDLING PROCEDURE

SOURCE DATA	DEFAULT SUPPLIED		ACTION	
	YES	NO	REJECT RECORD	REJECT FILE
Card type designator		X	X	
Heliostat number		X	X	
HFC assignment		X	X	
Heliostat Coordinates		X	X	
Pecking order assignment		X	X	
BCS target assignment		X	X	
Corridor assignment		X	X	
Wash Angles	X			
Alt 1 Stow angles	X			
Alt 2 Stow angles	X			
Aim-point number		X	X	X
Aim-point coordinates		X	X	X

Table 3.2.7-VII . Source Data Error Handling Procedure

- e. DINTTL - outputs descriptive titles to the hard-copy device when source data listings are requested. This submodule ejects a page, prints the requested title, and returns a variable containing the number of lines required to print the title.
- f. DINANG - this submodule verifies the format, limit checks the values, and performs data scaling and packing, and data base structuring. As each card image is processed, the card type is compared against the operator-requested data type for error processing. The source data values are processed in comparison to the values exhibited in Tables 3.2.7-II and 3.2.7-III. If during the processing of an input source record an error is encountered, the error-handling procedure shown in Table 3.2.7-VII is followed, and the source record is listed on the hard-copy device.
- g. DINPAC - scales and packs the azimuth and elevation angles into 16-bit integer words.
- h. DINFLG - indicates to the operator the occurrence of errors in source data. This submodule receives an array of flags corresponding to source data columns and a count of the errors. If the error count is zero, no output occurs; otherwise an 80-character buffer is filled with ASCII characters such that blank corresponds to no error in the column, or slash (/) corresponds to an error in the column. As the buffer is filled, the input indicator array is cleared. After the buffer is filled, it is output to the device indicated via an input parameter.
- i. DINGRF - initializes the graphic processors with the heliostat field locations. The locations are sent to the graphics in the format shown in Table 3.2.10-VIII. The heliostat locations are sent twice, first sorted by HFC-HC number, and then sorted by segment number, subsorted by pecking order. Execution of this submodule requires the concurrent execution of an initializing program in the graphics processors.
- j. DINLAC - processes source data from card-image types 1, 4, 5, and 6 (Tables 3.2.7-I, 3.2.7-IV, 3.2.7-V, 3.2.7-VI). Calls DINLOC for syntax and value checking; and if the data is O.K., it writes the data to the appropriate disk file. If an error is encountered, a message will be sent to the operator.
- k. DINCKI - this submodule checks that an integer value is between the upper and lower bounds passed by DINLOC or DINAEA.

1. DINCKR - this submodule checks that a real value is between the upper and lower bounds passed by DINLOC or DINAEA.
- m. DININC - this submodule checks for the upper and lower limits of the aimpoint values for the receiver. The upper and lower values are predetermined based on the row that the heliostat is in. If new values must be input, this program must be changed.
- n. DINAEA - This submodule processes source data from card-image types 2 and 3 (Tables 3.2.7-II and 3.2.7-III). Calls DINANG for syntax and value checking; and if the data is O.K., it writes the data to the appropriate disk file. If an error is encountered, a message will be sent to the operator.
- o. DINUPC - this submodule packs the data to be sent to the chromatics terminals into the format shown in Table 3.2.10-VIII.

The Data Base Initialization (DBI) task is composed of eleven submodules, which execute automatically upon operating system start-up. The basic purpose of each submodule is briefly described below:

- a. DBI - is a task that performs overall control of the initial operator and system interfacing, in-core data initialization, disk-resident data verification, and activation of the timing task (CLK) for start-up of the HAC operations tasks.
- b. DBIFAC - functions as a controlling submodule for three submodules. This submodule controls the process of initializing the operator's console, receiver trip, interrogation of the operator for prime or backup HAC designation, and operator notification of errors encountered.
- c. DBICPU - functions as a submodule of DBIFAC. DBICPU requests from the operator a designation of either "PRIME" or "BACKUP". When "PRIME" is input, submodule DBIPCS is called to configure the Peripheral Controller Switches (PCSs) to the Prime HAC and to master clear the 1907 communications handler. The global common word CPUSG of area COMIN11 is set to indicate the prime or backup designation.
- d. DBICRT - functions as a submodule of DBIFAC. The DBICRT clears the color terminal, selects color schemes, subdivides the console screen, and sets up screen protection boundaries. DBICRT notifies the operator that DBINIT is activated with the message: "DATA BASE INITIALIZATION".

- e. DBIPCS - functions as a submodule of DBICPU. When called, the DBIPCS functions to request the PCSs in order to connect one peripheral to the prime machine and master clear the 1907 communications handler.
- f. DBITRP - functions as a submodule of DBIDTA. The receiver trip initialization submodule connects the interrupt levels to the handlers, enables the interrupts, checks for proper state initialization, and reports trip status.
- g. DBIDTA - functions as a submodule of DBI. DBIDTA functions as a controlling submodule for five submodules. It controls the calling of submodules in performing global common initialization, bias and aim-point array initialization, data transfer for "Prime-Backup" configuration, receiver trip initialization, and error processing.
- h. DBICOR - functions as a submodule of DBIDTA. DBICOR initializes the global common data base for intertask coordination and communications. Section 3.3.1 displays the data words and arrays that comprise the global common data base. The data words and arrays are described in Section 3.3.1.4 (Global Common Data Base) as: Variable Name; Size; Initial Value; Description; All Modules (Tasks) that set; All Modules (Tasks) that use; Location; Transfer Frequency; and Format. DBICOR initializes these data to the initial values specified under the Initial Value descriptor. The global common data base is structured into three types of data: 1) data that is transferred to the "Backup" only at initialization; 2) data that is transferred every second; and 3) data that is transferred every eight seconds. Error monitoring is performed on all initialization which requires access to disk-resident initial values. An error code is returned to the calling module for operator notification and processing.
- i. DBIAIM - functions as a submodule of DBIDTA. DBIAIM verifies the disk-resident aim-point files through inclusion area processing; and when an aim-point disk file is verified, its file name is flagged in global common for use by the Man-Machine Interface task for UPATM processing. After the 20 files are processed and properly flagged, the flagged file with the lowest cardinal number designator is copied into the global common data base for real-time HAC tasks. An error code is returned from this submodule to its calling module for processing.

- j. DBIBCK - functions as a task of DBIBTA. DBIBCK initializes the "Backup" HAC computer. All the invariant global common data base areas are copied from the "prime" to the "Backup" computer.

- k. DBIERR - functions as a submodule of DBIDTA, DBIFAC, and the tasks DIN and CLK. DBIERR processes error codes and produces error messages for the operator. DBIERR processes an error code which is classified into three types: 1) no error; 2) non-fatal informative errors; and 3) fatal errors. An error code of zero results in no error; error codes in the range 1 to 20 are non-fatal errors; while error codes in the range 21 to 40 are fatal errors. Whenever a disk file cannot be accessed, or a task cannot be established or activated, the condition is considered fatal, and the DBINIT module will abort. This submodule writes the error message to the device(s) indicated through an input parameter(s).

The task CLK is composed of four submodules which are activated by the DBI task after DBI has completed its processing. The basic purpose of each submodule is briefly described below:

- a. CLK - is a task that controls start-up of the real-time HAC operations. CLK is defined as a one-shot task, activated by DBI. CLK controls the calling of three submodules which establish the real-time HAC tasks as resident in main memory, activate the TOK task, and activate the other real-time HAC operations tasks for field control.
- b. CLKEST - functions as a submodule of CLK. CLKEST establishes the HAC real-time tasks as main memory resident. The task three-character name is converted to "can" code for the REX, Establish service. Once the "can" codes are generated, the REX, Establish service #27 is utilized to process through the list of names. The condition code is monitored after each REX service for error detection, and reported via an error code returned to the calling module, CLK.
- c. CLKONL - functions as a submodule of CLK. CLKONL coordinates the starting of the HAC timing frame with the WWV device and the operator. When the WWV device is active, it is used as the primary timing device, but when WWV time is not available, this submodule accepts an initial time from the operator. When the operator initializes the time manually, the input must be coordinated with real time to within \pm TBD seconds for sun tracking accuracy. An error code is returned to the calling module for processing.
- d. CLKACT - functions as a submodule of CLK. CLKACT activates the HAC real-time tasks. The tasks previously established (CLKEST) are activated with a REX service #16. The condition code is checked for errors and an error code is returned to the calling module through the parameter list. If CLKACT is executing in the prime HAC it will also activate the SWI task in the backup HAC.

3.2.7.3.2

Resource Budgets

- a. Memory requirements (estimated)
 1. DIN - 10K words
 2. DBI - 15K words
 3. CLK - 5K words

- b. Timing - there are no timing constraints for DBINIT
- c. Priority
 - 1. DIN - Off-Line Task
 - 2. DBI - 75
 - 3. CLK - 165
- d. Global common and disk file usage
 - 1. Global common described in Section 3.3.1
 - 2. Disk file structure and format in Section 3.3.1

3.2.7.4 Design Description

3.2.7.4.1 Module Structure

DBINIT is composed of three tasks DIN, DBI, and CLK. The task DIN initializes the disk data base, DBI initializes the in-core global common data base and system configuration, and CLK establishes and activates the real-time HAC tasks for system operation. Each of these tasks, at the submodule level, is described below.

3.2.7.4.1.1 Submodule I - Task DIN

3.2.7.4.1.1.1 Description

- a. Language used - FORTRAN
- b. How invoked - Offline task activation by an operator.
- c. Constraints and limitations - The heliostat locations (card Type 1) must be processed before the graphics processors can be initialized. Heliostat locations (card Type 1) define a heliostat as being installed. Aim-point arrays must be separated in the input process by end-of-file marks. Errors found in processing an aim-point array cause the file to be rejected. Partial updates of the heliostat locations, Wash and Stow angles, and Alternate 1 and Alternate 2 Stow angles are accepted under operator direction. Aim-point array storage on the disk is performed in totality; all undefined aim points are stored as zeros (not defined).
- d. Processing -
 - 1. Define the disk files for usage by this task.
 - 2. Write the operator's options for input type to system console:

OPTION SELECT ONE DATA TYPE FOR INPUT

- 1 INPUT HELIOSTAT LOCATIONS
- 2 INPUT WASH AND STOW ANGLES
- 3 INPUT ALT1 AND ALT2 STOW ANGLES
- 4 INPUT AIM-POINT ARRAY
- 5 INPUT BCS TARGET LOCATIONS
- 6 INPUT CORRIDOR LOCATIONS

- 7 INITIALIZE GRAPHICS WITH HC LOCATIONS
- T TERMINATE TASK

3. Request the operator's response:

ENTER OPTION FROM ABOVE LIST:

Read the operator's response.

4. Test if the response is valid. If it is, go to step (5); if not, go to step (3).
5. Test if the terminate task option was selected. If it was, EXIT the task; if not, go to step (6).
6. Test if the selected option requires input source data. If it does, go to step (7); otherwise, go to step (19).
7. Write operator's options for a full or partial update:

OPTION

- 1 PARTIAL UPDATE (HC LOCATIONS, WASH, STOW, ALT1, ALT2)
- 2. FULL UPDATE
- T TERMINATE TASK

8. Request operator's response::

ENTER OPTION FROM ABOVE LIST:

Read the operator's response.

9. Test if the response is valid. If it is, go to step (10); otherwise go to step (8).
10. Test if the terminate task option was selected. If it was, EXIT the task; if not, go to step (11).
11. Write operator's options for source device:

OPTION	SOURCE INPUT DEVICE
1	MAG TAPE
2	CARD READER
T	TERMINATE TASK

12. Request operator's response:

ENTER OPTION FROM ABOVE LIST:

Read the operator's response.

13. Test if the response is valid. If it is, go to step (14); otherwise, go to step (12).

14. Test if the terminate task option was selected. If it was, EXIT the task; if not, go to step (15).

15. Write the operator's options for requesting a source listing:

OPTION	SOURCE LISTING
0	NO
1	YES
T	TERMINATE TASK

16. Request operator's response:

ENTER OPTION FROM ABOVE LIST:

Read the operator's response.

17. Test if the response is valid. If it is, go to step (18); otherwise, go to step (16).

18. Test if the terminate task option was selected. If it was, EXIT the task; if not, request from the operator if the source is ready for input:

OPTION	IS SOURCE READY FOR INPUT
C	CONTINUE
T	TERMINATE

ENTER OPTION FROM ABOVE LIST:

Read the response and check it for validity. Continue requesting response until a valid entry is made by the operator. When a valid response is received, test if it is the terminate option. If it is, EXIT the task; if not, go to step (19).

19. Branch on the source type selected. If source Type 1, 4, 5, or 6 was selected, go to step (20). If source Type 2 or 3 was selected, go to Step (21). If source Type 7 was selected, go to step (22).
20. Call DINLAC to coordinate the validity checking and subsequent disk update. Go to step 2.
21. Call DINAEA to coordinate the validity checking and subsequent disk update. Go to step 2.
22. Call DINGRF to coordinate the updating of the graphic processors.

e. Error messages and recovery - None

3.2.7.4.1.1.2 Data, Logic and Command Paths

Input data:

- a. Operator inputs via the system console.

Output data:

- a. Menus and prompts for operator.

Algorithms:

Three arrays (Figure 3.2.7-7) are internally created to convert from the MDAC numbering system to the HFC-HC numbering system and vice versa. The arrays MD2HCG and MDNPRG are used to map from the MDAC number to the HFC-HC number. The MDAC number is composed of two parts: the row number and row-position number. The row number is used to enter the MDNPRG array and get the subtotal of all HCs in the

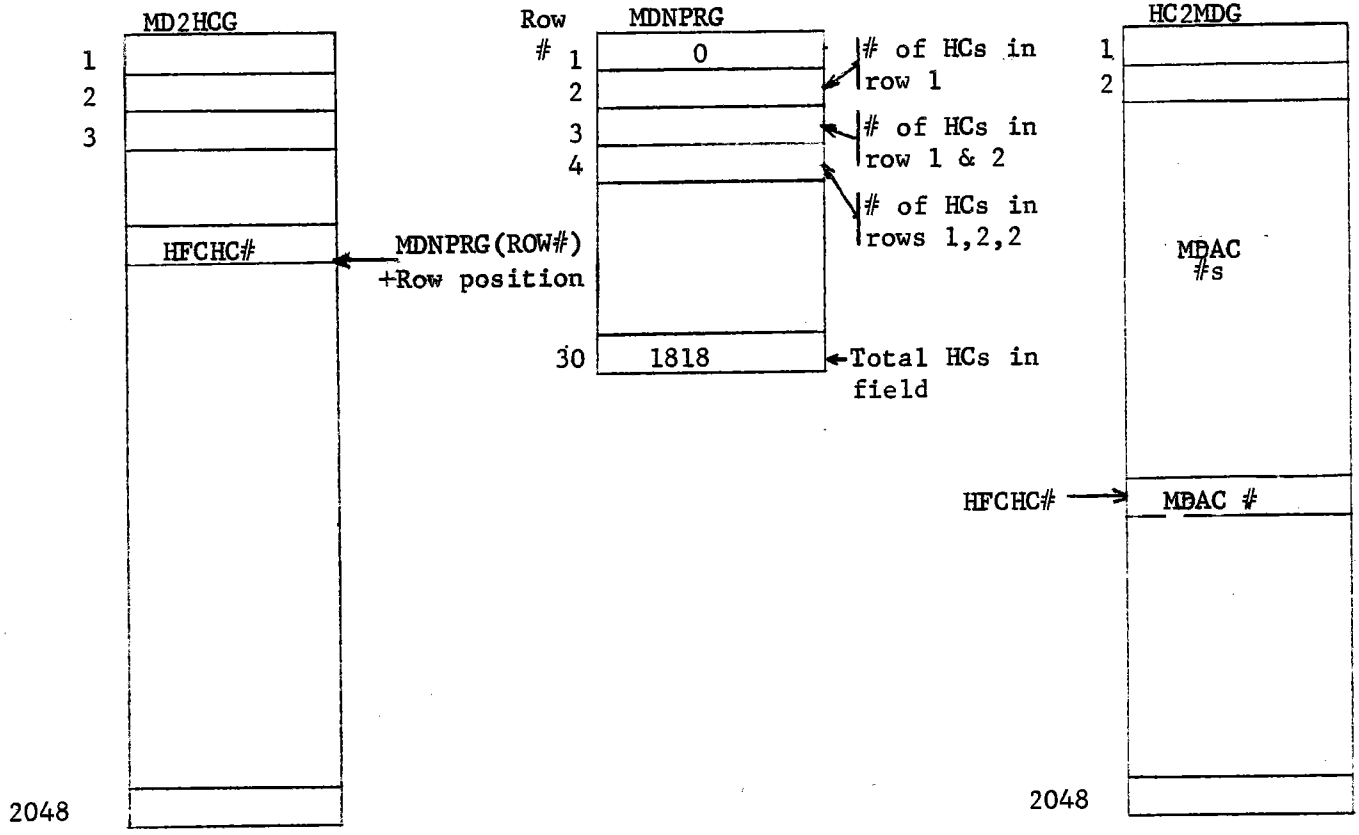


Figure 3.2.7-7 HC Numbering Scheme Mappings

lower number of rows. This is used as a base to which the row-position number is added for entry into the MD2HCG array. The MD2HCG array contains the HFC-HC number corresponding to the MDAC number. Error checking of the MDAC number is performed by comparing the entry number against contents of the next highest row number of the MDNPRG array. If the entry number is greater, then the MDAC number is invalid. The length of the MD2HCG array is 2048 elements, and the length of the MDNPRG is 30 elements where element one is zero and element 30 the total of all heliostats (i.e., 1818). The third array is the HC2MDG array and is used to map from the HFC-HC number to the MDAC number. The entry number (element index) is the HFC-HC number. The contents of the element is the MDAC number. Elements that have a value of zero are invalid HFC-HC numbers. The length of HC2MDG is 2048 elements.

When the heliostat locations are input, two arrays are created to map the segment numbers into groups of heliostat numbers. These two arrays (Figure 3.2.7-8) are the segment-mapping array (SEGMFG) and the segment-pointer array. The segment-pointer array contains pointers to the segment-mapping array; where the number of heliostats in that segment are stored (negative value) followed by the heliostat numbers of the heliostats in that segment in pecking order. If a zero value is found in the segment-pointer array, it means that there are no heliostats assigned to that segment number. The segment number is decomposed into its component parts of ring number and wedge number to index into the elements of the segment-pointer array. The value one is subtracted from the ring number, and the results are then multiplied by 12. The wedge number is added. This results in an index number from 1 to 60 (5 rings - 12 wedges).

3.2.7.4.1.1.3 Internal Data Description

a. Internal common areas - DINCOM:

1. 7 arrays of 2048 I*2 words used to process source data to disk files and vice-versa
2. 9 arrays of 8 I*2 words
3. HC2MDG array 2048 I*2 words
4. MD2HCG array 2048 I*2 words
5. MDNPRG array 30 I*2 words

b. Disk file DIN layout:

1. Random access file DIN; record size 128 words.

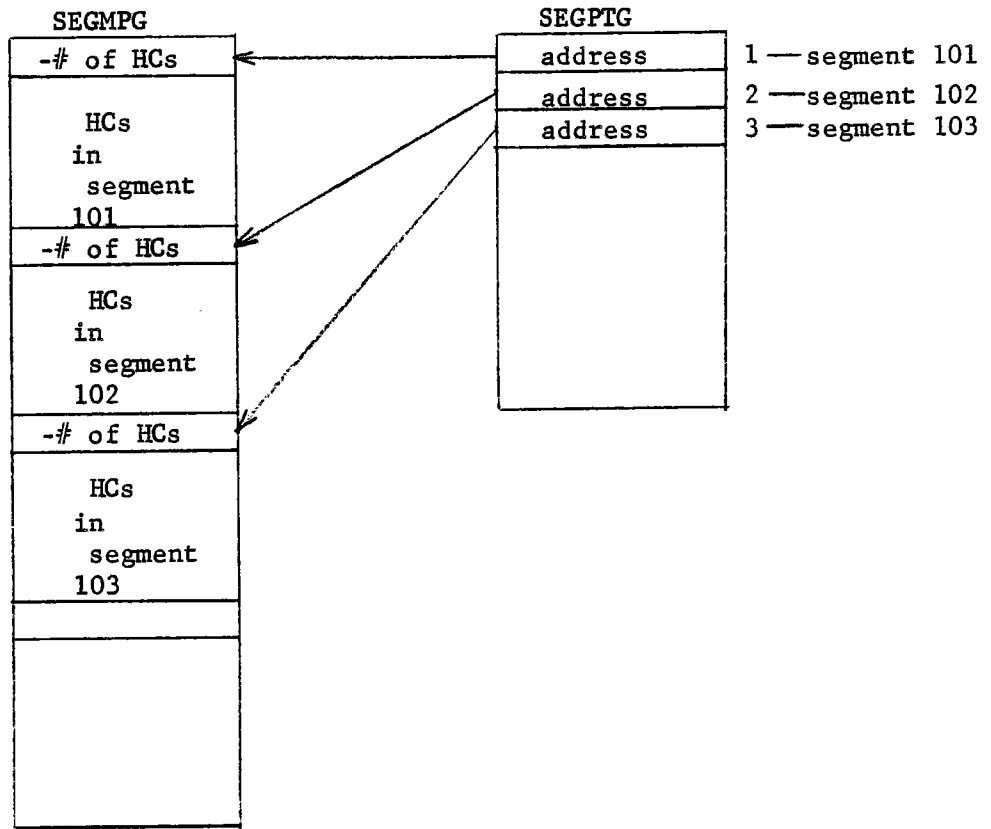


Figure 3.2.7-8 Segment Map and Segment Pointer Interaction

2. Records 1-16 contain the MDAC to HFC-HC number mapping MD2HCG array (2048 words).
3. Record 17 contains the MDAC number per row MDNPRG array (30 words - word 0 to word 29) and the segment-pointer array SEGPTG (60 words - word 50 to word 109).
4. Records 18-33 contain the HFC-HC to MDAC number mapping array HC2MDG (2048 words).
5. Records 34-50 contain the segment-mapping array SEGMPG (2108 words - Record 50 has the first 60 words used).
6. Record 50 (words 70 to word 93) contains the BCS target locations.
7. Record 51 contains the corridor locations (128 words).
8. Records 52-67 contain the HC segment number and pecking-order number packed together (2048 words).

3.2.7.4.1.1.4 Flowchart

See Figure 3.2.7-9 for DIN flowchart.

3.2.7.4.1.2 Submodule II - DINLOC

3.2.7.4.1.2.1 Description

- a. Language used - FORTRAN
- b. How invoked - Subroutine call by DINLAC
- c. Constraints and limitations - Records read from the source must contain an end-of-file indication at the conclusion of the file. Source input records cannot be duplicated. Card-image source types should not be intermixed. Aim-point arrays must be processed one at a time.
- d. Processing -
 1. Initialize:
 - a) End-of-file flag false
 - b) Read error counter to zero
 - c) Number of lines printed to zero
 - d) Number of records accepted to zero

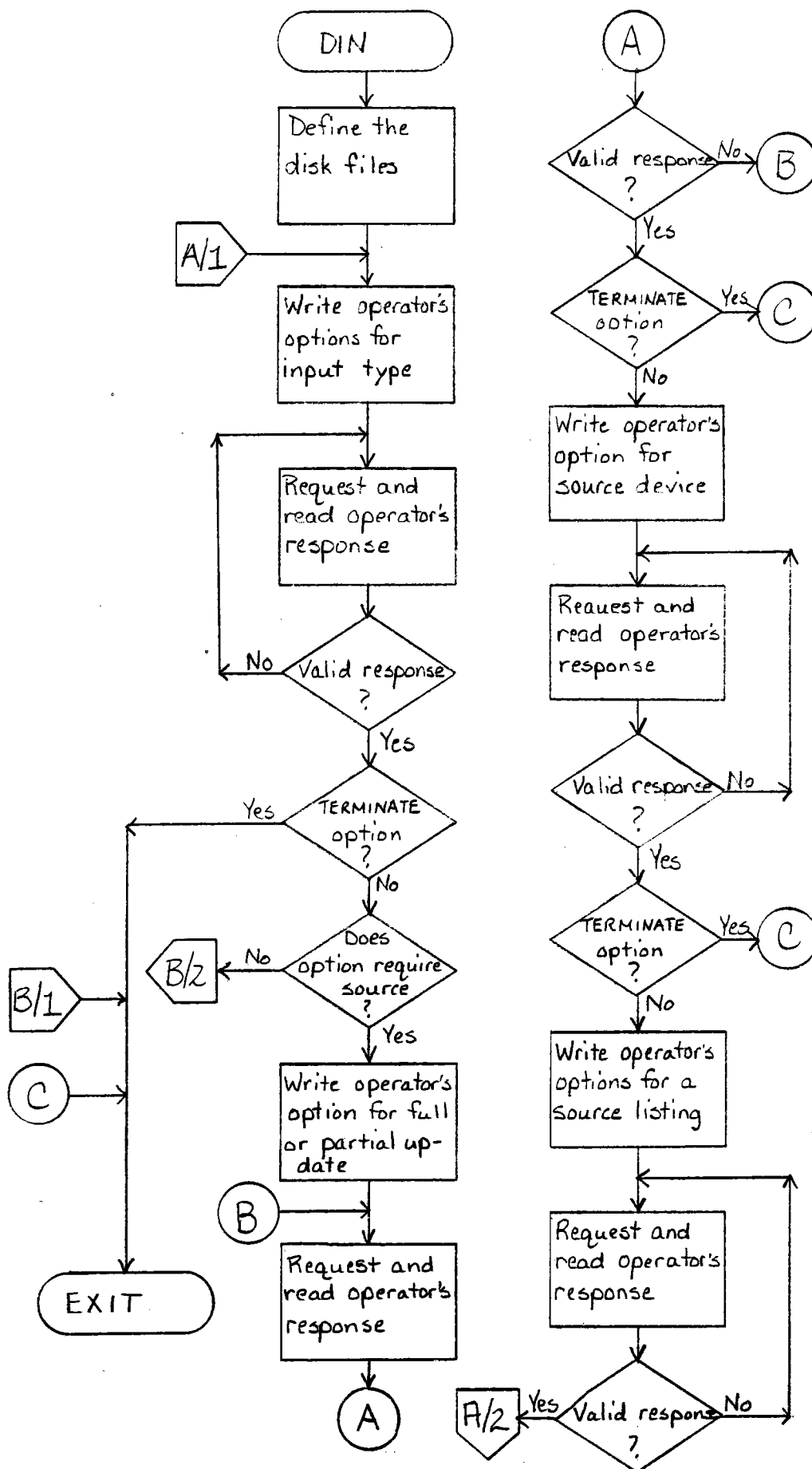


Figure 3.2.7-9 Flowchart - DIN

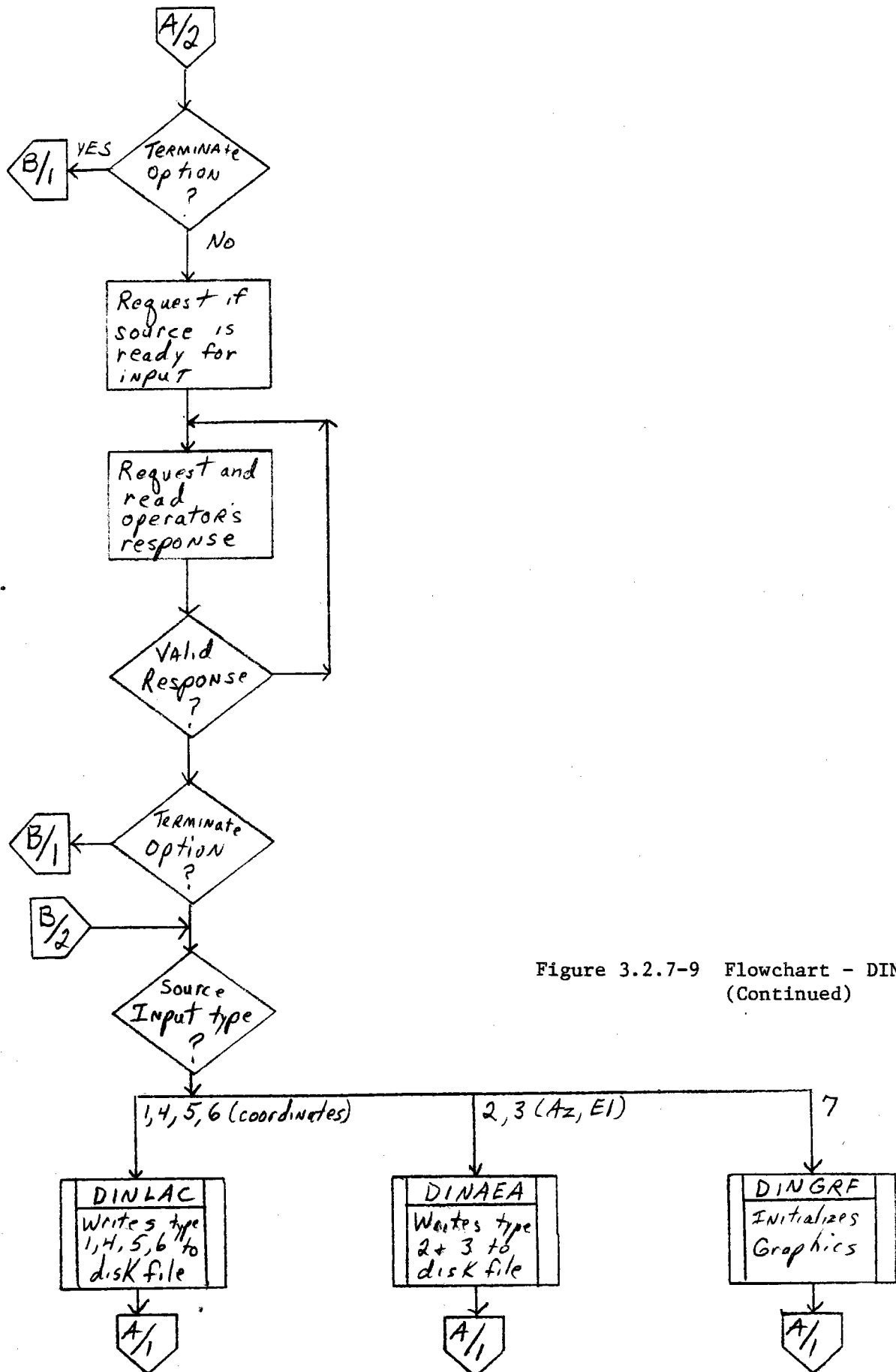


Figure 3.2.7-9 Flowchart - DIN (Continued)

- e) Number of records read to zero
 - f) Data accepted flag false
 - g) Reject file flag false
2. Branch on the source type being processed. Set the appropriate syntax masks for checking the source data types. Set the limit check values for the appropriate source data type.
 3. Initialize:
 - a) Error code to zero
 - b) Number of errors to zero
 - c) Column flag counter to zero
 4. Increment the number of records read, and read one record from the source device.
 5. Test if a read error occurred. If it did, set the error code, set the error counter to one, and increment the number of read errors. Then go to step (20). If no read error occurred, go to step (6).
 6. If an end-of-file was read, set the end-of-file flag true, and decrement the number of records read. Then go to step (20); otherwise, go to step (7).
 7. Initialize the syntax checking pointer to zero.
 8. Increment the syntax column pointer and syntax check the column with the syntax masks.
 9. If there is a syntax error, set the syntax error code, set the number of errors to one, set the error indicator flag in the column, and increment the flag counter; otherwise, go to step (10).
 10. Test if all the columns have been syntax checked. If they have, go to step (11); otherwise, go to step (8).
 11. Test if the card type and HC number are acceptable, if not, go to step (20); otherwise go to step (12).
 12. Decode the card type being processed. If a decode error occurs, set the reject-the-error code and increment the error counter. Then go to step (20); otherwise, go to step (13).

13. Set an error flag false. Then branch on the card type being processed. If card Type 1 is being processed, go to step (14). If card Type 4 is being processed go to step (28). If card Type 5 is being processed, go to step (33), and if card Type 6 is being processed go to step (38).
14. Card Type 1, check if the following are in range by calling either DINCKI or DINCKR:
 - a) HC number;
 - b) HFC number;
 - c) Segment number;
 - d) Pecking-order number;
 - e) BCS assignment number;
 - f) Corridor assignment letter; and
 - g) X, Y, Z coordinate values.

If any of these values are not in range, set the corresponding column indicator flags and set the error flag true.

15. Test if the error flag is set. If it is, set the error code word, increment the error counter and go to step (20). Otherwise, go to step (16).
16. Call the subroutine DINTRF to transform the coordinate values and scale and pack the values into five I*2 words.
17. Add the BCS and corridor assignments to the low-order byte of the fifth word.
18. Increment the number of records accepted and set the accept-file flag true.
19. Store the coordinate value words, pecking order, and segment number into local common arrays relative to the HFC-HC number address.
20. Test if a source listing or an error code was set. If either is true, go to step (21); otherwise, go to step (26).
21. Increment the line number counter.
22. Call the subroutine DINTTL which will produce a title heading if line number one is being printed.
23. Write the source ASCII record to the printer.

24. Call the subroutine DINFLG which will flag any indicated column errors.
25. Call subroutine DINERR which will write any error messages indicated.
26. Test if the end-of-file flag is true. If it is, go to step (27). Otherwise, test if three read errors have occurred. Go to step (3) if they have and set the data-accepted flag false; return to the calling subroutine.
27. Test if source type 4 is being processed (aim points); if false, return to the calling subroutine. Otherwise, test the reject-file flag. If the flag is true, return to the calling subroutine. If the flag is false, set the data-accepted flag true, and then return to the calling subroutine.
28. Card Type 4 - check if the following are in range by calling either DINCKI or DINCKR:
 - a) HC number; and
 - b) Aim-point coordinate values (X,Y,Z).

If any of these values are not in range, set the corresponding column indicator flags and set the error flag true.

29. Test if the error flag is set. If it is, set the error-code number and increment the error counter and flag counter, and go to step (20). otherwise, go to step (30).
30. Call the subroutine DINTRF to transform the aim-point coordinates, and scale and pack the values into five I*2 words.
31. Call the subroutine DININC to test if the aim point is in the inclusion area.
32. Test if the aim point is in the inclusion area. If it is, add the aim-point number to the low-order byte of word five; increment the number of records accepted; store the aim-point coordinate values into the local common arrays relative to the HFC-HC number; and store the aim-point number into a local common array relative to the HFC-HC number. Then go to step (20). If the aim point is not in the inclusion area, set the error code, increment error count, set the column flags and increment their count, and set the reject-file flag true. Then go to step (20).

33. Card Type 5 - check if the following are in range by calling either DINCKI or DINCKR:

- a) BCS target designator; and
- b) BCS target coordinates.

If any of these values are not in range, set the corresponding column-indicator flags and set the error flag true.

34. Test if the error flag is set. If it is, set the error code and increment the error counter; then go to step (20). Otherwise, go to step (35).
35. Call the subroutine DINTRF to transform the coordinate values; scale and pack the values into five I*2 words.
36. Store the scaled and packed values into local common arrays relative to the BCS target designator.
37. Increment the number of records stored, and set the data-accepted flag true; go to step (20).
38. Test if the corridor indicator is valid by calling DINCKI; if it is, go to step (38). If it is not valid, set the column flags, increment the count, and set the error flag true; then go to step (39).
39. Test if the lower-limit point is in range by calling DINCKR for X, Y, and Z. If it is not in range, set the column flags, increment the count, and set the error flag true.
40. Test if the upper-limit point is in range by calling DINCKR for X, Y, and Z. If it is, call the transformation subroutine DINTRF. If it is not in range, set the column flags, increment the count, and set the error flag true.
41. Calculate the delta, and call the transformation subroutine DINTRF.
42. Test if the error flag is set. If it is, set the error code and increment the error counter; then go to step (20). If it is not set, go to step (43).

43. Store the corridor values into local common relative to the corridor indicator.
44. Increment the number of records accepted and set the data-accepted flag true. Go to step (20).

e. Error messages and recovery -

1. Error messages:

Set appropriate error code, and call DBIERR.

2. Error recovery -

Return to calling subroutine with data-accepted flag false.

3.2.7.4.1.2.2 Data, Logic and Command Paths

Input data:

- a. Source device number;
- b. Source listing requested;
- c. Source type number;
- d. Source data; and
- e. Conversion mappings for MDAC to HFC-HC numbers.

Output data:

- a. Processed source data to local common;
- b. Number of records accepted; and
- c. Data-accepted flag.

3.2.7.4.1.2.3 Internal Data Description

Data processed from the source input is stored in local common area DINCOM. The storage of accepted records is always relative to an indicator on the source record. This storage procedure allows for partial updating of the processed records to disk. The HFC-HC numbering system to MDAC numbering system mapping arrays and vice-versa are contained in local common.

3.2.7.4.1.2.4 Flowchart

See Figure 3.2.7-10 for the DINLOC flowchart.

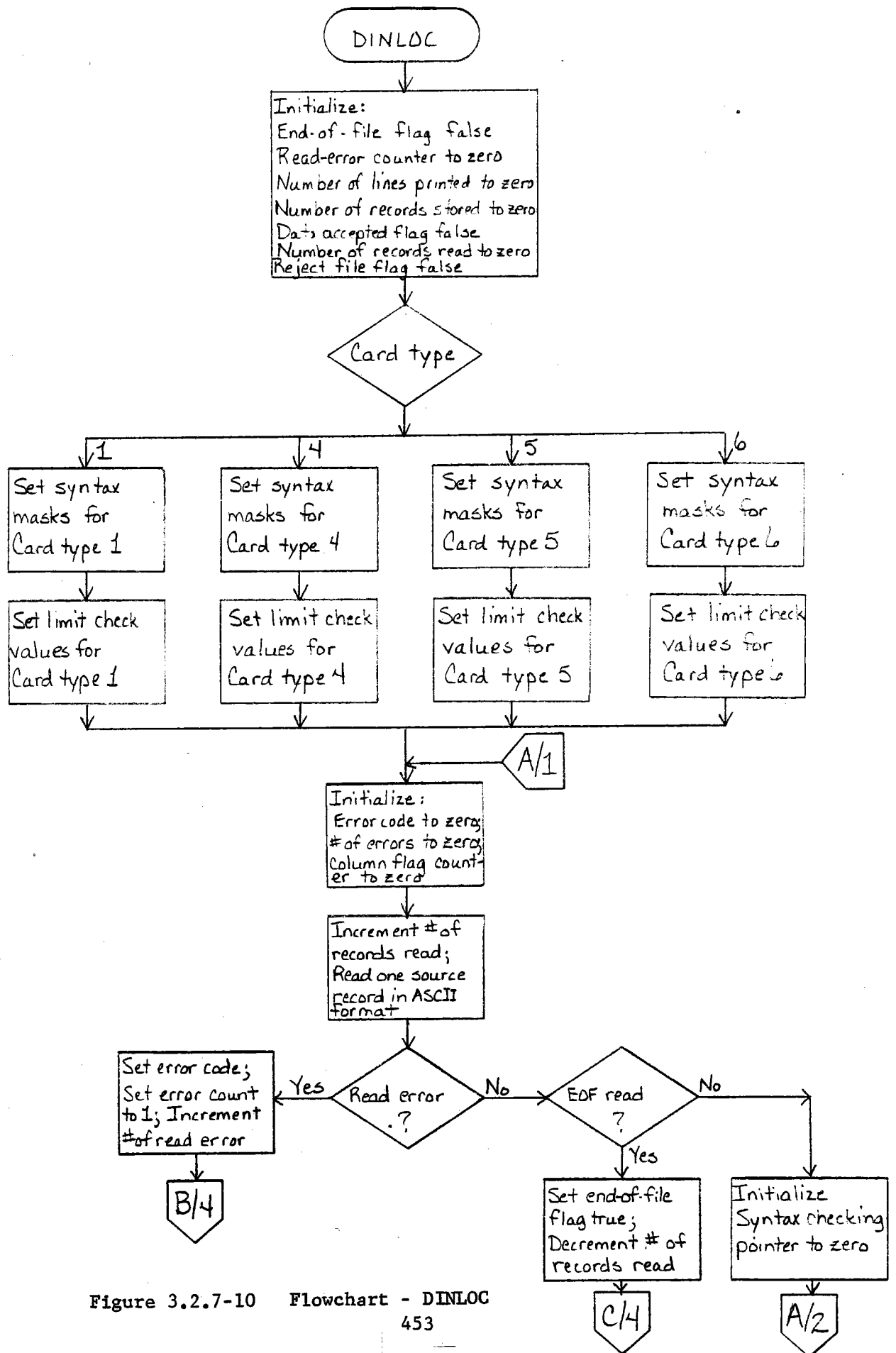


Figure 3.2.7-10 Flowchart - DINLOC
453

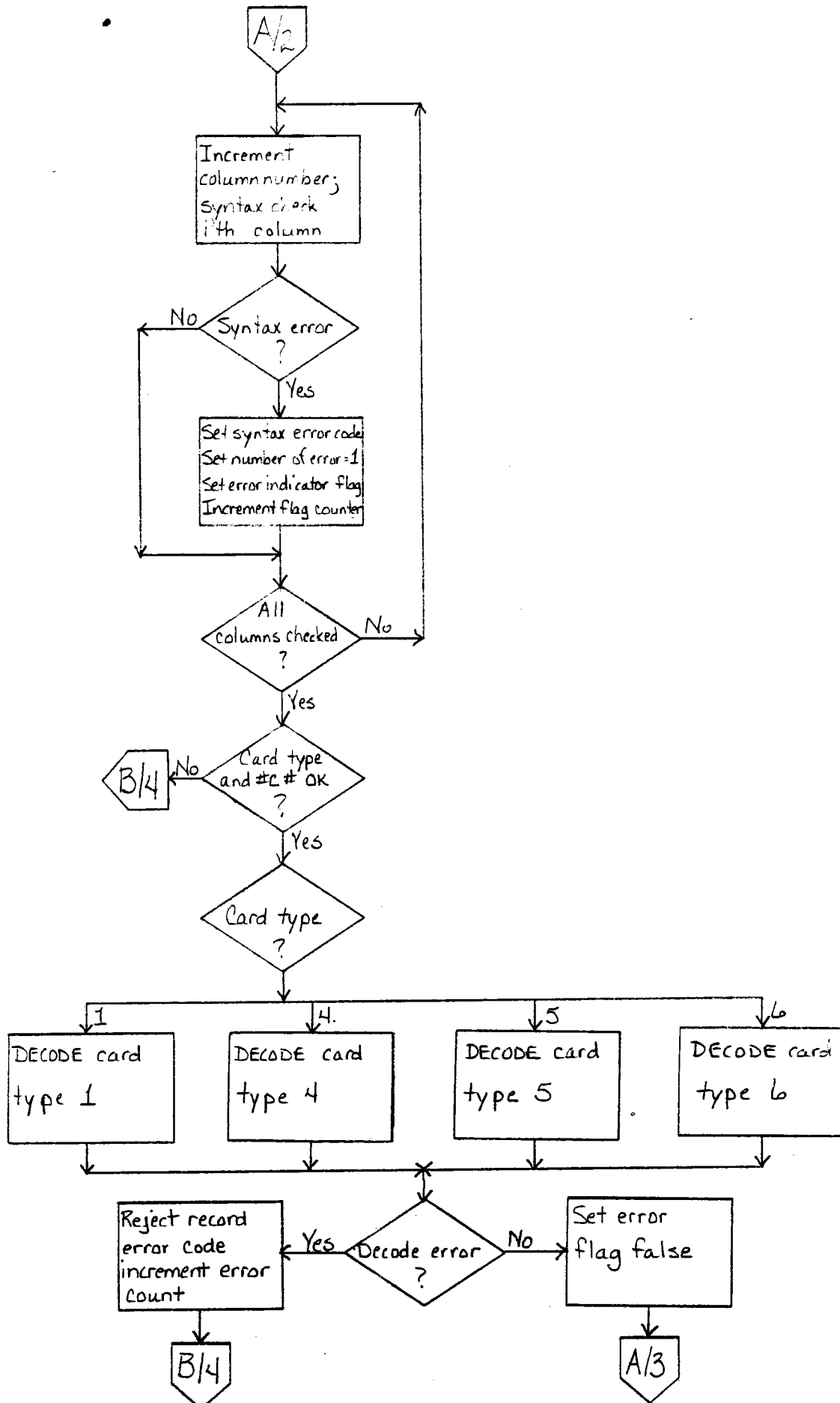


Figure 3.2.7-10 Flowchart - DINLOC (continued)

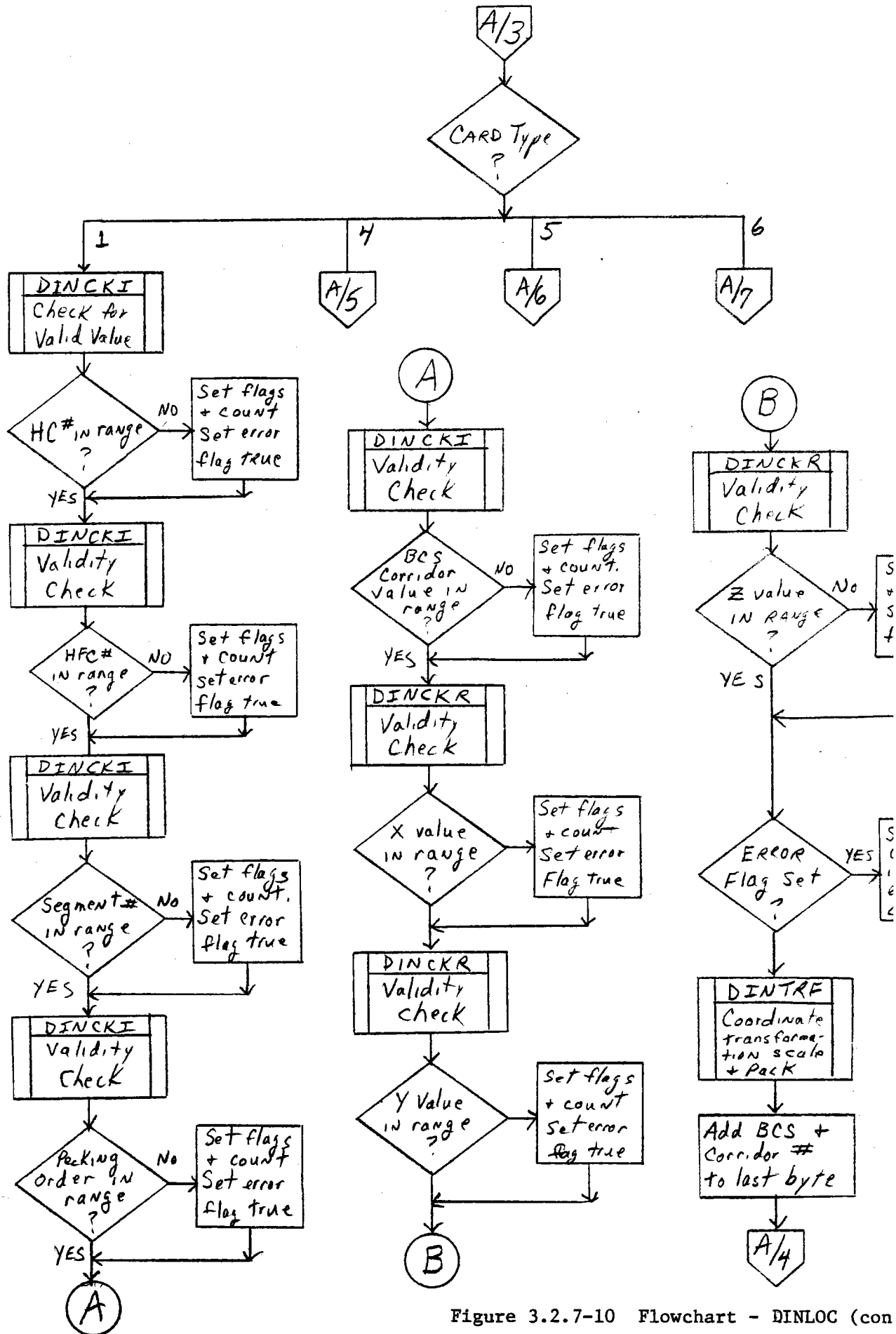


Figure 3.2.7-10 Flowchart - DINLOC (con

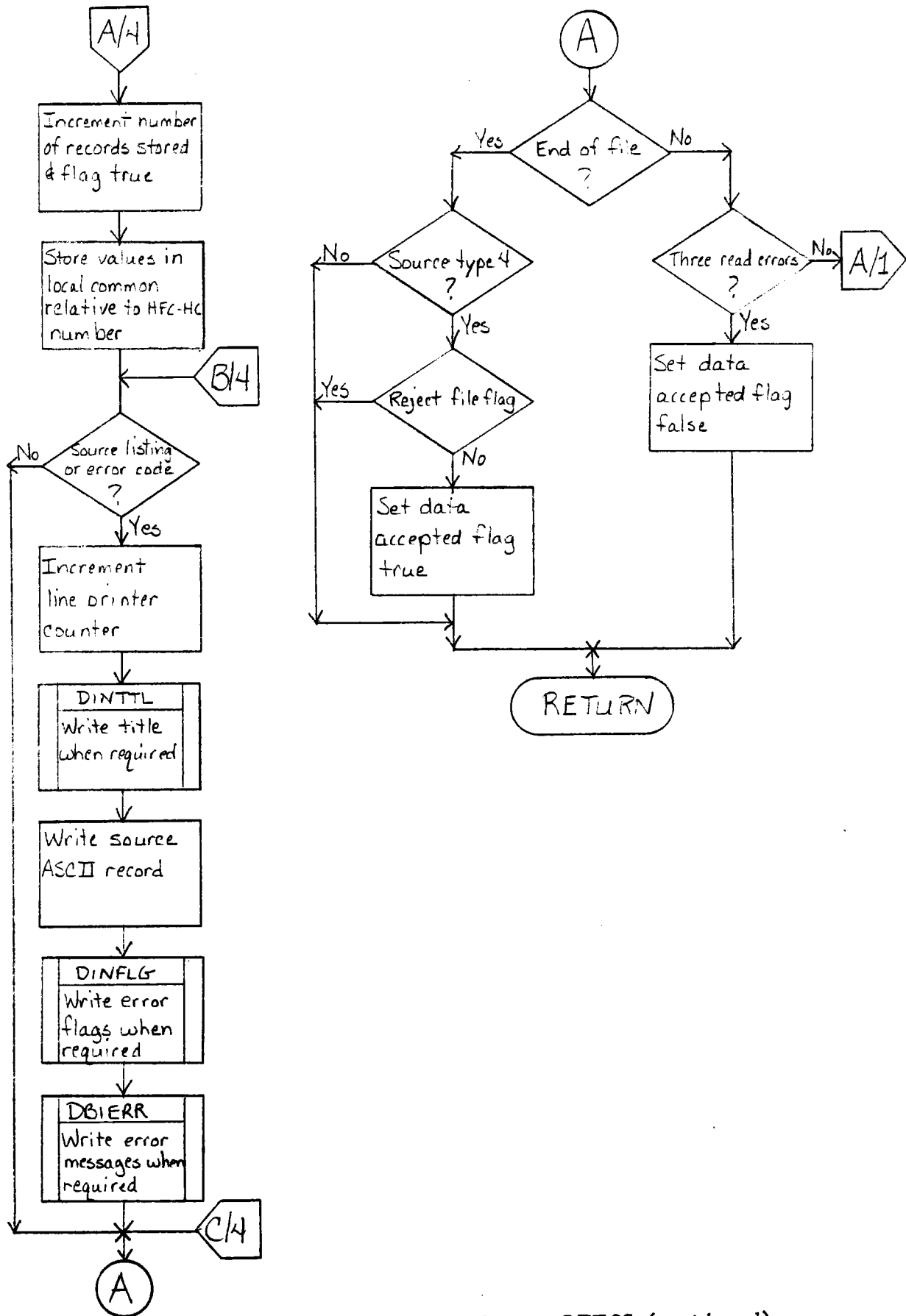


Figure 3.2.7-10 Flowchart - DINLOC (continued)

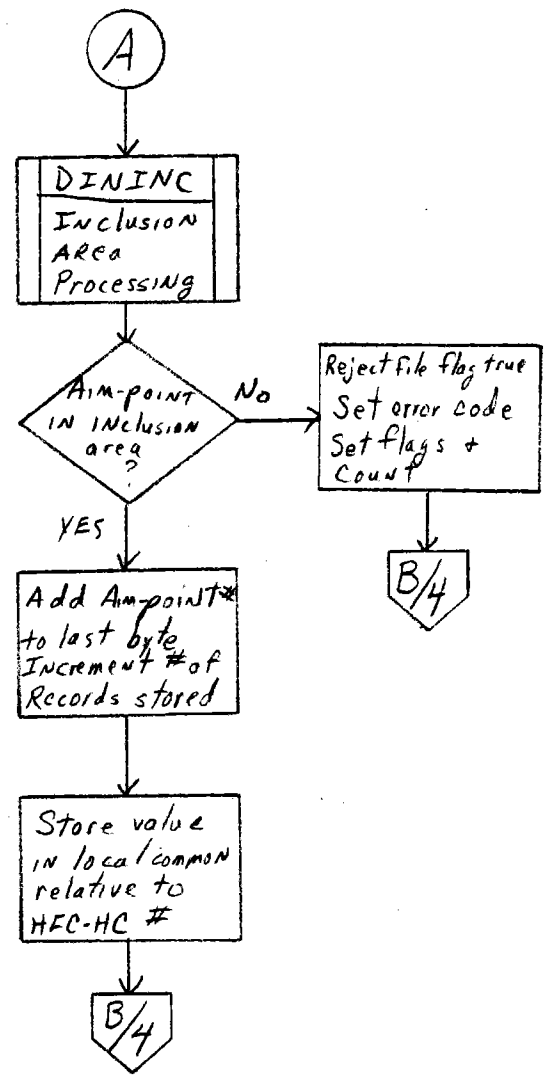
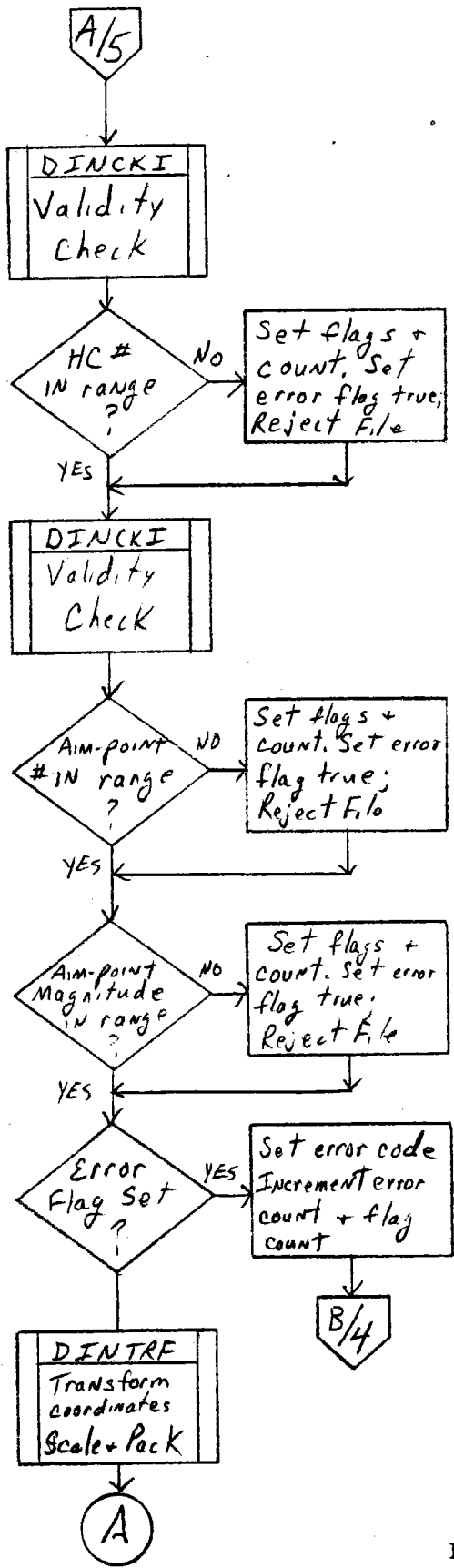


Figure 3.2.7-10 Flowchart - DINLOC (continued)

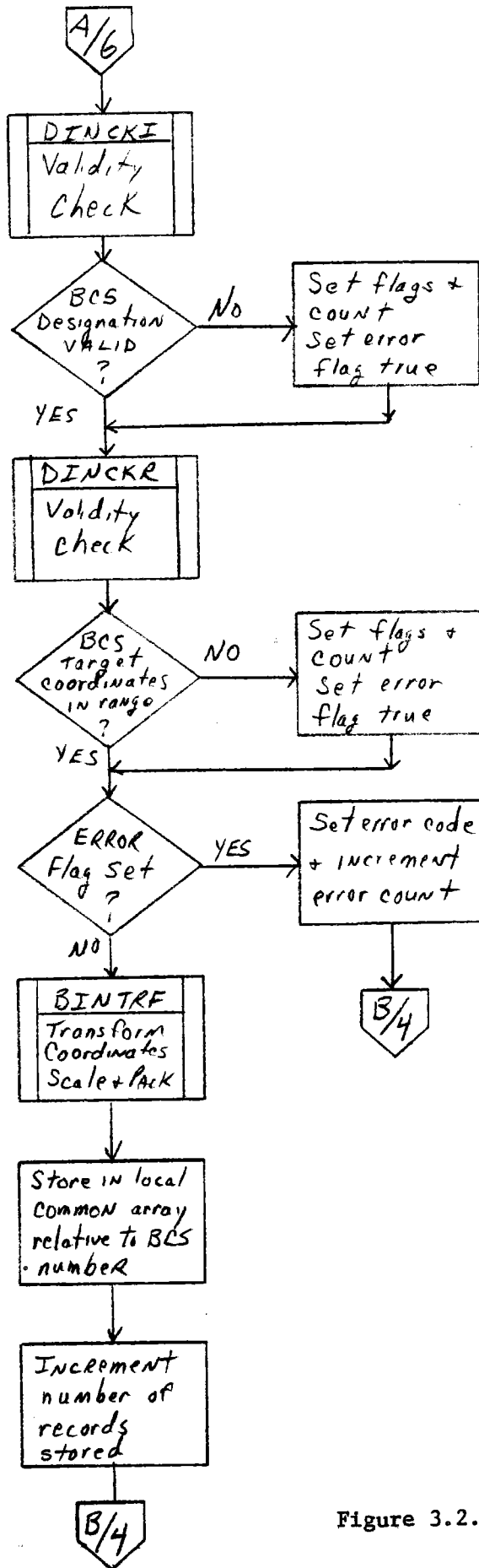


Figure 3.2.7-10 Flowchart - DINLOC (continued)

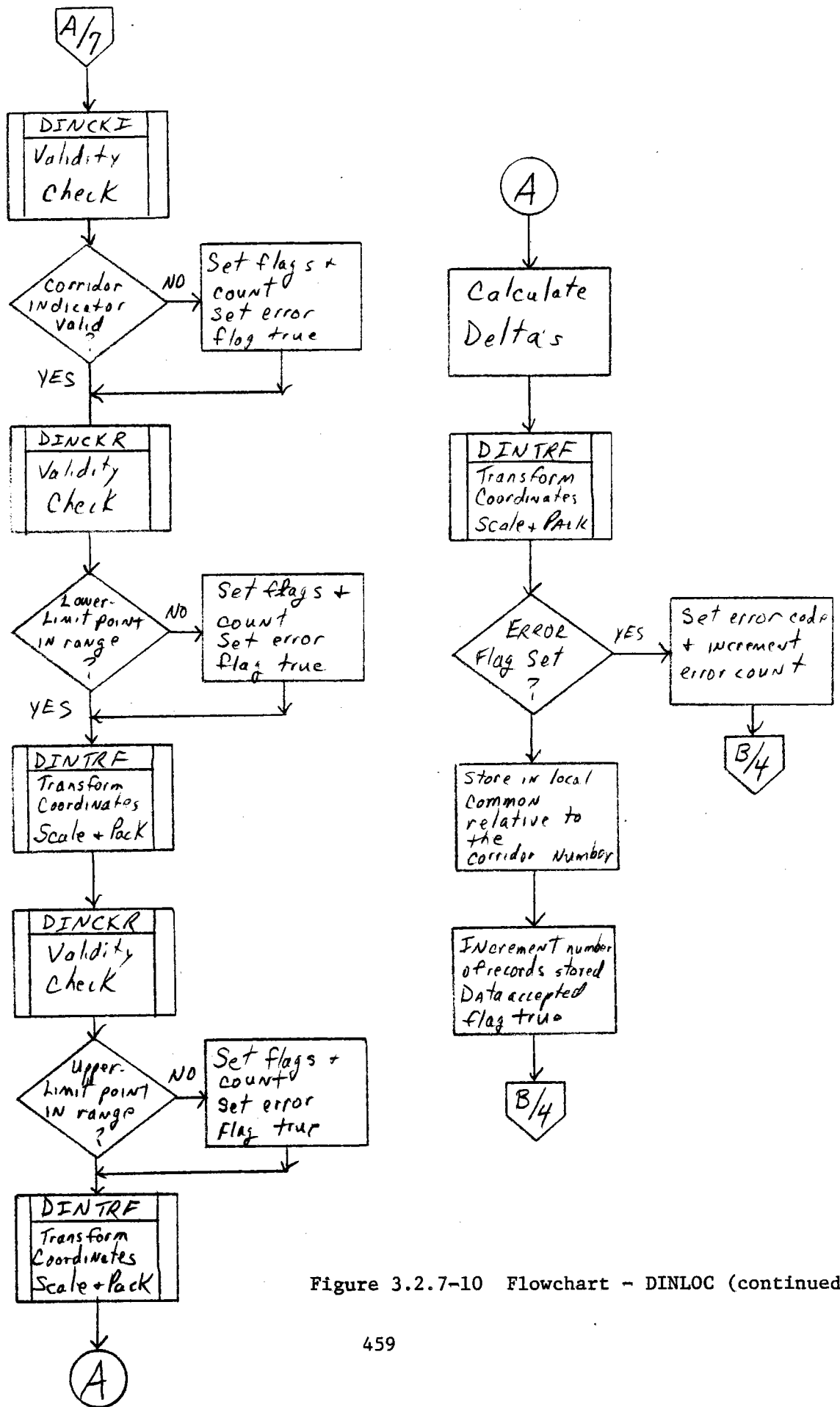


Figure 3.2.7-10 Flowchart - DINLOC (continued)

3.2.7.4.1.3 Submodule III - DINTRF

3.2.7.4.1.3.1 Description

- a. Language used - FORTRAN
- b. How invoked - Called by submodule DINLOC.
- c. Constraints and limitations - Coordinate magnitudes input must be less than 3,375 feet. This subroutine requires 32-bit integer words and a bit-shifting function.
- d. Processing -
 1. The processing begins by transforming the input coordinates, expressed in California Lambert grid, to coordinates expressed in site reference (East, North, Up). During the transformation, the coordinate value units are converted from feet to meters. Next the transformed coordinate values are scaled and packed into a 24-bit format.
 2. Each of the three values is scaled B14 and, respectively, stored into 32-bit integers.
 3. Each of the three 32-bit integer words is bit-shifted left eight places. This shift places the 24 bits of interest into the left-most 24 bits and the remaining right-most 8 bits are all zero.
 4. The Y-coordinate value is shifted right 24 bits. This results in 24 leading zeroes and 8 bits of the Y-value. This value is added to the X-coordinate integer word resulting in 32 bits which contain 24 bits of X value and 8 bits of Y.
 5. The Y-coordinate value is shifted left 8 places. The Z-coordinate is right shifted 16 places. The results of these shifts are added together resulting in a 32-bit word containing 16 bits of Y on the left and 16 bits of Z on the right.
 6. The Z-value is shifted left 16 bits resulting in a 32-bit word with 8 bits of Z and 8 bits of zero in the left half.
- e. Error messages and recovery - This submodule produces no error messages.

3.2.7.4.1.3.2 Data, Logic and Command Paths

Input data:

X, Y, and Z coordinate values in a floating-point array.

Output data:

- a. X, Y, and Z coordinates in Heliostat field reference system (units of meters) in a floating-point array; and
- b. X, Y, and Z coordinates in heliostat field reference system scaled B14 and formatted into a 16-bit integer word array.

Algorithms:

To transform from the California Lambert system (L) to the Heliostat field system (H):

$$X_H = X_L \sin a + Y_L \cos a$$

$$Y_H = X_L \cos a - Y_L \sin a$$

$$Z_H = Z_L - Z_B$$

Where: Z_B = offset constant (98) ft.

$$a = 0.6650407436$$

3.2.7.4.1.3.3 Internal Data Description

Equivalence an I*4 integer array of length three to an I*2 integer array of length five to obtain the 24-bit scaled data words in five I*2 integer words.

3.2.7.4.1.3.4 Flowchart

See Figure 3.2.7-11 for the DINTRF flowchart.

3.2.7.4.1.4 Submodule IV - DININC

3.2.7.4.1.4.1 Description

- a. Language used - FORTRAN
- b. How invoked - Subroutine call by DINLOC
- c. Constraints and limitations - The input aim-point vector must be expressed in the heliostat site reference system (East, North, Up). The upper and lower bounds are internal data, and this subroutine must be changed to incorporate new values.

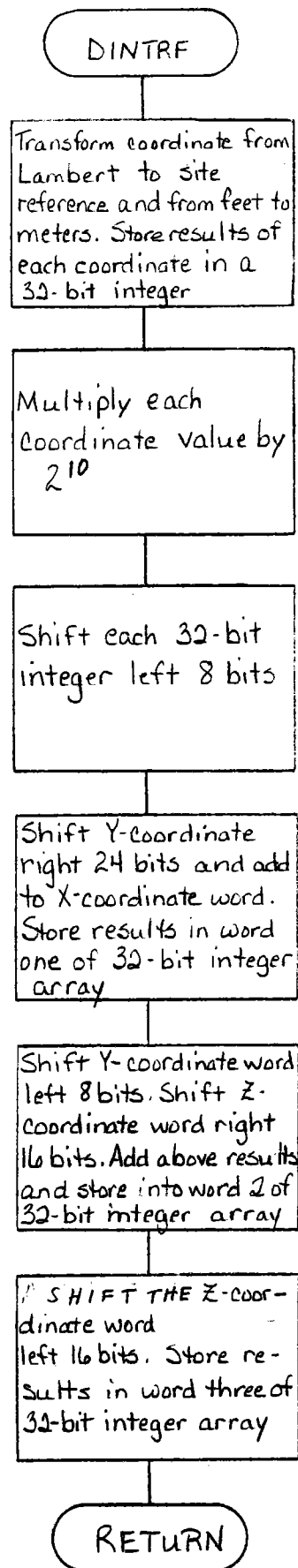


Figure 3.2.7-11 Flowchart - DINTRF

d. Processing -

1. Make sure that the X and Y values are on the receiver center line.
2. Check to see if the Z value is between the upper and lower limit for that row.
3. If either of the above two checks fails, set the error indicator and return.

e. Error messages and recovery - None.

3.2.7.4.1.4.2 Data, Logic, and Command Paths

Input data:

Aim-point vector

Output data:

Indicator flag for aim point being in inclusion area or not.

3.2.7.4.1.4.3 Internal Data Description

See Table 3.2.7-VIII for upper and lower limits.

3.2.7.4.1.4.4 Flowchart

See Figure 3.2.7-12 for the DININC flowchart.

Z COORDINATE (METERS)

<u>ROW</u>	<u>LOWER LIMIT</u>	<u>UPPER LIMIT</u>
1	78.233	88.480
2	77.114	88.008
3	76.758	87.648
4	76.480	87.322
5	76.258	87.066
6	76.075	86.853
7	75.935	86.688
8	75.816	86.551
9	75.694	86.408
10	75.584	86.286
11	75.493	86.179
12	75.411	86.085
13	75.344	86.002
14	75.273	85.920
15	75.197	85.882
16	75.133	85.755
17	75.078	85.688
18	75.027	85.631
19	74.978	85.576
20	74.918	85.506
21	74.865	85.445
22	74.822	85.396
23	74.783	85.350
24	74.746	85.307
25	74.716	85.268
26	74.679	85.231
27	74.646	85.189
28	74.615	85.152
29	74.591	85.118

Table 3.2.7-VIII Aim-Point Limits

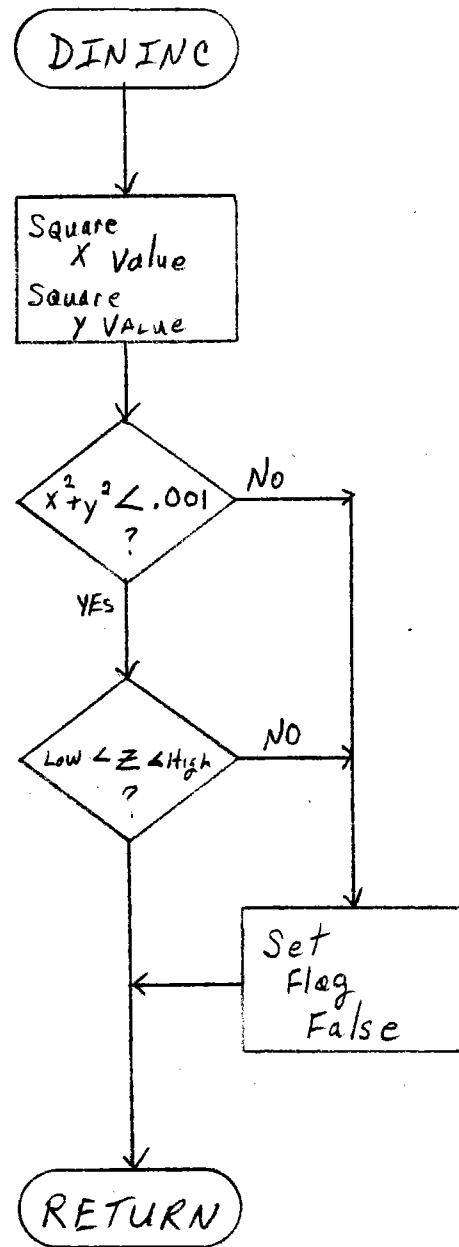


Figure 3.2.7-12 Flowchart - DINING

3.2.7.4.1.5

Submodule V - DINSRT

3.2.7.4.1.5.1

Description

- a. Language used - FORTRAN
- b. How invoked - Called by DINLOC
- c. Constraints and limitations - The pointer array input must be preestablished.
- d. Processing - Shell sorting algorithm:
 1. Store the number of elements to sort into a local variable.
 2. Integer divide the present local array length variable by two.
 3. Test if half of the present local array length is less than or equal to zero. If it is, the sort is complete; otherwise continue on to step (4).
 4. Store the local array length into index "j."
 5. Store the difference of index "j" and the local array length into index "l."
 6. Store the subject array pointer indexed by "j + 1" into temporary location "k."
 7. Test if the subject array value (pointed to by temporary pointer index "k") is greater than or equal to the subject array value (pointed to by the pointer array indexed by "j + 1"). If the test is true, go to step (10), if false, go to step (8).
 8. Store the pointer indexed by "i + 1" into the pointer location indexed by "i - h - 1," and compute a new index "i."
 9. Test if the new index "i" is greater than or equal to zero. If the test is true, go back to step (7). If the test is false, go to step (10).
 10. Store the temporarily stored pointer "k" into pointer location indexed by "i + h + 1."
 11. Test if the index "j" should be incremented or it has reached the end of the array. If "j" is not greater than or equal to the length of the array, increment "j" and go to step (5). If the test is true, go to step (2) to compute a new local array length.

- e. Error messages and recovery - None, but if a zero or negative number is input as the length of the array for sorting, this submodule will return to the calling submodule without attempting any sort.

3.2.7.4.1.5.2 Data, Logic and Command Paths

Input data:

- a. The subject array of integer values to sort into ascending order. The subject array is not disturbed;
- b. The array of pointers that point to the subject array in the order to consider for sorting. This array is rearranged during processing; and
- c. The number of elements in the subject array to sort. This input is not disturbed during processing.

Output data:

The array of pointers sorted so that they point to the subject array in ascending order.

Algorithm:

The algorithm used in DINSRT is the Shell Sort.

3.2.7.4.1.5.3 Internal Data Description

All the arrays and temporary storage are 16-bit integers.

3.2.7.4.1.5.4 Flowchart

See Figure 3.2.7-13 for the DINSRT flowchart.

3.2.7.4.1.6 Submodule VI - DINTTL

3.2.7.4.1.6.1 Description

- a. Language used - FORTRAN
- b. How invoked - Called by DINLOC and DINANG.
- c. Constraints and limitations - None.
- d. Processing -
 1. Initialize the write counter to one.
 2. Compute if the line number input requires a title (modulo function).

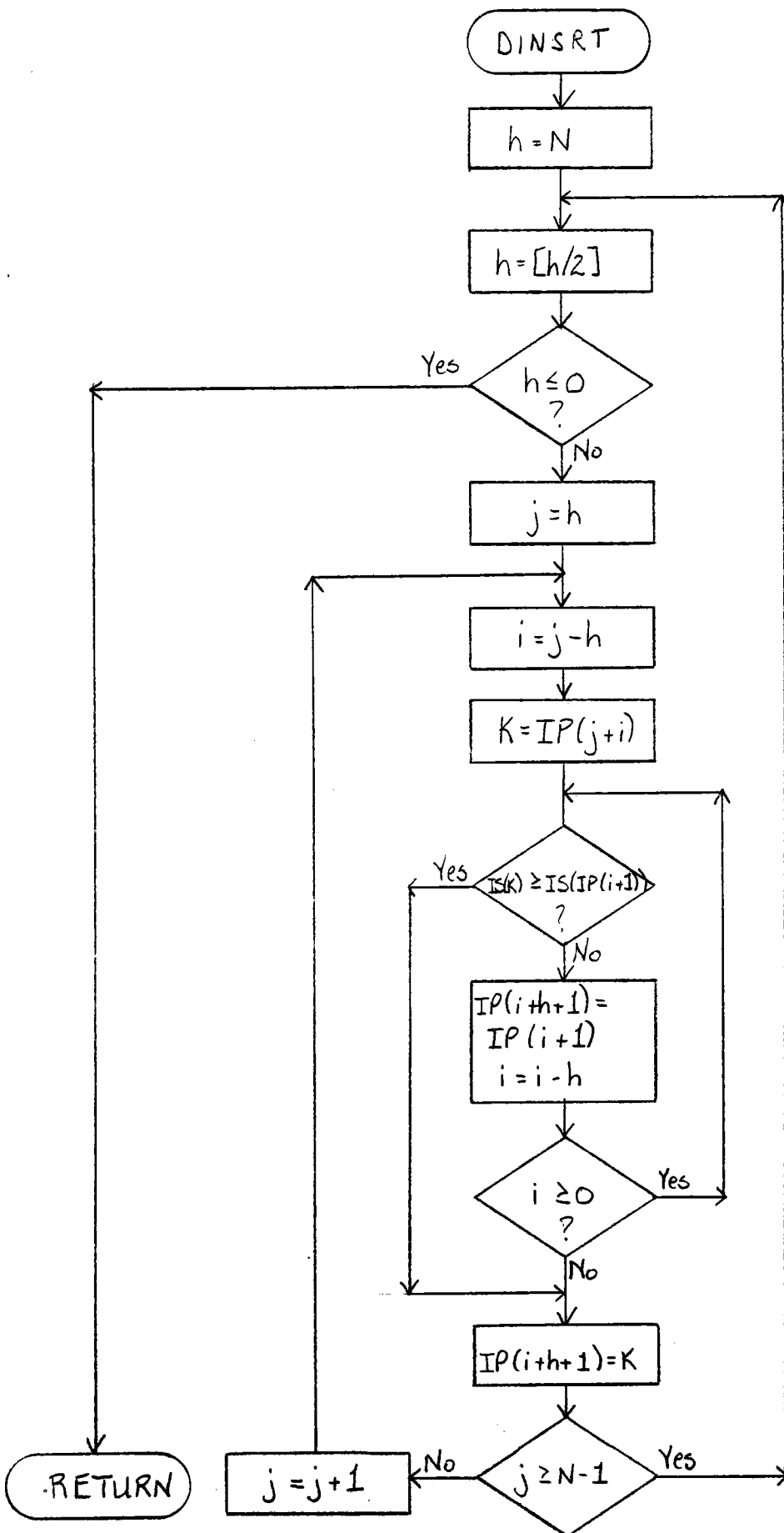


Figure 3.2.7-13 Flowchart - DINSRT

3. Test if writing on line one (from modulo function). If not on line one, return to calling submodule; otherwise enter a CASE construct to write the title.
4. The input title number causes a branch to the proper title for output. In the event an unassigned title number is requested, the top page default path is taken.
5. After the title is printed, the line counter is set back to the number of lines in the title plus one.
6. A test is made to determine if an output error occurred. If no error occurred, a return is made to the calling submodule.
7. If an output error occurred, the write counter is incremented and the operator is notified.

s

e. Error messages and recovery - None

3.2.7.4.1.6.2 Data, Logic and Command Paths

Input data:

- a. The number of the title desired, corresponding to the numbers of the card type source data;
- b. The present line number to write on; and
- c. The device number to write on.

Output data:

- a. If a title is printed, the line number counter is reinitialized for a new page. Otherwise, no changes; and
- b. Title for hard-copy device when output is going to line one.

3.2.7.4.1.6.3 Internal Data Description

The title messages are stored in format statements.

3.2.7.4.1.6.4 Flowchart

See Figure 3.2.7-14 for the DINTTL flowchart.

3.2.7.4.1.7 Submodule VII - DINANG

3.2.7.4.1.7.1 Description

- a. Language used - FORTRAN
- b. How invoked - Subroutine call from DINAEA
- c. Constraints and limitations - None
- d. Processing
 1. Initialize three counters which are routine dependent and the end-of-file false.
 2. Initialize the error code to zero, the number of errors to zero, and the number of syntax error flags to zero. These three variables are record dependent.
 3. Increment the number of records read counter.
 4. Read one record of the source data into a buffer formatted in ASCII characters.
 5. Test if a read error occurred. If the test is true, set the error code and increment the read error counter. Next set the number of error messages counter to one and transfer to step (26). If the test is false, go to step (6).
 6. Test if an end-of-file was read. If the test is true, set the end-of-file flag true and decrement the number of records read counter. Then transfer to step (26). If the test is false, go to step (7).
 7. Initialize the syntax checking column pointer.
 8. Syntax check contents of column presently pointed to.
 9. Test if there is a syntax error; if true, proceed to step (10). Otherwise transfer to step (13).

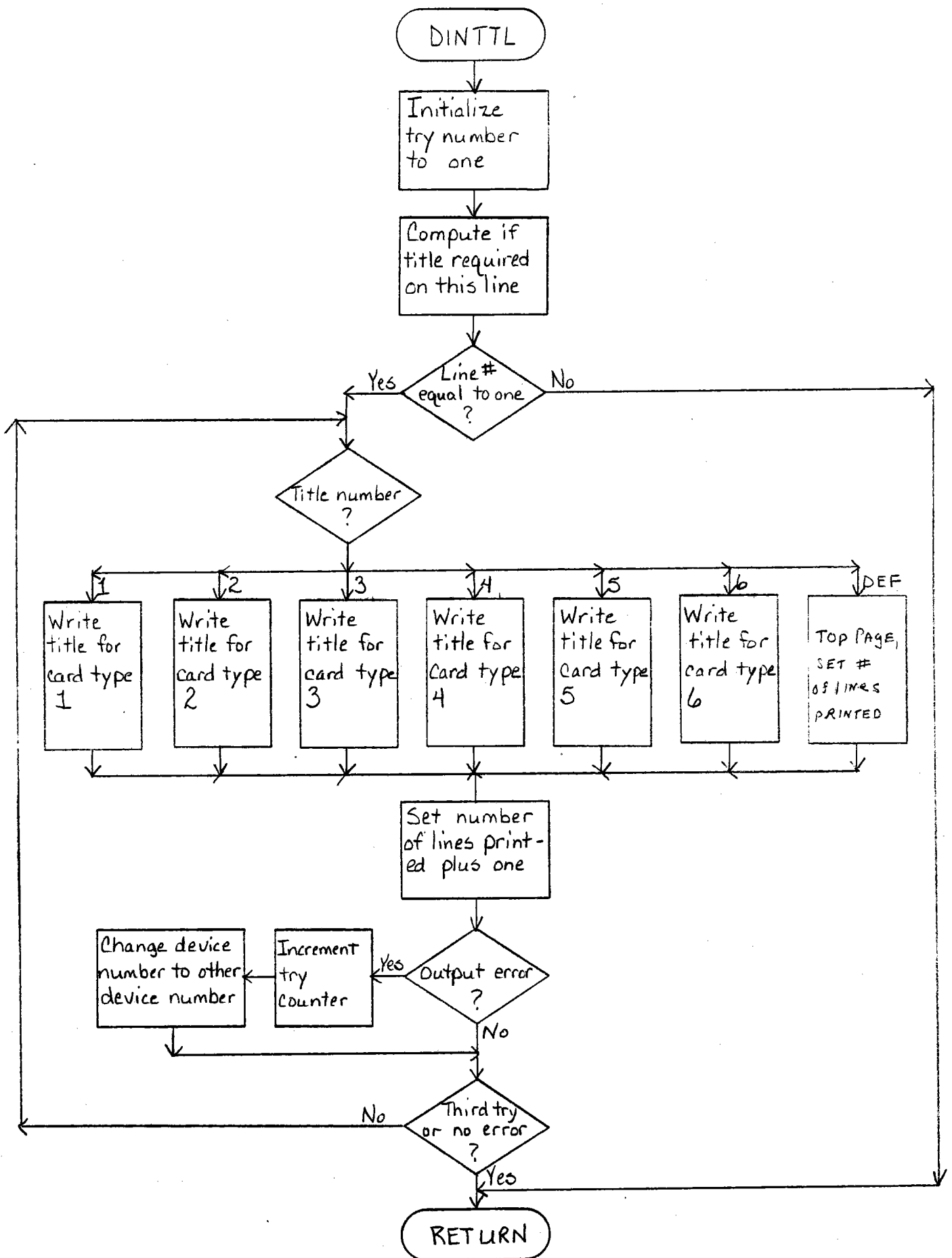


Figure 3.2.7-14 Flowchart - DINTTL
471

10. Set the error code indicating a syntax error, increment the number-of-errors counter, set the column indicator flags, and increment the column flag counter.
11. Test if a default value should be supplied. If a default value is needed, place the default in the columns (ASCII representation). Otherwise, go to step (12).
12. Set the pointers to the next column grouping and transfer to step (14).
13. Increment the column pointer.
14. Test if all the columns needing syntax checking are completed. If the test is true, go to step (15). Otherwise, transfer back to step (8).
15. Test if the card type and HC number are acceptable. If the test is true then go to step (16); otherwise go to step (26).
16. Decode the card type number and HC number.
17. Test if a DECODE error occurred. If the test is true, set the reject-record error code, increment the error counter and go to step (26). Otherwise, go to step (18).
18. Decode columns 16 through 32.
19. Test if a DECODE error or the resulting values are out of range. If the test is true, supply a default value(s), set the error code, increment the error counter, set the column indicator flags, and increment the column flag counter. Otherwise take the null false path.
20. Call the subroutine DINPAC to scale and pack the azimuth and elevation angles into a format compatible with the HFC and HC firmware.
21. Store the HC number and azimuth-elevation packed and scaled values into local common relative to the HC number.
22. Decode columns 39 through 55.
23. Test if a DECODE error occurred or the resulting values are out of range. If the test is true, supply a default value(s), set the error

code, increment the error counter, set the column indicator flags, and increment the column flag counter. Otherwise, take the null false path.

24. Call the subroutine DINPAC to scale and pack the azimuth and elevation angles into a format compatible with the HFC-HC firmware.
 25. Store the azimuth and elevation values into local common relative to the HC number.
 26. Test if a source listing was requested or any error codes exist. If the test is true, go to step (27); otherwise transfer to step (32).
 27. Increment the printer line-number counter.
 28. Call the DINTTL subroutine to produce a source listing title when required.
 29. Write the source ASCII to the printer.
 30. Call the subroutine DINFLG to print the error flags.
 31. Call the subroutine DBIERR to print the error messages.
 32. Test if the end-of-file flag is set or three read errors have occurred. If the test is true, return to the calling subroutine; otherwise transfer back to step (2).
- e. Error messages and recovery - Set error indicators and call DIBERR.

3.2.7.4.1.7.2 Data, Logic and Command Paths

Input data:

- a. Card type indicator;
- b. Source device number;
- c. Source listing desired indicator; and
- d. Output device number.

Output data:

- a. End-of-file encountered flag; and
- b. Local common is used to store the azimuth and elevation values for the HC number. These values are stored relative to the HC number.

3.2.7.4.1.7.3 Internal Data Description

An internal array of the ASCII characters "+," "-", "blank," "0," "1," "2," "3," ..., "9," and "." are used to syntax check the source record. Default values for the azimuth and elevation are stored both in ASCII and floating-point format.

The local common area is structured so that the first I*2 integer array corresponds to the source HC number. The remaining four I*2 integer arrays correspond respectively, to the azimuth, elevation, azimuth, and elevation.

3.2.7.4.1.7.4 Flowchart

See Figure 3.2.7-15 for the DINANG flowchart.

3.2.7.4.1.8 Submodule VIII - DINPAC

3.2.7.4.1.8.1 Description

- a. Language used - FORTRAN
- b. How invoked - Called by DINANG submodule
- c. Constraints and limitations - the input azimuth and elevation angles are limited in magnitude to less than 360 degrees.
- d. Processing -
 1. The input azimuth angle is divided by 360, then multiplied by the scaling factor 2^{13} . The result of this computation is stored in a 16-bit integer word.
 2. The input elevation angle is divided by 360, then multiplied by the scaling factor 2^{13} . The result of this computation is stored in a 16-bit integer word.

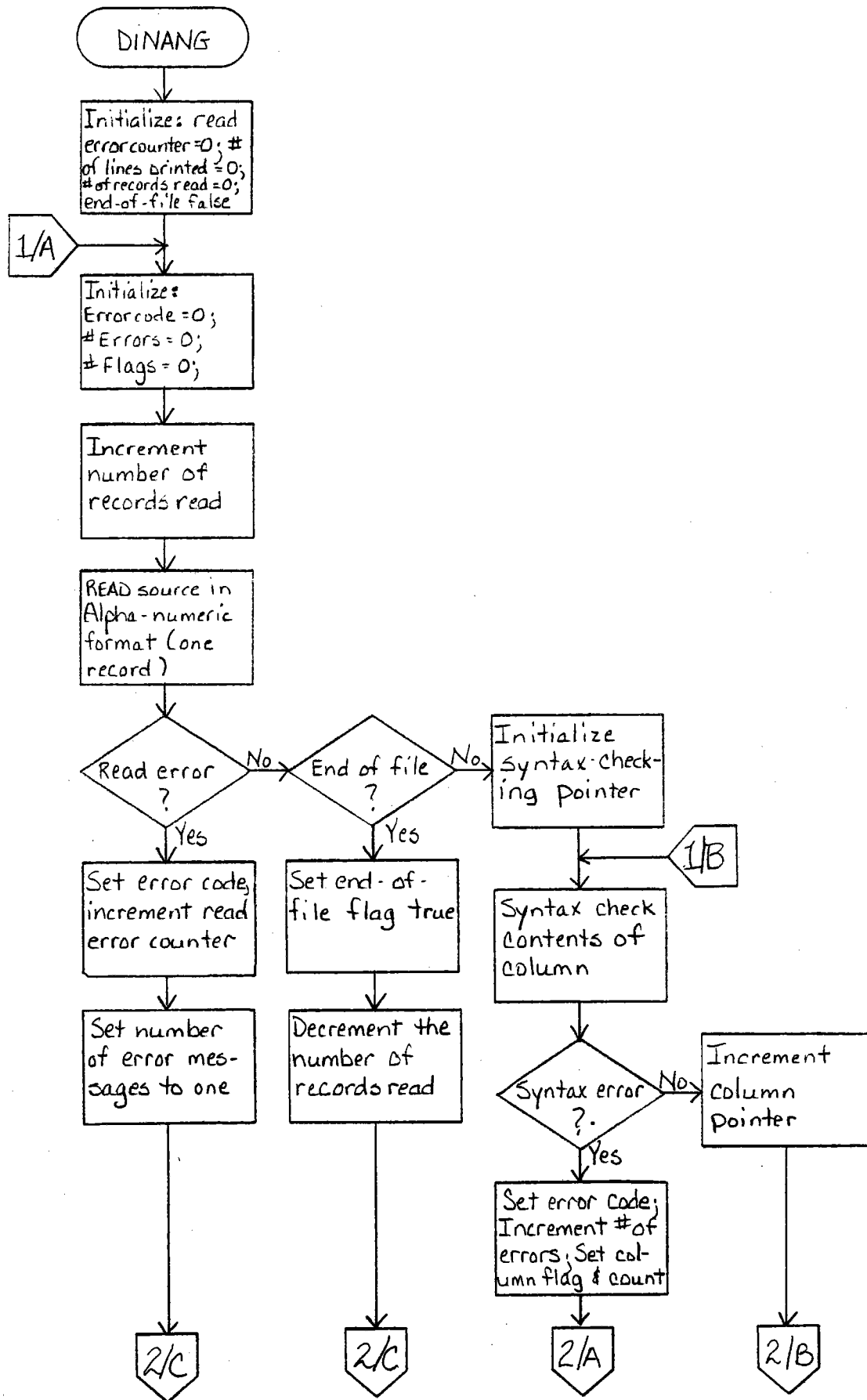


Figure 3.2.7-15 Flowchart - DINANG

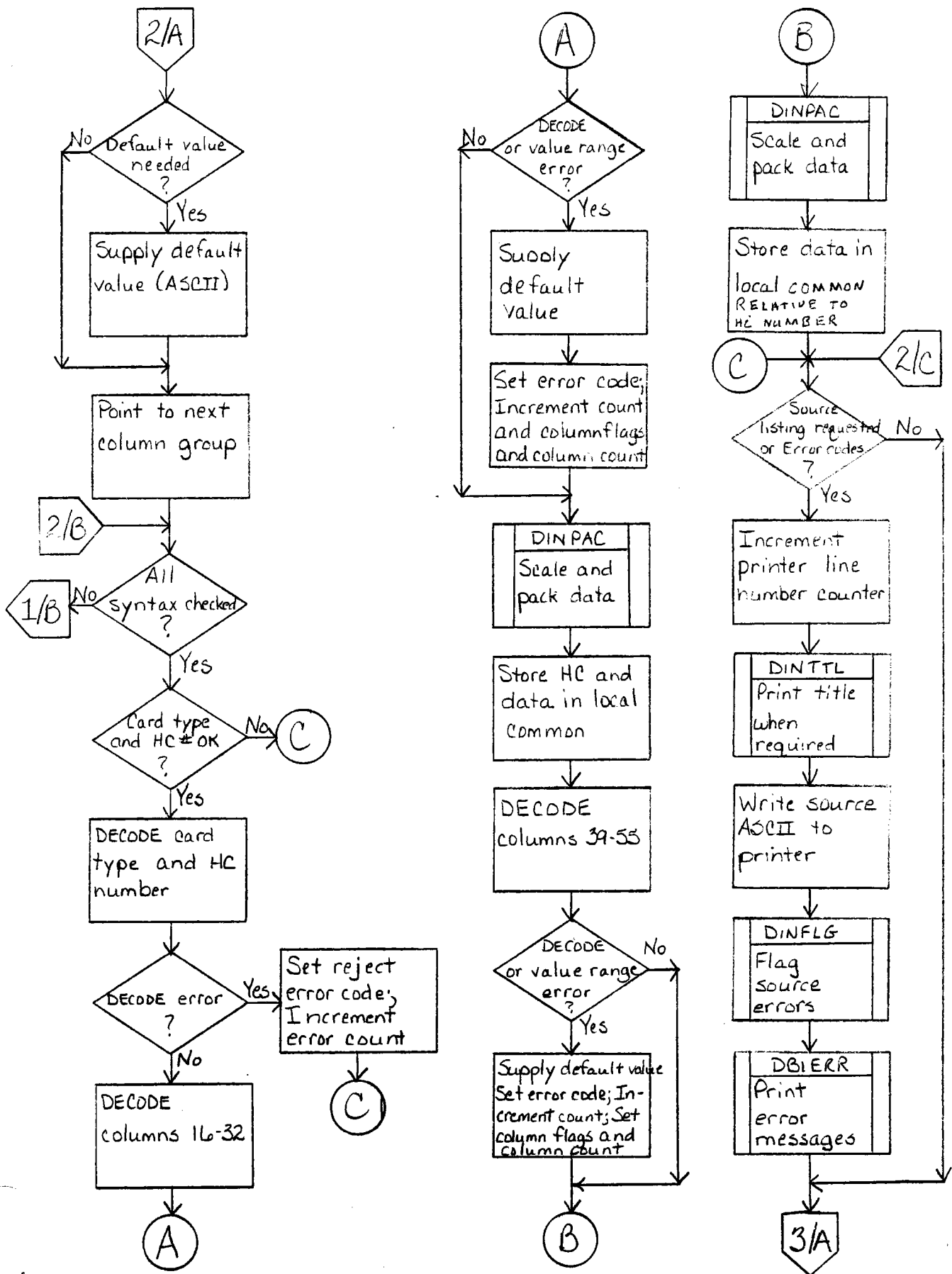


Figure 3.2.7-15 Flowchart - DINANG (continued)

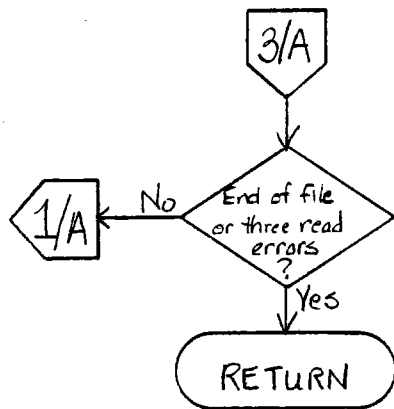


Figure 3.2.7-15 Flowchart - DINANG (continued)

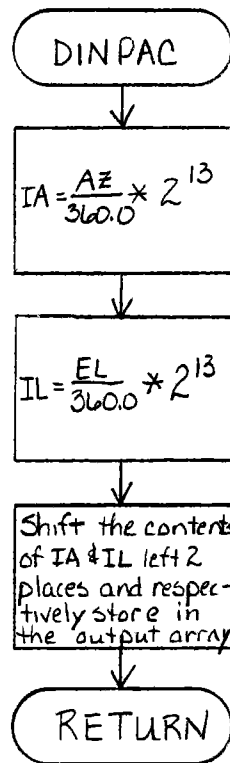


Figure 3.2.7-16 Flowchart - DINPAC

c. Constraints and limitations - None

d. Processing -

1. If there are no error flags to print, this routine returns to the calling submodule. Otherwise the processing continues.
2. A buffer for output to the printer is filled with either an ASCII blank or "-" depending on the state of an input indicator array.
3. After the output buffer is filled, the line number counter is incremented.

Next the ASCII message
buffer is written to the output device
requested.

4. Test if an output error occurred. If there is not an error, return to the calling submodule. Otherwise, set the output device number to the other device. Call the titles routine, write the ASCII message, and return.

e. Error messages and recovery - Error--cannot
access hard-copy device.

3.2.7.4.1.9.2 Data, Logic and Command Paths

Input data:

- a. An array of error flag indicators where zero means no flag and non-zero means flag it;
- b. The requested output device number;
- c. The number of error flags, where zero means no flags required and non-zero means get flag indicators from array;
- d. The line number of the output device; and
- e. The title number.

Output data:

- a. The input array of error flag indicators is cleared (set to zero) during processing; and
- b. The line number counter is incremented by one.

3.2.7.4.1.9.3 Internal Data Description

ASCII characters blank and "-" are used to fill the message output buffer. The input array of error flag indicators is initialized to zero with a data statement.

3.2.7.4.1.9.4 Flowchart

See Figure 3.2.7-17 for the DINFLG flowchart.

3.2.7.4.1.10 Submodule X - DINLAC

3.2.7.4.1.10.1 Description

- a. Language used - Fortran
- b. How invoked - Called by task DIN.
- c. Constraints and limitations - None
- d. Processing -
 1. Initialize buffers and pointers.
 2. Determine full or partial update.
 3. Call DINLOC for syntax and range checks.
 4. If card type 1, sort data and write to the HCC disk file.
 5. If card type 4, logic check aimpoint numbers and update the new aimpoint numbers on the AIM disk file.
 6. If card type 6, update new corridors on the DIN disk file.
 7. If card type 5, update the new BCS target coordinates on the DIN disk file.
- e. Error messages and recovery -
 1. Task DIN has an error in reading the disk.
 2. Source records are rejected.
 3. Error in accessing disk--do not use the data base.
 4. Aimpoint file rejected--not all same AIM number.

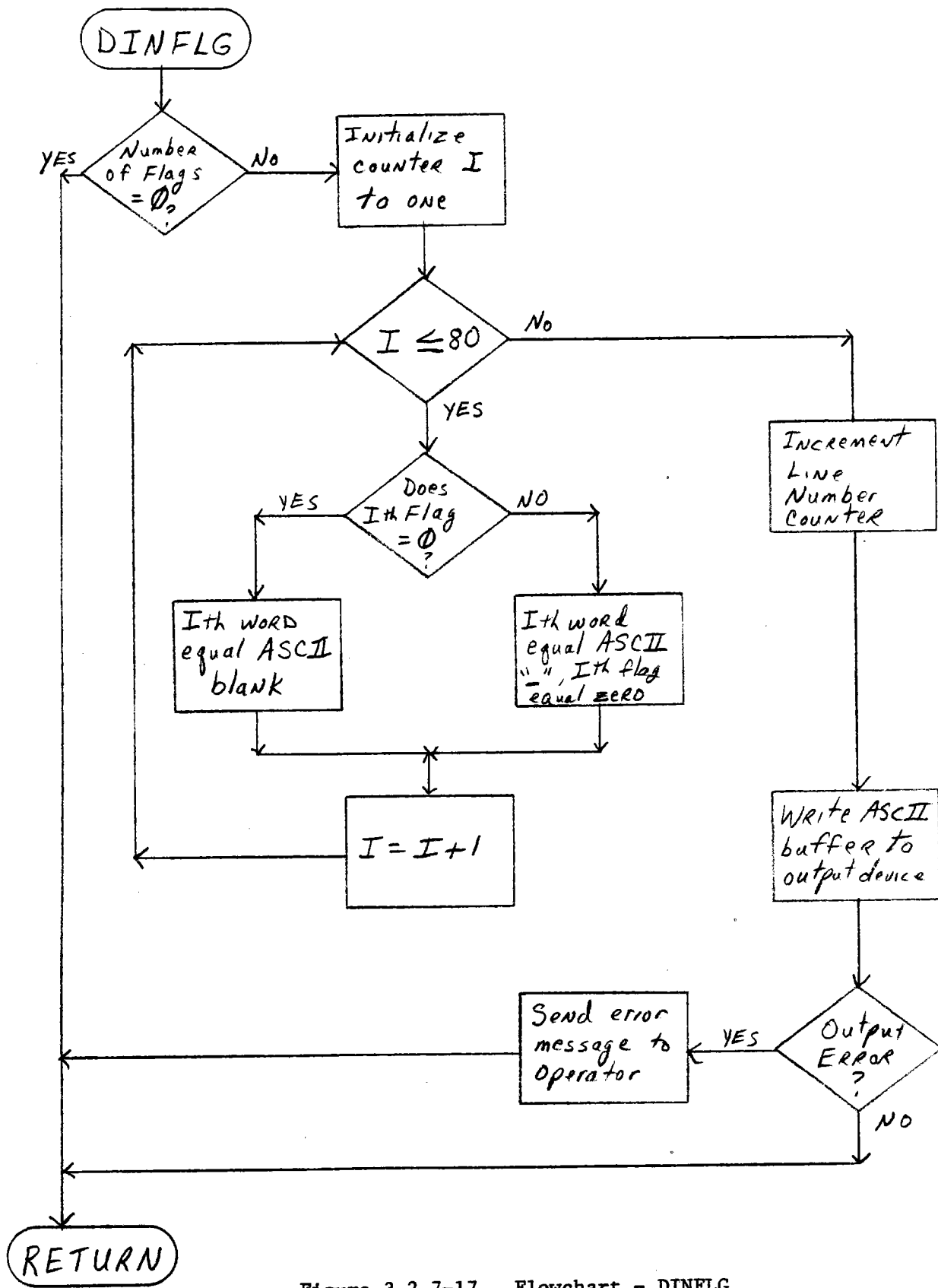


Figure 3.2.7-17 Flowchart - DINFLG

3.2.7.4.1.10.2 Data, logic, and command paths

Input Data:

- a. HCC, DIN, and AIM disk files
- b. Calling parameters from task DIN

Output data:

- a. Updated data to DIN, HCC, and AIM disk files
- b. Error messages to operator

3.2.7.4.1.10.3 Internal Data Description

DINCOM and local internal buffers are used to store and process data before the actual disk update.

3.2.7.4.1.10.4 Flowchart

See Figure 3.2.7-18 for DINLAC flowchart.

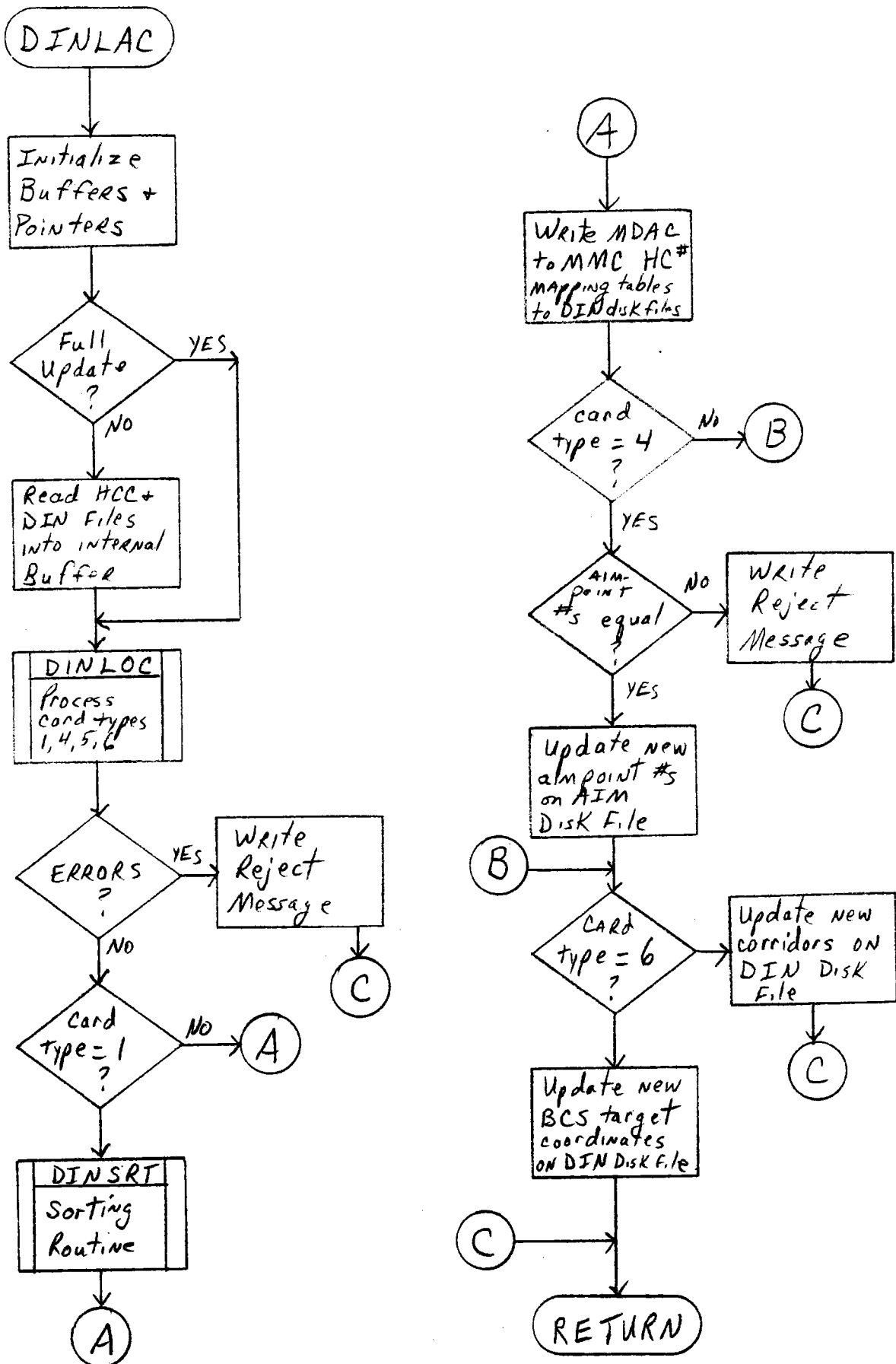


Figure 3.2.7-18 Flowchart - DINLAC

3.2.7.4.1.11 Submodule XI - DINCKI

3.2.7.4.1.11.1 Description

- a. Language used - FORTRAN
- b. How invoked - Called by DINLOC and DINANG
- c. Constraints and limitations - None
- d. Processing -
 1. Check to see if the integer variable passed as a parameter is between the high and low values also passed as parameters.
 2. If the value is not in range, return an error indicator and column number.
- e. Error messages and recovery -
 1. Set error indicator.

3.2.7.4.1.11.2 Data, Logic and Command Paths

Input Data:

- a. Calling parameters from DINLOC or DINANG:
 - Variable to be checked
 - The high and low values
 - Column numbers where variable resides

Output Data:

- Return parameters to DINLOC or DINANG:
 - Column number where error occurred
 - Number of columns checked
 - Error indicator

3.2.7.4.1.11.3 Internal Data Description - None

3.2.7.4.1.11.4 Flowchart

See Figure 3.2.7-19 for DINCKI Flowchart.

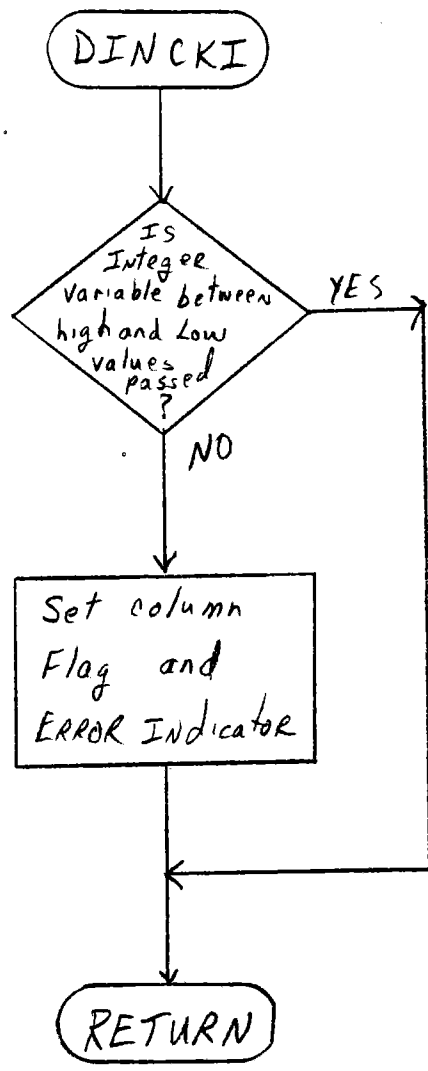


Figure 3.2.7-19 Flowchart - DINCKI

3.2.7.4.1.12 Submodule XII - DINCKR

3.2.7.4.1.12.1 Description

- a. Language used - FORTRAN
- b. How invoked - Called in DINLOC and DINANG
- c. Constraints and limitations - None
- d. Processing -
 1. Check to see if the real variable passed as a parameter is between the high and low values also passed as parameters.
 2. If the value is not in range, return an error indicator and column number.
- e. Error messages and recovery -
 1. Set error indicator

3.2.7.4.1.12.2 Data, Logic, and Command Paths

Input Data:

- a. Calling parameters from DINLOC or DINANG:
 - Variable to be checked
 - The high and low values
 - Column numbers where variable resides

Output Data:

- Return parameters to DINLOC or DINANG:
 - Column number where error occurred
 - Number of columns checked
 - Error indicator

3.2.7.4.1.12.3 Internal Data Description - None

3.2.7.4.1.12.4 Flowchart

See Figure 3.2.7-20 for DINCKR Flowchart.

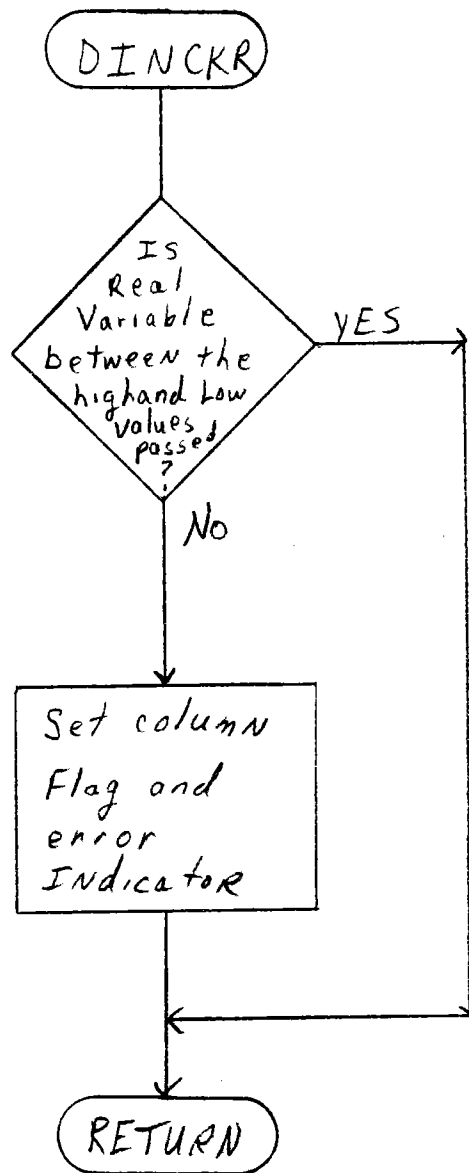


Figure 3.2.7-20 Flowchart - DINCKR

3.2.7.4.1.13 Submodule XIII - DINAEA

3.2.7.4.1.13.1 Description

- a. Language used - FORTRAN
- b. How invoked - Called by task DIN
- c. Constraints and limitations - None
- d. Processing -
 1. Zero buffers and preset variables.
 2. Determine if full or partial update.
 3. If partial update, read the disk data into local buffer for modification.
 4. Call DINANG for syntax and range checks.
 5. If errors have occurred, set the data to preset values and exit.
 6. Update the AL1, AL2, STO, WSH disk files, as appropriate.
- e. Error messages and recovery -
 1. The disk has a read error.
 2. The disk has a write error.
 3. No records from source were accepted.

3.2.7.4.1.13.2 Data, Logic, and Command Paths

Input Data:

1. AL1, AL2, WSH, STO disk files

Output Data:

1. Updated data to AL1, AL2, WSH, STO disk files.
2. Error messages to the operator.

3.2.7.4.1.13.3 Internal Data Description

For a partial update, the appropriate disk files are read into an internal buffer for subsequent modification.

3.2.7.4.1.13.4 Flowchart

See Figure 3.2.7-21 for DINAEA flowchart.

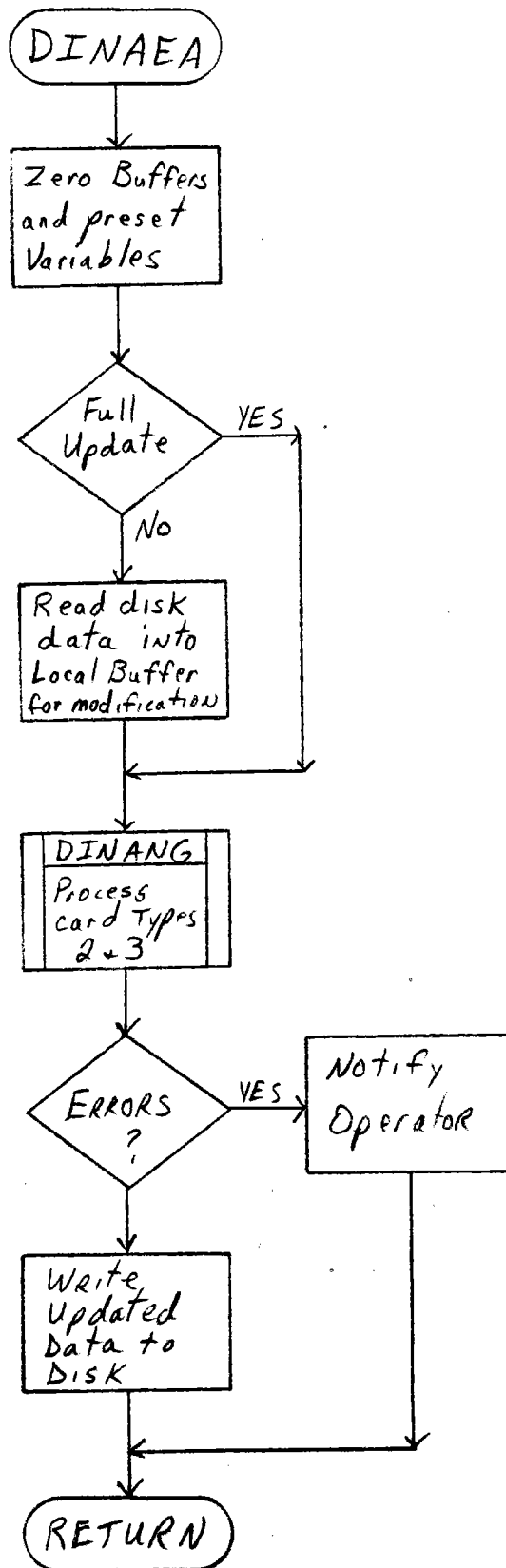


Figure 3.2.7-21 Flowchart - DINA EA

3.2.7.4.1.14 Submodule XIV - DINGRF

3.2.7.4.1.14.1 Description

- a. Language used - FORTRAN
- b. How invoked - Called by task DIN
- c. Constraints and limitations - Execution of this submodule requires the concurrent execution of an initializing program in the chromatix graphics processors.
- d. Processing -
 1. The HC location data is read from the HCC disk file into an internal buffer.
 2. The segment mapping data is read from the DIN disk file into an internal buffer.
 3. The operator inputs which console is to be initialized.
 4. Internal printers are set for sending the data in HC/HFC order.
 5. Call DINUPC to format the output.
 6. Send the data to the graphics processor, and go to Step 5 until all data is sent.
 7. Set up the pointers to send the data in segment order subordered by pecking order.
 8. Call DINUPC to format the output.
 9. Send the data to the graphics processors, and go to Step 8 until all data is sent.
3. Error messages and recovery - Error in accessing disk sent to operator.

3.2.7.4.1.14.2 Data, Logic, and Command Paths

Input Data;

1. HCC and DIN disk files.
2. Responses from the operator.
3. Response from graphics console.

Output Data:

1. Prompts, menus, and error messages to operator.
2. Data packets to the graphics processors.

3.2.7.4.1.14.3 Internal Data Description

The internal data buffer that is used to format the data for the graphics processor is accessed in two different orders; the first by HC/HFC number, and the second by pecking within segment order.

3.2.7.4.1.14.4 Flowchart

See Figure 3.2.7-22 for DINGRF flowchart.

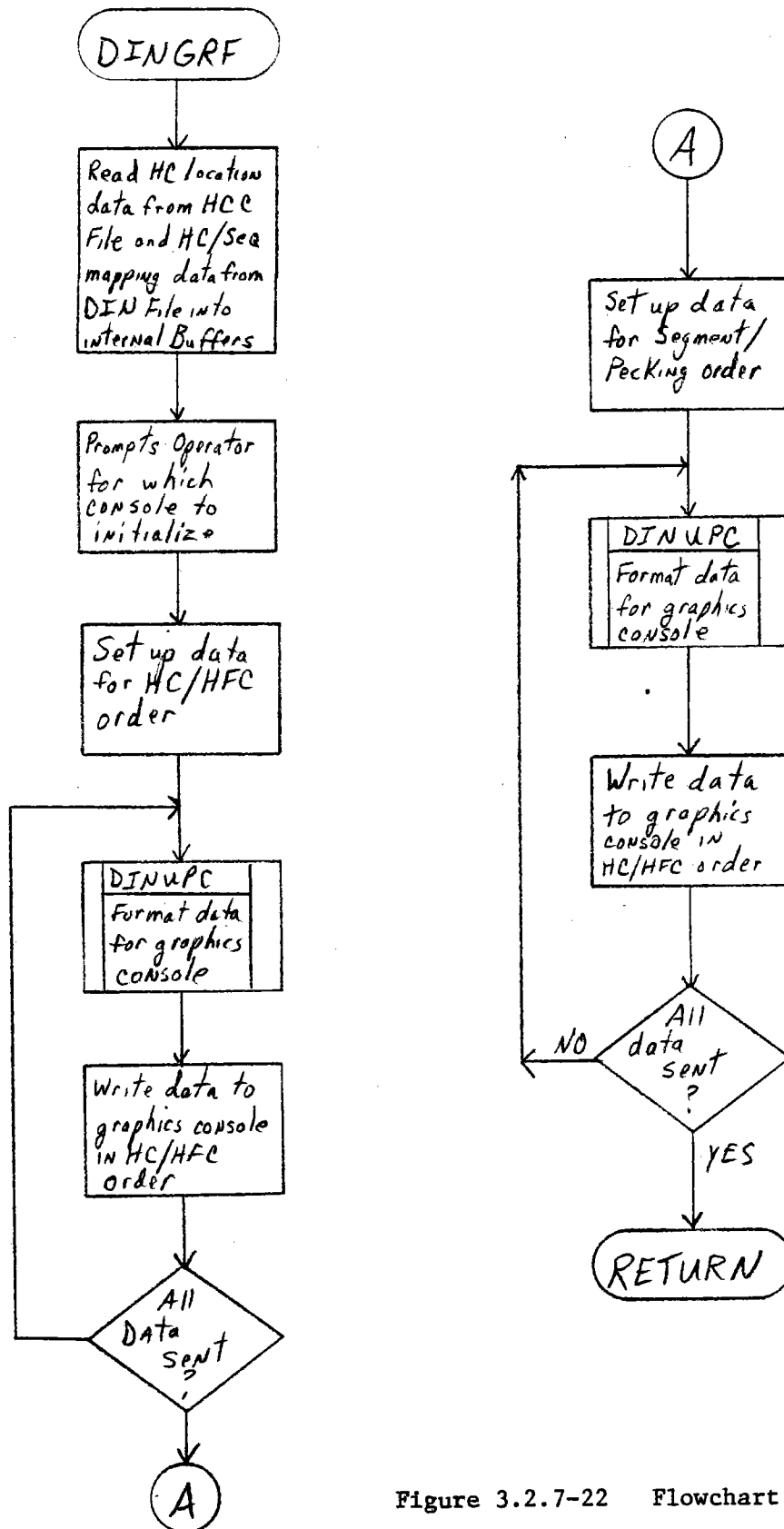


Figure 3.2.7-22 Flowchart - DINGRF

3.2.7.4.1.15 Submodule XV - DINUPC

3.2.7.4.1.15.1 Description

- a. Language used - FORTRAN
- b. How invoked - Called by DINGRF
- c. Constraints and limitations - None
- d. Processing -
 1. Preset internal variables.
 2. Shift and pack data into the message format.
 3. Change internal meters to feet for display.
- e. Error messages and recovery - None.

3.2.7.4.1.15.2 Data, Logic, and Command Paths

Input Data:

1. Parameters passed from DINGRF

Output Data:

1. Packed, converted, and shifted data as parameters

3.2.7.4.1.15.3 Internal Data Description

Constants for packing and shifting the data and a constant to convert meters to feet.

3.2.7.4.1.15.4 Flowchart

See Figure 3.2.7-23 for DINUPC flowchart.

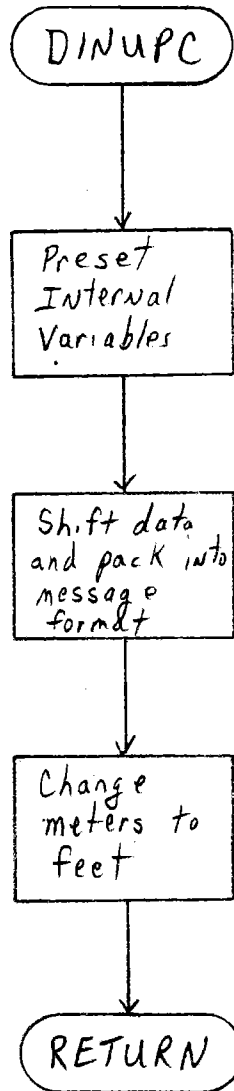


Figure 3.2.7-23 Flowchart - DINUPC

3.2.7.4.1.16 Submodule XVI - DBI (Task)

3.2.7.4.1.16.1 Description

- a. Language used - FORTRAN
- b. How invoked - Prescheduled task activated at system boot time.
- c. Constraints and limitations - The disk data base must be established.
- d. Processing -
 1. Call subroutine DBIFAC to initialize the system interfaces and establish the Prime-Backup HAC configuration.
 2. If execution is in the Prime HAC, go to step (3). Otherwise, establish the SWI task, call Submodule TRP, and go to step (5).
 3. Call subroutine DBIDTA to initialize the in-core global data base.
 4. If the ABORT flag is returned true by DBIDTA, go to step (9). Otherwise, go to step (5).
 5. Establish the CLK (HAC Start-up) task as main-memory resident.
 6. If either of the tasks could not be established, notify the operator, and go to step (9). Otherwise, go to step (7).
 7. Activates task CLK.
 8. If task could not be activated, notify the operator, and go to step (9). Otherwise, go to step (9).
 9. EXIT this task.

- e. Error messages and recovery - If either SWI or CLK can not be established or activated, the operator is notified and the task exits. Also, if the global common data base can not be initialized by subroutine DBIDTA, this task exits.

Error messages are:

1. TASK SWI FAILED TO ESTABLISH.

3.2.7.4.1.16.2 Data, Logic and Command Paths

Three conditions are considered fatal for the DBI task:

- a. Failure to initialize the global common data base;
- b. Failure to establish either SWI or CLK tasks; and
- c. Failure to activate either SWI or CLK tasks.

Input data:

- a. ABORT flag returned by DEIDTA; and
- b. Global common word, CPUSG.

3.2.7.4.1.16.3 Internal Data Description

This task does not use any local data structures.

3.2.7.4.1.16.4 See Figure 3.2.7-24 for the DBI flowchart.

3.2.7.4.1.17 Submodule XVII--DBIFAC

3.2.7.4.1.17.1 Description

- a. Language used - FORTRAN
- b. How invoked - Subroutine call by DBI.
- c. Constraints and limitations - None.
- d. Processing -
 1. Call subroutine DBICPU to determine whether execution is in the Prime or Backup HAC.
 2. Call subroutine DBIERR to output any error messages from DBICPU.

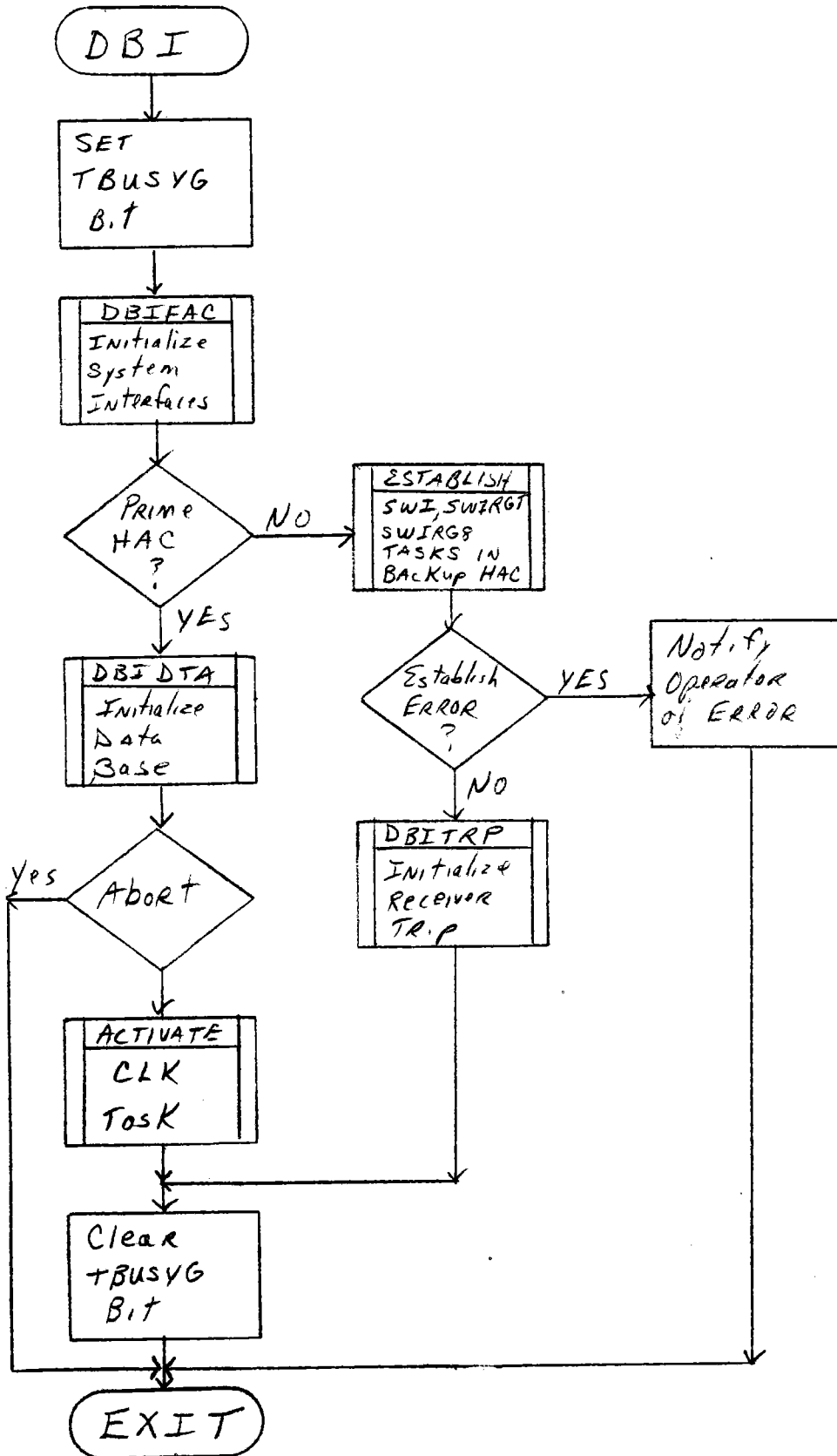


Figure 3.2.7-24 Flowchart - DBI

3. Call subroutine DBICRT if you are in the prime HAC. DBICRT will clear the ISC terminal and output the initial status display.
4. Call subroutine DBIERR to output any error messages from DBICRT.
5. Return processing control to the calling task DBI.

e. Error messages and recovery - None

3.2.7.4.1.17.2 Data, Logic, and Command Paths

Input data:

Prime or Backup indicator in global common word CPUSG.

Output data:

None

3.2.7.4.1.17.3 Internal Data Description

No internal data structures are utilized

3.2.7.4.1.17.4 Flowchart

See Figure 3.2.7-25 for the DBIFAC flowchart.

3.2.7.4.18 Submodule XVIII - DBICPU

3.2.7.4.1.18.1 Description

- a. Language used - FORTRAN
- b. How invoked - Called by subroutine DBIFAC
- c. Constraints and limitations - Require operator response to questions concerning designation of Prime or Backup HAC.
- d. Processing -
 1. Request the operator to designate Prime or Backup HAC.
 2. If Prime HAC designated, do the following steps; else, step (5).
 3. Request the operator if the Backup HAC is present, and set CPUSG word in global common COMIN4 accordingly. Also call Subroutine DBIPCS to gain control of the PCSs.

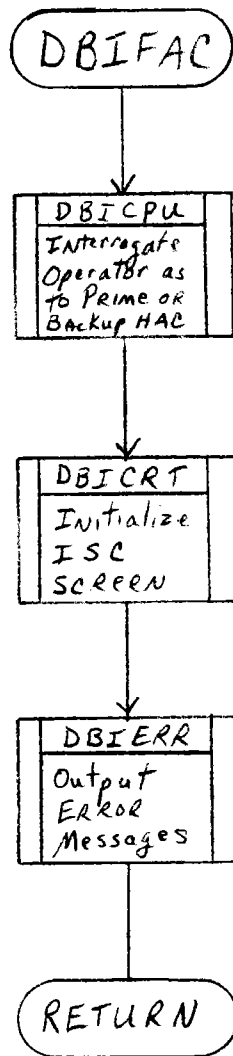


Figure 3.2.7-25 Flowchart - DIBFAC

4. Enable interrupt level seven to allow Prime HAC to be halted by Backup HAC upon failover, and return to calling submodule.

5. Indicate that this HAC is Backup, and return to calling Submodule.

e. Error messages and recovery - If an unexpected response is received from the operator, the question is repeated.

3.2.7.4.1.18.2 Data, Logic, and Command Paths

Input data:

Operator designated Prime or Backup

Output data:

-- Messages to operator

-- Prime/Backup designation in CPUSG

3.2.7.4.1.18.3 Internal Data Description

Miscellaneous constraints

3.2.7.4.1.18.4 Flowchart

See Figure 3.2.7-26 for DBICPU flowchart.

3.2.7.4.1.19 Submodule XIX - DBICRT

3.2.7.4.1.19.1 Description

a. Language used - FORTRAN

b. How invoked - Subroutine call DBIFAC

c. Constraints and limitations - None.

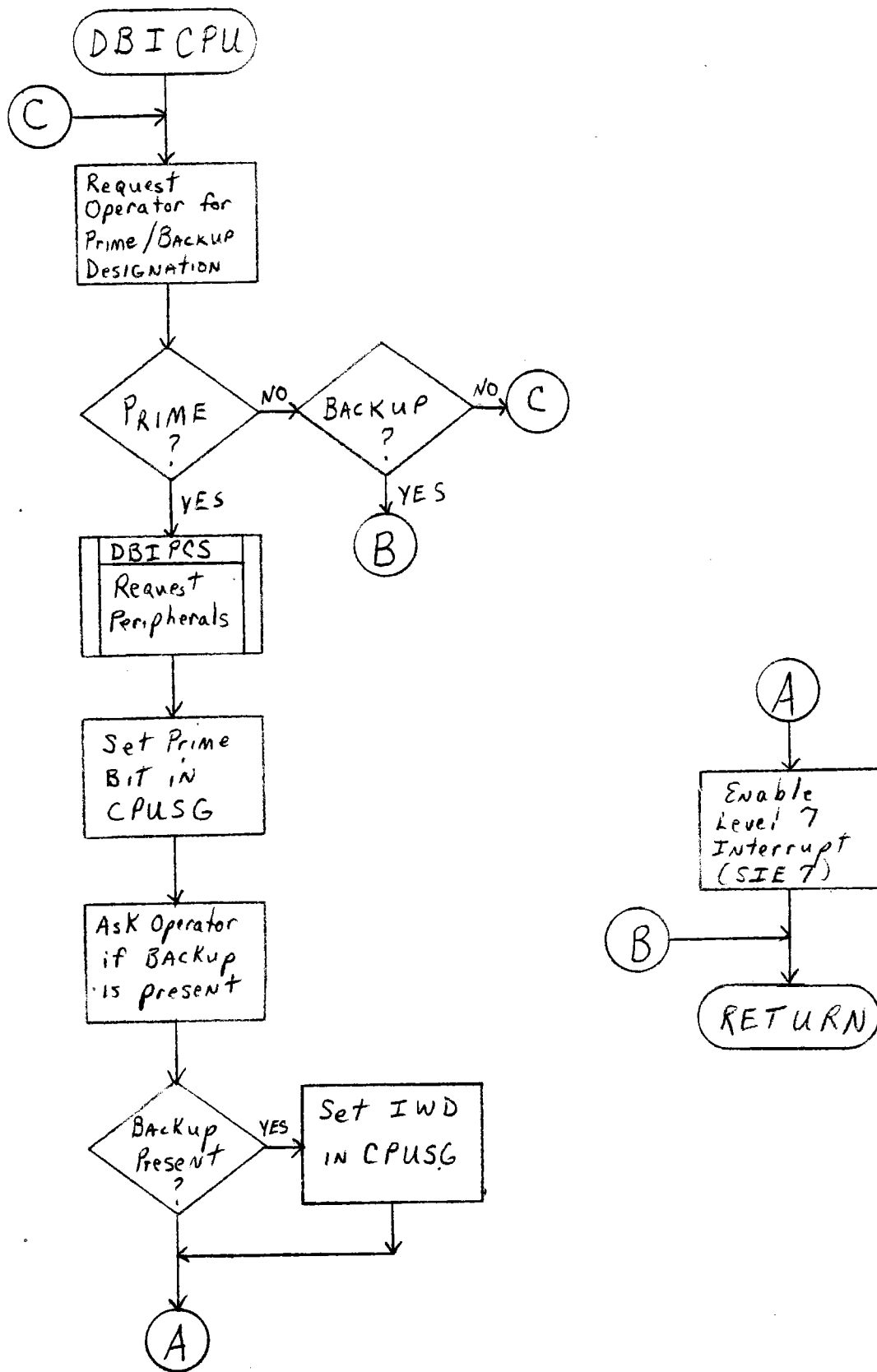


Figure 3.2.7.26 Flowchart - DBICPU

d. Processing -

1. Initialize ISC screen by erasing screen and subdividing into colors and writing initial headers.

e. Error messages and recovery - None

3.2.7.4.1.19.2 Data, Logic, and Command Paths

Input data:

Operator designated Prime or Backup

Output data:

None

3.2.7.4.1.19.3 Internal Data Description

Miscellaneous data to paint the ISC screen.

3.2.7.4.1.19.4 Flowchart

See Figure 3.2.7-27 for the DBICRT flowchart.

3.2.7.4.1.20 Submodule XX - DBIPCS

3.2.7.4.1.20.1 Description

- a. Language used - FORTRAN and MODCOMP ASSEMBLER
- b. How invoked - Subroutine call by DBICPU
- c. Constraints and limitations - Peripheral controller switches (PCSs) must be positioned to the auto mode.
- d. Processing -
 1. Initialize PCS status words.
 2. Change to assembler code, and load registers with I/O control codes for Master clearing 4811 controllers and request the PCSs.
 3. (a) Lockout DI interrupts (level 12).
(b) Disconnect SI interrupts from chromatics 4811's.
(c) Request PCS 8 and 9.
(d) Master clear 4811 controllers.
(e) Input status from PCS 8 and 9.
(f) Store status from PCS 8 and 9.
 4. If PCS had a bad status, then ask operator if he desires to retry the PCS request; if so, go to step (1).
 5. Else, Master clear 1907 controller.
 6. Return.
- e. Error messages and recovery - An error message is output if a PCS had a bad status and did not switch to the requesting HAC. The message is:

PCS ADDRESS (8/9) HAD BAD STATUS XXXX

when XXX is status from PCS. If the 1907 controller has a bad status, an error message:

1907 HAD BUSY STATUS = XXXX

where XXXX is the status received back from the 1907.

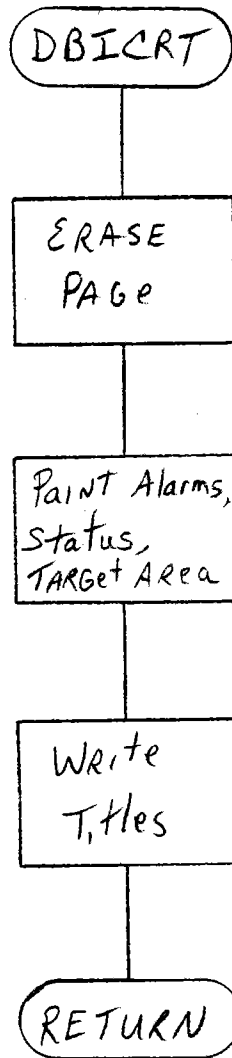


Figure 3.2.7-27 Flowchart - DBICRT

3.2.7.4.1.20.2 Data, Logic, and Command Paths

Input data:

Input status from 4811's and PCSs 8 and 9.

Operator responses to error messages.

Output data:

Error messages to operator.

3.2.7.4.1.20.3 Internal Data Description

Variables to hold status information from controller.

3.2.7.4.1.20.4 Flowchart

See Figure 3.2.7-28 for DBIPCS flowchart.

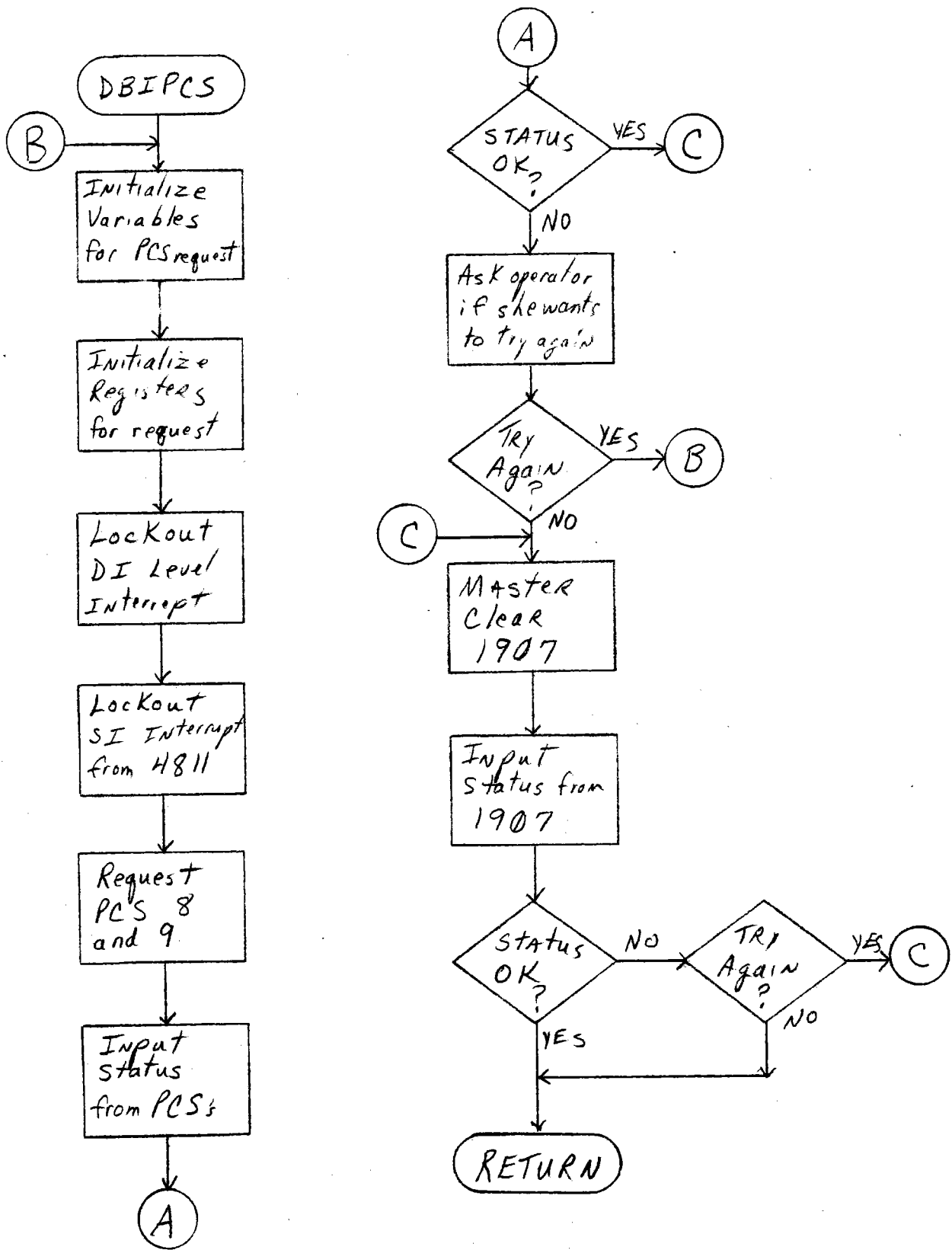


Figure 3.2.7-28 Flowchart - DBIPCS

3.2.7.4.1.21 Submodule XXI - DBITRP (Receiver Trip Initialization)
(For rationale, see Table 3.2.8.III - Receiver Trip Function)

3.2.7.4.1.21.1 Description

- a. Language used - FORTRAN
- b. How invoked - Subroutine call from DBI.
- c. Constraints and limitations - The only error undetected by DBITRP is the case of both interrupt levels high. In this case the two interrupt handlers in Receiver Trip will occupy 100 percent of the CPU time, and nothing more will be accomplished.
- d. Processing - Receiver Trip (R/T) initialization consists of establishing and connecting both receiver trip tasks, RTL and RTH, waiting one second, and then sensing the RTOP bit in CPUSG to see if one or the other of the interrupt levels has fired (one should have). If RTOP is set, initialization is successful. If not, there is an error in the receiver trip hardware. A message so stating is issued, and a request is made of the operator to retry or ignore the lack of receiver trip. Receiver trip has been built in such a way that even if initialization fails (i.e., the lack of signal is explicitly ignored) it can be made to initialize itself by providing the proper receiver trip signals at any point during operations. A spurious DEFOCUS command to task MMI (Man-Machine Interface Module) may result from such operation. RTOP will then indicate that R/T is operational.

Note further that RTOP will not indicate that the receiver trip hardware has gone non-operational once it has been initialized.

- e. Error messages and recovery - None, except as stated in "d." above.

3.2.7.4.1.21.2 Data, Logic and Command Paths

The bit RTOP is bit five in the global common word CPUSG. If on, it may be interpreted to mean the receiver trip hardware and software was operational at initialization; if off, then the receiver trip is not operational.

3.2.7.4.1.21.3 Internal Data Description

There is no data internal to this function.

3.2.7.4.1.21.4 Flowchart

See Figure 3.2.7-29 for the DBITRP flowchart.

3.2.7.4.1.22 Submodule XXII - DBIDTA

3.2.7.4.1.22.1 Description

- a. Language used - FORTRAN
- b. How invoked - subroutine call by DBI.
- c. Constraints and limitations - The disk data base must be initialized and accessible.
- d. Processing -

1. Call subroutine DBICOR to initialize the global common data base.
2. Call subroutine DBIERR to report any errors returned by DBICOR.
3. If the ABORT flag is returned true by DBICOR, go to step (13). Otherwise, go to step (7).
4. Call DBIAIM to initialize the global common aim-point array.
5. Call subroutine DBIERR to report any errors returned by DBIAIM.

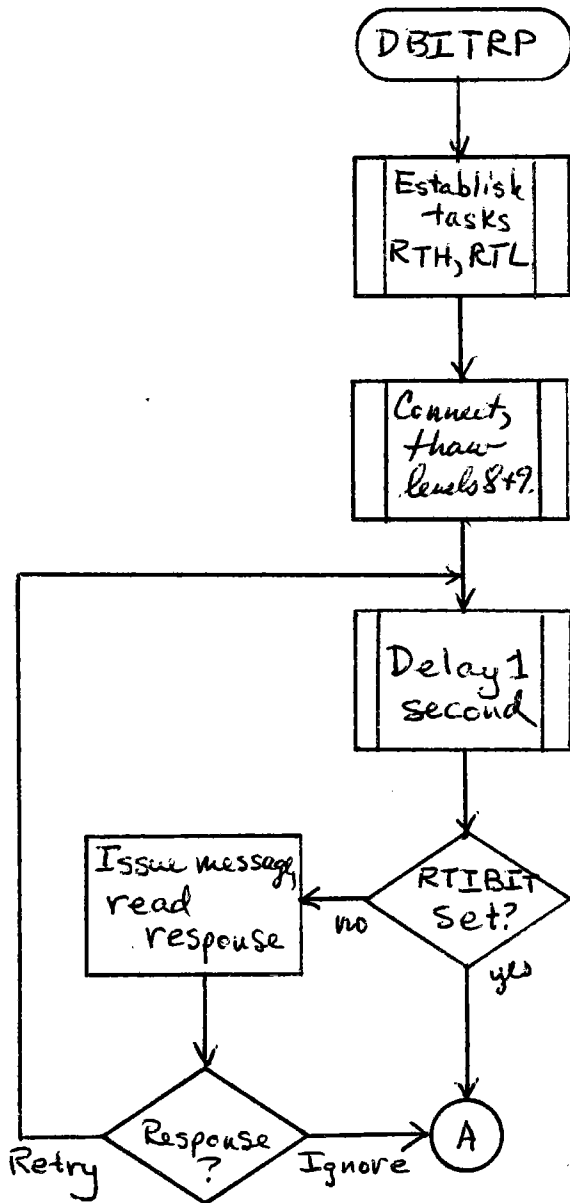


Figure 3.2.7.29 Flowchart - DBITRP

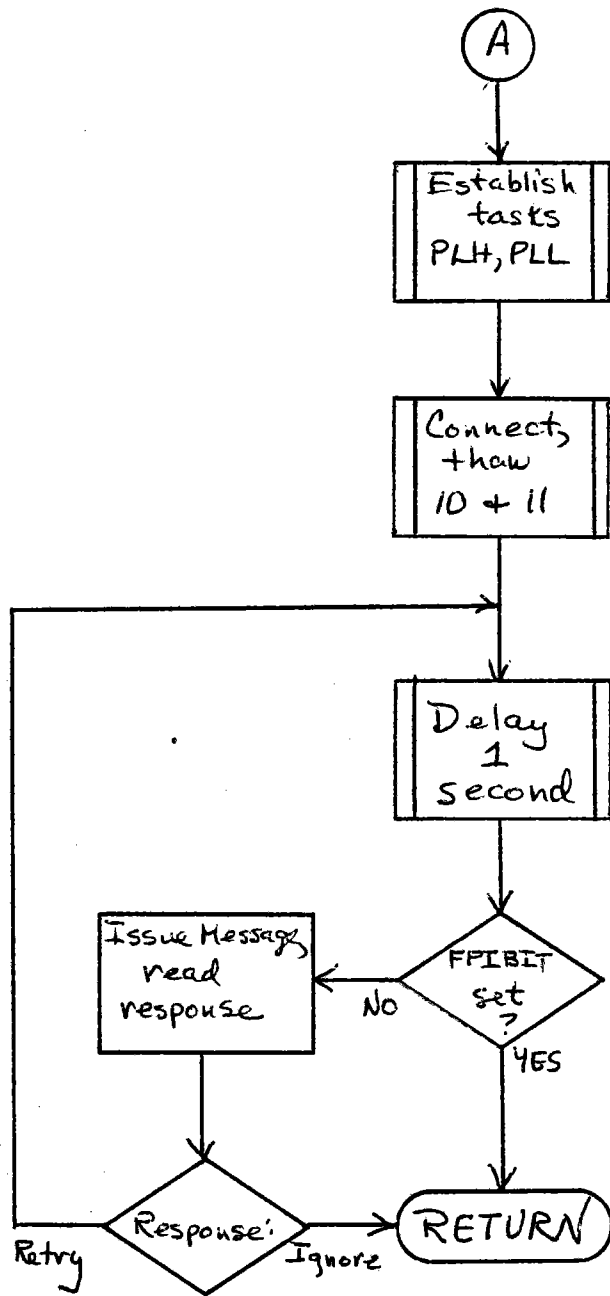


Figure 3.2.7.29 Flowchart - DBITRP (Continued)

6. If the ABORT flag is returned true by DBIAIM, go to step (13). Otherwise, go to step (10).
7. If bit 13 of global common word CPUSG is set, go to step (11). Otherwise, go to step (13).
8. Activate task DBIBCK to initialize the Backup HAC, and wait for DBIBCK to resume DBI task.
9. Return to the calling task DBI.

e. Error messages and recovery - None.

3.2.7.4.1.22.2 Data, Logic, and Command Paths

Input data:

- a. ABORT flag; and
- b. Global common word CPUSG.

Output data:

ABORT flag.

3.2.7.4.1.22.3 Internal Data Description

DBIDTA does not use any local data structures.

3.2.7.4.1.22.4 Flowchart

See Figure 3.2.7-30 for the DBIDTA flowchart.

3.2.7.4.1.23 Submodule XXIII - DBIBCK (Task)

3.2.7.4.1.23.1 Description

- a. Language used - FORTRAN
- b. How invoked - Activated by task DBI in submodule DBIDTA
- c. Constraints and limitations - Backup HAC must be loaded with M05 SYSGEN.
- d. Processing -
 1. Build user file table and can code device names for transfer.
 2. Move (Read) global common from global common area to local buffers.

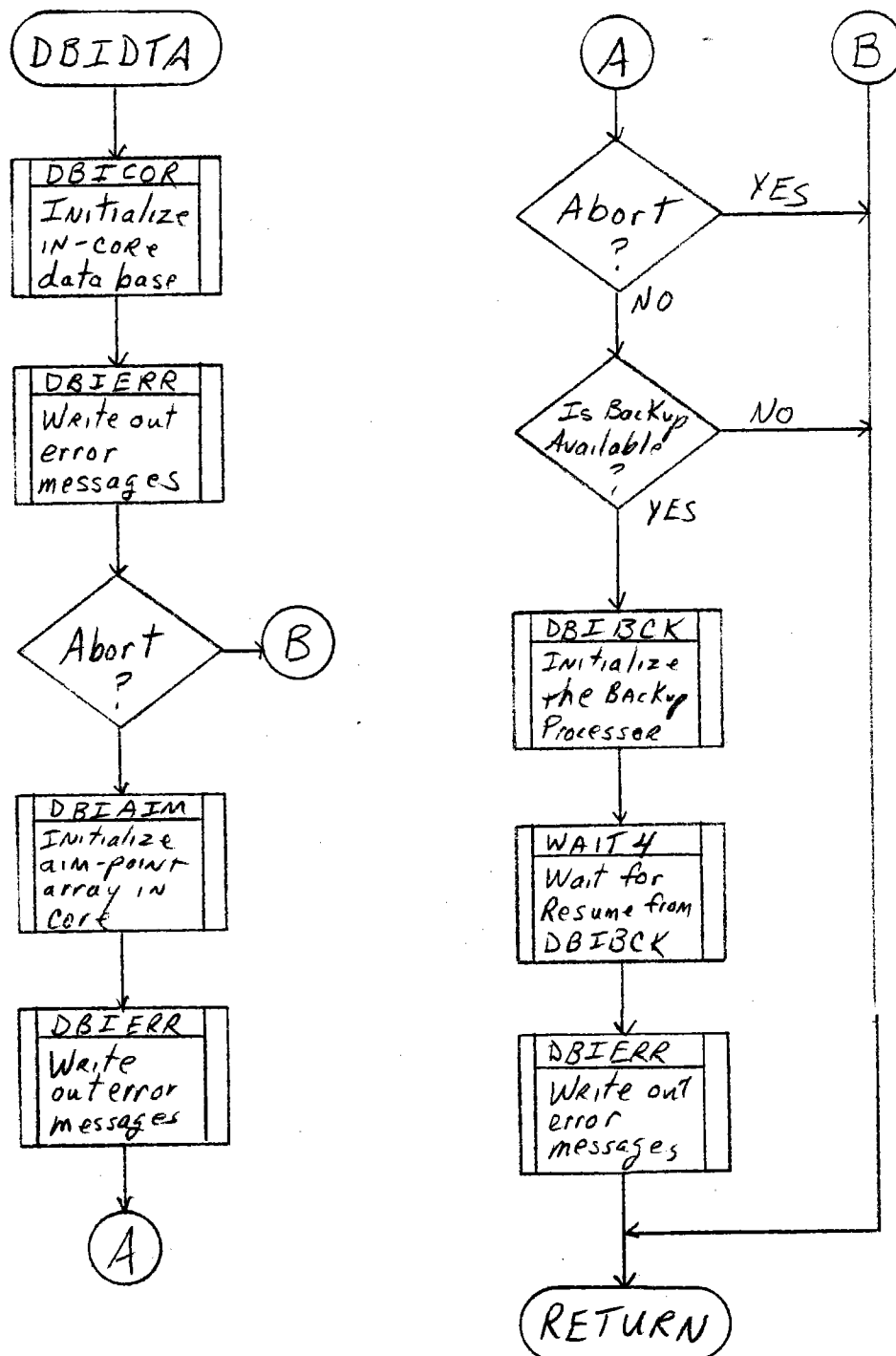


Figure 3.2.7-30 Flowchart - DBIDTA

3. Write local buffers across link to the backup HAC.
 4. If error in write, then write error message to operator and clear backup bit in CPUSG.
- e. Error messages and recovery - If an error occurs when the write is done to the backup HAC, and error message is written on the console, and CPUSG bit 13 is cleared, the message is:

'ERROR READ NUMBER' XXX UFT IS YY _ _ _ YY

where XXX is the index which points to which portion of global common was being written and YY - - YY is the user file table when the error occurred.

3.2.7.4.1.23-2 Data, Logic, and Command Paths

Input data:

- a. Global common areas
 - COMQUE
 - COMQU2
 - COMIN1
 - COMIN2
 - COMIN3
 - COMIN4

Output data:

Same as input data, except these areas are output to backup HAC.

3.2.7.4.1.23.3 Internal Data Description

Local buffers to hold global common during write operation to backup HAC.

3.2.7.4.1.23.4 Flowchart

See Figure 3.2.7-31 for DSIBCK flowchart.

3.2.7.4.1.24 Submodule XXIV - DBICOR

3.2.7.4.1.24.1 Description

- a. Language used - FORTRAN
- b. How invoked - Subroutine called by DIBDTA

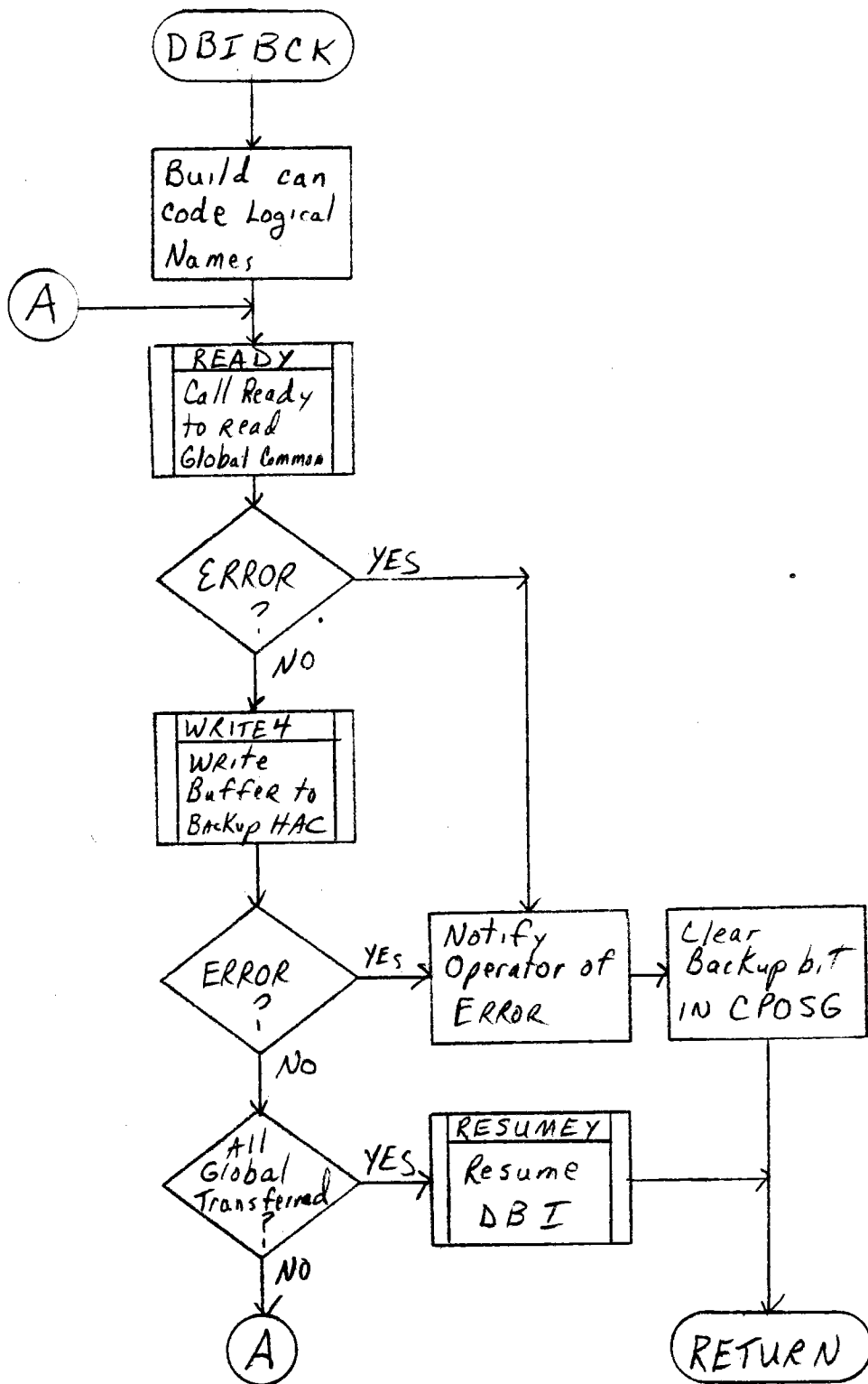


Figure 3.2.7-31 Flowchart - DBIBCK

c. Constraints and limitations - disk file DIN must be initialized prior to executing this subroutine.

d. Processing -

1. Initialize the abort flag false, the error code to zero, and the number of errors to zero. Initialize the message queues.

2. Initialize all of global common (COMDAT) to zero, except CPUSG and the TBUSYG words.

3. Initialize the 2048-word arrays, ELEVG and AZIMG, to:

ELEVG = $-90^{\circ} * 91.02222$

AZIMG = 0.0

4. Initialize the 2048-word array, GRSTSG, to a value of 12 (heliostats offline).

5. Initialize the 16-word array, LINESG, to:

Normal lines = #0800

Alternate lines = #0000

6. Initialize the 16-word array, SEQLSG, to the negative of its index number (i.e., SEQLSG(1) = -1, ..., SEQLSG(16) = -16).

7. Initialize the 64-word array, HFCS2G, to a value of #B000.

8. Initialize the double-precision word, SLATG, to the site latitude.

9. Initialize the double-precision word, SLONGG, to the site longitude.

10. Read disk file, DIN, records 1-16 to obtain the 2048-word array, MD2HCG, and store it in global common.

11. If a read error occurred, go to step (27); otherwise, go to step (12).

12. Read disk file, DIN, record 17 to obtain the 30-word array, MDNPRG, and the 60-word array, SEGPTG. Store both arrays in global common.

13. If a read error occurred go to step (27); otherwise, go to step (14).

14. Read disk file, DIN, records 18 to 33 to obtain the 2048-word array, HC2MDG, and store it in global common.
15. If a read error occurred, go to step (27); otherwise, go to step (16).
16. Read disk file, DIN, records 34 to 50 to obtain the 2108-word array, SEGMPG, and 20-word array, BCSTGG. Store both of these arrays in global common.
17. If a read error occurred, go to step (27); otherwise, go to step (18).
18. Read disk file, DIN, record 51 to obtain the corridor coordinates and store them into the 120-word global common array, CORRCG.
19. If a read error occurred, go to step (27); otherwise, go to step (20).
20. Initialize the 2048-word array, HCST2G, to a value of #8000 (not installed).
21. Using the SEGPTG and SEGMPG arrays, set the indicated HCST2G words to a value of #0800 (installed, not marked).
22. Read one record (one HFC) of HC location from the disk to obtain the HC's corridor and BCS assignments.
23. If a read error occurred, go to step (27); otherwise, go to step (24).
24. Initialize the HCST3G words (one HFC) with the corridor and BCS assignments (up to 32 words).
25. Initialize one word of the global common array, HFC3G, with the HFC corridor assignments.
26. If all the HFCs are processed, go to step (28); otherwise, go to step (22).
27. Set the abort flag true, set an error code and increment the number of errors.
28. Return to the calling subroutine.

- e. Error messages and recovery -Disk read errors result in an abort flag being set and an error code returned to the calling subroutine, DBIDTA.

3.2.7.4.1.24.2 Data, Logic and Command Paths

Input data:

Data stored on disk file DIN.

Output data:

- a. Abort flag;
- b. Error code;
- c. Number of errors; and
- d. Initial values into the global common area /COMDAT/.

3.2.7.4.1.24.3 Internal Data Description

The initial values for the arrays AZIMG and ELEVG are set in data statements. The initial values for SIATG and SLONGG (site latitude and longitude) are set in data statements.

3.2.7.4.1.24.4 Flowchart

See Figure 3.2.7-32 for DBICOR flowchart.

3.2.7.4.1.25 Submodule XXV - DBIAIM

3.2.7.4.1.25.1 Description

- a. Language used - FORTRAN
- b. How invoked - subroutine call by DBIDTA.
- c. Constraints and limitations - The global common array HCST2G must be initialized to its installed or not-installed state.
- d. Processing -
 1. Set the abort flag false, the error code to zero, and the number of errors to zero.
 2. Set the global common array AIMOKG equal to one, and set the installed flag false.
 3. Initialize a counter to one.
 4. If the counter I is greater than 2048, go to step (14); otherwise, go to step (5).

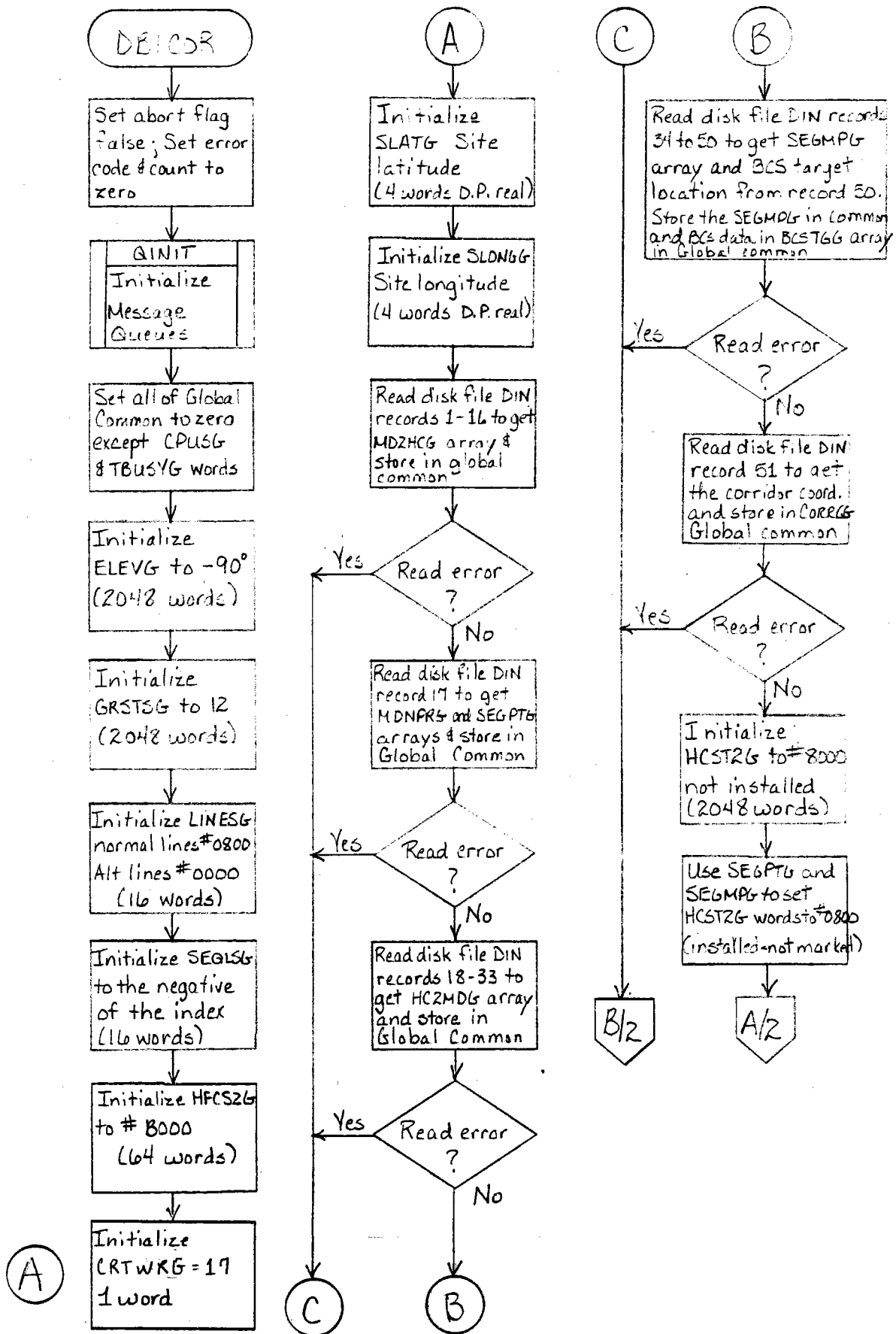


Figure 3.2.7- 32 Flowchart DBICOR

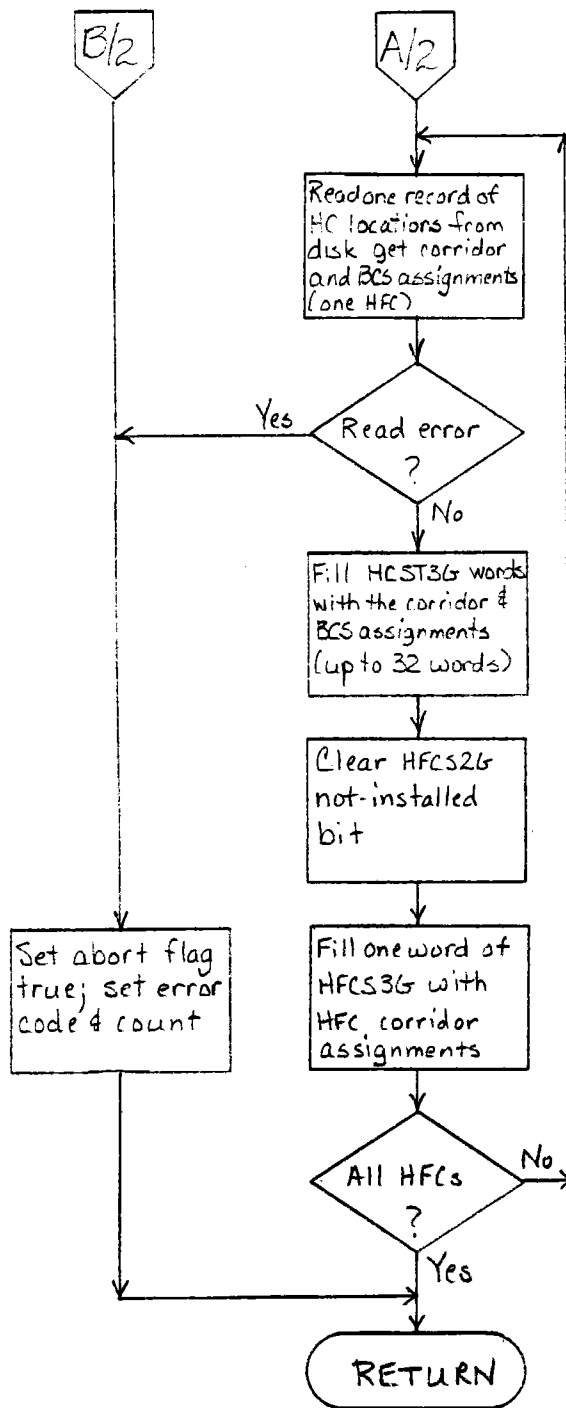


Figure 3.2.7-32 Flowchart - DBICOR (Continued)

5. If the HC is installed, go to step (6); otherwise, increment the counter I and go to step (4).
6. Set the installed flag true.
7. Read one record of the aim-point file indexed by the HC number (i.e., I).
8. If a read error occurs, set the abort flag true, set the error code, increment the number of errors, and go to step (16); otherwise, go to step (9).
9. Initialize the aim-point counter to one.
10. If the aim-point counter is greater than 20, increment the counter I and go to step (4).
11. If the aim-point counter equals the last byte of the five-word aim point, go to step (13). Otherwise, go to step (12).
12. Set the AIMOKG element to zero that corresponds to the aim-point counter.
13. Increment the aim-point counter, and go to step (10).
14. If the installed flag is true, go to step (17). Otherwise, go to step (15).
15. Set the abort flag true, set the error code, and increment the error count.
16. Set the AIMOKG global common array to zero.
17. If the AIMOKG array is all zero, set the abort flag true, set the error code, increment the number of errors, and go to step (20). Otherwise, go to step (18).
18. Read the lowest cardinal-numbered aim-point array from disk and store it into the global common array AIMPTG.
19. If a read error occurs, set the abort flag true, set the error code, increment the number of errors, and go to step (20). Otherwise, go to step (20).
20. Return to the calling subroutine.

- e. Error messages and recovery - If the processing finds that no heliostats are installed or that no valid aim-point array exists on the disk, an abort flag and error code are returned to the calling subroutine.

3.2.7.4.1.25.2 Data, Logic and Command Paths

Input data:

- a. HCST2G global common array; and
- b. Aim-point arrays from the disk.

Output data:

- a. Abort flag;
- b. Error code;
- c. Number of errors;
- d. Global common array AIMOKG; and
- e. Global common array AIMPTG.

3.2.7.4.1.25.3 Internal Data Description

A 128-word buffer is utilized to read records from the aim-point disk file.

3.2.7.4.1.25.4 Flowchart

See Figure 3.2.7-33 for the DBIAIM flowchart.

3.2.7.4.1.26 Submodule XXVI--DBIBCK

3.2.7.4.1.26.1 Description

- a. Language used - FORTRAN
- b. How invoked - Subroutine call by DBIDTA.
- c. Constraints and limitations - This subroutine should only be executed when there is a Backup processor.
- d. Processing -
 1. Set the error code to zero.
 2. Write the global common data base (COMDAT) to the Backup processor and exclude the CPU-peculiar status word, CPUSG.
 3. If there is a write error, set the error code, and go to step (8). Otherwise, go to step (10).

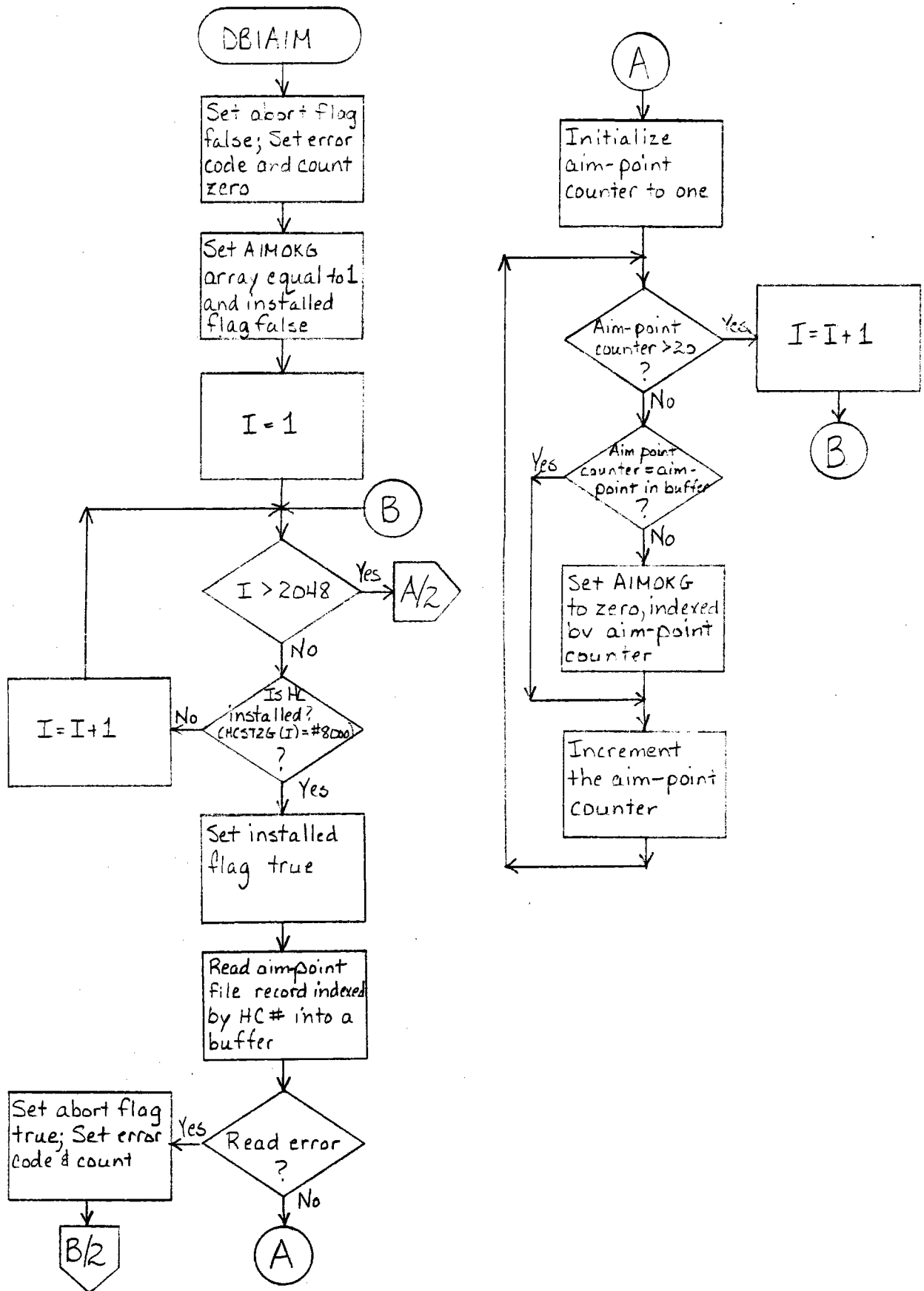


Figure 3.2.7-33 Flowchart DBIAIM

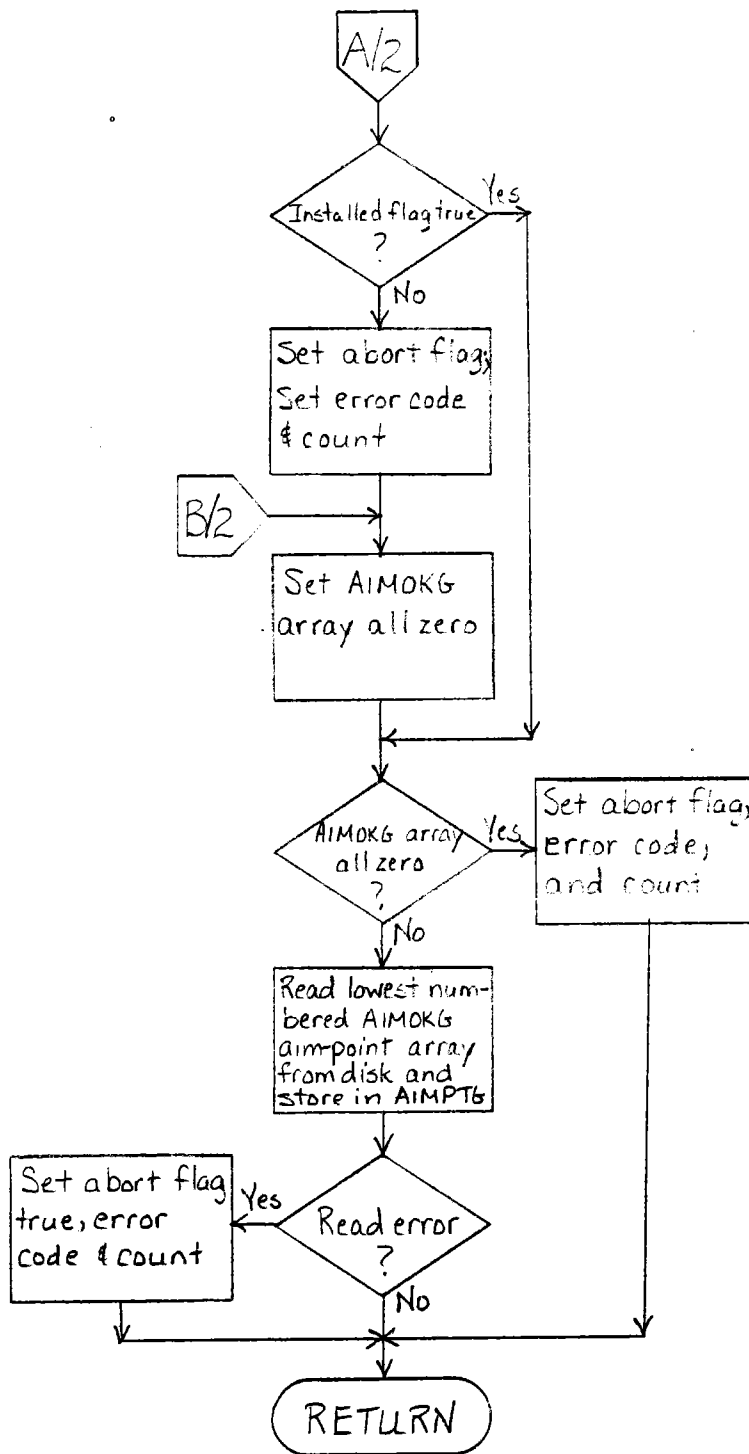


Figure 3.2.7-33 Flowchart - DBIAIM (Continued)

4. Write the disk data base to the Backup.
5. If there is a write error, set the error code, and go to step (8). Otherwise, go to step (10).
6. Write the global common queue area (CQUEUE) to the Backup processor.
7. If there is a write error, set the error code, and go to step (8). Otherwise, go to step (10).
8. Notify operator of backup initialization failure and request if another try is desired. If so go to step 1. Otherwise, go to step (10).
9. Clear backup bit in CPUSG.
10. Return to the calling subroutine DBIDTA.

- e. Error messages and recovery - If there is an I/O error, this subroutine repeats the complete Backup initialization up to three times. One pass through this subroutine which results in three I/O errors is sufficient to result in not trying again.

.3.2.7.4.1.26.2 Data, Logic and Command Paths

Input data:

- a. Global common area, COMDAT (exclude CPUSG);
- b. Disk data base files;
 1. DIN-disk file containing results of offline processing.
 2. HC locations.
 3. HC Wash angles.
 4. HC Stow angles.
 5. HC Alternate 1 Stow angles.
 6. HC Alternate 2 Stow angles.
 7. Field-status save file.

8. HC bias file.

9. HC aim-point file.

- c. Global common area, CQUEUE, which contains the initialized message queue area.

Output data:

All the data input is output to the Backup processor.

3.2.7.4.1.26.3 Internal Data Description

A 160-word local array is used to read the Prime disk data base.

3.2.7.4.1.26.4 Flowchart

See Figure 3.2.7-34 for the DBIBCK flowchart.

3.2.7.4.1.27 Submodule XXVII--DBIERR

3.2.7.4.1.27.1 Description

- a. Language used - FORTRAN
- b. How invoked - Subroutine call from tasks DIN, DBI and CLK.
- c. Constraints and limitations - This submodule attempts to write messages on the primary output device specified via a formal parameter. If a write error occurs, an alternate output device is selected and tried. If a write error occurs again, this submodule returns to the calling submodule with no output produced.
- d. Processing -
 1. Initialize the counter that tracks the number of times an alternate device is tried to one.
 2. Test if there are any error messages to output. If there are no error messages, return to the calling submodule. If there are messages to output, go to step (3).
 3. Call the subroutine DINTTL to write a heading title when required.
 4. Write the error message indicated by the formal-parameter array input, and increment the printer line-number counter.

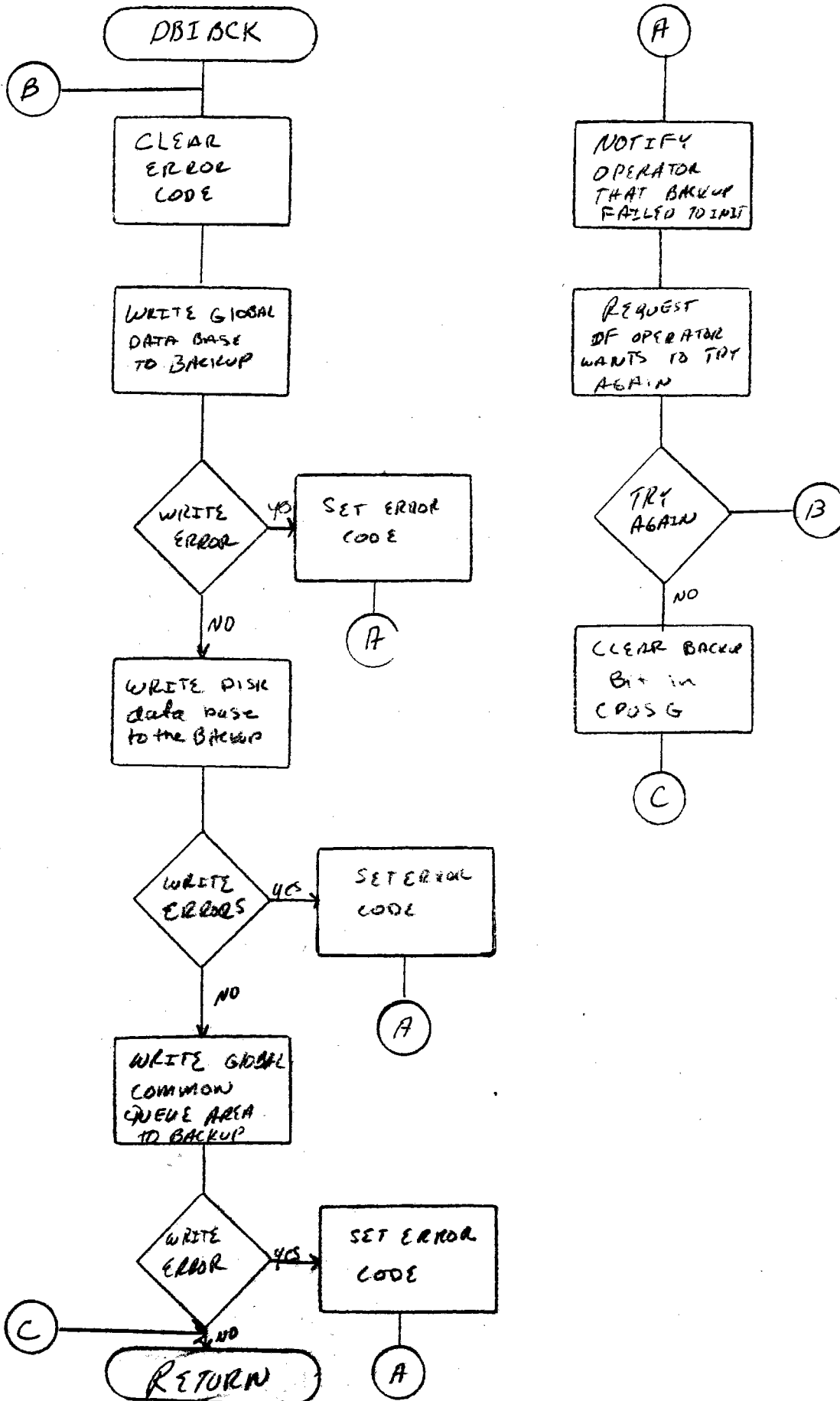


Figure 3.2.7-34 Flowchart - DBIBCK

5. If a write error occurs, go to the next step; otherwise, go to step (10).
 6. Increment the device try counter.
 7. Change the device number to the other device number.
 8. If the device try counter is three, return to the calling subroutine. Otherwise, decrement the printer line-number counter and go to step (3).
 9. If all the error messages are written, return to the calling subroutine. Otherwise, decrement the number of error messages counter, and go to step (3).
- e. Error messages and recovery - If the primary output device indicates an error, a secondary device is tried for the output.

Error messages - See Table 3.2.7- IX.

3.2.7.4.1.27.2 Data, Logic and Command Paths

Input data:

- a. Number of error messages to print;
- b. Address of error-message codes array;
- c. A number to tag onto an error message when desired;
- d. The output device number;
- e. The current line number to print to; and
- f. The number of the heading title desired.

Output data:

- a. Line number to print to;
- b. Output device number; and
- c. Error messages to output device.

3.2.7.4.1.27.3 Internal Data Description

The error messages are stored in arrays that are stored into a format array for output.

3.2.7.4.1.27.4 Flowchart

See Figure 3.2.7-35 for the DBIERR flowchart.

ERROR: DISK READ/WRITE ERROR
ERROR: READ IN SOURCE DEVICE REC:NNNN
ERROR: SYNTAX ERROR REC:NNNN
ERROR: DECODE ERROR REC:NNNN
ERROR: SOURCE VALUE OUT OF RANGE REC:NNNN
ERROR: AIM POINT NOT IN INCLUSION AREA; FILE REJECTED REC:NNNN
ERROR: SYNTAX ERROR IN SOURCE AZIMUTH OR ELEVATION, DEFAULT SUPPLIED REC:NNNN
ERROR: SYNTAX FOR CARD-TYPE DESIGNATOR REC:NNNN
ERROR: SYNTAX FOR HC NUMBER REC:NNNN
ERROR: DECODE OF CARD TYPE OR HC NUMBER REC:NNNN
ERROR: DECODE OF AZIMUTH OR ELEVATION, DEFAULT SUPPLIED REC:NNNN
ERROR: GRAPHICS CONSOLE NOT INITIALIZED AT OPERATOR'S REQUEST
ERROR: TASK XXX FAILED TO ESTABLISH
ERROR: TASK XXX FAILED TO ACTIVATE
ABORT: DISK DATA BASE HC LOCATIONS NOT VALID
ABORT: CAN NOT READ THE DISK DATA BASE
ABORT: NO HELIOSTATS ARE INSTALLED

ABORT: DISK DATA BASE WASH ANGLES NOT VALID
ABORT: DISK DATA BASE STOW ANGLES NOT VALID
ABORT: DISK DATA BASE ALT 1 STOW ANGLES NOT VALID
ABORT: DISK DATA BASE ALT 2 STOW ANGLES NOT VALID

ABORT: NO AIM-POINT ARRAYS EXIST
ERROR: CAN NOT WRITE TO BACKUP HAC GLOBAL COMMON
ERROR: CAN NOT WRITE TO BACKUP HAC DISK DATA BASE

Table 3.2.7- IX Error Messages

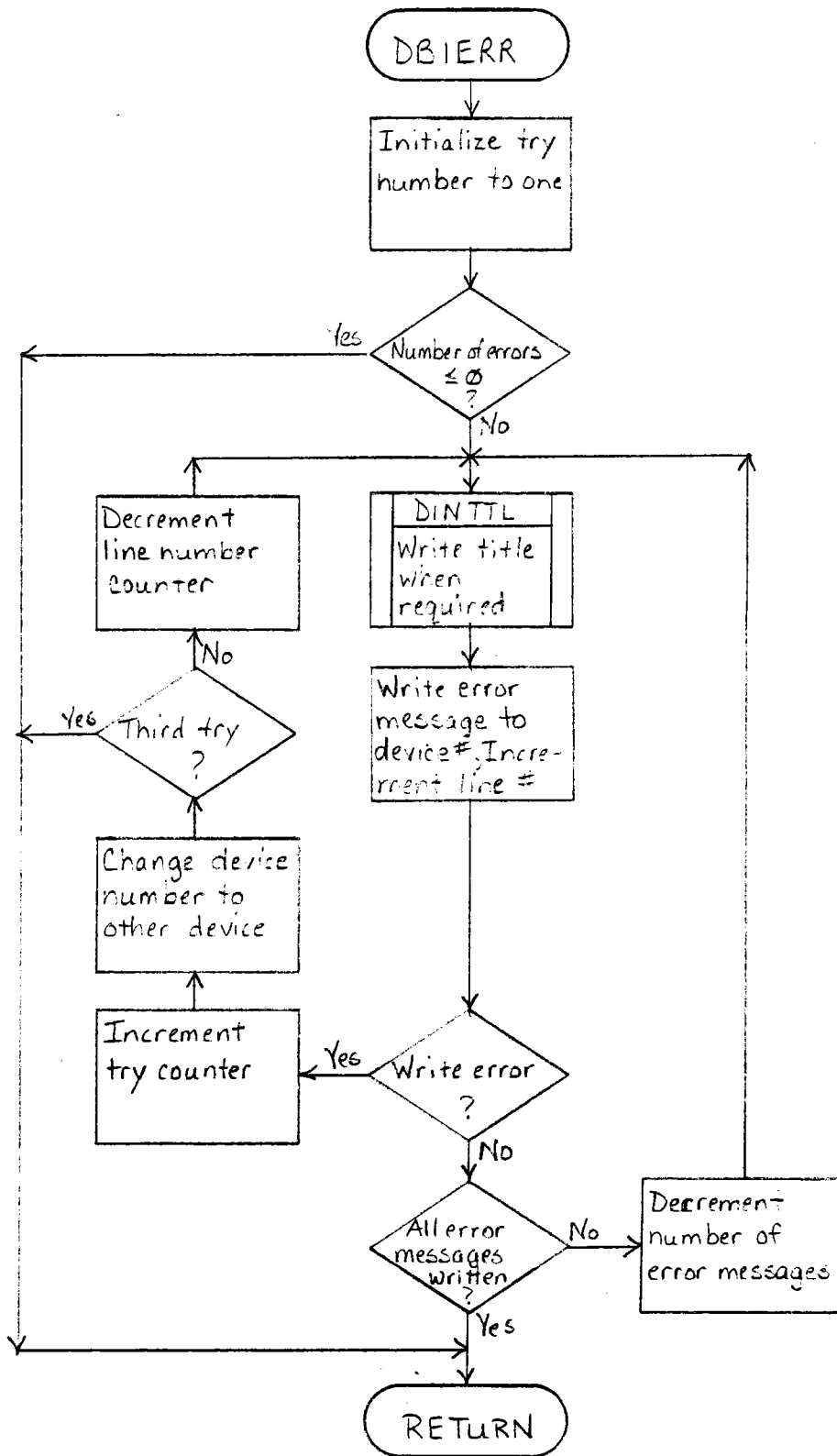


Figure 3.2.7-35 Flowchart - DBIERR

3.2.7.4.1.28 Submodule XXVIII--CLK (Task)

3.2.7.4.1.28.1 Description

- a. Language used - FORTRAN
- b. How invoked - Established and activated by DBI task.
- c. Constraints and limitations - System configuration must be initialized, the global common data base must be initialized, and the disk data base must be initialized and accessible.
- d. Processing -

1. Call subroutine CLKEST to establish the real-time HAC tasks in main memory.
2. If a task failed to establish, notify the operator and go to step (6). Otherwise, go to step (3).
3. If execution is in the Prime HAC, go to step (4). Otherwise, go to step (7).
4. Call subroutine CLKACT to activate the real-time asynchronous tasks.
5. If a task failed to activate, notify the operator and go to step (7). Otherwise, go to step (6).
6. Call subroutine CLKONL to initialize time-keeping and activate the HAC real-time synchronous tasks.
7. If any errors occurred, go to step (8); otherwise, go to step (11).
8. Abort all HAC tasks except CLK and SWI.
9. Request if the operator wants to try again.
10. If the operator requests to try again, go to step (7). Otherwise, go to step (11).
11. EXIT this task.

e. Error messages and recovery -

1. Error messages:

- a) TASK XXX FAILED TO ESTABLISH
- b) TASK XXX FAILED TO ACTIVATE
- c) ALL HAC TASKS ABORTED

OPTION	TRY AGAIN
0	NO
1	YES

2. Error recovery:

Error recovery is an operator's option.

3.2.7.4.1.28.2 Data, Logic and Command Paths

Input data:

- a. Global common word CPUSG; and
- b. Operator's response when required.

Output data: (a) Error Messages.

3.2.7.4.1.28.3 Internal Data Description

Task CLK does not use any local data structures.

3.2.7.4.1.28.4 Flowchart

See Figure 3.2.7-36 for the CLK flowchart.

3.2.7.4.1.29 Submodule XXIX - CLKEST

3.2.7.4.1.29.1 Description

- a. Language used - FORTRAN
- b. How invoked - Subroutine call by CLK.
- c. Constraints and limitations - The real-time HAC tasks must be cataloged on the disk.
- d. Processing -

- 1. Set the error flag false, and clear the task name.

- 3. Establish the HAC tasks: TOK, TIK, FCP, SUN, BHI, ALM, STS, BHC, CSI, GRF, CSO, EXI, and EXO.

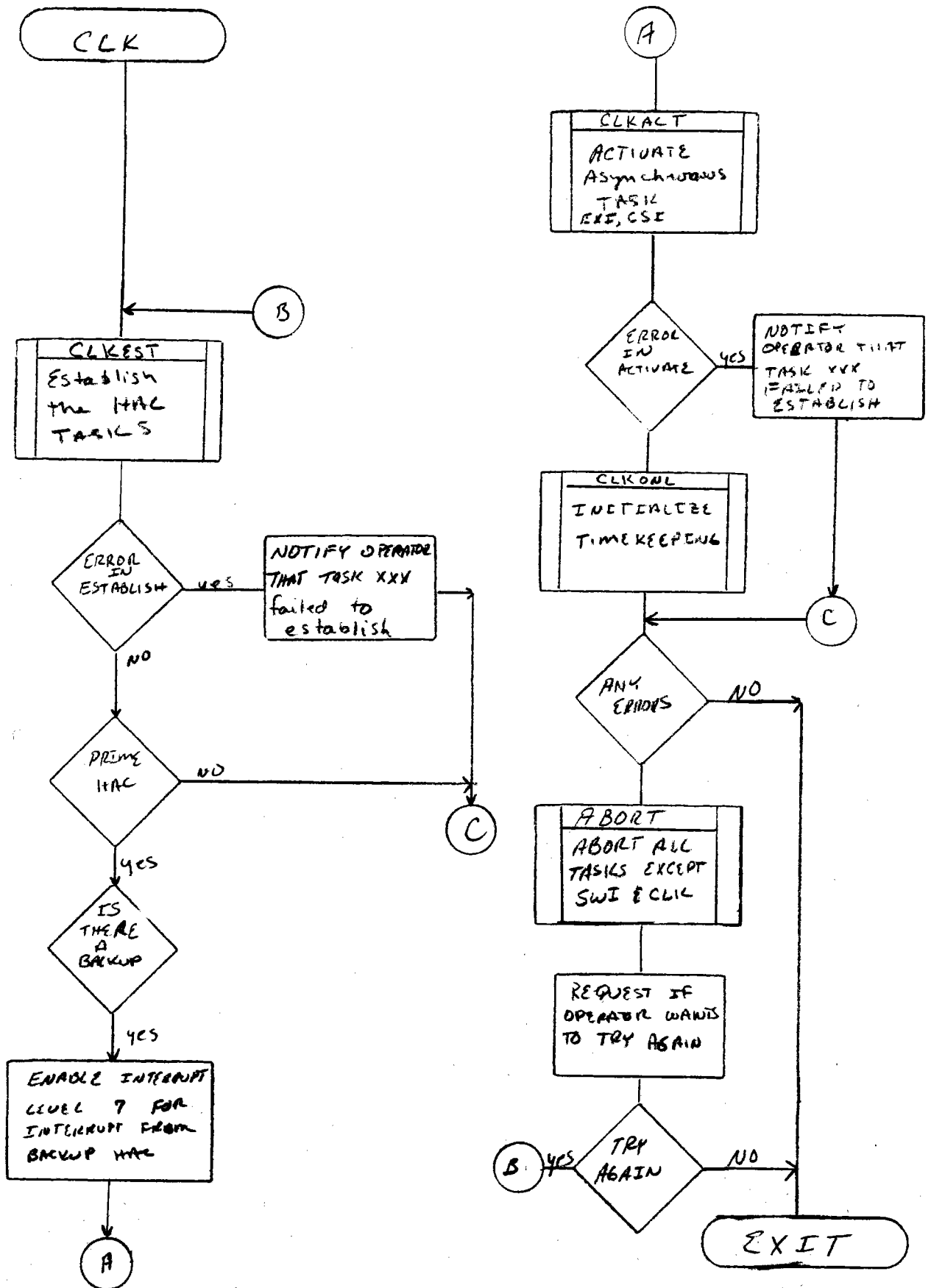


Figure 3.2.7-36 Flowchart - CLK

4. If any HAC real-time task fails to establish, go to step (8). Otherwise, go to step (5).
5. If in Prime, go to step (6). Otherwise, go to step (8).
6. Establish HAC tasks for Prime.
7. If any HAC real-time task fails to establish, go to step (8). Otherwise, go to step (10).
8. Set the error flag true, and set the task name in a parameter word.
9. De-establish all the HAC tasks that were established.
10. Return to the calling task CLK.

e. Error messages and recovery - Parameters returned to the calling task (CLK) for processing, and tasks de-established.

3.2.7.4.1.29.2 Data, Logic, and Command Paths

Input data: None

Output data:

- a. Error flag; and
- b. Task name.

3.2.7.4.1.29.3 Internal Data Description

The HAC real-time task names are initialized in ASCII code via data statements. The task names are: TOK, TIK, FCP, SUN, BHI, ALM, STS, BHC, CSI, GRF, MMI, CFO, DSK, CSO, CMD, GET, SEQ, EXI, STA, and ALO.

3.2.7.4.1.29.4 Flowchart

See Figure 3.2.7-37 for the CLKEST flowchart.

3.2.7.4.1.30 Submodule XXX - CLKONL (Timekeeping Initialization)

3.2.7.4.1.30.1 Description

- a. Language used - FORTRAN
- b. How invoked - Subroutine call from CLK (if Prime HAC).

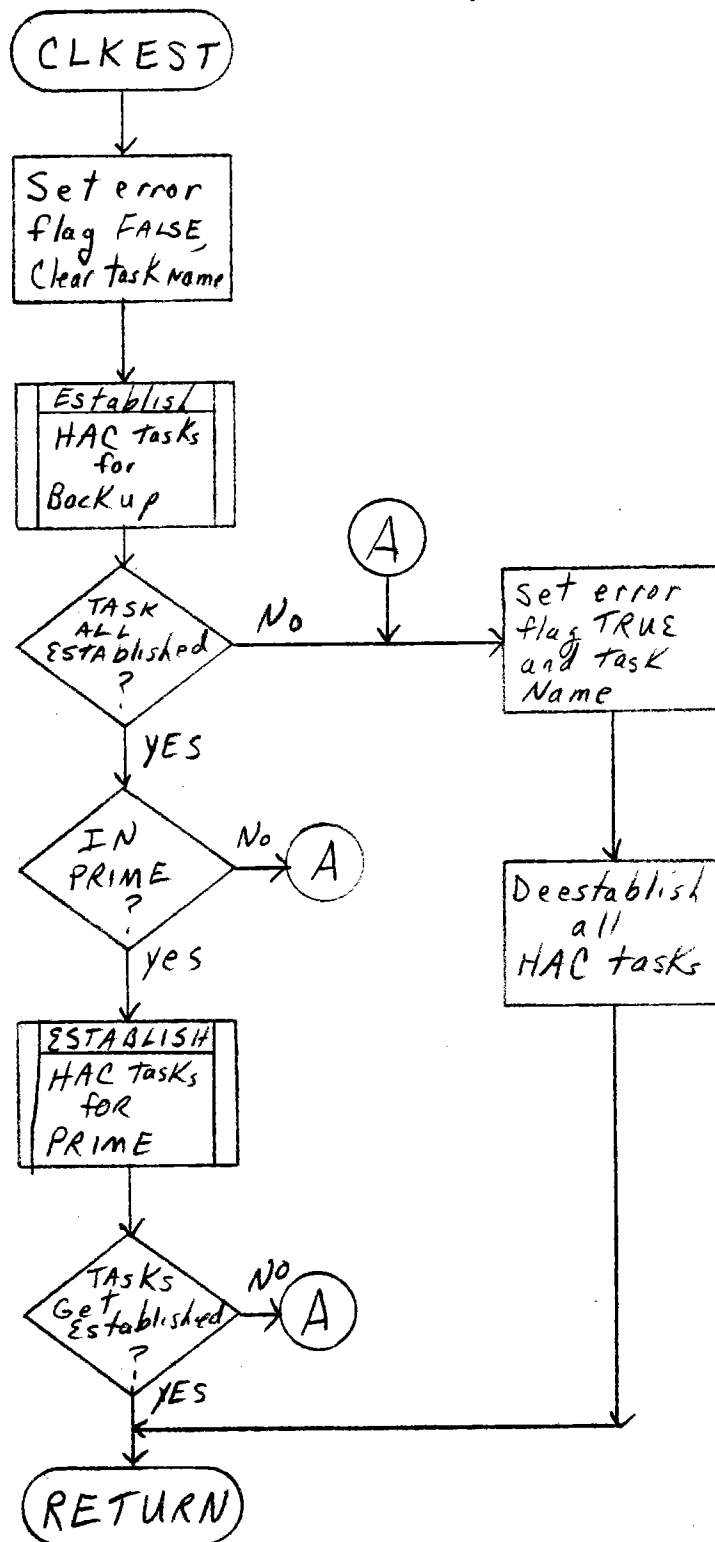


Figure 3.2.7-37 Flowchart - CLKEST

- c. Constraints and limitations - Time zones are limited to Pacific and Mountain, both Daylight and Standard times, and Greenwich Mean Time (GMT).
- d. Processing - CLKONL performs all the work required to initially establish and provide continued maintenance of the GTIMEG array (along with the CLOCK function, 3.2.8.I). Operator inputs nominally include the local time zone and the local year. All

other data is taken from the WWV receiver. In the event the receiver is not operational, the local month, day, hour, and minute, as well as a "mark" will be requested. TIK and TOK are activated, pertinent data is logged, GTIMEG is wholly initialized, and return is made to CLK.

- e. Error messages and recovery - None, except as mentioned in "d." above.

3.2.7.4.1.30.2 Data, Logic and Command Paths

Definitions:

GMT - Greenwich Mean Time, functionally identical to UTC, Universal Coordinated Time.

Local Time - Time kept in the time zone of interest, an integer number of hours offset from GMT.

The GTIMEG array contains all the variables used in time computations, as well as being CLKONL's final output:

<u>Offset in GTIMEG</u>	<u>FORTTRAN Array Index</u>	<u>Contents</u>	<u>Range</u>	<u>Mnemonic</u>
0	1	GMT year	1980 - 2100	GYR
1	2	GMT day of year	1-366	GDY
2	3	GMT hour of day	0-23	GHR
3	4	Minute in hour	0-59	MIN
4	5	Second in minute	0-59	SEC
5	6	Local year	1980-2100	LYR
6	7	Local month	1-12	LMO
7	8	Local day of month	1-31	LDY
8	9	Local hour of day	0-23	LHR
9	10	Time Quality	0-5	---
10	11	Days in GMT year	365-366	NDY
11	12	Days in local month	28-31	NDM
12	13	Hours offset, local to GMT	0-23	OFF
13	14	Days in local year	365-366	NDL

The algorithm converting GMT to local time is shown in Figure 3.2.7-38 and the converse algorithm in Figure 3.2.7-39.

3.2.7.4.1.30.3 Internal Data Structures

The only internal data structure in CLKONL is the formatted data received from the WWV receiver. It has the format:

<CTLA>DDD:HH:MM:SSQ<CR><LF>

Where <CTLA>, <CR>, and <LF> are the hexadecimal values #1, # , and # respectively; DDD, HH, MM, and SS are numbers, and Q is one of the set "?," "#," "-", "1," and "blank."

3.2.7.4.1.30.4 Flowchart

See Figure 3.2.7-40 for the CLKONL flowchart.

3.2.7.4.1.31 Submodule XXV - CLKACT

3.2.7.4.1.31.1 Description

- a. Language used - FORTRAN
- b. How invoked - Subroutine call by CLK.
- c. Constraints and limitations - Any task that this submodule activates must have been established as main-memory resident.
- d. Processing -
 1. Set the error flag false, and clear the task name word.
 2. Activate the HAC tasks: CSI and EXI.
 3. If all HAC real-time asynchronous tasks are activated, go to step (6). Otherwise, go to step (4).
 4. Abort the HAC real-time asynchronous tasks that were activated.
 5. Set the error flag true, and set the task name in a parameter word.
 6. If a backup HAC is present, then remotely activate the SWI task in the backup HAC.
 7. If remote activate is successful, then set backup list in CPUSG and go to step (9). Otherwise, go to step (8).

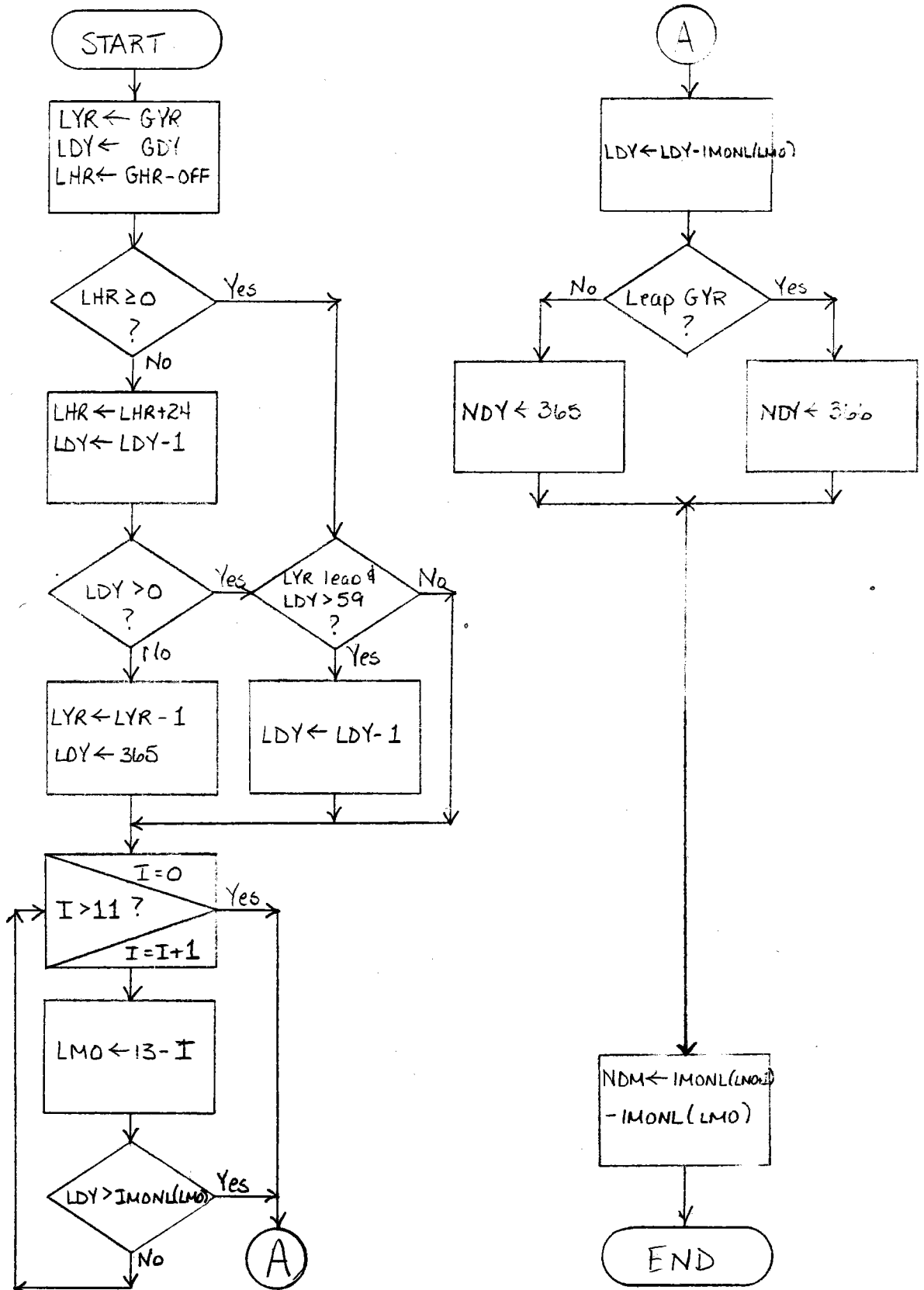


Figure 3.2.7- 38 Flowchart - Computation of Local Time

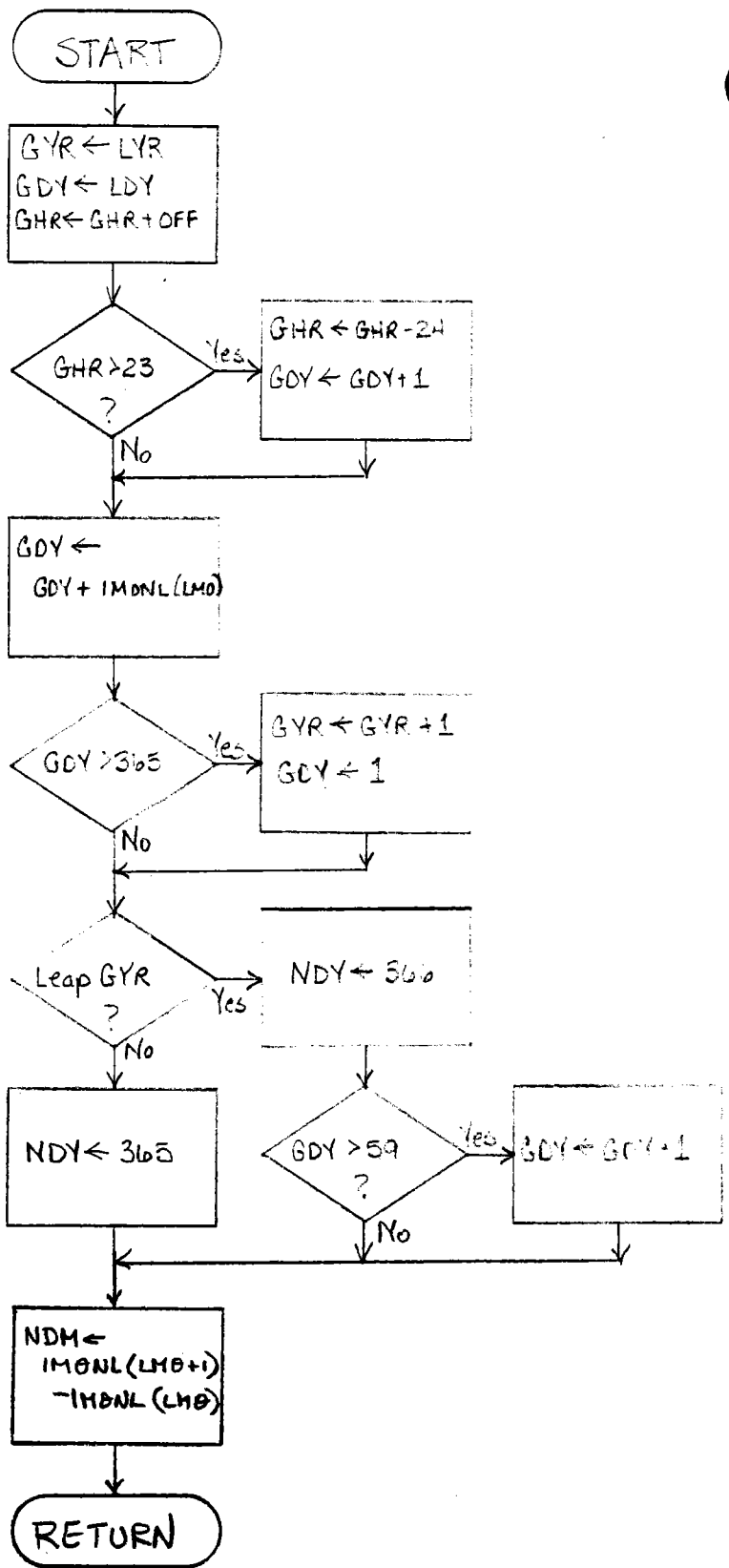


Figure 3.2.7-39 Flowchart - Computation of GMT Parameters

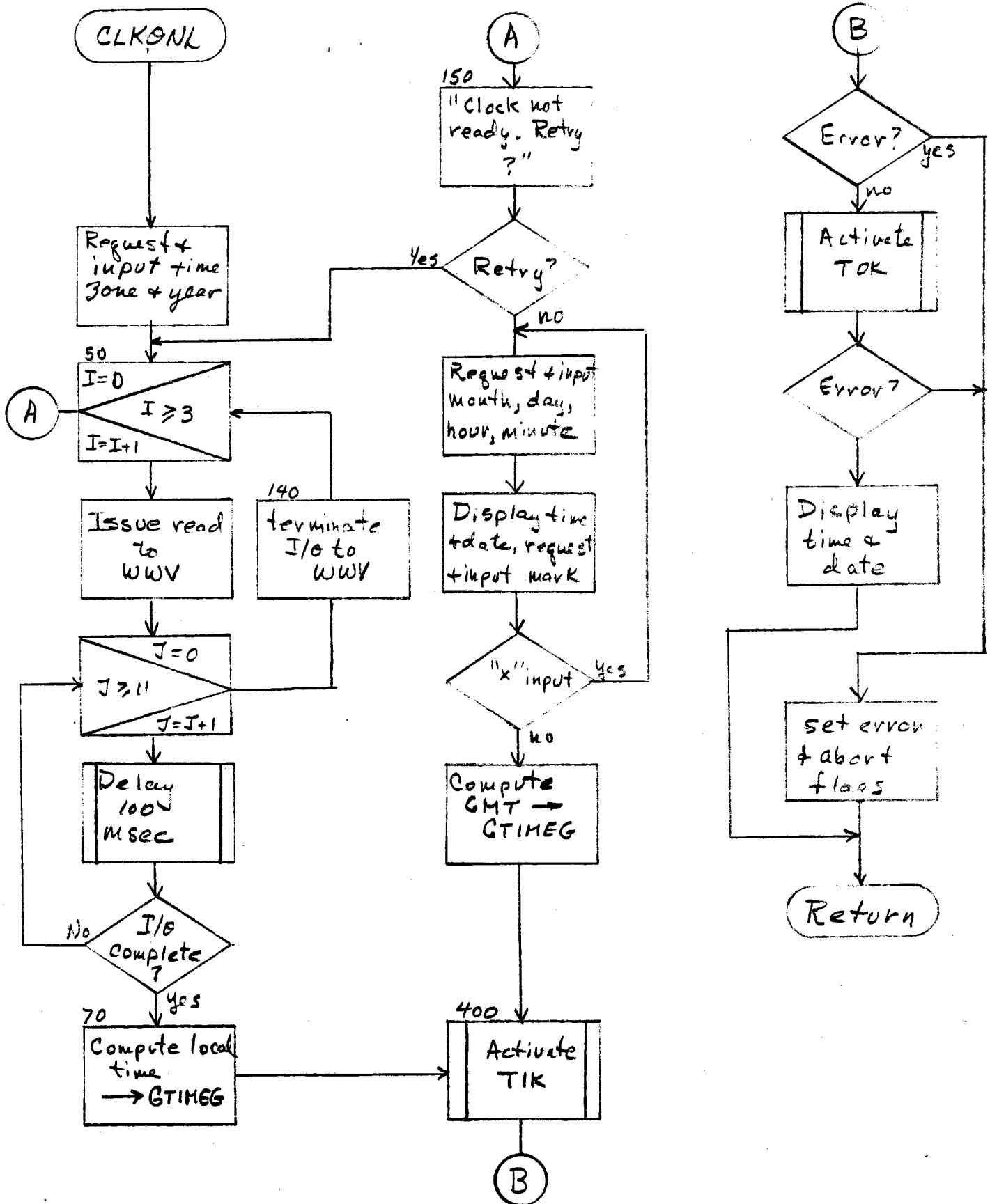


Figure 3.2.7.40 Flowchart - CLKONL

8. Set the error flag true and set the task name in parameter word.

9. Return to the calling submodule.

e. Error messages and recovery - Parameters are returned to the calling task (CLK) for processing, and tasks are aborted.

3.2.7.4.1.31.2 Data, Logic and Command Paths

Input data:

None

Output data:

- a. Error flag; and
- b. Task name.

3.2.7.4.1.31.3 Internal Data Description

The HAC real-time asynchronous task names are initialized in ASCII code via data statements. The task names are: CSI and EXI.

3.2.7.4.1.31.4 Flowcharts

See Figure 3.2.7-41 for the CLKACT flowchart.

3.2.7.5 Interface Description

The task DIN interfaces with the Disk Data Base, source records from either the magnetic tape or card reader, graphics processors, and the operator via the TI-820 console.

The task DBI interfaces with the Disk Data Base, Prime-Backup HAC configuration, OCS and DAS communications, graphics processors, and the operator via the TI-820 console. DBI also establishes and activates the HAC tasks CLK, RTH, and RTL.

The task CLK interfaces with the WWV device, and the operator for time initialization. Further, CLK establishes the real-time tasks in the Prime HAC and backup HAC. The CLK task activates the tasks in the prime HAC and the switchover task SWI in the Backup HAC.

3.2.7.6 Test Requirements

The Disk Data Base is dumped onto the line printer and verified that the source records are properly initialized. The global common data base is dumped onto the line printer and verified that the data is initialized correctly. The ISC initial state is verified by visual inspection, as is the graphics processors. Real-time task establishment and activation is verified through system status requests from the system console. Error reporting is verified by inputting deliberate errors in source data, and/or device state.

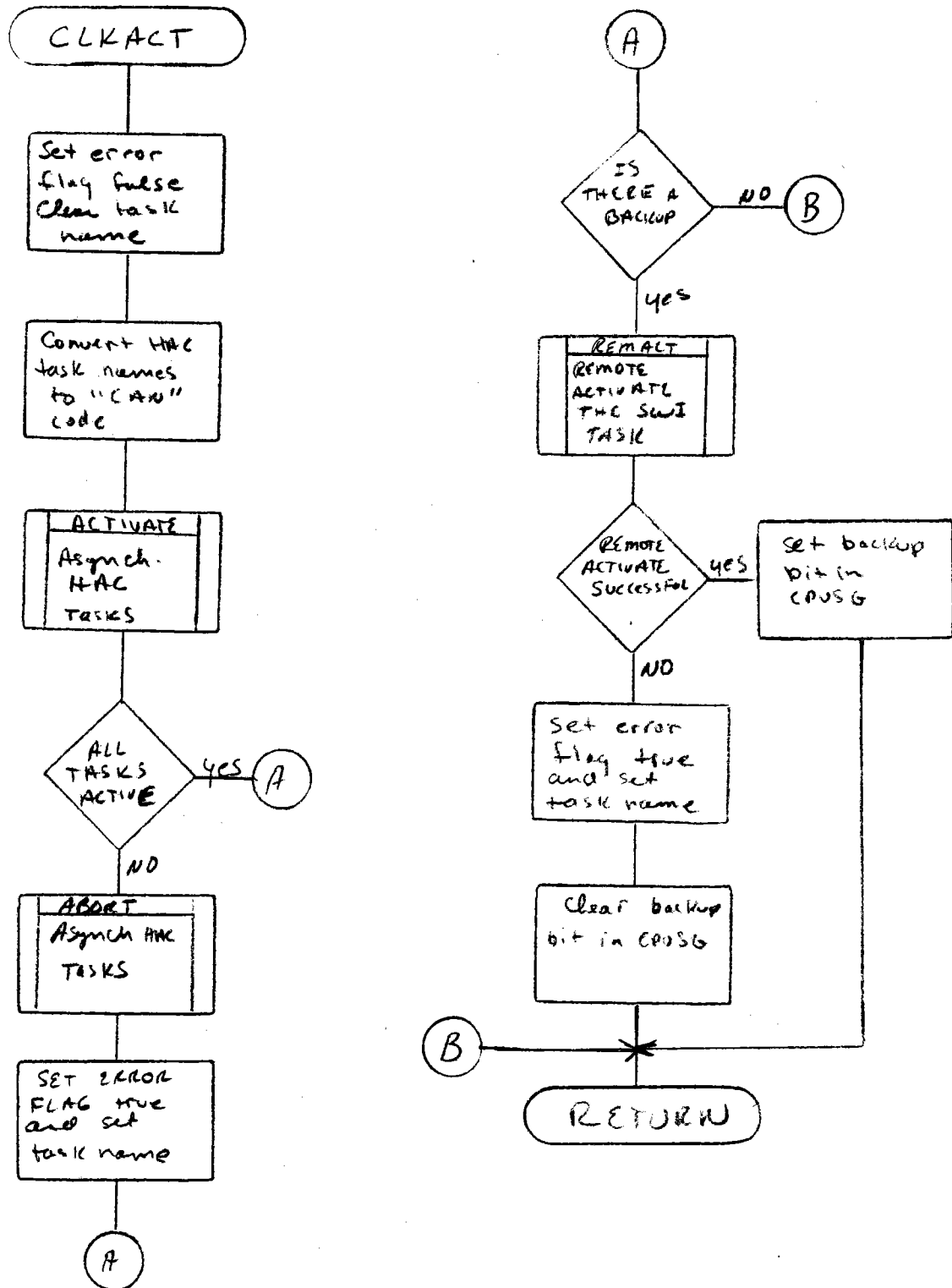


Figure 3.2.7-41 Flowchart - CLKACT

3.2.8 Operating System Modifications Module - MAXIVM

3.2.8.1 Buffer Management Function

Module Identification

The identities of the buffer management modules are:

QINIT
LEASE
FREE
ENQUE
DEQUE
QRST
BMFIF

3.2.8.1.1 Purpose

This group of operating system modules provides two primary user services:

- a. The ability to obtain reusable buffers for short-term use; and
- b. The ability to communicate between tasks, allowing some tasks to be asynchronous and event-driven.

QINIT initializes one or more buffer areas (one call per area) for use by the other services.

LEASE will attempt to acquire a buffer from a buffer area.

FREE will return a LEASE'd buffer to the buffer area pool.

ENQUE will LEASE a buffer and store the given message in it, link it into a queue of messages, and activate the task associated with that queue.

DEQUE will remove the highest item from a queue, pass the message to the caller, and FREE the buffer.

QRST will restart all queues in the event of failover to Backup.

BMFIF is a package which provides FORTRAN access to the above services.

3.2.8.1.2 Requirements

3.2.8.1.2.1 Design Requirements

There are no requirements in the Software/Firmware Functional Requirements document applicable to this function.

3.2.8.1.2.2

Derived Requirements

The ALARMS and other modules require a potentially unlimited number of buffers for message management. The dynamic allocation provided by these modules is deemed the most efficient way of approximating that requirement.

CRT management in previous control systems has been shown to be unwieldy in use. A common message enqueing system is needed to support CRT use satisfactorily.

3.2.8.1.3

Design Approach

3.2.8.1.3.1

Functional Allocations

Basic buffer management (i.e., allocation and deallocation of buffers) is provided by the LEASE and FREE modules. The ENQUE and DEQUE modules provide a moderated message dispatching capability from data sources to data sinks, decoupling their dependence upon each other. They, in turn, call LEASE and FREE to allow dynamic management of their queues.

All services require the initialization of the buffer area provided by QINIT.

3.2.8.1.3.2

Resource Budgets:

	QINIT	LEASE	FREE	ENQUE	DEQUE	QRST	BMFIF
Core requirements: (words)	70	60	40	75	55	50	50
Timing*: (μ secs, nominal estimate)	200	50- 500	100	100- 600	250- 500	100	20/call

*plus system overhead for REX call.

3.2.8.1.4

Design Description

3.2.8.1.4.1

Module Structure

The first seven submodules are stand-alone assemblies designed to be included in the resident portion of the operating system during system generation. They require no other subroutines not provided by the standard operating system itself.

BMFIF will reside in the FORTRAN library.

3.2.8.1.4.1.1

Submodule I - QINIT - Buffer Area Initializer

3.2.8.1.4.1.1.1 Description

- a. Language used - M5A Assembly language.
- b. How invoked - REX call.
- c. Constraints and limitations - No limits on the number or size of areas allowed. The entire area must, however, reside within the caller's address space. If more than one task is to use the area, it must be wholly within a particular global common area.
- d. Processing - The area is initialized for use of the other routines. Queue headers are built, and the lease area cleared.
- e. Error messages and recovery - error indications are returned in the event of invalid input parameters.

3.2.8.1.4.1.1.2 Data, Logic and Command Paths

QINIT must be supplied with the buffer area address, length, and a queue task association table (QTAT). The QTAT associates each queue number (zero to number of queues minus one) with a specific task name.

3.2.8.1.4.1.1.3 Internal Data Description

The initial buffer area configuration is shown in Figure 3.2.8.1.1.

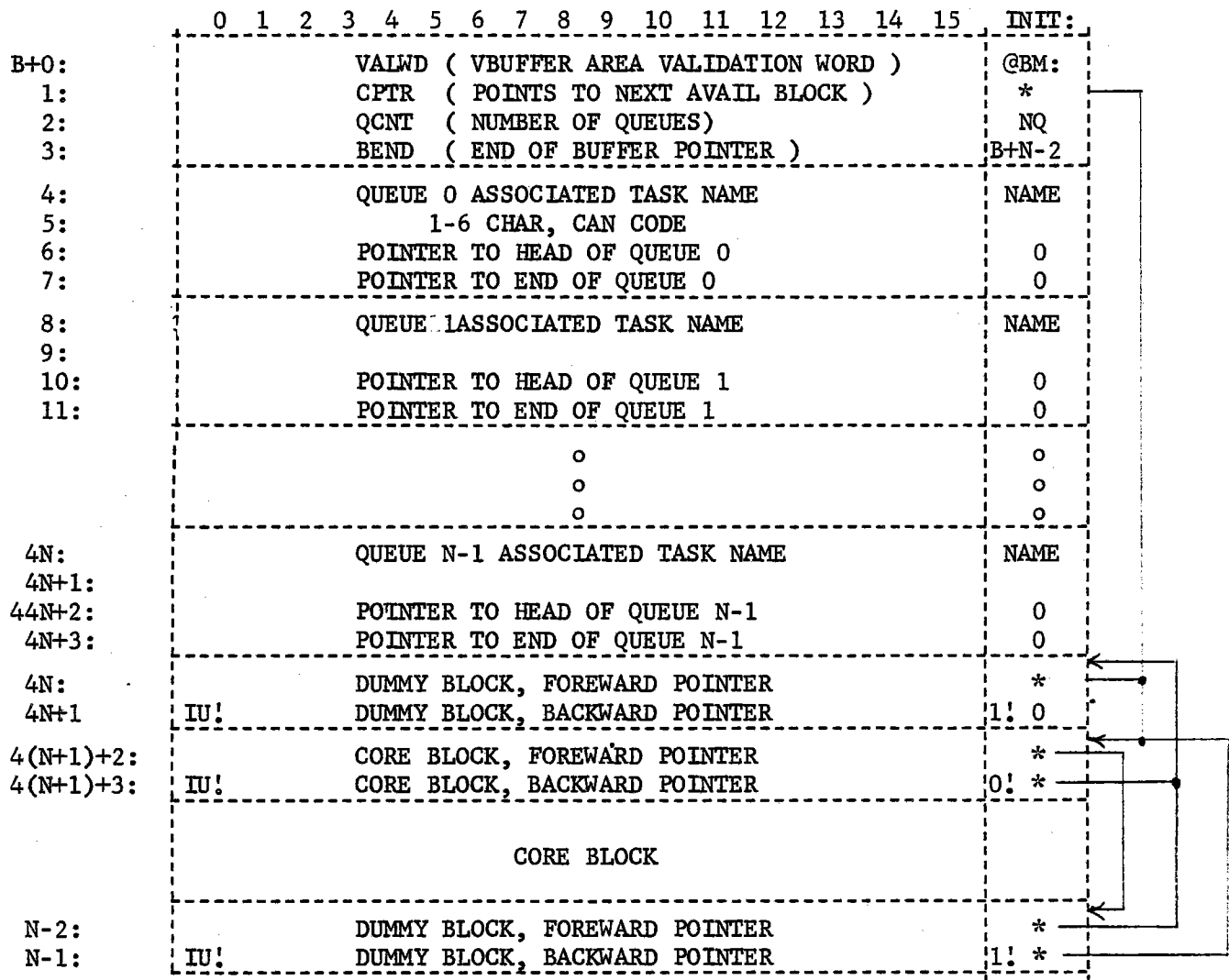
3.2.8.1.4.1.1.4 Flowchart

See Figure 3.2.8.1.2 for the QINIT flowchart.

3.2.8.1.4.1.2 Submodule II - LEASE - Lease a Buffer

3.2.8.1.4.1.2.1 Description

- a. Language used - M5A assembly language
- b. How invoked - REX call.
- c. Constraints and limitations - none
- d. Processing - Beginning from the end of the last leased buffer, the area is searched (with end-of-buffer wrap-around) until a sufficiently large area is found or until the entire area has been searched. If an area is found, it is set in use, any extra being left free. The address is returned to the caller.
- e. Error messages and recovery - error indications are returned for the following conditions:
 - 1. Invalid input parameters.
 - 2. No room for requested buffer.



NOTE: ALL POINTERS ARE ABSOLUTE

Figure 3.2.8.1.1 Internal Buffer Structure

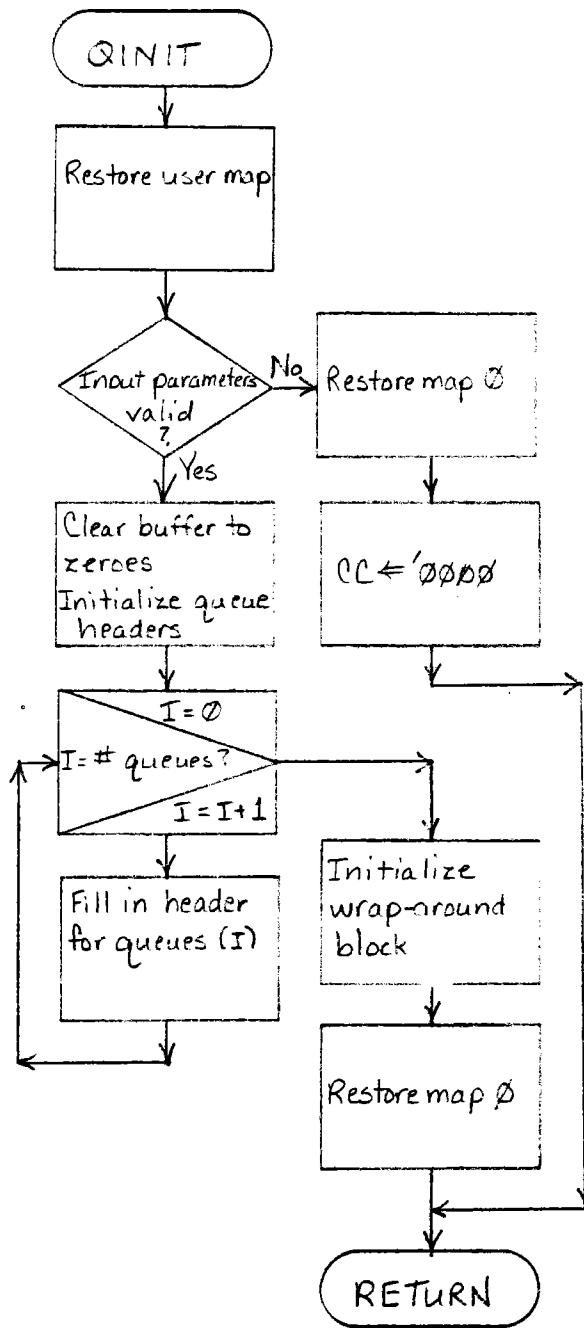


Figure 3.2.8.1.2 Flowchart - QINIT

- 3.2.8.1.4.1.2.2 Data, Logic, and Command Paths
None.
- 3.2.8.1.4.1.2.3 Internal Data Description
LEASE uses the same structure initialized by QINIT (Section 3.2.8.1.4.1.1).
- 3.2.8.1.4.1.2.4 Flowchart
See Figure 3.2.8.1.3 for the LEASE flowchart.
- 3.2.8.1.4.1.3 Submodule III - FREE - Return a LEASE'd Buffer.
- 3.2.8.1.4.1.3.1 Description
- a. Language used - M5A assembly language
 - b. How invoked - REX call
 - c. Constraints and limitations - none
 - d. Processing - FREE forcibly sets the returned block not busy and returns, if it was already in that state. If not, FREE will merge it with any adjacent free blocks. The primary requirement is that at no time should the chain of buffers contain two adjacent free blocks.
 - e. Error messages and recovery - errors are indicated to the user in the event of invalid input parameters, specifically an incorrect returned buffer pointer. No error is indicated if the buffer is found to be already free.
- 3.2.8.1.4.1.3.2 Data, Logic and Command Paths
None.
- 3.2.8.1.4.1.3.3 Internal Data Description
Same as QINIT (Section 3.2.8.1.4.1.2.2)
- 3.2.8.1.4.1.3.4 Flowchart
See Figure 3.2.8.1.4 for the FREE flowchart.
- 3.2.8.1.4.1.4 Submodule IV - ENQUE - Enque a Message and Activate the Associated Task
- 3.2.8.1.4.1.4.1 Description
- a. Language used - M5A assembly language
 - b. How invoked - REX call.

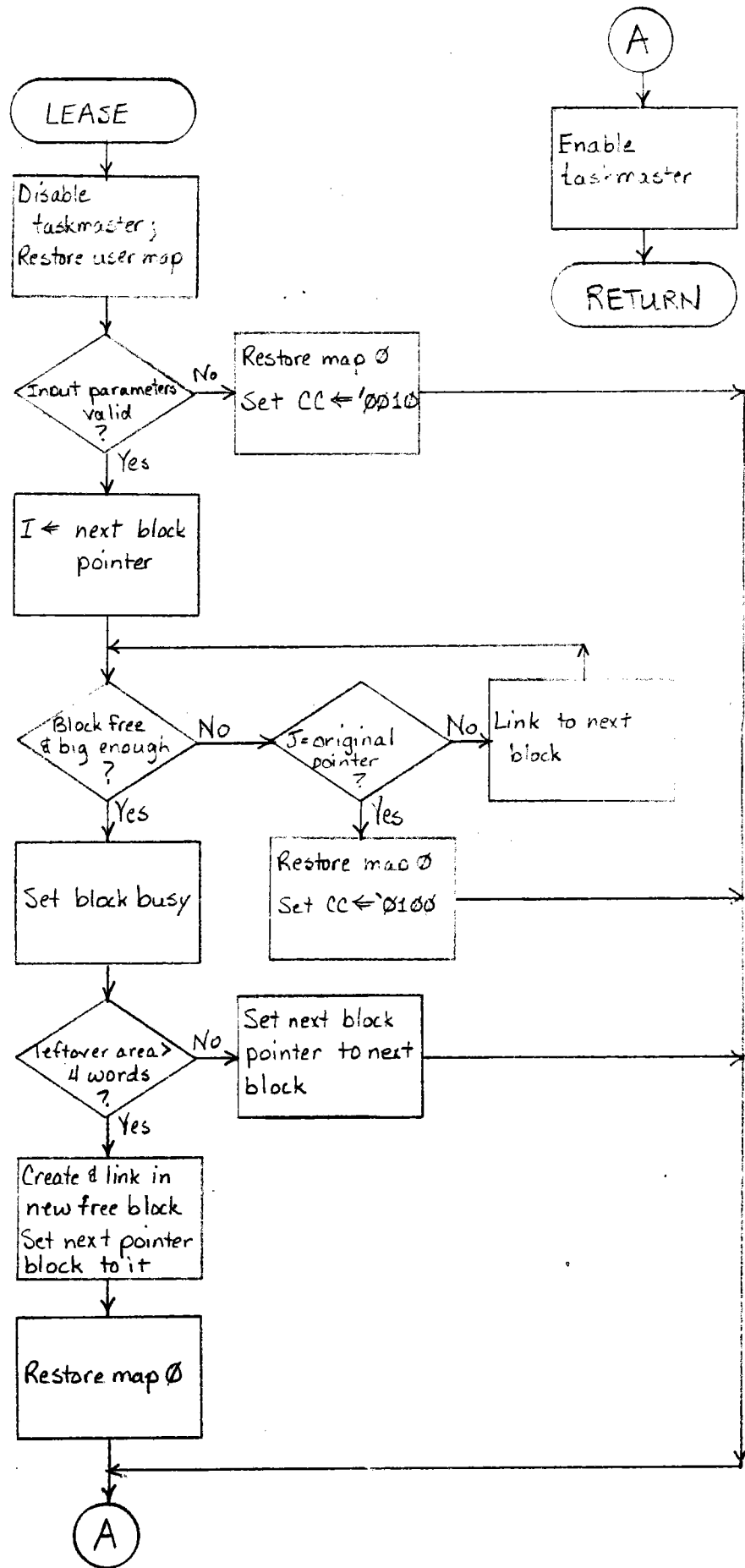


Figure 3.2.8.1.3 Flowchart - LEASE
550

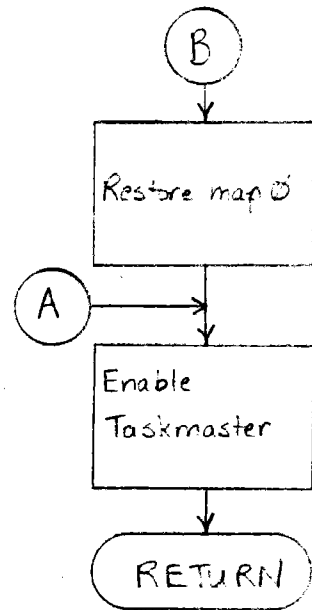
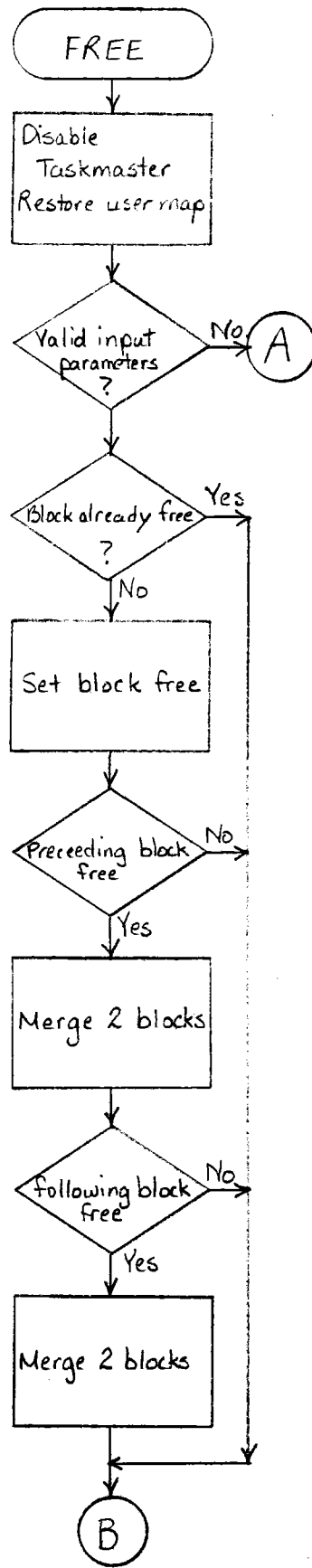


Figure 3.2.8.1.4 Flowchart - FREE

- c. Constraints and limitations - See paragraph 3.2.8.1.4.1.5
- d. Processing - a buffer of sufficient size to hold the input message and some queue control words is leased. The user's buffer contents are copied, and the leased buffer is enqueued into the desired queue. If the queue was previously empty, the associated task is activated, which will presumably DEQUE the message.
- e. Error messages and recovery - errors are indicated for the following conditions:
 - 1. No room for buffer LEASE.
 - 2. Invalid input parameter.
 - 3. Influence error (from ACTIVATE).
 - 4. Task not found (from ACTIVATE).
 - 5. No TCB available (from ACTIVATE).

3.2.8.1.4.1.4.2 Data, Logic and Command Paths

None.

3.2.8.1.4.1.4.3 Internal Data Description

The leased buffer is set up in the format shown in Figure 3.2.8.1.5. Null links, in queue items or the queue data areas, are maintained as zeros.

3.2.8.1.4.1.4.4 Flowcharts

See Figure 3.2.8.1.6 for the ENQUE flowchart.

3.2.8.1.4.1.5 Submodule V - DEQUE - Get the Oldest Queued Item

3.2.8.1.4.1.5.1 Description

- a. Language used - M5A Assembly code.
- b. How invoked - REX call.
- c. Constraints and limitations - ENQUE and DEQUE work in concert, attempting to empty queues by activating the tasks which use the queued data. Jammed queues are avoided as long as all the queues which address the same associated task are in the same buffer area.
- d. Processing - If the indicated queue is empty, return is made immediately. The foremost item in the queue is unlinked, and its contents transferred to the caller.

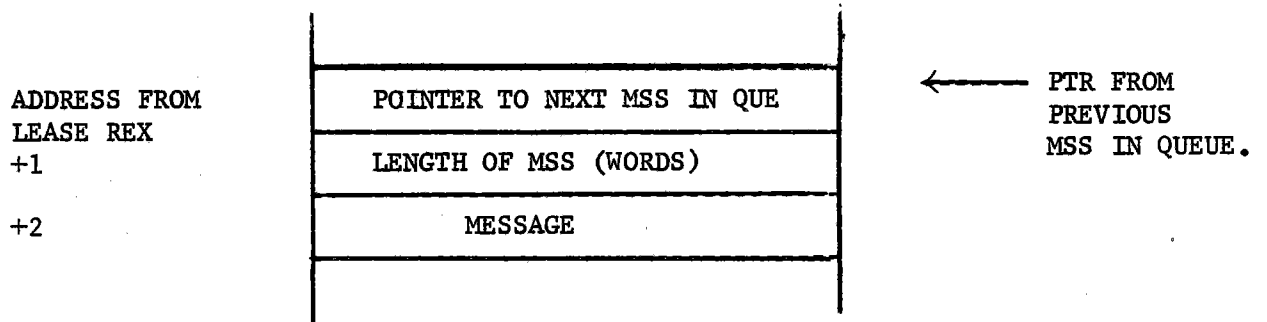


Figure 3.2.8.1.5 Message Structure

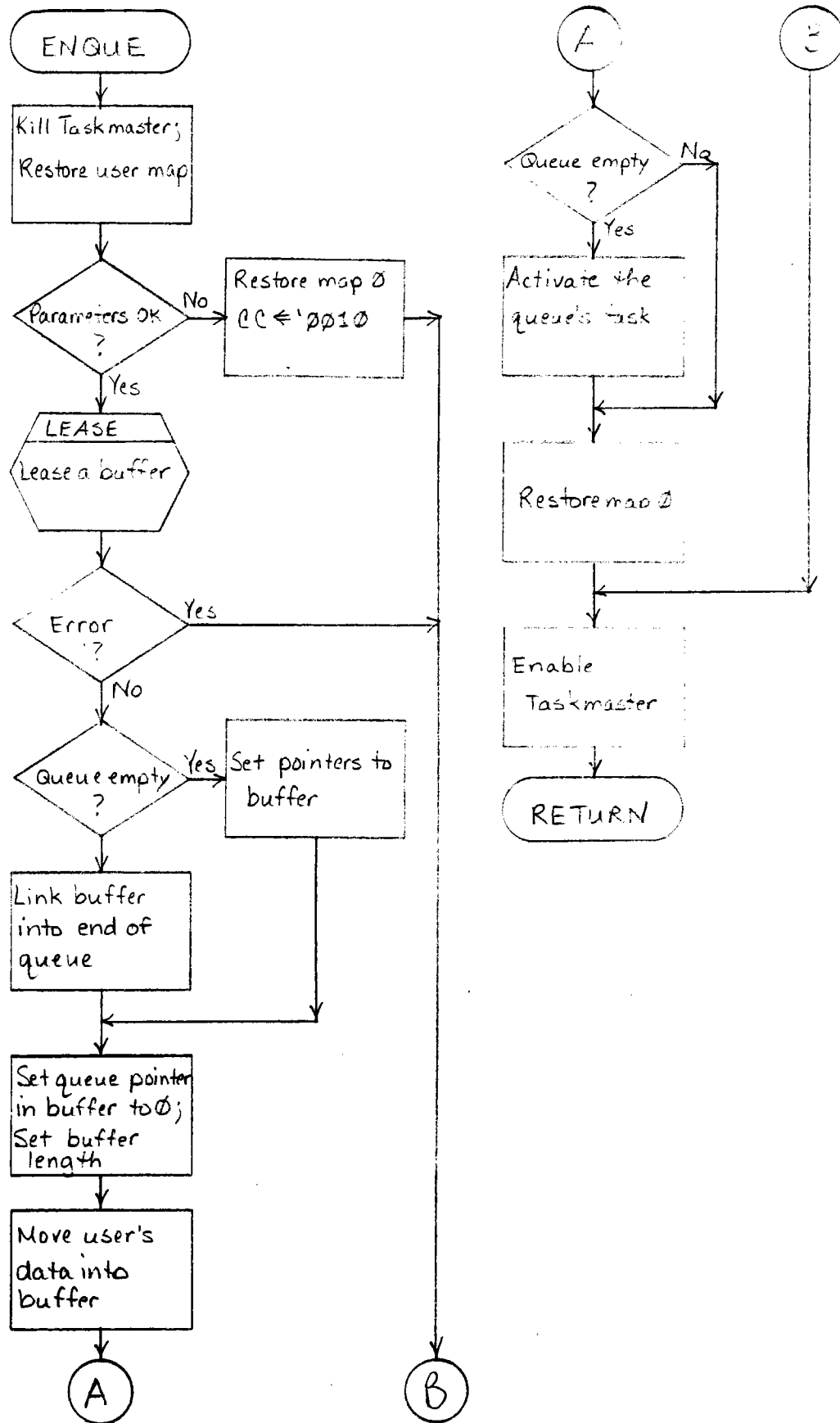


Figure 3.2.8.1.6 Flowchart - ENQUE

The area is then FREE'd. If the queue is not empty following this removal, an ACTIVATE of the associated task is done and return made. If the queue is empty, a search is made for any other non-empty queues with the same associated task. If any are found, an ACTIVATE is performed.

e. Error messages and recovery - errors are indicated for the following conditions:

1. Queue requested was empty.
2. Input parameter invalid.
3. Influence error (from ACTIVATE).
4. Task not found (from ACTIVATE).
5. No TCB available (from ACTIVATE).

3.2.8.1.4.1.5.2 Data, Logic and Command Paths

None.

3.2.8.1.4.1.5.3 Internal Data Description

The internal structure is the queue, described in 3.2.8.1.4.1.4.3.

3.2.8.1.4.1.5.4 Flowchart

See Figure 3.2.8.1-7 for the DEQUE flowchart.

3.2.8.1.4.1.6 Submodule VI - QRST - Restore Queue Activations

3.2.8.1.4.1.6.1 Description

- a. Language used - M5A assembly language
- b. How invoked - REX call
- c. Constraints and limitations - may cause spurious activations of queues which did not require restart. All tasks which DEQUE items should check for zero length returns, indicating nothing left in the queue, and exit as a result.
- d. Processing - QRST will scan the queues within a buffer area, activating all tasks whose queues are non-empty.
- e. Error messages and recovery - errors returned include invalid input parameters, and the activation errors: no TCB available; influence limit error; and task not found.

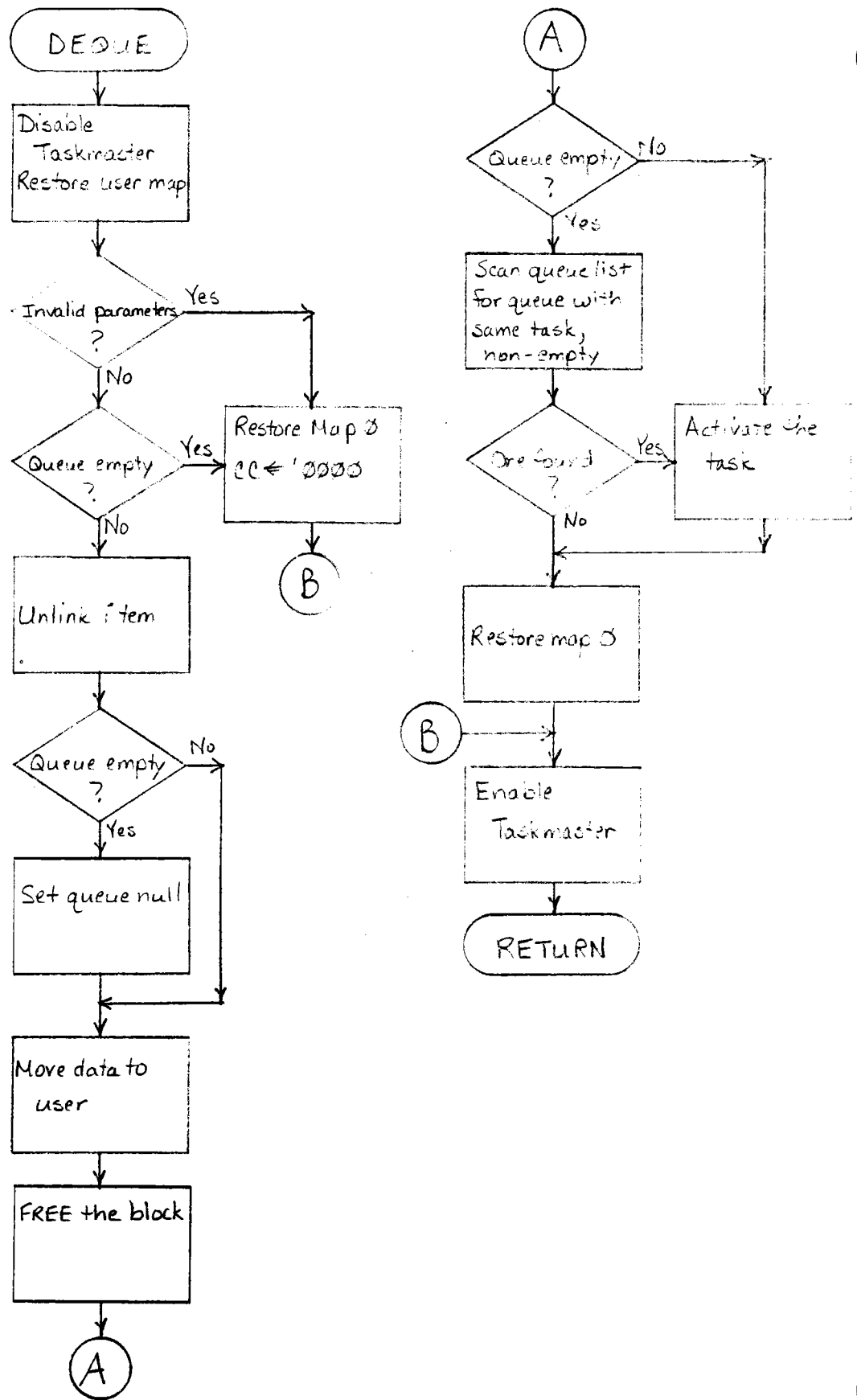


Figure 3.2.8.1.7 Flowchart - DEQUE

- 3.2.8.1.4.1.6.2 Data, Logic and Command Paths
Data structures as discussed in QINIT call.
- 3.2.8.1.4.1.6.3 Internal Data Description
None
- 3.2.8.1.4.1.6.4 Flowchart
See Figure 3.2.8.1.8 for the QRST flowchart.
- 3.2.8.1.4.1.7 Submodule VII - BMFIF - FORTRAN Interface to Buffer Management
- 3.2.8.1.4.1.7.1 Description
- a. Language used - M5A assembly language
 - b. How invoked - FORTRAN calls
 - c. Constraints and limitations - None
 - d. Processing - For each of the entry points, the parameters are gathered and a REX call to the appropriate function made. Upon return, parameters are passed and return made to the caller.
 - e. Error messages and recovery - The FORTRAN routine error package is invoked when unexpected errors occur. These return a "FER" error code, with subcode "QXn", where "n" indicates the error:
 - n = 0 - Task not found
 - n = 1 - Influence error
 - n = 2 - Parameter error
 - n = 3 - Buffer area invalid
 - n = 4 - No room available
 - n = 5 -
 - n = 6 -
 - n = 7 - No TCB available
- 3.2.8.1.4.1.7.2 Data, Logic and Command Paths
None
- 3.2.8.1.4.1.7.3 Internal Data Description
None
- 3.2.8.1.4.1.7.4 Flowchart
See Figure 3.2.8.1.9 for the BMFIF flowchart.
- 3.2.8.1.5 Interface Description
- 3.2.8.1.5.1 QINIT
See Figure 3.2.8.1.10.

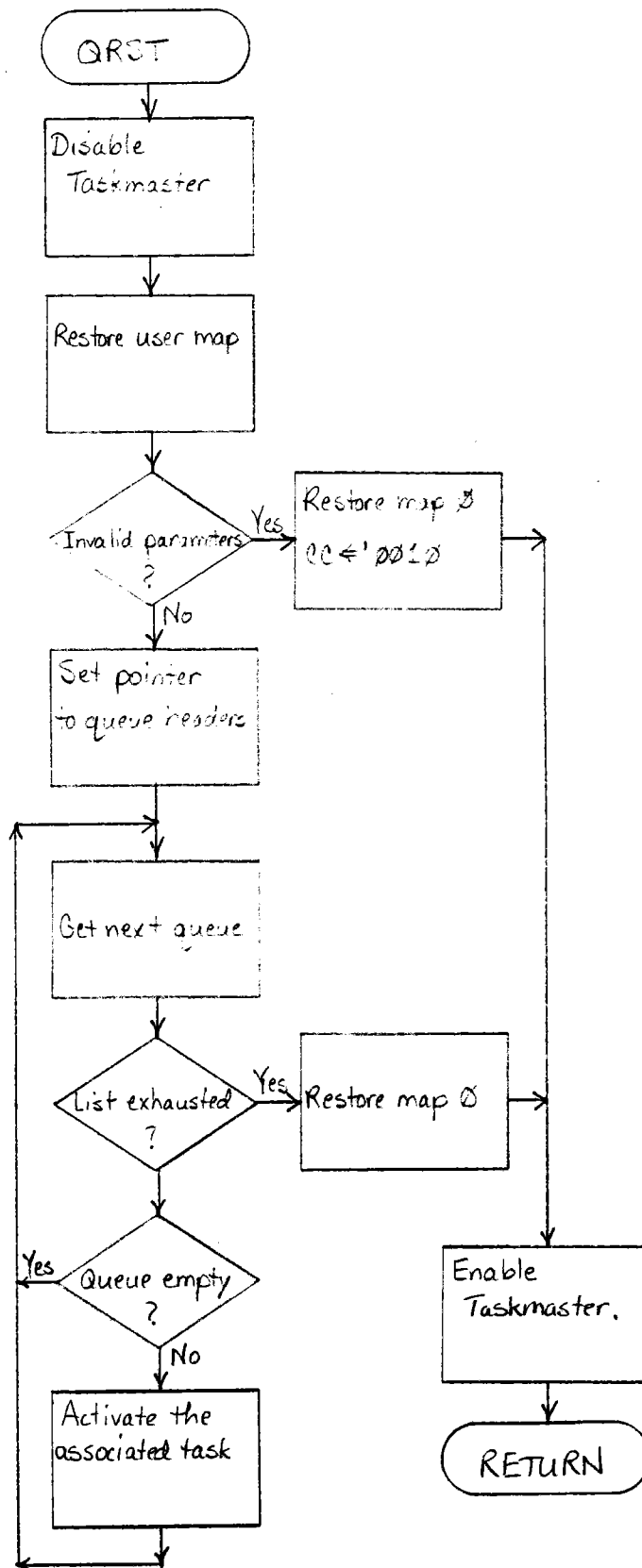


Figure 3.2.8.1.8 Flowchart - QRST

BMFIF

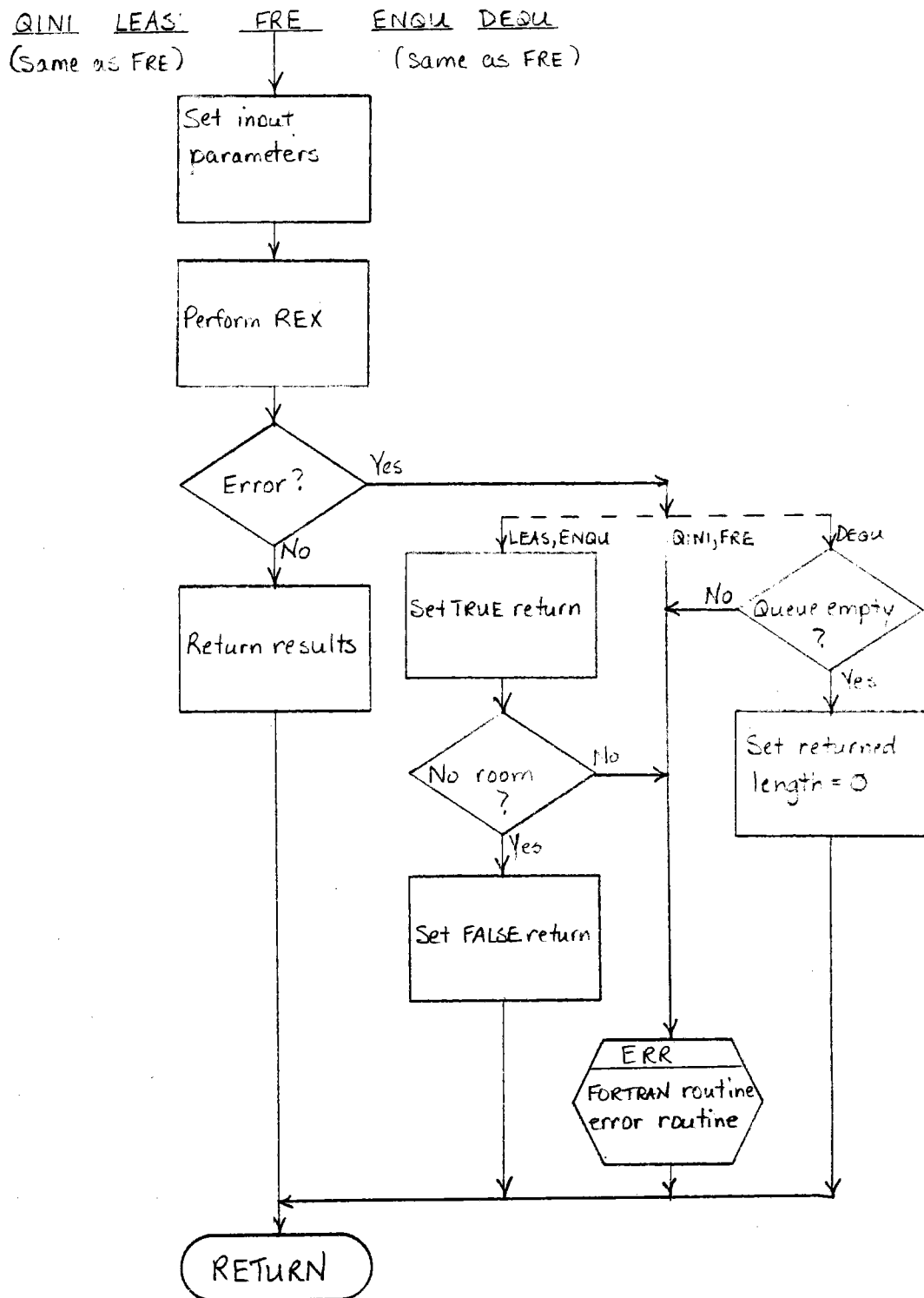


Figure 3.2.8.1.9 Flowchart - BMFIF

R15 - LENGTH OF THE COMMON BUFFER AREA IN WORDS (N);
 R14 - ADDRESS OF THE BUFFER AREA;
 R13 - ADDRESS OF THE QUEUE-TASK ASSOCIATION TABLE (SEE BELOW);
 R12 - NUMBER OF QUEUES TO BE SET UP IN THE BUFFER AREA (NQ)

$$N > = (NQ * 4) + 9;$$

LDI,R8 QINIT(#6A)
 REX,MAXIV
 RETURNED CONDITION CODES:

NZOC = '1000 -> NO ERROR, NORMAL RETURN.
 NZOC = '0010 -> PARAMETER ERROR.

THE QUEUE-TASK ASSOCIATION TABLE ASSOCIATES THE QUEUE NUMBERS (FROM 0 THROUGH NQ-1) TO SPECIFIC TASKS WHICH WILL BE ACTIVATED BY NAME WHEN A MESSAGE IS ENQUEUED IN THE ASSOCIATED QUEUE. IT HAS THE FORMAT:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0: 1:	TASK NAME ASSOCIATED WITH QUEUE 0 1-6 CHARS, CAN CODE															
2: 3:	TASK NAME ASSOCIATED WITH QUEUE 1															
.	.															
.	.															
.	.															
(NQ-1)*2: (NQ-1)*2+1:	TASK NAME ASSOCIATED WITH QUEUE N-1															

Figure 3.2.8.1.10 Interface Specification for QINIT

3.2.8.1.5.2 LEASE
See Figure 3.2.8.1.11.

3.2.8.1.5.3 FREE
See Figure 3.2.8.1.12.

3.2.8.1.5.4 ENQUE
See Figure 3.2.8.1.13.

3.2.8.1.5.5 DEQUE
See Figure 3.2.8.1.14.

3.2.8.1.5.6 QRST
See Figure 3.2.8.1.15.

3.2.8.1.5.7 BMFIF
See Figure 3.2.8.1.16.

3.2.8.1.6 Test Requirements

Testing will be performed with test drivers and by data structure inspection. Since the number of cases to be tested is relatively limited, this is deemed sufficient.

R14 - LOCATION OF THE BUFFER AREA, AS PASSED TO QINIT.

R15 - SIZE OF BUFFER REQUIRED (IN WORDS).

LDI,8 LEASE (#6B)

REX,MAXIV

R14 - ABSOLUTE ADDRESS OF THE BUFFER.

CONDITION CODES AS FOLLOWS:

NZOC = '0010 ->	PARAMETER ERROR.
NZOC = '0100 ->	NO ROOM LEFT IN THE BUFFER AREA.
NZOC = '1000 ->	NO ERROR, NORMAL RETURN.

Figure 3.2.8.1.11 Interface Specification for LEASE

R14 - BUFFER AREA ADDRESS, AS PASSED TO QINIT.

R15 - ADDRESS OF THE BLOCK TO BE FREED (AS GIVEN BY LEASE)

LDI,8 FREE (#6C)

REX,MAXIV,32

RETURNED CONDITION CODES:

NZOC = '1000 -> NO ERROR.
NZOC = '0010 -> PARAMETER ERROR

Figure 3.2.8.1.12 Interface Specification for FREE

R12 - WORD ADDRESS OF BUFFER CONTAINING MESSAGE.
R13 - QUEUE NUMBER (0 = NQ-1) FOR DESIRED QUEUE.
R14 - BUFFER AREA ADDRESS, AS PROVIDED TO QINIT.
R15 - LENGTH OF MESSAGE TO BE ENQUED, IN WORDS.

LDI,8 ENQUE (#6D)

REX,MAXIV

RETURNED CONDITION CODES ARE:

NZOC = '1000	->	NORMAL RETURN.
NZOC = '0100	->	NO ROOM FOR MESSAGE IN BUFFER AREA.
NZOC = '0010	->	PARAMETER ERROR.
NZOC = '0001	->	INFLUENCE ERROR (SEE ACTIVATE).
NZOC = '0000	->	TASK NOT FOUND.
NZOC = '0111	->	NO TCB AVAILABLE.

Figure 3.2.8.1.13 Interface Specification for ENQUE

R13 - QUEUE NUMBER (0 - NQ-1).

R14 - BUFFER AREA ADDRESS, AS PASSED TO QINIT.

R15 - USER'S BUFFER ADDRESS TO RECEIVE MESSAGE.

LDI,8 DEQUE (#68)

REX,MAXIV

R14 - RETURNS LENGTH OF THE MESSAGE (IN WORDS).

CONDITION CODES ON RETURN:

NZOC = '1000	->	NO ERROR, NORMAL RETURN.
NZOC = '0100	->	QUEUE WAS EMPTY.
NZOC = '0010	->	PARAMETER ERROR.
NZOC = '0001	->	INFLUENCE ERROR.
NZOC = '0000	->	TASK NOT FOUND.
NZOC = '0111	->	NO TCB AVAILABLE.

Figure 3.2.8.1.14 Interface Specification for DEQUE

R13 - QUEUE NUMBER (0 - N-1)

R14 - BUFFER AREA ADDRESS, AS PASSED TO QINIT.

LDI,8 QRST (#6F)

REX,MAXIV

CONDITION CODES ON RETURN:

NZOC = '1000 ->	NO ERROR, NORMAL RETURN.
NZOC = '0010 ->	PARAMETER ERROR.
NZOC = '0001 ->	INFLUENCE ERROR.
NZOC = '0000 ->	TASK NOT FOUND.
NZOC = '0111 ->	NO TCB AVAILABLE.

Figure 3.2.8.1.15 Interface Specification for QRST

CALL QINI (BUFFER, ISIZE, NQ, ITAB)

where:

BUFFER - the area to be used for buffer management;
ISIZE - length (in words) of the area;
NQ - number of queues to be managed;
ITAB - an INTEGER*4 table containing a six-character
task name (in CAN code) in each 4-byte entry.
The length of ITAB is NQ.

LOGICL = LEAS (BUFFER, NEEDSZ, BUFRTD)

where:

BUFFER - same as above;
NEEDSZ - required buffer size, in words;
BUFRTD - the returned subscript of the buffer leased
within BUFFER;
LOGICL - .TRUE. if lease was successful.

CALL FRE (BUFFER, BUFRTD)

where both are as described in LEAS.

LOGICL = ENQU (BUFFER, LOCBUF, ISIZE, IQ)

where:

BUFFER - as above;
LOCBUF - address of message to be enqueued;
ISIZE - length (in words) of LOCBUF;
IQ - queue number ($0 \leq IQ \leq NQ$) into which to enqueue
the message;
LOGICL - .TRUE. if successful.

CALL DEQU (BUFFER, LOCBUF, KSIZE, IQ)

where:

BUFFER - as above;
LOCBUF - buffer to receive the next message;
KSIZE - returned length of the message passed (zero
if nothing in queue);
IQ - the queue number from which to get the next
message.

Figure 3.2.8.1.16 Interface Specification for BMFIF

3.2.8.2 "Clock" Function

3.2.8.2.1 Purpose

This function will maintain a stable and accurate universal base within the global common area of memory. This (global common) time base may be accessed by any of the various tasks which are components of the heliostat system.

3.2.8.2.2 Requirements

3.2.8.2.2.1 Design Requirements

Section 3.1 of the 10 MWe Software/Firmware Functional Requirements Specification states the following requirement for the "Clock" function:

Maintain a stable time base.

Inasmuch as the apparent position of the sun is an important assumption of other requirements in the Software/Firmware FRS, it is assumed that a "stable time base" means an accurate representation of UTC (Universal Coordinated Time, same as GMT).

3.2.8.2.2.2 Derived Requirements

Section 3.2.1.8 of the 10 MWe Software/Firmware Functional Requirements Specification derives the following requirement:

Universal time maintenance, using the WWVB Clock as a reference standard.

In order to fulfill both of the Software/Firmware Functional Requirements Specification requirements, we derive the following:

- a. The primary requirement is to provide accurate maintenance of the time data base;
- b. This module, in order to assure accuracy, will execute at the highest software priority level allocated to heliostat system tasks;
- c. This module must be capable of detecting and reporting input errors which may be generated by the WWV device or the hardware which interfaces with that device;
- d. In the event of a WWV device error or failure, this module must be capable of maintaining (unassisted) a stable and accurate time base;

- e. In the event of a WWV device error or failure, the module must allow the operator, after correcting the problem, to place the device back on line;
- f. This module is, by nature, time critical and, therefore must be coded as efficiently as possible in order to maintain accuracy and minimize system overhead; and
- g. Local time (i.e., zone time at the operational location) must be maintained.

3.2.8.2.3 Design Approach

3.2.8.2.3.1 Functional Allocations

The "Clock" function consists of two component tasks:

- a. Task TOK, which provides/ensures alignment of system and WWV time; and
- b. Task TIK, which provides maintenance of global common time values and triggers the FCP task into execution each second.

3.2.8.2.3.2 Resource Budget

This program, when executing, will require use of the following system resources:

- a. Software priority levels: TIK - Very high
TOK - First priority below synchronous loop tasks
- b. Core/solid state memory: 400₁₀ words (approximate)
- c. Asynchronous input device: 1 Model 4811 (1 subchannel)
- d. WWV receiver: 1

3.2.8.2.4 Design Description

3.2.8.2.4.1 Module Structure

This function consists of two component tasks, the interaction of which is graphically represented in Figure 3.2.8.2.1

3.2.8.2.4.1.1 TOK - Major Interval Timer

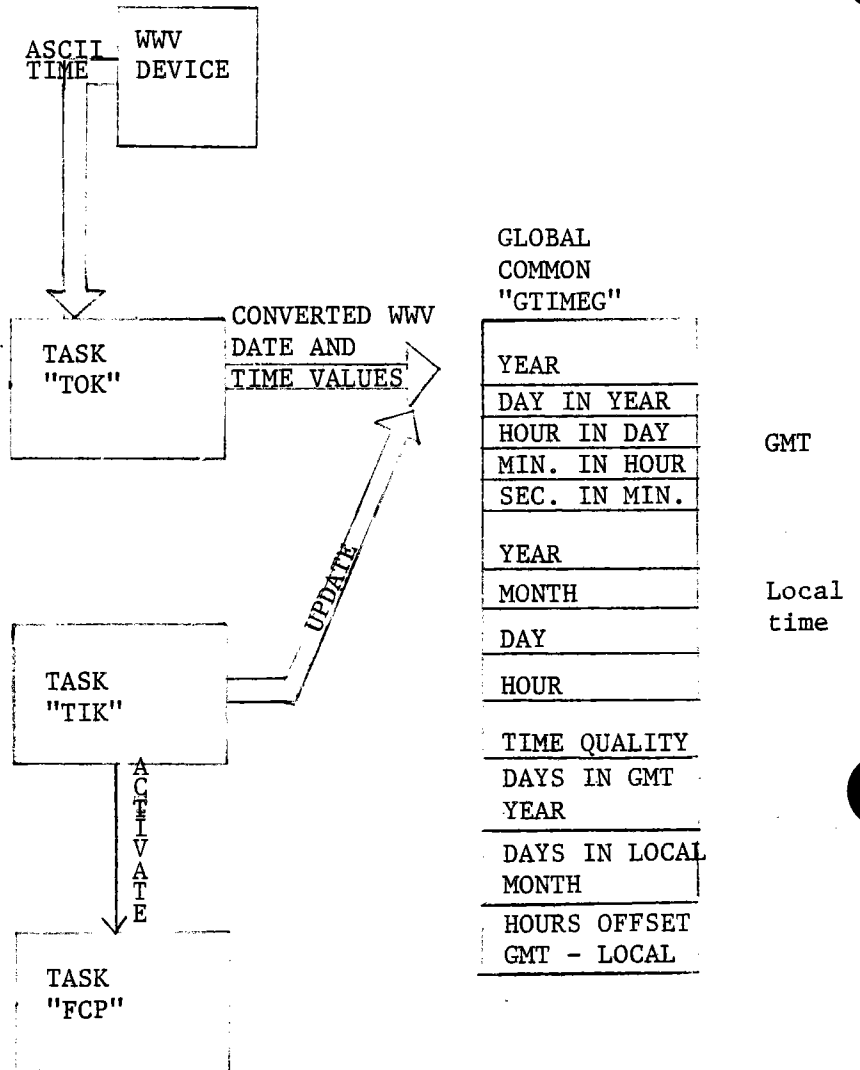
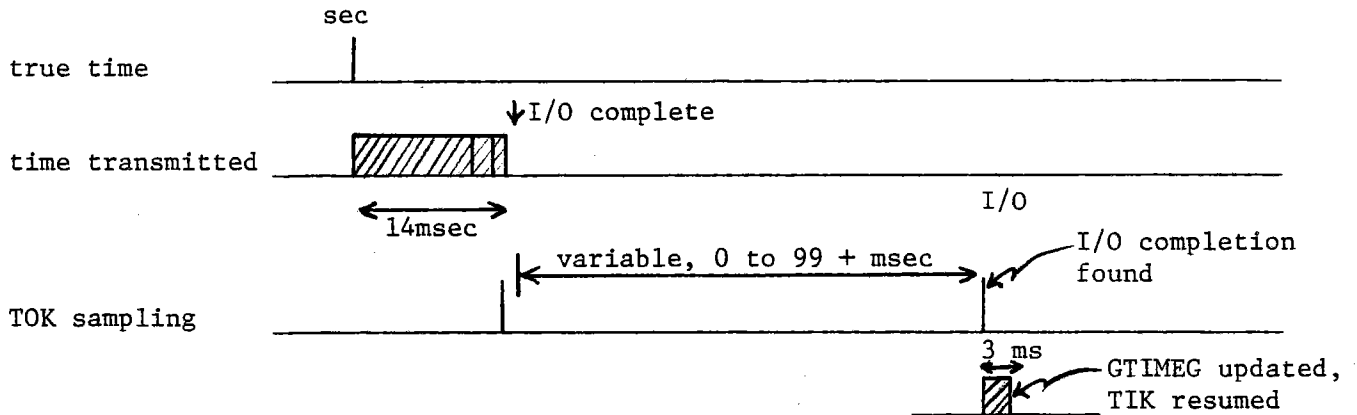


Figure 3.2.8.2.1 Clock Module, Component Task Interaction

3.2.8.2.4.1.1.1 Description

- a. Language used - MODCOMP MAC assembly language with MAX IV REX calls.
- b. How invoked - Activated by CLKONL (in DBINIT module).
- c. Constraints and limitations - NONE
- d. Processing - The major timer synchronizes the data base time to the WWV receiver time. Once a minute, TOK is reactivated, reads the WWV receiver (if it is on line), resumed TIK, updates the data base, and resets itself for execution again in one minute. An alarm is issued in the event that the time quality factor received from the WWV device changes.

TIK & TOK interact in such a way that if the WWV device is operating correctly, the time in GTIMEG at any instant is accurate to within the limits (+10.0, + 150 msec), excluding the WWV device quality error limit (specified below). This accuracy is made possible by the fact the TRU-TIME WWV receiver transmits the time to the CPU on the second. TOK opens a read, then checks 100msec intervals until the IO is complete. It then resumes TIK and communicates that fact to TIK via a bit (TIKTOK) in CPUSG. TIK, immediately reschedules itself one second later, and is now synchronized. A timing diagram of the interaction shows the best/worst case:



$$\text{worst case} = 14 + 99 + 3 = 116\text{msec}$$

$$\text{best case} = 14 + 3 = 17\text{msec}$$

If at anytime the WWV device becomes inoperable, an alarm is issued and timekeeping becomes internal. The device is still read each minute, however, and if the device becomes operable, will revert to normal operation.

- e. Error messages and recovery - None besides internal timekeeping.

3.2.8.2.4.1.1.2 Data, Logic and Command Paths

The useful output of TOK is the global common array GTIMEG. It is an integer array of the form:

<u>Address</u>	<u>Data</u>	<u>System</u>	<u>Range</u>
GTIMEG + 0	Year	GMT	1980 - ∞
+ 1	Day-of-year	GMT	1-366
+ 2	Hour	GMT	0-23
+ 3	Minute	GMT	0-59
+ 4	Second	GMT	0-59
+ 5	Year	Local	1980 - ∞
+ 6	Month	Local	1-12
+ 7	Day-of-Month	Local	1-31
+ 8	Hour	Local	0-23
+ 9	Time Quality	--	0-5*
+10	Days in GMT year	--	365-366
+11	Day in Local Month	--	28-31
+12	Hours offset GMT-local	--	0-23
+13	Spare		

NOTE: GMT is Universal Coordinated Time. Local time is local zone time, offset from GMT by an integer number of hours.

In addition, bit two of TBUSYG is set to one when TOK is executing and reset at exit. The TIKTOK bit (0) in CPUSG facilitates TIK-TOK communications.

*Values of indicator are:

<u>Value</u>	<u>Possible Error</u>
0	.001 sec
1	.001 sec
2	.005 sec
3	.050 sec
4	.500 sec
5	time internal

3.2.8.2.4.1.1.3 Internal Data Description

The only internal data items are a variable which maintains the status (up/down) of the WWV device, and the WWV buffer, which receives the time from the receiver as an ASCII string of the form:

<CTLA> DDD:HH:MM:SSQ <CR><LF>

where <CTLA> is Hex 1, <CR> is Hex D, and <LF> is Hex A.
"Q" is a time quality indication character.

3.2.8.2.4.1.1.4 Flowcharts

See Figure 3.2.8.2.2 for the TOK flowchart.

3.2.8.2.4.1.2 TIK - Minor Interval Timer

3.2.8.2.4.1.2.1 Description

- a. Language used - FORTRAN
- b. How invoked - Activated by CLKONL (in DBINIT module).
- c. Constraints and limitations - None
- d. Processing - TIK is resumed once a second by a MAX IV delay REX. Upon resumption, time in GTIMEG is advanced once second. If FCP is inactive, it is activated.
- e. Error messages and recovery - If TBUSY6 bit is on for FCI, then a message "FCI BIT ON IN TBUSY6" is printed on the console.

3.2.8.2.4.1.2.2 Data, Logic and Command Paths

See Section 3.2.8.2.4.2.2 for description of the global data base affected by TIK.

Bit three of TBUSYG is used to indicate TIK activity, bit 0 in CPUSG indicates that TOK forceably resumed TIK.

3.2.8.2.4.1.2.3 Internal Data Description

There is no data internal to this function.

3.2.8.2.4.1.2.4 Flowchart

See Figure 3.2.8.2.3 for the TIK flowchart.

3.2.8.2.5 Interface Description

TIK and TOK have a single hardware interface (the WWV receiver) and several software interfaces (the GTIMEG array, TBUSYG bits, and alarms). All have been detailed above.

Local time accuracy, as reflected in GTIMEG, presumes that GTIMEG has been initialized correctly at some time previous.

Time is maintained to well within ± 1 second as long as the WWV device is available (see processing above). On internal mode timekeeping, system clock drift (guaranteed to within 45 seconds per day, or about .05%) may occur.

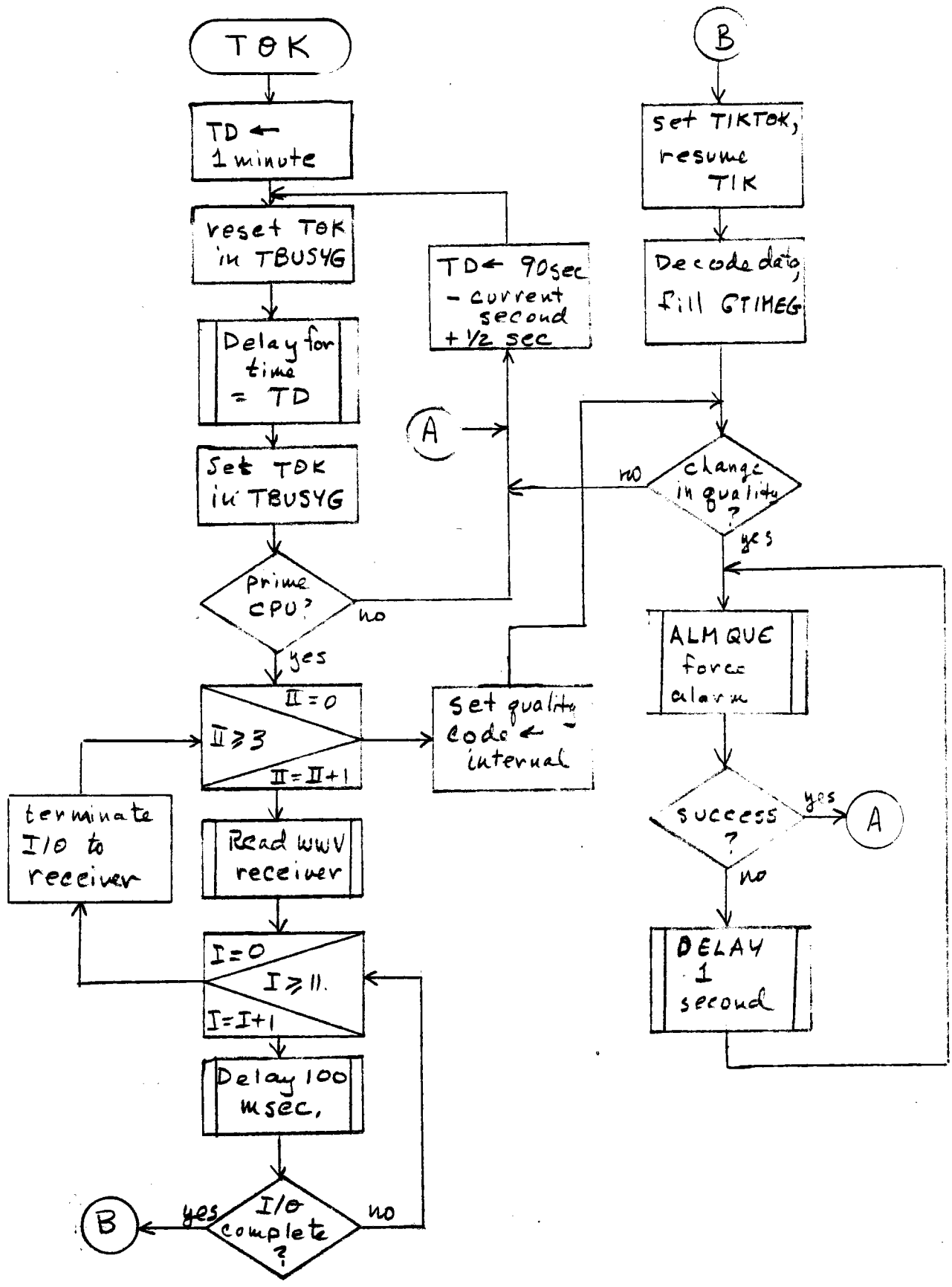


Figure 3.2.8.2.2 Flowchart - TOK

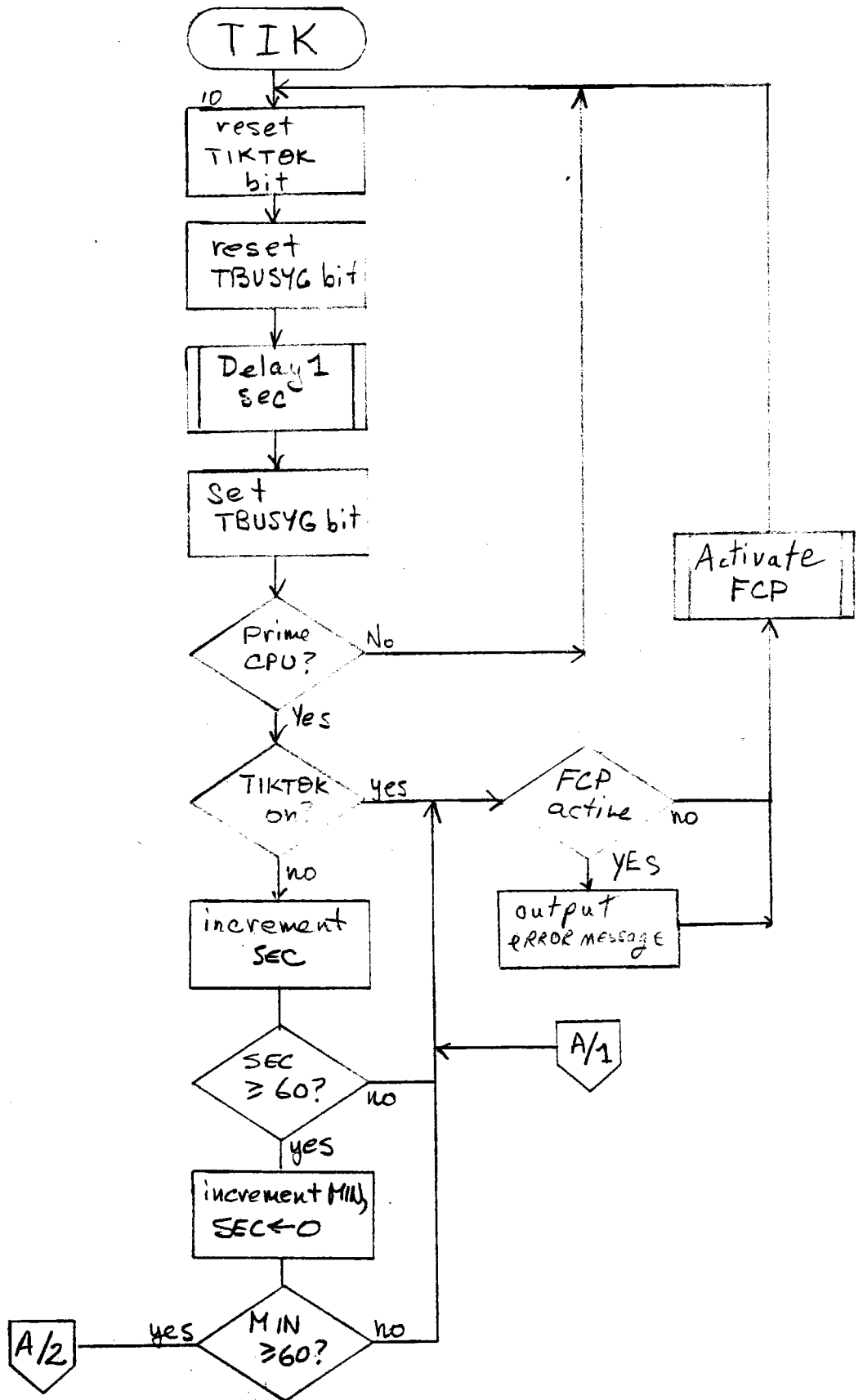


Figure 3.2.8.2.3 Flowchart - TIK
575

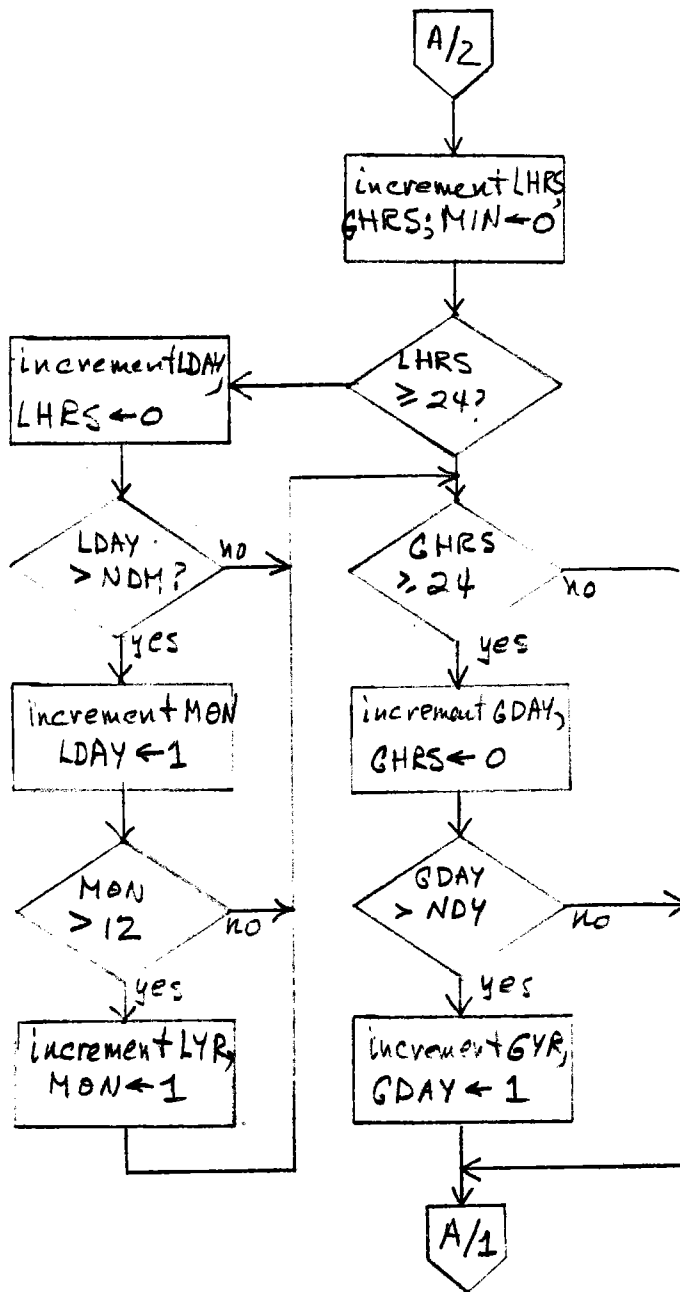


Figure 3.2.8.2.3 Flowchart - TIK (Cont'd)

3.2.8.2.6 Test Requirements

A special test program, THACK, will alias as FCP (triggered by TIK) and allow the current values of GTIMEG to be displayed for comparison to the WWV clock.

Testing TIK and TOK should be for an extended period (approximately 12 hours) on both computers, both with and without the WWV receiver in operation.

3.2.8.3 Receiver Trip

3.2.8.3.1 Purpose

The Receiver Trip function will keep a bit in the global data base (herein identified as the receiver trip bit (RTBIT)) updated, reflecting the status of the receiver trip signal. It will also alert MMI whenever the signal is raised.

3.2.8.3.2 Requirements

3.2.8.3.2.1 Design Requirements

Paragraph 3.3.1.1.5 of the Software/Firmware Functional Requirements Specification states:

"The HAC receives two signals (one the inverse of the other) from the Receiver System. One signal causes an emergency Defocus action; its inverse will allow an operator Defocus-Release action."

3.2.8.3.2.2 Derived Requirements

Assume that the signals mentioned are TTL compatible logic levels, one indicating (when high) that the Defocus action is required; the other (when high) that the release is possible.

The Receiver Trip module shall keep the RTBIT bit current, thus allowing the command interface to validate any Defocus Release command. Receiver Trip shall also inform MMI explicitly of the signal transition of receiver trip to high state (force Defocus).

The signals indicating receiver trip and its inverse will probably be noisy and subject to contact bounce; therefore, a signal will not be said to exist unless it is still detected .5 seconds after it is first sensed.

3.2.8.3.3 Design Approach

3.2.8.3.3.1 Functional Allocations

There will be three parts to the receiver trip module : two tasks and an initialization routine. Each of the two signals will be wired to separate external interrupt levels in the MODCOMP CPU (levels #8 & #9). The initialization routine (further described in the DBINIT module) will CONNECT the interrupt levels to the handlers, enable the interrupts, check for proper state initialization, and report trip status.

3.2.8.3.3.2 Resource Budgets

Priority Level - 30 (RTH)
- 30 (RTL)
Memory - 200 words
Timing - less than 500 microseconds/activation
Hardware Interrupts- 2

3.2.8.3.4 Design Description

3.2.8.3.4.1 Module Structure

The Receiver Trip module consists of two interrupt activated submodules. One submodule is activated when the receiver trips and the other is activated when the receiver returns to normal.

3.2.8.3.4.1.1 Submodule I - RTH

3.2.8.3.4.1.1.1 Description

- a. Language used - FORTRAN
- b. How invoked - When interrupt occurs on hardware level #8.
- c. Constraints and limitations - The receiver signals that are wired to the MODCOMP interrupt levels must be opposite for tasks RTH and RTL to successfully operate. If they are both high, the system will "hang" in a high priority loop. If they are both low, neither RTH or RTL will be activated.

The R/T signal must persist for at least .5 seconds before it will be recognized.

- d. Processing - Upon activation, the RTH bit in TBUSYG is set, and interrupt level #8 is frozen. The time of day in ticks (5msec each) is computed. The difference between that time and the time saved in RTHTIM is computed, and tested for being greater than 750msec. If it is, a 500msec delay is executed, level #8 thawed again, the time is saved in RTHTIM, and exit taken.

If the time difference was less than 750msec, a valid R/T signal has been detected. RTBIT in CPUSG is set, and the R/T init bit (RTIBIT) is tested. If zero (i.e., we are initializing) it is set to 1. If not, and this is the prime machine, a RTDEFocus command is enqueued to MMI. Level #9 is thawed, time is saved in RTHTIM, the RTH bit in TBUSYG is reset, and exit taken.

- e. Error Messages and Recovery - None

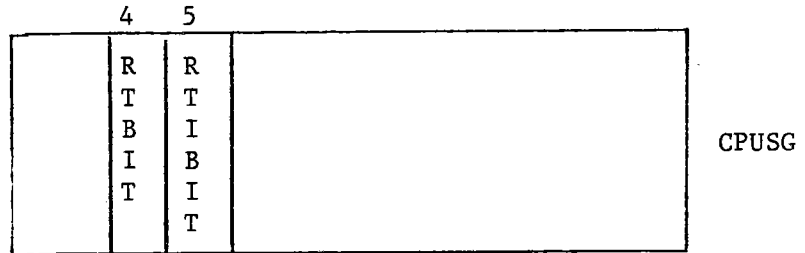
3.2.8.3.4.1.1.2 Data, Logic and Command Paths

Input data:

Global common word CPUSG

Output data:

Global common words CPUSG, TBUSYG



RTBIT - receiver trip state - 0 - receiver trip signal low

1 - receiver trip signal high

RTIBIT - initialization successful flag - 0 - receiver not operational

1 - receiver operational

3.2.8.3.4.1.1.3 Internal Data Description

RTHTIM is 32 bit time of day in ticks at last interrupt.

3.2.8.3.4.1.1.4 Flowchart

See Figure 3.2.8.3.1

3.2.8.3.4.1.2 Submodule II - RTL

3.2.8.3.4.1.2.1 Description

- a. Language Used - FORTRAN
- b. How Invoked - When interrupt occurs on hardware level #9.
- c. Constraints and Limitations - Same as RTH
- d. Processing - Upon activation, the RTL bit in TBUSYG is set, and interrupt level #9 is frozen. The time of day in ticks (5msec each) is computed. The difference between that time and the time saved in RTLTIM is computed, and tested for being greater than .75 sec. If it is, a 500msec delay is executed, level #9 thawed again, the time is saved in RTLTIM, and exit taken.

If the time difference was less than 750msec, a valid R/T -NOT signal has been detected. RTBIT in CPUSG is reset, and RTIBIT is set. Level #9 is thawed, time is saved in RTLTIM, the RTL bit in TBUSYG's reset, and exit taken.

- e. Error Messages and Recovery - None

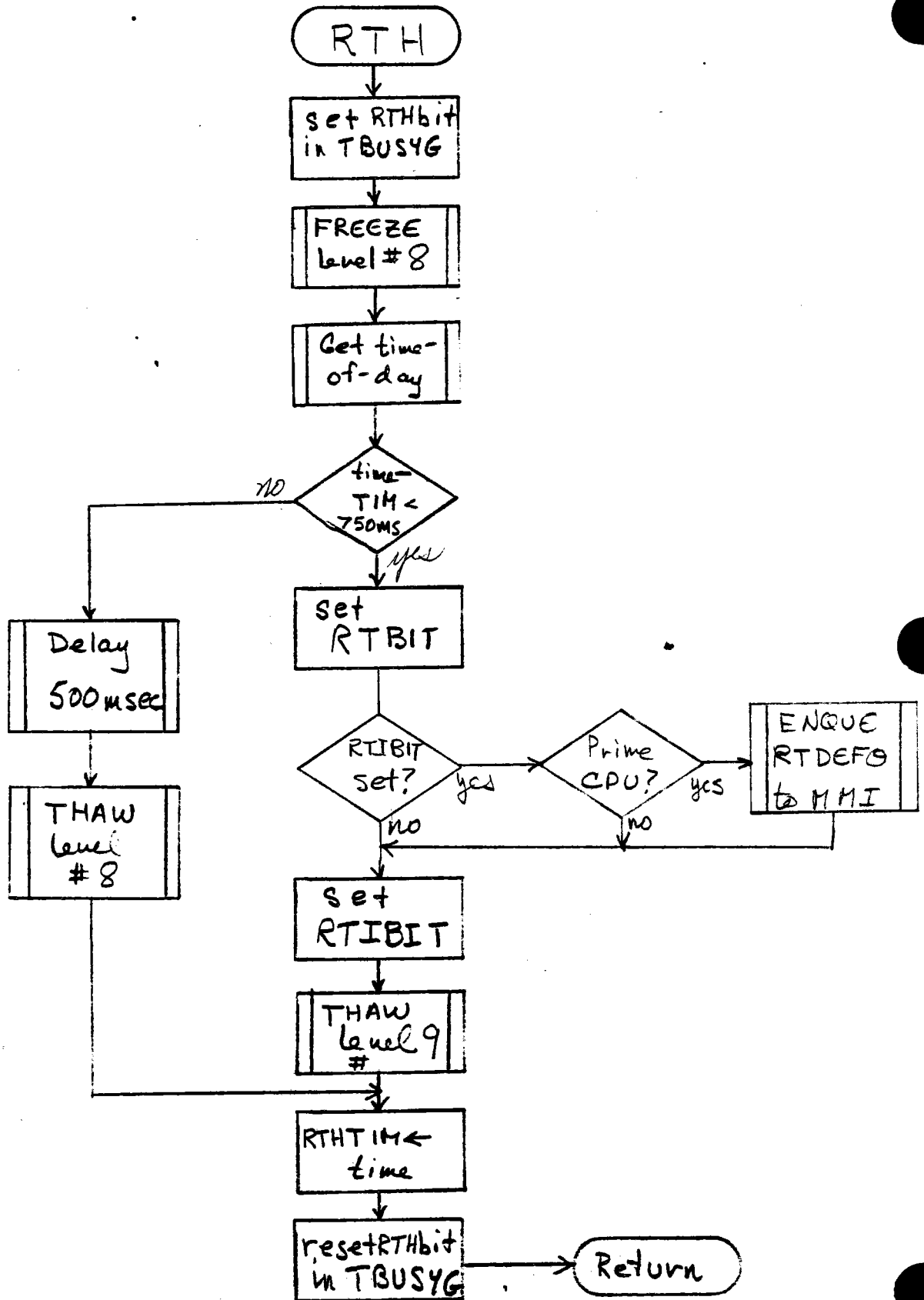


Figure 3.2.8.3.1 Flowchart - RTH
580

3.2.8.3.4.1.2.2 Data, Logic and Command Paths

Input data:

Global common word CPUSG (See 3.2.8.3.4.1.1.2)

Output data:

Global common words CPUSG, TBUSYG

3.2.8.3.4.1.2.3 Internal Data Description

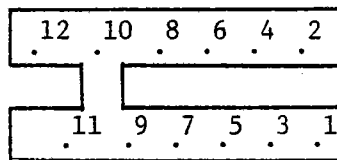
The 32 bit integer RTLTIM is a tick counter to record time of last call to RTL.

3.2.8.3.4.1.2.4 Flowchart

See Figure 3.2.8.3.2

3.2.8.3.5 Interface Description

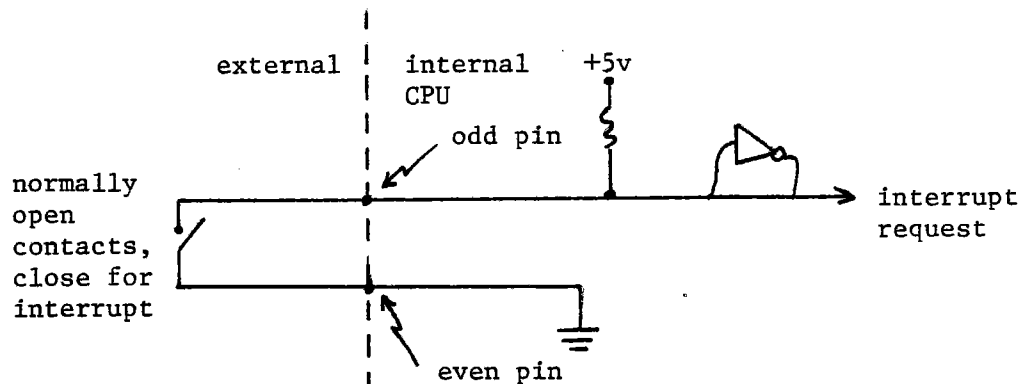
The CPU connection for all external interrupts is a twelve pin connector, as shown below (receptacle as seen on panel):



An interrupt is activated by connecting one of the add numbered pins to ground (as through a contact closure). The even numbered pins are all grounded. The add pins, and their corresponding interrupt levels, are:

<u>Pin</u>	<u>Level</u>
1	n/c
3	#B
5	#A
7	#9
9	#8
11	#7

Each interrupt's circuitry:



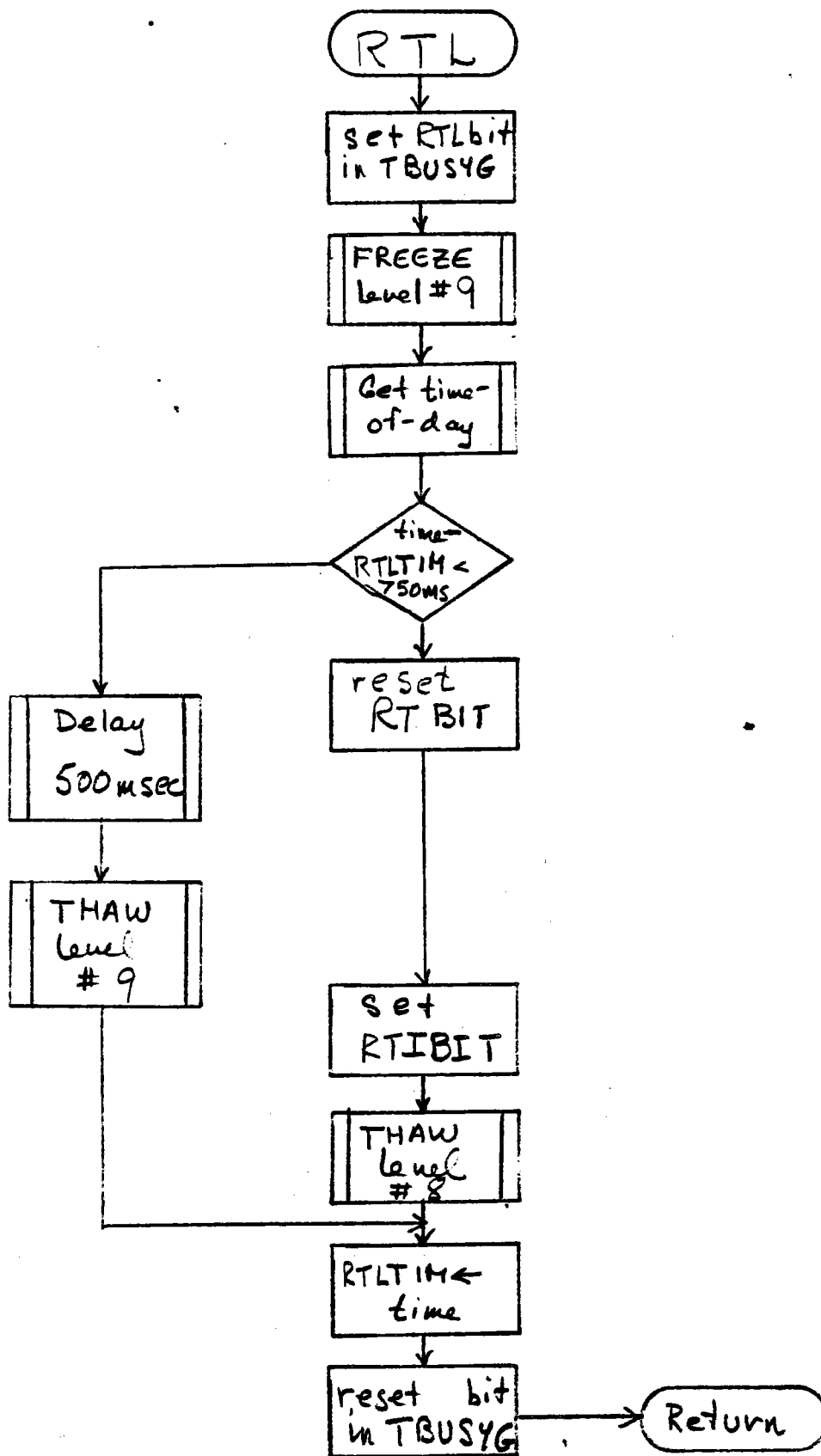


Figure 3.2.8.3.2 Flowchart - RTL

reveals that the interrupts are inverted logic, TTL levels, pulled up into the inactive state internally. If the switch is left closed, the interrupt will fire again every time it is cleared and reenabled. The software interfaces are two: The RTBIT and RTIBIT bits shall reside in global common, accessible to all, and the RTH task shall deliver the ASCII message "DEFOCUS" to the MMI module through the REX service "ENQUE", through queue number 0.

3.2.8.3.6

Test Requirements

Testing shall be performed to test the reaction of the Receiver Trip module in the following cases:

<u>Condition</u>	<u>Expected Results</u>
Both signals low RT high, RTNOT low	Error message from initialization DEFOCUS on, command to MMI on transition
RTNOT high, RT low	DEFOCUS off

3.2.8.4 Switching Function

3.2.8.4.1 Purpose

The purpose of the switching function is to prepare and maintain the data bases in the backup machine in the event of switchover. It must monitor switchover criteria and initiate and control switchover when it does occur.

3.2.8.4.2 Requirements

3.2.8.4.2.1 Design Requirements

The following paragraph from the 10 MWe Software/Firmware Functional Requirements Specification (FRS) states the basic switching requirement:

"3.1.d. Maintain a "Backup" system by providing redundant field communications and providing data transfer sufficient to allow one-way "Backup" fail-over with minimal degradation;"

3.2.8.4.2.2 Derived Requirements

The FRS further states:

- "3.2.1.8 f. Monitor status between computers to determine if the "Backup" should take over;
- g. Perform peripheral switching when necessary;"

Functionally, these three requirements from the FRS may be restated:

- a. Establish and maintain communications such that the backup system is able to perform the complete HAC task with minimum disruption in the event of switchover;
- b. Sense failure of the prime system and initiate and control CPU and peripheral switchover; and
- c. Prevent interference from a failed prime system.

3.2.8.4.3 Design Approach

3.2.8.4.3.1 Functional Allocations

To accomplish the required processing, the switching function consists of three tasks: SWI, SWIRG1, and SWIRG8, and three submodules of SWI: SWIINT, SWIHFC, and SWIPCS. There also is a failover interrupt routine, INT.7.

A brief description of each follows:

- a. SWI - is the main task of the switching function and executes once each second, activating tasks SWIRG1 and SWIRG8. The SWIRG1 and SWIRG8 tasks will read global common from the Prime HAC and resume the SWI task. The SWI task will detect a failover condition and call the submodule SWIINT to initiate HAC failover logic.
- b. SWIINT - This submodule is a subroutine which is responsible for initiating the Backup HAC to act as the Prime HAC. It is responsible for sending the interrupt to halt the Prime HAC, calling submodule SWIPCS to acquire the peripheral controller switches and activating one synchronous loop.
- c. SWIPCS - This subroutine is called by the SWIINT subroutine and is responsible for acquiring the peripheral controller switch and, hence, the peripherals attached to those controllers. It also master clears the 1907 communications handler.
- d. SWIHFC - This submodule is a subroutine called by SWIINT and will determine when the majority of HFCs are in their 300 millisecond window in which the HFCs will not respond to the HACs input status request. When this occurs, it is time delay 350 milliseconds, then get the synchronous loop (FCP) executing. This will insure the HFCs will be listening to the HAC when the sun synch message is transmitted.
- e. SWIRG1 - This task is activated by the SWI task once each second to read the areas of global common designated as one-second areas that need to be updated in the Backup HAC once each second.
- f. SWIRG8 - This task is activated by the SWI task once each second to read 1/8 of the eight-second global common area from the Prime HAC. Therefore, after eight seconds the entire eight-second data base will have been transferred to the Backup HAC.

Designation of which HAC is prime and which is backup, and initial transfer of the global common data base is done during DBI execution (refer to DBI paragraph 3.2.7-19).

The SWI task executes once each second in the backup HAC, transferring the global common data base from the primary HAC to the backup HAC and checking for the occurrence of a switchover condition.

The failover interrupt handler will halt the primary HAC once a switchover has been initiated.

3.2.8.4.3.2 Resource Budget

Code - SWI and handler - 2000 words

Timers - SWI will use one
- SWIRG1 - 2000 words
- SWIRG8 - 2000 words

3.2.8.4.4 Design Description

3.2.8.4.4.1 Module Structure

The SWI task is a task which executes in the backup HAC and activates two tasks, SWIRG1 and SWIRG8, to read global common and calls submodule SWIINT when a failover condition is encountered. The failover interrupt handler is loaded as part of the MAXNET IV operating system.

3.2.8.4.4.1.1 Submodule I - SWI task

3.2.8.4.4.1.1.1 Description

- a. Language used - FORTRAN
- b. How invoked - The SWI task is established in the backup machine by the PBI task during backup HAC initialization. It is remotely activated by the CLK task in the prime HAC following data base initialization in the prime HAC.
- c. Constraints and limitations - The switching function will execute properly only if the M04 SYSGEN is booted in the prime HAC and M05 is booted in the backup HAC. Also, once the switchover has been performed, no further switching of HACs is supported (i.e., only one switching of HACs is allowed).

- d. Processing - The SWI task is a periodic task which executes in the backup HAC once each second to read the global common data base from the primary HAC and update the backup HAC data base.

The global common data base is initially transferred to the backup HAC by the submodule DBIBCK of the DBINET module. The global common data base is divided into three basic types:

- (1) Data to be transferred once each second;
- (2) Data to be transferred once every eight seconds; and
- (3) Data to be transferred once at initialization (done by DBIBCK in prime HAC).

The SWI task transfers types a and b. Each second the SWI task will activate the SWIRG1 and SWIRG8 tasks to read all of the one-second (type a) global common and a portion of (type b) global common from the primary HAC. The data will be read using the MAXNET capability of core devices. The global common areas are equated to core devices.

If any read errors occur, or upon primary HAC request (done by setting the SW bit, bit 15 in global common word SVOVG), the SWI task will call the SWIINT subroutine to perform the switchover to the backup HAC.

The ALMQUE subroutine will also be called to alarm the operator of the HAC switchover condition. The SWI task, upon switchover, will de-establish the SWI, SWIRG1, and SWIRG8 tasks to free up main memory. Then SWI will establish the asynchronous tasks not established by CLK during system initialization. These tasks include: MMI, CMD, GET, STA, CFO, ALO. The SWI task will then stop execution by calling EXIT.

- e. Error messages and recovery - The SWI task issues the following error messages and warnings:

1. If the SWIRG1 or SWIRG8 tasks fail to establish, the following message is printed; and the SWI task will stop execution via a call exit.

CANNOT ESTABLISH BACKUP TASKS CONDITION CODE IS XX

where XX is bad condition code returned by the operating system.

2. In the event that transfer of global common takes longer than one second (usually due to link I/O retries of data transmissions), then the following warning message is printed:

SWI OVERRAN ONE SECOND FRAME I = XX

If XX = 1 then one second overrun occurred during transmission of one-second data. If XX = 2 then the overrun occurred during the eight second-data transmission.

3. When HAC failover occurs, the SWI task will print:

BACKUP SWITCHOVER HAS OCCURRED ERROR STATUS = XXXX

where XXXX is an error code telling why the failover took place. The parameter XXXX can take the following values:

<u>XXXX</u>	<u>MEANING</u>
1	HAC requested failover (i.e., total communication failure, disk errors)
14Z	The backup HAC timed out when attempting to read the Zth portion of one-second data from the primary HAC
15Z	The backup HAC encountered read errors while reading the Zth portion of the one-second data.
84Z	The backup HAC timed out when reading the Zth portion of the eight-second data from the primary HAC.
85Z	The backup HAC encountered read errors while reading the Zth portion of the eight-second data.

3.2.8.4.4.1.1.2 Data, Logic, and Command Paths

Global common directly used by SWI include the words CPUSG, SWOVG, and SPRIN4(27).

CPUSG (never transferred to backup):

13 14 15

B	S	P
R	W	R

3.2.8.4.4.1.1.3 Internal Data Description

Miscellaneous variables

3.2.8.4.4.1.1.4 Flowcharts

See Figure 3.2.8.4.1 for the SWI Task flowchart

3.2.8.4.4.1.2 Submodule II - Subroutine SWIINT

3.2.8.4.4.1.2.1 Description

- a. Language used - FORTRAN
- b. How invoked - Called by the SWI task to initiate a HAC switchover.
- c. Constraints and limitations - None
- d. Processing - Upon being called by the SWI task, the SWIINT subroutine will initiate the processing required to start the backup HAC. These processing steps are:
 1. Set bit in CPUSG to indicate that a switchover has occurred and that the backup HAC is now prime.
 2. Send interrupt to the primary HAC to halt its processing.
 3. Call subroutine SWIPCS to get control of the peripherals.
 4. Call subroutine SWIHFC to indicate when to start the synchronous tasks.
 5. Activate the asynchronous and synchronous tasks in the backup HAC (TOK, TIK, CSI, EXI).
 6. If all tasks got activated, then go to step (8). Otherwise, go to step (7).
 7. Notify operator of the activation error and ask if a retry is desired; if so, go to step (5). Otherwise, go to step (9).
 8. Check if any sequences were active in the primary HAC. If so, then send a notification message (enqueue) to the sequence task and go to step (9).
 9. Return to calling task.

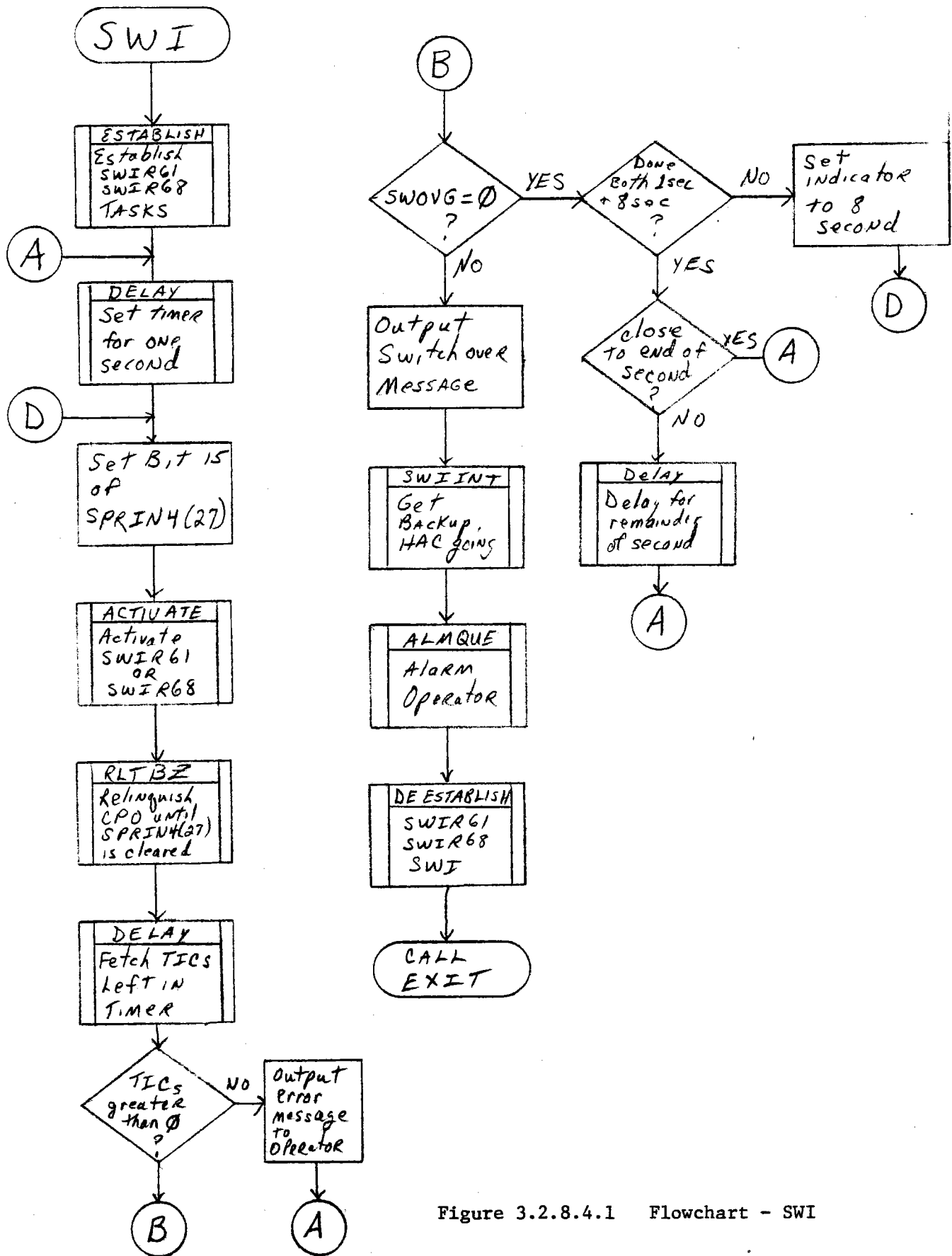


Figure 3.2.8.4.1 Flowchart - SWI

- e. Error messages and recovery - If one of the tasks fails to activate, the operator is informed and asked if he wants to try to activate these tasks once again. The error message reads:

TASK XXXX FAILED TO ACTIVATE
DO YOU WANT TO TRY AGAIN (Y OR N)

where XXXX is the task name.

3.2.8.4.4.1.2.2 Data, Logic, and Command Paths

Input data:

SEQNMG in global common to detect if sequences were active at the time of switchover.

Output data:

- a. Interrupt to the primary HAC (interrupt level #7)
- b. Enque message to the SEQ task as notification that sequences were active in the primary HAC when the switchover occurred.
- c. Global common word CPUSG.

3.2.8.4.4.1.2.3 Internal Data

Miscellaneous variables and pointers.

3.2.8.4.4.1.2.4 Flowcharts

See Figure 3.2.8.4.2 for SWIINT flowchart.

3.2.8.4.4.1.3 Submodule III - Subroutine SWIPCS

3.2.8.4.4.1.3.1 Description

- a. Language used - FORTRAN and MODCOMP assembly language.
- b. How invoked - Called by subroutine SWIINT in order to request control of the peripheral controller switches.
- c. Constraints and limitations - The peripheral controller switches must be placed in the auto (middle) position.
- d. Processing - Upon being called by the SWIINT subroutine, the SWIPCS subroutine will initialize variables and registers needed to send the demand request to both of the peripheral controller switches. The peripheral controller switches (PCSs) have physical address of unit 8 and 9.

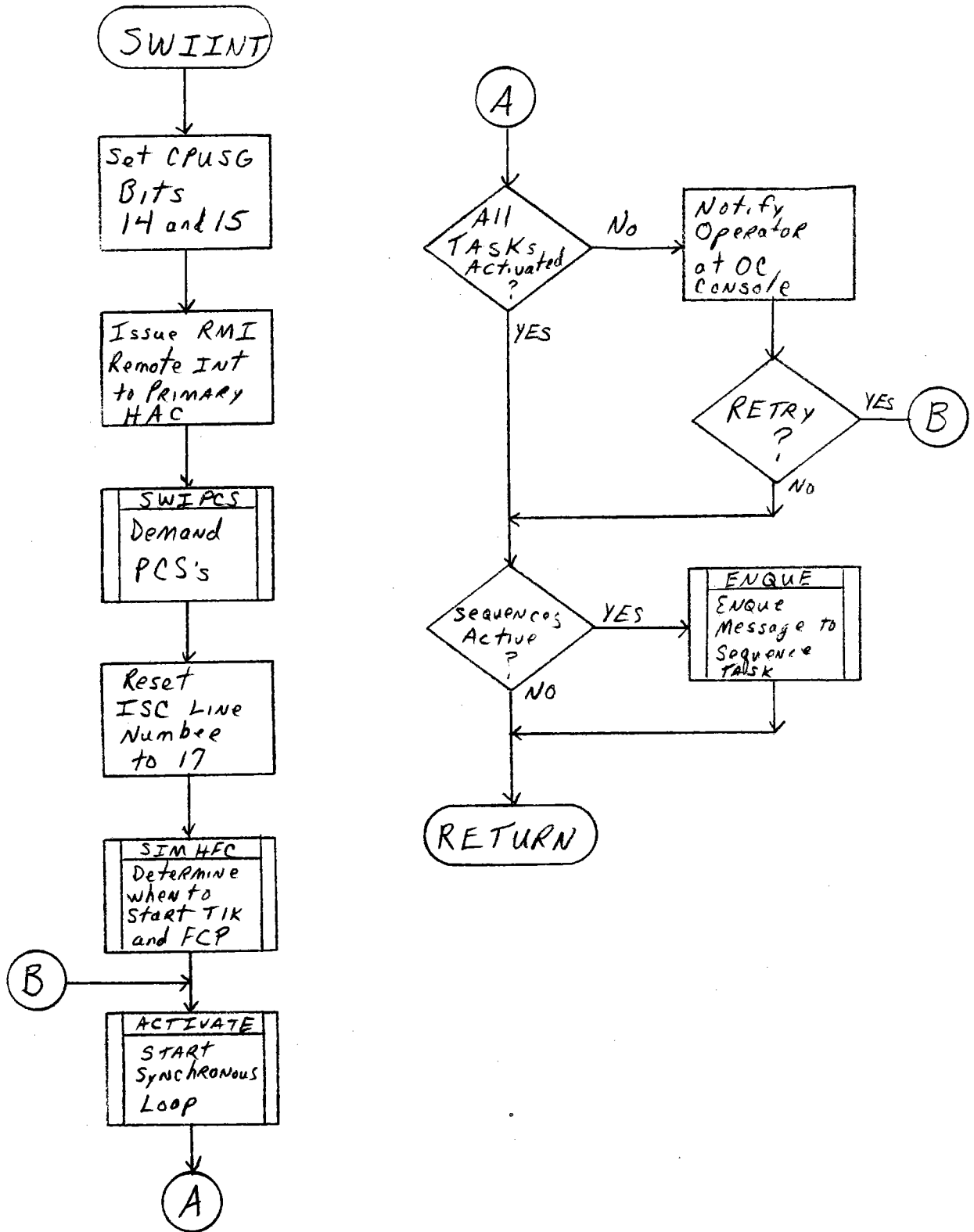


Figure 3.2.8.4.2 Flowchart - SWIINT

Prior to outputting the demand request to the PCSs, the SWIPCS submodule will lockout interrupt level #12. This will prevent an unsolicited data interrupt (DI) from halting the MODCOMP operating system.

Following the DI lockout, the SWIPCS subroutine will also lockout the SI (service interrupt) for the 4811 controllers which have the chromatics consoles on them. This again will prevent the MODCOMP operating system from halting due to an unsolicited SI interrupt originating from the graphics console.

Following the SI lockout, the SWIPCS submodule will issue the demand request to both PCSs 8 and 9. Master clear commands are then sent to all the 4811 controllers in the PCSs to reset their logic and terminate any I/O operations. The status of the PCSs 8 and 9 is then input and checked to insure that the PCSs are connected to the backup HAC.

If either PCS is not connected to the backup HAC, the SWIPCS submodule will notify the operator and request if he wishes to try the PCS demand request once again. If so, then the attempt is made once again; otherwise, SWIPCS will proceed to master clear the 1907 multiplexor and then exit.

A check is also made to insure the 1907 master clear was accepted; and if not, the operator is notified and asked if he wishes to try the master clear once again.

- e. Error messages and recovery - If either PCS 8 or 9 does not respond to the demand request properly, then the following error message is output on the console:

```
PCS ADDRESS X HAD BAD STATUS YYYY  
DO YOU WANT TO REQUEST PCS AGAIN (Y OR N)
```

where X is the PCS address either 8 or 9 and YYYY is the status actually received back from the PCS. The status should be one of the following:

8040 - means PCS connected to requesting CPU
in auto mode

0440 - PCS connected to requesting CPU in manual
mode

0420 - PCS connected to other CPU in manual mode.

If either of the last two responses are received, make sure both switches located on the front of the PCSs are in the middle position; then retry the demand request.

If the 1907 has an input status of busy following the master clear, the following message will be output:

```
1907 HAD BUSY STATUS = YYYY  
DO YOU WANT TO MASTER CLEAR AGAIN (Y OR N)
```

where YYYY is status returned from 1907. A simple retry should clear up the problem.

3.2.8.4.4.1.3.2 Data, Logic, and Command Paths

Input data

- a. Controller status from the PCSs 8 and 9
- b. 1907 controller status

Output data

- a. Demand request to PCS 8 and 9
- b. Master clear command to 4811 controllers in PCSs
- c. Master clear to 1907 multiplexor.

3.2.8.4.11.1.3.3 Internal Data

- a. Demand request command - #0800
- b. Service Interrupt (SI) disconnect command-#0400
- c. SI connect command-#0500
- d. Master clear for 4811 controller-#4440
- e. Compare mask for PCS input status compare-#8040
- f. Master clear for 1907 communication multiplexor-#8400

3.2.8.4.4.1.3.4 Flowchart

See Figure 3.2.8.4.3 for SWIPCS flowchart.

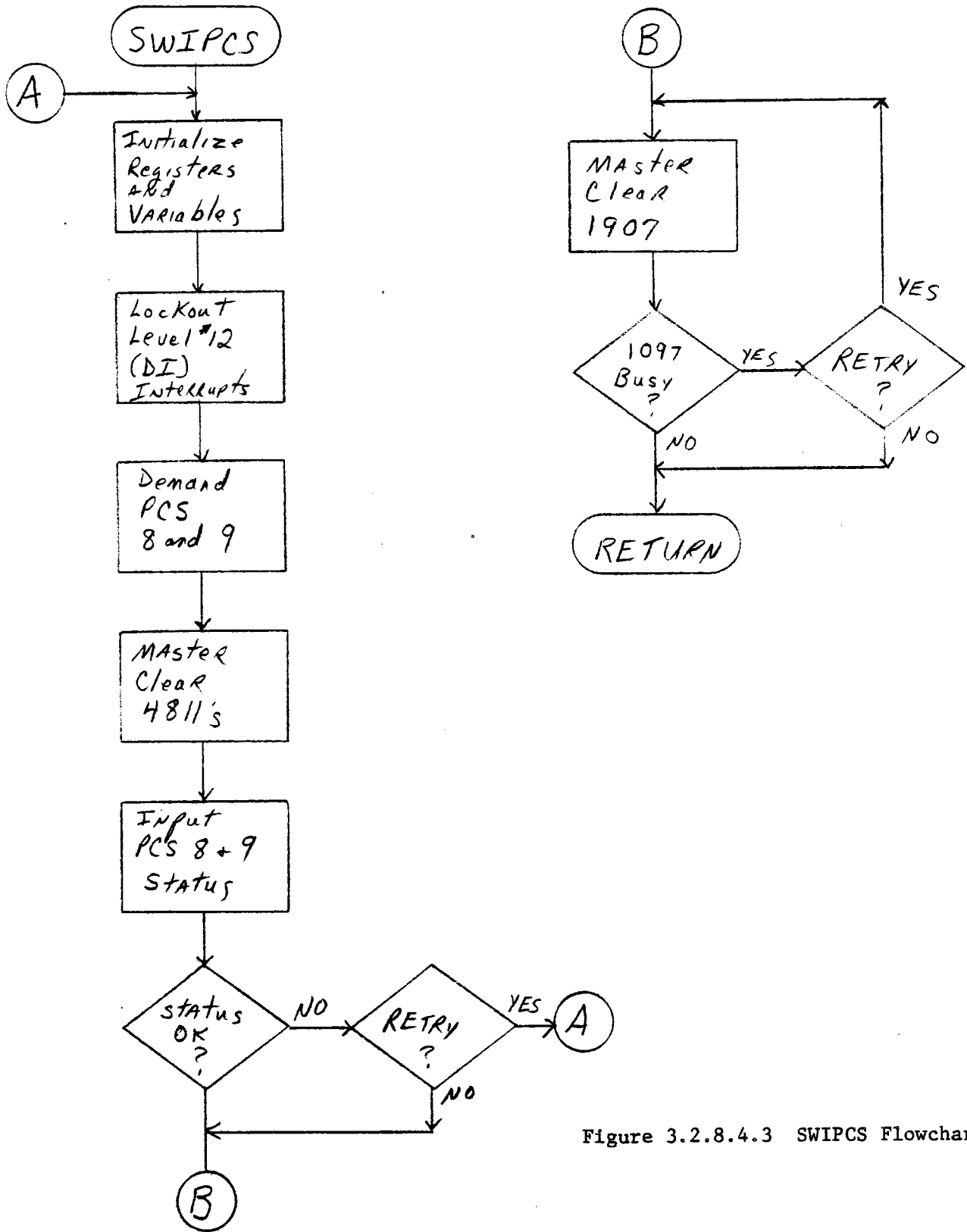


Figure 3.2.8.4.3 SWIPCS Flowchart

3.2.8.4.4.1.4 Submodule IV - Subroutine SWIHFC

3.2.8.4.4.1.4.1 Description

- a. Language used - FORTRAN
- b. How invoked - Called by submodule SWIINT prior to initiating the synchronous loop.
- c. Constraints and limitations - none.
- d. Processing - When called, the SWIHFC subroutine will begin processing by determining how many HFCs had a good status. The number of good HFCs will be used to determine when twenty-five percent of these HFCs have stopped answering the HAC. When this occurs, the HFCs have entered their 300 msec window where they talk to their HCs. To insure FCP task has HFCs listening when they receive the first sun synch command, this window must be determined accurately so the HFCs will respond. Therefore SWIHFC's only function is to detect this 300 msec window and return control to SWIINT so that submodule can start the synchronous loop at the correct time.

The SWIHFC submode will execute for a maximum of ten seconds, sending an input status message to the HFCs and waiting for their response.

When twenty-five percent have not responded (i.e., they are in their 300 msec window), then the SWIHFC submodule will return to the SWIINT. If ten seconds expire and twenty-five percent of the HFCs have not stopped responding, then control is returned to the SWIINT subroutine; and the synchronous loop (TIK, FCP, etc.) is started anyway.

- e. Error messages and recovery - none.

3.2.8.4.4.1.4.2 Data, Logic, and Command Paths

Input data

- a. Status response messages from the HFCs

Output data

- a. Status request messages from the HFCs
- b. A good indication back to the SWIINT subroutine when 25% of the HFCs have stopped talking.

3.2.8.4.4.1.4.3 Internal Data

- a. Input and output buffers for the HFC communications
- b. Miscellaneous variables for controlling the I/O operations.

3.2.8.4.4.1.4.4 Flowcharts

See Figure 3.2.8.4.4 for SWIHFC flowchart.

3.2.8.4.4.1.5 Submodule V - SWIRG1 (Task)

3.2.8.4.4.1.5.1 Description

- a. Language used - FORTRAN
- b. How invoked - Activated by SWI task
- c. Constraints and limitations - None
- d. Processing - The SWIRG1 task, when activated by the SWI task, will read, from the primary HAC, all of the one-second portion of global common. This global common includes COMIS1, COMIS2, COMIS3, COMIS4, COMIS5, COMQU2.

When activated, the SWIRG1 task will build the read user file tables required and then read the various global common areas into its local buffer area. The reads are performed by equating core devices to the global common areas and using MAXNET to read these areas across the link between HACs.

Once in local buffers, the global common areas will be transferred into global common using core devices within the backup HAC which have been equated to the corresponding global common in the backup HAC.

When the transfer to global common in the backup is done, the SWIRG1 will clear bit 15 of word SPRIN4(27) to resume the MWI task which has been waiting for the transfer to complete.

- e. Error messages and recovery - If a read error is encountered, or a read times out, an error indicator will be set in global common word SWOVG in COMIS8.

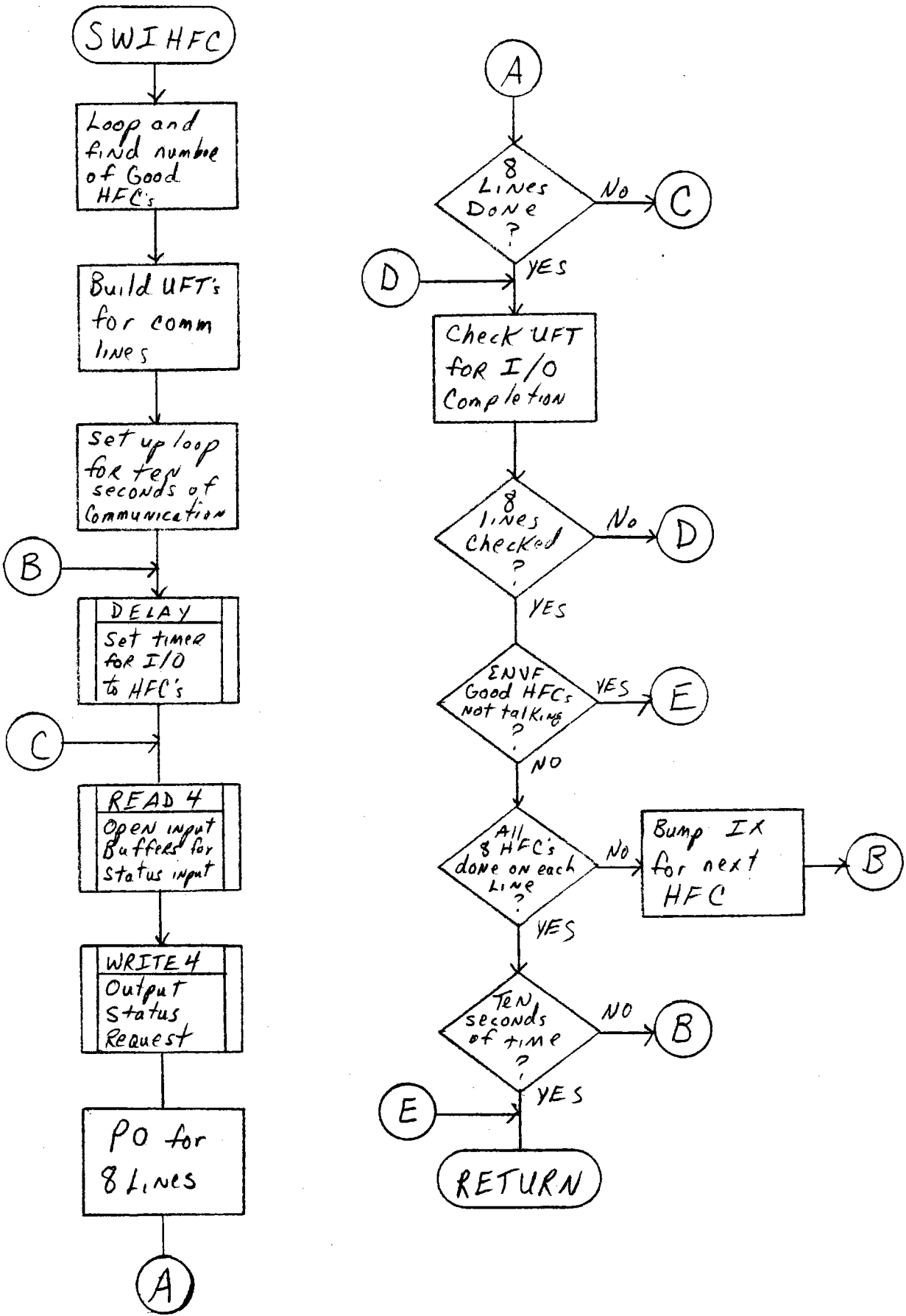


FIGURE 3.2.8.4.4 SWIHFC Flowchart

3.2.8.4.4.1.5.2 Data, Logic, and Command Paths

Input data: Input data will include global common from the primary HAC.

- a. COMIS1
- b. COMIS2
- c. COMIS3
- d. COMIS4
- e. COMIS5
- f. COMQUE

Output data: Same as input data except it is output to backup HAC data base; also work SPRIN4(27) in COMIN4.

3.2.8.4.4.1.5.3 Internal Data

Internal data buffers and variables to do global common transfer.

3.2.8.4.4.1.5.4 Flowcharts

See Figure 3.2.8.4.5 for SWIRG1 flowchart.

3.2.8.4.4.1.6 Submodule VI - SWIRG8 (Task)

3.2.8.4.4.1.6.1 Description

- a. Language used - FORTRAN
- b. How invoked - Activated by SWI task.
- c. Constraints and limitations - None
- d. Processing - The SWIRG8 task, when activated by the SWI task, will read one-eighth of the eight-second data. Therefore, in eight seconds all of the eight-second data will be updated in the backup HAC.

An index is kept to keep track of which portion of the eight-second data is transferred.

For each portion of global common, a core device is read from the primary HAC and transferred to global common in the backup HAC via the core device mechanism.

The SWI task is notified when SWIRG8 is finished by clearing bit 15 of global common SPRIN4(27). The SWIRG8 task will then exit until being activated in the next second by the SWI task.

- e. Error messages and recovery - None.

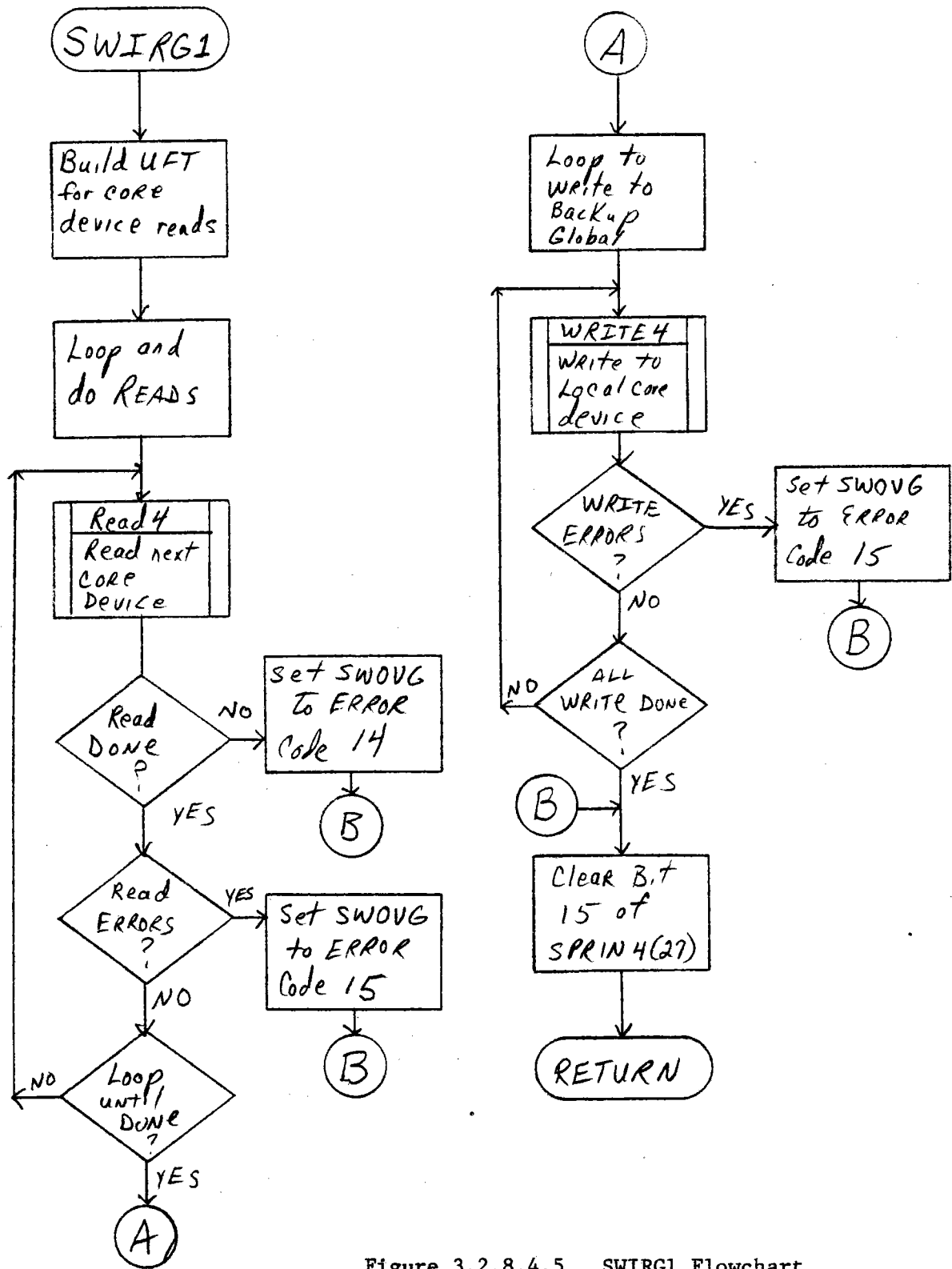


Figure 3.2.8.4.5 SWIRG1 Flowchart

3.2.8.4.4.1.6.2 Data, Logic, and Command Paths

Input data: Input data is all the eight-second global common areas from primary HAC, including:

- a. COM8S1
- b. COM8S2
- c. COM8S3
- d. COM8S4

Output data: Same as above except the output is to the backup HAC data base. Also variable SPRIN(27) clearing bit 15 will resume SWI task.

3.2.8.4.4.1.6.3 Internal Data

Internal data buffers and variables to do global common transfers.

3.2.8.4.4.1.6.4 Flowcharts

See Figure 3.2.8.4.6 for SWIRG8 flowchart.

3.2.8.4.4.1.7 Failover Interrupt Handler - INT. 7

3.2.8.4.4.1.7.1 Description

- a. Language used - M5A Assembly Language
- b. How invoked - Interrupt level seven occurs (Level #7)
- c. Constraints and limitations - None
- d. Processing - The failover interrupt is sent by the SWI task in the backup CPU to the prime CPU to indicate that it has taken over the prime's functions and causes the prime CPU to drop out of the control system.
- e. Error messages and recovery - None.

3.2.8.4.4.1.7.2 Data, Logic, and Command Paths

None

3.2.8.4.4.1.7.3 Internal Data Description

There is no data internal to this function.

3.2.8.4.4.1.7.4 Flowchart

See Figure 3.2.8.4.7 for the Failure Interrupt Handler flowchart.

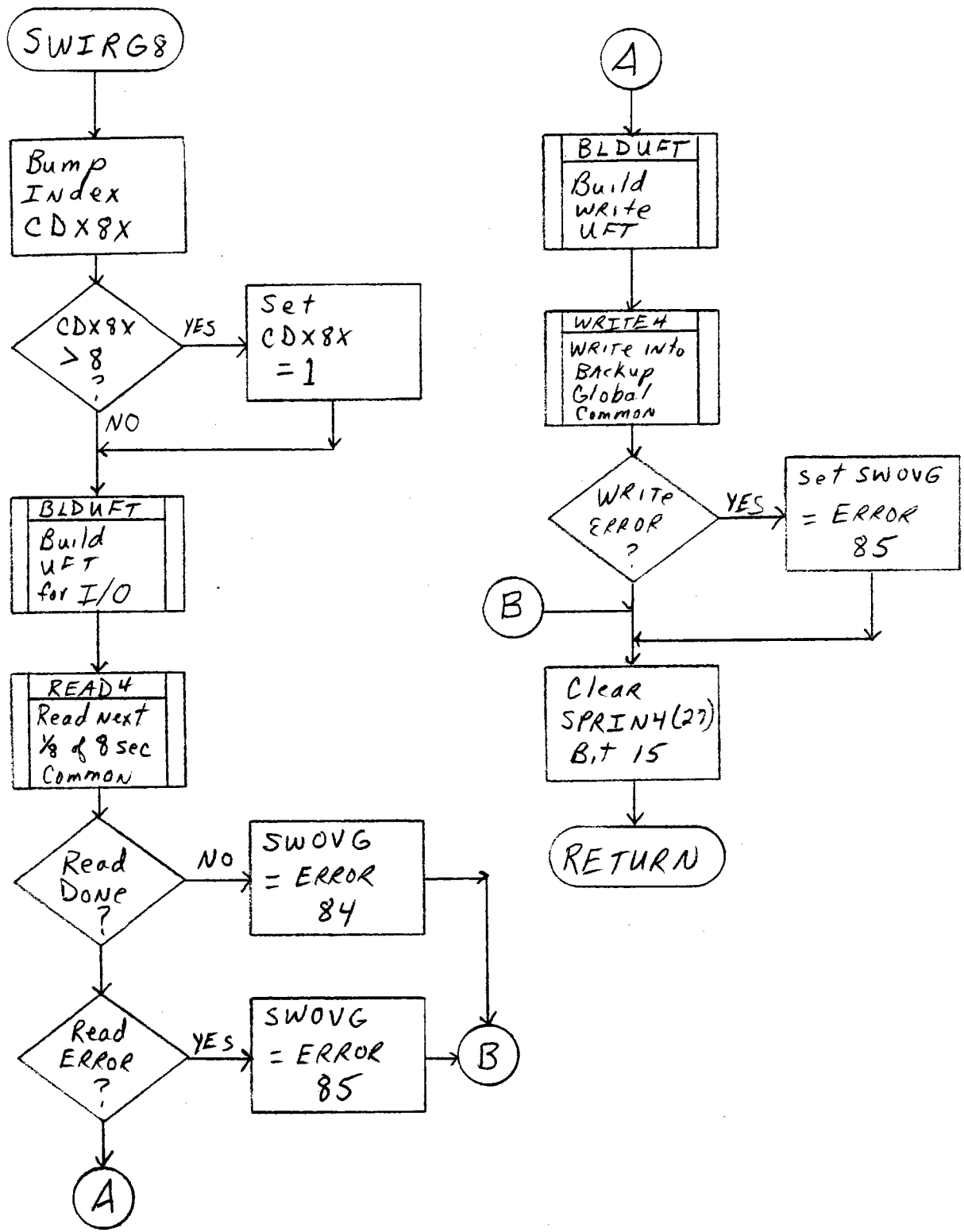


Figure 3.2.8.4.6 Flowchart - SWIRG8

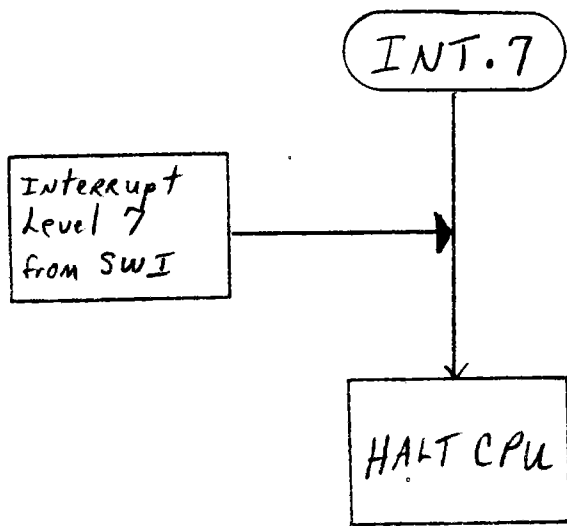


Figure 3.2.8.4.7 Flowchart - INT.7

3.2.8.4.5

Interface Description

The primary interface with the other software modules is via bit 15 of the global common word SWOVG which indicates to the SWI task if a switchover is desired by the primary HAC.

Also bits in CPUSG will be used to indicate which HAC is primary or backup and if a switchover has occurred.

Hardware interface for the PCS interrupt handler for demanding control of the peripherals is via the MAXNET operating system.

3.2.8.4.6

Test Requirements

Testing of the SWITCH function shall be performed through failing of various hardware components during integration testing. In particular, correct functioning will be verified in the following cases:

- a. Failed prime CPU;
- b. Failed backup CPU;
- c. Failed 4824 output (from prime CPU);
- d. Failed 4824 input (to prime CPU);
- e. Failed field communications in prime CPU; and
- f. External setting of the prime CPU's FS bit in CPUSG.

3.2.8.5 Field Power Loss

3.2.8.5.1 Purpose

The Field Power Loss (FPL) function will keep a bit in the global data base (herein identified as FPLBIT) updated, reflecting the status of the field power loss signal. It will also alert SEQ whenever the signal is raised, if any sequences are in progress.

3.2.8.5.2 Requirements

3.2.8.5.2.1 Design Requirements

Paragraph 3.2.1.8 of the Software/Firmware Functional Requirements Specification states:

- h. Detect a power loss signal and store this information in the data base for information to all tasks.

3.2.8.5.2.2 Derived Requirements

Assume that the signals mentioned are TTL compatible logic levels, one indicating (when high) that field power loss has occurred; the other (when high) that power is restored.

The Field Power Loss Module shall keep in FPLBIT in CPUSG the current status of the field power loss (FPL) signal, thus allowing all modules to know the field power state at anytime. Field Power Loss shall also explicitly alert the command sequencer task (SEQ) when field power is lost and transfer the contents of HCST1G to PWHC1G.

The signals indicating field power loss and its inverse will probably be noisy and subject to contact bounce; therefore a signal will not be said to exist unless it is still detected .5 seconds after it is first sensed.

3.2.8.5.3 Design Approach

3.2.8.5.3.1 Functional Allocations

There will be three parts to the field power loss module: two tasks and an initialization routine. Each of the two signals will be wired to separate external interrupt levels in the MODCOMP CPU (levels #A & #B). The initialization routine (further described in the DBINIT module) will CONNECT the interrupt levels to the handlers, enable the interrupts, check for proper state initialization, and report FPL status.

3.2.8.5.3.2

Resource Budgets

- Priority Levels - 30 (PLH), 30 (PLL)
- Memory - 200 words
- Timing - less than 500 microseconds/activation
- Hardware Interrupts - 2

3.2.8.5.4

Design Description

3.2.8.5.4.1

Module Structure

The Field Power Loss Module consists of two interrupt activated submodules. One submodule is activated when FPL is detected and the other is activated when the field power returns to normal.

3.2.8.5.4.1.1

Submodule I - PLH

3.2.8.5.4.1.1.1

Description

- a. Language Used - FORTRAN
- b. How Invoked - When interrupt occurs on hardware level #A.
- c. Constraints and Limitations - The FPL signals that are wired to the MODCOMP interrupt levels must be opposite for tasks PLH and PLL to successfully operate. If they are both high, the system will "hang" in a high priority loop. If they are both low neither PLH or PLL will be activated.

The FPL signal must persist for at least .5 seconds before it will be recognized.

- d. Processing - Upon activation, the PLH bit in TBUSYG is set, and interrupt level #A is frozen. The time of day in ticks (5msec each) is computed. The difference between that time and the time saved in OLDTIM is computed, and tested for being greater than 750msec. If it is, a 500msec delay is executed, level #A thawed again, the time is saved in PLHTIM, and exit taken.

If the time difference was less than 750msec, a valid FPL signal has been detected. FPLBIT in CPUSG is set, and the FPL init bit (FPIBIT) is tested. If zero (i.e., we are initializing), it is set to 1. If not, and this is the prime machine, the status array HCST1G is copied to PWHCLG, and if sequences are in progress, the Field Power Loss message is enqueued to SEQ. Level #B is thawed, time is saved in PLHTIM, the PLH bit in TBUSYG is reset, and exit taken.

- e. Error Messages and Recovery - None

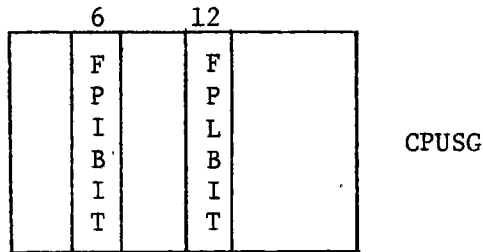
3.2.8.5.4.1.1.2 Data, Logic and Command Paths

Input data:

Global common words CPUSG, SEQNMG, and array HCST1G.

Output data:

Global common words CPUSG, TBUSYG, and array PWHC1G.



FPIBIT - Initialization

successful flag: 0 - field power sensing not operational

1 - field power sensing operational

FPLBIT - Field power loss state:

0 - power OK;

1 - power down

3.2.8.5.4.1.1.3 Internal Data Description

The 32 bit integer PLHTIM has tick time of last call.

3.2.8.5.4.1.1.4 Flowchart

See Figure 3.2.8.5.1

3.2.8.5.4.1.2 Submodule II - PLL

3.2.8.5.4.1.2.1 Description

a. Language Used - FORTRAN

b. How Invoked - When interrupt occurs on hardware level #B

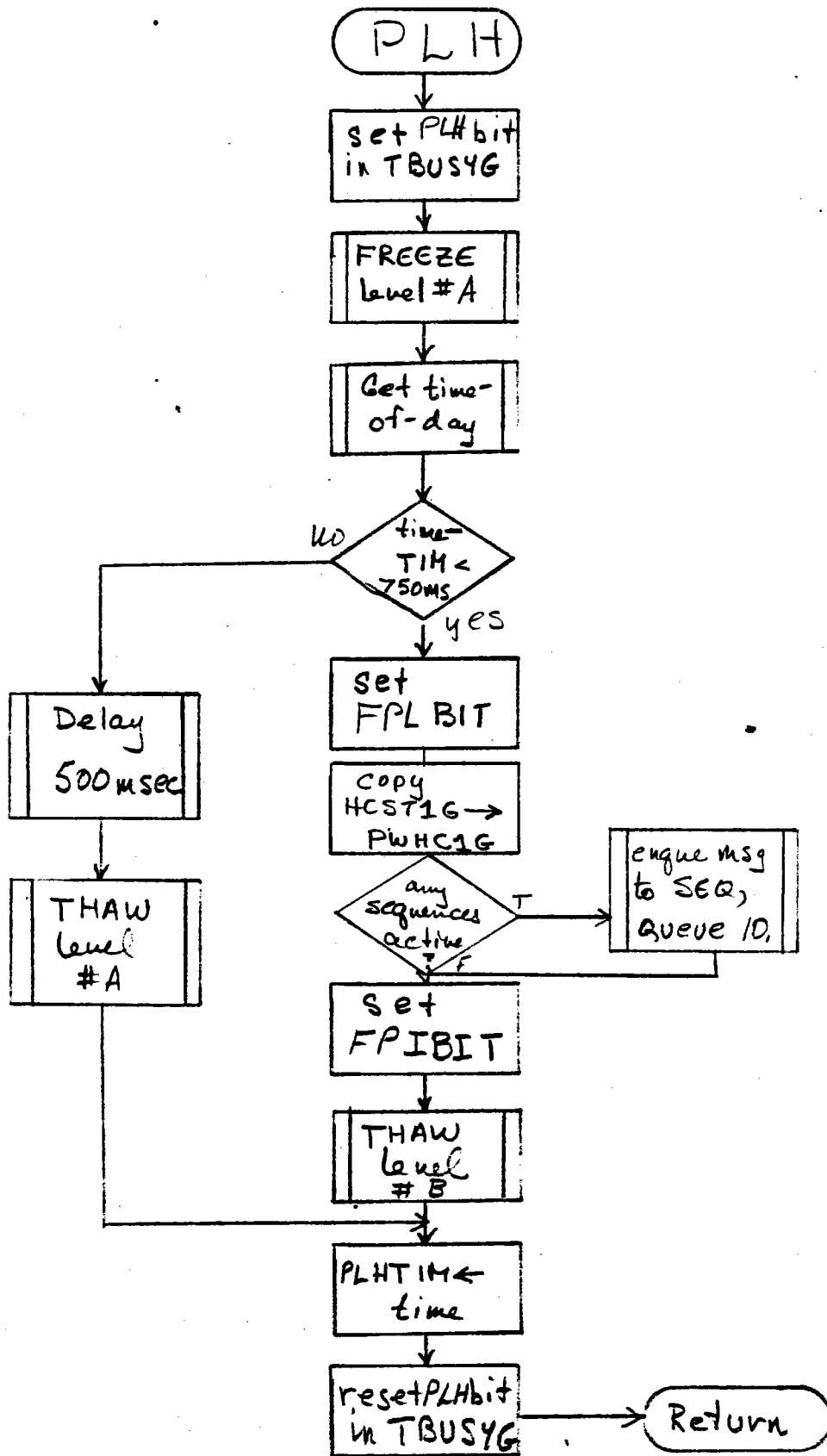


Figure 3.2.8.5.1 Flowchart - PLH

- c. Constraints and Limitations - Same as PLH
- d. Processing - Upon activation, the PLL bit in TBUSYG is set, and interrupt level #B is frozen. The time of day in ticks (5msec each) is computed. The difference between that time and the time saved in PLLTIM is computed, and tested for being greater than 750msec. If it is, a 500msec delay is executed, level #B thawed again, the time is saved in PLLTIM, and exit taken.

If the time difference was less than 750msec, a valid FPL-NOT signal has been detected. FPLBIT in CPUSG is reset, and the FPL init bit (FPIBIT) is set. Level #B is thawed, time is saved in PLLTIM, and exit taken.

- e. Error Messages and Recovery - None

3.2.8.5.4.1.2.2 Data, Logic and Command Paths

Input data:

Global common word CPUSG (See 3.2.8.5.4.1.1.2)

Output data:

Global common words CPUSG, TBUSYG

3.2.8.5.4.1.2.3 Internal Data Description

The 32 bit integer PLCTIM contains the time in ticks of the last call.

3.2.8.5.4.1.2.4 Flowchart

See Figure 3.2.8.5.2

3.2.8.5.5 Interface Description

The hardware interface consists of connection of the proper TTL signals to the HAC, (see 3.2.8.3.5). The software interfaces are two: The FPLBIT and FPIBIT bits shall reside in global common, accessible to all, and the PLH task shall deliver the power loss message to the SEQ module through the REX service "ENQUE", through queue number 10.

3.2.8.5.6 Test Requirements

Testing shall be performed to test the reaction of the Field Power Loss module in the following cases:

<u>Condition</u>	<u>Expected Results</u>
Both signals low	Error message from initialization
FPL high, FPLNOT low	FPBIT on, message to SEQ if sequence in progress
FPLNOT high, FPL low	FPLBIT off

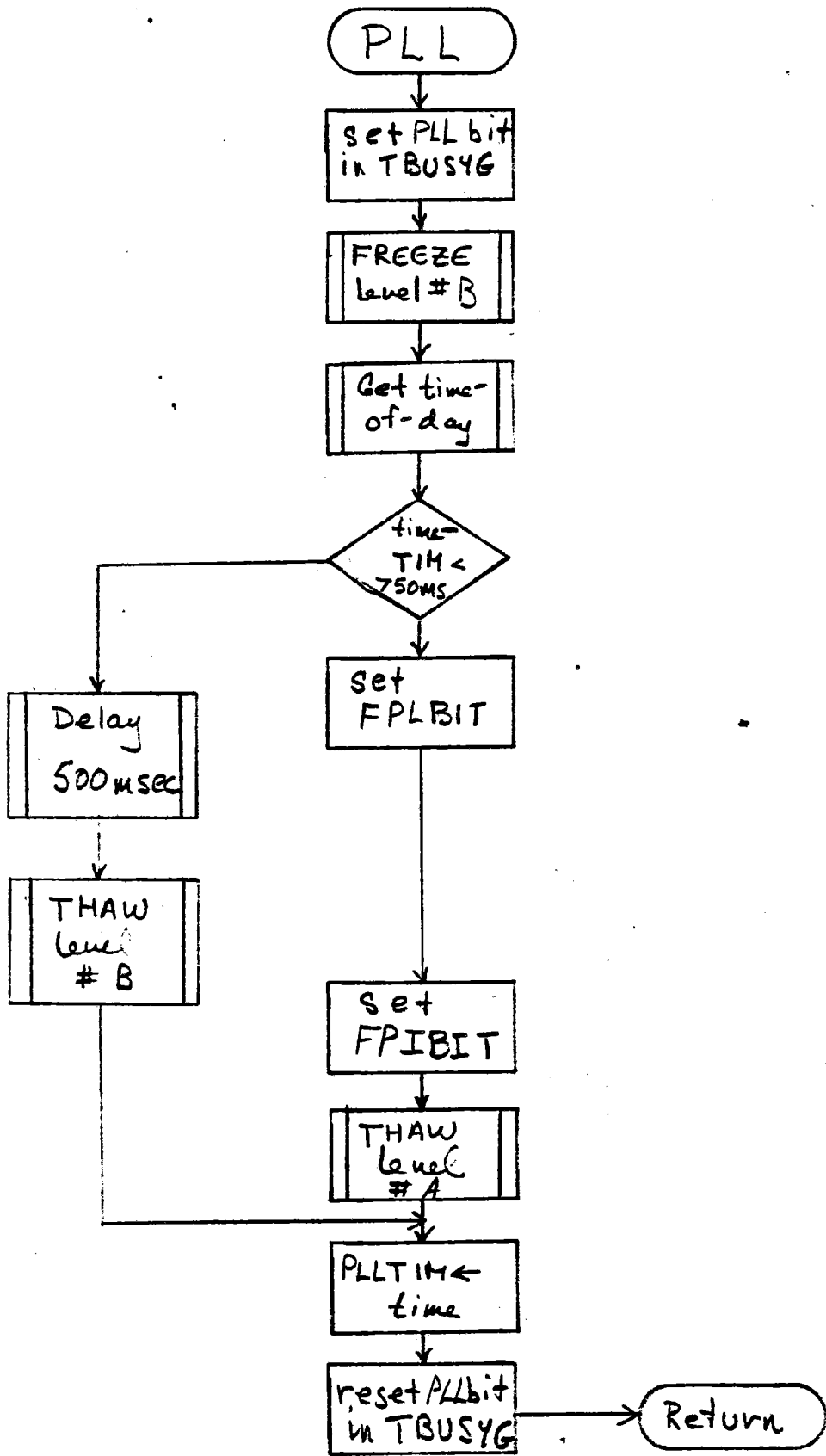


Figure 3.2.8.5.2 Flowchart - PLL
611

3.2.9 Graphics Display Processor Module (GRAPHC)

3.2.9.1 Purpose

The purpose of the Graphics Display Processor Module (GRAPHC) is to provide the functional interface with the Chromatics CG1999 intelligent color graphics systems in a command/response protocol. This module is software structured into one task (GRF) consisting of three submodules. GRAPHC includes the following functions:

- a. Reading requests from the Graphics processor and returning data to support the current display; and
- b. Reading messages from the Graphics requesting a High-Wind Stow or Defocus.

3.2.9.2 Requirements

3.2.9.2.1 Design Requirements

Software Requirements listed in Section 3.1 of the 10 MWe Collector Subsystem Software/Firmware Functional Requirements Specification (FRS) dated 12 June 1980 that apply to the GRAPHC module are:

- a. Control of up to 2048 heliostats in all modes required to operate the heliostats;
- b. Monitor and display the operational status of all heliostats;
- c. Provide graphic displays of the heliostat field or field segments; and
- d. Provide data transfer to the Chromatics Graphics Terminals sufficient for display of a finite number of graphic displays of the heliostat field or field segments and supporting alphanumeric information.

3.2.9.2.2 Derived Requirements

Section 3.2.1.9 of the FRS, states the following requirements for the Graphics Display Processor Module (GRAPHC):

- a. Provide the functional interface with the graphics system in a command/response protocol;
- b. Provide the graphics system with sufficient data for graphics displays;
- c. Accept High-Wind Stow (STHIWIND) and DEFOCUS commands from the CS control room graphics function switches;

- d. Output data to the Chromatics Graphics Terminals only when requested; and
- e. Interface with the Chromatics Graphics Terminals through two distinct Dual Asynchronous Communication Channels (#4811).

3.2.9.3

Design Approach

The Graphics Display Processor (GRAPHC) is composed of one asynchronous task (GRF) which is activated by the External Interface Module (EXTINF) or by self activation (via DEQUE). Figure 3.2.9-1 displays the functional interface diagram of the two Chromatics CG1999 graphic systems, the External Interface Module, the Man-Machine Interface Module, and the subject Graphics Display Processor Module. Command messages originate in the Chromatics CG1999 and are read by the External Interface Module. The External Interface Module checks for an emergency command from the CS control room graphics console and, upon receipt, the emergency command is placed into the Man-Machine Interface emergency command queue. Otherwise, the incoming message is source tagged and queued for the Graphics Display Processor. The command messages received by the Graphics Display Processor are shown in Tables 3.2.9-I through 3.2.9-IV. Detailed discussion of these messages and their resultant action by the Graphics Display Processor is given in the following subsection. Figure 3.2.9-2 exhibits the functional allocation of the graphics task (GRF) into three submodules.

3.2.9.3.1

Functional Allocation

GRAPHC is comprised of three submodules. The basic purpose of each submodule is briefly described in the following paragraphs:

- a. GRF (Graphics Display Processing Task) - dequeues a command message from the graphic's queue, and determines the message's source and type. The message type indicates what action GRF is to perform, and the source indicates where to send the action taken. Listed below are the message types received and the action taken:
 - 1. Message type 1 (Table 3.2.9-I) - Emergency High-Wind Stow (WTHIWIND) or DEFOCUS command from the Engineering Evaluation Room results in establishing an ASCII error message (Table 3.2.9-V) for the requesting source.
 - 2. Message type 2 (Table 3.2.9-II) - Resetting the Chromatics Terminal results in establishing a reset message (Table 3.2.9-VI) for the requesting Chromatics Processor.

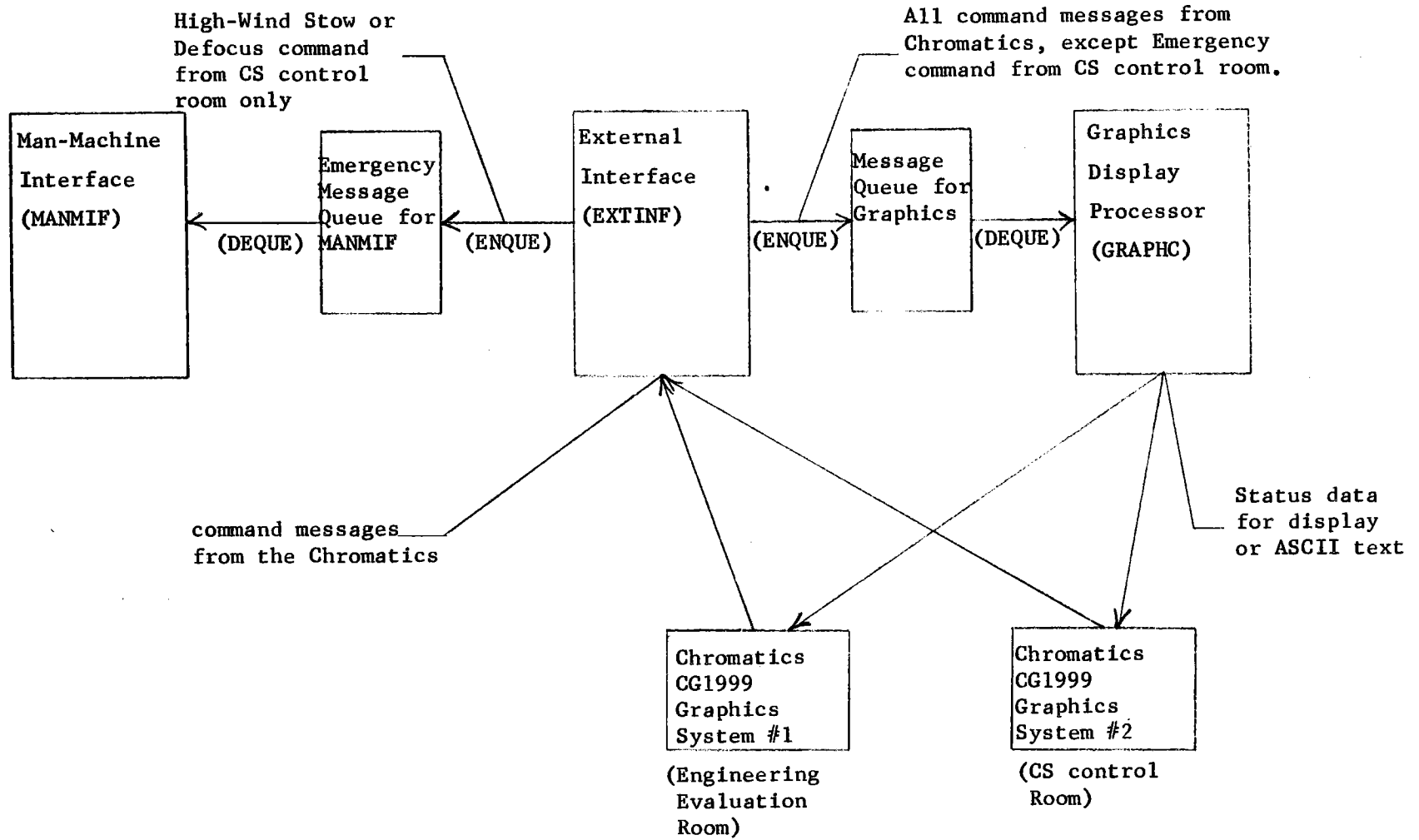


Figure 3.2.9-1 GRAPHC Functional Interface Diagram

MESSAGE LAYOUT

APPLICATION : GDC-HAC

MESSAGE TYPE : GDC Command

PROGRAMMER : D. Pettit

DATE : June 9, 1980

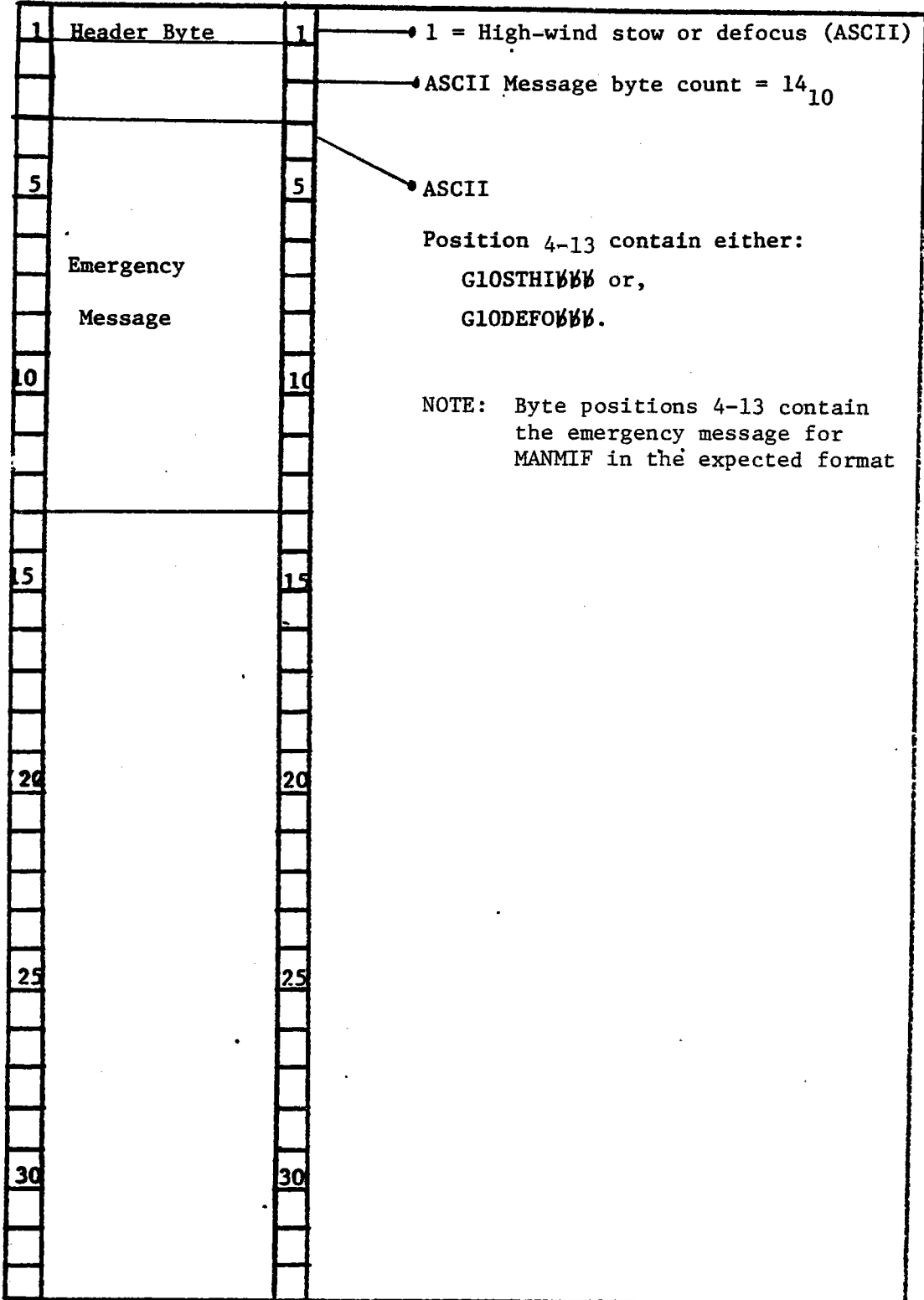


Table 3.2.9-II GDC-HAC GDC Command

MESSAGE LAYOUT

APPLICATION: GDC - HAC

MESSAGE TYPE: GDC Command

PROGRAMMER: D. Pettit

DATE: June 9, 1980

1	Header Byte	1	• 2 = Reset
5		5	
10		10	
15		15	
20		20	
25		25	
30		30	

Table 3.2.9-III GDC-HAC GDC Command

MESSAGE LAYOUT

APPLICATION: GDC - HAC

MESSAGE TYPE: GDC Command

PROGRAMMER: D. Pettit

DATE: June 9, 1980

1	Header Byte	1	<ul style="list-style-type: none"> → 3 = Full field or segment → ASCII Message byte count
5	Segment Number	5	<ul style="list-style-type: none"> → Position 4-6 contains segment number being displayed. Segment number 000 implies a full field display.
10		10	
15		15	
20		20	
25		25	
30		30	

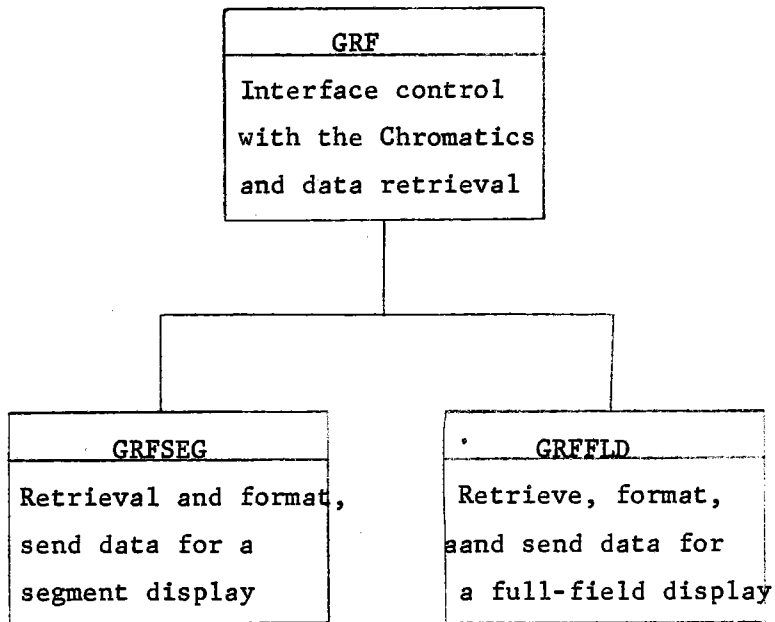


Figure 3.2.9-2 Functional Allocation of the GRF Task

Table 3.2.9-V HAC-GDC Text Message to GDC

MESSAGE LAYOUT

APPLICATION: HAC-GDC

MESSAGE TYPE: Text message to GDC

PROGRAMMER: D. Pettit

DATE: June 10, 1980

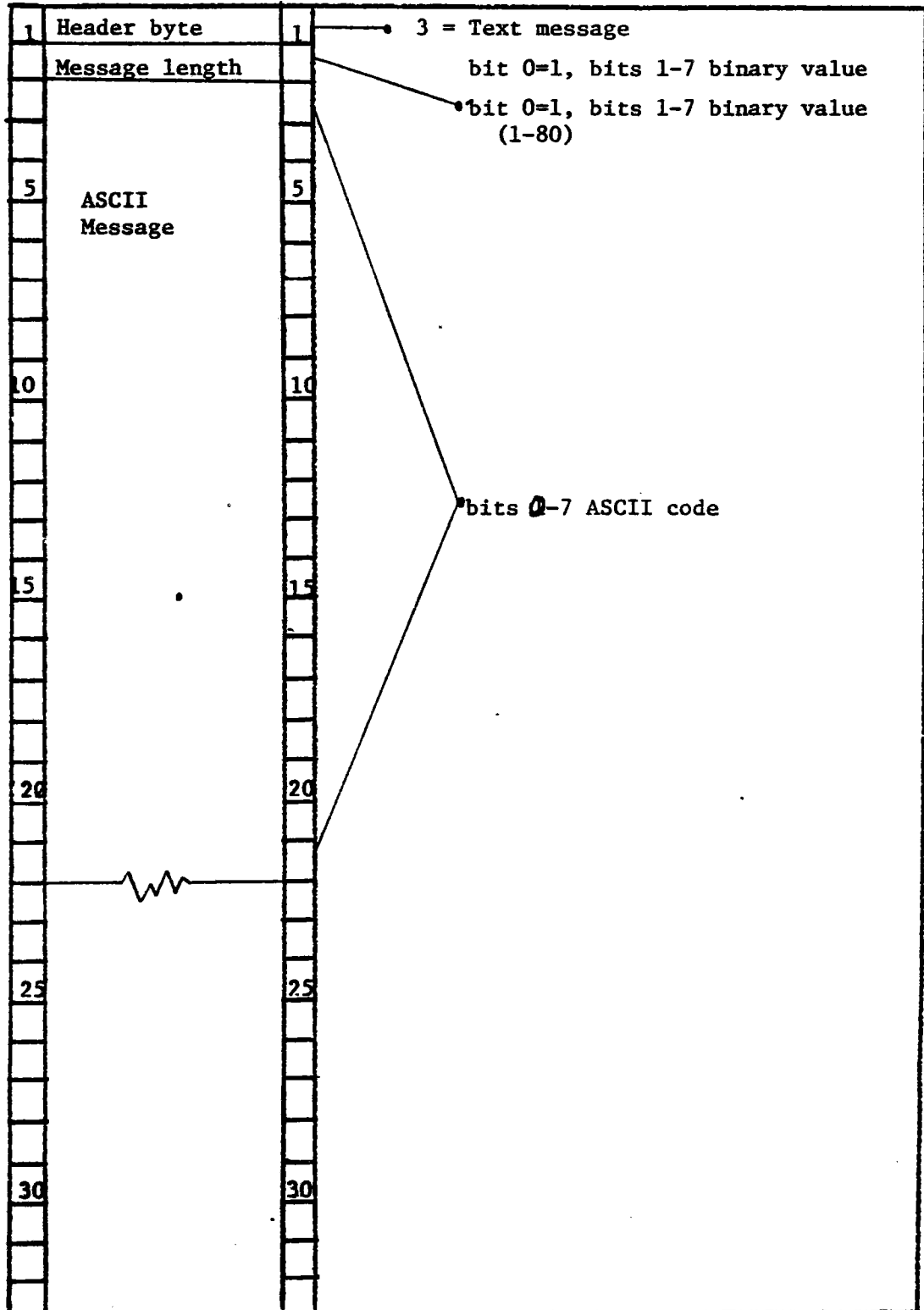


Table 3.2.9-VI HAC-GDC Graphics Initialization

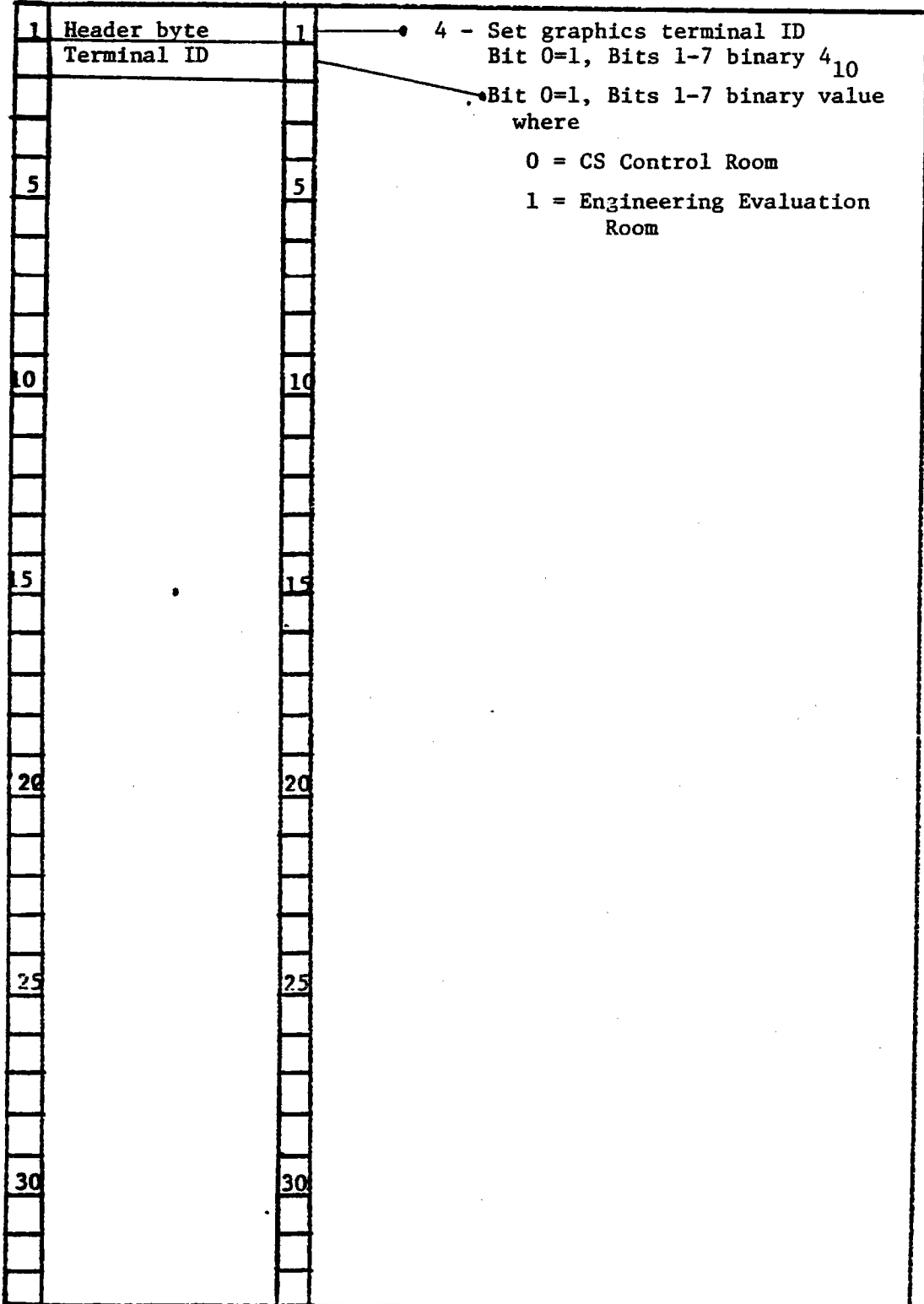
MESSAGE LAYOUT

APPLICATION : HAC-GDC

MESSAGE TYPE : Initialize Graphics

PROGRAMMER : D. Pettit

DATE : June 10, 1980



3. Message type 3 (Table 3.2.9-III) - Request status information for either a full-field display or a segment display, results in establishing either a full-field status message (Table 3.2.9-VII) or a segment status message (Table 3.2.9-VIII) - A segment number of zero implies send a full-field status; otherwise the segment number states which segment display is active. Pointers, counters, and source indicators are initialized to support the graphics request.

Note that the above three commands do not result in any direct output to the graphics processors, but they establish the response to be output upon receipt of the next message.

4. Message type 4 (Table 3.2.9-IV) - Send the next data or text stream to the Graphics Display Processor. Receipt of this message results in output being sent to the requesting graphics terminal. If a full-field status was requested, submodule GRFFLD is called to output the one-eighth of field status from the global common array, GRSTSG. If an individual segment status was requested, the submodule GRFSEG is called to output the status information required to support the currently active segment display. Otherwise a text message is output to the graphics.

This submodule dequeues one and only one command per activation. This implies that upon activation only one of the two graphics terminals will receive data, but only when the dequeued message is a type 4 command (send next message).

- b. GRFFLD-submodule supplies status for one-eighth of a full field. Formal parameters indicate for which one-eighth of the field to send status information, and another formal parameter specifies the destination. This submodule extracts status information from the global common array GRSTSG and packs it into the message format shown in Table 3.2.9-VII. After the message is formatted, it is output to the requesting graphics processor.
- c. GRFSEG - submodule supplies status data for the presently active segment display. Inputs, via the formal parameter list, are the total number of heliostats for status-ing, starting pecking-order number, source designator, and the segment number. The total number of heliostats must be positive and less than or equal to 21. The pecking-order number must be positive and less than or

Table 3.2.9-VII HAC-GDC Full Field Status

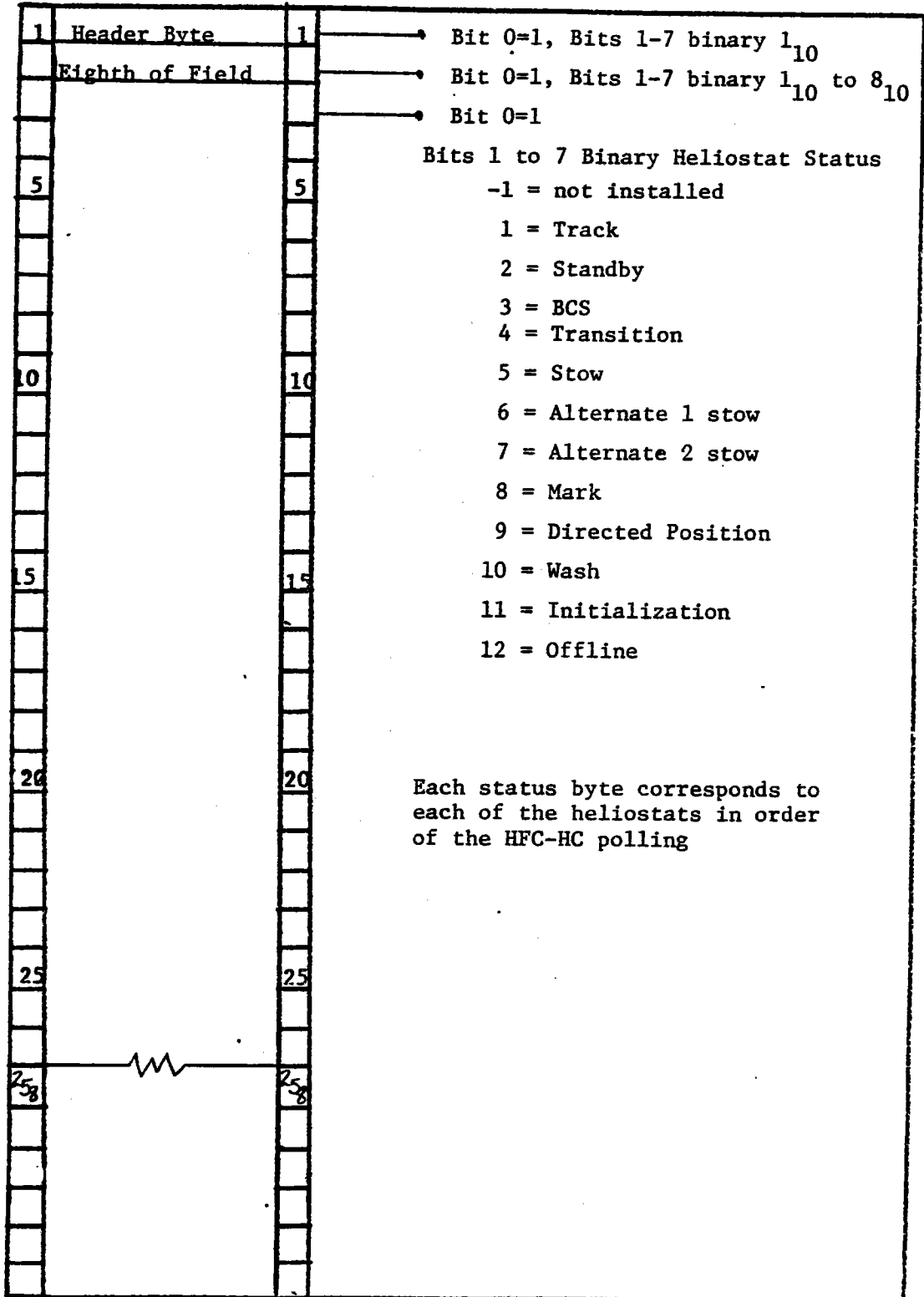
MESSAGE LAYOUT

APPLICATION : HAC-GDC

MESSAGE TYPE : HAC Full Field Status

PROGRAMMER : D. Pettit

DATE : June 9, 1980



equal to 50, and the segment number must be a valid segment designator. This submodule uses the formal parameters to process the segment's heliostats through the segment-to-heliostat mapping arrays, SEGPTG and SEGMPG. Once the heliostat number is obtained from the mapping, the present azimuth angle is obtained from the array AZIMG and the elevation angle is obtained from the array ELEVG. The present aim-point values and designator are obtained from the global array AIMPTG. The present status is obtained from the global common array GRSTSG. After all the information for a heliostat is collected, it is formatted and placed into the message format shown in Table 3.2.9-VIII. This procedure repeats for the total number of heliostats requested. When the entire message is built, it is output to the requesting graphics processor.

3.2.9.3.2 Resource Budgets

GRAPHIC uses approximately 2000 (estimated) words of memory. It is a low-priority module in the HAC system. It requires two Chromatic 1999 intelligent graphics consoles each interfaced to the MODCOMP through individual 4811s. One message queue is required to interface with the External Interface Module.

3.2.9.4.1.1 Submodule I - GRF (task)

3.2.9.4.1.1.1 Description

- a. Language used - FORTRAN
- b. How invoked - Activated by Enque in External Interface Module (EXTINF) or Deque in Graphics Display Module (GRAPHIC).
- c. Constraints and limitations - To support a segment graphics display, this task requires up to three activations to supply the full data stream.
- d. Processing -
 1. Set TBUSYG bit for Graphics and, dequeue a message from the Graphics Processor.
 2. Store the device number N, and store the message type number in local variables.
 3. If the message type number is one, set message code 1 word for device N, and go to step (32). Otherwise, go to step (4).
 4. If the message type number is two, set message

Table 3.2.9-VIII HAC-GDC HAC Segment Status

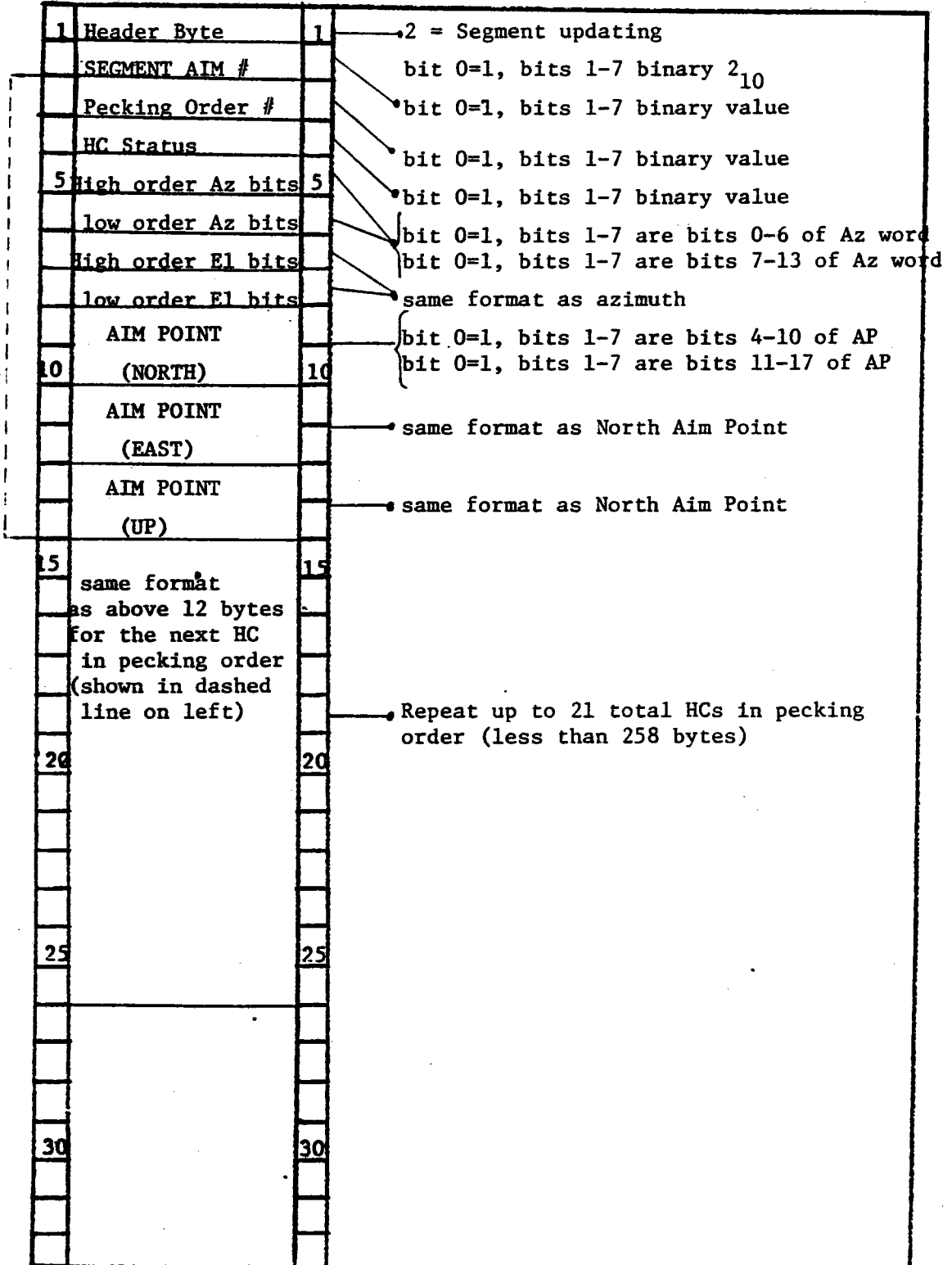
MESSAGE LAYOUT

APPLICATION: HAC-GDC

MESSAGE TYPE : HAC Segment Status

PROGRAMMER: D. Pettit

DATE June 9, 1980



- code 2 word for device N, and go to step (32). Otherwise, go to step (5).
5. If the message type number is three, go to step (6); otherwise, go to step (14).
 6. Get the segment number from types 4 to 6 of the message.
 7. If the segment number is zero, set message code 4 word and clear message code 3 word, both for device N. Set the one-eighth of field counter to one for device N, and go to step (). Otherwise, go to step (8).
 8. Get the number of HCs in the segment from global common.
 9. If the number of HCs is zero or the segment number is invalid, go to step (32). Otherwise, go to step (10).
 10. Set the number of HCs sent to zero for device N.
 11. Set message code 3 word and clear message code 4 word, both for device N.
 12. Retrieve the segment aim-point number from global common.
 13. Store the aim-point number for device N, and go to step (32).
 14. If the message type number is four, go to step (15); otherwise, go to step (32).
 15. If message code word 1 is set for device N, write an error message to device N, clear message code word 1 for device N, and go to step (32). Otherwise, go to step (16).
 16. If message code word 2 is set for device N, write the graphics reset message to device N, clear message code word 2 for device N, and go to step (32). Otherwise, go to step (17).
 17. If message code word 3 is set for device N, go to step (18); otherwise, go to step (29).
 18. Compute the number of HCs to send data for (number of HCs minus the number of HCs data output for), for device N.
 19. If the number of HCs to send data for is greater than 21, go to step (20); otherwise, go to step (24).

20. Compute the starting pecking-order number for device N.
21. Increment the number sent by 21 for device N.
22. Set the number to output to 21 for device N.
23. Set the corresponding HC numbers for device N into a buffer, and go to step (28).
24. Set the number to output equal to the number to send for device N.
25. Set the number sent for device N to zero.
26. Set the HC numbers for device N into a buffer.
27. Compute starting pecking-order number.
28. Call subroutine GRFSEG to output the segment display data for up to 21 HCs, and go to step (32).
29. Call subroutine GRFFLD to output one-eighth of the full-field status to graphics processor device N.
30. Increment the one-eighth of field counter for device N.
31. If the one-eighth of field counter is greater than or equal to eight, set the one-eighth of field counter to one, and go step (32). Otherwise, go to step (32).
32. Clear TBUSYG bit and EXIT this task.

- e. Error messages and recovery - Invalid segment number display requests result in no data output from this task. Receipt of a High-Wind Stow or Defocus command results in the error message:

EMERGENCY COMMAND NOT ALLOWED FROM THIS CONSOLE

An invalid message type from either graphics processor results in no data output to the graphics processor.

3.2.9.4.1.1.2 Data, Logic and Command Paths

Input data:

Graphic's messages from message queue tagged with a source code.

Output data:

- a. Reset message to a graphics processor; and

- b. ASCII text message to graphics processor.

Global Common:

- a. SEGMPG (segment mapping); and
- b. SEGPTG (segment pointer).

3.2.9.4.1.1.3 Internal Data Description

Two-word arrays are utilized to keep track of the internal command codes, counters, and buffers, and the two words correspond to the two graphics processors. The ASCII test message is initialized in a Data Statement, and most of the reset message is initialized in a Data Statement.

3.2.9.4.1.1.4 Flowchart

See Figure 3.2.9-3 for the GRF flowchart.

3.2.9.4.1.2 Submodule II-GRFSEG

3.2.9.4.1.2.1 Description

- a. Language used - Fortran
- b. How invoked - Subroutine call by GRF
- c. Limitations and constraints - Output buffer can only accommodate segment-display data for up to 21 heliostats.
- d. Processing -
 1. Fill word 1 of the message buffer with the Header byte and aim-point number byte. Set the leading byte bits on.
 2. Initialize two counters, I (HC counter) equal to one, and J (message word counter) equal to two.
 3. Fill message word J with the pecking-order number and HC status. Set the leading byte bits on.
 4. Fill message word J+1 with the high-order azimuth bits and the low-order azimuth bits. Set the leading byte bits on.
 5. Fill message word J+2 with the high-order elevation bits and the low-order elevation bits. Set the leading byte bits on.

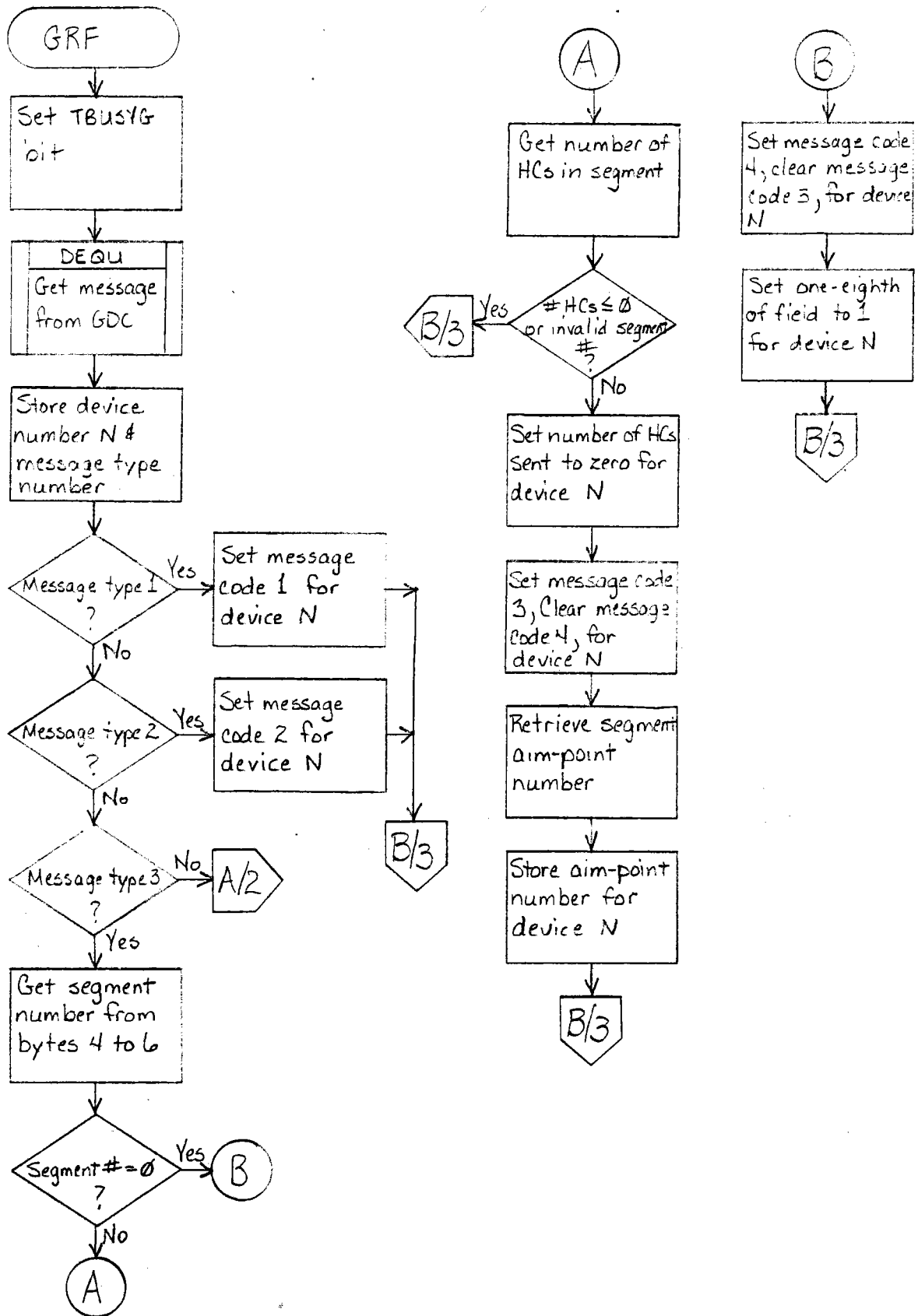


Figure 3.2.9-3 Flowchart - GRF

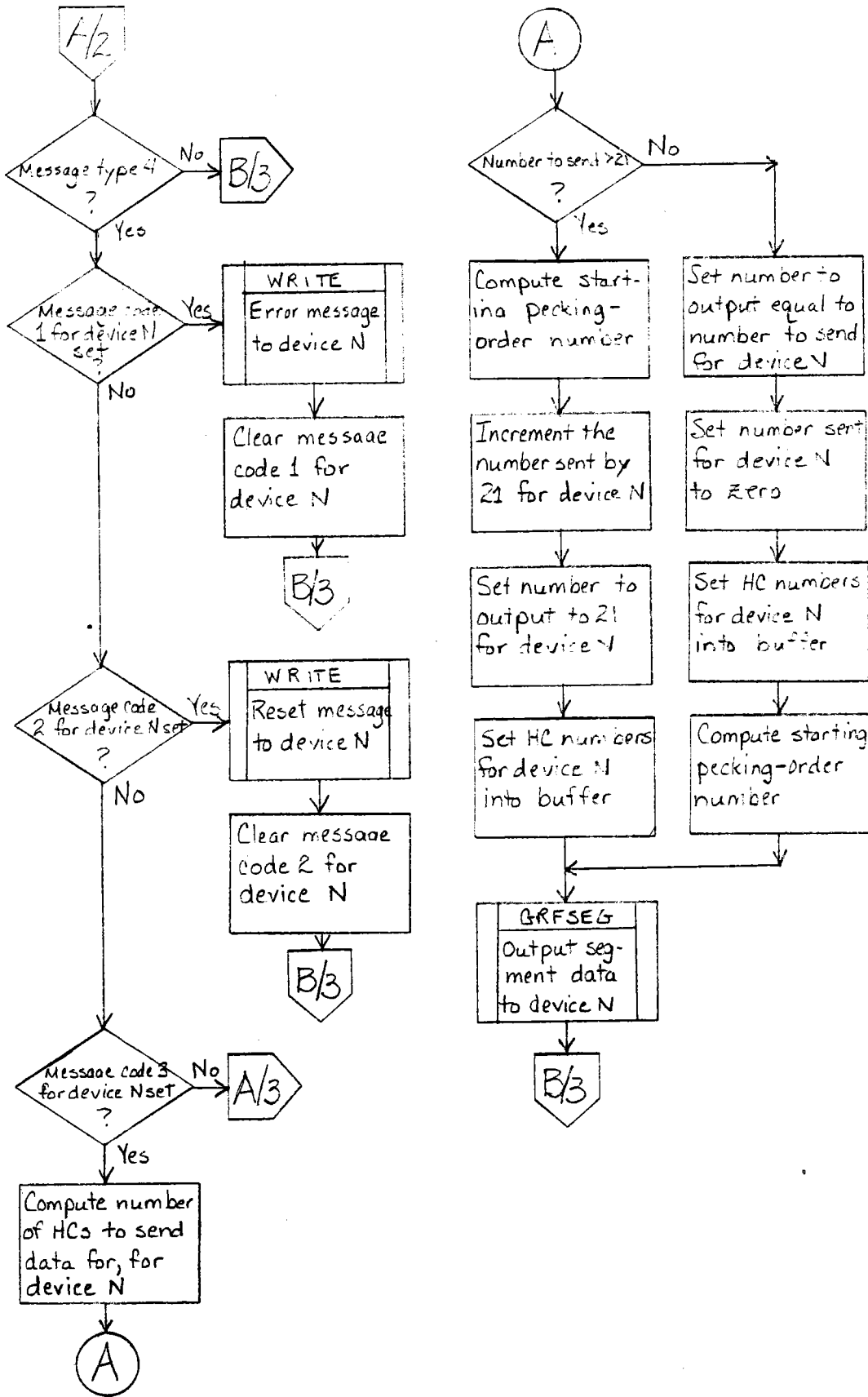


Figure 3.2.9-3 Flowchart - GRF (continued)

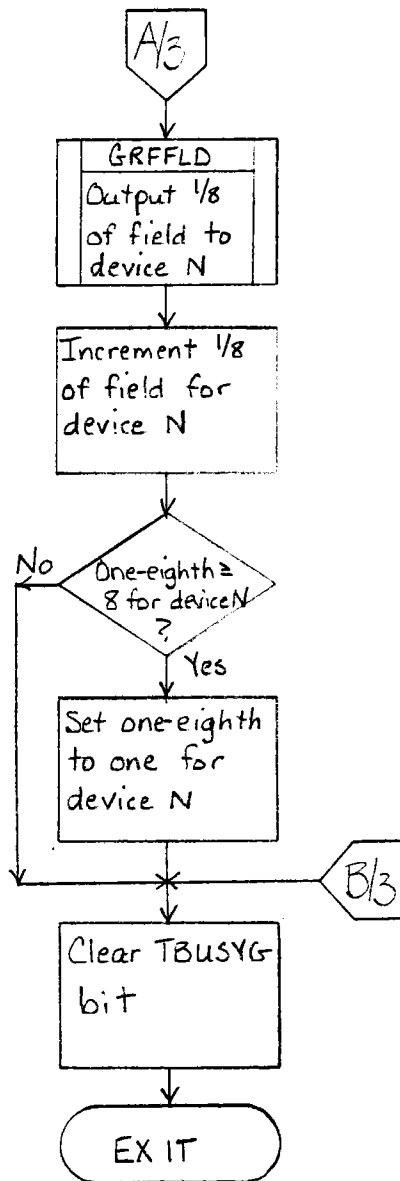


Figure 3.2.9-3 Flowchart - GRF (continued)
631

6. Fill message word J+3 with the north aim-point. Set the leading byte bits on.
7. Fill message word J+4 with the east aim-point. Set the leading byte bits on.
8. Fill message word J+5 with the up aim-point. Set the leading byte bits on.
9. Increment the message word counter J by six, and increment the HC counter by one.
10. If the number of HCs is greater than the counter I, go to step (3). Otherwise, go to step (11).
11. Write the message buffer to the graphics processor (device N).
12. Return to the calling submodule.

e. Error messages and recovery - None

3.2.9.4.1.2.2

Data, Logic and Command Paths

Input Data:

- a. Aim-point number;
- b. Pecking-order starting number;
- c. HC numbers in an array;
- d. Number of HCs to supply data for; and
- e. Number of device to write message to.

Output data:

HAC segment status message

Global common used:

- a. AIMPTG (aim points);
- b. AZIMG (azimuth angle);
- c. ELEVG (Elevation angle); and
- d. GRSTSG (heliostat status).

3.2.9.4.1.2.3

Internal Data Description

Bit masks are utilized to extract bits from the aim-points and

azimuth and elevation angles. The UFT array used by the output routine, WRITE, is initialized in Data Statements, except word I, the device-dependent word. A 258-byte message buffer is allocated.

3.2.9.4.1.2.4 Flowcharts

See Figure 3.2.9-4 for the GRFSEG flowchart.

3.2.9.4.1.3 Submodule III-GRFFLD

3.2.9.4.1.3.1 Description

- a. Language used - FORTRAN
- b. How invoked - Subroutine call by GRF
- c. Constraints and limitations - The one-eighth of field requested must be in the range one to eight.
- d. Processing -
 1. Retrieve the status words for the one-eighth of the field and store into a 256-word buffer.
 2. Fill message word one with the header byte and the one-eighth of field value. Set the high-order byte bits on. Initialize two counters I (HC counter) equal to one, and J (message and count) to two.
 3. Set bits 0 and 8 of message word J.
 4. If the i'th word of the status buffer is equal to -1, set the i'th word of the status buffer to HEX 7F00, and go to step (5). Otherwise, go to step (5).
 5. Shift status buffer word I left 8 places to put the value in the high-order byte.
 6. If the i'th+1 word of the status buffer is -1, set the status buffer word to Hex 007F, and go to step (7). Otherwise, go to step (7).
 7. Add status buffer word I and status buffer word I+1 to message word J, and store the results in message word J.
 8. Increment the counter J by one and I by two.
 9. If I is greater than or equal to 257, go to step (10). Otherwise, go to step (3).
 10. Call subroutine WRITE to output the message to graphics processor N.

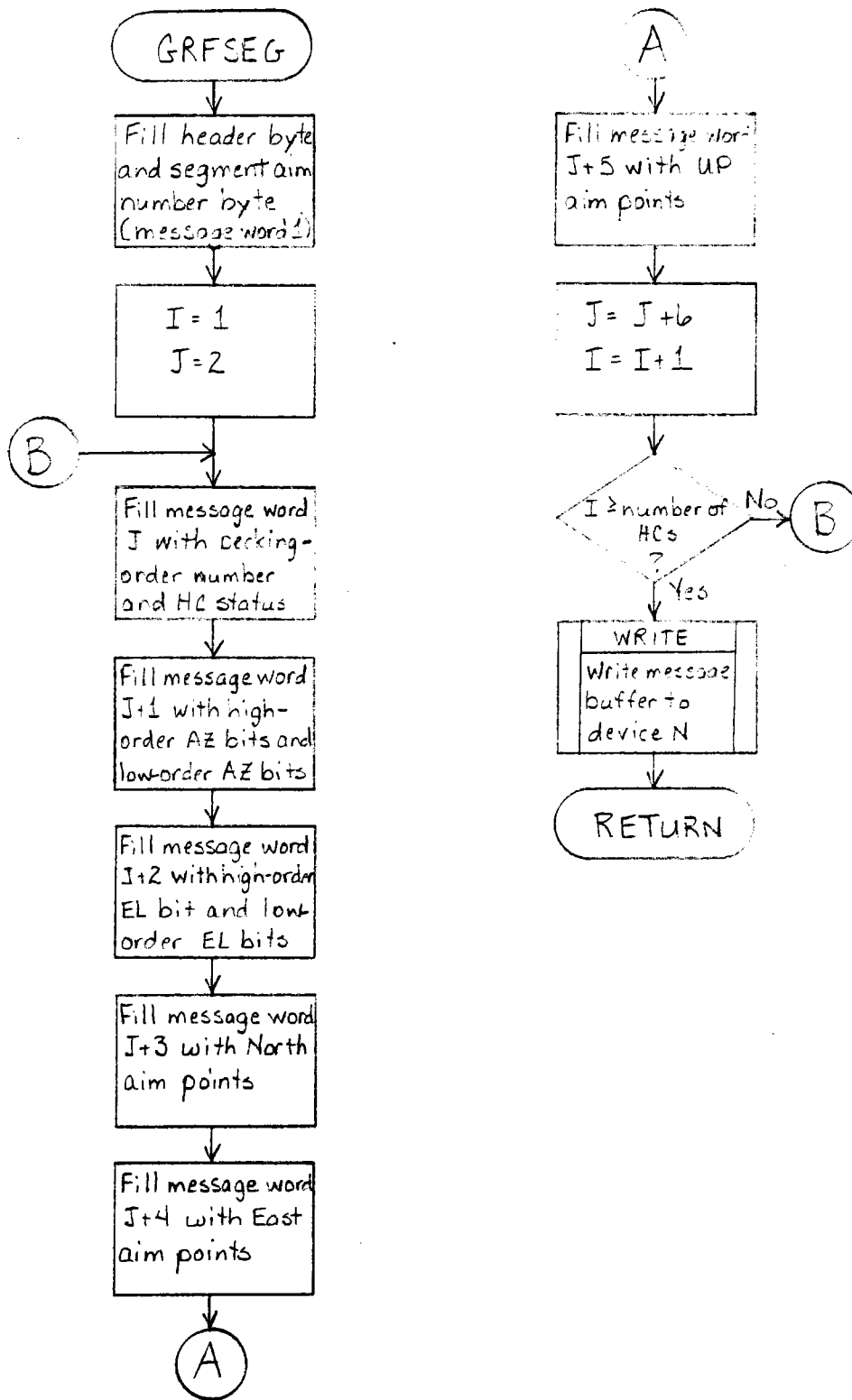


Figure 3.2.9-4 Flowchart - GRFSEG

11. Return to the calling submodule.

e. Error messages and recovery - None

3.2.9.4.1.3.2 Data, Logic and Command Paths

Input data:

- a. One-eighth of field (range 1 to 8); and
- b. Graphics processor device number.

Output data:

HAC full-field status message

Global common:

GRSTSG (heliostat status)

3.2.9.4.1.3.3 Internal Data Description

Variables for the Hex values 007F, 7F00, and 8080 are initialized in Data Statements. A 256-word buffer and a 129-word are allocated for use. The UET array is initialized in a Data Statement, except word 1, the device-dependent word.

3.2.9.4.1.3.4 Flowchart

See Figure 3.2.9-5 for the GRFFLD flowchart.

3.2.9.5 Interface Description

The task GRF interfaces with the External Interface Module (EXTINF) (Figure 3.2.9-1) via the task CSI. CSI enqueues messages from the graphics processors and GRF dequeues these messages. GRF outputs directly to the graphics processors with data messages. These message formats are shown in Tables 3.2.9-I to 3.2.9-VIII.

3.2.9.6 Test Requirements

Construct field status arrays, write the data to the graphics processors and visually inspect the color display for verification.

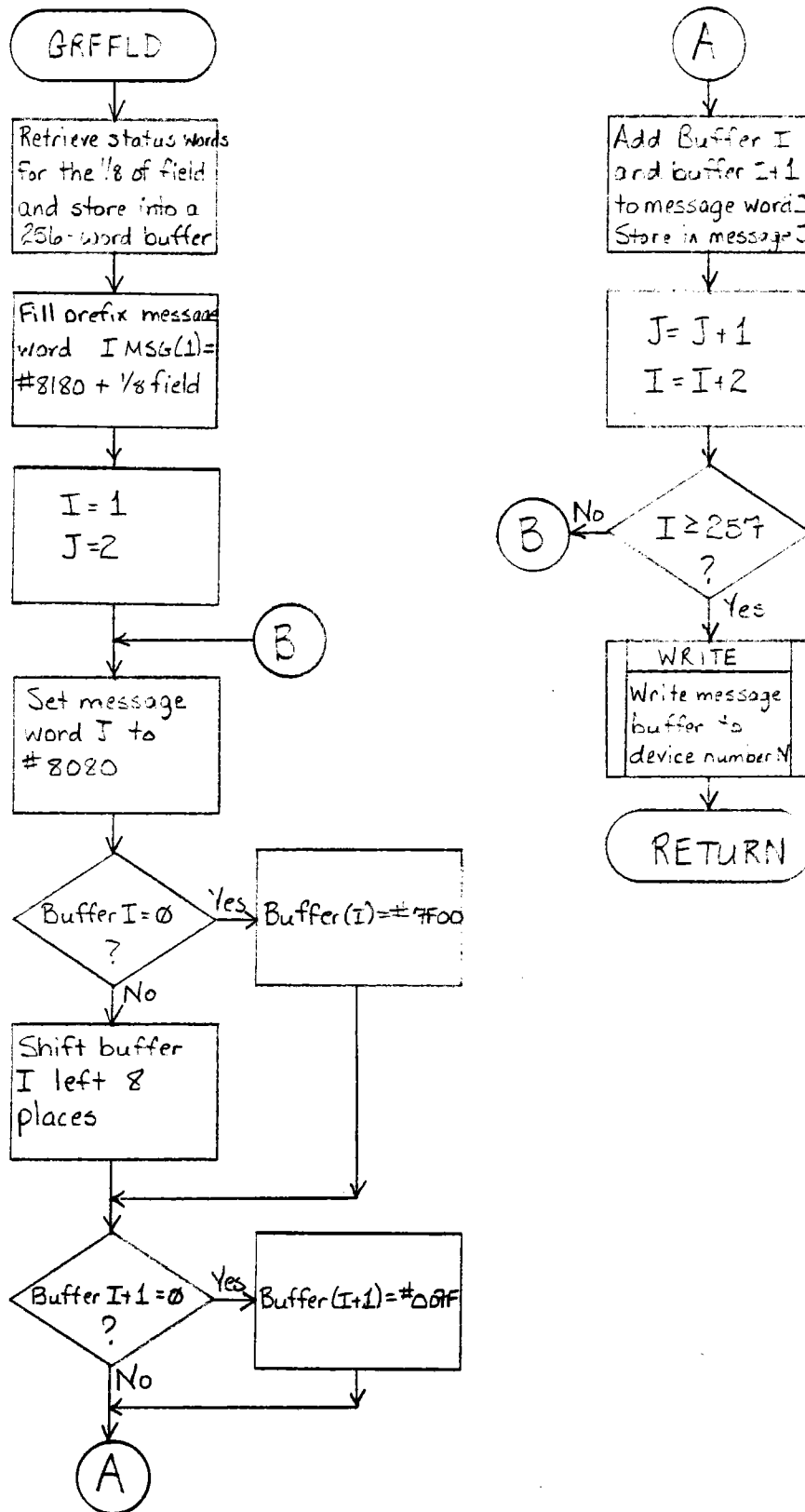


Figure 3.2.9-5 Flowchart - GRFFLD

3.2.10 External Interface Module - EXTINF

3.2.10.1 Purpose

The External Interface Module is responsible for maintaining the communication interface between the HAC computer and various external sources of I/O. These sources include the following:

- a. OCS computer;
- b. DAS computer;
- c. ISC console (CS control console);
- d. Graphics console 1 (CS control room); and
- e. Graphics console 2.

The EXTINF module is divided into the following submodules:

- a. External Interface Input (EXI) - task which is responsible for communicating with the OCS and DAS computers via the MAXNET operating system;
- b. External Interface Output (EXO) - task responsible for outputting responses to the OCS and DAS computers.
- c. Console Input (CSI) - task which is responsible for accepting inputs from the ISC and the two graphics terminals and sending these as messages to the responsible tasks; and
- d. Console Output (CSO) - task responsible for outputting to the ISC for the BCSMOD, MANMIF, STATUS, and ALARMS modules.

3.2.10.2 Requirements

3.2.10.2.1 Design Requirements

Section 3.1 of the Software/Firmware Functional Requirements Specification states the following requirements for the EXTINF module:

- a. Maintain command/response protocol and data transfer with the OCS;
- b. Provide status data to the DAS; and
- c. Monitor and display the operational status of all heliostats.

3.2.10.2.2 Derived Requirements

Section 3.2.1.10 of the Software/Firmware Functional Requirements Specification states the following derived requirements for the External Interface Module:

- a. Interface with the CS Control Console by:

1. Accepting inputs and passing these inputs to the respective modules.
 2. Regulating the outputs to the CRT so that only one submodule is outputting at a time.
- b. Interface with the Chromatics Terminals by:
1. Pass emergency STHIWIND and DEFOCUS commands to MANMIF module.
 2. Data requests to the GRAPHC module.
- c. Interface with the OCS computer by accepting commands; and
- d. Interface with the DAS by accepting status requests.

Additional derived requirements include:

- a. The requirement of interfacing with the Chromatics and CS control console implies the need for performing I/O to these devices; and
- b. Outputting to the CS control console implies a need to establish a message interface between the CSO task and the BCSMOD, MANMIF, STATUS, and ALARMS modules.

3.2.10.3

Design Approach

The External Interface module, to facilitate communications with the various external sources, is divided into tasks logically split into MAXNET and non-MAXNET interfaces. These external sources consist of the OCS, DAS and backup HAC computers, the CS control console, and the two Chromatics Graphics Terminals. The MAXNET capabilities are used to communicate with the OCS, DAS and backup HAC computers. The EXI task associated with the OCS/DAS interface is responsible for input messages from those computers. The EXI task processes messages from the OCS/DAS computers at a rate of one message from either computer per second. To maintain status on the backup HAC, the EXI task will request task status information once each second for the SWI task.

The EXO task will accept messages in response to OCS commands and DAS status requests and transfer them to the OCS and DAS computers.

There are two tasks associated with communications with the CS consoles and Chromatics Graphic Terminals. The CSI task is responsible for processing input from these consoles and activating the responsible module to process the input. The CSO task is responsible for outputting to the CS control console. User tasks wishing to display information on the CS control console queue messages to the CSO task.

3.2.10.3.1

Functional Allocations

To accomplish the required processing, the External Interface Module consists of submodules EXI, EXO, CSI, and CSO (see Figure 3.2.10-1). A brief description of the function performed by each of these submodules is given below.

- a. EXI - this task performs the processing required to support the communications interface with the OCS, DAS, and backup HAC computers. The command/response protocol between the computers is supported via the MAXNET operating system. Data transfer between the CS and the OCS, DAS, and backup HAC computers is accomplished using the MAXNET capabilities to read and write data between computers.
- b. EXO - This task performs processing required to send response messages to the OCS and DAS computers. The responses that will be dequeued from the message queue are COMQUE and sent to the OCS and DAS computers via core device capabilities.
- c. CSI - The CSI task interfaces with the ISC 8001G color graphic terminal and the two Chromatics Graphics Terminals. This task is responsible for inputting either operator commands from the ISC terminal or data messages from the Chromatic Terminals and queueing the message to the appropriate task for processing.
- d. CSO - The CSO task is responsible for outputting status messages and operator response messages to the ISC console.

3.2.10.3.2

Resource Budgets

- a. Memory requirements:

CSI task -	3400	
CSO task -	2500	
EXI task -	2800	
EXO task -	1900	
<u>Total -</u>	<u>10,600</u>	words of resident memory

- b. Mass storage (disk, tape) required: None; and
- c. Task Priority: The CSI task shall be of priority 100. The CSO task shall be of priority 110. The EXI task priority is 120. The EXO task priority is 122.

3.2.10.4

Design Description

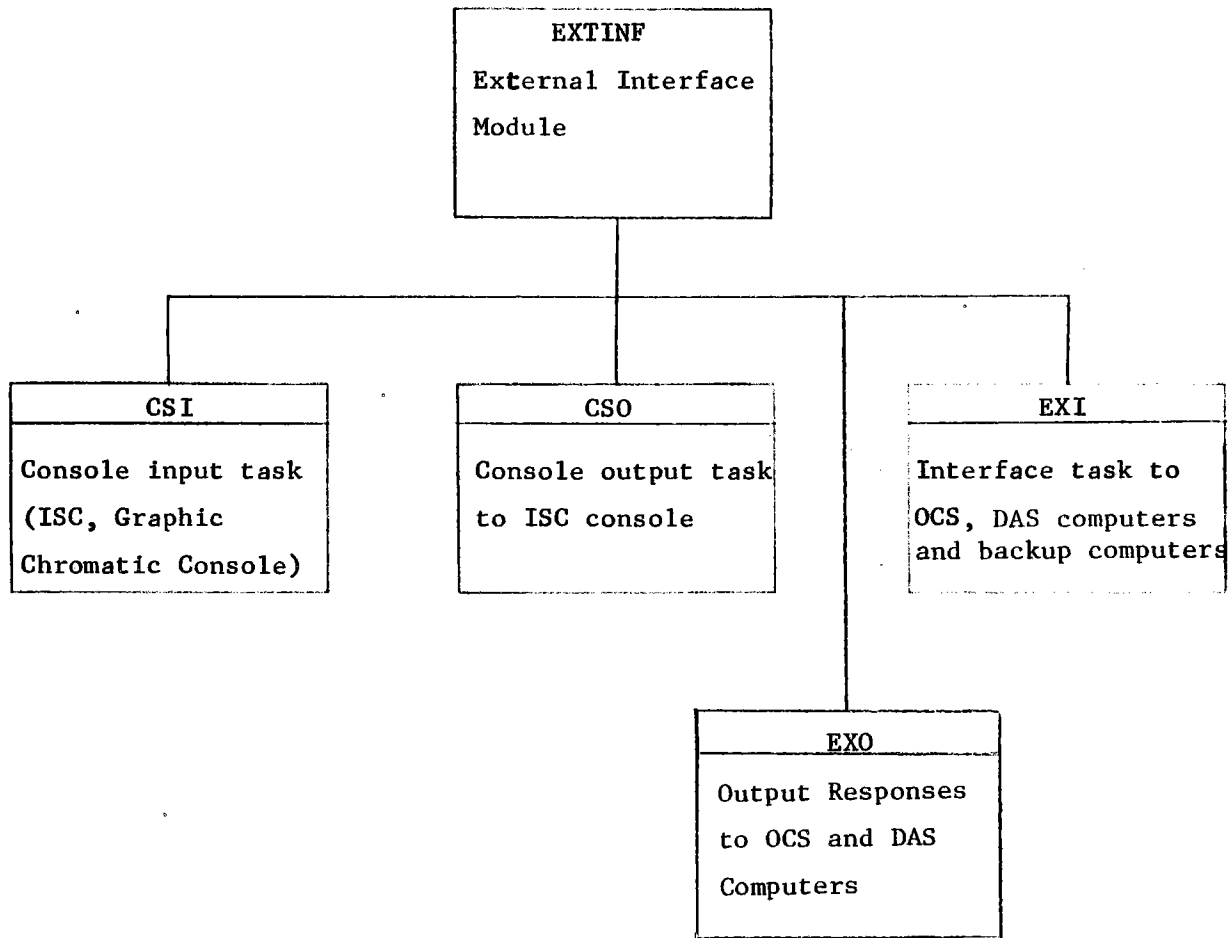


Figure 3.2.10-1 External Interface Module

3.2.10.4.1

Module Structure

CSI and CSO are tasks associated with inputting and outputting to the CS control console and Chromatics Graphics Terminals. The EXI and EXO tasks communicate with the OCS and DAS computers.

3.2.10.4.1.1

Submodule I - CSI Task (Main Routine)

3.2.10.4.1.1.1

Description

- a. Language used - MODCOMP FORTRAN IV
- b. How invoked - Activated by DBINIT during system initialization. Active throughout the operation of the CS computer.
- c. Constraints and limitations - The CSI task inputs one character at a time from the operator's console.
- d. Processing - When activated, the CSI task opens input buffers for the CS console and the two Chromatics Graphics Terminals via a REX Read. After initiating the read operations, the CSI task executes a REX I/O Wait, suspending itself until one of the input operations completes.

The MAX IV operating system will resume the CSI task upon completion of one of the input buffers. Upon being resumed, the CSI task examines all the input buffers to see which one has completed. If the ISC console buffer completed, the CSI console will move the first character input to an internal buffer and reinitiate another character-read operation. The character just read is then written back to the CS console to display the character to the operator.

If the character was a carriage return or if 80 characters have been input, signaling the end of an input, the CSI task will examine the internal buffer for an emergency command. An emergency command is queued to the MMI task's emergency queue; others go to the CS operator command queue.

Input buffers received from the Chromatics Graphics Terminals will be given to the GRF task for processing with an indicator of which graphics console sent the input.

A check is made of the input buffer for the Chromatics terminal in the CS control room to determine if the input was an emergency command. If an emergency command was found, the command is queued to the MMI task's emergency queue.

- e. Error messages and recovery - The only errors which occur are I/O errors during communication with the various consoles. When an I/O error is found, an error message is output to the operator indicating which interface is in error, and the UFT is output for diagnostic purposes.

3.2.10.4.1.1.2 Data, Logic, and Command Paths

Input data:

- a. The input data from the consoles;
- b. The error status from the MAX IV (MAXNET) operating system;
- c. CRTWDG word in global common COMLS5; and
- d. SPARE5(4) - word in global common COMLS5

Output data:

- a. Input message sent to the responsible task; and
- b. Global common word CRTWDG in COMLS5.

3.2.10.4.1.1.3 Internal Data Description

- a. 3 User File Tables (UFTS) for input buffers;
- b. 2 1-word task names;
- c. 1 40-word internal buffer for building ISC messages;
- d. 4 2-word tables to hold the ASCII representations of the emergency commands; and
- e. Miscellaneous pointers and indices.

3.2.10.4.1.1.4 Flowchart(s)

See Figure 3.2.10-2 for the CSI flowchart.

3.2.10.4.1.2 Submodule II - CSO Task

3.2.10.4.1.2.1 Description

- a. Language used - MODCOMP FORTRAN IV
- b. How invoked - Invoked via an Rex Enque call by either the MMI, BCS, CFO, DSK, or STA tasks when an output to the ISC console is required.

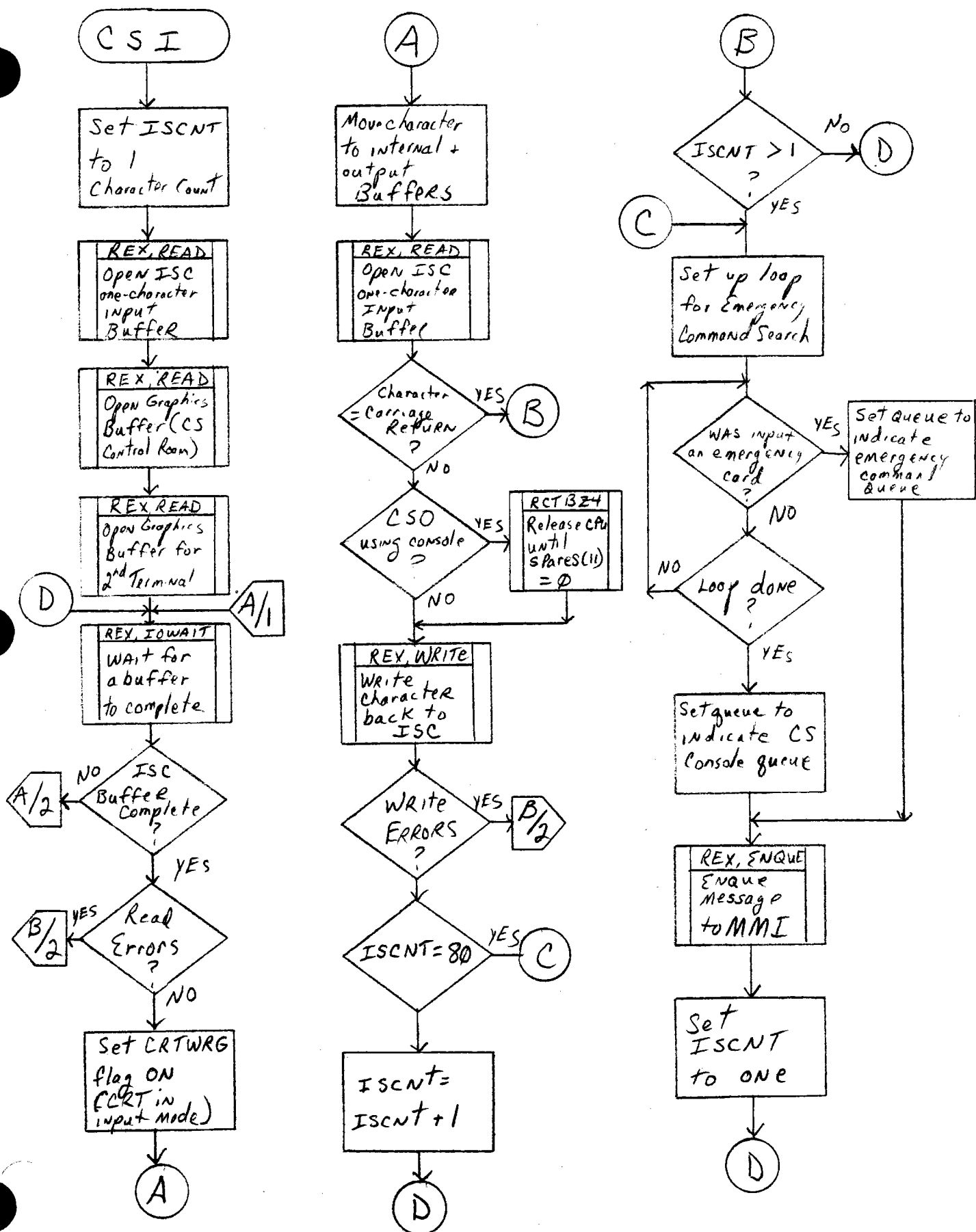


Figure 3.2.10-2 Flowchart - CSI

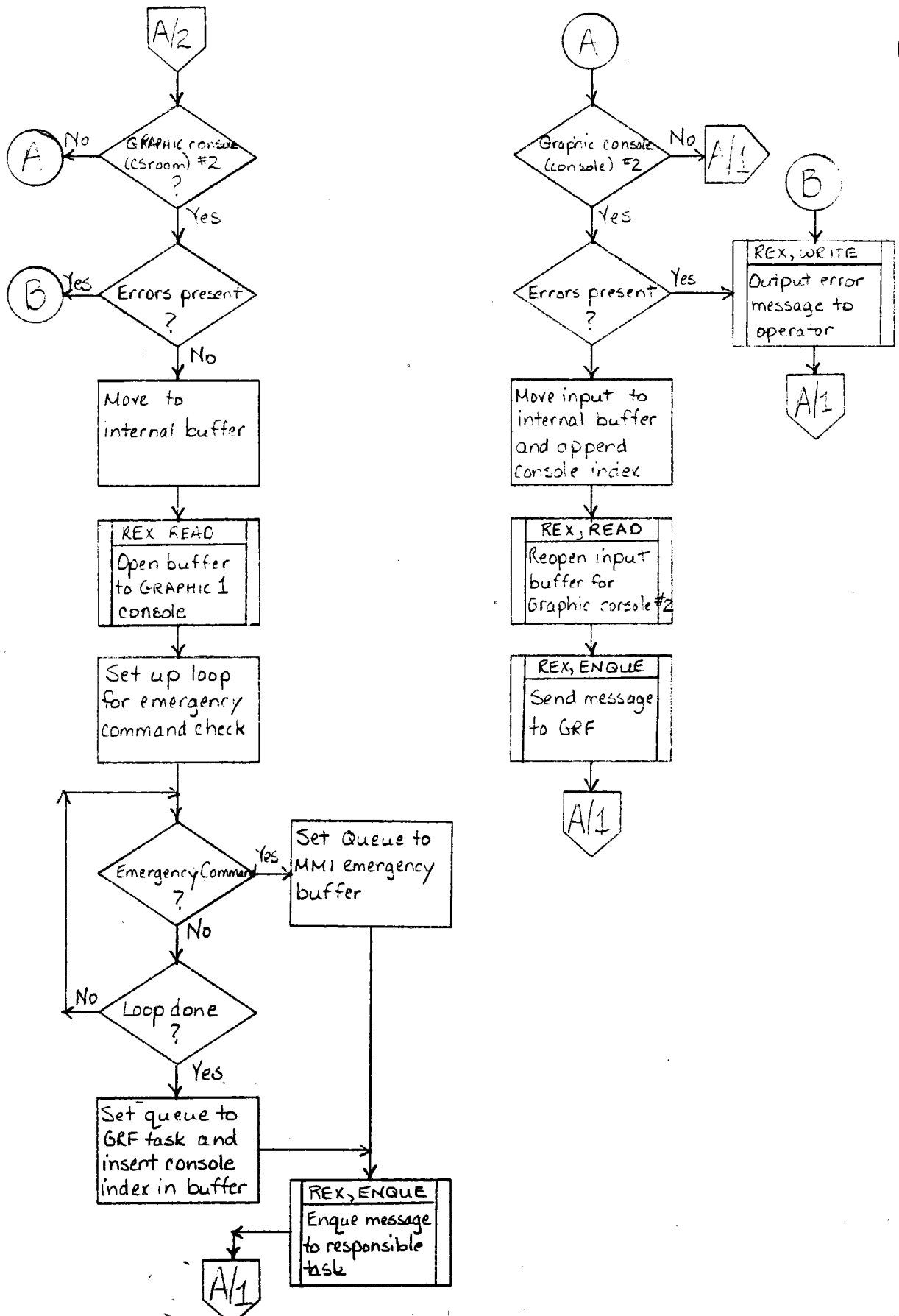


Figure 3.2.10-2 Flowchart - CSI (Continued)

- c. Constraints and limitations - Only one output line (maximum of 80 characters) can be queued at a time. If a line is to be output to the conversational area (lines 17-48 where operator places commands), then the message should be placed in queue number seven (7). If the user is supplying the control characters and placing the output into a reserved area of the screen (i.e., alarms area, status area, etc.), then the message with appropriate control characters should be placed in queue number twelve (12).
- d. Processing - When activated, the CSO task will deque from queue 12 first and determine if a message is present. If so, CSO will build the user file table and output the user's message. To avoid conflicting with the CSI tasks echoing command characters, a word of global common, COMIS5 - SPARE5(4), is used. CSO will set bit 15 of SPARE5(4) to slow the rate of I/O transmissions to the ISC console. Following output completion, CSO will clear this bit and allow CSI to continue.

If no message is found in queue 12, then queue 7 is checked for a message to be output to the conversational area of the ISC console.

If no message is present, the CSO task will simply exit and await further activations.

If a message is found, the CSO task will append control characters on the message and output to the next line on the screen. Before a write is initiated, a flag is checked in global common word COMIS5 - CRTWOG bit 15. If the flag is set, the CSO task wait for one second to allow CSI task to finish its output and then check the flag again. This check will prevent the operator commands from being overwritten with CSO output.

- e. Error message and recovery - When an I/O error is detected for the write operation, an error message will be output. The error message that is typed is:

"CS CONSOLE OUTPUT ERROR:

and the UFT is output for diagnostic purposes.

3.2.10.4.1.2.2 Data, Logic and Command Paths

Input data:

- a. Includes the output message queued by the MMI, STA, CFO, DSK and BCS tasks;
- b. The CS console word in global common for the CRT usage flag, (bit 0 in CRTWDG);
- c. CRT line number, (bits 8 - 15 in CRTWDG); and
- d. BCSLNG (global common indicates the number of lines BCSMOD is using).

Output data:

- a. CRTWDG word in global common COM1S5;
- b. SPARE5(4) word in global common COM1S5.

3.2.10.4.1.2.3 Internal Data Description

- a. 1 User File Table (UFT) for the output buffer to the ISC;
- b. 1 internal buffer for holding the output message; and
- c. Miscellaneous internal pointers.

3.2.10.4.1.2.4 Flowchart(s)

See Figure 3.2.10-3 for the CSO flowchart.

3.2.10.4.1.3 Submodule III - EXI Task

3.2.10.4.1.3.1 Description

- a. Language used - MODCOMP, FORTRAN IV
- b. How invoked - Activated in the prime HAC by CLK upon system startup. In the backup the SWI task activates the EXI task upon HAC failover.

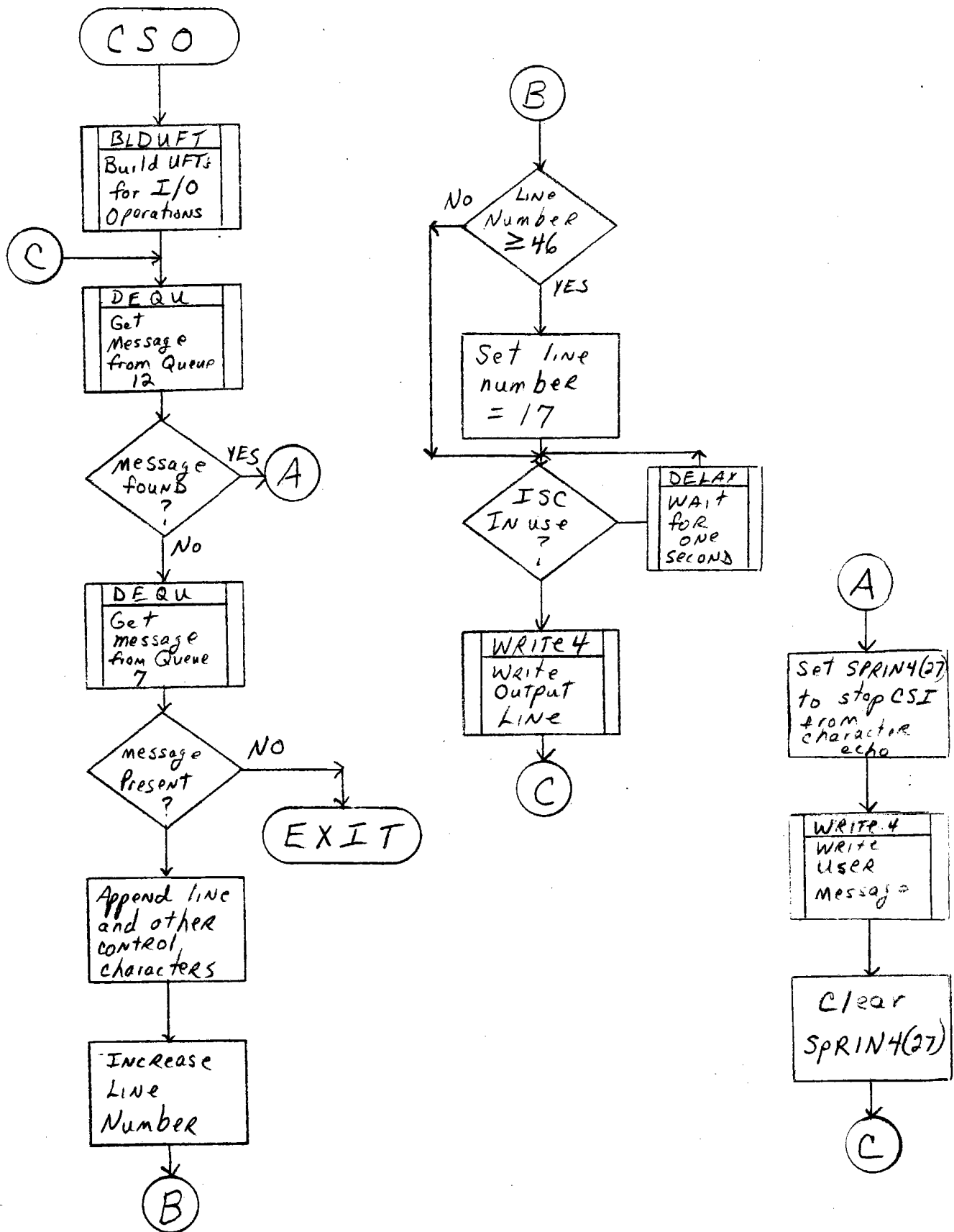


Figure 3.2.10-3 Flowchart - CSO

- c. Constraints and limitations - The EXI task will process one message per core device per second. There are three core devices for message transfers between the HACs and OCS and DAS. Core devices XST (status request) and XCS (commands) are core devices resident in the HAC and used by the OCS computer. The XOA core device is the core device used by the DAS computer for its status requests.
- d. Processing - When activated, the EXI task builds (via calls to the system routine BLDUFT) the user file tables (UFTs) that are required to perform the read operations to the core devices.

The EXI task will initiate read operations to three core devices: XCS, XST, and XDA resident in the HAC computer. After initiating the read operation, the EXI task will delay one second to allow the read operations to complete. After the delay, the EXI task will check all three user file tables to see if any read operations are finished. If a message was found in the core device, the EXI task will send the message to the MMI task for further processing.

A check will also be made to determine if the message received was an emergency message; and if so, the message will be placed in queue zero (\emptyset), MMI task's emergency queue. Upon receipt of the first message from the OCS computer, the EXI task will set a status bit in word SPARE5(5) bit 15 to indicate the OCS interface is operational, and call subroutine ALMQUE to notify the operator. Bit 14 of SPARE5(5) is used to indicate the DAS interface is operational.

After checking all the core devices, the EXI task will interrogate the backup HAC to determine if the backup HAC is operational. This enables the prime HAC to maintain the current operability status of the backup HAC. The request is made using the MAXNET special service request REMIFO, if the SWI task is established and active in the backup HAC. If the SWI task is not established, the backup bit is cleared and an alarm queued to the operator via a call to the ALMQUE subroutine.

- e. Error messages and recovery - Errors detected during the read operations to the OCS or DAS computer result in an error message to the operator indicating which interface has the error, and the UFT is output for diagnostic purposes. If a failure is detected in the backup HAC, a critical alarm is raised at the CS console

3.2.10.4.1.3.2

Data, Logic, and Command Paths

Input data:

- a. 80-byte commands from the OCS and DAS computers;
- b. Task status from backup HAC via MAXNET.

Output data:

- a. Commands to MMI;
- b. OCS/DAS status bits in SPARE5(5);
- c. Remote task information request.

3.2.10.4.1.3.3 Internal Data Description

- a. Three user file tables (each eight words long)
- b. Three 80-byte buffers to accept messages
- c. Various internal pointers
- d. One seven-word buffer to hold returned task status

3.2.10.4.1.3.4 Flowchart

See Figure 3.2.10-4 for the EXI flowchart.

3.2.10.4.1.4 Submodule IV - EXO task

3.2.10.4.1.4.1 Description

- a. Language used - FORTRAN
- b. How invoked - Activated by either MMI, ALO, or STA tasks via message enques in response to OCS/DAS requests.
- c. Constraints and limitations - There are four queues that the EXO task will use to transfer messages to the OCS and DAS computers. They are as follows:
 - (1) Queue 14 - Status responses to OCS
 - (2) Queue 15 - Command responses to OCS
 - (3) Queue 16 - Alarm responses to OCS
 - (4) Queue 17 - Status responses to DAS

Queues 14 and 16 are examined at a rate of ten times per second; whereas, queues 15 and 17 are serviced once per second.

- d. Processing - When activated, the EXO task will check all the message queues to see if any messages are to be sent to either the OCS or DAS computers. If a message is found, a write operation is done to the appropriate core device in either the OCS or DAS computers. There are four core devices that the HAC will write to; three in OCS computer, and one in the DAS computer.

All message queues are checked with each activation, and EXO will remain executing and sending messages until no more messages are present in any of the queues.

If an error is found attempting to write to the OCS or DAS computers, the corresponding bit (either 14 for OCS or 15 for DAS) is cleared in SPARE5(5), and an alarm sent to the operator.

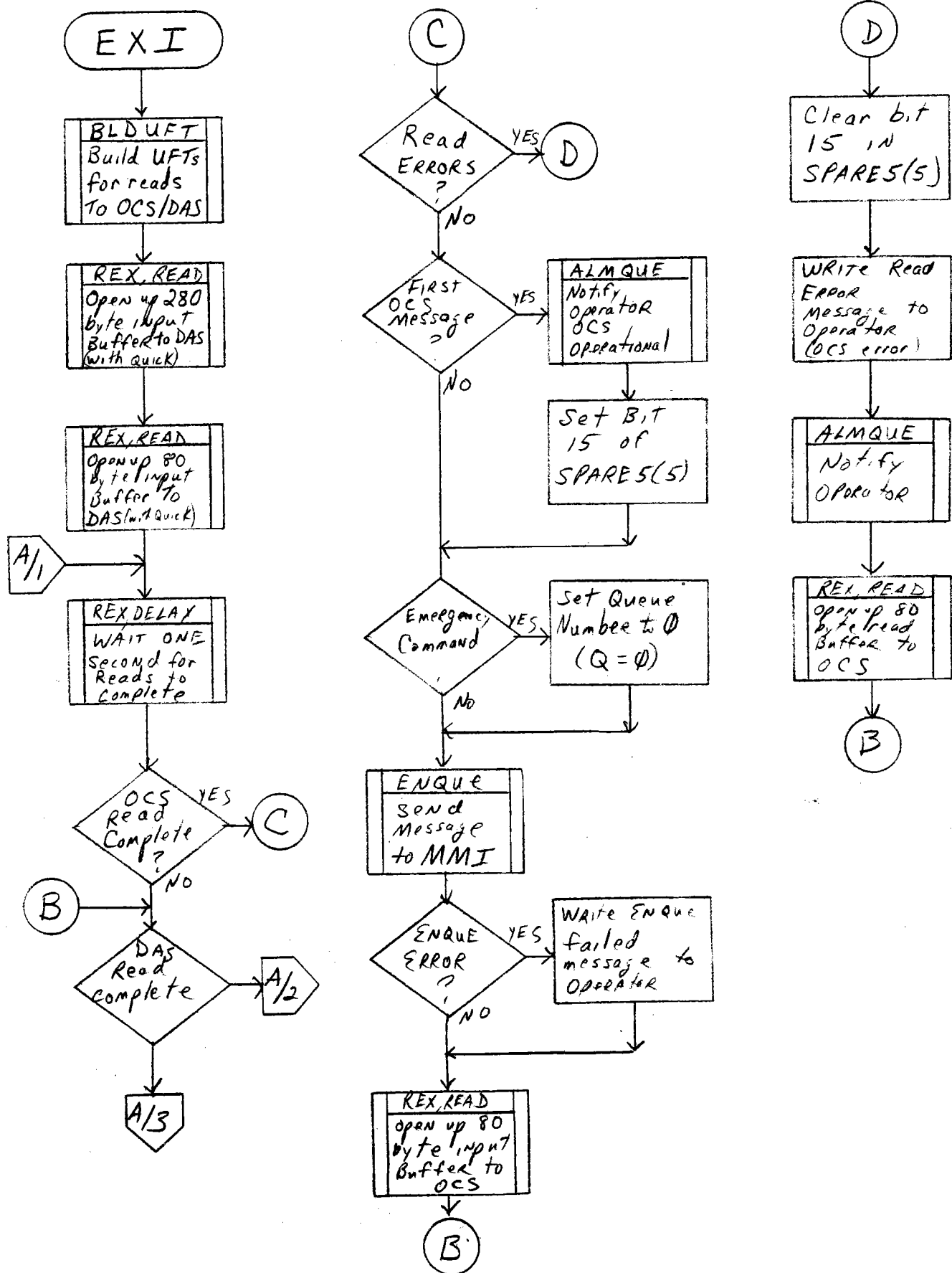


Figure 3.2.10-4 Flowchart - EXI
651

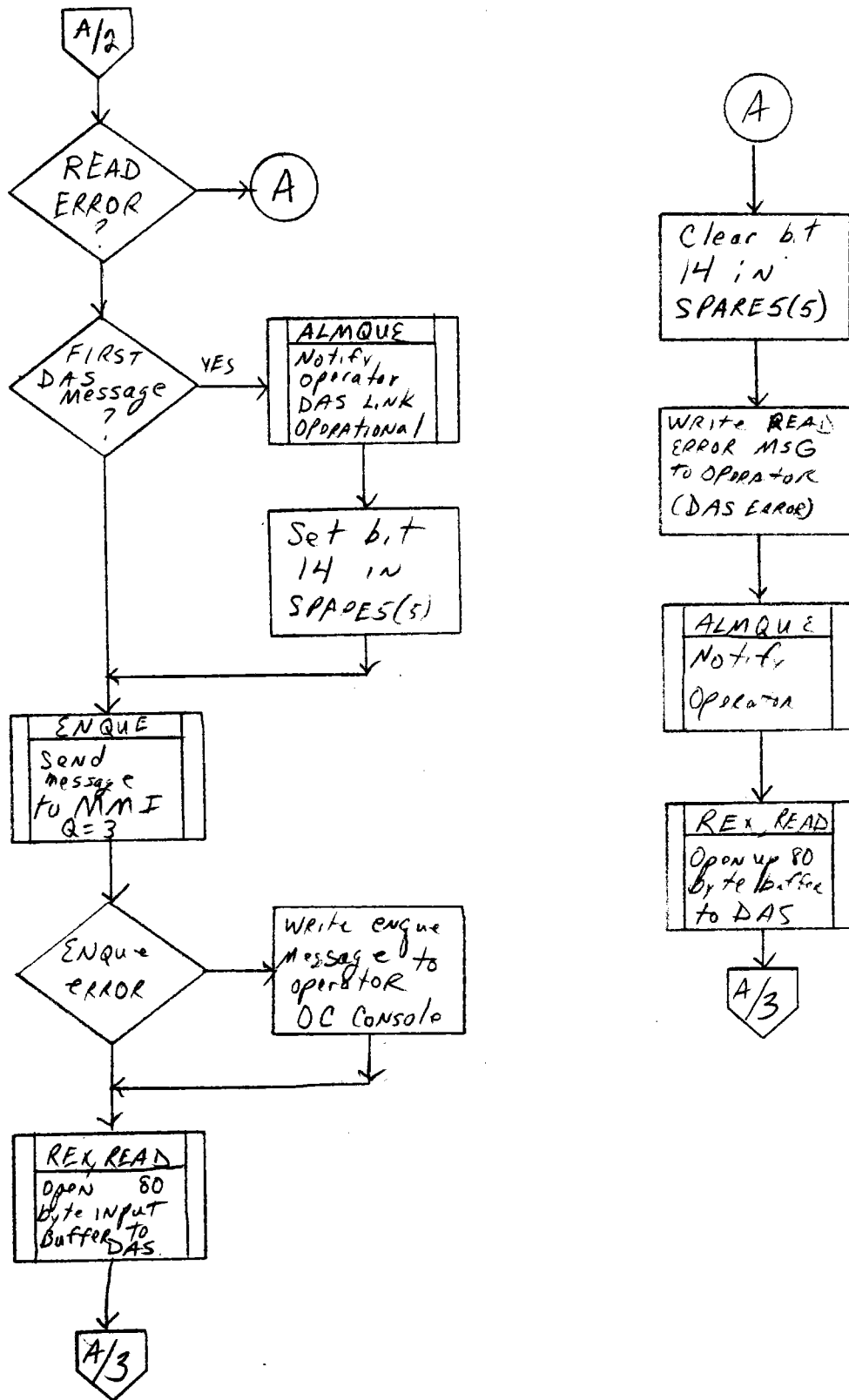


Figure 3.2.10-4 Flowchart - EXI (Continued)

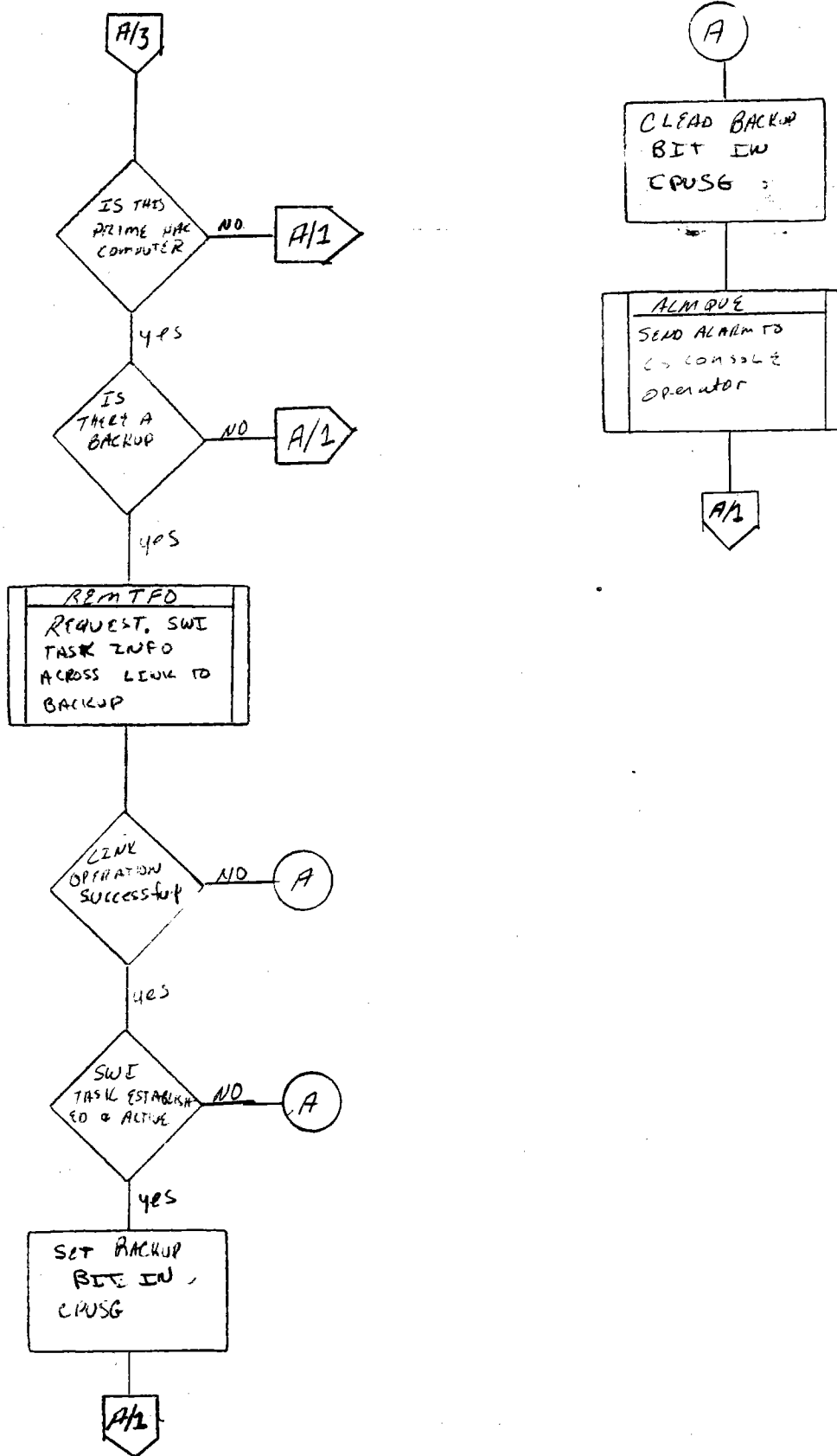


Figure 3.2.10-4 Flowchart - EXI (Continued)

- e. Error messages and recovery - Alarms are sent to the operator when a write operation to either the OCS or DAS computers has an error.

3.2.10.4.1.4.2 Data, Logic, and Command Paths

Input data:

- a. Response messages from the MMI, ALO, and STA tasks;
- b. SPARE5(5) bit 15 - OCS link status
bit 14 - DAS link status

Output data:

- a. SPARE5(5) bits 14 and 15 are cleared if write errors are found;
- b. Response messages sent to OCS and DAS computers.

3.2.10.4.1.4.3 Internal Data Description

- a. Four user file tables for write operation;
- b. Four local buffers for holding output messages;
- c. DEVTBL - to hold logical file names, and the status of each write operation, number of times per second this core device can be written to.
- d. Miscellaneous pointers.

3.2.10.4.1.4.4 Flowcharts

See Figure 3.2.10-5 for EXO flowchart.

3.2.10.5 Interface Description

a. Data Base Input:

CRTUSG-CRT usage flag
CRTLNG-CRT line index
BCSLNG-BCS line usage indicator
SPARE5(5)-OCS/DAS status

b. Data Base Output:

CRTUSG-CRT usage flag
CRTLNG-CRT line index
TBUSYG-task active flags
SPARE5(5)-OCS/DAS status

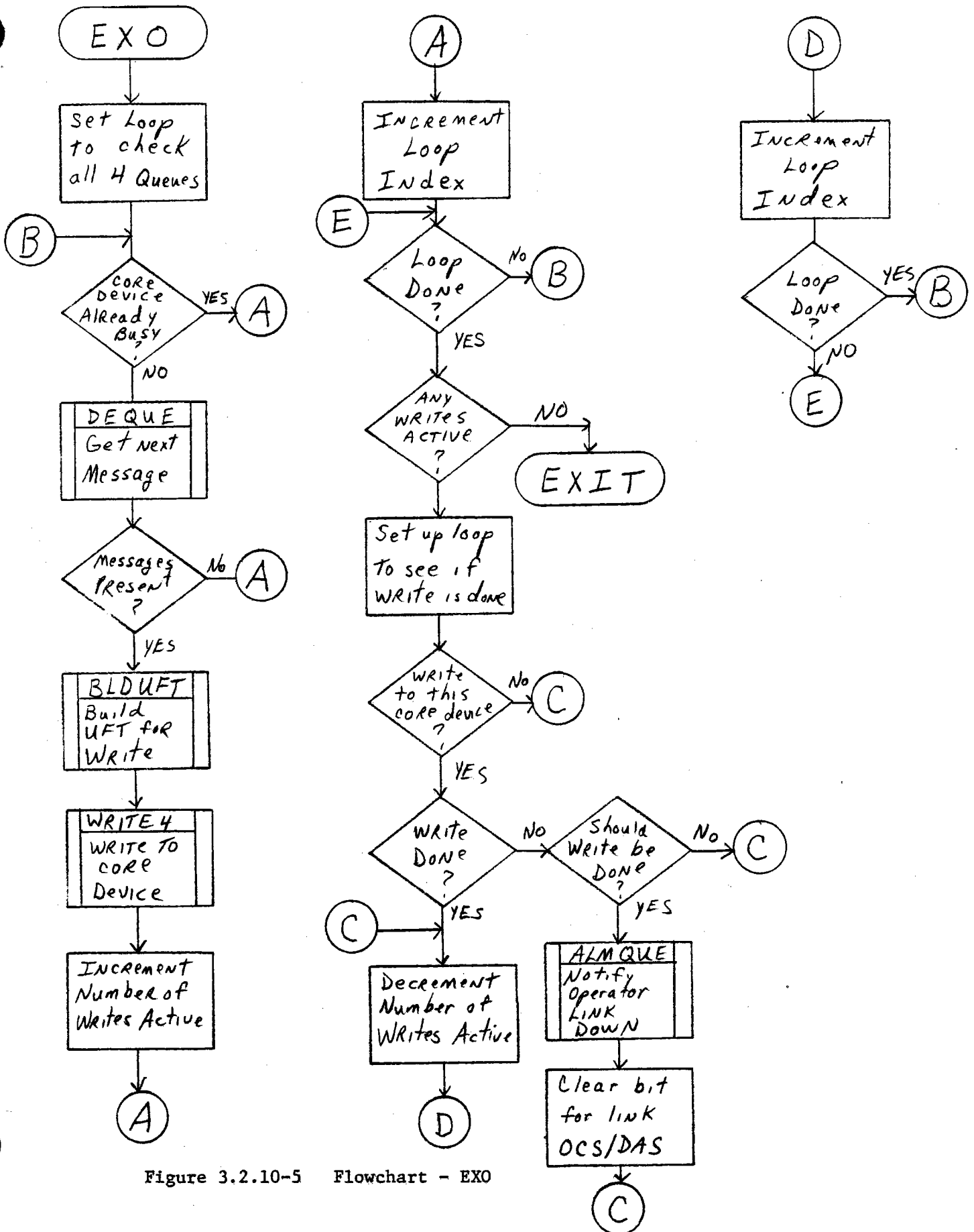


Figure 3.2.10-5 Flowchart - EXO

- c. Physical Output - Includes the write operation from the CSO task to the ISC terminal and the single-character buffer written by the CSI task.
- d. Physical Input - The read buffers associated with the ISC console (CS control console), the two Chromatics Graphic Terminals and the OCS, DAS, and backup HAC computers. The input from the ISC terminal is a one-character buffer. The input buffers from the graphics terminals are 10 words in length. (See Section 3.2.9 of the Software Design Specification.) The input from the OCS/DAS computers are a maximum of 80 bytes. The response to the remote task information request is a seven-word buffer.
- e. CSI is activated by DBINIT and CSO is activated by MMI, BCS, DSK, CFO, or STA tasks via Rex Enque.
- f. Messages sent from CSI to GRF are appended with a graphics console index either zero or one. This is used to indicate from which console the input came.
- g. Messages sent to the CSO task have the control characters required to place the output line in the desired location.
- h. Task CSI activates MMI or GRF.
- i. The interface between the BCSMOD and CSI/CSO tasks consists of the BCSLN6 word which contains the number of lines that the BCSMOD requires when it is active. CSI and CSO will check this word to insure that commands entered and outputs to conversational area of the ISC do not violate the area reserved for BCSMOD.
- j. The EXI task is activated by CLK in the prime HAC or by SWI if a HAC failover occurs. The EXI task will activate the MMI task by sending messages via the REX enqueue.
- k. The EXO task sends response messages to the OCS/DAS computers.

3.2.10.6

Test Requirements

- a. Verification of the ISC input and output will be done by operator key-in of various commands and observing the output on the CS control console.
- b. Verification of the Chromatics Graphic Terminal will be done by observing the updated screen displays.
- c. Logic of CSO and CSI will be verified by using MAX IV DEBUG. This system utility allows selective execution of code, tracing, and memory modification.

- d. The EXI/EXO task will be verified by entering commands on one OCS console and observing them displayed on the ISC and implemented accordingly.
- e. The EXI/EXO to DAS interface will be verified by observing data transmissions between the DAS and HAC via the testing of status requests.

3.2.11 BCS Automatic Processing Module - BCSMOD

3.2.11.1 Purpose

The purpose of this module is to implement an automatic method of cycling through heliostats in the field for beam characterization measurements. The results of these measurements can be used to calculate and update heliostat bias values for improving tracking accuracies.

3.2.11.2 Requirements

3.2.11.2.1 Design Requirements - Software Requirements listed in Section 3.1 of the 10 MWe Collector Subsystem Software/Firmware Functional Requirements Specification, 12 June 1980, that apply to the BCSMOD module are:

- a. Control of up to 2048 heliostats in all modes required to operate the heliostats.

This requirement is only partially applicable to BCSMOD, as only some of the heliostats need to be controlled, and only some of the modes (i.e.: TRACK, STANDBY, BCSTRACK, ALT2STOW, and STOW) are to be used.

- b. Detect, report, and respond to failures and irregularities. Only BCS failures and irregularities are to be detected and reported.

- c. Maintain a "Prime" and "Backup" system, as well as redundant field communications.

For BCSMOD this was interpreted to mean only that it would operate from beginning to end in either computer, but not be capable of switching from "Prime" to "Backup" during BCS operations.

- d. Maintain safe beam control.

Certain restrictions defined further in the CS-MCS and CS Plant Interface Requirements apply to BCS beam pointing.

- e. Respond to receiver trip for emergency defocus.

This requirement only applies to BCSMOD when it is trying to return heliostats to their original mode, which might have been TRACK.

- f. Maintain command/response protocol and data transfer with the OCS.

This only applies to "BCS" messages.

- g. Provide automatic control of beam characterization within the HAC.

This is the major BCS requirement.

Additional requirements that apply to the BCSMOD module are listed in Section 3.4 of the CS-MCS and CS-PLANT INTERFACE REQUIREMENTS (RADL ITEM 2-3-1) (dated October 1980) and are summarized below:

- a. Input three files from magnetic tape defining up to 60 heliostats to be measured per file and up to 5 heliostats to be moved out of the reflected beam path of the first heliostat.
- b. Display and select one of the three files and select the range of heliostats to be measured (first and last).
- c. Cycle through the selected heliostats by moving up to four (one per BCS target) at a time and their blocking heliostats as well. For each step the OCS issues a message to BCS, and at the completion of that step BCS responds to OCS.
- d. Results include bias calculation, displays, and logging of both.
- e. The BCSMOD module must be able to input manual offset data, calculate new biases, and update them.

3.2.11.2.2 Derived Requirements - Section 3.2.1.11 of the Software/Firmware Functional Requirement Specification (June 12, 1980) lists the following derived requirements for the BCSMOD module:

- a. Give the operator selection flexibility by using three different heliostat lists;
This implies no character online editing is necessary. Range of heliostats in list selection is required.
- b. Provide start capability of the automatic sequence that does not require further operator input;
- c. Execute control of a list of up to 60 heliostats and up to five blocking heliostats associated with each heliostat to be measured;
- d. Execute control over these measurements using up to four BCS targets in parallel;
- e. Communicate with the OCS for BCS measurements and offset values;
- f. Provide operator feedback for error in automatic sequences; and

- g. Calculate azimuth and elevation bias changes using OCS measured centroid offsets from up to three BCS measurements.

Up to three BCS measurements per heliostat can be evaluated.

Additionally, the BCSMOD module must be able to:

- a. Return heliostats to their original modes after measurements have been completed or after an alarm condition has been detected in the subsequence;
- b. Continue sequences that incur alarm conditions, but abort the subsequence causing the alarm;
- c. Store and access three files of BCS heliostat lists in either "Prime" or "Backup" computers;
- d. Execute special operator commands passed along by the MANMIF module. These commands and their parameters are listed in Table 3.2.11-I;
- e. Put heliostats in the Standby mode after putting them on the BCS target in response to a BCS-6 message or after a BCS-8 message if DEFOCUS is in progress and the heliostat was in the Track mode prior to the subsequence; and
- f. Put heliostats in the Track mode upon completion of a subsequence if it was in that mode prior to the subsequence.

3.2.11.3 Design Approach

BCSMOD is designed to operate in the software/hardware environment of the 10 MWe Phase II HAC on a minimal coupling (interfacing with existing modules) basis.

The following is a list of assumptions used in the design of the BCSMOD module:

- a. The major assumption is that BCSMOD is to provide an automatic service to the OCS/BCS operation that requires minimal operator intervention on the collector subsystem side.
- b. Assumption "a" implies, on a more detailed level, that errors and problems will be reported, but the software will not wait for an operator to act. Instead it will go on to the next subsequence, if possible.
- c. Assumption "b" implies that a CS operator cannot "fix" a problem and expect it to be considered again by the software. Such cases would have to be handled by starting new BCS sequences.

BCS COMMAND SUMMARY

PHASE 1

PHASE 2

COMMAND

DESCRIPTION

COMMAND

DESCRIPTION

BCSSTART

Activate BCS task

BCSMEAS

Do BCS measurements
for specified
heliostats

BCSXTAPE

Read magnetic tape to input 3 BCS files

BCSFILE FN

Select BCS file

BCSDISPL ON
OFF

Turn display of selected BCS file on or off

BCSRANGE R1,R2

Select range of heliostats contained on
selected BCS file and shown on display

T99

PARAMETER EXPLANATIONS:

FN File Number 1-3

R1,R2 Range of Heliostats R1 = low limit (1-60) $R1 \leq R2$
R2 = high limit (1-60) $R2 \geq R1$

Table 3.2.11-I

BCS COMMAND SUMMARY

PHASE 3			PHASE 4		
	COMMAND		DESCRIPTION	COMMAND	DESCRIPTION
METHOD 1	<u>BCSINPUT</u>	HC	Initiates method 1; specifies heliostat for which data is to be entered for bias calculation	<u>BCSEND</u>	Terminate BCS task
	<u>BCSOFFS</u>	D,T,OH,OV	Input 1, 2, 94 3 centroid offset points		
	<u>BCSCALC</u>	Input	Calculate bias from input data		
	<u>BCSUPD</u>	Y (yes) N (no)	Select to update new bias or not		
METHOD 2	<u>BCSCALC</u>	HC	Initiates method 2; do bias calculation for specified heliostat using OCS data base		
	<u>BCSUPD</u>	Y (yes) N (no)	Select to update new bias or not		
METHOD 3	<u>BCSCALC</u>	All	Initiates method 3; calculate biases; display if changed; unit for update decision;		
	<u>BCSUPD</u>	Y (yes) N (no)	Select to update new bias or not; continue with bias calculations if more heliostats arc specified on BCS file		

PARAMETER EXPLANATIONS:

HC	Heliostat Number	OH	Centroid Offset Horizontal
D	Date DD MM YY	OV	Centroid Offset Vertical
T	Time HH MM SEC	INPUT	Use data input via BCSINPUT and BCSOFFS for calculations

Table 3.2.11-I (continued)

- d. If OCS/BCS communications do not occur in a timely fashion, as mentioned in the interface document (CS-MCS and CS-Plant Interface Requirements), the BCS task will also self-abort.
- e. All self-aborts will include a return of HCs to normal operation, but will preclude new start-ups until this shutdown has been completed.

The BCS task is activated by the operator using the BCSSTART command. BCS is then controlled by the operator through the proper use of the commands summarized in Table 3.2.11-1. The operator is to do initialization first. He may read a new set of three BCS files from magnetic tape using the BCSXTAPE. One of the three BCS files must be chosen using the BCSFILE command. BCSRANGE is used to select a range of heliostats from the file chosen by the BCSFILE command. BCSRANGE will also initiate communication between the HAC and the OCS computers.

The operator is then permitted to select commands to do BCS OFFSET MEASUREMENTS or BIAS calculations and updating using any of three methods. The BCS measurement processing is initiated using the BCSMEAS command and proceeds automatically using the BCSFILE selected and the information supplied by the operator's initialization. Method 1 of the bias calculation and updating phase requires the operator to initiate the method with the BCSINPUT command and to manually input centroid offset data for up to three points using the BCSOFFS command. The bias calculations are commanded to proceed with the BCSCALC command. If a significant change in biases results, it is required that the operator decide whether the biases are to be updated or not with the BCSUPD command.

Method 2 is initiated using the BCSCALC command with a valid MACDAC heliostat number. The HAC will obtain the needed offset data from the OCS and do the bias calculations for the specified heliostat. If a significant change in biases is detected the operator is required to use the BCSUPD command to make the decision as to whether the biases should be updated or not.

Method 3 is initiated using the BCSCALC command with ALL as a parameter. This begins semiautomatic bias calculation and updating processing. The heliostats specified using the BCSFILE and BCSRANGE commands drive the processing. The biases for a heliostat are calculated; and if there is a significant difference, an update decision is required by the operator using the BCSUPD command. This is done for all the heliostats specified on the selected BCS file.

The BCSDISPL and BCSSEND commands are the only ones that can be entered at any time. BCSDISPL will turn on or off the BCS FILE display. BCSSEND will terminate the BCS task, discontinuing any processing currently in progress. If BCS measurements are in progress, all heliostats will be returned to the proper modes before the BCS task is terminated.

The BCS design is divided into two segments: the control processing segment and the measurement processing segment. These segments are controlled by the BCS task. Processing for each segment is divided into phases. Each phase is a collection of BCS processing states. The phases for control processing are shown in Table 3.2.11-II, while those for measurement processing are indicated in Table 3.2.11-V.

Each BCS segment is driven by its own set of control variables and STATE TRANSITION TABLE. CSTATE and MSTATE(T), where T is a target number, are the current state variables; and CCONDX and MCONDX(T) indicate a stimulus for a state transition defined in CTABLE and MTABLE for control and measurement processing respectively.

Control processing is responsible for the initiation of, or the actual processing of, BCS commands entered by the operator. Only certain commands are valid, depending on what the present CSTATE is. Control processing also handles messages to and from the OCS over core device 5. These messages deal with initialization, bias calculations, bias updating, and BCS termination, but excluding the messages concerned with the centroid offset measurement sequencing. Time-out processing for expected OCS messages is done in this area for core device 5. Each CSTATE does a specific processing function dependent upon the previous CSTATE (COLD) and the stimulus (CCONDX). Table 3.2.11-II describes the functions associated with CSTATES grouped according to control processing phases. CCONDX is set by an operator command, an OCS message on core device 5, or an expired timer for an expected OCS message on core device 5. Table 3.2.11-III shows the possible CCONDX stimuli. CTABLE will return the next valid CSTATE given the current CSTATE and the CCONDX stimulus. If an illegal transition is attempted, zero is returned. Table 3.2.11-IV is the state transition chart for control processing.

Measurement processing has the responsibility of controlling the centroid offset measurement sequencing. This includes commanding heliostats to be set up for the measurements, returning heliostats to the proper modes, and communication with the OCS computer through core devices 1 through 4. All the above is achieved for up to four BCS targets simultaneously. Time-out processing for expected OCS messages on core devices 1 through 4 is also done. Similar to the control processing each MSTATE does a specific processing function dependent upon the previous MSTATE (MOLD) and the stimulus (MCONDX). Table 3.2.11-V outlines the functions associated with MSTATES. MCONDX can be set by an OCS message, an expired timer for an expected OCS message and heliostat staggering, or a specific change of status of the heliostats being moved. Table 3.2.11-VI shows the possible MCONDX stimuli. MTABLE will return the next valid MSTATE given the current MSTATE and the MCONDX stimulus. Table 3.2.11-VII is the state transition chart for measurement processing. Due to the requirement that measurement sequencing is to occur simultaneously on up to four BCS targets, MSTATE, MCONDX, MOLD, and other variables are dimension to 4. Measurement processing is looped through up to four times to accomplish the processing for all operational targets. Messages

to and from the OCS OCCMR through core devices 1 through 4, each corresponding to a BCS target. The target numbers assigned for all BCS processing are shown in Table 3.2.11-VIII.

The heliostat and target status are kept throughout the BCS task. Heliostat status values are referenced for the BCS file display. Heliostat status (HSTAT) and target status (TSTAT) value explanations are shown in Table 3.2.11-IX and 3.2.11-X respectively.

TABLE 3.2.11-II CONTROL PROCESSING PHASE AND STATE DESCRIPTION

<u>CSTATE</u>	<u>FUNCTION</u>
<u>PHASE 1: INITIALIZATION</u>	
1	BCS Initialization
2	Read BCS Tape
3	BCS File Selection
4	Select Range from Selected File and Send BCS-1
5	Process BCS-2
24	Turn File Display ON/OFF
25	BCS-2 Timeout Processing
26	Termination Processing
<u>PHASE 2: BCS CENTROID OFFSET MEASUREMENT SEQUENCING</u>	
6	BCS Measurement Sequencing in Progress
7	BCS Measurement Sequencing Completed
<u>PHASE 3 - METHOD 1: BIAS CALCULATION AND UPDATE FROM MANUAL INPUT</u>	
8	Centroid Input Initialization
9	First Centroid Offset Input Processing
10	Second Centroid Offset Input Processing
11	Third Centroid Offset Input Processing
12	Bias Calculation from Input Data
13	Bias Update Decision Processing, Send BCS-13
<u>PHASE 3 - METHOD 2: BIAS CALCULATION AND UPDATE FROM OCS DATA</u>	
15	Send BCS-11 for OCS Data
16	Process BCS-12 (Up to 3)
17	Bias Calculations from OCS Data
18	Bias Update Decision Processing, Send BCS-13
<u>PHASE 3 - METHOD 3: AUTOMATIC BIAS CALCULATIONS AND UPDATING</u>	
19	Get Next HC from BCS File, Send BCS-11
20	Process BCS-12 (Up to 3)
21	Bias Calculations from OCS Data
22	Bias Update Decision Processing, Send BCS-13
23	Operation Complete
27	BCS-12 Time-out Processing

TABLE 3.2.11-III CSTATE CONDITION TABLE

CONDITION	COND NO.	(CCOND)
BCSTAPE OC CMD #1	1	
BCSFILE 2	2	
BCSDISPL 3	3	
BCSRANGE 4	4	
BCSMEAS 5	5	
BCSINPUT 6	6	
BCSOFFS 7	7	
BCSCALC INPUT 8	8	
BCSUPD 9	9	
BCSCALC HC 8	10	
BCSCALC ALL 8	11	
BCSEND 10	12	
(SPARE)	13	
(SPARE)	14	
MSTATE = (BCS MEAS SEQ COMPLETE)	15	
BCS-2	16	
TIME-OUT FOR BCS-2	17	
BCS-12	18	
TIME-OUT FOR BCS-12	19	
BCS MEASUREMENT SEQ. COMPLETED	20	
(SPARE)	21	

Table 3.2.11-IV STATE TRANSITION TABLE FOR CONTROL PROCESSING

CONDIX	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
USPHE																					
1	2	.3	.24	.0	.0	.0	.0	.0	.0	.0	.0	.26	.0	.0	.0	.0	.0	.0	.0	.0	.0
2	2	.3	.24	.3	.2	.0	.0	.0	.0	.0	.0	.26	.0	.0	.0	.0	.0	.0	.0	.0	.0
3	2	.3	.24	.4	.0	.0	.0	.0	.0	.0	.0	.26	.0	.0	.0	.0	.0	.0	.0	.0	.0
4	2	.3	.24	.0	.0	.0	.0	.0	.0	.0	.0	.26	.0	.0	.0	.0	.25	.0	.0	.0	.0
5	2	.3	.24	.7	.6	.0	.2	.4	.2	.15	.19	.26	.0	.0	.0	.0	.0	.0	.0	.0	.0
6	0	.0	.24	.0	.0	.0	.0	.0	.0	.0	.0	.26	.0	.0	.0	.0	.0	.0	.0	.0	.0
7	2	.3	.24	.0	.6	.8	.0	.0	.0	.0	.0	.26	.0	.0	.0	.0	.0	.0	.0	.0	.0
8	0	.0	.24	.0	.0	.0	.9	.0	.0	.0	.0	.26	.0	.0	.0	.0	.0	.0	.0	.0	.0
9	0	.0	.24	.0	.0	.7	.12	.12	.0	.0	.0	.26	.0	.0	.0	.0	.0	.0	.0	.0	.0
0	0	.0	.24	.0	.0	.0	.11	.12	.0	.0	.0	.26	.0	.0	.0	.0	.0	.0	.0	.0	.0
1	0	.0	.24	.0	.0	.0	.0	.12	.0	.0	.0	.26	.0	.0	.0	.0	.0	.0	.0	.0	.0
2	0	.0	.24	.0	.0	.0	.0	.0	.13	.0	.0	.26	.0	.0	.0	.0	.0	.0	.0	.0	.0
3	2	.3	.24	.7	.0	.0	.0	.0	.0	.0	.0	.26	.0	.0	.0	.0	.0	.0	.0	.0	.0
4	2	.0	.24	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0
5	0	.0	.24	.0	.2	.2	.2	.2	.2	.2	.2	.26	.2	.2	.2	.2	.2	.16	.27	.7	.2
6	2	.0	.24	.0	.0	.0	.0	.0	.0	.0	.0	.26	.0	.0	.0	.0	.0	.16	.27	.3	.0
7	0	.0	.24	.0	.0	.0	.0	.0	.18	.0	.0	.26	.0	.0	.0	.0	.0	.0	.0	.0	.0
8	2	.3	.24	.7	.6	.0	.0	.0	.0	.15	.19	.26	.0	.0	.0	.0	.0	.0	.0	.0	.0
9	0	.0	.24	.0	.0	.0	.0	.0	.0	.0	.0	.26	.0	.0	.0	.0	.0	.0	.20	.27	.0
0	0	.0	.24	.0	.0	.0	.0	.0	.0	.0	.0	.26	.0	.0	.0	.0	.0	.0	.27	.27	.0
1	2	.0	.24	.7	.0	.0	.0	.0	.22	.0	.0	.26	.0	.0	.0	.0	.0	.0	.0	.0	.0
2	0	.0	.24	.0	.0	.0	.0	.0	.0	.0	.0	.26	.0	.0	.0	.0	.0	.0	.20	.27	.0
3	2	.3	.24	.0	.6	.0	.0	.0	.0	.15	.19	.26	.0	.0	.0	.0	.0	.0	.0	.0	.0
4	0	.0	.24	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0
5	0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0
6	2	.7	.24	.7	.7	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0
7	2	.3	.24	.4	.6	.8	.0	.2	.0	.15	.19	.26	.0	.0	.0	.0	.0	.0	.0	.0	.0
8	0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0

TABLE 3.2.11-V BCS MEASUREMENT PROCESSING STATES (MSTATE(T))

A: SET UP HCs FOR BCS MEASUREMENT

1. Get Next HC and Send BCS-3
2. Process BCS-8 (Success Flag #2), Log, Get Next HC & Send BCS-3
3. Process BCS-4, Command HC & B/S HC
4. Set 2-Second Timer for Staggering B/S HCs When Setting Up
5. Command Next B/S HC
6. All HCs Commanded

B: DO BCS MEASUREMENTS

7. Send BCS-5 (HC on Target)
8. Process BCS-8, Begin Returning HCs
9. Process BCS-6, Command HC to Standby Mode
10. Send BCS-3
11. Process BCS-4, Command HC to Track BCS Target

C: RETURN HCs AND TIME-OUT PROCESSING

12. Begin Returning HCs (Due to Time-Out)
13. Set 2-Second Delay for Staggering B/S HC When Returning
14. Command Next B/S HC to Return
15. All HCs Commanded to Return
16. (Spare)

TABLE 3.2.11-VI MSTATE CONDITION TABLE

<u>CONDITION</u>	<u>CONDITION NO. (MCOND)</u>
BCSMEAS	1
TIME-OUT FOR BCS-4	2
TIME-OUT FOR BCS-8	3
BCS-4	4
BCS-6	5
BCS-8	6
B/S HC COMMAND QUEUED	7
STAGGER TIMER EXPIRED	8
ALL HCs IN COMMANDED MODE	9
HC POSITIONED ON BCS TARGET	10
(SPARE)	11
(SPARE)	12

TABLE 3.2.11-VII STATE TRANSITION TABLE FOR MEASUREMENT PROCESSING

CMCONDY	1	2	3	4	5	6	7	8	9	10	11	12
CMSTATE												
1	1	.16	.0	.3	.0	.2	.2	.0	.0	.2	.3	.2
2	.0	.16	.0	.3	.0	.2	.0	.0	.0	.0	.0	.2
3	.0	.2	.0	.0	.0	.0	.4	.2	.0	.0	.2	.2
4	.0	.0	.0	.0	.0	.0	.0	.5	.0	.2	.0	.0
5	.0	.0	.0	.0	.0	.0	.4	.0	.0	.0	.0	.0
6	.0	.0	.0	.0	.0	.0	.0	.0	.7	.0	.0	.0
7	.0	.0	.12	.0	.0	.0	.0	.0	.2	.2	.0	.2
8	.0	.0	.0	.0	.0	.0	.13	.0	.0	.0	.0	.0
9	.0	.0	.0	.0	.0	.0	.0	.0	.0	.10	.0	.0
0	.0	.12	.0	.11	.0	.0	.0	.0	.0	.0	.0	.0
1	.0	.0	.0	.0	.0	.0	.0	.0	.0	.7	.0	.2
2	.0	.0	.0	.0	.0	.0	.13	.0	.0	.0	.0	.0
3	.0	.0	.0	.0	.0	.0	.0	.14	.0	.0	.0	.0
4	.0	.0	.0	.0	.0	.0	.13	.0	.0	.0	.0	.0
5	.0	.0	.0	.0	.0	.0	.0	.0	.1	.0	.0	.0
6	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0	.0

TABLE 3.2.11-VIII BCS TARGET NUMBERS

<u>TARGET NUMBER</u>	<u>BCS TARGET</u>
1	SOUTH
2	WEST
3	NORTH
4	EAST

TABLE 3.2.11-IX HELIOSTAT STATUS

<u>HSTAT (1, POINT(T))</u>	<u>EXPLANATION FOR HC TO BE MEASURED</u>
0	No HC
1	In BCS File
2	In Selected Range (Not Measured)
3	Commanded to Track BCS Target or Tracking BCS Target
4	Commanded to Track Standby or Tracking Standby
5	Measurement Complete, HCs Commanded to be Returned
6	Not in Good Mode for BCS Measurements (HC or B/S HC)

(BHC>1)

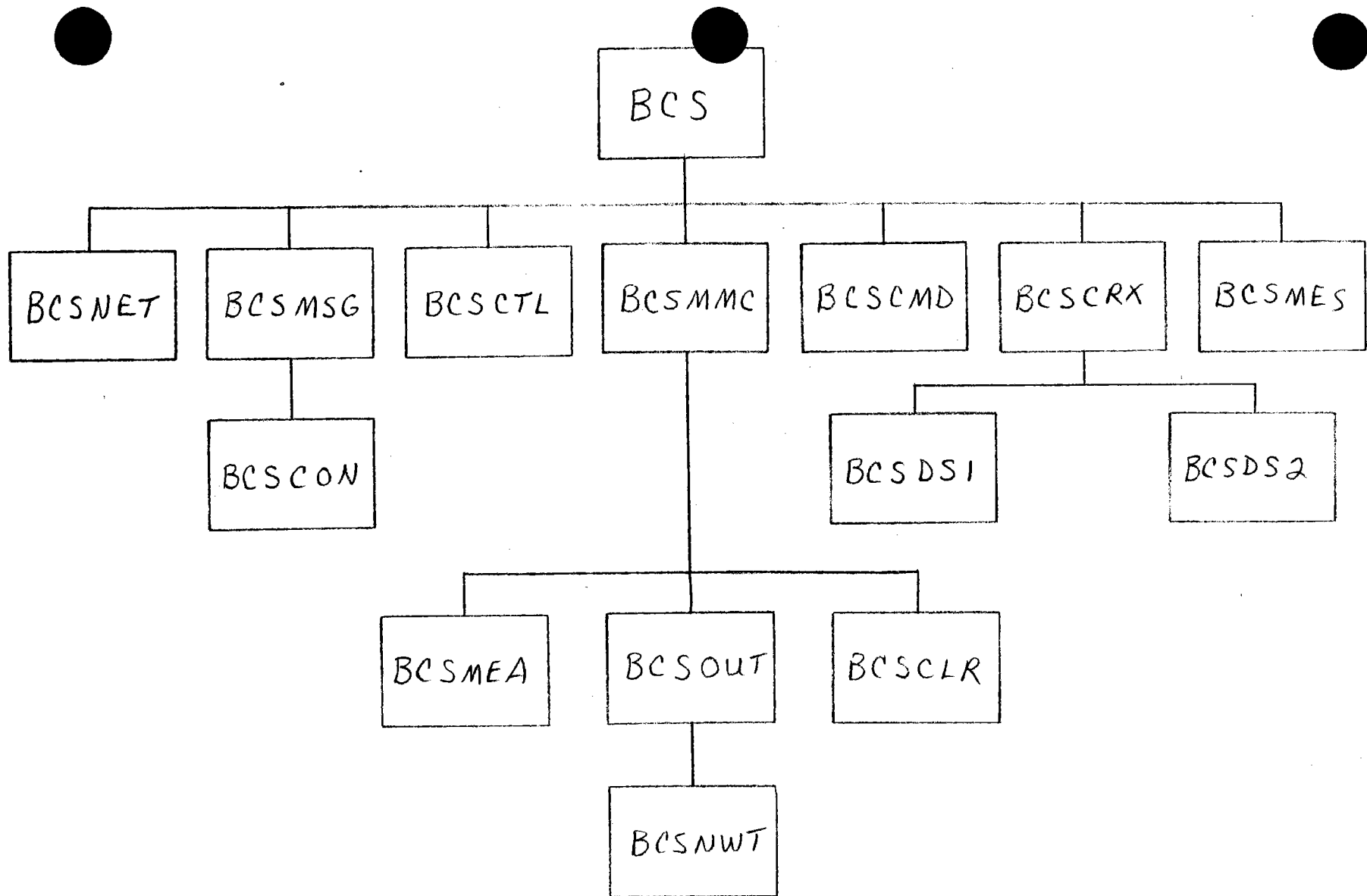
<u>HSTAT(BHC, POINT(T))</u>	<u>EXPLANATION FOR B/S HC</u>
8	Commanded To or In STOWALT2
9	In Stow or STOWALT2 Originally
0	HC in Original Mode or STBY

TABLE 3.2.11-X BCS TARGET STATUS

<u>TSTAT (T)</u>	<u>EXPLANATION</u>
0	Nonoperational or Already Busy
1	Operational and Not Busy
2	Measurements Completed for All HCs in Range and Assigned to Target T
3	HI-WIND STOW

3.2.11.3.1 Functional Allocations - BCSMOD consists of 43 submodules which are organized into four groups. The first group shown in Figure 3.2.11-1 contains the highest level control functions and those functions that are used throughout BCS. These submodules are described below:

- BCS - Main routine activated once per second. It checks for new stimulus such as operator input commands, OCS input, and expired timers relevant to control processing, and initiates the appropriate processing. If measurement processing is active, BCSMMC is called. Timers and displays are also updated.
- BCSCMD - Operator input routine to check information dequeued from MMI and to set CCONDX. Command parameters are validated.
- BCSCTL - This routine gets the new CSTATE resulting from the present CSTATE and a valid stimulus (CCOND). It initiates the appropriate control processing by calling the subroutine responsible for that phase of control processing.
- BCSMMC - When BCS measurement processing is active, this routine checks for new stimulus such as OCS input, expired timers, and changes in heliostat status concerning measurement processing, and initiates the appropriate processing. This is done for up to four targets.
- BCSMEA - This routine gets the new MSTATE resulting from the present MSTATE and a valid stimulus (MCOND). It initiates the appropriate measurement processing by calling the subroutine responsible for that phase of measurement processing.
- BCSMMSG - This routine is responsible for decoding any valid OCS-HAC message and setting the appropriate variables in local common. CCOND or MCOND are also set.
- BCSOUT - This routine will construct the specified HAC-OCS message and send it to the OCS.
- BCSMES - Outputs informative, alarm and error messages.
- BCSCON - Heliostat number conversion submodule. This subroutine converts a MDAC heliostat number into an MMA number.
- BCSCRX - This routine produces the selected BCS display on the ISC terminal. The possible displays are:
 - (1) BCS active display with CSTATE and MSTATE,
 - (2) BCS file display, and
 - (3) BCS biases display.



675

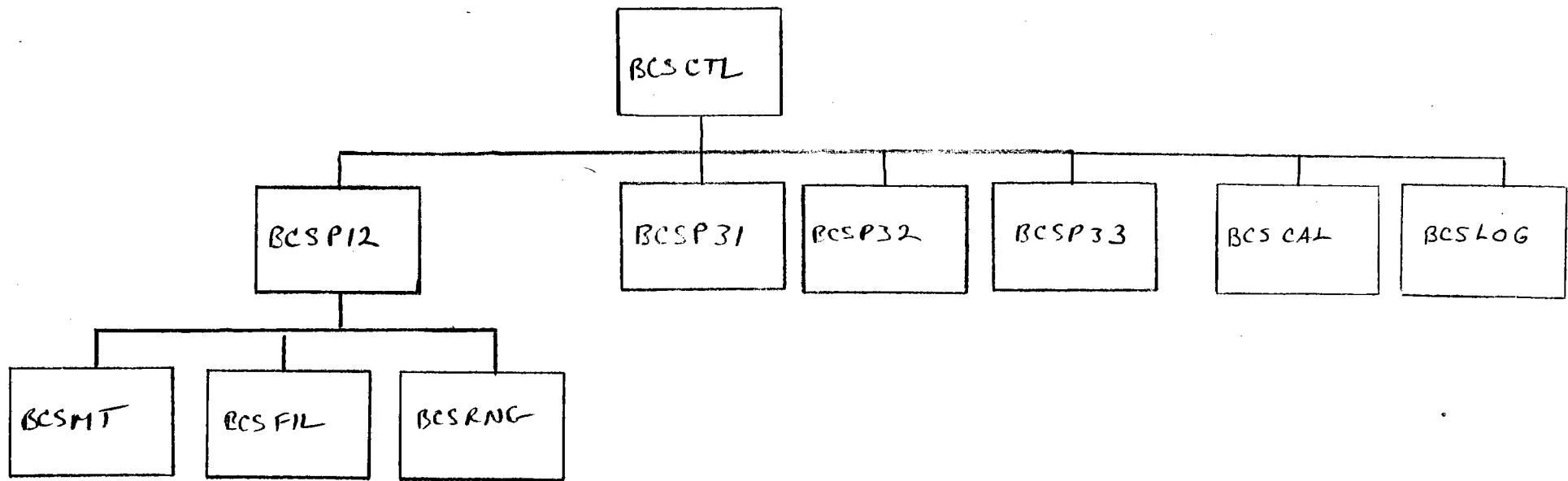
Figure 3.2.11-1 BCS SUBMODULE DIAGRAM--GROUP 1

The display may also be cleared by calling this routine.

- BCSDS1 - This routine is called by BCSCRX to produce the BCS file display.
- BCSDS2 - This routine is called by BCSCRX to produce the BCS bias display.
- BCSNET - This routine takes care of reading from any of the five core devices without waiting for a write. If no new write to the core device has been made, a stage is set and returned.
- BCSNWT - Write without wait are issued to the OCS link by this routine to the specified remote core device.

The second functional group is responsible for the actual control processing activities. These submodules are described below and depicted in Figure 3.2.11-2:

- BCSP12 - All phase 1 and phase 2 control processing is done here. This includes operator commands BCSTAPE, BCSFILE, BCSRANGE, BCSDISPL, BCSEND, and OCS messages BCS-1, BCS-2.
- BCSMT - This routine handles the reading of three BCS files from magnetic tape to disk.
- BCSFIL - This routine selects the chosen OCS file from disk and initializes the WRK and WRKARR arrays.
- BCSRNG - This routine initializes WRK and WRKARR arrays to the specified range and sends a BCS-1 message to the OCS.
- BCSP31 - Phase 3 Method 1 control processing is done here. This includes operator commands BCSINPUT, BCSSOFFS, BCSCAL INP, BCSUPD, and OCS message BCS-13.
- BCSP32 - Phase 3 Method 2 control processing is done here. This includes operator commands BCSCAL HC, BCSUPD, and OCS messages BCS-11, BCS-12, and BCS-13.
- BCSP33 - Phase 3 Method 3 control processing is done here. This includes operator commands BCSCALC ALL, BCSUPD, and OCS messages BCS-11, BCS-12, and BCS-13.
- BCSCAL - The bias calculations are the main responsibility of this routine. It also obtains the old biases and HC coordinates.
- BCSLOG - This routine will write to the logger the results of bias calculations similar to the information the BCS bias displays.



677

FIGURE 3.2.11-2 BCS SUBMODULE DIAGRAM--GROUP 2

The third functional group (Figure 3.2.11-3) does the BIAS calculations for Phase 3 control processing. Its submodules are described below:

- BCSBIA - This routine handles the direct site accessing and updating of heliostat biases on disk file HCB.
- BCSCOR - This routine handles the direct file accessing of heliostat coordinates from disk file HCC.
- BCSLSQ - Offset data algorithm. This subroutine uses least squares fitting to calculate new biases from one to three sets of offset data points.
- BCSSUN - Submodule to BCSLSQ to set up call for SUNPOS and reconvert the sun vector to floating point.
- BCSGIM - Submodule to BCSLSQ to calculate heliostat vector and gimbal angles.
- BCSNOR - Submodule to BCSLSQ to calculate heliostat normal vector from offsets.
- BCSLSC - Submodule to BCSLSQ to calculate least squares coefficients.
- BCSEPD - Submodule to BCSLSC to evaluate partial derivatives.
- BCSVEC - Submodule to BCSLSC to calculate vector terms for least squares coefficients.
- BCSLIN - Submodule to BCSLSQ to solve linear equations.

The fourth functional group (Figure 3.2.11-4) deals solely with measurement processing activities. These submodules are described below:

- BCSMPA - Phase A measurement processing is done here. This includes OCS messages BCS-3, BCS-4, and BCS-8 and setting up HCs for measurements.
- BCSBOS - This routine finds the next blocking/shadowing heliostat for the heliostat to be measured and commands it ALT2STOW.
- BCSAL2 - Gets ALT2STOW data from disk and sets up the HC command.
- BCSMPB - Phase B measurement processing is done here. This includes OCS messages BCS-5, BCS-8, BCS-6, BCS-3, BCS-4.
- BCSMP C - Phase C measurement processing is done here. This includes returning heliostats to their proper modes.

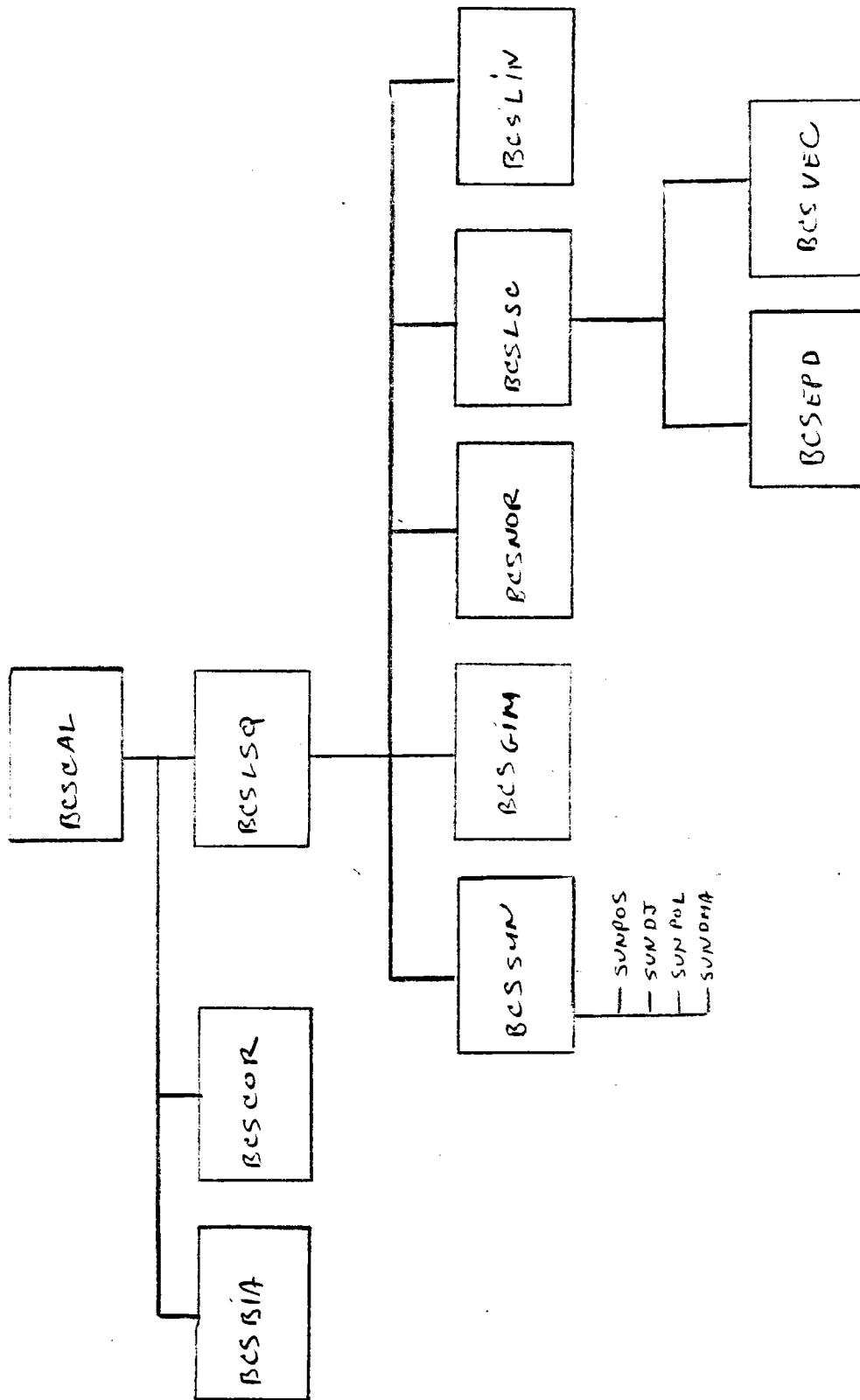


FIGURE 3.2.11-3 BCS SUBMODULE DIAGRAM--GROUP 3

- BCSGET - This routine finds the next heliostats to be measured from the BCS file for a particular target whose blocking/shadowing heliostats are in proper modes for BCS.
- BCSCLR - This routine clears the BCS busy bit for the measured heliostat and all its blocking/shadowing heliostats.
- BCSRTN - The next blocking/shadowing heliostat is returned to its proper mode.
- BCSSTB - The specified heliostat is commanded to STANDBY mode.
- BCSSTS - This routine checks heliostat STATUS changes and sets the appropriate MCONDX if a desired status change is observed.

3.2.11.3.2 Resource Budgets

- a. BCSMOD has the following memory requirements for its task (BCS) and associated subroutines:

16 16-bit words
(only while BCS is active)

- b. Disk and magnetic tape utilization:

1. Magnetic tape input during initial pahse.
2. BCS partition access (read and write) during initial phase. 186 records of 80 bytes = 14,880 bytes.
3. BIAS file (HCB) access (read and write) during bias update phase. 32 records of 256 bytes = 8,192 bytes.
4. HC coordiante file (HCC) access (read only) during bias update phase. 64 records of 320 bytes = 20,480 bytes.

- c. Timing

1. BCS will be activated about 3 - 4 times a day during good weather.
2. BCS will execute once per second to check on any inputs or events while it is active.

- d. Priorities

BCS is a background operation, so its priority requirement is low.

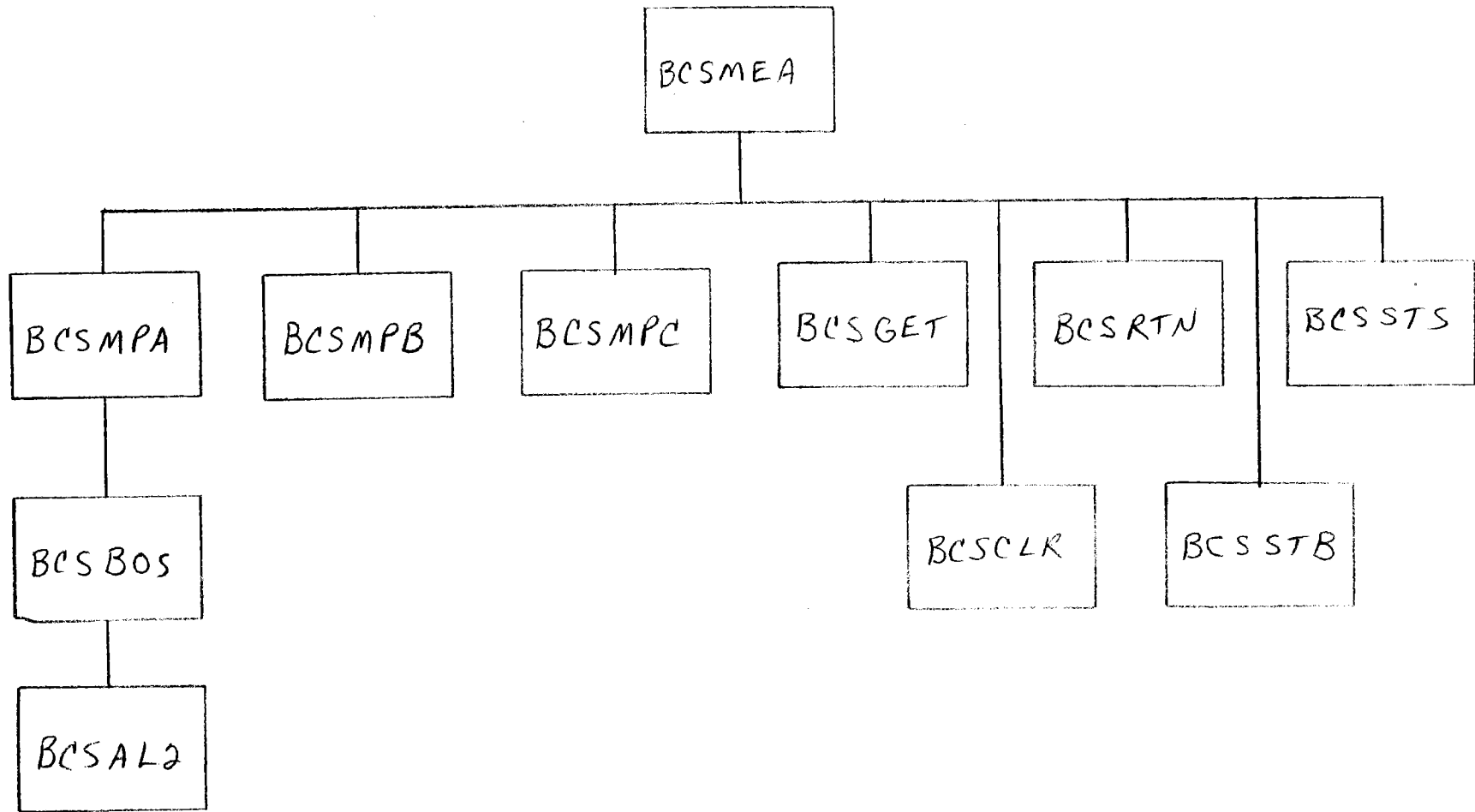


FIGURE 3.2.11-4 BCS SUBMODULE DIAGRAM--GROUP 4

e. Other

1. Line printer (logger) to print results.
2. Bottom of operator's CRT for table display, bias calculation results, and "BCS Active" message.

3.2.11.4 Design Description

3.2.11.4.1 Module Structure - BCSMOD consists of one task, called BCS, containing 41 submodules. Submodule BCS is the overall controlling routine and the highest level for control processing activities, with 30 submodules. Submodule BCMMC is the highest level for measurement processing activities, consisting of 10 submodules. A detailed discussion of all BCS submodules is given below.

3.2.11.4.1.1 Submodule I - BCS (Task)

3.2.11.4.1.1.1 Description -

- a. Language Used - FORTRAN.
- b. How Invoked - Activated by MMI in response to the operator command BCSSTART.
- c. Constraints and Limitations - None.
- d. Processing -
 1. Initialization of control variables.
 2. Dequeue message from MMI queue #13. If queue is empty, go to Step 5.
 3. Check and store parameters of dequeued command; set CCOND. If command is bad, display error message and go to Step 2.
 4. Do processing in response to dequeued command; go to Step 2. If termination is requested, go to Step 14. If the CSTATE transition is not valid, display error message and go to Step 2.
 5. Check for OCS message on Core Device 5; if none, go to Step 8.
 6. Decode received OCS message. If bad, display error message and go to Step 5; otherwise, set CCOND.
 7. Do processing in response to received OCS message, and go to Step 5. If termination is requested, go to Step 14. If the CSTATE transition is not valid, display error message and go to Step 5.

8. Check active control timers. If no timers expired, go to Step 10. (See Table 3.2.11-XII for TIMING VARIABLES.)
9. Do control processing in response to the expired timers. Go to Step 8.
10. If measurement processing is in effect, do the necessary measurement processing.
11. Update and/or change the BCS display.
12. Delay one second.
13. Update all active timers; go to Step 2.
14. Terminate the BCS task, suppressing all secondary activations.

3.2.11.4.1.1.2 Data, Logic, and Command Paths -

a. Input Data

1. Global Common:

COMIN4 - TBUSYG
COMQUE - IQUEUE

2. Local Common:

BCSCM1 - Timer, M, EXITF, TC
BCSCM2 - CBUFF, CSIZE, CRTF
BCSCM3 - MBUFFIMN
BCSCM6 - DS1F, DS2F, DS3F, MTIMER

b. Output Data

1. Global Common:

COMIN4 - TBUSYG

2. Local Common:

BCSCM1 - TIMER, MSTATE, MOLD, CSTATE, COLD, EXITF, M, CCONDX
BCSCM2 - CRTF
BCSCM3 - MN
BCSCM5 - HCCPTR, HCBPTR, CRTDN, PARM, NPARM
BCSCM6 - MTIMER, DS1F, DS2F, DS3F, IAVL2

c. Calls

1. BCS Submodules:

BCSCRX BCSNET
BCSCMD
BCSCTL
BCSMES
BCSMMSG
BCSMMC

2. Non-BCS Submodules:

DEQU BLDUFT

3. MAX IV REX Services:

DELAY
EXIT

3.2.11.4.1.1.3 Internal Data Description - BCS Local Common Areas:

BCSCM1 - See Table XIII
BCSCM2 - See Table XIV
BCSCM3 - See Table XV
BCSCM4 - See Table XVI
BCSCM5 - See Table XVII
BCSCM6 - See Table XVIII

3.2.11.4.1.1.4 Flowchart- See Figure 3.2.11-5.

TABLE 3.2.11-XI

BCS GLOBAL COMMON REFERENCE TABLE

BCS SUBMODULE	GLOBAL COMMON AREA	VARIABLE READ	VARIABLE WRITTEN
BCS	COMIN4	TBUSYG	TBUSYG
	COMQUE	(QUEUE #13)	
BCSMMC	COM1S4		BCSBYG
	COM1S5	LOKSTG	
BCSOUT	COM1S3	GRSTSG	
	COM1S5	GT1MEG	
		SUNPOG	
BCSMES	COMQUE		(QUEUE #7)
BCSCON	COMIN2	MD2HCG	
	COMIN3	MDNPRG	
BCSCRX	COMQUE		(QUEUE #7)
			(QUEUE #12)
	COM1S4	BCSLNG	
	COM1S5	CRTWRG	
BCSDS1	COMQUE		(QUEUE #12)
	COM1S4		
BCSDS2	COMQUE		(QUEUE #12)
	COM1S4		
BCSP12	COM1S4	BCSBYG	BCSBYG
	COM1S5	LOKSTG	
BCSP31	COM8S3	HCST3G	
BCSCAL	COMIN4	BCSTGG	
BCSLOG	COM1S5	GTIMEG	
BCSLSQ	COMIN4	SLATG	SLATG
		SLONGG	SLONGG
BCSSUN	COMIN4	SLATG	
		SLONGG	
BCSMPA	COMIN4	BCSTGG	HCCMDG
	COM1S2		HCDATG
BCSBOS	COM8S2	HCSTIG	
BCSAL2	COM1S1		
	COM1S2		HCCMDG
			HCDATG
BCSMPB	COMIN4	BCSTGG	
	COM1S2		HCCMDG
			HCDATG
BCSMPD	COMIN4		
	COM8S2		
	COM1S5	LOKDEG	HCCMDG
	COM1S2		HCDATG

BCS GLOBAL COMMON REFERENCE TABLE (Continued)

BCS SUBMODULE	GLOBAL COMMON AREA	VARIABLE READ	VARIABLE WRITTEN
BCSGET	COM8S2	HCSTIG	
	COM8S3	HCST3G	HCST3G
	COM1S2	HCCMDG HCDATG	
BCSCLR	COM8S3		HCST3G
BCSRTN	COM1S5	LOKDEG	
	COM1S2		HCCMDG HCDATG
BCSSTB	COMIN4	CORRCG	
	COM8S3	HCST3G	
	COM1S2		HCCMDG HCDATG
BCSSTS	COM8S2	HCSTIG	
	COM1S2	HCCMDG	

EXPECTED OCS MESSAGE	(TIMER) SUBSCRIPT	(TO) TIMEOUT PERIOD (SEC)	(TC) TIMEOUT CONDITION NO.
BCS-2	1	3	17
BCS-12	2	20	18
		(MTO)	(MTC)
BCS-4	3	28	2
BCS-8	4	720	3
STAGGER TIMER	5	2	8

NOTE: BCS-6 can never timeout because either a BCS-4 or BCS-8 is always expected

TABLE 3.2.11-XII TIMING VARIABLE

TABLE 3.2.11-XIII
LOCAL COMMON BCSCM1 -- CONTROL AND MEASUREMENT VARIABLES

VARIABLE	TYPE	EXPLANATION
CSTATE	I*2	Current or most recently completed control state
CCONDX	I*2	Stimulus or condition to proceed to new CSTATE
COLD	I*2	Previous CSTATE
CTABLE(21,27)	I*2	Contains next CSTATE given CCONDX and CSTATE
MSTATE(4)	I*2	Current or most recently completed measurement state
MCONDX(4)	I*2	Stimulus or condition to proceed to new MSTATE
MOLD(4)	I*2	Previous MSTATE
MTABLE(12,16)	I*2	Contains next MSTATE given MCONDX and MSTATE
M	I*2	Operator message no. (M=0 NO ERROR, M>0 ERROR)
T	I*2	Current target
EXITE	L*2	BCS termination flag
CHGF	L*2	Display update flag
WRK(6,60)	I*2	Same as below except MMC HC Numbers
WRKARR(6,60)	I*2	Measurement HCs and blocking/shadowing HCs from chosen BCS file in MDAC numbers
HSTAT(6,60)	I*2	Heliostat status
POINT(4)	I*2	Pointer to current HC in WRKARR for each target
TIMER(5)	I*2	OCS message timers
TO(5)	I*2	Timeout period
TC(5)	I*2	Timeout CONDX No.
BHC(4)	I*2	Pointer to current blocking/shadowing HC in WRKARR
ENDF	L*2	Measurement sequencing finished for target T
BOSF	L*2	Flag indicating more B/S HCs to be commanded to STOW ALT2
RTNF	L*2	Flag indicating more B/S HCs to be returned
SAVARR(60,6,8)	I*2	Array to save original HC command
STS7F(4)	L*2	Status flag for MCONDX 7
STS9F(4)	L*2	Status flag for MCONDX 9
STS10F(4)	L*2	Status flag for MCONDX 10
STSF	L*2	Heliostat status flag
BEFOR(4,6)	I*2	Position compare save array

TABLE 3.2.11-XIV

LOCAL COMMON BCSCM2 INPUT DATA BY BCS OPERATOR COMMANDS

VARIABLE	TYPE	EXPLANATION
CBUFF(11)	I*2	Message buffer
CSIZE	I*2	Size of message
FN	I*2	BCS file No. (1<FN<3) input by BCSFIL
R1,R2	I*2	Selected range of HCs (1<R1<60, R1<R2<60 Input by BCSRNG
IHC	I*2	MDAC HC No. Input by BCSINPUT
IMM,IDD,IYY	I*2	Month, day, year input by BCSOFFS
IH,IM,IS	I*2	Hours, mins, secs input by BCSOFFS
IU,IV	R*4	Horizontal and vertical offsets input by BCSOFFS
CHC	I*2	MDAC HC No. input by BCSCALL
YESF	L*2	Bias update flag
CRTF	L*2	Display On/Off flag

TABLE 3.2.11-XV

LOCAL COMMON BCSCM3 -- INPUT DATA BY OCS MESSAGES

VARIABLE	TYPE	EXPLANATION
MBUFF(40)	I*2	OCS Message buffer
MN	I*2	Current OCS message No. NOTE: WRKARR(1,POINT(T)) has MDAC HC No. sent in messages 3 and 5 associated with a target and expected in messages 4, 6, or 8.
MHC	I*2	MDAC HC No. received in OCS message
MT	I*2	BCS target no. received NOTE: Below is data provided by BCS-8
SCAN8	I*2	No. of BCS scans taken
CAF8	I*2	Centroid alarm flag
PAF8	I*2	Power alarm flag
MSF8	I*2	Measurement success flag
H8,M8,S8	I*2	Time of measurement
U8,V8	R*4	Vertical and horizontal offsets
IP8	R*4	Measured average image power
TIP8	R*4	Theoretical image power
FIL8	I*2	Field insolation level
		NOTE: Below is data provided by BCS-12.
MEAS	I*2	Measurement No. (1<MEAS<3)
NMEAS	I*2	Number of measurements being sent (1<NMEAS<3)
MML2,DD12,YY12	I*2	Date of measurement
H12,M12,S12	I*2	Time of measurement
U12,V12	R*4	Vertical and horizontal offsets
SCAN12	I*2	No. of BCS scans taken
CAF12	I*2	Centroid alarm flag
PAF12	I*2	Power alarm flag
MSF12	I*2	Measurement success flag
TSTAT(4)	I*2	BCS target status
GOF	L*2	BCS initialization permission from BCS-2

TABLE 3.2.11-XVI
 LOCAL COMMON BCSCM4 -- BIAS CALCULATION VARIABLES

VARIABLE	TYPE	EXPLANATION
XN(3),YN(3),ZN(3)	R*4	Heliostat Normal Vectors
XNA(3),YNA(3),ZNA(3)	R*4	Actual Heliostat Normals
XH,YH,ZH	R*4	Heliostat Coordinates
XS(3),YS(3),ZS(3)	R*4	Sun Unit Vectors
AZ(3),EL(3)	R*4	Heliostat Azimuths and Elevations
XT,YT,ZT	R*4	Target Center Coordinates
AT(4),BT(4)	R*4	Target Azimuths and Elevations
A(2,2),B(2),X(2)	R*\$	Linear System Matrix and Vectors
MM(3),DD(3),YY(3)	I*2	Month, Day, Year of Offset Data
HO(3),MI(3),SE(3)	I*2	Hour, Min, Sec of Offset Data
HOUR(3)	R*4	Time of Offset Data in Hours
U(3),V(3)	R*4	Horizontal and Vertical Offsets (Meters)
NP	I*2	Number of Offset Data Points
HCN	I*2	MMC HC No. to do Bias Calculations For
P	I*2	Current Offset Data Point
BIT	R*4	Value of 1/2 of an Encoder Bit
AZC(3),ELC(3)	R*4	Computer Leveling Corrected Values
AZ \emptyset ,EL \emptyset	R*4	Calculated Bias Delta in Radians
AZBD,ELBD	R*4	Calculated Bias Delta in Degrees
AZN,ELN	R*4	New Biases in Degrees
AZO,ELO	R*4	Old Biases in Degrees
AZNEW,ELNEW	I*2	New Biases in HAC Format
AZOLD,ELOLD	I*2	Old Biases in HAC Format

TABLE 3.2.11-XVII

LOCAL COMMON BCSCM5 -- MISCELLANEOUS DATA USED BY SEVERAL SUBMODULES

VARIABLE	TYPE	EXPLANATION
		NOTE: THE FOLLOWING CORRESPOND TO THE CURRENT BCS FILE
TITLE(5)	I*2	BCS File Title (10 Characters)
RDATE(3)	I*2	Date File is to be Run (DDMMYY)
ETIME(2)	I*2	Earliest Start Time (HHMM)
LTIME(2)	I*2	Latest Start Time (HHMM)
		NOTE: THESE VARIABLES ARE USED AS PARAMETERS FOR SEVERAL SUBROUTINES
MHCN0	I*2	MDAC HC Number
HFCHC	I*2	MMC HC Number Used by BCSBIA, BCSCOR
		NOTE: THESE VARIABLES ARE USED BY BCSCRX
CRTDN	I*2	Display Number
CRTHC	I*2	MDAC HC NO
		NOTE: THESE VARIABLES ARE USED BY BCSBIA
AZIM,ELEV	I*2	Bias to be Updated or Retrieved
REP	L*2	Update Bias if .True., Retreive if .False.
HCBPTR	I*2	Direct Access File Pointer for File HCB
HCCPTR	I*2	Direct Access File Pointer for File HCC
		NOTE: THESE VARIABLES ARE USED BY BCSMES
PARM(5)	R*4	Parameters for Alarms and Message
NPARM	I*2	No. of Parameters for Above
BTC	I*2	BCS Termination Code

TABLE 3.2.11-XVIII

LOCAL COMMON BCSCM6 -- OTHER VARIABLES

VARIABLE	TYPE	EXPLANATION
MTIMER(5,4)	I*2	Measurement Timers for 4 Targets
MTO(4)	I*2	Measurement Timeout Periods (4 Targets)
MTC(4)	I*2	Measurement Timeout MCONDX No. (4 Targets)
DS1F	L*2	Display Control Flag for Display 1
DS2F	L*2	2
DS3F	L*2	3
XF	L*2	Extra Flag
IAVAL2	I*2	Direct Access File Pointer for File AL2

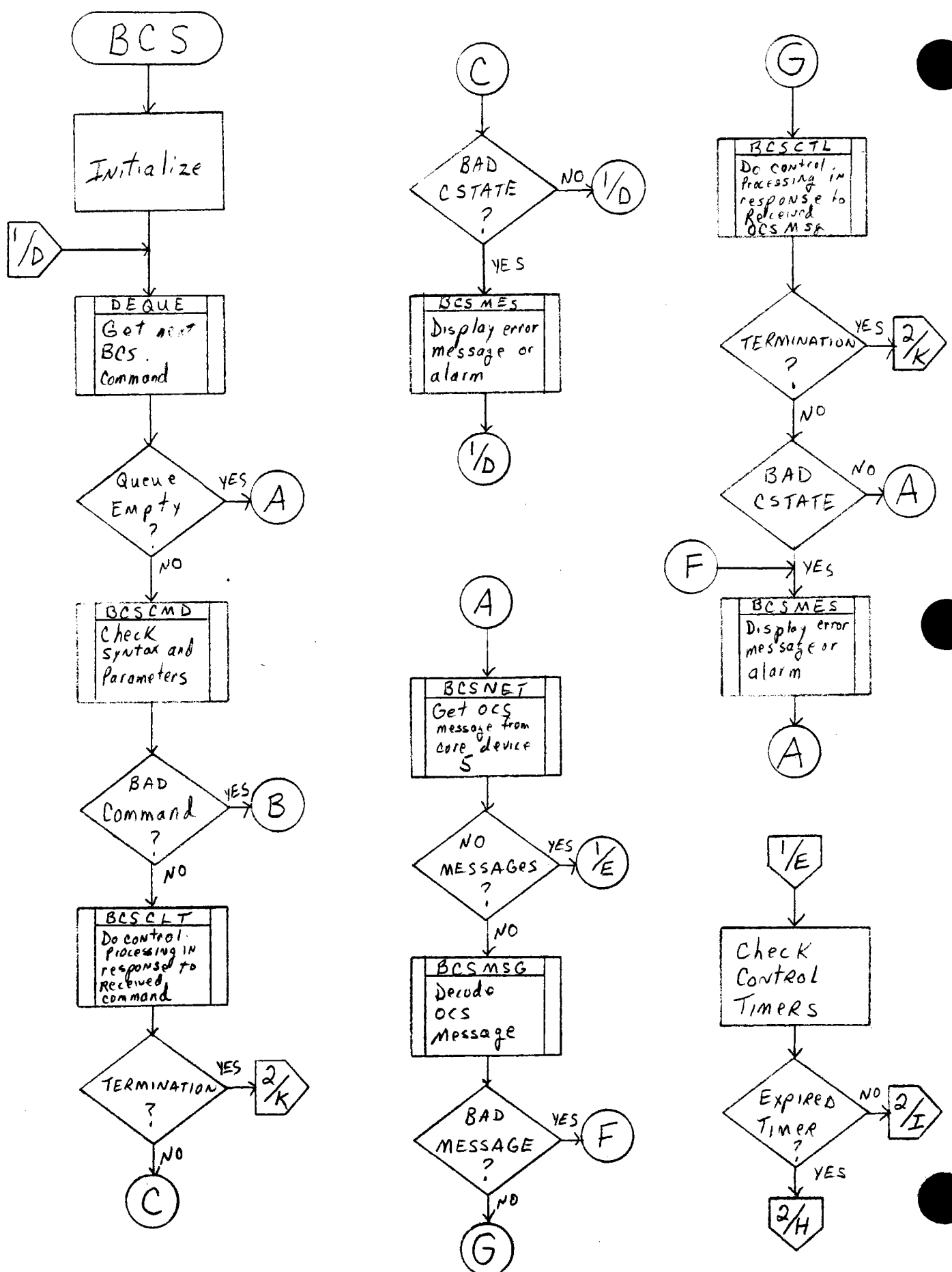


FIGURE 3.2.11-5 FLOWCHART BCS 694

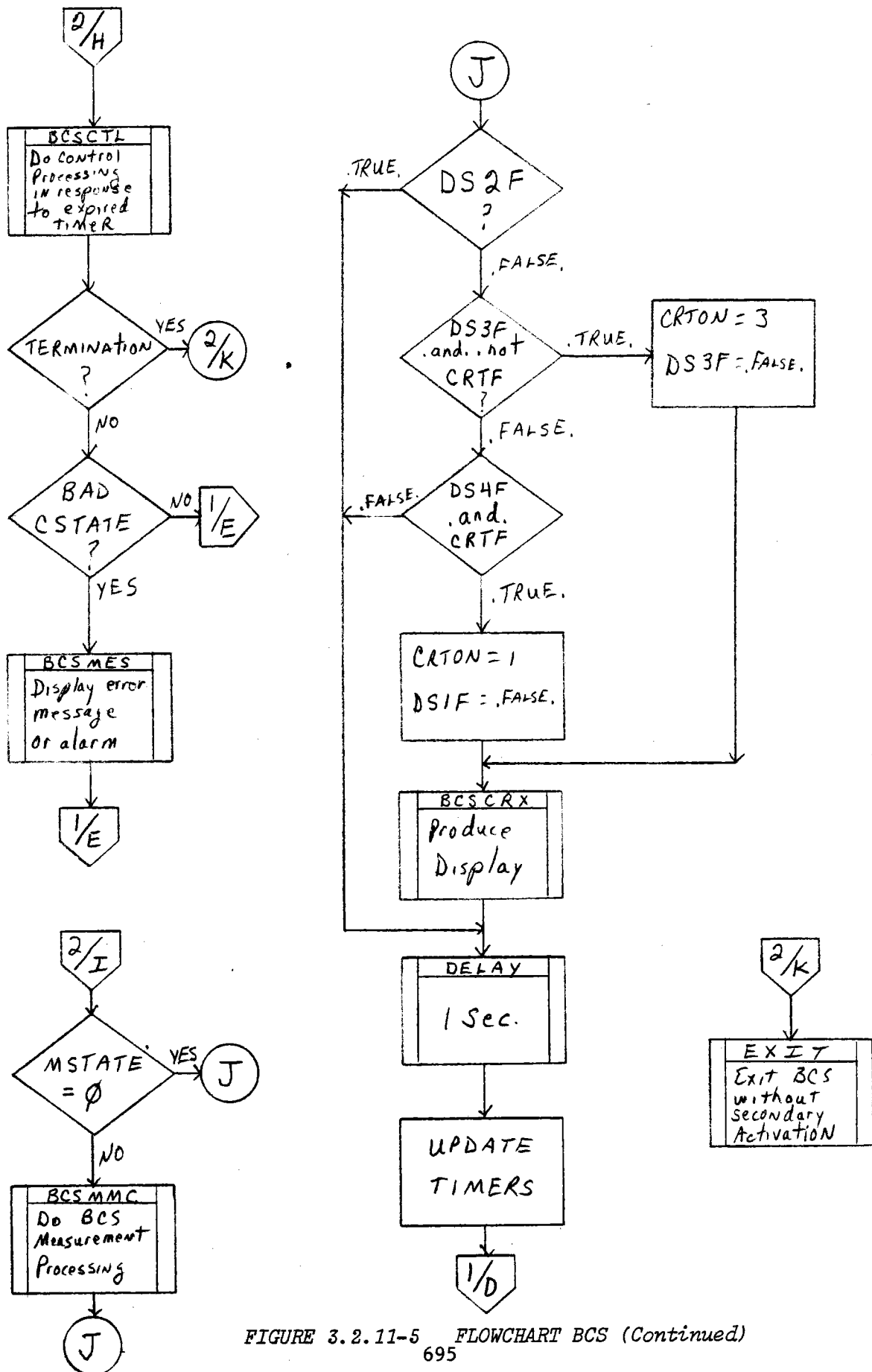


FIGURE 3.2.11-5 FLOWCHART BCS (Continued)
695

3.2.11.4.1.2 Submodule II -- BCSCMD -

3.2.11.4.1.2.1 Description -

- a. Language Used - FORTRAN.
- b. How Invoked - Subroutine called by BCS
- c. Constraints and Limitations - This subroutine should be executed only after a new message from MMI has been dequeued.
- d. Processing - Table 3.2.11-XIX shows the expected messages from MMI. The processing checks the parameters, if any, and sets CCONDX.
 1. If BCSFILE, check that file no. is between 1 and 3.
 2. If BCSDISPL, set CRTF ON or OFF.
 3. If BCSRANGE, make sure range is within the limits of the chosen BCS file.
 4. If BCSINPUT, convert MDHCNO to HFCHC and see if it's valid.
 5. If BCSSOFFS, move data into local common.
 6. If BCSCALC, check if parameter is 10000 (INP) or 10001 (ALL) or a valid HC NO. and convert.
 7. If BCSUPD, set YESF TRUE OR FALSE.

3.2.11.4.1.2.2 Data, Logic, and Command Paths -

a. Input Data

1. Local Common

BCSCM2-CBUFF

b. Output Data

1. Local Common

BCSCM1 -- CCONDX,M
BCSCM2 -- FN,CRTF,R1,R2,IHC,IDD,IMM,IYY,
IH,IM,IS,IU,IV,CHC,YESF
BCSCM5 -- PARM,NPARM
BCSCM6 -- DS1F,DS3F

c. Calls

1. BCS Subroutines

BCSCON

3.2.11.4.1.2.3 Flowchart - See Figure 3.2.11-6.

3.2.11.4.1.3 Submodule III - BCSTL

3.2.11.4.1.3.1 Description -

- a. Language Used - FORTRAN.
- b. How invoked - Subroutine called by BCS whenever a valid stimulus occurs for control processing activities.
- c. Constraints and Limitations - This subroutine should only be executed when a valid stimulus for control processing occurs.
- d. Processing -
 1. Get next CSTATE in response to current CSTATE and current stimulus (CCONDX).
 2. If new CSTATE is \emptyset , then an invalid transition has been attempted; so set error message parameters and return.
 3. Do the processing for the new CSTATE and return.

3.2.11.4.1.3.2 Data, Logic, and Command Paths -

a. Input Data

1. Local Common

BCSCM1 - CTABLE, CCONDX, CSTATE

b. Output Data

1. Local Common

BCSCM1 -- COLD, CSTATE, M

BCSCMG -- DS3F

c. Calls - BCS Submodules

BCSP12

BCSP31

BCSP32

BCSP33

3.2.11.4.1.3.3 Flowchart - See Figure 3.2.11-7.

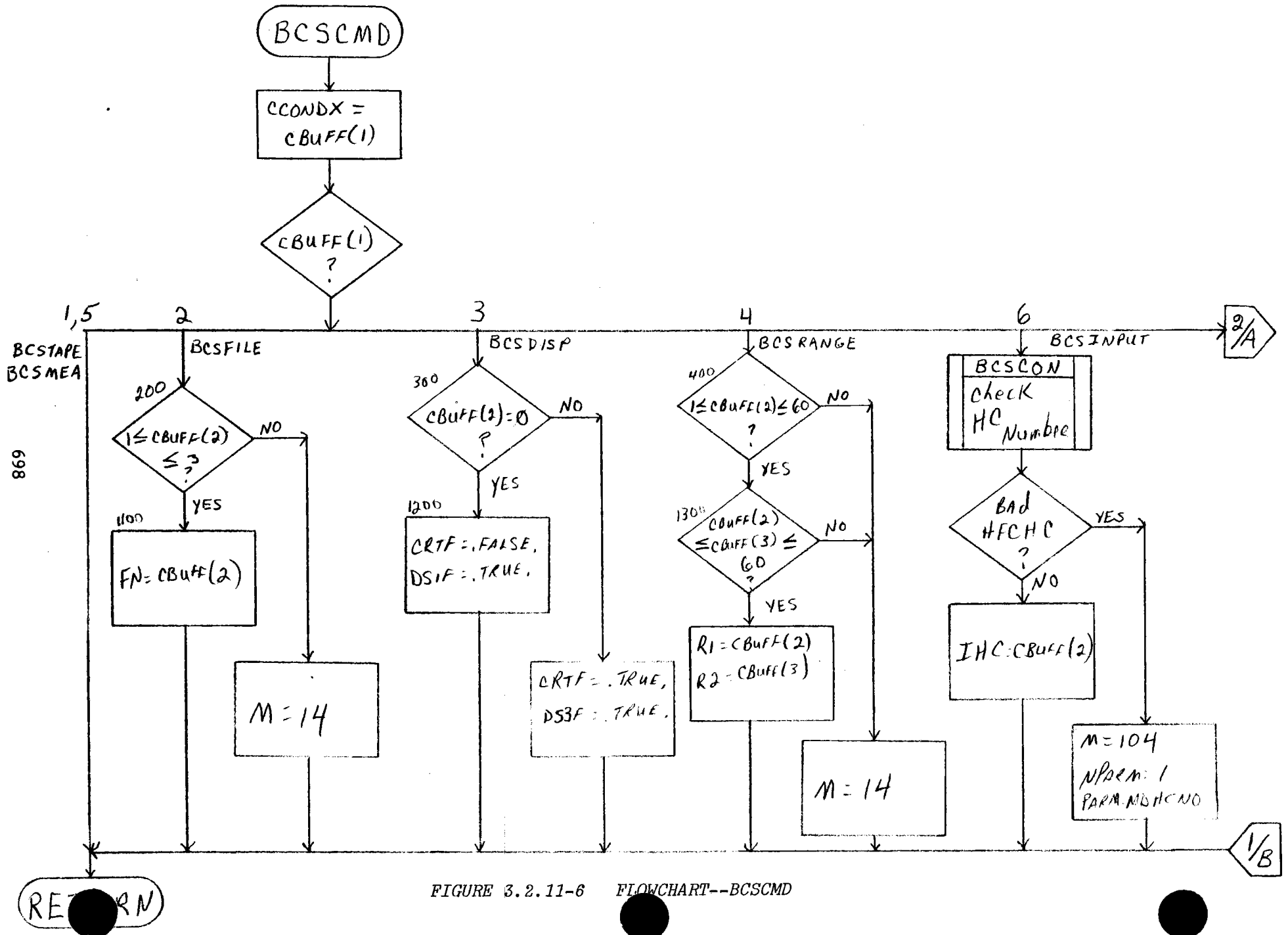


FIGURE 3.2.11-6 FLOWCHART--BCSCMD

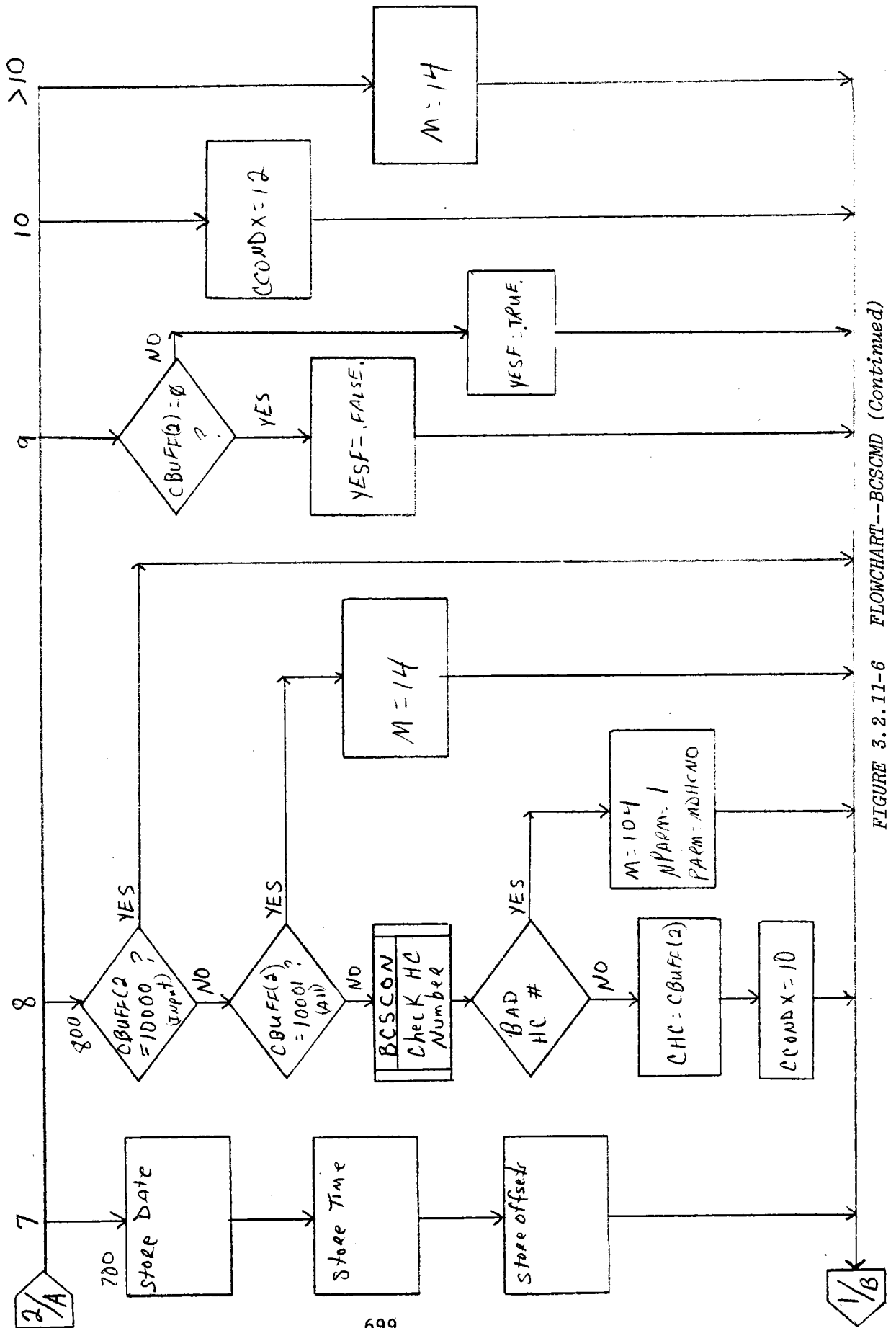


FIGURE 3.2.11-6 FLOWCHART--BCSCMD (Continued)

WORD IN CBUFF	INTEGER*2						REAL*4				
	1	2	3	4	5	6	7	8	9	10	11
BCSXTAPE	1										
BCSFILE FN	2	FN									
BCSDISPL ON		1									
OFF	3	0									
BCSRANGE R1 R2	4	R1	R2								
BCSMEAS	5										
BCSINPUT HC	6	HC									
BCSOFFS DATE, TIME, OFFSETS	7	MONTH	DAY	YEAR	HOUR	MIN	SEC	HORIZ.OFFSET	VERT.OFFSET		
BCSCALC INPUT	8	10000									
HC		HC									
ALL		10001									
BCSUPD YES	9	1									
NO		0									
BCSEND	10										

700

TABLE 3.2.11-XIX EXPECTED COMMAND MESSAGES FROM MMI

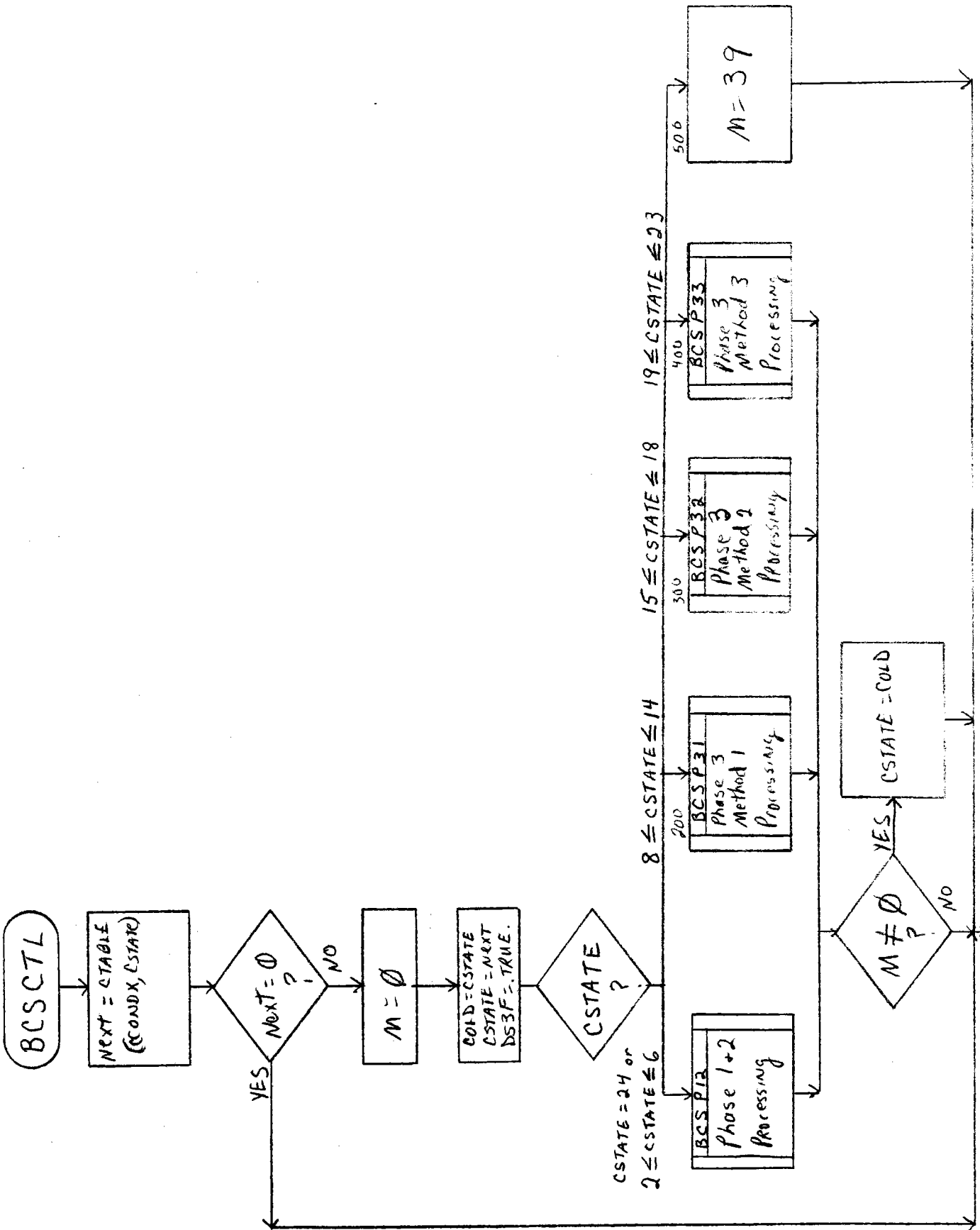


FIGURE 3.2.11-7 FLOWCHART -- BCSCCTL

3.2.11.4.1.4 Submodule IV -- BCSMMC

3.2.11.4.1.4.1 Description -

- a. Language Used - FORTRAN IV.
- b. How Invoked - Subroutine called by BCS submodule when CSTATE is 6.
- c. Constraints and Limitations - Executed only if at least one BCS target is operational and available.
- d. Processing -
 1. If "high wind STOW" clear, BCS busy, and target busy bits, send BCS-10 and Return.
 2. Get next good Target - T.
 3. Get OCS message from core device T.
 4. Decode OCS message; if bad, display message, go to Step 3.
 5. Do measurement processing in response to received OCS message; if bad MSTATE, display message, and go to Step 3; if processing terminated for Target T, go to Step 10.
 6. Check measurement timers; if no expired timers, go to Step 8.
 7. Do measurement processing in response to expired timer; if processing terminated for Target T, go to Step 10; if bad MSTATE, display message, and go to Step 6; otherwise go to Step 6.
 8. Check HC status; if status not changed, go to Step 10.
 9. Do measurement processing in response to HC status change; if processing terminated for Target T, go to Step 10 if MSTATE is bad display message and go to Step 8; otherwise go to Step 8.
 10. If processing for all targets is completed, clear busy bits, send BCS-10, and return; otherwise go to step 2.

3.2.11.4.1.4.2 Data, Logic, and Command Paths

a. Input Data

1. Local Common

BCSCM1 -- ENDF, STSF
BCSCM3 -- TSTAT
BCSCM6 -- MTIMER, MTO MTC

2. Global Common

COMIS5 -- LOKSTG

b. Output Data

1. Local Common

BCSCM1 -- M, MCONDX, MSTATE, CSTATE
BCSCM3 -- MN
BCSCM5 -- PARM, NPARM, BTC

2. Global Common

COMIS4 -- BCSBYG

c. Calls

1. BCS Submodules

BCSOUT BCSCLR
BCSMMSG BCSNET
BCSMEA
BCSMES

3.2.11.4.1.4.3 Flowchart - See Figure 3.2.11-8.

3.2.11.4.1.5 Submodule V -- BCSMEA

3.2.11.4.1.5.1 Description -

- a. Language Used - FORTRAN IV.
- b. How Invoked - Subroutine called by BCSMMC to perform the proper processing in response to the lasted stimulus.
- c. Constraints and Limitations - Processing in only done for a valid state transition.

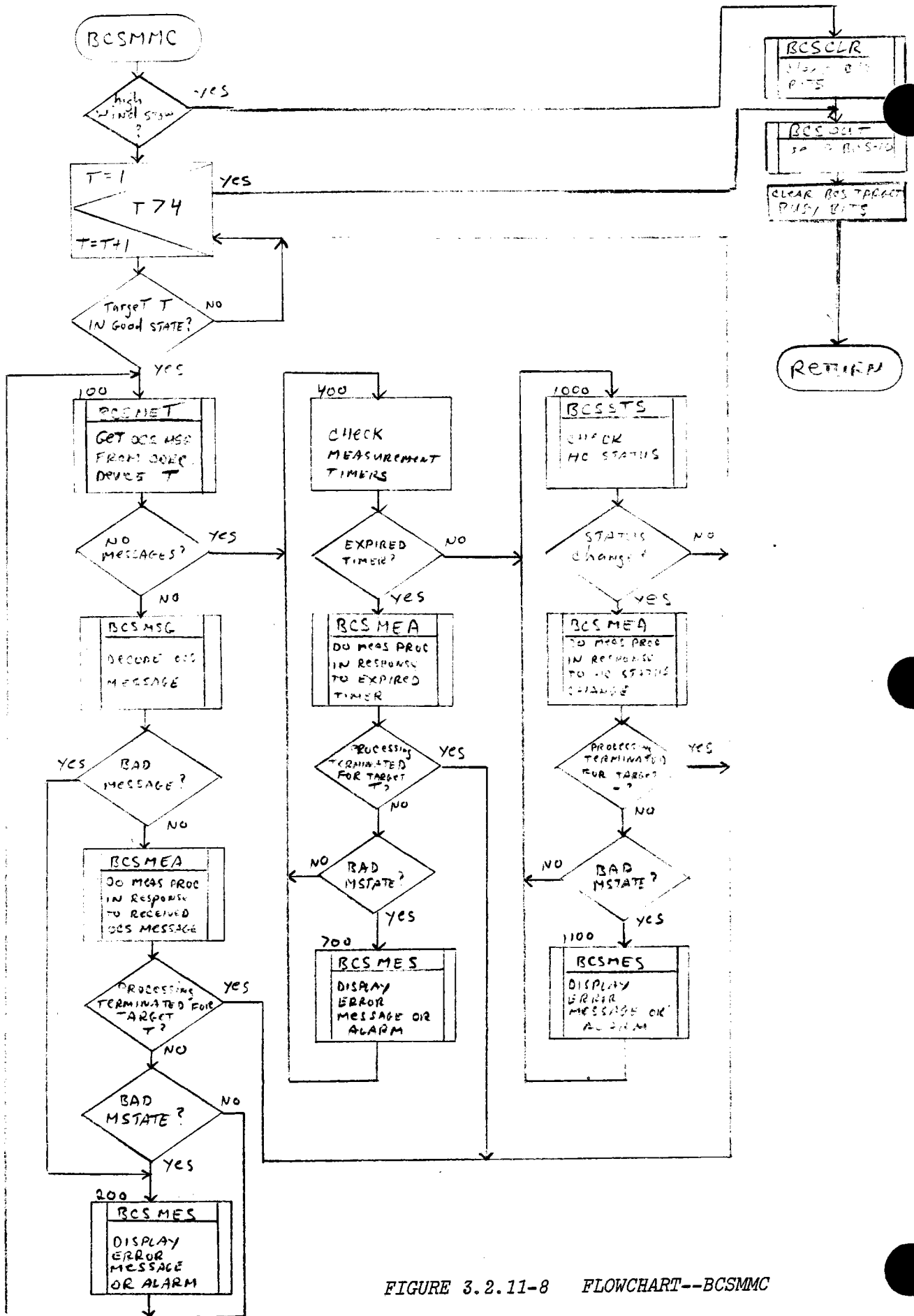


FIGURE 3.2.11-8 FLOWCHART--BCSMMC

d. Processing -

1. Get next MSTATE(T) given MCONDX, T and current MSTATE(T). If MSTATE(T) is \emptyset , return.
2. Set MSTATE(T) and MOLD(T).
3. If MSTATE(T) is in Phase A, do Phase A processing; go to Step 7.
4. If MSTATE(T) is in Phase B, do Phase B processing; go to Step 7.
5. If MSTATE(T) is in Phase C, do Phase C processing; go to Step 7.
6. Set error message and return.
7. If M is not \emptyset , set MSTATE(T) to MOLD(T) and return.

3.2.11.4.1.5.2 Data, Logic, and Command Paths

a. Input Data

1. Local Common

BCSCM1 - MCONDX, T, MSTATE, MTABLE

b. Output Data

1. Local Common

BCSCM1 - MOLD, MSTATE, M
BCSCMG - DS3F

c. Calls

1. BCS Submodules

BCSMPA
BCSMPB
BCSMPC

3.2.11.4.1.5.3 Flowchart - See Figure 3.2.11-9

BCSMEA

NEXT =
MTABLE
(MCOND(T), MSTATE(T))

YES
NEXT = \emptyset ?

NO

MOLD(T) = MSTATE(T)
MCOND(T) = \emptyset
M = \emptyset
MSTATE(T) = NEXT

MSTATE(T) ?

MSTATE = 16 or
 $1 \leq MSTATE \leq 5$

BCSMPA
Phase A
Processing

$7 \leq MSTATE \leq 11$

BCSMPB
Phase B
Processing

$12 \leq MSTATE \leq 14$

BCSMP C
Phase C
Processing

PARM(L) = MN
M = 111
NPARM = 1

MSTATE(T) =
MOLD(T)

YES

M \neq \emptyset

RETURN

FIGURE 2.11-9 FLOWCHART -- BCSMEA

3.2.11.4.1.6 Submodule VI -- BCMSG

3.2.11.4.1.6.1 Description -

- a. Language Used - FORTRAN IV.
- b. How Invoked - Subroutine called by BCS and BCMMC.
- c. Constraints and Limitations - This subroutine should be executed only after a new OCS message from core device 1 through 5.
- d. Processing -
 1. Decode and check first 8 bites of OCS message.
 2. If message number is 2, decode and check BCS file title, target status, permission flag, and set CCONDX and return.
 3. Decode and check HC number and target number.
 4. If message number is 4, set MCONDX and return.
 5. If message number is 6, set MCONDX and return.
 6. If message number is 8, set and check scans, centroid, power, and success flags, time, offsets, image power, and field insulation. Set MCONDX and return.
 7. If message number is 12, check and set measurement, number of measurement, scans, centroid, power, and success alarms. Set CCONDX and return.
 8. Anything else, display error message.

3.2.11.4.1.6.2 Data, Logic, and Command Paths

a. Input Data

1. Local Common

BCSCM3 - MBUFF
BCSCM5 - HFCHC, TITLE

b. Output Data

1. Local Common

BCSCM1 - CCONDX, MCONDX, M
BCSCM3 - MN, GOF, TSTAT, MHC, MT, CAN8, CAF8, PAF8,
MSF8, H8, M8, S8, U8, V8, IP8, TIP8, FIL8,
MEAS, NMEAS, MM12, DD12, YY12, H12, M12,
S12, U12, V12, SCAN12, CAF12, PAF12, MSF12,
MEA12, NMEA12, IBUFF
BCSCM5 - PARM, NPARM, MDHCNO,

c. Calls - BCS Submodule

BCSCON

3.2.11.4.1.6.3 Flowchart - See Figure 3.2.11-10.

3.2.11.4.1.7 Submodule VII - BCSOUT

3.2.11.4.1.7.1 Description -

- a. Language Used - FORTRAN IV.
- b. How Invoked - Subroutine called by any submode that needs to send a message to the OCS.
- c. Constraints and Limitations - The proper data values must be set for the message to be sent before it is called.
- d. Processing -
 1. Take care of first 8 bytes of message.
 2. If message number is not 1 or 10, go to Step 6.
 3. Encode time and BCS file title.
 4. If message number is 10, encode termination code.
 5. Go to Step 14.
 6. Encode HC number.
 7. If message number is 11, go to Step 14.
 8. If message number is not 13, go to Step 10.
 9. Encode date, time, new bias, old bias, and status; go to Step 14.
 10. Encode BCS target.
 11. If message number is 3, go to Step 14.
 12. If message number is not 5, set error message and return.
 13. Encode blocking/shadow HCs, sun vest. time, sun vector.
 14. Send message to OCS and return.

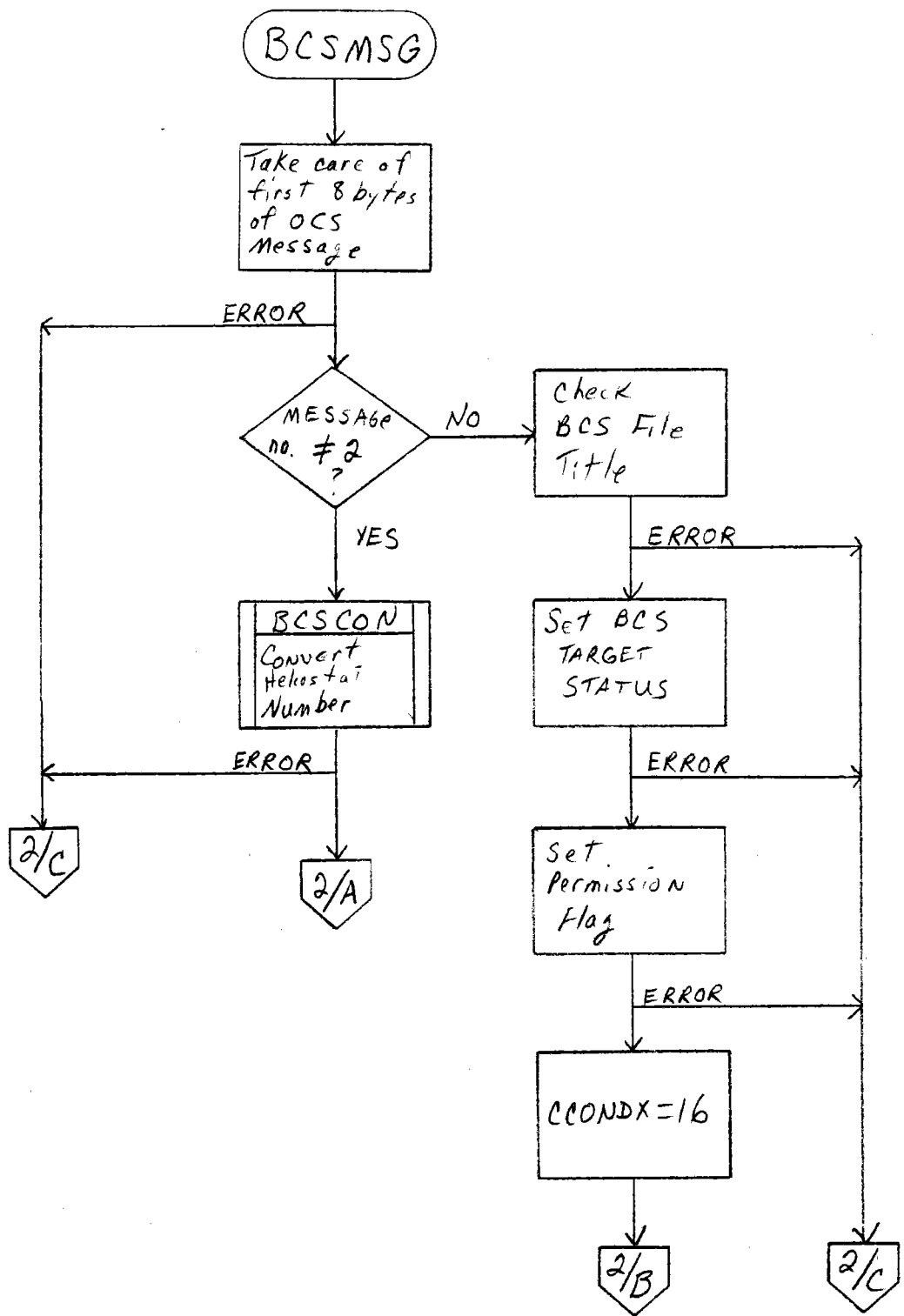


FIGURE 3.2.11-10 FLOWCHART -- BCSMSG

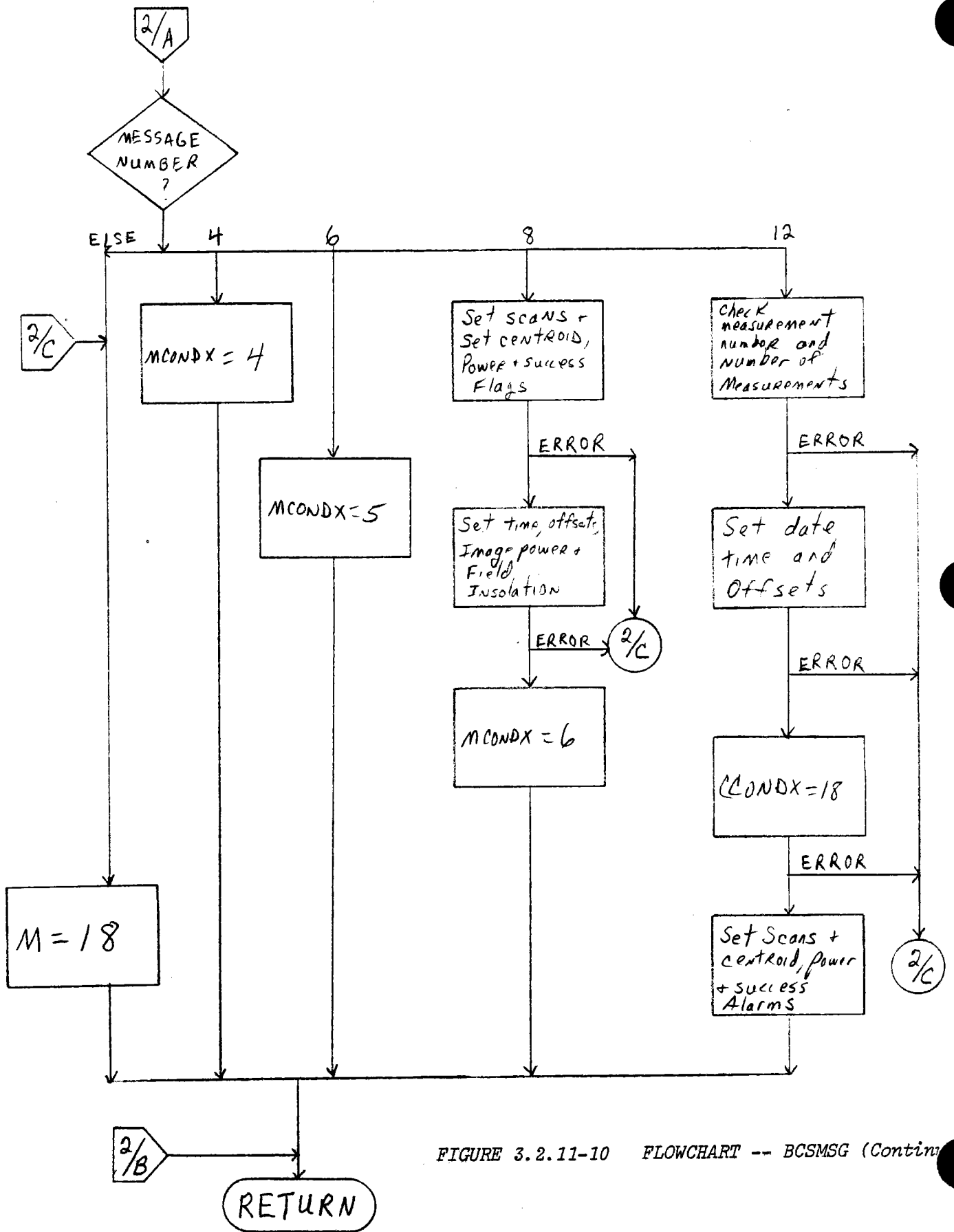


FIGURE 3.2.11-10 FLOWCHART -- BCSMSG (Continued)

3.2.11.4.1.7.2 Data, Logic, and Command Paths

a. Input Data

1. Local Common

BCSCM1 - WRK, POINT, T, WRKARR
BCSCM2 - YESF, CBUFF
BCSCM3 - MN
BCSCM4 - AZN, ELN, AZO, ELO
BCSCM5 - MDHCNO, BTC, TITLE

2. Global Common

COMIS5 - GTIMEG, SUNPOG
COMIS3 - GRSTSG

b. Output Data

1. Local Common

BCSCM1 - M

2. Other

HAC-OCS Message Sent to OCS

c. Calls - BCS Submodule -- BCSNWT

3.2.11.4.1.7.3 Flowchart - See Figure 3.2.11-11.

3.2.11.1.4.8 Submodule VIII - BCSMES

3.2.11.1.4.8.1 Description -

a. Language Used - FORTRAN.

b. How Invoked - Subroutine called whenever an error message needs to be displayed to the console, the CSO, or the alarms processor.

c. Constraints and Limitations - For alarms that have parameters, the parameters must be set up before BCSMES is called.

d. Processing -

1. If $1 < M < 16$, enqueue message to CSO and return.
2. If $99 < M < 121$, enqueue message to ALMQUE and return.
3. Write value of M to console and return.

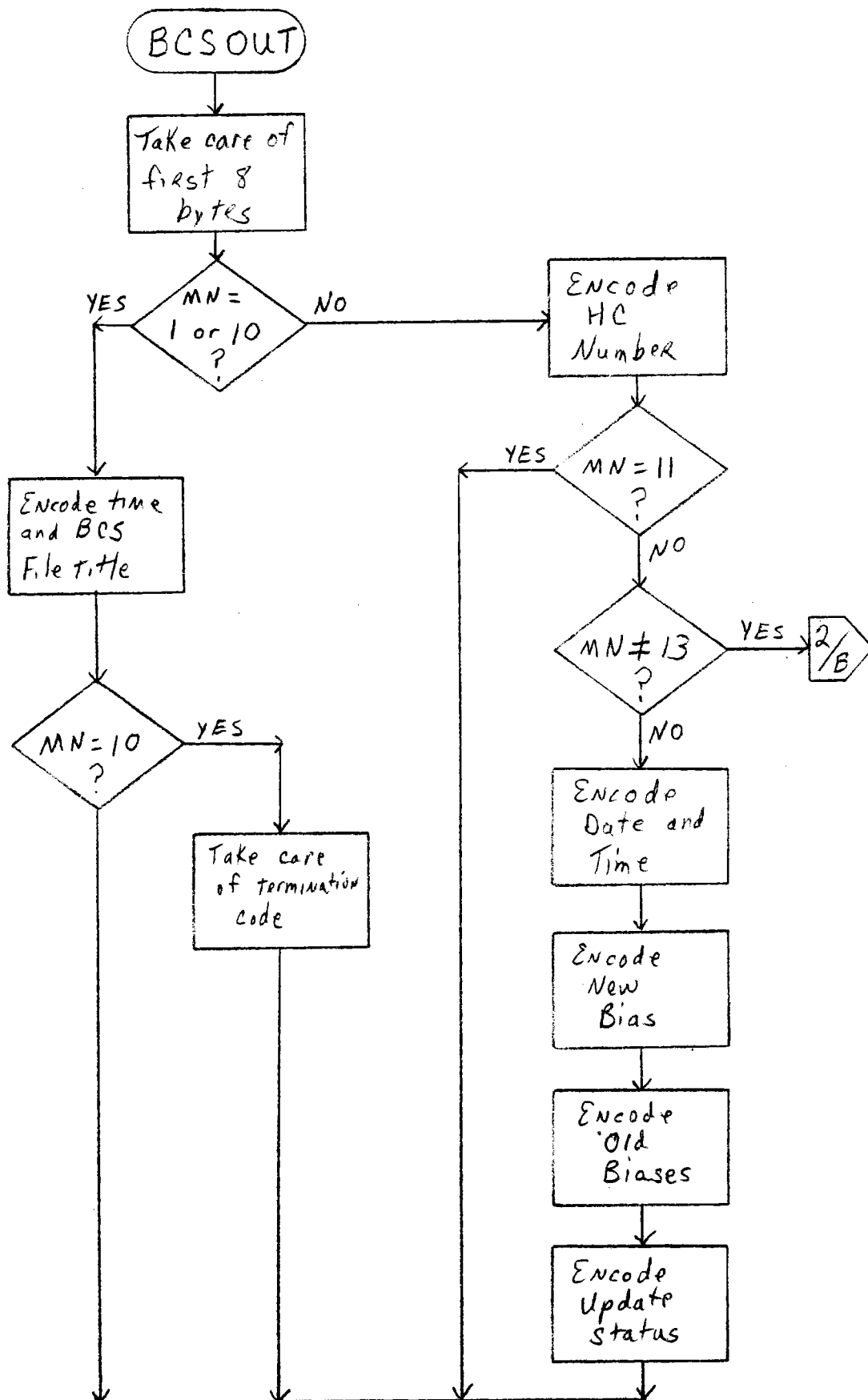


FIGURE 3.2.11-11 FLOWCHART -- BCSOUT

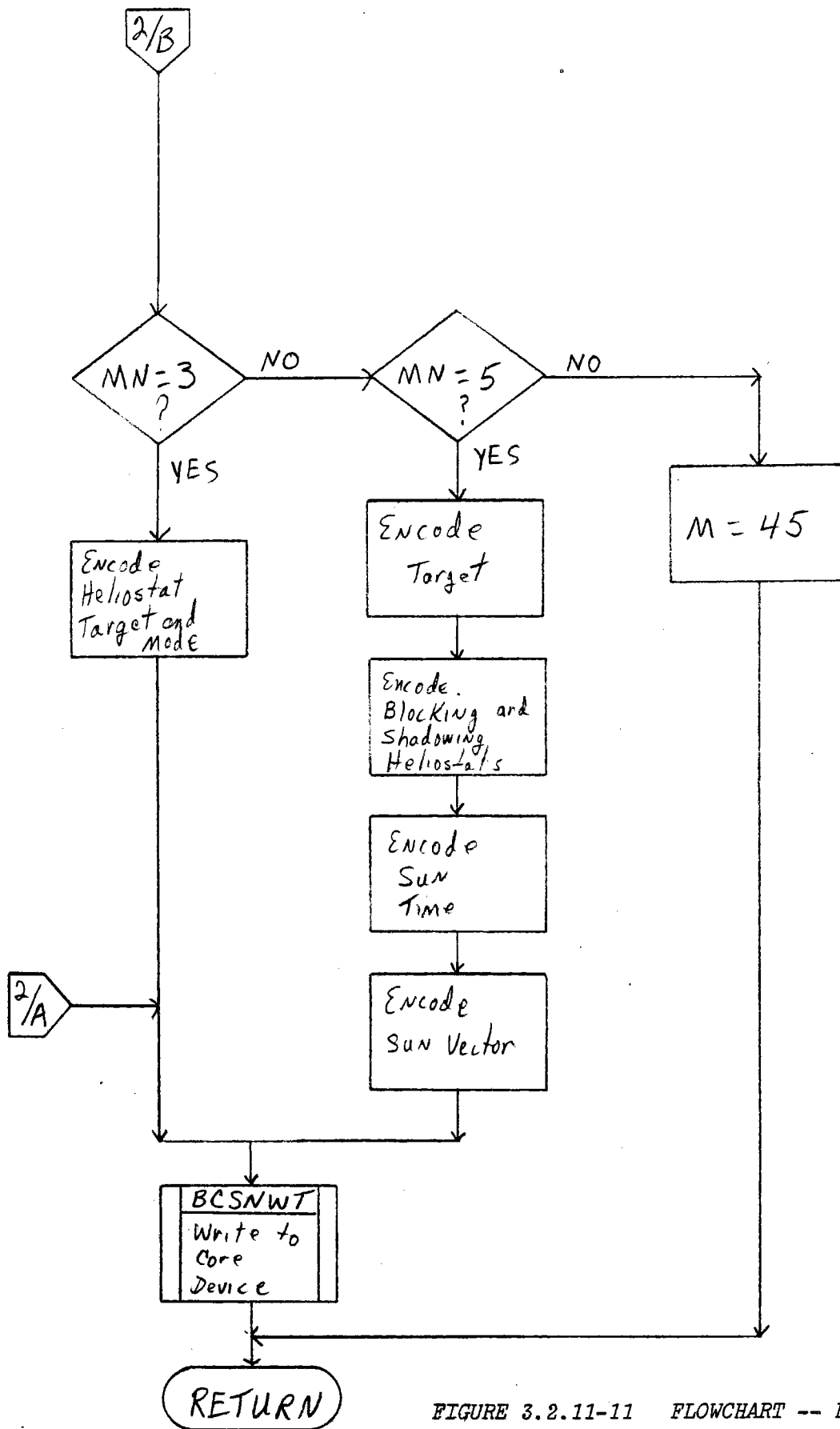


FIGURE 3.2.11-11 FLOWCHART -- BCSOUT (Continued)

3.2.11.1.4.8.2 Data, Logic, and Command Paths

a. Input Data

1. Local Common

BCSCM1 - M
BCSCM5 - PARM, NPARM

b. Output Data

1. BCS alarms (See Table 3.2.11-XX).
2. BCS messages to CSO (see Table 3.2.11-XXI).
3. Message number M on console. Meanings are described in Table 3.2.11-XXII.
4. Local Common BC5CM1 - MSTATE, T, CSTATE

c. Calls

1. Non-BCS Submodules

ENQU
ALMQUE

3.2.11.1.4.8.3 Flowchart - See Figure 3.2.11-12.

TABLE 3.2.11-XX BCS MESSAGES TO CSO

MESSAGE NO.	MESSAGE TEXT
1	Unexpected Command
2	Exceeded Iterations in Bias Calculations
3	Read Error on BCS Partition by BCS
4	Invalid File on BCS Partition
5	Magnetic Tape Read Error
6	Write Error on BCS Partition by BCS
7	Magnetic Tape File Format Invalid
8	Read Error on HCC Partition by BCS
9	End of File Found on HCC Partition by BCS
10	Read Error on HCB Partition by BCS
11	End of File Found on HCB Partition by BCS
12	Write Error on Partition HCB by BCS
13	Remote Write Error on Partition HCB by BCS
14	Invalid Parameter
15	Too Many Records on BCS File
16	BCS File Contains no Heliostats
17	BCS Bias Update Decision Required
18	BCS Bias Update Decision Not Required
19	BCS Offset Data Not Received From OCS
20	High Wind Stow Enabled
21	Range Not Allowed

TABLE 3.2.11-XXI UNEXPECTED BCS MESSAGES TO CONSOLE

MESSAGE NO.	MESSAGE TEXT
39	Bad CSTATE in BCSCCTL
40	Bad CSTATE in BCSP12
41	Bad CSTATE in BCSP31
42	Bad CSTATE in BCSP32
43	Bad CSTATE in BCSP33
44	Bad Field on BCS File
45	Bad Message Number in BCSOUT
46	Encoding Error in BCSOUT
47	Bad Message Number from MMI in BCSCMD
48	
49	HC Number Input by "BCSCALC" Doesn't Match HC Number Received From OCS in BCS-12 in BCSP32
50	Bad MSTATE in BCSMEA
51	Bad MSTATE in BCSMPA
52	Encode Error in BCSMPA
53	Encode Error in BCSGET
54	Encode Error in BCSCMD
55	Encode Error in BCSFILE
56	Bad MSTATE in BCSMPB
57	Bad MSTATE in BCSMPC
58	BCS-12 Timeout Processing Error in BCSP33
59	Encode Error in BCSMMC
60	Encode Error in BCSP12
61	Encode Error in BCSMPB
62	Encode Error in BCSP33
63	Read4 Not Done in BCSNET
64	Read Error on Link in BCSNET
66	Encode Error in BCS
67	Encode Error in BCSMPC
68	Write4 Not Done in BCSNWT
69	Write Error on Link in BCSNWT
70	Encode Error in BCSMEA

TABLE 3.2.11-XXII BCS ALARMS

99 'BCS TERMINATED BY OPERATOR'
 101 'NO TARTETS AVAILABLE FOR AUTOMATIC PROCESSING'
 102 'OK NOT RECEIVED FROM OCS FOR BCS PROCESSING'
 103 'TARGET ', A2, ' NOT AVAILABLE FOR BCS PROCESSING'
 104 'INVALID HC # ENCOUNTERED HC # A4
 105 'HC ', A4, ' NOT IN LEGAL MODE FOR BCS PROCESSING'
 106 'BLOCKING HC , A4, ' NOT IN LEGAL MODE'
 107 'ABORT HC MEASUREMENT DUE TO TIME OUT HC ', A4
 108 'BCS MEASUREMENT FINISHED FOR FILE ', 5A2
 109 'ENCOUNTERED ERRONEOUS OCS DATA IN MSG # ', A2
 110 'INVALID MSG RECEIVED FROM OCS MSG = ', A2
 111 'UNEXPECTED MSG FROM OCS MSG = ', A2
 112 'WRITE ERROR ON OCS DATA LINE POSSIBLE BCS DATA LOST'
 113 'BIAS CALCULATIONS AND UPDATES FINISHED FILE ', 5A2
 114 'BCS TERMINATED DUE TO LACK OF OCS RESPONSE'
 115 'CENTROID ALARM; HC ',A4 ' HORIZONTAL ',F6.2 ' FT VERTICAL ' F6.2 ' FT
 116 'POWER TOO LOW; HC ',A4, ' POWER ',F5.2, ' K WATTS'
 117 'BEAM OFF TARGET ',A1, ' HC ', A4
 118 'LOW LEVEL BEAM ON TARGET ',A1, ' HC ', A4
 119 'READ ERROR ON FILE AL2 BY BCS'
 120 'EOF EONCOUNTERED ON FILE AL2 BY BCS'
 121 'BCS PROCESSING STARTED ON FILE ', 5A2

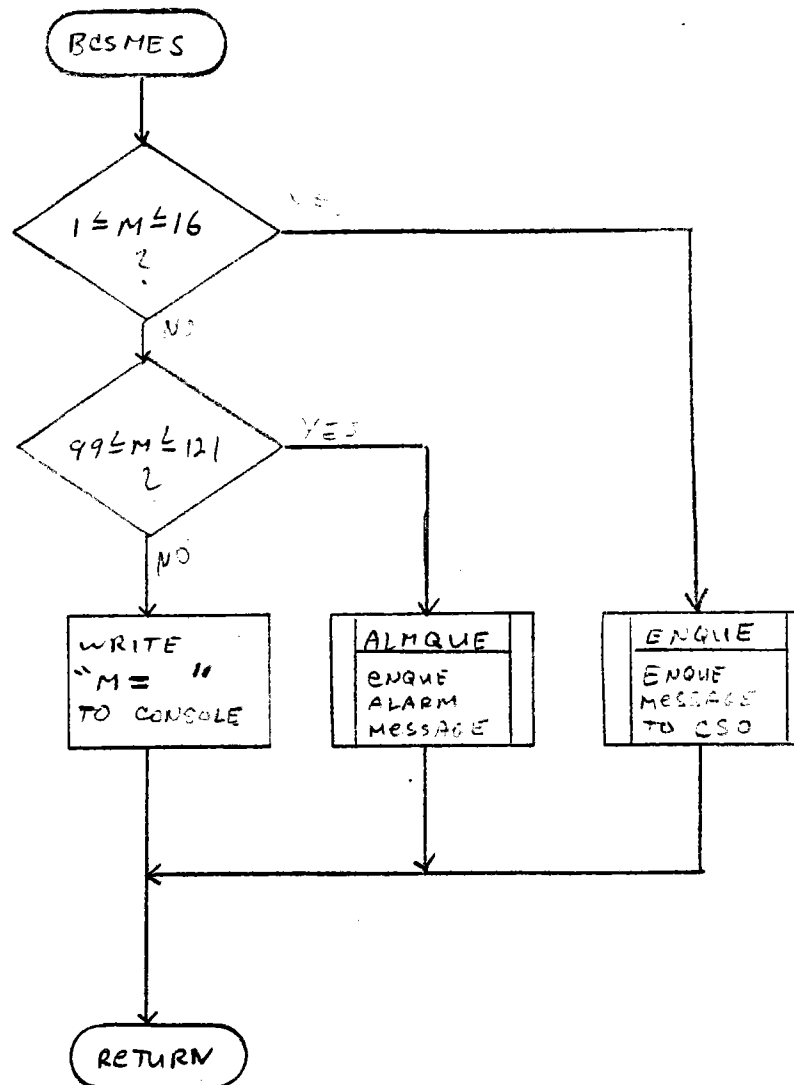


FIGURE 3.2.11-12 FLOWCHART - BCSMES

3.2.11.4.1.9 Submodule - BCSCON

3.2.11.4.1.9.1 Description -

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by various subroutines at various levels to convert a MDAC heliostat number to the corresponding MMA heliostat number.
- c. Constraints and Limitations - Input must be a valid MDAC heliostat number, and global common arrays MDNPRG and MD2HCG must be properly initialized.
- d. Processing - The MDAC row is isolated and checked for validity. If valid, the line number within the row is isolated. If the line number does not exceed the difference between the stored sum for this row and the next row, the HFCHC number is found directly in the MD2HCG array. If one of the above tests failed, an invalid MDAC number was input; and HFCHC is set to -1.

3.2.11.4.1.9.2 Data, Logic, and Command Paths -

a. Input Data

1. Local Common

BCSCM5 - MDHCNO

2. Global Common

COMIN3 - MDNPRG

COMIN2 - MD2HCG

b. Output Data

1. Local Common

BCSCM1 - M

BCSCM5 - HFCHC

3.2.11.4.1.9.3 Internal Data Description

ROW - First two digits of MDAC number

LINE - Last two digits of MDAC number

SUM - Sum in global common for above ROW.

3.2.11.4.1.9.4 Flowchart - See Figure 3.2.11-13 for the BCSCON flowchart.

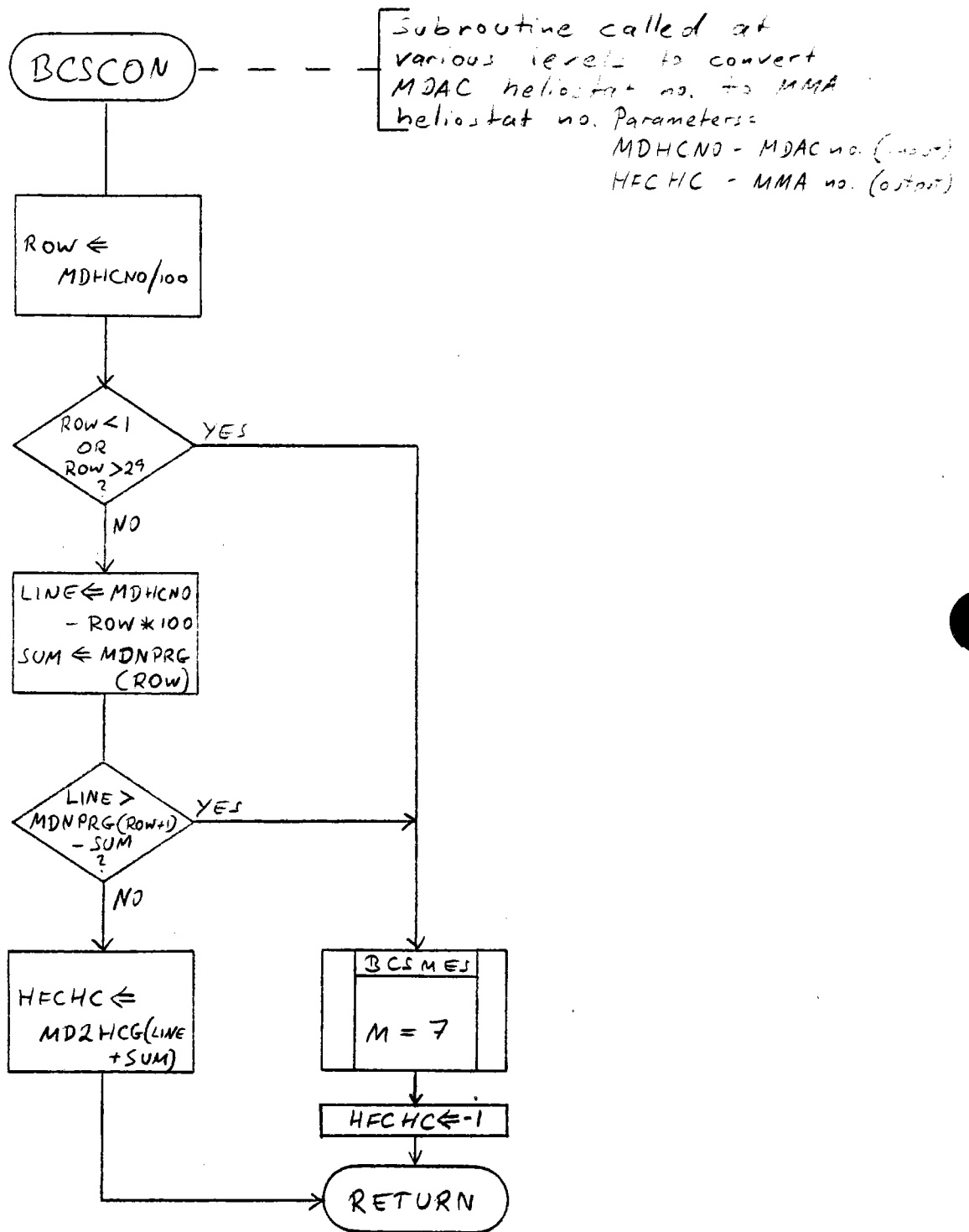


Figure 3.2.11-13 Flowchart - BCSCON

3.2.11.4.1.10 Submodule X - BCSCRX

3.2.11.4.1.10.1 Description

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by various subroutines when a BCS display needs to be displayed, updated, or cleared.
- c. Constraints and Limitations - A valid display number must be set for CRTDN.
- d. Processing -
 1. Check if cursor is in BCS portion of screen, and move the cursor if it is.
 2. Clear necessary lines on ISC.
 3. If CRTDN is 1, display BCS file.
 4. If CRTDN is 2, display biases.
 5. If CRTDN is 3, display BCS active.

3.2.11.4.1.10.2 Data, Logic, and Command Paths -

a. Input Data

1. Local Common

BCSCM1 - CSTATE, MSTATE
BCSCM5 - CRTDN

2. Global Common

COMIS5 - CRTWRG
COMIS4 - BCSLNG
COMQUE - IQUEUE

b. Output Data

1. Other

--Messages enqueued to CSO initialize
--Messages enqueued to CSO for BCS active
display on the ISC

BCS displays are shown in Figure 3.2.11-14.

Control File Display:

16 Lines of Alarm and Status Information		
17 Lines Conversational Area (command entry and feedback)		
*** ****		
BCS FILE: AAAAAAAAAA		
DATE: DD/MM/YY	EARLY TIME: HH:MM	LATE TIME: HH:MM
1 NNNN	11	51 NNNN
2		Color
.		Keys
.		
10 NNNN	20	60 NNNN

} 15 lines

Bias Calculation Display:

16 Lines of Alarm and Status Information		
26 Lines Conversational Area		
BCS Heliostat No: MMMM *** ***		
1.	DATE - TIME - H-OFFS - V-OFFS	
2.	" " " "	
3.	" " " "	
RESULTS:	NEW BIAS: AZ = + XXX.XX	EL = + XXX.XX
	OLD BIAS: AZ = + XXX.XX	EL = + XXX.XX

} 6 lines

BCS Active Display:

16 Lines of Alarm and Status Information		
30 Lines Conversational Area		
***** . . . *****		
BCS ACTIVE -		

} 2 lines

FIGURE 3.2.11-14 BCS DISPLAYS

c. Calls

1. BCS Submodules

BCSDS1
BCSDS2

2. Non-BCS Submodules

ENQU
INTLEN

3.2.11.4.1.10.3 Flowchart - See Figure 3.2.11-15.

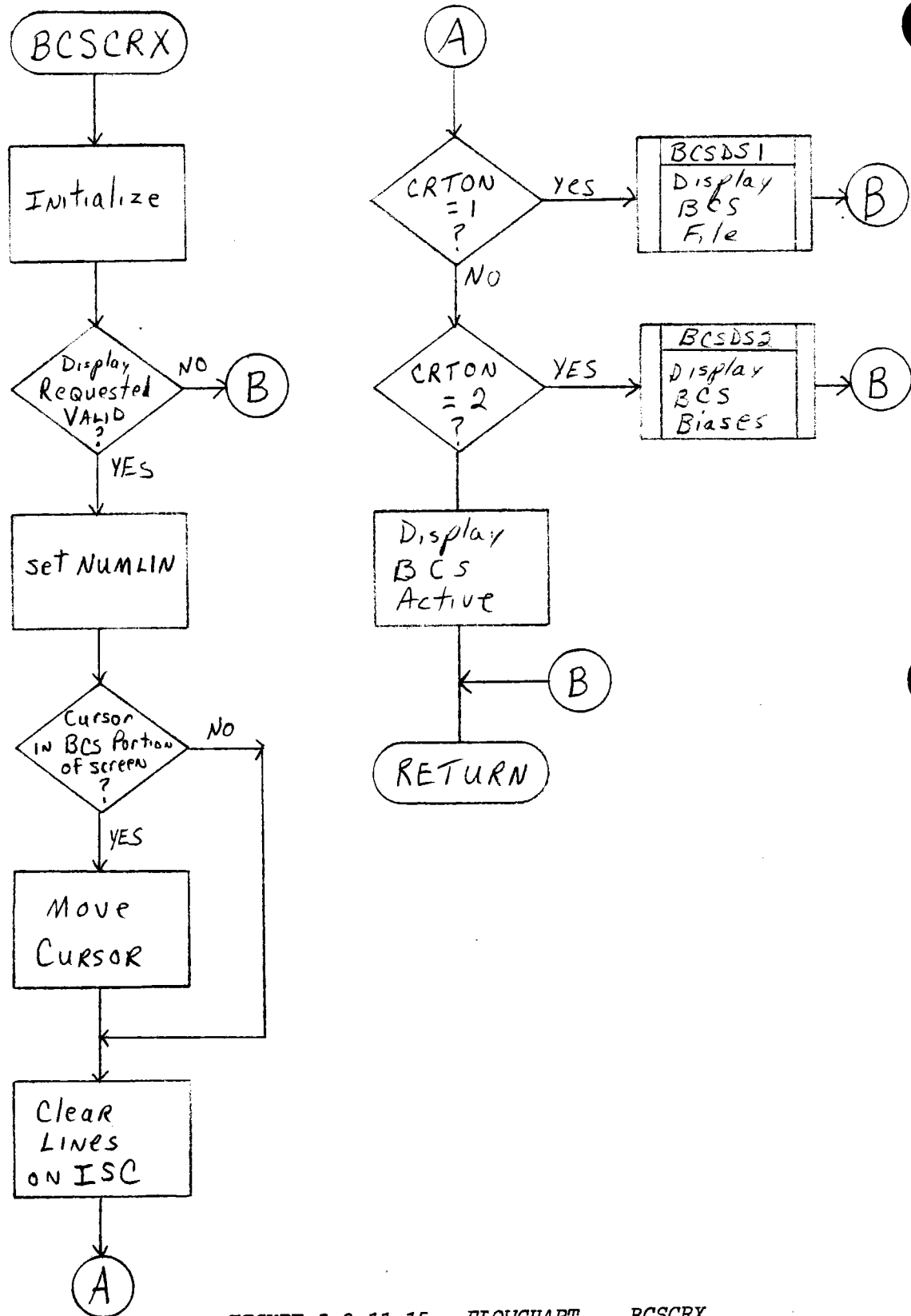


FIGURE 3.2.11-15 FLOWCHART -- BCSCRX

3.2.11.4.1.11 Submodule XI - BCSDS1

3.2.11.4.1.11.1 Description -

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by BCSCRX to produce the BCS file display.
- c. Constraints and Limitations - WRKARR and HSTAT must be initialized before calling.
- d. Processing -
 - 1. Set up and display the title.
 - 2. Take care of date and time.
 - 3. Set up each line up to 10 and display.

3.2.11.4.1.11.2 Data, Logic and Command Paths

a. Input Data

1. Local Common

BCSCM1 - WRKARR, HSTAT
BCSCM5 - TITLE, RDATE, ETIME, LTIME

2. Global Data

COMQUE - IQUEUE

b. Output Data

1. Other

Messages enqueued to CSO to produce the BCS file display on the ISC.

c. Calls

1. Non-BCS Submodules

ENQU

3.2.11.4.1.11.3 Flowchart - See Figure 3.2.11-16.

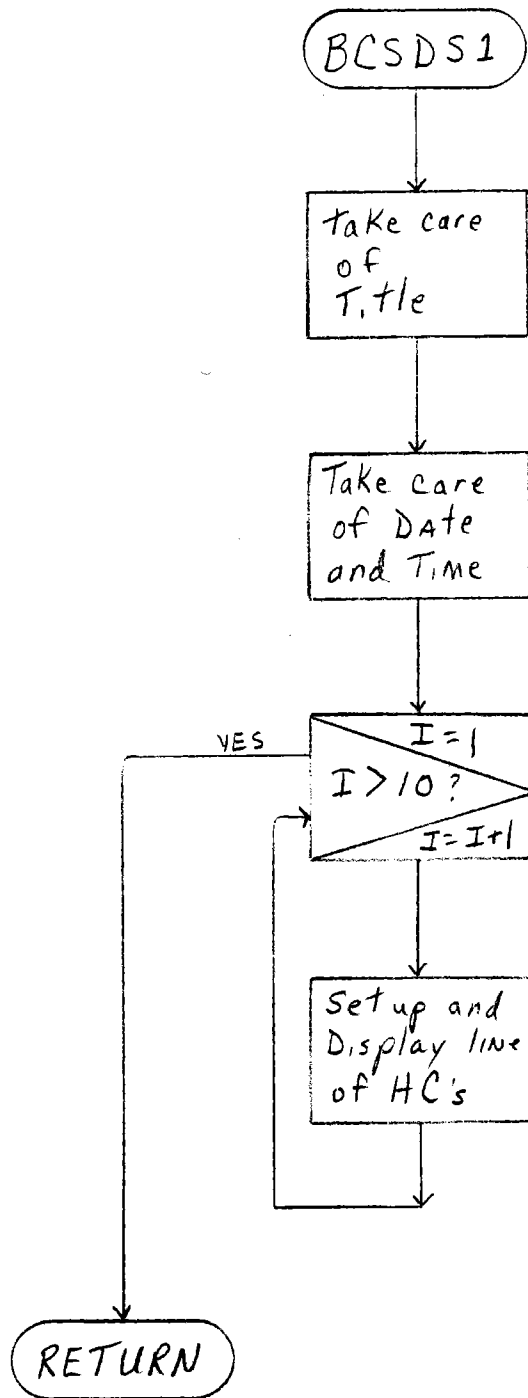


FIGURE 3.2.11-16 - BCSDS1

3.2.11.4.1.12 Submodule XII - BCSDS2

3.2.11.4.12.1 Description -

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by BCSCRX to produce the BCS bias display.
- c. Constraints and Limitations - The offset points with their date and time and the old and new biases must be set in local common before calling.
- d. Processing -
 1. Set up and display header for data points.
 2. Display date, time, and offsets for up to three data points.
 3. Set up and display header for biases.
 4. Display old and new biases.

3.2.11.4.1.12.2 Data, Logic, and Command Paths

a. Input Data

1. Local Common

BCSCM4 - MM, DD, YY, HO, MI, SE, U, V, AZN,
ELN, AZO, ELO

2. Global Common

COMQUE - IQUEUE

b. Output Data

1. Other

Messages enqueued to CSO to produce the BCS bias display on the ISC.

c. Calls

1. Non-BCS

ENQU

3.2.11.4.1.12.3 Flowchart - See Figure 3.2.11.17.

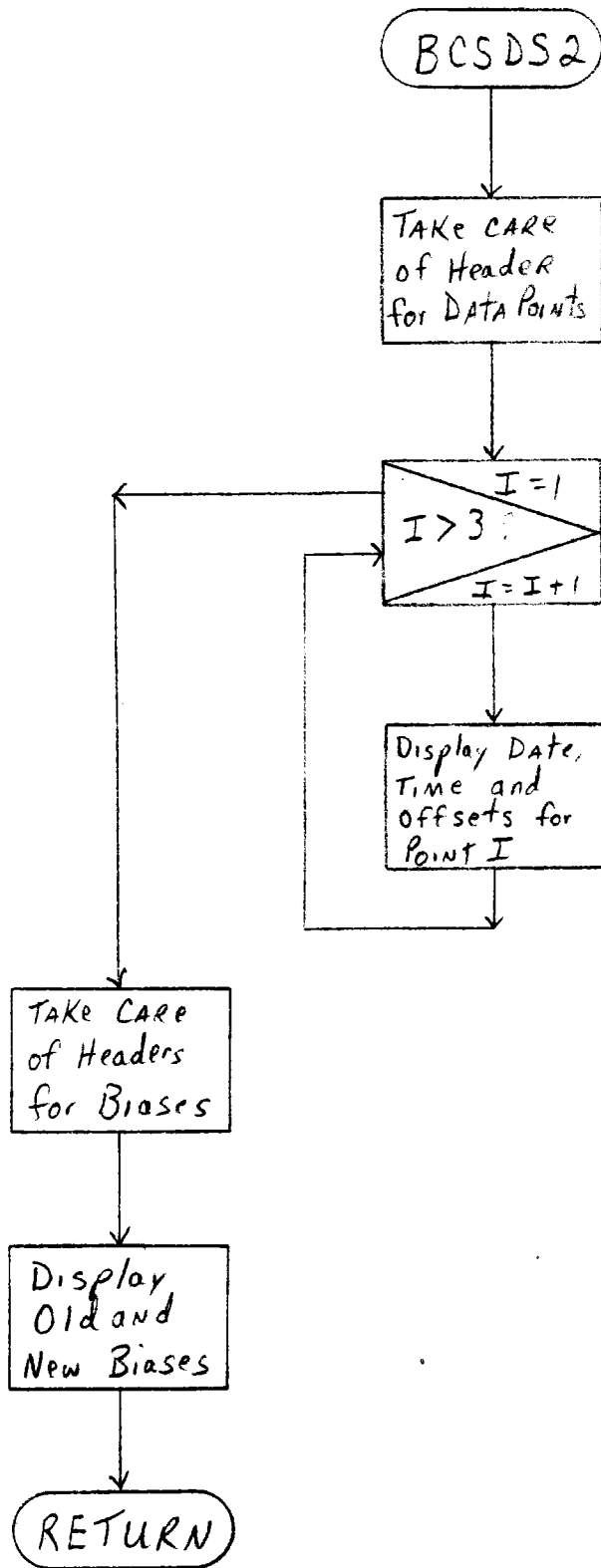


FIGURE 3.2.11-17 FLOWCHART -- BCSDS2

3.2.11.4.1.13 Submodule XIII - BCSP12

3.2.11.4.1.13.1 Description -

- a. Language Used - FORTRAN.
- b. How Invoked - Subroutine called by BCSCCTL when a new CSTATE requires Phase 1 or Phase 2 processing.
- c. Constraints and Limitations - This subroutine should only be called after a CSTATE transition has successfully occurred for Phase 1 and Phase 2 processing only.
- d. Processing -
 1. If CSTATE is 2, call BCSCMT and return.
 2. If CSTATE is 3, call BCSCFIL and return.
 3. If CSTATE is 4, call BCSCRNG and return.
 4. If CSTATE is 5, check BCS GO FLAG from BCS-2, check for busy targets, check if targets are operational, and return.
 5. If CSTATE is 24, set CSTATE to COLD and COLD to 24 and return.
 6. If CSTATE is 6, do initialization for measurement processing and return.
 7. If CSTATE is 25 or 26, set variables, call BCSCMES, call BCSCOUT, set EXITF to .TRUE., and return.

3.2.11.4.1.13.2 Data, Logic, and Command Paths

a. Input Data

1. Local Common

BCSCM1 - CSTATE, COLD
BCSCM2 - RI
BCSCM3 - GOF, TSTAT
BCSCM5 - Title

2. Global Common

COMIS4 - BCSCBYG

b. Output Data

1. Local Common

BCSCM1 - M, TIMER, T, POINT, MSTATE, MCONX,
COLD, CSTATE, EXITF, STSF, STS7F, STS9F, STS10F
BCSCM3 - MN
BCSCM5 - NPARM, PARM, BTC

2. Global Common

COMIS4 - BCSBYG

c. Calls

1. BCS Submodules

BCSMT	BCSOUT
BCSFIL	
BCSRNG	
BCSMES	
BCSMEA	

3.2.11.4.1.13.3 Flowchart - See Figure 3.2.11-18.

731

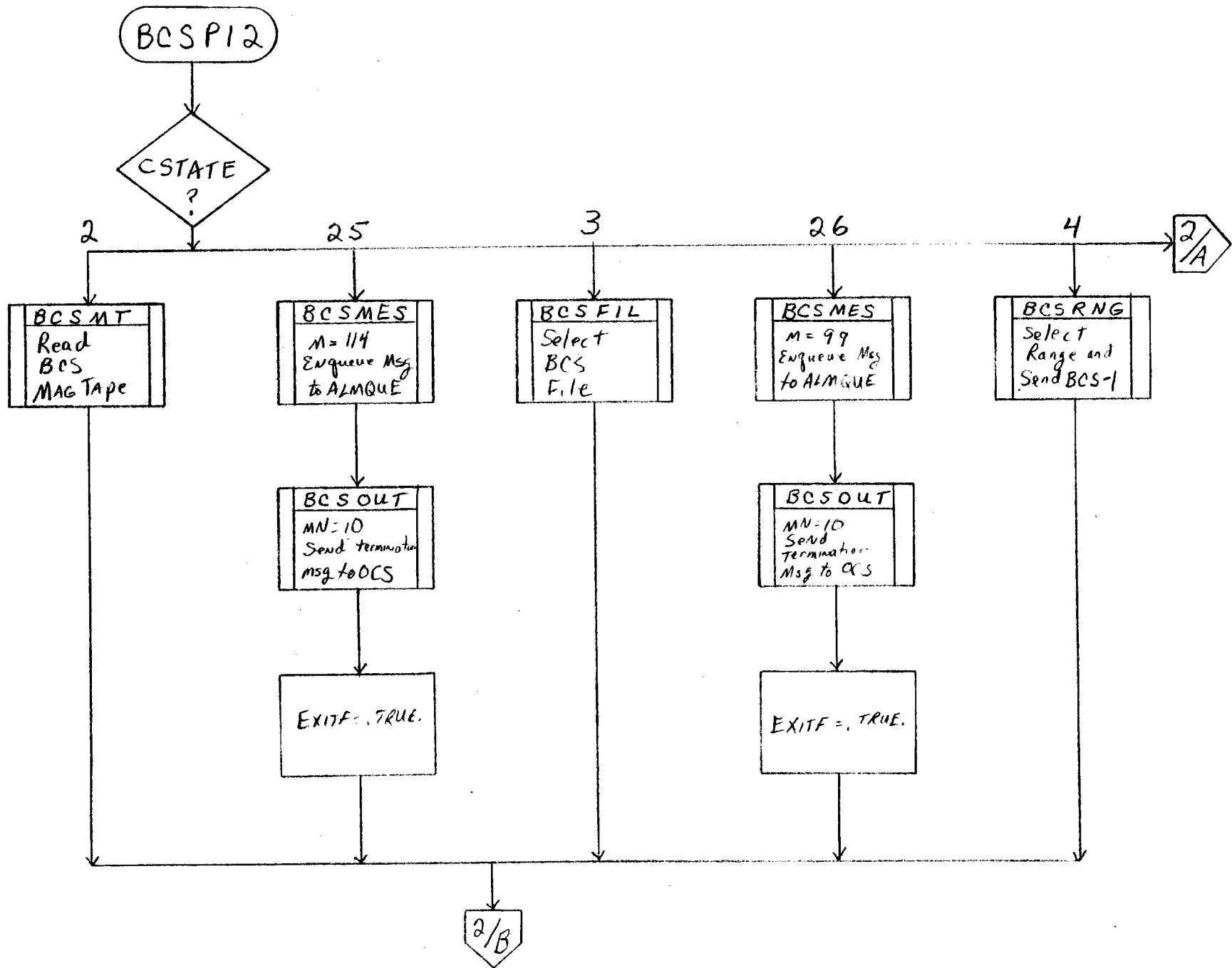


FIGURE 3.2.11-18 FLOWCHART - BCSP12

732

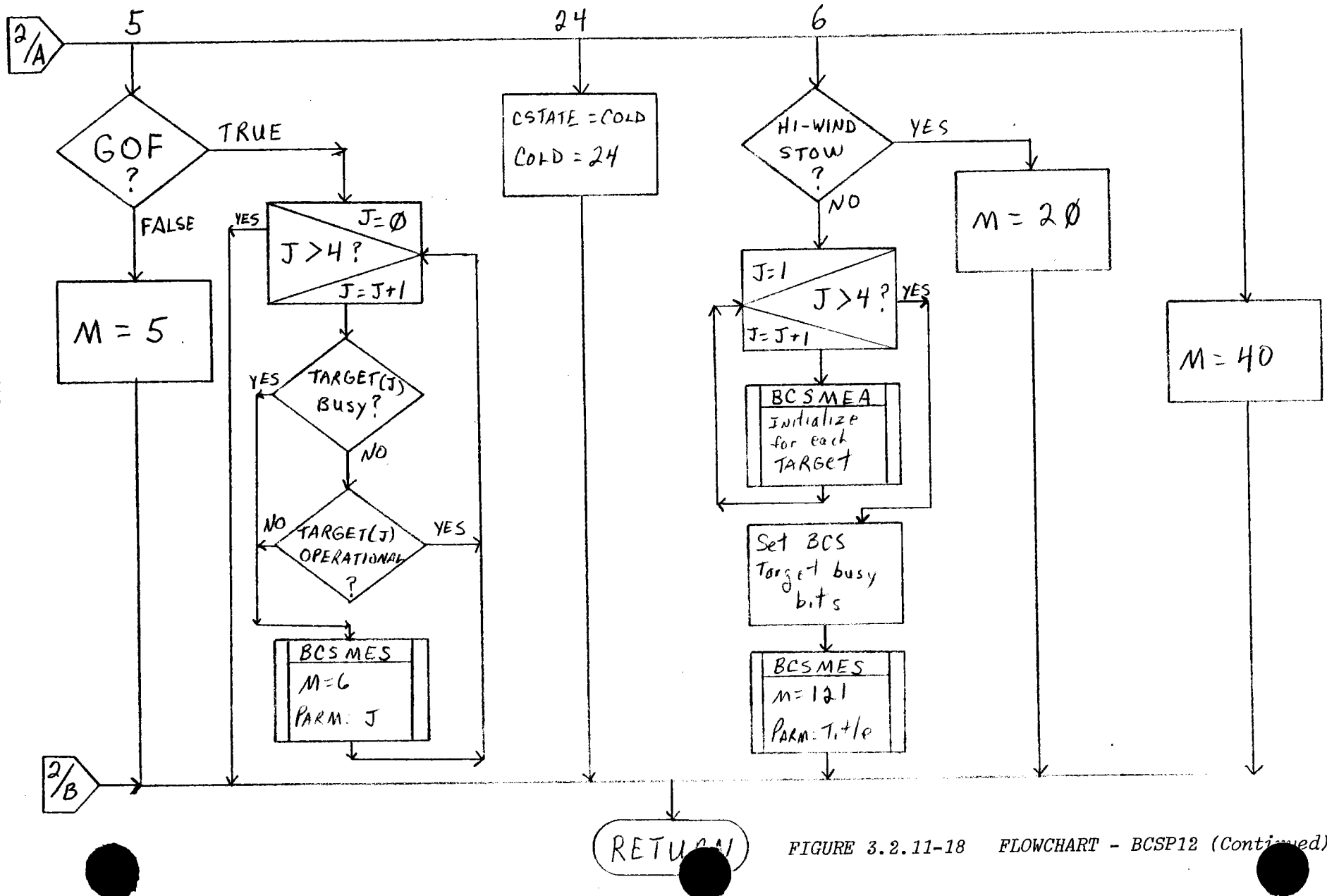


FIGURE 3.2.11-18 FLOWCHART - BCSP12 (Continued)

3.2.11.4.1.14 Submodule XIV - BCSMT

3.2.11.4.1.14.1 Description -

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by BCSP12 in response to a BCSXTAPE command.
- c. Constraints and Limitations - BCSMT will read only three BCS files in the appropriate format (see Table 3.2.11-XXIII); otherwise an error message is displayed.
- d. Processing -
 - 1. Read three BCS files from one nine-track magnetic tape and put on the BCS disk partition.
 - 2. Display file.

3.2.11.4.1.14.2 Data, Logic, and Command Paths

a. Input Data

1. Other

Magnetic tape containing three BCS files.

b. Output Data

1. Local Common

BCSCM1 - M

2. Other

Three BCS files to BCS disk partition

c. Calls

MAXIV Library Call
CMRI
CMR

3.2.11.4.1.14.3 Flowchart - See Figure 3.2.11-19.

TABLE 3.2.11-XXIII
BCS PROCESSING FILE FORMAT

Card	Card Column	Format	Description
1	1-8	Alpha	File designator, 8 characters, as follows: BCS file *
	20-29	Alpha	File title - user input into creation program, 10 characters
2	1-6	Alpha	Date this file is to be run MMDDYY
	10-13	Integer	Earliest time file is to be started (HHMM, plant local time) *
	20-23	Integer	Latest time file is to be started (HHMM, plant local time) *
3-62	1-4	Integer	Heliostat number to be measured
	10-13	Integer	Blocking/shadowing heliostat **
	20-23	Integer	Blocking/shadowing heliostat
	30-33	Integer	Blocking/shadowing heliostat
	40-43	Integer	Blocking/shadowing heliostat
	50-53	Integer	Blocking/shadowing heliostat

* 24-Hour clock format

** Zero fill will be used when less than 5 blocking/shadowing heliostats are required.

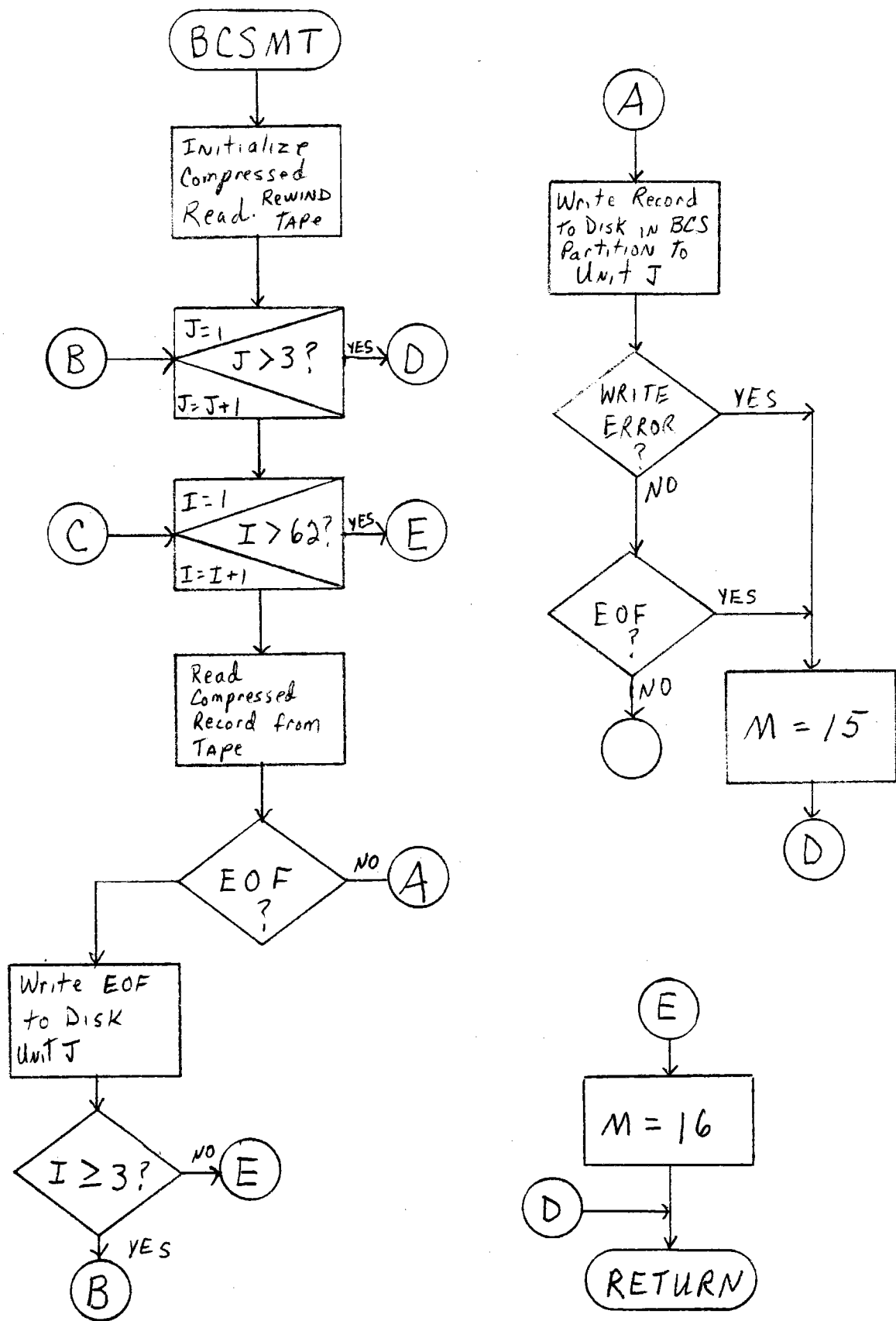


FIGURE 3.2.11-19 FLOWCHART - BCSMT

3.2.11.4.1.15 Submodule XV - BCSFIL

3.2.11.4.1.15.1 Description -

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by BCSP12 in response to a BCSFILE command.
- c. Constraints and Limitations - It is expected that three BCS files exist on partition BCS.
- d. Processing -
 1. Clear work arrays.
 2. Find proper file on BCS partition.
 3. Read BCS file header information into local common.
 4. Read HC and B/SHC numbers into WRKARR and convert MDHCNO to HFCHC and put in WRK.
 5. Display file.

3.2.11.4.1.15.2 Data, Logic, and Command Paths

- a. Input Data
 1. Local Common
 - BCSCM2 - FN
 2. Other
 - BCS DISK PARTITION
- b. Output Data
 1. Local Common
 - BCSCM1 - WRKARR, WRK, HSTAT, M
 - BCSCM2 - CRTF
 - BCSCM5 - TITLE, RDATE, ETIME, LTIME
 - BCSCM6 - DSIF
- c. Calls
 1. BCS Submodules
 - BCSCON

3.2.11.4.1.15.3 Flowchart - See Figure 3.2.11-20.

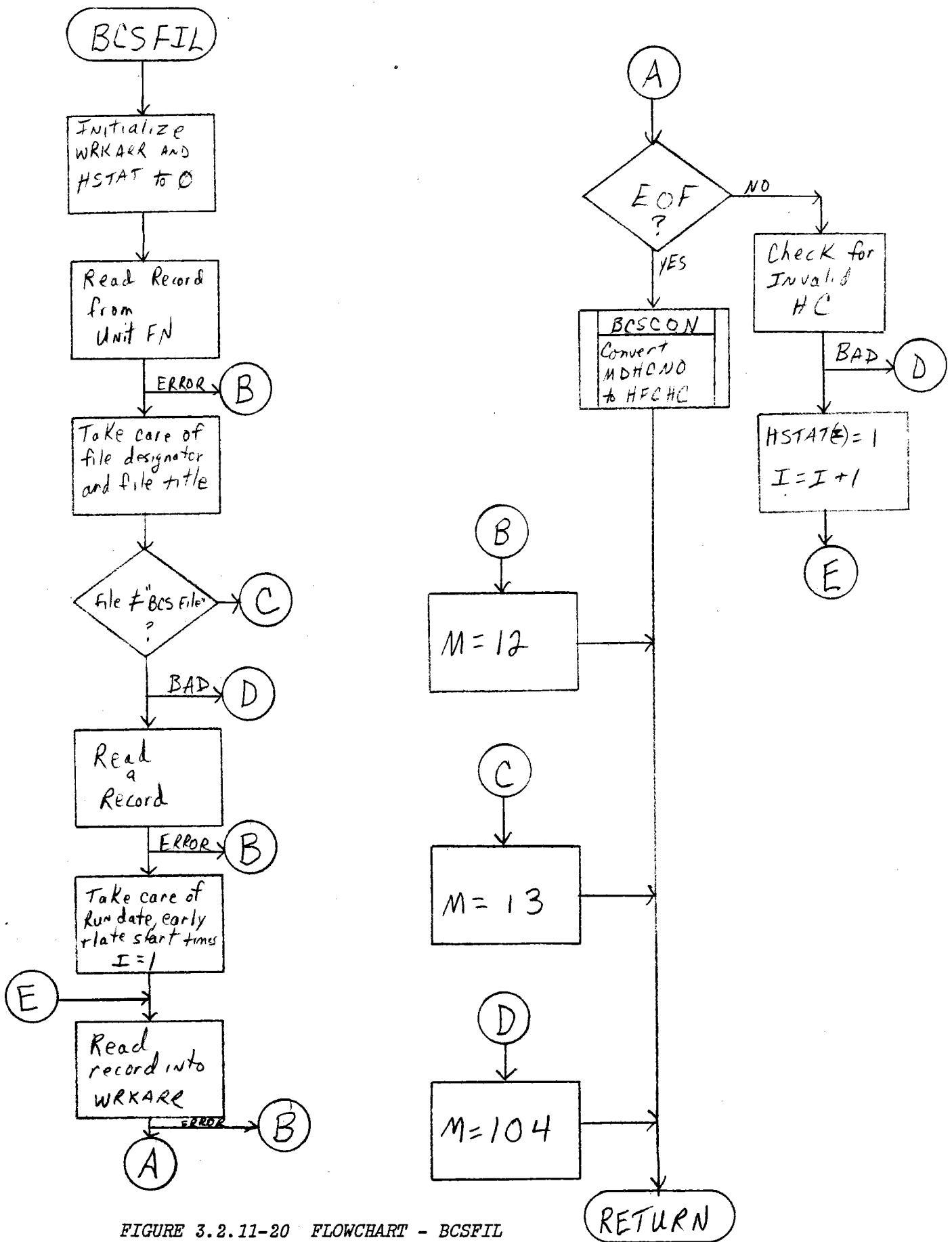


FIGURE 3.2.11-20 FLOWCHART - BCSFIL
737

3.2.11.4.1.16 Submodule XVI - BCSRNG

3.2.11.4.1.16.1 Description -

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by BCSP12 in response to BCSRNG command.
- c. Constraints and Limitations - Executed only after a BCS file has been selected by BCSFIL command.
- d. Processing -
 - 1. Set heliostat status (HSTAT) for range selected.
 - 2. Display file with selected range.
 - 3. Send BCS-1 message to OCS and set timer.

3.2.11.4.1.16.2 Data, Logic, and Command Paths

a. Input Data

1. Local Common

BCSCM1 - HSTAT
BCSCM2 - R1, R2

b. Output Data

1. Local Common

BCSCM1 - M, TIMER, HSTAT
BCSCM2 - CRTF
BCSCM3 - MN
BCSCM6 - DSIF

c. Calls

1. BCS Submodules

BCSOUT

3.2.11.4.1.16.3 Flowchart - See Figure 3.2.11-21.

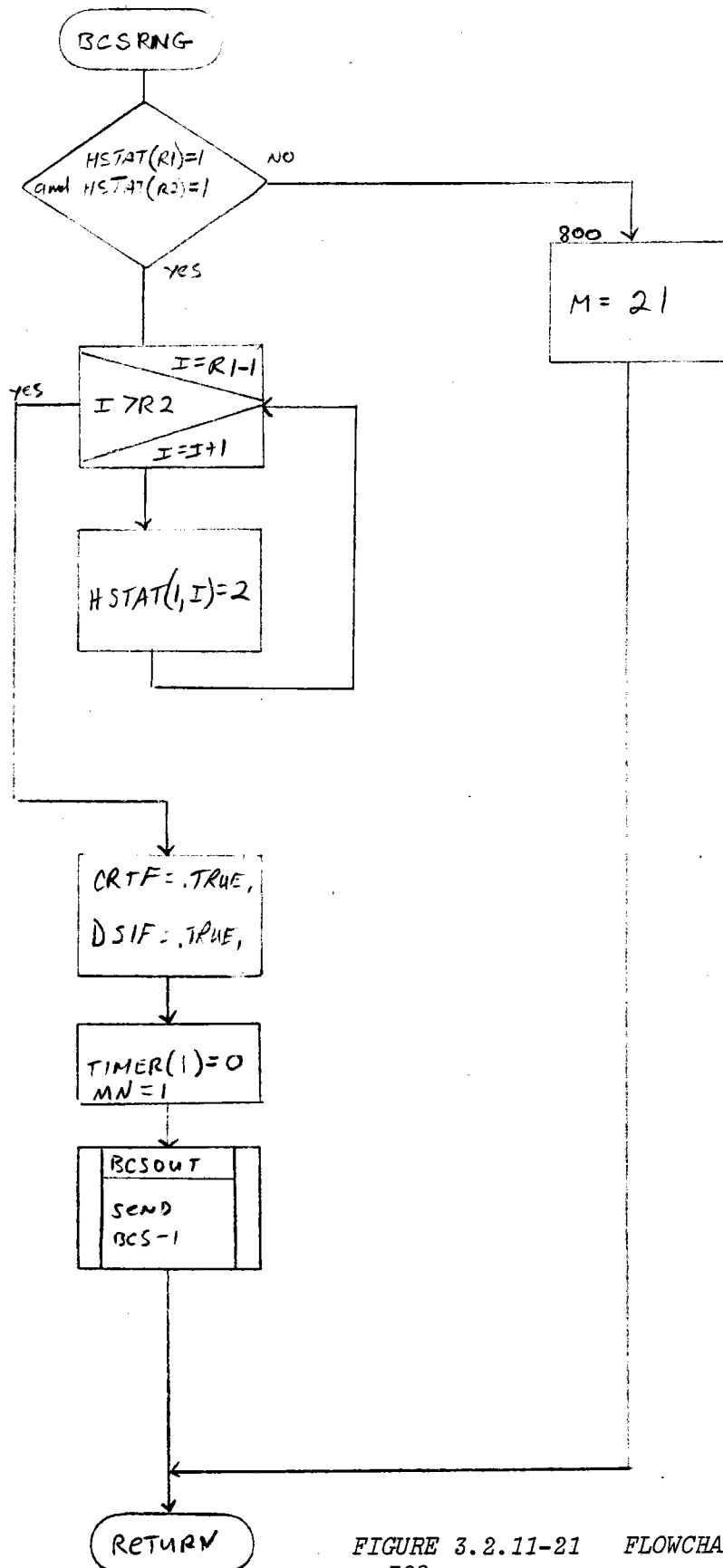


FIGURE 3.2.11-21 FLOWCHART - BCSRNG
739

3.2.11.4.1.17 Submodule XVII - BCSP31

3.2.11.4.1.17.1 Description -

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by submodule BCSCCTL when Phase 3, Method 1 processing is needed.
- c. Constraints and Limitations - Should only be called after a successful transition to a Phase 3, Method 1 CSTATE.
- d. Processing - Processing for CSTATES 8 through 13 are done here. The following describes each CSTATE:
 1. CSTATE 8 - Clear date, time, and offsets in local common.
 2. CSTATE 9 - Write message to logger and put date, time, and offsets in local common.
 3. CSTATE 10 & 11 - Put date, time, and offsets in local common for second and third data points.
 4. CSTATE 12 - Do bias calculations; log results; and if there is a significant change in biases, display results; otherwise, send BCS-13 message to OCS and set CSTATE = B.
 5. CSTATE 13 - Put old display back up and send BCS-13 message to OCS. If update desired, then update file HCB on disk.

3.2.11.4.1.17.2 Data, Logic, and Command Paths

a. Input Data

1. Local Common

BCSCM1 - CSTATE
BCSCM2 - IHC
BCSCM4 - AZBD, ELBD, BIT

2. Global Common

HCST3G

b. Output Data

1. Local Common

BCSCM1 - M, T, CSTATE
BCSCM3 - CAF12, PAF12
BCSCM4 - NP, MM, DD, YY, HO, MI, SE, HOUR, U, V
BCSCM6 - DS1F, DS2F, DS3F

c. Calls

1. BCS Submodule

BCSCAL BCSCON
BCSLOG BCSOUT
BCSCRX
BCSBIA

3.2.11.4.1.17.3 Flowchart - See Figure 3.2.11-22.

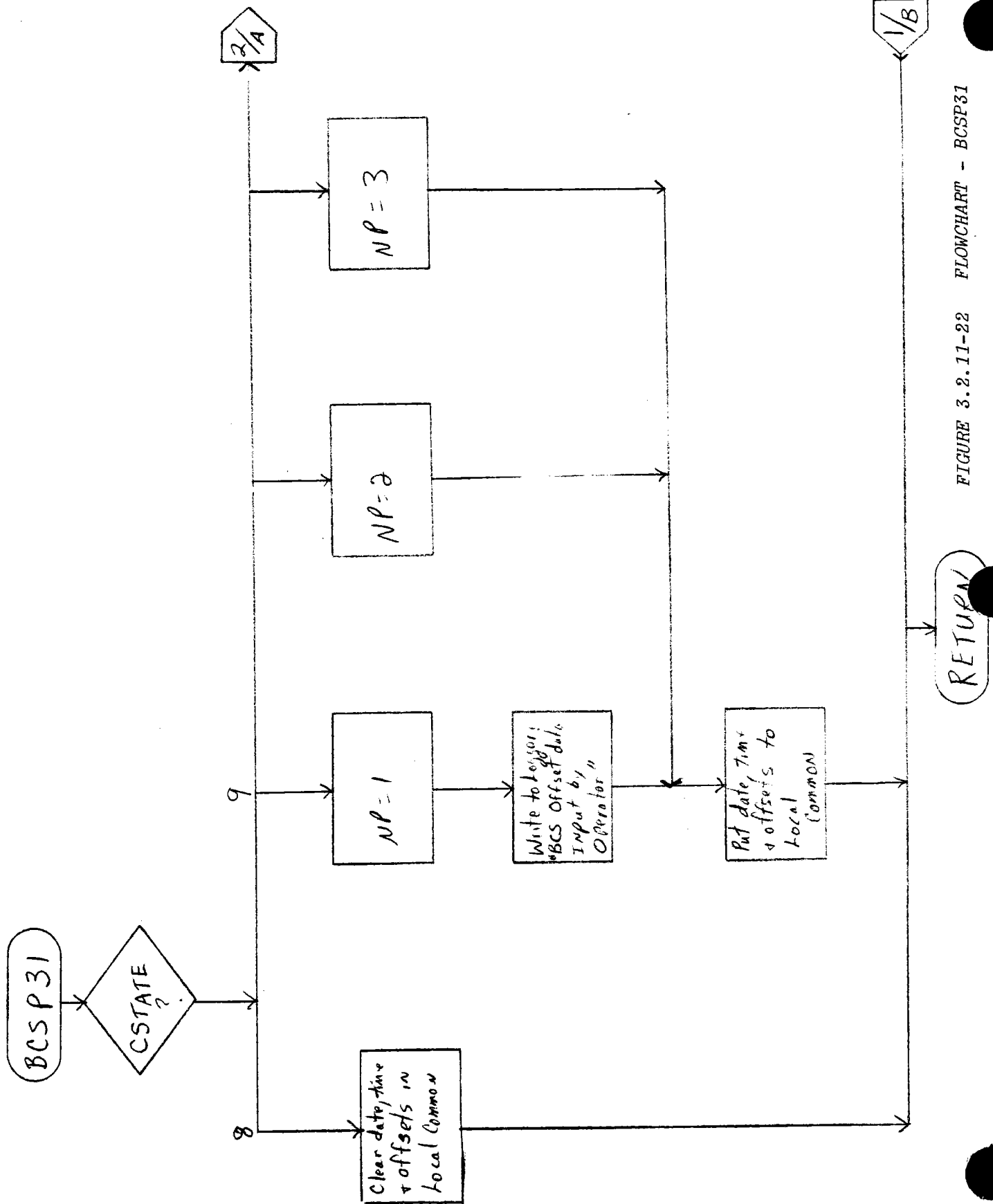


FIGURE 3.2.11-22 FLOWCHART - BCSF31

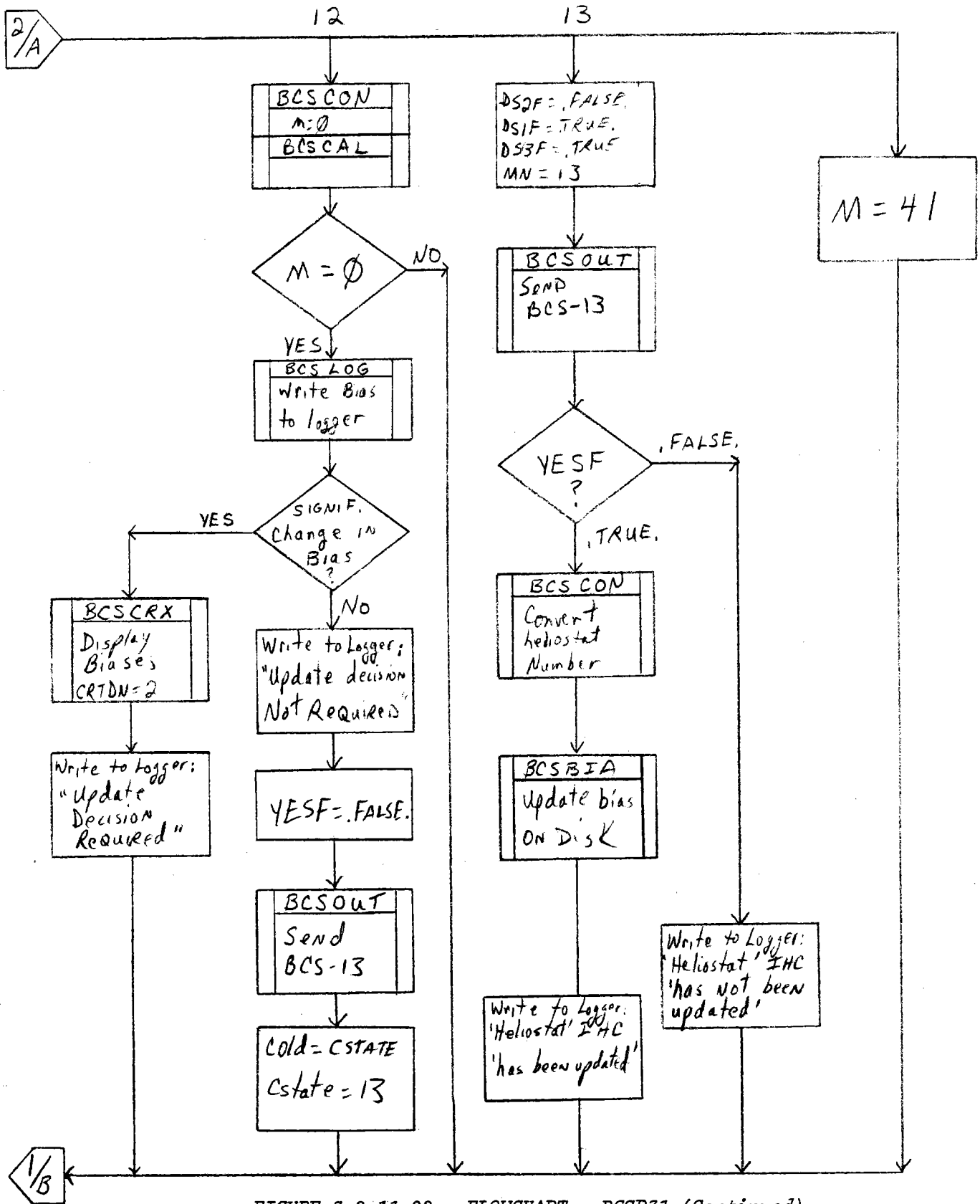


FIGURE 3.2.11-22 FLOWCHART - BCSP31 (Continued)

3.2.11.4.1.18 Submodule XVIII BCSP32

3.2.11.4.1.18.1 Description

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by BCCTL when Phase 3, Method 2 processing is needed.
- c. Constraints and Limitations - Should only be called after a successful transition to a Phase 3, Method 2 CSTATE.
- d. Processing - Processing for CSTATES 15, 16, 18 are done here. This is described below.
 1. CSTATE 15 - Send BCS-11 message to OCS.
 2. CSTATE 16 - If all BCS-12 message received from OCS, then do bias calculations and log; if there is significant change in the biases, display the results and set CSTATE = 17; otherwise, log and send BCS-13 message to OCS and set CSTATE = 18.
 3. CSTATE 18 - Put up old display and send BCS-13 to OCS. If biases are to be updated, then write new values to DISK FILE HCB.

3.2.11.4.1.18.2 Data, Logic, and Command Paths

a. Input Data

1. Local Common

BCSCM1 - CSTATE
BCSCM2 - CHC
BCSCM3 - MEAS, NMEAS, MHC, U12, V12, MM12,
DD12, YY12, H12, M12, S12
BCSCM4 - AZBD, ELBD, BIT, HOUR, AZNEW, ELNEW, HCN
BCSCM5 - HFCHC, AZIM, ELEV

b. Output Data

1. Local Common

BCSCM1 - MITIMER, CSTATE, M, COLD
BCSCM2 - YESF
BCSCM4 - MM, DD, YY, HO, MI, SE, HOUR, U, V
BCSCM6 - DS1F, DS2F, DS3F
BCSCM5 - MDHCNO, CRTDN

c. Calls

1. BCS Submodules

BCSOUT BCSBIA
BCSCON
BCSCAL
BCSLOG
BCSCRX

3.2.11.4.1.18.3 Flowchart - See Figure 3.2.11-23.

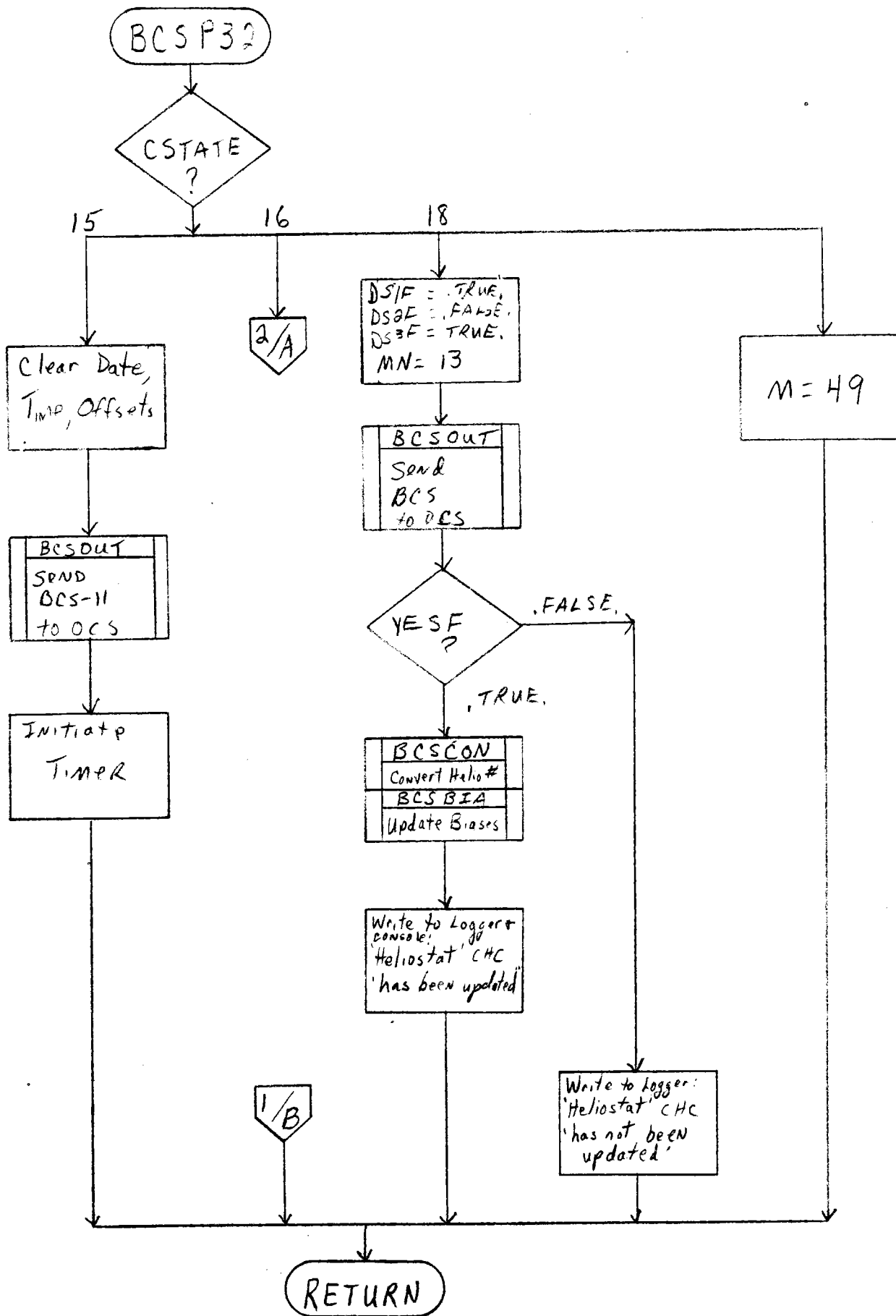


FIGURE 3.2.11-23 FLOWCHART - BCSP32
746

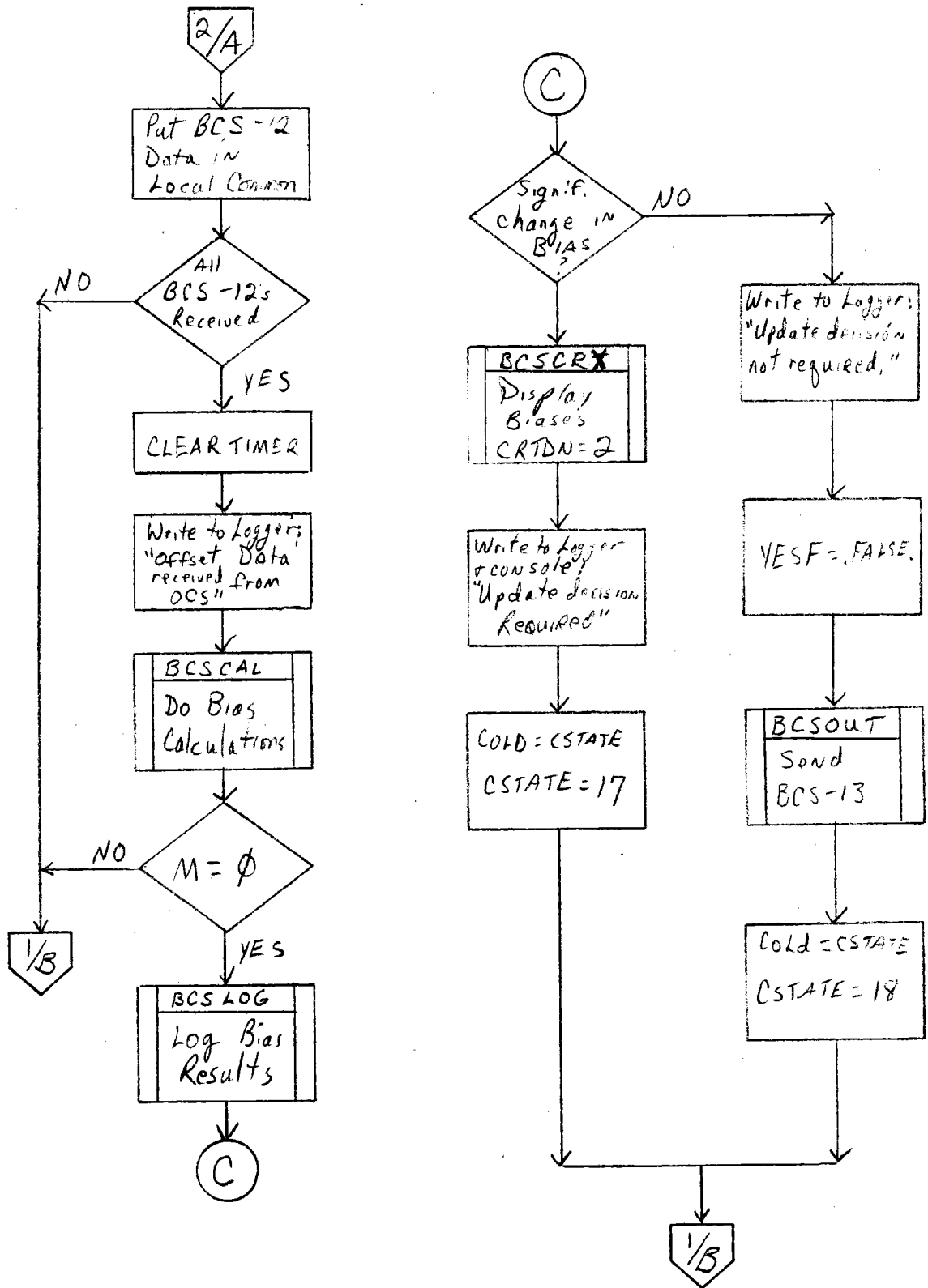


FIGURE 3.2.11-23 FLOWCHART - BCSP32 (Continued)

3.2.11.4.1.19 Submodule XIX - BCSP33

3.2.11.4.1.19.1 Description

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by BCCTL when Phase 3, Method 2 processing is needed.
- c. Constraints and Limitations - Should only be called after a successful transition to a Phase 3, Method 3 CSTATE.
- d. Processing - Processing for CSTATES 19, 20, 22 are done here and described below.
 1. CSTATE 19
 - (a) Initialize for semiautomatic bias calculations.
 - (b) Get next HC from file; if EOF display message and get CSTATE = 23.
 - (c) Send BCS-11 message to OCS.
 2. CSTATE 20
 - (a) If all BCS-12 message are received from OCS, do below.
 - (b) Do BCS bias calculations.
 - (c) Log results.
 - (d) If significant change in biases, display and set CSTATE = 21; otherwise, send BCS-13 message to OCS and set CSTATE = 19 and go to Step 1.(b).
 3. CSTATE 22
 - (a) Set up old display.
 - (b) Send BCS-13 to OCS.
 - (c) If update desired, write new biases to disk partition HCB.
 - (d) Go to Step 1.(b).

3.2.11.4.1.19.2 Data, Logic, and Command Paths

a. Input Data

1. Local Common

BCSCM1 - CSTATE, T, COLD, WRKARR
BCSCM2 - CHC, R1, R2, YESF
BCSCM3 - MEAS, NMEAS, MHC, U12, V12, MML2, DD12,
YY12, H12, M12, S12, ELTIME, I16BIT, MT, SUSPND
BCSCM4 - AZBD, ELBD, BIT, HOUR, AZNEW, ELNEW
BCSCM5 - HFCHC, TITLE

b. Output Data

1. Local Common

BCSCM1 - M, TIMER, CSTATE, COLD, HSTAT, T
BCSCM4 - MM, DD, YY, HO, MI, SE, HOUR, U, V, HCN, NP
BCSCM6 - DS1F, DS2F, DS3F
BCSCM5 - AZIM, CRTDN, CRTHC, ELEV, MDHCNO, NPARM, PARM, REP
PCSCM3 - MEAS, MN, NMEAS,

c. Calls

1. BCS Submodules

BCSOUT BCSMES
BCSCON
BCSCAL
BCSLOG
BCSCRX
BCSBIA

3.2.11.4.1.19.3 Flowchart - See Figure 3.2.11-24.

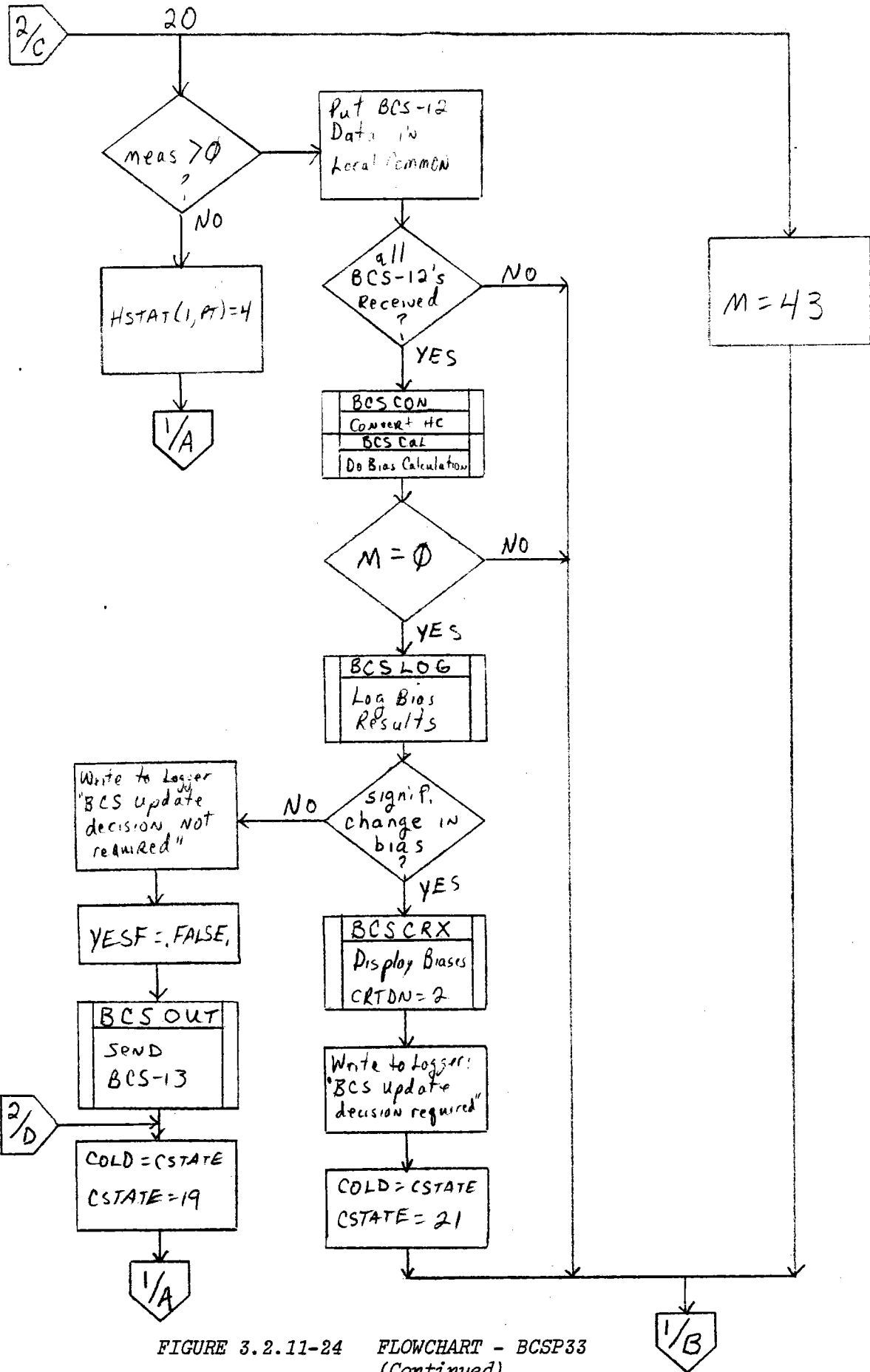


FIGURE 3.2.11-24 FLOWCHART - BCSP33 (Continued)

3.2.11.4.1.20 Submodule XX - BCSCAL

3.2.11.4.1.20.1 Description

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by BCSP31, BCSP32, BCSP33 to do bias calculations.
- c. Constraints and Limitations - Disk file HCB and HCC must be accessible. Local common must contain offset data information.
- d. Processing -
 1. Get old biases from disk file HCB.
 2. Get HC coordinates from disk file HCC.
 3. Get BCS target center coordinates.
 4. Perform least squares bias calculations.

3.2.11.4.1.20.2 Data, Logic, and Command Paths

a. Input Data

1. Local Common

BCSCM1 - T
BCSCM4 - HCN, NP
BCSCM5 - HFCHC

2. Global Common

COMIN4 - BCSTGG

b. Output Data

1. Local Common

BCSCM1 - M
BCSCM4 - AZOLD, ELOLD, AZO, ELO, XT, YT, ZT,
AZNEW, ELNEW, AZBD, ELBD, AZN, ELN
BCSCM5 - REP

c. Calls

1. BCS Submodules

BCSBIA
BCSCOR
BCSLSQ

3.2.11.4.1.20.3 Flowchart - See Figure 3.2.11-25.

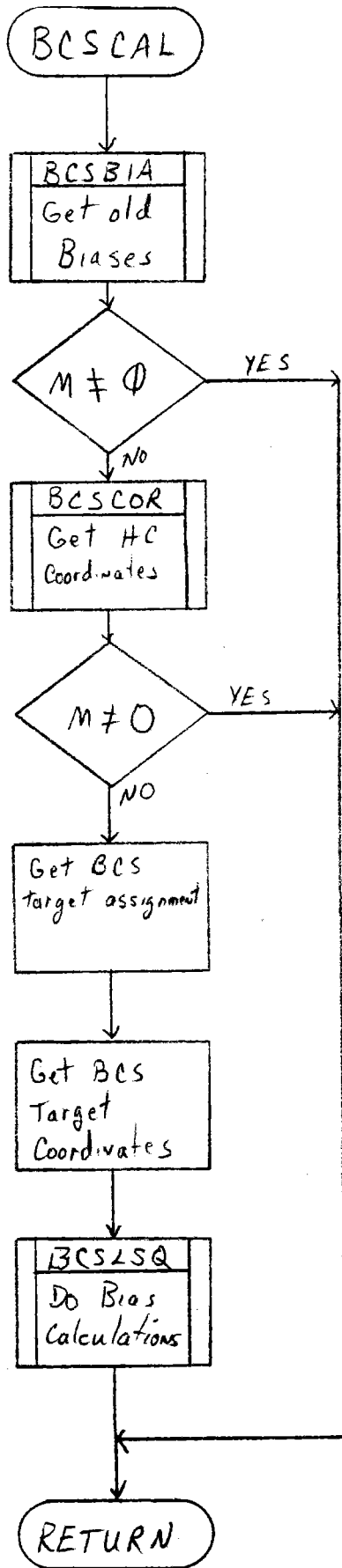


FIGURE 3.2.11-25 FLOWCHART - BCS CAL

3.2.11.4.1.21 Submodule XXI - BCSLOG

3.2.11.4.1.21.1 Description

- a. Language Used - FORTRAN IV.
- b. How invoked - Called by BCSP31, BCSP32, and BCSP33 to log results of a bias calculation on the logger.
- c. Constraints and Limitations - Up to three data sets can be logged. The data must be in local common before it is called.
- d. Processing - A header of information including the MDAC heliostat number is written to the logger. As many data points as available (up to three) are written to the logger. Old and new biases are formatted and written to the logger.

3.2.11.4.1.21.2 Data, Logic, and Command Paths

a. Input Data

1. Local Common

BCSCM3 - CAF12, PAF 12
BCSCM4 - NP, U. V, MM, DD, YY, HO, MI, SE
 AZO, ELO, AZN, ELN
BCSCM5 - MDMCNO

2. Global Common

COMIS5 - GTIMEG

b. Output Data

1. Other - Formatted logger output

HH:MM:SS BIAS CALCULATION RESULTS FOR HELIOSTAT MMMM:

DATA SET	DATE	TIME	HOR OFFS.	VERT. OFFS (FEET)	ALARMS	
					CENTROID	POWER
1	MO/DD/YY	HH:MM:SS	+ XX.XX	+ XX.XX	N	N
2	MO/DD/YY	HH:MM:SS	+ XX.XX	+ XX.XX	N	N
3	MO/DD/YY	HH:MM:SS	+ XX.XX	+ XX.XX	N	N

OLD BIASES: AZ = + XXX.XX EL = + XXX.XX (DEGREES)
NEW BIASES: AZ = + XXX.XX EL = + XXX.XX (DEGREES)

3.2.11.4.1.21.3 Flowchart - See Figure 3.2.11-26.

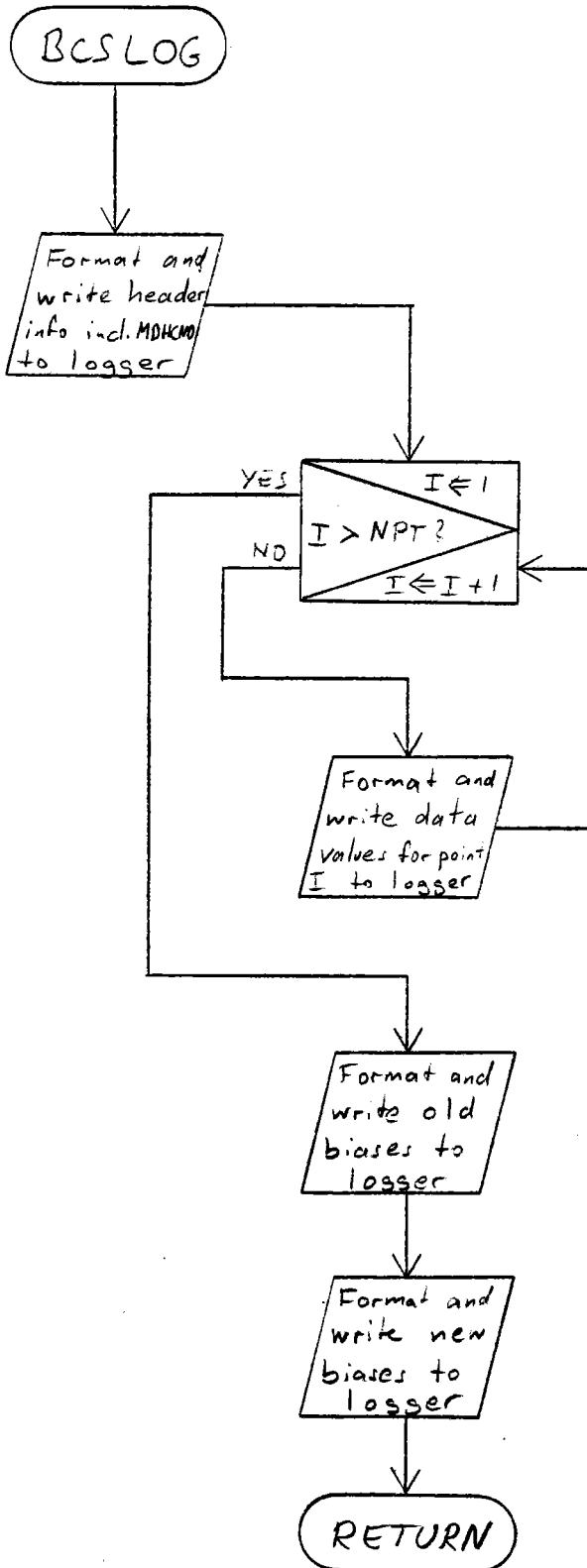


Figure 3.2.11-26 Flowchart - BCSLOG

3.2.11.4.1.22 Submodule XXII - BCSBIA

3.2.11.4.1.22.1 Description

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by BCSCAL, BCSP31, BCSP32, or BCSP33 to obtain or replace HC biases.
- c. Constraints and Limitations - Direct disk access to partition HCB only.
- d. Processing - BCSBIA calculates the appropriate disk record number and reads the complete record into a holding buffer. Read errors and EOFs are reported to the operator. Next a pointer into the holding buffer is calculated. If a replacement was not desired, AZIM and ELEV are loaded from the buffer. Otherwise, they are loaded into the buffer, and the record is written back to disk.

3.2.11.4.1.22.2 Data, Logic, and Command Paths

a. Input Data

1. Local Common

BCSCM5 - HFCHC - HFC/HC number
REP - Replacement flag
(If set, AZIM and ELEV are bias inputs)
HCBPTR

2. Other - HCB partition direct read.

b. Output Data

1. Local Common

BCSCM1 - M
BCSCM5 - AZIM, ELEV (if REP = .False.)

2. Other - HCB partition direct write.

3.2.11.4.1.22.3 Flowchart - See Figure 3.2.11-27.

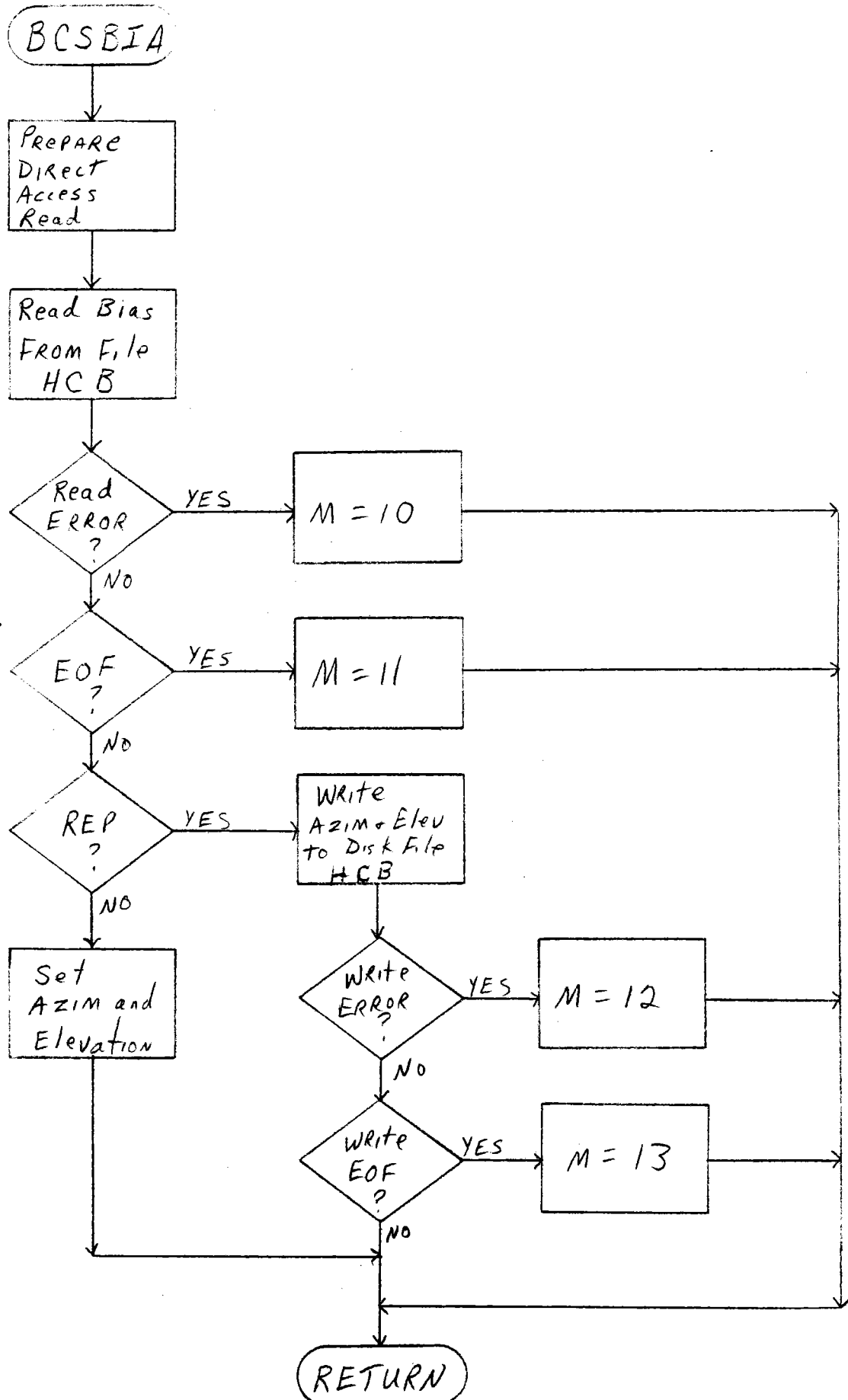


FIGURE 3.2.11-27 FLOWCHART - BCSBIA

3.2.11.4.1.23 Submodule XXIII - BCSCOR

3.2.11.4.1.23.1 Description

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by BCSCAL to obtain HC coordinates.
- c. Constraints and Limitations - BCSCOR can only read partition HCC and can only interpret 24 bit coordinates at binary point 14.
- d. Processing - BCSCOR calculates the appropriate record to be read and sets the partition pointer. Then it does a direct read of that record into a holding buffer. Read errors and EOFs are reported as noncritical alarms.

Next, BCSCOR calculates a pointer within the buffer and converts packed 24 bit data at binary point 14 into X, Y, and Z coordinates.

3.2.11.4.1.23.2 Data, Logic, and Command Paths

a. Input Data

1. Local Common

BCSCM5 - HFCHC, HCCPTR

2. Other

HCC partition direct read

b. Output Data

1. Local Common

BCSCM1 - M

BCSCM4 - XH, YH, ZH

3.2.11.4.1.23.3 Flowchart - See Figure 3.2.11-28.

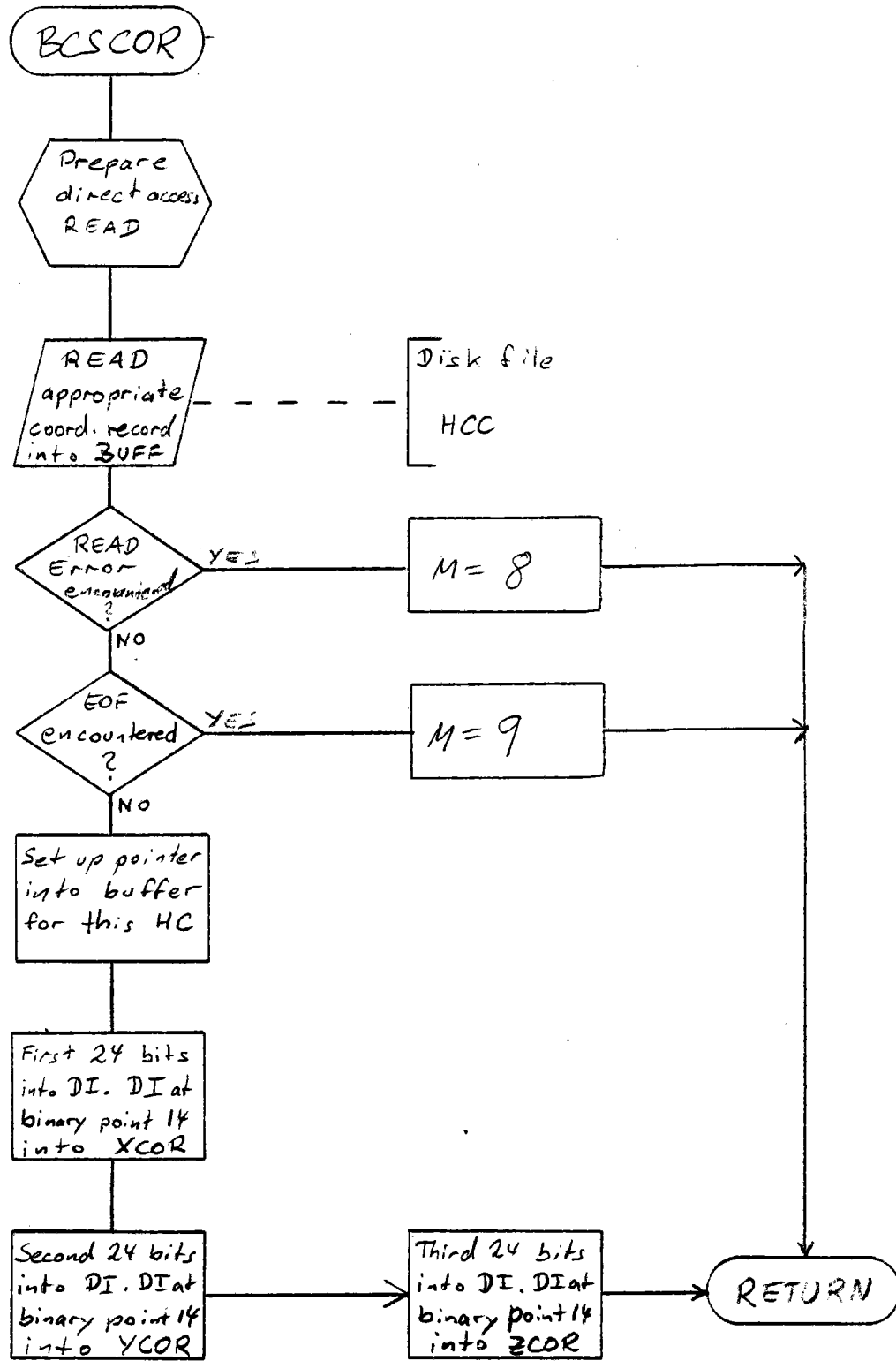


Figure 3.2.11-28 Flowchart - BCSCOR
759

3.2.11.4.1.24 Submodule XXIV - BCSSLQ

3.2.11.4.1.24.1 Description

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by BCSCAL to calculate new biases.
- c. Constraints and Limitations - BCSSLQ is limited to handle 1 - 3 sets of data points. It iterates up to 20 times to find a least sequence linear solution.
- d. Processing - A loop is entered and executed for each set of data points. In this loop the sun vector, heliostat gimbal angles, and heliostat normal vector are determined for a given data point. The resultant data is passed through a least squares coefficient algorithm and a linear equation solver until a solution is found or the maximum number of iterations is exceeded. In the latter case, a noncritical alarm is issued.

3.2.11.4.1.24.2 Data, Logic, and Command Paths

a. Input Data

1. Parameters

N - Number of data points
NT - Target number

2. Global Common ---

COMIN4 - SLATG, SLONG6

b. Output Data

1. Local Common

BCSCM1 - M
BCSCM4 - NT, XT, YT, ZT, AZO, ELO

c. Calls

1. BCS Submodules

BCSSUN
BCSGIM
BCSNOR
BCSLSC
BCSLIN

3.2.11.4.1.24.3 Internal Data Description

INTERNO - Number of iterations
REFAZ - Reference azimuth delta
REFEL - Reference elevation delta
Miscellaneous pointers and intermediate variables

3.2.11.4.1.24.4 Flowchart - See Figure 3.2.11-29 for the BCSSLQ flowchart.

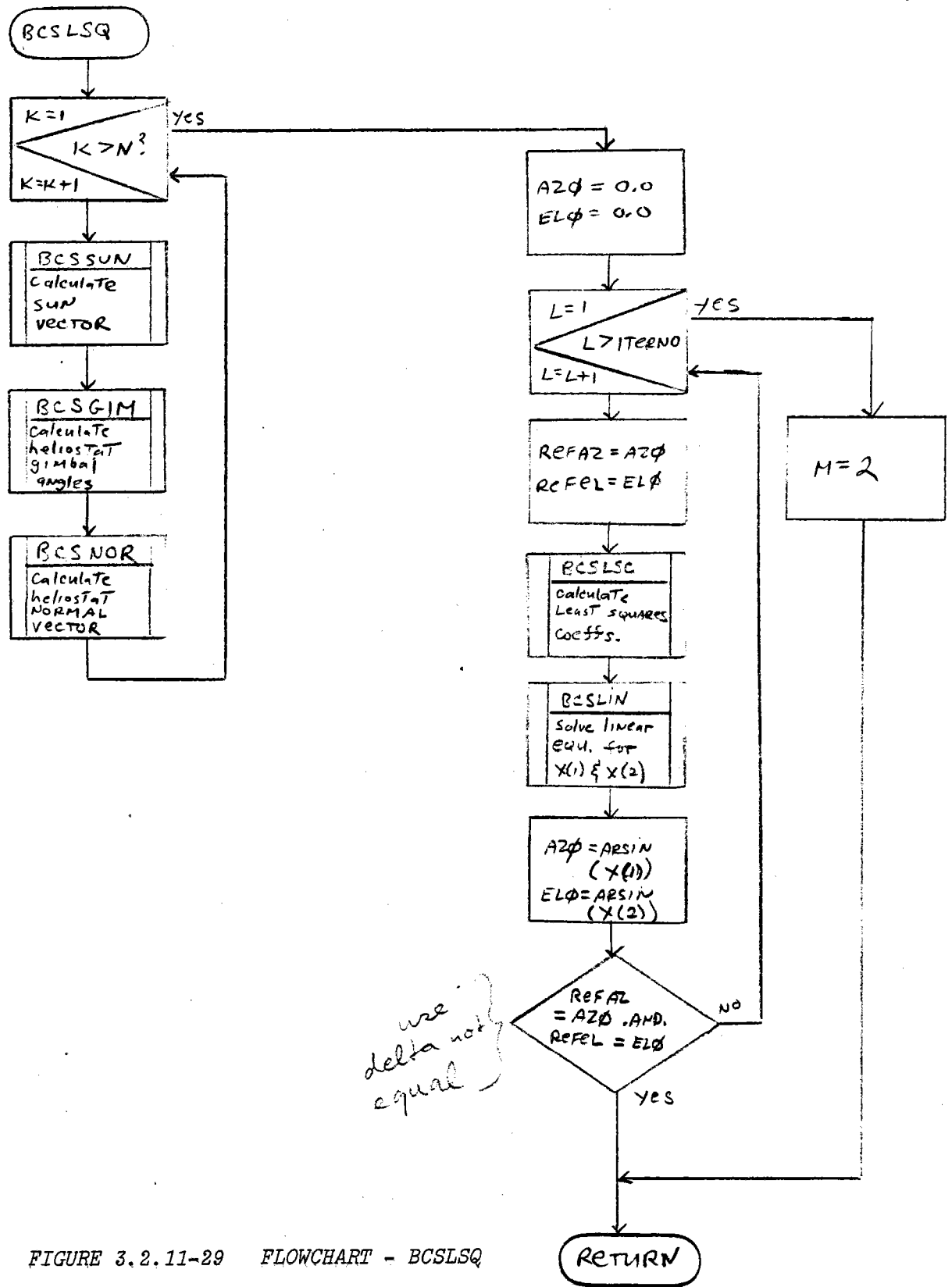


FIGURE 3.2.11-29 FLOWCHART - BCSLSQ

3.2.11.4.1.25 Submodule XXV - BCSSUN

3.2.11.4.1.25.1 Description

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by BCSSLQ to obtain the sun unit vector for the current data point (date and time).
- c. Constraints and Limitations - Local longitude and latitude must be initialized. Up to three sets of data can be processed. Existing interface to SUNPOS subroutine in SUNVEC module must not be altered.
- d. Processing - BCSSUN first initializes local common constants. Then it computes geocentric latitude in radians and converts longitude to radians. Before calling SUNPOS, it converts values in local common (BCSCM4) to date and time for that call and executes SUNPOS. Finally, BCSSUN places the returned sun unit vector in local common.

3.2.11.4.1.25.2 Data, Logic, and Command Paths

a. Input Data

1. Parameters

I = Number of data point

2. Local Common

BCSCM4 - HOUR, DD, MM, YY

3. Global Common

COMIN4 - SLATG, SLONGG

b. Output Data

1. Parameters - None

2. Local Common

BCSCM4 - XS, YS, ZS - Sun vector for data point

3. Global Common - None.

c. Calls

SUNPOS subroutine (of SUNVEC module)

3.2.11.4.1.25.3 Flowchart - See Figure 3.2.11-30 for the BCSSUN Flowchart.

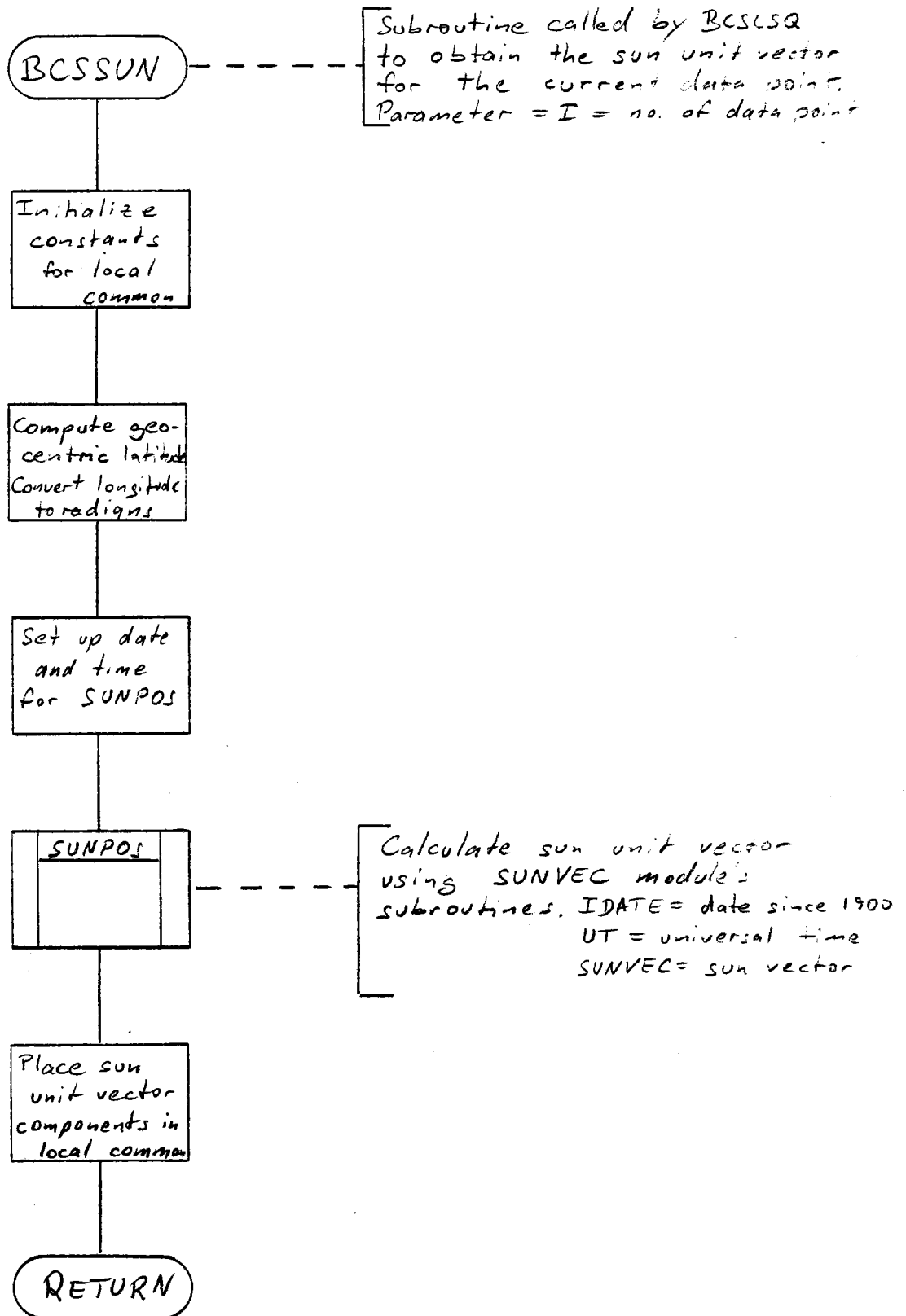


Figure 3.2.11-30 Flowchart - BCSSUN

3.2.11.4.1.26 Submodule XXVI - BCSGIM

3.2.11.4.1.26.1 Description

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by BCSLSQ to compute the commanded gimbal angles and heliostat normal for a particular set of data.
- c. Constraints and Limitations - Limited to three data sets and four targets.
- d. Processing - First the heliostat-to-target vector elements are calculated and normalized. Then these elements are added to the sun vector elements and normalized, resulting in the heliostat normal vector. From that the azimuth and elevation angles are calculated.

3.2.11.4.1.26.2 Data, Logic, and Command Paths

a. Input Data

1. Parameters

I = Number of data point (1-3)
NT = Number of BCS target (1-4)

2. Local Common

BCSCM4 - XT, YT, ZT - Target coordinates
 XH, YH, ZH - Heliostat coordinates
 XS, YS, ZS - Sun vector elements

3. Global Common - None.

b. Output Data

1. Parameters - None

2. Local Common

BCSCM4 - AZ, EL - Heliostat gimbal angles
 XN, YN, ZN - Heliostat normal vector

3. Global Common - None.

c. Calls

1. System Library Routines

ATAN2
ARSIN

3.2.11.4.1.26.3 Internal Data Description

- a. XTH, YTH, ZTH - Heliostat-to-target vector elements
XTN, YTN, ZTN - Normalized HC-to-target vector
XST, YST, ZST - Heliostat-to-sun vector
NORMT, NORMN - Normalizing of sums squared

3.2.11.4.1.26.4 Flowchart - See Figure 3.2.11-31 for BCSGIM flowchart.

3.2.11.4.1.27 Submodule XXVII - BCSNOR

3.2.11.4.1.27.1 Description

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by BCSLSQ to calculate the actual heliostat normal vector from given offset values.
- c. Constraints and Limitations - BCSNOR can handle three data points and four BCS targets.
- d. Processing - First BCSNOR calculates the target-to-heliostat vector. Then it converts target angles to radians and calculates horizontal and vertical offset unit vectors. Then it takes the actual offsets into account and calculates the target normal. With simple vector addition to the sun unit vector, the heliostat normal vector is determined.

3.2.11.4.1.27.2 Data, Logic, and Command Paths

a. Input Data

1. Parameters

I = Number of data point
NT = Number of BCS target

2. Local Common

BCSCM4 - XH, YH, ZH
 XT, YT, ZT, AT, BT
 U, V

3. Global Common - None.

b. Output Data

1. Parameters - None.

2. Local Common

BCSCM4 - XNA, YNA, ZNA

3. Global Common - None

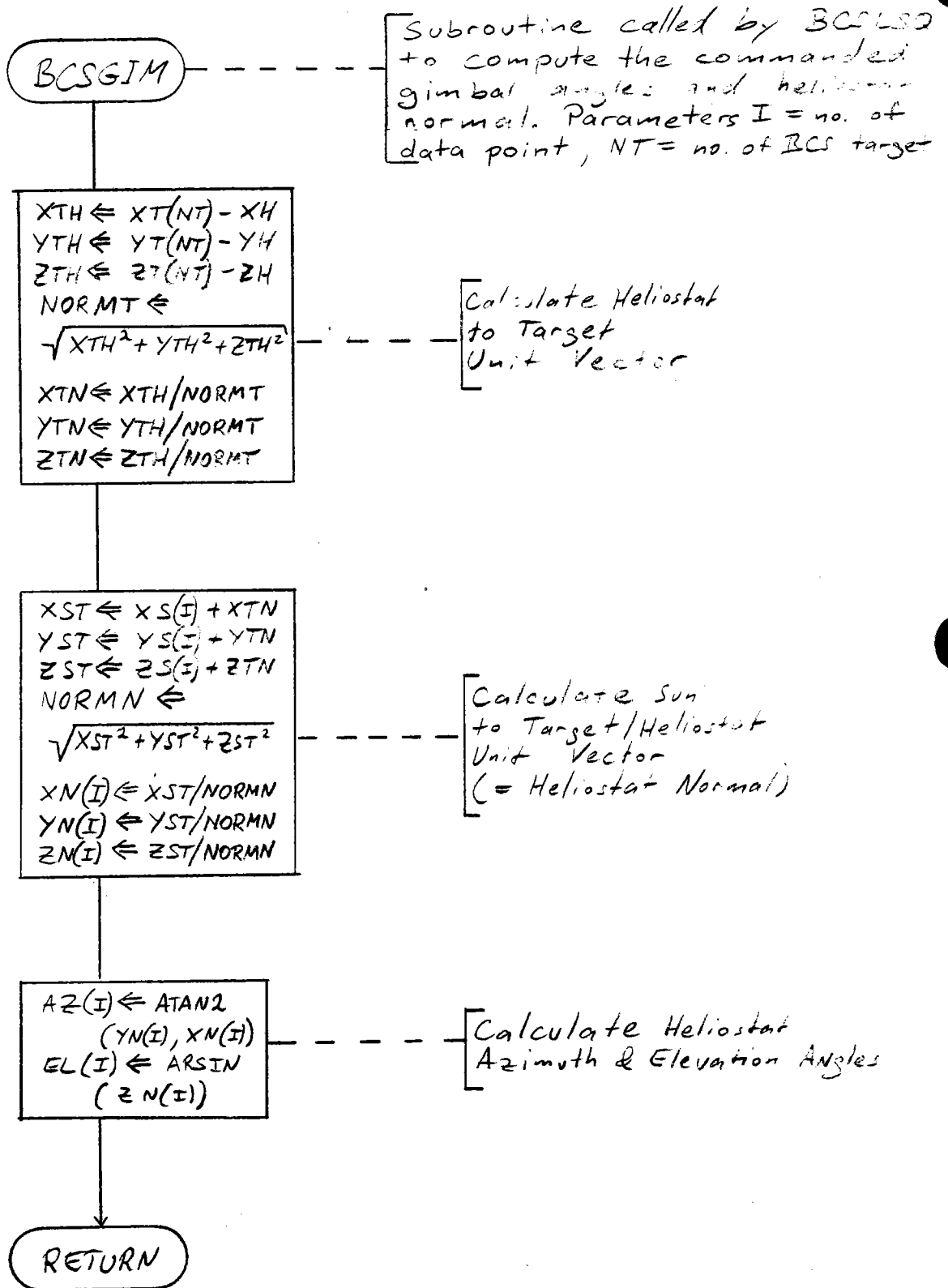


Figure 3.2.11-31 Flowchart - BCSGIM

c. Calls

1. System Library Routines

COS
SIN
SQRT

3.2.11.4.1.27.3 Flowchart - See Figure 3.2.11-32.

3.2.11.4.1.28 Submodule XXVIII - BCSLSC

3.2.11.4.1.28.1 Description

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by BCSLSQ to calculate the least squares linear system (normal equations) from measurement of the actual and commanded heliostat normal.
- c. Constraints and Limitations - Up to three data sets can be operated on.
- d. Processing - After initialization of the system of linear equation coefficients, BCSLSC enters a loop for the number of data points available. First BCSEPD is called to evaluate the partial derivatives; then BCSVEC calculates the vector components. Lambdas and deltas are added to find X, Y, and Z deltas. Then the B vector and A matrix are calculated, and the loop is repeated if more data points are available.

3.2.11.4.1.28.2 Data, Logic, and Command Paths

a. Input Data

1. Parameters

I = Number of data points

2. Local Common - None.

3. Global Common - None.

b. Output Data

1. Parameters - None.

2. Local Common

BCSCM4 - DELTAX, DELTAY, DELTAZ
A and B

3. Global Common - None.

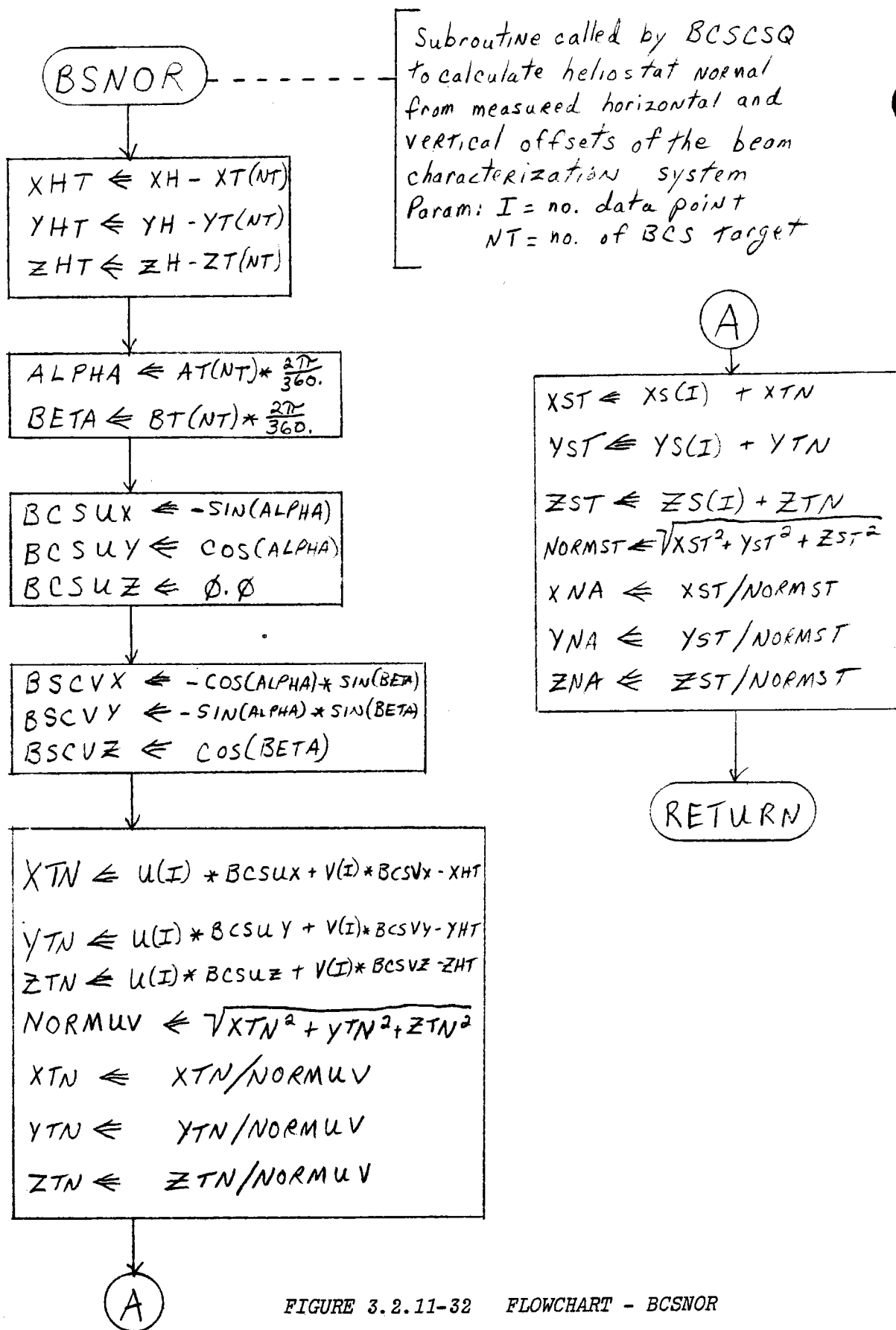


FIGURE 3.2.11-32 FLOWCHART - BCSNOR

c. Calls

1. BCS Submodules

BCSEPD
BCSVEC

3.2.11.4.1.28.3 Flowchart - See Figure 3.2.11-33 for the BCSLSC flowchart.

3.2.11.4.1.29 Submodule XXIX - BCSEPD

3.2.11.4.1.29.1 Description

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by BCSLSC to evaluate partial derivatives.
- c. Constraints and Limitations - BCSLSC can handle up to three data points and calculates a 3 x 2 matrix for each data point.
- d. Processing - Lines and cosines are calculated for heliostat azimuth and elevation. From that, the six elements of the 3 x 2 Jacobian matrix are calculated.

3.2.11.4.1.29.2 Data, Logic, and Command Paths

a. Input Data

1. Parameters

I = Number of data point

2. Local Common

BCSCM4 - AZ, EL, AZO, ELO

3. Global Common - None.

b. Output Data

1. Parameters

PP = Jacobian 3 x 2 matrix.

2. Local Common - None.

3. Global Common - None.

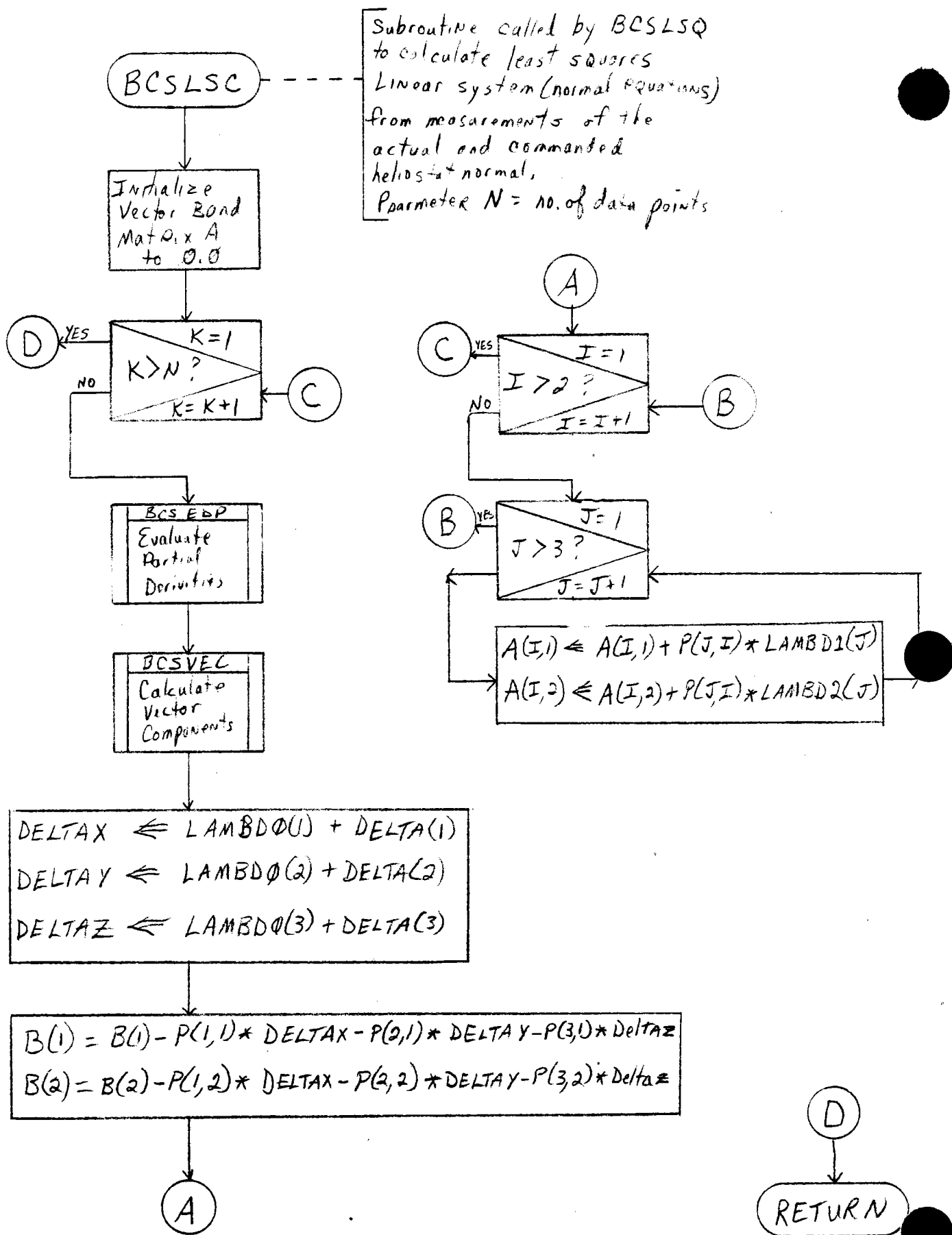


FIGURE 3.2.11-33 FLOWCHART - BCSLSC

c. Calls

1. System Library Routines

SIN
COS

3.2.11.4.1.29.3 Flowchart - See Figure 3.2.11-34 for the BCSEPD flowchart.

3.2.11.4.1.30 Submodule XXX - BCSVEC

3.2.11.4.1.30.1 Description

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by BCSLSC to calculate individual vector terms in encoder bias vector.
- c. Constraints and Limitations - BCSVEC can handle up to three data points.
- d. Processing - First the sines and cosines of the heliostat gimbal angles are calculated, then the 3 x 3 set of Lambdas. Finally the delta vector is initialized to zero.

3.2.11.4.1.30.2 Data, Logic, and Command Paths

a. Input Data

1. Parameters

I = Number of data point

2. Local Common

BCSCM4 - AZ, EL - Heliostat gimbal angles
XNA, YNA, ZNA - Heliostat actual normal vector.

3. Global Common - None.

b. Output Data

1. Parameters - None.

2. Local Common

BCSCM4 - LAMBDO, LAMBD1, LAMBD2
DELTA

3. Global Common - None.

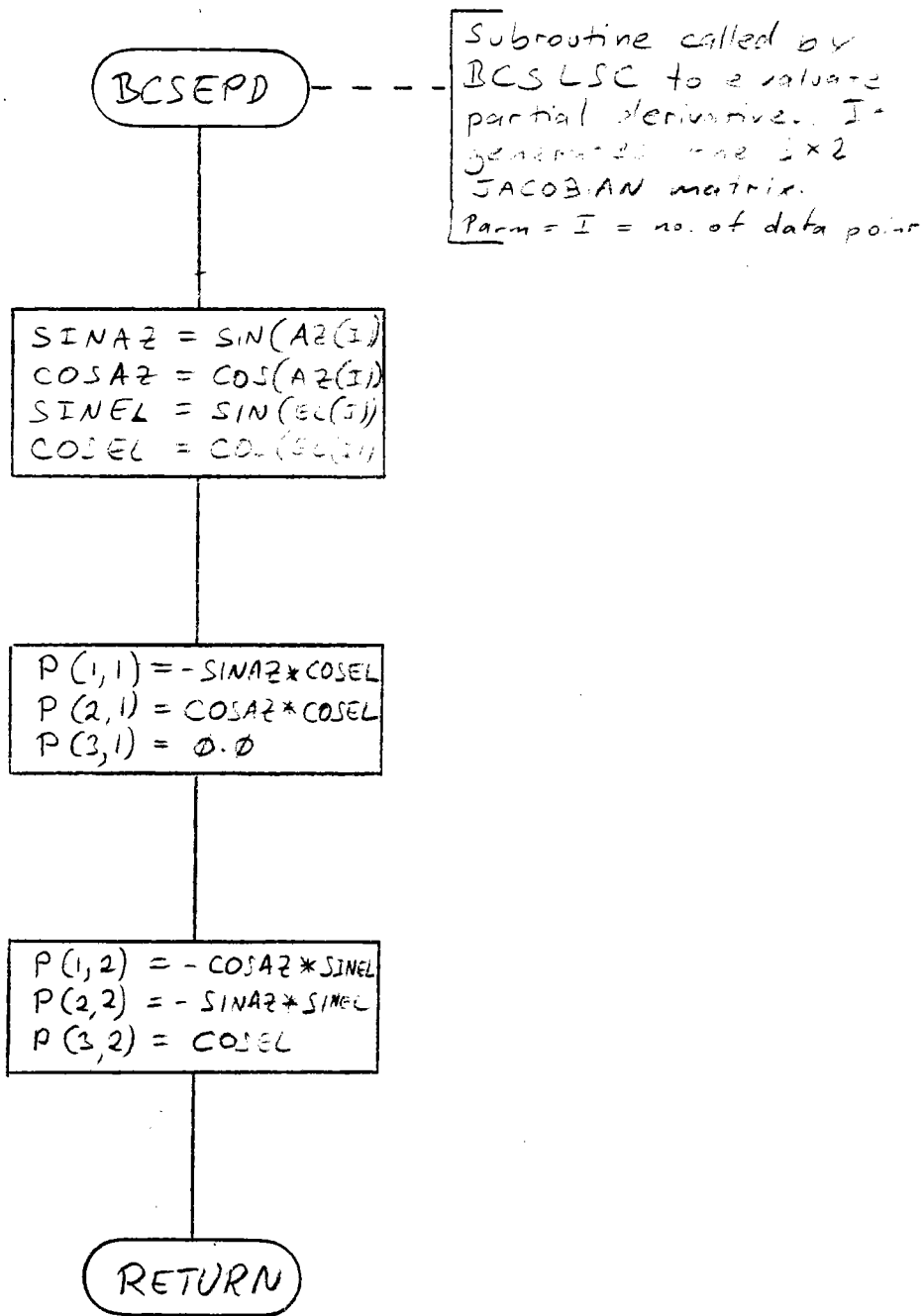


Figure 3.2.11- 34 Flowchart - BCSEPD

c. Calls

1. System Library Routines

COS
SIN

3.2.11.4.1.30.3 Flowchart - See Figure 3.2.11-35 for the BCSVEC flowchart.

3.2.11.4.1.31 Submodule XXXI - BCSLIN

3.2.11.4.1.31.1 Description

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by BCSSLQ to solve the system of linear equations $AX = B$ for X.
- c. Constraints and Limitations - A is a 2 x 2 matrix; X and B are two-dimensional vectors.
- d. Processing - The three determinants of the two equations are calculated, and X(1) and X(2) are the quotients of these determinants.

3.2.11.4.1.31.2 Data, Logic, and Command Paths

a. Input Data

- 1. Parameters - None.
- 2. Local Common
BCSCM4 - A, B
- 3. Global Common - None.

b. Output Data

- 1. Parameters - None.
X - Vector
- 2. Local Common
BCSCM4 - X
- 3. Global Common - None.

c. Calls - None.

BCSVEC

Subroutine called by BCSLSC
to calculate individual
vector terms in encoder
bias vector. Parm. = I =
No. of current data point

$\text{COSAZ} = \text{COS}(\text{AZ}(I))$
 $\text{SINAZ} = \text{SIN}(\text{AZ}(I))$
 $\text{COSAZ}\phi = \text{COS}(\text{AZ}\phi)$
 $\text{SINAZ}\phi = \text{SIN}(\text{AZ}\phi)$
 $\text{COSEL} = \text{COS}(\text{EL}(I))$
 $\text{SINEL} = \text{SIN}(\text{EL}(I))$
 $\text{COSEL}\phi = \text{COS}(\text{EL}\phi)$
 $\text{SINEL}\phi = \text{SIN}(\text{EL}\phi)$

$\text{LAMBDA}\phi(1) = \text{COSAZ} * \text{COJAZ}\phi * \text{COSEL} * \text{COSEL}\phi - \text{XNA}(I)$
 $\text{LAMBDA}\phi(2) = \text{SINAZ} * \text{COSAZ}\phi * \text{COSEL} * \text{COSEL}\phi - \text{YNA}(I)$
 $\text{LAMBDA}\phi(3) = \text{SINEL} * \text{COSEL}\phi - \text{ZNA}(I)$

$\text{LAMBDA}1(1) = -\text{SINAZ} * \text{COSEL} * \text{COSEL}\phi$
 $\text{LAMBDA}1(2) = \text{COSAZ} * \text{COSEL} * \text{COSEL}\phi$
 $\text{LAMBDA}1(3) = \phi.\phi$

$\text{LAMBDA}2(1) = -\text{COSAZ} * \text{COSAZ}\phi * \text{SINEL}$
 $\text{LAMBDA}2(2) = -\text{SINAZ} * \text{COSAZ}\phi * \text{SINEL}$
 $\text{LAMBDA}2(3) = \text{COSEL}$

$\text{DELTA}(1) = \text{SINAZ} * \text{SINAZ}\phi * \text{SINEL} * \text{SINEL}\phi$
 $\text{DELTA}(2) = -\text{COSAZ} * \text{SINAZ}\phi * \text{SINEL} * \text{SINEL}\phi$
 $\text{DELTA}(3) = \phi.\phi$

RETURN

Figure 3.2.11-35 Flowchart - BCSVEC

3.2.11.4.1.31.3 Internal Data Description

DETO - Determinant of Matrix A
DET1 - First solution determinant
DET2 - Second solution determinant

3.2.11.4.1.31.4 Flowchart - See Figure 3.2.11-36 for the BCSLIN flowchart.

3.2.11.4.1.32 Submodule XXXII - BCSMPA

3.2.11.4.1.32.1 Description

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by BCSMEA when Phase A measurement processing is needed.
- c. Constraints and Limitations - Should only be executed after a valid transition to a new MSTATE has occurred.
- d. Processing - Processing for MSTATES 1 through 5 is done here and described below.
 1. MSTATE 1 - Get next good HC from file for target T and send BCS=3 message to OCS.
 2. MSTATE 2 - Clear time for BCS-4, clear BCS busy bits. If success flag is 2, display alarm for bad message; otherwise, display alarm for M = 118 and get next good HC from file.
 3. MSTATE 3 -
 - (a) Process BCS-4 message from OCS and command HC to track BCS target.
 - (b) Clear stagger timer and command next B/S HC to go to STOWALT2.
 - (c) If more B/S HC need to be moved, set STS9F = .TRUE. Otherwise, set STS9F = TRUE. and MSTATE = 6.
 4. MSTATE 4 - Set 2 SEC STAGGER TIMER.
 5. MSTATE 5 - Go to Step 3.(b) and proceed.

3.2.11.4.1.32.2 Data, Logic, and Command Paths

a. Input Data

1. Local Common

BCSCM1 - MSTATE, T, POINT, WRKARR, WRK, BOSF
BCSCM3 - MSF8, MHC, MT
BCSCM6 - MTIMER

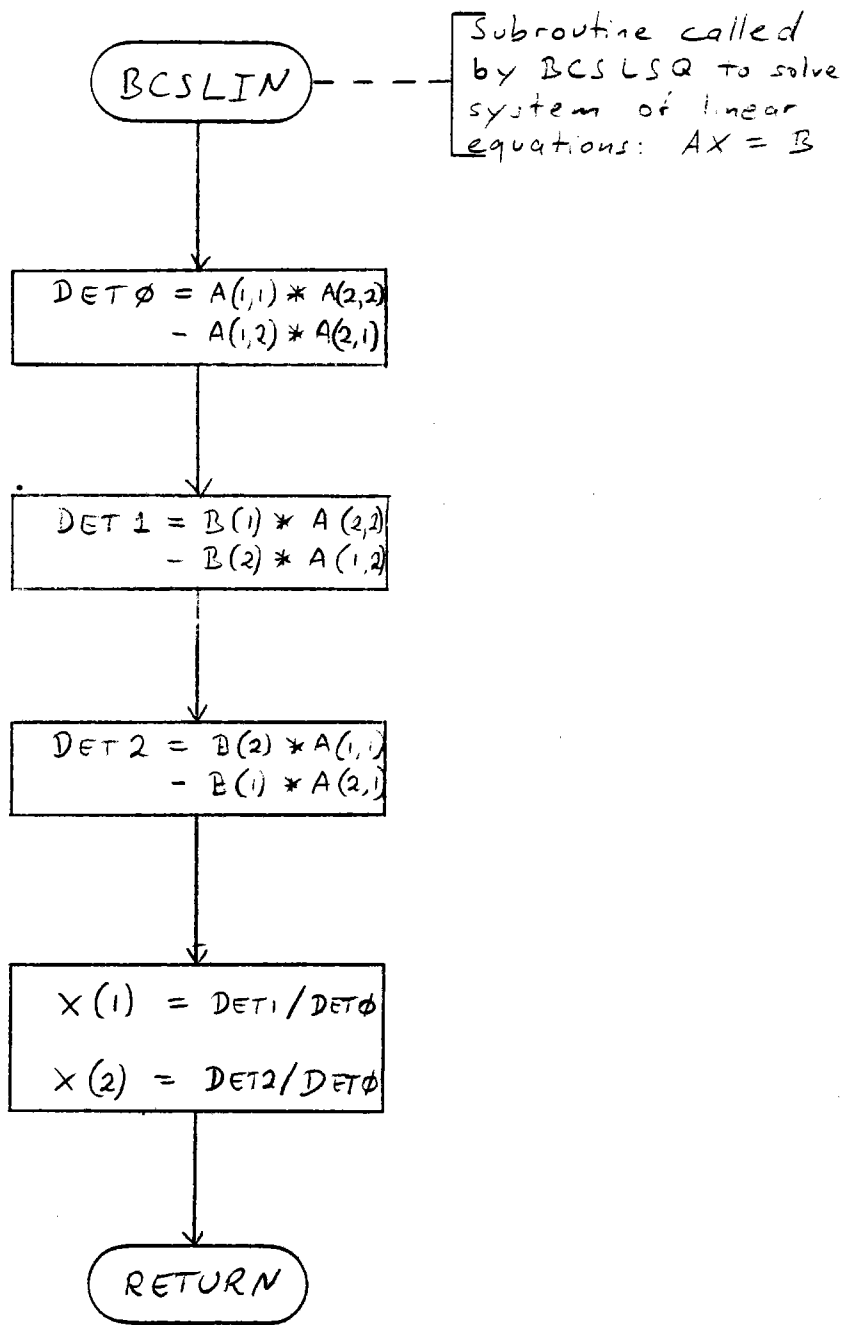


Figure 3.2.11-36 Flowchart - BCSLIN

2. Global Common
COMIN2 - HCDATG
COMIN4 - BCSTGG

b. Output Data

1. Local Common

BCSCM1 - M, HSTAT, BHC, STS7F, MSTATE, BEFOR, STS9F
BCSCM5 - PARM, NPARM, MDHCNO
BCSCM6 - MTIMER, DS1F

2. Global Common

COM1S2 - HCDATG, HCCMDG

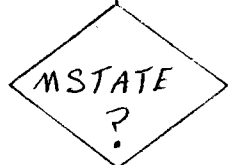
c. Calls

1. BCS Submodules

BCSGET
BCSCLR
BCSMES
BCSBOS

3.2.11.4.1.32.3 Flowchart - See Figure 3.2.11-37.

BCSMPA



RETURN

FIGURE 3.2.11-37 FLOWCHART - BCSMPA

3.2.11.4.1.33 Submodule XXXIII - BCSBOS

3.2.11.4.1.33.1 Description

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by BCSMPA for MSTATE = 3 or 5.
- c. Constraints and Limitations - Should only be called when processing is ready for a B/S HC to be commanded.
- d. Processing -
 1. Get next B/S HC; if none, set BOSF = .TRUE. and return.
 2. If B/S HC is in STOW or STOWALT2, set HSTAT = 9 and go to 1.
 3. Command B/S HC to STOWALT2 and set HSTAT = 8.
 4. Look ahead to next B/S HC; if none, set BOSF = .TRUE. and Return.
 5. If B/S HC in STOW or STOWALT2, set HSTAT = 9 and go to 4.
 6. Set BOSF = .FALSE., set STAGGER TIMER, and return.

3.2.11.4.1.33.2 Data, Logic, and Command Paths

a. Input Data

1. Local Common

BCSCM1 - BHC, T, POINT, WRK

2. Global Common

COM6S2 - HCST1G

b. Output Data

1. Local Common

BCSCM1 - HSTAT, BHC, BOSF
BCSCM6 - MTIMER

c. Calls

1. BCS Submodules

BCSAL2

3.2.11.4.1.33.3 Flowchart - See Figure 3.2.11-38.

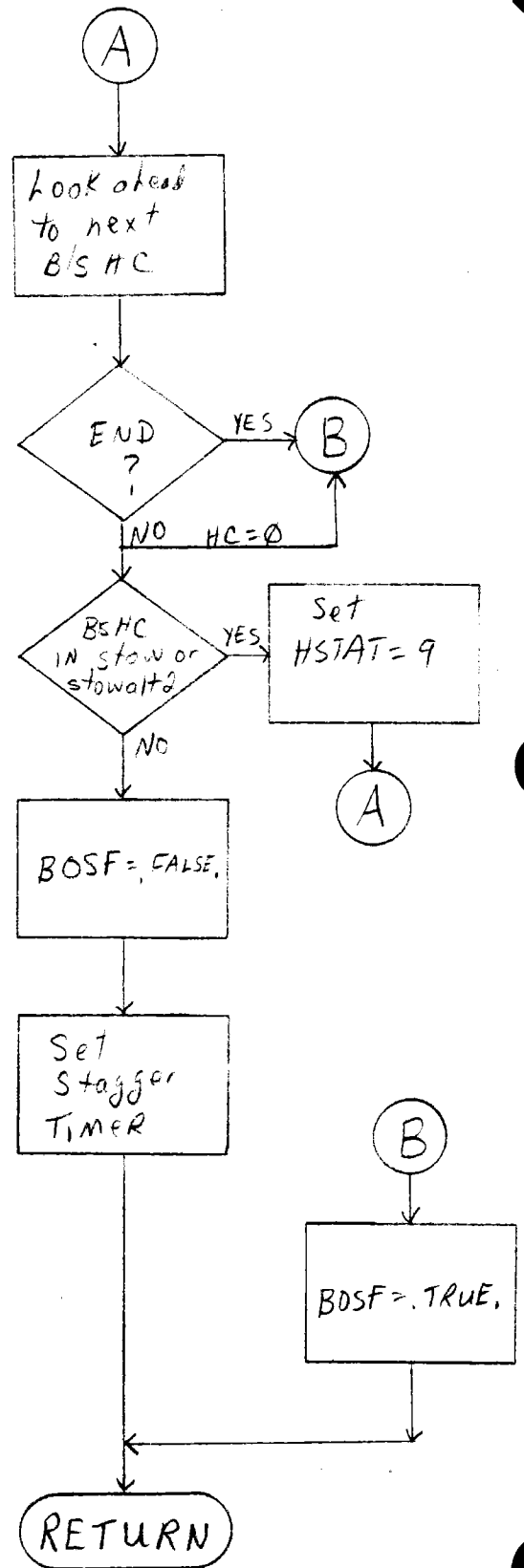
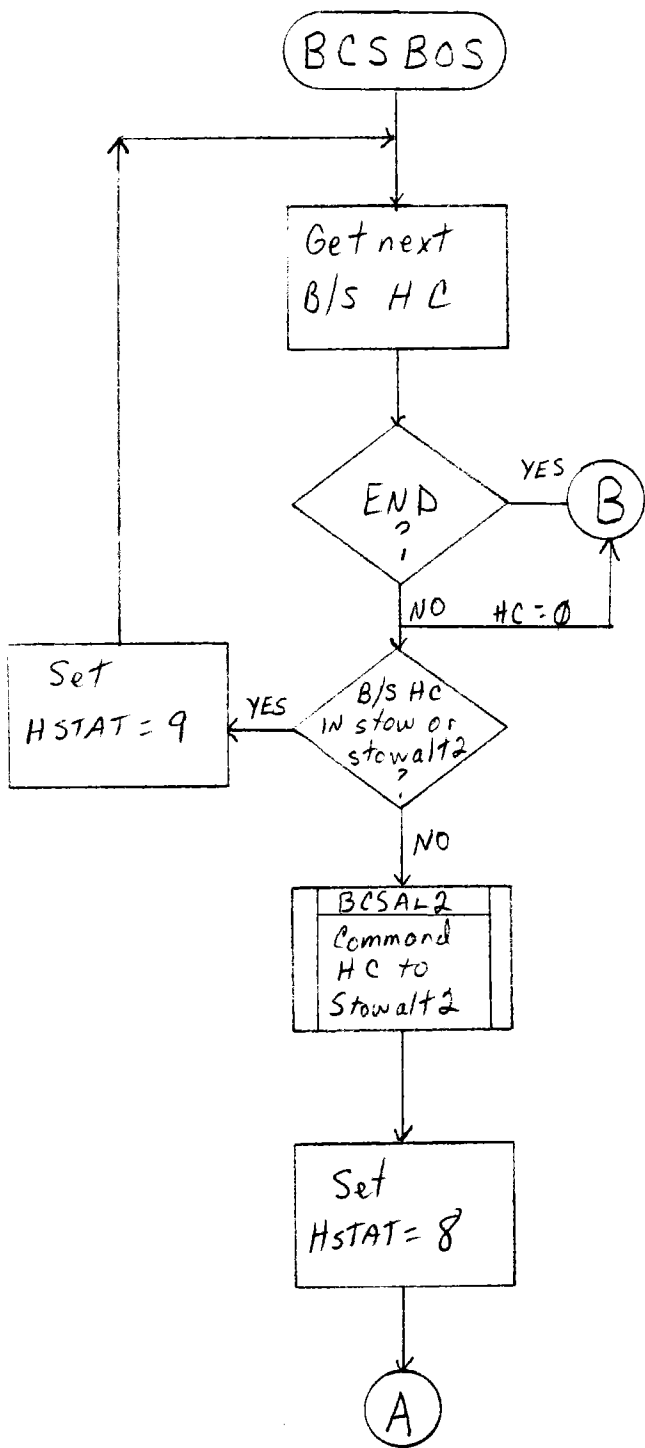


FIGURE 3.2.11-38 FLOWCHART - BCSBOS

3.2.11.4.1.34 Submodule XXXIV - BCSAL2

3.2.11.4.1.34.1 Description

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by BCSBOS to command HC to STOWALT2.
- c. Constraints and Limitations - BCSAL2 can only read disk partition AL2 for one set of values at one time.
- d. Processing - First BCSAL2 calculates which record to read and reads that record into a holding buffer. Any cursors of EOFs are reported as a noncritical alarm. Then BCSAL2 sets up a buffer pointer and finally commands HC to STOWALT2.

3.2.11.4.1.34.2 Data, Logic, and Command Paths

a. Input Data

1. Parameters - None.

2. Local Common

BCSCM5 - HFCHC
BCSCM6 - IAVAl2

3. Global Common

COM1S1 - AZIMG

b. Output Data

1. Parameters - None

2. Local Common

BCSCM1 - M
BCSCM5 - PARM, NPARM
BCSCM6 - IAVAl2

3. Global Common

COM1S2 - HCDATG, HCCMDG

3.2.11.4.1.34.3 Flowchart - See Figure 3.2.11-39 for the BCSAL2 flowchart.

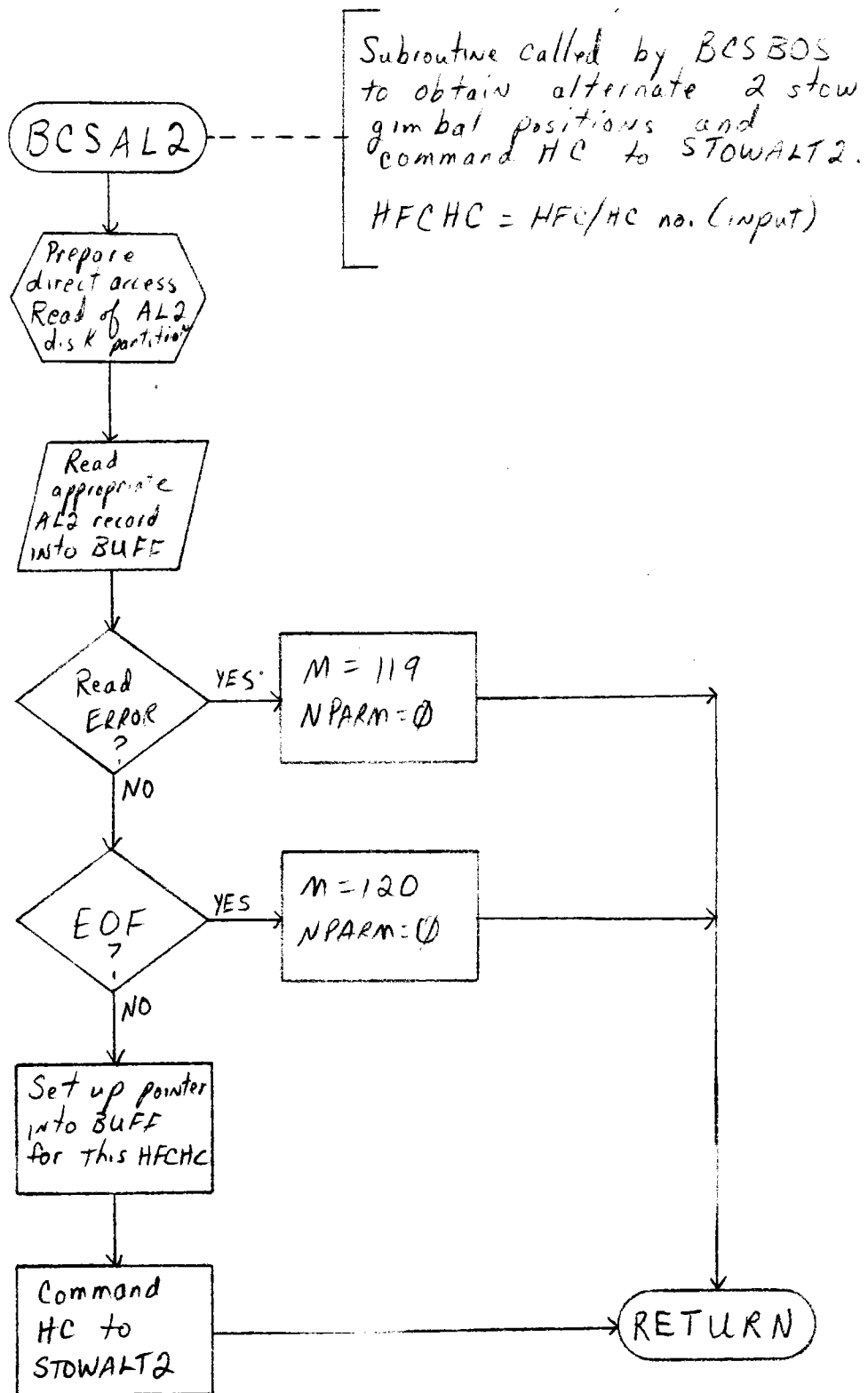


FIGURE 3.2,11-39 FLOWCHART - BCSAL2

3.2.11.4.1.35 Submodule XXXV - BCSMPB

3.2.11.4.1.35.1 Description

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by BCSMEA when Phase B processing is to be done.
- c. Processing - Processing for Phase B MSTATES through 11 are done here and described below.
 1. MSTATE 7 - Send BCS message to OCS and set timer.
 2. MSTATE 8 -
 - (a) Clear timer; if success flag is number 0, go to (c).
 - (b) Begin returning HCs to proper mode. If all HCs returned, then get next HC from file, set MSTATE = 1, and return.
 - (c) If success flag is 1, display alarm "Beam Off Target", and go to Step 2.(b).
 - (d) Otherwise, set error message and return.
 3. MSTATE 9 - Process BCS-6 information, command HC to standby, set HSTAT = 4, STS1OF = .TRUE., and clear BEFOR.
 4. MSTATE 10 - Send BCS-3 to OCS and set MTIMER.
 5. MSTATE 11 - Process BCS-4 and clear MTIMER, command HC to track BCS target, set HSTAT = 3, STSF = .TRUE., and clear BEFORE.

3.2.11.4.1.35.2 Data, Logic, and Command Paths

a. Input Data

1. Local Common

BCSCM1 - T, MSTATE, POINT, WRKARR, RTNF, WRK
BCSCM3 - MSF8, MHC, MT, CAF8, PAF8, IP8, U8, V8

2. Global Common

COMIN2 - HCDATG
COMIN4 - BCSTGG

b. Output Data

1. Local Common

BCSCM1 - M, BHC, MSTATE, MSTATE, BEFOR, STS10F, STS9F
BCSCM5 - PARM, NPARM, MDHCNO,
BCSCM6 - MTIMER, DS1F
BCSCM3 - MN

2. Global Common

COMLS2 - HDATG, HCCMDG

c. Calls

1. BCS Subroutines

BCSMES
BCSRTN

BCSSTB
BCSOUT

3.2.11.4.1.35.3 Flowchart - See Figure 3.2.11-40 for the BCSMPB flowchart.

3.2.11.4.1.36 Submodule XXXVI - BCSMPC

3.2.11.4.1.36.1 Description

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by BCSMEA when Phase C processing is to be done.
- c. Processing - Processing for MSTATES 12 through 14 are done here and described below.
 1. MSTATE 12 - Begin returning HCs to their proper modes; if more need to be returned, then return; otherwise set MSTATE - 1 and return.
 2. MSTATE 13 - Set two-second stagger timer.
 3. MSTATE 14 -
 - (a) Clear timer and MSTATE.
 - (b) If DEFOCUS, then command B/S HC to standby; otherwise command HC to original mode.
 - (c) If there is another B/S HC, then set STS7F = .TRUE.; otherwise clear BCS BUSY BITS, set HSTAT = 5, MSTATE = 1, and get next HC from file and return.

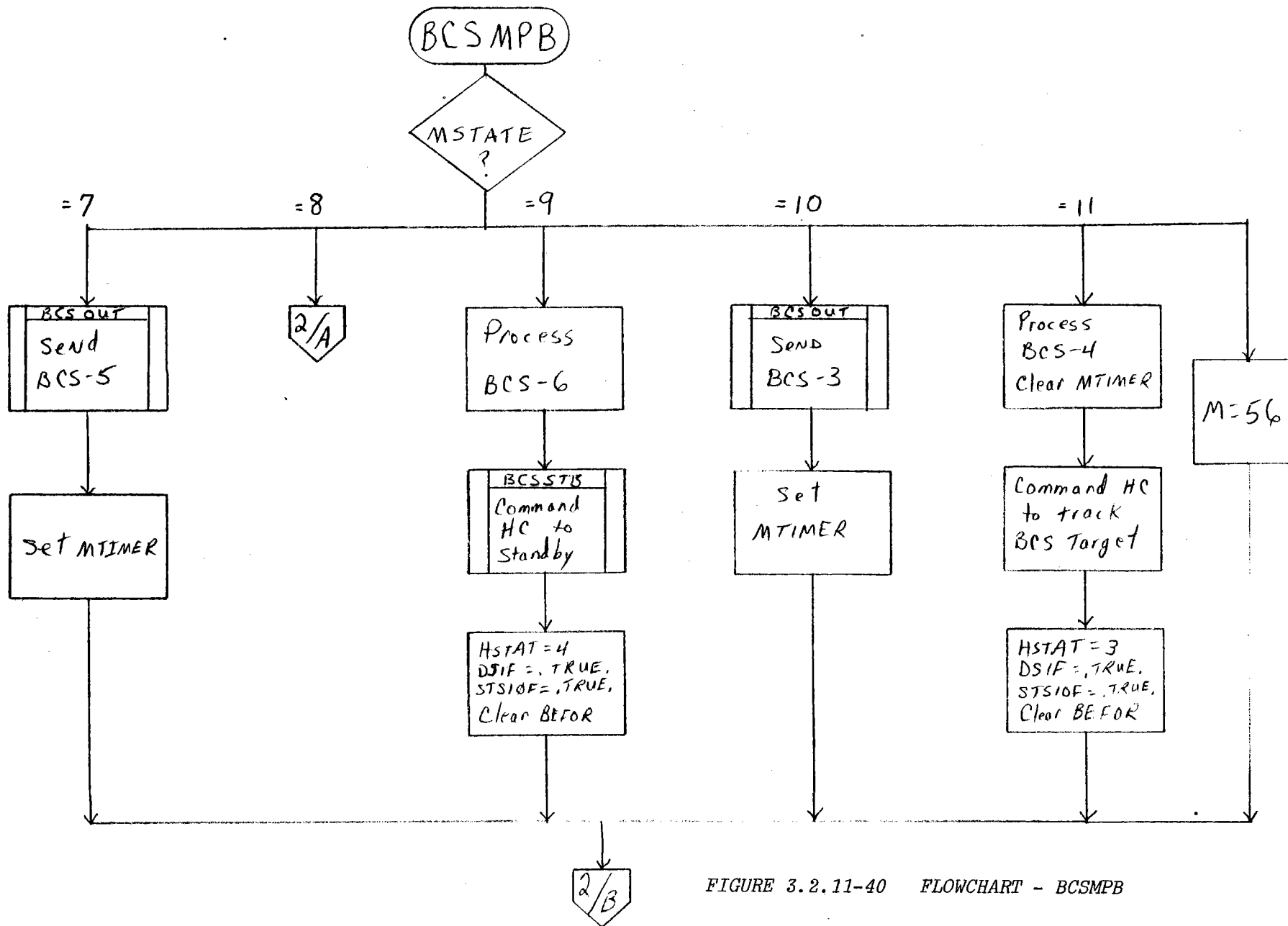


FIGURE 3.2.11-40 FLOWCHART - BCSMPB

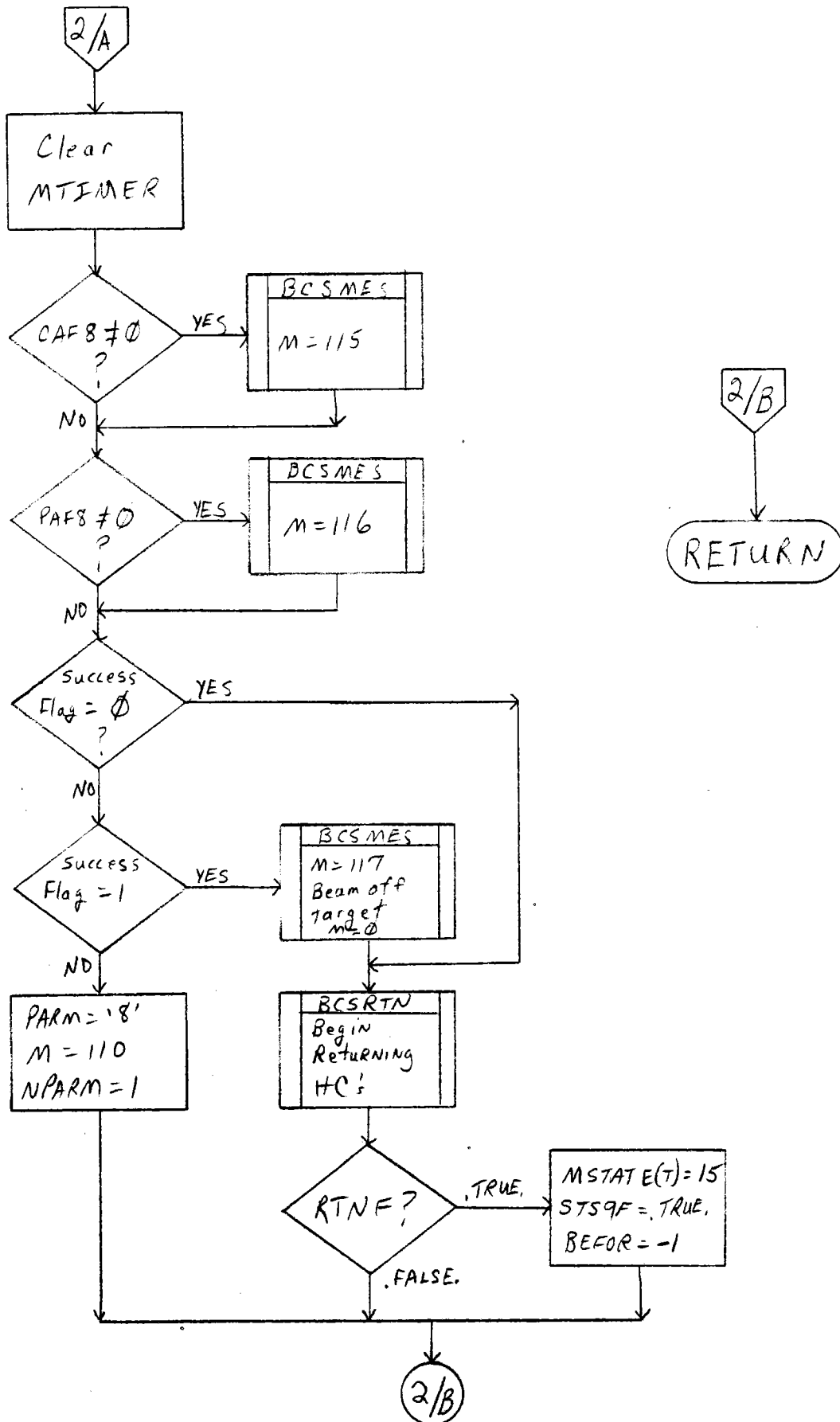


FIGURE 3.2.11-40 FLOWCHART - BCSMPB (Continued)
786

3.2.11.4.1.36.2 Data, Logic, and Command Paths

a. Input Data

1. Local Common

BCSCM1 - MSTATE, T, POINT, BHC, RTNF, WRK,
SAVARR, WRKARR, HSTAT

2. Global Common

COM1S2 - HCCMDG
COM1S5 - LOKDEG

b. Output Data

1. Local Common

BCSCM1 - M, BHC, MSTATE, HSTAT, STS7F, STS9F, BEFOR
BCSCM6 - MTIMER, DS1F
BCSCM5 - MDHCNO, PARM, NPARM

2. Global Common

COM1S2 - HCCMDG, HCDATG

c. Calls

1. BCS Submodules

BCSRTN BCSCLR
BCSSTB BCSMES

3.2.11.4.1.36.3 Flowchart - See Figure 3.2.11-41.

3.2.11.4.1.37 Submodule XXXVII - BCSGET

3.2.11.4.1.37.1 Description

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by BCSMPA when the next heliostat is desired from the file for a specific target.
- c. Constraints and Limitations - Should only be called when there are still more HCs in the file for that target.
- d. Processing -
 1. Get next HC from file; if EOF, set TSTAT = 2 and ENDF = .FALSE. and return.
 2. If HC is not in track or standby, display message and go to 1.

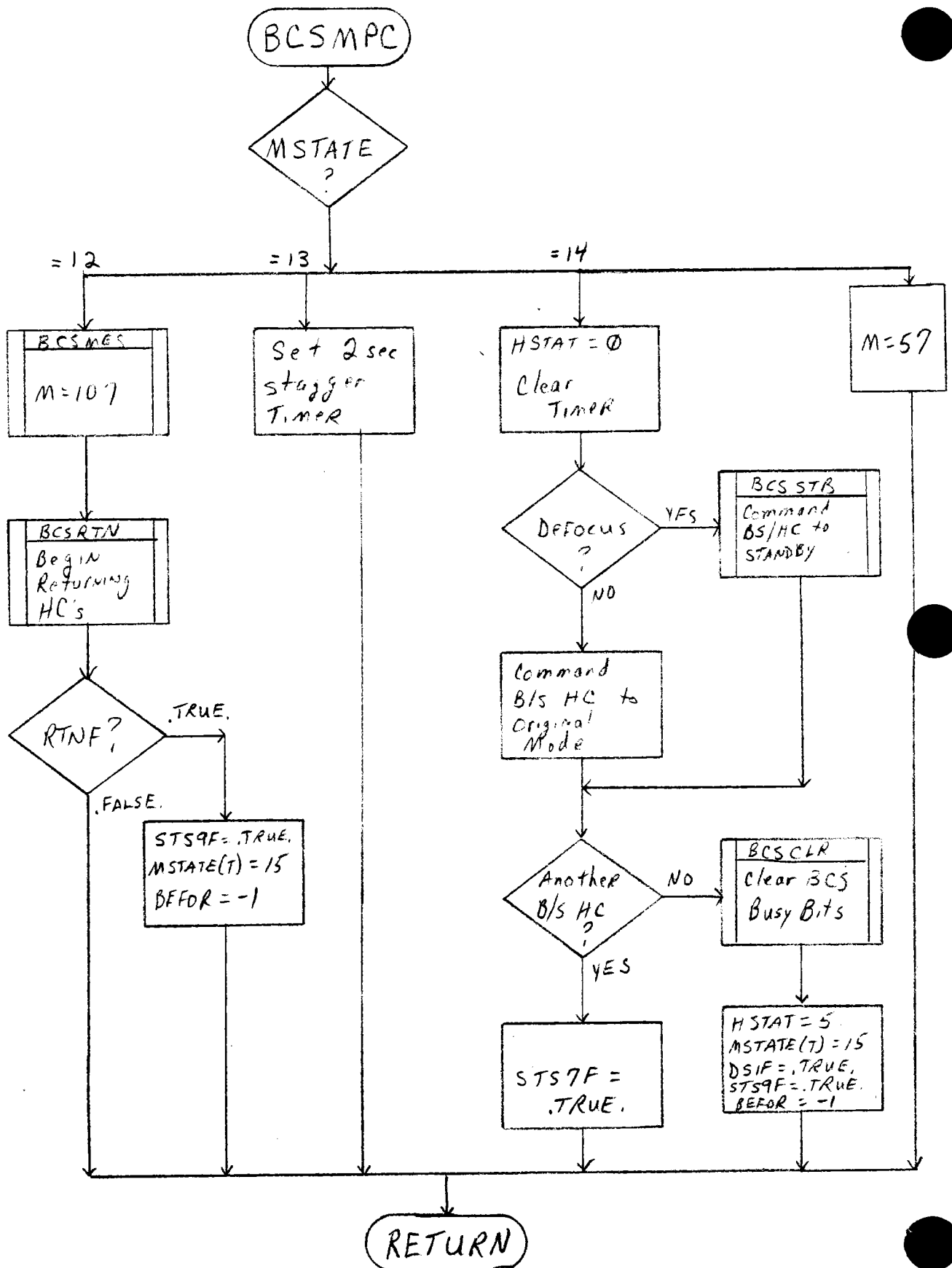


FIGURE 3.2.11-41 FLOWCHART - BCSMPC

3. Save current command for HC,
4. Get next B/C HC; if end, set BCS BUSY BITS for HCs, send BCS-3 to OCS, and return.
5. If HC in TRACK, STBY, STOW, or STOWALT2, save current command for HC and go to 4.
6. Display message and go to 1.

3.2.11.4.1.37.2 Data, Logic, and Command Paths

a. Input Data

1. Local Common

BCSCM1 - BHC, POINT, T, WRK, WRKARR
 BCSCM2 - R2

2. Global Common

COM8S3 - HCST3G
 COM8S2 - HCST1G
 COM1S2 - HCDATG, HCCMDG

b. Output Data

1. Local Common

BCSCM1 - BHC, SAVARR, M, HSTAT, ENDF
 BCSCM3 - TSTAT
 BCSCM6 - MTIMER

2. Global Common

COM8S3 - HCST3G

c. Calls

1. BCS Submodules

BCSMES
 BCSOUT

3.2.11.4.1.37.3 Flowchart - See Figure 3.2.11-42.

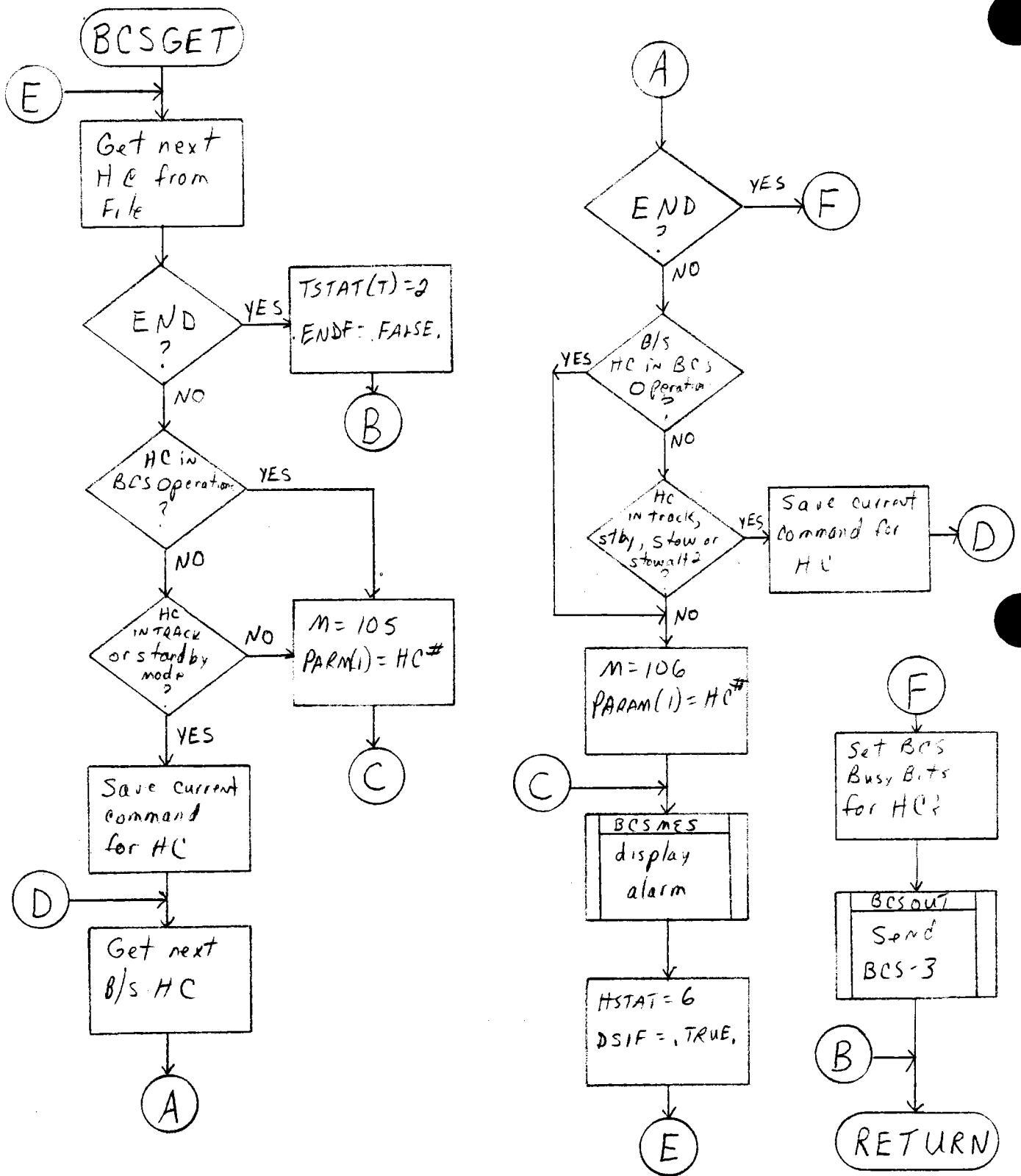


FIGURE 3.2.11-42 FLOWCHART - BCSGET

3.2.11.4.1.38 Submodule XXXVIII - BCSCLR

3.2.11.4.1.38.1 Description

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by BCSMPA, BCSMPC, and BCSMMC to clear BCS BUSY BIT for all HCs involved in a particular measurement.
- c. Constraints and Limitations - Should only be called when all HCs have been returned to their proper modes.
- d. Processing -
 1. Clear bit for measured HC.
 2. Clear bits for all B/S HCs.

3.2.11.4.1.38.2 Data, Logic, and Command Paths

- a. Input Data
 1. Local Common
BCSCM1 - POINT, T, WRK
- b. Output Data
 1. Global Common
COM8S3 - HCST3G

3.2.11.4.1.38.3 Flowchart - See Figure 3.2.11-43.

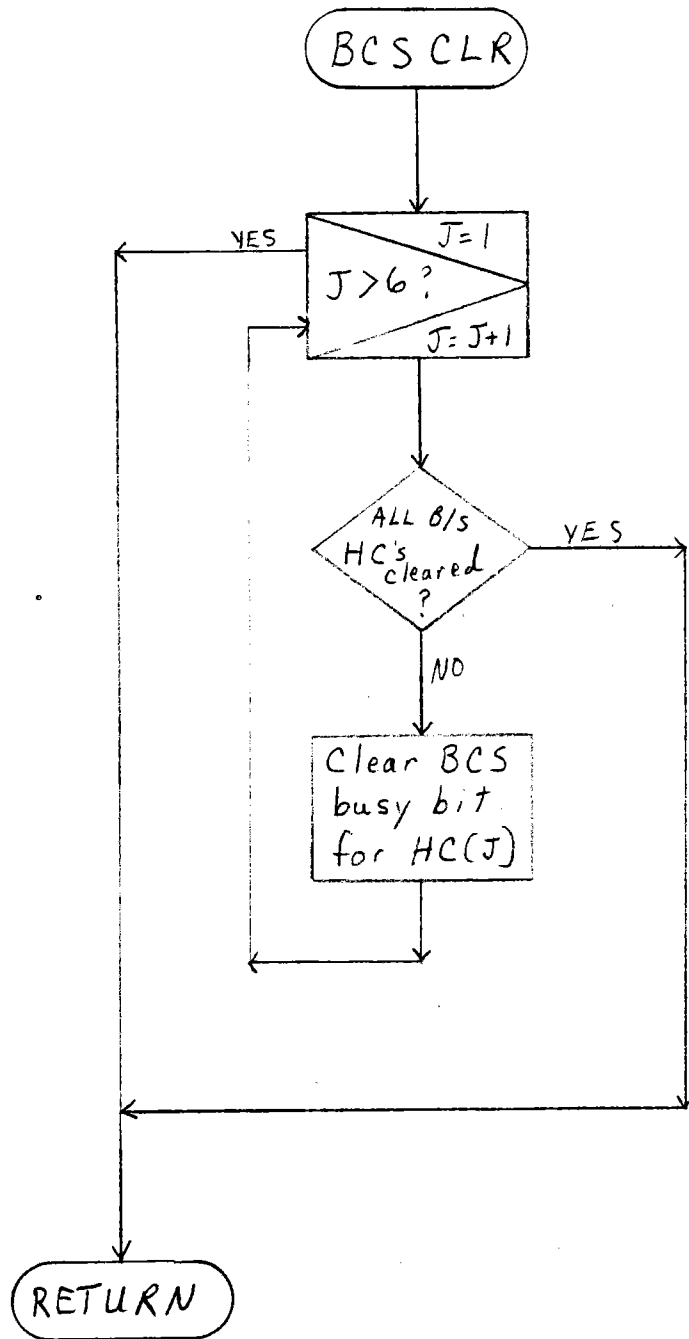


FIGURE 3.2.11-43 FLOWCHART - BCSCLR

3.2.11.4.1.39 Submodule XXXIX - BCSRTN

3.2.11.4.1.39.1 Description

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by BCSMPB and BCSMPC to return heliostats involved in a measurement to go to their previous modes.
- c. Constraints and Limitations - Should only be called to command HCs to return after measurements are finished or an abnormality occurs.
- d. Processing -
 1. Clear HSTAT; if DEFOCUS is in effect, command HC to STANDBY; otherwise, command to original mode.
 2. If no B/S HCs, go to Step 5.
 3. If DEFOCUS in in effect, command B/S HC to STANDBY; otherwise, command to original mode.
 4. If no more B/S HCs, go to Step 5. Otherwise, set STS7F and RTNF TO .TRUE., and return.
 5. Clear BCS BUSY BITS, set HSTAT = 5 and RTNF = .TRUE. and return.

3.2.11.4.1.39.2 Data, Logic, and Command Paths

a. Input Data

1. Local Common

BCSCM1 - T, BHC, POINT, WRK, SAVARR, WRKARR, HSTAT

2. Global Common

LOKDEG

b. Output Data

1. Local Common

BCSCM1 - HSTAT, BHC, RTNF, STS7F

BCSCM6 - DS1F

2. Global Common

COM1S2 HCDATG, HCCMDG

c. Calls

1. BCS Submodules

BCSCLR
BCSSTB

3.2.11.4.1.39.3 Flowchart - See Figure 3.2.11-44.

3.2.11.4.1.40 Submodule XL - BCSSTB

3.2.11.4.1.40.1 Description

a. Language Used - FORTRAN IV.

b. How Invoked - Called by BCSMPB, BCSRTN, and BCSMPC to command an HC to STANDBY mode.

c. Constraints and Limitations - The Global Data base must be initialized for corridor assignments.

D. Processing -

1. If XHC is not 1, set XHC to current B/S HC pointer.

2. Get corridor assignment.

3. Command HC to STANDBY mode.

3.2.11.4.1.40.2 Data, Logic, and Command Paths

a. Input Data

1. Parameters

XHC - If 1, command current HC to STBY;
not 1, command current B/S HC to STBY

2. Local Common

BCSCM1 - T, POINT, BHC, WRK

3. Global Common

COM8S3 - HCST3G
COMIN4 - CORRCG

b. Output Data

1. Global Common

COM1S2 - HCCMDG, HCDATG

3.2.11.4.1.40.3 Flowchart - See Figure 3.2.11-45.

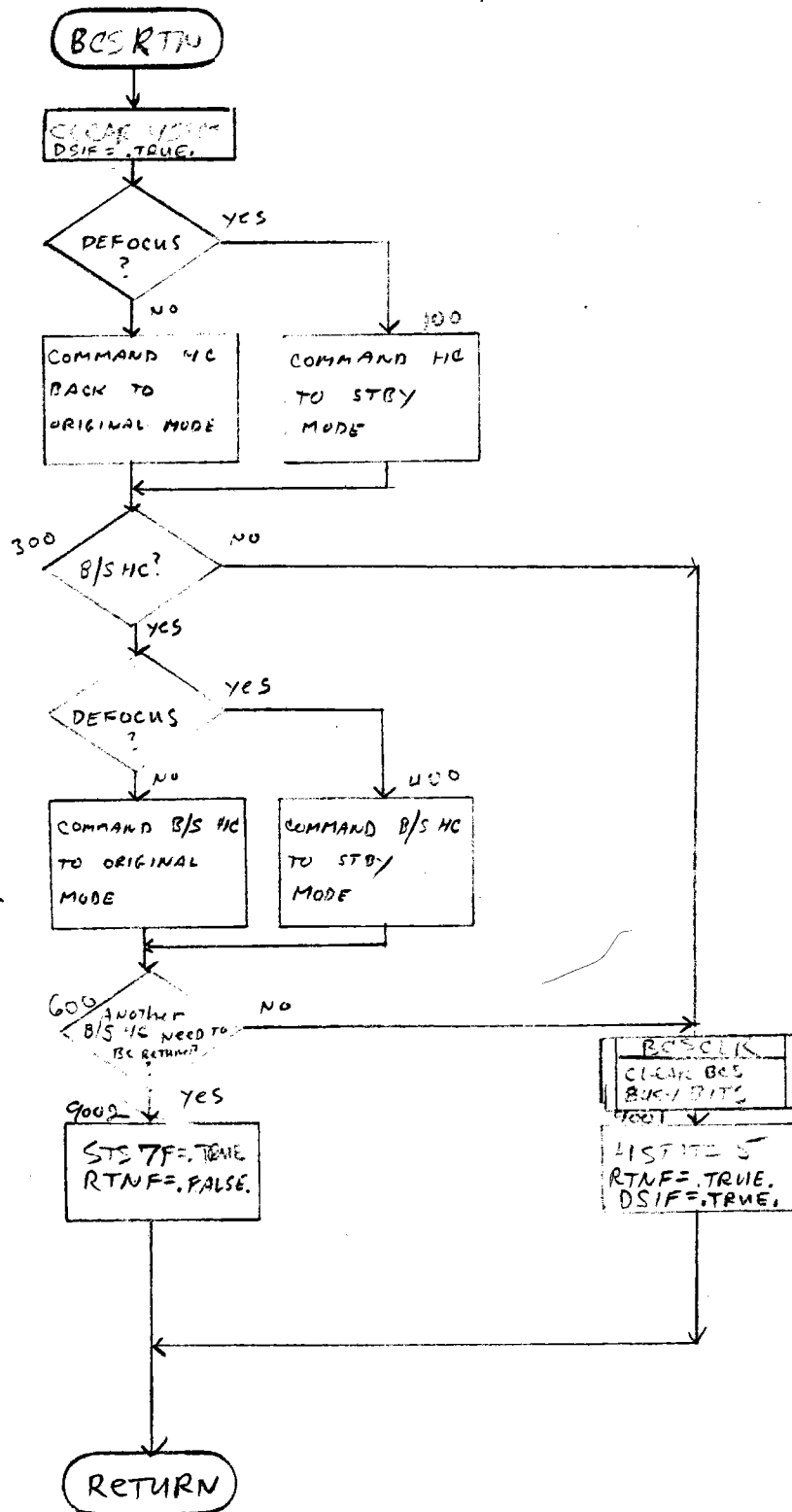


FIGURE 3.2.11-44 FLOWCHART - BCSRTN

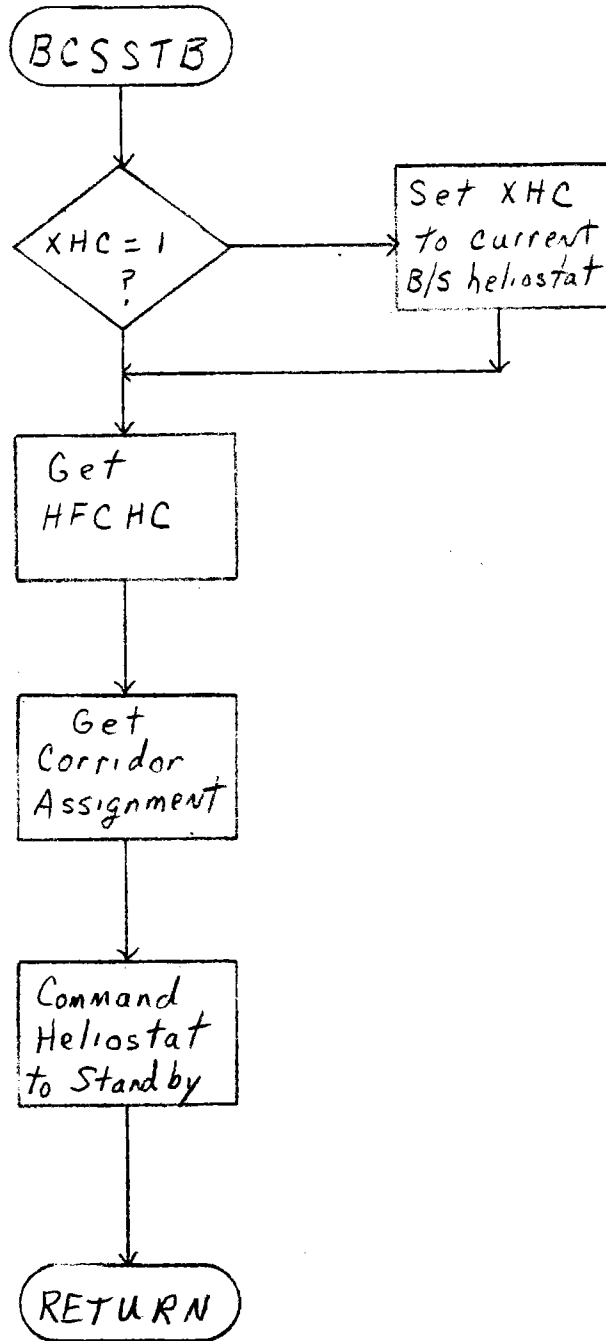


FIGURE 3.2.11-45 FLOWCHART - BCSSTB

3.2.11.4.1.41 Submodule XLI - BCSSTS

3.2.11.4.1.41.1 Description

- a. Language Used - FORTRAN IV.
- b. How Invoked - Call during each one-second loop by BCSMMC when measurement processing is active.
- c. Constraints and Limitations - Only called when measurement processing is active.
- d. Processing - Status of various HCs is checked, and MCOND_X is set accordingly.
 1. Check whether B/S HC command was queued; if so, set MCOND_X = 7.
 2. Check if HC and B/S HCs are all positioned for measurement. If so, set MCOND_X = 9.
 3. Check if HC is positioned; if so, set MCOND_X = 10.

3.2.11.4.1.44.2 Data, Logic, and Command Paths

a. Input Data

1. Local Common

BCSCM1 - STS7F, T, BHC, POINT, WRK, STS9F,
BEFOR, STS10F

2. Global Common

COM8S2 - HCSTIG
COMLS2 - HCCMDG

b. Output Data

1. Local Common

BCSCM1 - STSF, MCOND_X, STS9F, STSF, BEFOR,
STS10F, STS7F

3.2.11.4.1.44.3 Flowchart - See Figure 3.2.11-46.

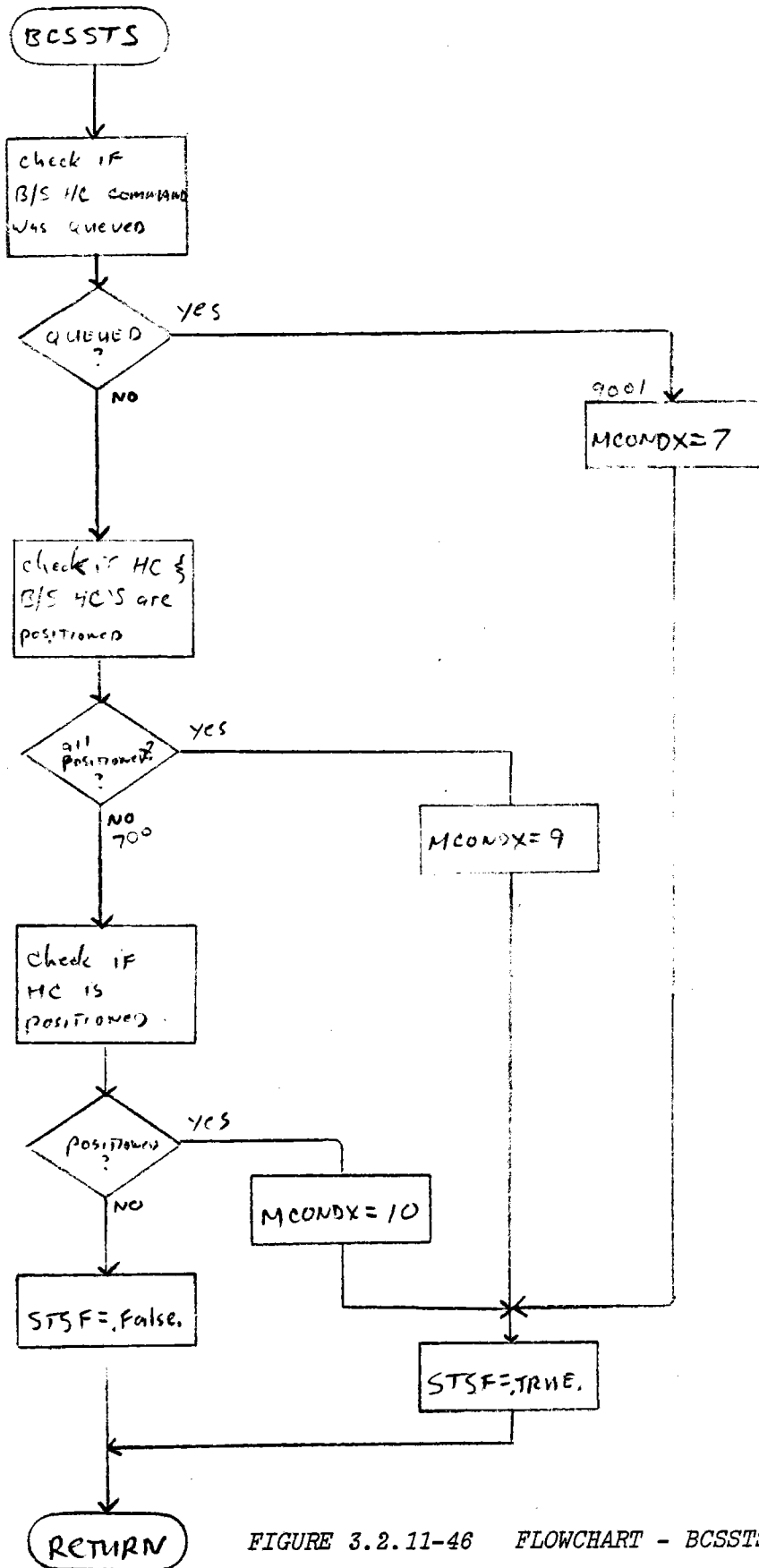


FIGURE 3.2.11-46 FLOWCHART - BCSSTS
798

3.2.11.4.1.42 Submodule XLII - BCSNET

3.2.11.4.1.42.1 Description

- a. Language Used - FORTRAN IV.
- b. How Invoked - Call by BCS and BCSMMC to check at least one and up to all five core devices for messages from the OCS every second.
- c. Constraints and Limitations - Limited to one core device read per call.
- d. Processing -
 1. Build UFT for core device M.
 2. Do rex service read from core device?
If read error, set M and return.
 3. Check write bit to determine if new information has been read, and set MSGF accordingly.

3.2.11.4.1.42.2 Data, Logic, and Command Paths

a. Input Data

1. Parameters

CORD - Core device number

b. Output Data

1. Local Common

BCSCM1 - M
BCSCM3 - Ibuff

2. Parameters

MSGF - New message received flag

3.2.11.4.1.42.3 Flowchart - See Figure 3.2.11-47.

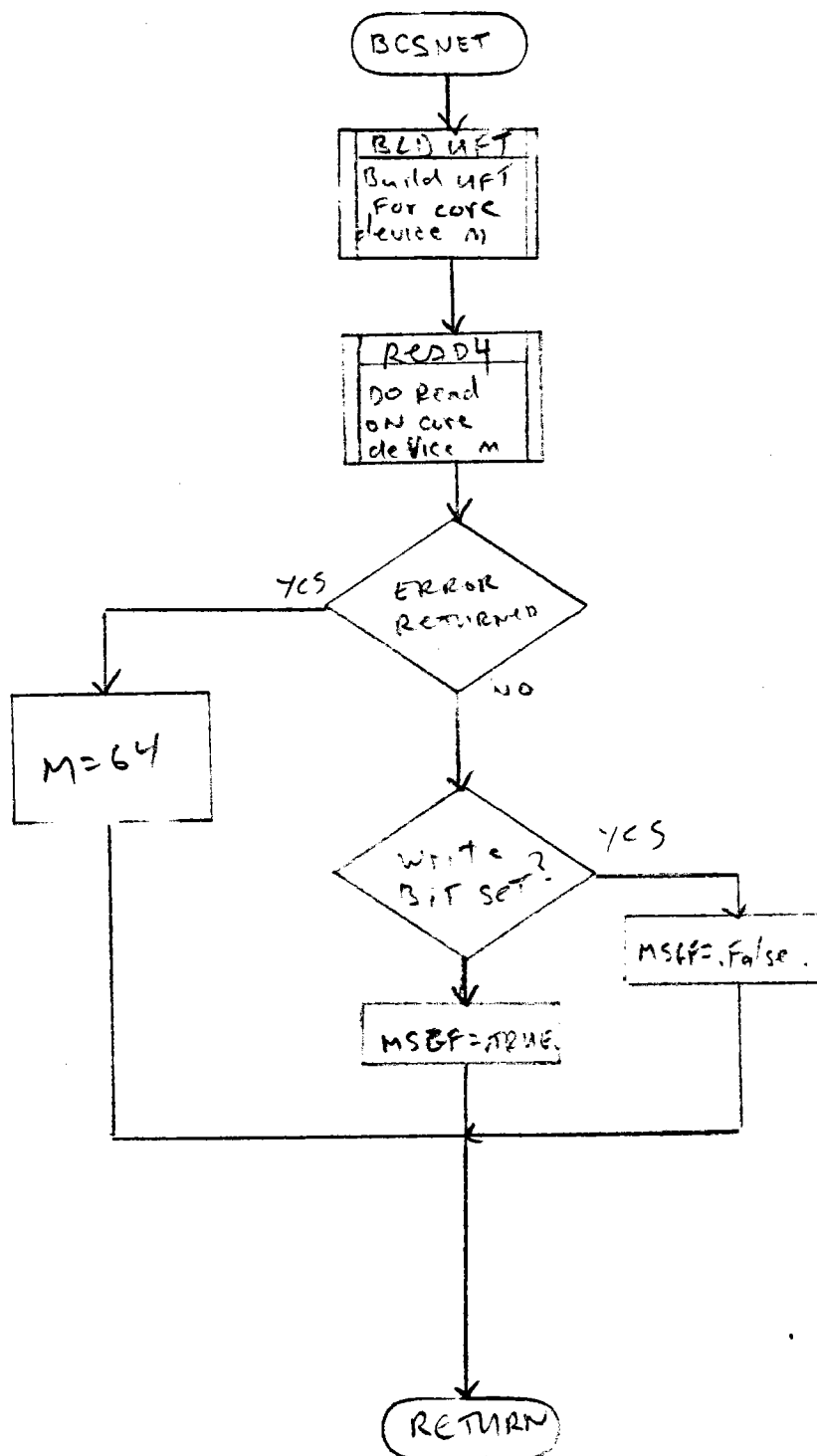


FIGURE 3.2.11-47 FLOWCHART - BCSNET

3.2.11.4.1.43 Submodule XLIII - BCSNWT

3.2.11.4.1.43.1 Description

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by BCSOUT to write a 40-word message to the OCS.
- c. Constraints and Limitations - It is assumed that at most two messages will be written to one core device before reads are done. If more than two writes are attempted, the buffer will be overwritten. This should not occur for the BCS design.
- d. Processing -
 1. A check is made to make sure last write has been completed. If not, a message is displayed and returned.
 2. The UFT is built. If core device 5 is specified, a separate UFT is built.
 3. A rex service write is performed. If a write error is detected, M is set for an alarm.

3.2.11.4.1.43.2 Data, Logic, and Command Paths

a. Input Data

1. Local Common

BCSCM3 - OBUFF

2. Parameters

N - Core device number

LU - Logical unit number option

b. Output Data

1. Local Common

BCSCM1 - M

BCSCMS - NPARM

2. Other

Message set to remote

Core device to OCS

3.2.11.4.1.43.3 Flowchart - See Figure 3.2.11-48.

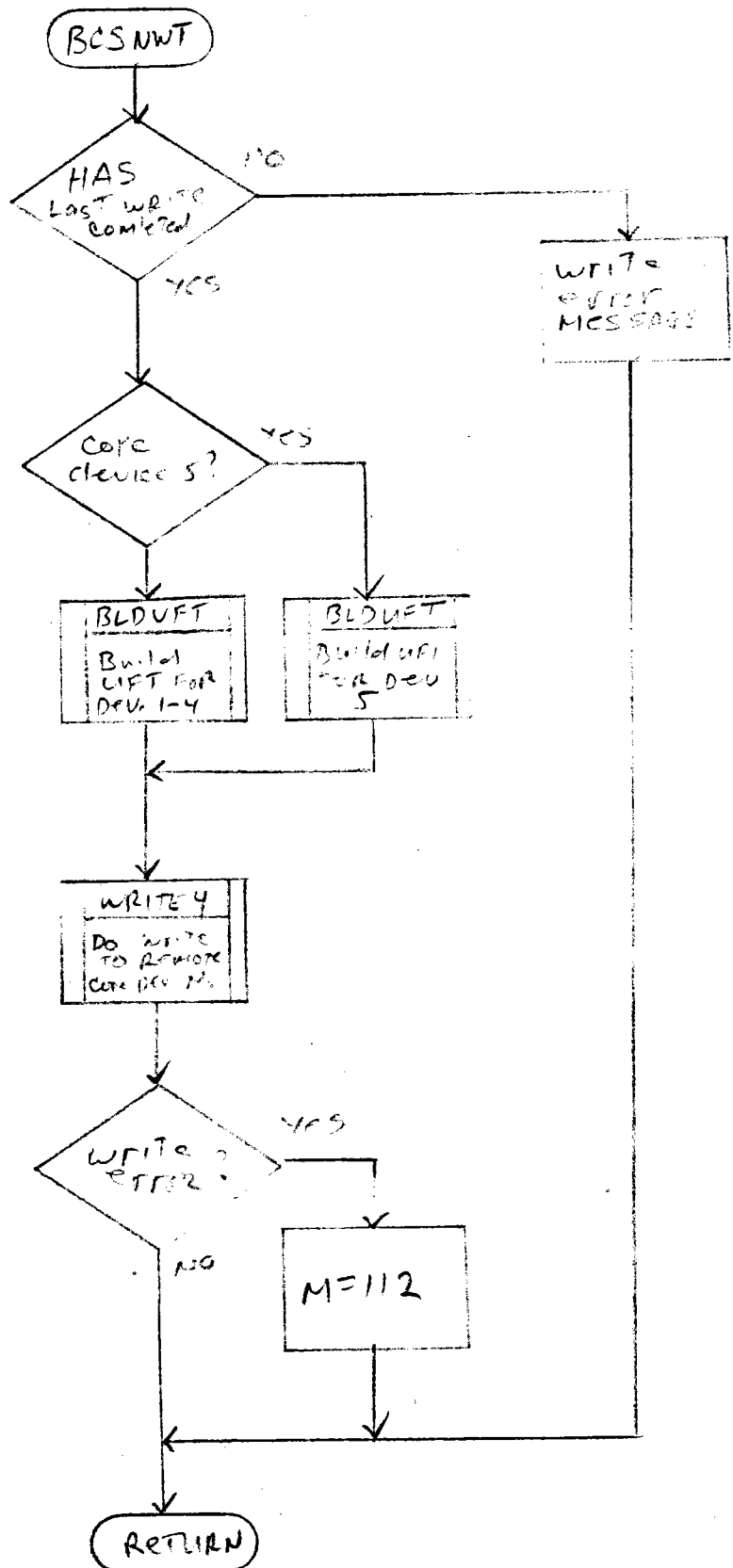


FIGURE 3.2.11-48 FLOWCHART - BCSNWT

3.2.11.5 Interface Description

- a. Input from the CS console via MMI and ENQUE/DEQUE.
- b. Input from OCS via local reads of core devices 1-5.
- c. Output to OCS via remake write to core devices 1-5.
- d. Output to CS console via CSO to visible cursor (error messages).
- e. Output to CS console via CSO to invisible cursor (file display, bias calculation results, and BCS active message).
- f. Output to CS console via ALMQUE for critical and non-critical alarms.
- g. Output to logger for bias calculation results.
- h. Disk access (read and write) to BCS and HCB partitions.
- i. Disk access (read only) to HCC and AL2 partitions.
- j. Magnetic tape read.
- k. Interface with MANMIF via TBUSYG global common.
- l. Interface with CSO via BCSLNG and CRTWRG global common.
- m. Interface with CMD via HCST3G and BCSBYG.

Some of the interfaces and the overall flow of the BCS task are depicted in Figure 3.2.11-28. An interface diagram within the HAC is shown in Figure 3.2.11-29.

3.2.11.6 Test Requirements

Verification of BCS's logic will be done by preparing a sample magnetic tape using SED. OCS responses will be simulated. BCS behavior will be observed using CRT displays, memory displays, and real-time behavior of real or simulated heliostats.

3.2.12 Heliostat Data Storage Capability - HDSMOD

3.2.12.1 Purpose

The purpose of this module is to record on magnetic tape, for a variable number of heliostats, at a variable time interval, the following data:

- (1) Time of Day
- (2) Sun Vector
- (3) Status
- (4) Azimuth and Elevation

3.2.12.2 Requirements

3.2.12.2.1 Design Requirements - Software requirements listed in Section 3.1 of the 10 MWe Collector Subsystem Software/Firmware Functional Requirements Specification, 12 June 1980, that apply to HDSMOD module are:

- a. Detect, report, and respond to failures and irregularities.
 - HDSMOD failures and irregularities only are detected and reported.
- b. Maintain a "Prime" and "Backup" system, as well as redundant field communications.
 - HDSMOD will function only in the "Backup" system. If failover occurs, HDSMOD will be terminated.

3.2.12.2.2 Derived Requirements - Software requirements described in the Statement of Work RC9132, March 18, 1981, are as follows:

- a. Read the magnetic tape mounted on the tape drive to obtain the list of heliostats for which data is to be recorded.
 - The data will be logged on this tape following the heliostat list. The maximum number of heliostats is 1,818.
- b. Interact with the operator via the MODCOMP Systems console for starting, time interval data, and stopping.
 - Stopping will occur when a specified time has elapsed or when a command is given. The time interval will be at least one minute. Start and end times are within a 24-hour period.

- c. Log on tape the time (GMT), sun vector, heliostat number, status (e.g., Stow, Tracking, Wash mode), azimuth, and elevation data at the specified time interval for each heliostat on the list.
- d. Notify operator via systems console that the tape is full.
 - This implies that the operator is to mount a new tape to proceed with logging.
- e. Terminate if a failover occurs (of prime HAC function from Prime HAC to Backup HAC).

3.2.12.3 Design Approach

HDSMOD is designed to operate in the 10 MWe Phase II Backup HAC. The basic approach is to:

- (1) Communicate with the operator in order to set up the logging as desired,
- (2) Do the logging automatically until the operator stops it, its end tape is past, or the tape becomes full. When the tape is full, the operator is asked to mount a new tape or to end. Any abnormalities will result in a message to the operator.

3.2.12.3.1 Functional Allocations - HDSMOD is organized into eight submodules, as shown in Figure 3.2.12-1, and described as follows:

- HDS - This is the main task of HDSMOD. It is the high-level control to read the magnetic tape, input information from the operator, and to initiate the logging process.
- HDSRDT - This submodule reads the magnetic tape converting the MDAC HC numbers to HFCHC numbers to initialized HCARR and NOHC.
- HDSCNV - Converts MDAC HC numbers to HFCHC numbers.
- HDSINP - Inputs start time, end time, and time interval from operator, and checks for errors.
- HDSGET - Inputs time (HHMM) from generator and displays message if bad.
- HDSLOG - This routine is the highest level of control for the actual logging occurring every time interval.
- HDSWRI - Writes a record to tape, displays message when tape is 90% full, and signals operator when tape is full and a new tape needs to be mounted.

HDSDEL - Delays for one time interval checking for the end time and for an operator message every second.

3.2.12.3.2 Resource Budgets

- a. HDS has the following memory requirements:
- b. Magnetic Tape Utilization
 - (1) Magnetic tape input for initialization and output for logging data while executing.
- c. Timing
 - (1) HDS will do processing once per time interval (one-minute minimum) to log all pertinent data, and once per second to check for operator input.
- d. Priority

The priority requirements for HDS are low. It operates in the background in the Backup machine.

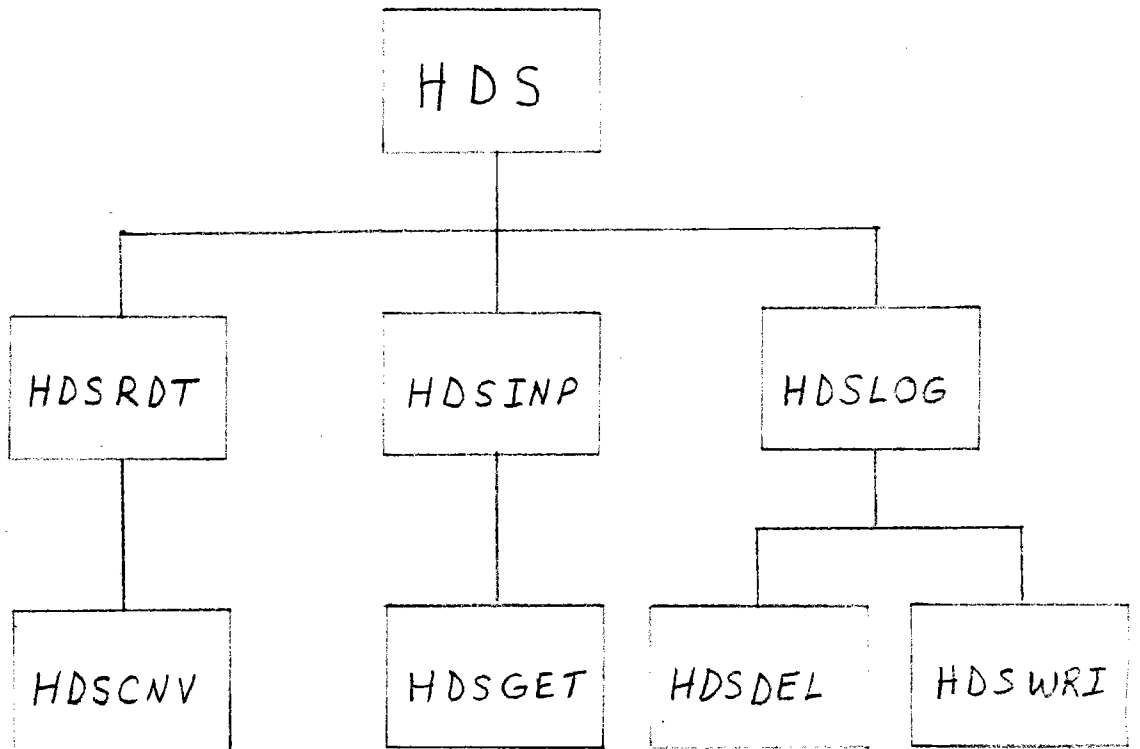


FIGURE 3.2.12-1 HDSMOD FUNCTION DIAGRAM

3.2.12.4 Design Description

3.2.12.4.1 Submodule I - HDS

3.2.12.4.1.1 Description

- a. Language Used - FORTRAN IV.
- b. How Invoked - Activated by operator.
- c. Constraints and Limitations - Must be activated in the Backup HAC. Must have magnetic tape with the desired heliostat list mounted on the proper tape drive.
- d. Processing
 - (1) The magnetic tape is read as part of initialization. If an error is detected, the operator is notified; and HDS is terminated.
 - (2) The start time, end time, and time interval are input from the operator.
 - (3) If a start time was entered, delay until the start time has passed.
 - (4) Do the logging.
- e. Error Messages and Recovery - None.

3.2.12.4.1.2 Data, Logic, and Command Paths

- a. Input Data
 - (1) Local Common - TSTART.
 - (2) Global Common - GTIMEG.
- b. Output Data - None.
- c. Calls
 - (1) HDS Submodules: HDSRDT
HDSINP
HDSLOG
 - (2) REX Services: DELAY

3.2.12.4.1.3 Internal Data Description

Local Common -

TBEGIN - Time in minutes that logging began
TSTART - Number of minutes to delay before logging is
 started
TEND - Number of minutes logging is to run
TWAIT - Time interval in minutes
HCARR - Contains all HCs specified on tape as HFCHC numbers
NOHC - Number of HCs specified in HCARR
NOREC - Number of records written to tape
FULL - Number of records written when tape is 90% full

3.2.12.4.1.4 Flowchart - See Figure 3.2.12-2.

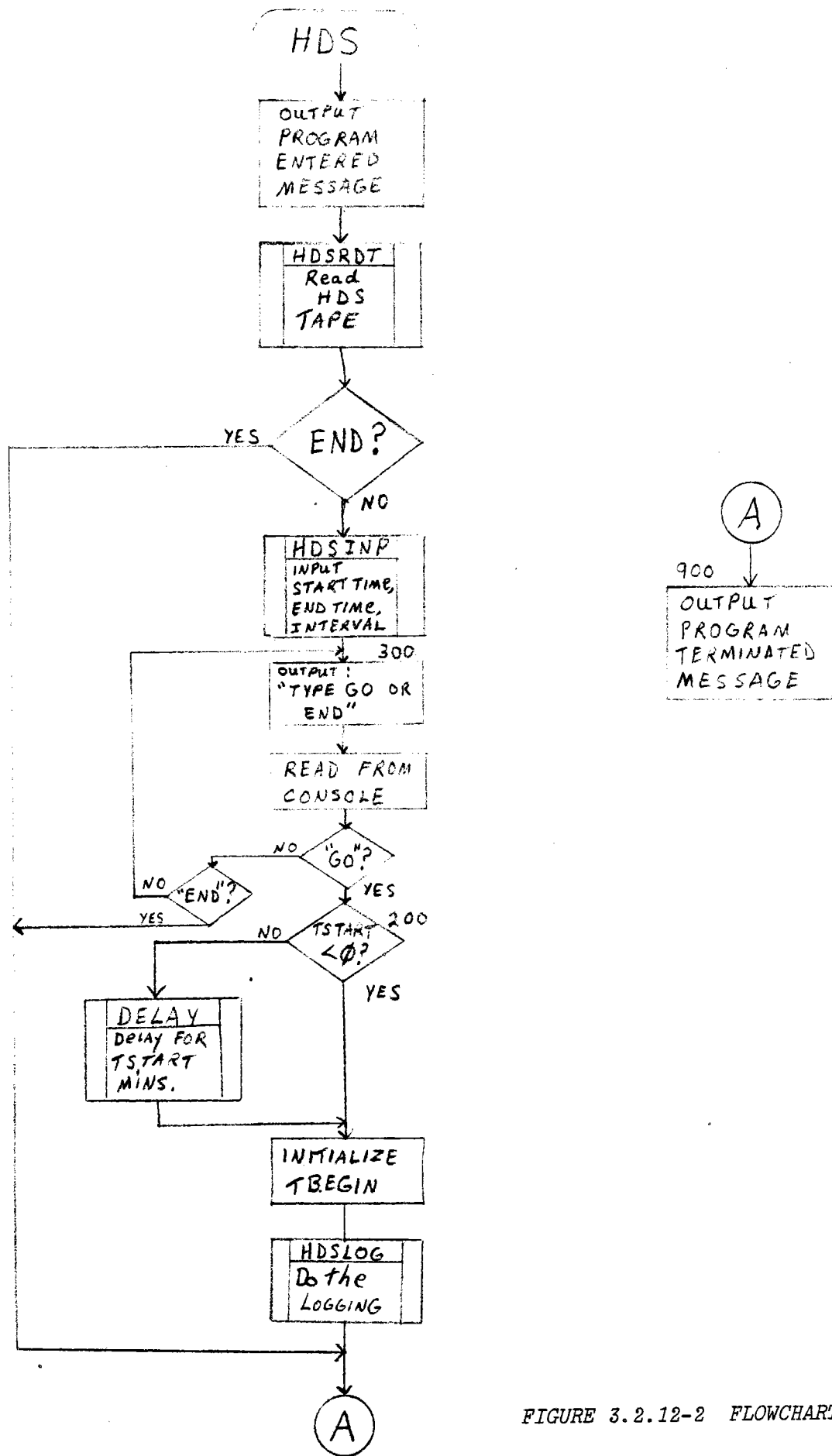


FIGURE 3.2.12-2 FLOWCHART HDS

3.2.12.4.2 Submodule II - HDSRDT

3.2.12.4.2.1 Description

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by HDS.
- c. Constraints and Limitations - The proper magnetic tape must be mounted.
- d. Processing -
 - (1) Rewind tape and initialize.
 - (2) Read first record; if EOF or not "HDS FILE", print error message and return.
 - (3) Read next record; if EOF, set HCPT and return.
 - (4) Decode record and set in HCS; if error, print message and return.
 - (5) Convert up to 15 MDAC HC numbers and increment HCPT. If bad HC number is found, print message and return. If all are good or \emptyset , go to Step (3).
- e. Error Messages and Recovery -
 - (1) If no file exists on tape, the message "NULL FILE ON TAPE" is issued.
 - (2) If a file is present on tape but is not preceded with the HDS file title, then the message "HDS FILE NOT FOUND" is issued.
 - (3) If any decodes fail while processing the HDS file, the message "BAD HC DATA ON TAPE" is issued.
 - (4) If any incorrect HC numbers are encountered on the file, the message "INVALID HC NUMBER" followed by the number is issued.

3.2.12.4.2.2 Data, Logic, and Command Paths

a. Input Data

Header and HC list from magnetic tape (see Table 3.2.12-I).

b. Output Data

- (1) Parameters - END - Flag set if task is to terminate.
- (2) Local Common - HCARR
NOHC

c. Calls

(1) HDS Submodules: HDSCNV

(2) Library CMRI
 CMR

3.2.12.4.2.3 Internal Data Description - None

3.2.12.4.2.4 Flowchart - See Figure 3.2.12-3.

INPUT TAPE FORMAT

<u>RECORD</u>	<u>COL</u>	<u>DATA</u>
1	1	HDS File
	10-80	Date or any comments
2-123	1, 6, 11, 16...71	Up to 15 HC numbers. Up to 122 records. The HC number following the last valid HC number should be all zeros. The four zeros should be followed by a blank and then a slash.
Last Record	1	END

Example:

HDS FILE HELIOSTAT DATA STORAGE TEST FILE 10/22/81

0001 0102 0103 0104 0105 0106 0107 0108 0109 0110 0111 0112 0113 0114 0115
0116 0117 0118 0119 0120 0000 /

END

TABLE 3.2.12-I

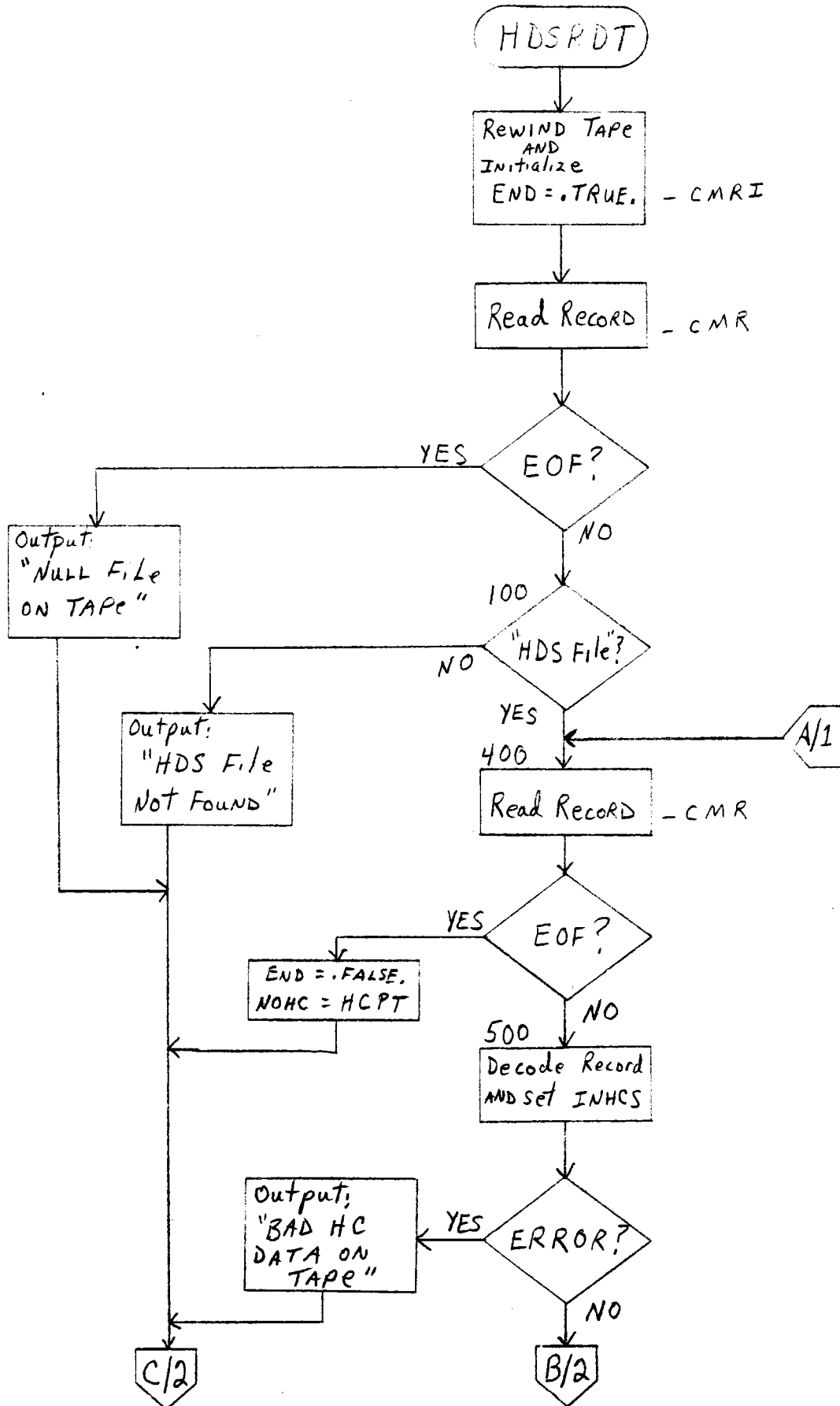


FIGURE 3.2.12-3 FLOWCHART HDSRDT

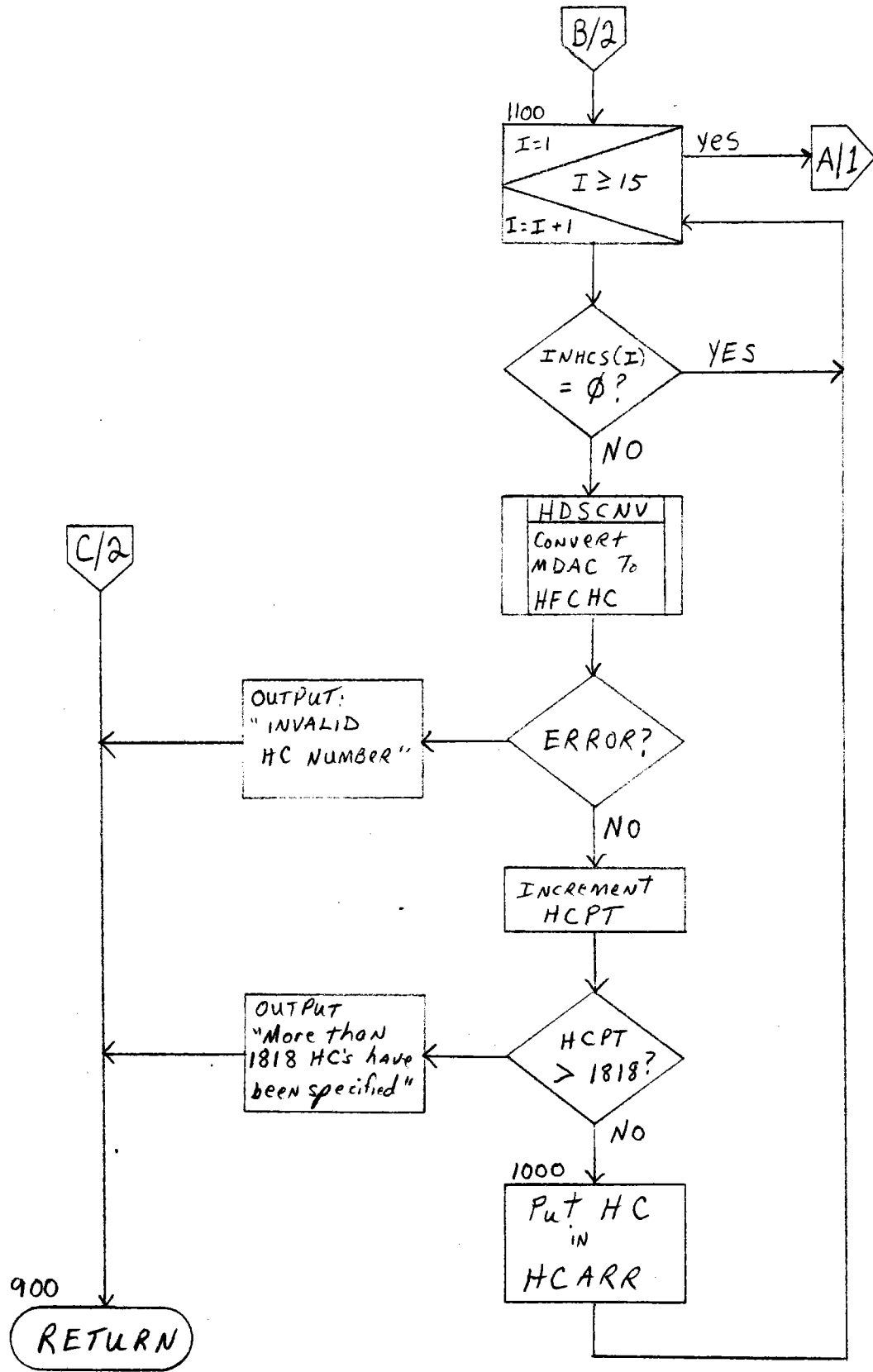


FIGURE 3.2.12-3 FLOWCHART HDSRDT (Continued)

3.2.12.4.3 Submodule III - HDSCNV

3.2.12.4.3.1 Description

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by HDSRDT.
- c. Constraints and Limitations - Input must be a valid MDAC heliostat number, and global common arrays MDNPRG and MD2HCG must be properly initialized.
- d. Processing - The MDAC row is isolated and checked for validity. If valid, the line number within the row is isolated. If the line number does not exceed the difference between the stored sum for this row and the next row, the HFCHC number is found directly in the MD2HCG array.
- e. Error Messages and Recovery - If one of the above tests failed, an invalid MDAC number was input; and error is set true while number is not changed.

3.2.12.4.3.2 Data, Logic, and Command Paths

a. Input Data

Parameters	NO - MDAC HC number to be converted
Global Common	MD2HCG MDNPRG

b. Output Data

Parameters	NO - HFCHC number ERROR - .TRUE. if bad MDAC HC is found
------------	---

3.2.12.4.3.3 Internal Data Description

ROW	- First two digits of MDAC number
LINE	- Last two digits of MDAC number
SUM	- Sum in global common for above ROW

3.2.12.4.3.4 Flowchart - See Figure 3.2.12-4 for the HDSRDT flowchart.

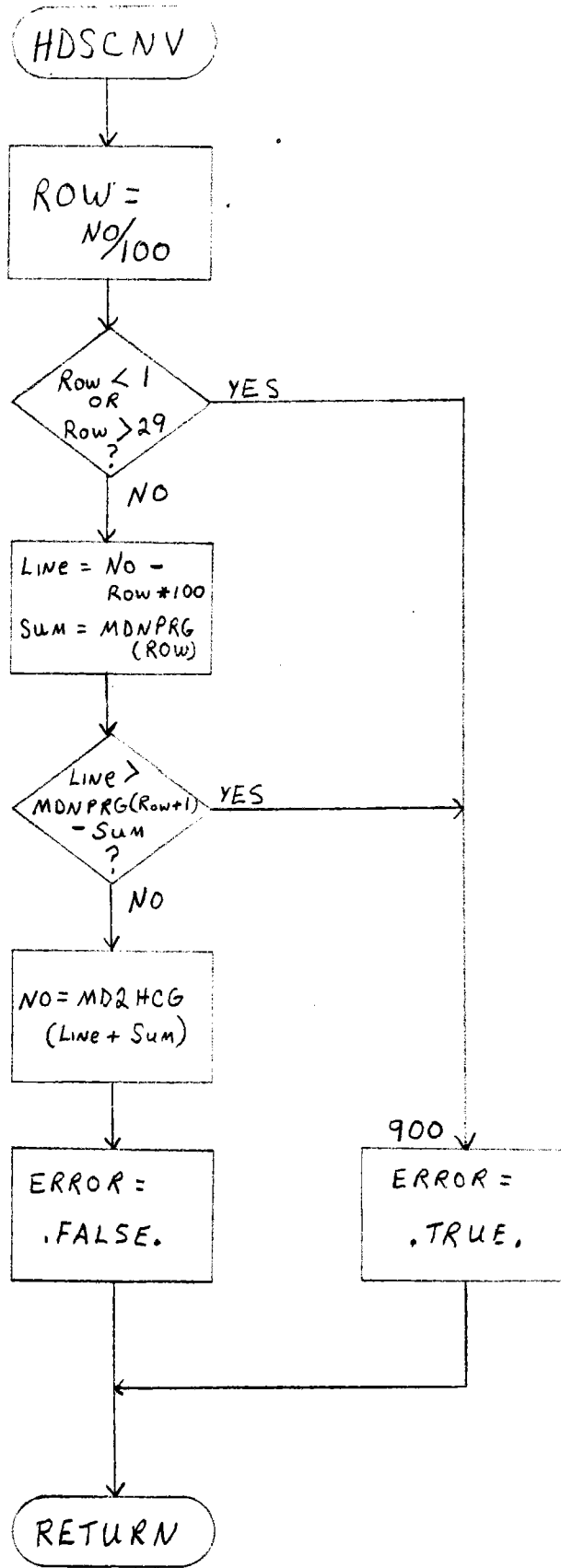


FIGURE 3.2.12-4 FLOWCHART HDSCNV

3.2.12.4.4 Subroutine IV - HDSINP

3.2.12.4.4.1 Description

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by HDS.
- c. Constraints and Limitations - None.
- d. Processing -
 - (1) The operator is asked if he wants to specify a start time. If not, set TSTART to -1 and go to Step (3).
 - (2) Input start time, check syntax, and initialize TSTART.
 - (3) Input end time, check syntax, and initialize TEND.
 - (4) Input time interval, check syntax, and initialize TWAIT.
- e. Error Messages and Recovery - None.

3.2.12.4.4.2 Data, Logic, and Command Paths

a. Input Data

Global Common - GTIMEG

b. Output Data

Local Common - TSTART, TEND, TWAIT

c. Calls

- (1) HDS Submodules - HDSGET

3.2.12.4.4.3 Internal Data Description - None.

3.2.12.4.4.4 Flowchart - See Figure 3.2.12-5.

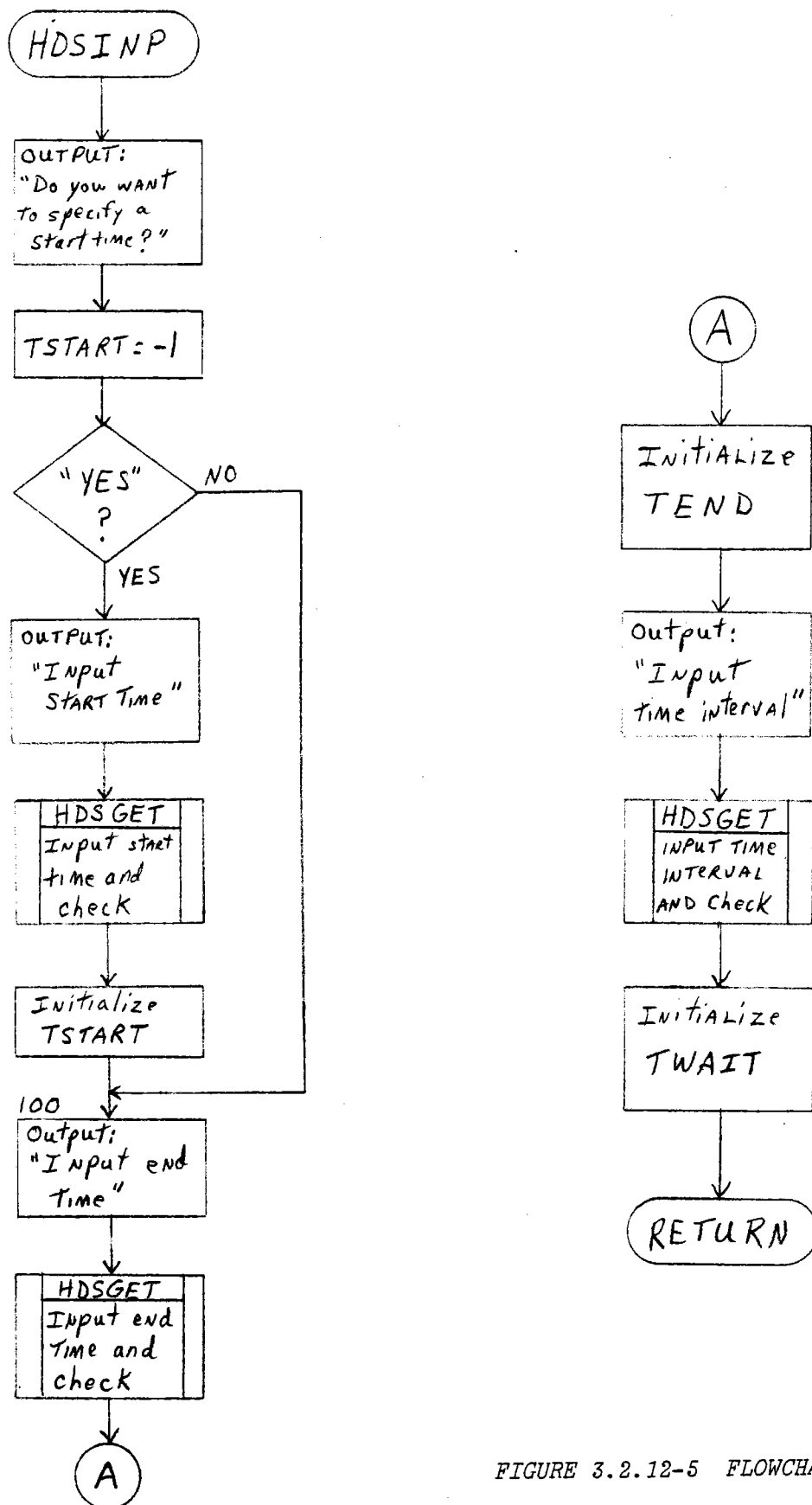


FIGURE 3.2.12-5 FLOWCHART HDSINP

3.2.12.4.5 Subroutine V - HDSGET

3.2.12.4.5.1 Description

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by MDSINP.
- c. Constraints and Limitations -
- d. Processing -
 - (1) Input time from system console.
 - (2) Check syntax if bad display message showing good format.
 - (3) Return.
- e. Error Messages and Recovery - Message to operator.

3.2.12.4.5.2 Data, Logic, and Command Paths

a. Input Data

System Console - Time input by operator.

b. Output Data

Parameters HH - Hours input
MM - Minutes input

c. Calls - None.

3.2.12.4.5.3 Internal Data Description

3.2.12.4.5.4 Flowchart - See Figure 3.2.12-6.

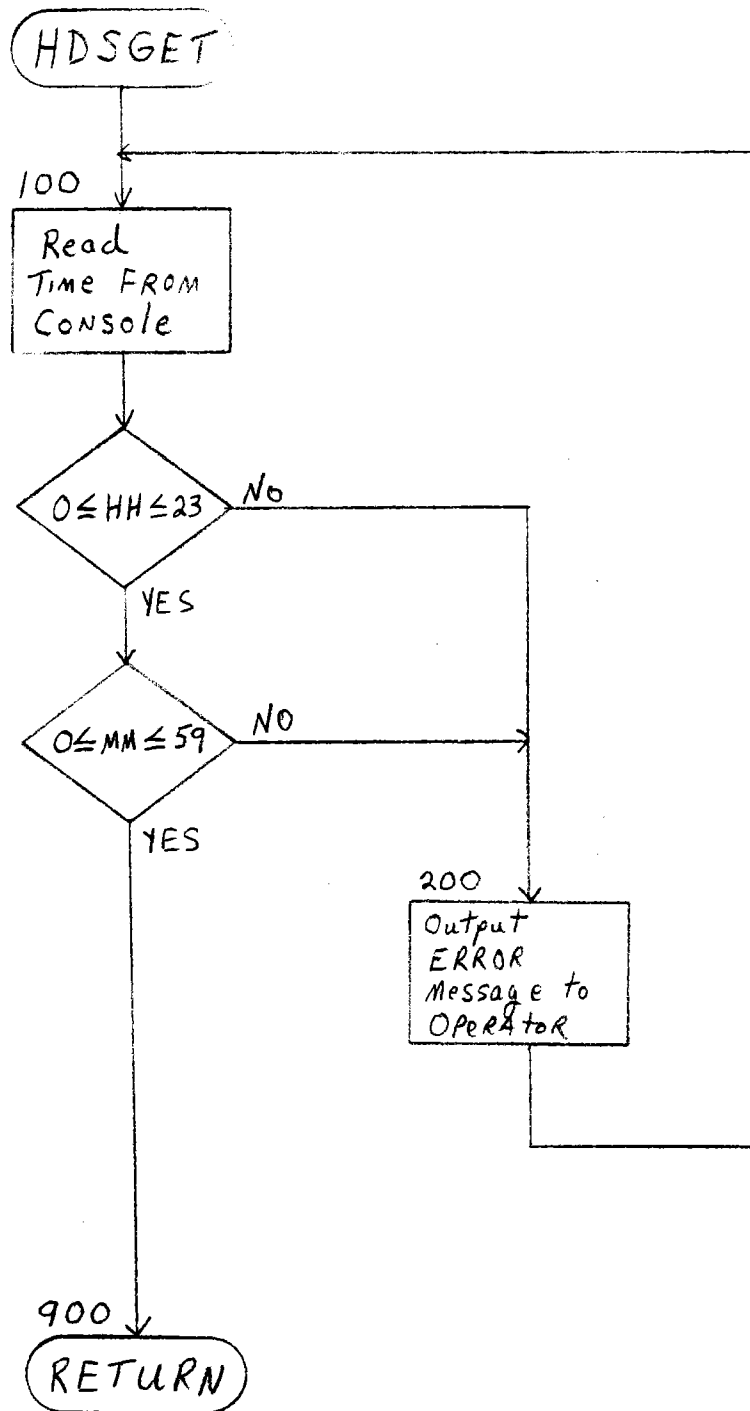


FIGURE 3.2.12-6 FLOWCHART HDSGET

3.2.12.4.6 Subroutine VI - HDSLOG

3.2.12.4.6.1 Description

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by HDS.
- c. Constraints and Limitations - TEND, TBEGIN, TWAIT, HCARR, and NOHC must be properly set.
- d. Processing -
 - (1) Initialize HCPT, get date and time for global common, and put in BUF.
 - (2) Get logging data from global common for up to 28 HC's and put in BUF. If HCPT is greater than NOHC, go to Step (4).
 - (3) Set remaining part of BUF to \emptyset , write BUF to tape; if END, return; otherwise, delay one time interval and look at console. If end is not requested, go to Step (1).
 - (4) Write BUF to tape; if end is not requested, go to Step (2); otherwise, return.
- e. Error Messages and Recovery - None.

3.2.12.4.6.2 Data, Logic, and Command Paths

- a. Input Data
 - (1) Local Common - HCARR, NOHC
 - (2) Global Common - GTIMEG
HCSTIG
AZIMG, ELEVG
SUNPOG
- b. Output Data
 - (1) Local Common - BUF
- c. Calls
 - (1) HDS Submodules - HDSWRI
HDSDEL

3.2.12.4.6.3 Internal Data Description - None.

3.2.12.4.6.4 Flowchart - See Figure 3.2.12-7.

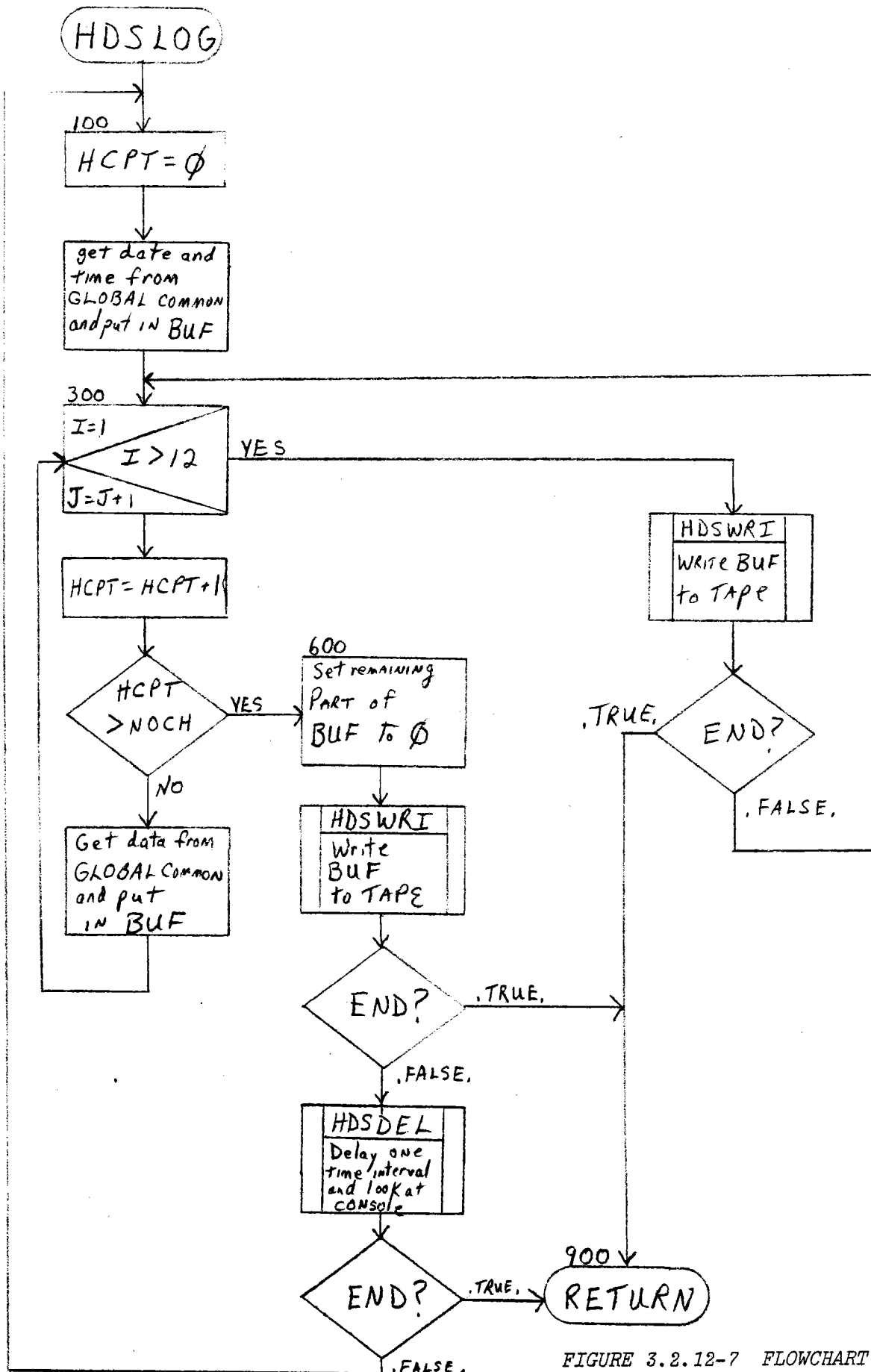


FIGURE 3.2.12-7 FLOWCHART HDSLOG

3.2.12.4.7 Subroutine VII - HDSWRI

3.2.12.4.7.1 Description

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by HDSLOG.
- c. Constraints and Limitations - BUF must be set up as desired; the appropriate tape must be mounted and ready.
- d. Processing -
 - (1) Increment record count (NOREC).
 - (2) If tape is 90% full, write message to operator.
 - (3) Write BUF TO TAPE; if error, write message to operator; if end is requested, return; otherwise, go to Step (3). If EOF, WEOF and rewind tape; write message to operator to mount new tape, or END. If new tape is mounted, go to Step (3); otherwise, return.
- e. Error Messages and Recovery -
 - (1) When the tape reaches 90% full, the message "THE TAPE IS 90% FULL" is issued to the systems console.
 - (2) If an error is encountered in writing to tape, the message "ERROR IN WRITING TO TAPE - ENTER GO OR END" is issued.
 - (3) When the tape is full, the message "ENTER END OR MOUNT NEW TAPE AND ENTER GO" is issued.

3.2.12.4.7.2 Data, Logic, and Command Paths

a. Input Data

- (1) Parameter - BUF - OUTPUT RECORD BUFFER
- (2) Local Common - NOREC, FULL

b. Output Data

- (1) Parameter - END - IF .TRUE. END REQUEST
- (2) Local Common - NOREC, EOT
- (3) Logging data written to tape (UNFORMATE), see Table 3.2.12-II.

3.2.12.4.7.3 Internal Data Description

3.2.12.4.7.4 Flowchart - See Figure 3.2.12-8.

OUTPUT TAPE FORMAT

<u>WORD</u>	<u>DATA</u>	
1-5	Year, Day, Hour, Min, Sec	} up to 65 records for one time interval
6-11	Sun Vector	
12	{ HC #	
13	* { Status (HCST2G)	
14, 15	{ Azimuth (AZIMG), Elevation (ELEVG)	
16-19	} up to 28 sets of *	
20-23		
24-27		
120-123		
124-126	Zeros	

TABLE 3.2.12-II

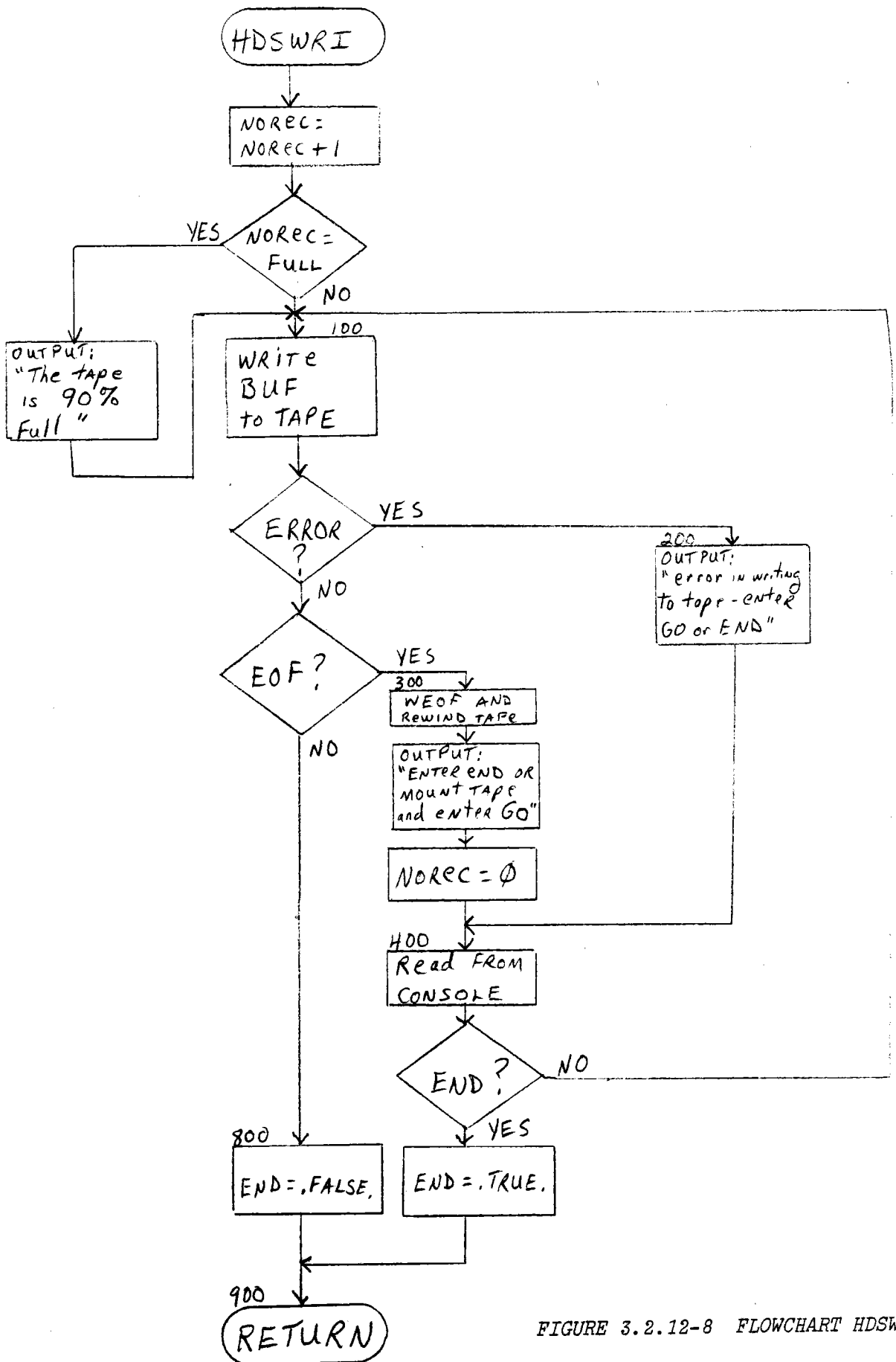


FIGURE 3.2.12-8 FLOWCHART HDSWRI

3.2.12.4.8 Subroutine VIII - HDSDEL

3.2.12.4.8.1 Description

- a. Language Used - FORTRAN IV.
- b. How Invoked - Called by HDSLOG.
- c. Constraints and Limitations - TWAIT and TBEGIN must be set.
- d. Processing -
 - (1) Initialize secs count.
 - (2) If time interval is expired, set END TO .FALSE. and return.
 - (3) Delay one sec and increment secs count.
 - (4) Check if end time is past; if so, WEOF and rewind tape, set END TO .TRUE. and return.
 - (5) Check if UFT is busy (read operation to console has not completed). If so, go to Step (2).
 - (6) If "END" message was entered, WEOF and rewind tape, set END TO .TRUE. and return.
 - (7) Build new UFT and do READ from console with QUICK and go to Step (2).
- e. Error Messages and Recovery -
 - (1) If an error is encountered calling delay, the message "DELAY TIMER ERROR C = " is issued.
 - (2) If the operator tries to stop the logging task with any message other than "END", the message "INVALID HOS COMMAND" is issued.
 - (3) When a failover occurs, the message "HAC FAILOVER LOGGING STOPPED" is issued.

3.2.12.4.8.2 Data, Logic, and Command Paths

- a. Input Data
 - (1) Local Common - TWAIT, TBEGIN, EOT
 - (2) Global Common - GTIMEG
- b. Output Data - None.

c. Calls

REX Service Calls - DELAY
BLDUFT
READY

3.2.12.4.8.3 Internal Data Description

3.2.12.4.8.4 Flowchart - See Figure 3.2.12-9.

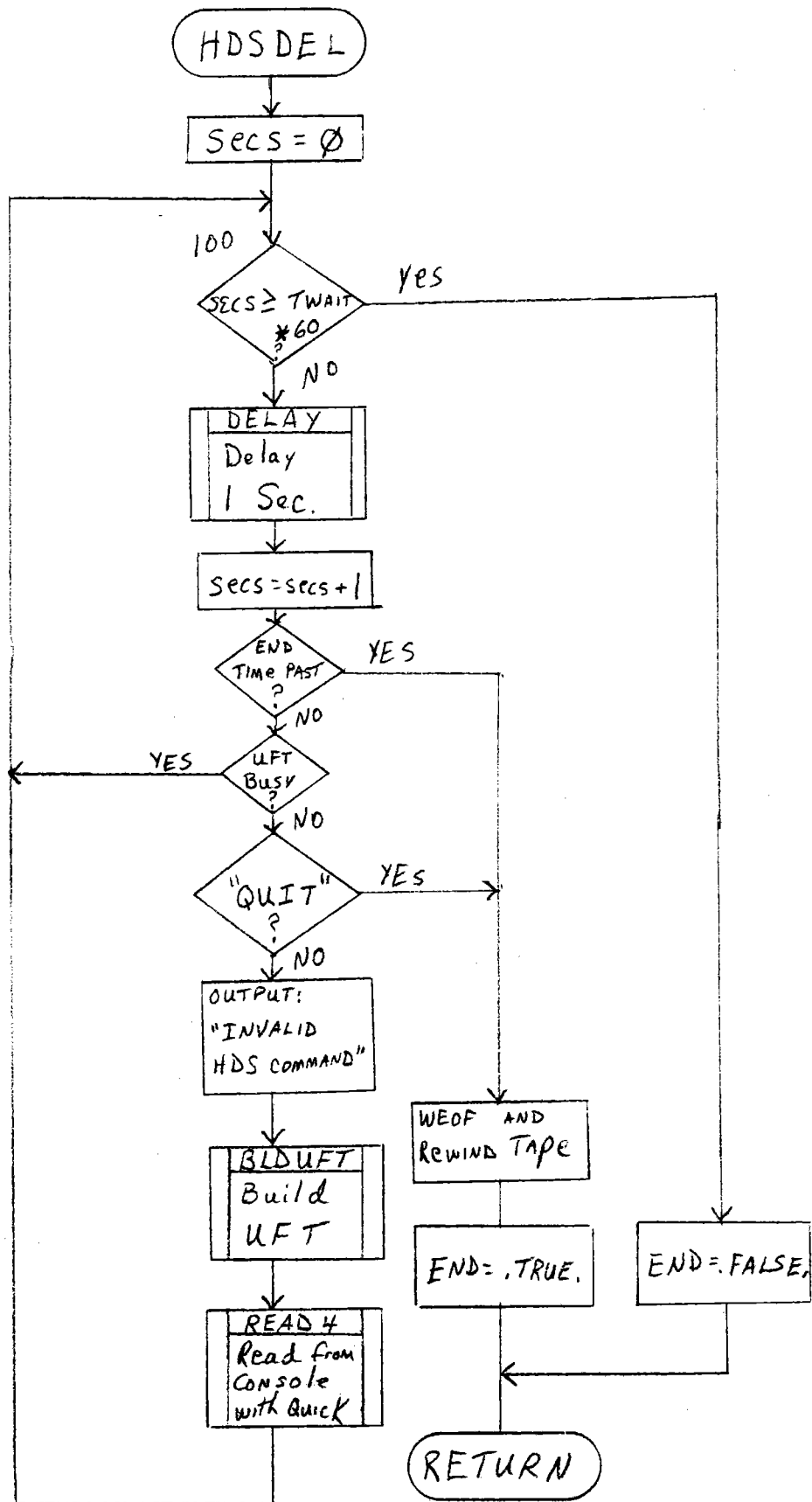


FIGURE 3.2.12-9 FLOWCHART HDSDEL

3.3.1 Data Base Design

3.3.1.1 Purpose

The data base, consists of in-core global common areas and disk-resident data files.

The in-core global common COMDAT serves as a means of communication between individual stand-alone tasks and between tasks and their associated support submodules. A number of status words, maintained by various external interface tasks, provide communication of real-time conditions describing the field of heliostats, devices, and software statuses. Further, data base tables of aim points, sun position, and corridor assignments are kept in global common to support command construction. Status words for the prime-backup HAC configuration are maintained for system integrity, and field address mapping schemes are maintained.

The global common area COMQUE supports communication processing with the OCS, DAS, CS, graphics, receiver trip and alarms interfaces.

The disk data base contains tables of:

- a. Heliostat location coordinates;
- b. Wash angles;
- c. Stow angles;
- d. ALT1 stow angles;
- e. ALT2 stow; angles;
- f. Heliostat corridor assignments;
- g. BCS assignments;
- h. Aim-point coordinate arrays;
- i. Bias angles;
- j. Field status snap-shot;
- k. Corridor coordinates;
- l. BCS coordinates;
- m. Field address mappings;
- n. Alarm messages (ASCII);
- o. Alarm messages packed; and
- p. Segment numbers and pecking order.

3.3.1.2 Requirements

3.3.1.2.1 Design Requirement

Software Requirements listed in Section 3.1 of the 10 MWe Collector Subsystem Software/Firmware Functional Requirements Specification, 12 June 1980, that apply to the data base are:

- a. Control of up to 2048 heliostats in all modes required to operate the heliostats;
- b. Monitor and display the operational status of all heliostats;
- c. Detect, report, and respond to failures and irregularities;
- d. Maintain a "Prime" and "Backup" system, as well as redundant field communications;
- e. Maintain a stable time base;
- f. Provide graphic displays of the heliostat field or field segments;
- g. Maintain safe beam control;
- h. Respond to receiver trip for emergency defocus;
- i. Provide the capability following power loss where the field may be commanded again within 60 seconds after power restoration;
- j. Maintain command/response protocol and data transfer with the OCS;
- k. Provide status data to the DAS; and
- l. Provide automatic control of beam characterization within the HAC.

In order to meet these requirements, the software in the HAC shall meet the following requirements supported in the data base:

- a. Control operational-phase sequences as required for integrated control of a field of up to 2048 heliostats;
- b. Generate and transmit heliostat mode commands (by HC, HFC, Segment, Wedge, Ring, Arc, Field) as required by the phase sequences during either operational or maintenance phases while maintaining safe beam control;
- c. Monitor the operational status of the heliostats as reported by the HFCs and report this status and detected irregularities in the form of displays and alarms;

- d. Maintain a "Backup" system by providing redundant field communications and providing data transfer sufficient to allow one-way "Backup" fail-over with minimal degradation;
- e. Provide a stable time base utilizing the WWV Trutime input for the "Prime" HAC time and internal time for the "Backup" HAC time. When the "Backup" HAC becomes "Prime", the WWV time will automatically be used, if available;
- f. Provide data transfer to the Chromatics Graphics Terminals sufficient for display of a finite number of graphic displays of the heliostat field or field segments and supporting alphanumeric information;
- g. Generate and transmit sun position information (once per frame) to the entire field;
- h. Maintain safety through controlled beam movement and inclusion area processing;
- i. Transmit emergency defocus command upon receiver trip;
- j. Provide automatic HFC initialization and the capability to command heliostats away from the receiver following initialization within 60 seconds after power restoration;
- k. Accept commands and transmit responses to the OCS including status and alarms;
- l. Respond to the STATUS command from the DAS by transmitting back the status message; and
- m. Provide automatic control of Beam Characterization upon operator request and establish communications and message transfer with the BCS task within the OCS.

3.3.1.2.2 Derived Requirements

The following derived requirements of individual modules have introduced the derived requirements of a data base:

Man-Machine Interface Module (MANMIF)

- a. Accept commands from the CS control console, OCS, DAS, CS graphics console, and read command files from disk;

- b. Decode command addresses for the Command Processor Module and convert commands to internal computer format;
- c. Check all commands for valid syntax;
- d. Generate error messages and route them to the appropriate output devices;
- e. Pass valid status commands to the Status Display Processor Module;
- f. Pass valid operational commands to the Command Processor Module;
- g. Log all commands on the console/printer with appropriate time stamps;
- h. Update a heliostat's bias disk file upon operator request; and
- i. Update aim-point array and perform inclusion-area checking.

Command Processor Module (CMDPRC)

- a. Check command arguments for reasonableness;
- b. Check heliostat's state for ability to execute commands;
- c. Translate operational commands and command sequences into individual HFC and HC commands;
- d. Maintain commanded heliostat mode table;
- e. Maintain sequence status arrays.

Sun Vector Module (SUNVEC)

- a. Calculate a unit sun vector as a function of universal time, on a once-per-second basis; and
- b. Format the sun-position unit vector into the format required for message packets and store in the data base.

Field Communications Processor Module (FLDCOM)

- a. Synchronize field operations by transmitting to the HFC computer a sun vector command once per frame;
- b. Generate, at the proper time in the time frame, the polling command for the HFC;

- c. Recieve the HC and HFC status in response to each polling command and save same in the data base;
- d. Transmit to the HFC, the operational commands generated by the Command Processor Module;
- e. Detect communications errors and automatically switch over to the redundant communications lines or the "Backup" computer and report these conditions to the Alarm Processor Module;
- f. Mark (calibrate) heliostats in both azimuth and elevation;
- g. Generate proper timing to accomplish all of the above in conjunction with HFC/HC firmware;
- h. Implement retransmission of HC commands (tracking and AZ/EL only; and
- i. Prevent communications failover upon detection of field power loss signal.

Alarm Processor Module (ALARMS)

- a. Monitor the heliostat status being returned from the field;
- b. Detect error conditions reported by the HC in that status;
- c. Report the alarms to the HAC operator using the alarms printer and an alarms area of the CS control console;
- d. Send alarm messages to the OCS through the interface;
- e. Display alarms detected by other software modules; and
- f. Maintain ONLINE/OFFLINE status for the heliostats.

Status Display Module (STATUS)

- a. Monitor heliostat status from status data reported by the HCs;
- b. Format field status;
- c. Display same on the STATUS area of the CS control console, updating display as required (see Figure 4);
- d. Respond to operator-entered commands for display status as reported from the MMI module, and format the status requested;
- e. Output the formatted status to the status printer; and

- f. Respond to OCS and DAS generated status requests by transmitting the requested status to the requestor.

Data Base Module (DBINIT)

- a. Define the global-common area, accessible to all applications tasks;
- b. Provide an initialization task to initialize the values in the global data base;
- c. Build appropriate data-base files for alarm messages, HC initialization coordinates, HC biases, control group mapping, BCS targets, STOW positions, WASH positions, corridor coordinates, initialization azimuth and elevation positions, and multiple aim-points;
- d. Read WWV time values from the WWV device if it is present and operating; and activate the "clock" task to perform time-base maintenance; and
- e. Initiate task execution sequences to the operating system; and allow for specification of "Prime" or "Bscup" computer and if "Prime" computer, make data available for transfer to "Backup", and if "Backup" computer, accept data from "Prime".

Operating System Modifications Module (MAXIVM)

- a. Generation of an operating system compatible with the peripheral equipment on the computer system;
- b. Universal time maintenance, using the WWVB Clock as a reference standard;
- c. Power fail/auto-restart capability;
- d. The HAC receives two signals (one the inverse of the other) from the Receiver System. One signal causes an emergency Defocus action--its inverse will allow an operator Defocus-Release action and a return to normal operations;
- e. Monitor status between computers to determine if the "Backup" should take over;
- f. Perform peripheral switching when necessary; and
- g. Detect a power loss signal and store this information in the data base for information to all tasks.

Graphics Display Processor Module (GRAPHC)

- a. Provide the functional interface with the graphics system in a command/response protocol;
- b. Provide the graphics system with sufficient data for graphics displays;
- c. Accept STHIWIND and DEFOCUS commands from the CS control room graphics function switches;
- d. Output data to the Chromatics Graphics Terminals only when requested; and
- e. Interface with the Chromatics Graphics Terminals through two distinct Dual Asynchronous Communications Channels (#4811).

External Interface Module (EXTINF)

- a. Interface with the CS Control Console by:
 1. Accepting inputs and passing these inputs to the respective modules; and
 2. Regulating the outputs to the CRT so that only one submodule is outputting at a time;
- b. Interface with the Chromatics Terminals by:
 1. Pass emergency STHIWIND and DEFOCUS commands to MANMIF module; and
 2. Data requests to the GRAPHC module.
- c. Interface with the OCS computer via a command/response and alarm-message protocol; and
- d. Interface with the DAS by accepting a STATUS request and transmitting a STATUS response.

BCS Automatic Processing Module (BCSMOD)

- a. Give the operator selection flexibility by using three different heliostat lists;
- b. Provide start capability of the automatic sequence that does not require further operator input;
- c. Execute control of a list of up to 60 heliostats and up to five blocking heliostats associated with each heliostat to be measured.

- d. Execute control over these measurements using up to four BCS targets in parallel;
- e. Communicate with the OCS for BCS measurements and offset values;
- f. Provide operator feedback for errors in automatic sequences; and
- g. Calculate azimuth and elevation bias changes using OCS measured centroid offsets from up to three BCS measurements.

In each case, inter-task communication is facilitated through global common storage of parameters, status, or flags. In cases where the data base is not specifically mentioned, its (data base) inclusion is designed to accent modular structure.

3.3.1.3 Design Approach

3.3.1.3.1 Functional Allocations

The in-core global common data base is designed to maintain the communications and temporary buffer storage for the software for support of a field of up to 2048 heliostats. As such, specific limitations exist on the data base and on certain variables within it. These constraints are outlined following each variable definition.

The general global common area COMDAT is subdivided into fourteen specific global commons, namely COM1S1, COM1S2, COM1S3, COM1S4, COM1S5, COM1S6, COM8S1, COM8S2, COM8S3, COM8S4, COMIN1, COMIN2, COMIN3, and COMIN4. A detailed description of each of these global common areas is given in Section 3.3.1.4. Table 3.3.1-I provides a cross reference for all the variables within the general global common area COMDAT and the related tasks. Table 3.3.1-II provides a cross reference between the global common areas and their associated tasks.

COMQUE is maintained as a communications global common as it must operate as a dynamic buffer. Its function is a message buffer for inter-processor communications, and for alarms awaiting output. Table 3.3.1-III provides a cross reference for all the message queues within global common area COMQUE.

The disk data base is used to maintain:

1. Externally supplied data for in-core global common initialization;
2. Heliostat locations for field initialization;
3. Corridor coordinates;
4. BCS coordinates;

5. Aim-point coordinates;
6. Stow angles;
7. Wash angles;
8. Alternate 1 stow angles;
9. Alternate 2 stow angles;
10. Heliostat bias angles;
11. Heliostat field status;
12. Alarm messages; and
13. BCS HC measurement files.

Tables 3.3.1-IV provides a cross reference for all the disk data base files and user tasks.

3.3.1.3.2 Resource Budgets

The global common data bases COMDAT and COMQUE require a total of 59,136 words of main memory at all times (COMDAT - 203 pages; COMQUE - 16 pages; COMQU2 - 16 pages). The disk data base requires 295,168 words.

3.3.1.4 Design Description

3.3.1.4.1 Structure

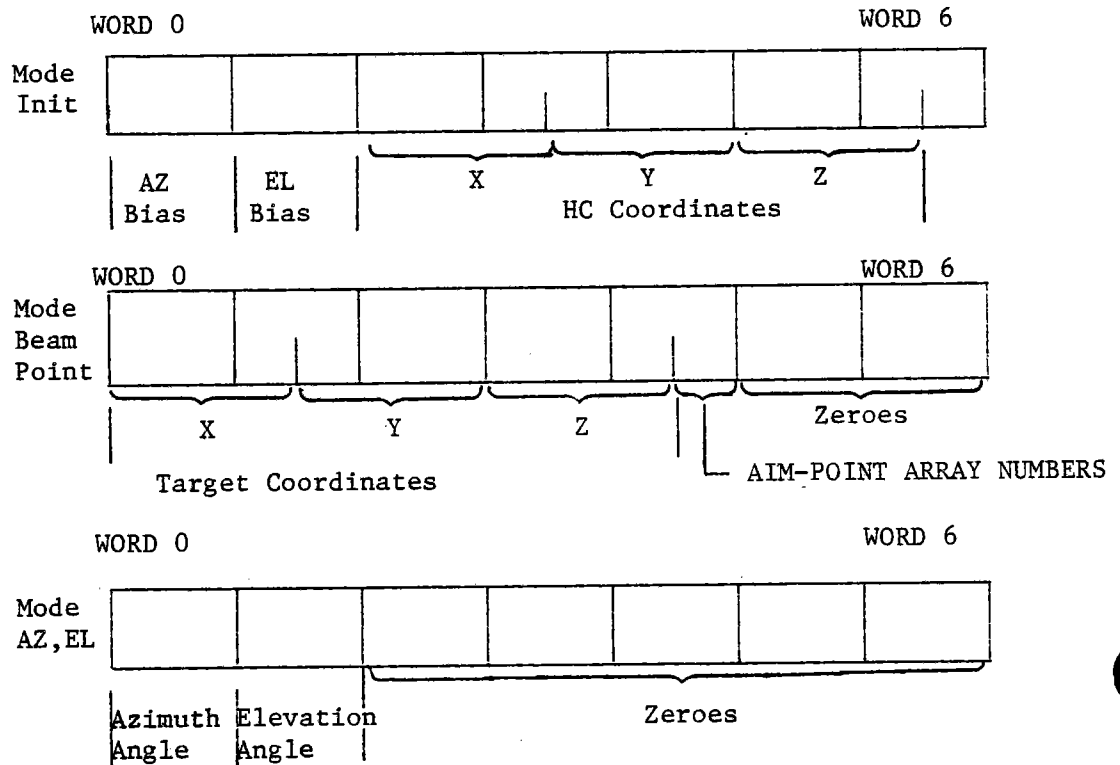
The following sections individually define variable and array structure. Variable and array contents are specifically defined and where applicable, individual bit settings are defined.

3.3.1.4.1.1 Global Common Area COM1S1

COM1S1 utilizes 16 pages of global common and is transferred to the backup HAC every second. Unless otherwise noted, variable type is INTEGER*2 and the size is in words.

<u>VARIABLE</u>	<u>SIZE</u>	<u>SET BY TASK(S)</u>	<u>USED BY TASK(S)</u>
b. <u>HCDATG</u>	14336 (7 x 2048)	BCS, CMD, GET, SEQ, DBI	BCS, BHC

Description - HCDATG is the most recent data buffer sent to HC



Initial value: 0

Constraints and limitation - HCDATG is designed to maintain commands for up to 2048 heliostats.

3.3.1.4.1.3

Global Common Area COM1S3

COM1S3 utilizes 8 pages of global common and is transferred to the backup HAC every second. Unless otherwise noted, variable type is INTEGER * 2 and the size is in words.

<u>VARIABLE</u>	<u>SIZE</u>	<u>SET BY TASK(S)</u>	<u>USED BY TASKS</u>
<u>GRSTSG</u>	2048	DBI, STS	GRF

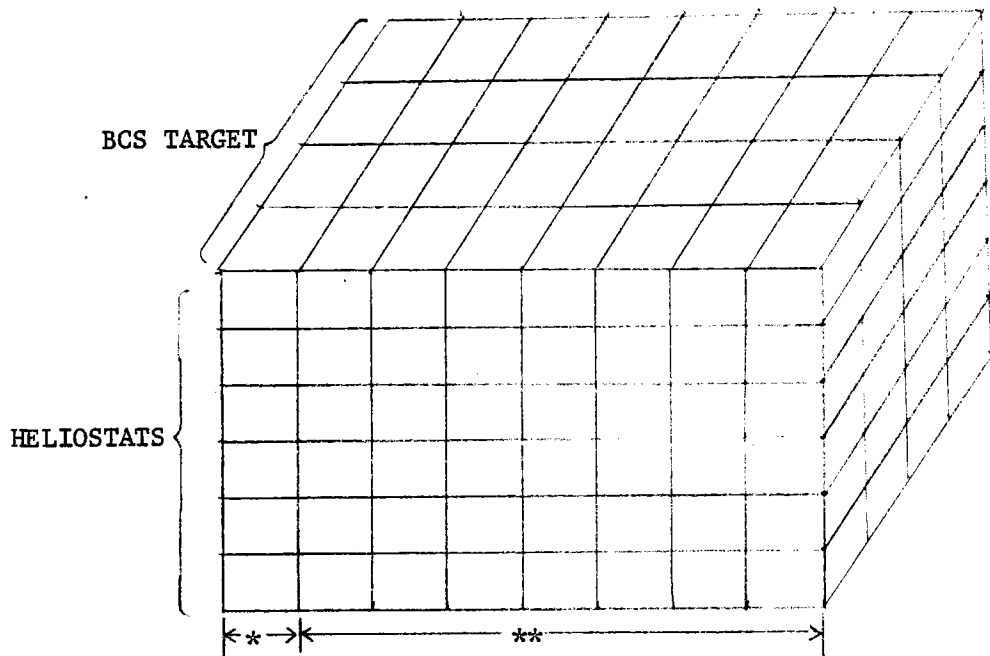
Description - Mode description for the graphics display, where:

- 0 = Not Installed
- 1 = Track
- 2 = Standby
- 3 = BCS
- 4 = Transition
- 5 = Stow
- 6 = Alternate 1 Stow
- 7 = Alternate 2 Stow
- 8 = Mark
- 9 = Directed Position
- 10 = Wash
- 11 = Initialized
- 12 = Offline

Initial value: 0 for those not installed, otherwise 12.

	<u>VARIABLE</u>	<u>SIZE</u>	<u>SET BY TASK(s)</u>	<u>USED BY TASK(s)</u>
d.	<u>BCSHSG</u>	192 (8x6x4)	BCS, DBI	BCS

Description - BCS active heliostat status.



Initial value: 0

Constraints and limitations - BCSHSG is designed to maintain BCS control for up to four BCS targets with six heliostats associated with each target during a BCS subsequence.

* Copy of HCCMDG (heliostat)

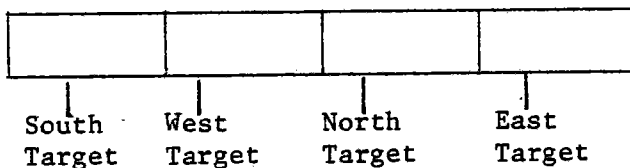
** Copy of HCDATG (heliostat, 1 to 7)

e.	<u>BCSTAG</u>	4	BCS, DBI	BCS
----	---------------	---	----------	-----

Description - Automatic BCS target status

Word 0

Word 3



Where each target word is defined as:

Bit 0

Bit 15



Operational
Mode

0 = Non-Operational

1 = Operational

2 = In Use

Initial value: 0

Constraints and limitations - BCSTAG is designed to maintain up to four BCS targets.

	<u>VARIABLE</u>	<u>SIZE</u>	<u>SET BY TASK(S)</u>	<u>USED BY TASK(S)</u>
f.	<u>CMDBFG</u>	640 (10x64)	BHC, BHI, DBI	FCP

Description - Command buffer, command information to HFC/HC. Each 10 word buffer corresponds to a 20 byte command for each of the 64 HFCs. Command formats can be found in the Software/Firmware Functional Requirements Specification.

Constraints and limitations - None

g. There are six spare words in this common.

Constraints and limitations -

3.3.1.4.1.5 Global common area COM1S5

COM1S5 utilizes 1 page of global common and is transferred to the backup HAC every second. Unless otherwise noted, variable type is INTEGER*2 and the size is in words.

<u>VARIABLE</u>	<u>SIZE</u>	<u>SET BY TASK(s)</u>	<u>USED BY TASK(s)</u>
a. <u>AIMBSG</u>	1	DBI, DSK, MMI	MMI

Description - Aimpoint array busy word used when the aim-point is being updated on the disk data base. Values in the word have the meaning:

- 0 = Not in use
- 1-20 = Aim-point array (1-20) being updated (do not access)

Initial value: 0

b. <u>ALRMSG</u>	1	ALM, ALO, DBI	ALO
------------------	---	---------------	-----

Description - Alarm message response word that has the meaning:

- 0 = Tells ALO to clear critical alarms line (i.e. acknowledge)
- 1 = Tells ALO to display next critical message, if any
- 2 = Tells ALO to wait for operator acknowledgement of the last critical alarm.

Initial value: 0

c. <u>CORRSG</u>	8	BHC, SEQ, DBI	BHC, SEQ
------------------	---	---------------	----------

Description - Corridor status array; used to initiate corridor "walks" and mark corridors in use.

Word 0

Word 7

1	2	3	4	5	6	7	8

Corridors

Where each word has the meaning:

	<u>Going up</u>	<u>Going down</u>
0 = Free	16 = A	17 = A
1 = Ready for up	18 = B	19 = B
2 = Ready for down	20 = C	21 = C
	22 = D	23 = D

Initial value: 0

Constraints and limitations - CORRSG is designed to maintain corridor status for up to eight corridors.

Initial value: 0

	<u>VARIABLE</u>	<u>SIZE</u>	<u>SET BY TASK(S)</u>	<u>USED BY TASK(S)</u>
j.	<u>FIRSTG</u>	1	ALM, ALO, DBI	ALM, ALO

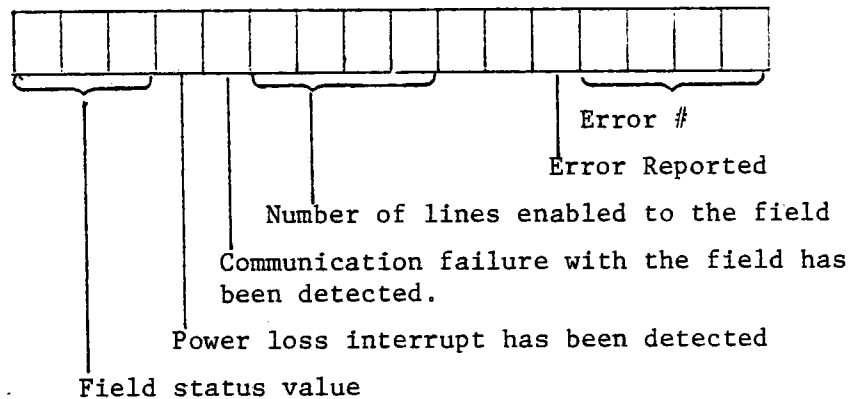
Description - Free storage index of first alarm in queue.
Initial value: 0

k.	<u>FLDSTG</u>	1	ALM, DBI, FCP	ALM, FCP
----	---------------	---	---------------	----------

Description - Field status (communications, power, and error).

Bit 0

Bit 15



0 = Field is in active state
1 = Field communications are enabled (a HFC has responded with a status response message)

Constraints and limitations - FLDSTG is designed to maintain up to eight communication lines and up to 16 error numbers.
Initial value: 0

l.	<u>GTIMEG</u>	14	TIK, TOK	ALO, BCS, MMI, STA, STS, SUN
----	---------------	----	----------	---------------------------------

Description - Time keeping array, where the words contain:

Word 0: Year (GMT), (1980-00)
Word 1: Day of year (GMT), (1-366)
Word 2: Hour in day (GMT), (0-23)
Word 3: Minutes in hour (GMT), (0-59)

Word 4: Second in minute (GMT), (0-59)
Word 5: Local year, (1980-0)
Word 6: Local month number, (1-12)
Word 7: Local day of month, (1-31)
Word 8: Local hour, (0-23)
Word 9: Time quality: (probable error)

0 - less than 1 msec
1 - 1 msec
2 - 5 msec
3 - 50 msec
4 - 500 msec
5 - internal time

Word 10: Number of days in GMT year, (365-366)
 Word 11: Number of days in local month, (28-31)
 Word 12: GMT - Local hours offset, (6-8)
 Word 13: SPARE

Initial value: 0

	<u>VARIABLE</u>	<u>SIZE</u>	<u>SET BY TASK(s)</u>	<u>USED BY TASK(s)</u>
m.	<u>LASTG</u>	1	ALM, ALO, DBI	ALM, ALO

Description - Free storage index of last alarm in queue.

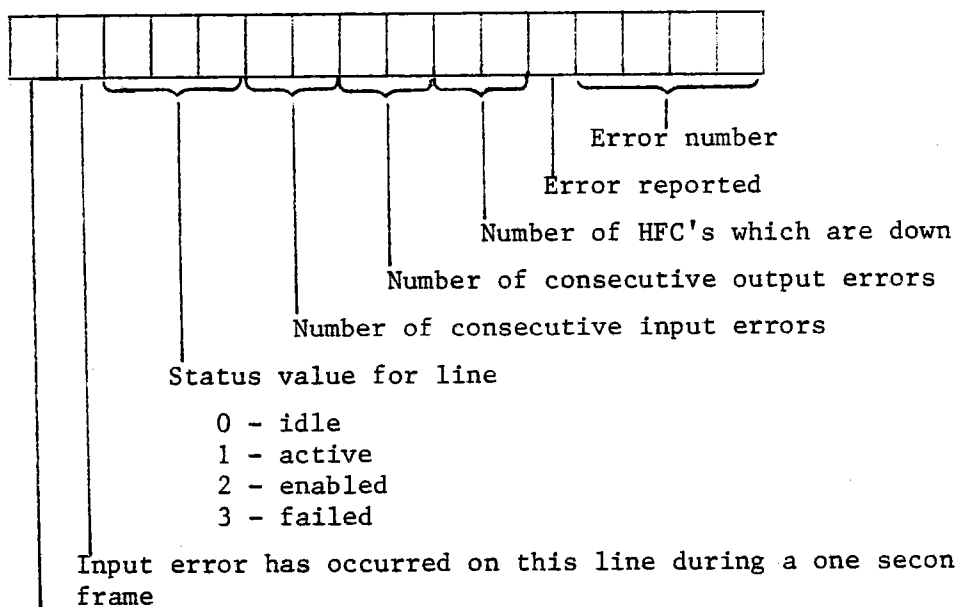
Initial value: 0

n.	<u>LINESG</u>	16	ALM, DBI, FCP	ALM, FCP
----	---------------	----	---------------	----------

Description - Communication line status.

Bit 0

Bit 15



Output error has occurred on this line during a one second frame

Initial value: #0800 for first 8 lines, 0 for remaining 8 lines.

Constraints and limitations - LINESG is designed for a maximum of sixteen communication lines.

o.	<u>LOKDEG</u>	1	MMI, DBI	BCS, CFO, DSK, MMI
----	---------------	---	----------	--------------------

Description - Defocus lock flag, which has the meaning:

ZERO = Defocus release command given
 NON-ZERO = Defocus command given

Initial value: 0

	<u>VARIABLE</u>	<u>SIZE</u>	<u>SET BY TASK(s)</u>	<u>USED BY TASK(s)</u>
p.	<u>LOKSTG</u>	1	DBI, MMI	BCS, CFO, DSK, MMI

Description - STOW-HI-WIND lock flag, which has the meaning:

ZERO = STOW-HI-WIND release command given
NON-ZERO = STOW-HI-WIND command given

Initial value: 0

q.	<u>PTRSG</u>	3	ALM, ALO, DBI	ALM, ALO
----	--------------	---	---------------	----------

Description - Free storage index for the next critical, non-critical, and system console output messages.

Initial value: 0

r.	<u>SUNPOG</u>	6	DBI, SUN	FCP
----	---------------	---	----------	-----

Variable type - INTEGER*4

Description: Sun position unit vector, where the double precision words have the meaning:

SUNPOG(1) = X component;

SUNPOG(2) = Y component;

SUNPOG(3) = Z component;

and each component is scaled B4.

Initial value: 0

s.	<u>SWOVG</u>	1	DBI, SWI, GET	SWI
----	--------------	---	---------------	-----

Description - Switchover to Backup HAC command word. If bit 15 is set, command the Backup HAC to initiate switchover.

aa.	<u>AIMOKG</u>	20	DBI, DSK	CMD, DSK
-----	---------------	----	----------	----------

Description - Array of 20 words corresponding to the 20 aim-point arrays, indicating whether the array can be used or not. The contents of the array words mean:

0 = disk data base aim-point array is not valid

1 = disk data base aim-point array is valid

Initial value: Determined from aimpoints that are loaded

bb.	<u>HFCS1G</u>	64	DBI, FCP	ALM, BHI
-----	---------------	----	----------	----------

	<u>VARIABLE</u>	<u>SIZE</u>	<u>SET BY TASK(s)</u>	<u>USED BY TASK(s)</u>
dd.	<u>MODEG</u>	12	DBI, STS	CMD, MMI, STA, STS

Description - Number of heliostats in each of the 12 modes.

Where the contents of MODEG have the Meaning.

- WORD 0: Number of HCs in TRACK
- WORD 1: Number of HCs in STANDBY
- WORD 2: Number of HCs in BCS
- WORD 3: Number of HCs in TRANSITION
- WORD 4: Number of HCs in STOW
- WORD 5: Number of HCs in ALT1STOW
- WORD 6: Number of HCs in ALT2STOW
- WORD 7: Number of HCs in MARK
- WORD 8: Number of HCs in DIRECTED POSITION
- WORD 9: Number of HCs in WASH
- WORD 10: Number of HCs in INITIALIZATION
- WORD 11: Number of HCs in OFFLINE

Initial value: 0

ee.	<u>SEQLSG</u>	16	CMD, DBI, SEQ	CMD, SEQ
-----	---------------	----	---------------	----------

Description - Sequence usage pool; unused entries are negative integers corresponding to the element number. While used entries are positive values corresponding to the element number.

Initial value: -1, -2, ..., -16

ff.	<u>SEQNMG</u>	1	CMD, SEQ, DBI	SEQ, SWI
-----	---------------	---	---------------	----------

Description - Active sequences counter; it reflects the number of positive numbers in the SEQLSG array.

gg.	<u>SPARE5(1)</u>	1	MMI	STS
-----	------------------	---	-----	-----

Description - Set by MMI for communication with STS.

Initial value: 0

hh.	<u>SPARE5(2)</u>	2	MMI	STS
-----	------------------	---	-----	-----

Description - Set by MMI for communication with STS.

Initial value: 0

ii. SPARE5(3) 1 FCP SEQ, FCP

Description - FCP simulation word. If bit 15 is set, FCP is in the simulation mode.

Initial value: \emptyset .

jj. SPARE5(4) 1 CSO CSO, CSI

Description: CSO sets bit 15 to prevent CSI from outputting during ALARM and STATUS outputs to the ISC.

Initial value: \emptyset .

kk. SPARE5(5) 1 EXI EXI, EXO

Description - Bit 15 is set by EXI when OCS is communicating with the HAC. Bit 14 is set by EXI when DAS is communicating with the HAC. Either EXI or EXO can reset these bits.

Initial value: \emptyset .

ll. SPARE5(6) - SPARE5(9) Unused

mm. SPARE5(10) 1 SEQ SEQ

Description - SEQ uses this bit for internal communication.

Initial value: \emptyset .

3.3.1.4.1.6 Global common area COM1S6

COM1S6 utilizes 4 pages of global common and is transferred to the backup HAC every second. Unless otherwise noted, variable type is INTEGER*2 and the size is in words.

a. SEQTPG 1 DBI, SEQ SEQ

Description - Time-link pointer for active sequences controlled by SEQ. It points to that cell (29 words) in the ACTVSG array which is to be visited next for sequence processing. Range is 1-16.

Initial value: 0

<u>VARIABLE</u>	<u>SIZE</u>	<u>SET BY TASK(s)</u>	<u>USED BY TASK(s)</u>
b. SEQAVG	1	DBI, SEQ	SEQ

Description - Next available cell in array ACTVSG for adding active sequences. Range is 1-16.

Initial value: 1

c. WAITSG	345 (15x23)	DBI, SEQ	SEQ
-----------	----------------	----------	-----

Description - Array used to store sequences waiting for all corridors to become free. A cell is 23 words long. See Figure 3.2.2-38 for format.

Initial value: 0

d. ACTVSG	464 (16x29)	DBI, SEQ	SEQ
-----------	----------------	----------	-----

Description - Array used to store active sequences (heliostats are moving). A cell is 29 words long. See Table 3.2.2-VII for format.

Initial value: 0 except the 4th word of Ith cell contains value I + 1 for cells 1-15; 4th word of cell 16 is 0.

e. There are 213 spare words of global common COM1S6.

3.3.1.4.1.7 Global common area COM8S1

COM8S1 utilizes 40 pages of global common and is transferred to the Backup HAC every eight seconds. Unless otherwise noted, variable type is INTEGER*2 and the size is in words.

<u>AIMPTG</u>	10240 (5x2048)	DBI, GET	CMD
---------------	-------------------	----------	-----

Description - Aim-point array currently in use, and the order is based on ascending HC order (five words/HC)

WORD 0

WORD 4

X	Y	Z	ARRAY #

Initial value: 1st valid aim-point array from disk file AIM

Constraints and limitations - AIMPTG is designed to maintain aim-point coordinates for up to 2048 heliostats.

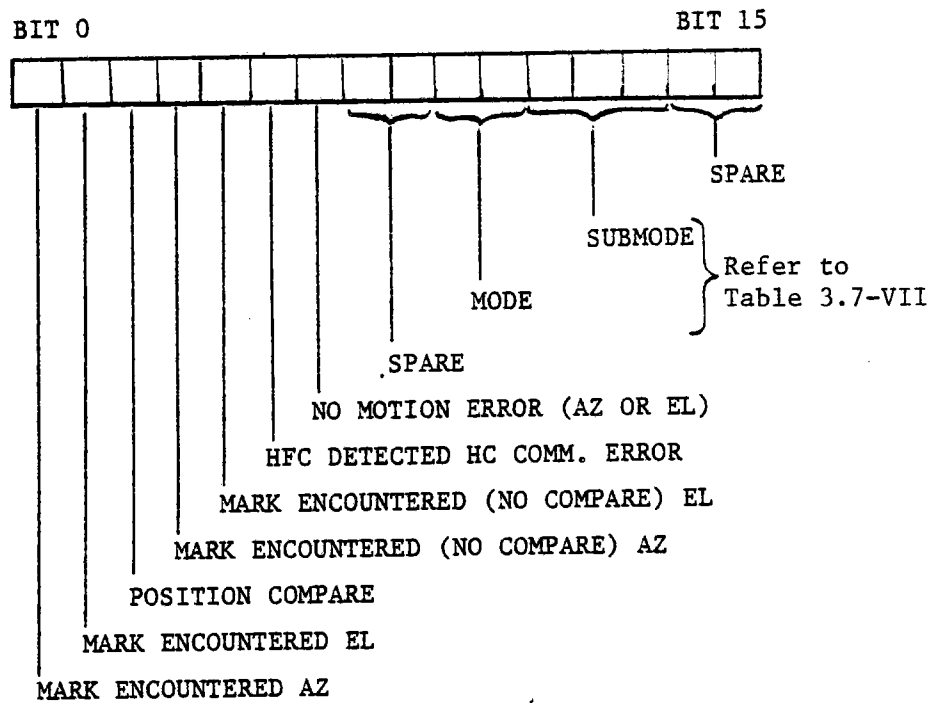
3.3.1.4.1.8

Global common area COM8S2

COM8S2 utilizes 16 pages of global common and is transferred to the Backup HAC every eight seconds. Unless otherwise noted, variable type is INTEGER*2, and the size is in words.

<u>VARIABLE</u>	<u>SIZE</u>	<u>SET BY TASK(s)</u>	<u>USED BY TASK(s)</u>
a. HCST1G	2048	DBI, FCP	ALM, BHC, CMD, GET, SEQ, STA, STS

Description - HC status

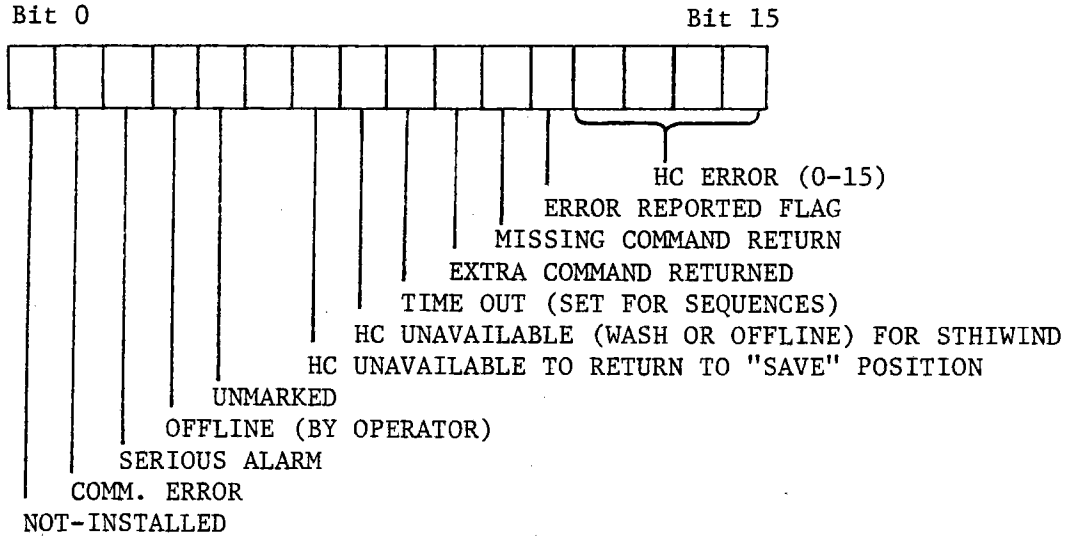


Initial value: 0

Constraints and limitations - HCST1G is designed to maintain status for a maximum of 2048 heliostats.

<u>VARIABLE</u>	<u>SIZE</u>	<u>SET BY TASK(s)</u>	<u>USED BY TASK(s)</u>
b. <u>HCST2G</u>	2048	ALM, CMD, DBI, FCP	ALM, BHC, CMD, STA, STS

Description - HC derived status



Initial value: #8000 if not installed otherwise #0800
unmarked

Constraints and limitations - HCST2G is designed to maintain the derived status for a maximum of 2048 heliostats.

3.3.1.4.1.9

Global common area COM8S3

COM8S3 utilizes 8 pages of global common and is transferred to the Backup HAC every eight seconds. Unless otherwise noted, variable type is INTEGER*2 and the size is in words.

HCST3G

2048

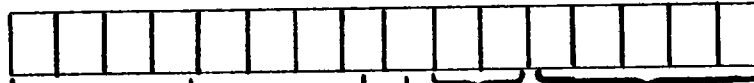
BCS, CMD, SEQ, DBI

BCS, BHC, CMD, SEQ

Description - Corridor, BCS, and sequence assignments

BIT 0

BIT 15



CORRIDOR ASSIGNMENT

0: NOT ASSIGNED

1-8: ASSIGNMENT

9-15: UNDEFINED

SEQUENCE # (DYNAMIC)

0: HC NOT ASSIGNED

1-16: ASSIGNMENT

BCS TARGET ASSIGNMENT

0: SOUTH

1: WEST

2: NORTH

3: EAST

HC INVOLVED IN AUTO BCS OPERATIONS

HC IN STOW SEQUENCE

Initial value: corridor and BCS assignments

Constraints and limitations - HCST3G is designed to maintain HC assignments for a maximum of 2048 heliostats. Bit assignments for the BCS target limit the total to a maximum of four. Bit assignments for the corridor assignment limit the total to a maximum of 16.

3.3.1.4.1.10 Global common area COM8S4

COM8S4 utilizes eight pages of global common and is transferred to the Backup HAC every eight seconds. Unless otherwise noted, variable type is INTEGER*2 and the size is in words.

<u>VARIABLE</u>	<u>SIZE</u>	<u>SET BY TASK(S)</u>	<u>USED BY TASK(S)</u>
PWHC1G	2048	DBI	CMD

Description - Last-reported heliostat status (HCST1G) when field power loss processing occurs.

Initial value: 0

Constraints and limitations - PWHC1G is designed to maintain power lost status for a maximum of 2048 heliostats.

3.3.1.4.1.11 Global common area COMIN1

COMIN1 utilizes eight pages of global common and is transferred to the Backup HAC only at initialization time. Unless otherwise noted, variable type is INTEGER*2 and the size is in words.

<u>VARIABLE</u>	<u>SIZE</u>	<u>SET BY TASK(S)</u>	<u>USED BY TASK(S)</u>
HC2MDG	2048	DBI	ALO, BCS

Description - Mapping array to convert from the HAC internal numbering scheme to the MDAC external numbering scheme. The HAC array element number contains the MDAC number.

Initial value: HAC/MDAC number correspondence

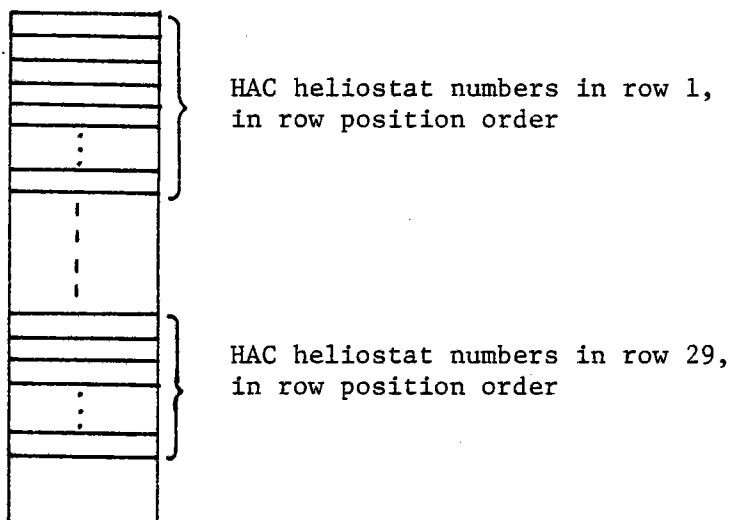
Constraints and limitations - HC2MDG is designed to maintain a maximum of 2048 heliostat numbers.

3.3.1.4.1.12 Global common area COMIN2

COMIN2 utilizes eight pages of global common and is transferred to the Backup HAC only at initialization time. Unless otherwise noted, variable type is INTEGER*2 and the size is in words.

<u>VARIABLE</u>	<u>SIZE</u>	<u>SET BY TASK(S)</u>	<u>USED BY TASK(S)</u>
<u>MD2HCG</u>	2048	DBI	BCS, MMI

Description - HAC heliostat numbers indexed through the array MDNPRG (MDAC number per row) resulting in a mapping from the MDAC numbering scheme to the HAC numbering scheme.



Initial value: MDAC/HAC number correspondence

Constraints and limitations - MD2HCG is designed to maintain a maximum of 2048 heliostat numbers.

3.3.1.4.1.13 Global common area COMIN3

COMIN3 utilizes seventeen pages of global common and is transferred to the Backup HAC only at initialization time. Unless otherwise noted, variable type is INTEGER*2 and the size is in words.

<u>VARIABLE</u>	<u>SIZE</u>	<u>SET BY TASK(S)</u>	<u>USED BY TASK(S)</u>
a. <u>HCMAPG</u>	2109	CMD, DBI, MMI	CMD, MMI

Description - MMI - CMD interface buffer for heliostats involved in a command. MMI passes CMD heliostat numbers for the command and CMD returns HCMAPG with the success rate from processing the command. This array is used in conjunction with CPPG.

Format: 2 types:

1. Multiple blocks

```

HCMAPG (1):  -N1           #HC's in first block
              (2):  HC#           N1 HC's in pecking order
              ⋮
              (N1+2): -N2           #HC's in second block
              HC#           N2 HC's in pecking order
              ⋮
NB
(N1 + NB+1): -999910       End of Data
i=1
  
```

where NB = # of blocks inherent in command addressing

2. Single block addressing:

```

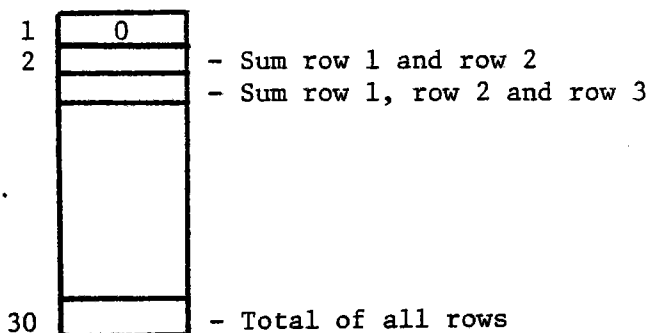
HCMAPG (1):  -Nt           Total #HC's in block
              (2):  HC# }           Nt HC's
              (Nt+1): }
              (Nt+2): -999910       End of Data
  
```

Initial value: 0

Constraints and limitations - HCMAPG is designed to maintain command control over a maximum of 2048 heliostats broken into a maximum of 60 subsets.

<u>VARIABLE</u>	<u>SIZE</u>	<u>SET BY TASK(S)</u>	<u>USED BY TASK(S)</u>
b. <u>MDNPRG</u>	30	DBI	BCS, DBI, MMI

Description - Each element contains the subtotal of the number of heliostats in each successive row of the field. MDNPRG is used to convert MDAC heliostats numbers into HAC heliostat numbers. The MDAC row number is the index into this array and the array element's content plus the MDAC row position is the index into the array MD2HCG.

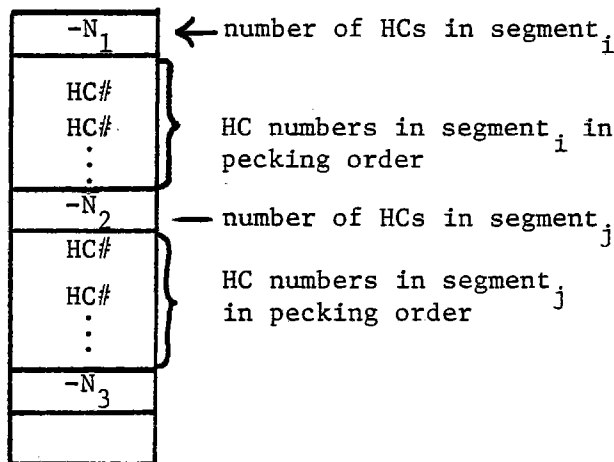


Initial value: input from record 17 of disk file DIN

Constraints and limitations - MDNPRG is designed to maintain a heliostat field with a maximum of 29 rows.

<u>VARIABLE</u>	<u>SIZE</u>	<u>SET BY TASK(S)</u>	<u>USED BY TASK(S)</u>
c. <u>SEGMPG</u>	2108	DBI	DBI, GRF, MMI

Description - HAC HC numbers ordered by segment number and subordered by pecking order. Each segment is preceded by the negative of the number of HCs in that segment. SEGMPG contains information only on segments that have installed heliostats. This array is used in conjunction with SEGPTG.

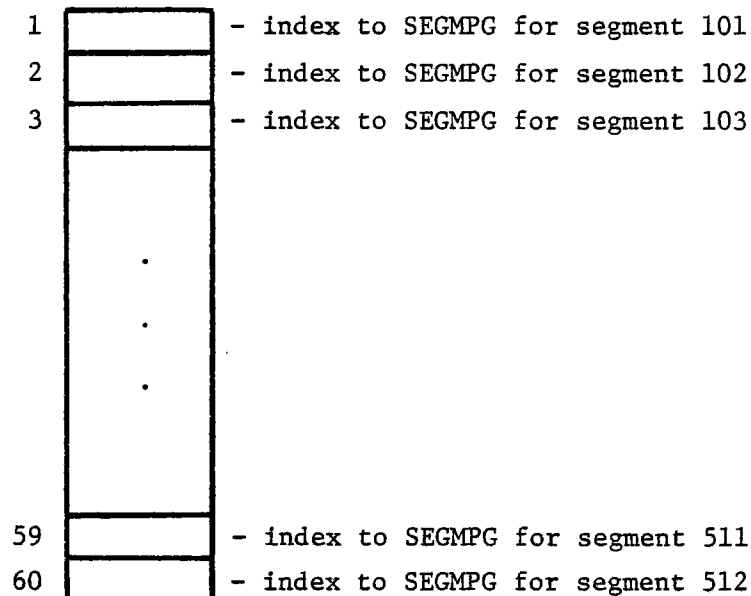


Initial value: input from records 34-50 of disk file DIN

Constraints and limitations - SEGMPG is designed to maintain a field of 2048 heliostats subdivided into a maximum of 60 segments

d. <u>SEGPTG</u>	60	DBI	DBI, GRF, MMI
------------------	----	-----	---------------

Description - Pointers to the array SEGMPG indicating the start of the number of HCs in a desired segment. When the contents of an element in SEGPTG is zero, it means that segment does not have any heliostats installed.



Initial value: input from record 17 of disk file DIN

Constraints and limitations - SEGPTG is designed to maintain a field subdivided into a maximum of 60 segments.

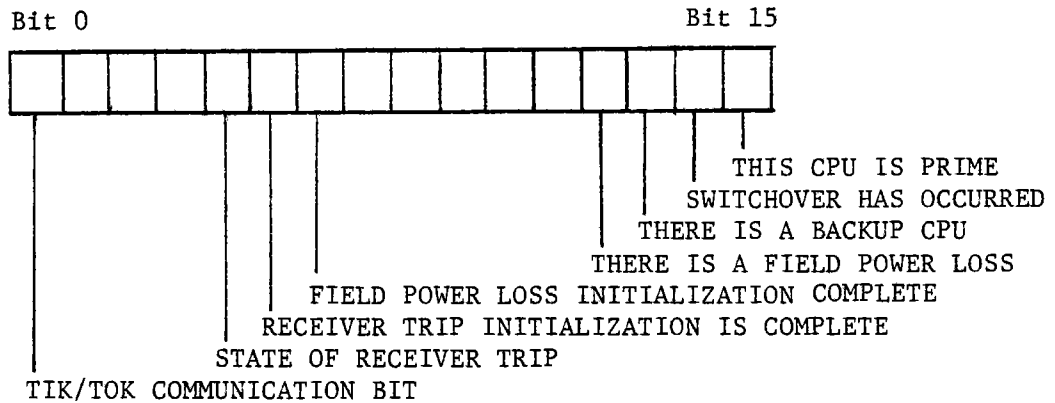
e. SPARI3 contains 45 words of spare for COMIN3

3.3.1.4.1.14 Global common area COMIN4

COMIN4 utilizes one page of global common and is transferred to the Backup HAC only at initialization time. Unless otherwise noted, variable type is INTEGER*2 and the size is in words.

<u>VARIABLE</u>	<u>SIZE</u>	<u>SET BY TASK(s)</u>	<u>USED BY TASK(s)</u>
a. <u>CPUSG</u>	1	DBI, SWI	CLK, DBI, TIK, TOK

Description - CPU peculiar status



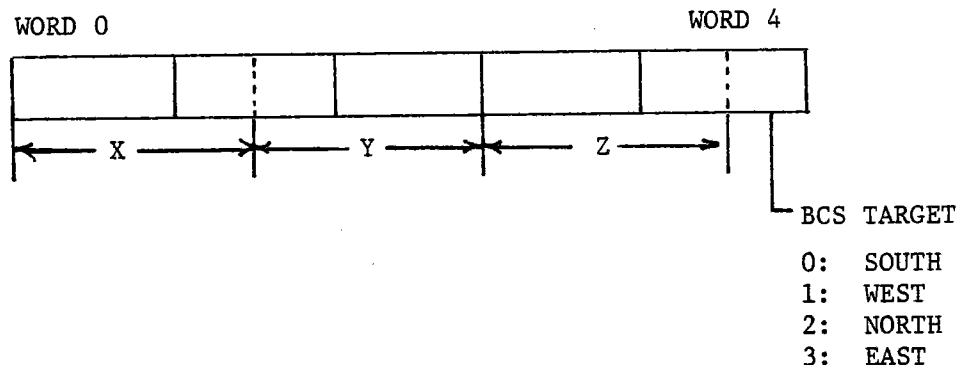
Initial value: determined from operator inputs to questions on system console

Constraints and limitations - The CPU peculiar status word CPUSG must never be transferred to the Backup HAC.

b. <u>BCSTGG</u>	20 (5x4)	DBI	BCS, CMD
------------------	-------------	-----	----------

Type - 24 bit fixed point, scaled B14

Description - BCS target in site reference orthogonal system



Initial value: input from record 50 of disk file DIN

Constraints and limitations - BCSTGG is designed to maintain BCS target coordinates for a maximum of four targets.

<u>VARIABLE</u>	<u>SIZE</u>	<u>SET BY TASK(S)</u>	<u>USED BY TASK(S)</u>
c. <u>CFABOG</u>	1	CFO, DBI, MMI	CFO

Description - Command file abort flag.
Initial value: 0

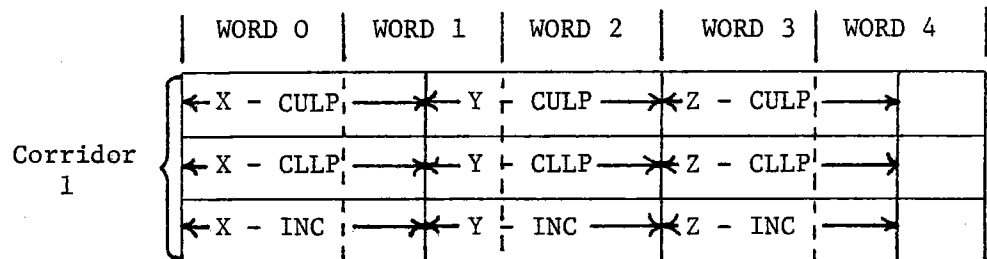
d. <u>CFWATG</u>	1	CFO, DBI, MMI	CFO
------------------	---	---------------	-----

Description - Delay execution of command file execution flag.
Initial value: 0

e. <u>CORRCG</u>	120 (5x3x8)	DBI	BHI, CMD, SEQ
------------------	----------------	-----	---------------

Type - 24 bit fixed point, scaled B14

Description - Corridor coordinates in site reference orthogonal system.

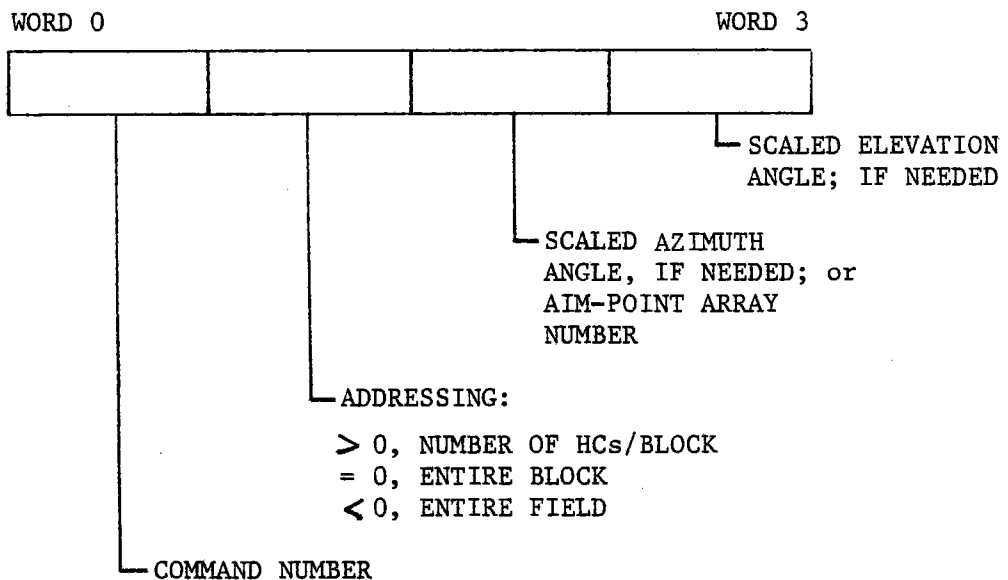


Initial value: input from record 51 of disk file DIN

Constraints and limitations - CORRCG is designed to maintain corridor coordinates for a maximum of eight corridors.

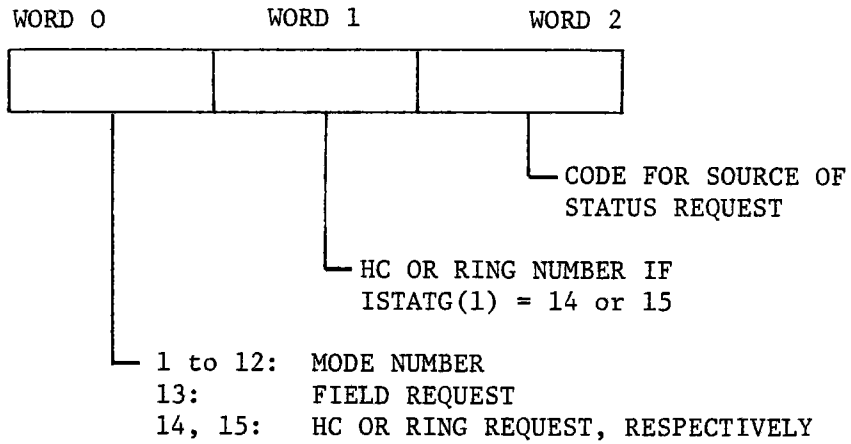
f. <u>CPPG</u>	4	DBI, MMI	CMO
----------------	---	----------	-----

Description - MMI - CMD interface command buffer



Initial value: 0

Description - MMI - STS interface status request indicator.



Initial value: 0

	<u>VARIABLE</u>	<u>SIZE</u>	<u>SET BY TASK(S)</u>	<u>USED BY TASK(S)</u>
k.	<u>SLATG</u>	4	DBI	SUN

Type - Double precision real

Description - Latitude of facility in degrees (38.4 degree north)

Initial value: 38.4 degrees

l.	<u>SLONGG</u>	4	DBI	SUN
----	---------------	---	-----	-----

Type - Double precision real

Description - Longitude of facility in degrees (-117.0 degrees west)

Initial Value: -117.0 degrees

m.	<u>TBUSYG</u>	2	All	-
----	---------------	---	-----	---

Variable type - INTEGER*4

Description - Tasks active word (1 = active, 0 = inactive)

Module	ALARMS			BCS MOD	CMDPRG					DBINIT		EXTINF			FLD COM	GRA PHC
Task	ALM	ALO	BCS	BHC	BHI	CMD	GET	SEQ	CLK	DBI	CSI	CSO	EXI	FCP	GRF	
Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Module	MANMIF			MAXIVM					STATUS		SUN VEC	MAXIVM				
Task	CFO	DSK	MMI	RTL	RTH	SWI	TOK	TIK	STA	STS	SUN	PLH	PLL			
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

The remaining variables in global common area COM1S5 are transferred to the Backup HAC every eight seconds.

Initial value: 0.

n. SPRIN4(1) 1 MMI CFO

Description: Used to hold the command file name from MMI.

Initial value: 0.

m. SPRIN4(2) 1 MMI CFO

Description: Set by MMI to indicate operator action to CFO.
 If set to 1 CFABORT has been entered.
 If set to 2 CFWAIT has been entered.
 If 0 normal command file processing.

Initial value: 0.

o. SPRIN4(3) - SPRIN4(26) Unused.

p. SPRIN4(27) 1 SWI SWI, SWIRG1, SWIRG8

Description: Bit 15 is set on by SWI while the one or eight second global common is being transferred across the HAC to HAC interface. Bit 15 is cleared by SWIRG1 or SWIRG8 when transfer has been complete.

Initial value: 0.

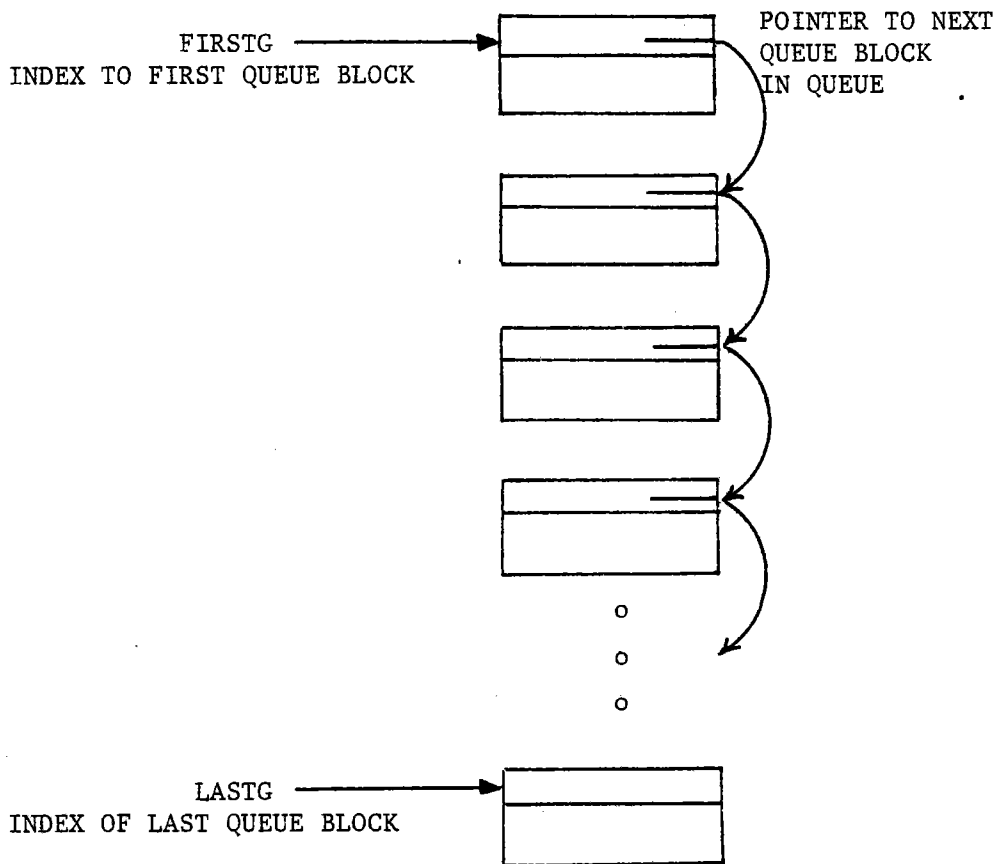
3.3.1.4.1.15 Free Storage (COMQUE)

COMQUE is a dynamic buffer, initialized by task DBI (via QINIT), and used to queue and dequeue messages for output. It is 4096 words in size, with a set of internal pointers monitored by submodules LEASE, FREE, ENQU, and DEQU.

The COMQUE global common area consists of the single COMQUE array IQUEUE.

<u>VARIABLE</u>	<u>SIZE</u>	<u>SET BY TASK(s)</u>	<u>USE BY TASK(s)</u>	<u>LOCATION</u>
<u>IQUEUE</u>	4096	EXI, CSI, MMI DBI, ALM, ALO CMO, GET, SEQ BCS	GRF, CSO, CFO ALM, ALO, GET SEQ, BCS	Global common COMQUE

Description - Free storage area for messages awaiting output.



3.3.1.4.1.16 Free Storage COMQU2

COMQU2 is a dynamic buffer, initialized by task DBI (via QINIT), and used to queue and dequeue command messages for the Command Processor Module. It is 4096 words in size (16 pages), with a set of internal pointers monitored by submodules LEASE, FREE, ENQU and DEQU

The layout of COMQU2 is similar to COMQUE in Section 3.3.1.4.1.14.

<u>VARIABLE</u>	<u>SIZE</u>	<u>SET BY TASK(S)</u>	<u>USED BY TASK(S)</u>
CMDQUE	4096	CMD, GET, SEQ	GET, SEQ

3.3.1.4.1.17 Disk Data Base

The Disk Data Base is composed of thirteen files residing on the fixed disk. A description of each file is given in the following:

- a. Disk File AIM - The AIM file contains the 20 aim-point arrays for the heliostat field. This file has 2048 records and each record is 100 words. Figure 3.3.1.4-1 provides a record layout description of the AIM file.
- b. Disk File AL1 - The AL1 file contains the Alternate 1 Stow azimuth and elevation angles. This file has 32 records and each record is 128 words. Figure 3.3.1.4-2 provides a record layout description of the AL1 file.

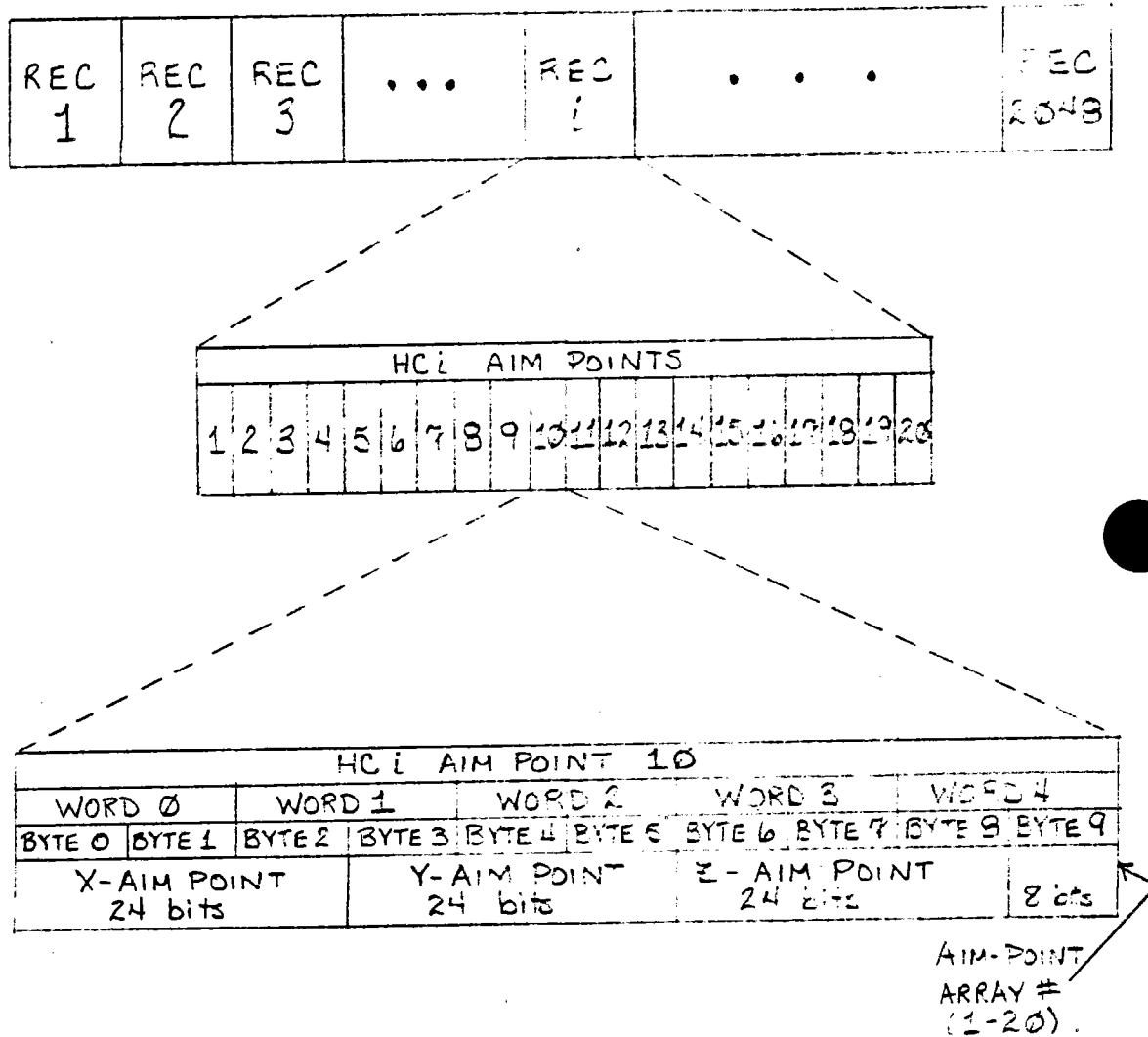


Figure 3.3.1.4-1 Format and Structure of the Aim-Point File (AIM)

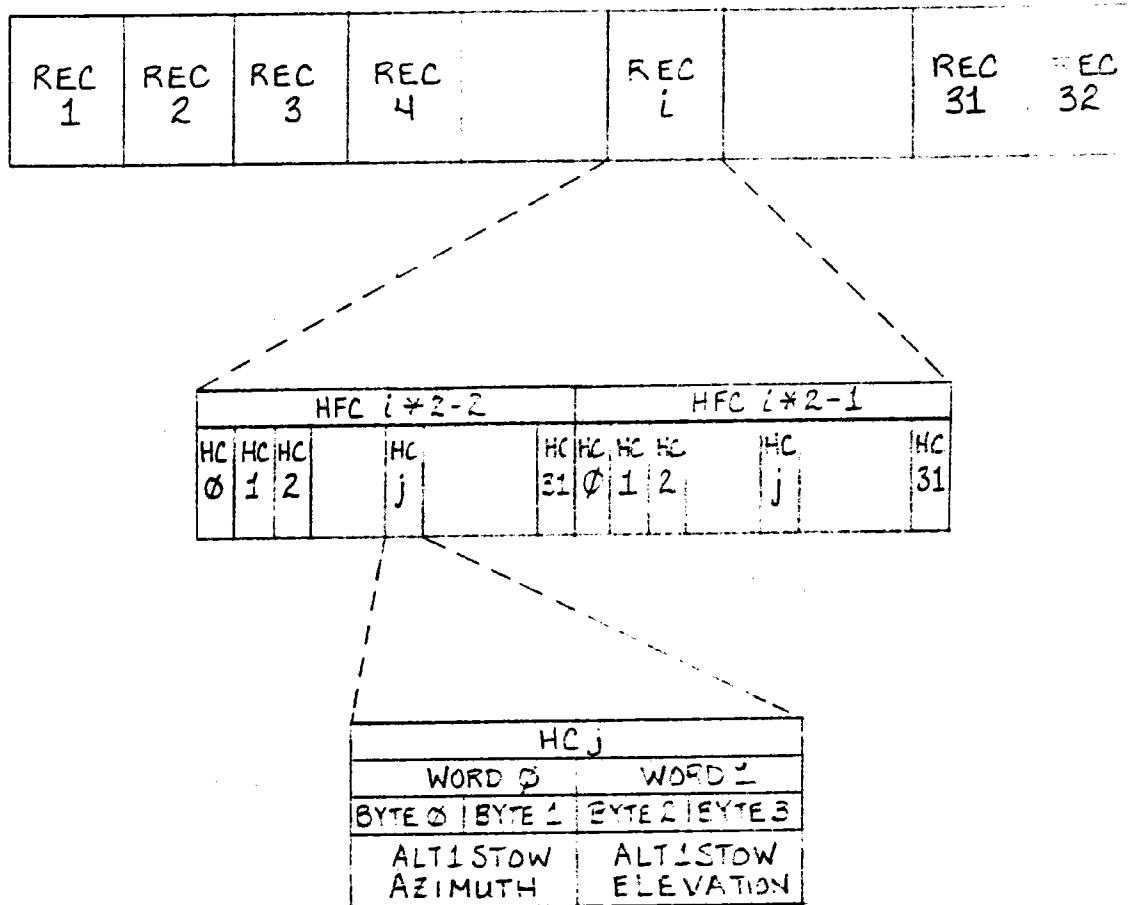


Figure 3.3.1.4-2 Format and Structure of the HC ALT1STOW Angles Disk File (AL1)

- c. Disk File AL2 - The AL2 file contains the Alternate 2 Stow azimuth and elevation angles. This file has 32 records and each record is 128 words. Figure 3.3.1.4-3 provides a record layout description of the AL2 file.
- d. Disk File BAM - The BAM file contains the alarm messages. This file has 255 records and each record is 64 bytes. Each sector contains four alarm messages (records).
- e. Disk File DIN - The DIN file contains five mapping arrays, the corridor coordinates, and the BCS coordinates. This file has 67 records and each record is 128 words. Figure 3.3.1.4-4 provides a record layout description of the DIN file.
- f. Disk File HCC - The HCC file contains the heliostat locations. This file has 64 records and each record is 160 words. Figure 3.3.1.4-5 provides a record layout description of the HCC file.
- g. Disk File HCB - The HCB file contains the heliostat bias azimuth and elevation angles. This file has 32 records and each record is 128 words. Figure 3.3.1.4-6 provides a record layout description of the HCB file.
- h. Disk File SAM - The SAM file contains the source alarm messages. This file has 255 records and each record is 128 words. The record contents are provided in Figure 3.3.1.4-7.
- i. Disk File AIM - The AIM file (words 123-128 for each record) contains the real-time status, tracking configuration status, perserving the HC, that are in Track or Standby. This file has 2 records and each uses 6 words per record. Figure 3.3.1.4-8 provides the record layout of disk file AIM.
- j. Disk File STO - The STO file contains the stow, azimuth and elevation angles. This file has 32 records and each record is 128 words. Figure 3.3.1.4-9 provides the record layout description of the STO file.
- k. Disk File WSH - The WSH file contains the Wash, azimuth and elevation angles. This file has 32 records and each record is 128 words. Figure 3.3.1.4-10 provides the record layout description of the WSH file.
- l. Disk File CFM - The CFM file contains command files in a SED directoried compressed source format. There may be up to 100 command files on this disk file.
- m. Disk File BCS - The BCS file contains up to three lists of HCs that are to be measured using the BCS automatic processing.

3.3.1.5 Interface Description

The global common data bases COMDAT, COMQU2, and COMQUE interface with tasks and submodules through FORTRAN INCLUDE statements and simple reference to those variables. Tables 3.3.1-I and 3.3.1-II shows cross-reference of all global common variables/arrays and the tasks which utilize them. Table 3.3.1-III cross-reference of all Disk Data Base files and the tasks which utilize them.

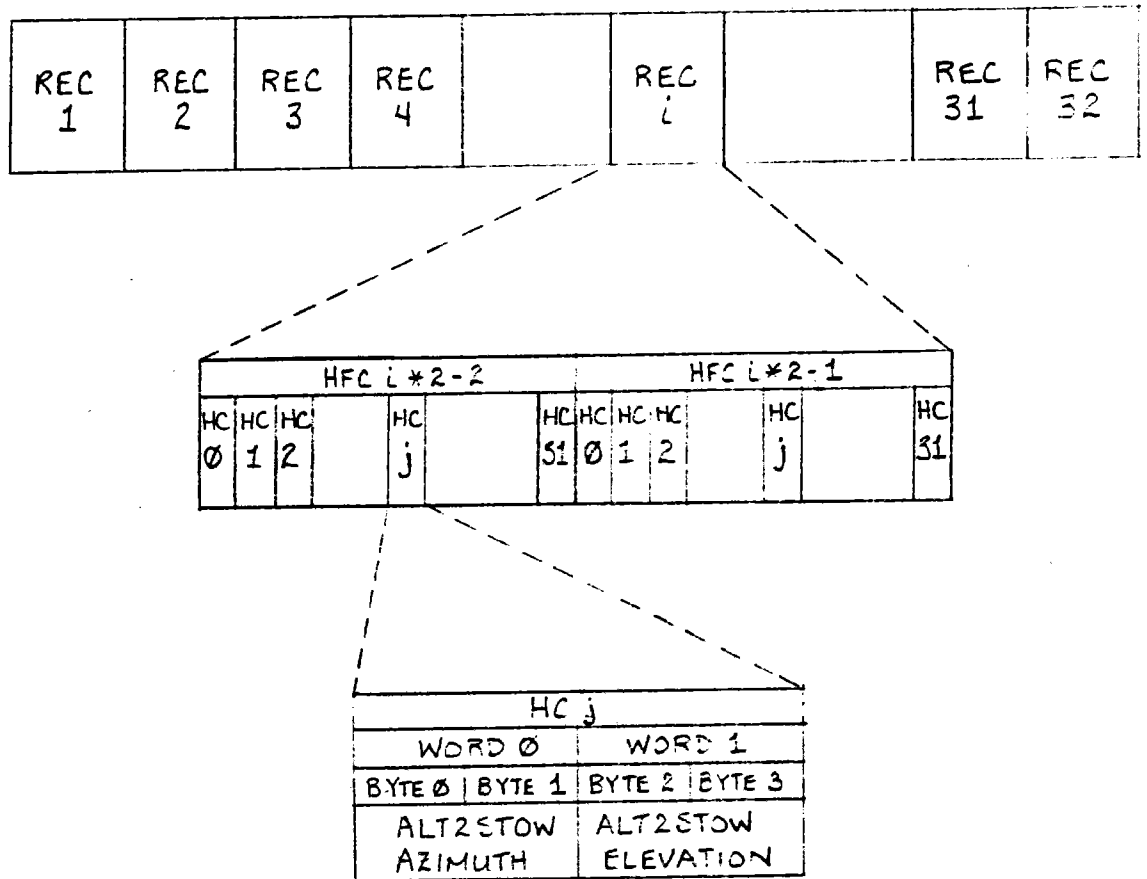


Figure 3.3.1.4-3 Format and Structure of ALT2STOW Angles Disk File (AL2)

REC 1	REC 2	REC 3	REC 4	REC 5	REC 6	REC 7	REC 8	REC 9	REC 10	REC 11	REC 12	REC 13	REC 67
-------	-------	-------	-------	-------	-------	-------	-------	-------	--------	--------	--------	--------	--------

W 0	W 1	W 2	W 3	W 4								W 126	W 127
-----	-----	-----	-----	-----	--	--	--	--	--	--	--	-------	-------

Records 1-16

These records are used to store the MDAC to HFCHC number mapping (MD2HCG), used in conjunction with the MDNPRG array. This file is filled from a-priori values whenever card Type 1 is input through task DIN, and is read from the disk to fill the global common array. The array is stored in sequential fashion with element 1 of the array in record 1, word zero, and element 2048 stored in record 16, word 127.

Record 17 contains the MDAC number per row (MDNPRG) array using the first 30 words.

Record 17

W 0	W 1	...	W 29		W 50	W 51	...	W 108	W 109		
Storage area for MDNPRG					Storage area for SEGPTG array, 60 words						

Records 18-33 contain the HC2MDG array. Each record contains 128 words of the HC2MDG array in ascending order

Record 34-50 contain the SEGMPG array stored in ascending elements manner. Record 50 contains the last 60 words of the 2108 word array.

Record 50

W 0	W 1	last of SEGMPG array	W 58	W 59		W 70		W 92	W 93	
						BCS target location				

Figure 3.3.1.4-4 Format and Structure of File DIN

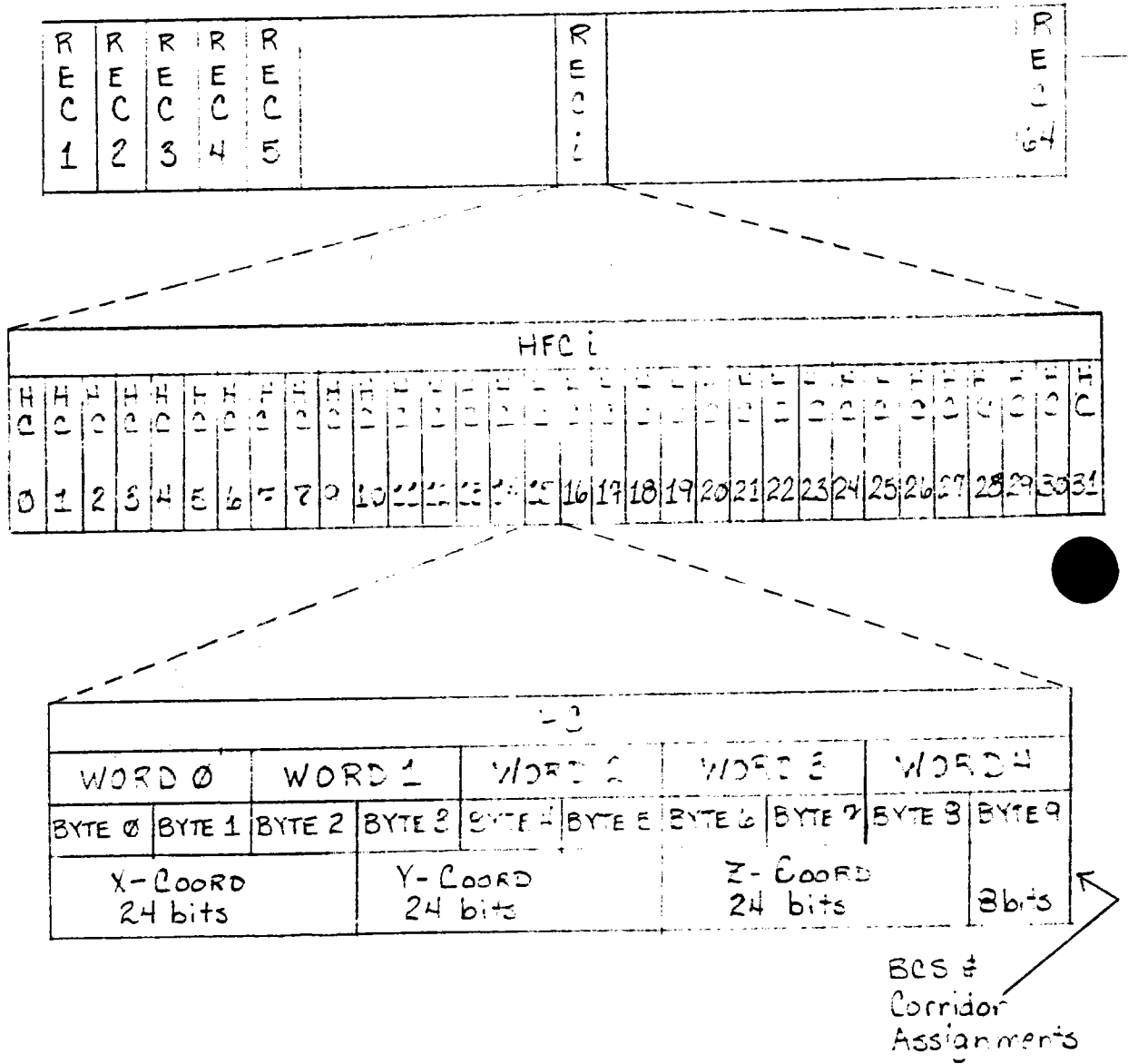


Figure 3.3.1.4-5 Format and Structure of the HC Locations Disk File (HCC)

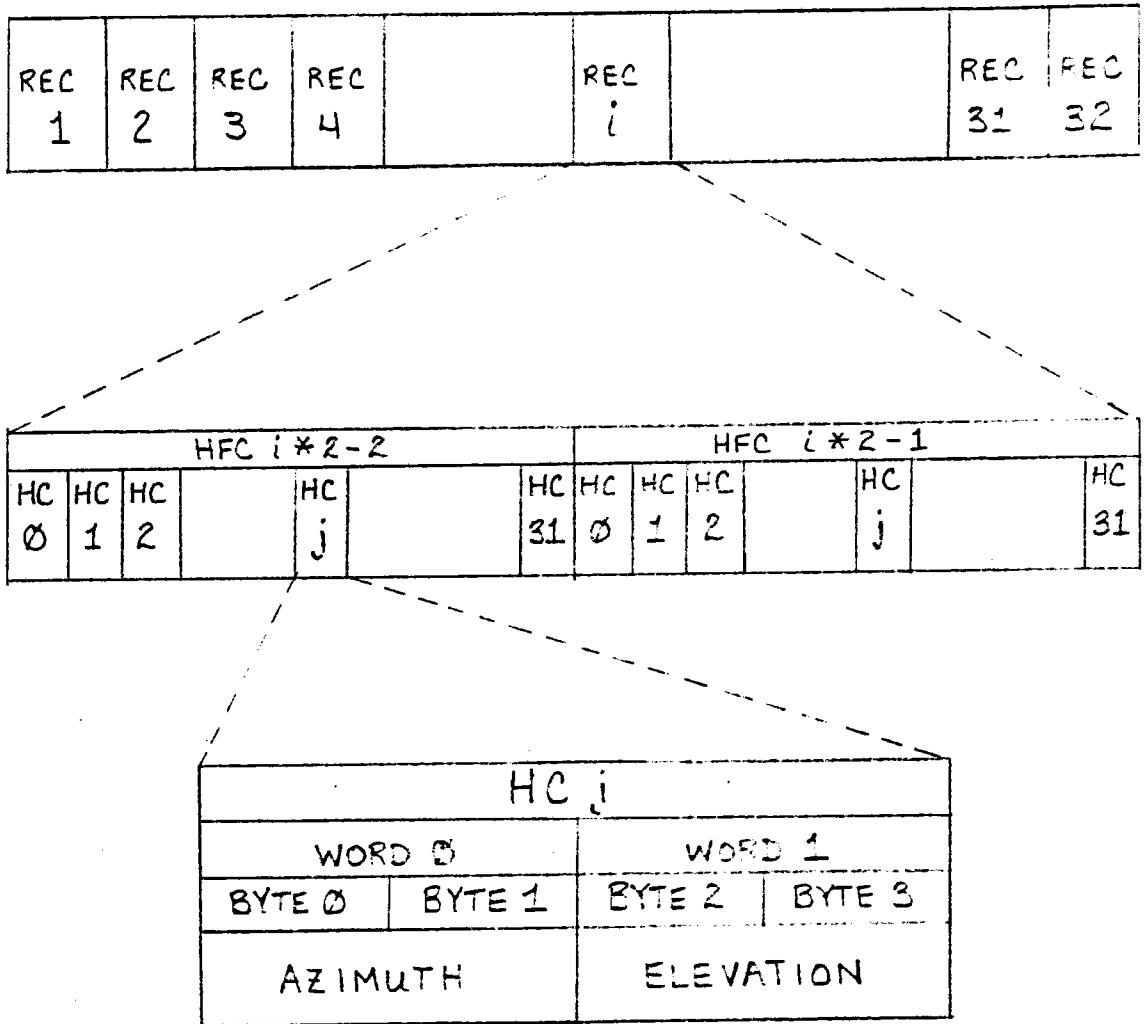


Figure 3.3.1.4-6 Format and Structure of the Bias Disk File (HCB)

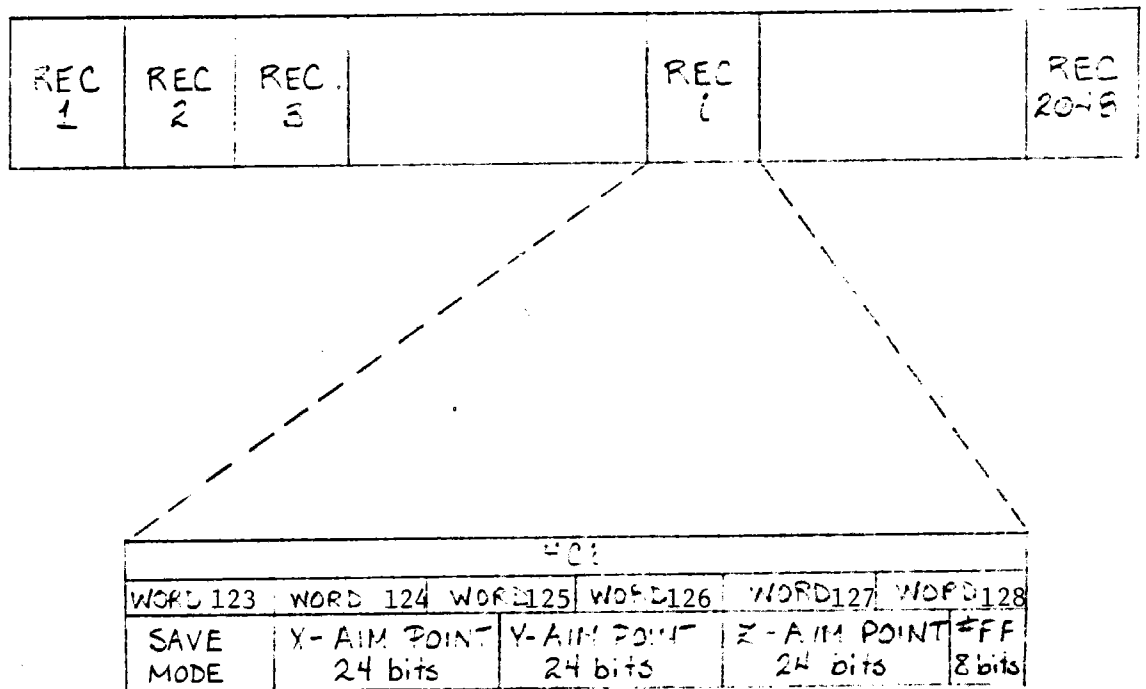
<u>LINE</u>	<u>ALARM NO.</u>	<u>MESSAGE</u>
1	ALARMDATA	REV.003 12/09/81
2	1	'MULTIPLE LINE ERRORS ',A2,' LINE',4A4
3	2	'MULTIPLE HFC ERRORS; LINE ',A2,'; HFC ',4A4
4	3	'HC ERRORS: HFC ',A2,'; HC ',3(A4,A4,A1,'/',A1),A2
5	4	'EL MARK ENCOUNTERED: HFC 'A2,'; HC 'A4,'; EL=#'Z4,', BIAS=#'Z4
6	5	'AZ MARK ENCOUNTERED: HFC 'A2,';HC 'A4,'; AZ=#'Z4,', BIAS=#'Z4
7	10	'MISSING COMMAND RETURN; HFC 'A2,'; HC 'A4
8	11	'NO MOTION ERROR (AZ OR EL); HFC 'A2,'; HC 'A4
9	12	'HC UNAVAILABLE (WASH/OFFLINE) TO STHWIND: HFC 'A2,'; HC 'A4
10	13	'HC CANNOT RETURN TO SAVE POSITION: HFC 'A2,'; HC 'A4
11	14	'SEQUENCE COMMAND TIMED OUT; HFC 'A2,'; HC 'A4
12	15	'EXTRA COMMAND RETURNED; HFC 'A2,'; HC 'A4
13	16	'HC MARK POSITION OUTSIDE TOLERANCE; HFC 'A2,'; HC 'A4
14	17	'HFC DETECTED HC/HFC COMM ERR--NO RESPONSE: HFC ',A2,'; HC ',A4
15	18	'HC LOST POWER - NEEDS LOADED: HFC ',A2,'; HC ',A4
16	19	'HC DETECTED HFC/HFC COMM ERROR; HFC ',A2,'; HC ',A4
17	20	'HFC DETECTED HAC/HFC COMMUNICATIONS ERROR; HFC ',A2
18	22	'HAC TO HFC COMMUNICATIONS OUTPUT TIMED OUT; HFC ',A2
19	23	'HAC TO HFC COMMUNICATIONS INPUT TIMED OUT; HFC ',A2
20	24	'INVALID DATA RECEIVED FROM HFC 'A2
21	25	'HFC DID NOT RECEIVE LAST COMMAND; HFC ',A2
22	26	'FIRMWARE ERROR - HFC ',A2,' STATUS = ',Z4
23	30	'LINE COMMUNICATIONS OUTPUT ERROR; LINE 'A2
24	31	'LINE COMMUNICATIONS INPUT ERROR; LINE 'A2
25	32	'LINE FAILURE AND SWITCHOVER; LINE ',A2
26	40	'FIELD POWER LOSS'
27	41	'COMMUNICATION FAILURE WITH FIELD'
28	50	'OPERATOR CONSOLE MALFUNCTION'
29	51	'CRT MALFUNCTION'
30	56	'DISK READ ERROR: TASK ',A4,'; FILE ',A4
31	57	'DISK WRITE ERROR: TASK ',A4,'; FILE ',A4
32	60	'POWER LOSS SIGNAL RECEIVED'
33	61	'POWER SIGNAL INDICATES FIELD POWER IS READY FOR OPERATION'
34	62	'RECEIVER TRIP SIGNAL RECEIVED'
35	63	'RECEIVER SIGNAL INDICATES RECEIVER IS READY FOR OPERATION'
36	89	'RESTORE COMMAND ENCOUNTERED A HELIOSTAT IN THE WRONG MODE'
37	90	'CHANGE IN CLOCK ACCURACY STATUS FROM ',2A4,' TO ',2A4
38	99	'BCS TERMINATED BY OPERATOR'
39	100	'AUTO-BCS COMMANDED HELIO ',A4,' TO ',A4
40	101	'NO TARGETS AVAILABLE FOR AUTO-BCS MEASUREMENTS'
41	102	'OK NOT RECEIVED FROM OCS FOR BCS PROCESSING'
42	103	'TARGET ', A1, ' NOT AVAILABLE FOR AUTO-BCS'
43	104	'INVALID HC # ENCOUNTERED (HC # = ', A4, ') DURING AUTO-BCS'
44	105	'HC ', A4, ' NOT IN LEGAL MODE FOR AUTO-BCS'
45	106	'BLOCKING HC ', A4, ' NOT IN LEGAL MODE'
46	107	'ABORT HC MEASUREMENT DUE TO OCS TIME OUT HC = ', A4
47	108	'BCS MEASUREMENTS FINISHED FOR FILE ', 5A2
48	109	'ENCOUNTERED ERRONEOUS OCS DATA IN MSG # ', A2
49	110	'INVALID MSG DURING AUTO-BCS RECEIVED FROM OCS MSG = ', A2
50	111	'UNEXPECTED MSG DURING AUTO-BCS FROM OCS MSG = ', A2

Figure 3.3.1.4-7 Alarm Messages Stored in File MSGS/DBI
(Continued on next page)

<u>LINE</u>	<u>ALARM NO.</u>	<u>MESSAGE</u>
51	112	'BCS WRITE DID NOT COMPLETE ON OCS DATA LINE (CORE DEV=' ,A1,')'
52	113	'BIAS CALCULATIONS AND UPDATES FINISHED FILE ', 5A2
53	114	'BCS TERMINATED DUE TO LACK OF OCS RESPONSE'
54	115	'CENTROID ALARM; HC ',A4,' HORIZ ',F6.2, ' FT VERT ',F6.2,' FT'
55	116	'POWER TOO LOW; HC ',A4, ' POWER ',F5.2, ' K WATTS'
56	117	'BEAM OFF TARGET ',A1, ' HC ', A4
57	118	'LOW LEVEL BEAM ON TARGET ',A1, ' HC ', A4
58	119	'EOF ENCOUNTERED BY TASK ', A4, ' READING FILE ', A4
59	120	'EOF ENCOUNTERED BY TASK ', A4, ' WRITING TO FILE ', A4
60	121	'BCS MEASUREMENTS STARTED USING FILE ', 5A2
61	122	'ABORT BCS HC MEASUREMENT DUE TO HC TIME OUT HC = ', A4
62	123	'BCS CALCULATIONS AND UPDATES STARTED FILE ',5A2
63	124	'TIME-OUT FOR BCS-12 MESSAGE (OFFSET DATA) FOR HC = ', A4
64	125	'AUTO-BCS IS NOW USING TARGET ', A1
65	126	'AUTO-BCS RELEASED TARGET ', A1
66	127	'AUTO-BCS DETECTED OCS DATA LINK ERROR'
67	150	'OCS DATA LINK OPERATIONAL'
68	151	'OCS DATA LINK NON-OPERATIONAL'
69	152	'DAS DATA LINK OPERATIONAL'
70	153	'DAS DATA LINK NON-OPERATIONAL'
71	170	'HAC FAILOVER HAS OCCURRED BACKUP HAC NOW PRIME'
72	171	'BACKUP HAC HAS FAILED--NO BACKUP CAPABILITY EXISTS'
73		

LIST

Figure 3.3.1.4-7 Alarm Messages Stored in File MSGS/DBI(Continuation)



Where Save Mode = 0 : other
 > 0 : HC was at Track and its Aim point is in words 1-5
 < 0 : HC was at Stand by

Figure 3.3.1.4-8 Tracking Configuration Save File (AIM)

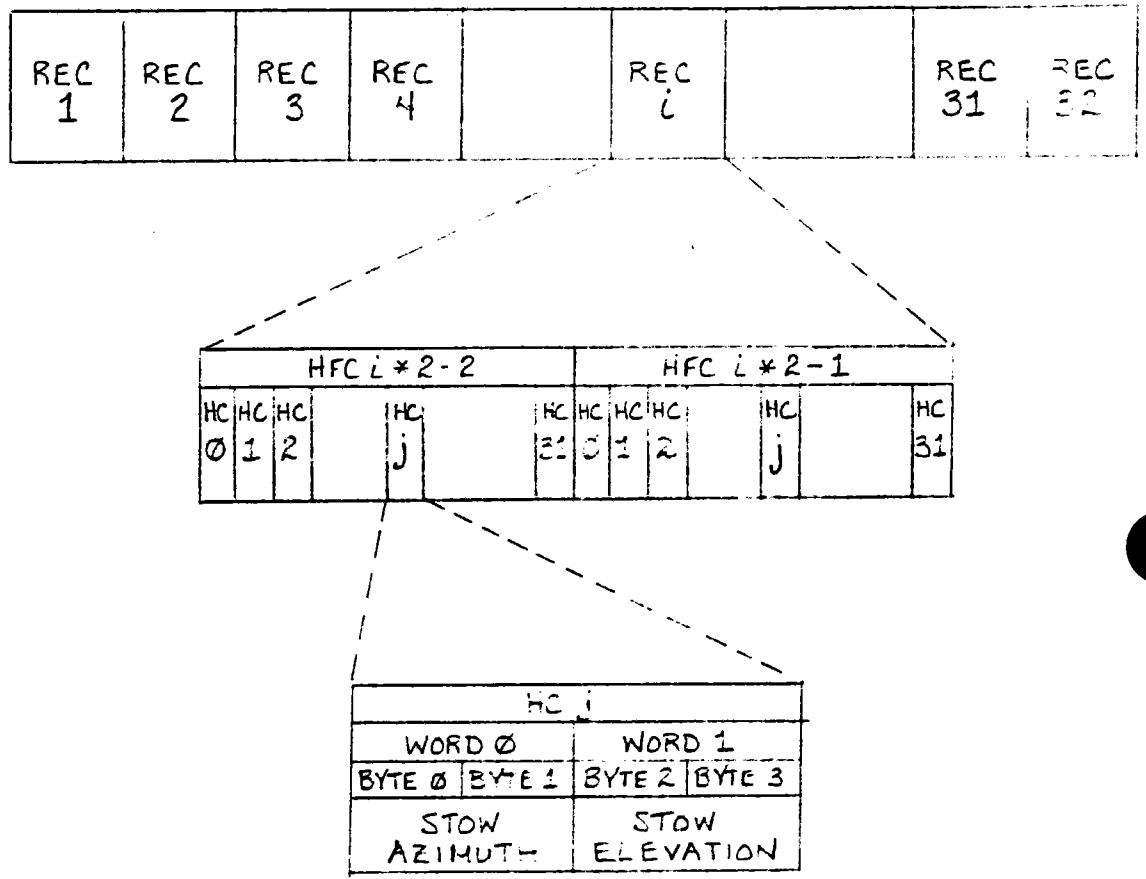


Figure 3.3.1.4-9 Format and Structure of HC Stow Angles Disk File (STO)

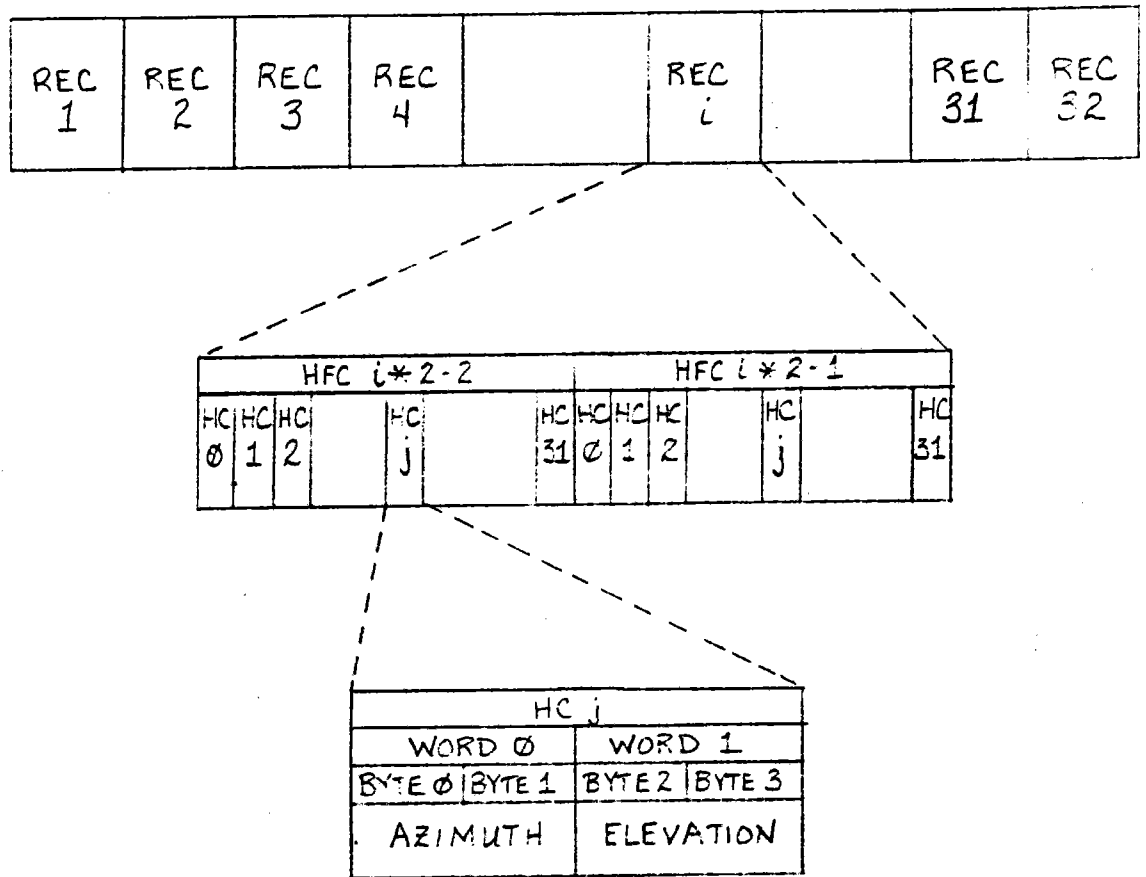


Figure 3.3.1.4-10 Format and Structure of the Wash Angles Disk File (WSH)

3.3.1.6

Test Requirements

Verification of global common data bases COMDAT and COMQUE and Disk shall be achieved through utilization of "snapshot" techniques, indicating global common contents. Modification of global common through independent test routine simulation shall be accomplished to insure proper design and efficient use.

COMDAT VARIABLE	ALM	ALO	BCS	BHC	BHI	CMD	GET	SEQ	CLK	DBI	DIN	CSI	CSO	EXI	FCP	GRF	CFO	DSK	MMI	RTL	RTH	SWI	TCK	TIK	STA	STS	SUN
AIMBSG										S								S	S/U								
ALRMSG	U/S	S/U								S																	
AZIMG	U			U		U	U			S					S												
BCSBYG			S/U			S/U				S																	
BCSHLG			S/U							S																	
BCSHSG			S/U							S																	
BCSTAG			S/U							S																	
CMDBFG				S	S					S					U												
CORRSG				S/U				S/U		S																	
CORSQG								S/U		S																	
COUNTG	S/U	S								S																	
CRTWRG										S		S/U	S/U						S							S	
CURHSG	U									S					S/U												
ELEVG	U		U	U		U				S					S												
EMCC1G				S/U		S	S	S		S																	
EMSEQG			U			S/U		S		S							U	U	U								
FIRSTG	S/U	S/U								S																	
FLDSTG	S/U									S					S/U												
GRSTSG										S						U										S	
GTIMEG		U	U																U				S	S	U	U	U
HCCMDG			S/U	S/U		S	S	S/U		S																	
HCDATG			S/U	U		S	S	S		S																	
LASTG	S/U	S/U								S																	

S = Set
U = Used

Table 3.3.1-I Task Utilization of Global Common COMDAT
(1 Sec Transfer Frequency to Backup HAC)

COMDAT VARIABLE	ALM	ALO	BCS	BHC	BHI	CMD	GET	SEQ	CLK	DBI	DIN	CSI	CSO	EXI	FCP	GRF	CFO	DSK	MMI	RTL	RTH	SWI	TOK	TIK	STA	STS	SUN
LINESG	S U									S					S U												
LOKDEG			U							S							U	U	S U								
LOKSTG			U							S							U	U	S U								
PTRSG	S U	S U								S																	
SUNPOG										S					U												S
SWOVG										S												S U					
TBUSYG	S	S	S	S	S	S	S	S	S	S		S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S

S = Set
U = Used

Table 3.3.1-I Task Utilization of Global Common COMDAT (con't.)
(1 Sec Transfer frequency to Backup HAC)

COMDAT VARIABLE	ALM	ALO	BCS	BHC	BHI	CMD	GET	SEQ	CLK	DBI	DIN	CSI	CSO	EXI	FCP	GRF	CFO	DSK	MMI	RTL	RTH	SWI	TOK	TIK	STA	STS	SUN
AIMOKG						U				S								S U									
AIMPTG						U	S			S																	
HCST1G	U			U		U	U	U		S					S										U	U	
HCST2G	S U			U		S U				S					S										U	U	
HCST3G			S U	U		S U		S U		S																	
HFCS1G	U				U					S					S												
HFCS2G	S U			S U	S U					S					S U												
MODEG						U				S									U						U	S U	
PWHC1G						U				S																	
SEQLSG						S U		S U		S																	
SEQNMG						S		S U		S												U					

S = Set
U = Used

Table 3.3.1-I Task Utilization of Global Common COMDAT (con't.)
(8 Sec Transfer Frequency to Backup HAC)

COMDAT VARIABLE	ALM	ALO	BCS	BHC	BHI	CMD	GET	SEQ	CLK	DBI	DIN	CSI	CSC	EXI	FCP	GRF	CFO	DSK	MMI	RTL	RTH	SWI	TOX	TIK	STA	STS	SUN
BCSTGG			U			U				S																	
CFABOG										S							S U		S								
CFWATG										S							S U		S								
CORRCG					U	U		U		S																	
CPPG						U				S									S								
CPPRTG						S				S									U								
DISKRG										S								U	S								
HCMAPG						S U				S									S U								
HC2MDG		U	U							S																	
HFCS3G				U	U					S															U		
ISTATG										S									S						U	U	
MDNPRG			U							S U	S U								U								
MD2HCG			U							S	S U								U								
SEGMPG										S U	S U					U			U								
SEGPTG										S U	S U					U			U								
SLATG										S																	U
SLONGG										S																	U

S = Set
U = Used

Table 3.3.1-I Task Utilization of Global Common COMDAT (con't.)
(Transferred to Backup HA only at Initialization)

GLOBAL COMMON	ALM	ALO	BCS	BHC	BHI	CMD	GET	SEQ	CLK	DBI	DIN	CSI	CSO	EXI	FCP	GRF	CFO	DSK	MMI	RTL	RTH	SWI	TOK	TIK	STA	STS	SUN
COM1S1	X			X		X	X			X					X												
COM1S2			X	X		X	X	X		X																	
COM1S3										X						X										X	
COM1S4			X	X	X	X				X					X												
COM1S5	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
COM8S1						X	X			X																	
COM8S2	X			X		X	X	X		X					X										X	X	
COM8S3			X	X		X		X		X																	
COM8S4						X				X																	
COMIN1		X	X							X																	
COMIN2			X							X									X								
COMIN3			X			X				X						X			X								
COMIN4			X	X	X	X		X	X								X	X	X			X	X	X	X	X	X
COMQUE	X	X	X									X	X	X		X	X		X		X	X				X	
COMQU2						X	X	X																			

Table 3.3.1-II Global Common to Task Assignments

COMQUE QUEUES	ALM	ALO	BCS	BHC	BHI	CMD	GET	SEQ	CLK	DBI	DIN	CSI	CSO	EXI	FCP	GRF	CFO	DSK	MMI	RTL	RTH	SWI	TOX	TIK	STA	STS	SUN
1														E					D		E						
2														E					D								
3														E					D								
4														E					D								
5																	E		D								
6																	E		D								
7	E	D																									
8													D						E							E	
9			E											D					E							E	
10																											
11																											
12												E				D											

E = Enqueue
D = Dequeue

Table 3.3.1-III Task Utilization of Message Queue Global Common COMQUE
(Transfer frequency to Backup, Every Second)

3.4 HFC Firmware Design

3.4.1 HFC Firmware Module - HFCMOD

3.4.1.1 Purpose

HFCMOD is the firmware which performs all HFC functions. These functions are:

- a. HAC I/O interfacing;
- b. HC I/O interfacing;
- c. HAC command processing;
- d. HC operations;
- e. Corridor walk calculations;
- f. Emergency corridor walk operations; and
- g. HFC timer maintenance.

HFCMOD is composed of nine major submodules which perform the seven HFC functions. These submodules are:

- a. HACIN which interfaces to the HAC/HFC input port;
- b. HACOUT which interfaces to the HAC/HFC output port;
- c. HCIN which interfaces to the HFC/HC input port;
- d. HCOU which interfaces to the HFC/HC output port;
- e. CMDI which processes HAC inputs and sun vector time-outs;
- f. HCOPS which sequences and performs all HC operations;
- g. CWCALC which handles corridor walking sequences;
- h. ECWOPS which controls emergency corridor walk-down sequencing in the event of HAC communications loss; and
- i. HFCTMR which generates two HFC system timers from the HFC hardware timer.

3.4.1.2 Requirements

3.4.1.2.1 Design Requirements

Section 3.1 of the Software/Firmware Functional Requirements requires the following of the HFC:

- a. Transmit commands to the HCs;
- b. Transmit status and data to the HAC;
- c. Initiate safe stowage commands to the HCs upon loss of HAC communication; and
- d. Control groups of HCs.

3.4.1.2.2

Derived Requirements

The following are the derived requirements for this module:

- a. Maintain communications with the HAC computer;
- b. Transmit the received commands for up to 32 HCs connected on the HFC and store the parameters for four corridors;
- c. Poll the specified four HCs in response to a polling command from the HAC;
- d. Transmit the status received from the four HCs to the HAC, upon request from the HAC;
- e. Detect communications errors in messages from HAC and HC; and
- f. On loss of communications with HAC computer, control stow of up to 32 heliostats using an approximate corridor walk (using last received sun vector).

3.4.1.3

Design Approach

3.4.1.3.1

Functional Allocations

HFCMOD is composed of nine submodules which are briefly described below:

- a. HACIN - collects message bytes received from the HAC, detects the start and end of each message, performs checksum calculation, and activates CMDI when a complete message is collected;
- b. HACOUT - transmits messages a byte at a time to the HAC, calculates the checksum, and appends the checksum to the end of the output message;
- c. HCIN - collects message bytes received from the HC, detects the start and end of each message, performs checksum calculation, and reactivates the calling submodule when a complete message is detected;

- d. HCOU - transmits messages a byte at a time to the HC, calculates the checksum and appends the checksum to the end of the output message;
- e. CMDI - controls the mode switching of the HC and associated processing according to current mode and current input, processes all HAC commands, and invokes CWCALC, HCOPS, and ECWOPS;
- f. HCOPS - performs all operations to the HCs including sun/command output, command response receipt, HC status polling and receipt, and status reformatting;
- g. CWCALC - performs the once-per-second updating of the corridor coordinates and handles corridor endpoints;
- h. ECWOPS - generates heliostat mode masks from the normal heliostat status buffer, sequences the emergency corridor walk when HAC communications is lost, and generates HC commands to accomplish the emergency corridor walk; and
- i. HFCTMR - maintains an 833 usec and a 50 msec timer for use by the other HFC submodules. The 833 usec timer is used for timing I/O intervals. The 50 msec timer is used to detect HAC communications failure and to provide a one second time base during emergency corridor-walk operations.

3.4.1.3.2

Resource Budgets

The HFC is limited to 4096 bytes of read-only memory (1080 of which are to be unused spare), 1152 bytes of read/write memory (732 of which are to be unused spare), 1.2288 MHz cycle time, and one counter/timer. The HFC firmware must fit in the available memory and execute within a one second cycle time.

HFCMOD is estimated to require 3016 bytes of read-only memory and 420 bytes of read/write memory.

3.4.1.4

Design Description

3.4.1.4.1

Module Structure (See figure 3.4.1-1)

HFCMOD consists of nine submodules. Five are interrupt driven and the remaining four are activated serially under software control. CMDI is the main controller of the four ground-level tasks (analogous to an executive or taskmaster in a large computer). Interrupt routines and ground level routines communicate through flag settings. Whenever the HFC is waiting for something to do (HAC input or sun vector time-out), CMDI loops and tests the HAC input flag and sun vector time-out flag. CWCALC, HCOPS, and ECWOPS are

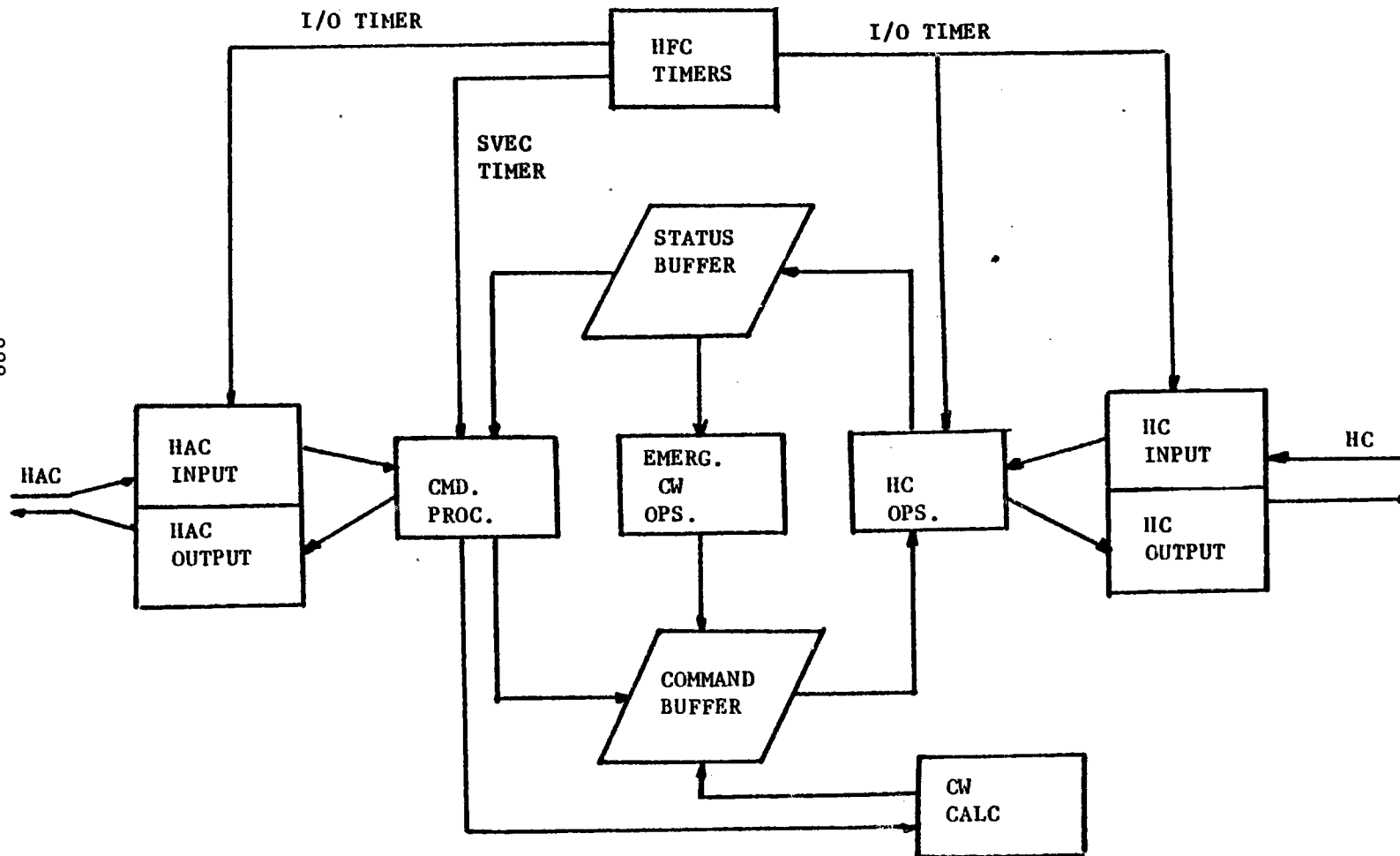


Figure 3.4.1-1 HFC Module Structure

activated by CMDI when the sun vector is received from the HAC or each second by sun vector time-out when in the emergency corridor-walk mode. HACIN, HACOUT, HCIN, and HCCOUT are I/O interrupt service routines which perform byte oriented I/O and checksum calculations. In addition, the input routines detect message end by byte time-out when there is a gap in transmission. HFCTMR services the timer interrupts and provides task timers for the remainder of the HFC submodules.

3.4.1.4.2 Submodule I - Command Interpreter (CMDI)

3.4.1.4.2.1 Main Routine

3.4.1.4.2.1.1 Description

CMDI has three major functions: taskmaster, HFC mode determination, and HAC command processing. CMDI determines what processing is to be performed and what mode the HFC should be in according to the current HFC mode and the type of input stimulus. The HFC may be in the following modes:

- a. RESTART - this mode is entered on initial power-up and after completion of emergency corridor walk. During this mode the HFC switches between the primary and secondary communications lines at a 13-second rate until HAC communications is established. The HFC remains on the established line either primary or secondary until communications is lost;
- b. NORMAL - normal mode of HFC;
- c. PRE-EMERGENCY CORRIDOR WALK - this mode is entered when the sun vector times out (loss of HAC communications) to allow ten seconds for HAC recovery; and
- d. EMERGENCY CORRIDOR WALK - after eight seconds, emergency corridor walk begins and the HFC turns off all HAC I/O.

The input stimuli are:

- a. SUN VECTOR TIME-OUT - this is also used as a one-second time base when no sun vector is received from the HAC;
- b. SUN VECTOR RECEIVED;
- c. HC COMMAND RECEIVED - these are allowed only in the normal mode and the HFC must have been initialized;
- d. HFC INITIALIZATION RECEIVED - allowed only in normal mode, and they must occur in order (A & B

CULPs, A & B CLLPs, A & B DELTAs, C & D CULPs, C & D CLLPs, C & D DELTAs, and CORRIDOR-ASSIGNS); and

- e. HAC STATUS POLL RECEIVED - only allowed in normal mode.

Table 3.4.1-I is a state table describing the processes and mode changes of the HFC based on current mode and stimulus.

When in the normal mode, CMDI will process all commands from the HAC. HC commands are formatted into the HC command buffer for HCOPS, except corridor-walk commands which are channeled to CWCALC. Status commands are processed and completed in CMDI. A normal HAC status poll only requires output to the HAC since HCOPS has precollected and formatted the status buffer. HFC initialization commands are checked for correct order, and then the parameters (CULPs, CLLPs, etc...) are stored in the HFC data base. Receipt of a sun/sync packet from the HAC causes CMDI to act as a taskmaster, invoking HCOPS, CWCALC, and ECWOPS in that order. During normal and emergency corridor-walk modes, this happens once every second.

3.4.1.4.2.1.2 Data, Logic, and Command Paths

CMDI input are commands from the HAC. See section 3.4.1.5 for HAC/HFC communications description. CMDI's output is task activations, sun vector and HC command data to the HC command buffer, corridor-walk data to CWCALC, and HFC initialization data to the HFC data base. CMDI's logic and command paths are described above and in the flowcharts.

3.4.1.4.2.1.3 Internal Data Description

CMDI has no internal tables or data structures.

3.4.1.4.2.1.4 Flowcharts (See figures 3.4.1-2 and 3.4.1-3)

3.4.1.4.2.2 CMDI Subroutine I - CMDCMD

3.4.1.4.2.2.1 Description

CMDCMD is called by CMDI when the HAC is in the normal mode, a HAC command has been received, and the HFC is fully initialized. CMDCMD copies the data from the HAC message into the HC command buffer if the command is AZ/EL pointing, beam pointing, or HC initialization. Also, the selected HCs are removed from the corridor-walk command masks. If CMDCMD decodes a corridor-walk start command, CMDCMD generates the first corridor-walk command and sets up CWCALC to continue the corridor-walk operation. In all cases, CMDCMD sets the command return field in the HFC status.

ECW
DONE

	POWER-UP	SUN-VECTOR TIME-OUT	SUN-VECTOR RECEIVED	HFC COMMAND RECEIVED	HFC INIT RECEIVED	HFC STAT POLL RECEIVED
RESTART MODE		SWAP HAC LINES, RESET THE TIMER AND SET THE INIT PHASE BACK TO ZERO	RESET TIMER, GO NORMAL, THEN	IGNORE	IGNORE	IGNORE
NORMAL MODE		GO TO ECW MODE	RESET TIMER, DO C.W. CALC THEN HCOPS AND STATUS REFORMAT	PROCESS IT IF INITIALIZED	PROCESS IT, BUT IF WRONG ORDER, SET INIT PHASE BACK TO ZERO	OUTPUT IF STAT 4 HC's
PRE-EMERG. CW HOLD MODE 0-10 SECS.		SWAP HAC LINES, RESET TIMER, DO C.W. CALC THEN HCOPS AND STATUS REFORMAT	RESET TIMER, GO TO NORMAL MODE BUT WAIT FOR NEXT INPUT	IGNORE	IGNORE	IGNORE
* EMERG. CW AFTER 10 SECS.		RESET TIMER, DO C.W. CALC THEN HCOPS AND STATUS REFORMAT	IGNORE	IGNORE	IGNORE	IGNORE

* When emergency CW is all done, HFC goes to the restart mode.

Table 3.4.1-1

HFC State Transition Matrix

891

FLOW CHART & BLOCK DIAGRAM

HFC COMMAND INTERPRETER

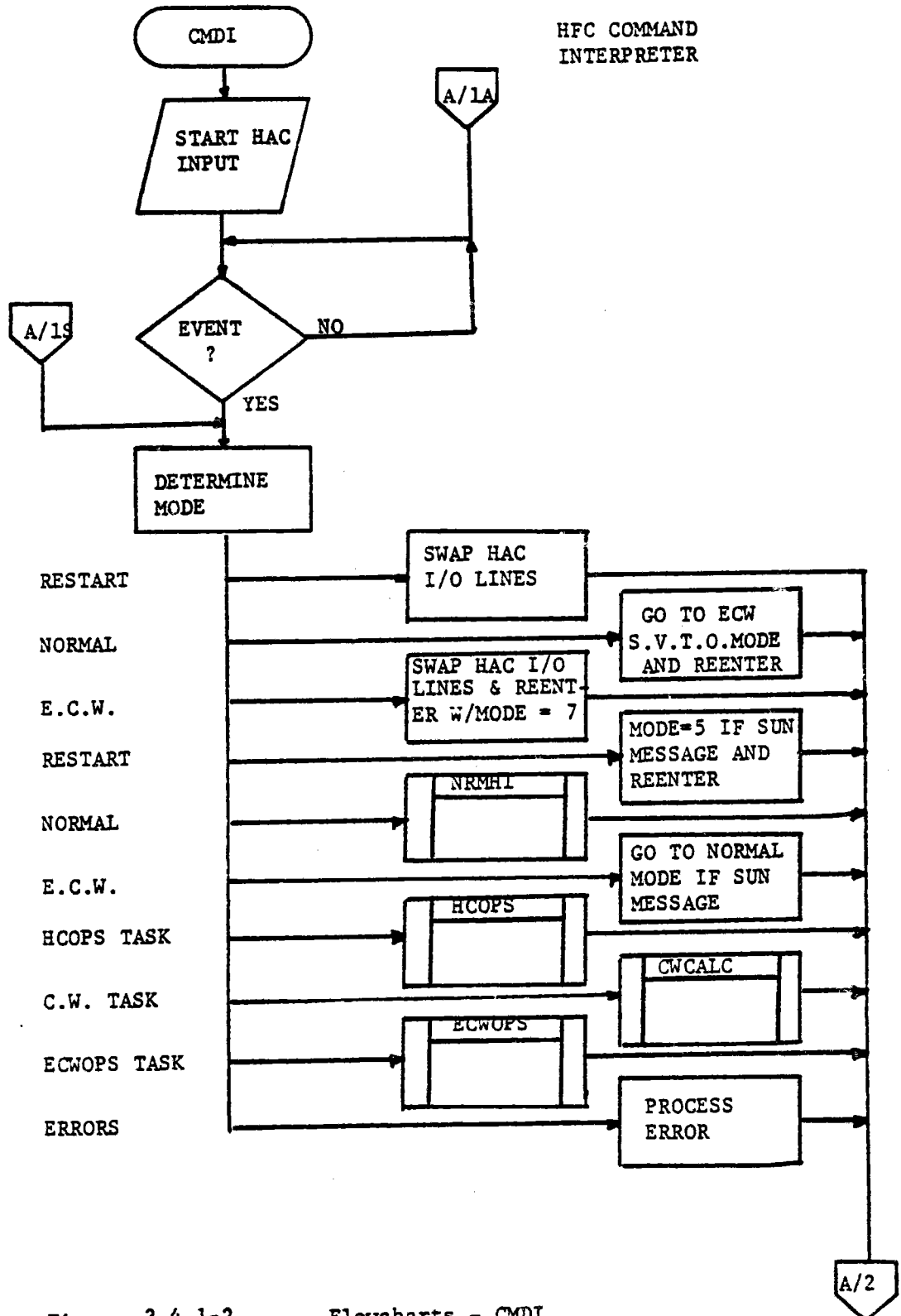


Figure 3.4.1-2

Flowcharts - CMDI

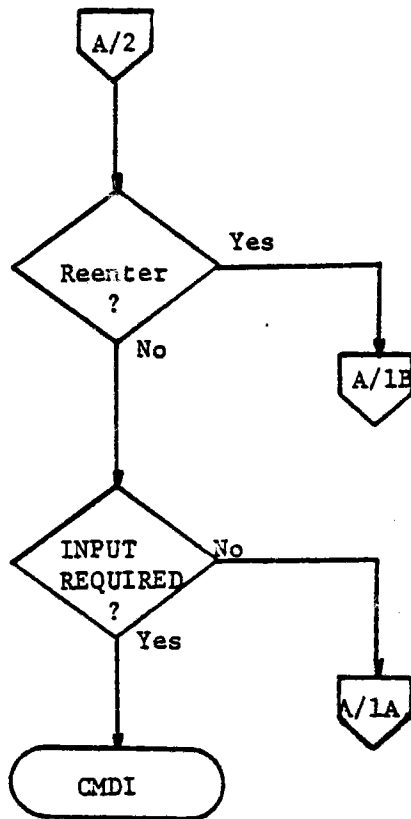


Figure 3.4.1-2

Flowcharts - CMDI (Cont.)

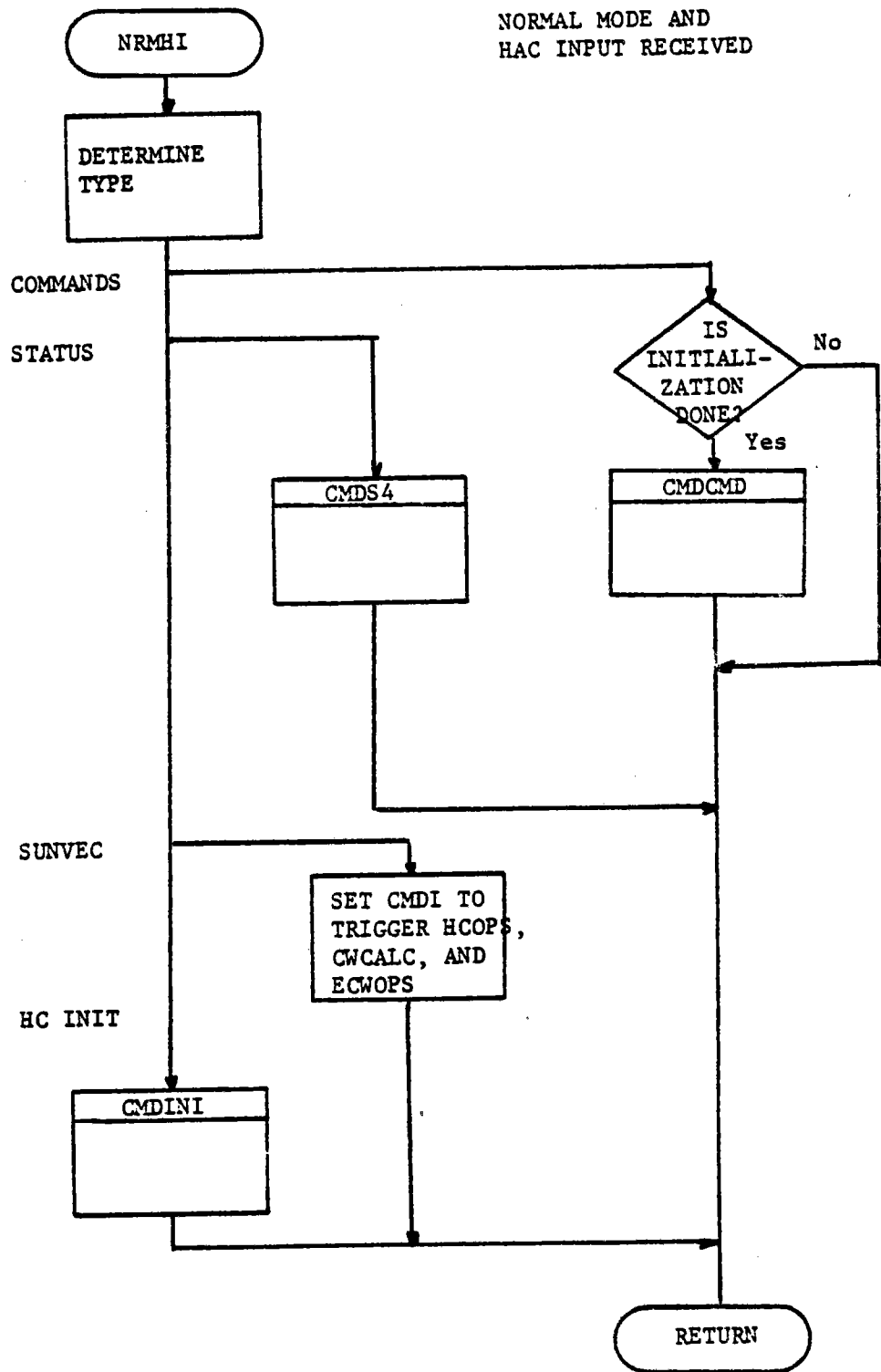


Figure 3.4.1-3

3.4.1.4.2.2.2 Data, Logic and Command Paths

CMDCMD's input data is the HAC message from the HAC input buffer. CMDCMD outputs data to the HC command buffer and to CWCALC as described above.

3.4.1.4.2.2.3 Internal Data Description

CMDCMD has no internal tables or data structures.

3.4.1.4.2.2.4 Flowcharts (See figure 3.4.1-4)

3.4.1.4.2.3 CMDI Subroutine II - CMDINI

3.4.1.4.2.3.1 Description

CMDINI is called by CMDI when the HFC is in the normal mode and an HFC initialization command has been received. Because there are seven initialization message packets (A & B CULPs, A & B CLLPs, A & B Corridor Deltas, C & D CULPs, C & D CLLPs, C & D Deltas, and Corridor Assignments) which must be received to fully initialize the HFC, CMDINI insures they are all received by requiring them to be sent in sequence. Corridor assignments are the exception because they are allowed any time. If the current packet is received in sequence, the data is stored in the HFC data base and the initialization phase is advanced so the CMDINI expects the next packet in sequence. However, the corridor assignments being last, cause no phase update, and the initialization flags are reset. If the received packet is out of order, the initialization phase is reset to the start to force reinitialization. Finally, CMDINI sets the command return in the HFC status.

3.4.1.4.2.3.2 Data, Logic, and Command Paths

CMDINI's input data is the HAC message from the HAC input buffer. CMDINI outputs data to the HFC data base.

3.4.1.4.2.3.3 Internal Data Description

CMDINI's only internal data is the initialization phase, IPHASE.

3.4.1.4.2.3.4 Flowcharts (See figure 3.4.1-5)

3.4.1.4.2.4 CMDI Subroutine III - CMDS4

3.4.1.4.2.4.1 Description

CMDS4 is called by CMDI when the HAC requests the normal status for four heliostats. CMDS4 starts the HAC output of the status buffer and waits in a loop until it is complete.

CMDI Command Process

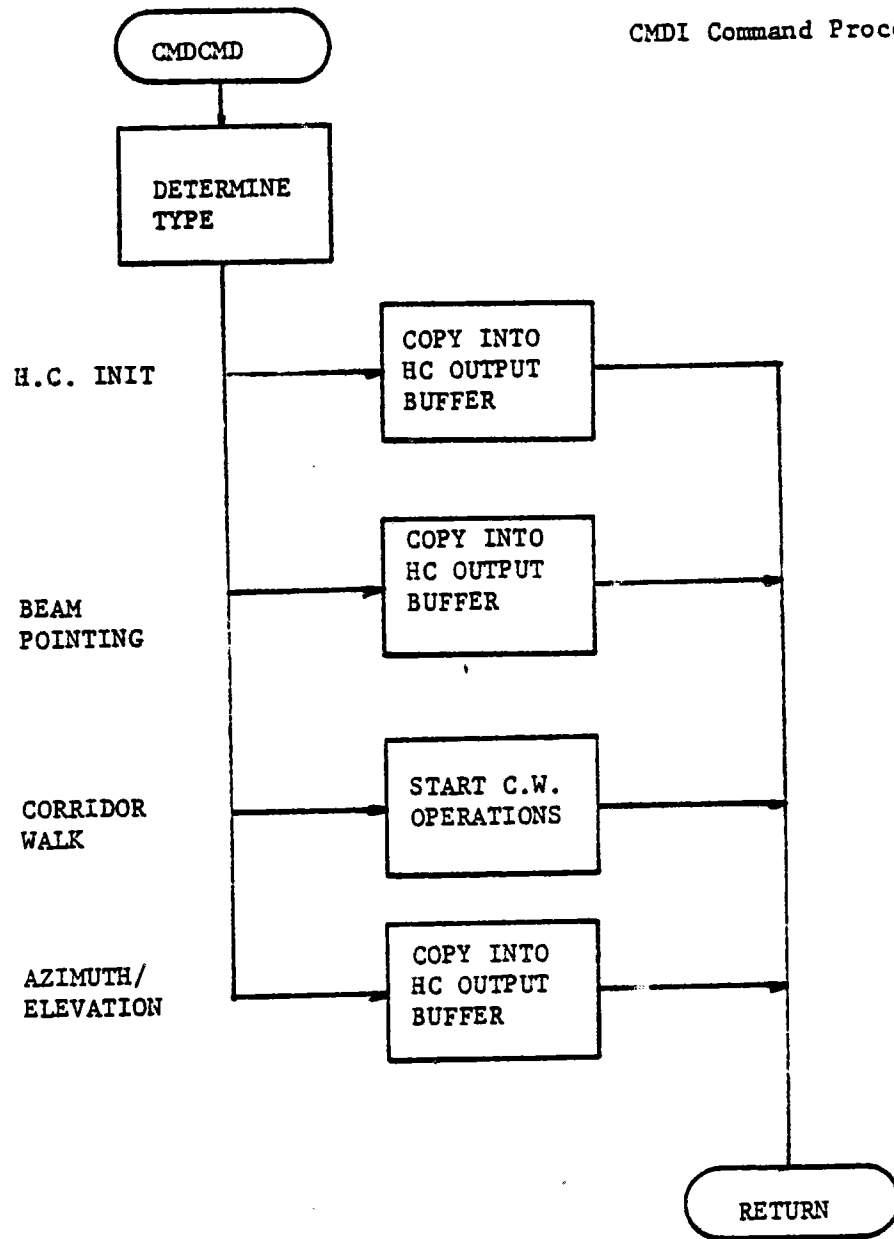


Figure 3.4.1-4 Flowcharts - CMDCMD

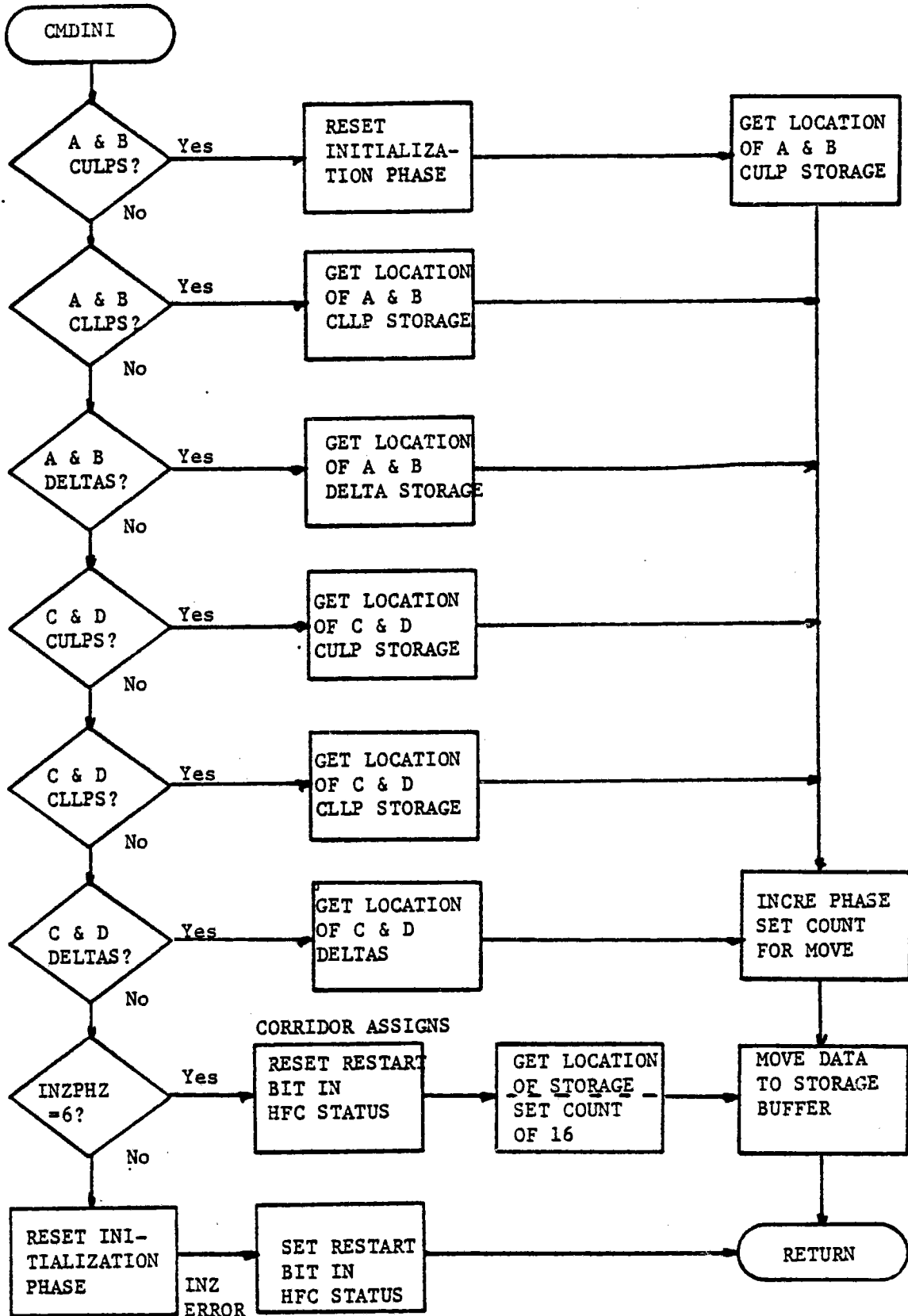


Figure 3.4.1-5

Flowcharts - CMDINI

3.4.1.4.2.4.2 Data, Logic and Command Paths

CMDS4's input is the four HC status buffer which was collected by HCOPS. CMDS4's output is to the HAC via HACOUT.

3.4.1.4.2.4.3 Internal Data Description

CMDS4 has no internal data.

3.4.1.4.2.4.4 Flowcharts (see figure 3.4.1-6)

3.4.1.4.2.5 CMDI Subroutine IV - POLIHC

3.4.1.4.2.5.1 Description

POLIHC status polls a single HC, specified by input argument, when called by either CMDI or HCOPS. POLIHC first sends out a status poll command via HCOU to the requested HC and then waits in a loop for it to complete. POLIHC then sets up a read via HCIN and a deadman timer via HFCTMR and waits for the event to occur. If HCIN received a byte before the timer times out, it will be reset for each byte until end-of-message. When POLIHC continues in response to timer time-out, it returns either one HC's status or a no-response code depending on the received byte count.

3.4.1.4.2.5.2 Data, Logic and Command Paths

POLIHC receives the desired HC number as an argument and returns one HC's status to the calling routine.

3.4.1.4.2.5.3 Internal Data Description

POLIHC contains a skeleton copy of the HC polling command.

3.4.1.4.2.5.4 Flowcharts (See figure 3.4.1-7)

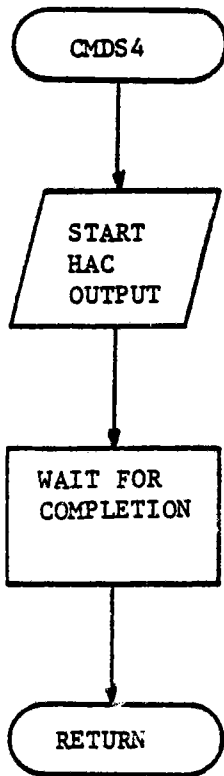
3.4.1.4.3 Submodule II - Corridor-Walk Calculation (CWCALC)

3.4.1.4.3.1 Main Routine

3.4.1.4.3.1.1 Description

CWCALC performs the corridor-walking operations in the HFC. CWCALC processes in two distinct steps. First, corridor coordinates are incremented (or decremented) and updated in the HC command buffer. The second step of CWCALC is corridor end-point detection. If, after the corridor coordinates are updated, the corridor Z-coordinate is greater than (less than) the Corridor Upper (Lower) Limit Point. For a walk-up (down)

CMDI Return 4 HC's Status



FLAG SET BY HACOUT

Figure 3.4.1-6

Flowcharts - CMDI4

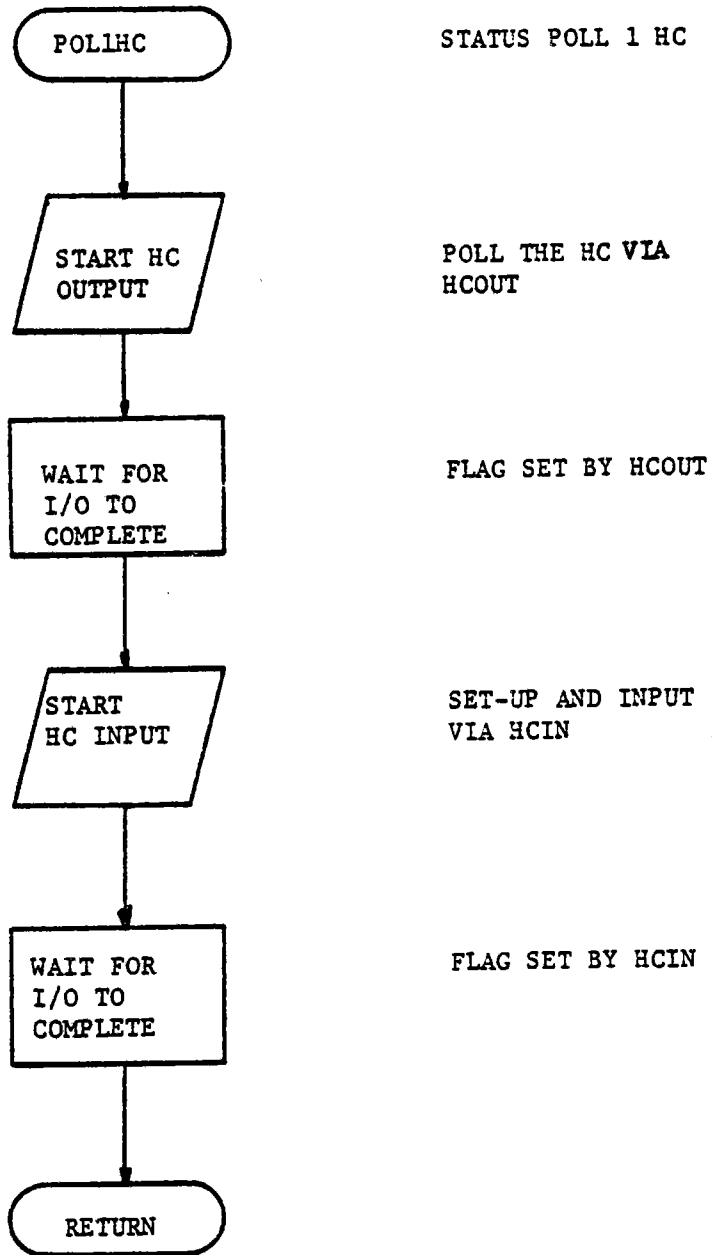


Figure 3.4.1-7

Flowcharts -POL1HC

operation, CWCALC generates a final corridor-walk command to the Corridor Upper (Lower) Limit Point. One second later, CWCALC sets the corridor status as "not in use" and reenables the HC command response for walking the HCs to the HAC.

CWCALC utilizes two subroutines, CKDELTA and CKDELTC, to determine if a delta add or a delta subtract is required to update the current corridor vector. Endpoint detection is also a function of these subroutines.

The flowchart for CKDELTA is detailed in figure 3.4.1-9. The CKDELTC is identical in function to CKDELTA which controls the corridor delta updates and endpoint detection for corridors C and D.

3.4.1.4.3.1.2 Data, Logic and Command Paths

CWCALC's input consists of CULP's, CLLPs, DELTAs from the HFC data base and corridor-walk start-up information from CMDI. CWCALC's output is corridor-walk commands to the HC command buffer and corridor status to the HFC status buffer.

3.4.1.4.3.1.3 Internal Data Description

CWCALC has no internal tables or data structures because it operates directly on the HC command buffer.

3.4.1.4.3.1.4 Flowcharts (See figures 3.4.1-8 and 3.4.1-9)

3.4.1.4.4 Submodule III - HC Operations (HCOPS)

3.4.1.4.4.1 Main Routine

3.4.1.4.4.1.1 Description

HCOPS performs all normal operations to the HCs except special, single-HC status polls. HCOPS has three subfunctions: HC sun/sync/command message output, command-response reception and formatting, and HC status polling and formatting. These three subfunctions are performed by subroutines HCOSSC, HCOCCR and HCOPOL, respectively. HCOPS calls these three subroutines in order.

3.4.1.4.4.1.2 Data, Logic and Command Paths

HCOPS calls the three subroutines once each, in order: HCOSSC, HCOCCR, HCOPOL. HCOPS input is the HC sun/sync/command buffer, the number of the first HC to be status polled, and HC inputs. HCOPS outputs commands to the HCs and updates the HFC/HC status buffer.

3.4.1.4.4.1.3 Internal Data Description

HCOPS has no internal tables or data structures because it operates on the HC input and output buffers directly.

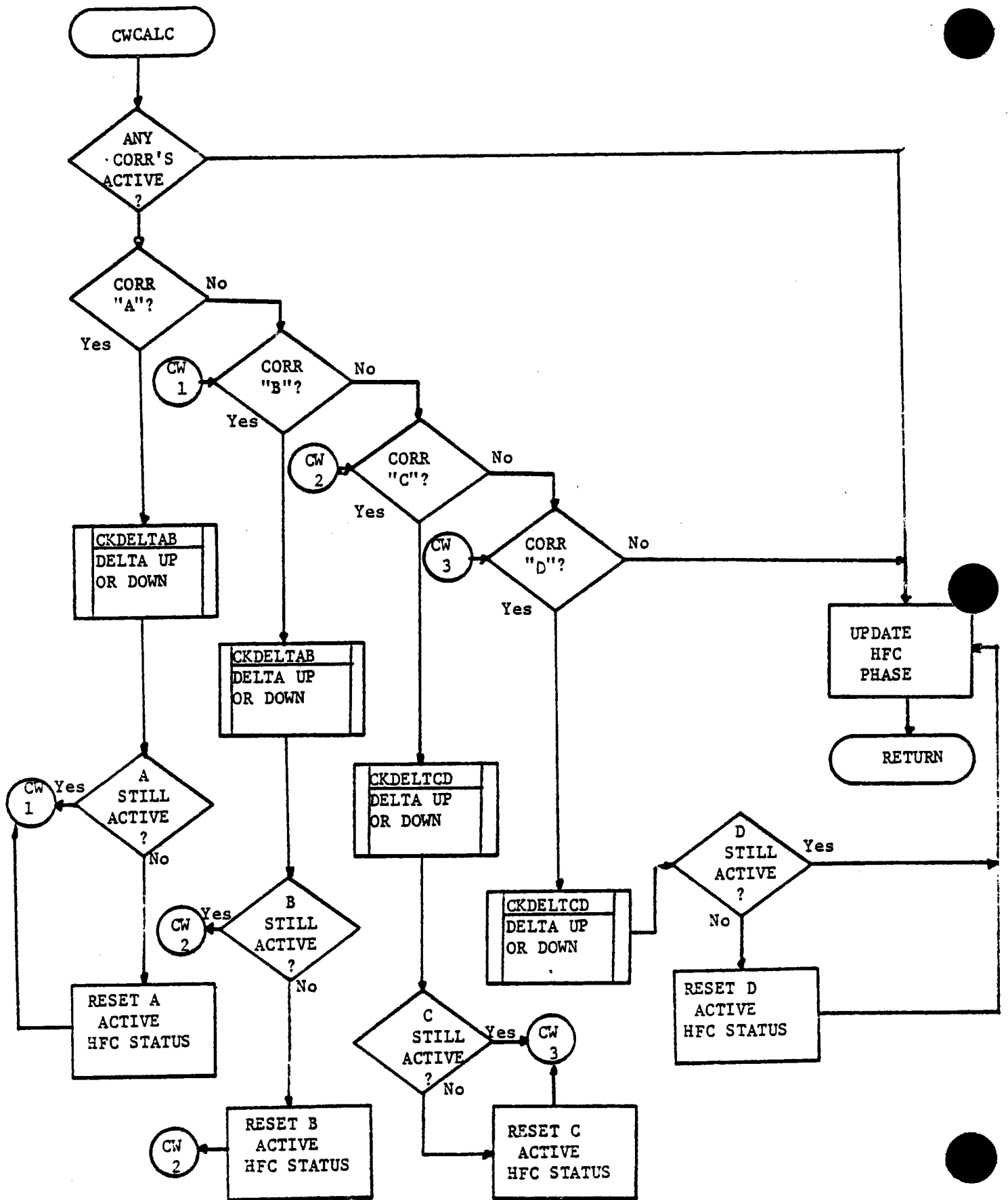


Figure 3.4.1-8

Flowchart - CWCALC

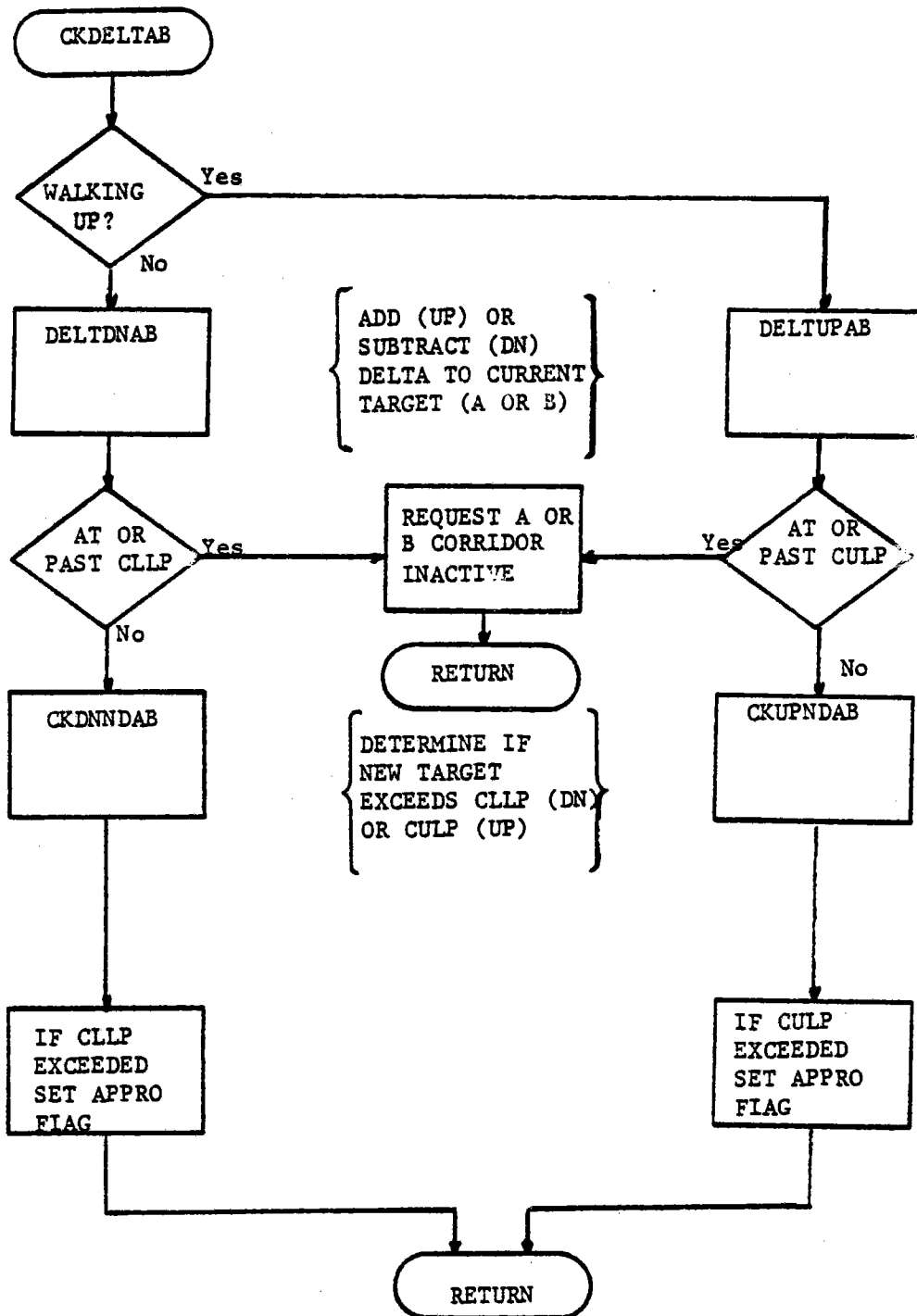


Figure 3.4.1-9

Flowchart - CKDELTAB

- 3.4.1.4.4.1.4 Flowcharts (See figure 3.4.1-10)
- 3.4.1.4.4.2 HCOPS Subroutine I - HCOSSC
 - 3.4.1.4.4.2.1 Description

HCOSSC outputs the sun/sync/command message to the HCs when called by HCOPS in response to a HAC sun vector message or emergency Corridor-walk sun vector time-out. HCOSSC starts the output operation (carried on by HCOU) and waits in a loop until it completes. HCOSSC then clears the command mask and returns to HCOPS.
 - 3.4.1.4.4.2.2 Data, Logic and Command Paths

HCOSSC's input command is the sun/sync/command HC output buffer.
 - 3.4.1.4.4.2.3 Internal Data Description

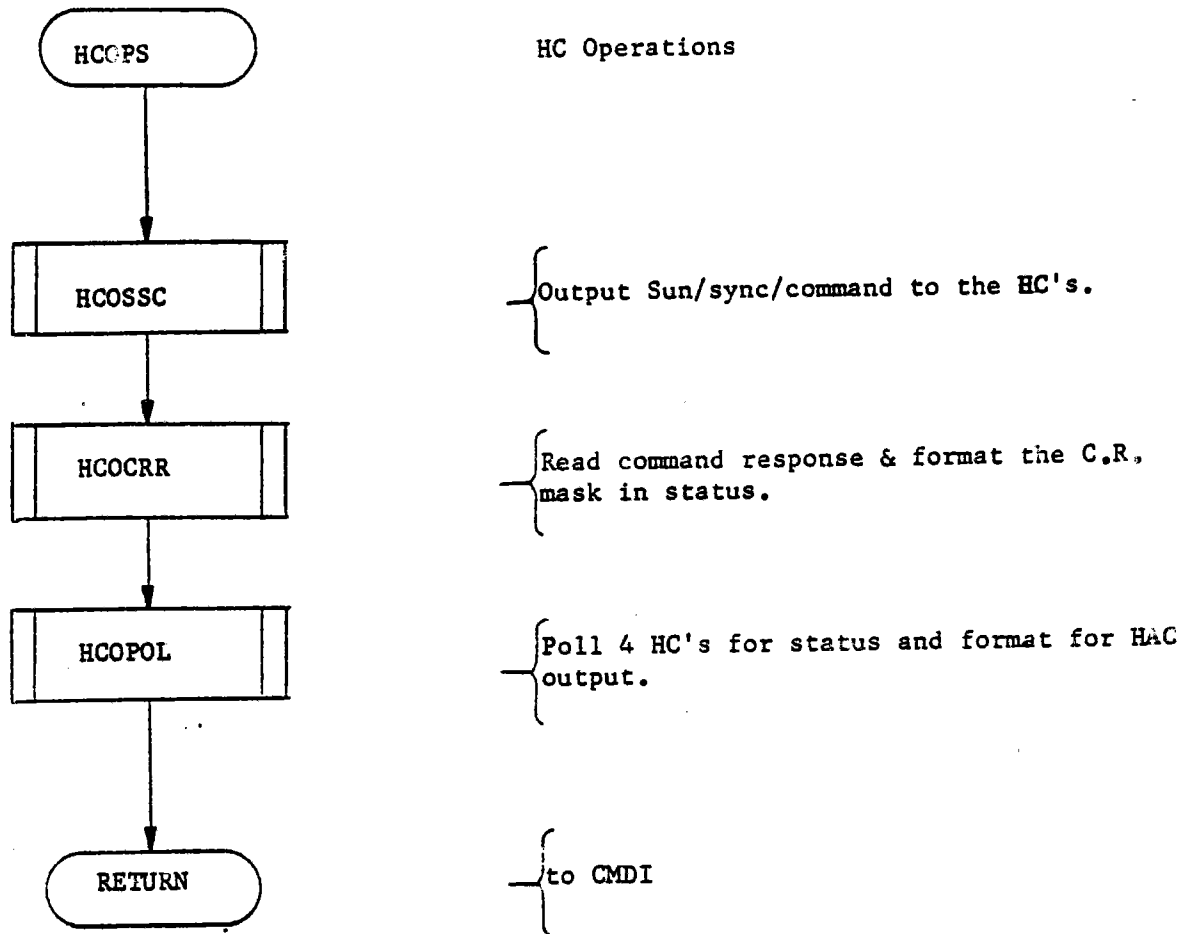
HCOSSC has no internal data.
 - 3.4.1.4.4.2.4 Flowcharts (See figure 3.4.1-11)
- 3.4.1.4.4.3 HCOPS Subroutine II - HCOCRR
 - 3.4.1.4.4.3.1 Description

HCOCRR sets up the input of the command responses from the HCs and reformats them into a 32-bit mask in the HFC output status. HCOCRR first sets up a 112 msec deadman timer and starts an HC input. HCOCRR then waits in a loop until the 112 msec timer times out. HCOCRR next decodes the HC number from each byte of command response input to know which bits in the command response mask to set. HCOCRR then returns to HCOPS.
 - 3.4.1.4.4.3.2 Data, Logic and Command Paths

HCOCRR's input is the command response bytes from the HCs via HCIN. HCOCRR's output is the HC command response mask.
 - 3.4.1.4.4.3.3 Internal Data Description

HCOCRR has no internal data.
 - 3.4.1.4.4.3.4 Flowcharts (See figure 3.4.1-12)
- 3.4.1.4.4.4 HCOPS Subroutine III - HCOPOL
 - 3.4.1.4.4.4.1 Description

HCOPOL collects and formats for HAC output from HC's status. HCOPOL performs the following sequence four times and then



Flowchart - HCOPS
 Figure 3.4.1-10

HCOSSC

HCOPS Sun/sync/command Output Subroutine

START
HC
OUTPUT

{ Send the sun/sync/command message via HCOU

WAIT FOR
I/O COMPLETE

{ Flag set by HCOU

CLEAR COMMAND
MASK ON
S/S/C

{ But don't clear CW command masks

RETURN

{ to HCOPS

Flowchart - HCOSSC
Figure 3.4.1-11

HCOCRR

HCOPS COMMAND RETURN RECEPTION SUBROUTINE

SET 112 ms
TIMER

via HFCTMR

START HC
INPUT

Read 32 bytes via HCIN

WAIT FOR
112 ms
TIMER

HC reply only if and received
& HC selected

MAKE 32 BIT
MASK FROM
RESPONSES

1 bit per HC

RETURN

to HCOPS

Flowchart - HCOCRR

Figure 3.4.1-12

returns to HCOPS:

- a. Delay five msec to allow message dead space on the HFC/HC line;
- b. Call POL1HC to get the status for the next HC; and
- c. Reformat the status into the status output buffer (destined for the HAC). If no response from HC, set the "HFC detected HC command error" bit in the output status.

HCOPOL receives from the HAC sun message the starting HC number (0, 4, 8, 12, 16, 20, 24, 28) and increments it at the end of the loop.

3.4.1.4.4.4.2 Data, Logic and Command Paths

HCOPOL's input is the HC number to be polled and the HC status from POL1HC. HCOPOL outputs reformatted HC status to the HAC output buffer.

3.4.1.4.4.4.3 Internal Data Description

HCOPOL operates directly on the HC input buffer and HAC output buffer.

3.4.1.4.4.4.4 Flowcharts (See figure 3.4.1-13)

3.4.1.4.5 Submodule IV - Emergency Corridor-Walk Operations (ECWOPS)

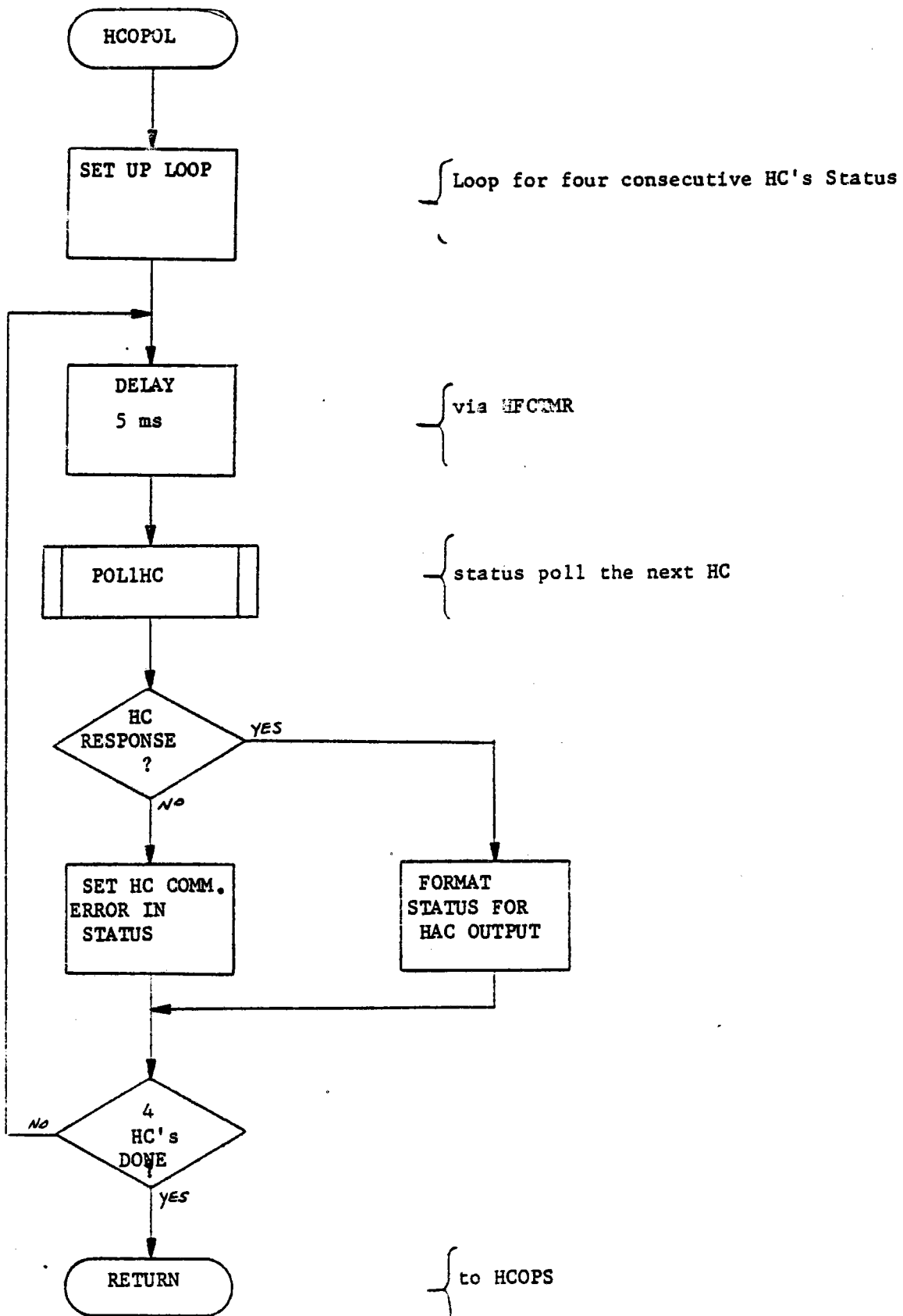
3.4.1.4.5.1 Main Routine

3.4.1.4.5.1.1 Description

ECWOPS is activated every second by CMDI either in response to a received sun vector from the HAC or to a sun vector time-out during emergency corridor-walk mode. During ECW mode, the sun vector time-out is the one-second time base. Normally, ECWOPS returns to CMDI, but when enabled in the ECW mode, ECWOPS sequences the emergency corridor-walk operations. ECWOPS calls two subroutines to perform the ECW operations. ECWOPS generates, from the HC status collected by HCOPS, five mode masks used by the second subroutine, ECWSEQ, which does the actual step-by-step sequencing. ECWOPS calls ECWGMG then ECWSEQ once each second and returns to CMDI.

3.4.1.4.5.1.2 Data, Logic and Command Paths

ECWOPS uses the HC status from HCOPS to generate the mode masks. ECWOPS' output is commands to the HC sun/sync/command output buffer.



Flowchart - HCOPOL
Figure 3.4.1-13

3.4.1.4.5.1.3 Internal Data Description

ECWOPS main routine has four masks generated by ECWGM for ECWSEQ. See ECWGM, section 3.4.1.4.5.2.

3.4.1.4.5.1.4 Flowcharts (See figure 3.4.1-14)

3.4.1.4.5.2 ECWOPS Subroutine I - ECWGM

3.4.1.4.5.2.1 Description

ECWGM is called by ECWOPS once-per-second during ECW mode. ECWGM generates six mode masks which are used by ECWSEQ to perform ECW operations. These masks are:

- a. UPHCS - 32-bit mask, one bit per HC, which indicates which heliostats are pointing on receiver or at stand-by;
- b. WRUPHCS - 32-bit mask, one bit per HC, which indicates which heliostats are walking up the corridor;
- c. WRDNHCS - 32-bit mask, one bit per HC, which indicates which heliostats are walking down the corridor;
- d. DNHCS - 32-bit mask, one bit per HC, which indicates which heliostats are at any of the lower limit points (CLLPs);
- e. HCPCS - 32-bit mask, one bit per HC, which indicates which heliostats have "position compare" status bit set; and
- f. BADHCS - 32-bit mask, one bit per HC, which indicates which heliostats do not respond to commands.

These mode masks have four HCs updated per second from the normal HC status buffer.

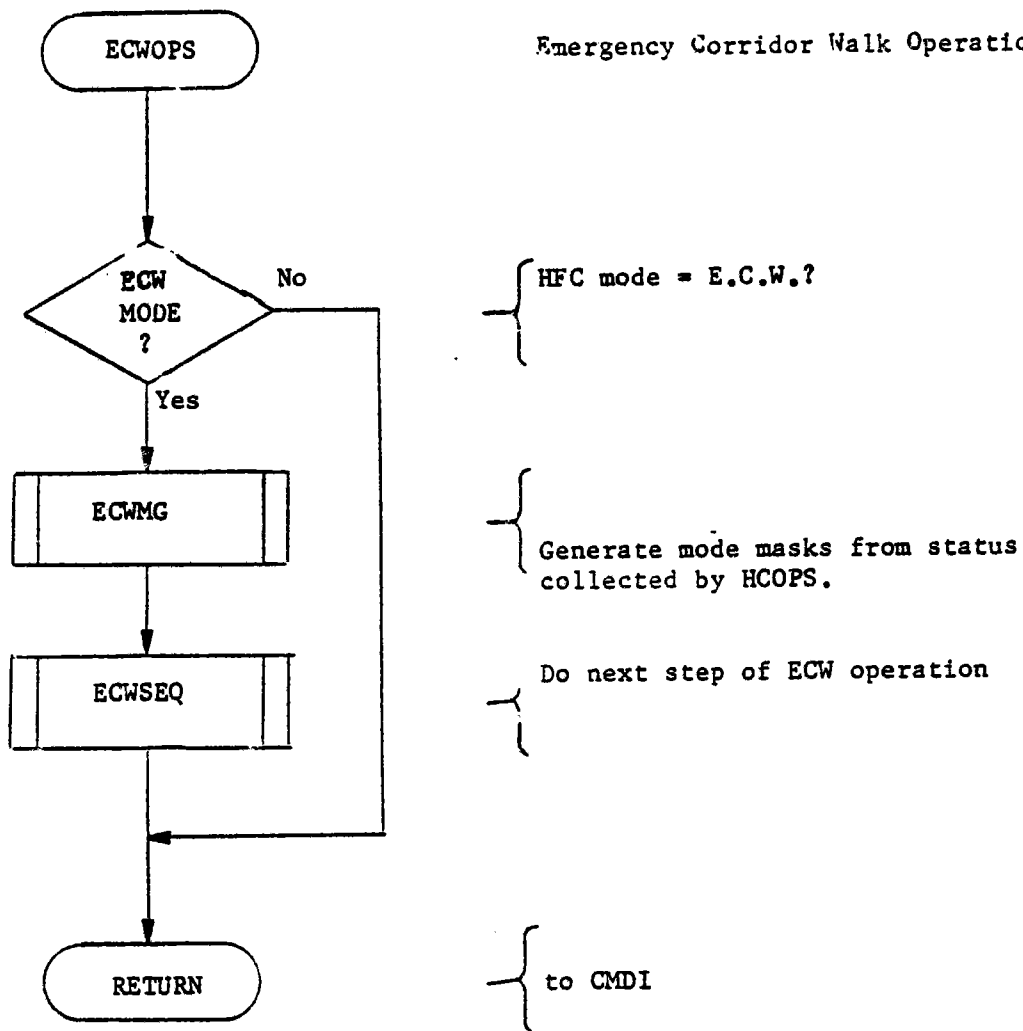
3.4.1.4.5.2.2 Data, Logic and Command Paths

ECWGM generates the five masks according to the following formulae:

UPHCS	(CMD + CR)	.	(RCVR + CULP + BCS)
WRUPHCS	(CMD + CR)	.	(UPWIREA + UPWIREB + UPWIREC + UPWIRED)
WRDNHCS	(CMD + CR)	.	(DNWIREA + DNWIREB + DNWIREC + DNWIRED)
DNHCS	A-CLLP + B-CLLP + C-CLLP + D-CLLP		
HCPCS	PC		

Where:

Emergency Corridor Walk Operations



Flowchart - ECWOPS
Figure 3.4.1-14

- a. CMD is the bit mask of all heliostats commanded in the past second;
- b. CR is the command response mask generated by HCOPs;
- c. CULP, RCVR, BCS, CORRIDOR-A, CORRIDOR-B, CORRIDOR-C, CORRIDOR-D, and CLLP are heliostat submodes;
- d. PC is the collection of the position compare bits from the HCs;
- e. \oplus is logical EXCLUSIVE-NOR;
- f. \cdot is logical AND; and
- g. $+$ is logical OR.

Each second a four-bit slice of each command mask is updated for the four HCs that were just polled. (CMD + CR) is used full 32 bits each second so that failing heliostats will be removed from the masks.

ECWGM's input consists of the command response mask from HCOPs, four HC status from HCOPs, and the command mask for the previous second from ECWSEQ.

3.4.1.4.5.2.3 Internal Data Description

ECWGM's only internal data is used as temporary storage during calculation.

3.4.1.4.5.2.4 Flowcharts (See figures 3.4.1-15 and 3.4.1-16)

3.4.1.4.5.3 ECWOPS Subroutine II - ECWSEQ

3.4.1.4.5.3.1 Description

ECWSEQ is called once-per-second by ECWOPS during emergency corridor-walk mode. ECWSEQ used the mode masks generated by ECWGM and MGIHC to command HCs from all points at or above the CLLPs to stow. Eight seconds after ECWSEQ is first called, it inhibits all HAC I/O until the emergency walk is completed.

ECWSEQ then commands all HCs at or above the CULP to their respective CULPs, commands all HCs at the CLLP to stow, and continues any corridor walks in progress.

Seven minutes later ECWSEQ commands any HCs at the top of the corridor to track their respective CULPs.

Ten seconds later ECWSEQ commands all HCs tracking the CULP to walk down their corridors.

ECWGM

Emergency Corridor Walk Mode
Generation Subroutine

UPHCS ← UPHCS . (CMD ⊕ CR)
WRUPHCS ← WRUPHCS . (CMD ⊕ CR)
WRDNHCS ← WRDNHCS . (CMD ⊕ CR)

Delete HC's which didn't respond
to a command

UPHCS ← UPHCS . MSK4
WRUPHCS ← WRUPHCS . MSK4
WRDNHCS ← WRDNHCS . MSK4
HCPHCS ← HCPHCS . MSK4
DNHCS ← DNHCS . MSK4

Mask out the 4 HC's polled this

SET UP 4 HC
LOOP

Do for four HC's polled this second

MGIHC

Set correct mode & state bit in masks

4
HC's
DONE
?

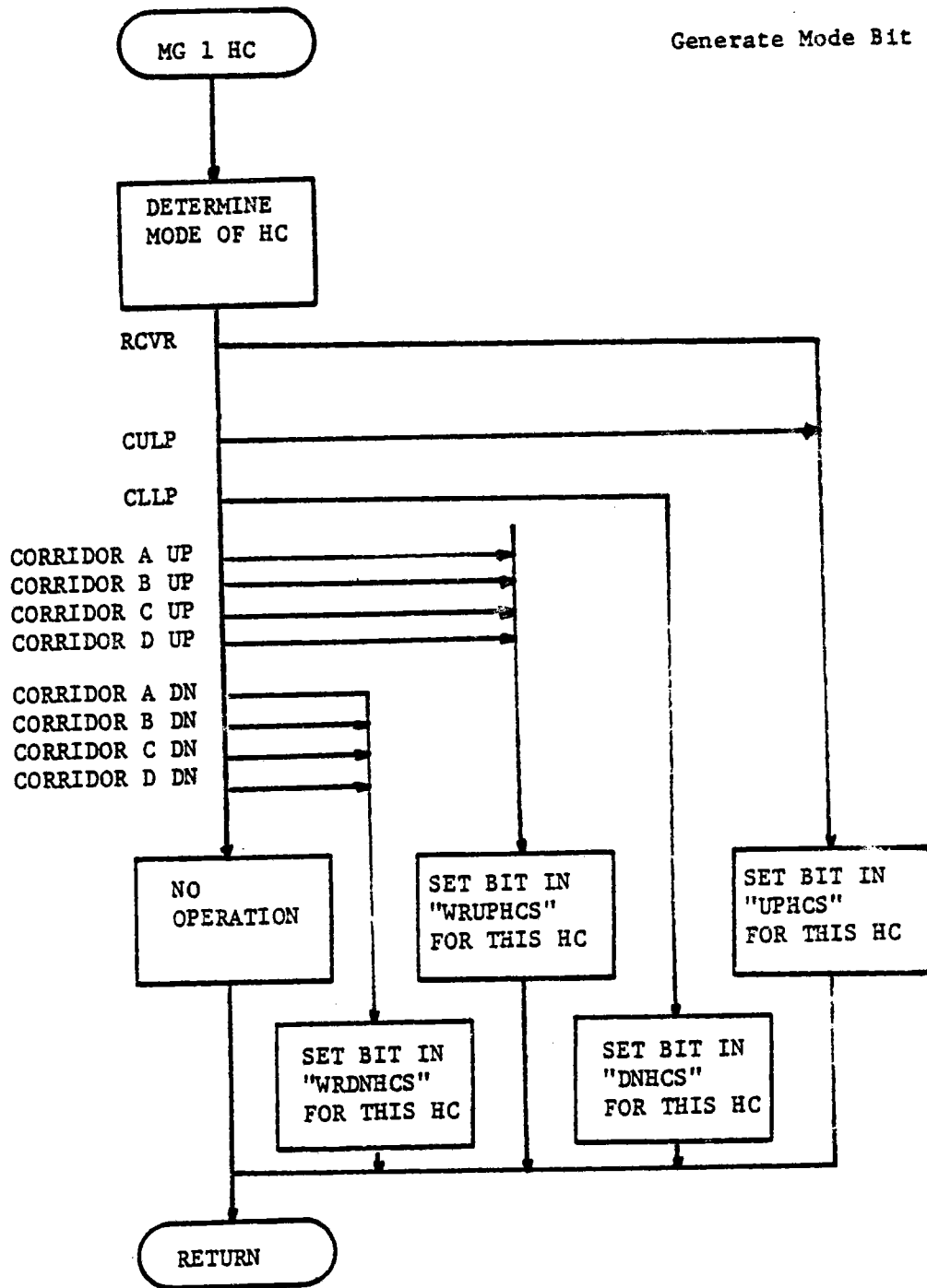
RETURN

to ECWOPS

Flowchart - ECWGM

Figure 3.4.1-15

Generate Mode Bit for 1 HC



Flowchart - MGLHC

Figure 3.4.1-16

ECWSEQ then commands all HCs at the bottom of the corridor to stow.

Seven minutes later, after any corridor walk-downs are complete, ECWSEQ commands all HCs at the bottom of the corridor to stow.

ECWSEQ then initiates an eleven minute wait to verify all HCs have achieved the stow position. ECWSEQ then disables its internal clocks and the HFC hardware initiates a power-on restart.

3.4.1.4.5.3.2 Data, Logic and Command Paths

ECWSEQ sequences through nineteen steps or phases during the course of emergency corridor walking. The phases and their descriptions follow:

- a. PHASE 1 - Wait eight seconds to get full mode status from ECWVG, inhibit HAC I/O, advance phase, and command UPHCS on Corridor-A to A-CULP;
- b. PHASE 2 - Command UPHCS on Corridor-B to B-CULP and advance phase;
- c. PHASE 3 - Command UPHCS on Corridor-C to C-CULP and advance phase;
- d. PHASE 4 - Command UPHCS on Corridor-D to D-CULP and advance phase;
- e. PHASE 5 - command DNHCS to stow and advance phase;
- f. PHASE 6 - Initiate seven-minute delay to allow HCs going up the corridor to complete and advance phase;
- g. PHASE 7 - If delay has expired, command WRUPHCS on Corridor-A to A-CULP and advance the phase; if not, return;
- h. PHASE 8 - Command WRUPHCS on Corridor-B to B-CULP and advance phase;
- i. PHASE 9 - Command WRUPHCS on Corridor-C to C-CULP and advance phase;
- j. PHASE 10 - Command WRUPHCS on Corridor-D to D-CULP advance phase, and initiate a ten-second wait to allow HCs to reach their respective CULPs;
- k. PHASE 11 - Command UPHCS on Corridor-A to walk down Corridor-A and advance phase;

- l. PHASE 12 - Command UPHCS on Corridor-B to walk down Corridor-B and advance phase.
- m. PHASE 13 - Command UPHCS on Corridor-C to walk down Corridor-C and advance phase.
- n. PHASE 14 - Command UPHCS on Corridor-D to walk down Corridor-D, advance phase, and initiate a ten-second wait.
- o. PHASE 15 - Command HCs at CLLP to STOW and advance phase.
- p. PHASE 16 - Initiate seven-minute delay to allow HCs going down the corridor to complete and advance phase.
- q. PHASE 17 - Command HCs at CLLP to STOW, initiate an eleven-minute wait and advance phase.
- r. PHASE 18 - Wait until delay is complete and advance phase.
- s. PHASE 19 - Force restart of HFC by disabling all clocks.

3.4.1.4.5.3.3 Internal Data Description

ECWSEQ creates, each second, a mask for use by ECWMG which is the logical OR of those heliostats commanded on CORRIDOR-A, CORRIDOR-B, CORRIDOR-C, CORRIDOR-D, and either to beam or AZ/EL point. ECWPHZ is the phase index which controls the ECW sequence.

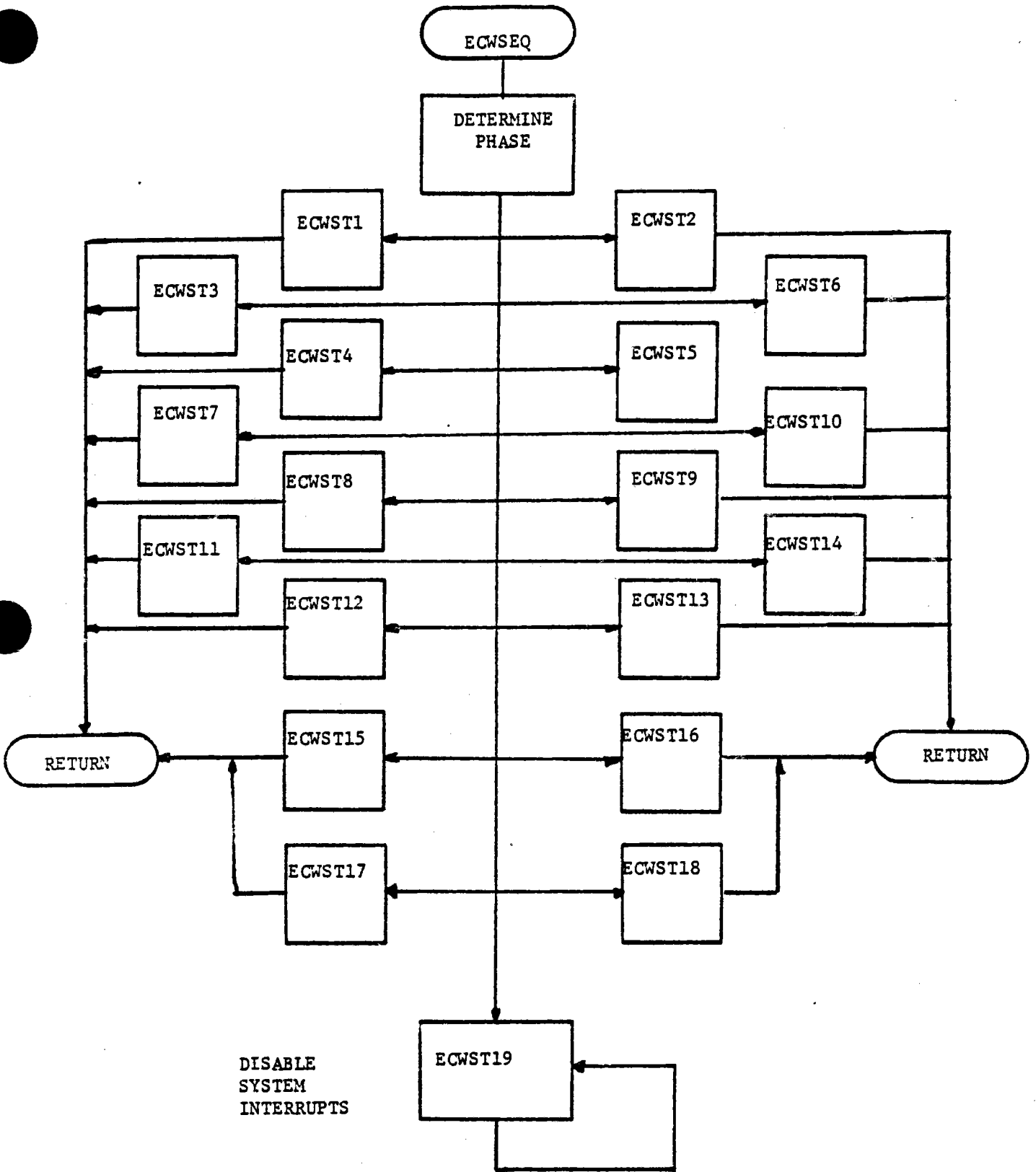
3.4.1.4.5.3.4 Flowcharts (see Figure 3.4.1-17)

3.4.1.4.6 Submodule V - HFC Timer Interrupt Handlers (HFCTMR)

3.4.1.4.6.1 Main Routine

3.4.1.4.6.1.1 Description

HFCTMR consists of two interrelated interrupt routines, HFCTOC and HFCTOF, which handle the MC6803's TOC and TOF timer interrupts. These routines maintain 833 usec and 50 msec granularity delay timers, respectively, for use by the HFC tasks. The routines count down the timers if active, and trigger the associated task when the count expires. HFCTMR is not a physical body of code but a logical grouping of two related functions.



DISABLE
SYSTEM
INTERRUPTS

Figure 3.4.1-17

Flowchart - ECWSEQ

3.4.1.4.6.1.2 Data, Logic and Command Paths

See HFCTOC (3.4.1.4.6.2.2) and HFCTOF (3.4.1.4.6.3.2).

3.4.1.4.6.1.3 Internal Data Description

See HFCTOC (3.4.1.4.6.2.3) and HFCTOF (3.4.1.4.6.3.3).

3.4.1.4.6.1.4 Flowcharts (see Figures 3.4.1-18 and 3.4.1-19).

3.4.1.4.6.2 HFCTMR Subroutine I - HFCTOC

3.4.1.4.6.2.1 Description

HFCTOC receives the MC6803's TOC (Timer Output Compare) interrupt when the TOC register contents matches the value of the MC6803's free running counter. HFCTOC adds 0400_{16} ($833 \mu\text{sec}$ at 1.2288 MHz) to the TOC register, subtracts one from all active timers, and triggers any tasks whose timers have just expired. HFCTOC then exits the interrupt level. HFCTOC's timers are used as I/O byte timers and I/O delay timers.

3.4.1.4.6.2.2 Data Logic and Command Paths

HFCTOC's input is the TOC interrupt and delay timer cell settings. HFCTOC's output is task triggers.

3.4.1.4.6.2.3 Internal Data Description

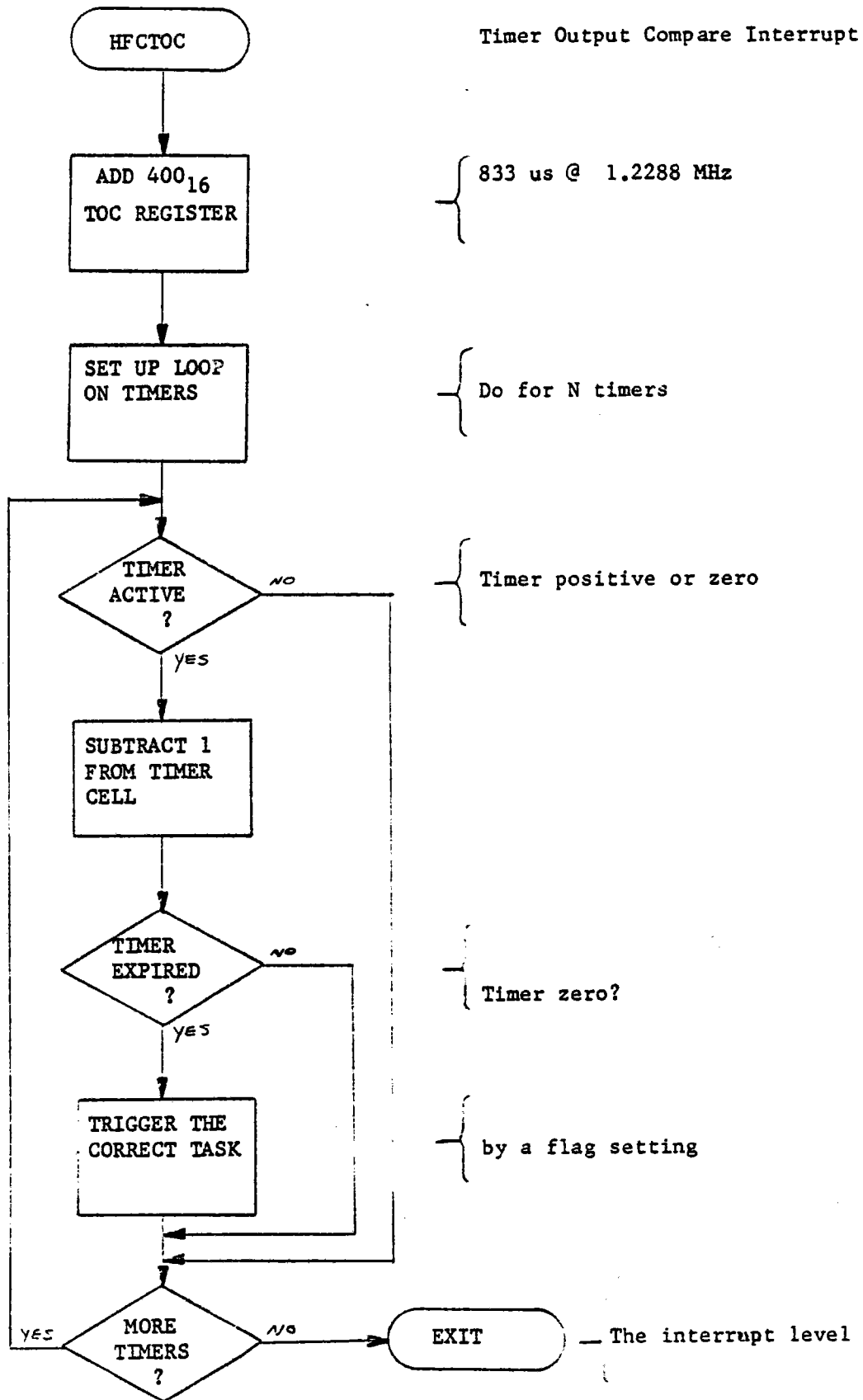
HFCTOC's timers are each one byte. A positive number indicates the timer is counting down. Zero means a task is to be triggered. Negative indicates an inactive timer.

3.4.1.4.6.2.4 Flowcharts (see Figure 3.4.1-18)

3.4.1.4.6.3 HFCTMR Subroutine II - HFCTOF

3.4.1.4.6.3.1 Description

HFCTOF receives the MC6803's TOF (Timer Over Flow) interrupt when the free running MC6803 counter goes from $FFFF_{16}$ to 0000_{16} . HFCTOF adds 1000_{16} to the counter to generate the 50 msec TOF interrupt (1000_{16} is an up count of 61440_{10} which at 1.2288 MHz is 50 msec). HFCTOF also adds 1000_{16} to the TOC register to resynchronize the TOC interrupt. Because the TOC is started out at 1200_{16} , this results in an adjustment from $\text{TOC} = 0200_{16}$ to $\text{TOC} = 1200_{16}$. HFCTOF then subtracts one from the sum vector timer and triggers CMDI if it has expired. HFCTOF then exits the interrupt level.



Flow Chart - HFCTOC

Figure 3.4.1-18

3.4.1.4.6.3.2 Data, Logic and Command Paths

HFCTOF's input consists of the MC6803 TOF interrupt, the TOC register, and the sun vector timer cell. HFCTOF's output is the trigger of CMDI.

3.4.1.4.6.3.3 Internal Data Description

HFCTOF's sun vector timer is a one-byte down counter. Positive indicates counting down, zero means task is to be triggered, and negative is past trigger until reset by CMDI.

3.4.1.4.6.3.4 Flowcharts (see Figure 3.4.1-19).

3.4.1.4.7 Submodule VI - HAC Input Interrupt Handler (HACIN)

3.4.1.4.7.1 Main Routine

3.4.1.4.7.1.1 Description

HACIN receives bytes from the HFC's ACIA device (essentially a UART) and stores them in the HAC input buffer until the HAC I/O byte timer expires indicating no byte received in approximately two I/O byte times. HACIND, a subroutine executed when I/O timer expires, HACIN also accumulates a checksum while receiving the message. HACIN resets the byte timer each time a new byte is received. HAC input start-up is accomplished by setting HACIN's I/O byte count positive. HACIN stores message bytes until its byte count goes negative, zeroing the checksum and resetting the store address. Reading bytes and accumulating checksum will continue until I/O byte time-out, even though no bytes are stored.

3.4.1.4.7.1.2 Logic, Data and Command Path

HACIN's input is message bytes, one byte per interrupt and the I/O byte timer. HACIN's output is the received message, byte-count and checksum accumulation.

3.4.1.4.7.1.3 Internal Data Description

HACIN's internal data consists of a byte-count, transfer address, and checksum accumulator.

3.4.1.4.7.1.4 Flowcharts (see Figure 3.4.1-20 and Figure 3.4.1-21).

HFCTOF

Timer Overflow Interrupt

ADD 1000₁₆ TO BOTH THE
FREE RUNNING COUNTER
AND TOC REGISTER

{ 61440₁₀ up count is 50 ms @ 1.2288 MHz

SUBTRACT 1
FROM THE SUN
VEC TIMER

{ Timer is reset positive by CMDI

EXPIRED ?

{ zero?

TRIGGER
CMDI

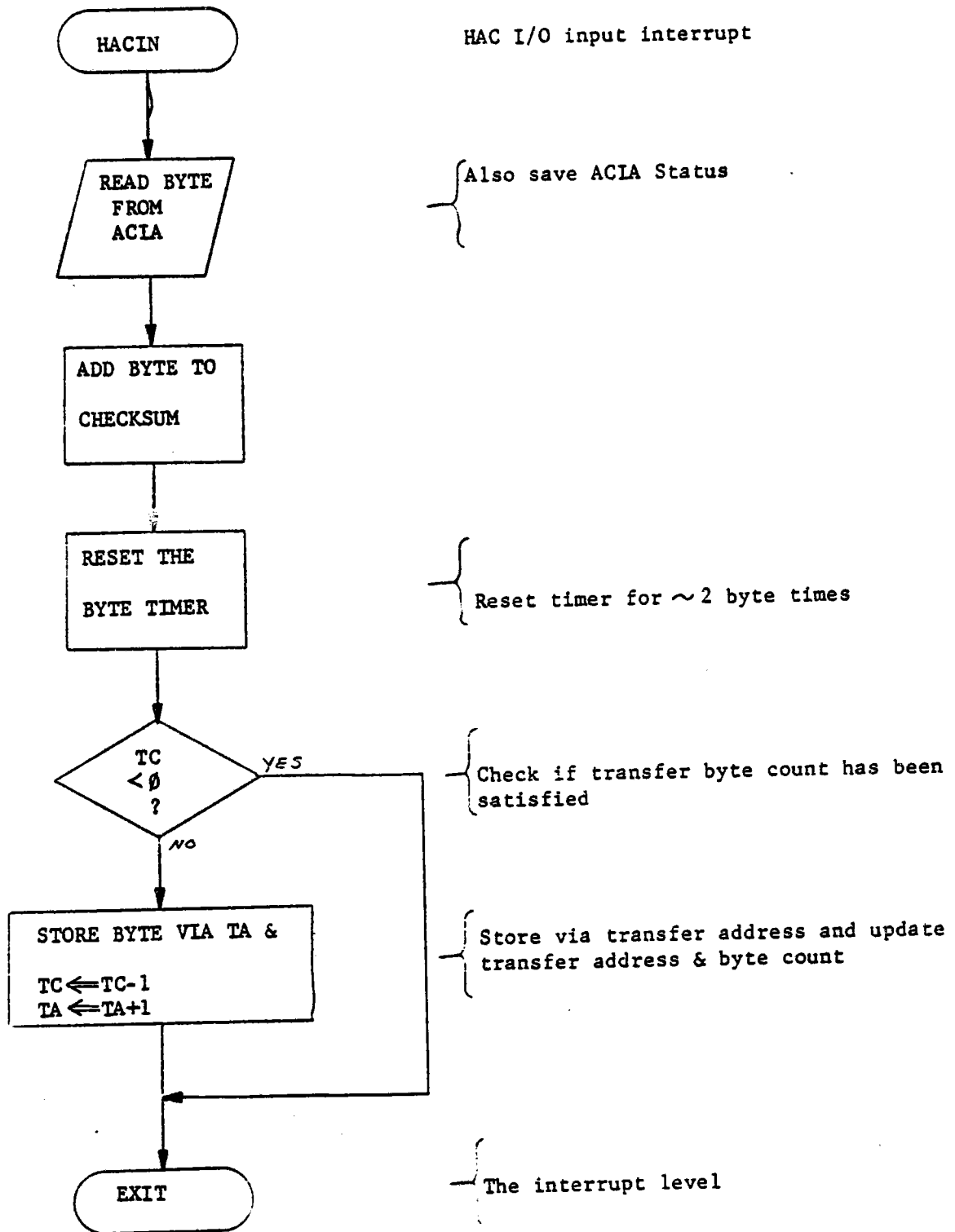
{ by a flag settling

EXIT

{ the interrupt level

Flowchart - HFCTOF

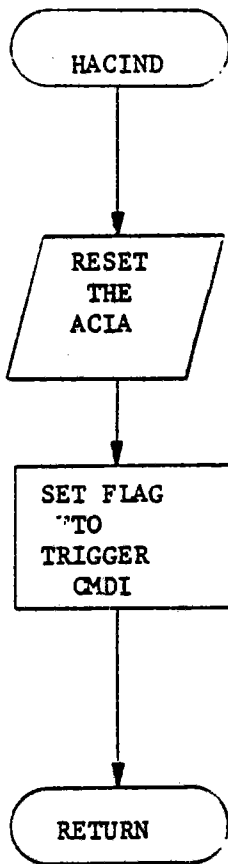
Figure 3.4.1-19



(CONTINUED)

Flowchart - HACIN

Figure 3.4.1-20



HAC Input Timer Time-Out

{ Clean up any end conditions on the channel

{ To HFCTOC

Flowchart - HACIND

Figure 3.4.1-21

3.4.1.4.8 Submodule VII - HAC Output Interrupt Handler (HACOUT)

3.4.1.4.8.1 Main Routine

3.4.1.4.8.1.1 Description

HACOUT takes a message buffer created by another HFC task and outputs a byte at a time to the HFC's ACIA device (essentially a UART) when the "transmitter buffer empty" interrupt is received. HACOUT accumulates a checksum and outputs it as the last message byte. HACOUT retriggers the user task after the checksum has been output. HAC output startup is accomplished by setting HACOUT's byte count, transfer address, and checksum and then writing the first byte to the ACIA.

3.4.1.4.8.1.2 Logic, Data and Command Paths

HACOUT's input is the message buffer from the user task. HACOUT's output is a byte at a time to the ACIA.

3.4.1.4.8.1.3 Internal Data Description

HACOUT's internal data is the byte count, transfer address and checksum.

3.4.1.4.8.1.4 Flowcharts (see Figure 3.4.1-22).

3.4.1.4.9 Submodule VIII - HC Input Interrupt Handler (HCIN)

3.4.1.4.9.1 Main Routine

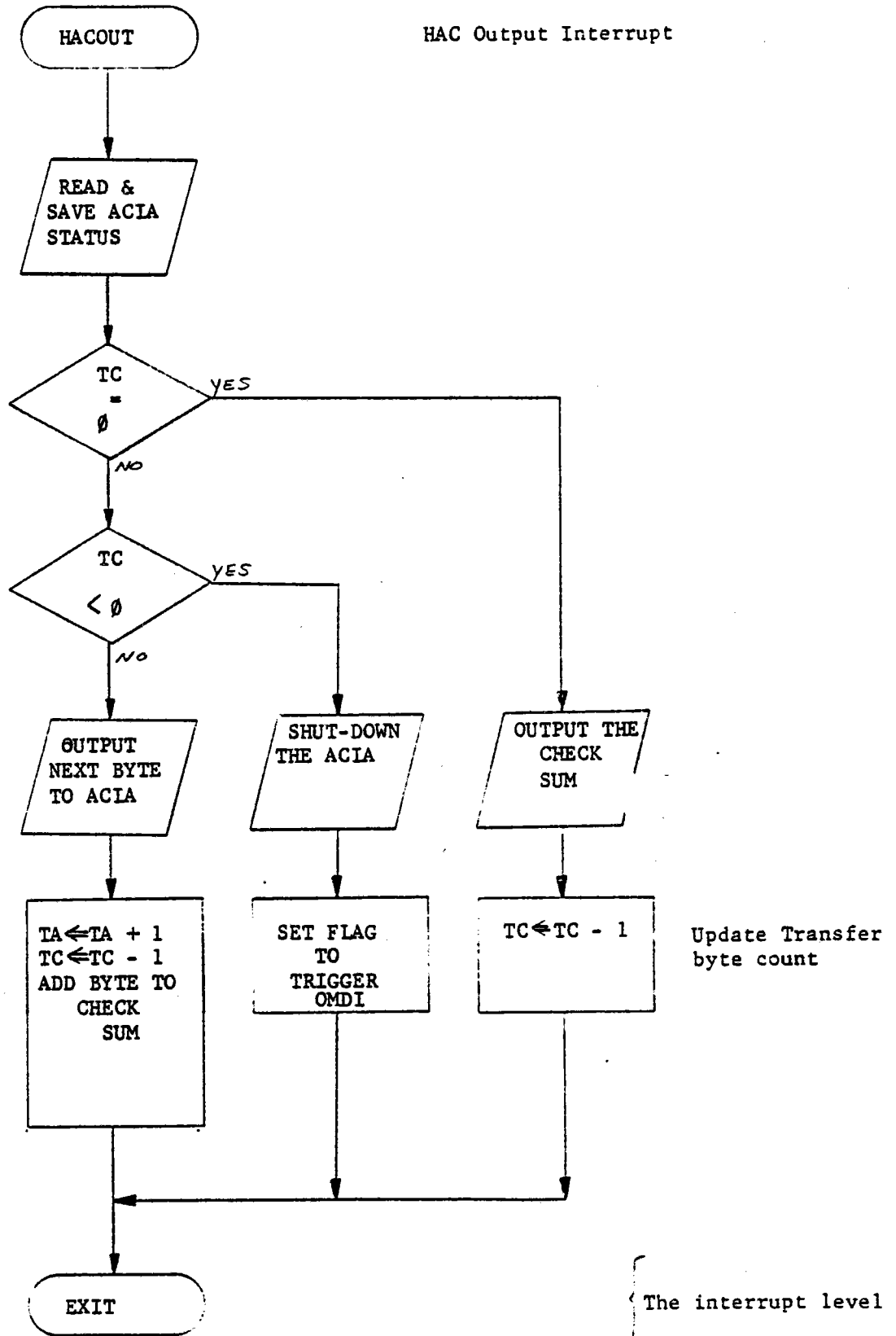
3.4.1.4.9.1.1 Description

HCIN receives bytes from the MC6803's internal UART and stores them in the HC input buffer until the HC I/O byte timer expires indicating no byte received in two I/O byte times. HCIND, a subroutine executed when I/O timer expires, then retriggers the user task. HCIN also accumulates a checksum while receiving the message. HCIN resets the byte timer each time a byte is received. HC input start-up is accomplished by setting HCIN's I/O byte count positive, zeroing the checksum, and resetting the store address. HCIN stores message bytes until the byte count goes negative. Reading bytes and accumulating checksum continues until I/O byte time-out even though no bytes are stored.

3.4.1.4.9.1.2 Logic, Data and Command Paths

HCIN's input is a message byte per interrupt and the I/O byte timer. HCIN's output is the received message, byte count, and checksum accumulation.

HAC Output Interrupt



Flowchart - HACOUT

Figure 3.4.1-22

3.4.1.4.9.1.3 Internal Data Description

HCIN's internal data consists of a byte count, transfer address, and checksum accumulator.

3.4.1.4.9.1.4 Flowcharts (see Figures 3.4.1-23 and 3.4.1-24).

3.4.1.4.10 Submodule IX - HC Output Interrupt Handler (HCOU)

3.4.1.4.10.1 Main Routine

3.4.1.4.10.1.1 Description

HCOU takes a message buffer created by another HFC task and outputs a byte at a time to the MC6803's internal UART when the "transmitter buffer empty" interrupt is received. HCOU accumulates a checksum and outputs it as the last message byte, and then retriggers the user task. HC output startup is accomplished by setting HCOU's bytes count, transfer address, and checksum and then writing the first byte to the internal UART.

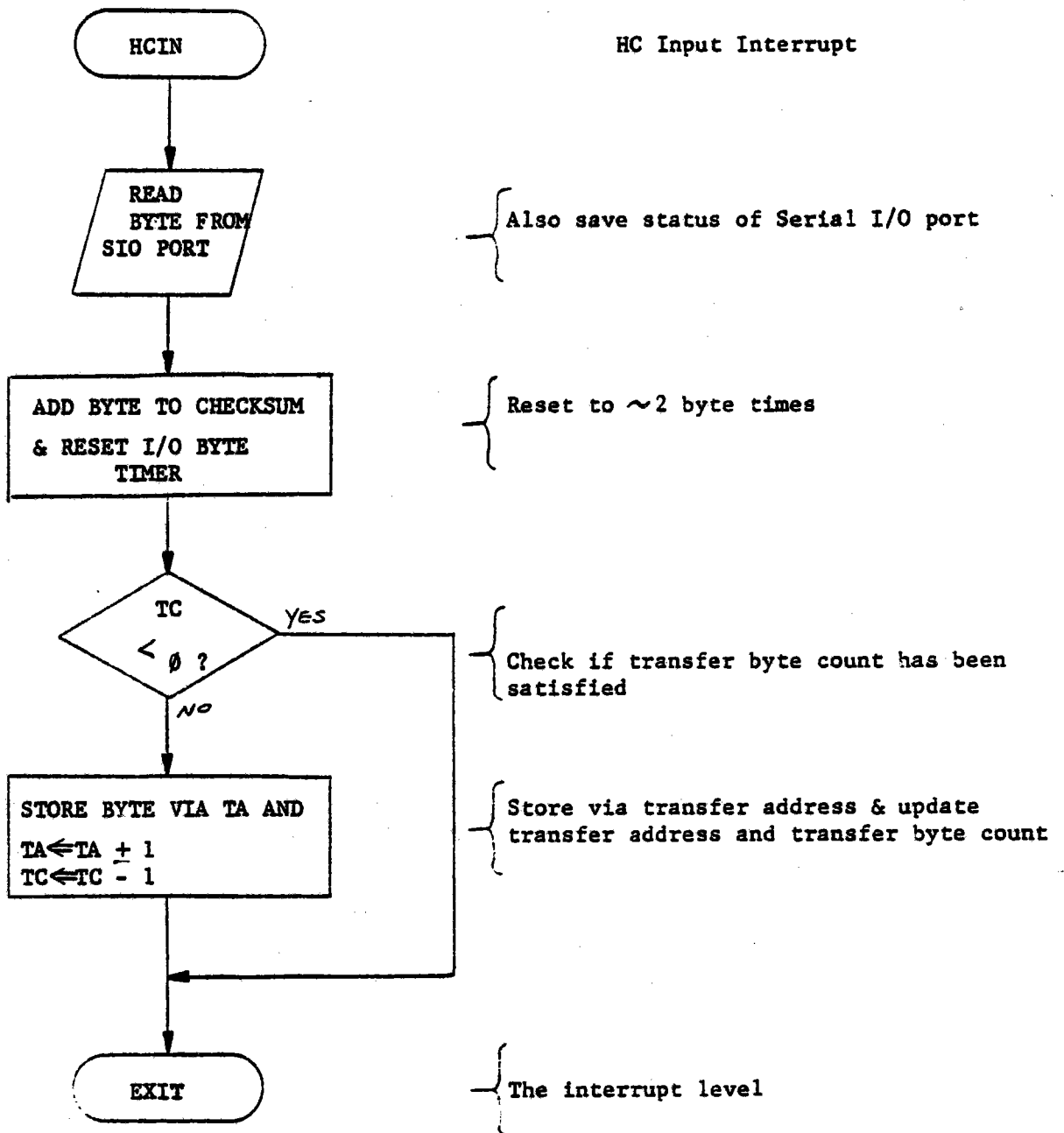
3.4.1.4.10.1.2 Logic, Data and Command Paths

HCOU's input is the message buffer from the user task. HCOU's output is a byte at a time to the MC6803's internal UART.

3.4.1.4.10.1.3 Internal Data Description

HCOU's internal data is the byte count, checksum, and transfer address.

3.4.1.4.10.1.4 Flowcharts (see Figure 3.4.1-25).



(CONTINUED) -

Flowchart - HCIN
Figure 3.4.1-23

HCIND

HC Input Timer Time-out

RESET THE
SERIAL
I/O PORT

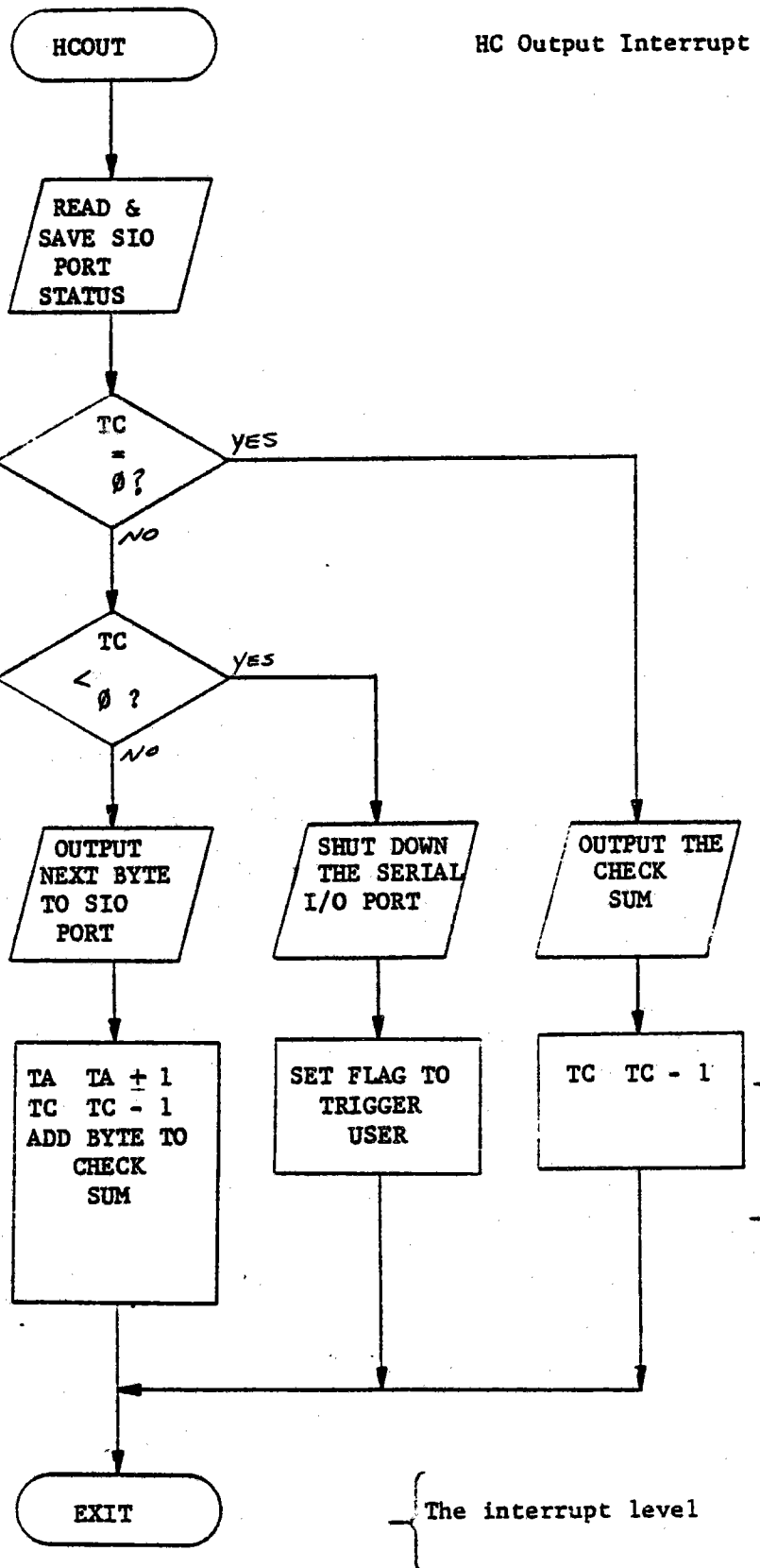
{ Clean up any end conditions on the channel

SET FLAG TO
TRIGGER USER
TASK

RETURN

{ To HFCTOC

Flowchart - HCIND
Figure 3.4.1-24



Flowchart - HCOUT

Figure 3.4.1-25

3.5 HC Firmware Design

3.5.1 HC Firmware Module - HCMOD

3.5.1.1 Purpose

The following is a description of the Operation of the Heliostat Controller Firmware, hereafter referred to as HCMOD. HCMOD consists of the following submodules:

- a. INIT - System initialization;
- b. SYSCLK - System clock and task activation control;
- c. POINT - Encoder data acquisition and motor control;
- d. SCIO - Serial communications handler; and
- e. CALC - Control Algorithm.

3.5.1.2 Requirements

3.5.1.2.1 Design Requirements

Section 3.1 of the Software/Firmware Functional Requirements requires the following of the HC:

- a. Determine heliostat azimuth and elevation position requirements;
- b. Control gimbals drive motors;
- c. Check gimbal axis sensors; and
- d. Provide gimbal axis position data to HAC.

3.5.1.2.2 Derived Requirements

HC firmware derived requirements are as follows:

- a. Maintain communications with the HFC, detecting communications errors;
- b. Calculate the desired azimuth and elevation angles based on sun vector and operational commands from the HFC;
- c. Control the azimuth and elevation motors to achieve the desired position;
- d. Maintain the azimuth and elevation encoder interfaces to provide gimbal angle information;
- e. Provide HC status information to the HFC upon a poll command;

- f. Execute an automatic reset upon detection of major errors; and
- g. Detect errors of heliostat operation (i.e., in-operative motors).

3.5.1.3

Design Approach

The following is intended to show the relationship of the various HCMOD submodules to system requirements (see Figure 3.5.1-1).

3.5.1.3.1

Functional Allocation

INIT - This submodule, activated by SYSTEM RESET, initiates system restart operations. System operating configuration is set, and internal registers and system memory locations are initialized (see Figure 3.5.1-2).

SYSCLK - This submodule, activated by a real-time clock, controls the time-critical functions of the HC. Time critical functions include the following:

- a. HC to HFC command response;
- b. Motor direction switching delays;
- c. Data acquisition; and
- d. Communications time-outs.

Non time-critical functions, such as command decoding, are enabled for activation by this submodule (see Figure 3.5.1-3 and Figure 3.5.1-4).

POINT - This submodule, activated by SYSCLK, is responsible for the real-time data acquisition of the gimbal angle incremental encoders.

Motor control is also a function of this submodule. Motor speed control is accomplished by comparison of commanded or calculated position and actual gimbal position error. If this error is significant (greater than eight bits), Hi speed operation is initiated. If the opposite condition exists, Lo speed operation is then initiated (see Figures 3.5.1-5, 3.5.1-6, and 3.5.1-7).

SCIO - This submodule, enabled by SYSCLK, handles serial communication with the HFC. Activation of this submodule is accomplished by receiver and transmitter interrupts. These interrupts, if enabled by SYSCLK, are vectored into SCIO directly.

HC SYSTEM MEMORY MAP

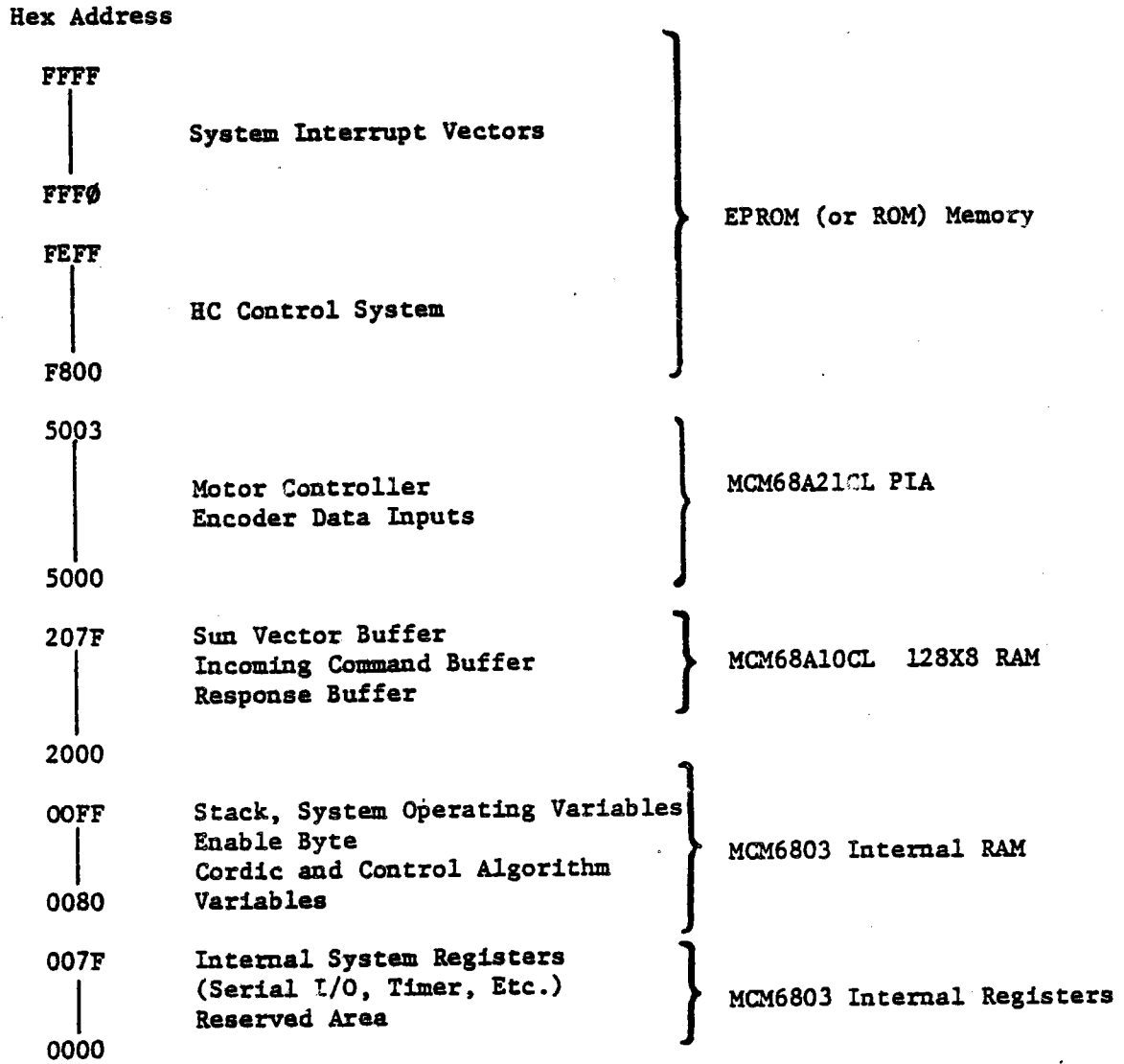
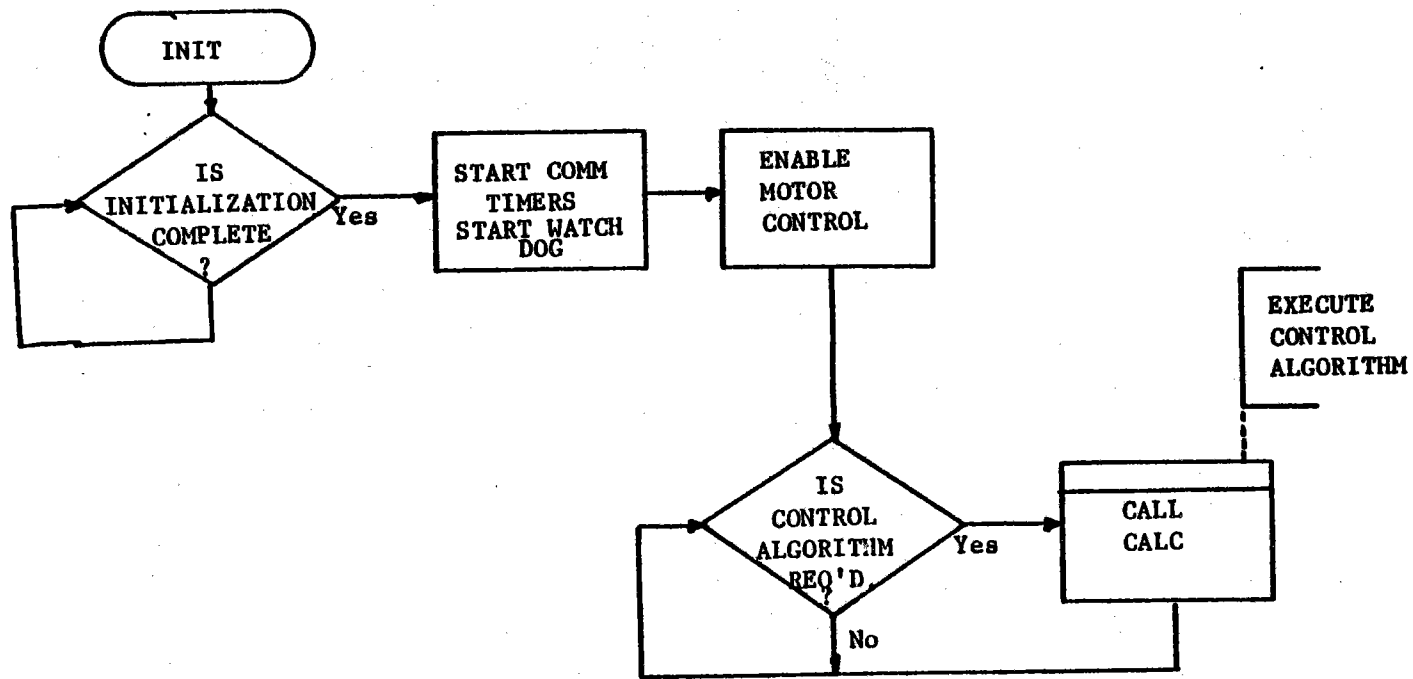


Figure 3.5.1-1

BACKGROUND TASKS



933

Figure 3.5.1-2 Flowcharts - INIT

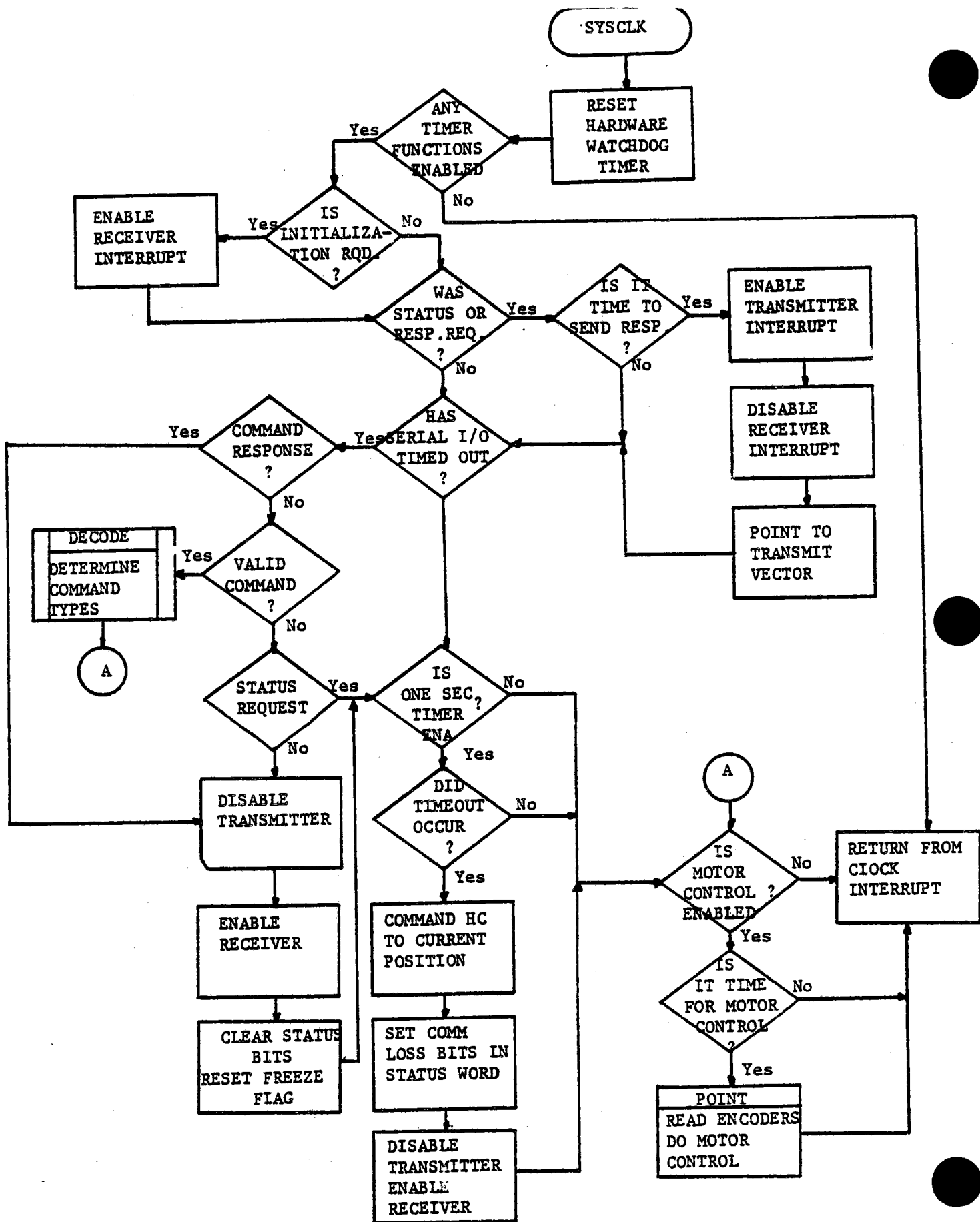


Figure 3.5.1-3 SYSCLK
934

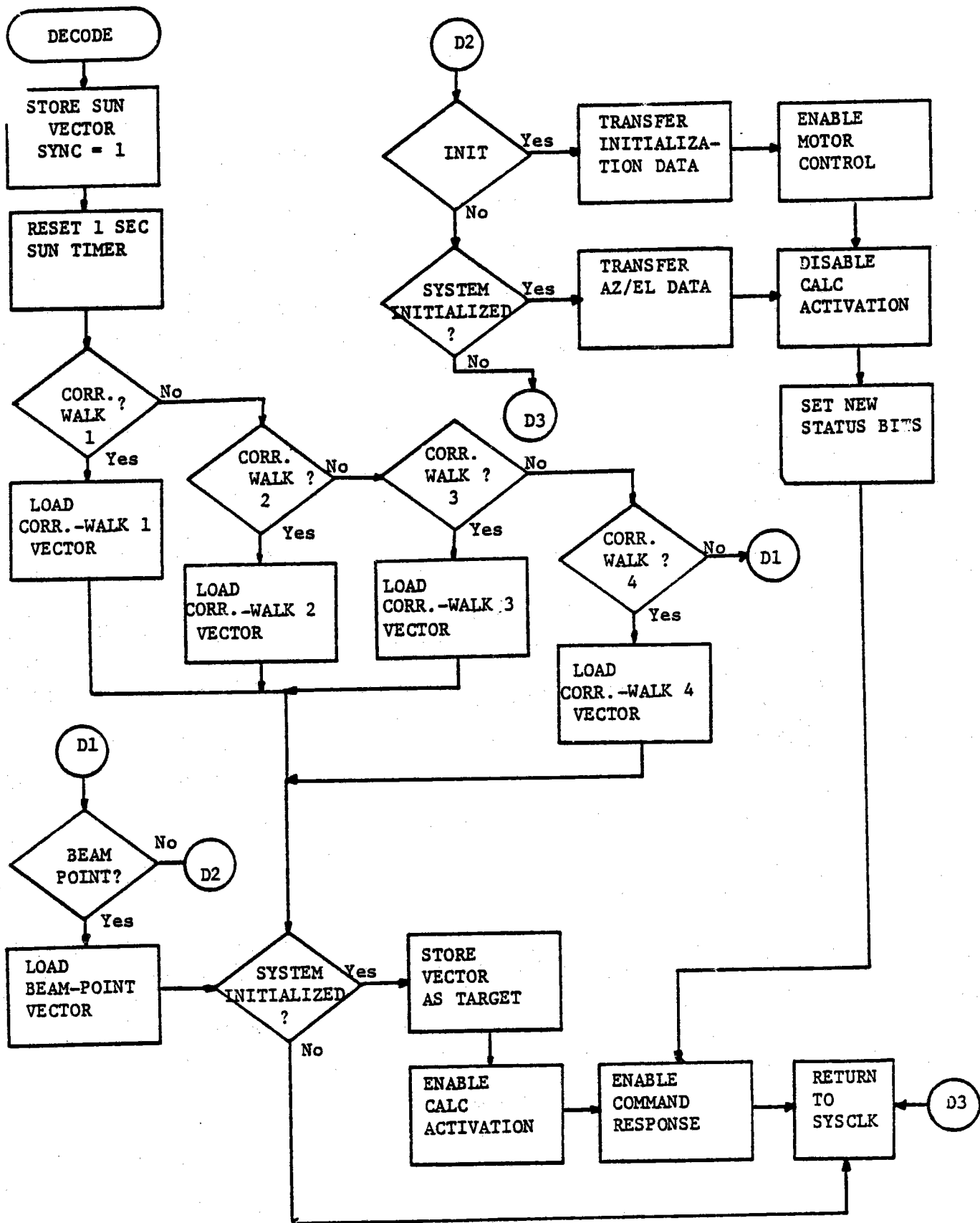


Figure 3.5.1-4 Flowchart - DECODE

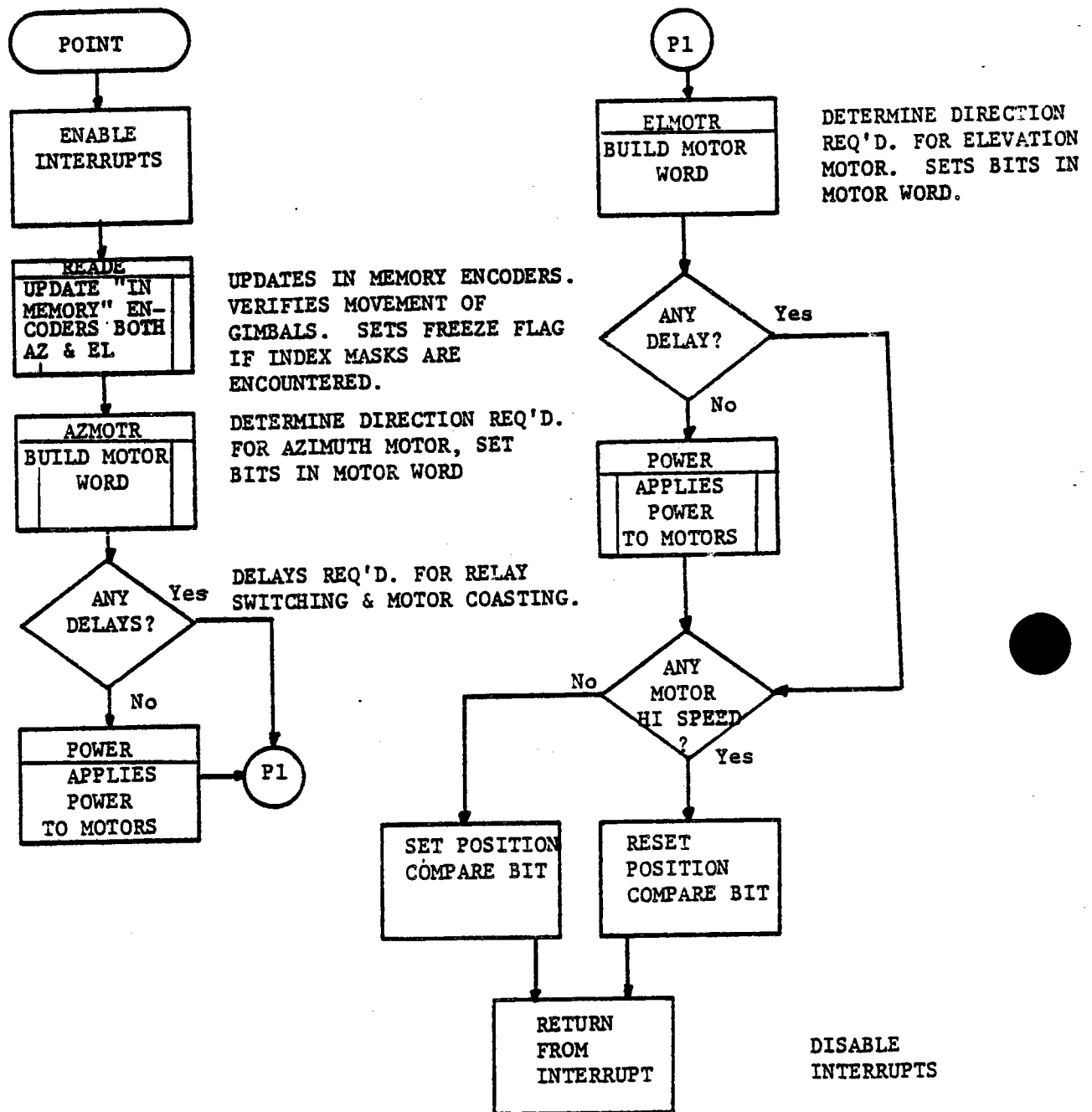


Figure 3.5.1-5 Flowchart - POINT

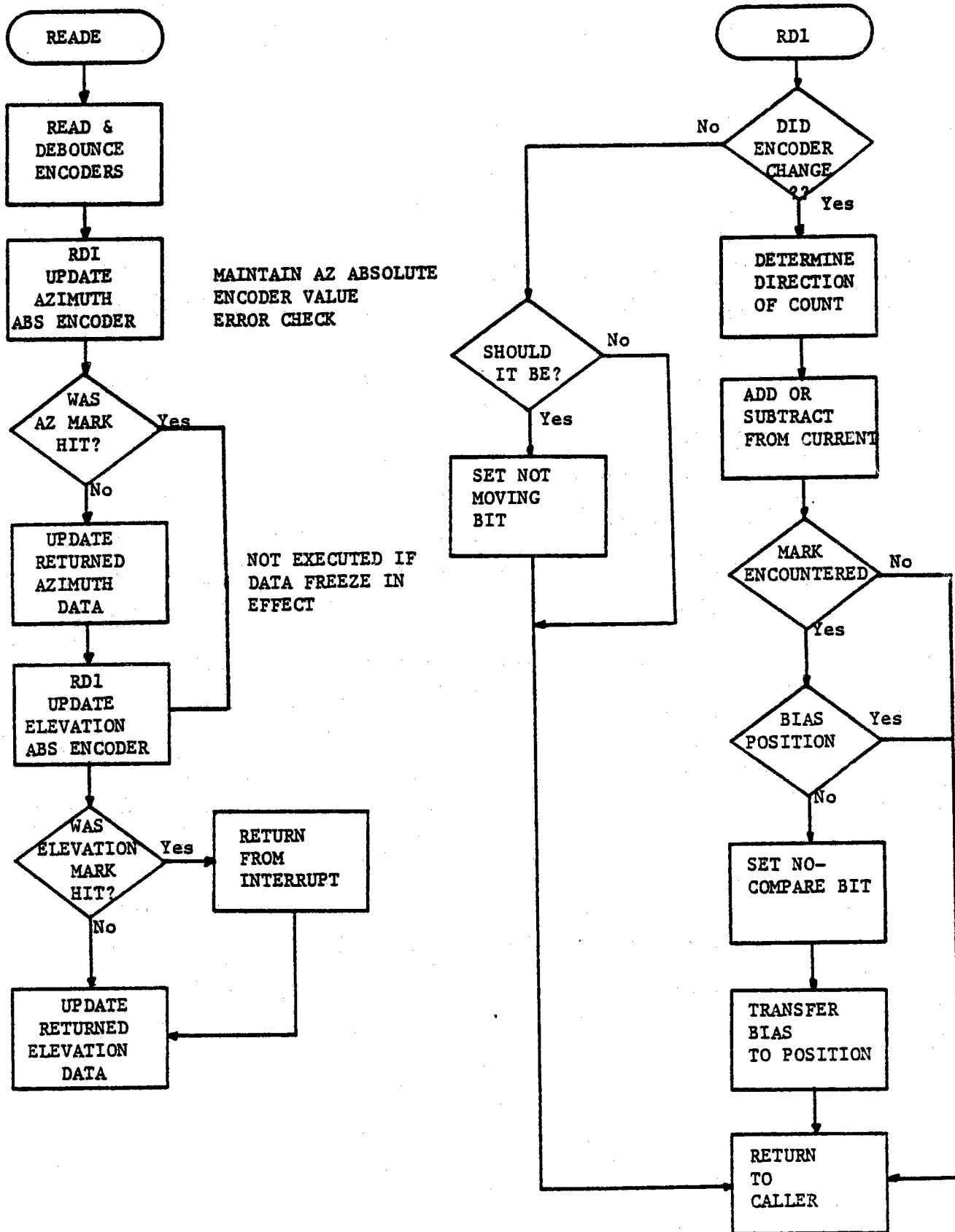


Figure 3.5.1-6. Flowchart - READE
937

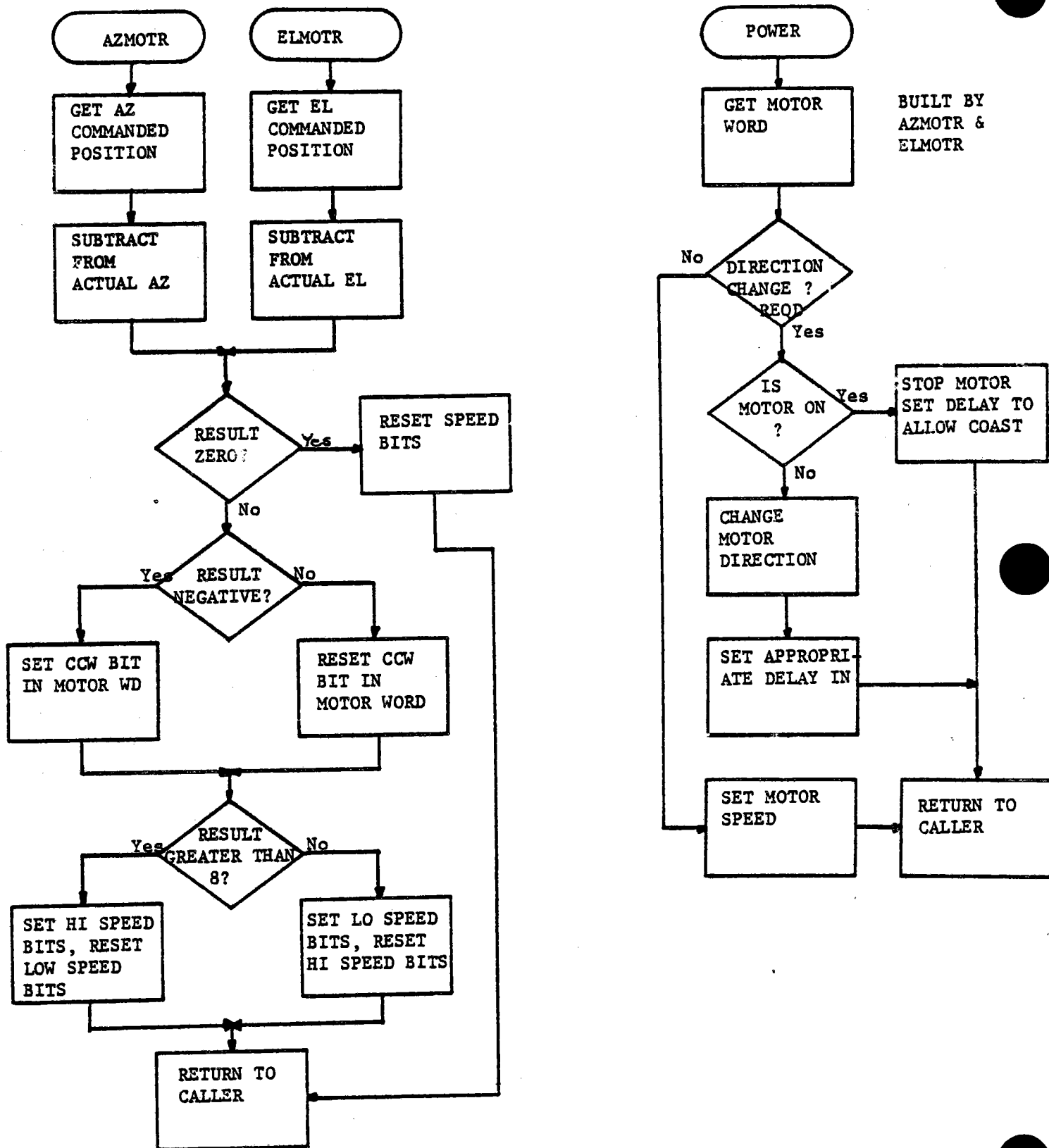


Figure 3.5.1-7 Flowchart - AZMOTR, ELMOTR, POWER

Data checksum and communication error detection (framing and overrun errors) are also handled in this submodule. Communication line turnaround is also a function of this submodule (see Figure 3.5.1-8).

CALC - This submodule, activated by SCIO, is a Background Task enabled for operation when a new sun vector is received from the HFC. This submodule contains the Cordic Positioning Algorithm (see Figure 3.5.1-9).

3.5.1.3.2

Resource Budgets

INIT	--	91 Bytes
SYSCLK	--	469 Bytes
POINT	--	457 Bytes
SCIO	--	263 Bytes
CALC	--	752 Bytes
VECTORS	--	<u>16 Bytes</u>
Total		2,048 Bytes Read-Only Program Memory

SYSTEM STACK	--	35 Bytes
SYSTEM VARIABLES	--	44 Bytes
CONTROL ALGORITHM	--	49 Bytes
COMMAND AND RESPONSE BUFFERS	--	<u>121 Bytes</u>

249 Bytes Read/Write Data Memory

3.5.1.3.2.1

Submodule Priority

Basic operating priority is as follows:

INIT
SYSCLK

SCIO
POINT
CALC

3.5.1.3.3

Operating System Variables

See Table 3.5.1-I.

3.5.1.4

Design Description

The HCMOD is a complete, stand-alone, firmware package designed to reside in 2048 bytes of mask programmed read-only memory.

HCMOD is essentially a foreground - background system designed to take advantage of the vectored interrupt capability of the MCM6803 microcomputer.

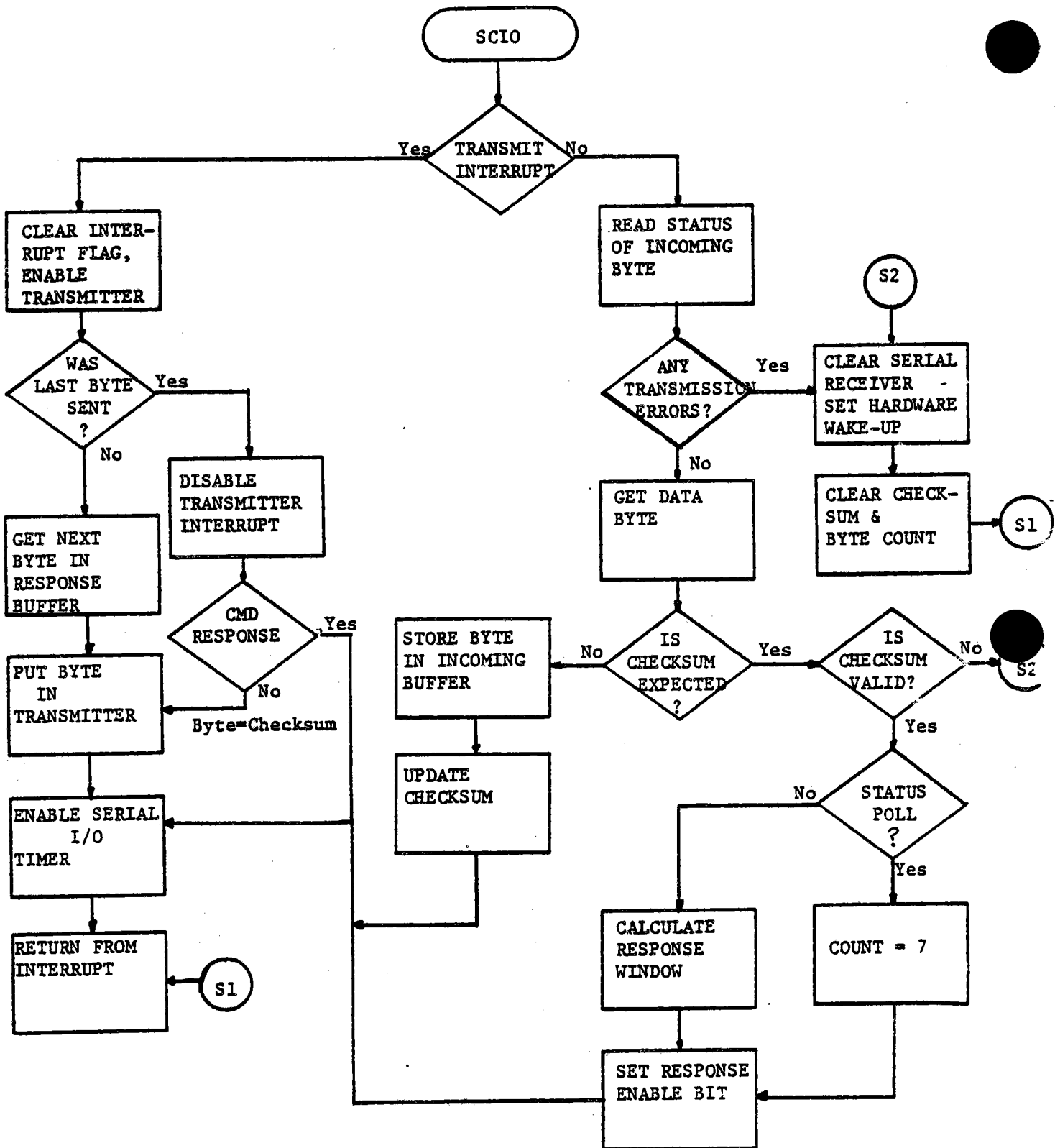
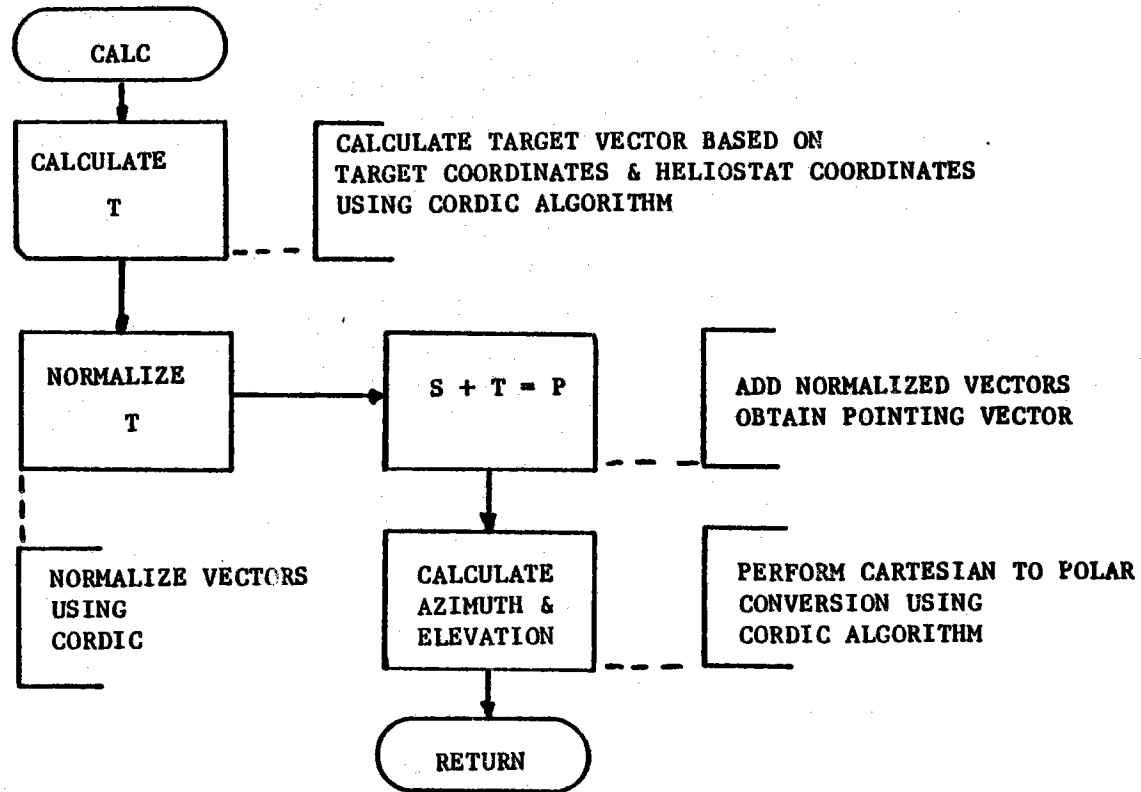


Figure 3.5.1-8 SCIO

POSITION CALCULATIONS



941

Figure 3.5.1-9 Flowcharts - CALC

HC SYSTEM OPERATING VARIABLES

VARIABLE	HEX LOCATION	SIZE IN BYTES
ENABLE	00A3	1
BIT BREAKDOWN		
LSB	BIT 0 - 1 = New Sun Vector Received and Queued	
	BIT 1 - 1 = Message Length Timer Enabled	
	BIT 2 - 1 = Comm Loss (ISEL) Timer Enabled	
	BIT 3 - 1 = Status Response Enabled	
	BIT 4 - 1 = System Requests Initialization	
	BIT 5 - Spare	
	BIT 6 - 1 = Control Algorithm Enabled	
MSB	BIT 7 - 1 = Motor Control Enabled	
TRXOK	00A4	1
	Loaded with calculated value that determines when HC responds with Command Response or Status -- when "0" transmitter is enabled.	
SLOSS	00A5	1
	Loaded at receipt of incoming byte, also loaded at time of outgoing byte -- when "0" Command Decode is initiated or transmitter operation is disabled.	
CLOSS	00A6	2
	Loaded at Sun Vector Receipt Time - when "0" HFC Comm Loss has been detected -- HC loads current position into Az and El Buffers and indicates Comm Loss Via Status Bits - Motor Control still active.	
NEXVEC	00A8	2
	Used by SCIO Routine to control entry point of Receiver Buffer Full Interrupt.	
IOVEC	00AA	
	Used by SCIO Routine in conjunction with "NEXVEC" to determine entry point for Receiver Buffer Full and Transmitter Buffer Empty Interrupts.	

Table 3.5.1-I

HC SYSTEM OPERATING VARIABLES (Cont'd.)

VARIABLE	HEX LOCATION	SIZE IN BYTES
BYTECT	00AC	1
Used by SCIO Routine to count bytes of incoming and outgoing messages used with "CHARS".		
CHARS	00AD	1
Used by SCIO Routine to count actual message bytes received and sent (including Checksum).		
CKSM	00AE	1
Accumulates bytes for message checking - compared with received Checksum byte to determine message validity.		
MASK	00AF	1
8 Bit MASK with 1 bit set - used to determine if received beam pointing, Az, El, or Corridor Walk Command is valid for HC - Set bit position determined by Heliostat Address.		
MCOUNT	00B0	1
Counter loaded at motor control time - used to determine sample time of Encoders and Motor Control operation - Value currently loaded equals 2.5 ms.		
ACOUNT	00B1	(Azimuth) 2
ECOUNT	00B3	(Elevation) 2
Counters loaded at motor power time - Various counts appear in these locations based on delay time, required for Relay Switching, Motor Const, Direction Change.		

Table 3.5.1-I (continued)

HC SYSTEM OPERATING VARIABLES (Cont'd.)

VARIABLE	HEX LOCATION	SIZE IN BYTES
OLDAZ	00BB	1
Contains old 2 Bit Reading used to determine Direction of Count for Azimuth Encoder.		
ASTUCK	00BC	2
Used to determine length of time Heliostat is not moving - Not necessarily an error condition.		
AZABS	00BE	2
Actual absolute Azimuth Encoder position - Updated during read of Azimuth Encoder.		
AZBAZ	00CO	2
Downloaded Heliostat Azimuth Encoder Bias.		
FRZAZ	00C2	1
Set to non-zero value by Index Mark Interrupt Handler - Indicates AZ Mark Encountered. Moves Bias Value (AZBZ) to Position Buffer (AZABS).		
AZNC	C3	1
AZCP	C4	1
Variables used in Compare/No Compare Logic.		
OLDEL	00C5	1
ESTUCK	00C6	2
ELABS	00C8	2
ELBAZ	00CA	2
FRZEL	00CC	1
ELNC	00CD	1
Similar to above variables - for Elevation Motor Control.		
ELCP	00CE	1

HC SYSTEM OPERATING VARIABLES (Cont'd.)

VARIABLE	HEX LOCATIONS	SIZE IN BYTES
MOTWD	00B5	1
<p>Bit positions in this memory word correspond directly to bit positions in Motor Controller Output Register. This motor word is built by point routine before actually powering motors.</p>		
MCODE	00B6	1
<p>Parameter used by point routines to determine which motor is being acted upon.</p>		
ADDR	00B7	1
<p>Used to store Heliostat Address.</p>		
MSAVE	00B8	1
<p>Temporary location to store current Motor On/Off Dir Status while altering.</p>		
FREEZE	00B9	1
<p>Indicates Status Poll has not yet been received. Disables update of returned Heliostat position.</p>		
SYNC	00BA	1
<p>Indicates a new Sun Vector has been received - Initiates Control Algorithm activation in background.</p>		
<p>Locations 00CF - 00FF are variables used by Control Algorithm.</p>		

Table 3.5.1-I (continued)

HC SYSTEM OPERATING VARIABLES (Cont'd.)

VARIABLE	HEX LOCATION	SIZE IN BYTES
SUNX	2000	3
SUNY	2003	3
SUNZ	2006	3
Contain Downloaded Sun Vector Information.		
CMDT	2012	1
Contains first byte of last message command type and/or HC address.		
CMD5	2013	9
Latest Sun Vector sent from HFC.		
CMDW1M	2013	4
Latest Corridor-Walk 1 Mask.		
CMDW1V	2020	9
Latest Corridor-Walk 1 Vector.		
CMDW2M	2029	4
Latest Corridor-Walk 2 Mask.		
CMDW2V	202D	9
Latest Corridor-Walk 2 Vector.		
CMDW3M	2036	4
Latest Corridor-Walk 3 Mask.		
CMDW3V	203A	9
Latest Corridor-Walk 3 Vector.		
CMDW4M	2043	4
Latest Corridor-Walk 4 Mask.		
CMDW4V	2047	9
Latest Corridor-Walk 4 Vector.		
CMD	2050	
Latest Beam Point Mask if Beam Pointing. Latest AZ-EL Mask if being positioned. Azimuth and Elevation Commanded position.		
VECTOR	2058	9
Beam Pointing Vector if command was Point, Heliostat Vector if command was Initialization.		

The above variables are referred to as the Incoming Command Buffer.

Table 3.5.1-I (continued).

HC SYSTEM OPERATING VARIABLES (Cont'd.)

VARIABLE	HEX LOCATIONS	SIZE IN BYTES
HEADER	2061	1
Contains Heliostat Address and Mode Bits.		
STATUS	2062	2
Contains Heliostat Status Information.		
HZPOS	2064	2
ELPOS	2066	2
Contains Azimuth and Elevation Position data Freeze is set, contains position at marks.		
The above variables are referred to as Response Buffer.		
AZB2	2068	2
ELBZ	206A	2
Bias Values.		
HELIX	2066	3
HELIY	206F	3
HELIZ	2072	3
Current Heliostat Vector if system is Initialized.		
CMDAZ	2075	2
CMDEL	2077	2

These locations are loaded by Azimuth-Elevation Commands or by the Control Algorithm upon completion of its calculation (if Enabled). The Point Routine uses this information to position the Heliostat.

These locations are loaded with the current Heliostat position if loss of communications occurs.

Table 3.5.1-I (continued)

3.5.1.4.1

Model Structure

Refer to Figure 3.5.1-10, System Block Diagram and Figure 3.5.1-11, System State Diagram.

On power up, initiated by the RESET interrupt, the system sets parameters, I/O devices, and system status tables. The system watchdog timer and system clock interrupts are enabled. At this time the HC will accept HC initialization and status commands.

Once the initialization command has been decoded and accepted by the serial I/O handler (SCIO), motor control (POINT) activation is enabled. Motor power, direction, and speed are determined from the comparison of calculated or commanded positions and actual gimbal positions. Previous encoder readings are stored in memory to determine the existence of motor or encoder error conditions. Activation is accomplished during the servicing of the system clock interrupt (SYSCLK).

During the command checking process, performed by the serial communications handler, command response data is queued and the proper HC response window is calculated. Transmitter operation is initiated when the system clock routine determines that the proper delay period has elapsed.

Upon the receipt of a valid beam pointing or corridor walk command, HC target vectors are updated and the control algorithm (CALC) is enabled for activation.

The following system state diagram shows the interrelationships of the firmware system.

System State Table

State Number	State Description
1	Waiting to be initialized
2	Initialized, no action
3	Timer happens
4	Motor control started
5	Motor control running
6	Serial character out
7	Serial character in
8	Command decoded
9	Control algorithm

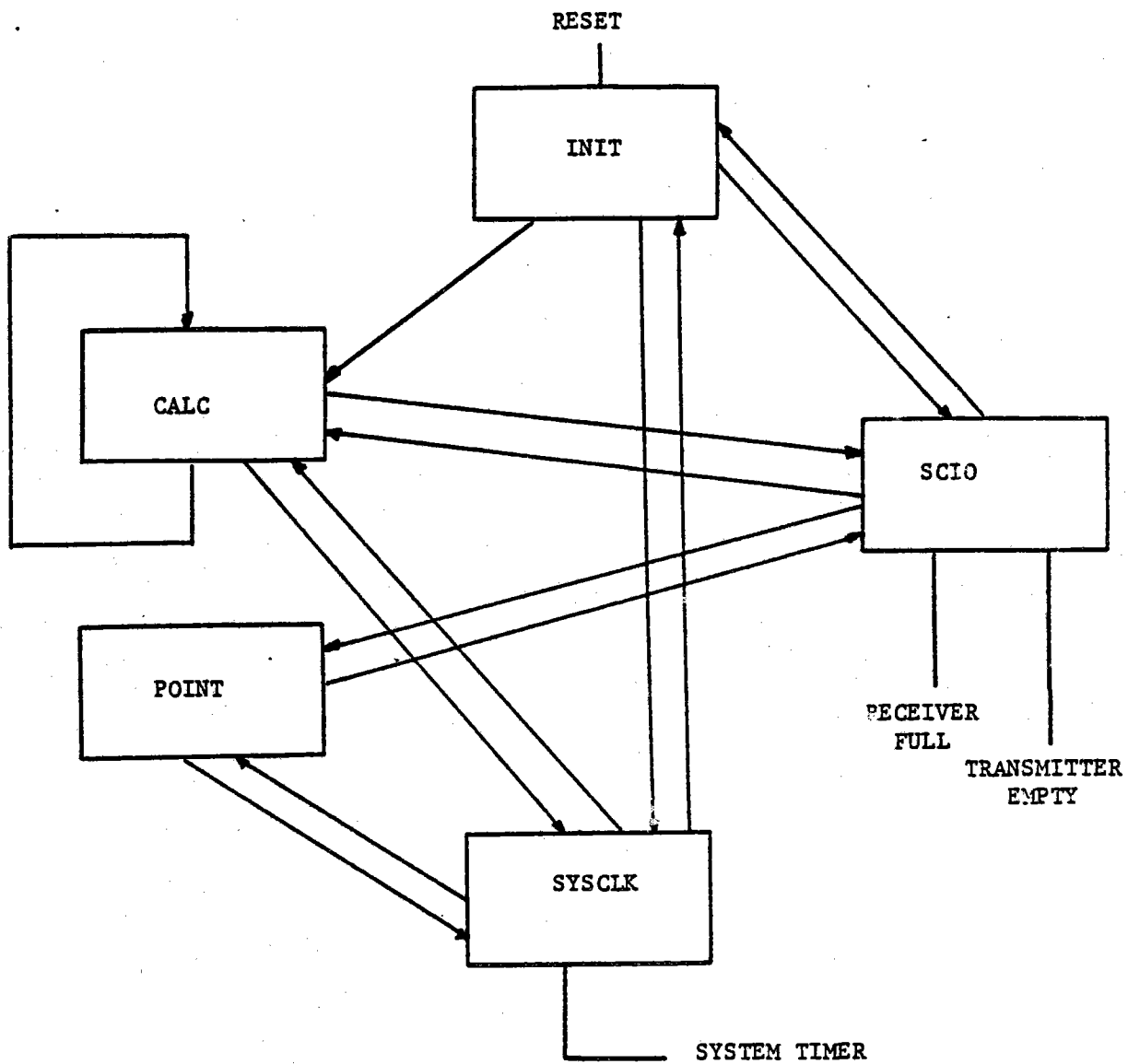


Figure 3.5.1-10

System Block Diagram

SYSTEM STATE DIAGRAM

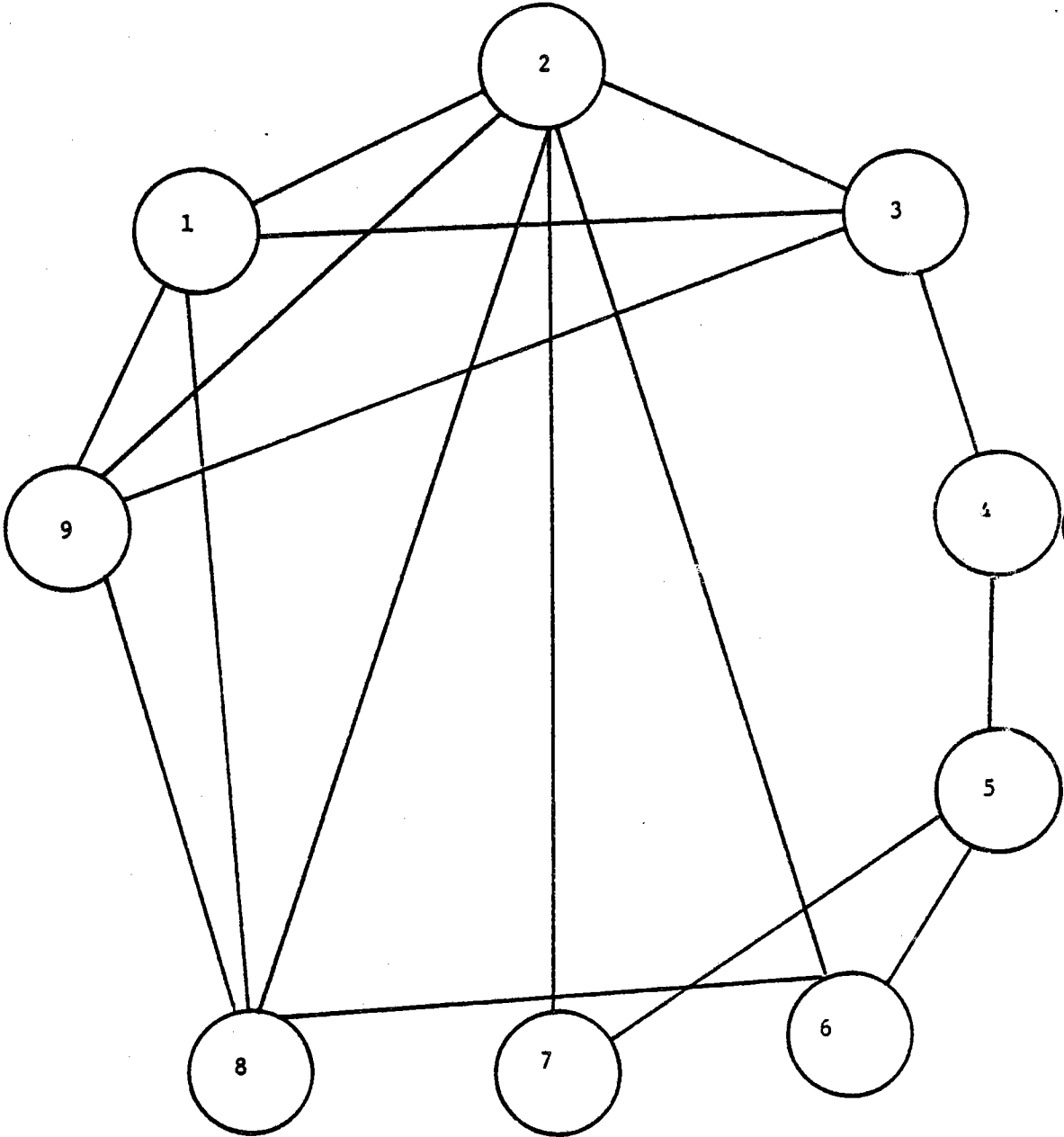


Figure 3.5.1-11 System State Diagram

3.5.1.5

Interface Description

A complete interface description is detailed in Section 3.6.2.

3.5.1.6

Test Requirements

- a. Verification of HCMOD position algorithm will be accomplished by the use of the SIGMA 5 M6801 Software Simulator.
- b. Verification of HFCMOD and HCMOD communication timing will be accomplished by logic analyzer and oscilloscope inspection.
- c. Verification of HFCMOD and HCMOD logic will be accomplished by the use of the Heliostat Simulator P/N 40E5005132775.
- d. A complete functional test will be performed to determine pointing accuracy, motor speed control, timing, communication protocol, and error detecting capability.

3.6 Graphic Display Console (GDC) Software

3.6.1 Purpose

The primary purpose of the Graphic Display Console (GDC) is to provide graphic and alphanumeric displays which enhance operator understanding of heliostat field operations. The secondary purpose of the GDC is to provide field commanding capabilities complementary to those of the CSS operator console.

The GDC-resident software system implements controls and displays which support these functions, and maintains necessary communications with the HAC. The allocation of processing functions between the HAC and GDC is designed to minimize risk to the time-critical functions of the HAC without unduly penalizing performance of the GDC.

3.6.2 Requirements

3.6.2.1 Design Requirements

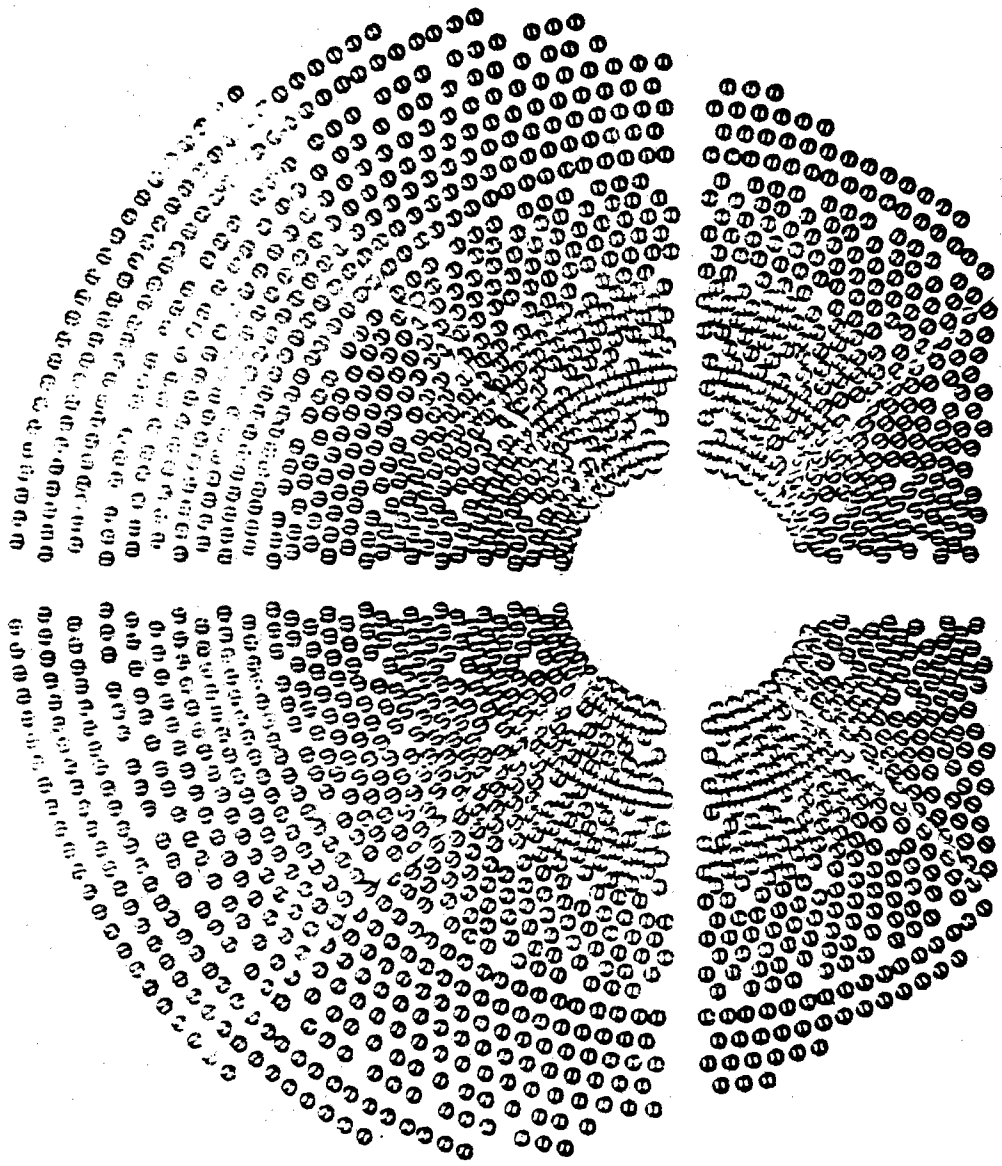
Section 3.1 of the Software/Firmware Functional Requirements Specification (12 June 1980) requires:

- a. Control of up to 2048 heliostats in all modes;
- b. Display of the operational status of all heliostats; and
- c. Graphic displays of the heliostat field or field segment

Section 3.2.1.9 of the Software/Firmware Functional Requirements Specification requires that the GDC module:

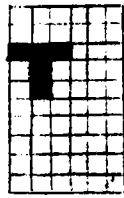
- a. Minimize the workload for the HAC to support graphic displays;
- b. Interface with the HAC using a command/response protocol;
- c. Provide full field and segment displays on operator demand. Figure 3.6.2-1 depicts the format for the full-field display. The shape and color encoding of the 12 different heliostat status symbols for the full-field display are shown in Figure 3.6.2-2. Figure 3.6.2-3 depicts the format for the segment displays. The shape and color encoding of the 12 different heliostat status symbols for the segment displays are shown in Figure 3.6.2-4.

The eight rectangles in the lower left portion of the screen are labeled and color coded to match the functions of the BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, YELLOW, and WHITE function keys. The "Command>" line reflects the



<p>Command> display full-field</p> <p>Message></p> <div style="border: 1px solid black; width: 50px; height: 20px; display: inline-block;"></div> <div style="border: 1px solid black; width: 50px; height: 20px; display: inline-block; vertical-align: top; padding: 2px;">display full- field</div> <div style="border: 1px solid black; width: 50px; height: 20px; display: inline-block; vertical-align: top; padding: 2px;">display segment</div> <div style="border: 1px solid black; width: 50px; height: 20px; display: inline-block; vertical-align: top; padding: 2px;"></div> <div style="border: 1px solid black; width: 50px; height: 20px; display: inline-block; vertical-align: top; padding: 2px;">defocus</div> <div style="border: 1px solid black; width: 50px; height: 20px; display: inline-block; vertical-align: top; padding: 2px;"></div> <div style="border: 1px solid black; width: 50px; height: 20px; display: inline-block; vertical-align: top; padding: 2px;">high wind stow</div> <div style="border: 1px solid black; width: 50px; height: 20px; display: inline-block; vertical-align: top; padding: 2px;">help</div>	<p>T STOW 1 OFFLINE 2 TRACK</p> <p>1 ALT1STOW 2 INITIAL 2 STANDBY</p> <p>2 ALT2STOW 2 MARK 2 BCS</p> <p>1 WASH 2 DIR POS 2 TRANSIT</p>
---	--

Figure 3.6.2-1 Full Field Display
953



STOW
(green)



OFFLINE
(blue)



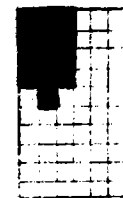
TRACK
(red)



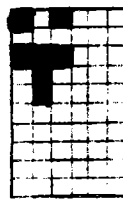
ALT1STOW
(green)



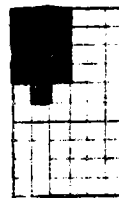
INITIAL
(blue)



STANDBY
(yellow)



ALT2STOW
(green)



MARK
(green)



BCS
(cyan)



WASH
(green)



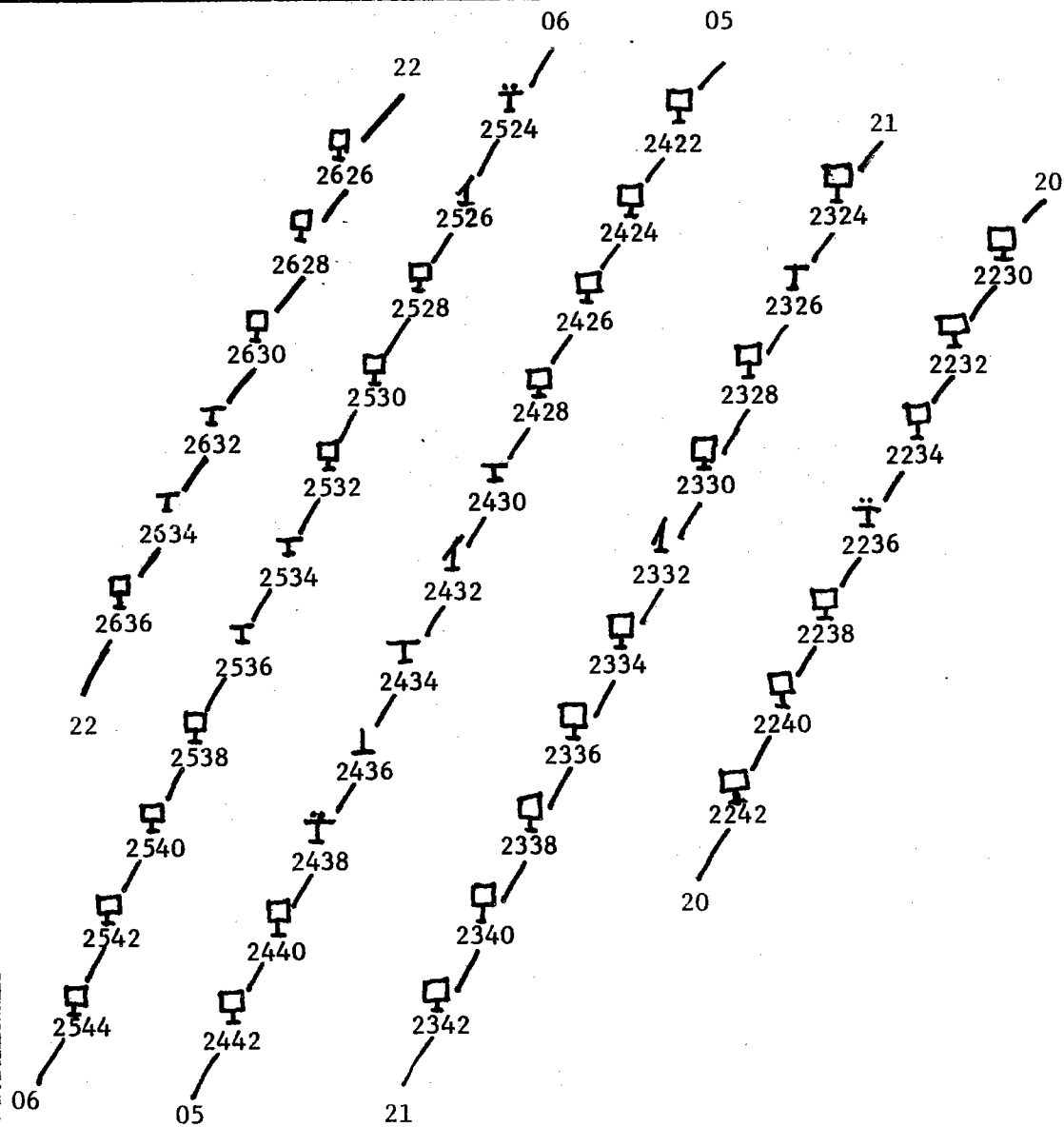
DIR POS
(magenta)



TRANSIT
(white)

Figure 3.6.2-2 - 3 X 5 Symbol/Color Encoding for Full Field Display

Figure 3.6.2-3 -- Segment Display
955



hel#	mode	aimpoint (ft)			gimbals (deg)	
		north	east	up	az	el

2230	TRACK	-59.7	32.3	368.4	-270.35	90.47
------	-------	-------	------	-------	---------	-------

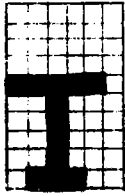
nnnn AAAAAA + -nn.n + -nn.n nnn.n + -nn.n + -nn.n

Segment 405 Aimpoint 16

Command> display segment RNN = 405
Message>

I STOW	↓ OFFLINE	☐ TRACK
↓ ALT1STOW	☐ INITIAL	☐ STANDBY
⊥ ALT2STOW	☐ MARK	☐ BCS
∟ WASH	☐ DIR POS	☐ TRANSIT

display full-field
 display segment
 defocus
 high wind stow
 help



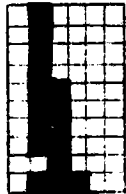
STOW
(green)



OFFLINE
(blue)



TRACK
(red)



ALT1STOW
(green)



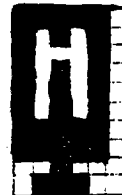
INITIAL
(blue)



STANDBY
(yellow)



ALT2STOW
(green)



MARK
(green)



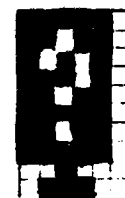
BCS
(cyan)



WASH
(green)



DIR POS
(magenta)



TRANSIT
(white)

Figure 3.6.2-4 - 5 X 10 Symbol/Color Encoding for Segment Displays

current command being entered by operator function key and/or keyboard entries, while the "Message>" line displays prompting or error messages to the operator; and

- d. Transmit function switch "HIGH WIND STOW" and "DEFOCUS" commands to the HAC.

3.6.2.2

Derived Requirements

The following requirements are derived from the explicit requirements listed in Section 3.6.2.1:

- a. The GDC module shall be capable of formatting and transmitting operator-entered emergency commands to the HAC;
- b. The GDC module shall be capable of receiving field status information from the HAC and formatting it into displays requested by the operator;
- c. Failure of the GDC or the need to restart the GDC shall not adversely impact operation of the HAC.

3.6.3

Design Approach

3.6.3.1

Functional Allocations

GDC software is comprised of the following modules:

- a. DIALOG - a background task activated by availability of inputs from the operator. The inputs which are valid at any given time depend on the particular command being entered. This task implements the operator interface, including:
 - 1. Displaying prompting and feedback messages.
 - 2. Processing display requests, including initiation of display maintenance tasks and sending status requests to HAC.
 - 3. Processing (emergency) commands including sending command to HAC.
- b. FFDISP and SEGDISP - each is a set of three background subtasks which initialize displays of full-field or segments, update the current display to reflect new information received from the HAC, and delete displays when they are no longer needed.

- c. HACDAT, FFPP and SEGPP - offline modules which manage data base transfer from HAC and preprocess field configuration data into formats convenient for rapid generation of run-time displays.
- d. EXEC - the executive service routine which provides for task synchronization and switching.
- e. PROHAC - a background task activated by receipt of a message from the HAC. This task checks validity of the message and then activates the appropriate task to update the current display with the new data received from the HAC.
- f. RCV - an interrupt handler activated by receipt of data on the HAC/GDC serial communication line. This module assembles incoming bytes into message packets which are passed on to PROHAC for processing.
- g. KBD - an interrupt handler activated by the keyboard/function-key console strobe interrupt. This module signals availability of input data from the keyboard/function-key input port to an application task waiting for that data.
- h. SEND - a module which formats and transmits message packets to the HAC.
- i. HELP - a module which displays explanatory material on the GDC screen.

3.6.3.2

Resource Budgets

Resource utilization of the GDC module is estimated as follows:

	<u>number bytes of code</u>	<u>number bytes of data</u>	<u>number of disk files (I/O)</u>	<u>total number of disk sectors</u>
DIALOG	N/A		-	-
FFDISP		12K	1/-	64
SEGDISP	N/A	2K	120/-	500
FFPP	N/A		1/1	64
SEGPP	N/A		1/120	500
EXEC	N/A		-	-
PROHAC	N/A		-	-
RCV			-	-
KBD	144		-	-
SEND	N/A		-	-
HELP	N/A		1/-	3

3.6.4 Design Description

3.6.4.1 Module Structure

The GDC software module consists of foreground (interrupt driven) submodules, background (real time) processing submodules, and offline (non-real time) preprocessing submodules.

The offline preprocessors (HACDAT, FFPP and SEGPP) are run whenever the field configuration data base on the HAC is updated. They manage transfer of formatted data from the HAC and conversion of that data into run-time efficient, unformatted, disk-resident files.

GDC online software includes foreground submodules which handle interrupts from the SIO link (RCV) and keyboard (KBD). These submodules provide data buffering and signal availability of data to the EXEC, which invokes appropriate background tasks to process the data. These background tasks include the interactive user/GDC dialogue controller (DIALOG), and real-time display updating tasks (FFDISP and SEGDISP). A collection of utility functions provides application-oriented numeric and character-handling capabilities to both offline and online processes.

Figure 3.6.4-1 shows the GDC on-line module structure.

3.6.4.1.1 Submodule I - EXEC (Main Routine)

3.6.4.1.1.1 Description

The EXEC module will activate the background tasks as data is received from the HAC and operator keyboard. The interrupt handler, RCV, will set a flag when a complete message packet is received from the HAC. This flag is tested by the EXEC which will activate the background module PROHAC whenever the flag is set. PROHAC checks that the message packet received is correct for the current state of the GDC and either issues an error message or activates the appropriate screen update task.

Similarly, when the KBD interrupt handler receives a keyboard/function-key interrupt, it will set a flag which causes the EXEC to activate the DIALOG task. The DIALOG task will echo keystrokes to the display, and then activate the appropriate tasks for updating the display or sending commands to the HAC.

3.6.4.1.1.2 Data, Logic and Command Paths

The EXEC routine is both the idle loop and task dispatcher for the GDC. It is initiated at system start-up and runs indefinitely.

3.6.4.1.1.3 Internal Data Description

There is no data internal to this submodule.

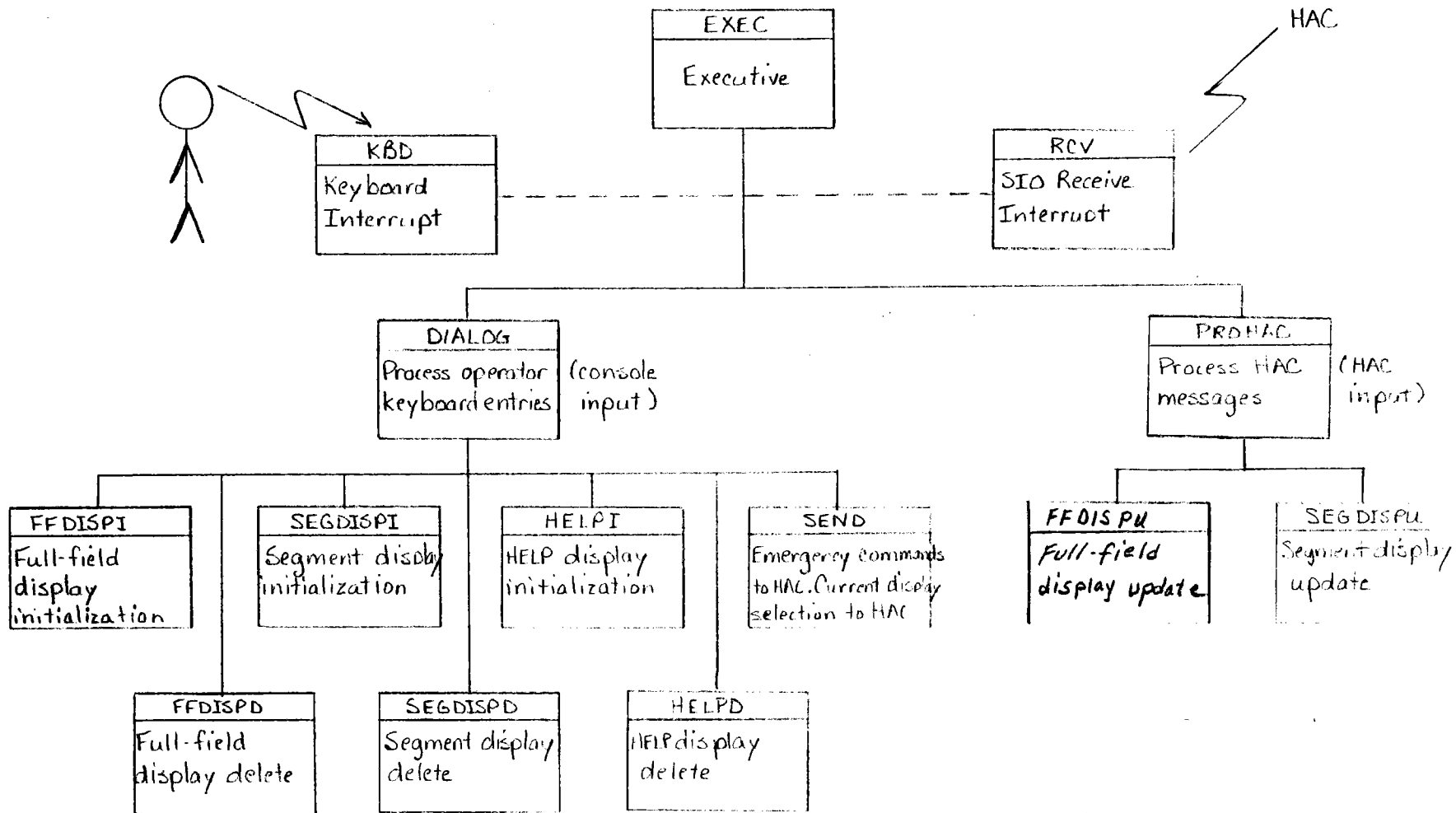


Figure 3.6.4-1 GDC Module Structure

3.6.4.1.1.4 Flowchart

See Figure 3.6.4-2 for the EXEC flowchart.

3.6.4.1.2 Submodule II - RCV

This module is the device driver for the receiving side of the serial I/O (SIO) link to the HAC. This module follows the standard Chromatics conventions for device drivers and thus includes three entry points:

- a. RCVISR - the interrupt service routine;
- b. HACAVAIL - indicates availability of data in the receive buffer; and
- c. HACDATA - returns the next character from the buffer.

This module is modified from the standard device driver in PROM as follows:

- a. A circular buffer of twice the maximum HAC message length (258 bytes) is used instead of a 2K-byte buffer;
- b. All characters prior to an STX character are discarded;
- c. All characters following an ETX character are discarded;
- d. Eight-bit binary data is stored in the buffer instead of masking off the high-order bit; and
- e. The flag HWAIT is cleared when a complete message is received to indicate that the GDC is no longer waiting for a HAC message.

3.6.4.1.2.1 Subroutine 1 - RCVISR

3.6.4.1.2.1.1 Description

This routine stores messages from the HAC in the receive buffer.

3.6.4.1.2.1.2 Data, Logic and Command Paths

This routine is activated at the interrupt level on receipt of a character on the asynchronous communication line from the HAC. This routine maintains the IN pointer to the circular buffer, BUF. It looks for an STX character and then stores all subsequent characters up to an ETX character in BUF. When an ETX is received, the HWAIT flag is cleared to indicate that a complete message packet has been received.

3.6.4.1.2.1.3 Internal Data Description

This routine maintains an internal flag indicating whether an STX character has been received.

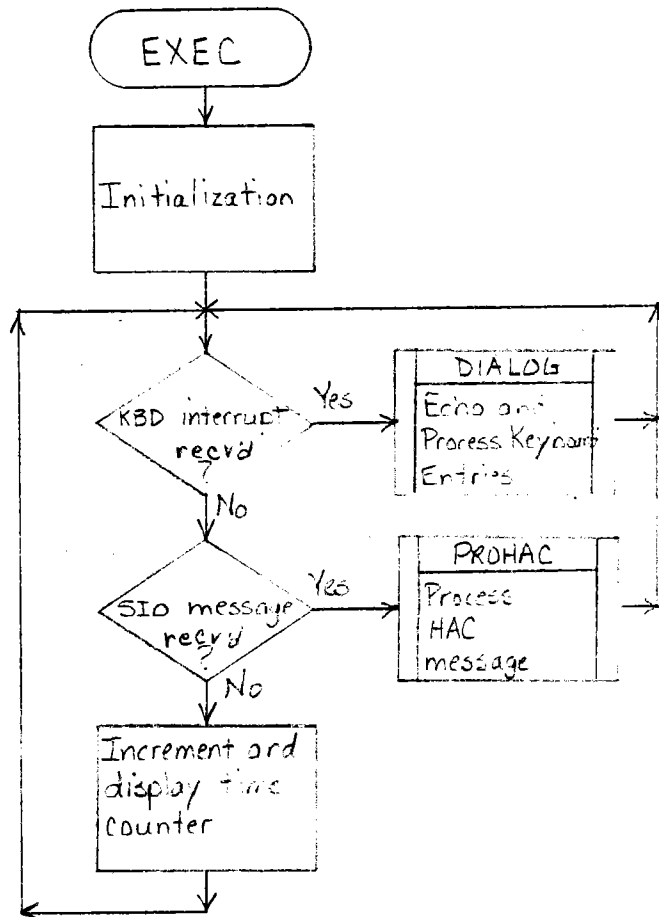


Figure 3.6.4-2 Flowchart - EXEC

- 3.6.4.1.2.1.4 Flowchart
See Figure 3.6.4-3 for RCVISR flowchart.
- 3.6.4.1.2.2 Subroutine 2 - HACAVAIL
- 3.6.4.1.2.2.1 Description
This routine is used to check for data in the SIO receiver buffer.
- 3.6.4.1.2.2.2 Data, Logic and Command Paths
This routine is called by EXEC to test for the availability of a HAC message. It returns a non-zero value if a message is available. It tests the HWAIT flag, which is set by SEND when a clear-to-send message is sent to the HAC, and is cleared by RCVISR when a complete message packet has been received from the HAC.
- 3.6.4.1.2.2.3 Internal Data Description
There is no data internal to this subroutine.
- 3.6.4.1.2.2.4 Flowchart
See Figure 3.6.4-4 for the HACAVAIL flowchart.
- 3.6.4.1.2.3 Subroutine 3 - HACDATA
- 3.6.4.1.2.3.1 Description
This routine returns the next character from the SIO receive buffer.
- 3.6.4.1.2.3.2 Data, Logic and Command Paths
This routine is called by all the HAC message-processing modules to access the HAC message. This routine maintains the OUT pointer to the circular buffer, BUF.
- 3.6.4.1.2.3.3 Internal Data Description
There is no data internal to this subroutine.
- 3.6.4.1.2.3.4 Flowchart
See Figure 3.6.4-5 for the HACDATA flowchart.
- 3.6.4.1.3 Submodule III - KBD
This module is the device driver for the alphanumeric keyboard and function keys. This module follows the standard Chromatics conventions for device drivers and thus includes three entry points:

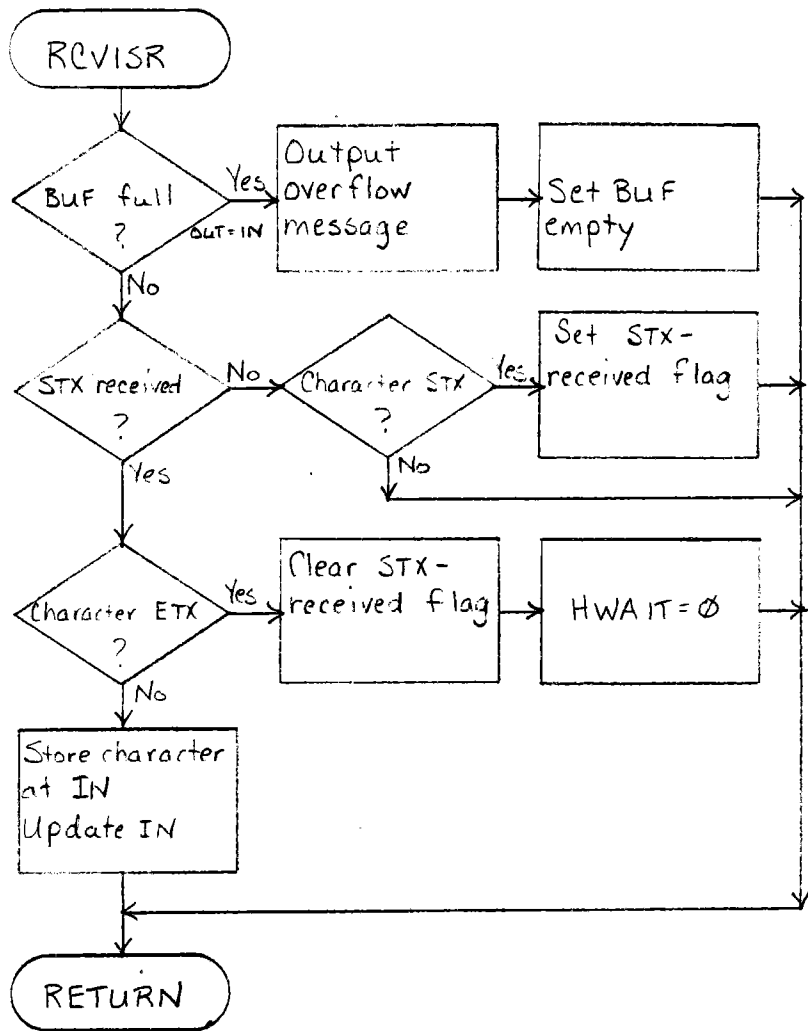


Figure 3.6.4-3 Flowchart - RCVISR

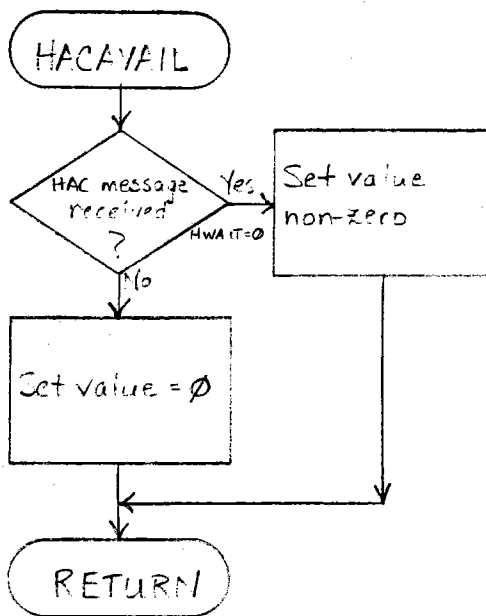


Figure 3.6.4-4 Flowchart - HACAVAIL
965

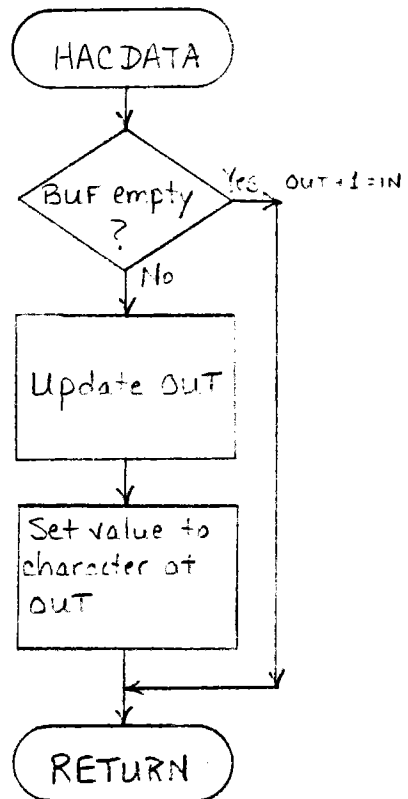


Figure 3.6.4-5 Flowchart HACDATA

- a. KBDISR - the interrupt service routine;
- b. KBDVAAIL - indicate availability of data in the keyboard buffer; and
- c. KBDDATA - returns the next character from the buffer.

This module is modified from the standard device driver in PROM as follows:

- a. All keystrokes are passed directly to the keyboard buffer without expansion into escape sequences; and
- b. An output to ring the bell is made if the keyboard buffer overflows.

3.6.4.1.3.1 Subroutine 1 - KBDISR

3.6.4.1.3.1.1 Description

This routine stores input characters in the keyboard buffer.

3.6.4.1.3.1.2 Data, Logic and Command Paths

This routine is activated at the interrupt level on receipt of a keyboard input interrupt. This routine maintains the KIN pointer to the circular buffer, KBUF. It stores the incoming characters in KBUF. If KBUF is full, a keyboard output is done to ring the bell.

3.6.4.1.3.1.3 Internal Data Description

There is no data internal to this subroutine.

3.6.4.1.3.1.4 Flowchart

See Figure 3.6.4-6 for the KBDISR flowchart.

3.6.4.1.3.2 Subroutine 2 - KBDVAAIL

3.6.4.1.3.2.1 Description

This routine is used to check for data in the keyboard buffer.

3.6.4.1.3.2.2 Data, Logic and Command Paths

This routine is called by EXEC to test for availability of keyboard input. It returns a non-zero value if one or more keystrokes have been stored in the keyboard buffer. The presence of data in the buffer is detected by comparing the KIN and KOUT pointers.

3.6.4.1.3.2.3 Internal Data Description

There is no data internal to this subroutine.

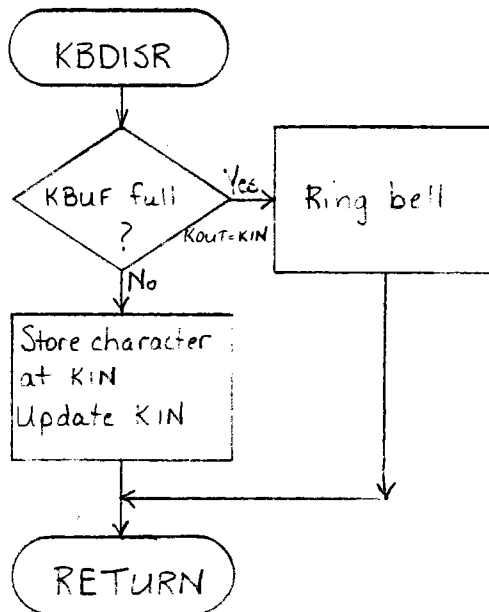


Figure 3.6.4-6 Flowchart - KBDISR

- 3.6.4.1.3.2.4 Flowchart
See Figure 3.6.4-7 for the KBDVAIL flowchart.
- 3.6.4.1.3.3 Subroutine 3 - KBDDATA
- 3.6.4.1.3.3.1 Description
This routine returns the next character from the keyboard buffer.
- 3.6.4.1.3.3.2 Data, Logic and Command Paths
This routine is called by the DIALOG module to access the operator inputs. This routine maintains the KOUT pointer to the circular buffer, KBUF.
- 3.6.4.1.3.3.3 Internal Data Description
There is no data internal to this subroutine.
- 3.6.4.1.3.3.4 Flowchart
See Figure 3.6.4-8 for the KBDDATA flowchart.
- 3.6.4.1.4 Submodule IV - SEND (Main Routine)
- 3.6.4.1.4.1 Description
The SEND module is called to transmit a message packet from the GDC to the HAC. Since the Chromatics does not support SIO transmit interrupts, the calling task is suspended until transmission is complete. The SEND module checks each message sent to the HAC, and if the message being sent is a clear-to-send (CTS, message ID = 4), it will set the HWAIT flag to indicate that the GDC is waiting for a HAC message. Since the chromatics disk routines run with non-disk interrupts masked, data sent by the HAC during a disk access would be lost. This problem is avoided by never accessing the disk unless the HWAIT flag is clear.
- 3.6.4.1.4.2 Data, Logic and Command Paths
The SEND module is called by PROHAC to send a CTS message to the HAC. The SEND module is called by DIALOG to send emergency commands to the HAC, and to notify the HAC of which status information is required for the current display.
- 3.6.4.1.4.3 Internal Data Description
There is no data internal to this submodule.
- 3.6.4.1.4.4 Flowchart
See Figure 3.6.4-9 for the SEND flowchart.

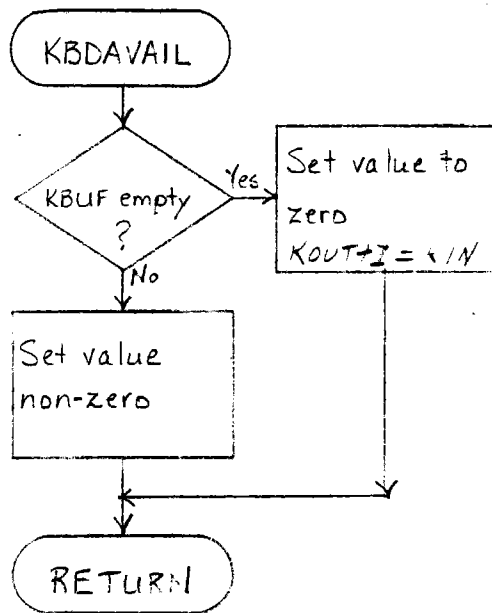


Figure 3.6.4-7 Flowchart - KBDVAAIL

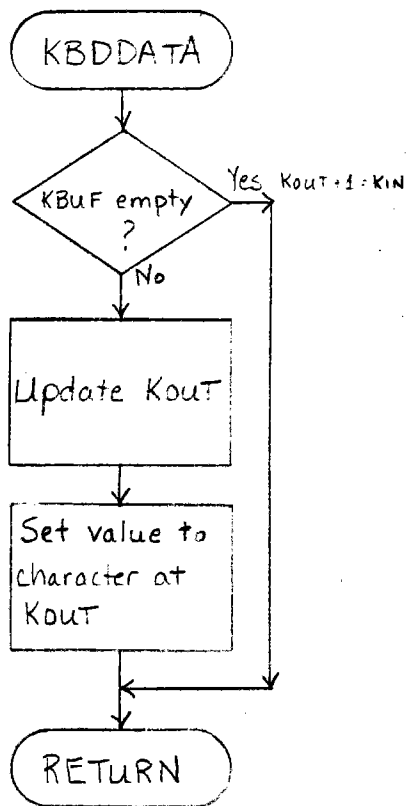


Figure 3.6.4-8 Flowchart - KBDDATA
971

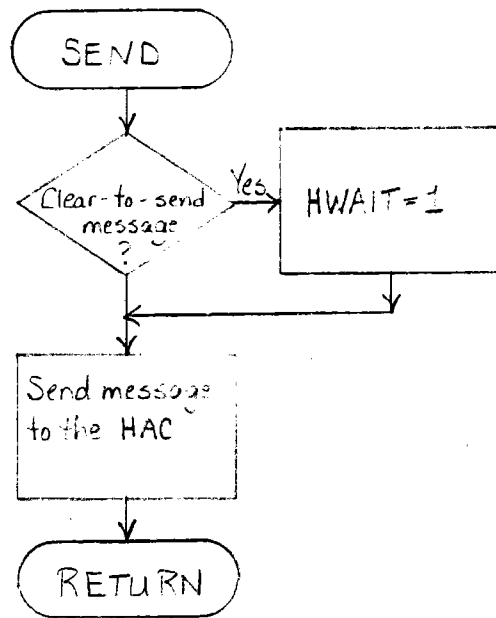


Figure 3.6.4-9 Flowchart - SEND
972

3.6.4.1.5

Submodule V - DIALOG

3.6.4.1.5.1

Description

This section describes software modules responsible for implementing the GDC/operator interface. Specifically, it covers the handling of keyboard/function-key input, echoing of operator inputs, and interpretation of commands, as detailed below:

a. Keyboard Input

1. Keyboard Interrupt Service Routine
2. Return Keyboard Buffer Data

b. Input Processing - This section describes the protocol for echoing and interpreting keyboard/function-key inputs. Due to the very limited number of commands available to the operator, a dialogue format which, though operator initiated, is relatively constraining with respect to errors the operator can make, seems desirable. Thus, it should be impossible for the operator to construct syntactically illegal commands. The following commands are available as function keys:

1. display full-field
2. display segment RNN =
3. high-wind stow
4. defocus
5. help

When a function-key is pressed, the COMMAND line will reflect that command. Pressing another key causes the new selection to replace the previous one. Commands are only acted upon when the RETURN key is pressed. In general, parameters to commands will be entered via the keyboard. The command key echo will include prompting information illustrating the type of parameter required. Among these commands, only the segment display requires any kind of amplifying information, namely the segment to be displayed. When this command is selected, the user will be prompted to enter the numeric designator for a segment as RNN, where R is a decimal digit (1:5) giving the Ring and NN is a decimal number (01:12) giving the wedge. Only decimal digits will be accepted as inputs to these fields.

A "menu" of color and position-encoded labeled function-keys will be displayed at the bottom of the screen during normal system operations.

c. Command Execution - Recognition of commands and arguments will be communicated to other modules for execution via calls, global variables and flags. In particular, display

commands will invoke the (FULL-FIELD) or (SEGMENT) display modules, emergency field commands will invoke the (SEND) module, and help requests will invoke the (HELP) module.

Emergency field commands will only be accepted at the control-room GDC.

3.6.4.1.5.2 Data, Logic and Command Paths

DIALOG is invoked by EXEC whenever KBD signals availability of input. It interprets keyboard and function-key input according to programmed dialog specifications and signals actions to be taken to EXEC. Display echoing is handled via calls to utility routines. Parsing of command input is controlled by a table-driven finite-state recognizer.

3.6.4.1.5.3 Internal Data Description

DIALOG maintains internal state information pertaining to commands and command arguments entered by the operator and the status of the command-echo display.

3.6.4.1.5.4 Flowchart

See Figure 3.6.4-10 for the DIALOG flowchart.

3.6.4.1.6 Submodule VI - FFDISP

The FFDISP module provides initialization and real-time updating of the full-field display. It includes the following functions:

- a. FFDISPI - full-field display initiation;
- b. FFDISPU - full-field display update; and
- c. FFDISPD - full-field display delete.

3.6.4.1.6.1 Subroutine 1 - FFDISPI (Full-Field Display Initialization)

3.6.4.1.6.1.1 Description

Disk-resident files generated by the full-field preprocessor module are read in to provide screen position information. The full-field symbol set is selected, and routines to draw symbol keys and titles are executed. A message is sent to the HAC informing it of the current display selection.

3.6.4.1.6.1.2 Data, Logic and Command Paths

FFDISPI is invoked by EXEC whenever the full-field display is selected by the operator via DIALOG. After initializing memory-resident data structures from disk, FFDISPI schedules (i.e. sets the current-display-flag to) FFDISPU.

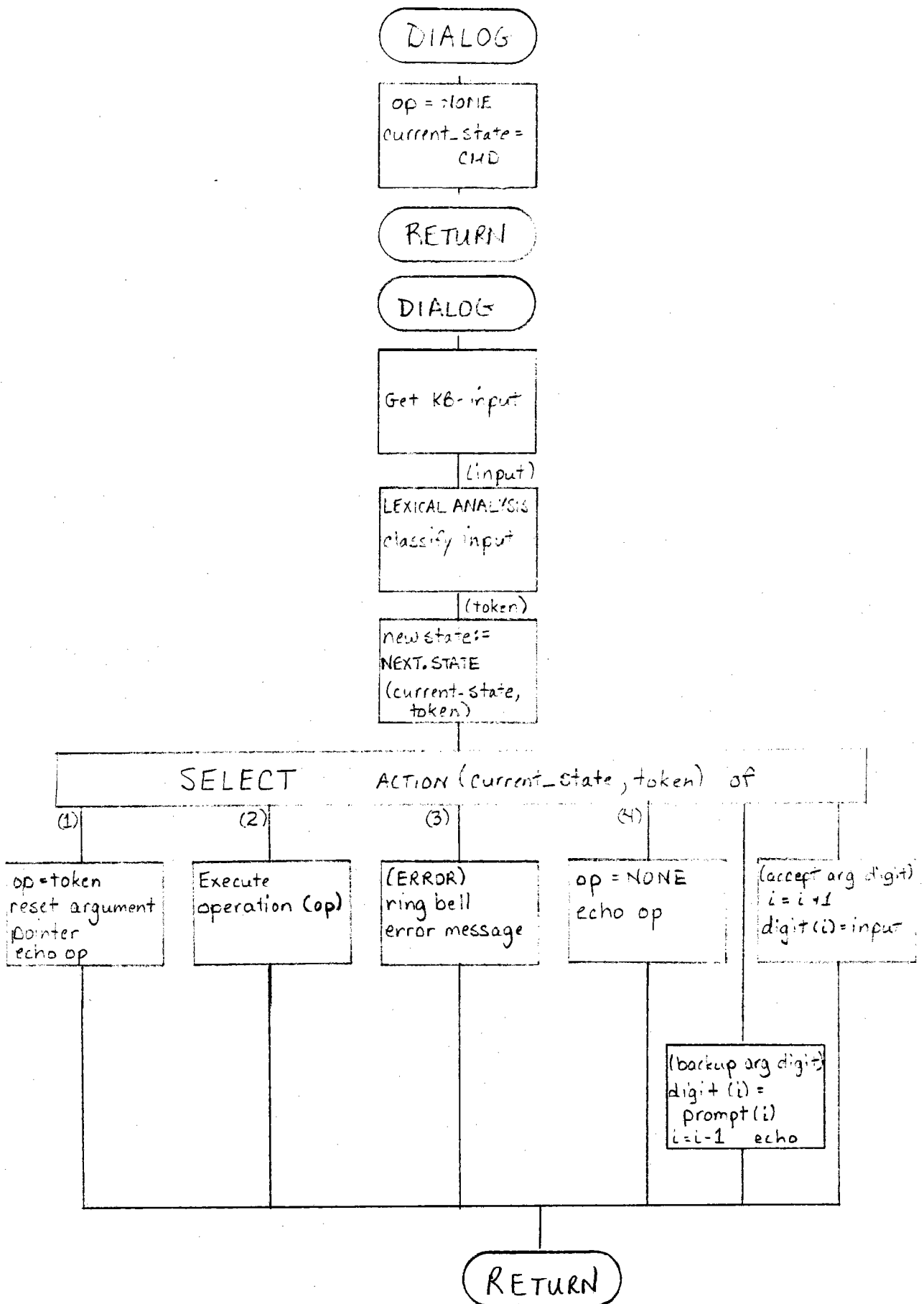


Figure 3.6.4-10 Flowchart - DIALOG
975

3.6.4.1.6.1.3 Internal Data Description

FFDISPI allocates and initializes FFDISPU local status data base to "unknown."

3.6.4.1.6.1.4 Flowchart

See Figure 3.6.4-11 for the FFDISPI flowchart.

3.6.4.1.6.2 Subroutine 2 - FFDISPU (Full-Field Display Update)

3.6.4.1.6.2.1 Description

As full-field status messages are received from the HAC, this subroutine is executed to update the screen display to reflect status changes. This involves comparing new status information to previously-displayed data and updating the color and symbol for changed heliostats.

3.6.4.1.6.2.2 Data, Logic and Command Paths

FFDISPU is invoked by EXEC and PROHAC whenever the full-field display has been selected by the operator and a full-field status message is received from the HAC.

The FFDISPU module uses data from two principal sources:

- a. Memory-resident copy of disk data base generated by offline procedure FFPP; gives screen location of each heliostat in full-field display; and
- b. Incoming real-time status messages, received via RCV.

This information is used to update the full-field display. Symbols for heliostats are redrawn only when a change from most-recently drawn status is detected.

The assembly language subroutine FULFLD performs the actual checking and updating operation.

3.6.4.1.6.2.3 Internal Data Description

FFDISPU maintains a local data base containing the status of each heliostat as most recently drawn on the display. This is used and maintained by subroutine FULFLD to eliminate redundant display updating. The most significant 7 bits of the screen X coordinate array entry for each heliostat are used to store this data.

3.6.4.1.6.2.4 Flowchart

See Figure 3.6.4-12 for the FFDISPU flowchart.

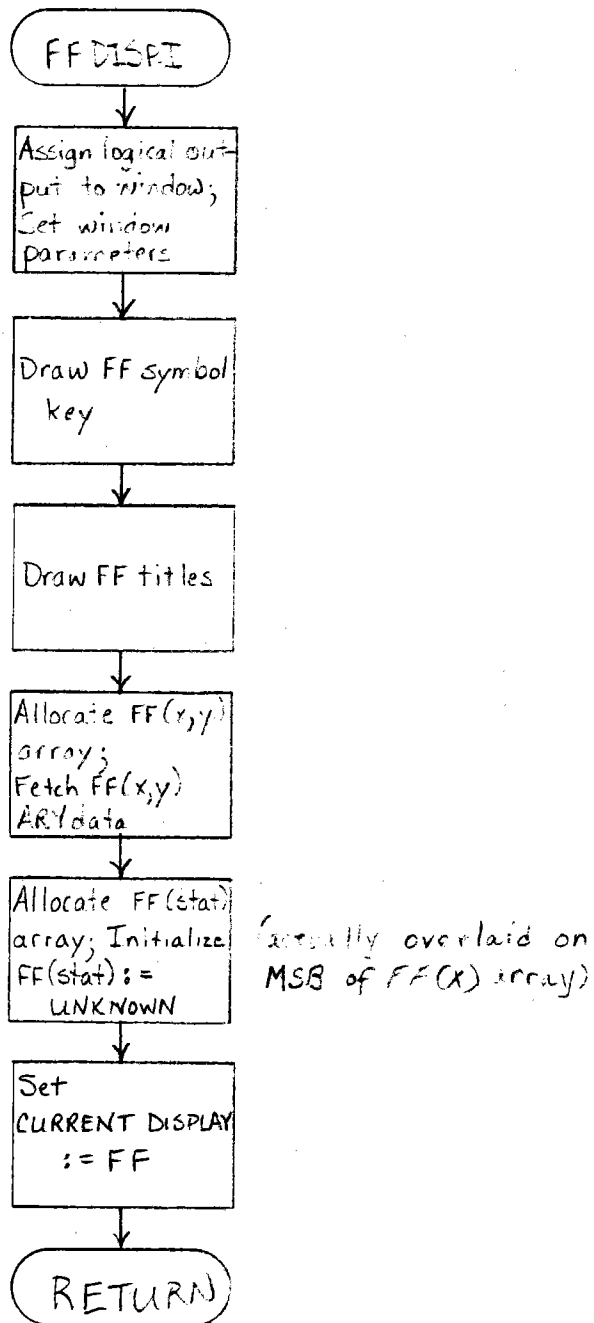


Figure 3.6.4-11 Flowchart - FFDISPI

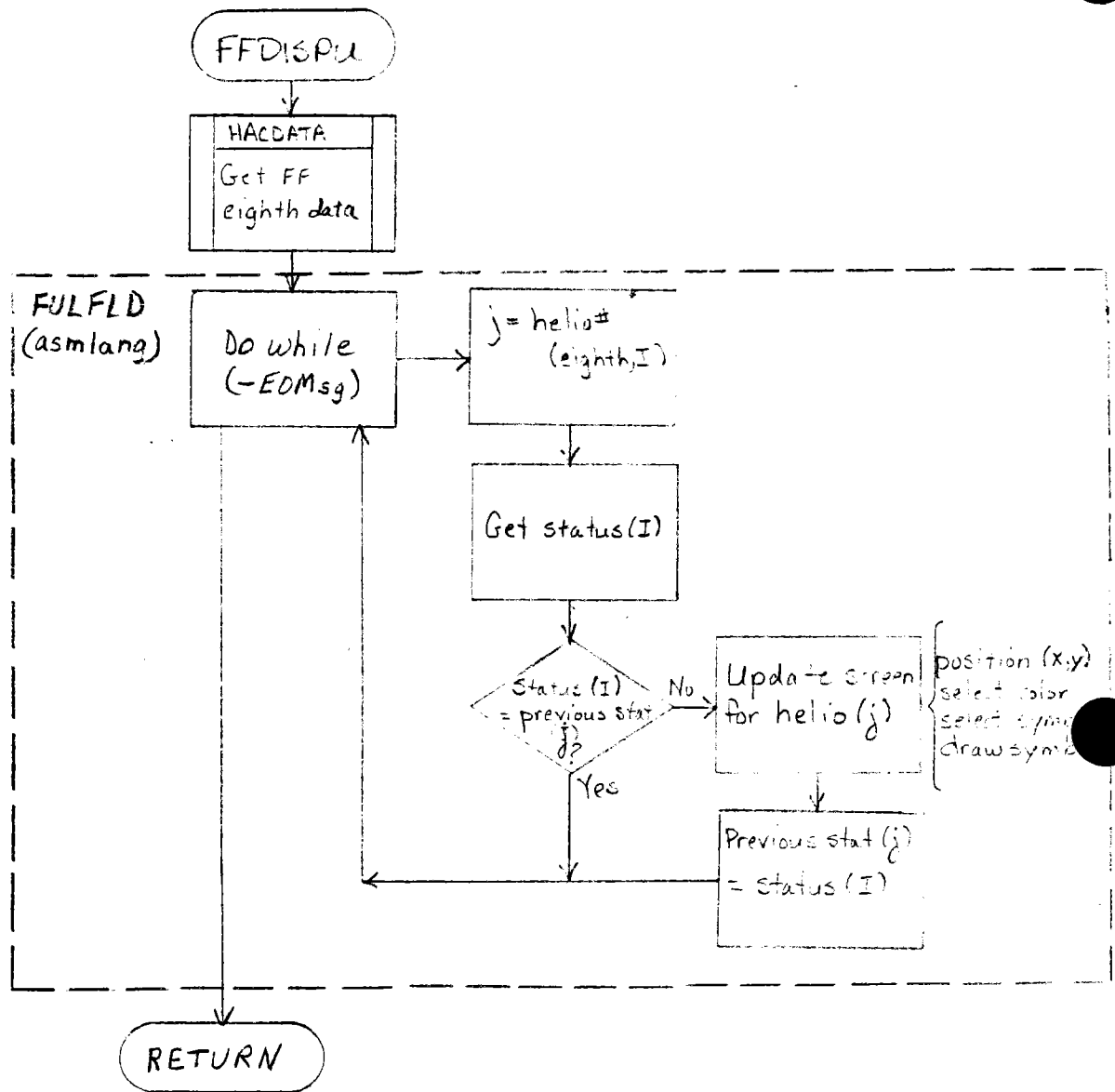


Figure 3.6.4-12 Flowchart - FFDISPU
978

3.6.4.1.6.3 Subroutine 3 - FFDISPD (Full-Field Display Delete)

3.6.4.1.6.3.1 Description

When the full-field display is no longer requested, the module clears the viewport(s) assigned to it, releases storage, and exits.

3.6.4.1.6.3.2 Data, Logic and Command Paths

FFDISPD is called by DIALOG whenever the full-field display is active and another display is selected by the operator.

It de-schedules FFDISPU (i.e. sets current-display-to NONE) to complete the shutdown operation.

3.6.4.1.6.3.3 Internal Data Description

Local data base storage used by FFDISP is deallocated.

3.6.4.1.6.3.4 Flowchart

See Figure 3.6.4-13 for the FFDISPD flowchart.

3.6.4.1.7 Submodule VII - SEGDISP

The SEGDISP module provides initialization and real-time updating of the segment-level display. It includes the following functions:

- a. SEGDISPI - Segment display initialization;
- b. SEGDISPU - Segment display update; and
- c. SEGDISPD - Delete segment display.

3.6.4.1.7.1 Subroutine 1 - SEGDISPI (Segment Display Initialization)

3.6.4.1.7.1.1 Description

Disk-resident files generated by the segment preprocessor module are read in to provide screen position information for both graphical symbols and A/N table information. Precomputed display lists for heliostat (HFC) connection lines and lables are drawn. The segment symbol set is selected and routines which draw table headers and symbol keys are called. A message is sent to the HAC informing it of the current display selection.

3.6.4.1.7.1.2 Data, Logic and Command Paths

SEGDISPI is invoked by EXEC whenever a segment-level display is selected by the operator via DIALOG.

After initializing memory-resident data structures from disk, SEGDISPI schedules (i.e. sets the current-display-flag to) SEGDISPU. Data includes heliostat screen locations (in pecking order) from SEGnnn.ARY and background display from SEGnnn.BUF.

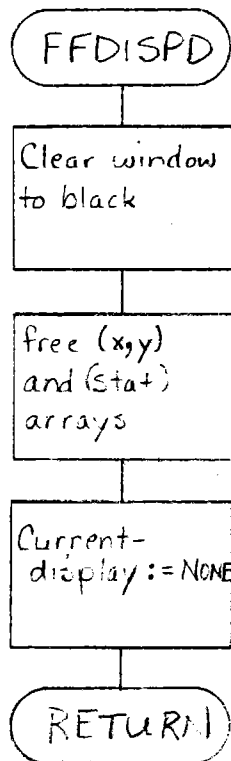


Figure 3.6.4-13 Flowchart - FFDISPD
980

3.6.4.1.7.1.3 Internal Data Description

SEGDISP allocates and initializes SEGDISPU local status data base to "unknown."

3.6.4.1.7.1.4 Flowchart

See Figure 3.6.4-14 for the SEGDISPI flowchart.

3.6.4.1.7.2 Subroutine 2 - SEGDISPU (Segment Display Update)

3.6.4.1.7.2.1 Description

As full-field status messages are received from the HAC, this subroutine is executed to update the screen display to reflect status changes. This involves comparing new status information to previously-displayed data and updating the color and symbol for changed heliostats. The A/N table is updated to reflect received information.

3.6.4.1.7.2.2 Data, Logic and Command Paths

SEGDISPU is invoked by EXEC and PROHAC whenever a segment display has been selected by the operator and a segment status message is received from the HAC. The SEGDISP module uses data from two principal sources:

- a. Memory-resident copy of local disk data base prepared by SEGPREP: file SEGnmm.ARY contains screen location of each heliostat in given segments, and screen location of A/N table entry for each heliostat; file SEGnmm.BUF contains a display-list which draws the display background for the given segment; and
- b. Real-time heliostat status update messages from HAC via RCV: contain mode, azimuth/elevation, and target information.

SEGDISP updates both the graphical and A/N tabular displays to reflect changes in heliostat status.

3.6.4.1.7.2.3 Internal Data Description

SEGDISPU maintains a local data base containing the status of each heliostat as most recently drawn on the graphic display and A/N table. This is used to eliminate redundant display updating.

3.6.4.1.7.2.4 Flowchart

See Figure 3.6.4-15 for the SEGDISPU flowchart.

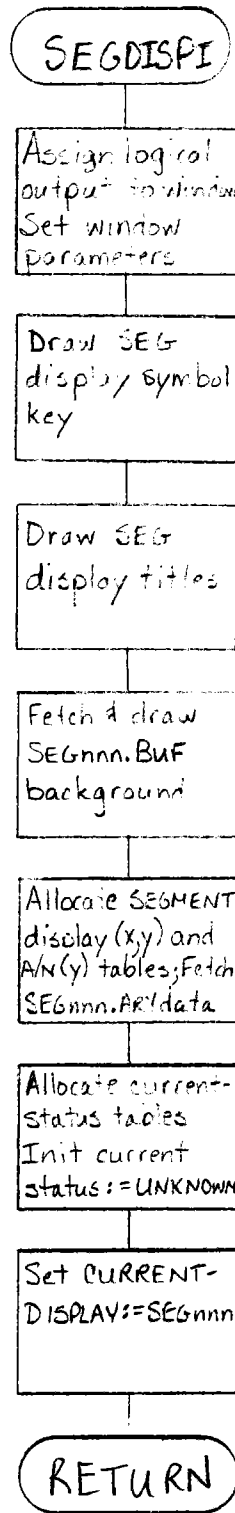


Figure 3.6.4-14 Flowchart - SEGDISP1
982

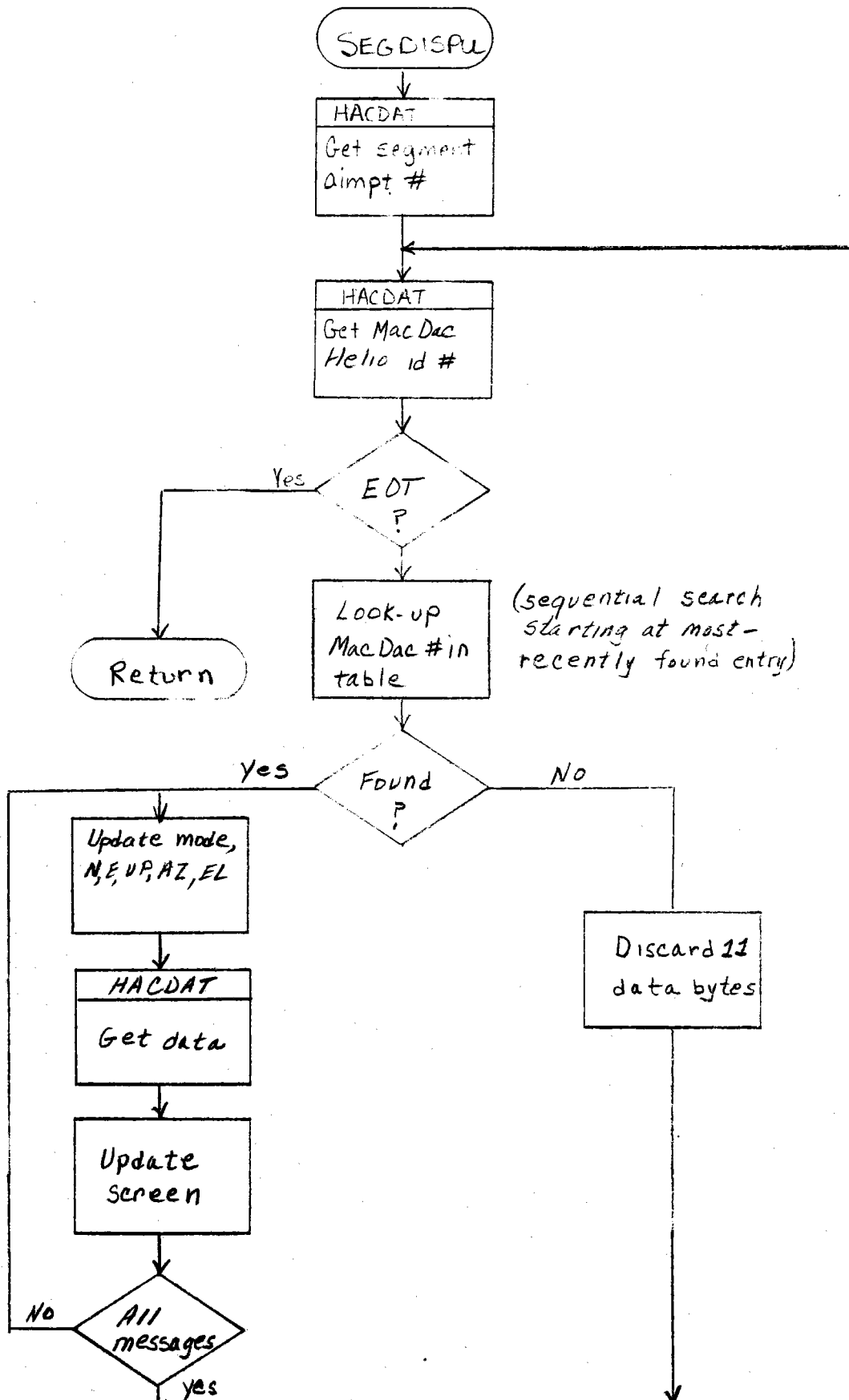


Figure 3.6.4-15 Flowchart - SEGDISPU
983

- 3.6.4.1.7.3 Subroutine 3 - SEGDISPD (Delete Segment Display)
- 3.6.4.1.7.3.1 Description
- When the segment display is no longer requested, the module clears the viewport(s) assigned to it, releases storage, and exits.
- 3.6.4.1.7.3.2 Data, Logic and Command Paths
- SEGDISPD is invoked by DIALOG whenever a segment display is active and a different display is selected by the operator. It deschedules SEGDISPU (i.e. sets current-display-to NONE) to complete the shutdown process.
- 3.6.4.1.7.3.3 Internal Data Description
- Local data base storage used by SEGDISP is deallocated.
- 3.6.4.1.7.3.4 Flowchart
- See Figure 3.6.4-16 for the SEGDISPD flowchart.
- 3.6.4.1.8 Submodule VIII - HELP
- The HELP module displays explanatory material on the GDC screen. It includes the following functions:
- a. HELPI - Initialize HELP display; and
 - b. HELPD - Delete HELP display.
- 3.6.4.1.8.1 Subroutine 1 - HELPI (Initialize HELP Display)
- 3.6.4.1.8.1.1 Description
- This routine reads in a file of explanatory text material and writes it to the GDC main-display area. The text explains the overall operating scheme of the GDC.
- 3.6.4.1.8.1.2 Data, Logic and Command Paths
- The HELPI module is invoked by DIALOG when requested by the operator. It displays the (single page) HELP file and returns.
- 3.6.4.1.8.1.3 Internal Data Description
- HELPI reads in the display buffer file, HELP, and redraws it.
- 3.6.4.1.8.1.4 Flowchart
- See Figure 3.6.4-17 for the HELPI flowchart.

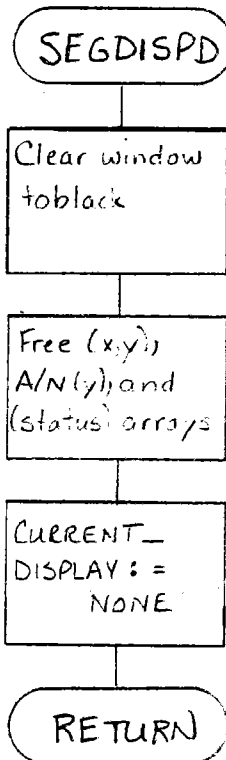


Figure 3.6.4-16 Flowchart - SEGDISPD

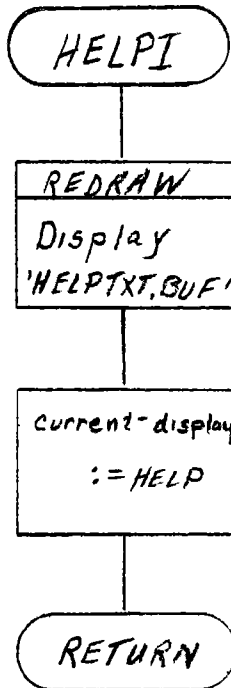


Figure 3.6.4-17 Flowchart - HELPI
986

- 3.6.4.1.8.2 Subroutine 2 - HELPD (Delete HELP Display)
- 3.6.4.1.8.2.1 Description
- This routine clears the screen area used by the HELP display.
- 3.6.4.1.8.2.2 Data, Logic and Command Paths
- This routine is invoked by DIALOG when the HELP display is the current display and another display is selected.
- 3.6.4.1.8.2.3 Internal Data Description
- There is no data internal to this subroutine.
- 3.6.4.1.8.2.4 Flowchart
- See Figure 3.6.4-18 for the HELPD flowchart.
- 3.6.4.1.9 Submodule IX - PROHAC
- 3.6.4.1.9.1 Description
- This module processes HAC messages and is called by EXEC whenever a complete message packet is received from the HAC.
- The main routine verifies that the message received is valid for the current state of the GDC and either issues an error message, displays the error message packet received from the HAC, or causes the appropriate screen-update program to be invoked.
- 3.6.4.1.9.2 Data, Logic and Command Paths
- This module is invoked by EXEC whenever a HAC message is received. This module maintains the TERMID variable and tests the DISP variable which indicates which display is currently on the screen. The formats of the HAC messages are described in Tables 3.7-XXXIX to XLIII
- 3.6.4.1.9.3 Internal Data Description
- This module extracts MSGID from the first byte of the current HAC message.
- 3.6.4.1.9.4 Flowchart
- See Figure 3.6.4-19 for the PROHAC flowchart.
- 3.6.4.1.10 Submodule X - FFPP
- 3.6.4.1.10.1 Description
- This section describes the offline processing required to acquire non-dynamic data from the HAC and convert it into forms suitable

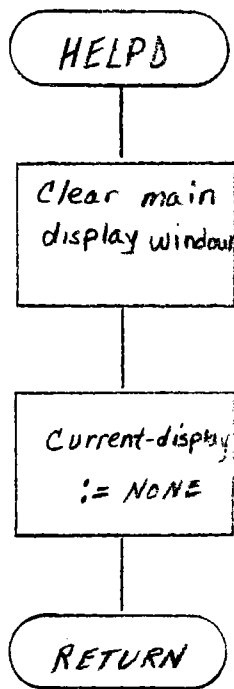


Figure 3.6.4-18 Flowchart - HELPD
988

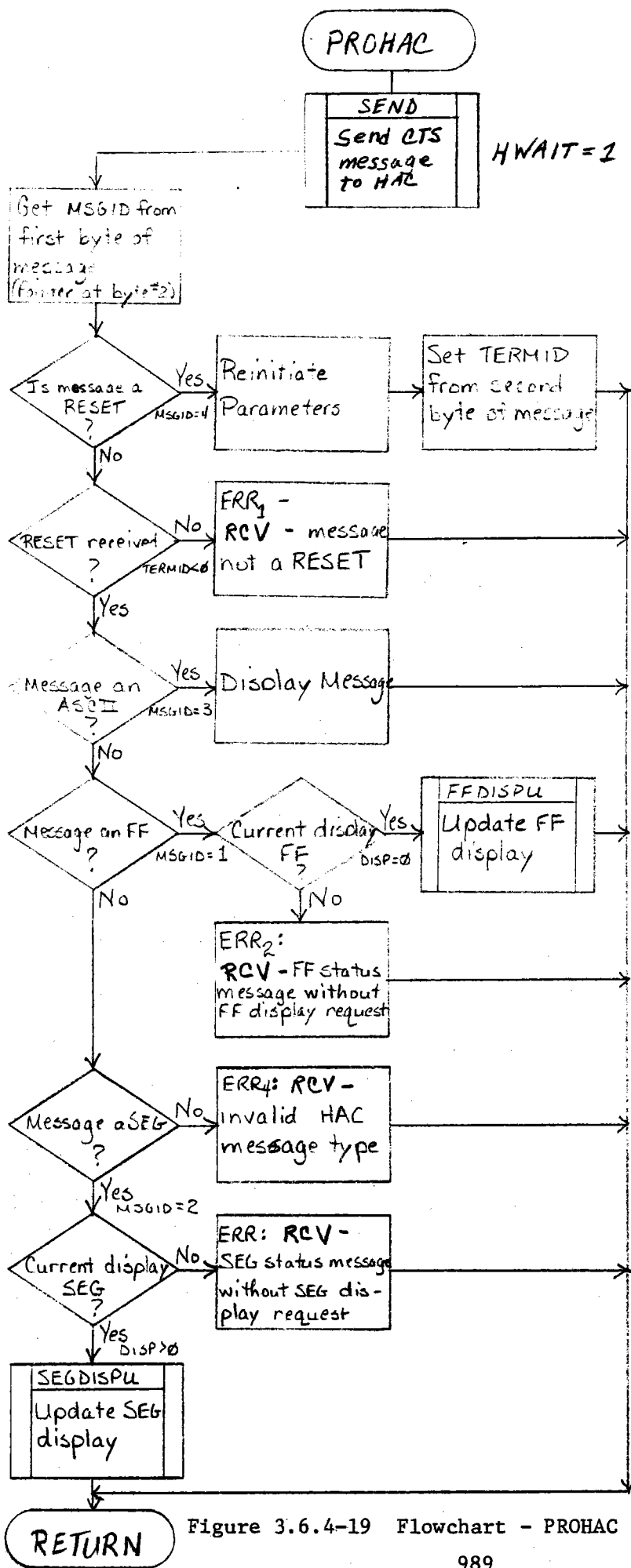


Figure 3.6.4-19 Flowchart - PROHAC

for use by the real-time dynamic full-field display module. FFPP includes the following functions:

- a. Data Acquisition (see program HACDAT) - Data for this operation is sent from the HAC in formatted ASCII decimal. Each record has the format described in Table 3.7-XXXVIII and is terminated by a CR. The records are sorted by eighth of field polling order. The full complement of records (2048 nominal) will be sent even if some heliostats are not installed.
- b. Data Processing - The principle processing required for the full-field display is the mapping of heliostat positions from the real-world coordinate system to the screen space used by this display. This module will accept window (real world) and viewport (screen) limits to be used. The screen x,y pairs thus generated will be saved as a BASIC array file. Since HFC/HC numbering is implicit in this file, it is not stored. The segment assignment of each heliostat in this ordering will be saved to allow generation of a segment assignment display of the full field.

3.6.4.1.10.2 Data, Logic and Command Paths

This routine controls data communication, processing, and storage during full-field preprocessing. Data is obtained from the HAC via a (software) handshaking protocol controlled by the GDC. In response to each REQ to SEND, the HAC sends one ASCII record ended by a CR.

FFPP creates files on the GDC local floppy disk which supports real-time processing by FFDISP.

3.6.4.1.10.3 Internal Data Description

- RECORD\$ - string containing ASCII data record from HAC
- NUM\$ - string containing ASCII representation of numeric field
- HXY% - integer array (2,2048) containing screen x,y coordinates of heliostats in MMC (HFC,HC) order
- WL - real array containing limits of user space window i.e., geographic coordinate boundaries of field
- VL - real array containing limits of display VIEWPORT, scaled to same proportions as WINDOW.

3.6.4.1.10.4 Flowchart

See Figure 3.6.4-20 for the FFPP flowchart.

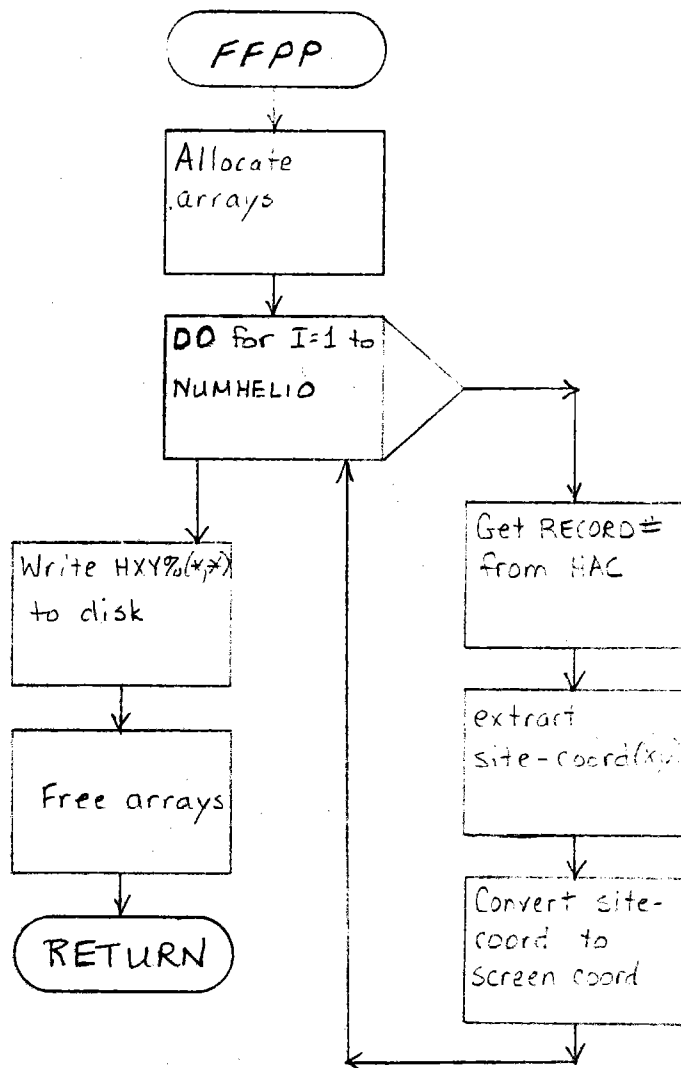


Figure 3.6.4-20 Flowchart - FFPP

3.6.4.1.11 Submodule XI - SEGPP

3.6.4.1.11.1 Description

This section describes acquisition and processing of data needed to support real-time generation of individual segment level displays. SEGPP includes the following functions:

- a. Data Acquisition (See program HACDAT) - Data for this operation is sent from the HAC in formatted ASCII decimal. Each record is formatted as per Table 3.7-XXXVIII and is terminated by a CR. The records are sorted by segment and pecking order. Segment boundaries in the data stream are marked by a record containing the characters "EOF" in positions 1-3.
- b. Data Processing - For each segment, the maximum and minimum north and east coordinate values over all heliostats in that segment must be determined to allow scaling of the segment geometry to optimally fit the screen viewport.

The alphanumeric tabular data accompanying each display will be sorted on MDAC heliostat number; i.e., the RRNN = row, position designation. Thus, each segment must be sorted on this field to compute the A/N table line assigned to each heliostat. Additionally, this ordering is needed to construct the HFC line connections which are displayed at the segment level.

For each segment, a display buffer file containing HFC connection lines, HFC labeling, and HC labels (in both the graphic and A/N table areas) will be constructed to minimize the time required to bring up displays at run time. A BASIC array file will be constructed for each segment. It will contain the (x,y) screen position of each heliostat symbol and its A/N table position (y).

3.6.4.1.11.2 Data, Logic and Command Paths

This routine controls data communication, processing and storage during segment-display preprocessing. Data is obtained from the HAC via a (software) handshaking protocol controlled by the GDC. In response to each REQ to SEND, the HAC sends one ASCII record ended by a CR.

SEGPP creates files on the GDC local floppy disk which support real-time processing by SEGDISP.

3.6.4.1.11.3

Internal Data Description - See Figure 3.6.4-21

- RECORD\$ - string containing ASCII data record from HAC
- NUM\$ - string containing ASCII representation of numeric field
- XYYN% - integer array (4,50) containing screen (x,y), MDAC HC number, and A/N table (y) per heliostat in pecking order
- WL - real array containing geographic limits of segment
- VL - real array containing screen viewport limits
- HFCHC% - integer array (50) containing MMC number per heliostat
- IN% - sort index array (50) used to sort segment by HC number (MDAC) = row, position for HFC connection lines
- ROW% - index table pointing to IN% indicating start of each row; integer array (15)

3.6.4.1.11.4

Flowchart

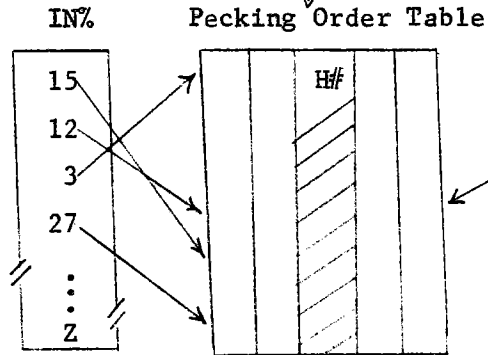
See Figure 3.6.4-22 for the SEGPP Flowchart.

per Segment in Pecking Order

X	Y	H# (MDAC)	HFC/HC# (MMC)

H# order

- 1
- 2
- 3
- ⋮
- n



Status message data:

#	Stat	AZ/EL	XYZ

index into Pecking-Ordered run-time tables

Row Table:

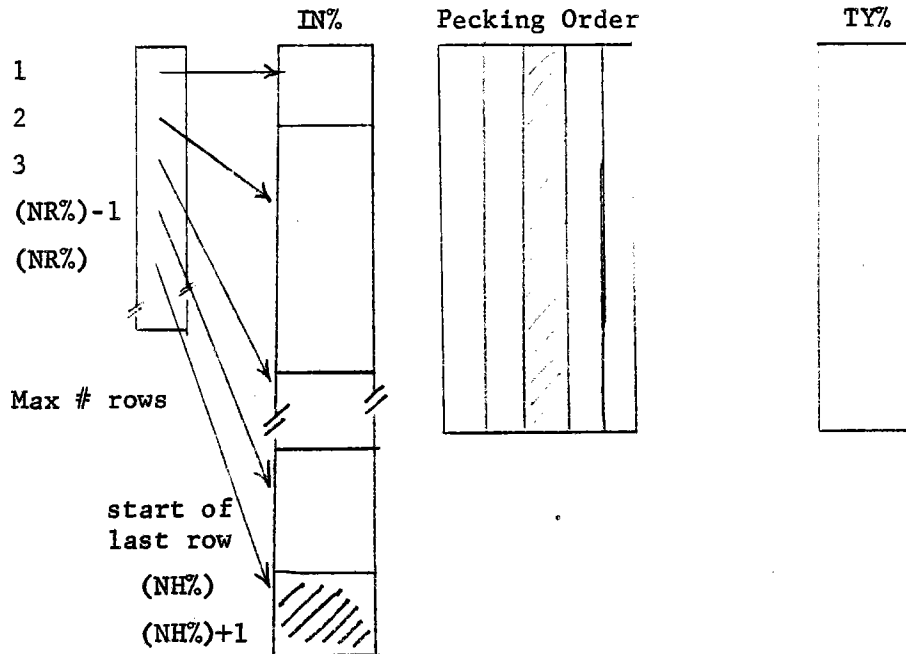


Figure 3.6.4-21 SEGPP Internal Data Structures

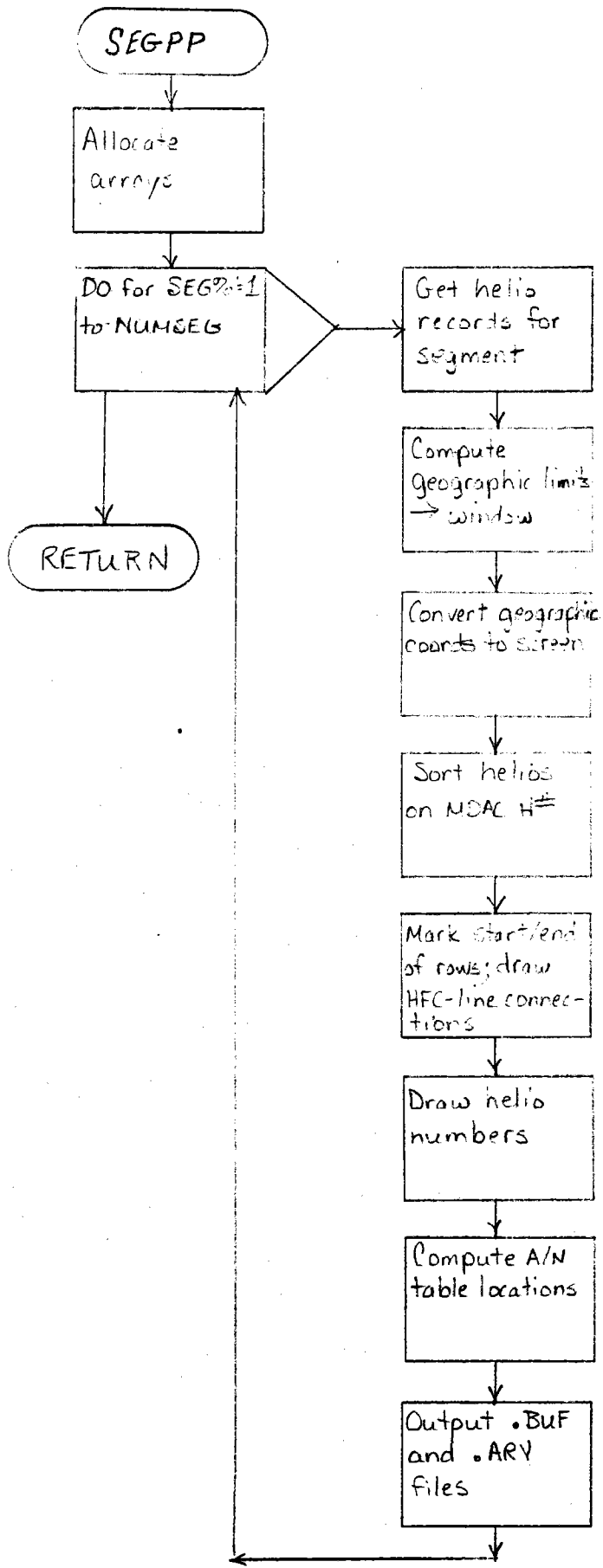


Figure 3.6.4-22 Flowchart - SEGPP

3.7 System Interfaces

The major system interfaces within the Collector Subsystem are as follows: HAC/HFC interface, HFC/HC interface, HAC/OCS-DAS interface, HAC/GDC interface, HAC/Receiver interface, and the Man-Machine interface. Each of these interfaces is detailed in the following sections.

3.7.1 HAC/HFC Interface Description

This interface is a computer-to-computer interface over which the HAC computer transmits to the HFC computer sun position information, operational commands, and a status poll request. The HFC computer transmits to the HAC computer the HFC/HC status information. The transmissions are on one-second cycles, and the timing of the cycle is shown in Figure 3.7-1. The data is transmitted at a rate of 19.2 kilobaud. The format of the message packets are shown in Tables 3.7-I through 3.7-XII.

3.7.2 HFC/HC Interface Description

This interface is a computer-to-computer interface over which the HFC computer transmits to the HC computer sun position information, operational commands, and a status request. The HC computer transmits to the HFC computer its command response information and status information. These transmissions are cyclic in nature and are made on one-second intervals. The data is transmitted at a rate of 9.6 kilobaud. The format of the message packets is shown in Tables 3.7-XIII through 3.7-XVIII.

3.7.3 HAC/OCS-DAS Interface Description

This interface is a computer-to-computer interface over which the OCS computer transmits commands, status requests and BCS requests. The HAC computer responds with command responses, error messages, status information and BCS messages. The DAS computer transmits over a separate line status requests to which the HAC computer responds with the appropriate status data. These transmissions are of a stimulus/response nature and are transmitted at a 19.2 kilobaud rate. The OCS and DAS have separate lines to the HAC and to the backup HAC. The format of the message packets is shown in Tables 3.7-XIX through 3.7-XXXVII.

3.7.4 HAC/GDC Interface Description

This interface is a computer-to-Intelligent Terminal interface over which the HAC transmits field initialization data, status information and text messages. The GDC transmits to the HAC emergency commands, status requests, send-next-message and a reset command. These transmissions are of a stimulus/response nature and are transmitted at a 9.6 kilobaud rate. The format of the message packets is shown in Tables 3.7-XXXVII through 3.7-XLVI.

3.7.5

HAC/Receiver Interface Description

This interface is a physical four-wire interface with TTL-type signal levels. Each of the two pairs shall have the opposite signal of the other. The receiver will transmit a trip signal and it will remain on the line from the receiver until it has returned to normal. The HAC will respond to the trip signal by automatically creating a DEFOCUS command. A set of four wires will be provided to both the HAC and its backup with the wires grounded at the HAC. The Trip Logic is shown in Table 3.7-XLVII.

3.7.6

Man-Machine Interface Description

The man-machine interfaces to the Collector Subsystem can best be described in five parts: 1) the command input mode; 2) the command file execute mode; 3) the command logging mode; 4) the alarms response mode; and 5) the status display mode. All of these modes exist in the system simultaneously.

3.7.6.1

Command Input Mode

Commands may be input to the system from the CS Control Console, Control Room Graphics Console, and the OCS via a data link or from a command file on disk (see Section 3.7.6.2). The list of commands available to the operator is shown in Tables 3.2.1-I, and 3.2.1-II, along with the valid sources, addressing formats, mode constraints, and command descriptions. All commands must be entered with at least four characters of the command, and may be entered with up to as many characters as the command has. The operator may enter the two emergency commands via function keys on the Control Room Graphics Console. This input involves pressing the appropriate function key rather than typing in a particular command. Commands coming from the OCS will be in the same format at those entered from the CS Control Console. Incorrectly entered commands will result in error messages transmitted to the inputting source. Table 3.2.1-III shows the error messages which may be displayed.

3.7.6.2

Command File Execute Mode

This mode allows a sequence of commands and their respective execute times to be prestored on disk for subsequent automatic execution. The execute times are relative to the command file start time, which requires an operator input.

3.7.6.3

Command Logging Mode

All commands input to the system are echo printed on the CS Control Room line printer (logger) with appropriate time stamps. This includes commands input from the CS Control Console, command

files, CS Control Graphics emergency commands, OCS commands and DAS status requests.

3.7.6.4

Alarm Response Mode

Alarms for irregular Collector Subsystem operation are generated automatically and reported to the operator. Two modes of reporting are employed. A fixed, protected portion of the CS Control console's screen is dedicated to alarm reporting. Two types of alarms are reported on this console. The first class of alarm is the system critical alarm, which requires an operator response, or acknowledgement, through the keyboard before another critical message may be displayed. The second class of alarms is the system non-critical alarm, which requires no response and is overwritten automatically by the next non-critical alarm. A historical record of critical and non-critical alarms is also maintained on the CS Control Room line printer. Table 3.2.5-IV shows a sample of the types of alarm messages which will be generated.

3.7.6.5

Status Display Mode

The status of the Collector Subsystem is displayed in two ways. A fixed, protected portion of the CS Control console's screen is reserved for a continuously generated display of field status. This display shows date and time, and gives a title of the twelve modes which the heliostats can be in at any time. Beneath each of the mode titles is the total number of heliostats in that mode. (See Figure 3.2.6-2.) This field status is updated on a once-per-second basis, using the one-eighth of field status which was received in that second. In addition to the continuous status displayed on the CS Control console, additional status may be requested by the operator through the keyboard. This demand status may be in one of four formats, at the option of the operator. If the argument entered with the STATUS command is ALL, the format is a snapshot of the display on the CS Control console printed on the line printer. (See Figure 3.2.6-2 for format.) If the argument entered is an "H/NNNN," where the "NNNN" indicates the HC number whose status is being requested, then the HC number, its segment number, its azimuth and elevation angles, and the commanded mode and actual mode are printed on the line printer. (See Figure 3.2.6-3 for format.) If the argument entered with the STATUS command is "M/AAA", where "AAA" is the mode whose status is requested, then all HC numbers that are in the requested mode will be listed on the line printer. (See Figure 3.2.6-4 for format.) If the argument entered is "R/N," where "N" is the ring number whose status is requested, then all segments within that ring will be printed on the line printer. Associated with each segment printed will be the number of HCs in that segment, the number of HCs in the TRACK mode and the number of HCs in the STANDBY mode. In addition, totals will be summarized for the whole ring. (See Figure 3.2.6-5 for format.)

NOTE: For M, R or ALL status requests, the status can also be displayed on the CS Control Console by optionally typing "CRT" after the command argument.

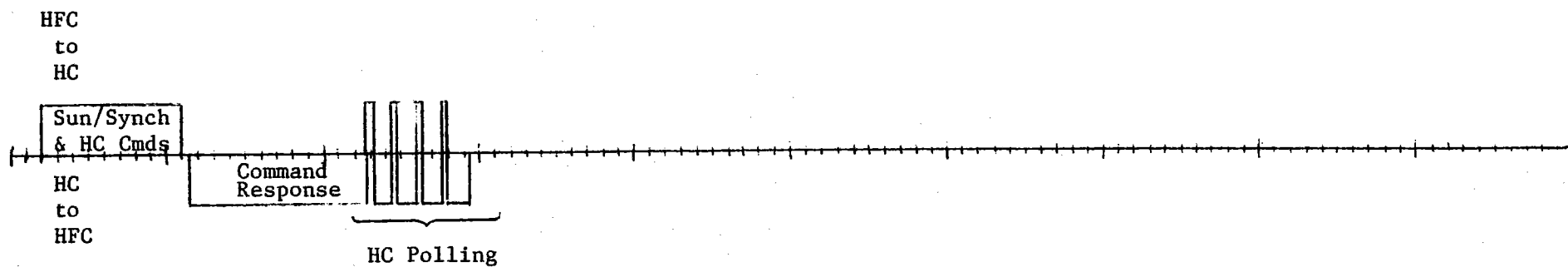
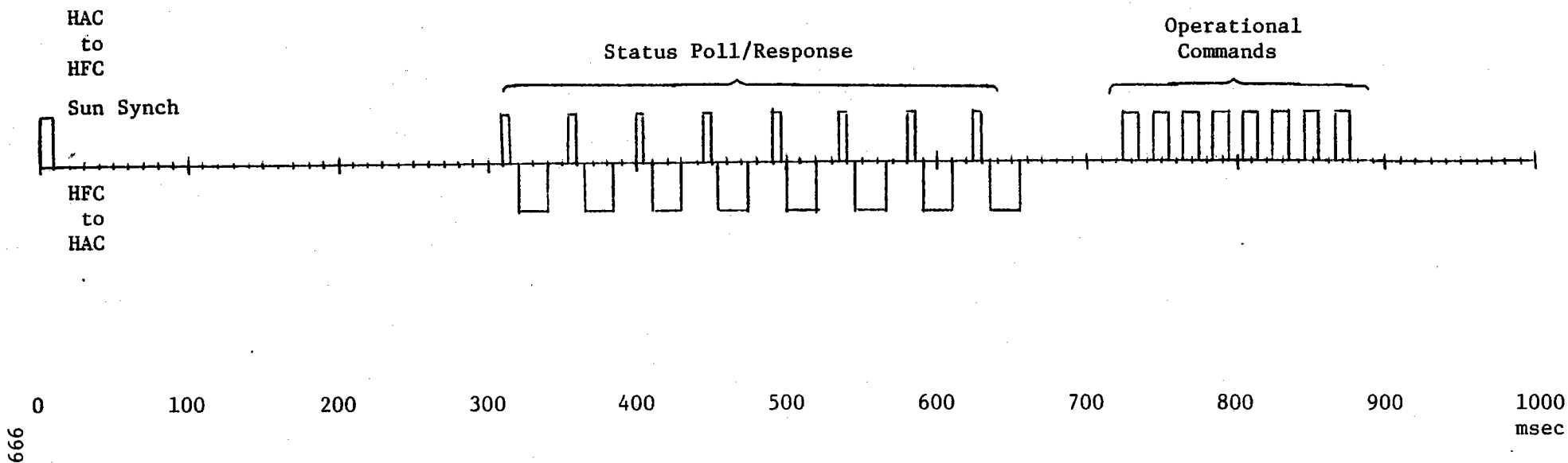


Figure 3.7-1 FLDCOM One-Second Communications Time Frame

TABLE 3.7-I HC INITIALIZATION COMMAND FORMAT
MESSAGE LAYOUT

APPLICATION HAC HFC _____ MESSAGE TYPE COMMAND _____
HC
INITIALIZATION

PROGRAMMER D. POWELL _____ DATE 2/8/80 _____

1	HEADER BYTE	1	0016
	#		3 bits HFC #; 5 bits HC#
	CAZ		HC current azimuth
5	CEL	5	HC current elevation
	BAZ		HC bias azimuth
	BEL		HC bias elevation
10		10	
	HX		Heliostat field x-coord. (scaled binary point 14.)
15	HY	15	Heliostat field y-coord. (scaled binary point 14.)
	HZ		Heliostat field z-coord. (scaled binary point 14.)
20	CHECKSUM	20	8 bit checksum such that <u>all</u> message bytes sum to zero.
25		25	
30		30	

TABLE 3.7-II BEAM POINTING COMMAND FORMAT

MESSAGE LAYOUT

APPLICATION HAC HFC _____ MESSAGE TYPE _____ BEAM POINTING
COMMAND

PROGRAMMER D. POWELL _____ DATE 2/8/80

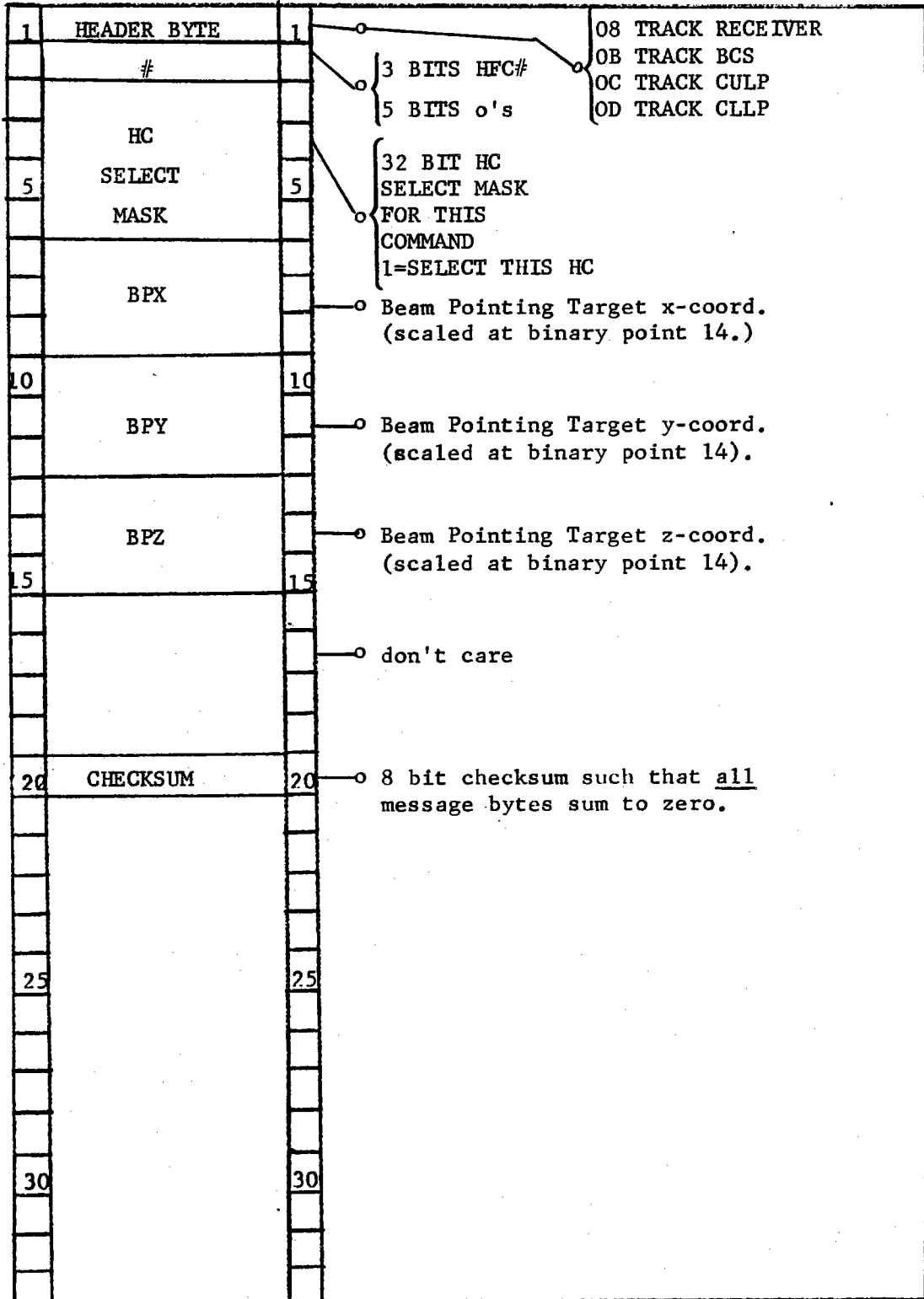


TABLE 3.7-V STATUS POLL COMMAND FORMAT
MESSAGE LAYOUT

APPLICATION HAC HFC MESSAGE TYPE STATUS POLL

PROGRAMMER D. POWELL DATE 2/8/80

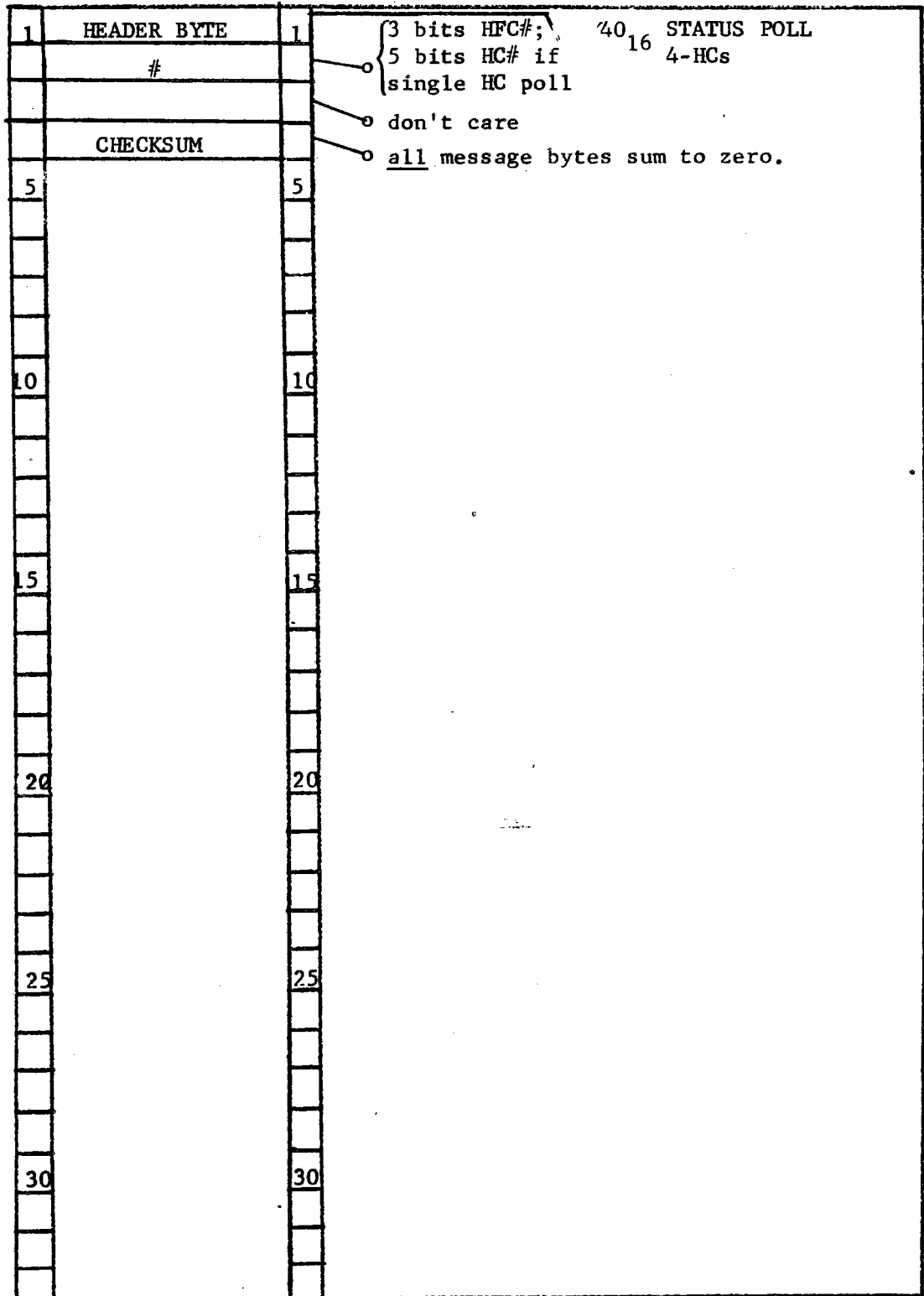


TABLE 3.7-VI FOUR HELIOSTAT STATUS RESPONSE FORMAT

MESSAGE LAYOUT

4-HC
STATUS

APPLICATION HFC HAC

MESSAGE TYPE RESPONSE

PROGRAMMER D. POWELL

DATE 2/8/80

1	HEADER BYTE	1	50 ₁₆
	#		3 bit HFC#; 5 bit HC# of the first HC in the group
	COMMAND RECEIVED MASK		32 bits of command received mask 0=this HC did not receive a command 1=HC received a command OK
5		5	
	HC STATUS		See breakdown of HC status-- HC status for HC#n
	CAZ		Current azimuth for HC#n
10		10	
	CEL		Current elevation for HC#n
	HC STATUS		See breakdown of HC status-- HC status for HC# n+1
	CAZ		Current azimuth for HC# n+1
15		15	
	CEL		Current elevation for HC# n+1
	HC STATUS		See breakdown of HC status-- HC status for HC# n+2
20		20	
	CAZ		Current azimuth for HC# n+2
	CEL		Current elevation for HC# n+2
	HC STATUS		See breakdown of HC status-- HC status for HC# n+3
25		25	
	CAZ		Current azimuth for HC# n+3
	CEL		Current elevation for HC# n+3
30		30	
	HFC STATUS		See breakdown of HFC status-- HFC status this HFC

MESSAGE LAYOUT

APPLICATION HC STATUS BYTES MESSAGE TYPE HC STATUS BIT BREAKDOWN

PROGRAMMER D. Powell DATE 2/8/80

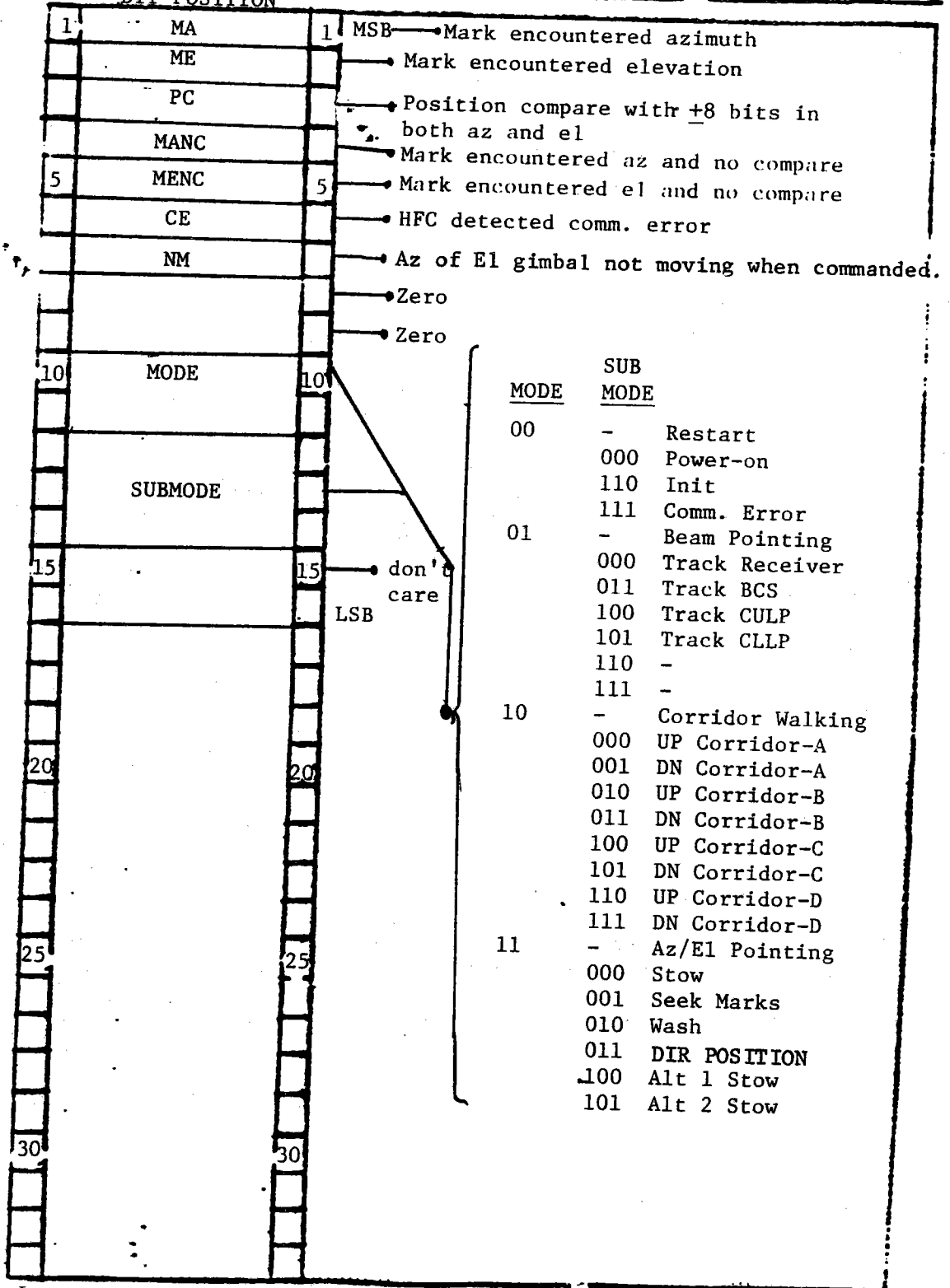


TABLE 3.7-VII HELIOSTAT CONTROLLER STATUS BIT BREAKDOWN
 1007

MESSAGE LAYOUT

APPLICATION HFC STATUS BYTES MESSAGE TYPE HFC STATUS BREAKDOWN

PROGRAMMER D. A. Powell DATE 2/8/80

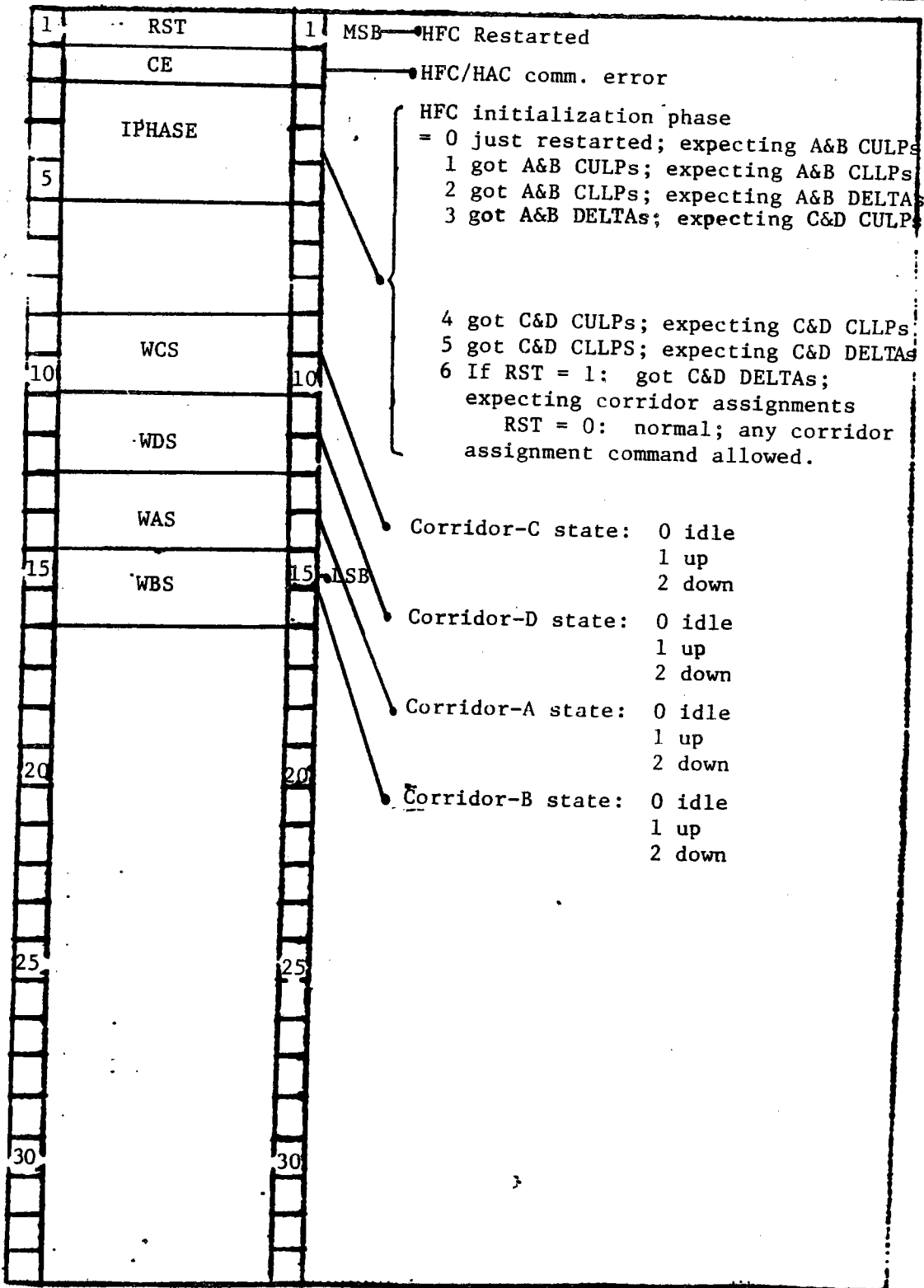


TABLE 3.7-VIII HFC STATUS BREAKDOWN

MESSAGE LAYOUT

APPLICATION HAC HFC

MESSAGE TYPE SUN/SYNC

PROGRAMMER D. A. Powell

DATE 2/8/80

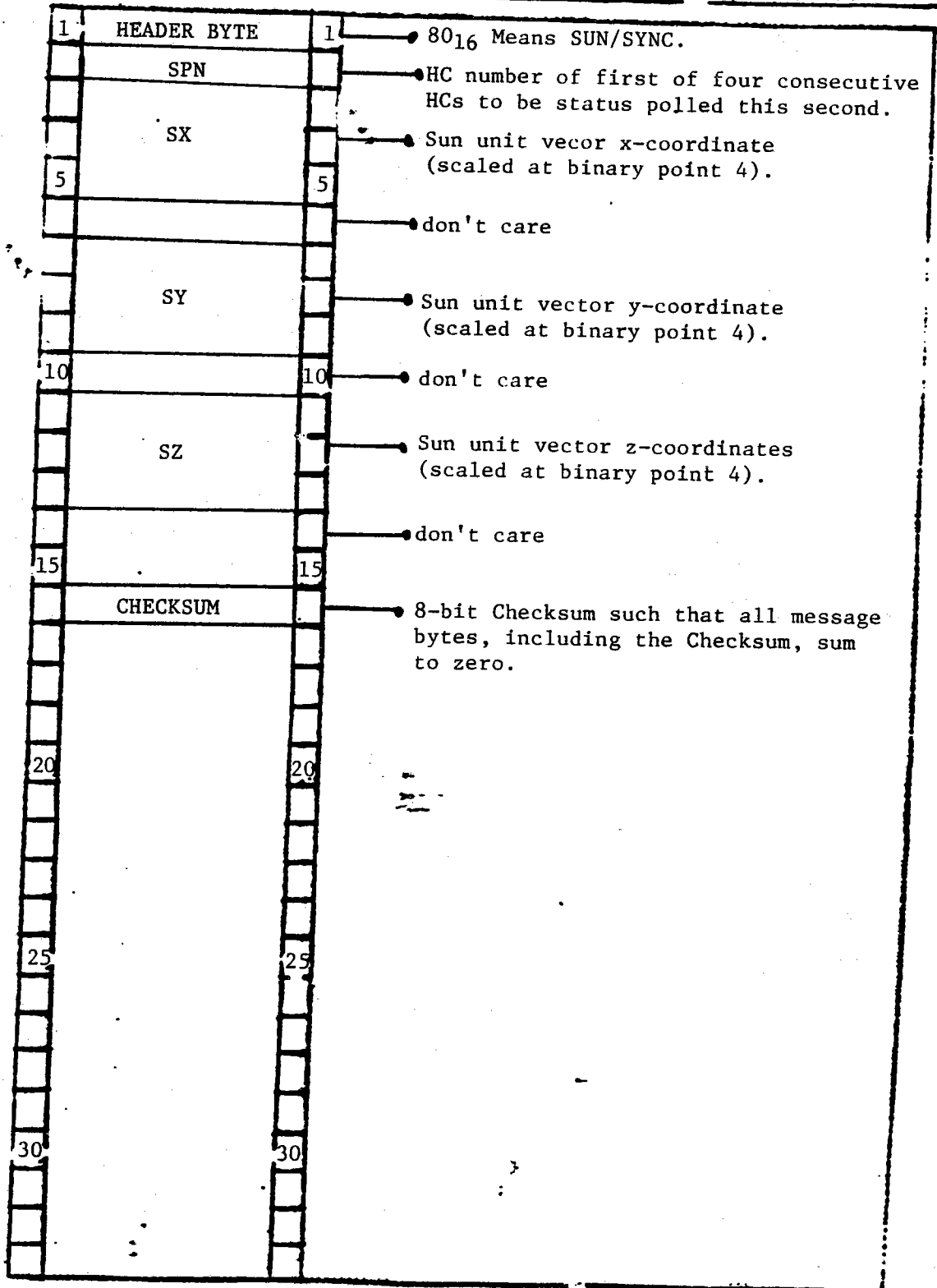


TABLE 3.7-IX SUN POSITION COMMAND FORMAT

MESSAGE LAYOUT

APPLICATION HAC HFC MESSAGE TYPE HFC INIT. COMMAND SUBTYPES 0,1,2

PROGRAMMER D. A. Powell DATE 2/8/80

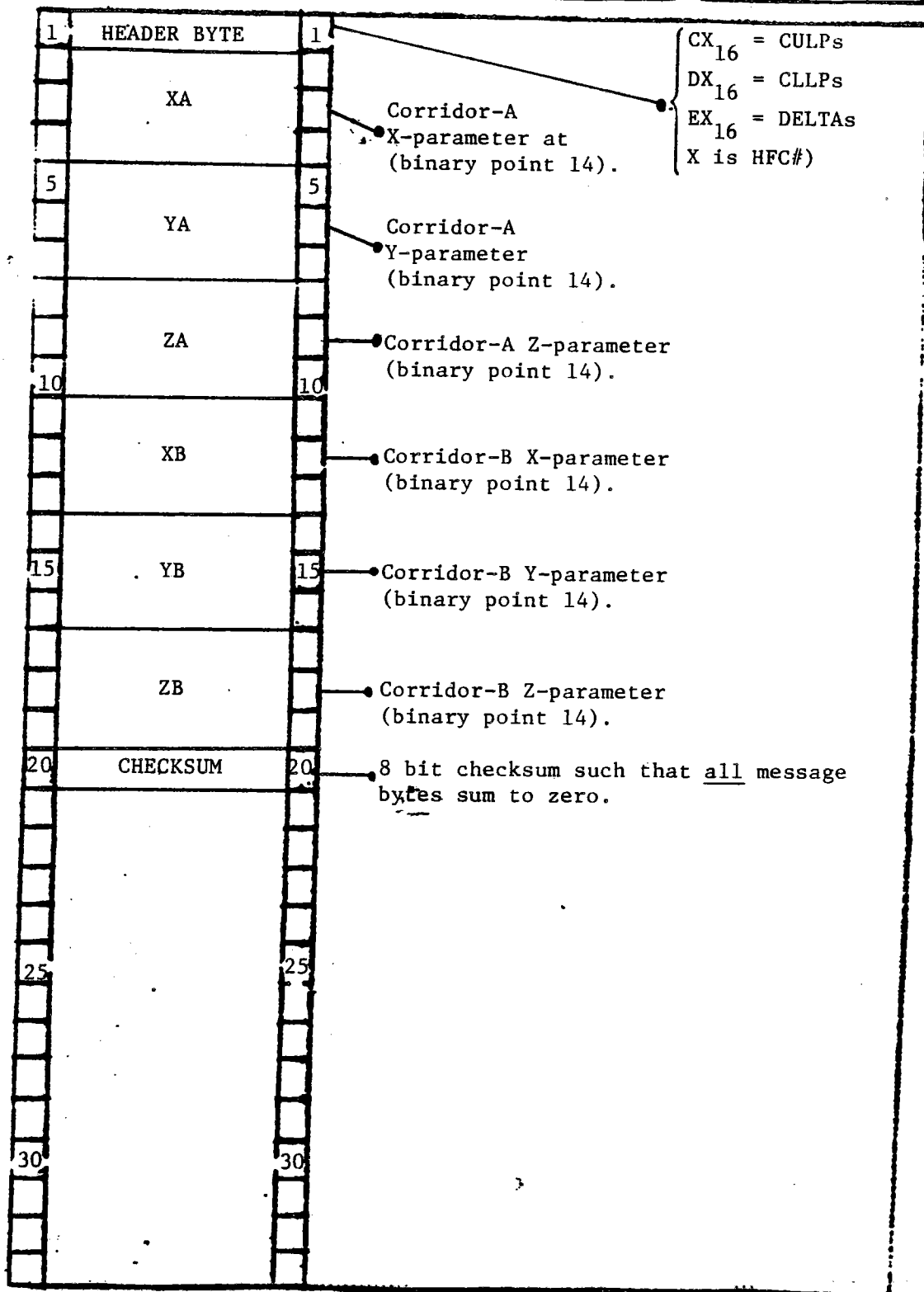


TABLE 3.7-X HFC INITIALIZATION COMMAND FORMAT (SUBTYPES 0,1,2)

TABLE 3.7-XI. HFC INITIALIZATION COMMAND FORMAT (SUBTYPES 3,4,5,) MESSAGE LAYOUT

APPLICATION HAC HFC MESSAGE TYPE SUBTYPES 3,4,5 HFC INIT. COMMAND

PROGRAMMER E. Madigan DATE 2/8/80

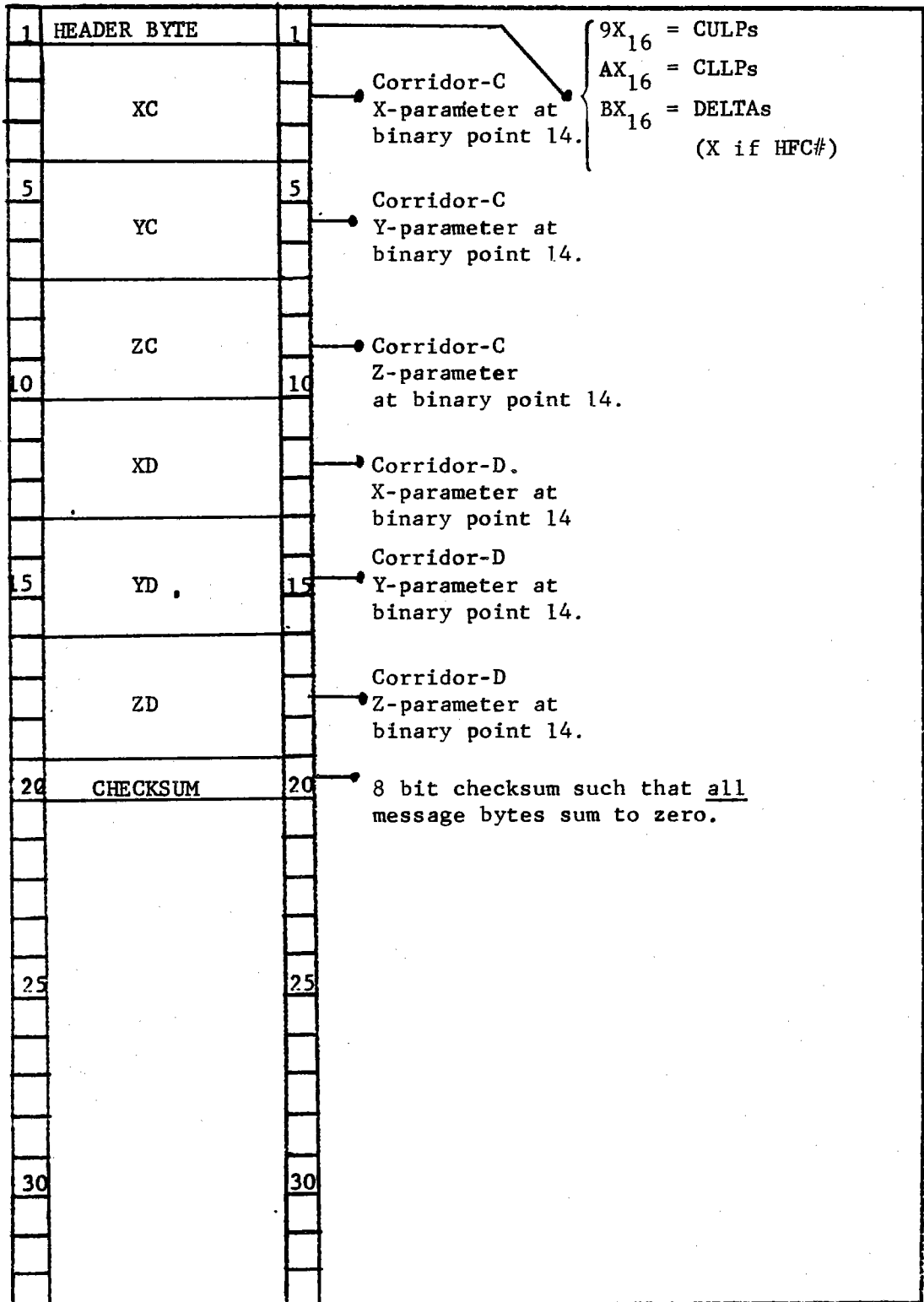


TABLE 3.7-XII HFC INITIALIZATION COMMAND FORMAT (SUBTYPE 6)
MESSAGE LAYOUT

APPLICATION HAC HFC _____ MESSAGE TYPE HFC INIT. COMMAND SUBTYPE 6

PROGRAMMER E. Madigan _____ DATE 7/8/80

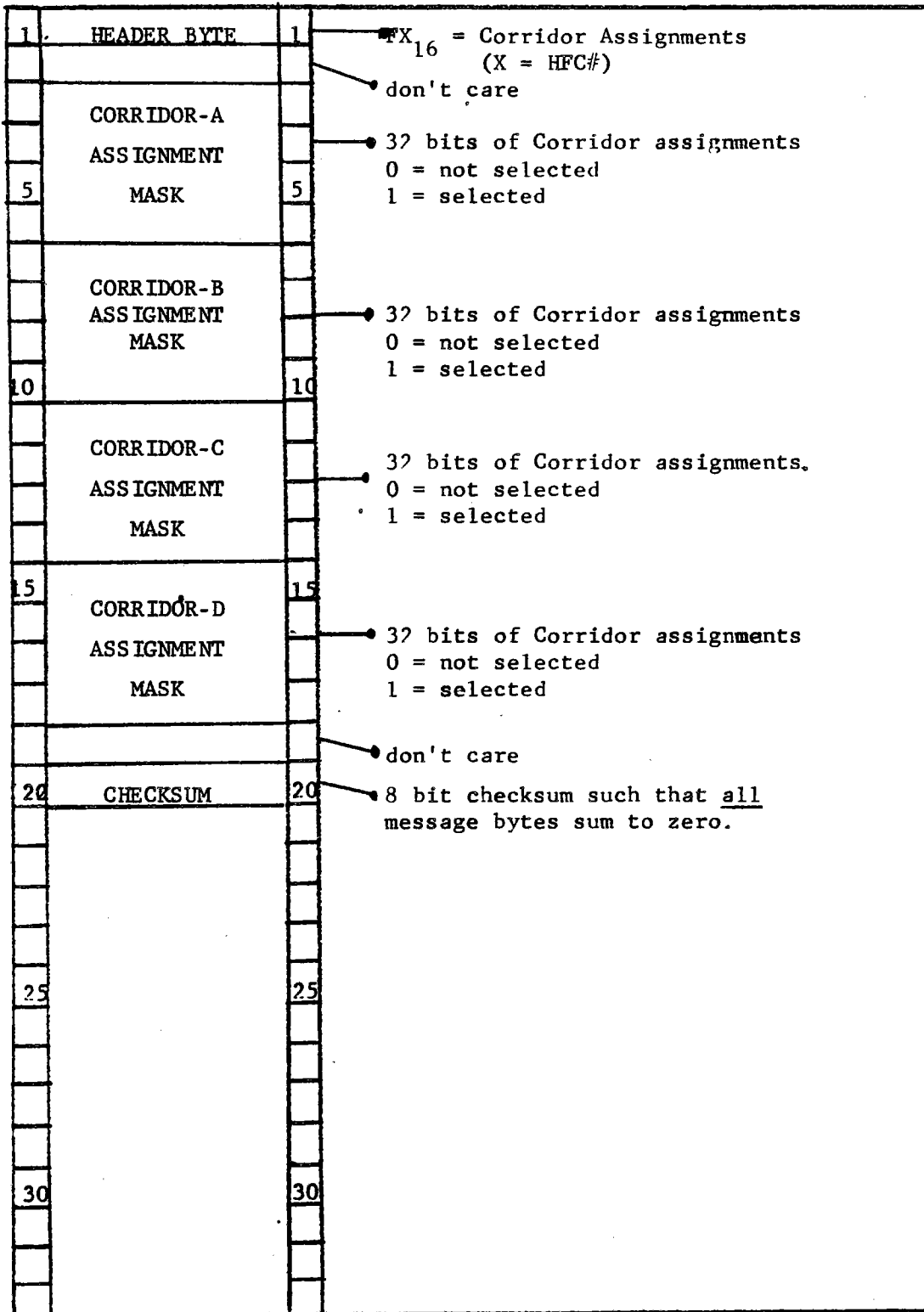


TABLE 3.7-XIII HC SUN/SYNCHRONIZATION COMMAND FORMAT (BEAM POINTING)
MESSAGE LAYOUT

APPLICATION HFC HC SUN/SYNC/CMD
MESSAGE TYPE BEAM POINTING

PROGRAMMER D. Powell DATE 2/8/80

1	HEADER BYTE	1	Receiver
			BCS
	SX		CULP (standby)
			CLLP
5	SY	5	Sun unit vector y-coord. at binary point 4.
	SZ		Sun unit vector z-coord. at binary point 4.
10		10	
	CORRIDOR-A SELECT MASK		32 bit HC select mask of HCs to point at corridor-A.
15	WAX	15	Corridor-A x-coord. at binary point 14.
	WAY		Corridor-A y-coord. at binary point 14.
20		20	
	WAZ		Corridor-A z-coord. at binary point 14.
25	CORRIDOR-B SELECT MASK	25	32 bit HC select mask of HCs to point at Corridor-B.
	WBX		Corridor-B x-coord. at binary point 14.
30		30	
	WBY		Corridor-B y-coord. at binary point 14.

TABLE 3.7-XIII HC SUN/SYNCHRONIZATION COMMAND FORMAT (BEAM POINTING) (con't.)
MESSAGE LAYOUT

APPLICATION HFC HC		MESSAGE TYPE		SUN/SYNC/CMD BEAM POINTING (CONTINUED)
PROGRAMMER D. Powell		DATE 2/8/80		
35	WBZ	35	Corridor-B z-coord. at binary point 14.	
	CORRIDOR-C SELECT MASK		32 bit HC select mask of HCs to point at Corridor-C	
40	WCX	40	Corridor-C x-coord. at binary point 14.	
45	WCY	45	Corridor-C y-coord. at binary point 14.	
	WCZ		Corridor-C z-coord. at binary point 14.	
50	CORRIDOR-D SELECT MASK	50	32 bit select mask of HCs to point at Corridor-D	
55	WDX	55	Corridor-D x-coord. at binary point 14.	
	WDY		Corridor-D y-coord. at binary point 14.	
60	WDZ	60	Corridor-D z-coord. at binary point 14.	
65	BEAM POINTING SELECT MASK	65	32 bit select mask of HCs to beam point	

TABLE 3.7-XIV HC SUN/SYNCHRONIZATION COMMAND FORMAT (AZIMUTH/ELEVATION POINTING)

MESSAGE LAYOUT

APPLICATION HFC HC SUN/SYNC/CMD
 MESSAGE TYPE AZ/EL POINTING

PROGRAMMER D. Powell DATE 2/8/80

1	HEADER BYTE	1	
	SX		Sun unit vector x-coord at binary point 4.
			78 ₁₆ Stow
			79 ₁₆ Seek Marks
			7A ₁₆ Wash
			7B ₁₆ DIR POSITION
5	SY	5	Sun unit vector y-coord at binary point 4.
			7C ₁₆ ALT1STOW
			7D ₁₆ ALT2STOW
	SZ		Sun unit vector z-coord at binary point 4.
10		10	
	CORRIDOR-A SELECT MASK		32 bit HC select mask of HCs to point at corridor-A.
15		15	
	WAX		Corridor-A x-coord. at binary point 14.
	WAY		Corridor-A y-coord at binary point 14.
20		20	
	WAZ		Corridor-A z-coord. at binary point 14.
	CORRIDOR-B SELECT MASK		32 bit HC select mask of HCs to point at Corridor-B.
25		25	
	WBX		Corridor-B x-coord at binary point 14.
30		30	
	WBY		Corridor-B y-coord at binary point 14.

TABLE 3. 7-XIV HC SUN/SYNCHRONIZATION COMMAND FORMAT (AZIMUTH/ELEVATION POINTING) (con't.)

MESSAGE LAYOUT

SUN/SYNC/CMD
AZ/EL POINTING

APPLICATION HFC HC _____ MESSAGE TYPE (CONTINUED) _____

PROGRAMMER D. Powell. _____ DATE 2/8/80 _____

35	WBZ	35	Corridor-B z-coord at binary point 14.
	CORRIDOR-C SELECT MASK		32 bit HC select mask of HCs to point at Corridor-C
40		40	
	WCX		Corridor-C x-coord at binary point 14.
45	WCY	45	Corridor-C y-coord. at binary point 14.
	WCZ		Corridor-C z-coord. at binary point 14
50	CORRIDOR-D SELECT MASK	50	32 bit HC select mask of HCs to point at Corridor-D
55	WDX	55	Corridor-D x-coord at binary point 14.
	WDY		Corridor-D y-coord at binary point 14.
60	WDZ	60	Corridor-D z-coord at binary point 14.
	AZ/EL POINTING SELECT MASK		32 bit select mask of HCs to AZ/EL point.
65		65	

TABLE 3.7-XV SUN/SYNCHRONIZATION COMMAND FORMAT (HC INITIALIZATION)
MESSAGE LAYOUT

APPLICATION HFC HC SUN/SYNC/CMD
MESSAGE TYPE HC INITIALIZATION

PROGRAMMER D. Powell

DATE 2/8/80

1	HEADER BYTE	1	3 bits = zero 5 bits HC#
	SX		Sun unit vector x-coord at binary point 4.
5	SY	5	Sun unit vector y-coord at binary point 4.
	SZ		Sun unit vector z-coord at binary point 4.
10	CORRIDOR-A SELECT MASK	10	32 bit HC select mask of HCs to point at Corridor-A
15	WAX	15	Corridor-A x-coord at binary point 14.
	WAY		Corridor-A y-coord. at binary point 14.
20	WAZ	20	Corridor-A z-coord. at binary point 14.
25	CORRIDOR-B SELECT MASK	25	32 bit HC select mask of HCs to point at Corridor-B.
	WBX		Corridor-B x-coord at binary point 14.
30	WBY	30	Corridor-B y-coord at binary point 14.

TABLE 3.7-XV SUN/SYNCHRONIZATION COMMAND FORMAT (HC INITIALIZATION) (con't.)

MESSAGE LAYOUT

APPLICATION		HFC	HC	MESSAGE TYPE (CONTINUED)
PROGRAMMER			D. Powell	DATE 2/8/80
35	WBZ	35		Corridor-B z-coord at binary point 14.
	CORRIDOR-C SELECT MASK			32 bit HC select mask of HCs to point at Corridor-C
40	WCX	40		Corridor-C x-coord at binary point 14.
45	WCY	45		Corridor-C y-coord at binary point 14.
	WCZ			Corridor-C z-coord at binary point 14.
50	CORRIDOR-D SELECT MASK	50		32 bit HC select mask of HCs to point at Corridor-D
55	WDX	55		Corridor-D x-coord at binary point 14.
	WDY			Corridor-D y-coord at binary point 14.
60	WDZ	60		Corridor-D z-coord at binary point 14.
	CAZ			Current azimuth
65	CEL	65		Current elevation

TABLE 3.7-XVI HC COMMAND RESPONSE FORMAT

MESSAGE LAYOUT

APPLICATION HC HFC _____ MESSAGE TYPE RESPONSE _____

COMMAND

PROGRAMMER D. Powell _____ DATE 2/8/80 _____

1	C. R. BYTE	1	3 bits = 6 ₁₆ 5 bits HC#
5		5	
10		10	
15		15	
20		20	
25		25	
30		30	

TABLE 3.7-XVII HC STATUS POLL COMMAND FORMAT

MESSAGE LAYOUT

APPLICATION HFC HC STATUS
 MESSAGE TYPE POLL

PROGRAMMER D. Powell DATE 2/8/80

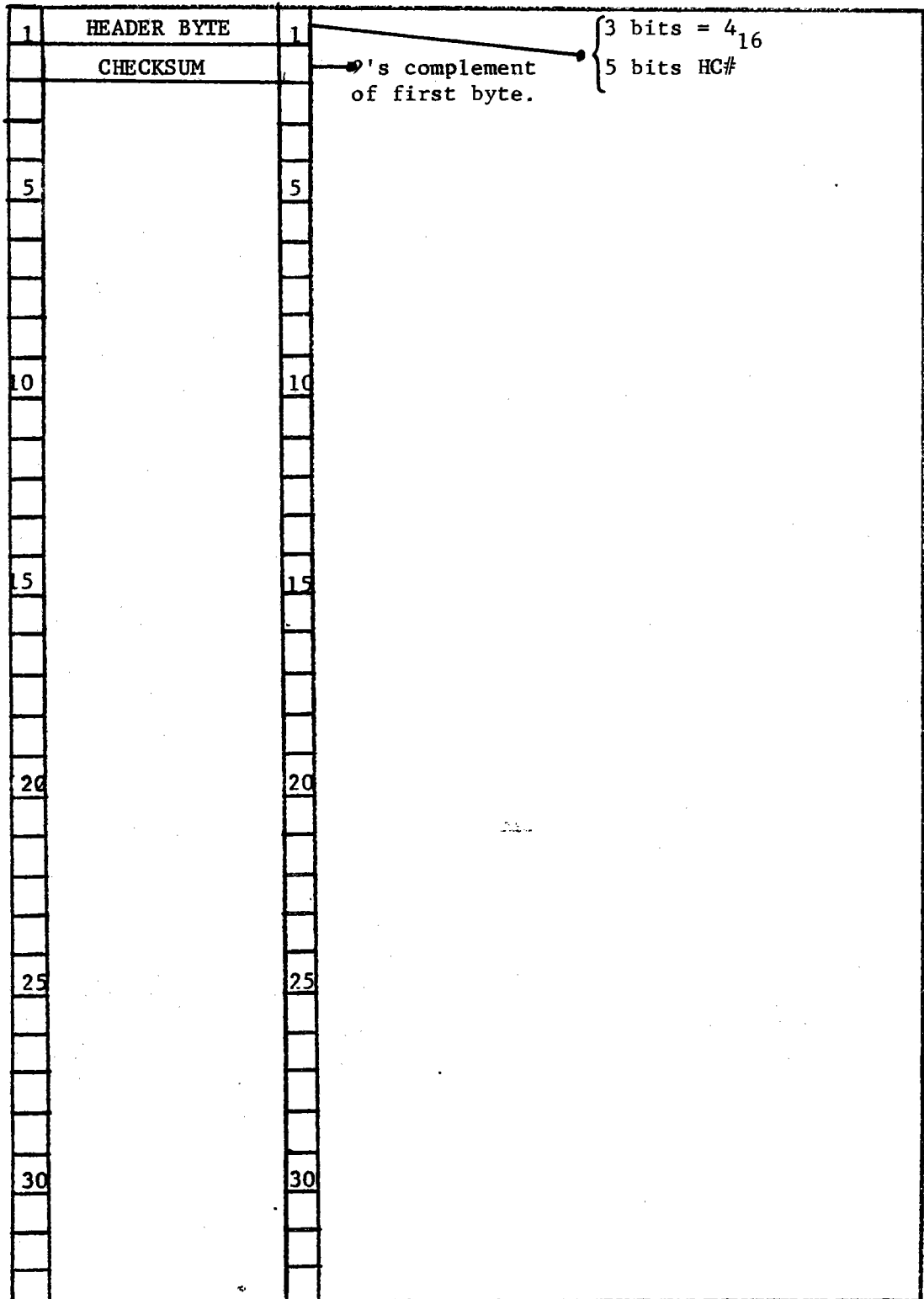


TABLE 3.7-XVIII HC STATUS RESPONSE FORMAT

MESSAGE LAYOUT

APPLICATION HC HFC _____ MESSAGE TYPE _____ STATUS RESPONSE _____

PROGRAMMER D. Powell _____ DATE 2/8/80 _____

1	HEADER BYTE	1	{ 3 bits = 5 16 { See HC status bit breakdown for HFC HAC status response { 5 bits HC#
	HC STATUS		
	CAZ		Current azimuth
5		5	
	CEL		Current elevation
	CHECKSUM		8 bit checksum such that <u>all</u> message bytes sum to zero.
10		10	
15		15	
20		20	
25		25	
30		30	

TABLE 3.7-XIX OCS/DAS-HAC COMMANDS FORMAT

MESSAGE LAYOUT

APPLICATION OCS/DAS - HAC OCS / DAS
MESSAGE TYPE COMMANDS

PROGRAMMER T. Ladewig DATE 2/8/80

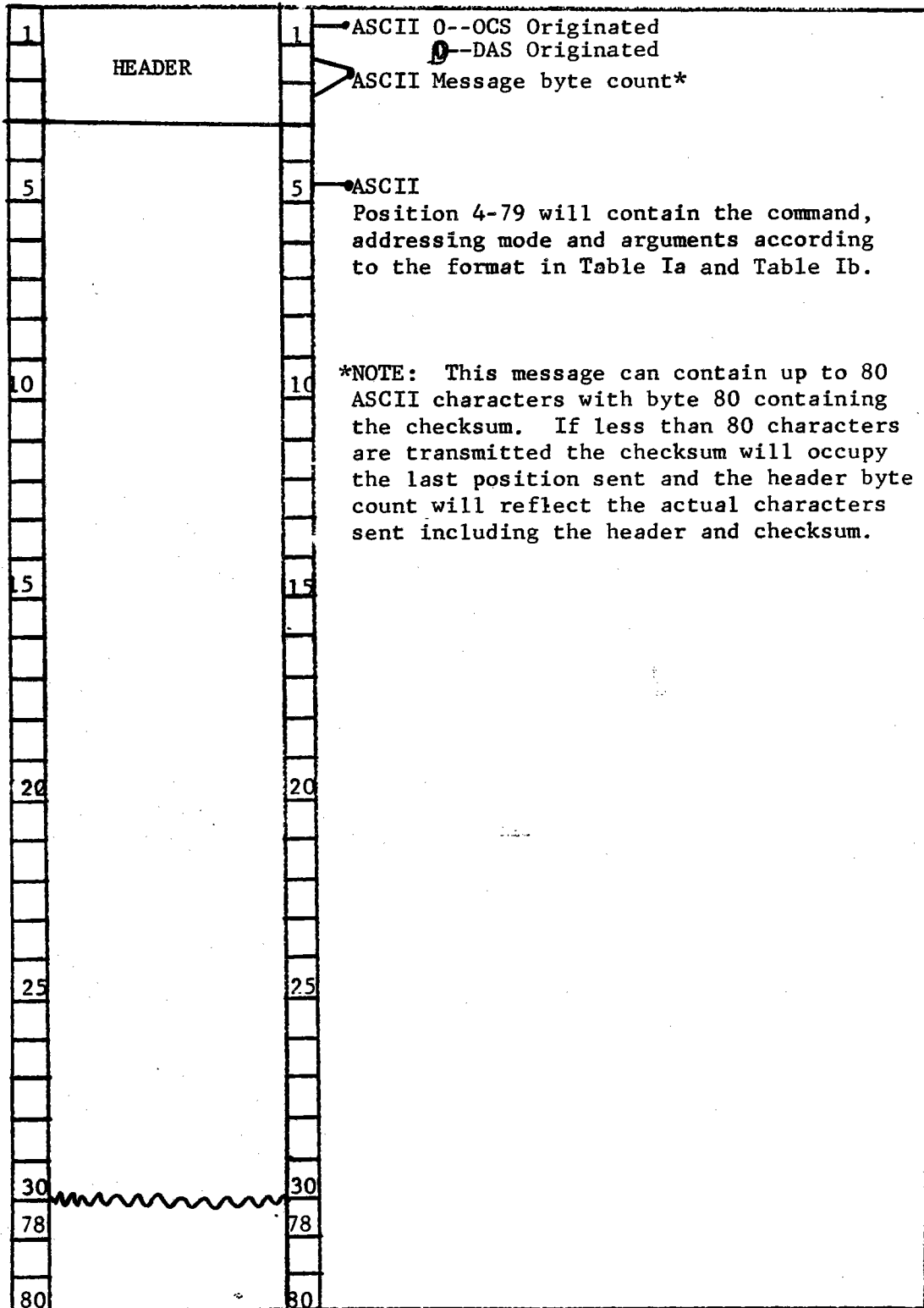


TABLE 3.7-XXI HAC-OCS/DAS ENVIRONMENT STATUS (FIELD) FORMAT

MESSAGE LAYOUT

APPLICATION HAC - OCS/DAS

MESSAGE TYPE STATUS - FIELD

PROGRAMMER T. Ladewig

DATE 2/8/80

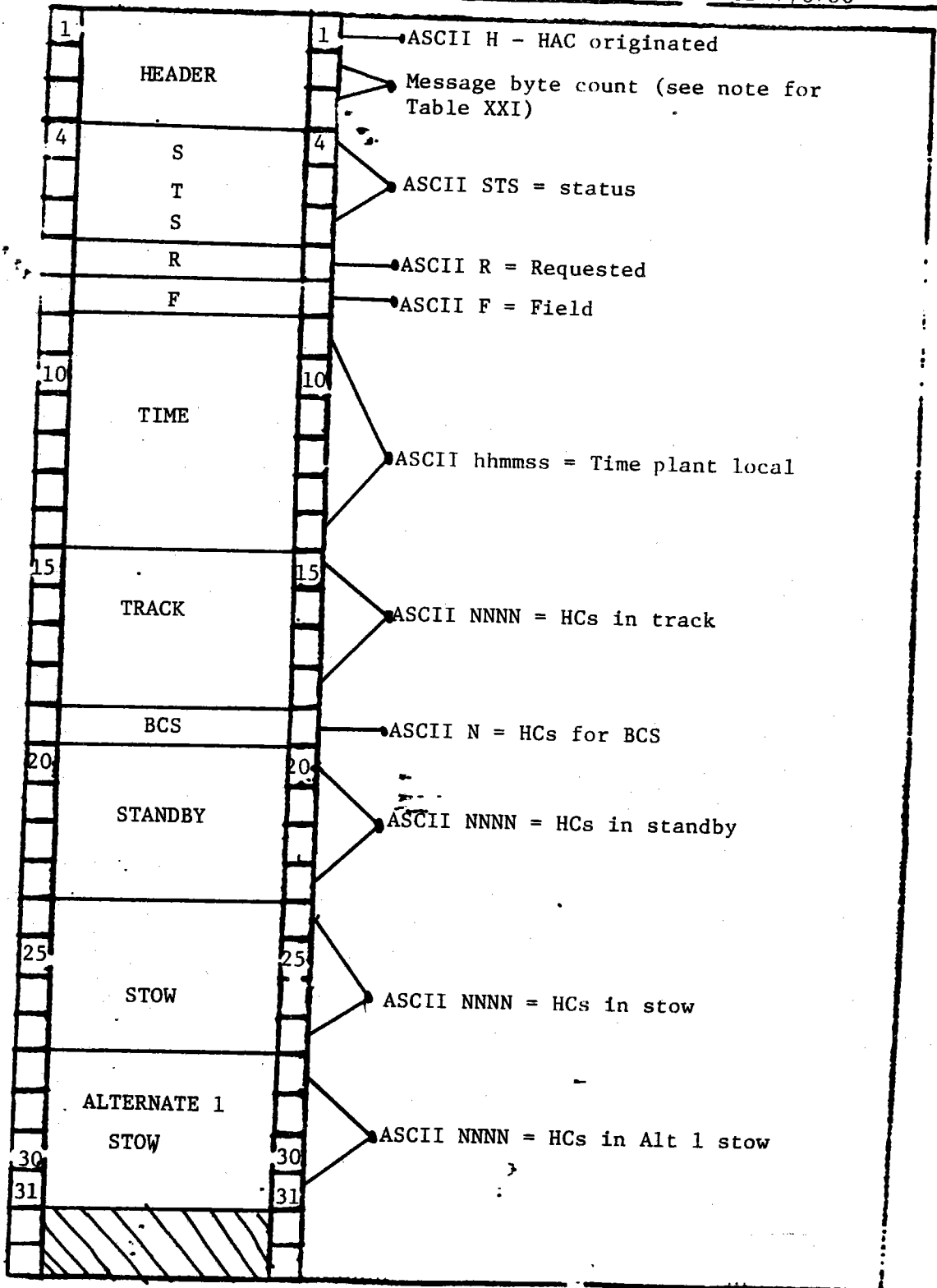


TABLE 3.7-XXI HAC-OCS/DAS ENVIRONMENT STATUS (FIELD) FORMAT (CONTINUED)

MESSAGE LAYOUT

APPLICATION HAC OCS/DAS STATUS-FIELD
MESSAGE TYPE (CONTINUED)

PROGRAMMER T. Ladewig DATE 2/8/80

32	ALTERNATE ? STOW	32	ASCII NNNN = HCs in ALT?STOW
35	TRANSITION	35	ASCII NNNN = HCs in Transition
40	WASH	40	ASCII NNNN = HCs In Wash
45	DIRECTED POSITION	45	ASCII NNNN = HCs in directed position
50	OFFLINE	50	ASCII NNNN = HCs in offline
55	MARK	55	ASCII NNNN = HCs in Mark
60	UNUSED	60	
77	INIT	77	ASCII NNNN = HCs in INIT
80	CHECKSUM	80	8 bit checksum such that all bytes sum to zero.

TABLE 3.7-XXII HAC-OCS/DAS ENVIRONMENT STATUS (MODE) FORMAT
MESSAGE LAYOUT

APPLICATION HAC-OCS/DAS MESSAGE TYPE STATUS--MODE

PROGRAMMER T. Ladewig DATE 2/8/80

1		1	ASCII H = HAC originated
	HEADER		Message byte count (see Note for Table XXI)
5		5	ASCII STS - STATUS
	CONDITION		ASCII R = Requested A = Alarmed*
	M		ASCII M = Mode
10		10	ASCII TRK = Track BCS = BCS STB = Standby STO = Stow AL1 = ALT1STOW AL2 = ALT2STOW TRN = Transition WSH = Wash DPO = Directed Position OFF = Offline MRK = Mark INI = Init
	MODE		
15		15	ASCII hhhmss - Time
	TIME		
	HC#		ASCII 4 digit HC number. Additional messages will be used to transmit additional HCs.
20		20	
	HC#		
25		25	
	...		
	UNUSED		
80	CHECKSUM	80	Checksum

*NOTE: Alarmed will be used to report HCs unable to respond to a STHIWIND command. HCs in WASH, OFFLINE, and DIRECTED POSITION will be transmitted

TABLE 3.7-XXIII HAC-OCS/DAS ENVIRONMENT STATUS (HC) FORMAT

MESSAGE LAYOUT

APPLICATION HAC OCS /DAS MESSAGE TYPE STATUS
 INDIVIDUAL HC

PROGRAMMER T. Ladewig DATE 2/8/80

1	HEADER	1	• ASCII H = HAC Originated
			• Message byte count (See note for Table XXI)
5	STS	5	• ASCII STS = Status
	R		• ASCII R = Requested
	H		• ASCII H = Individual Status
10	TIME	10	
15	HC#	15	• ASCII 4 Digit HC Number
20	AZIMUTH	20	
			• ASCII 7 Digit SNNN.NN Azimuth
25	ELEVATION	25	
			• ASCII 7 Digit SNNN.NN Elevation
30		30	
32		32	

TABLE 3.7-XXIV HAC-OCS/DAS ENVIRONMENT STATUS (RING) FORMAT

MESSAGE LAYOUT

APPLICATION HAC-OCS/DAS STATUS RING
MESSAGE TYPE RING

PROGRAMMER T. Ladewig DATE 5/12/80

1	HEADER	1	ASCII H = HAC Originated
			Message byte count (see note for Table XXI.)
5	STS	5	ASCII STS = STATUS
	R		ASCII R = Requested
	R		ASCII R = Ring
10	TIME	10	ASCII Time Plant Local
15	RING	15	ASCII 1-digit Ring number requested
	SEGMENT		ASCII 3-digit Segment number
20	HCs in SEGMENT	20	ASCII 2-digit number of HCs in Segment
	TRACK		ASCII 2-digit number of HCs in TRACK in Segment
	STANDBY		ASCII 2-digit number of HCs in STANDBY in Segment
	REPEAT ↓		REPEAT as many times as necessary to cover complete ring - multiple messages may be necessary.
80	CHECKSUM	80	

TABLE 3.7-XXV HAC/BCS BCS INITIATION REQUEST MESSAGE FORMAT

APPLICATION HAC/BCS MESSAGE TYPE BCS Measurement

PROGRAMMER T. Ladewig DATE 5/12/80

1	HEADER	1	ASCII H - HAC originated
			Message Byte Count (See note for Table XXI.)
5	B C S	5	ASCII BCS = BCS message
			Message number = 1
	0 1		
10	TIME	10	ASCII hhmmss = Time
15	BCS FILE TITLE	15	
20		20	ASCII BCS file title
25		25	
30		30	
80	CHECKSUM		CHECKSUM

TABLE 3.7-XXVI BCS/HAC BCS INITIATION RESPONSE MESSAGE FORMAT
MESSAGE LAYOUT

APPLICATION BCS/HAC

MESSAGE TYPE BCS Measurement

PROGRAMMER T. Ladewig

DATE 5/12/80

1		1	• ASCII 0 - OCS originated
	HEADER		• Message Byte Count (See note for Table XXI)
5	B C S	5	• ASCII BCS = BCS Message
	0 2		• Message number = 2
10		10	
	BCS FILE TITLE		
15		15	• ASCII BCS file title from BCS-1 title
	SOUTH		
20	WEST	20	
	NORTH		• BCS targets status: 1 = operational 0 = non-operational
	EAST		
	GO/STOP		• BCS initiation permission: 1 = go 2 = no-go
25		25	
30		30	
80	CHECKSUM		checksum

TABLE 3.7-XXVII HAC/BCS BCS MEASUREMENT INITIATION REQUEST MESSAGE FORMAT
MESSAGE LAYOUT

APPLICATION HAC/BCS MESSAGE TYPE BCS Measurement

PROGRAMMER T. Ladewig DATE 5/12/80

1		1	• ASCII H - HAC originated
	HEADER		• Message byte count (see note for Table XXI)
5	B C S	5	• ASCII BCS = BCS message
	0 3		• Message number = 3
10	HELIOSTAT	10	• Heliostat number
	BCS TARGET		• BCS Target: 1 = South 2 = West 3 = North 4 = East
15	MODE	15	• Mode that heliostat is presently in (for coding see byte 9-11, Table XXIV.)
20		20	
25		25	
30		30	
80	CHECKSUM		Checksum

TABLE 3.7-XXVIII BCS/HAC BCS MEASUREMENT INITIATION RESPONSE MESSAGE FORMAT
MESSAGE LAYOUT

APPLICATION BCS/HAC MESSAGE TYPE BCS Measurement

PROGRAMMER T. Ladewig DATE 5/12/80

1		1	• ASCII 0 - OCS originated
	HEADER		• Message byte count (See note for Table XXI)
5	B C S	5	• ASCII BCS = BCS Message
	0 4		• Message number = 4
10	HELIOSTAT	10	• Heliostat number
	BCS TARGET		• BCS TARGET: (see byte 13, Table XXIX)
15		15	
20		20	
25		25	
30		30	
80	CHECKSUM		Checksum

TABLE 3.7-XXIX HAC/BCS HELIOSTAT ON BCS TARGET MESSAGE FORMAT
MESSAGE LAYOUT

APPLICATION HAC/BCS MESSAGE TYPE BCS Measurement

PROGRAMMER T. Ladewig DATE 5/12/80

1		1	• ASCII H - HAC originated
	HEADER		• Message Byte Count (see note for Table XXI)
5	B C S	5	• ASCII BCS = BCS Message
	0 5		• Message number = 5
10	HELIOSTAT	10	• Heliostat number
	BCS TARGET		• BCS Target (see byte 13, Table XXIX)
15	HELIOSTAT	15	• Blocking/shadowing heliostat numbers (up to 5)
20	HELIOSTAT	20	
25	HELIOSTAT	25	
30	HELIOSTAT	30	

TABLE 3.7-XXX BCS/HAC HELIOSTAT BCS REMOVAL REQUEST MESSAGE FORMAT
MESSAGE LAYOUT

APPLICATION BCS/HAC MESSAGE TYPE BCS Measurement

PROGRAMMER T. Ladewig DATE 5/12/80

1		1	• ASCII 0 - OCS Originated
	HEADER		• Message Byte Count (See Note for Table XXI)
5	B C S	5	• ASCII BCS = BCS Message
	0 6		• Message number = 6
10	HELIOSTAT	10	• Heliostat number
	BCS TARGET		• BCS Target (See byte 13, Table XXIX)
15		15	
20		20	
25		25	
30		30	
80	CHECKSUM	80	Checksum

TABLE 3.7-XXXI HELIOSTAT BCS REMOVAL RESPONSE MESSAGE FORMAT
MESSAGE LAYOUT

APPLICATION HAC/BCS MESSAGE TYPE BCS Measurement

PROGRAMMER T. Ladewig DATE 5/12/80

1		1	• ASCII H - HAC originated
	HEADER		• Message Byte Count (See note for Table XXI)
5	B C S	5	• ASCII BCS = BCS Message
	0 7		• Message number = 7
10	HELIOSTAT	10	• Heliostat number
	BCS TARGET		• BCS Target (see byte 13, Table XXIX)
15	MODE	15	• Mode that heliostat was removed to (for coding see byte 9-11, Table XXIV)
20		20	
25		25	
30		30	
80	CHECKSUM		Checksum

TABLE 3. 7-XXXII BCS/HAC - BCS MEASUREMENT RESULTS

MESSAGE LAYOUT

APPLICATION BCS/HAC

MESSAGE TYPE BCS MEASUREMENT

PROGRAMMER T. Ladewig

DATE 5/12/80

1		1	ASCII 0 - OCS originated
	HEADER		Message Byte Count (see note for Table XXI)
	B		
5	C	5	ASCII BCS = BCS Message
	S		
	0		Message Number = 8
	8		
10	HELIOSTAT	10	Heliostat Number
	BCS TARGET		BCS TARGET (see byte 13, Table XXIX)
	MEASUREMENTS		Number of BCS scans taken
15		15	
	CENTROID ALARM		Centroid Alarm Flag (1=Alarm)
	POWER ALARM		Power Alarm Flag (1=Alarm)
	SUCCESS FLAG		Measurement Success Flag
			0 = good measurement
			1 = spilling target
			2 = off target
			3 = inconsistent measurements
20	TIME	20	ASCII hhhmss = Time of day of BCS measurement
25	CENTROID OFFSET	25	
			ASCII Horizontal average centroid offset (±99.99 ft.)
30	CENTROID OFFSET	30	
			ASCII Vertical average centroid offset (±99.99 ft.)

TABLE 3.7-XXXIII HAC/BCS BCS MEASUREMENT RESULTS RESPONSE
MESSAGE LAYOUT

APPLICATION HAC/BCS MESSAGE TYPE BCS Measurement

PROGRAMMER T. Ladewig DATE 5/12/80

1		1	• ASCII H - HAC originated
	HEADER		• Message Byte Count (see note for Table XXI)
5	B C S	5	• ASCII BCS = BCS Message
	0 9		• Message number = 9
10	HELIOSTAT	10	• Heliostat number
	BCS TARGET		• BCS Target (see byte 13, Table XXIX)
15	TIME	15	• ASCII hhmmss = Time of day of BCS measurement (from bytes 19-24, Table XXXIV)
20	CENTROID OFFSET HORIZONTAL	20	• ASCII Horizontal centroid offset (from bytes 25-30, Table XXXIV)
25	CENTROID OFFSET VERTICAL	25	• ASCII Vertical centroid offset (from bytes 31-36, Table XXXIV)
30	AZIMUTH	30	• ASCII Azimuth Bias from measurement data

TABLE 3.7-XXXIII HAC/BCS - BCS MEASUREMENT RESULTS RESPONSE (CONTINUED)

MESSAGE LAYOUT

APPLICATION HAC/BCS

MESSAGE TYPE BCS MEASUREMENT

PROGRAMMER T. Ladewig

DATE 5/12/80

34		34	
35	AZIMUTH	35	ASCII ± NNN.NN
	(continued)		
40	ELEVATION	40	ASCII ± NNN.NN Heliostat Elevation Bias from Measurement Data
	BIAS		
45	CURRENT	45	ASCII ± NNN.NN Current Heliostat Azimuth BIAS
	AZIMUTH		
	BIAS		
50		50	
	CURRENT		ASCII ± NNN.NN Current Heliostat Elevation BIAS
	ELEVATION		
	BIAS		
55		55	
59		59	
80	CHECKSUM	80	

TABLE 3.7-XXXIV HAC/BCS BCS TERMINATION MESSAGE FORMAT

MESSAGE LAYOUT

APPLICATION HAC/BCS

MESSAGE TYPE BCS Measurement

PROGRAMMER T. Ladewig

DATE 5/12/80

1	HEADER	1	•ASCII H - HAC originated
			•Message Byte Count (see note for Table XXI.)
5	B C S	5	•ASCII BCS = BCS message
	1 0		•Message number = 10
10	TIME	10	•ASCII hhmmss = Time of termination (all heliostats are back to original mode)
15	BCS FILE TITLE	15	
20		20	•ASCII BCS file title (see bytes 15-24, Table XXVII.)
25		25	
30		30	
80	CHECKSUM	80	

TABLE 3.7-XXXV HAC/BCS HELIOSTAT MEASUREMENT HISTORICAL DATA REQUEST
 MESSAGE FORMAT
 MESSAGE LAYOUT

APPLICATION HAC/BCS MESSAGE TYPE BCS Measurement

PROGRAMMER T. Ladewig DATE 5/12/80

1		1	ASCII H - HAC originated
	HEADER		Message Byte Count (See note for Table XXI.)
	B		
5	C	5	ASCII BCS = BCS message
	S		
	1		Message number = 11
	1		
10	HELIOSTAT	10	Heliostat number
15		15	
20		20	
25		25	
30		30	
80	CHECKSUM	80	Checksum

TABLE 3.7-XXXVII HAC/BCS HELIOSTAT BIAS RESULTS

MESSAGE LAYOUT

APPLICATION HAC/BCS MESSAGE TYPE BCS Measurement

PROGRAMMER T. Ladewig DATE 5/12/80

1		1	• ASCII H - HAC Originated
	HEADER		• Message Byte Count (see note for Table XXI).
	B		
5	C	5	• ASCII BCS - BCS Message
	S		
	1		• Message Number = 13
	3		
10	HELIOSTAT	10	• Heliostat Number
15	DATE	15	• ASCII yymmdd = Date of bias update (if applicable)
20	NEW AZIMUTH BIAS	20	• ASCII \pm 999.99 New Heliostat Azimuth Bias
25	NEW ELEVATION BIAS	25	• ASCII \pm 999.00 New Heliostat Elevation Bias
30		30	
32		32	

TABLE 3.7-XXXVIII HAC-GDC Field Position Initialization

MESSAGE LAYOUT

APPLICATION : HAC-GDC

MESSAGE TYPE: Initialize field position
(1 of 2)

PROGRAMMER : D. Pettit

DATE : June 10, 1980

1	Blank	1	All ASCII
	Mac Dac Heliostat number		
5	Blank	5	Number composed of row number and row position number
	HFC-HC number		
10	Blank	10	HFC * 32+1+HC = (HFC-HC number) range 1-2048
	+/- x x x x x		
15	Blank	15	Site reference x-coordinate (East)
	+/- x x x x x		
20	Blank	20	Site reference y-coordinate (North)
	Blank		
25	Blank	25	Segment number assignment
	Segment Number		
30	Blank	30	
	Blank		

TABLE 3.7-XL HAC-GDC Segment Status

MESSAGE LAYOUT

APPLICATION: HAC-GDC MESSAGE TYPE : HAC Segment Status

PROGRAMMER: D. Pettit

DATE June 9, 1980

1	Header Byte	1	• 2 = Segment updating
	SEGMENT AIM #		• bit 0=1, bits 1-7 binary 2_{10}
	Pecking Order #		• bit 0=1, bits 1-7 binary value
	HC Status		• bit 0=1, bits 1-7 binary value
5	High order Az bits	5	• bit 0=1, bits 1-7 binary value
	low order Az bits		• bit 0=1, bits 1-7 are bits 0-6 of Az word
	High order El bits		
	low order El bits		• same format as azimuth
	AIM POINT		• bit 0=1, bits 1-7 are bits 4-10 of AP
10	(NORTH)	10	
	AIM POINT		• same format as North Aim Point
	(EAST)		
	AIM POINT		• same format as North Aim Point
	(UP)		
15	same format as above 12 bytes	15	• Repeat up to 21 total HCs in pecking order (less than 258 bytes)
	For the next HC in pecking order (shown in dashed line on left)		
20		20	
25		25	
30		30	

TABLE 3.7-XLI HAC-GDC Text Message to GDC

MESSAGE LAYOUT

APPLICATION: HAC-GDC

MESSAGE TYPE: Text message to GDC

PROGRAMMER: D. Pettit

DATE: June 10, 1980

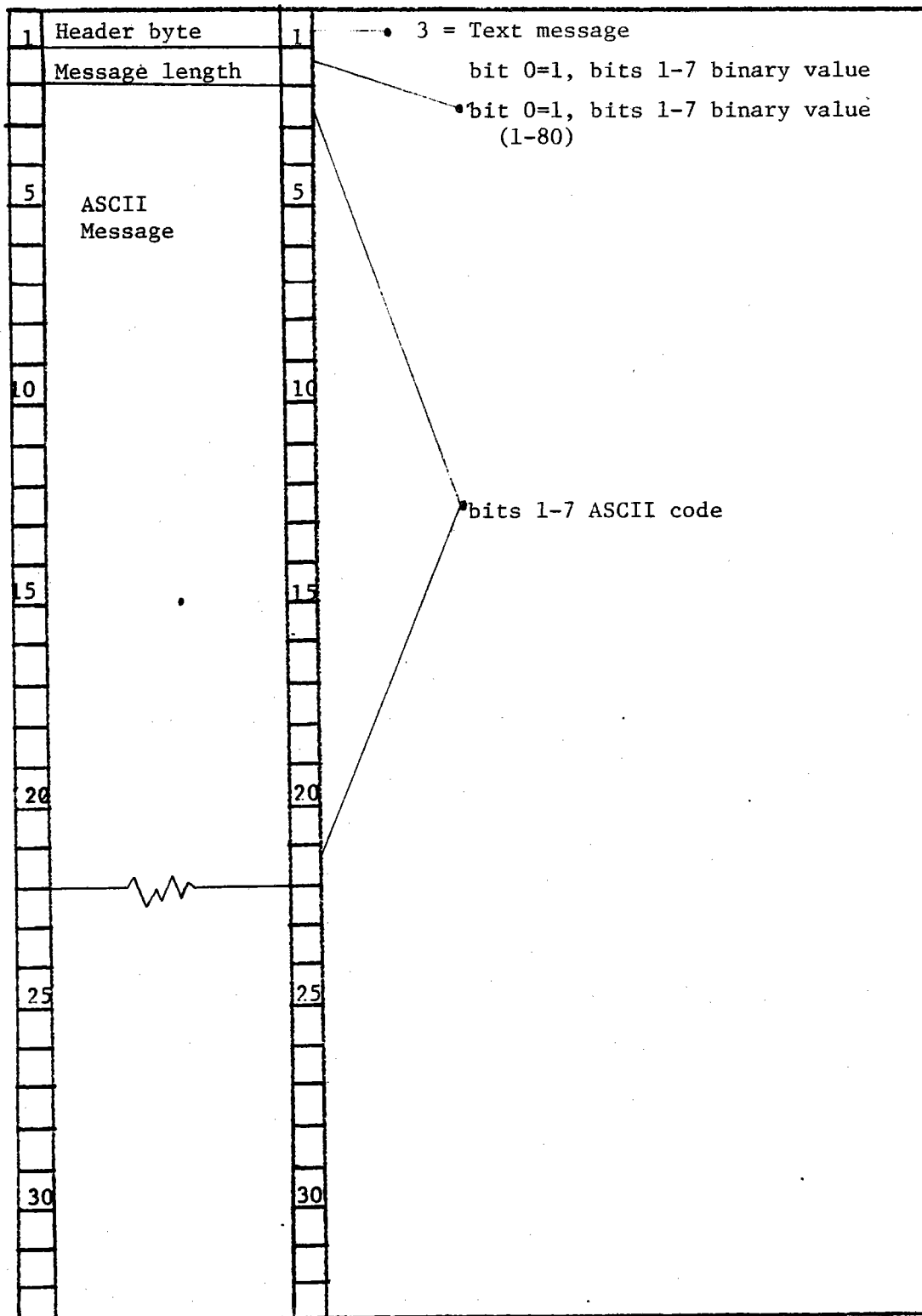


TABLE 3.7-XLII HAC-GDC Graphics Initialization

MESSAGE LAYOUT

APPLICATION : HAC-GDC

MESSAGE TYPE : Initialize Graphics

PROGRAMMER : D. Pettit

DATE : June 10, 1980

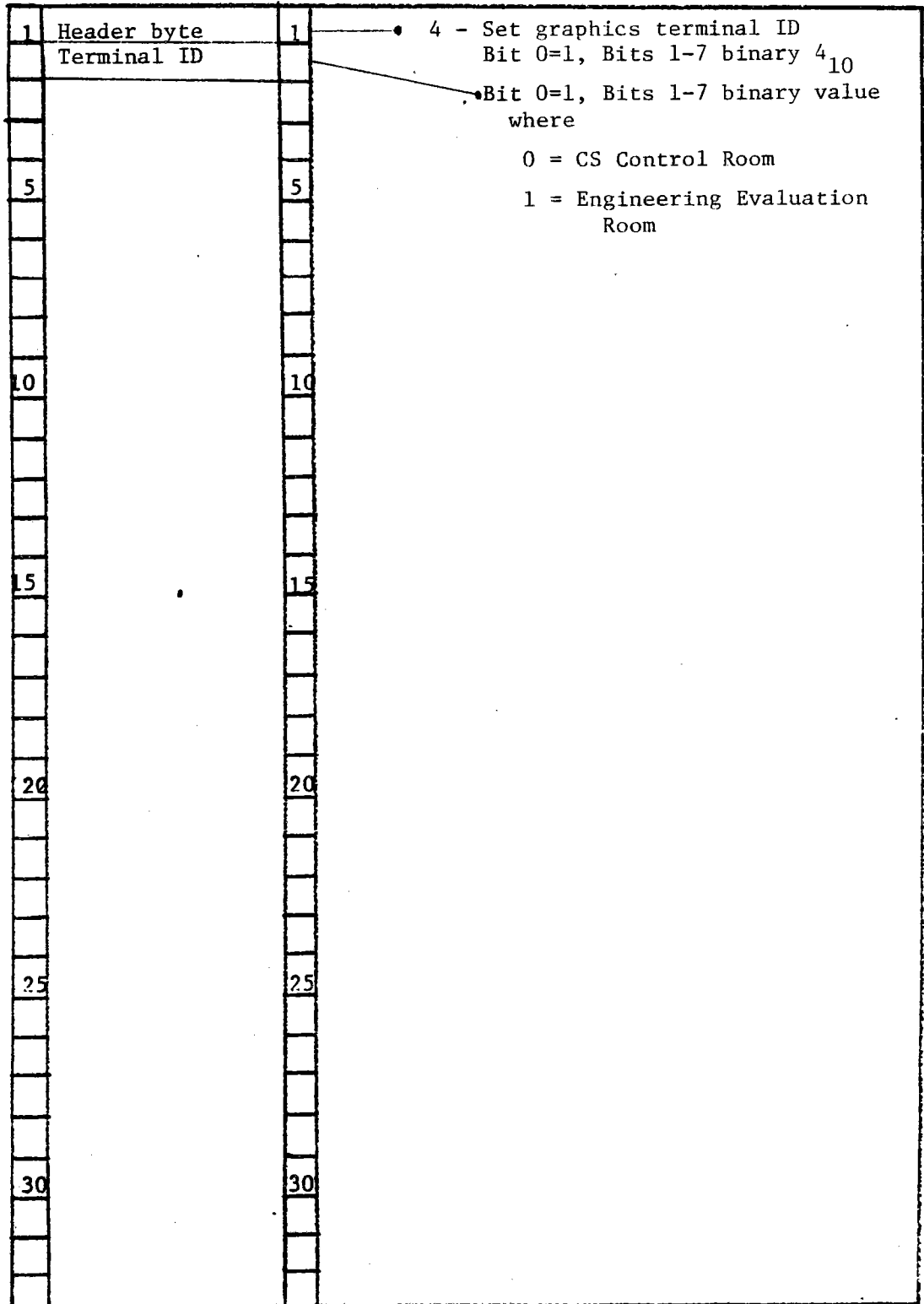


TABLE 3.7-XLIII GDC-HAC GDC Command

MESSAGE LAYOUT

APPLICATION : GDC - HAC

MESSAGE TYPE : GDC Command

PROGRAMMER : D. Pettit

DATE : June 9, 1980

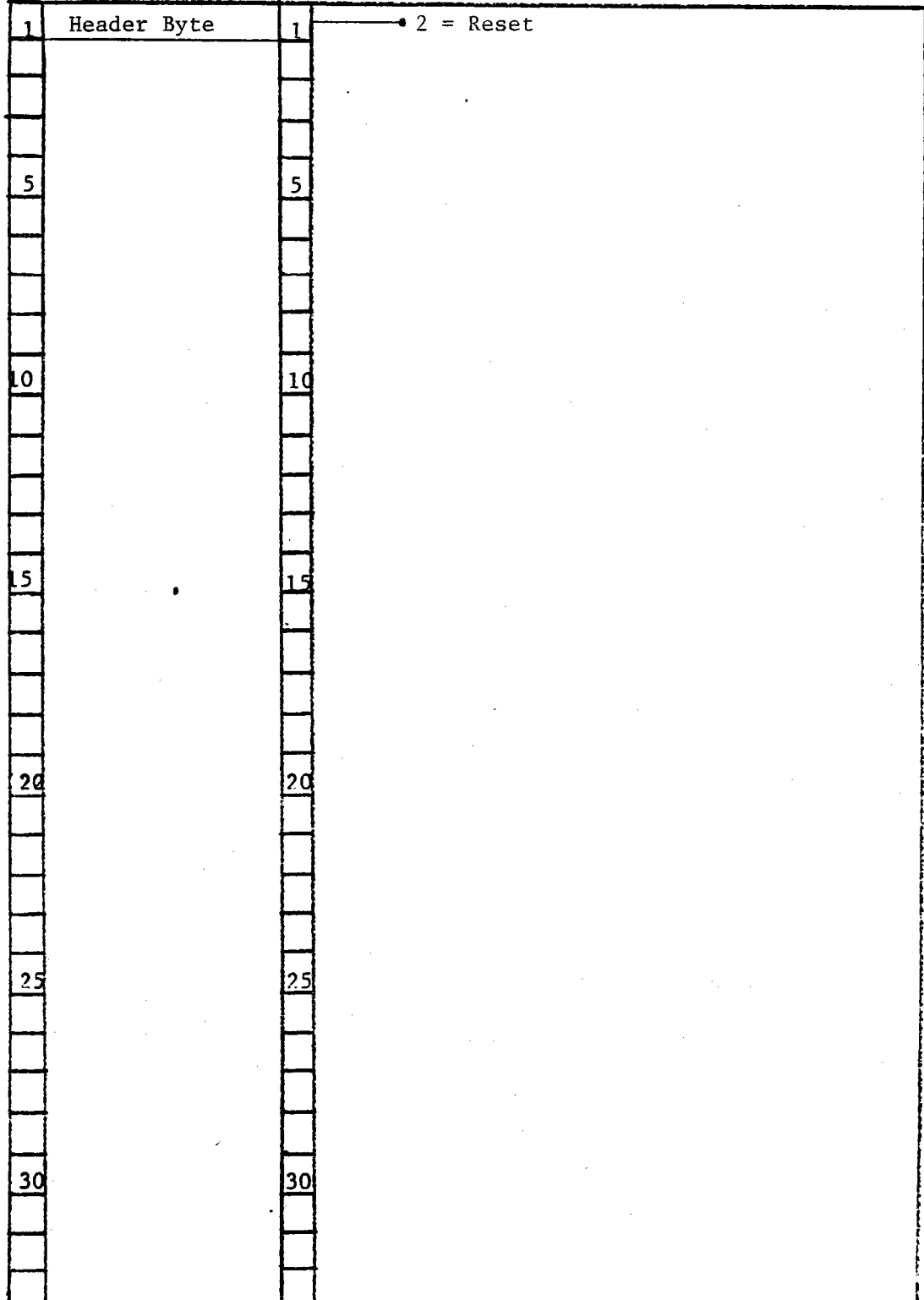


TABLE 3.7-XLIV GDC - HAC GDC Command

MESSAGE LAYOUT

APPLICATION: GDC - HAC

MESSAGE TYPE: GDC Command

PROGRAMMER: D. Pettit

DATE: June 9, 1980

1	Header Byte	1	<ul style="list-style-type: none"> • 3 = Full field or segment • ASCII Message byte count
5	Segment Number	5	<ul style="list-style-type: none"> • Position 5-7 contains segment number being displayed. Segment number 000 implies a full field display.
10		10	
15		15	
20		20	
25		25	
30		30	

TABLE 3.7-XLV GDC-HAC Command

MESSAGE LAYOUT

APPLICATION : GDC-HAC

MESSAGE TYPE : GDC Command

PROGRAMMER : D. Pettit

DATE : June 9, 1980

1	Header Byte	1	→ 4 = Send next message
5		5	
10		10	
15		15	
20		20	
25		25	
30		30	

TABLE 3.7-XLVI GDC-HAC GDC Command

MESSAGE LAYOUT

APPLICATION : GDC-HAC

MESSAGE TYPE : GDC Command

PROGRAMMER : D. Pettit

DATE : June 9, 1980

1	Header Byte	1	1 = High wind stow or defocus (ASCII)
			ASCII Message byte count = 14 ₁₀
5	Emergency Message	5	ASCII
			Position 5-14 contain either: G1OSTHI 0000 or, G1ODEFO 0000 .
10			
15		15	
20		20	
25		25	
30		30	

Receiver-to-HAC Trip Signal Logic

Computer/twisted pair *	Trip	Operational
HAC A - pair 1	< +2 VDC	+5 VDC
HAC A - pair 2	+5 VDC	< +2 VDC
HAC B - pair 1	< +2 VDC	+5 VDC
HAC B - pair 2	+5 VDC	< +2 VDC

* All signals TTL Level, 300 ma

TABLE 3.7-XLVII

4.0

SOFTWARE/FIRMWARE SYSTEM VALIDATION

The validation that the software/firmware developed for the Collector Subsystem meets the requirements imposed by the 10 MWe Collector Subsystem Software/Firmware Functional Requirements Specification and this design specification shall be accomplished by a tiered level of testing. Each element of a level of testing which leads to the next higher level of testing will be completed to the satisfaction of the responsible engineer for that level before commencing to the next higher level. At the completion of each level of testing, a memo will be generated by the responsible engineer for that level detailing the tests performed, the results of the test, and listing any special software or tools used in the test.

4.1

Test Phases

All software and firmware developed for the 10 MWe Collector Subsystem project will be subjected to four levels of testing:

- a. Function (or unit) testing;
- b. Integration testing;
- c. Breadboard level testing; and
- d. System level testing.

4.2

Functional Testing

Functional testing will be accomplished by the engineer responsible for the development of the software or firmware module, as each submodule becomes available. Each submodule will be tested against the Software/Firmware Design Specification. Testing at this level will be conducted informally and to the satisfaction of the responsible engineer. All hard-copy test results generated at this level will be filed in the Module Development Folder. Software modules developed for the HAC will be tested in the HAC computer. Firmware modules developed for the HFC and HC may be tested on the Motorola 6801 simulator running in the Sigma V computer system, the Motorola MEK6800D2 development kits, or breadboard models of the HFC, HC, or Stimulator.

4.3

Integration Testing

Integration testing will be accomplished under the cognizance of the HAC Lead Engineer for software to run in the HAC computer, and the HFC/HC Lead Engineer for firmware to operate in the HFC and HC microcomputers. The software and firmware modules will be integrated into the software system and firmware system, as the modules become available, and will be tested for compliance with interface definitions and functions, as defined in the Software/Firmware Design Specification and Software/Firmware Functional Requirements Specifications documents. Integration

testing will be informal and to the satisfaction of the HAC Lead Engineer and HFC/HC Lead Engineer. Hard-copy results and notes generated from this testing will be filed in a file kept in close proximity to the module development folders and will be used in preparation of the "as-built" Software/Firmware Design Specification.

4.4. Breadboard Integration Testing

Breadboard Integration testing will be accomplished under the cognizance of the Controls Manager. The software and firmware systems will be tested with the HAC computer connected to and communicating with a breadboard version of the HFC and HC microprocessors, with HFC and HC firmware systems operating, respectively. The software and firmware modules will be tested for compliance with the interface definitions, the software and firmware functions, and overall system performance. This testing will be informal and to the satisfaction of the Controls Manager. Any hard-copy results and notes generated from this testing will be filed in a Breadboard Test File and will be used in the preparation of the "as-built" Software/Firmware Design Specification. Upon completion of this testing, the software and firmware systems will be placed under configuration control.

4.5 System Level Testing

System Level testing will be accomplished under the cognizance of the Collector Subsystem Operations Manager. This level of testing will be accomplished under formal controls against the Collector Subsystem Functional Test Plan. The test will verify that the code satisfies the functional objectives and requirements specifications. Discrepancy reports will be filed for each failure detected. Any discrepancies will be corrected by the responsible engineers. Retesting will be performed to verify that the discrepancies have been corrected. System Level Test will be run at the Martin Marietta Plant in Denver, Colorado, by use of a heliostat located in the solar test area and using the HAC installed at the Space Physics Laboratory. After successful completion of the System Level Test in Denver, the HAC will be moved to the 10 MWe Solar One plant site and used for heliostat installation procedures. The HAC will undergo Acceptance Testing with the field as part of the Collector Subsystem Acceptance Test.

ACRONYMS

The following is a list of acronyms used with this project:

ASCII - American Standard Code for Information Interchange
AZ - Azimuth
BCS - Beam Characterization Subsystem
CLLP - Corridor Lower Limit Point
CPU - Central Processor Unit - CPU-CPU Link - 4824/4824 link
between Prime and Backup HAC

CRT - Cathode Ray Tube

CS - Collector Subsystem

CULP - Corridor Upper Limit Point

DAS - Data Acquisition System

DOE - Department of Energy

EL - Elevation

EPGS - Electrical Power Generation System

GFE - Government Furnished Equipment

HAC - Heliostat Array Controller

HC - Heliostat Controller

HFC - Heliostat Field Controller

ICD - Interface Control Document

I/F - Interface

I/O - Input/Output

ISC - Intelligent Systems Corporation

JCL - Job Control Language

MDAC - McDonnell Douglas Corporation

MAX IV - MODCOMP Operating System

MAXNET - MODCOMP Net Working System Superset of MAX IV

ACRONYMS (Cont'd)

MCS - Master Control System
MMC - Martin Marietta Corporation
MODCOMP - Modular Computer Corporation
O&M - Operation and Maintenance
OCS - Operational Control System
OPDD - Overall Plant Design Description
PCS - Peripheral Control Switch
QA - Quality Assurance
REX - Request Executive Services
RFP - Request for Proposal
RS - Receiver System
SCE - Southern California Edison
SDP - Software Development Plan
SED - Source Editor
SFDI - Solar Facility Design Integrator
SFRS - Software/Firmware Functional Requirements Specification
STMPO - Solar Ten Megawatt Project Office
S/W - Software
TBD - To be determined
TBS - To be supplied
TCB - Task Control Block
TI - Texas Instruments
TSS - Thermal Storage System
UFT - User File Table
UPS - Uninterruptable Power Supply
WWV - National Bureau of Standards Universal Time
broadcasting station at Fort Collins, CO

STMLPO-299

MARTIN MARIETTA AEROSPACE

DENVER AEROSPACE
POST OFFICE BOX 179
DENVER, COLORADO 80201
TELEPHONE (303) 977-3000

May 3, 1982

Refer to: DAC-82-389

To: U. S. Department of Energy
P. O. Box 808
Livermore, California 94550

Attn: Roger S. Gaither, Esq.
Assistant Chief for Prosecution
Office of Patent Counsel, L-376

Subj: Contract DE-ACO3-80SF10539

1. Attached is the Patent Certification on the subject contract.
2. If you have any questions, please contact me at (303) 977-6109.

Very truly yours,

MARTIN MARIETTA CORPORATION

Phillip L. DeArment
Phillip L. DeArment
Assistant Patent Counsel

PLD:jes

PATENT CERTIFICATION

DOE CONTRACT NO. DE-ACO3-80SF10539

1. The following is a complete list of technical reports prepared during the course of the work under this contract and the DOE office to which the reports were sent:

See Attachment I

2. Technical data of this contract other than reports (i.e., notebooks, drawings, etc.) are completely listed, as follows:

See Attachment II

3. Each of the above-listed documents under paragraphs 1 and 2 has been examined for invention subject matter by me and/or technical personnel under my direction; to the best of my knowledge and belief, no inventions or discoveries were made or conceived in the course of or under this contract other than the following:

<u>CONTRACTOR NO.</u>	<u>TITLE</u>	<u>DATE REPORTED</u>	<u>DOE NO.</u>
Docket # 80YD41	FASTON TERMINAL INSTALLATION TOOLS	03/09/81	S-55,956, RL-8354
Docket # 81YD19	FASTON TERMINAL EXTRACTION TOOLS	05/03/82	S-59,046, RL-8843

4. There were no subcontracts or purchase orders involving research and development, except as follows:

NONE

5. The completion date of this contract is as follows: February 13, 1982

6. The following period is covered by this certification:

December 3, 1979	to	February 13, 1982
Month Day Year		Month Day Year

Martin Marietta Corporation
Contractor Denver Aerospace
P. O. Box 179
Denver, Colorado 80201
Address

Phillip L. DeArment
Signature Phillip L. DeArment
Assistant Patent Counsel

Title

Submit in duplicate to:

Roger S. Gaither
Assistant Chief for Prosecution
California Patent Group, L-376
U. S. Department of Energy
P. O. Box 808
Livermore, California 94550

Form completed by: M. Frohardt
Date: Feb 23, 1982
M. Frohardt

Date of Certification

February 22, 1982
Attachment I
Patent Certification
DOE Contract No. DE-AC03-80SF10539

1. The following is a complete list of technical reports prepared during the course of the work under this contract and the DOE office to which the reports were sent:

<u>Report Type</u>	<u>Office Mailed</u>
Monthly Submittals	Huntington Beach
Technical Status Report Issue 1 through Issue 25	Huntington Beach
Design and Manufacturing Drawings	Huntington Beach and Canoga Park
Technical Correspondence	Huntington Beach and Canoga Park
Contract Correspondence	Oakland, CA.
Deliverables	Huntington Beach and Canoga Park
Periodic Reports	Huntington Beach and Canoga Park

2. Technical data of this contract other than reports (i.e., notebooks, drawings, etc.) are completely listed, as follows:

All correspondence, reports, etc., mailed to Huntington Beach office.

Monthly submittals, Issue 1 through 25 as follows:

- a. Milestone Schedule and Status Report
- b. Cost Management Report
- c. Project Status Report
- d. Documentation Tab Run
- e. List of active changes/modifications
- f. List of drawing revisions/levels
- g. Indentured Parts List

Minority Business Reports

Updated OPDD Documentation

Controls Hardware Drawings

Software Design Specification

Safety Plan

Hazard Analysis

Preliminary Design Review Package

Preliminary Design Review Package Final

Manufacturing Plan

Quality Assurance Plan

Functional Test Plan

Functional Test Report

C/S Integrated Acceptance Test Plan

February 22, 1982
Attachment II
Patent Certification
DOE Contract No. -DE-AC03-80SF10539

Page Two

C/S Integrated Test Report

Supplemental Spares Plan

Two Copies of Drawings with Latest Revisions

Operational and Maintenance Manuals

Collector Subsystem Instrumentation

Maintenance Instructions

Control System Theory of Operations

Software/Firmware Design Specifications

STMP10-249

MARTIN MARIETTA AEROSPACE

DENVER DIVISION
POST OFFICE BOX 179
DENVER, COLORADO 80201
TELEPHONE (303) 977-3000

10 November 1982

Ms. Mary Jane Holliday
Contract Examiner
Department of Energy
California Patent Group
San Francisco Operations Office
Oakland, California 94612

Re: Final Patent Certification for DOE
Contract DE-AC03-80SF10539

Dear Ms. Holliday:

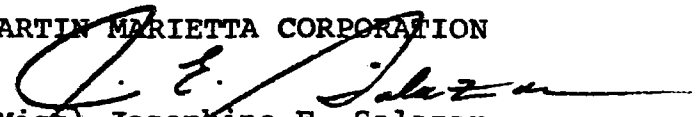
Pursuant to your letter of September 7, 1982, addressed to Mr. Cecil W. Duclon, I have made the corrections you requested thereon.

With respect to Invention Disclosures 81YD16, 81YD24 and 81YD25, please see our attached letters dated March 11, 1982 wherein you were notified that these inventions are not reportable. I am also attaching Mr. Carnahan's letter dated March 22, 1982.

If we can be of further assistance, please contact our office. Mr. DeArment may be reached at (303) 977-6109 and I may be reached at (303) 977-6501/6110.

Very truly yours,

MARTIN MARIETTA CORPORATION



(Miss) Josephine E. Salazar
Assistant to Phillip L. DeArment
Assistant Patent Counsel

cc: C. W. Duclon
W. A. Brever
J. T. Weber



STMPU-289

Department of Energy
San Francisco Operations Office
1333 Broadway
Oakland, California 94612

Reply to:
DOE Site Office
P.O. Box 366
Daggett, CA 92327
ATTN: S. D. Elliott, Jr.

Melvin W. Frohardt
Martin Marietta Aerospace
P.O. Box 179
Denver, CO 80201

JUN 25 1983

Subj.: Request for patent clearance and TIC Distribution of Documents from
DOE Contracts ET21007 and SF10539 (Solar One Heliostats, Phases I & II)

Dear Mel:

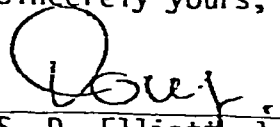
We are about to come out, with the help of EPRI, with a bibliography of key Project documents. To cope with anticipated requests for copies, I would like to arrange for properly cleared documents to be filed with and distributed through the DOE Technical Information Center at Oak Ridge. A check with TIC shows that only MCR-80-1377 has been cleared by them to date. Can you provide me with signed-off Patent Clearance Requests for:

- o The five indicated documents from the Phase I study (ET21007);
- o The twelve indicated documents from Phase II (SF10539);
- o The as-built drawing set provided via Sandia at the end of Phase II;
- o Any other Project documents generated by MMC you think the utility/industry community should have ?

I'd also appreciate a check on the Phase I CDR handout; was it MCR-78-1325?

Your help is greatly appreciated; it will save me (and you) a lot of running about once the bibliography comes out. I will insure that you get a copy; it lists about 500 documents, not including drawings (these we will provide to TIC in aperture card form at a later date, with a full index). Please call me ((619) 254-2672/-2142) if you have any questions or concerns.

Sincerely yours,


S. D. Elliott, Jr.
DOE Project Director



STYPO-299

Department of Energy
San Francisco Operations Office
1333 Broadway
Oakland, California 94612

Reply to:
DOE Site Office
Post Office Box 366
Daggett, CA 92327

Mr. Melvin T. Frohardt
Martin Marietta Aerospace
Post Office Box 179
Denver, CO 80201

DEC 06 1983

Subj.: Closeout Actions on Martin Marietta Contracts with DOE San Francisco Operations Office

Dear Mel:

Nearly six months ago, I requested your assistance in finalizing patent clearance on a number of the documents from the Collector Phase I and Phase II contracts which we wish to enter into the DOE Technical Information Center system. Thus far, I have not had any response to this request. We are about to issue the bibliography developed by Burns & McDonnell under the EPRI-funded "Lessons Learned and Project Documentation" study (I assume you have received a copy of Vol. 1, "Lessons Learned" - if not, let me know and I will send you one), and we and TIC anticipate a substantial number of requests for key documents, including yours.

In addition, SAN Contracts Closeout (Sonia Jackson) advises me that several of the final documents needed to complete closeout (and release final payment of withheld funds), ~~is~~ as yet lacking, not only on the above two contracts, but also on the old Preliminary Design contract. I would greatly appreciate your assistance (or your guidance as to who can assist us) in getting this wrapped up and off both of our desks. To recapitulate (adding the items needed by SAN) for the three contracts:

DE-AC03-76ET20422 (Old Contract -1110), Central Receiver System Prel. Design:

- o A "Final Invoice", to be submitted to Sonia Jackson, with copy to me;
- o "Contractors Assignment of Refunds and Rebates", to Sonia;
- o "Contractors Release", to Sonia;
- o "Contractor Request for Patent Clearance" (send to me, only), for:
 - MCR-77-161, "System Safety Design Criteria for Central Receiver...System",
 - MCR-77-162, "System Safety Program Requirements for Solar Thermal Systems".(These were done under an extension to the Preliminary Design contract, and are valuable background documents.)

DE-AC03-78ET21007 Collector System, Phase I:

- o "Final Invoice", to Sonia, copy to me;
- o "Assignment of Funds and Rebates", to Sonia;

- o "Contractors Release", to Sonia;
- o "Contractor Request for Patent Clearance", to me, for:
 - MCR-78-1323, "10-MWe Solar Thermal Pilot Plant Conceptual Design Review";
 - MTR-78-1330, "10-MWe Solar Thermal Pilot Plant Preliminary Design Review";
 - MCR-79-1302, "10-MWe Solar Thermal Pilot Plant Final Design Review (2 Vols.)";
 - 40-0-500-4P, "10-MWe Solar Thermal Pilot Plant Phase II O&M Equipment";
 - 40-0-500-6P, "10-MWe Solar Thermal Pilot Plant Phase II Planning."

DE-AC03-80SF10539, Collector System Phase II

- o "Final Invoice", to Sonia, copy to me;
- o "Assignment of Funds and Rebates", to Sonia;
- o "Contractors Release", to Sonia;
- o "Contractor Request for Patent Clearance", to me, for:
 - MCR-79-1352B; "Quality Assurance Plan for 10-MWe Phase II Collector..";
 - MCR-80-1304, "10-MWe Solar Pilot Plant Collector Subsystem Safety Plan";
 - MCR-81-1331B, "Hazard Analysis for 10-MWe ...Pilot Plant";
 - 40-0-500-2P, "10-MWe ...Pilot Plant Phase II Mfg. Plan, Rev. 2";
 - MCR-80-1341A, "10-MWe Collector Sybsystem Software/Firmware Functional Req'ts.";
 - MCR-80-1362, "System Description Document, Collector Subsystem...";
 - MCR-80-1376; "Heliostat Stimulator Operators' Manual";
 - MCR-81-1708, "Operation Instructions, Heliostat Field Subsystem...";
 - MCR-81-1709A, "Maintenance Instructions, Heliostat Field Subsystem...";
 - MTR-81-1769, "...Collector Subsystem Functional Test Report";
 - MCR-81-1770, "Supplemental Spares Plan, Heliostat Field...";
 - MCR-80-1377A, "Software/Firmware Design Specifications...";
 - MCR-82-1701, "Control System Theory of Operation";
 - Drawing Set, as Identified in "Drawing Tree 400500 5132701";
 - Source Listing of Code for Heliostat Controller ROM or EPROM*
 - Source Listing of Code for Heliostat Field Controller ROM/EPROM*

Our files do not have current copies of the following other items identified in the Drawing Tree (400500 5132701):

Documents: 40M500-2S, "Foundation Req'ts.", 40M500-1T, "Installation Instructions", 40M500-2M, "Canting Procedures", 40M500-5P, "Acceptance Plan", MCR-80-1361, "Collector System Functional Test Plan", and MCR-81-1715, "Collector System Integrated Acceptance Test Plan."

* Current copies of these four items are lacking from the Project files; your assistance in obtaining at least one copy of each will be most appreciated.

Drawings: 40E500 5132788, "Adapter Plate/Control Arm Heat Tool", 40E500 5132771, "Field Canting Tool", and 40E500 5132776, "Drive Unit Checkout Console".

While these items are not carried in the current version of the Bibliography (none of the Plant as-built drawings have been entered as yet), many, if not all, of them may be expected to be of interest to the solar community. I would appreciate at least one copy of each, again with your release. To save you considerable effort in preparing the Patent Clearance Request forms (I am enclosing several copies of the form), you may combine many of the above by simply clearing the "Drawing Tree", with its contents.

If you need the other closeout forms cited above (your Contract Administration staff should have them in stock), please call Sonia Jackson at FT5 536-4179, or write her at:

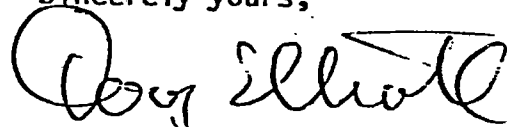
Ms. Sonia Jackson (CM)
 Department of Energy
 1333 Broadway
 Oakland, CA 94612

Finally, since we are required to forward two clean, reproducible copies of each document to DOE/TIC, as well as needing one clean copy for our on-site archives, any "extras" you can turn up around your offices would be greatly appreciated; certainly, rather than throw anything of possible interest out, send it to me.

Mel, I know (believe me!) that this is all a significant amount of work, and I wish I didn't have to ask you (or your staff) to go through it, but it will be to our mutual benefit in the end to get these three contracts all cleaned up, and a comprehensive package of Project documentation (currently, over 550 documents, plus drawings) into the archives. If there is anything further I can do to assist you in this effort, please call on me.

Encl.: DOE Proj. Ofc. ltr. 6/25/83
 Patent Clearance Req. Forms

Sincerely yours,



S. D. Elliott, Jr., Director,
 DOE Project Office, Barstow

cc: H. C. Wroton, MMC
 Sonia Jackson, DOE/SAN (CM)

PS: I keep running across references to a document I can't identify: MCR-78-1325; what was it?

STMDO-299

MARTIN MARIETTA AEROSPACE

DENVER DIVISION
POST OFFICE BOX 179
DENVER, COLORADO 80201
TELEPHONE (303) 877-3000

January 30, 1984

Mr. Doug Elliott
DOE Site Office
Post Office Box 366
Daggett, CA 92327

Subject: Closeout Actions on Martin Marietta Contracts with DOE San Francisco Operations Office

Reference: Letter of December 06, 1983, S.D. Elliott, Jr. to M. Frohardt, Closeout of Contracts

In regard to the referenced letter, following is the status and actions in process to close out these items:

1. Contract Closeout Status

In regard to the closeout of cost type contracts DE-AC03-76ET20422, Central Receiver Test Facility, and DE-AC03-78ET21007, Collector System Phase I, we include the "Contractors Assignment of Refunds and Rebates" and "Contractors Release" with our final invoice package. The final invoices for these two contracts will be submitted upon completion of final settlement negotiations for our 1979 overhead and G&A rates which is currently in progress. In reference to the closeout of contract DE-AC03-80SF10539, Collector System Phase II, please see Attachment 1, the letter to Ms. Joann Littlehales dated January 23, 1984, for the current status.

2. Patent Clearance

The following documents are in the process of being cleared by our Patent office. When this transmittal is available, I will send a copy to you.

- MCR-78-1323, "10-MWe Solar Thermal Pilot Plant Conceptual Design Review"
- MCR-78-1330, "10-MWe Solar Thermal Pilot Plant Preliminary Design Review"
- MCR-79-1302, "10-MWe Solar Thermal Pilot Plant Final Design Review (2 Vols)"
- 40-0-500-4P, "10-MWe Solar Thermal Pilot Plant Phase II O&M Equipment"
- 40-0-500-6P, "10-MWe Solar Thermal Pilot Plant Phase II Planning"

Mr. Doug. Elliott
January 27, 1984
Page 2

The remainder of the documents have been previously cleared by the following letters, copies of which are included in Attachment 2.

Letters from Phillip DeArment to Roger Gaither:

DAC-83-417, dated May 24, 1983
80-Y-15555, dated July 28, 1980
DAC-82-389, dated May 3, 1982
Letter dated March 11, 1982
Letter dated November 10, 1982

3. Documents

You requested copies of some documents and drawings in the referenced letter. Copies of the following drawings and documents are being submitted under Attachment 3.


MCR-78-1330, "Preliminary Design Review Package"
MCR-79-1352B, "Quality Assurance Plan for 10-MWe Phase II
Collector"
MCR-80-1376, "Heliostat Stimulator Operators' Manual"
40M500-2S, "Foundation Requirements"
40M500-2M, "Canting Procedures"
40M500-1T, "Installation Instructions"
40M500-5P, "Acceptance Plan"
MCR-81-1715, "Collector System Integrated Acceptance Test Plan"
MCR-80-1361, "Collector System Functional Test Plan"
40M500 5132788, "Adapter Plate/Control Arm Heat Tool"
40M500 5132771, "Field Canting Tool"
Source Listing of Code for Heliostat Controller ROM or EPROM*
Source Listing of Code for Heliostat Field Controller ROM/EPROM

No drawing exists for 40E500 5132776, "Drive Unit Checkout Console" as this checkout console consisted of a stimulator to operate a production Drive Mechanism Assembly. Also MCR-78-1330 is the correct document number for the Preliminary Design Review Package rather than MCR-78-1325. MCR-78-1325 is the document number assigned to all the Monthly Progress Reports written during the Phase I contract.

Doug, I hope this will help in getting the documentation finalized. I will follow-up with the additional information identified. If you have any questions please call on me.

Sincerely yours,

MARTIN MARIETTA CORPORATION


Melvin W. Frohardt
Solar Programs

Enclosures

cc: H. Wroton
Sonia Jackson

STMP0-299

U.S. DEPARTMENT OF ENERGY

memorandum

DATE: MAY 14 1984

REPLY TO
ATTN OF

S. D. Elliott, Jr., Director, DOE Solar One Project Office

SUBJECT

Submission of Thirteen Reports Prepared for 10-MWe Pilot Plant ("Solar One") Project by Martin Marietta Corporation under Contract DE-AC03-80SF10539

TO

Roger S. Gaither, DOE/SAN Office of Patent Counsel
William D. Matheny, DOE/TIC Document Control


Enclosed are thirteen documents prepared by the Martin Marietta Corporation, Denver Aerospace Division, for the Solar Ten-Megawatt Project Office in conjunction with design and fabrication of the Pilot Plant Collector (Heliostat) System, under Contract DE-AC03-80SF10539:

<u>Primary Document No.</u>	<u>Secondary No.</u>	<u>Brief Title</u>
DOE/SF/10539-01	(STMP0-288)	"Quality Assurance Plan..."
DOE/SF/10539-02	(STMP0-289)	"...System Safety Plan"
DOE/SF/10539-03	(STMP0-290)	"Hazard Analysis..."
* DOE/SF/10539-04	(STMP0-291)	"Phase II Manufacturing Plan (Revision 2)"
DOE/SF/10539-05	(STMP0-292)	"Software/Firmware Functional Requirements..."
* DOE/SF/10539-06	(STMP0-293)	"System Description Document..."
DOE/SF/10539-07	(STMP0-294)	"Heliostat Stimulator Operator's Manual"
DOE/SF/10539-08	(STMP0-295)	"Operations Instructions, Heliostat Field..."
* DOE/SF/10539-09	(STMP0-296)	"Maintenance Instructions, Heliostat Field..."
DOE/SF/10539-10	(STMP0-297)	"...Functional Test Report"
DOE/SF/10539-11	(STMP0-298)	"Supplemental Spares Plan..."
* DOE/SF/10539-12	(STMP0-299)	"Software/Firmware Design Specification..."
DOE/SF/10539-13	(STMP0-300)	"Control System Theory of Operation"

One copy of each document, accompanied by a SAN Form 70 prepared by the Project Office (on the basis of Attachment 1, the Contractor's Patent Certification as submitted May 3, 1982), is provided for SAN/OPC review and clearance. The fabrication and maintenance materials indicated in the above list by an asterisk should be reviewed in the light of the two disclosures filed with Atch. 1, as well as the Martin Marietta Dockets 81YD16, -24, and -25, claimed as developed outside the scope of the contract in Atch. 2, MMC letter of November 10, 1982. Please return the "feedback" copies of the Form 70's to this office; the clearance copies of the documents themselves may be returned to Mr. Mike Lopez, SAN/FGS.

Two copies of each document, accompanied by a completed DOE Form RA-426, are submitted for archiving and announcement by the DOE Technical Information Center and for forwarding to the National Technical Information Service.

- Attchs.: 1. Martin Marietta Ltr. 5/3/82
- 2. Martin Marietta ltr. 11/10/82



S. D. Elliott, Jr., Director,
DOE Solar One Project Office

Encls.: 13 Documents w/transmittal forms

- cc: Mike Lopez, DOE/SAN (FGS)
- Don Holz, DOE/SAN (ISEA)
- Mary Soderstrum, Burns & McDonnell



DEPARTMENT OF ENERGY
SAN FRANCISCO OPERATIONS OFFICE

CONTRACTOR REQUEST FOR PATENT CLEARANCE
FOR RELEASE OF UNCLASSIFIED DOCUMENT

Prime Contract No. DE-AC03-80SF10539
Subcontract No. (N/A)
Report No. DOE/SF/10539-12 (STMP0-299)
Date of Report January 1982
Name & Phone No. of DOE Technical Representative S. D. Elliott, Jr. (619) 254-2672

TO: Roger S. Gaither, Asst. Chief for Prosecution
Office of Patent Counsel/Livermore Office
P.O. Box 808, L-376
Livermore, California 94550

FROM: DOE Solar One Project Office
Post Office Box 336
Daggett, CA 92327

- Document Title: "Software/Firmware Design Specification for 10 MWe Solar Thermal Central Receiver Pilot Plant"
- Type of Document: Technical Report, Conference Paper, Journal Article, Abstract or Summary, Copy of Oral Presentation, Other (please specify): _____
- In order to meet a publication schedule or submission deadline, patent clearance by _____ (routine) would be desired.

SENDER IS TO CHECK BOX #4 OR #5 BELOW.

4. I have reviewed (or have had reviewed by technically knowledgeable personnel) this document for possible inventive subject matter (Subject Inventions) and that no inventions or discoveries (Subject Inventions) are deemed to be disclosed in this document except as stated below:

a. Attention should be directed to pages _____ of this document.

b. This document describes matter relating to an invention:

- Contractor Invention Docket No. 80YD41, 81YD19
- A disclosure of the invention was submitted to DOE on 3/9/81, 5/3/82 (date)
- A disclosure of the invention will be submitted shortly _____ (approximate date)
- A waiver of DOE's patent rights to the contractor:
 - has been granted, has been applied for; or will be applied for _____ (date)

5. This document is being submitted, but no review has been made of this document for possible inventive subject matter.

6. Remarks: See Martin Marietta ltr. 5/3/82 for Patent Certification and disclosure data

Reviewing/Submitting Official: Name (Print/Type) S. D. Elliott, Jr., Director
 Title DOE Solar One Project Office
 Signature *S. D. Elliott* Date 12 May, 1984

TO: INITIATOR OF REQUEST

FROM: ASSISTANT CHIEF FOR PROSECUTION
Office of Patent Counsel/Livermore Office

- No patent objection to above-identified release.
- Please defer release until advised by this office.

Signed _____ Date Mailed _____

DOE AND MAJOR CONTRACTOR RECOMMENDATIONS FOR
ANNOUNCEMENT AND DISTRIBUTION OF DOCUMENTS

See Instructions on Reverse Side

1. DOE Report No. DOE/SF/10539-12 (STMP0-299)	2. Contract No. DE-AC03-80SF10539	3. Subject Category No. UC-62
--	--------------------------------------	----------------------------------

4. Title
"SOFTWARE/FIRMWARE DESIGN SPECIFICATION FOR 10 MWe SOLAR THERMAL CENTRAL RECEI-

5. Type of Document ("x" one)
 a. Scientific and technical report
 b. Conference paper: Title of conference _____

VER PILOT PLANT"

_____ Date of conference _____

Exact location of conference _____ Sponsoring organization _____
 c. Other (specify planning, educational, impact, market, social, economic, thesis, translations, journal article manuscript, etc.)

6. Copies Transmitted ("x" one or more)
 a. Copies being transmitted for standard distribution by DOE-TIC.
 b. Copies being transmitted for special distribution per attached complete address list.
 c. Two completely legible, reproducible copies being transmitted to DOE-TIC. (Classified documents, see instructions)
 d. Twenty-seven copies being transmitted to DOE-TIC for TIC processing and NTIS sales.

best avail-
able copies

7. Recommended Distribution ("x" one)
 a. Normal handling (after patent clearance): no restraints on distribution except as may be required by the security classification.
 b. To U.S. Government agencies and their contractors.
 c. within DOE and to DOE contractors.
 d. within DOE.
 f. Other (Specify) Archive/issue on request

8. Recommended Announcement ("x" one)
 a. Normal procedure may be followed. b. Recommend the following announcement limitations:

9. Reason for Restrictions Recommended in 7 or 8 above.
 a. Preliminary information. b. Prepared primarily for internal use. c. Other (Explain)

10. Patent, Copyright and Proprietary Information
 Does this information product disclose any new equipment, process or material? No Yes If so, identify page nos. _____
 Has an invention disclosure been submitted to DOE covering any aspect of this information product? No Yes (see 13, below)
 If so, identify the DOE (or other) disclosure number and to whom the disclosure was submitted.
 Are there any patent-related objections to the release of this information product? No Yes If so, state these objections.
 Does this information product contain copyrighted material? No Yes
 If so, identify the page number _____ and attach the license or other authority for the government to reproduce.
 Does this information product contain proprietary information? No Yes If so, identify the page numbers _____
 ("x" one) a. DOE patent clearance has been granted by responsible DOE patent group.
 b. Document has been sent to responsible DOE patent group for clearance.

11. National Security Information (For classified document only; "x" one)
 Document a. does b. does not contain national security information

12. Copy Reproduction and Distribution
 Total number of copies reproduced 15 Number of copies distributed outside originating organization 5

13. Additional Information or Remarks (Continue on separate sheet, if necessary)
 Disclosures S-55, 956, RL-8354 and S-59, 046, RL-8843, submitted to SAN/OPC (MMC ltr. 5/3/82). SAN/OPC will review for patent issues.

14. Submitted by (Name and Position) (Please print or type)
 S. D. Elliott, Jr., Director, DOE Solar One Project Office

Organization
 Post Office Box 366, Daggett, CA 92327 (619) 254-2672

Signature S. D. Elliott Date MAY 12 1984



DEPARTMENT OF ENERGY
SAN FRANCISCO OPERATIONS OFFICE

CONTRACTOR REQUEST FOR PATENT CLEARANCE
FOR RELEASE OF UNCLASSIFIED DOCUMENT

Roger S. Gaither, Asst. Chief for Prosecution
Office of Patent Counsel/Livermore Office
P.O. Box 808, L-376
Livermore, California 94550

Prime Contract No. DE-AC03-80SF10539
Subcontract No. (N/A)
Report No. DOE/SF/10539-12 (STMP0-299)
Date of Report January 1982
Name & Phone No. of DOE Technical Representative S. D. Elliott, Jr. (619) 254-2672

FROM: **DOE Solar One Project Office**
Post Office Box 336
Daggett, CA 92327

- Document Title: **"Software/Firmware Design Specification for 10 MWe Solar Thermal Central Receiver Pilot Plant"**
- Type of Document: Technical Report, Conference Paper, Journal Article, Abstract or Summary, Copy of Oral Presentation, Other (please specify): _____
- In order to meet a publication schedule or submission deadline, patent clearance by _____ (routine) would be desired.

SENDER IS TO CHECK BOX #4 OR #5 BELOW.

- I have reviewed (or have had reviewed by technically knowledgeable personnel) this document for possible inventive subject matter (Subject Inventions) and that no inventions or discoveries (Subject Inventions) are deemed to be disclosed in this document except as stated below:
 - Attention should be directed to pages _____ of this document.
 - This document describes matter relating to an invention:
 - Contractor Invention Docket No. **80YD41, 81YD19**
 - A disclosure of the invention was submitted to DOE on **3/9/81, 5/3/82** (date)
 - A disclosure of the invention will be submitted shortly _____ (approximate date)
 - A waiver of DOE's patent rights to the contractor:
 has been granted, has been applied for; or will be applied for _____ (date) by me
 - This document is being submitted, but no review has been made of this document for possible inventive subject matter.
 - Remarks: **See Martin Marietta ltr. 5/3/82 for Patent Certification and disclosure data**
- Reviewing/Submitting Official: Name (Print/Type) **S. D. Elliott, Jr., Director**
 Title **DOE Solar One Project Office**
 Signature *[Signature]* Date **12 May, 1984**

TO: **INITIATOR OF REQUEST**
FROM: **ASSISTANT CHIEF FOR PROSECUTION**
Office of Patent Counsel/Livermore Office

- No patent objection to above-identified release.
- Please defer release until advised by this office.

M. Lopez, SAN
JEC
5/21/84
5/21/84

Signed *[Signature]* Date Mailed **5/21/84**