



VORTEX II

Operating System

Programmer Reference

Mini-Computer Operations
2722 Michelson Drive
P.O. Box C-19504
Irvine, California 92713

98A 9952 246





VORTEX II OPERATING SYSTEM

Programmer Reference Manual

UP-8677 Rev. 2
98A 9952 248

JUNE 1980

UPDATE B

The statements in this publication are not intended to create any warranty, express or implied. Equipment specifications and performance characteristics stated herein may be changed at any time without notice. Address comments regarding this document to Sperry Univac, Mini-Computer Operations, Publications Department, 2722 Michelson Drive, P.O. Box C-19504, Irvine, California, 92713.

COPYRIGHT ©1979, 1980 by
SPERRY RAND CORPORATION
ALL RIGHTS RESERVED

Sperry Univac is a division of Sperry Corporation

Printed in U.S.A.



PAGE STATUS SUMMARY

ISSUE: Update B - UP-8677 Rev. 2
(98A 9952 248)

Part/Section	Page Number	Update Level	Part/Section	Page Number	Update Level	Part/Section	Page Number	Update Level
Cover				16a	B*	10	1 thru 6	Orig.
Title				17	B	11	1 thru 7	Orig.
PSS	1, 2	B		18 thru 24	Orig.	12	1 thru 18	Orig.
CR	1	B				13	1 thru 4	Orig.
Contents	1 thru 10	B	4	1	Orig.		5 thru 6	B
1	1	Orig.		2 thru 3	B		7 thru 10	Orig.
	2	B		4 thru 9	Orig.		11	B
	3	Orig.		10 thru 11	B	14	1 thru 8	Orig.
	4	B	5	1 thru 23	Orig.		9 thru 10	B
	5 thru 6	Orig.	6	1 thru 7	Orig.		11 thru 16	Orig.
2	1 thru 11	Orig.		8	B		17 thru 19	B
	12 thru 15	B	7	1	B		20 thru 32	Orig.
	16	Orig.		2 thru 7	Orig.		33	B
3	1 thru 2	B		8	A		34 thru 35	Orig.
	2a	B*		9 thru 11	B		36	B
	3	B		12	Orig.		37 thru 53	Orig.
	4	Orig.	8	1	Orig.		54 thru 55	B
	5 thru 7	B		2	B			
	8 thru 13	Orig.		3 thru 8	Orig.			
	14 thru 16	B	9	1 thru 4	Orig.			
				5 thru 8	B	15	1	B

***New pages**

All the technical changes are denoted by an arrow (→) in the margin. A downward pointing arrow (↓) next to a line indicates that technical changes begin at this line and continue until an upward pointing arrow (↑) is found. A horizontal arrow (→) pointing to a line indicates a technical change in only that line. A horizontal arrow located between two consecutive lines indicates technical changes in both lines or deletions.

PAGE STATUS SUMMARY

ISSUE: Update 8 - UP-8677 Rev. 1
(98A 9952 248)

Part/Section	Page Number	Update Level	Part/Section	Page Number	Update Level	Part/Section	Page Number	Update Level
16	2	Orig.	21	1 thru 2	8		23	8
	3 thru 4	8		3 thru 6	Orig.		24 thru 27	Orig.
	5 thru 7	Orig.		7	8		28	8
	8 thru 10	8		8	8*		29	Orig.
	1 thru 2	Orig.		22	1 thru 10		Orig.	30
17	3	8		11	8	Appendix B	1 thru 2	Orig.
	4 thru 7	Orig.	23	1 thru 2	Orig.	Appendix C	1 thru 4	Orig.
	1	Orig.		3 thru 8	8	Appendix D	1 thru 2	Orig.
	2	8	24	1 thru 14	Orig.	Appendix E	1	Orig.
18	3 thru 7	Orig.	25	1 thru 4	8	Appendix F	1	Orig.
	1	Orig.	Appendix A	1	8	2	Orig. A	
	2	A		2 thru 3	Orig.	3	Orig.	
	3 thru 4	Orig.		4 thru 6	8	Appendix G	1 thru 6	Orig.
19	5	8		6a	8*	Appendix H	1 thru 5	Orig.
	1 thru 11	8		7 thru 8	8	6	8	
	12 thru 14	8*		9 thru 13	Orig.	7	8*	
20	1	Orig.		14	8	Appendix I	1 thru 5	8*
	2	8		15 thru 17	Orig.	Index	1 thru 7	8
	3 thru 13	Orig.		18	8			
	14 thru 15	8		19 thru 22	Orig.			

*New pages

All the technical changes are denoted by an arrow (→) in the margin. A downward pointing arrow (↓) next to a line indicates that technical changes begin at this line and continue until an upward pointing arrow (↑) is found. A horizontal arrow (→) pointing to a line indicates a technical change in only that line. A horizontal arrow located between two consecutive lines indicates technical changes in both lines or deletions.

CHANGE RECORD

Change Designation	Issue Date	Change Description
	1-77	Incorporated Addendum 1 issued August 1976
	4-77	F.1 revision changes (plus additional minor revisions)
	2-78	Deleted references to Varian
Rev. 1	7-79	G.1 (7R1.0) revision changes. Added information pertaining to micro-VORTEX and operation of VORTEX II on the V77-800. Incorporated Addendum 1 issued 11-77. Addendum 2 issued 11-77, and Addendum 3 issued 9-78.
Rev. 2	10-79	Miscellaneous editing changes. Replaced system generation information with patch program information (Section 15).
Update A	2-80	Included flexible diskette (F3064) information in Section 18 and other appropriate sections.
Update B	5-80	8R0 revision changes. Added Appendix I, deleted portions of Sections 13, 18, 19, and 20. Moved information from Section 26 to Section 19. Incorporated corrections and editorial changes.

Change Procedure:

When changes are made to this manual, updated pages are issued. These updated pages are either added to this manual or used to replace obsolete pages. The specific pages affected by each change are identified on the PAGE STATUS SUMMARY page.

TABLE OF CONTENTS

SECTION 1 INTRODUCTION

1.1	SYSTEM REQUIREMENTS.....	1-1
1.2	SYSTEM FLOW AND ORGANIZATION.....	1-2
1.2.1	Computer Memory.....	1-2
1.2.2	Rotating Memory Device.....	1-4
1.2.3	Secondary Storage.....	1-4
1.3	MEMORY MAP CONCEPT.....	1-4
1.4	BIBLIOGRAPHY.....	1-6

SECTION 2 REAL-TIME EXECUTIVE SERVICES

2.1	REAL-TIME EXECUTIVE MACROS.....	2-1
2.1.1	SCHED (Schedule) Macro.....	2-2
2.1.2	SUSPND (Suspend) Macro.....	2-3
2.1.3	RESUME Macro.....	2-3
2.1.4	DELAY Macro.....	2-3
2.1.5	LDELAY Macro.....	2-4
2.1.6	PMSK (PIM Mask) Macro.....	2-5
2.1.7	TIME Macro.....	2-5
2.1.8	OVLAY (Overlay) Macro.....	2-5
2.1.9	ALOC (Allocate) Macro.....	2-6
2.1.10	DEALOC (Deallocate) Macro.....	2-7
2.1.11	EXIT Macro.....	2-7
2.1.12	ABORT Macro.....	2-8
2.1.13	IOLINK (I/O Linkage) Macro.....	2-8
2.1.14	PASS Macro.....	2-9
2.1.15	TBEVNT (Set or Fetch TBEVNT) Macro.....	2-9
2.1.16	ALOCPG (Allocate Memory Pages) Macro.....	2-9
2.1.17	DEALPG (Deallocate Memory Pages) Macro.....	2-10
2.1.18	MAPIN (Map-In Specified Physical Pages of Memory) Macro.....	2-10
2.1.19	PAGNUM (Identify Physical Page Number) Macro.....	2-11
2.2	RTE SYSTEM FLOW.....	2-12
2.3	TASK LIMITATIONS AND DIFFERENCES.....	2-12
2.4	ABORT PROCEDURE.....	2-14
2.5	CHECKPOINTING OF TASKS.....	2-14
2.6	PAGE ALLOCATION SCHEME.....	2-14

Table of Contents

Section/Para	Title	Page	2.2	RTE SYSTEM FLOW	2-13
			2.3	TASK LIMITATIONS AND DIFFERENCES	2-14
1	INTRODUCTION	1-1	2.4	ABORT PROCEDURE	2-16
1.1	SYSTEM REQUIREMENTS	1-1	2.5	CHECKPOINTING OF TASKS	2-16
1.2	SYSTEM FLOW AND ORGANIZATION	1-2	2.6	PAGE ALLOCATION SCHEME	2-16
1.2.1	COMPUTER MEMORY	1-2			
1.2.2	ROTATING MEMORY DEVICE	1-4			
1.2.3	SECONDARY STORAGE	1-4			
1.3	MEMORY MAP CONCEPT	1-4	3	INPUT/OUTPUT CONTROL	3-1
1.4	BIBLIOGRAPHY	1-6			
2	REAL-TIME EXECUTIVE SERVICES	2-1	3.1	LOGICAL UNITS	3-1
2.1	REAL-TIME EXECUTIVE MACROS	2-1	3.1.1	SPECIAL LOGICAL UNITS	3-1
2.1.1	SCHED (SCHEDULE) MACRO	2-2	3.2	RMD FILE STRUCTURE	3-4
2.1.2	SUSPND (SUSPEND) MACRO	2-3	3.3	I/O INTERRUPTS	3-5
2.1.3	RESUME MACRO	2-3	3.4	SIMULTANEOUS PERIPHERAL OUTPUT OVERLAP (SPOOL)	3-5
2.1.4	DELAY MACRO	2-3	3.4.1	SPOOL OPERATION	3-6
2.1.5	LDELAY MACRO	2-4	3.4.2	SPOOL FILES	3-6
2.1.6	PMSK (PIM MASK) MACRO	2-5	3.5	I/O-CONTROL MACROS	3-7
2.1.7	TIME MACRO	2-5	3.5.1	OPEN MACRO	3-10
2.1.8	OVLAY (OVERLAY) MACRO	2-5	3.5.2	CLOSE MACRO	3-11
2.1.9	ALOC (ALLOCATE) MACRO	2-6	3.5.3	READ MACRO	3-12
2.1.10	DEALOC (DEALLOCATE) MACRO	2-7	3.5.4	WRITE MACRO	3-12
2.1.11	EXIT MACRO	2-7	3.5.5	REW (REWIND MACRO)	3-13
2.1.12	ABORT MACRO	2-8	3.5.6	WEOF (WRITE END OF FILE) MACRO	3-13
2.1.13	IOLINK (I/O LINKAGE) MACRO	2-8	3.5.7	SREC (SKIP RECORD) MACRO	3-13
2.1.14	PASS MACRO	2-9	3.5.8	FUNC (FUNCTION) MACRO	3-14
2.1.15	TBEVNT (SET OR FETCH TBEVNT) MACRO	2-9	3.5.9	STAT (STATUS) MACRO	3-15
2.1.16	ALOCPG (ALLOCATE MEMORY PAGES) MACRO	2-9	3.5.10	DCB (DATA CONTROL BLOCK) MACRO	3-16
2.1.17	DEALPG (DEALLOCATE MEMORY PAGES) MACRO	2-10	3.5.11	FCB (FILE CONTROL BLOCK) MACRO	3-16
2.1.18	MAPIN (MAP-IN SPECIFIED PHYSICAL PAGES OF MEMORY) MACRO	2-10	3.6	DISK DEVICE I/O OVERVIEW	3-17
2.1.19	PAGNUM (IDENTIFY PHYSICAL PAGE NUMBER) MACRO	2-11	3.6.1	ALTERNATE SECTOR PARTITION	3-17
2.1.20	RECOV (ERROR RECOVERY) MACRO	2-12	3.6.2	ALTERNATE SECTOR PROCESSING	3-18
2.1.21	AFAULT (ARITHMETIC FAULT SETUP) MACRO	2-12	3.6.3	DATA RECORD BLOCKING AND DEBLOCKING	3-18
2.1.22	SRFAULT (SAVE/RESTORE ARITHMETIC FAULT STATUS) MACRO	2-13	3.6.4	IDENTIFICATION BUFFER DESCRIPTION	3-18
2.1.23	SETPAR (SET PARITY ENABLE) MACRO	2-13	3.6.5	STATUS BUFFER DESCRIPTION	3-19
			3.6.5.1	Primary Status	3-19
			3.6.5.2	Secondary Status Buffer	3-19
			3.6.5.3	Disk Access Methods	3-19
			3.7	70-755x, 70-7560, 70-7561, 70-7562 I/O CONTROL	
			3.7.1	MACRO DESCRIPTIONS	3-20
			3.7.2	DEMAND	3-20
				RELEAS	3-20

Table of Contents (continued)

9.2.6	INIT (INITIALIZE) DIRECTIVE 9-5	11.2.17	ENDSORT DIRECTIVE 11-6
9.2.7	INPUT DIRECTIVE 9-6	11.3	USER EXITS 11-6
9.2.8	ADD DIRECTIVE 9-6	11.3.1	CALLING SEQUENCE 11-6
9.3	VORTEX FILE MAINTENANCE	11.3.2	IMPLEMENTATION 11-6
9.3.1	DRIVER (VZFMA) 9-6	11.4	VSORT MESSAGES 11-7
9.3.1.1	IOC FUNC REQUESTS TO VZFMA 9-8	11.5	BACKGROUND SORT 11-7
9.3.1.2	Exit Conditions 9-8	12	DATAPLOT II
9.3.1.3	Errors 9-8	12.1	SYSTEM FLOW OUTLINE 12-1
9.3.1.3	Explanation of FSB Contents 9-8	12.2	HARDWARE REQUIREMENTS . 12-1
10	INPUT/OUTPUT UTILITY PROGRAM	12.3	GENERAL DESCRIPTION 12-1
10.1	ORGANIZATION 10-1	12.3.1	DATAPLOT II ORGANIZATION... 12-1
10.2	I/O UTILITY DIRECTIVES 10-1	12.3.2	SYSTEM CONSIDERATION 12-3
10.2.1	COPYF (COPY FILE) DIRECTIVE .. 10-1	12.3.3	VORTEX CONSIDERATIONS 12-3
10.2.2	COPYR (COPY RECORD) DIRECTIVE 10-2	12.4	DATAPLOT II SUBROUTINES .. 12-5
10.2.3	SFILE (SKIP FILE) DIRECTIVE 10-3	12.4.1	DPINIT (SYSTEM FILE INITIALIZATION) 12-5
10.2.4	SREC (SKIP RECORD) DIRECTIVE 10-3	12.4.2	PLOTS (WORK BUFFER INITIALIZATION) 12-5
10.2.5	DUMP (FORMAT AND DUMP) DIRECTIVE 10-3	12.4.3	PLOT (GENERATE PLOT)..... 12-6
10.2.6	PRNTF (PRINT FILE) DIRECTIVE . 10-4	12.4.4	SCALE (GENERATE SCALE FACTOR) 12-6
10.2.7	WEOF (WRITE END OF FILE) DIRECTIVE 10-4	12.4.5	AXIS (GENERATE SEGMENTAL AXIS) 12-7
10.2.8	REW (REWIND) DIRECTIVE 10-4	12.4.6	SYMBOL (GENERATE SYMBOLS) 12-8
10.2.9	PFILE (POSITION FILE) DIRECTIVE 10-4	12.4.7	NUMBER (GENERATE NUMBER) 12-9
10.2.10	CFILE (CLOSE FILE) DIRECTIVE 10-5	12.4.8	LINE (GENERATE GRAPH LINE)..... 12-10
10.2.11	PACKB (PACK BINARY) DIRECTIVE 10-5	12.4.9	MLTIPLE (MULTIPLE PLOT) 12-11
10.3	MULTI-VOLUME TAPE HANDLING (V\$RSW) 10-5	12.4.10	FACTOR (ALTER PLOT SIZE) 12-11
11	VSORT (SORT/MERGE)	12.4.11	WHERE (LOCATE COORDINATES)..... 12-11
11.1	ORGANIZATION 11-1	12.4.12	APPEND (APPEND FILE) 12-12
11.2	VSORT DIRECTIVES 11-1	12.4.13	TOPFRM (TOP-OF-FORM)..... 12-12
11.2.1	SORT DIRECTIVE 11-2	12.4.14	CUT (CUT PAPER) 12-12
11.2.2	INPUT DIRECTIVE 11-2	12.4.15	ENDCUT (EJECT AND CUT PAPER)..... 12-13
11.2.3	ALTIN DIRECTIVE 11-2	12.4.16	DPSORT (SORT PLOT FILE) 12-13
11.2.4	OUTPUT DIRECTIVE 11-2	12.4.17	DPLOT (OUTPUT FILE)..... 12-13
11.2.5	WORK1-3 DIRECTIVE 11-3	12.4.18	DPCLOS (CLOSE PLOT FILE) 12-14
11.2.6	SORTKEY DIRECTIVE 11-3	12.4.19	ORIG -- OFFSETTING THE ORIGIN ENTRY POINT 12-14
11.2.7	ALTSEQ DIRECTIVE 11-3	12.4.20	VECT -- VECTOR ENTRY POINT 12-14
11.2.8	INCLF DIRECTIVE 11-3	12.4.21	SPECIAL SYMBOL SUBROUTINE 12-15
11.2.9	INCLC DIRECTIVE 11-4	12.4.22	DENSTY (ALTER STYLII/ INCH) 12-15
11.2.10	OMITF DIRECTIVE 11-4	12.5	PLOT FILE DATA FORMAT 12-15
11.2.11	OMITC DIRECTIVE 11-4	12.5.1	VECTORS 12-15
11.2.12	MOVEF DIRECTIVE 11-5	12.5.2	CHARACTERS 12-16
11.2.13	MOVEC DIRECTIVE 11-5	12.5.3	END-OF-PLOT INDICATOR 12-16
11.2.14	LOT DIRECTIVE 11-5	12.6	EXAMPLE OF APPLICATION OF DATAPLOT II 12-16
11.2.15	INEXIT DIRECTIVE 11-5		
11.2.16	OUTEXIT DIRECTIVE 11-5		

Table of Contents (continued)

12.6.1	PROGRAM TO GENERATE SINE WAVE..... 12-16	14.5.3	ITE INTERTASK COMMUNICATION MODULE... 14-42
12.6.2	PROGRAM TO GENERATE COMMUNICATION NETWORK . 12-16	14.6	MEMORY PARITY CONSIDERATIONS 14-54
12.7	OPERATING PROCEDURES AND ERROR MESSAGES 12-17	14.6.1	MEMORY PARITY CONSIDERATION (V70/V77-600) 14-54
12.7.1	VORTEX OPERATING PROCEDURES 12-17	14.6.2	MEMORY PARITY CONSIDERATIONS (V7-200/400) 14-54
12.7.2	UNSORTED PLOT FILES 12-17	14.6.3	V77-600 PARITY ERROR HANDLING..... 14-54
12.7.3	PRESORTED PLOT FILES 12-17	14.6.4	V77-800 PARITY ERROR HANDLING..... 14-55
12.7.4	VORTEX SPECIAL PROCEDURES 12-18		
13	SUPPORT LIBRARY		
13.1	CALLING SEQUENCE..... 13-1	15	THE PATCH PROGRAM
13.2	NUMBER TYPES AND FORMATS 13-1	15.1	GENERAL 15-1
13.3	SUBROUTINE DESCRIPTIONS.. 13-2	15.1.1	USER INTERFACE 15-1
		15.2	CONTROL DIRECTIVES 15-1
		15.2.1	.PTCH DIRECTIVE 15-1
		15.2.2	.DUMP DIRECTIVE 15-1
		15.2.3	.APND DIRECTIVE..... 15-2
		15.2.4	.HIST DIRECTIVE..... 15-2
		15.2.5	.BASE DIRECTIVE 15-2
		15.2.6	.LIBR DIRECTIVE 15-3
		15.2.7	.EXIT DIRECTIVE 15-3
		15.2.8	.SLCT DIRECTIVE 15-4
		15.2.9	.MANL DIRECTIVE 15-4
		15.2.10	.RECD DIRECTIVE 15-4
		15.2.11	.CNTL DIRECTIVE 15-5
		15.3	CHANGE DIRECTIVES 15-5
		15.4	PATCH IMAGE FILE FORMAT ... 15-7
		15.5	PATCH DIRECTIVE LOG FILE FORMAT 15-7
		15.6	BTPTCH INTERFACE 15-7
		15.7	SYSTEM GENERATION INTERFACE 15-8
		15.8	PATCHING CONSIDERATIONS . 15-8
		15.9	ERROR MESSAGES 15-9
		15.10	EXAMPLES 15-9
		16	SYSTEM MAINTENANCE
		16.1	ORGANIZATION 16-1
		16.1.1	CONTROL RECORDS 16-2
		16.1.2	OBJECT MODULES 16-3
		16.1.3	SYSTEM-GENERATION LIBRARY 16-3
		16.2	SYSTEM-MAINTENANCE DIRECTIVES 16-3
		16.2.1	IN (INPUT LOGICAL UNIT) DIRECTIVE 16-3
		16.2.2	OUT (OUTPUT LOGICAL UNIT) DIRECTIVE 16-4
		16.2.3	ALT (ALTERNATE LOGICAL UNIT) DIRECTIVE..... 6-4
		16.2.4	ADD DIRECTIVE 16-4
14	REAL-TIME PROGRAMMING		
14.1	INTERRUPTS 14-1		
14.1.1	EXTERNAL INTERRUPTS 14-1		
14.1.2	INTERNAL INTERRUPTS 14-3		
14.1.3	INTERRUPT-PROCESSING 14-4		
14.1.4	INTERRUPT STATE 14-4		
14.2	SCHEDULING 14-4		
14.2.1	SYSTEM FLOW 14-4		
14.2.2	PRIORITIES 14-5		
14.2.3	TIMING CONSIDERATIONS (APPROXIMATE) 14-28		
14.3	REENTRANT SUBROUTINES . 14-29		
14.4	CODING AN I/O DRIVER 14-30		
14.4.1	I/O TABLES 14-30		
14.4.2	I/O DRIVER SYSTEM FUNCTIONS 14-30		
14.4.3	ADDING AN I/O DRIVER TO THE SYSTEM FILE 14-31		
14.4.4	ENABLING AND DISABLING PIM INTERRUPTS 14-32		
14.4.5	DIRECTLY CONNECTED INTERRUPT HANDLER..... 14-34		
14.4.6	VORTEX USE OF BICS AND BTCS 14-34		
14.4.7	VORTEX II AND VORTEX COMPATABILITY..... 14-35		
14.4.8	MICRO-VORTEX (CPU-3) AND VORTEX II COMPATABILITY ... 14-36		
14.4.8.1	Functional Changes 14-36		
14.4.8.2	Restrictions..... 14-37		
14.4.9	RESIDENT TASKS..... 14-37		
14.4.10	PURGE CACHE COMMAND 14-38		
14.5	INTERTASK COMMUNICATION 14-38		
14.5.1	ITC MODULE OPERATION 14-38		
14.5.2	ITC CALLING SEQUENCES..... 14-38		

Table of Contents (continued)

16.2.5	REP (REPLACE) DIRECTIVE	16-5	18.1.3	LINE PRINTER	18-1
16.2.6	DEL (DELETE) DIRECTIVE	16-6	18.1.4	STATOS-31	18-1
16.2.7	LIST DIRECTIVE	16-6	18.1.5	33/35 ASR TELETYPE	18-1
16.2.8	END DIRECTIVE	16-6	18.1.6	HIGH-SPEED PAPER-TAPE	
16.3	SYSTEM-MAINTENANCE			READER	18-1
	OPERATION	16-6	18.1.7	MAGNETIC-TAPE UNIT	18-1
16.4	PROGRAMMING EXAMPLES ..	16-6	18.1.8	MAGNETIC-DRUM AND FIXED-	
				HEAD DISC UNITS	18-1
			18.1.9	MOVING-HEAD DISC UNITS	18-1
			18.1.10	MOVING-HEAD DISC UNITS	18-2
17	OPERATOR COMMUNICATION		18.1.11	MOVING-HEAD DISC UNITS	18-2
			18.1.12	MOVING-HEAD DISC UNITS	18-2
17.1	DEFINITIONS	17-1	18.1.13	MOVING-HEAD DISC UNITS	18-2
17.2	OPERATOR KEY-IN REQUESTS	17-1	18.2		
17.2.1	;SCHED (SCHEDULE			SYSTEM BOOTSTRAP	
	FOREGROUND TASK) KEY-IN			LOADER	18-2
	REQUEST	17-2	18.2.1	AUTOMATIC BOOTSTRAP	
17.2.2	;TSCHED (TIME-SCHEDULE			LOADER	18-2
	FOREGROUND TASK) KEY-IN		18.2.2	CONTROL PANEL LOADING	18-3
	REQUEST	17-2	18.2.3	SYSTEM BOOT HALT	18-3
17.2.3	;ATTACH KEY-IN		18.3	DISC PACK HANDLING	18-3
	REQUEST	17-3	18.3.1	PRT (PARTITION) DIRECTIVE	18-4
17.2.4	;RESUME KEY-IN REQUEST	17-3	18.3.2	FRM (FORMAT ROTATING	
17.2.5	;TIME KEY-IN REQUEST	17-3		MEMORY) DIRECTIVE	18-4
17.2.6	;DATE KEY-IN REQUEST	17-3	18.3.3	INL (INITIALIZE) DIRECTIVE	18-5
17.2.7	;ABORT KEY-IN REQUEST	17-4	→ 18.3.4	EXIT DIRECTIVE	18-5
17.2.8	;TSTAT (TASK STATUS)		18.4	WRITABLE CONTROL STORE	
	KEY-IN REQUEST	17-4		(WCS)	18-5
17.2.9	;ASSIGN KEY-IN REQUEST	17-5			
17.2.10	;DEVDN (DEVICE DOWN)			PROCESS INPUT/OUTPUT	
	KEY-IN REQUEST	17-5	19		
17.2.11	;DEVUP (DEVICE UP)			INTRODUCTION	19-1
	KEY-IN REQUEST	17-5	19.1	F2963 DATA ACQUISITION	
17.2.12	;IOLIST (LIST I/O)		19.2	AND CONTROL SYSTEM	19-1
	KEY-IN REQUEST	17-5		I/O MACROS	19-1
17.3	BLDSKD	17-6	19.2.1	REQUIRED HARDWARE	19-1
17.3.1	SET-UP REQUIREMENTS	17-6	19.2.2	HARDWARE DESCRIPTION	19-2
17.3.2	TASK SCHEDULING	17-6	19.2.3	SOFTWARE ADDRESSING	19-2
17.3.3	'A' COMMAND (ATTACH A		19.2.4	INPUT	19-3
	TASK TO A CONTROL		19.2.5	OUTPUT	19-6
	CHARACTER	17-6	19.2.6	THROUGHPUT	
17.3.4	'D' COMMAND (DELETE		19.2.7	CONSIDERATIONS	19-8
	A TASK)	17-7		FORTRAN PROCESS CONTROL	
17.3.5	'L' COMMAND (LIST		↓ 19.3	SUBROUTINES	19-9
	ASSIGNED TASKS)	17-7		PROCESS I/O PARAMETERS ..	19-9
17.3.6	'E' COMMAND (EXIT		19.3.1	PROCESS I/O CALLS	19-10
	BLDSKD)	17-7	19.3.2		
17.4	TASK SCHEDULING	17-7	↑ 19.4	IEEE STD 488-1975	
				DRIVER	19-11
			19.4.1	DESCRIPTION	19-11
18	OPERATION OF THE VORTEX SYSTEM		19.4.2	USER INTERFACE	19-11
			19.4.3	DATA AND INSTRUCTION	
18.1	DEVICE INITIALIZATION	18-1		FORMATS	19-13
18.1.1	CARD READER	18-1	19.4.4	SYSTEM AND USER	
18.1.2	CARD PUNCH	18-1		REQUIREMENTS	19-14

Table of Contents (continued)

20	WRITABLE CONTROL STORE AND FLOATING-POINT PROCESSOR		
20.1	MICROPROGRAMMING SOFTWARE 20-1	22.1	COMPRESSION/EDIT SYSTEM (COMSY)
20.1.1	MICROPROGRAM ASSEMBLER 20-1	22.1.1	ORGANIZATION 22-1
20.1.2	MICROPROGRAM SIMULATOR 20-1	22.1.2	COMSY COMPRESSION 22-1
20.1.3	MICROPROGRAM UTILITY 20-1	22.1.3	SEQUENTIAL FILES 22-2
20.1.4	WCS RELOAD TASK, WCSRLD 20-2	22.1.4	RANDOM FILES 22-2
20.2	STANDARD FIRMWARE 20-2	22.1.5	COMMON FILES 22-2
20.2.1	FIXED-POINT ARITHMETIC FIRMWARE..... 20-2	22.2	SEQUENCE AND EDITION NUMBERS..... 22-2
20.2.1.1	Fixed-Point Multiple (FIMPY) 20-3	22.3	INPUT/OUTPUT 22-2
20.2.2	FLOATING—POINT ARITHMETIC FIRMWARE 20-3	22.3.1	COMSY DIRECTIVES 22-2
20.2.3	DATA TRANSFER FIRMWARE..... 20-3	22.3.2	ASSIGN DIRECTIVE 22-3
20.2.4	FORTRAN-ORIENTED FIRMWARE 20-3	22.3.3	UNIT DIRECTIVE 22-4
20.2.5	BYTE MANIPULATION FIRMWARE..... 20-5	22.3.4	SET DIRECTIVE 22-4
20.2.6	STACK FIRMWARE 20-6	22.3.5	GANG DIRECTIVE 22-5
20.2.7	FIRMWARE MACROS 20-9	22.3.6	DECK DIRECTIVE 22-6
20.2.8	COMMERCIAL FIRMWARE 20-14	22.3.7	COMDECK DIRECTIVE 22-6
21	FILE MAINTENANCE UTILITY	22.3.8	COPY DIRECTIVE 22-7
21.1	INTRODUCTION 21-1	22.3.9	RANDOM DIRECTIVE 22-7
21.2	ORGANIZATION 21-1	22.3.10	APPEND DIRECTIVE 22-7
21.3	OUTPUT LISTINGS 21-1	22.3.11	EDIT DIRECTIVE 22-8
21.4	FMUTIL - OVERVIEW 21-1	22.3.12	LIST DIRECTIVE 22-8
21.5	D DIRECTIVE 21-2	22.3.13	CHECK DIRECTIVE 22-8
21.5.1	DUMP FILE 21-2	22.3.14	INSERT (ADD) DIRECTIVE 22-8
21.5.2	DUMP PARTITION 21-3	22.3.15	REPLACE (DELETE) DIRECTIVE 22-9
21.5.3	DUMP FILE-NAME DIRECTORY 21-3	22.3.16	COMMON DIRECTIVE 22-9
21.6	L DIRECTIVE 21-3	22.3.17	COMSY DIRECTIVE 22-10
21.6.1	LOAD FILE 21-3	22.3.18	FILE DIRECTIVE 22-10
21.6.2	LOAD PARTITION 21-4	22.4	END DIRECTIVE 22-10
21.6.3	LOAD DIRECTORY 21-5	22.5	COMSY LOAD MODULE GENERATION 22-11
21.7	R DIRECTIVE 21-5	22.6	COMSY EXECUTION 22-11
21.8	E DIRECTIVE 21-5	22.7	ERROR PROCESSING 22-11
21.9	S DIRECTIVE 21-6	23	COMSY EXAMPLE 22-11 ←
21.10	P DIRECTIVE 21-6	23.1	MULTITASK SPOOLER
21.11	U DIRECTIVE 21-6	23.1.1	GENERAL 23-1
21.12	T DIRECTIVE 21-6	23.2	COMMAND SUMMARY 23-1
21.13	X DIRECTIVE 21-7	23.2.1	SPOOL FILE OPEN AND CLOSE CALLS 23-2
21.14	I DIRECTIVE 21-7 ←	23.2.2	SPOP (SPOOL FILE OPEN) 23-2
21.15	C DIRECTIVE 21-7	23.3	SPCLS (SPOOL FILE CLOSE) 23-2
		23.4	SCHEDULING A SPOOL PRINT TASK
		23.4.1	SPOOL COMMUNICATION 23-2
		23.4.2	EX (SPOOL PRINT TASK EXIT) 23-3
			SG (SPOOL PRINT TASK GO) 23-3

Table of Contents (continued)

23.5	INITIATION OF COMMUNICATION WITH A SPOOL PRINT TASK... 23-3	25.4.3	PRINT COMMAND 25-2
23.6	SPOOL FILE STATUS DISPLAY 23-3	25.4.4	SWITCH COMMAND 25-2
23.7	DD (SPOOL FILE DELETE) 23-4	25.4.5	HELP COMMAND 25-2
23.8	PA, PH, PD (SPOOL FILE PRINT) 23-4	25.4.6	EXIT COMMAND 25-2
23.9	SC (SPOOL PRINT CANCELLATION) 23-5	25.4.7	STATUS COMMAND 25-2
23.10	ST (SPOOL PRINT TERMINATION) 23-5	25.5	STATUS WORD 25-2
23.11	SPOOL FILE ALLOCATION 23-6	25.6	OUTPUT FORMAT 25-3
23.12	COMPONENT DESCRIPTION 23-6	25.7	TELETYPEWRITER DRIVER (V\$TYB) ADDITIONS 25-4
23.13	SYSTEM GENERATION 23-7	25.8	ERROR MESSAGES 25-4
23.14	CONSOLE MESSAGES 23-8		
24 TAPE LABELING			
24.1	INTRODUCTION TO TAPE PROCESSING 24-1	→ Appendix A	ERROR MESSAGES
24.2	LABEL DEFINITIONS 24-1	A.1	ERROR MESSAGE INDEX A-1
24.3	LABEL ORGANIZATION 24-4	A.2	REAL-TIME EXECUTIVE A-1
24.4	DATA SET DEFINITION 24-7	A.3	I/O CONTROL A-5
24.5	TAPE MOUNTING 24-10	A.4	JOB-CONTROL PROCESSOR ... A-8
24.6	DATA SET ACCESS 24-10	A.5	LANGUAGE PROCESSORS A-9
24.7	INPUT DATA SETS 24-10	A.5.1	DASMR ASSEMBLER A-9
24.7.1	OPENING AN INPUT DATA SET 24-10	A.5.2	FORTRAN IV COMPILER AND RUNTIME COMPILER A-10
24.7.2	ACCESSING AN INPUT DATA SET 24-11	A.5.3	RPG IV COMPILER AND RUNTIME COMPILER A-11
24.7.3	CLOSING AN INPUT DATA SET 24-11	A.6	LOAD-MODULE GENERATOR . A-13
24.8	OUTPUT DATA SETS 24-11	A.6.1	RELINK A-14
24.8.1	OPENING AN OUTPUT DATA SET 24-11	A.7	DEBUGGING AIDS A-15
24.8.2	ACCESSING AN OUTPUT DATA SET 24-11	A.8	SOURCE EDITOR A-16
24.8.3	CLOSING AN OUTPUT DATA SET 24-11	A.9	FILE MAINTENANCE A-16
24.9	TAPE INITIALIZATION 24-11	A.10	I/O UTILITY A-18
24.10	SYSTEM GENERATION 24-12	A.11	SORT ERROR MESSAGES A-19
24.11	ERROR MESSAGES 24-12	A.12	DATAPLOT A-19
24.12	EXAMPLES OF USAGE 24-12	A.13	SUPPORT LIBRARY A-20
		A.14	REAL-TIME PROGRAMMING .. A-20
		A.15	SYSTEM GENERATION A-20
		A.16	SYSTEM MAINTENANCE A-21
		A.17	OPERATOR COMMUNICATION A-22
		A.18	RMD ANALYSIS AND INITIALIZATION A-22
		A.19	PROCESS INPUT/OUTPUT A-23
		A.20	WRITABLE CONTROL STORE .. A-23
		A.20.1	MICROPROGRAM ASSEMBLER . A-23
		A.20.2	MICROPROGRAM SIMULATOR . A-24
		A.20.3	MICROPROGRAM UTILITY A-25
		A.21	VTAM NETWORK CONTROL MODULE A-26
		A.22	FILE MAINTENANCE UTILITY (FMUTIL) ERRORS A-27
		A.23	COMSY ERROR MESSAGES .. A-28
		A.24	PARITY ERROR MESSAGES... A-29
		A.25	ERROR CODES A-29
		A.25.1	ERRORS RELATED TO DIRECTIVES A-30
		A.25.2	ERRORS RELATED TO PROGRAMS A-29
		A.25.3	ERRORS RELATED TO MEMORY SIZE A-30
		A.25.4	ERRORS RELATED TO HARDWARE A-30
25 CONSOLE LOGGING PROGRAM			
25.1	INTRODUCTION 25-1		
25.2	PROGRAM REQUIREMENTS 25-1		
25.3	PROGRAM DESCRIPTION 25-1		
25.4	COMMAND DESCRIPTION 25-1		
25.4.1	ON COMMAND 25-1		
25.4.2	OFF COMMAND 25-2		

Table of Contents (continued)

<p>Appendix B I/O DEVICE RELATIONSHIPS</p>		<p>Appendix H RMD STATUS WORDS</p>	
<p>Appendix C DATA FORMATS</p>		<p>H.1 70-76x0 RMD H-1</p> <p>H.2 70-7500 AND 70-7510 RMD ... H-1</p> <p>H.3 70-7520/7530 RMD H-2</p> <p>H.4 70- 7603/7613 RMD H-3</p> <p>H.5 70-755x, 70-7560, 707561, and 70-7562 RMD H-4</p> <p>H.5.1 PRIMARY STATUS H-4</p> <p>H.5.2 SECONDARY STATUS H-5</p> <p>H.6 F3064 FLEXIBLE DISKETTE H-5 ↓</p> <p>H.7 VZLPC LINE PRINTER DRIVER (SUL/ACORN PRINTER) H-6</p>	
<p>C.1 PAPER TAPE C-1</p> <p>C.1.1 BINARY MODE C-1</p> <p>C.1.2 ALPHANUMERIC MODE C-1</p> <p>C.1.3 UNFORMATTED MODE C-1</p> <p>C.1.4 SPECIAL CHARACTERS C-1</p> <p>C.2 CARDS C-2</p> <p>C.2.1 BINARY MODE C-2</p> <p>C.2.2 ALPHANUMERIC MODE C-2</p> <p>C.2.3 UNFORMATTED MODE C-4</p> <p>C.2.4 SPECIAL CHARACTER C-4</p> <p>C.3 MAGNETIC TAPE C-4</p> <p>C.3.1 SEVEN-TRACK C-4</p> <p>C.3.2 NINE-TRACK C-4</p> <p>C.4 STATOS PRINTER/PLOTTER ... C-4</p> <p>C.4.1 ALPHANUMERIC MODE C-4</p> <p>C.4.2 UNFORMATTED MODE C-4</p>		<p>Appendix I VORTEX II 8-BIT ASCII SUPPORT</p> <p>I.1 GENERAL I-1</p> <p>I.2 IOC MODIFICATIONS I-2</p> <p>I.3 IOUTIL MODIFICATIONS I-2</p> <p>I.3.1 OPTION S I-2</p> <p>I.3.2 NO OPTION I-4</p> <p>I.3.3 NOTES AND GUIDELINES FOR THE USER I-4</p> <p>I.4 SEDIT MODIFICATIONS I-4</p> <p>I.5 DASMR ASSEMBLER MODIFICATIONS I-5 ↑</p>	
<p>Appendix D STANDARD CHARACTER CODES</p>			
<p>Appendix E ASCII CHARACTER CODES</p>			
<p>Appendix F VORTEX HARDWARE CONFIGURATIONS</p>			
<p>Appendix G OBJECT MODULE FORMAT</p>			
<p>G.1 RECORD STRUCTURE G-1</p> <p>G.2 PROGRAM IDENTIFICATION BLOCK G-1</p> <p>G.3 DATA FIELD FORMATS G-1</p> <p>G.4 LOADER CODES G-1</p> <p>G.5 EXAMPLE G-3</p> <p>G.5.1 SOURCE MODULE G-3</p> <p>G.5.2 OBJECT MODULE G-3</p> <p>G.5.3 CORE IMAGE G-5</p> <p>G.6 END LOAD RECORD G-6</p>			
		<p>INDEX</p>	

List of Illustrations

1-1	VORTEX System Flow	1-2	14-2	VORTEX Priority Structure	14-6
1-2	VORTEX Nucleus, Map 0	1-3	14-3	Driver Interface	14-33
1-3	VORTEX RMD Storage Map	1-4	16-1	SMAIN Block Diagram	16-1
2-1	Matrix of Nucleus Module		16-2	SMAIN LIST Directive Listing	16-7
	Access Mode	2-15	20-1	Base and Limit of Stack	20-6
2-2	V\$PAGE, Page Allocation Table	2-16	20-2	Stack Control Block	20-6
3-1	Spooling Subsystem Flow	3-6	20-3	Stack Multiply	20-7
5-1	VORTEX Macro Definitions for		20-4	Stack Divide	20-7
	DASMR	5-2	20-5	Stack Push	20-7
5-2	Sample Assembly Listing	5-10	20-6	Stack Pop	20-7
5-3	Sample Concordance Listing	5-13	20-7	Stack Double Push	20-8
5-4	FORTTRAN I/O Execution		20-8	Stack Double Pop	20-8
	Sequences	5-19	21-1	Sample Output Produced by Dump	
6-1	Load-Module Overlay Structure			File Directive	21-2
	(Virtual Memory)	6-2	22-1	COMSY Data Flow	22-1
12-1	DATAPLOT II Graphics System Data		23-1	Multitask Spool System Flow	23-1
	Flow	12-2	24-1	Layout of Single Data Set - Single	
12-2	DATAPLOT II Organization	12-2	24-2	Volume	24-4
12-3	Minimum and Maximum Plot			Layout of Single Data Set - Multiple	
	Values	12-4	24-3	Volumes	24-4
12-4	+ X Axis and + y Axis Relative to			Layout of Multiple Data Set - Single	
	Paper Direction	12-14	24-4	Volume	24-5
12-5	Vector-Data Format	12-15		Layout of Multiple Data Sets - Multiple	
12-6	Character Data Format	12-16	26-1	Volumes	24-6
12-7	Character Orientation Data		C-1	Typical System Configuration	26-1
	Format	12-16	C-2	Paper Tape Binary Record Format ...	C-1
12-8	End-of-Plot Indicator	12-16	C-3	Paper Tape Alphanumeric Record	
12-9	Sine Wave Plot Generated by		C-4	Format	C-2
	DATAPLOT II	12-17		Card Binary Record Format	C-3
12-10	Communication Network Plot Generated			Card Alphanumeric Record Format	
	by DATAPLOT II	12-17		(IBM 026)	C-3
14-1	Interrupt Line Handlers	14-2			

List of Tables

1-1	Executive Mode States	1-6	15-1	Supplementary Defined Mnemonics ..	15-6
2-1	RTE Service Request Macros	2-1	17-1	Physical I/O Devices	17-1
3-1	VORTEX Logical-Unit Assignments ..	3-2	17-2	Task Status (TIDB Words 1 and 2) ..	17-4
3-2	Valid Logical-Unit Assignments	3-3	18-1	Key-In Loader Programs	18-2
3-3	FCB Words Under I/O Macro Control	3-16a	20-1	Firmware Availability	20-2
3-4	IOC Word Usage for 70-755x, 70-7560,		21-1	FMUTIL Directives	21-1
	70-7561, 70-7562 Disk	3-22	22-1	COMSY Logical Units	22-2
5-1	Directives Recognized by the DASMR		24-1	Volume Label 1 (VOL1) Format	24-1
	Assembler	5-1	24-2	Header Label 1 (HDR1) Format	24-2
5-2	RTE Macros Available Through		24-3	Header Label 2 (HDR2) Format	24-3
	FORTTRAN IV	5-13	25-1	Status Word (CLSTAT)	25-3
7-1	DEBUG Directives	7-1			
9-1	File Maintenance Control Block	9-7			
13-1	DAS Coded Subroutines	13-2			
13-2	OM Library Subroutines	13-7	↓		
13-3	FORTTRAN IV Coded Subroutines ..	13-8	I-1	8-Bit ASCII Character Set	I-1
14-1	TIDB Layout	14-7	I-2	8-Bit Special Characters, Katakana ..	I-2
14-2	TIDB Description	14-12	I-3	IOUTIL COPY Directive Conversions,	
14-3	Map of Lowest Memory Sector	14-18	I-4	Option S	I-3
14-4	TIDB Status-Word Bits	14-26		IOUTIL COPY Directive Output,	
			↑	Without Option S	I-4

SECTION 1 INTRODUCTION

SPERRY UNIVAC VORTEX II is a modular software operating system for controlling, scheduling, and monitoring tasks in a real-time multiprogramming environment. VORTEX II supports memory map operation to a maximum of 1024K of central memory. VORTEX II also provides for background operations such as compilation, assembly, debugging, or execution of tasks not associated with the real-time functions of the system. The basic features of VORTEX II are:

- Memory map management
- Real-time I/O processing
- Provision for directly connected interrupts
- Interrupt processing
- Multiprogramming of real-time and background tasks
- Overlapping output to peripherals with spooling
- Priority task scheduling (clock time or interrupt)
- Load and go (automatic)
- Centralized and device-independent I/O system using logical unit and file names
- Operator communications
- Batch-processing job-control language
- Program overlays
- Background programming aids: FORTRAN and RPG IV compilers, DAS MR assembler, load-module generator, library updating, debugging, and source editor.
- Use of background area when required by foreground tasks
- Disc/drum directories and references
- System generator
- Individual task protection

NOTE: Throughout this manual, all references to VORTEX imply VORTEX II.

1.1 SYSTEM REQUIREMENTS

VORTEX requires the following minimum hardware configuration:

- a. SPERRY UNIVAC 70 series computers with 32K memory
- b. 33/35 ASR Teletype or compatible CRT on a priority interrupt module
- c. Priority Interrupt Module (PIM)
- d. Rotating memory device (RMD) on a PIM with either a buffer interlace controller (BIC), block transfer controller (BTC), or direct memory interface.
- e. One of the following on a PIM:
 - (1) Card reader with a BIC
 - (2) Paper-tape system or a paper-tape reader
 - (3) Magnetic-tape unit with a BIC
- f. Memory map hardware

The system supports and is enhanced by the following optional hardware items:

- a. Additional main memory (up to a total of 1024K)
- b. Additional rotating memory devices
- c. Automatic bootstrap loader with VORTEX II (device dependent) system boot
- d. Card reader, if one is not included in the minimum system with BIC and PIM
- e. Card punch with BIC and PIM
- f. Line printer with BIC and PIM
- g. Paper-tape punch, if one is not included in the minimum system
- h. Process input and output
- i. Data communications multiplexor
- j. Electrostatic printer/plotter
- k. Writable control store
- l. Floating-point processor
- m. V75 extended instruction set.

All BICs, BTCs, and DCMs must have memory mapping capability.

The rotating-memory device (RMD) serves as storage for the VORTEX operating system components, enabling real-time operations and a multiprogramming environment for solving real-time and nonreal-time problems. Real-time processing is implemented by hardware interrupt controls and software task scheduling. Tasks are scheduled for

INTRODUCTION

execution by operator requests, other tasks, device interrupts, or the completion of time intervals.

Background processing (nonreal-time) operations, such as FORTRAN compilations or DAS MR assemblies, are under control of the job-control processor (section 4), itself a VORTEX background task. These background processing operations are performed simultaneously with the real-time foreground tasks until execution of the former is suspended, either by an interrupt or a scheduled task.

1.2 SYSTEM FLOW AND ORGANIZATION

VORTEX executes foreground and background tasks scheduled by operator requests, interrupts, or other tasks. All tasks are scheduled, activated, and executed by the real-time executive component on a priority basis. Thus, in the VORTEX operating system, each task has a level of priority that determines what will be executed first when two or more tasks come up for execution simultaneously.

The job-control processor component of the VORTEX system manages requests for the scheduling of background tasks.

Upon completion of a task, control returns to the real-time executive. In the case of a background task, the real-time executive schedules the job-control processor to determine if there are any further background tasks for execution.

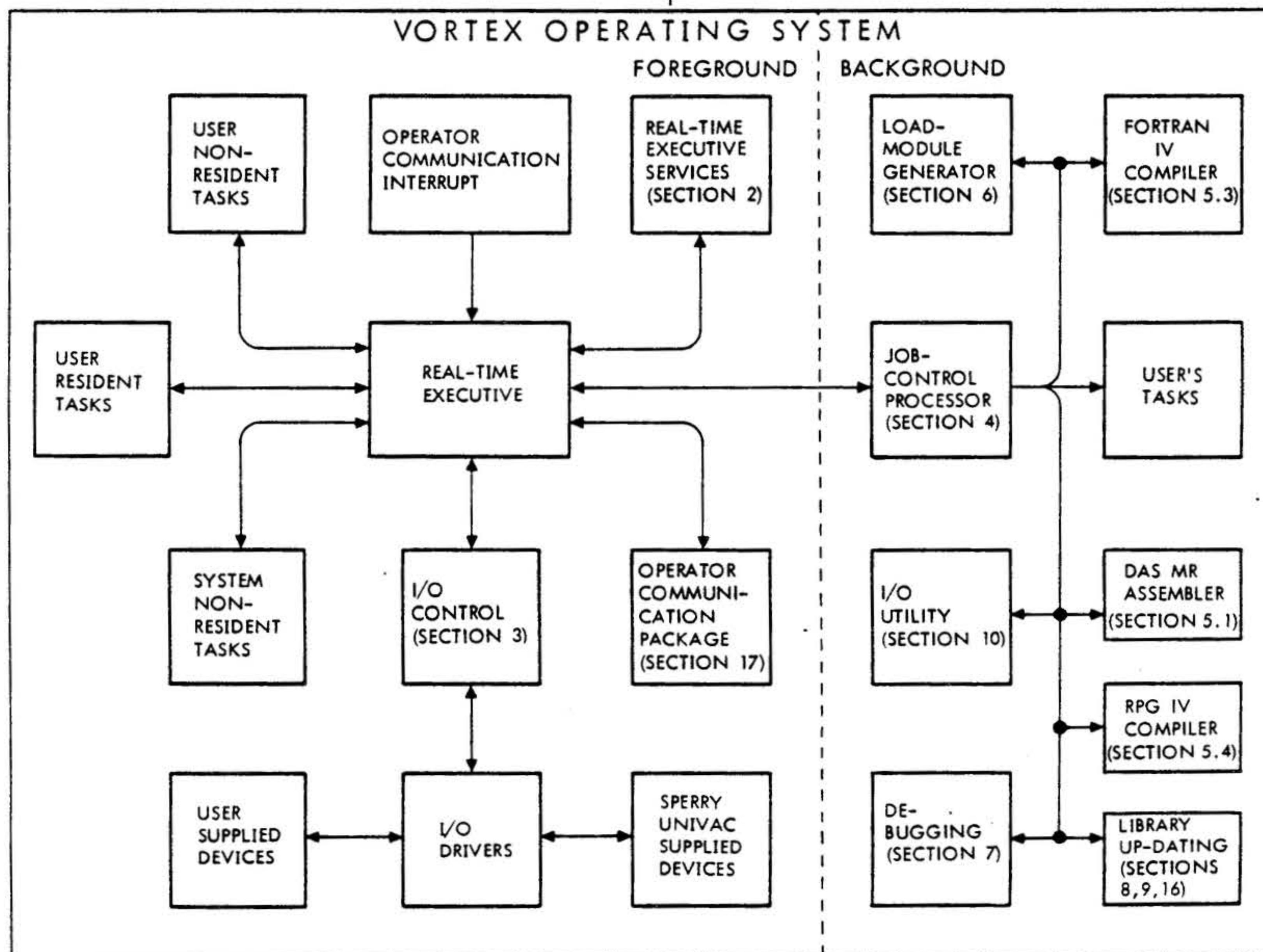
During execution, any foreground task can use any real-time executive service (section 2.1).

Figure 1-1 is an overview of the flow in the VORTEX operating system. Section numbers refer to further discussion of this manual.

1.2.1 Computer Memory

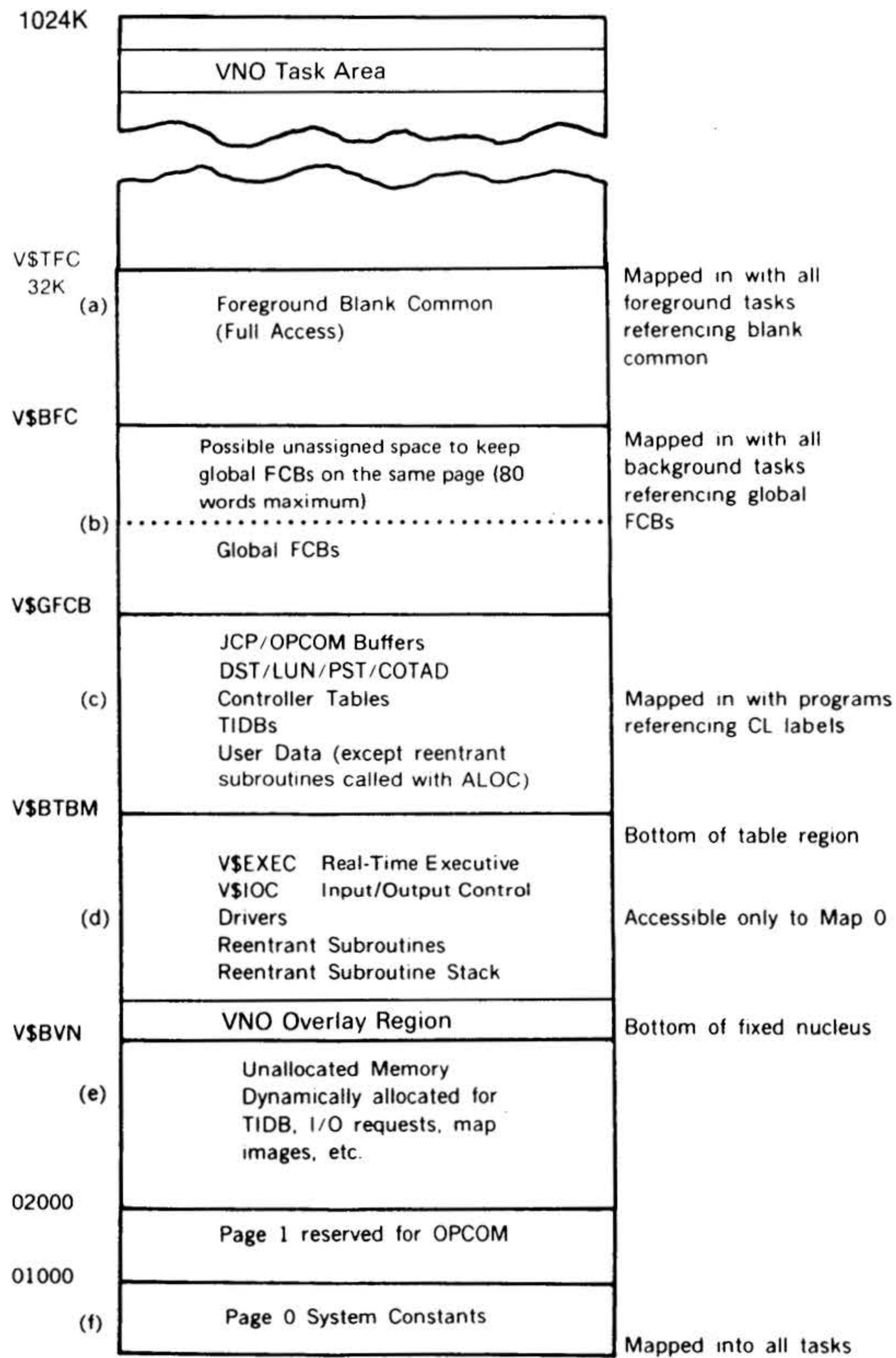
VORTEX requires a minimum of 32K words of main memory and supports up to a maximum of 1024K words.

↓ The system generation programs (see VORTEX II System Generation User Guide/Programmer Reference, UP-9083) execute in a non-memory map environment and consequently utilize only the first physical 32K words of main memory. All resident tasks and data reside in the first 32K of logical memory and are considered as part of the VORTEX nucleus. The nucleus is assigned to be in the executive mode, map 0, virtual memory (see section 1.3).



VTII-1314

Figure 1-1. VORTEX System Flow



NOTE: V\$TFC, V\$BFC, etc. are system pointers in page 0 described in section 14, table 14-1.

NOTE: V\$TFC, top of nucleus, is specified on SGEN MRY directive (described in section 15.5.1).

Figure 1-2. VORTEX Nucleus, Map 0

INTRODUCTION

memory. All resident tasks and data reside in the first 32K of logical memory and are considered as part of the VORTEX nucleus. The nucleus is assigned to be in the executive mode, map 0, virtual memory (see section 1.3).

Figure 1-2 illustrates the map 0 nucleus memory layout. The 32K words memory space is grouped into several modules:

- a. **Foreground Blank Common Module:** This module is mapped with all foreground tasks referencing blank common.
- b. **Global FCB Module:** This module is mapped with all background tasks referencing the global FCBs. It is read only access mode for priority 0 tasks and read/write for priority 1 tasks. This module is of approximately 90 words.
- c. **Nucleus Table Module:** This module is mapped with all tasks with an external name defined in the CL library. Read-only access mode for priority 0 tasks and read/write access for all other tasks. The bottom of this module is defined in V\$BTBM and is determined by SGEN during the nucleus module building. Control record CTL,21 specifies the end of the nucleus table module. All user data and programs which are to be included in this module must precede the CTL,21 control record. The approximate size of this module is 1000 words (RMD, line printer, card reader, Teletype, CRT).
- d. **Nucleus Programs Module:** This module consists of V\$EXEC, V\$I/O, I/O drivers, reentrant subroutines, stacks, and any user programs inserted between the CTL,21 and CTL,PART0003 SGEN tasks. The bottom of this module is defined by V\$CRDR.
- e. **Map 0 Allocable Memory Space:** The virtual memory space between page two and V\$CRDR is available for dynamic allocation. I/O request block, TIDB block, and map image memory space are allocated in this region. Page one is reserved for the OPCOM task. The actual physical memory assigned to the virtual memory space is memory management performed by the RTE component.
- f. **Page 0:** Always reserved for system constants, interrupt traps, and background literal pool (a description is found in section 14, table 14-3).

The unused physical memory in the first 32K and all physical memory above 32K are designated as allocable memory. This is the physical memory which is dynamically allocated for map 0 memory space as described in e, and which is allocated to a user mode task's logical memory.

1.2.2 Rotating Memory Device

At least one RMD (disc or drum) is required for storage of VORTEX operating system components. The RMD is divided into a fixed number of variable-length areas called **partitions**. These are defined at system-generation time (see VORTEX II System Generation User Guide/Programmer Reference, UP-9083).

The following reside on the RMD (figure 1-3):

- a. System initializer, loader, and VORTEX nucleus in absolute format
- b. Checkpoint file
- c. GO file
- d. User library
- e. Transient files
- f. Relocatable object-module library
- g. Relocatable load-module library

System Initializer and Loader
VORTEX Nucleus in Absolute Format
CL Directory
Relocatable Object-Module Library
Relocatable Load-Module Libraries
Checkpoint File
GO File
User Library
Transient Files

Figure 1-3. VORTEX RMD Storage Map

1.2.3 Secondary Storage

The VORTEX operating system supports any secondary storage devices that have been specified at system-generation time.

1.3 MEMORY MAP CONCEPT

VORTEX logical (virtual) memory is defined to be 32K words. This is the maximum memory space that any single task can address, even though the physical memory space may be as great as 1024K words. Where in actual or

physical memory that task resides is transparent to the task and is a memory management function performed by the RTE component of VORTEX.

Each logical memory space (32K) is organized into fixed-size blocks of 512 words (01000 in octal), called logical (virtual) pages. Hence, there are 64 logical pages within a 32K logical memory space. The size of the logical memory available to a task is reduced by:

- a. Page 0: The first page of 512 words is reserved for system constants, interrupt trap locations, background literal pool and communication link for IOC and V\$EXEC calls. This page is mapped in all logical memories.
- b. Nucleus Modules: A task referencing an external name which is defined in the CL library will have the corresponding VORTEX nucleus module mapped in logical memory for a task. (Section 1.2.1 describes in greater detail the nucleus modules.) These are:
 - (1) Foreground blank common module.
 - (2) Global FCB module, and/or
 - (3) Nucleus table module
- c. Any FORTRAN program performing input/output operation will have the nucleus table module mapped into its virtual memory. FORTRAN runtime package requires access to the device specification table (DST), logical unit tables (LUT), and controllers tables for linking information. The maximum available logical memory space available is V\$BTBM (bottom of nucleus table module, location 0331) minus 01000 (program start logical address). V\$BTBM is defined on the SGEN listing.
- d. For background priority 1 tasks, page 0 is set to read/write access mode to permit tasks, e.g., JCP, to modify low memory pointers V\$JCFG, V\$CRDM, etc. Hence, the method of transferring control from user mode to executive mode for I/O and RTE calls is to map in the pages containing the entry to V\$IOC (I/O calls), V\$EXEC (RTE calls), and V\$IOST (STAT calls). Therefore a priority 1 task making an I/O call (or RTE call, or STAT call), executes a JSR,X to location 0404. Because page 0 is set to read/write access mode, the instruction at 0404 (JMP V\$IOC) is executed. The first instruction in V\$IOC (likewise, V\$EXEC and V\$IOST) is a disable PIM (EXC 0444) instruction. Execution of an I/O type instruction in the user map generates a memory-protection interrupt, which forces the system to the executive mode and hence the means of transferring control to the map 0 tasks. Therefore, the available memory space for a background task is from location 01000 to the page where V\$IOC (which is lower in memory than V\$EXEC) resides. V\$IOC address is defined on the SGEN output listing.

All user mode tasks are loaded from logical address 01000. A task not referencing external names defined in the CL library has all of the logical memory available to it except page 0.

Physical memory is also organized into fixed-size blocks of 512 words, referred to as physical pages. A system with

physical memory size of 256K words contains 512 physical pages (64 physical pages for each 32K words of memory).

Allocation of logical memory to physical memory is accomplished by pages. A task of 010000 (4096 in decimal) words will reside in eight physical pages of physical memory. These physical pages need not be contiguous. However, that fact is transparent to the task. During execution, the task assumes that its eight pages are contiguous. The linking of physical pages is performed by the memory map hardware. All user program object modules are assembled relative to location 0. Load modules are generated by SGEN and LMGGEN to be relative to logical address 01000.

A map defines the 64 logical pages within a logical memory. Each logical page can be set to one of four possible access modes:

Unassigned	The logical addresses within that virtual page are unassigned.
Read/Write	All accesses including write operation permitted to/from the logical page.
Read Operand Only	Only operand fetches permitted from the logical page.
Read Only	Only instruction or operand fetches permitted within the logical page.

Each logical page, except for the pages with unassigned status, must be assigned to a physical page. The RTE task sets the status for each page, allocates a physical page to each logical page, and loads the corresponding mapping registers.

The memory map hardware provides a 4-bit map register for the 16 possible maps. This 4-bit map register is set by the RTE component to select the proper map (0-15). Map 0 is defined as the executive mode. All other map selections (1-15) are designated as being in the user mode. However, when the system is forced to the executive mode, state 0, by an I/O, real-time, or memory map interrupt, the map register will continue to contain the currently executing user map selection number.

Executive Mode

All instructions except HALT are permitted in this mode. Any interrupt will force the hardware to enter this mode in executive mode state 0. The interrupt will not disable the map. VORTEX Real-Time Executive (RTE), Input/Output Control (IOC), I/O drivers, and other resident tasks and constants are mapped into the executive mode. The instructions and data which comprise the VORTEX nucleus are mapped in the executive mode. Any task executing I/O instructions (EXC, OAR, SEN, etc.) must execute in map 0.

A HALT instruction executed in the executive mode with the map enabled will generate an interrupt. The HALT is permitted only in the disabled map state.

INTRODUCTION

There are four executive modes states as shown in table 1-1. A map 0 task will normally execute in state 0. In state 0, all instruction fetches and operand fetches and stores are performed in map 0 logical memory. If a map 0 task must fetch and store data to or from a user map (1-15), the map 0 task must switch to the proper executive mode state (1, 2 or 3), then upon completion of the fetch or store, restore the executive mode to state 0. A convenient way of switching executive or mode states is to output one of the control words established by the RTE component in the page 0 system data region, locations 0334-0337: V\$ST0, V\$ST1, V\$ST2, and V\$ST3 for executive mode states 0 through 3 respectively. An example of switching to executive mode 3 is OME 046, V\$ST3, where 046 is the memory-map device address.

User Mode

All operands and instructions are mapped in accordance with the map register contents. Error conditions will cause interrupts, which force the system to the executive mode. User mode is entered from the executive mode under control of RTE.

Privileged instructions (e.g., EXC, HALT) are not permitted in this mode. An interrupt is generated if a task attempts to execute a privileged instruction. Foreground tasks may execute disable and/or enable PIMS and RT clock instructions (EXC 0444, EXC 0244, EXC 0147, EXC 0747). Section 14.4.4 describes this subject further.

Section 2.2, RTE System Flow, describes the user mode and executive mode tasks.

Table 1-1. Executive Mode States

State	Instruction Fetch	Operand Fetch	Store
0	MAP 0	MAP 0	MAP 0
1	MAP 0	MAP 0	*MAP N
2	MAP 0	MAP N	MAP 0
3	MAP 0	MAP N	MAP N

+ MAP 0 refers to the executive task map.

*MAP N refers to the task map specified by the map register. (n = 1-15)

1.4 BIBLIOGRAPHY

The following Sperry Univac manuals are pertinent to the use of VORTEX and the 70/620 series computers:

Title	Document Number
FORTRAN IV (VORTEX) Reference Manual	98 A 9952 04x
HASP/RJE Operator's Manual	98 A 9952 21x
VTAM Reference Manual	98 A 9952 22x
VORTEX Installation Manual	98 A 9952 25x
TOTAL Reference Manual	98 A 9952 41x
RPG II Reference Manual	98 A 9952 42x
COBOL Reference Manual	98 A 9952 43x
TSS Reference Manual	98 A 9952 44x
Assembly Language Reference Manual	98 A 9952 45x
VIDEO Reference Manual	98 A 9952 46x
V70 Architecture Reference Manual	98 A 9906 000

SECTION 2 REAL-TIME EXECUTIVE SERVICES

The VORTEX **real-time executive (RTE) component** processes, upon request by a task, operations that the task itself cannot perform, including those involving linkages with other tasks. RTE **service requests** are made by macro calls to V\$EXEC, followed by a parameter list that contains the information required to process the request.

The contents of the volatile A and B registers and the setting of the overflow indicator are saved during execution of any RTE macro. After completion of the macro, these values are returned. The contents of the X register are lost. If the task uses the V75 registers 3 through 7, the contents of R3 through R7 are also saved.

There are 32 priority levels in the VORTEX system, numbered 0 through 31. Levels 0 and 1 are for background tasks and levels 2 through 31 are for foreground tasks. If a background task is assigned a foreground priority level, or vice versa, the task automatically receives the lowest valid priority level for the correct environment. Lower numbers assign lower priority. If more than one task has the same priority level, they are selected for execution on a first-in, first-out basis. Background and foreground RTE service requests are similar.

Table 2-1. RTE Service Request Macros

Mnemonic	Description	Level 0	FORTTRAN
SCHED	Schedule a task	Yes	Yes
SUSPND	Suspend a task	Yes	Yes
RESUME	Resume a task	No	Yes
DELAY	Delay a task	No	Yes
LDELAY	Delay and reload from specified logical unit	No	Yes
PMSK	Store PIM mask register	No	Yes
TIME	Obtain time of day	Yes	Yes
OVLAY	Load and/or execute an overlay segment	Yes	Yes
ALOC	Allocate a reentrant stack	No	Yes
DEALLOC	Deallocate the current reentrant stack	No	No
EXIT	Exit from a task (upon completion)	Yes	Yes
ABORT	Abort a task	No	Yes

IOLINK	Link background I/O	Yes	No
PASS	Pass map 0 data	Yes	Yes
TBEVNT	Set/fetch task's TBEVNT	Yes	No
ALOCPG	Allocate memory page(s) (Priority 0 in map 0)	Yes	No
DEALPG	Deallocate memory page(s) (Priority 0 in map 0)	Yes	No
MAPIN	Map in specified memory page(s)	No	No
PAGNUM	Identify physical page number	Yes	No

Whenever a task is aborted, all currently active I/O requests are completed. Pending I/O requests are de-queued. Only then is the aborted task released.

There are 18 RTE service request macros. Certain of them are illegal in unprotected background (level 0) tasks. Table 2-1 lists the RTE macros, indicates whether they are legal in level 0 tasks, and indicates whether there is a FORTRAN library subroutine (section 13) provided.

Note: A **task name** comprises one to six alphanumeric characters (including \$), left-justified and filled out with blanks. Embedded blanks are not permitted.

2.1 REAL-TIME EXECUTIVE MACROS

This section describes the RTE macros given in table 2-1.

The general form of an RTE macro is

label **mnemonic**,*p(1)*,*p(2)*,...,*p(n)*

where

label permits access to the macro from elsewhere in the program

mnemonic is one of those given in table 2-1

each *p(n)* is a parameter defined under the descriptions of the individual macros

The omission of an optional parameter is indicated by retention of the normal number of commas unless the omission occurs at the end of the parameter string. Thus, in the macro (section 2.1.1)

REAL-TIME EXECUTIVE SERVICES

SCHED 8,,106,, 'TA', 'SK', 'A'

the first double comma indicates a default value for the wait option and the second double comma indicates omission of a protection code.

Error messages applicable to RTE macros are given in Appendix A.2.

2.1.1 SCHED (Schedule) Macro

This macro schedules the specified task to execute on its designated priority level. The scheduling task can pass two values in the A and B registers to the scheduled task (a task using the V75 registers 3 through 7 can also pass parameters in R3 through R7). A TIDB is created for each scheduled task, (see section 14 for a description of TIDB). The macro has the general form.

label SCHED level, wait, lun, key, 'xx','yy','zz', f

where

- level** is the value from 0 (lowest) to 31 (highest) of the priority level of the scheduled task
- wait** is 0 (default value) if the scheduling and scheduled task obtain CPU time based on priority levels and I/O activity, or 1 if the scheduling task is suspended until completion of the scheduled task
- lun** is the name or number of the logical unit whose library contains the scheduled task, zero to schedule a resident foreground task, or 106 to schedule a nonresident task from the foreground library. If a zero is specified and the task is not found in the resident directory, the RTE component (SAL) will automatically search for the task on the foreground library (FL)
- key** is the protection code, if any, required to address lun (0306 or 'F' to schedule a nonresident task from the foreground library). The foreground library logical unit and its protection key are specified by the user at system-generation time
- xyyzz** is the name of the scheduled task in six ASCII characters, coded in pairs between single quotation marks and separated by commas; e.g., the task named BIGJOB is coded 'BI','GJ','OB' and the task named ZAP is coded 'ZA','P',' '
- f** is 1 if the TIDB address of the scheduled task is to be returned in the A register, or 0 (default value) if the original value is to be returned.

The FORTRAN calling sequence for this macro is

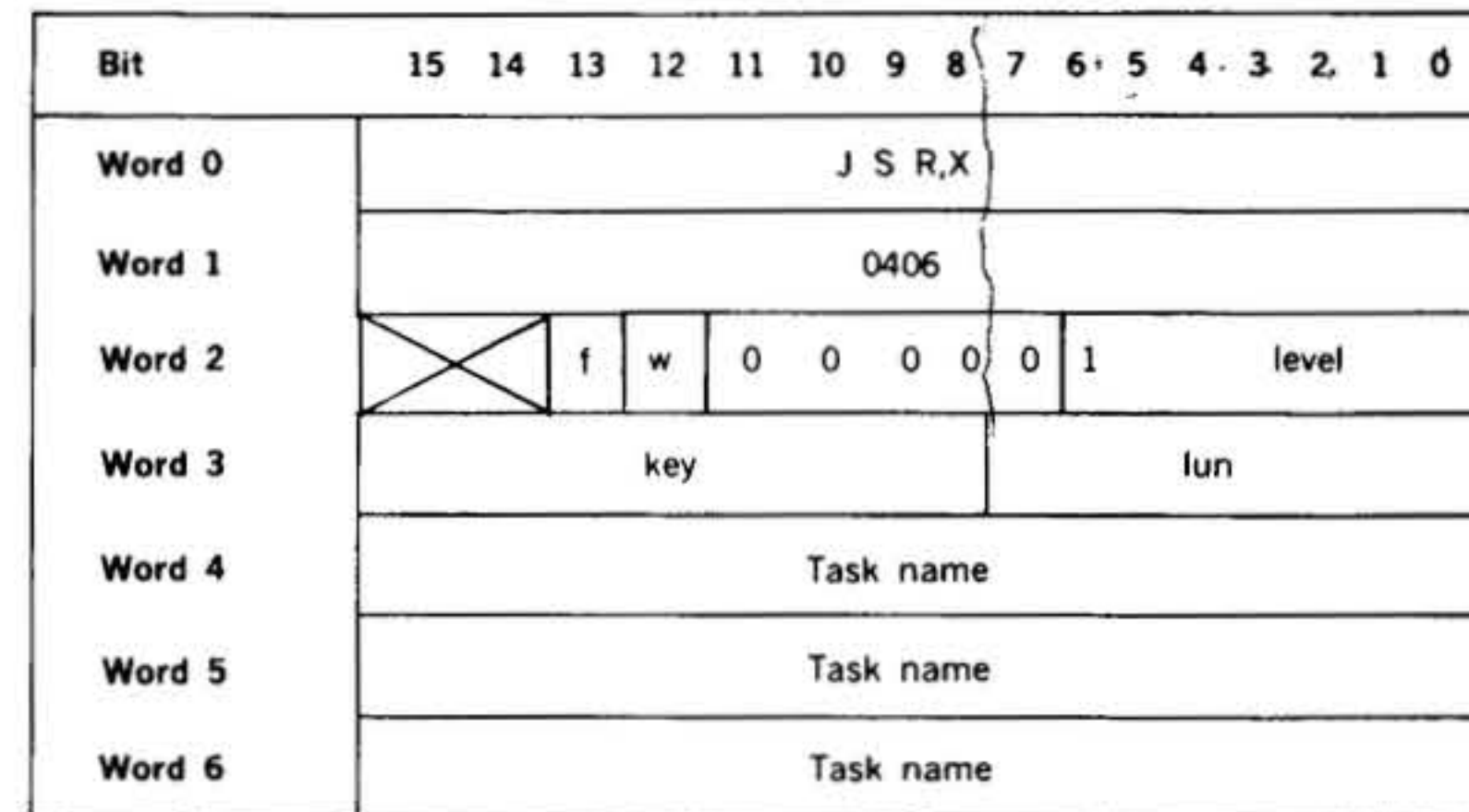
CALL SCHED(level,wait,lib,key,name)

where lib is the number of the library logical unit containing the task, and name is the three-word Hollerith array containing the name of the scheduled task. The other parameters have the definitions given above.

All tasks are activated at their entry-point locations, with the A and B registers (and the V75 registers if available) containing the value to be passed. The scheduled task executes when it becomes the active task with the highest priority.

The specified logical unit (which can be a background library, a foreground library, or any user-defined library on an RMD) must be defined in the schedule-calling sequence.

Expansion: The task name is loaded two characters per word. The wait option flag is bit 12 of word 2 (w). The scheduled task TIDB address flag is bit 13 or word 2 (f).



Examples: Schedule the foreground library task named TSKONE on priority level 5. Use the no-wait option so that scheduled and scheduling tasks obtain Central-Processor-Unit (CPU) time based on priority levels and I/O activity.

```

FL      EQU      106      (LUN assigned to
                          foreground library FL)
KEY     EQU      0306    (Protection code
                          for FL)
.
.
.
SCHED  5, 0, FL, KEY, 'TS', 'KO', 'NE'
.
.
.
        (Control return to
        highest priority)
    
```

Note: the KEY line can be coded with the equivalent ASCII character enclosed in single quotation marks.

```
KEY     EQU      'F'
```

The same request in FORTRAN is

```

DIMENSION N1, N2(3)
DATA N1/2H F/
DATA N2(1), N2(2), N2(3)/2HTS, 2HKO, 2HNE/
CALL SCHED(5, 0, 106, N1, N2)
or
CALL SCHED(5, 0, 106, 2H F, 6HTSKONE)
    
```


2.1.2 SUSPND (Suspend) Macro

This macro suspends the execution of the task initiating the macro. The task can be resumed only by an external interrupt, a simulated interrupt caused by IOC or I/O completion events for the task, or a RESUME (section 2.1.3) macro. The macro has the general form

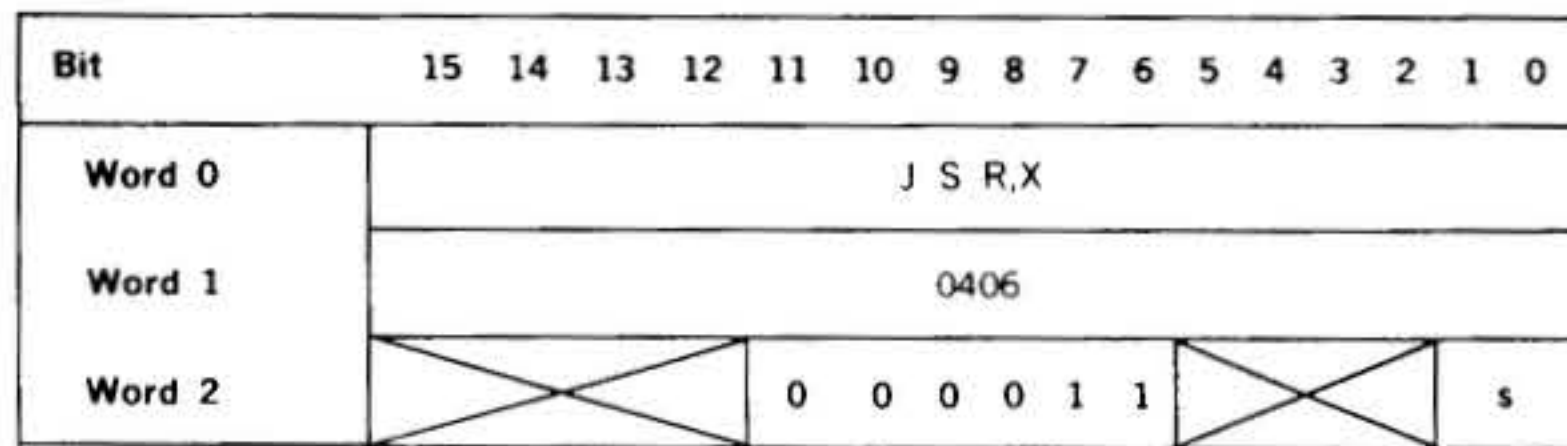
label **SUSPND** *susp*

where *susp* is 0 if the task is to be resumed by RESUME or 1 if the task is to be resumed by external interrupt, or 2 if the task is to be resumed by external interrupt or by IOC or I/O completion events via a simulated interrupt (i.e., TBEVNT word in task's TIDB is set to 1).

The FORTRAN calling sequence for this macro is

CALL SUSPND(susp)

Expansion: The *susp* flag is bit 0 of word 2 (s).



Example: Suspend a task from execution. Provide for resumption of the task by interrupt, which reactivates the task at the location following SUSPND.

SUSPND 1

The same request in FORTRAN is

CALL SUSPND (1)

2.1.3 RESUME Macro

This macro resumes a task suspended by the SUSPND macro. The RESUME macro has the general form

label **RESUME** 'xx','yy','zz'

where *xyyzz* is the name of the task being resumed, coded as in the SCHED macro (section 2.1.1).

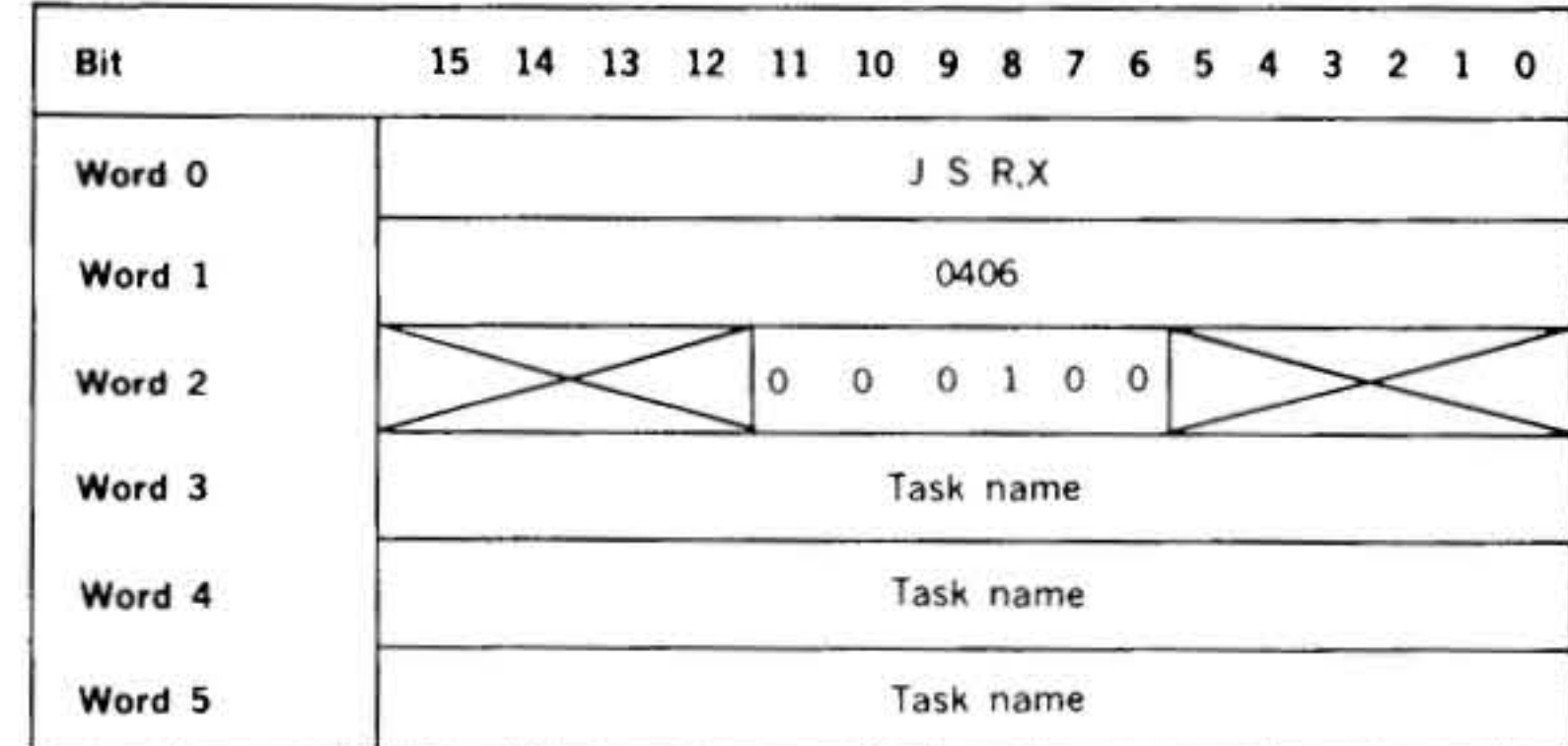
The RTE searches for the named task and activates it when found. The task will execute when it becomes the task with the highest active priority. If the priority of the specified task is higher than that of the task making the request, the specified task executes before the requesting task and immediately if it has the highest priority.

The FORTRAN calling sequence for this macro is

CALL RESUME(name)

where *name* is the three-word Hollerith array containing the name of the specified task.

Expansion: The task name is loaded two characters per word.



Example: Resume (reactivate) the task TSKTWO, which will execute when it becomes the task with the highest active priority.

RESUME 'TS', 'KT', 'WO'
(Control return)

Control returns to the requesting task when it becomes the task with the highest active priority. Control returns to the location following RESUME.

The same request in FORTRAN is

DIMENSION N1(3)
DATA N1(1),N1(2),N1(3)/2HTS,2HKT,2HWO/
CALL RESUME(N1)

or

CALL RESUME(6HTSKTWO)

2.1.4 DELAY Macro

This macro suspends the requesting task for the specified time, which is given in two increments. The first increment is the number of 5-millisecond periods, and the second, the number of minutes. The macro has the general form

label **DELAY** *milli,min,type*

where

milli is the number of 5-millisecond increments delay

min is the number of minutes delay

type is 0 (default value when the task is to be suspended for the specified delay, remain in memory, and automatically resume following the DELAY macro

1 when the task is to exit from the system, relinquishing memory, and

REAL-TIME EXECUTIVE SERVICES

after the specified delay, be automatically rescheduled and reloaded in a elapsed time mode, or

2 when the task is to resume automatically after the specified delay or upon receipt of an external interrupt whichever comes first, and automatically resume following the DELAY macro; or

3 when the task is to resume automatically after the specified delay, or upon receipt of an external interrupt, or completion of an I/O request initiated previously, whichever comes first, and automatically resume following the DELAY macro.

IOC resumes execution of the task by setting the TBEVNT word in the task's TIDB to 1.

The FORTRAN calling sequence for this macro is

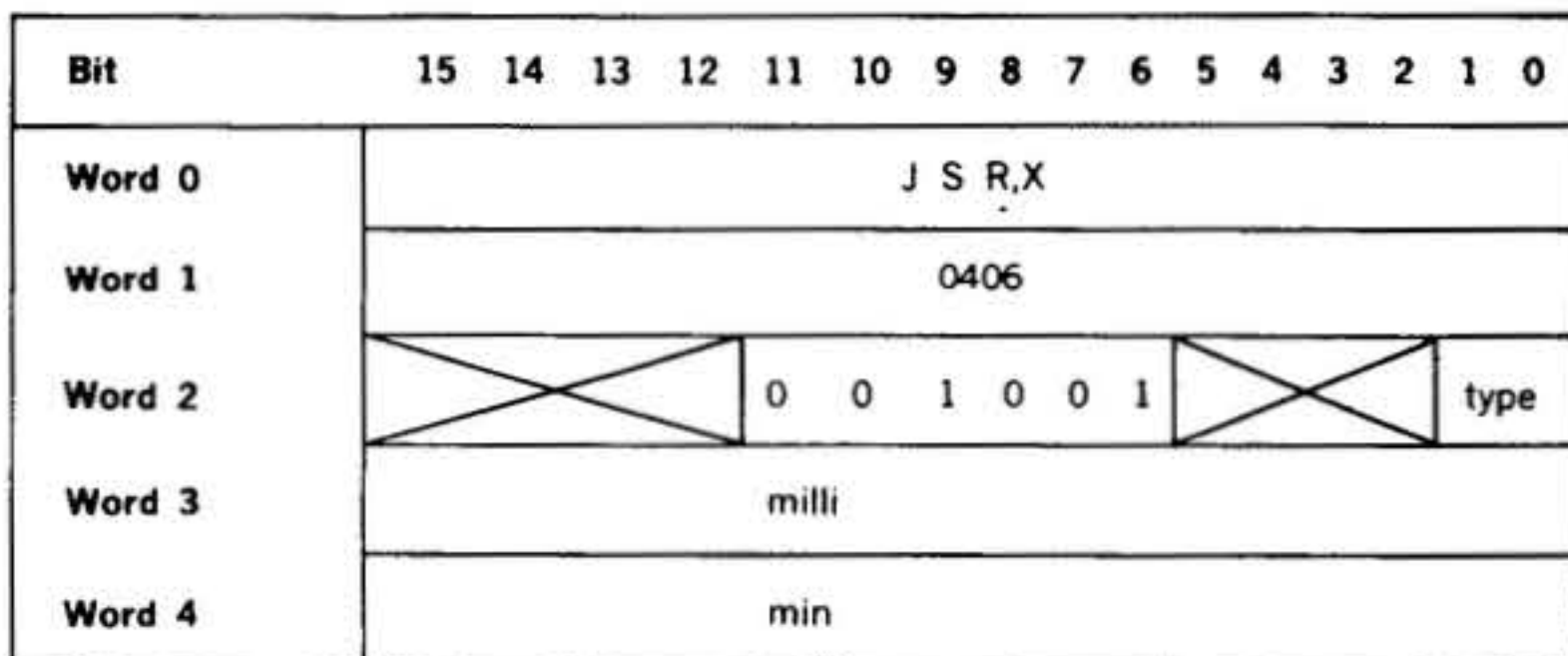
CALL DELAY(milli,min,type)

where the integer-mode parameters have the definitions given above.

The maximum value for either *milli* or *min* is 32767. Any such combination given the correct sum is a valid delay definition; e.g., for a 90-second delay, the values could be 6000 and 1, respectively, or 18000 and 0. After the specified delay, the task becomes active. When it becomes the highest-priority active task, it executes.

Note that the resolution of the clock is a user-specified variable having increments of 5 milliseconds. The time interval given in a DELAY macro must be equal to or greater than the resolution of the clock. The delay interval is stored in minute increments and real-time clock resolution increments.

Expansion: The *type* flag is bits 0 and 1 of word 2.



Examples: Assuming a 5-millisecond clock increment, delay the execution of a task for 90 seconds. At the end of this time, the task becomes active. When it becomes the highest-priority task, it executes.

DELAY 6000,1

Delay the execution of a task for 90 seconds or until receipt of an external interrupt, whichever comes first, at which

time the task becomes active. Such a technique can test devices that expect interrupts within the delay period.

DELAY 18000,0,2

Delay the execution of a task for 90 seconds, or until receipt of an external interrupt, or the completion of a previously initiated I/O request, whichever comes first.

DELAY 18000,0,3

2.1.5 LDELAY Macro

This macro is a type 1 DELAY macro with additional parameters to specify the logical unit from which the task is to be reloaded after the delay. The macro has the general form:

label LDELAY milli,min,lun,key

where

milli is the number of 5-millisecond increments delay

min is the number of minutes delay

lun is the number of the logical unit from which the task is to be loaded after the delay (DELAY tape 1 reloads from FL library)

key is the protection code for the logical unit

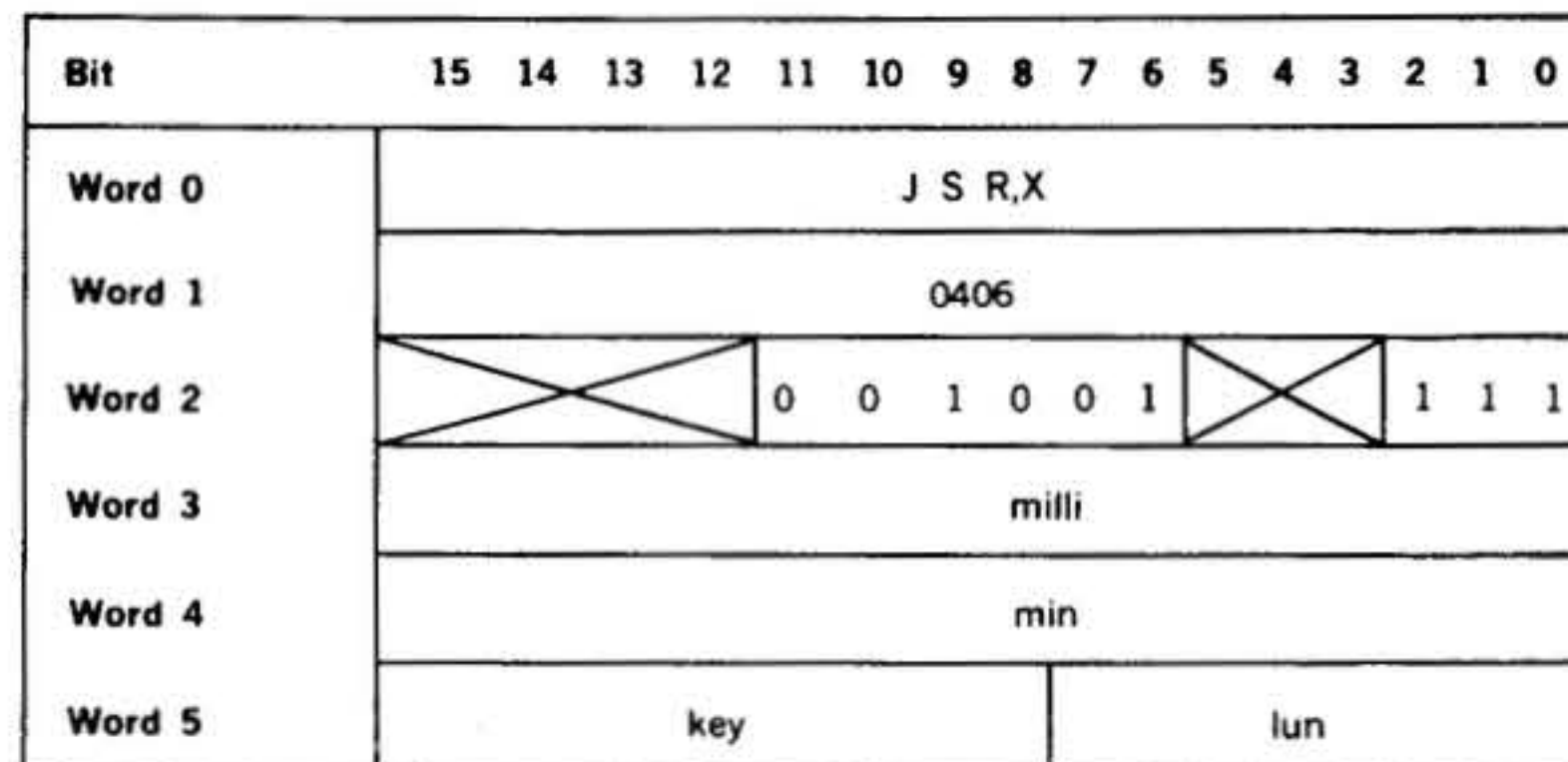
The FORTRAN calling sequence for this macro is

CALL LDELAY (milli,min,lun,key)

where the integer-mode parameters have the definitions given above.

Time is the same as specified for DELAY.

Expansion:



Example: Assuming a 5-millisecond clock increment, delay the execution of a task for 90 seconds. At the end of this time, the task becomes active. When it becomes the highest priority task, it is loaded from logical unit 128 which has protection key A, and executed.

LDELAY 6000,1,128,0301

2.1.6 PMSK (PIM Mask) Macro

This macro redefines the PIM (priority interrupt module) interrupt structure, i.e., enables and/or disables PIM interrupts. The macro has the general form

label **PMSK** *pim,mask,opt*

where

- pim** is the number (1 through 8) of the PIM being modified
- mask** indicates the changes to the mask, with the bits indicating the interrupt lines that are either to be enabled or disabled, depending on the value of *opt*, and with the other lines unchanged
- opt** is 0 (default value) if the set bits in **mask** indicate newly enabled interrupt lines, or 1 if the set bits in **mask** indicate newly disabled interrupt lines

The FORTRAN calling sequence for this macro is

CALL PMSK(pim,mask,opt)

where the integer-mode parameters have the definitions given above.

The eight bits of the mask correspond to the eight priority interrupt lines, with bit 0 corresponding to the highest-priority line.

VORTEX operates with all PIM lines enabled unless altered by a PMSK macro. Normal interrupt-processing allows all interrupts and does one of the following: a) posts (in the TIDB) the interrupt occurrence for later action if it is associated with a lower-priority task, or b) immediately suspends the interrupted task and schedules a new task if the interrupt is associated with a higher-priority task. PMSK provides control over this procedure.

Note: VORTEX (through system generation) initializes all undefined PIM locations to nullify spurious interrupts that may have been inadvertently enabled through the PMSK macro.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 0	J S R, X															
Word 1	0406															
Word 2	0 0 1 0 0 0 0															
Word 3	pim								mask							

Examples: Enable interrupt lines 3, 4, and 5 on PIM 2. Leave all other interrupt lines in the present states.

PMSK **2, 070**

The same request in FORTRAN is

CALL PMSK(2, 56, 0)

Disable the same lines.

PMSK **2, 070, 1**

2.1.7 TIME Macro

This macro loads the current time of day in the A and B registers with the B register containing the minute, and the A register the 5-millisecond, increments. The macro has the form

label **TIME**

The FORTRAN calling sequence for this macro is

CALL TIME(min,milli)

where **min** is the integer minutes to the 24 hour total, and **milli** is the seconds in 5-millisecond integer increments.

Expansion:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 0	J S R, X															
Word 1	0406															
Word 2	0 0 1 0 1 0															

Example: Load the current time of day in the A (5-millisecond increments) and B (1-minute increments) registers.

TIME
(Return with time in A and B registers)

2.1.8 OVLAY (Overlay) Macro

This macro loads and/or executes overlays within an overlay-structured task. It has the general form

label **OVLAY** *type,'xx','yy','zz'*

where

type is 0 (default value) for load and execute, or 1 for load and return following the request. If only load is specified, the load address is returned in the X register.

xyyzz is the name of the overlay segment, coded as in the SCHED macro (section 2.1.1)

REAL-TIME EXECUTIVE SERVICES

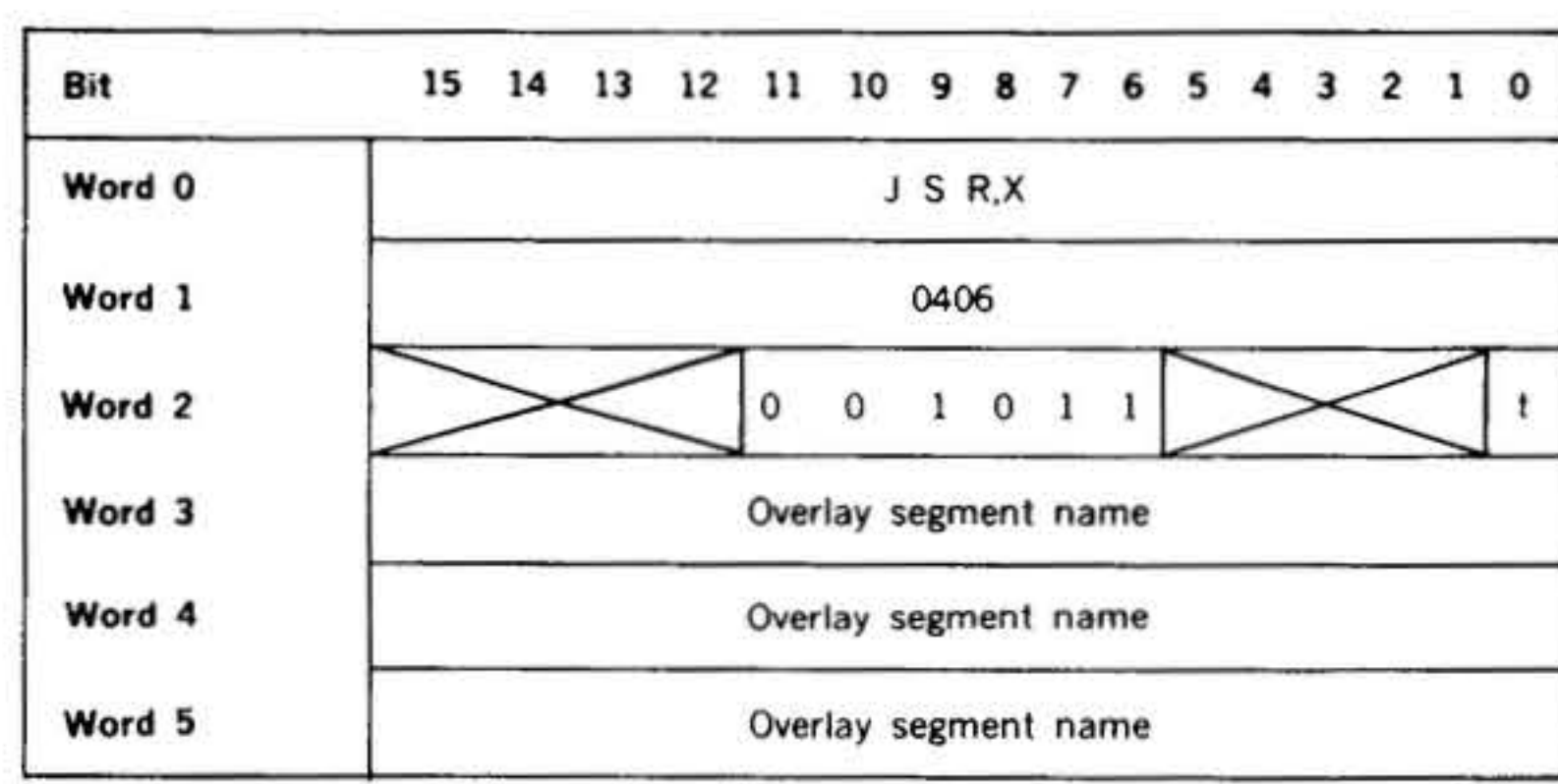
The FORTRAN calling sequence for this macro is

```
CALL OVLAY(type,reload,name)
```

where **type** is a constant or name whose value has the definition given above, **reload** is a constant or name with the value **zero** to load or non-zero to load only if not currently loaded, and **name** is a three-word Hollerith array containing the overlay segment name.

FORTRAN overlays must be subroutines if called by a FORTRAN call.

Expansion: The overlay segment name is loaded two characters per word. The **type** flag is bit 0 of word 2 (t).



When the load and execute mode is selected in the OVLAY macro RTE executes an equivalent of a root segment JSR instruction to enter the overlay segment. Therefore, the return address of the root segment is available to the overlay segment in the X register.

Example: Find, load, and execute overlay segment OVSG01 without return.

```
OVLAY 0, 'OV', 'SG', '01'
      (No return)
```

The same request in FORTRAN is

```
DIMENSION N1(3)
DATA N1(1),N1(2),N1(3)/2HOV,2HSG,2H01/
CALL OVLAY(0,0,N1)
```

or

```
CALL OVLAY(0,0,6HOVSG01)
```

External subprograms may be referenced by overlays. If a subprogram S is called in several overlays, and S is not in the main segment, each overlay will be built with a separate copy of S.

When using FORTRAN overlays containing I/O statements for RMD files defined by CALL V\$OPEN or CALL V\$OPNB statements (described in section 5.3.2), the main segment must contain an I/O statement so that the runtime I/O program (V\$FORTIO) will be loaded with the main segment. FCB arrays must be in the main segment or in common, so they are linked in memory and cannot be in any overlay.

2.1.9 ALOC (Allocate) Macro

This macro allocates space in a push-down (LIFO) stack of variable length for reentrant subroutines. The macro has the general form

```
label ALOC address
```

where **address** is the address of the reentrant subroutine to be executed.

The FORTRAN calling sequence for this macro is

```
EXTERNAL subr
CALL ALOC(subr)
```

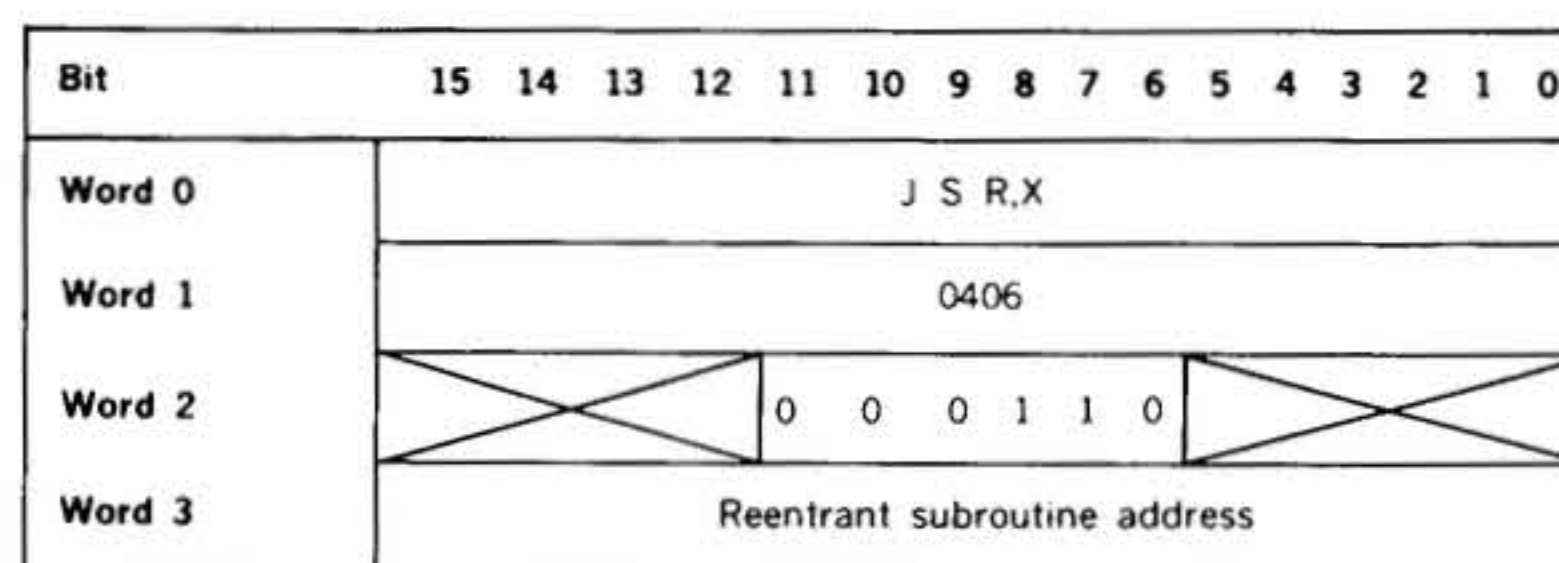
where **subr** is the name of the DAS MR assembly language subroutine.

The first location of the LIFO stack is V\$FLRS, and that of the current position in the stack is V\$CRS. The first word of the reentrant subroutine, whose address is specified in the general form of ALOC, contains the number of words to be allocated. If fewer than five words are specified, five words are allocated.

Control returns to the location following ALOC when a DEALOC macro (section 2.1.10) is executed in the called subroutine. Between ALOC and DEALOC, (1) subroutine cannot be suspended, (2) no IOC calls (section 3) can be made, and (3) no RTE service calls can be made.

Reentrant subroutines are normally included in the resident library at system-generation time so they can be concurrently accessed by more than one task. The maximum size of the push-down stack is also defined at system-generation time.

Expansion:



Reentrant subroutine: The reentrant subroutine called by ALOC contains, in entry location x, the number of words to be allocated. Execution begins at x + 1. The reentrant subroutine returns control to the calling task by use of a DEALOC macro.

The reentrant stack is used to store register contents and allocate temporary storage needed by the subroutine being called. The location V\$CRS contains a pointer to word 0 of the current allocation in the stack. By loading the value of the pointer into the X (or B) register, temporary storage cells can be referenced by an assembly language M field of 5,1 for the first cell; 6,1 for the second; etc.

A stack allocation generated by the ALOC macro has the format:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 0	Contents of the A register															
Word 1	Contents of the B register															
Word 2	Contents of the X register															
Word 3	ovfl	Contents of the P register														
Word 4	Stack-control pointer (for RTE use only)															
Word 5	For reentrant subroutine use (temporary storage)															
.
Word n
Words n+1 to n+5	V75 registers 3-7															

where ovfl is the overflow indicator bit.

The current contents of the A and B registers are stored in words 0 and 1 of the stack and are restored upon execution of the DEALOC macro. The same procedure is used with the setting of the overflow indicator bit in word 3 of the stack. The contents of word 2 (X register) point to the location of the reentrant subroutine to be executed following the setting up of the stack. The contents of word 3 (bits 14-0) point to the return location following ALOC.

Example: Allocate a stack of six words. Provide for deallocation and returning of control to the location following ALOC.

```

EXT      SUB 1
ALOC     SUB 1
          (Return Control)
.
.
SUB 1    NAME  SUB 1
          DATA 6
          .
          .
          .
          DEALOC
          END
    
```

Each time SUB1 is called, six words are reserved in the reentrant stack. Each time the reentrant subroutine makes a DEALOC request (section 2.1.10), six words are deallocated from the reentrant stack. If the calling task uses the V75 registers, 11 words are allocated/deallocated.

2.1.10 DEALOC (Deallocate) Macro

This macro deallocates the current reentrant stack, restores the contents of the A and B (and V75) registers and the setting of the overflow indicator to the requesting

task, and returns control to the location specified in word 3 (P register value) of the reentrant stack (section 2.1.9). The macro has the form

```
label      DEALOC
```

Expansion:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 0	J S R,X															
Word 1	0406															
Word 2	0 0 0 1 1 1															

Example: Release the current reentrant stack, restore the contents of the volatile registers and the setting of the overflow indicator and return control to the location specified in word 3 of the stack.

```

.
.
.      (Reentrant subroutine)
DEALOC
END
    
```

2.1.11 EXIT Macro

This macro is used by a task to signal completion of that task. The requesting task is terminated upon completion of its I/O. The macro has the form

```
label      EXIT
```

The FORTRAN calling sequence (no parameters specified) is

```
CALL      EXIT
```

If the task making the EXIT is in unprotected background memory, the macro schedules the job-control processor (JCP) task (section 4).

Expansion:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 0	J S R,X															
Word 1	0406															
Word 2	0 0 0 0 1 0															

Example: Exit from a task. The task making the EXIT call is terminated upon completion of its I/O requests.

```

.
.
.      (No return)
EXIT
    
```

2.1.12 ABORT Macro

This macro aborts a task. Active I/O requests are completed, but pending I/O requests are dequeued. The macro has the general form

```
label      ABORT      'xx','yy','zz'
```

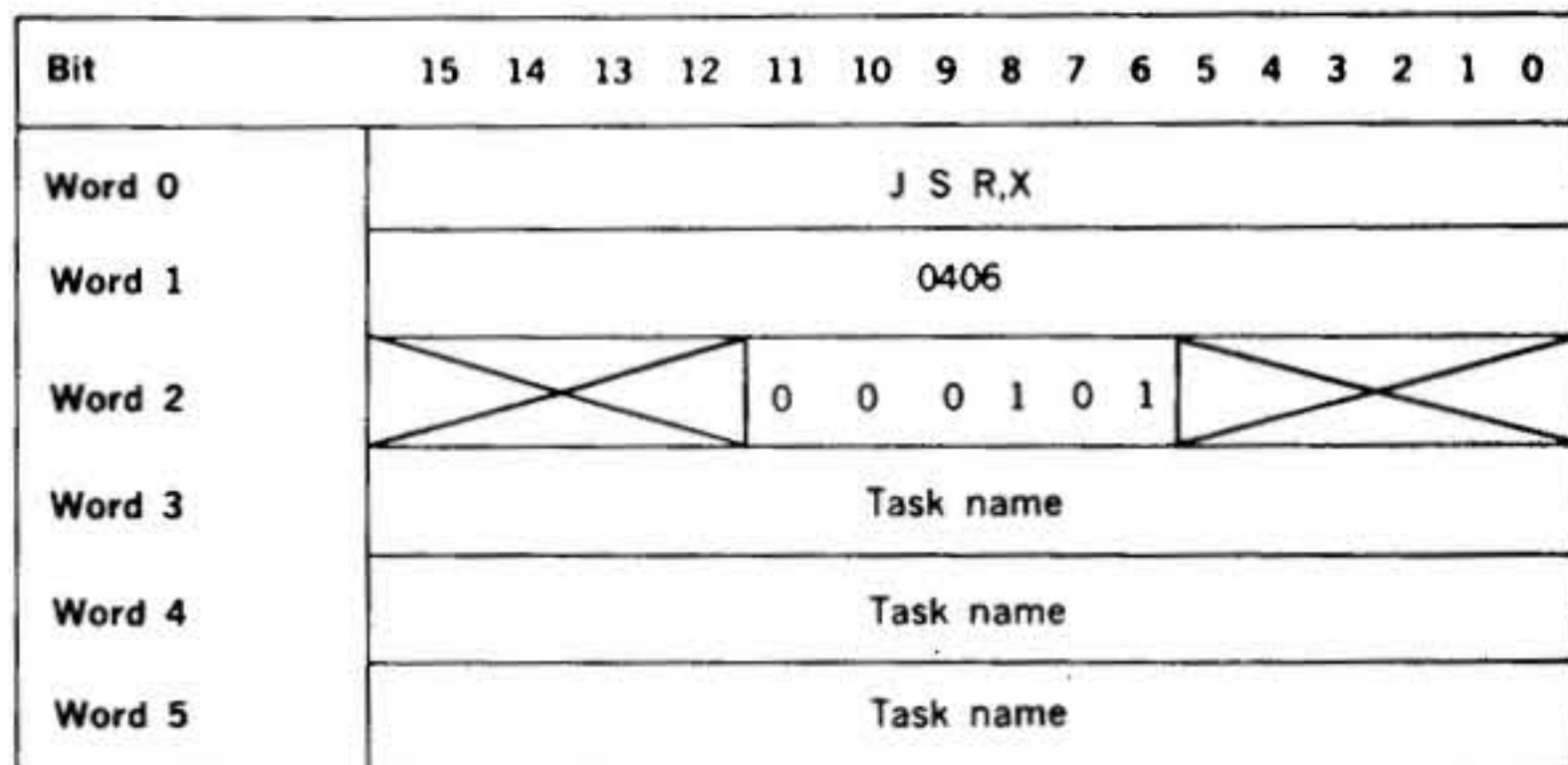
where *xyyzz* is the name of the task being aborted, coded as in the SCHED macro (section 2.1.1).

The FORTRAN calling sequence for this macro is

```
CALL ABORT(name)
```

where *name* is the three-word Hollerith array containing the name of the task being aborted.

Expansion: The task name is loaded two characters per word.



Example: Abort the task TSK and return control to the location following ABORT.

```

.
.
.
ABORT  'TS','K',' '
.      (Control return)
.
.

```

The same request in FORTRAN is

```
DIMENSION N1(3)
DATA N1(1),N1(2),N1(3)/2HTS,2HK ,2H /
CALL ABORT(N1)
```

or

```
CALL ABORT(6HTSK )
```

2.1.13 IOLINK (I/O Linkage) Macro

This macro enables background tasks to pass buffer address and buffer size parameters to the system background global FCBs. It has the general form

```
label      IOLINK      lungsd,bufloc,bufsiz
```

where

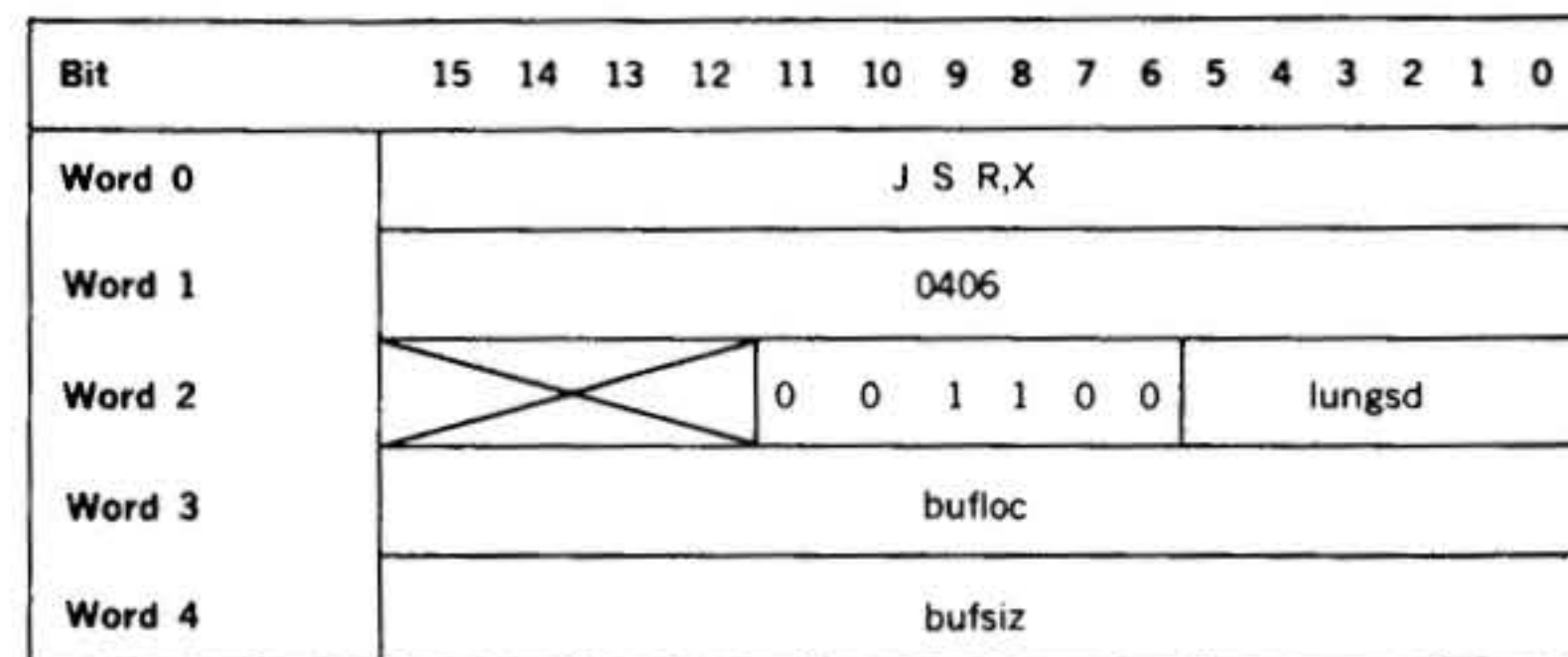
lungsd is the logical unit number of the global system device

bufloc is the address of the input/output buffer

bufsiz is the size of the buffer (maximum and default value: 120)

```
ABORT  'TS',' ',' '
```

Global file control blocks: There are eight global FCBs (section 3.5.11) in the VORTEX system reserved for background use. System background and user programs can reference these global FCBs. JCP directive /PFILE (section 4.2.11) stores the protection code and file name in the corresponding FCB before opening/rewinding the logical unit. The IOLINK service request passes the buffer address and the size of the record to the corresponding logical-unit FCB. The names of the global FCBs are *SIFCB*, *PIFCB*, *POFCB*, *SSFCB*, *BIFCB*, *BOFCB*, *GOFCB*, and *LOFCB*, where the first two letters of the name indicate the logical unit.



Example: Pass the address and size specifications of a 40-word buffer at address BUF to the PI global FCB.

```

PI      EQU      4
EXT     PIFCB
.      (PI logical-unit number 4)
.
.
IOLINK  PI, BUF, 40
READ   PIFCB, P1, 0, 1
.      (Read 40 ASCII words
.      from PI)
.
.
BUF     BSS      40
END

```

If the PI file is on an RMD, reassign the PI to the proper RMD partition, and then position the PI file using JCP directive /PFILE.

2.1.14 PASS Macro

This macro fetches map 0 data into the user map. It has the general form

label **PASS** **count,from,to**

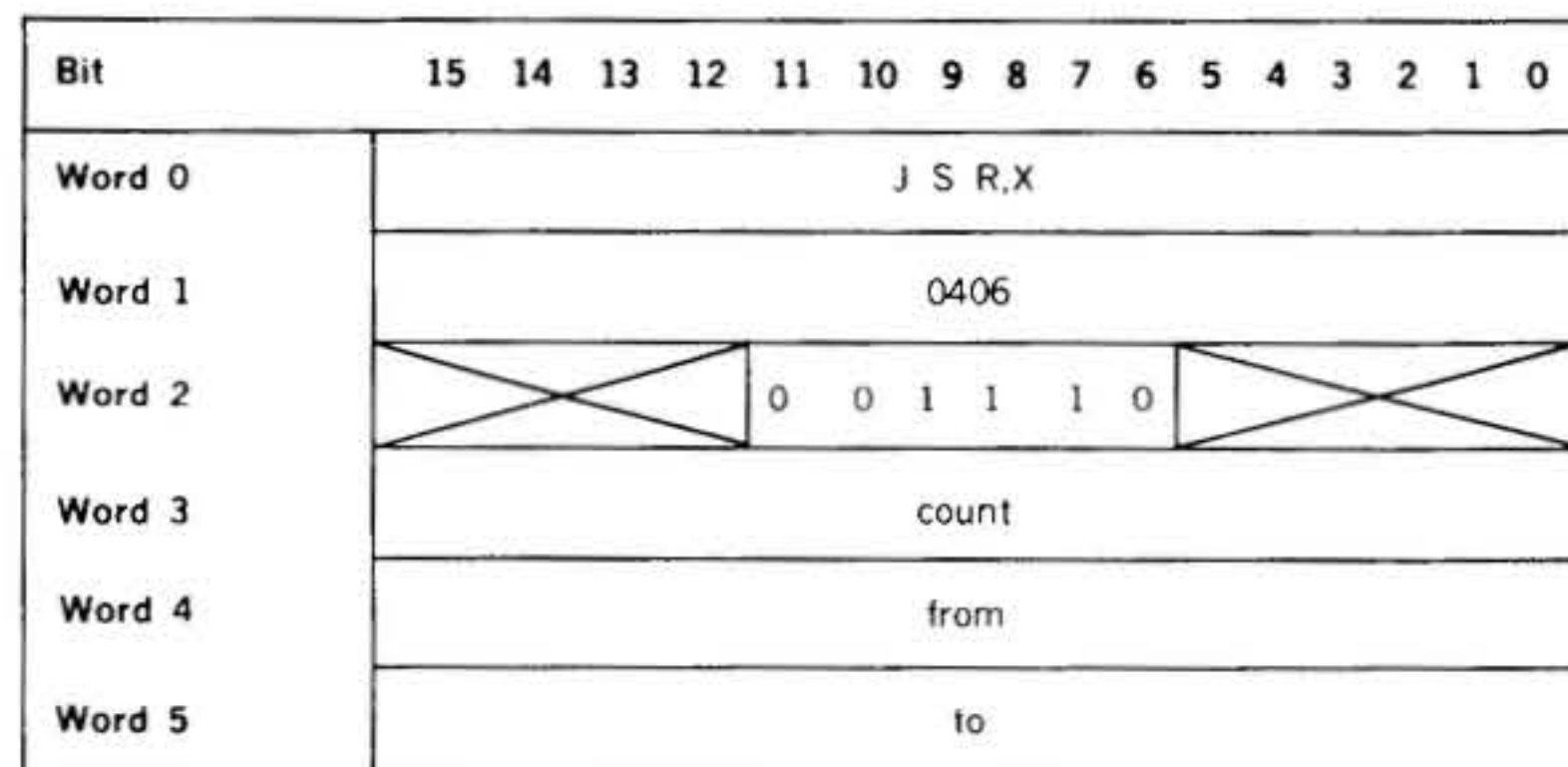
where

count is the number of words to be passed
from is the map 0 fetch address
to is the user map store address

The FORTRAN calling sequence for this macro is:

CALL PASS(count,from,to)

Expansion:



If a negative or zero word count is specified, an EX16 error message is posted and the task aborted. Any memory protection violation will result in an EX20-EX25 error message.

Example: Pass the TIDB information into PBUF

```
V$CTL EQU 0300

      LDA V$CTL      (Get TIDB address)
      STA P1+4
P1    PASS 29,*,PBUF
      .
      .
      .
PBUF  BSS 29
      END
```

2.1.15 TBEVNT (Set or Fetch TBEVNT) Macro

This macro fetches or sets the requesting task's event word, TBEVNT, as well as alters other TIDB entries. It should be noted here that most changes to TIDB entries

could cause irrecoverable errors, so TBEVNT should be used with caution.

The macro has the general form:

label **TBEVNT** **value, disp, c/s**

where:

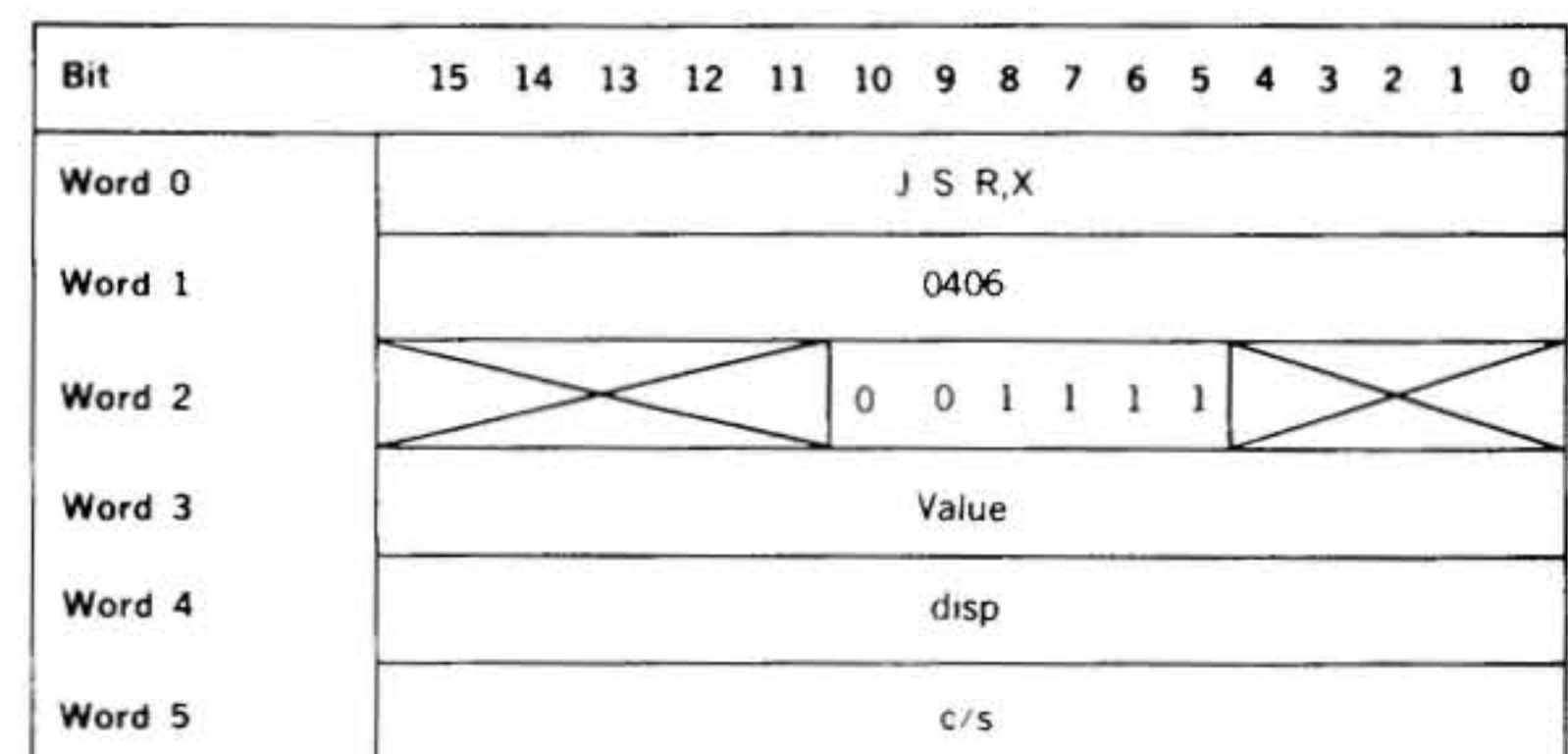
value is 0177777 (mask)
disp is the TIDB word ordinal number (displacement) to be altered
c/s is the clear/set indication

Explanation:

If *disp* = 0, the following is done according to the value parameter. If value is 0-0177776 it is set into the requesting task's TIDB event word, TBEVNT. If the **value** is 0177777, the request will fetch TBEVNT from the requester's TIDB and return with the A register set to the TBEVNT content. (See section 14 for information on use of the event word.)

If *disp* ≠ 0, the action depends on the *c/s* indication. When *c/s* = 1 (i.e., set), the corresponding TIDB (word number displacement) bits are set according to the ones in the mask value.

When *c/s* = 0 (i.e., reset), the corresponding TIDB (word number displacement) are reset according to the zero bits in the mask value.



Default values: *disp* = 0 *c/s* = 0

Example: Reset TBPL (word 2 of TIDB) bit 8 and then set it again.

TBEVNT 0177377, 2, 0 (reset)
TBEVNT 0400, 2, 1 (set)

2.1.16 ALOCPG (Allocate Memory Pages) Macro

This macro allocates in physical pages from the pool of available pages to logical pages starting at the specified logical address, modulo 01000. The logical pages to be mapped must not have been previously assigned. The logical pages are assigned as read/write access mode. If an

REAL-TIME EXECUTIVE SERVICES

error condition occurs, an EX27 error message is output and the task resumes operation at the specified reject address. The general form is

label **ALOCPG** *n*,logical addr,reject addr

where

n is the number of pages to be allocated

logical addr is the logical address, modulo 01000, where the *n* pages are allocated. If the logical address is negative (1's complement) the address is assumed to be in map 0. If the logical address is positive, the address is assumed to be the requestor's map (priority tasks cannot allocate memory in map 0)

reject addr is the error return address when a task exits or is aborted all ALOEPG pages are automatically deallocated.

Expansion:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 0	J S R,X															
Word 1	0406															
Word 2	X		0		1		0		0		0		0		X	
Word 3	n															
Word 4	logical addr															
Word 5	reject addr															

Example: Allocate 4 pages of memory to the requesting task's virtual memory starting at logical address 06000. If error, go to ERR01.

```

        ALOCPG    4,06000,ERR01
        .
        .
        .
ERR01  STA      (Error routine)
    
```

2.1.17 DEALPG (Deallocate Memory Pages) Macro

This macro deallocates *n* pages of memory starting at the specified logical address, modulo 01000. The deallocated logical pages are set to unassigned access mode. Deallocated physical pages, which were not assigned by MAPIN requests, are returned to the pool of available pages. Specifying logical page 0 or non-read/write page results in

EX30 error message to be posted and the task's operation resumed at the reject address. The general form is

label **DEALPG** *n*,logical addr,reject addr

where

n is the number of pages to be deallocated

logical addr is the logical address, modulo 01000, where the *n* pages are deallocated if negative, 1's complement of map 0 logical address (illegal for priority 0 tasks)

reject addr is the error return address

Expansion:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 0	J S R,X															
Word 1	0406															
Word 2	X		0		1		0		0		0		1		X	
Word 3	n															
Word 4	logical addr															
Word 5	reject addr															

Example: Deallocate 4 pages of memory in the requesting task's virtual memory starting at logical address 06000. If error, go to ERR02.

```

        .
        DEALPG   4,06000,ERR02
        .
        .
ERR02  LDA      (Error routine)
        .
        .
    
```

2.1.18 MAPIN (Map-In Specified Physical Pages of Memory) Macro

This macro allows the requestor to specify physical pages of memory to be assigned to the requestor's logical memory starting at the specified logical address, modulo 01000. Priority 0 tasks are not permitted to execute the MAPIN request. The specified logical pages to be mapped must not have been previously assigned except by a previous MAPIN request. All logical pages are set to the read/write access mode. Pages mapped in by this request do not effect the pool of available pages. The requested physical pages cannot include page 0 nor any of the pages assigned to the nucleus program module. Any error condition causes EX31

to be output and the task resumed at the reject address. The general form is

label **MAPIN** *n*,log add, {*buff* }_{pg} ,reject

where

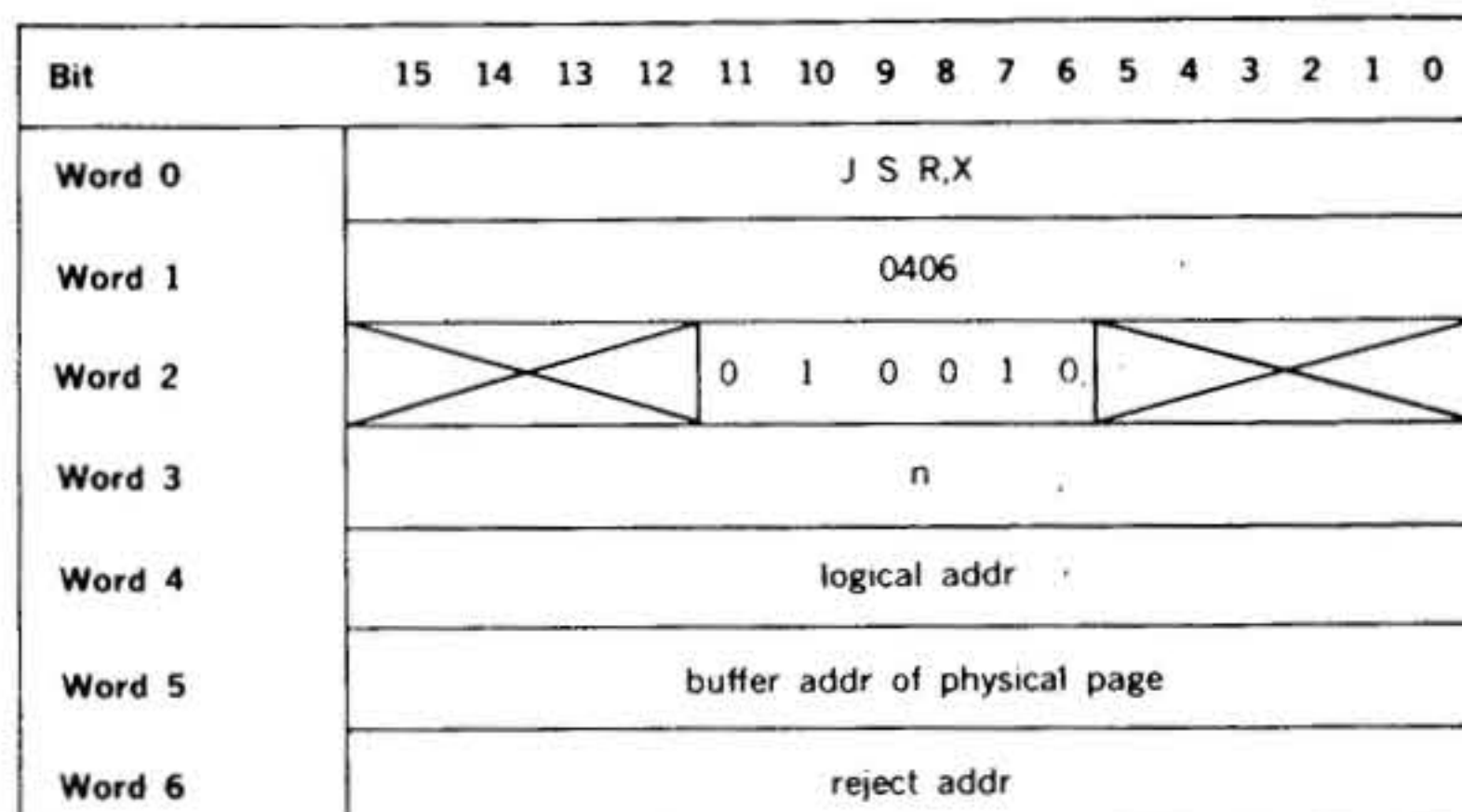
n if positive is the number of pages of memory to be allocated. If negative, it is assumed to be the one's complement of the number of pages to be allocated

log add is the requestor's logical address, modulo 01000, where the specified physical pages are to be mapped

{*buff* }_{pg} is the actual physical page number to be mapped, or the address of the buffer containing the physical page numbers. If *n* is positive and this parameter is positive and less than 2048, this parameter is interpreted as a physical page number. If *n* is greater than 1, all physical pages assigned will be consecutive. If *n* is negative and this parameter is greater than 1023, this parameter is assumed to be a map 0 buffer address, e.g., TIDB map image address. If *n* is positive and this parameter is negative, this parameter is assumed to be the one's complement of the buffer address within the requestor's logical space, which contains the physical page numbers

reject is the error return address

Expansion:



Example: Copy the same 2 physical pages as used by task A, logical address ABUF, into task B's logical memory at logical address BBUF. Task A scheduled task B, passing task A's TIDB address to task B.

```

TASK A
NAME      TASKA
TITLE     TASKA
FL        EQU      106
KEY       EQU      0306
V$CTL    EQU      0300
.
LDBI     ABUF      (B = Buffer Address)
LDA      V$CTL     (A = Task A's TIDB)
SCHED    2,0,FL,KEY,'TA','SK','B'
.
.
.
ABUF     BSS      02000
END

TASK B
NAME      TASKB
TITLE     TASKB
TBMING   EQU      27
TASKB    STA      P1+4      (Set task A's TIDB addr)
P1       PASS     29,* ,PBUF (Pass task A's TIDB
                              into PBUF)
.
.
.
TBA      (B = ABUF addr)
TZB
LLSR     9        (A = Page number, B =
                  offset in page)
ADDE     TBMING+PBUF
STA      M1+5    (Add task A's map image
                  addr)
M1       MAPIN   2,BBUF,* ,RA (MAPIN same 2 physical
                              pages at BBUF shared by
                              task A at ABUF)
.
.
.
TBA      (B = Offset into page)
LSRA     7
ADDI     BBUF    (Add BBUF addr)
TAB      (B = Start of ABUF)
.
.
PBUF     BSS      29        (TIDB buffer)
BBUF     BSS      TASKB-#+512 (Set to page boundary)
.
.
EQU      *          (Assume task B < 512
                    words)
END
    
```

2.1.19 PAGNUM (Identify Physical Page Number) Macro

This macro allows the requestor to identify the physical page number assigned to a specified logical address. If an unassigned logical address is specified, return is to the requestor with the A register = 0. Otherwise, return is made with the A register set to the physical page number and the B register set to the task's map image address for the specified logical address. The general form is

label **PAGNUM** logical addr

where **logical addr** is the address where the identity of the assigned physical page is requested.

Expansion:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 0	J S R, X															
Word 1	0406															
Word 2	X				0 1 0 0 1 1				X							
Word 3	logical addr															

Example: Identify the physical page assigned to PBUF.

```

        .
        LDAI    PBUF    (Get RBUF addr)
        STA    P1+3
P 1     PAGNUM *      (Identify physical page)
        .
        .
PBUF   BSS      100
    
```

2.1.20 RECOV (Error Recovery) Macro

This macro allows the requestor to pass the address of a recovery routine to VORTEX. Control will be passed to the routine if VORTEX attempts to terminate the task abnormally. Return from the user's recovery routine should be made via the EXIT macro. The macro has the general form.

```

        LABEL    RECOV    ADDR
    
```

where

ADDR is the address of the error recovery routine

The recovery address is kept in TBENTY of the user's TIDB. Repeated calls to RECOV are allowed but the last specified recovery address is always the address used. Note that if an abend occurs and control is passed back to the user, registers are not preserved. When using RECOV with a driver, the user must be aware of, and handle the fact that IOC uses TBENTY when activating the driver. Thus the recovery routine becomes the driver initialization routine.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 0	JRS, X															
Word 1	0406															
Word 2	X				1 1 0 0 1				X							
Word 3	Address of error handling routine															

Example:

```

        BEGIN    RECOV    ERROR
        .
        .
        ERROR    EQU      (error recovery routine)
    
```

2.1.21 AFAULT (Arithmetic Fault Setup) Macro

This paragraph applies to V77-800 users only.

The AFAULT macro allows V77-800 users to specify the action to be taken when an arithmetic fault occurs.

Expansion:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 1	JSR, X															
Word 2	0406															
Word 3	X				010011				X							
Word 4	FLAGS															
Word 5	MASK															
Word 6	EXITS															

Word 4: FLAGS:

- Bit 0(OV) = 0: Reset overflow interrupt enable.
1: Set overflow interrupt enable.
- Bit 1(UN) = 0: Reset underflow interrupt enable.
1: Set underflow interrupt enable.
- Bit 2(AOV) = 0: Don't abort on overflow interrupt.
1: Abort on overflow interrupt.
- Bit 3(AUN) = 0: Don't abort on underflow interrupt.
1: Abort on underflow interrupt.
- Bit 4(DOM) = 0: Don't disable overflow error message.
1: Disable overflow error message.
- Bit 5(DUM) = 0: Don't disable underflow error message.
1: Disable underflow error message.

Word 5: MASK: This is a bit mask. 1-bits indicate which of the above bits are to be modified

Word 6: EXITS: This is a pointer to a two word block

- EXITS(0) Address of overflow processor. Null if AOV is set. Continue in line if zero.
- EXITS(1) Address of underflow processor. Null if AUN is set. Continue in line if zero.

2.1.22 SRFAULT (Save/Restore Arithmetic Fault Status) Macro

This paragraph applies to V77-800 users only.

The SRFAULT macro saves and restores arithmetic fault interrupt enable status. Its function is to allow DASMR subroutines which are part of an executable FORTRAN program (i.e., with a FORTRAN main program) to use the V70 Overflow Indicator (e.g., the Run-Time I/O module) without getting trapped out by an arithmetic fault interrupt. Such a subroutine would issue a SAVE request on entry which would disable overflow fault interrupts, and RESTORE request on exit.

The two requests are distinguished by the contents of the A-register:

A = 0: Save arithmetic fault interrupt enable status.

A ≠ 0: Restore arithmetic fault interrupt enable status.

Expansion:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 0	JSR, X															
Word 1	0406															
Word 2	X				0 1 0 0 1				X							

2.1.23 SETPAR (Set Parity Enable) Macro

This paragraph applies to V77-800 systems only.

The SETPAR macro allows V77-800 users to selectively enable or disable memory parity during execution of the requesting task.

Expansion:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 0	JRS, X															
Word 1	0406															
Word 2	X				1 1 0 0 1				X							
Word 3	mask															

where
mask

is the parity enable mask. The values and corresponding functions of mask are

- 00 disable all parity
- 01 enable double bit parity error only
- 03 enable all parity

Example:

Disable single bit parity.

```

SETPAR,1      disable single bit parity
EXC DISPIM    disable PIMS
EXC DISCLK    disable clock
    
```

non-interruptible code

```

EXC ENAPIM    Restore parity upon returning from
               the first clock or PIM interrupt.
EXC ENACKL    
```

2.1.24 SELECT(Select a Partition) Macro

This macro allows the user to change, under program control, the partition to which an assignable logical unit number is assigned. The logical unit number can only be reassigned to another partition on the disk unit to which the logical unit is currently assigned (i.e., the same drive). Any defined partition on the disk unit can be selected or the logical unit number can be set to indicate a "special" logical unit by specifying a partition number zero; this latter choice is only possible if the disk unit has been defined during system generation to be a special logical unit (see section 3.1.1.). The specified logical unit must be currently assigned to a disk partition (or to the disk unit in the case of a special logical unit). For this reassignment of logical unit number, the logical unit table assignability rules apply and the selected partition must be defined during system generation. Any error condition causes the EX37 message to be output. The task is then resumed at the reject address, described below. The general form of this macro is:

label **SELECT** lun,part,reject addr

where:

- lun** is the logical unit number.
- part** is the desired partition number, or 0 if the lun is to be made a special logical unit
- reject addr** is the error return address.

REAL-TIME EXECUTIVE SERVICES

Expansion:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 0	JSR,X															
Word 1	0406															
Word 2	X				0 1 1 0 1 0				X							
Word 3	logical unit number															
Word 4	partition number															
Word 5	reject address															

Example: Logical unit 50 is currently assigned to D11(07). Change the assignment of logical unit 50 to D11(09). The address of the appropriate error routine procedure is "ERRADR".

SELECT 50,9,ERRADR

2.2 RTE SYSTEM FLOW

The RTE component loads and executes a task depending on the category of that task:

Executive Mode Tasks

These are the VORTEX system and user tasks designated during system generation (SGEN) to be resident (excludes tasks specified on SGEN TSK directives). The RTE, IOC, I/O drivers, and common interrupt processors are examples of system executive mode tasks (map 0). OPCOM is loaded into and executed from page 1 of map 0. All other non-resident tasks are defined to be user mode tasks.

User Mode Tasks

- a. Background tasks with a priority of zero: Tasks that are executed via a DASMR or FORTRAN load-and-go operation and those that are loaded and executed from the BL library with a JCP/LOAD directive are in this group.

These tasks are loaded with the first page of physical memory (0-0777) designated as read operand only. The literal and indirect pointer pool is loaded in the first page at locations 0500-0777. The remainder of the background task is loaded in whatever physical pages are available at the time the task is loaded. These pages are designated as read/write access. If a nucleus module is referenced, that module is mapped as read operand only. All other pages in the logical memory are designated as unassigned. The RTE component designates an available map key (1-15) to the background task and sets the appropriate mapping registers to reflect the task's logical memory.

- b. Background priority 1 tasks: System tasks such as the Job-Control Processor (JCP), Input/Output Utility (IOUTIL), System Maintenance (SMAN), Source Editor (SEdit), DAS MR, FORTRAN, RPG IV, MIDAS, MICSIM, and File Maintenance (FMAIN) require full access to the nucleus (to modify tables or utilize the global FCBs). These tasks are loaded with the required nucleus modules designated as read/write access mode permitting fetches and stores into these areas. The literal and indirect pointer pool is loaded in the first page at locations 0500-0777. The task is loaded starting at logical address 01000.

- c. Foreground tasks: Page 0 is mapped read operand only for a foreground task. Nucleus modules (including blank common) referenced by foreground tasks, are mapped in the read/write access mode (see figure 2-1). The maximum logical memory space available to a foreground task is thus dependent on the number and type of nucleus module referenced by the task. The pages within the logical memory not utilized are mapped as unassigned. All foreground tasks are loaded at logical memory address 01000.

- d. Read-only pages: During the creation of a load module by LMGEM, the user has the capability to specify pages within the load module as read-only pages. The designated read-only pages are indicated in the pseudo TIDB block. When the task is loaded, the RTE component will designate those pages in the task's logical memory as read-only pages.

2.3 TASK LIMITATIONS AND DIFFERENCES

In the VORTEX environment, background and foreground tasks either share or are differentiated by the following characteristics:

- a. A background task has a priority level of 0 or 1. A foreground task can have a priority of 2 through 31.
- b. Only one background task can be executed at a time. Excluding the RTE, IOC, and I/O driver tasks, a maximum of 15 (user mode of 1 through 15) user foreground tasks can be in operation concurrently, provided physical memory size is adequate. See section 2.5 for a description of checkpointing of tasks.
- c. A background task can be checkpointed and its operation pre-empted by a foreground task. A foreground program memory space is not checkpointed (see section 2.5).
- d. A background task can have literals and indirect pointers, a foreground task cannot.

- e. All tasks whether background or foreground have individual task protection.
- f. If allocable memory is not available to load a background task, the RTE component will output an error message (EX05) and abort the operation. If a foreground task is to be loaded and allocatable memory is not available, the RTE component will reattempt the load when memory becomes available.
- g. Background level 0 or 1 task can schedule a task from the background library only.
- h. Foreground tasks can utilize foreground blank common. Background tasks cannot.
- i. Background level 0 tasks have restricted RTE requests (see table 2-1). Foreground tasks have no restriction on RTE service requests.

	Background Priority	Priority of Task Background Priority	Foreground Priorities
Nucleus Modules	0	1	2-31
Foreground Blank COMMON Nucleus Module	UN	UN	RW
Global FCT Nucleus Module	ROP	RW	UN
System Table Nucleus Module	ROP	RW	RW
System Resident Tasks Nucleus Module	UN	UN	UN
Page 0 System Constants	ROP	RW ROP*	ROP

Key: RW Read-Write Access Mode
 ROP Read Operand Only Access Mode
 RO Read-Only Access Mode
 UN Unassigned Access Mode
 * for micro-VORTEX only

Note: Since the upper three modules are defined contiguously, without regard to page boundaries, and since maps are full pages, a map for any of these modules may include a partial page of an adjoining module, with the same access mode.

Figure 2-1. Matrix of Nucleus Module Access Mode

2.4 ABORT PROCEDURE

Whenever a task is aborted, all currently active I/O operations are allowed to complete. All I/O requests that are threaded (queued, or waiting to be activated) are not activated. Upon completion of all active I/O operations and after all pending requests are dethreaded, the aborted task is released.

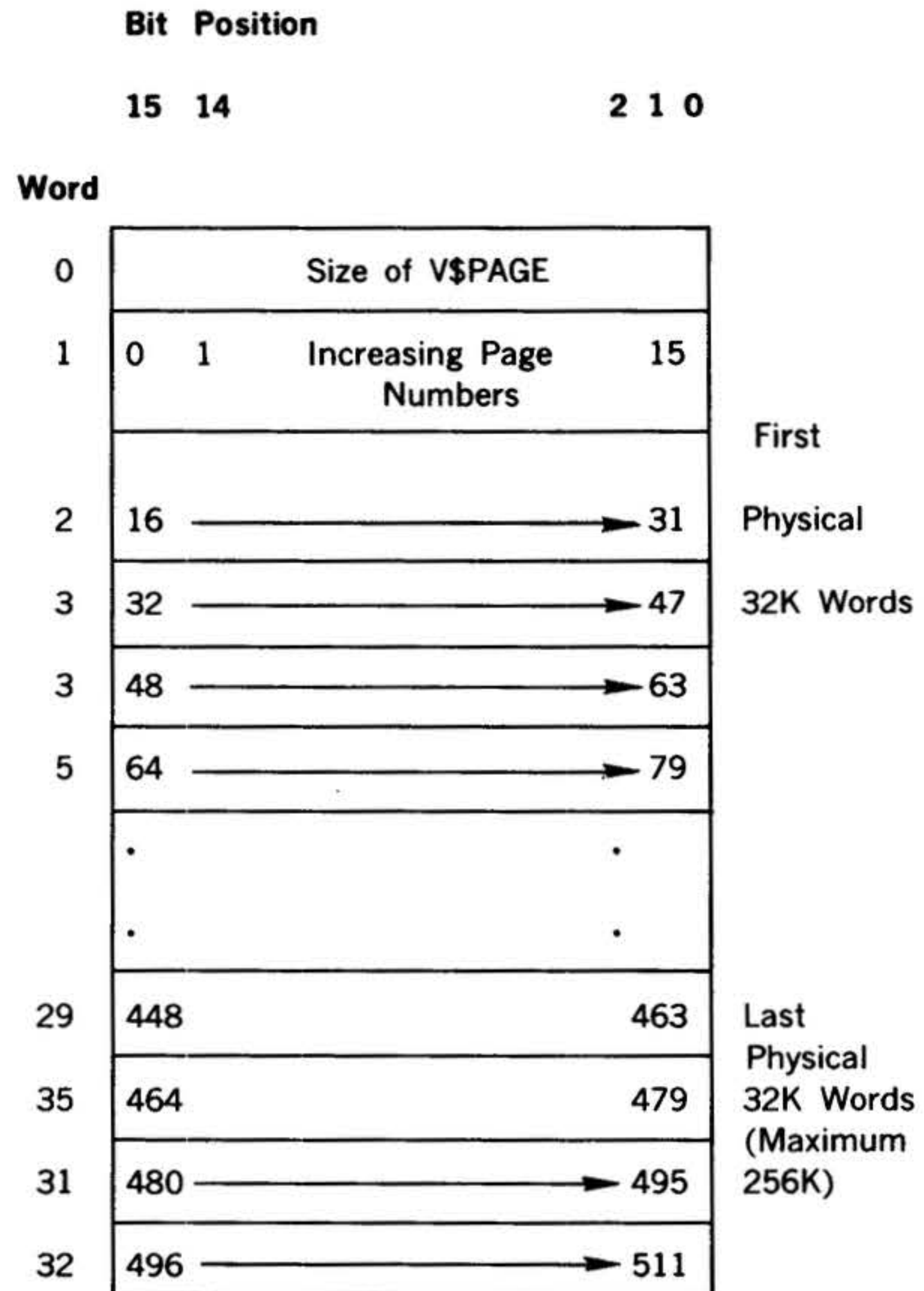
2.5 CHECKPOINTING OF TASKS

A background task's memory space and/or assigned map may be checkpointed for use by a foreground task. The background task is restarted when memory space and/or a map key becomes available.

A foreground task may be checkpointed by a higher priority foreground task. It may also be checkpointed by a lower priority task depending on the value of TBST bit 8. The default value of this bit is on (=1) i.e., "may be checkpointed by a lower priority task". In order to turn this bit off, a usage of TBEVNT (2.1.15) is recommended. The foreground task's memory space is never checkpointed. More than one foreground task's map may be checkpointed.

2.6 PAGE ALLOCATION SCHEME

The page allocation routine scans the page bit mask table, V\$PAGE (figure 2-2) to determine the allocable physical pages. To expedite the process, the allocation routine first checks the page 0 system word V\$NPAG to find the total number of allocable pages in V\$PAGE. If the required number of pages exceeds V\$NPAG, scanning of V\$PAGE is not attempted. The allocation routine scans V\$PAGE starting with the word number specified in V\$LPP (page 0 system pointer). The system generation program initially sets V\$LPP to the deallocated pages. In micro-VORTEX, V\$PAGE is scanned from the top instead of the bottom.



Corresponding Page Bit Positions:

1 = Page is allocatable

0 = Page is unallocatable

V\$PGT Address of V\$PAGE

V\$LPP 0, Pointer to last word tested

V\$NPAG Number of available pages

Figure 2-2. V\$PAGE, Page Allocation Table

The size of V\$PAGE is determined by SGEN based on the physical memory size specified on the MRY directive.

SECTION 3 INPUT/OUTPUT CONTROL

The VORTEX **input/output-control component (IOC)** processes all requests for I/O to be performed on peripheral devices. The IOC comprises an I/O-request processor, a find-next-request processor, an I/O-error processor, and I/O drivers. The IOC thus provides a common I/O system for the overall VORTEX operating system and eliminates the programmer's need to understand the computer hardware.

All I/O with remote devices connected through the Data Communications Multiplexor (DCM) uses the VORTEX Telecommunications Access Method (VTAM). VTAM interfaces with IOC. Use of VTAM is described in the VTAM Reference Manual.

The contents of the volatile A and B registers and the setting of the overflow indicator are saved during execution of any IOC macro. After completion of the macro, these data are returned. The contents of the X register are lost.

If a physical-device failure occurs, the I/O drivers perform error recovery as applicable. Where automatic error recovery is possible, the recovery operation is attempted repeatedly until the permissible number of recovery tries has been reached, at which time the I/O driver stores the error status in the user I/O-request block, and the I/O-error processor posts the error on the OC logical unit. The user can then try another physical device or abort the task.

3.1 LOGICAL UNITS

A **logical unit** is an I/O device or a partition of a rotating-memory device (RMD). It is referenced by an assigned number or name. The logical unit permits performance of I/O operations that are independent of the physical-device configurations by making possible references to the logical-unit number. The standard interfaces between the program and the IOC, and between the IOC and the I/O driver, permit substitution of peripheral devices in I/O operations without reassembling the program.

VORTEX permits up to 256 logical units. The numbers assigned to the units are determined by their reassignability:

- a. *Logical-unit numbers 1-100* are used for units that can be reassigned through the operator communications component (OPCOM, section 17) or the job-control processor (JCP, section 4).
- b. *Logical-unit numbers 101-179* are used for units that are not reassignable.
- c. *Logical-unit numbers 180-255* are used for units that can be reassigned through OPCOM only.
- d. *Logical-unit number 0* indicates a dummy device. The IOC immediately returns control from a dummy device to the user as if a real I/O operation had been completed.
- e. Each disk partition requires a device specification table (DST) and each DST is treated as a potential logical unit. The limit of 256 logical units can result in the inability to assign a unique logical unit to each partition if the number of non-disk devices and partitions on the system exceeds 256. To access disk partitions which have DSTs greater than the 256th, the user can issue JCP or OPCOM assignment statements prior to program execution or he can specify disk drives as "special" drives, where one of the 256 logical unit numbers is associated with all the partitions of the drive. For special disk drives, the desired partition is selected by file name.

VORTEX logical unit assignments for all systems are specified in Table 3-1. All logical unit numbers that are not listed are available for reassignment as indicated above.

Table 3-2 indicates possible logical unit assignments. The system names for physical devices are listed in Table 17-1

3.1.1 SPECIAL LOGICAL UNITS

Special logical units are defined at system generation time by specifying partition number zero (0) in the PRT directive. The user can then assign a logical unit number to the entire disk unit by using the device name Dcu(0) in the System Generation ASN directive or by using the name Dcu@ in the JCP or OPCOM ASSIGN statement. The partition accessed on the special logical unit is indicated by the fifth and sixth characters of the file name. Thus a file name on a special logical unit is limited to 4 unique characters because the fifth and sixth characters of the file name are the partition number. These characters must be numbers between 01 and 63, inclusive. For example, if the ninth partition is to be accessed on a special logical unit, the file name in the FCB for the OPEN request is of the form xxxx09. An I/O error in special logical unit file names results in an IO02 diagnostic message.

If the user wishes to perform direct VORTEX I/O using the logical unit, he assigns a specific VORTEX logical unit to the desired partition with a JCP or OPCOM command. Under program control the RTE SELECT macro can select specific partitions on a particular logical unit, overriding the file name format above (see section 2.1.24).

The rules for using special logical units also apply when files are created or maintained using VZFMA. To use other VORTEX utilities such as FMAIN or FMUTIL on a special logical unit, the user must first assign a VORTEX logical unit to the specific partition needed.

Table 3-1. VORTEX Logical-Unit Assignments

Number	Name	Description	Function
0	DUM	Dummy	For I/O simulation
1	OC	Operator communication	For system operator communication with immediate return to user control; Teletype or CRT only
2	SI	System input	For inputs of all JCP control directives to any device
3	SO	System output	For display of all input control directives and output system messages; teletypewriter or CRT only. VORTEX allows "SO" to be assigned to DUM. If a background program detects an error, it usually goes to "SO" for corrected input. If SO = DUM, the input buffer is reprocessed, causing an infinite loop
4	PI	Processor input	For input of source statements from all operating system language processors
5	LO	List output	For output of operating system input control directives, system operations messages, and operating system language processors' output listings
6	BI	Binary input	For input of object-module records from operating system processors
7	BO	Binary output	For output of object-module records from operating system language processors
8	SS	System scratch	For system scratch use; all operating system language processors that use an intermediate scratch unit input from this unit
9	GO	Go unit	For output of the same information as the BO unit by the system assembler and compiler; RMD partition or MT.
10	PO	Processor output	For processor output; all operating system language processors that use an intermediate scratch unit output to this unit; PO and SS are assigned to the same device at system-generation time
11	DI	Debugging input	For all debugging inputs
12	DO	Debugging output	For all debugging outputs

INPUT/OUTPUT CONTROL

Table 3-1. VORTEX Logical-Unit Assignments

(continued)

Number	Name	Description	Function
101	CU	Checkpoint unit	For use by VORTEX to checkpoint a background task; partition protection key S; RMD partition only
102	SW	System work	For generation of a load module by the system load-module generator component; or for cataloging, loading, or execution by other system components; partition protection key B; RMD partition only
103	CL	"Core" -resident library	For all "core" -resident system entry points; partition protection key C; RMD partition only (12 names per 2 sectors)
104	OM	Object-module library	For the VORTEX system object-module library; partition protection key D; RMD partition only
105	BL	Background library*	For the VORTEX system background library; partition protection key E; RMD partition only
106	FL	Foreground library*	For the VORTEX system foreground library; partition protection key F; RMD partition only

* Other units can be assigned as user foreground libraries provided they are specified at system-generation time. However, there is only one background library in any case.




Table 3-2. Valid Logical-Unit Assignments

Logical Unit Unit No.	OC 1	SI 2	SO 3	PI 4	LO 5	BI 6	BO 7	SS 8	GO 9
Device									
Dummy				DUM	DUM	DUM	DUM	DUM	
Card punch					CP		CP		
Card reader		CR		CR		CR			
CRT device	CT	CT	CT	CT	CT				
RMD (disc/drum) partition		D		D	D	D	D	D	D
Line printer					LP				
Magnetic-tape unit		MT		MT	MT	MT	MT	MT	MT
Paper-tape reader/ punch		PT		PT	PT	PT	PT		
Teletype	TY	TY	TY	TY	TY				
Remote Teletype		TC	TC	TC	TC				
Logical Unit Unit No.	PO 10	DI 11	DO 12	CU 101	SW 102	CL 103	OM 104	BL 105	FL 106
Device									
Dummy	DUM		DUM						
Card punch	CP								
Card reader		CR							
CRT device	CT	CT	CT						
RMD (disc/drum) partition	D			D	D	D	D	D	D
Line printer	LP		LP						
Magnetic-tape unit	MT								
Paper-tape reader/ punch	PT								
Teletype	TY	TY	TY						
Remote Teletype		TC	TC						

INPUT/OUTPUT CONTROL

3.2 RMD FILE STRUCTURE

Each RMD (rotating-memory device) is divided into up to 20 memory areas called **partitions**. Each partition is referenced by a specific logical-unit number. The boundaries of each partition are recorded in the core-resident **partition specification table (PST)**. The first word of the PST contains the number of VORTEX physical records per track. The second word of the PST contains the address of the bad-track table, if any, or zero. Subsequent words in the PST comprise the partition entries. Each PST entry is in the format:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 0	Beginning partition address (track number)															
Word 1	ppb											Protection key				
Word 2	Number of bad tracks in the partition															
Word 3	Ending partition address + 1															

Section 9.1 describes the full PST format.

The **partition protection bit**, designated ppb in the above PST entry map, when set, requires the correct protection key to read/write from this partition.

Note that PST entries overlap. Thus, word 3 of each PST entry is also word 0 of the following entry. The length of the PST is $3n + 2$, where n is the number of partitions in the system. The relative position of each PST entry is recorded in the **device specification table (DST)** for that partition.

The **bad-track table**, whose address is in the second word of the PST, is a bit string constructed at system-generation time and thereafter constant. The bits are read from right to left within each word, and forward through contiguous words, with set bits flagging bad tracks on the RMD.

The 70-755x and 70-756x, RMD drivers use a bad sector table (BST) instead of a bad track table. This RMD may also contain up to 63 partitions instead of 20, and so the PST occupies the first 2 logical 120 word records of the RMD. The third logical record contains the second half of the boot program, so that the BST begins on the fourth logical 120 word record instead of the second as for other RMDs. Because the second half of the boot program must be on the

third logical record, 70-755x system disks must be formatted using the FORMAT-H routine before the first and between any subsequent system generations. The BST format is logically the same as the BTT (bad track table).

Each RMD partition can contain a **file-name directory** of the files contained in that partition. The beginning of the directory is in the first sector of that partition. The directory for each partition has a variable number of entries arranged in n sectors, 19 entries per sector. Sectors containing directory information are chained by pointers in the last word of each sector. Thus, directory sectors need not be contiguous. (**Note:** Directories are not automatically created when the partitions are defined at system-generation time. It is possible to use a partition with no directory, e.g., by a foreground program that is collecting data in real time.) Each directory entry is in the format:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 0	File name															
Word 1	File name															
Word 2	File name															
Word 3	Current position of file															
Word 4	Beginning file address															
Word 5	Ending file address															

The file name comprises six ASCII characters packed two characters per word. Word 3 contains the current address at which the file is positioned, is initially set to the ending file address, and is manipulated by the OPEN and CLOSE macros (sections 3.5.1 and 3.5.2). The extent of the file is defined by the addresses set in words 4 and 5 when the file is created, and which remain constant.

At system-generation time, the first sector of each partition is assigned to the file-name directory and a zero written into the first word. Once entries are made in the file-name directory, the first word of each sector contains a count of the entries in that sector.

The last entry in each sector is a one-word entry containing either the value 01 (end of directory), or the address of the next sector of the file-name directory.

The file-name directories are created and maintained by the VORTEX file-maintenance component (section 9) for IOC use. User access to the directories is via the IOC, which references the directories in response to the I/O macros OPEN and CLOSE. The file-maintenance component sets words 0, 1, 2, 4, and 5 of each directory entry, which then remain constant and unaffected by IOC operations. The IOC can modify only the current position-of-file parameter.

In the case of a file containing a directory, an OPEN is required before the file is accessible. The macro searches the file directory for the entry corresponding to the name in the file-control block (FCB) in use. When the entry is found, the file boundary addresses and the current position-of-file value from the directory entry are stored in the FCB. If the OPEN macro

- a. Specifies the option to rewind, the FCB current position is set equal to the address of the beginning of file.
- b. Specifies the option not to rewind, the FCB current position is set equal to the address of the position of file.

Once a file is thus opened, READ and WRITE operations are enabled. The IOC references the file by the file boundary values set by the OPEN, rather than by the file name. READ and WRITE operations are under control of the FCB current position value, the extent of the file, and the current record number.

A CLOSE macro disables the IOC and user access to the file by zeroing the four file-position parameters in the FCB. If the CLOSE macro

- a. Specifies the option to update, the current position-of-file value in the directory entry is set to the value of the FCB current position, allowing reference by a later OPEN.
- b. Specifies the option not to update, the file-directory entry remains unmodified.

Special directory entries: A **blank entry** is created when a file name is deleted, in which case the file name is "*****" and words 3 through 5 give the extent of the blank file. A **zero entry** is created when one name of a multiname file is deleted, in which case the deleted name is converted to a *blank entry* and all other names of the multiname file are set to zero.

When using sequential access with file extension (access method 9) VORTEX will automatically create a file extension for the user if an EOD (end of device) status occurs on an RMD device during a write operation. The file extension is created by the manager routine VZFMA. The file size is equal to the size of the original file and is completely separate (non contiguous and non linked) from the base file segment and its file extensions are loosely linked by sharing the same file name and containing a unique file extension number. The current file extension number is maintained in word 2, bits 12-13 of the current FCB. A file may contain up to 3 extensions. If these are exceeded, a standard EOD status is returned to the user. All file extensions must reside in the same logical unit as the base segment thus the number of extensions may be limited due to available space on the logical unit. On read requests, when an EOD status is returned, the next sequential file extension is searched for

and, if found, I/O continues using the file extension. For both reads and writes, the handling of file extensions is transparent to the user.

Note: Files that contain extensions should not be used with direct access I/O as the user FCB file extents are subject to modification by the file extension logic. If a file created with sequential I/O, has had extensions created, and is to then be used with direct access I/O, that file must be copied into a contiguous single file.

3.3 I/O INTERRUPTS

VORTEX uses a complete, interrupt-driven I/O system, thus optimizing the allocation of CPU cycles in the multiprogramming environment.

3.4 SIMULTANEOUS PERIPHERAL OUTPUT OVERLAP (SPOOL)

The VORTEX spooler is a generalized set of routines which permit queuing of a task's output to intermediate RMD files. This avoids the user task waiting for the device transfer completion. Total system throughput will be increased because waiting for transfers to be completed, both in the use of I/O calls with suspended returns and that of tasks which are terminating, will be minimized.

Also, non-resident tasks may transfer to a spooled device and immediately exit, instead of remaining resident until completion of the transfer.

At system generation, the user may have the output of some logical units, such as LO, automatically spooled. During operation, the operator may assign device outputs to the spooler through JCP or OPCOM assign directives.

Components

The SPOOL subsystem consists of two components: (1) an IOC driver to which data output may be assigned and which transfers output for its associated logical unit to a circular RMD file or directly to the output listing task, and (2) an output listing task which accepts messages from this circular RMD file or directly from the IOC driver and transfers them to the appropriate output device.

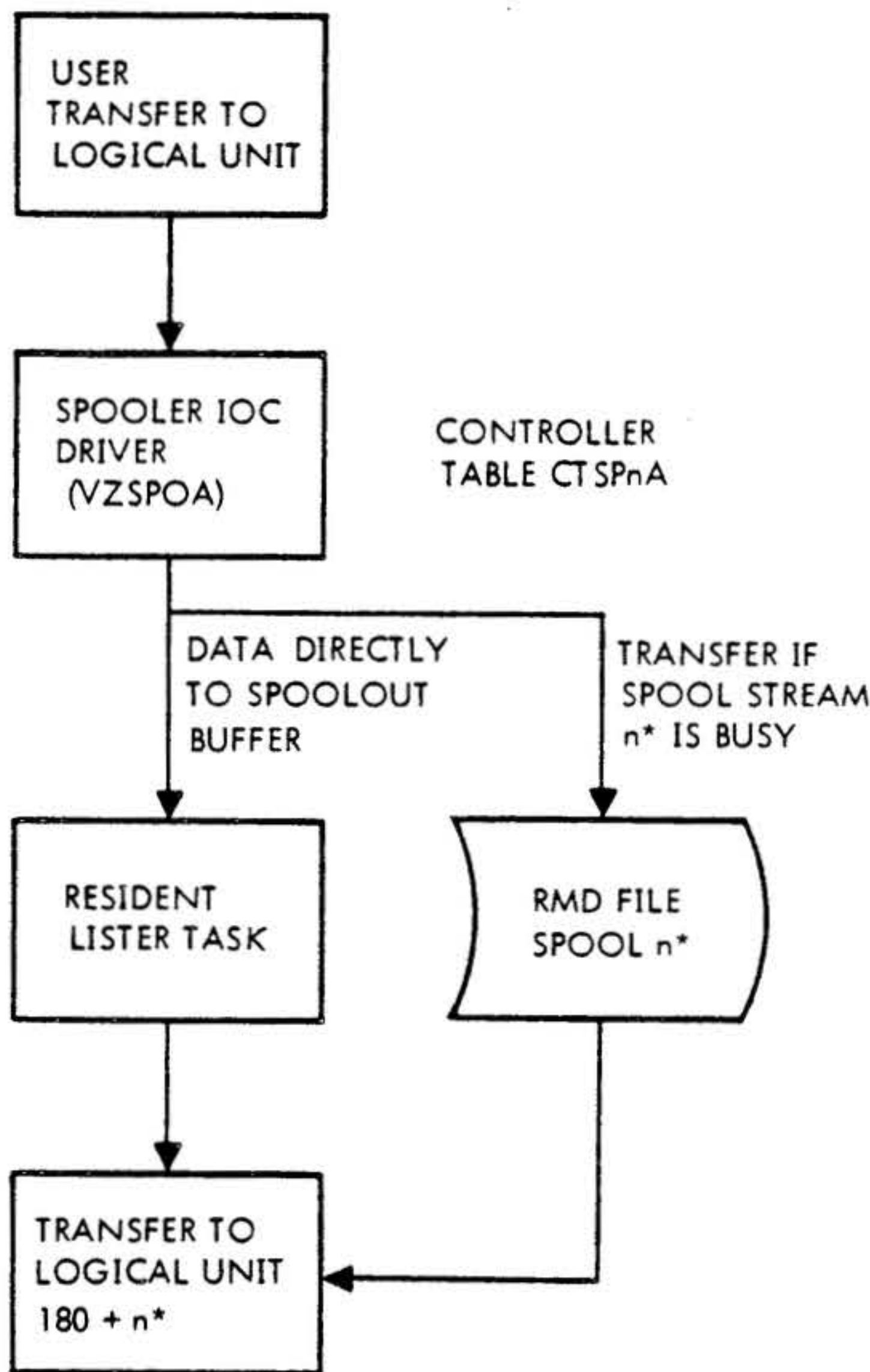
Communication between these two tasks is accomplished through parameters within the listing task which are

INPUT/OUTPUT CONTROL

established by the IOC driver. When these and other system parameters indicate that the listing task has caught up with the spoolout task, output messages will be transferred directly to the listing task, instead of going through the RMD SPOOL file. (This avoids the overhead of two RMD transfers).

All data records transferred to the circular RMD file will contain record length and a key signifying whether the transfer is to be write or a function as well as other synchronization data. Data will be transferred to RMD in an unpacked mode (one record per sector) in order to avoid delays caused by unwritten still-to-be packed records. SPOOL file overflow messages will be output when appropriate after allowing the RMD circular file certain amounts of time to remove its oldest entry.

Figure 3-1 shows a simplified flow of output data through the SPOOL subsystem.



* WHERE n IS AN INTEGER FROM ZERO TO SEVEN

VT11-2123

Figure 3-1. Spooling Subsystem Flow

3.4.1 SPOOL Operation

During the system generation, up to eight spool pseudo devices may be defined. These pseudo-devices, SPOA through SP7A are dummies which can be assigned to any logical unit used only for output. Such assignments can be made permanently at SGEN time, or dynamically through JCP or OPCOM.

Each pseudo-device, SPiA, has a corresponding RMD file name, SPOOLi. These files must be defined on an RMD partition which is permanently assigned to logical unit 107 (named SX). Each spool pseudo-device and file is then associated with a logical unit (180-187) to which the LISTER writes unit record output. For example, a user issuing a WRITE request to an LUN assigned to device SPiA, will have data transferred to file SPOOLi on RMD.

The data will be read from the RMD and written to LUN $180 + i$, whose name is Si, as time and the device allow.

If the output device is not busy when a user request is made, and if the RMD stream is inactive, the user data is moved directly to the output device via a SPOOL buffer. In this case, the user request is set complete as soon as the buffer is queued for the device.

If a user's I/O requests are made and a spool pseudo-device number for the appropriate SPOOLi file could not be found, or if the RMD is inoperative for any reason, the RMD is bypassed. That is, each user request causes a SPOOL buffer containing the user's data to be queued directly to the output device, up to a maximum of two buffers per stream. If the user should issue a request that would require a third buffer for that stream, then the SPOOL driver enters a delay loop until the two buffer limit can be satisfied. During this wait time, the user's I/O is active.

If the output device to which a user is spooling output should go down or become not ready, data continues to be accepted and spooled to RMD, but not more than two SPOOL buffers will be tied up waiting for the device to become usable. If an RMD is down when this case occurs, user's requests will be delayed after two buffers are allocated to the stream.

Should the user fill the RMD file for a stream with data before the device can catch up, the next user request remains active until space is available in the RMD.

3.4.2 SPOOL Files

Certain RMD files are required for maximum spooler operation. Without these, the SPOOL subsystem will function at a reduced rate. Files SPOOL0 through SPOOL7, where the last digit is the SPOOL stream number, are used as circular files and may be established at varying lengths to improve system performance. SPOOL operation will be slower if RMD files are totally filled with data to be output.

Files must be created after SGEN but before the first user of the SPOOL program. To establish files in a manner consistent with SPOOL, an exact procedure must be followed. If LO is assigned to SPOOL, it must be reassigned temporarily to a non-spooled device through OPCOM using a command such as:

```
;ASSIGN,LO=LP
```

where LP is not a spooled device. After this step, the actual file or files must be created using FMAIN in the following manner:

```
/FMAIN  
INIT,107,S  
CREATE,107,S,SPOOL0,120,n  
CREATE,107,S,SPOOL1,120,n  
.  
.  
.  
CREATE,107,S,SPOOL7,120,n  
/FINI
```

The last parameter n of the CREATE directives is the number of records. A CREATE directive is required for each data stream. As many CREATE directives as data streams are required.

The number of 120-word records to be established within the file is given as the last parameter of the CREATE directive. SPOOL files are circular files; entries are being placed on one end while being removed from the other end. When the SPOOL subsystem determines that the file is full, i.e., that another entry cannot be placed on the file without destroying one which has not been removed, transfers to the spooler driver will not be completed until a new file entry becomes available (the oldest entry has been removed from the file). As file size is increased, the likelihood of a full file is decreased. File size should be a function of expected stream utilization and device output speed, which determines how quickly entries are moved from circular spooler files. The IO60 error message indicates that a file is full. If this message is received frequently the number of records in that file should be increased for maximum spooling efficiency.

This procedure for creation of SPOOL files needs to be done only once. It is performed immediately after completion of SGEN when the "VORTEX SYSTEM READY" message is output. If these file sizes are found to be unsatisfactory, the system may be rebooted and file sizes modified by executing the procedure again.

As part of the SGEN for systems using the SPOOL program, controller table 0 (stream 0) must be included since the initialization routine is included in its buffers. Additional controller tables may be included as desired. However, storage requirements may be varied by using different controller tables: all even addresses contain four 74-word buffers, and odd streams contain only two 74-word buffers. For systems with a large amount of SPOOL throughput, it is recommended that four buffers be specified for controller tables, otherwise two-buffer tables should be sufficient.

3.5 I/O-CONTROL MACROS

I/O requests are written in assembly language programs as I/O macro calls. The DAS MR assembler provides the following I/O macros to perform I/O operations, thus simplifying coding:

- OPEN Open file
- CLOSE Close file
- READ Read one record
- WRITE Write one record
- REW Rewind
- WEOF Write end of file
- SREC Skip one record
- FUNC Function
- STAT Status
- DCB Generate data control block
- FCB Generate file control block

The following I/O macros apply to the 70-755x RMD only:

- DEMAND Demand the controller for this unit
- RELEAS Release the controller after a DEMAND
- BCB Generate Buffer Controller Block
- RESERV Reserve the controller for this unit
- RELRSV Release the controller after a RESERV

INPUT/OUTPUT CONTROL

The IOC performs a validity check on all I/O requests. It then queues (according to the priority of the requesting task) each valid request to the controller assigned to the specified logical unit. Finally, the IOC schedules the appropriate I/O driver to service the queued request.

The assembler processes the I/O macro to yield a macro expansion comprising data and executable instructions in the form of assembler language statements.

Certain I/O operations require parameters in addition to those in the I/O macro. These parameters are contained in a table, which, according to the operation requested, is called either a file control block (FCB, section 3.5.11) or a data control block (DCB, section 3.5.10). Embedded but omitted parameters (e.g., default values) must be indicated by the normal number of commas.

Error messages applicable to these macros are given in Appendix A.3.

I/O Macros: *The general form of I/O macros is:*

label name cb,lun,wait,mode

where the symbols have the definitions given in section 3.5.1.

If the **cb** is for an FCB, it is mandatory. If it is for a DCB, it is optional.

The expansion of an I/O macro is:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 0	J S R,X															
Word 1	0404															
Word 2	c	Status				e	cc				Priority*					
Word 3	w	Mode				Op-code				Logical-unit number						
Word 4	FCB or DCB address															
Word 5	User task identification block address*															
Word 6	IOC thread address*															

where

c	set indicates completion of I/O tasks
Status	is the status of the I/O request
e	set indicates an irrecoverable I/O error
cc	is the completion code
Priority	is the priority level of the task making the request
w	is the wait/immediate-return option
Mode	is the mode of operation
Op-code	specifies the I/O operation to be performed
*	indicates an item whose initial value is zero

The wait option causes the task to be suspended until its I/O is complete. The immediate option causes control to be returned immediately to the task after the I/O request is queued. Therefore, to multiprogram effectively within VORTEX, the wait option is preferred.

Word 2 contains the following information:

- Bit 15 indicates whether the I/O request is complete.
- Bits 14 through 9 contain one of the error-message status codes described in Appendix B.2.
- Bit 8 indicates an irrecoverable I/O error.
- Bits 7 through 5 contain a completion code: 000 indicates a normal return; 101, an error; 110, an end of file, beginning of device, or beginning of tape; and 111, end of device, or end of tape.
- Bits 4 through 0 indicate the priority level of the task making the request.

Word 3 contains the following information:

Bits 0-7 Logical Unit (LUN)

When an I/O request is made to V\$IOC, V\$IOC uses the LUN as an index into the logical unit table (LUT). V\$IOC then uses the current assignment pointer of that entry in the LUT to determine the address of the DST on which the

The last parameter *n* of the CREATE directives is the number of records. A CREATE directive is required for each data stream. As many CREATE directives as data streams are required.

The number of 120-word records to be established within the file is given as the last parameter of the CREATE directive. SPOOL files are circular files; entries are being placed on one end while being removed from the other end. When the SPOOL subsystem determines that the file is full, i.e., that another entry cannot be placed on the file without destroying one which has not been removed, transfers to the spooler driver will not be completed until a new file entry becomes available (the oldest entry has been removed from the file). As file size is increased, the likelihood of a full file is decreased. File size should be a function of expected stream utilization and device output speed, which determines how quickly entries are moved from circular spooler files. The IO60 error message indicates that a file is full. If this message is received frequently the number of records in that file should be increased for maximum spooling efficiency.

This procedure for creation of SPOOL files needs to be done only once. It is performed immediately after completion of SGEN when the "VORTEX SYSTEM READY" message is output. If these file sizes are found to be unsatisfactory, the system may be rebooted and file sizes modified by executing the procedure again.

As part of the SGEN for systems using the SPOOL program, controller table 0 (stream 0) must be included since the initialization routine is included in its buffers. Additional controller tables may be included as desired. However, storage requirements may be varied by using different controller tables: all even addresses contain four 74-word buffers, and odd streams contain only two 74-word buffers. For systems with a large amount of SPOOL throughput, it is recommended that four buffers be specified for controller tables, otherwise two-buffer tables should be sufficient.

3.5 I/O-CONTROL MACROS

I/O requests are written in assembly language programs as I/O macro calls. The DAS MR assembler provides the following I/O macros to perform I/O operations, thus simplifying coding:

- OPEN Open file
- CLOSE Close file
- READ Read one record
- WRITE Write one record
- REW Rewind
- WEOF Write end of file
- SREC Skip one record
- FUNC Function
- STAT Status
- DCB Generate data control block
- FCB Generate file control block

The IOC performs a validity check on all I/O requests. It then queues (according to the priority of the requesting task) each valid request to the controller assigned to the specified logical unit. Finally, the IOC schedules the appropriate I/O driver to service the queued request.

The assembler processes the I/O macro to yield a macro expansion comprising data and executable instructions in the form of assembler language statements.

Certain I/O operations require parameters in addition to those in the I/O macro. These parameters are contained in a table, which, according to the operation requested, is called either a file control block (FCB, section 3.5.11) or a data control block (DCB, section 3.5.10). Embedded but omitted parameters (e.g., default values) must be indicated by the normal number of commas.

Error messages applicable to these macros are given in Appendix A.3.

I/O Macros: *The general form of I/O macros is:*

label name cb,lun,wait,mode

where the symbols have the definitions given in section 3.5.1.

If the **cb** is for an FCB, it is mandatory. If it is for a DCB, it is optional.

INPUT/OUTPUT CONTROL

The expansion of an I/O macro is:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 0	J S R,X															
Word 1	0404															
Word 2	c	Status										e	cc	Priority*		
Word 3	w	Mode				Op-code				Logical-unit number						
Word 4	FCB or DCB address															
Word 5	User task identification block address*															
Word 6	IOC thread address*															

where

c	set indicates completion of I/O tasks
Status	is the status of the I/O request
e	set indicates an irrecoverable I/O error
cc	is the completion code
Priority	is the priority level of the task making the request
w	is the wait/immediate-return option
Mode	is the mode of operation
Op-code	specifies the I/O operation to be performed
*	indicates an item whose initial value is zero

The wait option causes the task to be suspended until its I/O is complete. The immediate option causes control to be returned immediately to the task after the I/O request is queued. Therefore, to multiprogram effectively within VORTEX, the wait option is preferred.

Word 2 contains the following information:

- Bit 15 indicates whether the I/O request is complete.
- Bits 14 through 9 contain one of the error-message status codes described in Appendix B.2.
- Bit 8 indicates an irrecoverable I/O error.
- Bits 7 through 5 contain a completion code: 000 indicates a normal return; 101, an error; 110, an end of file, beginning of device, or beginning of tape; and 111, end of device, or end of tape.

- Bits 4 through 0 indicate the priority level of the task making the request.

Word 3 contains the following information:

Bits 0-7 Logical Unit (LUN)

When an I/O request is made to V\$IOC, V\$IOC uses the LUN as an index into the logical unit table (LUT). V\$IOC then uses the current assignment pointer of that entry in the LUT to determine the address of the DST on which the I/O is to be performed. To determine the DST address, the current assignment value less one is multiplied by the length of a DST (3 words) and added to the base address of the DST block. V\$IOC verifies the validity of the specified LUN.

If the LUN is invalid, a parameter error has occurred (refer to sections 3.1 and 3.3).

Bits 8-11 Op-Code

Op-codes can range in value from 0 to 15; however, not all op-codes are applicable for every device. V\$IOC, using the op-code as an index gets an entry from a bit table. This word contains a 1 in the bit position associated with the op-code and is compared with the controller table item CTOPM. If the corresponding bit in CTOPM is set to 1, it means that the device connected to the controller can perform the requested operation. If the corresponding bit in CTOPM is zero, the I/O request is not performed, and the I/O complete indicator (bit 15) set.

Bit 8-11	Meaning
0000	Read
0001	Write
0010	Write EOF
0011	Rewind
0100	Skip record
0101	Function
0110	Open
0111	Close
1000-1111	Not used

Bits 12-14 Mode

The mode bits are not used by V\$IOC nor V\$FNR. The I/O driver use this information whenever applicable to the op-code.

Bit 15 Wait Option

V\$IOC uses this bit to determine whether the requesting task is to be suspended until I/O is completed or whether an immediate return is required.

- Bit 15 = 0 Suspend until I/O completed. V\$IOC sets bit 14 in TBST in the requesting task's TIDB.
- Bit 15 = 1 Immediate return required (via V\$DISP). V\$IOC clears bit 14 in TBST in the requesting task's TIDB.

Word 5 initially points to the user's task identification block. Upon completion of a READ or WRITE macro (sections 3.5.3 and 3.5.4), the IOC sets word 5 to the actual number of words transmitted.

Word 6 initially contains the request block thread. Upon completion of a READ or WRITE operation the IOC sets word 6 to the number of retries attempted prior to completion of the request.

Status macro: *The general form of the status (STAT) macro is:*

label **STAT** *req,err,aaa,bbb,busy*

where the symbols have the definitions given in section 3.5.9.

The normal return is to the first word following the macro expansion.

The expansion of the STAT macro is:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 0	J S R,X															
Word 1	0373															
Word 2	Address of the I/O macro															
Word 3	Address of the I/O error routine															
Word 4	aaa															
Word 5	bbb															
Word 6	Address of the busy or I/O-not-complete routine															

where **aaa** is the address of the end of file, beginning of device or beginning of the tape routine and **bbb** is the address of the end of the tape or end of the device routine.

Control block macro: *The general form of the DCB macro is:*

label **DCB** *rl, buff, fun*

where the symbols have the definitions given in section 3.5.10.

The expansion of the DCB macro is:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 0	Record length															
Word 1	Direct Address of user data area															
Word 2	Function code															

The function code applies only to I/O drivers that allow:

- a. The line printer to slew to top of form or to space through the channel selection for paper-tape form control.
- b. The paper-tape punch to punch leader.
- c. The card punch to eject a blank card as a separator.

The general form of the FCB macro is:

label **FCB** *rl, buff, acc, key, 'xx', 'yy', 'zz'*

where the symbols have the definitions given in section 3.5.11.

The expansion of the FCB macro is:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 0	Record length															
Word 1	Address of user data area															
Word 2	Access method								Protection key							
Word 3	Current record number															
Word 4	Current end-of-file address															
Word 5	Beginning file address															
Word 6	Ending file address															
Word 7	File name															
Word 8	File name															
Word 9	File name															

The access method (word 2, bits 15 through 8) specifies one of the four methods of reading or writing a file:

- a. *Direct access by logical record:* The I/O driver uses the contents of FCB word 3 as the number of the logical record within a file to be processed, but does not alter word 3 after reading or writing. Word 3 is set by the user to the desired record number prior to each read/write. Specifying FCB word three to zero will cause access to the partition directory. Care should be taken when supplying this value so that directories are not accidentally destroyed.
- b. *Sequential access by logical record:* The I/O driver uses the contents of word 3 as the number of the logical record within a file to be processed, then increments the contents of word 3 by one. Word 3 is set initially to zero when the FCB macro expands. Successive reading and writing thus accesses records sequentially.

INPUT/OUTPUT CONTROL

- c. *Direct access by physical record:* The I/O driver uses the contents of FCB word 3 as the number of the VORTEX physical record to be processed within a file (120-word length), but does not alter word 3 after a read or write. Word 3 is set by the user to the desired record number prior to each read/write. Specifying FCB word three to zero will cause access to the partition directory. Care should be taken when supplying this value so that directories are not accidentally destroyed.
- d. *Sequential access by physical record:* The I/O driver uses the contents of FCB word 3 as the number of the VORTEX physical record to be processed within a file (120-word length), then increments the contents of word 3 by one. Word 3 is set initially to zero when the FCB macro expands. Successive reading and writing thus accesses records sequentially.

3.5.1 OPEN Macro

This macro, which applies only to RMDs or magnetic-tape units, enables I/O operations on the devices by initializing the file information in the specified FCB. The macro has the general form

```
label OPEN fcb,lun,wait,mode
```

where

- fcb** is the address of the file control block
- lun** is the number of the logical unit being opened
- wait** is 1 for an immediate return, or 0 (default value) for a return suspended until the I/O is complete
- mode** is 0 (default value) for rewinding or 1 for not rewinding. In the former case, word 3 (current record number) of the FCB is set to 1, word 4 (current position-of-file address) is set to the current position-of-file address given by the RMD file directory, and rewinds the magnetic-tape unit. In the latter case, the current position-of-file address given by the RMD file directory is copied into word 4, converted to a record number and stored in word 3 of the FCB, thus initializing the user FCB, enabling reading or writing from a previously specified location, and the magnetic-tape position is left unchanged (not rewound).

OPEN must precede any other I/O request (except REW) because the FCB file information must be complete before any file-oriented I/O is possible. If a file has already been opened, an OPEN will be accepted.

The OPEN macro is file-oriented, while the REW macro is oriented to the logical unit. An REW destroys information completed by a previous OPEN on the same logical unit.

The OPEN macro changes words 3, 4, 5, and 6 of the FCB (section 3.5.11).

If an attempt is made to apply the OPEN macro to any device other than an RMD or a magnetic-tape unit, the I/O request is processed internally by the IOC but not by an I/O driver. The IOC indicates the status as I/O complete.

Example: Read a 120-word record from the FI10 on logical unit 18, an RMD partition with sequential, record-oriented access. BUFF is the address of the user's buffer area. Use the wait and rewind options, and set the logical-unit protection key to 1.

```
X1 EQU 18 (LUN assigned to unit X1)
RL EQU 120 (Record length 120)
WAIT EQU 0 (Wait option)
REW EQU 0 (Rewind option)
KEY EQU 1 (Logical-unit protection key)
SEQR EQU 1 (Sequential, record-oriented
access)
OPEN OPEN FCB,X1,WAIT,REW
READ READ FCB,X1,WAIT
.
.
.
FCB FCB RL,BUFF,SEQR,KEY,
'FI','10',' '
```

3.5.2 CLOSE Macro

This macro, which applies only to RMDs or magnetic-tape units, updates information in the specified FCB file. This records and retains the current position within the file. The *mode* option ignores the updating, thus retaining the previously defined position in the file. The macro has the general form

```
label CLOSE fcb,lun,wait,mode
```

where

- fcb** is the address of the FCB
- lun** is the number of the logical unit being closed
- wait** is 1 for an immediate return, or 0 (default value) for a return suspended until the I/O is complete

mode is 0 (default value) for not updating, or 1 for updating. In the former case, there is no change to the current position-of-file address in the RMD file directory, words 3, 4, 5, and 6 of the FCB are set to zero, and the magnetic-tape position is left unchanged (not rewound). In the latter case, the contents of FCB word 3 (current record number) are converted to an address and stored in the current position-of-file address in the RMD file directory, words 3, 4, 5, and 6 of the FCB are set to zero, and an end-of-file mark written on the magnetic tape.

The CLOSE macro cannot be used if there is no such file defined in the FCB (section 3.5.11).

If an attempt is made to apply the CLOSE macro to any device other than an RMD or magnetic-tape unit, the I/O request is processed internally by the IOC, but not by an I/O driver. The IOC indicates the status as I/O complete.

Example: Close the file MATRIX on logical unit 180, an RMD partition with sequential, record-oriented access. Use the wait and update options.

```

SEQR    EQU    1    (Sequential, record-
                oriented access)
UPDATE  EQU    1    (Update option)
WAIT    EQU    0    (Wait option)
.
.
.
CLOSE   CLOSE   FCB, 180, WAIT, UPDATE
.
.
.
FCB     FCB     , , SEQR, , 'MA', 'TR', 'IX'
    
```

3.5.3 READ Macro

This macro retrieves a record of specified length from the specified logical unit, and places it in the specified area of main memory. The macro has the general form

```
label    READ    cb,lun,wait,mode
```

where

cb is the address of the data control block, or of the file control block

lun is the number of the logical unit from which the record is read

wait is 1 for an immediate return, or 0 (default value) for a return suspended until the I/O is complete

mode specifies the I/O mode: 0 (default value) for system binary, 1 for ASCII, 2 for BCD, or 3 for unformatted I/O (see appendix C for format)

The number of words read is stored in word 5 of the I/O macro.

Example: Read a record from logical unit 4, a magnetic-tape unit. Use system binary mode and the immediate return option. The record length is 60 words, and the address of the user's data area is BUFF.

```

IM      EQU    1    (Immediate return)
BIN     EQU    0    (System binary mode)
MT      EQU    4    (LUN assigned to
                magnetic-tape unit)
RECL    EQU    60   (Record length 60 words)
.
.
.
MTRD    READ    TAPE, MT, IM, BIN
.
.
.
TAPE    DCB     RECL, BUFF (Data control block)
BUFF    BSS     60      (User data area)
    
```

Note that the READ macro had a mode value of zero. Since this is the default value, the macro could have been coded:

```
MTRD    READ    TAPE, MT, IM
```

3.5.4 WRITE Macro

This macro takes a record of specified length from the specified area of main memory, and transmits it to the specified logical unit. The macro has the general form

```
label    WRITE    cb,lun,wait,mode
```

where the parameters have the same definitions and take the same values as in the READ macro (section 3.5.3).

The number of words written is stored in word 5 of the I/O macro. The first byte of each print line is treated as a print control character and not echoed when outputting to a listing device.

Example: Obtain a system binary record 60 words in length from the user's data area BUFF, and transmit it to logical unit 16, a magnetic-tape unit. Use the immediate-return option.

```

IM      EQU    1    (Immediate return)
BIN     EQU    0    (System binary mode)
MT      EQU    16   (LUN assigned to magnetic-
                tape unit)
RECL    EQU    60   (Record length 60 words)
.
.
.
MTWT    WRITE    TAPE, MT, IM, BIN
.
.
.
TAPE    DCB     RECL, BUFF (Data control block)
BUFF    BSS     60      (User data area)
    
```

INPUT/OUTPUT CONTROL

3.5.5 REW (Rewind) Macro

This macro, which applies only to magnetic-tape or rotating-memory devices, repositions the specified logical unit to the beginning-of-unit position. It has the general form

```
label REW cb,lun,wait
```

where

cb is the address of the FCB or DCB, which is optional

lun is the number of the logical unit being rewound

wait is 1 for an immediate return, or 0 (default value) for a return suspended until the I/O is complete

Note that the DCB address is an optional parameter, but that the FCB address is mandatory.

To reposition a named file on an RMD, use the OPEN macro (section 3.5.1).

Magnetic-tape devices: REW rewinds the specified unit and, upon successful completion of the task, returns a beginning-of-device (BOD) status.

Rotating-memory devices: REW places the start-RMD-partition and end-RMD-partition addresses in words 5 and 6, respectively, of the FCB (section 3.5.11).

Examples: Rewind logical unit 23, a magnetic-tape unit. Use the wait option, here specified by default.

```
MT EQU 23 (LUN assigned to magnetic-
.
.
.
REWT REW ,MT
.
.
.
```

Rewind logical unit 10, an RMD partition. Use the wait option, here specified by default. Note that the REW for an RMD must have an associated FCB (section 3.5.11).

```
DISC EQU 10 (LUN assigned to RMD
partition)
RECL EQU 120
.
.
.
REWD REW FCB,DISC
.
.
.
FCB FCB RECL,BUFF,,, 'SY', 'ST', 'EM'
(section 3.5.11)
BUFF BSS 120
```

3.5.6 WEOF (Write End of File) Macro

This macro writes an end of file on the specified logical unit. It has the general form

```
label WEOF cb,lun,wait
```

where

cb is the address of the control block

lun is the number of the affected logical unit

wait is 1 for an immediate return, or 0 (default value) for a return suspended until the I/O is complete

Example: Write an end of file on logical unit 10. Use the wait option, here specified by default.

```
TAPE EQU 10
.
.
.
EOF WEOF CB,TAPE
.
.
.
```

3.5.7 SREC (Skip Record) Macro

This macro, which applies only to magnetic-tape, card reader, or rotating-memory devices, skips one record in either direction on the specified logical unit. It has the general form

```
label SREC cb,lun,wait,mode
```

where

cb is the address of the control block

lun is the number of the logical unit being manipulated

wait is 1 for an immediate return, or 0 (default value) for a return suspended until the I/O is complete

mode specifies the direction of the skip: 0 (default value) for a forward skip, or 1 for a reverse skip. Reverse skip does not apply to the card reader.

If applied to an RMD, SREC adds or subtracts from the value of word 3 of the FCB (section 3.5.11).

If an attempt is made to apply this macro to a device other than a magnetic-tape or rotating-memory unit, the I/O request is processed internally by the IOC but not by an I/O driver. The IOC indicates the status as I/O complete.

3.5.5 REW (Rewind) Macro

This macro, which applies only to magnetic-tape or rotating-memory devices, repositions the specified logical unit to the beginning-of-unit position. It has the general form

```
label      REW      cb,lun,wait
```

where

cb is the address of the FCB or DCB, which is optional

lun is the number of the logical unit being rewound

wait is 1 for an immediate return, or 0 (default value) for a return suspended until the I/O is complete

Note that the DCB address is an optional parameter, but that the FCB address is mandatory.

To reposition a named file on an RMD, use the OPEN macro (section 3.5.1).

Magnetic-tape devices: REW rewinds the specified unit and, upon successful completion of the task, returns a beginning-of-device (BOD) status.

WARNING

VSIOC returns to the issuing program prior to rewind completion. When rewinding is complete, the issuing RQBLK is updated. Therefore, the issuing RQBLK must not be modified prior to rewind completion.

Rotating-memory devices: REW places the start-RMD-partition and end-RMD-partition addresses in words 5 and 6, respectively, of the FCB (section 3.5.11).

Examples: Rewind logical unit 23, a magnetic-tape unit. Use the wait option, here specified by default.

```
MT      EQU      23      (LUN assigned to magnetic-
      .          .          tape unit)
      .
      .
REW     REW      MT
      .
      .
```

Rewind logical unit 10, an RMD partition. Use the wait option, here specified by default. Note that the REW for an RMD must have an associated FCB (section 3.5.11).

```
DISC   EQU      10      (LUN assigned to RMD
      .          .          partition)
RECL   EQU      120
      .
      .
REW     REW      FCB,DISC
      .
      .
FCB     FCB      RECL,BUFF,,, 'SY', 'ST', 'EM'
      .          .          (section 3.5.11)
BUFF   BSS      120
```

3.5.6 WEOF (Write End of File) Macro

This macro writes an end of file on the specified logical unit. It has the general form

```
label      WEOF      cb,lun,wait
```

where

cb is the address of the control block

lun is the number of the affected logical unit

wait is 1 for an immediate return, or 0 (default value) for a return suspended until the I/O is complete

Example: Write an end of file on logical unit 10. Use the wait option, here specified by default.

```
TAPE   EQU      10
      .
      .
      .
EOF     WEOF      CB,TAPE
      .
      .
      .
```

3.5.7 SREC (Skip Record) Macro

This macro, which applies only to magnetic-tape, card reader, or rotating-memory devices, skips one record in either direction on the specified logical unit. It has the general form

```
label      SREC      cb,lun,wait,mode
```

where

cb is the address of the control block

lun is the number of the logical unit being manipulated

wait is 1 for an immediate return, or 0 (default value) for a return suspended until the I/O is complete

mode specifies the direction of the skip: 0 (default value) for a forward skip, or 1 for a reverse skip. Reverse skip does not apply to the card reader.

If applied to an RMD, SREC adds or subtracts from the value of word 3 of the FCB (section 3.5.11).

If an attempt is made to apply this macro to a device other than a magnetic-tape or rotating-memory unit, the I/O request is processed internally by the IOC but not by an I/O driver. The IOC indicates the status as I/O complete.

INPUT/OUTPUT CONTROL

Example: Skip back one record on logical unit 57, a magnetic-tape unit. Use the immediate-return option.

```

MT      EQU      57  (LUN assigned to magnetic-
                    tape unit)
REV     EQU      1  (Reverse)
IM      EQU      1  (Immediate return)
.
.
SKIP    SREC     CB, MT, IM, REV
.
.
    
```

↓

↑

4	Slew channel control on programmable VFU. Slew printer on specified channel.
7	Advance paper to bottom of form
8	Normal print size*
9	Large print size*

* Only if supported by Statos hardware character generator.

3.5.8 FUNC (Function) Macro

This macro performs a miscellaneous function on a specified logical unit. The function (when present) cannot be defined by any of the preceding I/O control functions. The macro has the general form

```

label    FUNC      dcb, lun, wait
    
```

where

- dcb** is the address of the data control block
- lun** is the number of the logical unit being manipulated
- wait** is 1 for an immediate return, or 0 (default value) for a return suspended until the I/O is complete

FUNC causes certain I/O drivers to perform special functions specified by the function code *fun* in a DCB macro (section 3.5.10):

I/O Driver	Function Code	Function
Card punch	0	Eject blank card
Paper-tape punch	0	Punch 256 blank frames for leader
Line printer and Teletype printer	0	Advance paper to top of next form, or on Teletype 3 lines *
	1	Advance paper one line
	2	Advance paper two lines
	3	Initialize Vertical Forms Unit (VFU). Load form definition into Controller Programmable VFU memory.

I/O Driver	Function Code	Function
Statos	00	Advance paper to top of form
	01	Advance paper one line
	02	Advance paper two lines
	07	Advance paper to bottom of form
	08	Step plotter one raster line
	10	Select small/upright
	11	Small/ +90 degrees
	12	Small/ 180 degrees
	13	Small/ -90 degrees
	14	Large/upright
	15	Large/ +90 degrees
70-755x	16	Large/ 180 degrees
	17	Large/ -90 degrees
	20	Cut paper
	21	End cut
	0	Identify unit (see 3.6.4)
	1	Status of unit

Plot data may be transmitted to the Statos by specifying unformatted mode, 3, in the WRITE macro. Each 1 bit will cause a dot to be printed in its corresponding position in the output line. The most significant bit in the first word output represents the leftmost dot position. Functions 10-21 are processed only if hardware capability is available.

Statos The WRITE macro enables the transfer of one data buffer to the printer/plotter and allows for five different modes of operation:

- Mode 1** -- Compatible line printer (70-6701) mode
- Mode 3** -- Plot (raster) mode (binary raster data transfer)
- Mode 4** -- Print mode selectable size and orientation
- Mode 5** -- Simultaneous print/plot mode (ASCII data transfer)

Mode 6 -- Simultaneous print/plot mode (binary raster data)

All other modes default to mode 1. Modes 4, 5, and 6 are processed only if hardware capability is available.

If an attempt is made to apply the FUNC macro to any other device, the I/O request is processed internally by the IOC but not by an I/O driver. The IOC indicates the status as I/O complete.

Example: Skip two lines on the printer, which is logical unit 5. Use the wait option, here specified by default.

```

LP      EQU      5      (LUN assigned to line
CNT     EQU      2      printer) (Paper-tape
                          channel 2)
      .
      .
      .
UPSP    FUNC     DCB,LP
      .
      .
      .
DCB     DCB      0, 0,CNT
    
```

↓ **Example:** Slew printer to definition mark on VFU channel 7. The printer is logical unit 5.

```

LP      EQU      5      (Lun assigned to
SLEW    DATA    7      line printer)
                          (Slew channel)
      .
      .
      .
      FUNC     SDCB,LP
      .
      .
      .
↑ SDCB   DCB      1,SLEW,4
    
```

3.5.9 STAT (Status) Macro

This macro examines bits 5-6 of the status word in an I/O macro to determine the result of an I/O function request. The STAT macro has the general form

label **STAT** *req,err,aaa,bbb,busy*

where

req is the address of the I/O macro (e.g., READ)

err is the address of the I/O-error routine

aaa is the address of the end of file, beginning of device, or beginning of tape routine

bbb is the address of the end of device or end of tape routine

busy is the address of the I/O-not-complete routine

All parameters (except the label) are mandatory. The contents of the overflow indicator and the A and B registers are saved. Upon normal completion, control returns to the user at the first word after the end of the macro expansion.

CAUTION

Foreground tasks should not loop to check for completion of I/O tasks because this inhibits all lower-level tasks.

Example: Rewind logical unit 12, a magnetic-tape unit, and check for beginning of device (load point). Use the immediate-return option.

```

MT      EQU      12     (LUN assigned to magnetic-
IM      EQU      1     tape unit)
                          (Immediate return)
      .
      .
      .
REW     REW      0,MT,IM (DCB can be omitted
                          for REW)
      .
      .
      .
BUSY    STAT     REW,ERR,BOT,EQT,BUSY
      .
      .
      .
BOT
      .
      .
      .
ERR
    
```

INPUT/OUTPUT CONTROL

3.5.10 DCB (Data Control Block) Macro

This macro generates a DCB as required by I/O macro requests to devices other than RMDs. Note that not all such requests (e.g., rewinding a magnetic-tape unit) require a DCB. The macro has the general form

label	DCB	rl, buff, fun
where		
<i>rl</i>		is the length, in words, of the record to be transmitted
<i>buff</i>		is the address of the user's data area
<i>fun</i>		is the function code for a FUNC request and is unused for other requests (section 3.5.8)

Example: Read a record from logical unit 4, a magnetic-tape unit. Use system binary mode and the immediate-return option. The record length is 60 words, and the address of the user's data area is BUFF.

```

IM      EQU      1      (Immediate return)
BIN     EQU      0      (System binary mode)
MT      EQU      4      (LUN assigned to magnetic-
                        tape unit)
RECL    EQU      60     (Record length 60 words)
.
.
.
MTRD    READ     TAPE, MT, IM, BIN
.
.
.
TAPE    DCB      RECL, BUFF (Data control block)
    
```

3.5.11 FCB (File Control Block) Macro

This macro generates an FCB required by any I/O macro request to an RMD. The macro has the general form

label	FCB	rl, buff, acc, key, 'xx', 'yy', 'zz'
where		
<i>rl</i>		is the length, in words, of the record to be transmitted
<i>buff</i>		is the address of the user's data block
<i>acc</i>		specifies the access method and is 0 (default value) for the direct access by logical record, 1 for sequential access by logical record, 2 for direct access using the relative sector number (beginning

with 1) within the file, 3 for sequential access using the relative sector number within the file, 4 for direct access by physical sector, 5 for sequential access by physical sector, and 9 for sequential access with file extension.

<i>key</i>	is the protection code, if any, required to address that logical unit. This is a single alphanumeric ASCII character coded between single quotation marks (e.g., the protection code H would be coded 'H') or as the eight-bit octal equivalent, in which case no quotation marks are used (e.g., 0310 for the protection code H). The default value is binary zero (not the character 0).
------------	---

<i>xyyzz</i>	is the name of the file being referenced. The file name is one to six ASCII characters, coded in pairs between single quotation marks and separated by commas, e.g., the file named ARRIBA is coded 'AR', 'RI', 'BA'. Embedded blanks are illegal.
--------------	--

Table 3-3 shows the use of FCB words 3, 4, 5, and 6 for the I/O macros.

Example: Create an FCB for the file FILEXX. Use the logical-record-oriented, sequential-access method with a record length of 120 words. The user's data area is BUFF and the protection code is Z.

```

SEQR    EQU      1      (Sequential, record-
                        oriented access)
RECL    EQU      120    (Record length 120
                        words)
.
.
.
DISC    FCB      RECL, BUFF, SEQR, 'Z',
.              'FI', 'LE', 'XX'
.
.
BUFF    BSS      120
    
```

Note that the protection code character Z is coded between single quotation marks, i.e., 'Z', but it can also be coded as the octal value of the ASCII character, in which case no quotation marks are used, i.e., 0332. Thus, the statement given in the example above is equivalent to

```

DISC    FCB      RECL, BUFF, SEQR,
.              0322, 'FI', 'LE', 'XX'
    
```

Table 3-3. FCB Words Under I/O Macro Control

Word	OPEN	READ	WRITE	SREC	CLOSE	REW
Sequential-Access Method						
3	Set to position of current record by mode chosen	Increments record number by one	Increments record number by one	Adds or subtracts one	Put into position of file on directory by mode chosen	Current record set (directory partition) to one or beginning address of logical unit (non-directory partition)
4	Set to current position of file as noted on directory	Checks end of file	No action	Checks end of file	Cleared	Set to ending address of logical unit
5	Set to beginning of file address put in this word	No action	No action	No action	Cleared	Set to beginning address of logical unit (non-directory partition) Skip first directory sector (directory partition)
6	Set to end of file address	No action	No action	No action	Cleared	Set to ending address of logical unit



Table 3-3. FCB Words Under I/O Macro Control (continued)

Word	OPEN	READ	WRITE	SREC	CLOSE	REW
Direct-Access Method						
3	Set to position of current record by mode chosen	No action	No action	No action	Put into position of file on directory by mode chosen	Current record set (directory partition) to one or beginning address of logical unit (non-directory partition)
4	Set to current position of file as noted on directory	No action	No action	No action	Cleared	Set to ending address of logical unit
5	Set to beginning of file address	No action	No action	No action	Cleared	Set to beginning address of logical unit (non-directory partition) Skip first directory sector (directory partition)
6	Set to end of file address	No action	No action	No action	Cleared	Set to ending address of logical unit

3.6 DISK DEVICE I/O OVERVIEW

This section applies only to the 70-755x, 70-7560, 70-7561 and 70-7562 disk devices.

3.6.1 Alternate Sector Partition

Each disk spindle contains an alternate sector partition for that spindle. The first 2n sectors (where n = number of

INPUT OUTPUT CONTROL

partitions per spindle) contain the alternate sector directory for that spindle. The remainder of the alternate sector partition is used for subsequent sector assignments and additional directories if needed. The alternate sector partition is the first partition on the spindle and can either be specified at, or defaulted at system generation. The default size is 1 percent of the spindle capacity. The partition is an unkeyed partition with a partition designator of * (asterisk).

Bad sectors are located and identified during the disk pack formatting operation. The formatter creates a table of bad sectors for the entire disk pack. During system generation, partitions and their associated PST entries are created, alternates to bad sectors are assigned, and the alternate sector table is built. Alternates to bad sectors are always allocated from the alternate sector partition.

3.6.2 Alternate Sector Processing

When a bad sector is detected during a data transfer operation, the alternate sector table for spindle being accessed is read into memory and the table is searched for an entry containing the address of the sector flagged bad. If the entry is not found, the I/O request is terminated with the appropriate error status. Refer to Appendix A for a list of error conditions. When a match is obtained, the following actions are taken by the driver:

1. A channel control block (CCB) specifying command buffer out is created.
2. A command buffer containing the position to transfer data command and an alternate sector address is created.
3. A channel control block specifying data buffer in/out is created for the data buffer corresponding to the alternate sector.
4. The command buffer out CCB is command chained to the data buffer in/out CCB, then an initiate channel function is performed.
5. Upon receipt of the operation complete interrupt, the read/write heads are repositioned to the sector following the bad sector and the data transfer operation is continued to normal completion.

3.6.3 Data Record Blocking And Deblocking

The operating system utilizes records of 120 words and multiples of 120 words for I/O operations on disks. The use of 120-word physical sectors for the disks yields a utilization efficiency of approximately 73%. Increasing the physical sector size improves the disk utilization efficiency by

increasing the data storage area with respect to fixed areas of sectors. The disk controller provides the capability to select one of several different hardware sector sizes through the use of plug-in PROMs. When sector sizes larger than 120 words are selected, the disk driver will perform automatic blocking and deblocking of user data records on all I/O requests, unless this feature is overridden.

On write requests which specify one of the access modes 0 through 3, the disk driver will pack as many whole and partial data records as a hardware sector can contain using the following procedures.

1. User data records which reside on the same hardware sector as the record to be written, are read into memory.
2. The new data record is data-chained to the other records which were read in during step a.
3. All of the data records are written onto the sector.

A write request requiring a blocking operation will take a minimum of 1, a maximum of 2 and an average of 1.5 revolutions to complete.

On read requests which specify one of the access modes 0 through 3, the disk driver will perform automatic deblocking of user data records. On deblocking operations, the driver utilizes the "transfer-in" channel capability of the disk controller to eliminate data on the same sector preceding the desired record. Consequently, an additional revolution of the disk is not required to perform a deblocking operation.

On read and write requests which specify access modes 4 and 5, user data records are read or written starting on hardware sector boundaries. Blocking and deblocking have no significance when these access modes are selected.

3.6.4 Identification Buffer Description

A three-word record containing unit information is placed in a user-supplied buffer in response to an identify function (mode = 0). The format of the three word buffer is:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 0							CN		ua							
Word 1							type				class					
Word 2	size															

SECTION 4

JOB-CONTROL PROCESSOR

The **job-control processor (JCP)** is a background task that permits the scheduling of VORTEX system or user tasks for background execution. The JCP also positions devices to required files, and makes logical-unit and I/O-device assignments.

4.1 ORGANIZATION

The JCP is scheduled for execution whenever an unsolicited operator key-in request to the OC logical unit has a slash (/) as the first character.

Once initiated, the JCP processes all further JCP directives from the SI logical unit.

If the SI logical unit is a Teletype or a CRT device, the message **JC**** is output to indicate the SI unit is waiting for JCP input. The operator is prompted every 15 seconds (by a bell for the Teletype or tone for the CRT) until an input is keyed in.

If the SI logical unit is a rotating-memory-device (RMD) partition, the job stream is assumed to comprise unblocked data. In this case, processing the job stream requires an /ASSIGN directive (section 4.2.6).

A JCP directive has a maximum of 80 characters, beginning with a slash. Directives input on the Teletype are terminated by the carriage return.

All JCP directives are echoed to the SO logical unit if SI ≠ SO. All directives, except /C and /P have the time of day appended onto the front of the directive when echoed to SO. The format is

HH:MM:SS /JCP directive

4.2 JOB-CONTROL PROCESSOR DIRECTIVES

This section describes the JCP directives:

a. Job-initiation/termination directives:

/JOB	Start new job
/ENDJOB	Terminate job in progress
/FINI	Terminate JCP operation
/C	Comment
/P	Pause
/MEM	Allocate extra memory for background task

b. I/O-device assignment and control directives:

/AFILE	Assign RMD file and open file.
/ASSIGN	Make logical-unit assignment(s)
/SFILE	Skip file(s) on magnetic-tape unit

/SREC	Skip record(s) on magnetic-tape unit or RMD partition
/WEOF	Write end-of-file mark
/REW	Rewind magnetic-tape unit or RMD partition
/PFILE	Position rotating memory-unit file
/FORM	Set line count on LO logical unit
/KPMODE	Set keypunch mode
/OPEN	Open VTAM line or terminal
/CLOSE	Close VTAM line or terminal
/CFILE	Close file on global logical unit

c. Language-Processor directives:

/DASMR	Schedule DAS MR assembler
/FORT	Schedule FORTRAN compiler
/RPG	Schedule RPG II compiler
/DBGEN	Schedule TOTAL system
/COBOL	Schedule COBOL compiler

d. Utility directives:

/CONC	Schedule system-concordance program
/SEDT	Schedule symbolic source-editor task
/FMAIN	Schedule file-maintenance task
/LMGEN	Schedule load-module generator
/IOUTIL	Schedule I/O-utility processor
/SMAIN	Schedule system-maintenance task
/COMSY	Schedule compression/edit system
/FMUTIL	Schedule file utility task

e. Program-loading directives:

/EXEC	Schedule loading and execution of a load-module from the SW unit file
/LOAD	Schedule loading and execution of a user background task
/ALTLIB	Schedule the next background task from the specified logical unit rather than from the background library
/PLOAD	Schedule loading and execution of a user background task at priority level 1
/DUMP	Dump background at completion of task execution
/TRACE	Invoke trace mode (V77-800 only)

JCP directives begin in column 1 and comprise sequences of character strings having no embedded blanks. The character strings are separated by commas (,) or by equal signs (=). The directives are free-form and blanks are permitted between the individual character strings of the directive, i.e., before or after commas (or equal signs). Although not required, a period (.) is a line terminator. Comments can be inserted after a period.

Each JCP directive begins with a slash (/).

The general form of a job-control statement is

/name,p(1),p(2),...,p(n)

JOB-CONTROL PROCESSOR

where

name is one of the directive names given (any other character string produces an error)

each $p(n)$ is a parameter required by the JCP or by the scheduled task and defined below under the descriptions of the individual directives

Numerical data can be octal or decimal. Each octal number has a leading zero.

For greater clarity in the descriptions of some directives, optional periods, optional blank separators between character strings, and the optional replacement of commas by equal signs are omitted from descriptions.

Error messages applicable to JCP directives are given Appendix A.4.

4.2.1 /JOB Directive*

This directive initializes all background system pointers and flags, and stores the job name if one is specified. It has the general form

/JOB, name

where *name* is the name of the job and comprises up to eight ASCII characters (additional characters are permitted but ignored by the JCP).

The job name, if any, is then printed at the top of each page for all VORTEX background programs.

The occurrence of the /JOB directive causes the scheduling of the background task V\$ACT1. V\$ACT1 is a dummy task on BL which only performs an EXIT. However, V\$ACT1 may be replaced by a user task to perform any desired accounting function.

Example: Initialize the job TASKONE.

/JOB, TASKONE

4.2.2 /ENDJOB Directive*

This directive initializes all background system pointers and flags, and clears the job name. It has the form

/ENDJOB

The occurrence of the /ENDJOB directive causes the scheduling of the background task V\$ACT2. V\$ACT2 is a dummy task on BL which only performs an EXIT. However, V\$ACT2 may be replaced by a user task to perform any desired accounting function.

Example: Terminate the job in process.

/ENDJOB

4.2.3 /FINI (Finish) Directive*

This directive terminates all JCP background operations and makes an EXIT request to the real-time executive RTE component (section 2.1.11). It has the form

/FINI

To reschedule JCP after a FINI, input any JCP directive from the OC unit

The occurrence of the /FINI directive causes the scheduling of the background task V\$ACT3. V\$ACT3 is a dummy task on BL which only performs an EXIT. However, V\$ACT3 may be replaced by a user task to perform any desired accounting function.

Example: Terminate JCP operations.

/FINI

- ↓
- * The JCP directives JOB, ENDJOB, and FINI reset all logical unit table 1 units to their default (system) values. These directives also reinitialize the global FCBs (SI, PI, PO, BI, BO, SS, GO) to default record sizes based on assignment: to 40 words for non-RMD units, or to 120 words for RMD units. JOB and ENDJOB do not set the SI logical unit. Logical unit 1 (OC) is not reset by the occurrence of /JOB, /ENDJOB, or /FINI.
- ↑

4.2.4 /C (Comment) Directive

This directive outputs the specified comment to the SO and LO logical units, thus permitting annotation of the listing. It is not otherwise processed. It has the general form

/C, comment

where *comment* is any desired free-form comment.

Example: Annotate a listing with the comment *Rewind all mag tapes.*

/C, REWIND ALL MAG TAPES

4.2.5 /MEM (Memory) Directive

This directive assigns additional 512-word blocks of main memory to the next scheduled background task. It has the general form

/MEM,n

where *n* is the number of 512-word blocks of main memory to be assigned.

/MEM permits larger symbol tables for FORTRAN compilations and DAS MR assemblies.

The total area of the 512-word blocks of memory plus the background program itself cannot be greater than the total area available for background and nonresident foreground tasks. An attempt to exceed this limit causes the scheduled task to be aborted.

Example: Allocate an additional 1,024 words of main memory to the next scheduled task.

/MEM, 2

4.2.6 /ASSIGN Directive

This directive equates and assigns particular logical units to specific I/O devices. It has the general form

/ASSIGN,l(1)=r(1),l(2)=r(2),...,l(n)=r(n)

where

each *l(n)* is a logical-unit number (e.g., 102) or name (e.g., SI)

each *r(n)* is a logical-unit number or name, or a physical-device system name (e.g., TY00, table 17-1)

The logical unit to the left of the equal sign in each pair is assigned to the unit/device to the right.

If the controller and unit numbers are omitted from the name of a physical device, controller 0 and unit 0 are assumed. When *r(n)* is alphabetic and consists of two characters, it is first interpreted as a logical name and then as an abbreviated physical name. If a matching logical name exists, the assignment will be made using that unit.

An inoperable device, i.e., one declared down by the ;DEVDN operator key-in request (section 17.2.10), cannot be assigned. A logical unit designated as unassignable cannot be reassigned.

Example: Assign the PI logical unit to card reader CR00 and the LO logical unit to Teletype TY00.

/ASSIGN,PI=CR,LO=TY

4.2.7 /SFILE (Skip File) Directive

This directive, which applies only to magnetic-tape units and card readers, causes the specified logical unit to move the tape forward the designated number of end-of-file marks. It has the general form

/SFILE,lun,neof

where

lun is the number or name of the affected logical unit

neof is the number of end-of-file marks to be skipped

If the end-of-tape mark is encountered before the required number of files has been skipped, the JCP outputs to the SO and LO logical units the error message **JC05,nn**, where **nn** is the number of files remaining to be skipped.

Example: Skip three files on the BI logical unit.

/SFILE, BI, 3

4.2.8 /SREC (Skip Record) Directive

This directive, which applies only to magnetic-tape units, card readers, and RMDs, causes the specified logical unit to move the tape the designated number of records in the required direction. In the case of RMDs, word 4 of the FCB is adjusted the appropriate number of records. It has the general form

/SREC,lun,nrec,direc

where

lun is the number or name of the affected logical unit

nrec is the number of records to be skipped

direc indicates the direction to be skipped; F (default value) for forward, or R for reverse. Reverse skip does not apply to the card reader.

If a file mark (forward skip only), end of tape, or beginning of tape is encountered before the required number of records has been skipped, the JCP outputs to the SO and LO logical units the error message **JC05,nn**, where **nn** is the number of records remaining to be skipped.

Note: File marks are not detected on a reverse skip.

JOB-CONTROL PROCESSOR

Example: Skip nine records forward on the BO logical unit.

```
/SREC,BO,9
```

4.2.9 /WEOF (Write End of File) Directive

This directive writes an end-of-file mark on the specified logical unit. It has the general form

```
/WEOF,lun
```

where **lun** is the number or name of the affected logical unit. (Not accepted for RMD.)

Example: Write an end-of-file mark on the BO logical unit.

```
/WEOF,BO
```

4.2.10 /REW (Rewind) Directive

This directive, which applies only to magnetic-tape units and RMDs, causes the specified logical unit(s) to rewind to the beginning of tape. It has the general form

```
/REW,lun,lun,...,lun
```

where **lun** is the number or name of a logical unit to be rewound.

Example: Rewind the BO and PI logical units.

```
/REW,BO,PI
```

4.2.11 /PFILE (Position File) Directive

This directive, which applies only to RMDs and MT assigned to global logical units causes the specified logical unit to move to the beginning of the designated file. It has the general form

```
/PFILE,lun,key,name
```

where

lun is the number or name of the affected logical unit. The logical unit must be one of the system defined logical units which has a global FCB

key is the protection code required to address **lun**

name is the name of the file to which the logical unit is to be positioned

Global file control blocks: There are eight global file control blocks (FCB, section 3.5.11) in the VORTEX system that are reserved for background use. System background and user programs can reference these global FCBs. The /PFILE directive stores **key** and **name** in the corresponding FCB before opening/rewinding the logical unit. To pass the buffer address and size of the record to the corresponding logical-unit FCB, make an RTE IOLINK service request (section 2.1.13). The names of the global FCBs are *SIFCB*, *PIFCB*, *POFCB*, *SSFCB*, *BIFCB*, *BOFCB*, *GOFCB*, and *LOFCB*, where the first two letters of the name indicate the logical unit.

Example: Position the PI logical unit to beginning of file FILEXY, whose protection key is \$.

```
/PFILE,PI,$,FILEXY
```

4.2.12 /FORM Directive

This directive sets the specified line count on the LO logical unit. This is the number of lines printed by DAS MR assembler or FORTRAN compiler before a top of form is issued. The directive has the general form

```
/FORM,lines
```

where **lines** is the number (from 5 to 9999, inclusive) of lines to be printed before a top of form is issued.

The default value of **lines** is defined at system-generation time. If the directive contains a value outside the legal range, the default value is used.

Example: Set a line-count value of 100.

```
/FORM,100
```

4.2.13 /KPMODE (Keypunch mode) Directive

This directive specifies the mode, 026 or 029, (BCD or EBCDIC respectively) in which VORTEX is to read and punch cards. It has the general form

KPMODE,m

where *m* is 0 for 026 mode, or 1 for 029 mode.

Example: Specify that cards be read and punched in 029 keypunch mode.

`/KPMODE, 1`

4.2.14 /DASMR (DAS MR Assembler) Directive

This directive schedules the DAS MR assembler (section 5.1) with the specified options for background operation on priority level 1. It has the general form

`/DASMR,p(1),p(2),...,p(n)`

where each *p(n)*, if any, is a single character specifying one of the following options:

Parameter	Presence	Absence
B	Suppresses binary object	Output binary object
L	Outputs binary object on GO file	Suppresses output of binary object on GO file
M	Suppresses symbol-table listing	Output symbol-table listing
N	Suppresses source listing	Outputs source listing
E	Assembles V75 extended instructions.	Flags V75 extended instructions with '*OP error'.
I	Flags implicit indirect instructions with '*II error'.	Assembles implicit indirect instructions.

The `/DASMR` directive can contain up to four such parameters in any order.

The DAS MR assembler reads source records from the PI logical unit on the first pass. The PI unit must have been set to the beginning of device before the `/DASMR` directive. This can be done with an `/ASSIGN` (section 4.2.6), `/SFILE`

(section 4.2.7), `/REW` (section 4.2.10), or `/PFILE` (section 4.2.11) directive.

A load-and-go operation requires, in addition, an `/EXEC` directive (section 4.2.22).

Example: Schedule the DAS MR assembler with no source listing, but with binary-object output on the GO file.

`/JOB,EXAMPLE`
`/PFILE,BO,,BO`
`/DASMR,N,L`

`/JOB` initializes the GO file to start of file. If `BO` is assigned to a rotating memory partition, a `/PFILE,BO,,BO` must precede the `/DASMR` directive to initialize the file (unless the assembly is part of a stacked job - see section 4.3 for sample deck setup).

4.2.15 /FORT (FORTRAN Compiler) Directive

This directive schedules the FORTRAN compiler (section 5.3) with the specified options for background operation on priority level 1. It has the general form

`/FORT,p(1),p(2),...,p(n)`

where each *p(n)*, if any, is a single character specifying one of the following options:

Parameter	Presence	Absence
A	Produce DASMR listing	Suppress DASMR listing
B	Suppresses binary object	Output binary object
D	Assigns two words to integer array items and to integer and logical variables (ANSI standard)	Assigns one word to integer array items and to integer and logical variables
H	Generate code using Floating-Point Processor (FPP)	Generate no FPP instructions
L	Outputs binary object on GO file	Suppresses output of binary object on GO file
M	Suppresses symbol-table listing	Outputs symbol-table listing

JOB-CONTROL PROCESSOR

N	Suppresses source listing	Outputs source listing
O	Outputs object-module listing	Suppresses object-module listing
X	Compiles conditionally	Compiles normally
F	Generates code with calls to faster firmware routines (see section 20.2)	Generates subroutine calls

The /FORT directive can contain any or all such parameters in any order.

Sample deck formats are illustrated in section 4.3.

The FORTRAN compiler reads source records from the PI logical unit. The PI unit must have been set to the beginning of device before the /FORT directive. This can be done with an /ASSIGN (section 4.2.6), /SFILE (section 4.2.7), /REW (section 4.2.10), or /PFILE (section 4.2.11) directive.

A load-and-go operation requires, in addition, an /EXEC directive (section 4.2.22).

Example: Schedule the FORTRAN compiler with binary-object, source, symbol-table, and object-module listings; normal compilation; and no binary-object output on the GO file.

`/FORT, O`

4.2.16 /CONC (System Concordance) Directive

This directive schedules the system concordance program (section 5.2) for background operation. It has the form

`/CONC, L`

where L is an optional parameter to request that all symbols in a source program be listed. Normally, CONC only lists those symbols which were referenced.

The concordance program inputs from the SS logical unit and uses the same source statements that are input to the

DAS MR assembler. It outputs to the LO logical unit a listing of all symbols and their referenced locations in the same input program.

The SS unit is set to the beginning of device before the /CONC directive.

Example: Schedule the system concordance program.

```
/ASSIGN, PI=MT00  
/REW, PI  
/DASMR  
/PFILE, SS, , SS  
/CONC, L
```

4.2.17 /SEdit (Source Editor) Directive

This directive schedules the symbolic source editor (section 8) for background operation on priority level 1. It has the form

`/SEdit`

Example: Schedule the symbolic source editor.

`/SEdit`

4.2.18 /FMAIN (File Maintenance) Directive

This directive schedules the file maintenance task (section 9) for background operation on priority level 1. It has the form

`/FMAIN`

Example: Schedule the file maintenance task.

`/FMAIN`

4.2.19 /LMGEN (Load-Module Generator) Directive

This directive schedules the load-module generator (section 6) for background operation on priority level 1. A memory map is output unless suppressed. The directive has the general form

/LMGEN, M, X

where

- M** if present, suppresses the output of a memory map.
- X** if present, specifies to print both an alphabetical and numerical sorted map.

Example: Schedule the load-module generator task without a memory map.

/LMGEN, M

4.2.20 /IOUTIL (I/O Utility) Directive

This directive schedules the I/O utility processor (section 10) for background operation on priority level 0. The directive has the form

/IOUTIL

Example: Schedule the I/O utility processor.

/IOUTIL

4.2.21 /SMAIN (System Maintenance) Directive

This directive schedules the system maintenance task (section 16) for background operation on priority level 1. The directive has the form

/SMAIN

Example: Schedule the system maintenance task.

/SMAIN

4.2.22 /EXEC (Execute) Directive

This directive schedules the load-module loader to load and execute a load module from the SW logical unit file. Since this is not a VORTEX system task, execution is on priority level 0. The directive has the general form

/EXEC, D

Where D, if present, dumps all of the background upon completion of execution. The dump format consists of eight memory locations per line. Both octal and ASCII representation appear in the dump. During ASCII dump non-ASCII characters appear as blanks. ASCII dump is suppressed if dump is to a TY or CT device.

The dump format consists of eight memory locations per line as follows:

XXXXXX AAAAAA BBBBBB... HHHHHH

where XXXXXX is the starting memory address location of the eight following data words and AAAAAA through HHHHHH are the octal values of those locations. The occurrence of an asterisk between two lines indicates that all dump lines between those lines have the same value as the previous line.

/EXEC can be used to create a load module (named SW) on the SW logical unit and then schedule it, or to execute an existing load module on the SW logical unit. The action taken depends on the setting of bit 2 of the low core flag V\$JCPF. If the bit is set, LMGEN is scheduled to read binary from the GO logical unit and catalog the task on SW. If the bit is not set, the current load module on SW is executed. This bit is set by performing a "load-and-go" assembly or compilation using the "L" option flag. This bit is cleared by the loading of any background program. (**Note:** JCP directives which do not load tasks, for example, /ASSIGN, /PFILE, do not clear this bit.). The load module on SW may be executed at anytime until SW is modified (i.e., another load-and-go, LMGEN, COMSY, or any other task that writes to SW).

Example: Schedule the loading of a user load module from the SW unit file without a background dump.

/EXEC

Schedule a FORTRAN load-and-go operation.

/FORT, L
/EXEC

When a dump has been specified the dump will be output to the List Output unit after the task exits or is aborted. Once the dump has started, it may be terminated by use of the Operator Communication ;ABORT. When the dump is aborted in this manner, it is required that the executing task be aborted by a previous action.

Example:

- /EXEC, D** Executes a load module from SW unit file requesting background dump on exit
- ;ABORT, SW** Causes the task to abort and dump the background
- ;ABORT, JPDUMP** Causes the background dump to be aborted
- ;ABORT, SW** Causes the task to be released and JCP to be reloaded

JOB-CONTROL PROCESSOR

Note: When executing load and go, /EXEC has additional optional parameters. These parameters are used to replace (R1,...,R4) or insert (I1,I2,I3) "canned" LMGEN directives. Standard LMGEN canned load and go directives are (see section 6 for further explanation):

```
TIDB, SW, 1, 0
LD, GO, , GO
LIB
END
```

As LMGEN processes the canned directives (if insert or replace have been specified), LMGEN will process further directives from device SI until an end-of-file is input. The end-of-file will resume LMGEN processing of the remainder of the canned directives.

Insert and replace parameters must be entered in the following order:

```
R1, I1, R2, I2, R3, I3, R4
```

Note: Any number of parameters may be used.

Example: Request load and go execution, replacing the canned LMGEN LIB directive.

```
/EXEC, R3
```

The device PI specifies a library search from user library logical unit 115 with protection code M. Directives will be processed as follows:

	Canned	PI
1.	TIDB, SW, 1, 0	
2.	LD, GO, , GO	
3.		LIB, 115, m
4.		End of File
5.	END	

4.2.23 /LOAD Directive

This directive schedules a user task, which must be present in the background library or alternate library, for background execution on priority level 0. The directive has the general form

```
/LOAD,name,p(1),p(2),...,p(3)
```

where

name	is the name of the user task being scheduled
each <i>p(n)</i> (if any)	is a parameter required by the user task

Each parameter specified, if any, will be in the job-control buffer when the user task is scheduled. The parameter string, which can extend to the end of the 80-character buffer, will appear in the buffer exactly as it does in the input directive. The address of the first word of the parameter string is in location V\$JCB.

Example: Schedule the user task TSKONE with parameters ALPHA1 and ALPHA2.

```
/LOAD, TSKONE, ALPHA1, ALPHA2
```

4.2.24 /ALTLIB (Alternate Library) Directive

This directive replaces the background library with the specified alternate library unit as the unit from which a background task is to be loaded. The directive has the general form:

```
/ALTLIB,lun,key
```

where

lun is the number or name of the alternate library logical unit

key is the protection code required to address lun

This directive affects the loading of the next task which would normally be loaded from the background library. It affects the loading of VORTEX language processors and utility tasks in addition to user tasks loaded with the /LOAD directive.

It has no effect on a /EXEC directive. After execution of the background task, the background library is restored as the logical unit from which background tasks are to be loaded.

Example: Schedule the user task TSKONE from logical unit 25, protection key N.

```
/ALTLIB, 25, N
/LOAD, TSKONE
```

4.2.25 /DUMP Directive

This directive causes all of background to be dumped upon completion of execution of a task executed from the background library or an alternate library. The dump format is the same as the format for /EXEC,D (see section 4.2.22).

Example: Schedule the execution of user task TSKONE with a dump at completion of execution.

```
/DUMP
/LOAD, TSKONE
```

4.2.26 /CFILE Directive

This directive, which applies only to RMDs and MTs assigned to global logical units, causes the file currently attached to the global FCB file on a logical unit to be closed with an update. It has the general form.

```
/CFILE,lun
```

where

lun is the name or number of the affected logical unit. The logical unit must be one of the global logical units.

Example: Close the file currently attached to the PO global FCB.

```
/CFILE,PO
```

4.2.27 /DBGEN (Data Base Generator) Directive

This directive schedules the Data Set Generator Program (see TOTAL Manual for more detailed information) for background operation on priority level 1. It has the form

```
/DBGEN
```

Example: Schedule the Data Base Generator for TOTAL.

```
/DBGEN
```

4.2.28 /PLOAD Directive

This directive schedules a user task, which must be present in the background library or alternate library, for background execution on priority level 1. The directive has the general form

```
/PLOAD,xxxxxx,p(1),p(2),...p(n)
```

where

xxxxxx is the name of the user task being scheduled. The name must not contain numeric characters.

p(n) is a parameter required by the user task.

Each parameter specified, if any, will be in the job-control buffer when the user task is scheduled. The parameter string, which can be extended to the end of the 80-character buffer, will appear in the buffer exactly as it does in the input directive. The address of the first word of the parameter string is in location V\$JCB.

4.2.29 /FMUTIL Directive

This directive causes files, directories, and/or partitions to be dumped or loaded from RMD's or magnetic tapes, and schedules the file maintenance utility (section 21) for background operation on priority level 1. The directive has the form

```
/FMUTIL
```

Examples: Schedule File Maintenance Utility.

```
/FMUTIL
```

4.2.30 /RPG (RPG II Compiler) Directive

This directive schedules the RPG II compiler (section 5.5) with the specified options for background operations on priority level 1. It has the general form

```
/RPG,p(1),p(2),...,p(n)
```

where

p(n) is a single character specifying one of the following options:

Parameter	Presence	Absence
B	Suppresses binary object.	Output binary object.
D	Include RPG debug features in object module.	Suppress debug features.
L	Outputs binary object on GO file.	Suppresses output of binary object on GO file.

JOB-CONTROL PROCESSOR

The /RPG directive can contain up to five such parameters in any order.

Sample deck formats are illustrated in section 4.3.

The RPG II compiler reads source records from the PI logical unit. The PI unit must have been set to the beginning of device before the /RPG directive. This can be done with an /ASSIGN (section 4.2.6), /SFILE (section 4.2.7), /REW (section 4.2.10), or /PFILE (section 4.2.11) directive.

Example: Schedule the RPG II compiler with binary object, source, and symbol-table listings; normal compilation; and no binary object output on the GO file.

```
/RPG
```

Example: Schedule RPG II for normal compilation but with binary object output on the GO file instead of the BO file.

```
/RPG, L, B
```

4.2.31 /P (Pause) Directive

This directive outputs the specified comment to the SO and LO logical units and then causes JCP to be suspended. In addition to the specified comment, instructions are output to SO on how to resume JCP. It has the general form

```
/P,comment
```

where

comment is any desired free-form comment.

Example: Request that the current job requires MT # 800 on MT00 before it continues.

```
/P, Mount MT #800 on MT00
```

in addition, JCP will output:

```
Pause---WHEN READY, TYPE --;RESUME, JCP
```

4.2.32 /AFILE Directive

This directive equates and assigns the specified global logical unit to the specified RMD device and then opens (with rewind) the specified file. It has the general form

```
/AFILE,I(1),I(2),key,filename
```

where

I(1) is the global logical unit name or number.
I(2) is the logical unit name or number of the RMD device to be assigned.
key is the protect code.
filename is the name of the file to which the logical unit is to be positioned.

See note on global files in section 4.2.11.

Example: Position the SI logical unit to file PROG1 on logical unit 30 (no protect key).

```
/AFILE,SI,30,,PROG1
```

↓

4.2.33 /TRACE DIRECTIVE

The trace facility is a debugging aid available only on V77-800 systems. It permits the user to verify and, under certain user-specified conditions, to print register contents on the LO device following each instruction executed in the traced program.

The /TRACE directive causes the next scheduled background task to be executed in the TRACE mode on the V77-800 system. This directive has the format

```
/TRACE
```

Example: Schedule the execution of the user task TSKONE in the TRACE mode.

```
/TRACE  
/LOAD,TSKONE
```

Foreground tasks can also be executed in the TRACE mode by including the parameter T in the OPCOM ;SCHED command. For detailed information regarding the use of the trace facility, see section 7.4.

↑

4.3 SAMPLE DECK SETUPS

The batch-processing facilities of VORTEX are invoked by JCP control directives in combination with programs and data. These elements form the input job stream to VORTEX. The input job stream can come from various peripherals and be carried on various media. These examples illustrate common job streams and deck-preparation techniques.

Example 1 - Card Input: Compile a FORTRAN IV main program (with source listing and octal object listing), and assemble a DAS MR subprogram. Then load and execute the linked program.

```

/JOB, EXAMPLE1
/FORT, L, O
.
.
.
(Source Deck)
.
/DASMR, L
.
(Source Deck)
.
.
.
/EXEC
/ENDJOB

```

Example 2 - Card Input: Assemble a DAS MR program (with source listing and load-and-execute) and generate a concordance listing. The DAS MR program is cataloged on RMD partition D00K under file name USER1 with protection key U. Assign the PI logical unit to RMD partition D00K, open file name USER1 for the assembler, assemble the program, and execute the program with a dump.

```

/JOB, EXAMPLE2
/ASSIGN, PI=D00K
/PFILE, PI, U, USER1
/DASMR, L
/PFILE, SS, , SS
/CONC
/EXEC, D
/ENDJOB

```

Example 3 - Card Input: Assemble a DAS MR program (with source listing and object-module output on the BO logical unit). Assign the PI logical unit to magnetic-tape unit MT00, the PO logical unit to dummy device, the SS logical unit to the PI logical unit, the BO logical unit to RMD partition D00J, and output the object module to file name USER2 with no protection key. Before assembly, position the PI logical unit to the third file. Allocate four additional 512-word blocks for the DAS MR symbol-table area.

```

/JOB, EXAMPLE3
/ASSIGN, PI=MT00, PO=DUM, SS=PI, BO=D00J
/REW, PI
/SFILE, PI, 2

```

```

/PFILE, BO, , USER2
/MEM, 4
/DASMR
/ENDJOB

```

Example 4 - Card Input: After generation of a VORTEX system, use FMAIN to initialize and add object modules to the object-module library (OM) with protection key D. Assign the BI logical unit to CR00.

```

/JOB, EXAMPLE4
/ASSIGN, BI=CR00
/FMAIN
INIT, OM, D
INPUT, BI
ADD, OM, D
.
.
.
(Object Modules)
.
(2-7-8-9 EOF Card)
.
.
.
/ENDJOB

```

Example 5 - Card Input: Load and go operation. Compile a FORTRAN IV main program, a subprogram and assemble a DASMR subprogram. Save output on BO. Execute the linked programs.

```

/JOB, EXAMPLE5
/PFILE, BO, , BO
/FORT, L
.
.
.
(Source deck FORTRAN main program)
.
(Source deck FORTRAN subprogram)
.
/DASMR, L
.
(Source deck DASMR subprogram)
.
.
.
/EXEC
/FINI

```


SECTION 5 LANGUAGE PROCESSORS

The VORTEX operating system supports three language processors: the *DAS MR assembler* (section 5.1), the *FORTRAN IV compiler* (section 5.3), and the *RPG IV compiler* (section 5.4), plus the ancillary *concordance program* (section 5.2.).

5.1 DAS MR Assembler

DAS MR is a two-pass assembler scheduled by job-control directive `/DASMR` (section 4.2.14). DAS MR uses the secondary storage device unit for pass 1 output. It reads a source module from the PI logical unit and outputs it on the PO unit. The source input for pass 2 is entered from the SS logical unit.

When an END statement is encountered, the SS unit is repositioned and reread. During pass 2, the output can be directed to the BO and/or GO units for the object module and the LO unit for the assembly listing. The SS or PO file, which contains a copy of the source module, can be used as input to a subsequent assembly.

A DAS MR symbol consists of one to six characters, the first of which must be alphabetic, with the rest alphabetic or numeric. Additional alphanumeric characters can be appended to the first six characters of the symbol to form an extended symbol up to the limit imposed by a single line of code. However, only the first six characters are recognized by the assembler.

DAS MR symbols may also be formed from the pound sign, exclamation mark or dollar sign, in initial and other positions.

Since the DAS MR assembler is used within the VORTEX system under VORTEX I/O control, the VORTEX user can specify the desired I/O devices. However, the PO and SS logical units must be the same magnetic-tape unit or RMD partition. Except when PI is equal to SS as shown in section 4.3 (example 3).

DAS MR has a symbol-table area for 175 symbols at five words per symbol. To increase this area, input before the `/DASMR` directive a `/MEM` directive (section 4.2.5), where each 512-word block enlarges the capacity of the table by 100 symbols.

A VORTEX physical record on an RMD is 120 words. Source records are blocked three 40-word records per VORTEX physical record, and object modules are blocked two 60-word modules per record. However, in the case where $SI = PI = RMD$, records are not blocked but assumed to be one per VORTEX physical record. When an input file contains more than one source module each new source module must start at a physical record boundary. Unused portions of the last physical record of the previous source modules should be padded with blank records. Proper blocking may

be ensured by following the END statement of the previous source module with two blank records.

Detailed references to the DAS MR assembly language are given in the appropriate Sperry Univac reference manuals (see section 1.3). These references include descriptions of the directives recognized by the assembler (table 5-1), except for the title directive which is discussed below. DAS MR will assemble the entire V70 extended instruction set if the E parameter is specified in the `/DASMR` directive.

Table 5-1. Directives Recognized by the DAS MR Assembler

BES	IFF
BSS	IFT
CALL	LIST
COMN	LOC
CONT	MAC
DATA	MZE
DETL	NAME
DUP	NLIS
EJEC	NULL
END	OPSY
EMAC	ORG
ENTR	PZE
EQU	RETU
EXT	SET
FORM	SPAC
GOTO	SMRY
	TITLE

Error messages applicable to the DAS MR assembler are given in Appendix A.5.1.

5.1.1 TITLE Directive

This directive changes the title of the assembly listing and the identification of the object program. It has the general form

TITLE *symbol*

where *symbol* is the new title of the assembly listing; the label field being ignored by the assembler. There are a maximum of eight characters in *symbol*.

At the beginning of assembler pass 1, the title of the assembly listing and the identification of the object program are initialized as blanks. When a TITLE directive is encountered, title and identification assume the *symbol* given in the directive.

Examples: Entitle the assembly listing and object program NEWTITLE.

TITLE **NEWTITLE**

Reinitialize the title and identification, obliterating the old title.

TITLE

LANGUAGE PROCESSORS

5.1.2 VORTEX Macros

The DAS MR assembler contains macro definitions for the real-time executive (RTE, section 2.1) and I/O control (IOC, section 3.5) macros. Figure 5-1 illustrates these definitions.

```
*
M1      MAC
        EXT      V$IOC
        JSR      0404,1
        DATA    0100000
F       FORM     1,3,4,8
        F        P(1),P(2),P(3),P(4)
        DATA    P(5),0,0
        EMAC

*
*       VORTEX READ MACRO DEFINITION
*       READ     DCB,LUN,W,M
*               WHERE DCB = FCB OR DCB ADDRESS
*                   LUN = LOGICAL UNIT NO.
*                   W = WAIT OPTION
*                   M = I/O MODE
READ    MAC
M1      P(3),P(4),0,P(2),P(1)
        EMAC

*
*       VORTEX WRITE MACRO DEFINITION
*       WRITE    DCB,LUN,W,M
*               WHERE DCB = FCB OR DCB ADDRESS
*                   LUN = LOGICAL UNIT NO.
*                   W = WAIT OPTION
*                   M = I/O MODE
WRITE   MAC
M1      P(3),P(4),1,P(2),P(1)
        EMAC

*
*       VORTEX WRITE END OF FILE MACRO DEFINITION
*       WEOF     DCB,LUN,W
*               WHERE DCB = FCB OR DCB ADDRESS
*                   LUN = LOGICAL UNIT NO.
*                   W = WAIT OPTION
WEOF    MAC
M1      P(3),0,2,P(2),P(1)
        EMAC

*
*       VORTEX REWIND MACRO DEFINITION
*       REW      DCB,LUN,W
*               WHERE DCB = FCB OR DCB ADDRESS
*                   LUN = LOGICAL UNIT NO.
*                   W = WAIT OPTION
REW     MAC
M1      P(3),0,3,P(2),P(1)
        EMAC

*
*       VORTEX SKIP RECORD MACRO DEFINITION
*       SREC     DCB,LUN,W,M
*               WHERE DCB = FCB OR DCB ADDRESS
*                   LUN = LOGICAL UNIT NO.
*                   W = WAIT OPTION
*                   M = I/O MODE
```

Figure 5-1. VORTEX Macro Definitions for DAS MR

```

SREC      MAC
          M1      P(3),P(4),4,P(2),P(1)
          EMAC

*
*
* VORTEX FUNCTION MACRO DEFINITION
* FUNC      DCB,LUN,W
*           WHERE DCB = FCB OR DCB ADDRESS
*           LUN = LOGICAL UNIT NO.
*           W = WAIT OPTION
FUNC      MAC
          M1      P(3),0,5,P(2),P(1)
          EMAC

*
*
* VORTEX OPEN MACRO DEFINITION
* OPEN     FCB,LUN,W,M
*           WHERE FCB = FCB OR DCB ADDRESS
*           LUN = LOGICAL UNIT NO.
*           W = WAIT OPTION
*           M = I/O MODE
OPEN     MAC
          M1      P(3),P(4),6,P(2),P(1)
          EMAC

*
*
* VORTEX CLOSE MACRO DEFINITION
* CLOSE    FCB,LUN,W,M
*           WHERE FCB = FCB OR DCB ADDRESS
*           LUN = LOGICAL UNIT NO.
*           W = WAIT OPTION
*           M = I/O MODE
CLOSE    MAC
          M1      P(3),P(4),7,P(2),P(1)
          EMAC

*
*
* VORTEX STATUS MACRO DEFINITION
* STAT     FCB,ERR,EOF,EOD,BUSY
*           WHERE FCB = FCB OR DCB ADDRESS
*           ERR = ERROR RETURN ADDRESS
*           EOF = END OF FILE, BEGINNING
*           OF DEVICE, OR BEGINNING OF
*           TAPE RETURN ADDRESS
*           EOD = END OF DEVICE OR END OF TAPE
*           RETURN ADDRESS
*           BUSY = BUSY RETURN ADDRESS
STAT     MAC
          EXT     V$IOST
          JSR     0373,1
          DATA   P(1),P(2),P(3),P(4),P(5)
          EMAC

*
*
* VORTEX DEVICE CONTROL BLOCK MACRO DEFINITION
* DCB      RL,BUF,CNT
*           WHERE RL = RECORD LENGTH
*           BUF = DATA ADDRESS
*           CNT = COUNT
DCB      MAC
          DATA   P(1),P(2),P(3)
          EMAC

```

Figure 5-1. VORTEX Macro Definitions for DAS MR (continued)

LANGUAGE PROCESSORS

```

*      VORTEX FILE CONTROL BLOCK MACRO DEFINITION
*      FCB      RL,BUF,AC,KEY,'N1','N2','N3'
*              WHERE RL = RECORD LENGTH
*              BUF  = DATA ADDRESS
*              AC   = ACCESS METHOD
*              KEY  = PROTECTION KEY
*              N1  = FIRST 2 ASCII FILE NAME
*              N2  = SECOND 2 ASCII FILE NAME
*              N3  = THIRD 2 ASCII FILE NAME
FCB      MAC
        DATA      P(1),P(2)
F        FORM      6,2,8
        F          0,P(3),P(4)
        DATA      0,0,0,0,P(5),P(6),P(7)
        EMAC

*
M2      MAC
        EXT        V$EXEC
        JSR        0406,1
        EMAC

*
*      VORTEX SCHEDULE MACRO DEFINITION
*      SCHED    PL,W,LUN,KEY,'N1','N2','N3',F
*              WHERE PL = PRIORITY LEVEL
*              W      = WAIT OPTION
*              LUN    = LOGICAL UNIT NO.
*              KEY    = PROTECTION KEY
*              N1    = FIRST 2 ASCII TASK NAME
*              N2    = SECOND 2 ASCII TASK NAME
*              N3    = THIRD 2 ASCII TASK NAME
*              F      = TIDB ADDRESS FLAG
SCHED    MAC
        M2
F        FORM      2,1,1,6,1,5
        F          0,P(8),P(2),1,0,P(1)
F        FORM      8,8
        F          P(4),P(3)
        DATA      P(5),P(6),P(7)
        EMAC

*
*      VORTEX EXIT MACRO DEFINITION
*      EXIT
EXIT      MAC
        M2
        DATA      0200
        EMAC

*
*      VORTEX SUSPEND MACRO DEFINITION
*      SUSPND   T
*              WHERE T = TYPE OF SUSPENSION
SUSPND   MAC
        M2
F        FORM      4,6,5,1
        F          0,3,0,P(1)
        EMAC

*
*      VORTEX RESUME MACRO DEFINITION
*      RESUME   'N1','N2','N3'
*              WHERE N1 = FIRST 2 ASCII TASK NAME
*              N2  = SECOND 2 ASCII TASK NAME
*              N3  = THIRD 2 ASCII TASK NAME

```

Figure 5-1. VORTEX Macro Definitions for DAS MR (continued)

```

RESUME      MAC
            M2
            DATA      0400,P(1),P(2),P(3)
            EMAC

*
*
*           VORTEX ABORT MACRO DEFINITION
*
*           ABORT      'N1','N2','N3'
*                       WHERE N1 = FIRST 2 ASCII TASK NAME
*                       N2 = SECOND 2 ASCII TASK NAME
*                       N3 = THIRD 2 ASCII TASK NAME
ABORT      MAC
            M2
            DATA      0500,P(1),P(2),P(3)
            EMAC

*
*
*           VORTEX ALLOCATE MACRO DEFINITION
*
*           ALOC      ADDR
*                       WHERE ADDR = ADDRESS OF REENTRANT
*                               SUBROUTINE
ALOC      MAC
            M2
            DATA      0600,P(1)
            EMAC

*
*
*           VORTEX DEALLOCATE MACRO DEFINITION
*
*           DEALOC
DEALOC     MAC
            M2
            DATA      0700
            EMAC

*
*
*           VORTEX PRIORITY INTERRUPT MASK MACRO DEFINITION
*
*           PMSK      NUM,MSK,TYP
*                       WHERE NUM = PIM NUMBER
*                       MSK = PIM LINE MASK
*                       TYP = ENABLE OR DISABLE TYPE
PMSK      MAC
            M2
F1         FORM      4,6,5,1
            F1       0,010,0,P(3)
F          FORM      8,8
            F        P(1),P(2)
            EMAC

*
*
*           VORTEX DELAY MACRO DEFINITION
*
*           DELAY     T5,TM,DT
*                       WHERE T5 = DELAY TIME IN 5 MILLI-
*                               SECOND INCREMENTS
*                       TM = DELAY TIME IN 1 MINUTE
*                               INCREMENTS
*                       DT = DELAY TYPE
DELAY     MAC
            M2
F          FORM      4,6,4,2
            F        0,011,0,P(3)
            DATA    P(1),P(2)
            EMAC

```

Figure 5-1. VORTEX Macro Definitions for DAS MR (continued)

LANGUAGE PROCESSORS

```

*
*   VORTEX LDELAY MACRO DEFINITION
*   LDELAY      T5, TM, LUN,KEY
*               WHERE T5 = DELAY TIME IN 5-MILLISECOND
*                   INCREMENTS
*                   TM = DELAY TIME IN 1-MINUTE
*                   INCREMENTS
*                   LUN = LOGICAL UNIT NUMBER FOR TASK LOAD
*                   KEY = PROTECTION KEY
LDELAY      MAC
           M2
           DATA      01107,P(1),P(2)
           FORM       8,8
           F          P(4),P(3)
           EMAC
*
*   VORTEX TIME REQUEST MACRO DEFINITION
*   TIME
*   TIME
TIME      MAC
           M2
           DATA      01200
           EMAC
*
*   VORTEX OVERLAY MACRO DEFINITION
*   OVLAY      TF, 'N1', 'N2', 'N3'
*               WHERE TF = TYPE FLAG
*                   N1 = FIRST 2 ASCII TASK NAME
*                   N2 = SECOND 2 ASCII TASK NAME
*                   N3 = THIRD 2 ASCII TASK NAME
OVLAY     MAC
           M2
F         FORM       4,6,5,1
           F          0,013,0,P(1)
           DATA     P(2),P(3),P(4)
           EMAC
*
*   VORTEX IOLINK MACRO DEFINITION
*   IOLINK     LUN,BUF,NUM
*               WHERE LUN = LOGICAL UNIT NO.
*                   BUF = USER'S BUFFER LOCATION
*                   NUM = BUFFER SIZE
IOLINK    MAC
           M2
F         FORM       4,6,6
           F          0,014,P(1)
           DATA     P(2),P(3)
           EMAC
*
*   VORTEX PASS MACRO DEFINITION
*   PASS      COUNT, FROM, TO
*               WHERE COUNT = WORD COUNT
*                   FROM = FROM ADDRESS
*                   TO = TO ADDRESS

```

Figure 5-1. VORTEX Macro Definitions for DAS MR (continued)


```

*
PASS      MAC
          M2
F         FROM      4,6,6
          F         0,016,0
          DATA     P(1),P(2),P(3)
          EMAC
*
*         VORTEX TBEVNT MACRO DEFINITION
*         TBEVNT   VALUE, , DISP, ,C/S
*
*         WHERE
*
*         VALUE = IS A BIT MASK
*
*         DISP  = IS THE TIDB WORD TO BE ALTERED.
*               IT IS EXPRESSED BY WAY OF A NUMBER,
*               THE DISPLACEMENT (OR POSITION) OF THIS
*               WORD IN THE TIDB.
*
*         C/S   = IS THE CLEAR/SET INDICATION (0 = CLEAR,
*               1 = SET)
*
*         OPTIONS:      BOTH DISP AND C/S ARE OPTIONAL AND
*                       THE DEFAULT FOR BOTH IS 0.
*
*         IMPLEMENTATION:
*                       WHEN DISP = 0 THE ACTION DEPENDS ON
*                       THE VALUE OF VALUE:
*
*                       VALUE, IF 0-177776, IS SET INTO
*                       THE REQUESTING TASK'S TIDB TBEVNT
*                       WORD. IF VALUE IS 0177777, RETURN
*                       IS WITH THE REQUESTOR'S TBEVNT IN
*                       THE A REGISTER
*
*                       WHEN DISP = 0, DISP WILL BE ALTERED
*                       ACCORDING TO VALUE AND C/S.
*
*                       C/S = 0, ALL THE BITS IN DISP CORRESPONDING
*                       TO THE ZERO (0) BITS IN VALUE
*                       WILL BE RESET TO 0.
*
*                       C/S = 1, ALL THE BITS IN DISP CORRESPONDING
*                       TO THE ONE (1) BITS IN VALUE
*                       WILL BE SET TO 1.
*
TBEVNT   MAC
          M2
          DATA     01700
          DATA     P(1),P(2),P(3)
          EMAC
*

```

Figure 5-1. VORTEX Macro Definitions for DAS MR (continued)

LANGUAGE PROCESSORS

```

*      VORTEX ALLOCATE PAGE MACRO DEFINITION
*      ALOCPG      N,LOGICAL ADDR,REJECT ADDR
*                  WHERE N = NUMBER OF PAGES TO ALLOCATE
*                  LOGICAL ADDR = LOGICAL ADDRESS
*                               MODULO 01000, WHERE
*                               PAGES ARE ALLOCATED
*                  REJECT ADDR = ERROR RETURN ADDRESS
*
ALOCPG      MAC
            M2
            DATA      02000
            DATA      P(1)
            DATA      P(2)
            DATA      P(3)
            EMAC

*
*      VORTEX DEALLOCATE PAGE MACRO DEFINITION
*      DEALPG     N,LOGICAL ADDR,REJECT ADDR
*                  WHERE N = NUMBER OF PAGES TO DEALLOCATE
*                  LOGICAL ADDR = LOGICAL ADDRESS,
*                               MODULO 01000, WHERE
*                               PAGES ARE TO BE
*                               DEALLOCATED
*                  REJECT ADDR = ERROR RETURN ADDRESS
*
DEALPG     MAC
            M2
            DATA      02100
            DATA      P(1)
            DATA      P(2)
            DATA      P(3)
            EMAC

*
*      VORTEX MAPIN MACRO DEFINITION
*      MAPIN      N,LOGICAL ADDR,BUFFER ADDR,REJECT ADDR
*                  WHERE N = NUMBER OF PAGES TO BE MAPPD
*                  LOGICAL ADDR = LOGICAL ADDRESS, MODULO
*                               01000, WHERE PAGES ARE TO
*                               BE ALLOCATED
*                  BUFFER ADDR = PHYSICAL PAGE NUMBER
*                               OR BUFFER ADDRESS CON-
*                               TAINING PHYSICAL PAGES
*                               TO BE MAPPED
*                  REJECT ADDR = ERROR RETURN ADDRESS
*
MAPIN      MAC
            M2
            DATA      02200
            DATA      P(1)
            DATA      P(2)
            DATA      P(3)
            DATA      P(4)
            EMAC

```

Figure 5-1. VORTEX Macro Definitions for DAS MR (continued)

```

*      VORTEX PAGE NUMBER MACRO DEFINITION
*      PAGNUM      LOGICAL ADDR
*                               WHERE LOGICAL ADDR = ADDRESS WITHIN THE
*                               REQUESTING TASK'S VIRTUAL
*                               MEMORY WHERE IDENTIFICATION
*                               OF THE ASSIGNED PHYSICAL
*                               PAGE IS REQUIRED
*
PAGNUM      MAC
            M2
            DATA      02300
            DATA      P(1)
            EMAC

```

Figure 5-1. VORTEX Macro Definitions for DAS MR (continued)

LANGUAGE PROCESSORS

5.1.3 Assembly Listing Format

Figure 5-2 is a sample listing following the format described in this section.

Page format: The assembly listing is limited to the number of lines per page specified by the VORTEX resident

constant V\$PLCT, with each line containing no more than 120 characters. Each page has a page number and title line followed by one blank line, and then the program listing containing two lines less than the number specified by V\$PLCT. (This specification can be changed through the job-control processor (JCP).)

```

PAGE 23 01/22/72 PROG1 VORTEX DASMR V$JCP
      588 EJEC
      589 *
      590 * SUBROUTINE PRINTS JCP DIRECTIVE ON SO AND LO DEVICE
      591 *
000660 074056 A 592 JCPRT STX JSPRX
000661 064056 A 593 STB JCPRB
000662 010412 A 594 LDA V$JCB GET BUFFER ADDRESS
000663 005311 A 595 DAR
000664 054003 A 596 STA *+4 SETUP LOFCB
      597 IOLINK LO,*,41
000665 006505 A
000666 000604 E
000667 001405 A
000670 000665 R
000671 000051 A
000672 030400 A 598 LDX V$LUT1 ADRS OF LOG UNIT TBL
000673 015003 A 599 LDA SO,X
000674 150463 A 600 ANA BM377 SO CUR ASSIGNMT
000675 054274 A 601 STA JCTA
000676 015002 A 602 LDA SI,X
000677 150463 A 603 ANA BM377 SO CUR ASSIGNMT
000700 144271 A 604 SUB JCTA SO, SI SAME LUN
000701 001010 A 605 JAZ JCPR1
000702 000714 R
000703 017000 I 606 LDA JCFBCS+3 STORE 'LOFCB' ADRS IN CALL
000704 054004 A 607 STA *+5
      608 WRITE LOFCB,SO,0,1 NO - WRITE TO SO
000705 006505 A
000706 000630 E
000707 100000 A
000710 010403 A
000711 000633 E
000712 000000 A
000713 000000 A
000714 030400 A 609 JCPRT LDX V$LUT1
000715 015005 A 610 LDA LO,X
000716 150463 A 611 ANA BM377 LO CUR ASSIGNMT
000717 144252 A 612 SUB JCTA LO, SO SAME LUN
000720 001010 A 613 JAZ JCPRE YES
000721 000733 R
000722 017000 A 614 LDA JCFCBS+3 STORE 'LOFCB' ADRS IN CALL
000723 054004 A 615 STA *+5
      616 WRITE LOFCB,LO,0,1 NO - WRITE TO LO

```

Figure 5-2. Sample Assembly Listing

At the end of the assembly, the following information is printed after the END statement:

- a. A line containing the subheading ENTRY NAMES
- b. All entry names (in four columns), each preceded by its value and a flag to denote whether the symbol is absolute (A), relocatable (R), or common (C).
- c. A line containing the subheading EXTERNAL NAMES
- d. All external names (in four columns), each preceded by its value and a flag to denote that the symbol is external (E)
- e. A line containing the subheading SYMBOL TABLE
- f. The symbol table (in four columns), each symbol preceded by its value and a flag to denote whether the symbol is absolute (A), relocatable (R), common (C), or external (E)
- g. A line containing the subheading mmmm ERRORS ASSEMBLY COMPLETE, where mmmm is the accumulated error count expressed as a decimal integer, right-justified and left-blank-filled

Line format: Beginning with the first character position, the format for a title line is:

- a. One blank
- b. The word PAGE
- c. One blank
- d. Four character positions that contain the decimal page number
- e. Two blanks
- f. Eight character positions that contain the current date obtained from the VORTEX resident constant V\$DATE
- g. Two blanks
- h. Eight character positions that contain the program identification obtained from the VORTEX resident constant V\$JNAM
- i. Two blanks
- j. The word VORTEX
- k. Two blanks
- l. The word DASMR
- m. Two blanks
- n. Eight character positions that contain the program title from the TITLE directive
- o. Blanks through the 120th character position

Beginning with the first character position, the format for an assembly line is:

- a. One blank
- b. Six character positions to display the location counter (octal) of the generated data word
- c. One blank
- d. Six character positions to display the generated data word (octal)
- e. One blank
- f. One character position to denote the type of generated data word: absolute (A), relocatable (R), common (C), external (E), literal (L), or indirect-address pointer generated by the assembler (I)
- g. One blank
- h. Four character positions containing the decimal symbolic source statement line number, right-justified and left-blank-filled
- i. One blank
- j. Eighty character positions that contain the image of the symbolic source statement. (If the symbolic source statement is not a comment statement, the label, operation, and variable fields are reformatted into symbolic source statement character positions 1, 8, and 16, respectively. If commas separate the label, operation, and variable fields, they are replaced by blank characters.)
- k. Blanks, if necessary, through the 120th character position

Error Chaining: If syntax errors occur during an assembly error, chaining is provided to assist in finding the errors. If errors occur, the error message at the end of the assembly contains a decimal value within parentheses corresponding to the source line number at which the last error occurred. The line number referenced in turn references the next line number containing an error. The last line number containing an error does not have a chaining reference. If no errors occurred, the error message does not contain a chaining reference.

5.2 CONCORDANCE PROGRAM

The background **concordance program (CONC)** provides an indexed listing of all source statement symbols, giving the number of the statement associated with each symbol and the numbers of all statements containing a reference to the symbol. CONC is scheduled by job-control directive /CONC (section 4.2.16). Upon completion of the concordance listing, control returns to the JCP via EXIT.

Input to CONC is through the SS logical unit. The concordance is output on the LO unit. CONC uses system

LANGUAGE PROCESSORS

global file control block SSFCB. If the SS logical unit is an RMD, a /REW or /PFILE directive (section 10) establishes the FCB before the /CONC directive is input to the JCP.

CONC has a symbol-table area to process 400 no-reference symbols at five words per symbol, plus 400 referenced symbols (averaging five references per symbol) at ten words per symbol. To increase this area, input before the /CONC directive a /MEM directive (section 4.2.5), where each 512-word block enlarges the capacity of the table by approximately 75 symbols.

CONC processes both packed records (three source statements per 120-word VORTEX physical record) and unpacked records (one source statement per record).

5.2.1 Input

CONC receives source-statement input from the SS logical unit. There is, however, no positioning of the SS unit prior to reading the first record. The source statements are identical with those input to the VORTEX assembler and thus conform to the assembler syntax rules.

As the inputs are read, each source statement is assigned a line number, 1, 2, etc., which is identical with that printed on the assembly listing. When a symbol appears in the label field of a symbolic source statement, the line number of that source statement is assigned to the symbol. When the symbol appears in the variable field of a source statement, the line number of that statement is used as a reference for the symbol.

5.2.2 Output

CONC outputs the concordance listing on the LO logical unit. Output begins when one of the following events occurs:

- a. CONC processes the source statement END
- b. Another job-control directive is input
- c. An SS end of file or end of device is found
- d. A reading error is found
- e. *The symbol-table area is filled*

If the output occurred because the symbol-table area of memory was full, CONC clears the concordance tables, outputs error message CN01, and continues until one of the other terminating conditions is encountered. In all other cases, CONC terminates by calling EXIT.

The concordance listing is made in the order of the ASCII values of the characters comprising the symbols.

Beginning with the first character position, the format for a title line is:

- a. One blank
- b. The word PAGE
- c. One blank
- d. Four character positions that contain the decimal page number
- e. Two blanks
- f. Eight character positions that contain the date obtained from the VORTEX resident constant V\$DATE
- g. Two blanks
- h. Eight character positions that contain the program identification obtained from the VORTEX resident constant V\$JNAM
- i. Two blanks
- j. The word VORTEX
- k. Two blanks
- l. The word CONC
- m. Blanks through the 72nd character position

Beginning with the first character position, the format for a concordance cross-reference listing is:

- a. Two blanks
- b. Four character positions that contain the decimal line number of the source statement assigned to the symbol in item (e) below
- c. One blank
- d. One character position containing an asterisk (*) if there are no references to that symbol (otherwise blank)
- e. Six character positions containing the symbol being listed
- f. Two blanks
- g. Four character positions that contain the decimal line number of a source statement referencing the symbol in item (e) above
- h. Items (f) and (g) are repeated as necessary for each source statement referencing the symbol in item (e) above, where up to nine references are placed on the first line, and subsequent references on the next line(s). Continuation lines that may be required for ten or more references to the same symbol do not repeat items (a) through (e)
- i. Blanks through the 72nd character position of the last line of the entry

Figure 5-3 illustrates the concordance listing.

PAGE	1	09/22/71	V\$OPCM	VORTEX	CONC						
509	B		841	859	879	990	1001	1002	1012	1068	1072
			1074	1112	1230	1231					
261	B10	*									
262	B11	*									
263	B12	*									
1206	ODATE		1180	1182	1190						
1937	ONUM		895	928	936	1017	1182	1190	1196	1254	1284
			1406	1418							

Figure 5-3. Sample Concordance Listing

5.3 FORTRAN IV COMPILER

The **FORTRAN IV compiler** is a one-pass compiler scheduled by job-control directive /FORT (section 4.2.15). The compiler inputs a source module from the PI logical unit and produces an object module on the BO and/or GO units and a source listing on the LO unit. No secondary storage is required for a compilation.

If a fatal error is detected, the compiler automatically terminates output to the BO and GO units. LO unit output continues. The compiler reads from the PI unit until an END statement is encountered or a control directive is read. Compilation also terminates on detection of an I/O error or an end-of-device, beginning-of-device, or end-of-file indication from I/O control.

The output comprises relocatable object modules under all circumstances: main programs and subroutines, function, and block-data subprograms.

Error messages applicable to the FORTRAN IV compiler are given in Appendix A.5.2.

FORTRAN IV has conditional compilation facilities implemented by an X in column 1 of a source statement. When the X appears in the /FORT directive, all source statements with an X in column 1 are compiled (the X appears on the LO listing as a blank). When the X is not present, all conditional statements are ignored by the compiler. X lines are assigned listing numbers in either case, but the source statement is printed only when the X is present.

FORTRAN IV has a symbol-table area for approximately 70 symbols (i.e., names), if none of the logical units used is assigned to an RMD device. Each RMD assignment requires buffer space of 120 words (except when BO = GO = RMD, in which case BO and GO use the same buffer) and the symbol capacity is reduced by 24 symbols per buffer. To increase the symbol-table area, input before the /FORT directive a /MEM directive (section 4.2.5), where each 512-word block enlarges the capacity of the table by 100 symbols. If a larger symbol-table is used, greater subexpression optimization is possible.

A VORTEX physical record on an RMD is 120 words. Source records are blocked three 40-word records per VORTEX physical record, object modules are blocked two 60-word modules per record, and list modules are output one record per physical record. However, in the case where SI = PI =

RMD, records are not blocked but assumed to be one per VORTEX physical record. When the file contains more than one source module, each new source module must start at a physical record boundary. The unused portion of the last physical record of the previous module should be padded with blanks.

Table 5-2 lists the VORTEX real-time executive (RTE) service request macros available through FORTRAN IV. These macros are detailed in section 2.1.

Table 5-2. RTE Macros Available Through FORTRAN IV

ABORT	EXIT	SCHED
ALOC	OVLAY	SUSPND
DELAY	PMSK	TIME
LDELAY	RESUME	PASS

5.3.1 FORTRAN IV Enhancements

The VORTEX FORTRAN IV language additions and enhancements make the VORTEX FORTRAN compiler more consistent with IBM FORTRAN (level G). Except for these additions and enhancements, FORTRAN compilation and execution with the VORTEX operating system is the same as with the Master Operating System (MOS) described in the FORTRAN IV Reference Manual (98 A 9902 03x).

FORTRAN-complied programs can execute in either foreground or background.

Detailed information on the VORTEX FORTRAN IV language additions and enhancements are given in the VORTEX FORTRAN IV Reference Manual (98 A 9902 04x).

5.3.1.1 Variables

VORTEX FORTRAN IV variables are identifiers which consist of a string of one to six alphanumeric characters and correspond to the type of data the variable represents. Variables are classified into the following five fundamental types: INTEGER, REAL, DOUBLE PRECISION, COMPLEX, and LOGICAL.

The following list shows each variable type with its associated standard and optional length (in bytes):

LANGUAGE PROCESSORS

Variable Type	Standard	Optional
INTEGER	2	4
REAL	4	8
COMPLEX	8	-
LOGICAL	2	-
DOUBLE PRECISION	8	-

5.3.1.2 Constants

There are four categories of VORTEX FORTRAN IV constants: NUMERICAL, LOGICAL LITERAL, and HEXADECIMAL. These four constant data constructions are discussed below.

NUMERICAL constants are integer, real, or complex numbers. Integer constants may be positive, zero, or negative. If the constant has no sign, it is interpreted as representing a positive value. If a zero is specified, with or without a preceding sign, the sign will have no effect on the value zero. The constant has the general form

sn

where

s is the optional signed character (+ or -).
 n is a decimal character string (maximum magnitude is 1073741823).

LOGICAL constants allow for the use of logical operations through the medium of the logical expression. Thus, two logical constants are provided to represent the "true" and "false" logical values. The constant has the general form

.TRUE. or **.FALSE.**

LITERAL constants are a string of alphanumeric and/or special characters. If apostrophes delimit the literal, a single apostrophe within the literal is represented by two apostrophes. The number of characters in a string, including blanks, may not be less than 1 or greater than 255. Blanks within the character string will be considered part of the string. The constant has the general form

wHs or $'s'$

where

w is a positive non-zero constant denoting the width of the character string.
 s denotes the character string.

HEXADECIMAL constant consists of the letter Z followed by 1 to 16 hexadecimal digits. The constant has the general form

Zn

where

n is a 1 to 16 hexadecimal digit string.

The maximum number of digits allowed in a hexadecimal constant depends on the length specification of the variable being initialized. If the number of digits is greater than the maximum, the left-most digits are truncated. If the number is less than the maximum, the left-most positions are filled with zeros.

5.3.1.3 IMPLICIT Statement

The IMPLICIT statement must be the first statement in a main program or the second statement in a subprogram. The statement enables the user to specify the type, including length of all variables, arrays, and function names. The statement has the general form

IMPLICIT type $*s(a1, \dots)$

where

type is a type name.
 $*s$ is optional; and, represents one of the permissible length specifications (see variable).
 a is an initial character string (A, B, ..., Z, \$,) in that order.

5.3.1.4 Explicit Type Statements

The Explicit Type Specification statement declares the type of variable, function name, statement function name, or array by its name rather than by its initial character. Optionally, it may also initialize the variable. The statement overrides the IMPLICIT statement, which in turn overrides the predefined convention. The statement has the general form

type $*s$ $a1*s1(k1)/x1, \dots$

where

type is a type name.
 $*s$ is optional; and, represents one of the permissible length specifications.
 a is a variable, array, or function name.
 (k) is optional; and, gives dimension information for arrays. When the TYPE statement in which it appears is in a subprogram, k may contain

integer variables of length 2 (section 5.3.1.1), provided that the array is a dummy argument.

/x/ is optional; and, represents initial data values (see DATA statement).

5.3.1.5 DOUBLE PRECISION Statement

The DOUBLE PRECISION statement overrides any specification of a variable made by either the predefined convention or the IMPLICIT statement. The statement has the general form

DOUBLE PRECISION *a(k),...*,

where

a represents a variable, array, or function name.

(k) is optional; and, is composed of one to seven unsigned integer constants that represent the maximum value of each subscript in the array. *k* may contain integer variables of length 2, provided that the array is a dummy argument.

5.3.1.6 PAUSE Statement

The execution of the PAUSE statement causes the unconditional suspension (SUSPND) of the object program being executed pending operator action. To resume the suspended task, input the operator-communication key-in request RESUME. The statement has the general form

PAUSE
or
PAUSE *n* or **PAUSE** *m*

where

n is a string of one to five decimal digits.

m is a literal constant enclosed in apostrophes.

5.3.1.7 STOP Statement

The execution of the STOP statement causes the unconditional termination of the execution of the object program being executed. The statement has the general form

STOP
or
STOP *n* or **STOP** *m*

where

n is a string of one to five decimal digits.

m is a literal constant enclosed in apostrophes.

5.3.1.8 CALL Statement

The execution of the CALL statement causes the specified subroutine to be executed. The CALL statement arguments must agree in number and order of appearance with the dummy arguments in the SUBROUTINE statement. The statement has the general form

CALL *name (a1,a2),...*,

where

name is the name of a SUBROUTINE subprogram.

a is an actual argument that is being supplied to the SUBROUTINE subprogram. The argument may be a variable array element, array name, literal, or arithmetic or logical expression. Each *a* may also be of the form *&n*, where *n* is a statement number.

5.3.1.9 RETURN Statement

The RETURN statement provides the method by which the calling program is reentered following the execution of a subprogram. The normal sequence of execution following the RETURN statement of a SUBROUTINE subprogram is to the next statement following the CALL statement in the calling program. The statement has the general form

RETURN or **RETURN** *i*

where

i is an integer constant or variable whose value, for example *n*, denotes the *n*-th asterisk in the argument list of a SUBROUTINE statement. RETURN *i* may be specified only in a SUBROUTINE subprogram.

5.3.1.10 READ/WRITE Statements

VORTEX FORTRAN IV allows two optional parameters to the READ/WRITE statements. These optional parameters allow for conditional exits on an end-of-data or transmission error.

LANGUAGE PROCESSORS

Example: READ(4,10,ERR = 105,END = 200)A,B

In the above example, control will be transferred to statement 105 if an I/O error occurs, or to statement 200 if an end-of-data occurs on unit 4.

5.3.1.11 ENCODE/DECODE Statement

ENCODE/DECODE statements perform data conversion according to a FORMAT statement without performing external I/O operations. ENCODE statement takes an I/O list, converts each element and places it in a specified buffer. DECODE statement words from the buffer into the I/O list. For example:

```
DIMENSION I(40)
READ(CDR,10)I
10 FORMAT(40A2)
   DECODE(10,20,I)K,L
20 FORMAT(2I5)
```

These statements read an ASCII card image into array I. The first two fields of five ASCII characters are then decoded into their integer equivalent and placed into the variables K and L.

5.3.1.12 Direct-Access INPUT/OUTPUT Statements

The direct-access INPUT/OUTPUT statements allows a programmer to go directly to any point in a file which resides on an RMD, and process a record without having to process all the records within the file. To use direct-access INPUT/OUTPUT statements (READ, WRITE, and FIND), the file(s) to be operated on must be described with a DEFINE FILE statement. The statement has the general form

```
DEFINE FILE a1(m1,r1,f1,v1),...
```

where

- a specifies the unit number.
- m represents the relative position of a record within the file.
- r specifies the maximum size of each record in the file.
- f specifies whether the file is to be read or written with or without format control.
- v specifies an integer variable (not an array element) called an associated variable, which

points to the record immediately following the last record transmitted.

5.3.1.13 Direct-Access READ Statement

The READ statement causes data to be transferred from a direct-access device into internal storage. The statement has the general form

```
READ(a'r,b,ERR = Ec)list
```

where

- a specifies the unit number and must be followed by an apostrophe.
- r represents the relative position of a record within the file.
- b is optional; and, if given, is either the statement number of the FORMAT statement, or the name of an array that contains an object-time format.
- ERR = Ec is optional; and, specifies the number of a statement to which control is given when an error condition is encountered
- list is optional; and, is an I/O list. The I/O list must not contain the associated variable.

5.3.1.14 Direct-Access WRITE Statement

The WRITE statement causes data to be transferred from internal storage to a direct-access device. The statement has the general form

```
WRITE (a'r,b)list
```

where

- a specifies the unit number and must be followed by an apostrophe.
- r represents the relative position of a record within the file.
- b is optional; and, if given, is either the statement number of the FORMAT statement, or the

name of an array that contains an object-time format.

list is optional; and, is an I/O list. The list must not contain the associated variable.

5.3.1.15 FIND Statement

The FIND statement causes the next input record to be found while the present record is being processed. The statement has the general form

FIND (a'r)

where

a specifies the unit number and must be followed by an apostrophe.

r represents the relative position of a record within the file.

At the conclusion of a FIND operation, the associated variable points to the record found.

5.3.1.16 DATA Statement

The DATA statement is used to define initial values of variables, array elements, and arrays. This statement cannot precede any specification statement that refers to the same variables, array elements, or arrays. The DATA statement may not precede an IMPLICIT statement. It has the general form

DATA k/d/...

where

k is a list containing variables, array elements, or array names.

d is a list of constants (integer, real, complex, hexadecimal, logical, or literal), any of which may be preceded by i^* , where i^* indicates that the constant is to be specified i times.

5.3.1.17 TITLE Statement

The TITLE statement declares a module name which is output to the top of each page of the source listing and to the object module. It has the general form

TITLE name

where

name is the title to be output. The title contains up to eight characters, and is output in the object text as the name by which the program is to be referenced by SMAIN.

If a TITLE statement is used, it must be the first source statement. A TITLE statement forces a page eject on the LO listing.

5.3.1.18 Subprogram Multiple Entry

VORTEX FORTRAN IV facilitates multiple entry into SUBROUTINE and FUNCTION subprograms by specifying a CALL statement or a FUNCTION reference that refers to an ENTRY statement in the subprogram. Entry is made at the first executable statement following the ENTRY statement. The statement has the general form

ENTRY name(a1,a2,a3),...

where

name is the name of an entry point.

a is a dummy argument corresponding to an actual argument in a CALL statement or FUNCTION reference.

5.3.1.19 SUBROUTINE Subprogram

The SUBROUTINE subprogram may contain any FORTRAN IV statement except a FUNCTION statement, another SUBROUTINE statement, or an BLOCK DATA statement. If an IMPLICIT statement is specified, it must immediately follow the SUBROUTINE statement. SUBROUTINE has the general form

SUBROUTINE name(a1,a2,a3),...

where

name is the SUBROUTINE name.

a is a distinct dummy argument. Each argument used must be a variable or array name, the dummy name of another SUBROUTINE, FUNCTION subprogram, or an asterisk "*" which denotes a return point specified by a statement number in the calling program.

The actual arguments can be:

LANGUAGE PROCESSORS

- A literal, arithmetic, or logical constant
- Any type of variable or array element
- Any type of array name
- Any type of arithmetic or logical expression
- The name of a FUNCTION or SUBROUTINE subprogram
- A statement number

5.3.1.20 FUNCTION Subprogram

The FUNCTION subprogram is an independent subprogram consisting of a FUNCTION statement and at least one RETURN statement. It has the general form

```
type FUNCTION name*s(a1,a2,a3),...
```

where

type	is INTEGER, REAL, DOUBLE PRECISION, COMPLEX, or LOGICAL. Its inclusion is optional.
name	is the name of the FUNCTION.
*s	represents one of the permissible length specifications.
a	is a dummy argument or dummy SUBROUTINE name or other FUNCTION subprogram.

5.3.1.21 Subscripts

A subscript is a set of integer subscript quantities that are associated with an array name to identify a particular element of the array. A maximum of seven subscript quantities, separated by commas, can appear in a subscript. The following rules apply to the construction of subscript quantities:

- Subscript quantities may contain arithmetic expressions that use any of the arithmetic operators: +, -, *, /, **
- Subscript quantities may contain FUNCTION references
- Subscript quantities may contain array elements
- Integer and real mixed-mode expressions within subscript quantities are evaluated according to normal

FORTRAN rules. If the evaluated expression is real, it is converted to integer

- The evaluated result of a subscript quantity should always be greater than zero

5.3.1.22 Z Format Code

The hexadecimal Z format code causes a string of hexadecimal digits to be interpreted as a hexadecimal value and to be associated with the corresponding I/O list element for purposes of data transmitting. It has the general form

```
Zw
```

where

w denotes a string of hexadecimal digits. The maximum value that can be read is FFFFFFFFFFFFFFFF

On input, if an input field contains an odd number of digits, the number will be padded on the left with a hexadecimal zero when it is stored.

On output, if the number of characters in the storage location is less than w, the left-most print positions are filled with blanks. If the number of characters in the storage location is greater than w, the left-most digits are truncated and the rest of the number is printed.

5.3.2 Execution-Time I/O Units

All FORTRAN I/O statements (FORTRAN IV manual) include a FORTRAN unit number (FUN) or name, which may or may not be identical with the logical unit containing the required file(s). Four different cases of FORTRAN units must be distinguished as indicated in figure 5-4.

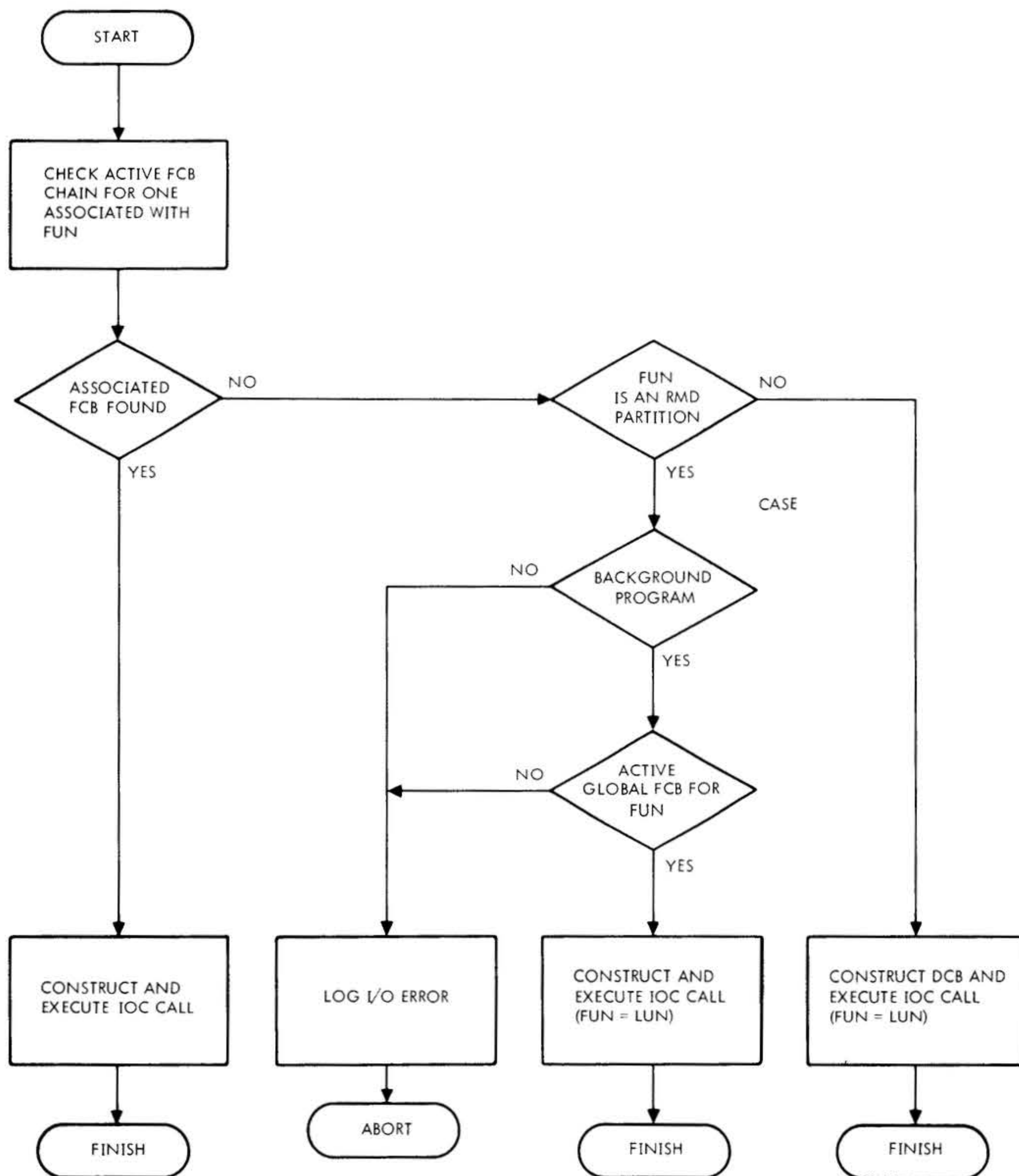
Case 1, non-RMD unit: The logical-unit number is assigned to the device by SGEN (section 15) or by the JCP /ASSIGN directive (section 4.2.6), where the FORTRAN unit number is identical with that of the file unit. Thus, to rewind the PO logical unit (unit 10, magnetic-tape unit 0), the job stack can be:

```
•  
•  
•  
/ASSIGN, PO=MT00  
/FORT  
•  
•  
•  
REWIND 10  
•  
•
```

Case 2, RMD file executing in background only: The JCP /PFILE directive (section 4.2.11) positions the PI unit to a background reassignable logical unit, and loads a global FCB. As in case 1, the FORTRAN unit number is identical with that of the file unit. Thus, to read the file FILE1 on logical unit 50 (protection code X) where PI is logical unit 4, the job stack can be:

```

/FORT, L, B
.
.
.
READ (4, ...
.
.
END
    
```



NOTE: THE FORTRAN LOGICAL UNIT FUN IS NOT NECESSARILY IDENTICAL WITH THE FILE LOGICAL UNIT (LUN) UNLESS SO INDICATED. V\$OPEN OVERRIDES A /PFILE ASSIGNMENT.

Figure 5-4. FORTRAN I/O Execution Sequences

LANGUAGE PROCESSORS

```

/ASSIGN,PI=50
/PFILE,4,X,FILE1
/EXECC
    
```

Case 3, normal RMD file executing in foreground or background: the CALL V\$OPEN statement associates any specified RMD file with the FORTRAN unit number. The CALL V\$OPEN statement overrides any /PFILE assignment (case 2). The format of the statement is:

```
CALL V$OPEN(fun,lun,name,mode)
```

where

- fun** is the name or number of the FORTRAN unit which may be numeric value, defined by a DATA statement, or an assignment statement
- lun** is the name or number of the logical unit which may be numeric value, defined by a DATA statement, or an assignment statement
- name** is the name of the 13-word array containing the file name and the protection code
- mode** is the mode of the I/O-control open macro (section 3.5.1)

V\$OPEN constructs an FCB in the first ten words of the specified 13-word array, performs an IOC OPEN on this FCB, and links it with the active FCB chain. The remaining three words of the array contain an FCB-chain link, the FORTRAN unit number, and the file logical unit number. Thus, to reference file FIL on logical unit 20 (protection code Q) by the number 2, rewinding upon opening, the job stack can be:

```

.
.
.
/FORT
.
.
.
DIMENSION IFCB(13)
DATA IFCB(3)/2H Q/
DATA IFCB(8),IFCB(9),IFCB(10)/2HFI,2HL ,2H /
.
.
.
CALL V$OPEN(2,20,IFCB,0)
.
.
.
    
```

File FIL can now be referenced by FORTRAN statements by using 2 as the designation of the FORTRAN logical unit. For instance,

```
READ (2,...
```

executes an IOC READ call, reading from FIL using IFCB as the FCB.

Note: V\$OPEN sets the record length to 120 words and the access method to 3, sequential access using relative VORTEX physical record number within the file. The user should not change the record length or access method parameters in the FCB because the FORTRAN Run-Time I/O package has reserved only a 120 word buffer.

Any record in a file opened by V\$OPEN can be directly accessed by operating on the FCB array. Thus, using the job stack in the previous example, record 61 in file FIL is read by inputting

```

.
.
.
IFCB(4)=61
READ(2,...
```

To dissolve an existing association between an RMD file and a FORTRAN logical unit, use the CALL V\$CLOS statement of the format.

```
CALL V$CLOS(fun,mode)
```

where

- fun** is the name or number of the FORTRAN logical unit
- mode** is the mode of the I/O-control CLOSE macro (section 3.5.2)

Thus, when the processing of file FIL in the previous example is complete, to close/update FIL and take IFCB off the active FCB chain so that FORTRAN statements with fun = 2 no longer reference FIL, the job stack can be:

```

.
.
.
CALL V$CLOS(2,1)
.
.
.
    
```

Note: the auxiliary FORTRAN I/O statements REWIND, BACKSPACE, and ENDFILE cannot be used with RMD files. Use instead (where IFCB is the ECB array):

```

IFCB(4) = 1 For rewind
IFCB(4) = IFCB(4) - 1 For backspace
CALL V$CLOS(fun, 1) For endfile
    
```

Case 4, blocked RMD file executing in foreground or background: the CALL V\$OPNB statement associates any specified RMD file with a FORTRAN unit number. This statement overrides any /PFILE statement. The format is:

CALL V\$OPNB (fun, lun, name, mode, recsz, buff, rbwfl)

where

fun	is the name or number of the FORTRAN unit which may be numeric value, defined in a DATA statement, or an assignment statement
lun	is the name or number of the file logical unit which may be numeric value, defined in a DATA statement, or an assignment statement
name	is the name of a 14-word FCB array
mode	is the mode of the I/O control OPEN macro
recsz	is the logical record size in words
buff	is the address of a blocking buffer array
rbwfl	is the read-before-write flag

The first parameters are identical in function to those of the CALL V\$OPEN statement. The other three specify blocking information.

An RMD file opened by a CALL V\$OPNB statement is processed as though it were a consecutive series of logical records, each one **recsz** words in length. These logical records continue across physical record boundaries with no space wasted (except possibly at the end of file). Input and output is buffered through the user-supplied buffer array **buff** as specified above.

Since actual physical I/O is performed on **buff**, the file must be large enough to do I/O on the end of the last logical record. It is sufficient to allocate RMD space for one more logical record than will ever be used.

It is the user's responsibility to declare the size of the buffer array **buff** sufficiently large, remembering that it is a function of the logical record size **recsz**, that it must be a multiple of the basic record size of 120, and that it must be large enough to include enough basic 120-word physical records to cover a logical record, even though the physical record may overlap the physical record boundaries. The following tables specify all conditions, where:

$Q(x/y)$ means the quotient of x/y
 $R(x/y)$ means the remainder of x/y

$recsz < 120$

$R(120/recsz)$	Size of Array Buff
= 0	120 words
≠ 0	240 words

$recsz \geq 120$

$R(recsz/120)$	Size of Array Buff
= 0	recsz
= 1	$120 * (1 + Q(recsz/120))$
> 1	$120 * (2 + Q(recsz/120))$

If **recsz** is not a multiple or factor of 120 words, the blocking buffer **buff** must allow room for an extra 120-word physical record at the start or end of a logical record.

On a WRITE operation where **recsz** is not a multiple of 120 words, data on the RMD can be overwritten unless a read-before-write is performed. In some situations, such as initial file creation in a strictly sequential fashion, this is unnecessary and slow.

The parameter **rbwfl** allows the user to select this feature. If **rbwfl** is zero, read-before-write is disabled. Any non-zero value enables read-before-write.

Example: An RMD file opened by CALL V\$OPNB can be accessed randomly, as with CALL V\$OPEN, by a replacement statement using the logical record number.

```

/FORT
  DIMENSION IFCB(14), IBUFF(120)
  DATA IFCB(3), IFCB(8), IFCB(9), IFCB(10)
        /0, 2HBL, 2HFI, 2HLE/
  CALL V$OPNB(2, 10, IFCB, 0, 10, IBUFF, 1)
  IFCB(4) = 5
  READ (2) I
  READ (2) J

```

This sequence causes the unkeyed file name BLFILE on logical unit 10 to be opened and assigned FORTRAN unit number 2. The first READ statement causes the entire first 120-word physical record (first 12 logical records) to be input into blocking buffer IBUFF, and the first word of the fifth logical record to be transferred to I. The second READ would not require another physical input for record 6 in IBUFF. This READ statement would simply transfer the first word of logical record 6 to J.

To flush the blocking buffer, close the file and disassociate the FORTRAN and logical unit numbers the CALL V\$CLSB statement is provided. Its format is:

CALL V\$CLSB (fun,mode)

LANGUAGE PROCESSORS

where

fun is the FORTRAN unit number

mode is the mode of the I/O control CLOSE macro

The end-of-file information in a FILE NAME DIRECTORY refers to a physical 120-word record number. Therefore, if logical record size is not a multiple of 120 words, the user may need to define his own end-of-file mark. Close and Update, Open and Leave, and IOCHK (section 5.3.4) EOF features all operate on this File Name Directory parameter referring strictly to 120-word physical record numbers.

5.3.3 Runtime I/O Exceptions

The FORTRAN runtime I/O program allows a program to detect I/O errors and end-of-file or end-of-device conditions. Status of a READ or WRITE operation is available immediately after the operation is complete and before another I/O operation is executed. This status can be checked by executing a subroutine or function call in the form.

CALL IOCHK(status)

where status is the name of an integer variable which is to receive the result of the status check.

If the last I/O operation had been completed normally, the value of zero will be returned. If an error had occurred, the value minus one is returned. If either an end-of-file or an end-of-device had occurred, the value positive one will be returned.

The status may be checked and the result tested in a single statement by use of the form:

IF (IOCHK(status)) label(1), label(2), label(3)

where

status is the name of an integer variable which receives the result of the status check. A value of zero indicates normal completion. A negative non-zero value indicates an error. A positive non-zero value indicates EOF or EOD.

label(1) is a statement label to which control is transferred, if an I/O error occurred.

label(2) is a statement label to which control is to be transferred if the operation was completed normally.

label(3) is a statement label to which control is transferred, if an end-of-file or end-of-device was encountered.

If the program does not check the status of a READ or WRITE operation in which an error occurs, FORTRAN will abort execution of the task upon the next entry to the runtime I/O routine. At that time the diagnostic message will be output to the System Output device. Any data which is input to a read in which an error occurred will be invalid. After a call to IOCHK is executed, any error status is reset and the program may proceed with additional input and/or output.

5.3.4 Reentrant Runtime I/O

The VORTEX runtime I/O program processes all FORTRAN READ, WRITE, auxiliary I/O, and open and close statements at execution time. It is composed of two modules, V\$FORTIO and the reentrant task V\$RERR. Both are in the OM library. V\$RERR is also in the nucleus portion of the SGL. SGEN then automatically loads V\$RERR in the VORTEX nucleus, and all FORTRAN programs automatically link to it. If V\$RERR is not desired in the VORTEX nucleus, the SGEN directive DEL, V\$RERR must be entered during system generation. Each FORTRAN program will then get its own copy of V\$RERR from the OM library. V\$RERR is approximately 3K words long.

5.4 RPG IV COMPILER

5.4.1 Introduction

The VORTEX RPG IV System is a software package for general data processing applications. It combines versatile file and record defining capabilities with powerful processing statements to solve a wide range of applications. It is particularly effective in processing data for reports. The VORTEX RPG IV system consists of the RPG IV compiler and RPG IV runtime/loader program.

The VORTEX RPG IV compiler and the runtime/loader execute as level zero background programs in unprotected memory. Both the compiler and the runtime/loader will operate in 6K of memory with limited work stack space. The stack space may be expanded and consequently larger RPG programs compiled and executed by use of the /MEM directive.

The RPG IV language and its compilation and execution under VORTEX are described in the Sperry Univac 70/620 RPG IV User's Manual (98A 9947 03x).

Error messages applicable to the RPG IV compiler are given in Appendix A.

5.4.2 RPG IV I/O Units

The RPG IV compiler reads source records from the Processor Input (PI) file, write object records on the Binary Output (BO) file, and lists the source program on the List Output (LO) file.

The RPG IV runtime/loader will normally load the RPG object program from the Binary Input (BI) file. When the program executes, the READ CARD, PUNCH and PRINT statements are performed on logical units 13, 14, and 15 respectively, statements for performing input and output to logical units 16 through 22.

5.4.3 Compiler and Runtime Execution

The RPG compiler and the runtime package should be cataloged into the background library (BL) using LMGEN.

The compiler and runtime package should be defined as background unprotected tasks with the names PRGC and RPGRT, respectively.

The compiler is scheduled from the background library by the directive

```
/LOAD, RPGC
```

The compiler terminates when the required END statement in the RPG program is encountered. The compiler exits to the executive. There is no provision for stacking multiple compilations or for operating in compile-and-go mode.

The compiler rewinds the PI, BO, and LO files at the beginning of the compilation.

The runtime/loader is scheduled from the background library by the directive

```
/LOAD, RPGRT
```

The loader expects the RPG object program is on the Binary Input (BI), and loads and executes it. If the load directive contains the name of an RPG program to be loaded in the form,

```
/LOAD, RPGRT, name
```

the runtime/loader will assume the program mentioned is in the background library and will load it from there. An RPG object program may be 'cataloged' into the background library by creating a directory entry and allocating file space with FMAIN and copying the RPG object program into the file with IOUTIL.

5.5 RPG II COMPILER

5.5.1 Introduction

The VORTEX RPG II System is an industry compatible software package for general data processing applications. It combines versatile file and record defining capabilities with powerful processing statements to solve a wide range of applications. It is particularly effective in processing data for reports. The VORTEX RPG II system consists of the RPG II compiler and RPG II runtime interpreter.

The VORTEX RPG II compiler executes as a level one background program in unprotected memory. The compiler will operate in 4K of memory with limited work space. The work space may be expanded and consequently larger RPG programs may be compiled by use of the /MEM directive.

The RPG II language, and its compilation and execution under VORTEX is described in the RPG II User's Manual.

5.5.2 RPG II I/O Units

The RPG II compiler reads source records from the Processor Input (PI) file, writes object records on the Binary Output (BO) file, and lists the source program on the List Output (LO) file. Optionally, object records may be written on the GO file.

5.5.3 Compiler and Runtime Execution

The RPG II compiler and the runtime package should be cataloged into the background library (BL) using LMGEN.

The compiler and runtime package should be defined as a background unprotected task, with the name RPG.

The compiler is scheduled from the background library by the directive:

```
/RPG
```

The compiler terminates when the required /* statement in the RPG program is encountered. The compiler exits to the executive. There is no provision for stacking multiple compilations or for operating in compile-and-go mode.

The compiler rewinds PI, BO, and LO files at the beginning of the compilation.

An RPG object program may be 'cataloged' into the background library by creating a directory entry and allocating file space with FMAIN and copying the RPG object program into the file with IOUTIL.

SECTION 6

LOAD-MODULE GENERATOR

The **load-module generator (LMGEN)** is a background task that generates background and foreground tasks from relocatable object modules. The tasks can be generated with or without overlays, and are in a form called **load modules**.

To be scheduled for execution within the VORTEX operating system, all tasks must be generated as load modules.

6.1 ORGANIZATION

LMGEN is scheduled for execution by inputting the job-control processor (JCP) directive /LMGEN (section 4.2.19).

LMGEN has a symbol-table area for 200 symbols at five words per symbol. To increase this area, input a /MEM directive (section 4.2.5), where each 512-word block will enlarge the capacity of the table by 100 symbols.

INPUTS to the LMGEN comprise:

- *Load-module generator directives* (section 6.2) input through the SI logical unit.
- *Relocatable object modules* from which the load module is generated.
- *Error-recovery inputs* entered via the SO logical unit.

Load-module generator directives define the load module to be generated. They specify the task types (unprotected background or protected foreground) and the locations of the object modules to be used for generation of the load modules. The directives supply information for the cataloging of files, i.e., for storage of the files and the generation of file-directory entries for them. LMGEN directives also provide overlay and loading information. The directives are input through the SI logical unit and listed on the LO logical unit. If the SI logical unit is a Teletype or a CRT device, the message **LM**** is output on it to indicate that the SI unit is waiting for LMGEN input.

Relocatable object modules are used by LMGEN to generate the load modules. The outputs from both the DAS MR assembler and the FORTRAN compiler are in the form of relocatable object modules. Relocatable object modules can reside on any VORTEX system logical unit and are loaded until an end-of-file mark is found. The last execution address encountered while generating a segment (root or overlay, section 6.1.1) becomes the execution address for that segment. (**Note:** If the load module being generated is

a foreground task, no object module loaded can contain instructions that use addressing modes utilizing the first 2K of memory, other than the base page (page 0). No assembler generated indirects or literals are allowed.

A VORTEX physical record on an RMD is 120 words. Object-module records are blocked two 60-word records per VORTEX physical record. However, in the case of an RMD assigned as the SI logical unit, object modules are not blocked but assumed to be one object module record per physical record.

Error-recovery inputs are entered by the operator on the SO logical unit to recover from errors in load-module generation. Error messages applicable to this component are given in Appendix A.6.

Recovery from the type of error represented by invalid directives or parameters is by either of the following:

- a. Input the character C on the SO unit, thus directing LMGEN to go to the SI unit for the next directive.
- b. Input the corrected directive on the SO unit for processing. The next LMGEN directive is then input from the SI unit.

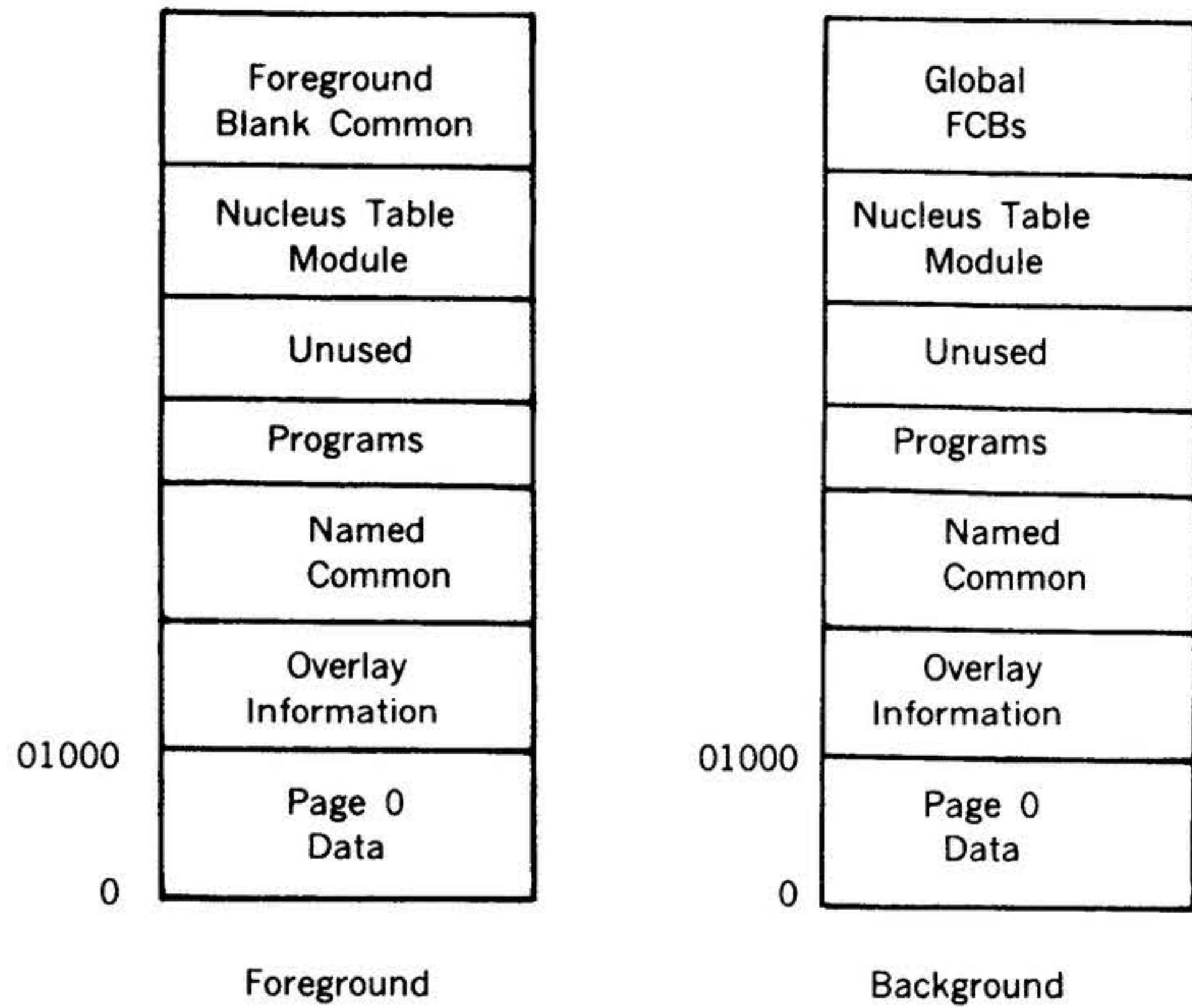
If recovery is not desired, input a JCP directive (section 4.2) on the SO unit to abort the LMGEN task and schedule the JCP for execution. (**Note:** An irrecoverable error, e.g., I/O device failure, causes LMGEN to abort. Examine the I/O error messages and directive inputs to determine the source of such an error.)

OUTPUTS from the LMGEN comprise:

- *Load modules* generated by the LMGEN
- *Error messages*
- *Load-module maps* output upon completion of a load-module generation

Load modules are LMGEN-generated absolute or relocatable tasks with or without overlays. They contain all information required for execution under the VORTEX operating system. During their generation, LMGEN uses the SW logical unit as a work unit. Upon completion of the load-module generation, the module is thus resident on the SW unit. LMGEN can then specify that the module be cataloged on another unit, if required, and output the load module to that unit. Figure 6-1 shows the structure of a load module.

LOAD-MODULE GENERATOR



All foreground tasks share the foreground blank common area but may have their own named common area.

Figure 6-1. Load-Module Overlay Structure (virtual memory)

Note: LMGGEN locks out the partition while it is modifying the directory.

Error messages applicable to the load-module generator are output on the SO and LO logical units. The individual messages, errors, and possible recovery actions are given in appendix A.6.

Error messages which apply to problems with a particular object module will include the title of the object module to which the error relates.

Load-module maps are output on the LO logical unit upon completion of the load-module generation, unless suppressed. The maps show all entry and external names and labeled data blocks. They also describe the items given as defined or undefined, and as absolute or relocatable, and indicate the relative location of the items. The load-module map lists the items in the format, four entries per line:

Print position	2 3 4 5 6 7 8	9	10	11	12 13 14 15 16
	<i>item</i>	<i>b</i>	<i>x</i>	<i>b</i>	<i>location</i>

where

- item* is a left-justified entry or external name or labeled data block
- b* is a blank
- x* is A for an absolute or R for a relocatable item
- location* is the left-justified relative location of the item

The load-module map is sorted in alphabetical order (right justified) by default. An additional numerically sorted map may be output by specifying the 'X' option (see section 4.2.19).

The following appear at the end of the LMGEN map.

[\$IAP]	Top of indirect address pool, which begins at 0500
[\$LIT]	Bottom of literal pool, which begins at 0777
[\$PED]	Last loaded location. Foreground, word size of load module. Background, last location loaded (loading begins at 01000).

LMGEN performs special handling for an external of the form '\$PED'. LMGEN satisfies this external with the last loaded location plus one of the load modules for both overlayed and non-overlayed tasks. This external can be used for specifying table areas behind tasks that link with external routines.

6.1.1 Overlays

Load modules can be generated with or without overlays. Load modules with overlays are generated when task requirements exceed core allocation. In this case, the task is divided into overlay segments that can be called as required. Load modules with overlays are generated by use of the OV directive (section 6.2.3) and comprise a root segment and two or more overlay segments (figure 6-1), but only the root segment and one overlay segment can be in memory at any given time. Overlays can contain executable codes, data, or both.

When a load module with overlays is loaded, control transfers to the root segment, which is in main memory. The root segment can then call overlay segments as required.

Called overlay segments may or may not be executed, depending on the nature of the segment. It can be an executable routine, or it can be a table called for searching or manipulation, for example. Whether or not the segment consists of executable data, it must have an entry point.

The generation of the load module begins with the root segment, but overlay segments can be generated in any order.

The root segment can reference only addresses contained within itself. An overlay segment can reference addresses contained within itself or within the root segment. Thus, all entry points referenced within the root segment or an overlay segment are defined for that segment and segments subordinate to it, if any.

For an explanation of DAS MR and FORTRAN calls to overlays see section 2.1.8.

6.1.2 Common

Common is the area of memory used by linked programs for data storage, i.e., an area common to more than one program. There are two types of common: named common and blank common. (Refer to the FORTRAN IV Reference

Manual, document number 98 A 9902 03x, or the DAS MR COMN directive description in the computer handbook, for the system being used.

Named common is contained within a task and is used for communication among the subprograms within that task.

Blank common can be used like named common or for communication among foreground tasks.

The extent of blank common for foreground tasks is determined at system generation time. The size of the foreground blank common can vary within each task without disturbing the positional relationship of entries but cannot exceed the limits set at system generation time.

The extent of blank common for background tasks is allocated within the load module. The size of the background blank common can vary within each task, but the combined area of the load module and common cannot exceed available memory.

Each blank common is accessible only by the corresponding tasks, i.e., foreground tasks use only foreground blank common, and background tasks use only background blank common.

All definitions of named and blank common areas for a given load module must be in the first object module loaded to generate that load module.

6.1.3 Shared Procedures

Load modules can be generated with or without shared procedures. Shared procedures are uniquely defined (shared procedure checksum) by name (1 to 6 ASCII characters), size, logical starting address, and external and entry point resolution. Shared procedures should be reentrant programs with no internal modification.

When a task with shared procedures is initiated, VORTEX will search for a previously loaded shared procedure with the same checksum. If one is found currently loaded, the shared procedure will not be loaded, but the remainder of the task will be loaded and the shared procedure currently in memory will be mapped into the task logical memory space. The shared procedure will be "shared" by two different tasks. Only one copy of the shared procedure will occupy physical memory.

Load module generation with shared procedures has the following restrictions:

LOAD-MODULE GENERATOR

- a. A load module may have up to ten shared procedures
- b. All shared procedures must be loaded before any non-shared object modules
- c. Each shared procedure must have no unresolved externals at the completion of shared procedure generation (externals may be satisfied by previously loaded shared procedures or the LIB directive)
- d. Generation of a shared procedure is terminated by a LIB directive

6.2 LOAD-MODULE GENERATOR DIRECTIVES

- TIDB Create task-identification block
- LD Load relocatable object modules
- OV Overlay
- LIB Library search
- CLD Load relocatable object modules without re-opening or repositioning
- MEM Default extra memory pages
- SP Create shared procedure
- END

Load-module generator directives begin in column 1 and comprise sequences of character strings having no embedded blanks. The character strings are separated by commas (,) or by equal signs (=). The directives are free-form and blanks are permitted between the individual character strings of the directives, i.e., before or after commas (or equal signs). Although not required, a period (.) is a line terminator. Comments can be inserted after the period.

The general form of a load-module generator directive is

name,*p(1)*,*p(2)*,...,*p(n)*

where

name is one of the directive names given above

each *p(n)* (if any) is a parameter required by the directive and defined below under the descriptions of the individual directives

Numerical data can be octal or decimal. Each octal number has a leading zero.

For greater clarity in the descriptions of the directives, optional periods, optional blank separators between character strings, and the optional replacement of commas (,) by equal signs (=) are omitted.

Error messages applicable to load-module generator directives are given in Appendix A.6.

6.2.1 TIDB (Task-Identification Block) Directive

This directive must be input before any other LMGEN directives can be accepted. It permits task scheduling and execution, and specifies the overlay and debugging characteristics of the task. The directive has the general form

TIDB,*name*,*type*,*segments*,*DEBUG*, *ropages*

where

name is the name (1 to 6 ASCII characters) of the task

type is 1 for an unprotected background task on BL, or 2 for a protected foreground task or 3 for a background task on an alternate library

segments is the number (2 to 9999) of overlay segments in a task with overlays, or 0 for a task without overlays (note that the number 1 is invalid)

DEBUG is present when debugging is desired

ropages is an optional read-only page specifier (1-77). It can be a single number or a range of consecutive numbers (e.g., 3,5).

The *DEBUG* parameter includes the DEBUG object module as part of the task. If the task is a load module without overlays, DEBUG is the last object module loaded. If the task is a load module with overlays, DEBUG is the last object module loaded in the root segment (section 6.1.1).

The **ropage** parameter allows specification of a range of virtual pages as read-only.

Examples: Specify an unprotected background task named DUMP as having no overlays but with debugging capability.

TIDB, *DUMP*, 1, 0, *DEBUG*

Specify a protected foreground task named PROC as having a root segment and four overlay segments.

TIDB, *PROC*, 2, 4

6.2.2 LD (Load) Directive

This directive specifies the logical unit from which relocatable object modules are to be loaded. It has the general form

LD,*lun*,*key*,*file*

for loading from RMD logical units, and

LD,*lun*

for loading from any other logical unit, where

lun is the name or number of the logical unit where the object module resides

key is the protection code required to address **lun**

file is the name of the RMD file

From the object modules, LMGGEN generates load modules (with or without overlays) on the SW logical unit. Loading of object modules from the specified logical unit continues until an end-of-file mark or an end-of-load module record (appendix G.6) is encountered.

Successive LD directives permit the loading of object modules that reside on different logical units. The execution address for the load module is the last encountered execution address.

Examples: Load the relocatable object modules from logical unit 6 (BI) until an end-of-file mark is encountered.

LD, 6

Open a file named DUMP on logical unit 9 (GO) with no protection code. (LMGGEN loads the relocatable object modules and closes the file.)

LD, 9, , DUMP

6.2.3 OV (Overlay) Directive

This directive specifies that the named segment is an overlay segment. It has the general form

OV, segname

where **segname** is the name (1 to 6 ASCII characters) of the overlay segment.

Example: Specify SINE as an overlay segment.

OV, SINE

6.2.4 LIB (Library) Directive

This directive indicates that all load (LD, section 6.2.2) directives have been input, i.e., all object modules have been loaded except those required to satisfy undefined externals. LIB also specifies the libraries to be searched (and the order in which the search is made) to satisfy all undefined externals. The directive has the general form

LIB, lun(1), key(1), lun(2), key(2), ..., lun(n), key(n)

where

each **lun(n)** is the name or number of a resident-library RMD logical unit to be searched

each **key(n)** is the protection code required to address the preceding logical unit

The search is conducted in the order in which the logical units are given in the LIB directive. When not specified by LIB, the core-resident (CL) and object-module (OM) libraries are searched after all specified libraries have been searched. However, if LIB specifies the CL and/or OM libraries, they are searched in the order given in LIB.

If the generation of the load module involves overlays, a LIB directive follows each overlay generation.

Examples: Specify to the LMGGEN a sequence of libraries to be searched to satisfy undefined externals. Use logical unit 115, a user library, having protection code M; followed by logical unit 103, the CL library, having protection code C; and the OM library, having protection code D. (Because the last two libraries are searched in any case, note that the two inputs following are equivalent.) Input

LIB, 115, M, 103, C, 104, D

or, more briefly,

LIB, 115, M

To change the order of search to logical units 104, 115, and 103, input

LIB, 104, D, 115, M, 103, C

or, more briefly,

LIB, 104, D, 115, M

To search only the CL and OM libraries to satisfy undefined externals, input

LIB

6.2.5 END Directive

This directive terminates the generation of the load module and, if specified, causes the creation of a file and a directory entry (section 9) for the load-module contents on the indicated logical unit. The indicated logical unit, if any, is an RMD, and thus may require a protection code. The directive has the general form

END, lun, key

LOAD-MODULE GENERATOR

where

lun is the name or number of the logical unit on which the file containing the load module will reside

key is the protection code, if any, required to address *lun*

If TIDB (section 6.2.1) specified an unprotected background task (TIDB directive **type** = 1), the logical unit, if any, specified by the END directive must be that of the BL unit, i.e., unit 105. If TIDB specified a protected foreground task (TIDB directive **type** = 2), the logical unit, if any, specified by the END directive must be that of the FL unit, i.e., unit 106, or that of any available assigned RMD partition. If TIDB specified an alternate library background task (TIDB directive **type** = 3), the logical unit, if any, specified by the END directive, may be that of any available assigned RMD partition.

If the END directive does not specify a logical unit, the load module resides on the SW logical unit only.

If there are still undefined externals, the load module is not cataloged even if END specifies a legal logical unit. In this case, the load module resides on the SW unit only.

Examples: Specify that the load module is complete (no more inputs to be made), create a file and a directory entry on the BL logical unit (105), and catalog the module. The protection code is E. (**Note:** The load module will also reside on the SW unit.)

```
END, 105, E
```

Specify that the load module is complete (no more inputs to be made) and is to reside on the SW unit only.

```
END
```

6.2.6 CLD Directive

This directive specifies the logical unit from which relocatable object modules are to be loaded. It has the general forms

```
CLD,lun,key,file  
  
or  
  
CLD,lun
```

Where use of the two forms and the meaning of *lun*, *key*, and *file* is as for the LD directive (section 6.2.2). This directive specifies the same action as for the LD directive except that successive CLD directives do not cause re-opening or repositioning of the specified logical unit.

6.2.7 MEM (Memory) Directive

This optional directive is used to specify the default number of extra memory blocks to be attached to a background task in a similar manner to the /MEM directive of JCP. This value is in addition to a /MEM request and is stored in word 12 of the task's pseudo TIDB. The directive has the general form

```
MEM,n
```

where

n is the number of 512 word blocks (pages)

This directive, if used, must appear after the last LIB directive and before the END directive.

6.2.8 SP (Shared Procedure) Directive

This directive specifies that a shared procedure is to be generated as part of the load module. It has the general form

```
SP,name
```

where

name is the name (1 to 6 ASCII characters) of the shared procedure

Example: Specify AIDX as the name of the shared procedure.

```
SP,AIDX
```

6.3 SAMPLE DECKS FOR LMGEN OPERATIONS

Example 1: Card and Teletype Input

Generate a background task without overlays using LMGEN with control records input from the Teletype and object module(s) on cards. Assign the BI logical unit to card reader unit CR00. Assign the task name EXC4 and catalog to the BL logical unit, and load DEBUG as part of the task from the OM library.

```
/JOB,EXAMPLE4 (Teletype input)  
/ASSIGN,BI=CR00  
/LMGEN  
TIDB,EXC4,1,0,DEBUG  
LD,BI  
LIB  
END,BL,E  
/ENDJOB
```

Note: The object module deck must be followed by an end of file (2-7-8-9 in card column 1).

Example 2: Card Input

Generate a foreground task with overlays using LMGEN with control records and object modules input from the card reader. Assign the BI and SI logical units to card reader unit CR00. Assign the task name EXC5, overlay names SGM1, SGM2, and SGM3, and catalog to the FL logical unit.

```

/JOB, EXAMPLE5
/ASSIGN, BI=CR00, SI=CR00
.
(Deck)
.
/LMGEN
TIDB, EXC5, 2, 3
LD, BI
(Object Module(s) -- root segment)
(End of File)
LIB
OV, SGM1
LD, BI
(Object Module(s))
(End of File)
LIB
OV, SGM2
LD, BI
(Object Module(s))
(End of File)
LIB
OV, SGM3
LD, BI
(Object Module(s))
(End of File)
LIB
END, FL, F
/ENDJOB

```

Example 3: Teletype and RMD Input

Generate a foreground task without overlays using LMGEN with control records input from the Teletype and object module(s) from an RMD. The object module resides on RMD 107 under the name PGEX. Assign the task name EXC6, search the OM library first to satisfy any undefined externals, and catalog on RMD 120.

```

/JOB, EXAMPLE6
/LMGEN
TIDB, EXC6, 2, 0
LD, 107, Z, PGEX
LIB, OM, D
END, 120, X
/ENDJOB

```

Example 4: Shared Procedure Generation

Generate a background task with two shared procedures. Control records are input from the Teletype, and object

modules from an RMD. The first shared procedure is name PART1 and is comprised of object module PRT10B on RMD 107. The second shared procedure is named PART2 and is comprised of object module PRT20B on RMD 107. The remainder of the object modules are in the card reader. The taskname is SHARED and cataloged to the BL logical unit.

```

/JOB, EXAMPLES
/ASSIGN, BI=CR00
/LMGEN
TIDB, SHARED, 1, 0
SP, PART1
LD, 107, Z, PRT10B
LIB
SP, PART2
LD, 107, Z, PRT20B
LIB
LD, BI
LIB
END, BL, E
/ENDJOB

```

6.4 RELINK

RELINK is a background program which is used to "relink" (reestablish VORTEX nucleus pointers) an existing load module with a modified nucleus (one recreated with a SYSGEN and for which changes have been introduced that moved certain nucleus pointers or addresses). The load module may be restored on a library partition by any copy process such as FMUTIL, IOUTIL, or equivalent program. RELINK makes the load module executable on the new system without the need to perform an entire LMGEN process. RELINK needs to be executed only once for each load module after the nucleus has been modified and the load module restored. RELINK may also be used for relinking a load module that has been transferred from another system, e.g. through CPU to CPU data links, on disk pack, or on magnetic tape. This is particularly useful in a master-slave configuration where the slave system is not capable of supporting development activities: programs are compiled and LMGENed on the host system and transferred to the slave system.

RELINK uses a Core Resident Symbol Table (CRST) located at the end of the load module file (this is not part of the user program space) which has been established by LMGEN. RELINK extracts entries from this table and searches for the nucleus pointer name in the system CL directory. When this name is found, RELINK "patches" the appropriate locations in the load module RMD file and continues this process until the CRST table is complete. The VORTEX system loader (VSSAL) does not use the CRST, since the load module has been patched. There is no limit to the number of times an existing load module may be relinked. However, relinking is possible only if all load module referenced nucleus pointers exist on the new system on which the load module is to be relinked.

LOAD-MODULE GENERATOR

Note: The RELINK program cannot be used to relink load modules cataloged by SYSGEN (e.g., OPCOM, JCP, FMAIN). It may only be used to relink load modules cataloged by LMGEN. This means that SYSGEN cataloged modules cannot be transported from one system to another. Also, programs which use Foreground Blank Common cannot be relinked if the Foreground Blank Common is not currently at the same address that it occupied when the load module was originally cataloged.

RELINK may be executed by JCP command of the form

```
/LOAD,RELINK
```

RELINK will respond with the prompt:

```
LUN,KEY,LD MODULES
```

The user then inputs the name of the load modules that are to be relinked in the format:

```
LUN,KEY,mod name, mod name,...
```

where

LUN	is the logical unit number on which the load module resides (may be lun name)
KEY	is the protection key, if any, required for that LUN
mod name	is the name of the load module on that LUN, or the keyword word "ALL" if all load modules on the specified LUN are to be relinked.

Up to ten mod names may be specified on a single input line; however, the directive may not contain any embedded blanks, since a blank character terminates the directive line processing. All the "mod names" must reside on the specified LUN.

A continuation directive may be used for "mod names" on

the previously specified LUN by starting the directive with a comma and omitting the LUN and KEY parameters. A continuation line may not be used as the first directive line. Each directive line is processed before the next directive (either new or continuation) is accepted. If a load module resides on a different LUN, it must be specified by a new directive giving the different LUN and KEY.

As each load module is relinked, the following message appears on the SO device:

```
**xxxxxx RELINKED**
```

where

xxxxxx is the name of the module which has been relinked.

The prompt LUN,KEY,LD MODULES occurs only once.

To exit from RELINK and return to JCP the user may enter the command

```
END
```

If a directive beginning with a slash (/) is encountered, an 1013 diagnostic message is output and RELINK exits.

Example: Relink the load modules PROG1, PROG2, and PROG3 from the background library, and load modules PROG4 and PROG5 from the foreground library.

```
/LOAD,RELINK
LUN,KEY,LD MODULES
BL,E,PROG1,PROG2,PROG3
**PROG1 RELINKED**
**PROG2 RELINKED**
**PROG3 RELINKED**
FL,F,PROG4,PROG5
**PROG4 RELINKED**
**PROG5 RELINKED**
END
```

SECTION 7

DEBUGGING AIDS

The VORTEX II system contains two debugging aids: the *debugging program (DEBUG)* and the *snapshot dump program (SNAP)*.

7.1 DEBUGGING PROGRAM

The 1414-word VORTEX debugging program (DEBUG) is added to a task load module whenever the DEBUG option is specified by a load-module generator TIDB directive (section 6.2.1). The DEBUG object module is the last object module loaded of the root segment if the task is an overlay load module. The load-module generator sets the load-module execution address equal to that of DEBUG.

If DEBUG has been attached to the load module, DEBUG will execute when the load module is loaded or scheduled.

During the execution of DEBUG, the A, B, and X pseudoregisters save the contents of the real A, B, and X registers, and restore the contents of these registers before terminating DEBUG. If the task uses V75 registers, the contents of R3 through R7 are also saved and restored.

When debugging is complete, the input of any job-control directive (section 4.2) returns control to the VORTEX system.

INPUTS to DEBUG comprise the directives summarized in table 7-1 input through the DI logical unit. When DEBUG is first entered, it outputs on the Teletype or CRT device the message **DG**** followed by the TIDB task name and the address of the first allocatable memory cell. This message indicates that the system is ready to accept DEBUG directives on the DI unit.

Table 7-1. DEBUG Directives

Directive	Description
A	Display and change the contents of the A pseudoregister
Ax	Change, but do not display, the contents of the A pseudoregister
B	Display and change the contents of the B pseudoregister
Bx	Change, but do not display, the contents of the B pseudoregister
*Rn	Display and change the contents of the V75 register n (n = 0-7).
*Rnx	Change, but do not display, the contents of the V75 register n.
Cx	Display and change the contents of memory address x
Fx	Set base address to value of x
Gx	Load the contents of the pseudoregisters into the respective A, B, and X registers, and transfer to memory address x
Ix,y,z	Initialize memory addresses x through y with the value of z
O	Display and change the overflow indicator
P	Read DEBUG directives from BI unit until EOF
QC	Clear all traps and restore original contents to trap cells
QL	Display current trap locations
QSV,w,x,y,z	Place traps at memory addresses v,w,x,y,z
Sx,y,z,m	Search memory addresses x through y for the z value, using mask m
Ty,x	Place a trap at memory address y, starting execution at address x

Table 7-1. DEBUG Directives (continued)

Directive	Description
Ty	Place a trap at memory address y, starting execution at the last trap location
X	Display and change the contents of the X pseudoregister
Xy	Change, but do not display, the contents of the X pseudoregister
xxxxxx	Display the contents of memory address xxxxxx
xxxxxx,yyyyyy	Display the contents of memory addresses xxxxxx through yyyyyy

* = V75 systems only

Each DEBUG directive has from 0 to 72 characters and is terminated by a carriage return. Directive parameters are separated by commas, but DEBUG treats commas, periods, and equal signs as delimiters.

Numerical data are always interpreted as octal by DEBUG. Negative numbers are accepted, but they are converted to their two's complements by DEBUG.

The user may specify a "base address" using the F command. The purpose of this command is to allow the use of relocatable addresses in DEBUG. If the base address is set to the load address of the program being debugged, all subsequent DEBUG directive address parameters will have this base address added before accessing memory. Thus, an absolute location may be accessed by specifying its relocatable address. Also, all addresses output on the DO logical unit will be relocatable addresses, not absolute addresses.

The default value for the base address is zero, allowing the use of absolute addressing. Although the base address can be changed at any time, to avoid confusion, the user should set the base address as desired at the start of the debugging operation and not modify it thereafter.

The base address can be used in a change command which sets (but does not display) memory locations. The parameters to which the base address is to be added should specify a "*" before the numerical data. These parameters (normally representing relocatable addresses) will have the base address added before storing in memory.

↓ Traps can be placed at as many as five memory addresses, but no two of these can be consecutive addresses. If the user attempts to place a trap (with the QS or T command) where one already exists, the trap is left in place and processing continues. An attempt to set more than five different traps will be unsuccessful and will result in the issuance of an error message.

↑

An error message, EX20-EX25, is output and the task is aborted, if a memory-map protection violation occurs.

OUTPUTS from DEBUG consist of corrections to registers and memory, displays, listings on the DO logical unit, and error messages. Numerical data are always to be interpreted as octal.

Error messages applicable to the debugging program are given in Appendix A.7.

Examples of DEBUG directive usage: Note that, in the following examples, operator inputs are in bold type. Entries in *italics*, are program responses to the directives.

Display the contents of a pseudoregister A:

A
(001200)

Display and change the contents of a pseudoregister B:

B
(001200) 010406

Change, but do not display, the contents of a pseudoregister X:

X02050

Display, but do not change, the status of the overflow indicator:

O
(000001)

Load the contents of the pseudoregisters into the respective A, B, and X registers, and start execution at memory address 001001:

G001001

Initialize memory addresses 000200 through 000210 to the value 077777:

I000200,000210,077777

Search memory addresses 000200 through 000240 for the value 000110 using the mask 000770, and display addresses that compare:

S000200,000240,000110,000770
 000220 (017110)
 000234 (000110)
 000237 (001110)

Load the contents of the pseudoregisters and the overflow indicator status into the respective registers, and start execution at memory address 001234, specifying a trap address of 001236. Display the contents of the A, B, and X registers and the setting of the overflow indicator when the trap address is encountered:

T001236,001234
 001236 (142340) 002000 010405 012345 000001

Execute the same trap if the task uses V75 instructions (assuming Rn = n):

T001236,001234
 001236 (142340) 002000 010405 012345 000001
 000003 000004 000005 000006 000007

Display the contents of memory address 001234:

001234
 (001200)

Display the contents of memory addresses 001234 through 001237:

001234,001237
 001230 005000 005000
 └──────────────────────────┘
 Total of 8 values

7.2 SNAPSHOT DUMP PROGRAM

The 294-word **snapshot dump program (SNAP)** provides on the DO logical unit both register displays and the contents of specified areas of memory. It is added to a task load

module if the task contains a SNAP request and calls the SNAP external routine. SNAP is entered directly upon execution of the SNAP display request CALL SNAP. The SNAP display request is an integral part of the task and is assembled with the task directives. Thus, no external intervention is required to output a SNAP display.

SNAP outputs the message **SN**** followed by the task TIDB name before listing the requested items. The calling sequence for a SNAP display is

```

EXT          SNAP
CALL         SNAP
DATA        start
DATA        end
DATA        tidb
    
```

where

- start** is the first address whose contents are to be displayed
- end** is the last address whose contents are to be displayed
- tidb** is less than zero if dump of task TIDB is desired, is positive if TIDB dump is to be suppressed

If **start** is a negative number, there is no memory dump. If more than one location is specified to be displayed, the output dump will be in complete lines of eight addresses, e.g., if **start** is 01231 and **end** is 01236, the dump will display the contents of addresses 01230 through 01237, inclusive. SNAP displays octal data.

If there is an error in the SNAP display request, only the contents of the A, B, and X (and V75 if present) registers and the setting of the overflow indicator are displayed.

Output examples: with the snap request at 01234, display the contents of the A (017770), B (001244), and X (037576) registers, and the overflow indicator (on).

```

SN** TASK01
001234 017770 001244 037576 000001
*000003 000004 000005 000006 000007
    
```

Using the same data, display, in addition, the contents of memory addresses 001002 through 001025, inclusive and request a dump of the active TIDB.

DEBUGGING AIDS

```
SN** SW      000500
001023 000000 000000 001023 000000
*000003 000004 000005 000006 000007
```

TIDB LOC 055013 =CONTENTS=

```
055010 000000 000000 000000 000000 000001 000000 000000 001527
055020 001527 067001 001326 141146 001000 065604 000007 001302
055030 000001 001541 000002 000000 002000 151727 120240 120240
055040 000500 000000 074627 064604 055075 000003 000004 000005
*055050 000006 000007 000000 000000 000000 000000 000000 000000
```

SNAP DUMP

```
001000 006505 070275 001402 001031 000050 006505 066270 100000
001010 010002 075334 000000 000000 006505 070137 001005 001101
001020 001101 001101 001014 002000 001107 001000 001027 001000
```

* These lines appear only if the task uses V75 register

7.3 SYSTEM MEMORY DUMP

The system memory dump facility consists of the following three major components:

- DSYSTEM and VZRMm (nucleus resident)
- DSPMEM (background task)
- FILINT (foreground task)

These components combine to provide the user with a system memory dump under three conditions:

- a. pre-determined occurrence of an error
- b. at any time by use of the CPU console interrupt
- c. upon a system halt

The general run of events leading up to a system memory dump are:

- a. The foreground task FILINT is activated upon system boot, opening the appropriate RMD dump file and transferring control to a portion of the nucleus task DMEMORY, which saves the RMD set-up information for later use. Control then returns to FILINT, which exits.
- b. Upon a selected condition (as mentioned above), the VORTEX system passes control to DSYSTEM, which shuts down VORTEX and dumps all assigned maps to the previously opened RMD file via a standalone RMD driver VZRMm. Upon completion of the dump, DSYSTEM updates the file number, activates FILINT, and reactivates VORTEX.

c. FILINT then outputs the dump file used to the OC devices and opens the next sequential file. If all files have been used, FILINT returns to the first file.

d. The user may, at some later point in time, examine the selected dump file via the background program DSPMEM.

7.3.1 DSPMEM Program

DSPMEM is a background utility program used to display the contents of the dump image file created by the VORTEX II nucleus program DSYSTEM. The contents of the file are formatted and output to the system LO device. DSPMEM is controlled by directives input from the system SI device.

7.3.2 DSPMENT Directives

The following are DSPMENT directives:

```
.FIL
.TID
.TSK
.END
```

DSPMEM directives begin in column 1 and comprise sequences of character strings having no embedded blanks. The character strings are separated by commas. Embedded blanks are not allowed. The occurrence of the first blank terminates the directive. The general form of the directive is

name,p(1),p(2),p(3)

where

name is one of the directive names given above

each **p(n)** is a parameter required by the individual directive (if any exists) described below.

the TID and TSK directives apply, and has the general form

FIL,name

where

name is the image file name to be examined.

Error messages applicable to DSPMEM are given in appendix A.7.

7.3.2.1 FIL (Image File) Directive

This directive must be input before using either the TID or TSK directives. It is used to specify the image file to which

No further FIL directives are needed until it is desired to examine a different image file. The status at dump time is output to LO in the format

```
EX22 ERROR IN TASK  DEBUG
ADDR= 002476, INST= 056000
A=000003  B=000200  X=002323
3=000000  4=000000  5=000000  6=000000  7=000000
```

Example: Specify image file IMAGE0 to be examined.

FIL, IMAGE0

TID,name

where

name is the TIDB name or address (in octal); or the characters 'ALL', signifying that all TIDBs are to be displayed

7.3.2.2 TID (TIDB) Directive

This directive is used to request a list of one or all TIDBs on the TIDB chain at DMEMORY dump time. The directive has the general form

Format of the TID directive output is

```
*****DEBUG *****
011000--TBTHRD= 071640  TBST = 001421  TBPL = 040402  TBEVNT= 000000
011004--TBRSA = 002476  TBRSE = 000000  TBRSE = 011150  TBRSP = 002613
011010--TBRSTS= 143152  TBENTY= 001000  TBTMS = 071527  TBTMIN= 000000
011014--TBISA = 001306  TBISB = 000001  TBISX = 001251  TBISP = 001237
011020--TBISRS= 000000  TBIO = 000000  TBKN1 = 142305  TBKN2 = 141325
011024--TBKN3 = 143640  TBTLC = 001000  TBCPTH= 000000  TBATSK= 071767
011030--TBRSE = 010122  TBSIZ = 006000  TBNUCL= 000001  TBMIMG= 011050
011034--TBIST = 100041  TBRSR3= 000000  TBRSR4= 000000  TBRSR5= 000000
011040--TBRSR6= 000000  TBRSR7= 000000  TBISR3= 000000  TBISR4= 000000
011044--TBISR5= 000000  TBISR6= 000000  TBISR7= 000000
```

If the subject TIDB is a driver TIDB, then the first fifteen words of the controller table (if attached) is output in the format

```
CONTROLLER TABLE, FIRST 12 WORDS
071527--CTTIDB= 173001  CTADNC= 000020  CTOPM = 000047  CTDST = 063460
071533--CTRQBK= 011000  CTRTY= 000772  CTDVAT= 000001  CTIOA = 000000
071537--CTSTAT= 000000  CTBICB= 000000  CTFCB = 004364  CTWDS = 000000
```

DEBUGGING AIDS

If request blocks are queued to the controller, then they are output in the format

```
REQUEST BLOCK QUEUED TO CONTROLLER
011000--RSTPR = 000000  ROPWD = 010002  RFCB = 004364  RTIDB = 011100
011004--RADNR = 000000
-----
```

Multiple TID directives can be issued to display more than one TIDB. The TIDB of the task specified by the TSK directive is output together with the TSK directive output. The TID directive is thus not needed if a single TIDB is to be displayed and the task is to be displayed using the TSK directive.

Examples: Specify the displaying of all TIDBs at dump time.

TID, ALL

Specify the displaying of the TIDB for task IOUTIL at dump time.

TID, IOUTIL

7.3.2.3 TSK (Task) Directive

This directive is used to request a display of a specified task's memory space at dump time. The directive has the general form

TSK, name, p(1), ..., p(n)

where

name is the task TIDB name or address (in octal)

each p(i) is a dump limit parameter of the following:

P is dump through program region only

T to dump through table region only

F to dump through foreground common only

M to not dump mapped-in page (MAPIN RTE call)

Sxxx where xxx is the starting page number (octal or decimal) to dump

Exxx where xxx is the ending page number (octal or decimal) to dump

V to exclude the dumping of VNO tasks. Note: this parameter is required to suppress VNO tasks even if P, T, F, or E are specified

blank to dump entire task space

Parameters P, T, and F are mutually exclusive. The last encountered parameter takes precedence.

The output of the TSK parameter has the format

```
*****DEBUG *****
011000--TBTHRD= 071640  TBST = 001421  TBPL = 040402  TBEVNT= 000000
011004--TBRSA = 002476  TBRSE = 000000  TBRSE = 011150  TBRSP = 002613
011010--TBRSTS= 143152  TBENTY= 001000  TBTMS = 071527  TBTMIN= 000000
011014--TBISA = 001306  TBISB = 000001  TBISX = 001251  TBISP = 001237
011020--TBISRS= 000000  TEIO = 000000  TBKN1 = 142305  TBKN2 = 141325
011024--TBKN3 = 143640  TBTLC = 001000  TBCPTH= 000000  TBATSK= 071767
011030--TBRSE = 010122  TBSIZ = 006000  TBNUCL= 000001  TBMIMG= 011050
011034--TBIST = 100041  TBRSR3= 000000  TBRSR4= 000000  TBRSR5= 000000
011040--TBRSR6= 000000  TBRSR7= 000000  TBISR3= 000000  TBISR4= 000000
011044--TBISR5= 000000  TBISR6= 000000  TBISR7= 000000

MAP 01 TIDB MAP IMAGE
011050--002000  001002  001003  001004  000000  000000  000000  000000
**
011140--000000  000000  000000  000000  000000  000000  000000  000000
```



```
MAP 01 ACTUAL IMAGE AS READ
000000--002000 001002 001003 001004 000000 000000 000000 000000
**
000070--000000 000000 000000 000000 000000 000000 000000 000000
```

```
PAGE 00, RP ACCESS CONTROL, PHYSICAL PAGE 0000
000000--002000 021556 000000 000000 000000 000000 000000 000000
000010--000000 000000 000000 000000 000000 000000 005000 072153
000020--002000 043344 002000 043133 002000 043113 002000 042773
000030--002000 042763 002000 043103 002000 042757 002000 042757
000040--002000 01151 001000 043762 002000 042724 001403 000000
000050--002000 120240 120240 120240 000000 000000 000000 000000
```

where (going from top to bottom) the first section is the tasks TIDB contents at dump time, the second section is the TIDB map image as kept in memory, the third section is the map RAM image as read in, the fourth section is the task's memory space at dump time displayed on a page basis, and the fifth section (if a map 0 task and VNO (Virtual Nucleus Overlay) tasks are present) is a dump of all VNO task on a page basis.

Multiple TSK directives can be issued to display more than one task. A task's TIDB address ought to be used, where the specified task has the same TIDB name as another.

Examples: Specify the displaying of the nucleus memory at dump time. Use V\$EROR as the task name.

TSK, V\$EROR

Specify the displaying of the program USEROO memory at dump time. Display only the program region and do not display mapped-in pages.

TSK, USEROO, M, P

7.3.2.4 END Directive

This directive is employed to exit DSPMEM and return to JCP. The directive has the form

END

There are no parameters to this directive.

7.3.2.5 Usage Considerations

An image file is not protected from being used again before it is examined by DSPMEM. To save a file for later examination, use FMAIN to rename the file to a name other than IMAGEx and then create a new image file to take its place. Note that procedure should not be used if the system only has one dump image file (IMAGE0).

FILINT uses logical unit 190 to open the next dump image file. Attempting to reassign logical unit 190 to another disc partition or another disc unit or type may result in dump errors or possible damage to disc files. Hence do not reassign 190 after VORTEX is active.

7.3.3 System Generation Requirements

The following SYSGEN directives must be used to support the System Memory Dump:

- a. Assign logical unit 190 to the RMD partition which is to contain the dump image files. This file must have a protection key of 'Z' or be unprotected. The partition should contain at least 1-1/2 * 5 * n sectors, where n is the number of 512-word pages in the system.
- b. EQP, RM0x, 0, 1, 0, 0 where x is the RMD model code for the RMD which is to contain the dump files.

7.3.4 Post SYSGEN Requirements

The following post system generation requirements must be met to support the System Memory Dump:

- a. The foreground program FILINT and the background program DSPMEM must be cataloged. This is done by standard VORTEX II job streams.
- b. The dump image files must be created by using the following FMAIN directive

CREATE, 190, Z, IMAGEx, 512, n

where

- x is the file number (0-9), which must be contiguous (i.e., 0,1,2,3) and begin with zero
- n is the number of 512-word records required, at least 1-1/2 times the number of pages in the system.

Example: Direct DSYSTM to dump to the DOON partition of a 70-7500 disc, using three dump files on a 96K system.

```
SYSGEN
EQP, RM0C, 0, 1, 0, 0
PRT, DOON, x, z
ASN, 190=DOON
```

```
FMAIN
CREATE, 190, z, IMAGE0, 512, 288
CREATE, 190, z, IMAGE1, 512, 288
CREATE, 190, z, IMAGE2, 512, 288
```

7.3.5 Invoking a Dump

A system memory dump may be invoked by using any one of the following three methods.

- a. A patch may be made to a nucleus program, causing a call to DSYSTM when a predetermined condition occurs. This is done by inserting a call to DSYSTM. Thus the following patch ought to be employed to cause a dump on any E20-E24 memory map violation:

```
V$PSTR, CALL, DSYSTM, JMP, V$DISP
V$FUNC+2065, V$PSTR
```

- b. The console interrupt may be used to cause a dump at any time when VORTEX II is running. By this method, VORTEX II will halt with I = 0444 and

A = dump file number (the 'x' of 'IMAGEx')

B = error code (the 'xx' of 'DMxx', equal to zero if there are no errors; see appendix A.7)

Engaging console run continues VORTEX execution from the point where it was shut down by the pressing of console interrupt.

- c. When the system has halted or has "locked up", a DSYSTM dump can be invoked by commencing execution at location zero. The results are the same as for the employment of console interrupt save that the system must be rebooted to bring up VORTEX instead of merely pressing console run.

After each dump (except from a halt or "locked up" system) the dump file name used is output to the OC device and the next sequential file opened. If the dump file just used is the last sequential file, an IO10 diagnostic will occur for FILINT and the first sequential file will be opened. After any dump file is used, it may be examined using DSPMEM at any time until it is used again.

Rebooting the system always re-selects dump image file IMAGE0. To protect dump image file IMAGE0, use the method described in section 7.3.1.6 and then immediately reboot the system. In general, whenever dump image file IMAGE0 is recreated, the system should be rebooted immediately after its recreation.

```

PMAIN
CREATE,190,Z,IMAGE0,512,288
CREATE,190,Z,IMAGE1,512,288
CREATE,190,Z,IMAGE2,512,288
    
```

7.3.5 Invoking a Dump

A system memory dump may be invoked by using any one of the following three methods.

- a. A patch may be made to a nucleus program, causing a call to DSYSTM when a predetermined condition occurs. This is done by inserting a call to DSYSTM. Thus the following patch should be employed to cause a dump on any EX20-EX24 memory map violation:

```

V$PSTR, CALL, DSYSTM, JMP, V$DISP
V$DSYS, JMP, V$PSTR
    
```

To use DMEMORY/DCORE instead of DSYSTM use

```

V$PSTR,100747,100444,JMP,DMEMRY
    
```

- b. The console interrupt may be used to cause a dump at any time when VORTEX II is running. By this method, VORTEX II will halt with I = 0444 and

A = dump file number (the 'x' of 'IMAGEx')

B = error code (the 'xx' of 'DMxx', equal to zero if there are no errors; see appendix A.7)

Engaging console run continues VORTEX execution from the point where it was shut down by the pressing of console interrupt.

- c. When the system has halted or has "locked up", a DSYSTM dump can be invoked by commencing execution at location zero. The results are the same as for the employment of console interrupt save that the system must be rebooted to bring up VORTEX instead of merely pressing console run.

After each dump (except from a halt or "locked up" system) the dump file name used is output to the OC device and the next sequential file opened. If the dump file just used is the last sequential file, an IO10 diagnostic will occur for FILINT and the first sequential file will be opened. After any dump file is used, it may be examined using DSPMEM at any time until it is used again.

Rebooting the system always re-selects dump image file IMAGE0. To protect dump image file IMAGE0, use the method described in section 7.3.1.6 and then imme-

diately reboot the system. In general, whenever dump image file IMAGE0 is recreated, the system should be rebooted immediately after its recreation.

7.4 V77-800 TRACE FACILITY

The trace facility is only implemented on the V77-800 processor. TRACE is a foreground task which allows the user to specify conditions under which TRACE will print current register contents after execution of each instruction of the traced task. The TRACE output is directed to the LO device. TRACE commands are input from the DI device; prompts and error messages are output on the DO device.

The trace facility can be used for any foreground or background task, but only one task can be traced at a time. The TRACE request for a background task is made using the JCP /TRACE directive. This directive specifies that the next task scheduled by JCP will execute in the trace mode. A foreground task is traced if the parameter T is included in the OPCOM ;SCHED command.

7.4.1 TRACE COMMANDS

When the traced task is ready for execution, TRACE will request input from the user. If the DI device is a user terminal, the prompt 'TR**' is issued to the DO device. The commands accepted by the trace facility are

Command type	Format	Name
Execution specifier	EX	Exit
Execution specifier	RE	Restart
Test specifier	WD,r,l,aaaaaa	Wild Card
Test specifier	RG,r,l,aaaaaa,bbbbbb	Range

where:

r is the register selected (P,I,A,B,X,3,4,5,6 or 7).

l is the logical test required (A or O).

aaaaaa, bbbbbb are command parameters discussed below.

7.4.1.1 Wild Card Command

The Wild Card command allows a user to specify particular octal values for testing each digit position in the selected register. Multiple values can be specified for any position by enclosing the set of permissible values in parentheses. An asterisk (*) in place of a particular value indicates that any octal digit is valid in this position.

DEBUGGING AIDS



Each of the six octal digit values must be specified in the wild card command. The digit positions, symbolized by 'aaaaaa', are numbered 5 through 0 as follows:

```

a a a a a a    16-bit register value
5 4 3 2 1 0    digit position numbers
    
```

Example: Specify a wild card which defines a set of octal numbers containing a 1, 2, or 7 in digit position 1 for Register A.

```
WD,A,O,*****127*
```

This test specifies that only digit position 1 is to be tested. All other digits can be any value, but digit position 1 must contain a 1, 2, or 7 for the test to pass. The third parameter of this command, O, is a logical test specifier. Its function is described in section 7.4.2.

7.4.1.2 Range Command

The Range command specifies the range of octal values for which the specified register is to be tested. The parameter 'aaaaaa' of the Range command is the lowest value in the range; the parameter 'bbbbbb' is the highest value in the range. The default low value is 0 and the default high value is 077777. These values are used by TRACE whenever the corresponding range parameter is not specified in the Range command. Negative numbers can be specified as parameters by using the two's complement format.

Example: Specify a Range command which defines a range of octal numbers from 777 to 3456, inclusive, for the P register.

```
RG,P,A,777,3456
```

This test will pass only if the P register contains an octal number between 777 and 3456. The third parameter of the command, A, is a logical test specifier. Its function is described in section 7.4.2.

7.4.1.3 Restart Command

The Restart command is used to restart the TRACE input command sequence and destroy the previously input command specifications.

7.4.1.4 Exit Command

The Exit command is used to terminate TRACE command input and begin tracing the designated traced task.



7.4.2 LOGICAL SPECIFICATIONS

The logical specification parameter of the WD or RG command allows the user to specify the operation performed by TRACE when the test passes, fails, or the end of tests is encountered. The operations performed by each specification choice for each condition are listed below. All TRACE commands are evaluated in the order they were input.

Condition encountered	Operation Performed	
	Choice A	Choice O
Test passes	Continue testing	Print registers
Test fails	Do not print	Continue testing
End of tests	Print registers	Do not print

7.4.3 TRACE EXAMPLES

Example 1: Trace the foreground task TASKA, located on the Foreground Library. Require the trace facility to print the contents of all registers whenever the A register is negative.

```

;SCHED,TASKA,2,FL,F,T
TR**                (trace prompt)
RG,A,O,100000,177777 (range command)
TR**                (trace prompt)
EX                  (exit and begin trace)
    
```

If the A register contains a negative value, the test passes. Since an O logical specifier was used, register contents are printed when the test passes. The end of test condition is then encountered, and this does not cause the register contents to be printed.

Example 2: Trace TASKB and print registers when any JAZ or JAZM instruction occurs. TASKB is a background task.

```

/JOB,TRACJOB
/TRACE
/LOAD,TASKB
TR**                (trace prompt)
WD,I,O,00(12)010   (wild card)
TR**                (trace prompt)
EX                  (exit and begin trace)
    
```



Update B



The octal representatives of the JAZ and JAZM instructions are 001010 and 002010, respectively. This test will pass if either of these is contained in the I (instruction) register. The O logical specification causes the register contents to be printed when the test passes (i.e., when one of these instructions is found).

Example 3: Trace TASKC, a foreground task, and print registers when register 3 contains 0, or if register B is positive, or if an LDA instruction occurs between addresses 0123 and 03456.

```

;SCHED,TASKC,2,FL,F,T
  TR**          (trace prompt)
RG,2,0,0,0     (range test for R3=0)
  TR**          (trace prompt)
RG,B,0,0,77777 (range test for RB 0)
  TR**          (trace prompt)
RG,P,A,123,3456 (range test for address
                bounds)
  TR**          (trace prompt)
WD,I,O,01**** (wild card for LDA
                instruction)
  TR**          (trace prompt)
EX             (exit and begin trace)
    
```



If register 3 contains 0, the register values will be printed. This is indicated by the O logical specification. If this test fails, the succeeding tests will be evaluated. If register B contains a positive number, the registers will also be dumped. Because the LDA test is required only for addresses in the range 0123 through 03456, the logical specification 'A' is used. This eliminates addresses outside these bounds. If this test passes, the next test is evaluated; otherwise, register contents are not printed. The final test specifies a wild card test of the instruction register which will pass only if positions 5 and 4 contain the digits 0 and 1, respectively. This condition is true only for the LDA instruction.

7.4.4 TRACE ERROR CONDITIONS

TRACE error messages are sent to the DO device. They are of the form

TRxx

where:

xx is one of the following error codes.

Code	Error Condition
01	Invalid character encountered in TRACE directive
02	Unbalanced parenthesis or too many commas
03	Invalid command, register, or logical specification
04	Too many tests specified



SOURCE EDITOR

Error messages applicable to SEDIT are output on the SO and LO logical units. The individual messages and errors are given in Appendix A.8.

The listing of the SEDIT directives is made as the directives are read. Source records, when listed, are listed as they are input or output. The VORTEX standard heading appears at the top of each page of the listing.

LOGICAL UNITS referenced by SEDIT are either fixed or reassignable units. The three fixed logical units are:

- a. The SI logical unit, which is the normal input unit for SEDIT directives.
- b. The SO logical unit, which is used for error-processing.
- c. The LO logical unit, which is the output unit for SEDIT listings.

The three reassignable logical units are:

- a. The SEDIT input (IN) logical unit, which is the normal input unit for source records. This is assigned to the PI logical unit when SEDIT is loaded, but the assignment can be changed by an AS directive with an IN parameter (section 8.2.1).
- b. The SEDIT output (OUT) logical unit, which is the normal output unit for source records. This is assigned to the PO logical unit when SEDIT is loaded, but the assignment can be changed by an AS directive with an OU parameter.
- c. The SEDIT alternate input (ALT) logical unit, which is the alternate input unit used for new or replacement source records. This is assigned to the BI logical unit when SEDIT is loaded, but the assignment can be changed by an AS directive with an AL parameter.

Note: If PI, or PO, or BI are assigned to RMD units when SEDIT is loaded, SEDIT will attempt to open files named "PI", or "PO" or "BI" on the respective units. If the file does not exist an "SE02" error message is presented; this message is also generated if PI, PO, or SI is assigned to DUM.

8.2 SOURCE-EDITOR DIRECTIVES

This section describes the SEDIT directives:

- a. Copying group:
 - AS Assign logical units
 - AD Add record(s)
 - SA Add string
 - REPL Replace record(s)
 - SR Replace string
 - DE Delete record(s)
 - SD Delete string
 - MO Move record(s)
- b. Auxiliary group:
 - FC Copy file
 - SE Sequence records
 - LI List records
 - GA Gang-load all records
 - WE Write end-of-file
 - REWI Rewind
 - CO Compare records

SEdit directives begin in column 1 and comprise sequences of character strings having no embedded blanks. The character strings are separated by commas (,) or by equal signs (=). The directives are free-form and blanks are permitted between individual character strings of the directive, i.e., before or after commas (or equal signs). Although not required, a period (.) is a line terminator. Comments can be inserted after the period.

The general form of an SEDIT directive is

name,*p*(1),*p*(2),...,*p*(*n*)

where

name is one of the directive names given above or a longer string beginning with one of the directives names (e.g., AS or ASSIGN)

each *p*(*n*) is a parameter defined below under the descriptions of the individual directives

Where applicable in the following descriptions, a field specification of the format (first,last) or (n1,n2,n3) is still separated from other parameters by parentheses even though it is enclosed in commas. Note also that the character string *string* is coded within single quotation marks, which are, of course, neither a part of the string itself nor of the character count for the string.

8.2.1 AS (Assign Logical Units) Directive

This directive specifies a unit assignment for an SEDIT reassignable logical unit (section 8.1). It has the general form

AS,*nn* = *lun*,*key*,*file*

where

nn is IN if the directive is making an assignment of the IN logical unit, OU if the OUT logical unit, or AL if the ALT logical unit

lun is the name or number of the logical unit being assigned as the IN, OUT, or ALT unit

key is the protection code, if any, required to address *lun*

file is the name of an RMD file, if required

If the SEDIT reassignable units are to retain the assignments made when SEDIT was loaded (default assignments: IN = PI, OUT = PO, ALT = BI), no AS direc-

tive is required. Each AS directive can make only one reassignment (e.g., if both IN and OUT are to be reassigned, two AS directives are required).

Any RMD affected by an AS directive is automatically repositioned to beginning of file.

The AS directive merely fixes parameters in I/O control calls within SEDIT. It does not alter I/O control assignments in the logical-unit table (table 3-1).

Note: AS resets the corresponding record counter; however, no physical rewinding of devices occurs.

Examples: Assign the PI logical unit as the SEDIT reassignable IN unit.

AS, IN=PI

or, the unabbreviated form

ASSIGN, INPUT=PI

Assign logical unit 8 as the SEDIT reassignable OUT unit.

AS, OU=8

Assign as the SEDIT reassignable IN unit the file FILEX on logical unit 111, an RMD partition without a protection key.

AS, IN=111, FILEX

8.2.2 AD (Add Records) Directive

This directive adds source records. It has the general form

AD,recno

where **recno** is the number of the record last copied from the IN logical unit before switching to the ALT unit for further copying.

The AD directive copies source records from the IN logical unit onto the OUT logical unit beginning with the current position of the IN unit and continuing up to and including the record specified by **recno**. Then, source records are copied from ALT onto OUT from the current position of the unit up to but *not* including the next end-of-file mark.

Example: Copy records from IN onto OUT from the current position of IN up to and including IN record 7. Then, switch to ALT and copy records from the current position of that unit up to but *not* including the next end-of-file mark.

AD, 7

8.2.3 SA (Add String) Directive

This directive inserts a character string into a source-record field. It has the general form

SA,recno,(first,last),'string'

where

recno is the number of the source record in which the character string is to be inserted

first is the number of the first character position to be affected

last is the number of the last character position to be affected

string is the string of characters to be inserted in the field delimited by character positions **first** and **last** in record number **recno**

The SA directive copies source records from the IN logical unit onto the OUT logical unit beginning with the current position of the IN unit and continuing up to but *not* including the record specified by **recno**. The record **recno** is read into the memory buffer. The character string **string** shifts into the left end of the specified field **first,last**, with characters shifted out of the right end of the field being lost. There is no check on the length of **string** and shifting continues until it is left-justified in the field with excess characters, if any, being truncated on the right.

The record remains in the memory buffer, thus permitting multiple string operations on the same record. (If IN is already positioned at **recno** because of a previous string operation, there is, of course, no change in position.)

The record **recno** is read out of the memory buffer and onto the OUT unit when an SEDIT directive affecting another record is input.

The field specification **first,last** is lost after one manipulation. Subsequent string operations must specify the character positions based on the *new* configuration. For example, for the character string ACDEGbb in positions 1 through 7, addition of the character B in position 2 requires the field specification (2,7). Then, to add the character F between E and G, one must specify the field (6,7) rather than (5,7) because of the shift previously caused by insertion of the character B.

Example: Change the erroneous DAS MR source-statement operand in character positions 16-21 of the 32nd record from LOCXbb to LOC,Xb.

SA, 32, (19,20), ' , '

8.2.4 REPL (Replace Records) Directive

This directive replaces one sequence of source records with another sequence of records. It has the general form

REPL,recno1,recno2

where

recno1 is the number of the first record to be replaced

recno2 is the number of the last record to be replaced

If **recno2** is omitted, it is assumed equal to **recno1**, i.e., one record will be replaced.

The REPL directive copies source records from the IN logical unit onto the OUT logical unit beginning with the current position of the IN unit and continuing up to but not including the record specified by **recno1**. Then, records are read from IN, but not copied onto OUT, up to and including the record specified by **recno2**. Thus, the records **recno1** through **recno2**, inclusive, are deleted. Then, source records are copied from the ALT logical unit from the current position of the unit up to but not including the next end-of-file mark.

Example: Copy records from IN onto OUT from the current position of IN up to and including record 9. Replace IN records 10 through 20, inclusive, with records on ALT, copying those between the current position of ALT and the next end-of-file mark onto OUT. Do not copy the end-of-file mark.

REPL, 10, 20

8.2.5 SR (Replace String) Directive

This directive replaces one character string within a source record with another character string. It has the general form

SR,recno,(n1,n2,n3),'string'

where

recno is the number of the source record in which the character string is to be replaced

n1 is the number of the first character position of the string to be replaced

n2 is the number of the last character position of the string to be replaced

n3 is the number of the last character position of the field in which the string to be replaced occurs

string is the string of characters to be inserted in the field delimited by character positions **n1** and **n3** in record number **recno** after shifting out the characters in positions **n1** through **n2**, inclusive

The SR directive copies source records from the IN logical unit onto the OUT logical unit beginning with the current position of the IN unit and continuing up to but not including the record specified by **recno**. The record **recno** is read into the memory buffer. Field **n1,n3** is then shifted to the left and filled with blanks until the field **n1,n2** is shifted out. Then, the character string **string** shifts into the left end of the field **n1,n3**. There is no check on the length of **string** and shifting continues until it is left-justified in the field **n1,n3** with excess characters, if any, being truncated on the right.

The record remains in the memory buffer, thus permitting multiple string operations on the same record. (If IN is already positioned at **recno** because of a previous string operation, there is, of course, no change in position.)

The record **recno** is read out of the memory buffer and onto the OUT unit when a SEDIT directive affecting another record is input.

The field specification **n1,n2,n3** is lost after one manipulation. Subsequent string operations must specify the character positions based on the new configuration.

Example: Copy records from IN onto OUT up to and including record 49, and replace the present contents of character positions 10 through 12, inclusive, in IN unit source record 50 with the character string XYb.

SR, 50, (10, 12, 12), 'XY '

8.2.6 DE (Delete Records) Directive

This directive deletes a sequence of source records. It has the general form

DE,recno1,recno2

where

recno1 is the number of the first record to be deleted

recno2 is the number of the last record to be deleted

If **recno2** is omitted, it is assumed equal to **recno1**, i.e., one record will be deleted.

The DE directive processing is exactly like that of the REPL directive (section 8.2.4) except that there is no copying from the ALT unit after the deletion of the records **recno1** through **recno2**, inclusive.

Examples: Copy records from IN onto the OUT logical unit up to and including record 49, but delete records 50 through 54, inclusive.

DE, 50, 54

Position IN at record 2, deleting record 1.

DE, 1

8.2.7 SD (Delete String) Directive

This directive deletes a character string from a source record. It has the general form

SD,recno,(n1,n2,n3)

where

recno	is the number of the source record from which the character string is to be deleted
n1	is the number of the first character position of the string to be deleted
n2	is the number of the last character position of the string to be deleted
n3	is the number of the last character position of the field in which the string to be deleted occurs

The SD directive processing is exactly like that of the SR directive (section 8.2.5) except that no new character string is shifted into field **n2,n3** after the field **n1,n2** is shifted out.

Example: Copy records from IN onto OUT up to and including record 99, and delete characters 2 through 4, inclusive, from record 100, shifting characters 5 through 10, inclusive, three places to the left, with blank fill on the right.

SD, 100, (2, 4, 10)

8.2.8 MO (Move Records) Directive

This directive moves a block of records forward on a unit. It has the general form

MO,recno1,recno2,recno3

where

recno1	is the number of the first record to be moved
recno2	is the number of the last record to be moved
recno3	is the number of the record after which the block of records delimited by recno1 and recno2 is to be inserted

If **recno2** is omitted, it is assumed equal to **recno1**, i.e., one record will be moved.

The MO directive copies source records from the IN logical unit onto the OUT logical unit beginning with the current position of the IN unit and continuing up to but *not* including the record specified by **recno1**. The records **recno1** through **recno2** are then read into a special MOVE area in memory. The position of IN is now **recno2**+1. When OUT reaches (by some succeeding directive) **recno3**+1, the contents of the MOVE area are copied onto OUT. Multiple MO operations are legal.

Example: Copy records from IN onto OUT up to and including record 4, save records 5 through 10, inclusive, in the MOVE area of memory, copy records 11 through 99, inclusive, from IN onto OUT, and then copy records 5 through 10 from the MOVE area to OUT. This gives a record sequence on OUT of 1-4, 11-99, 5-10 (FC directive, section 8.2.9.).

MO, 5, 10, 99
FC

8.2.9 FC (Copy File) Directive

This directive copies blocks of files, including end-of-file marks. It has the general form

FC,nfiles

where **nfiles** (default value = 1) is the number of files to be copied.

If the IN logical unit and/or the OUT logical unit is an RMD partition, **nfiles** must be 1 or absent. If OUT is a named file on an RMD, there will be an automatic close/update. Whenever an end-of-file mark is encountered, all record counters are reset to zero.

SOURCE EDITOR

Examples: Copy files from IN onto OUT up to and including the next end-of-file mark on the IN unit.

FC

Copy the next six IN files (including end-of-file marks) onto OUT. This includes the sixth end-of-file mark. (Note: If IN and/or OUT is an RMD partition, there will be an error.)

FC, 6

8.2.10 SE (Sequence Records) Directive

This directive assigns a decimal sequence number to each source record output to the OUT logical unit. It has the general form

SE,(first,last),initial,increment

where

<i>first</i>	is the first character position of the sequence name field
<i>last</i>	is the last character position of the sequence number field, where the default value of <i>first,last</i> is 76,80
<i>initial</i>	is the initial number to be used as a sequence number (default value = 10)
<i>increment</i>	is the increment to be used between successive sequence numbers (default value = 10)

There is also a special form of the SE directive to stop sequencing:

SE,N

where there are no parameters other than the letter **N**.

Examples: In the next record output to OUT, place 00010 in character positions 76 through 80, and increment the field by 10 in each succeeding record.

SE

In the next record output to OUT, place 030 in character positions 15 through 17, and increment the field by 7 on each succeeding record.

SE, (15, 17), 30, 7

Stop sequencing.

SE, N

8.2.11 LI (List Records) Directive

This directive lists, on the LO logical unit, the records copied onto the OUT unit. The LI directive has the general form

LI,list

where *list* is A (default value) if all OUT records are to be listed, C if only changed records are to be listed, or N if listing is to be suppressed. Source records output to the OUT file are listed with their OUT record number at the left of the print list.

Examples: List all records output to OUT.

LI

Suppress all listing except that of SEDIT directives.

LI, N

8.2.12 GA (Gang-Load All Records) Directive

This directive loads the same character string into the specified field of every record copied onto the OUT logical unit. It has the general form

GA,(first,last),'string'

where

<i>first</i>	is the first character position of the field to be gang-loaded
<i>last</i>	is the last character position of the field to be gang-loaded, where the default value of <i>first,last</i> is 73,75
string	is the string of characters to be gang-loaded into character positions <i>first</i> through <i>last</i> , inclusive in all records copied onto out

There is also a special form of the GA directive to stop gang-loading:

GA

where there are no parameters in the directive.

In every OUT record, GA clears the specified field, and loads the string into it. There is no check on the length of **string** and shifting continues until it is left-justified in the specified field with excess characters, if any, being truncated on the right.

Examples: Load character string VDMbb in character positions 11 through 15, inclusive, of every record copied onto OUT.

```
GA, ( 11, 15 ), 'VDM '
```

Stop gang-loading.

```
GA
```

8.2.13 WE (Write End of File) Directive

This directive writes an end-of-file mark on the OUT logical unit. It has the form

```
WE
```

without parameters. If OUT is a named file on an RMD, there will be an automatic close/update.

Example: Write an end-of-file mark on OUT, a magnetic-tape unit.

```
WE
```

8.2.14 REWI (Rewind) Directive

This directive rewinds the specified SEDIT logical unit(s). It has the general form

```
REWI,p(1),p(2),p(3)
```

where each *p(n)* is a name of one of the SEDIT logical units: IN, OUT, or ALT. These can be coded in any order.

Example: Rewind all SEDIT logical units.

```
REWI, IN, ALT, OUT
```

8.2.15 CO (Compare Inputs) Directive

This directive compares the specified field in the inputs from the IN logical unit with those from the ALT logical unit and lists discrepancies on the LO logical unit. The directive has the general form

```
CO,(first,last),limit
```

where

<i>first</i>	is the first character position of the field to be compared
<i>last</i>	is the last character position of the field to be compared, where the default value of <i>first,last</i> is 1,80.

limit is the maximum number of discrepancies to be listed before aborting the comparison and passing to the next directive.

Any discrepancy between the IN and ALT inputs is listed in the format

```
I recordnumber or EOF inrecord
A recordnumber or EOF altrecord
```

If the comparison terminates by reaching the *limit* number of discrepancies, SEDIT outputs on the LO the message

```
SEEDIT COMPARE ABORTED
```

to prevent long listings of errors, for example, where a card is misplaced or missing on one input. A normal termination of a comparison (at the next end-of-file mark) concludes with the message

```
SEEDIT COMPARE FINISHED
```

Example: Compare character positions 1 through 80, inclusive, from the IN and ALT units until either an end of file is found or there have been 5 discrepancies listed on the LO.

```
CO, , 5
```

8.3 EXAMPLE OF EDITING A FILE

Following is a sample job stream for editing an existing file on a magnetic tape onto a new file on magnetic tape. The input file consists of 80-character records followed by an end-of-file mark. The job stream and the edit cards are read through the system input device.

```
/JOB, EDIT
/ASSIGN, PI=MT00, PO=MT10
/REW, PI, PO
/SEEDIT
AS, IN=PI
AS, OUT=PO
AS, ALT=SI
DE, 5
REPL, 8, 10
                                LDA    TEMP
(EOF card, 2-7-8-9 punch)
ADD, 17
TBL    BSS    5
(EOF card, 2-7-8-9 punch)
FC
REWI, IN, OUT
/ENDJOB
```

SOURCE EDITOR

The result of running the preceding source editor example would be the following:

Input File				Output File			
1	*			1	*		
2	*	CATALOG	ROUTINE	2	*	CATALOG	ROUTINE
3	*			3	*		
4	A\$3	EQU	6	4	A\$3	EQU	6
5	B\$3	EQU	9	5	*		
6	*			6	CATLOG	DATA	0
7	CATLOG	DATA	0	7		LDA	TEMP
8		LDA	TMX	8		ADD	PARM6
9		LDB	TMY	9		ANAI	0770
10		JBZM	ODER	10		STA	TBL+2
11		ADD	PARM6	11		LRLA	6
12		ANAI	0770	12		STA	TBL+4
13		STA	TBL+2	13		TZB	
14		LRLA	6	14		JMP*	CATLOG
15		STA	TBL+4	15	TBL	BSS	5
16		TZB					
17		JMP*	CATLOG				

SECTION 9 FILE MAINTENANCE

The VORTEX file-maintenance component (**FMAIN**) is a background task that manages file-name directories and the space allocations of the files. It is scheduled by the job-control processor (JCP) upon input of the JCP directive /FMAIN (section 4.2.18).

Only files assigned to rotating-memory devices (disc or drum) can be referenced by name.

File space is allocated within a partition forward in contiguous sectors of the same cylinder, skipping bad tracks. The only exception to this continuity is the file-name directory itself, which is a sequence of linked sectors that may or may not be contiguous.

9.1 ORGANIZATION

FMAIN inputs file-maintenance directives (section 9.2) received on the SI logical unit and outputs them on the LO logical unit and on the SO logical unit if it is a different physical device from the LO unit. Each directive is completely processed before the next is input to the JCP buffer.

If the SI logical unit is a Teletype or a CRT device, the message **FM**** is output on it before input to indicate that the SI unit is waiting for FMAIN input.

If there is an error, one of the error messages given in Appendix A.9 is output on the SO logical unit, and a record is input from the SO unit to the JCP buffer. If the first character of this record is /, FMAIN exits via the EXIT macro. If the first character is C, FMAIN continues. If the first character is neither / nor C, the record is processed as a normal FMAIN directive. FMAIN continues to input and process records until one whose first character is / is detected, when FMAIN exits via exit. (An entry beginning with a carriage return is an exception to this, being processed as an FMAIN directive).

FMAIN has a symbol-table area for 200 symbols at five words per symbol. To increase this area, input a /MEM directive (section 4.2.5), where each 512-word block will enlarge the capacity of the table by 100 symbols.

9.1.1 Partition Specification Table

Each rotating-memory device (RMD) is divided into up to 20 memory areas called **partitions**. Each partition is

referenced by a specific logical-unit number. The boundaries of each partition are recorded in the core-resident **partition specification table (PST)**. The first word of the PST contains the number of VORTEX physical records per track. The second word of the PST contains the address of the bad-track table, if any. Subsequent words in the PST comprise the four-word partition entries. Each PST is in the format:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 0	Size of bad track table (120-words)															
Word 1	Address of bad track table (0 if none) relative to word 0															
Word 0	Beginning partition track address															
Word 1	PPB	Not used						Protection code								
Word 2	Number of bad tracks in partition															
Word 3	Ending partition address + 1															
	• • •															

The partition protection bit, designated ppb in the above PST entry map, is unused in file maintenance procedures.

Note that PST entries overlap. Thus, word 3 of each PST entry is also word 0 of the following entry. The relative position of each PST entry is recorded in the **device specification table (DST)** for that partition.

The **bad-track table**, whose address is in the second word of the PST, is a bit string read from left to right within each word, and forward through contiguous words, with set bits flagging bad tracks on the RMD. (If there is no bad-track table, the second word of the PST contains zero.)

9.1.2 File-Name Directory

Each RMD partition contains a **file-name directory** of the files contained in that partition. The beginning of the directory is in the first sector of the partition. The directory for each partition has a variable number of entries arranged in n sectors, 19 entries per sector. Sectors containing directory information are chained by pointers in

FILE MAINTENANCE

the last word of each sector. Thus, directory sectors need not be contiguous. Each directory entry is in the format:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 0	File name															
Word 1	File name															
Word 2	File name															
Word 3	Current position of file															
Word 4	Beginning file address															
Word 5	Ending file address															

The file name comprises six ASCII characters packed two characters per word, left justified, with blank fill. Word 3, which contains the current address at which the file is positioned, is initially set to the ending file address, and is manipulated by I/O control macros (section 3). The extent of the file is defined by the addresses set in words 4 and 5 when the file is created, and remains constant.

The first sector of each partition is assigned to the file-name directory. FMAIN allocates RMD space forward in contiguous sectors, skipping bad tracks. Following the last entry in each directory sector is a one-word tag containing either the value 01 (end of directory), or the address of the next sector of the file-name directory.

The file-name directories are created and maintained by the file-maintenance component for the use of the I/O control component (section 3). User access to the directories is via the I/O control component.

Special entries: A **blank entry** is created when a file name is deleted, in which case the file name is ********* and words 3 through 5 give the extent of the blank file. A **zero entry** is created when one name of a multiname file is deleted, in which case the deleted name is converted to a *blank entry* and all other names of the multiname file are set to zero.

WARNING

To prevent possible loss of data from the file-name directory during file-maintenance operations, FMAIN sets the **lock bit** (bit 12 of word 2 of the DST) before any directory operation, thus inhibiting all foreground requests for I/O with the partition being modified. Upon completion of the directory operation, FMAIN clears the lock bit. Except for the use of protection codes, **this is the only protection for the file-name directory**. Manipulation of foreground files with FMAIN is at the user's risk. For example, VORTEX does not prevent deletion of a file name from a file-name directory that has been opened and is being written into by a foreground program. Therefore, foreground files should be reassigned prior to manipulation by FMAIN.

9.1.3 Relocatable Object Modules

Outputs from both the DAS MR assembler and the FORTRAN compiler are in the form of relocatable object modules. Relocatable object modules can reside on any VORTEX-system logical unit. Before object modules can be read from a unit by the FMAIN INPUT and ADD directives (sections 9.2.7 and 9.2.8), an I/O OPEN with rewinding (section 3.5.1) is performed on the logical unit, i.e., the unit (except paper-tape or card readers) is first positioned to the beginning of device or load point for that unit. Object modules can then be loaded until an end-of-file mark is found.

The system generator (section 15) does not build any object-module library. FMAIN is the only VORTEX component used for constructing user object-module libraries.

A VORTEX physical record on an RMD is 120 words. Object-module records are blocked two 60-word records per VORTEX physical record. However, in the case of an RMD assigned as the SI logical unit, object modules are not blocked but assumed to be one object-module record per physical record.

9.1.4 Output Listings

FMAIN outputs four types of listing to the LO logical unit:

- **Directive listing** lists, without modification, all FMAIN directives entered from the SI logical unit.
- **Directory listing** lists file names from a logical unit file-name directory in response to the FMAIN directive LIST (section 9.2.5).
- **Deletion listing** lists file names deleted from a logical unit file-name directory in response to the FMAIN directive DELETE (section 9.2.2).
- **Object-module listing** lists the object-module input in response to the FMAIN directive ADD (section 9.2.8).

All FMAIN listings begin with the standard VORTEX heading.

The *directory listing* is further described under the discussion of FMAIN directive LIST (section 9.2.5), the *deletion listing* under DELETE (section 9.2.2), and the *object-module listing* under ADD (section 9.2.8).

9.2 FILE-MAINTENANCE DIRECTIVES

This section describes the file-maintenance directives:

- CREATE file
- DELETE file
- RENAME file
- ENTER new file name
- LIST file names
- INIT (initialize) directory
 - INPUT logical unit for object module
 - ADD object module

File-maintenance directives comprise sequences of character strings having no embedded blanks. The character strings are separated by commas (,) or by equal signs (=). The directives are free-form and blanks are permitted between the individual character strings of the directive, i.e., before or after commas (or equal signs). Although not required, a period (.) is a line terminator. Comments can be inserted after the period.

The general form of a file-maintenance directive is

directive,lun,p(1),p(2),...,p(n)

where

directive	is one of the directives listed above in capital letters
lun	is the number or name of the affected logical unit
each p(n)	is a parameter defined under the descriptions of the individual directives below

Numerical data can be octal or decimal. Each octal number has a leading zero.

For greater clarity in the descriptions of the directives, optional periods, optional blank separators between character strings, and the optional replacement of commas (,) by equal signs (=) are omitted.

Error messages applicable to file-maintenance directives are given in Appendix A.9.

9.2.1 CREATE Directive

This directive creates a new file on the specified logical unit, allocates RMD space to the file, adds a corresponding entry to the file-name directory, and sets the current end-of-file value to one greater than the address of the last sector assigned to the new file.

The directive has the general form

CREATE,lun,key,name,words,records

where

lun	is the number or name of the logical unit where the new file is to be created
key	is the protection code, if any, required to address lun
name	is the name of the file being created
words	is the number of words in each record of the file
records	is the number of records in the file

Size parameters merely allocate space for the file and do not limit file use to the specified record size. To each record in the created file, FMAIN assigns n records of 120 words each where n is the smallest integer such that $\text{words}/120$ is less than or equal to n . The file size is n times **records** words. This value is converted to a sector count to make assignments. Neither the file size value nor the sector count value is saved.

Example: Create the file XFILE with ten records of 120 words each on logical unit 112, whose protection code is K.

CREATE, 112, K, XFILE, 120, 10

9.2.2 DELETE Directive

This directive deletes the designated file and all file-name directory references to it from the specified logical unit. It converts the specified file-name directory entry to a blank entry (name field = *****, section 9.1.2) and all other directory references to this file to zero entries (all fields = zero, section 9.1.2), and outputs a listing of deleted file-names on the LO logical unit. The directive has the general form

DELETE,lun,key,name

where

lun	is the number or name of the logical unit from which the file is being deleted
key	is the protection code, if any, required to address lun
name	is the name of the file being deleted (in the case of a multiname file, any one of the names can be used, all names are deleted)

The output format has, following the FMAIN heading, a two-line heading

DELETE LISTING FOR lun
FILE NAME START END CURRENT

where **lun** is the number of the logical unit from which the file is being deleted. This heading is followed by a blank line and a listing of all file-names being deleted, one per line. Words 0-2 of the file-name directory entry (section 9.1.2) are placed in the FILE NAME column; word 3, (in octal) in the CURRENT column; word 4, (in octal) in the START column; and word 5, (in octal) in the END column. After the last file name, there is an entry describing the blank file created by the deletion, where the FILE NAME column contains *****, the START column contains the next available address (word 2 of the PST entry), and both the CURRENT and END columns contain the last address + 1 (word 3 of the PST entry).

FILE MAINTENANCE

Example: Delete the file ZFILE (and all file-name directory entries referencing it) from logical unit 112, whose protection code is P).

DELETE, 112, P, ZFILE

The name ZFILE is replaced in the file-name directory by ***** and the space allocation for this blank entry extended in both directions to include adjacent blank entries, if any. Any blank entries thus absorbed are converted to zero entries, as are all other entries that reference the file ZFILE. All affected file-name directory entries are listed on the LO logical unit.

9.2.3 RENAME Directive

This directive changes the name of a file, but does not otherwise modify the file-name directory. The directive has the general form

RENAME, lun, key, old, new

where

- lun** is the number or name of the logical unit where the file to be renamed is located
- key** is the protection code, if any, required to address **lun**
- old** is the old name of the file being renamed
- new** is the new name of the file being renamed

Following RENAME, **old** can no longer be used to reference the file.

Example: On logical unit 112, whose protection code is P, change the name of the file XFILE to YFILE.

RENAME, 112, P, XFILE, YFILE

9.2.4 ENTER Directive

This directive adds a new file name to be used in referencing an existing file, but does not otherwise modify the file-name directory. ENTER thus permits multiname access to a file. The directive has the general form

ENTER, lun, key, old, new

where

- lun** is the number or name of the logical unit where the affected file is located
- key** is the protection code, if any, required to address **lun**
- old** is an old name of the affected file
- new** is the new name by which the file can also be referenced

Example: On logical unit 113, whose protection code is K, make the file X1 accessible by using either the name X1 or the name Y1.

ENTER, 113, K, X1, Y1

9.2.5 LIST Directive

This directive outputs on the LO logical unit the file-name directory of the specified logical unit. The output comprises the file names, file extents, current end-of-file positions, logical-unit name or number, and the extent of unassigned space in the partition. All numbers are in octal. The directive has the general form

LIST, lun, key

where

- lun** is the number or name of the logical unit whose contents are to be listed
- key** is the protection code, if any, required to address **lun**

The output format has a two-line heading

FILE DIRECTORY FOR LUN lun
FILE NAME START END CURRENT

where **lun** is the number or name of the logical unit whose contents are being listed. This heading is followed by a blank line and a listing of all file names from the directory, one name per line. Words 0-2 of the file-name directory entry (section 9.1.2) are placed in the FILE NAME column; word 4, (in octal) in the START column; word 3, (in octal) in the CURRENT column; and word 5, (in octal) in the END column. After the last file name, if there is any unassigned space in the partition, there is an entry describing the unassigned space in the partition, where the FILE NAME column contains *UNAS*, the START column contains the next available address, and both the CURRENT and END columns contains the last address + 1. All numerical values are octal sectors.

Example: List the file-name directory of logical unit 114, which has no protection code.

LIST, 114

9.2.6 INIT (Initialize) Directive

This directive clears the entire file-name directory of the specified logical unit, deletes all file names in it, and releases all currently allocated file space in the partition by reducing the file-name directory to a single end-of-directory entry. The directive has the general form

INIT, lun, key

where

- lun** is the number or name of the logical unit being initialized
- key** is the protection code, if any, required to address **lun**

9.2.3 RENAME Directive

This directive changes the name of a file, but does not otherwise modify the file-name directory. The directive has the general form

RENAME,lun,key,old,new

where

lun is the number or name of the logical unit where the file to be renamed is located

key is the protection code, if any, required to address **lun**

old is the old name of the file being renamed

new is the new name of the file being renamed

Following RENAME, **old** can no longer be used to reference the file.

Example: On logical unit 112, whose protection code is P, change the name of the file XFILE to YFILE.

RENAME, 112, P, XFILE, YFILE

9.2.4 ENTER Directive

This directive adds a new file name to be used in referencing an existing file, but does not otherwise modify the file-name directory. ENTER thus permits multiname access to a file. The directive has the general form

ENTER,lun,key,old,new

where

lun is the number or name of the logical unit where the affected file is located

key is the protection code, if any, required to address **lun**

old is an old name of the affected file

new is the new name by which the file can also be referenced

↓ This directive will also include the creation date, access date, extension numbers, and file type information in the corresponding shadow directory.
↑

9.2.5 LIST Directive

This directive outputs to the LO logical unit the file-name and shadow directories of the specified logical unit. The output comprises the file name, file extent, current end-of-

file position, logical unit name or number, extent of unassigned space in the partition, file type, file extension number, and the dates the file was created and last accessed. All numbers are in octal except for dates and extensions. The directive has the general form:

LIST,lun,key

where

lun is the number or name of the logical unit whose contents are to be listed.

key is the protection code, if any, required to access **lun**.

The output format has a two line heading:

```
FILE DIRECTORY FOR LUN XXX
FILE NAME START END CURRENT F-TYPE EX CREATED
ACCESSED
```

where

XXX is the number or name of the logical unit whose contents are being listed.

The header is followed by a blank line and a listing of all file name from the directory. See section 9.1.2 for a description of header items. After the last file name, if there is any unassigned space in the partition, there is an entry describing the unassigned space in the partition, where the FILE NAME column contains *UNAS*, the START column contains the next available address, and both the CURRENT and END columns contain the last address + 1. All numerical values are octal sectors.

Example: List the file name directory of logical unit 114 which has no protection code.

LIST,114

9.2.6 INIT (Initialize) Directive

This directive clears the entire file-name directory of the specified logical unit, deletes all file names in it, and releases all currently allocated file space in the partition by reducing the file-name directory to a single end-of-directory entry. The directive has the general form

INIT,lun,key

where

lun is the number or name of the logical unit being initialized

key is the protection code, if any, required to address **lun**

FILE MAINTENANCE

Example: Initialize the file-name directory on logical unit 115, which has protection code X.

```
INIT, 115, X
```

9.2.7 INPUT Directive

This directive specifies the logical unit from which object modules are to be input. Once specified, the input logical-unit number is constant until changed by a subsequent INPUT directive. The directive has the general form

```
INPUT, lun, key, file
```

where

lun is the number or name of the logical unit from which object modules are to be input

key is the protection code, if any, required to address lun

file is the name of the RMD file containing the required object module(s)

Neither *key* nor *file* are required unless *lun* is a RMD partition.

NOTE

There is no default value. Thus, if an attempt is made to input an object module (ADD directive, section 9.2.8) without defining the input logical unit by an INPUT directive, an error message will be output.

Example:

Open and rewind the file ARCTAN on logical unit 104, which has protection code D.

```
/PFILE, 104, D, ARCTAN  
/FMAIN  
INPUT, 104, D, ARCTAN
```

```
INPUT, 104, D, ARCTAN
```

9.2.8 ADD Directive

This directive reads object modules from the INPUT unit (section 9.2.7) and writes them onto the SW logical unit, checking for entry names and validating check-sums, record sizes, loader codes, sequence numbers, and record

structures. Reading continues until an end of file is encountered. Entry names are then added to the file-name directory of the specified logical unit and the object modules are copied from the SW logical unit onto the specified logical unit. The ADD directive also sets up the date in word 5 of the corresponding shadow directory and sets up the "file type" variable in word 3 of the shadow directory to "object module" (bit 0 is set). The ADD directive has the general form

```
ADD, lun, key
```

where

lun is the number or name of the logical unit onto which object modules are to be written

key is the protection code, if any, required to address lun

The specified logical unit *lun* references a system or user object-module library.

The names of the object modules and their date of generation, size in words (zero for FORTRAN modules), entry names, and referenced external names are listed on the LO logical unit.

To recover from errors in object-module-processing, reposition the logical unit to the beginning of the module.

Example: Add object modules to logical unit 104, which has protection code D.

```
ADD, 104, D
```

Note: When using the INPUT and ADD directives with an RMD device, the files must be previously positioned using the /PFILE directive.

9.3 VORTEX FILE MAINTENANCE DRIVER (VZFMA)

The VORTEX File Maintenance driver provides a user-programmable subset of the VORTEX FMAIN services. VZFMA operates as a system driver assigned to logical unit 115. All requests to VZFMA must be made through the OM library resident interface routine, V\$FILE. Direct calls to VZFMA are not allowed. This is because conflicts arise in calling sequences if VZFMA services should be augmented.

The calling sequence to request a file service is as follows:

```

EXT      V$FILE
LDAI    code
LDBI    fmcB
JSR     V$FILE,X
    
```

where

code is the operation code for the requested service

- 0 = create
- 1 = delete
- 2 = rename
- 3 = enter
- 4 = unused
- 5 = find

fmcB is the address of the file maintenance control block (see table 9-1)

file which is that area of the file beyond the current end-of-file.

Upon exit from a file request the A register contains the completion status code.

The completion status codes are as follows:

- 0 request completed without error
- 1 invalid request code
- 2 name already in directory
- 3 name not found
- 4 insufficient space
- 5 input/output error occurred
- 6 directory structure error
- 15 file is empty

The create, delete, rename and enter requests perform the same operations as in the VORTEX FMAIN program. The unused request releases the unused portion of the named

Note: Completion code 5 can also indicate invalid LUN, invalid "CREATE" sector count, (≤ 0) or invalid protection key.

The file maintenance control blocks for the requests must be arranged as shown in Table 9-1.

Table 9-1. File Maintenance Control Block

The file maintenance control blocks for the requests must be arranged as follows:

Word	Create	Delete, Unused, Find	Rename, Enter
0	logical unit	logical unit	logical unit
1	key	key	key
2 } 3 } 4 }	file name	file name	current file name
5	number of sectors		} new file name
6	0		
7	0		

Words 0 and 1 are right justified and zero filled.

The file name (words 2-4) consists of two characters per word, left justified and blank filled.

For CREATE, word 6 must be zero and word 5 is right justified and zero filled. Also, if the number of sectors specified in word 5 is an odd number, it is rounded up to the next even number.

FILE MAINTENANCE

9.3.1 IOC FUNC Requests to VZFMA

Note: In most instances, the user should call VZFMA via V\$FILE as describes in Section 9.3

Requests to VZFMA may be via an IOC FUNC request of the form

FUNC fsb,lun,wait

where

- fsb** is the address of the file specification block
- lun** is the VZFMA logical unit number (usually 115)
- wait** is 0 for wait until request is complete

The FSB has the format:

Word	Contents		
0	Opcode		
1	LUN Range		
2	Key		
3	Record Count		
4 } 5 } 6 } 7 }	Secondary File Name		
8 } 9 }		Primary File Name	
10 } 11 } 12 }			File Attributes
		file type	
	access date		
	creation date		

9.3.1.1 Exit Conditions

Upon exit, VZFMA returns **lun** in high byte of FSB word 2 for FIND.

9.3.1.2 Errors

Error codes are returned in bits 9-14 of I/O request block status word. Error codes are:

- 1 = invalid opcode
- 2 = invalid lun or high lun not \geq low lun
- 3 = lun not assigned to RMD
- 4 = insufficient RMD space
- 5 = I/O error
- 6 = directory structure error
- 7 = file name already in directory
- 8 = file name not found
- 9 = invalid "CREATE" sector count (i.e. ≤ 0)
- 10 = partition locked out too long
- 11 = invalid protect key

9.3.1.3 Explanation of FSB Contents

Opcodes

- 0 = Create
- 1 = Delete
- 2 = Rename
- 3 = Enter
- 4 = Delete Unused
- 5 = Find
- 6 = Find extension file
- 7 = Create extension file
- 8 = Create file using supplied attributes
- 9 = Enter file using supplied attributes

LUN Range

High byte = lowest lun of range of FIND
 = lun for other opcodes
 Low byte = highest lun of range for FIND

Key

Partition protection key in low byte

Record Count

Number of records for CREATE

Secondary file name

Name of new file new RENAME and ENTER

Primary file name

Name of old file for RENAME and ENTER. Name of file for other opcodes.

File attributes

The format of words 10-12 of the FSB is the same as that of words 3-5 of the shadow directory (Section 9.1.2).

SECTION 10 INPUT/OUTPUT UTILITY PROGRAM

The I/O utility program (IOUTIL) is a background task for copying records and files from one device onto another, changing the size and mode of records, manipulating files and records, and formatting the records for printing or display.

10.1 ORGANIZATION

IOUTIL is scheduled for execution by inputting JCP directive /IOUTIL (section 4.2.20) on the SI logical unit. If the SI logical unit is a Teletype or a CRT device, the message IU** is output to indicate that the SI unit is waiting for IOUTIL input. Once activated, IOUTIL inputs and executes directives from the SI unit until another JCP directive (first character is a slash) is input, at which time IOUTIL terminates and the JCP is again scheduled.

"The IOUTIL buffer is usually 1024 words long. The /MEM directive can be used to increase this size by increments of 512 words."

IOUTIL has the option of calling V\$RSW (multi-volume reel-switch routine), when using a copy file, copy record, skip file, skip record, format and dump, position file, and pack binary.

Error Messages applicable to IOUTIL are given in Appendix A.10. Recovery from an error is by either of the following:

- a. Input the character C on the SO unit, thus directing IOUTIL to go to the SI unit for the next directive.
- b. Input the corrected directive on the SO unit for processing. The next IOUTIL directive is then input from the SI unit.

If recovery is not desired, input a JCP directive (section 4.2) on the SO unit to abort IOUTIL and schedule the JCP for execution.

10.2 I/O UTILITY DIRECTIVES

This section describes the IOUTIL directives:

- COPYF Copy file
- COPYR Copy record
- SFILE Skip file
- SREC Skip record
- DUMP Format and dump
- PRNTF Print file
- WEOF Write end of file
- REW Rewind
- PFILE Position file
- CFILE Close file
- PACKB Pack binary

IOUTIL directives begin in column 1 and comprise sequences of character strings having no embedded

blanks. The character strings are separated by commas (,) or by equal signs (=). The directives are free-form and blanks are permitted between individual character strings of the directive, i.e., before or after commas (or equal signs). Although not required, a period (.) is a line terminator. Comments can be inserted after the period.

The general form of an IOUTIL directive is

name,*p*(1),*p*(2),...,*p*(*n*)

where

name is one of the directive names given above

each *p*(*n*) is a parameter defined below under the descriptions of the individual directives

Numerical data can be octal or decimal. Each octal number has a leading zero.

For greater clarity in the descriptions of the directives, optional periods, optional blank separators between character strings, and the optional replacement of commas (,) by equal signs (=) are omitted.

The IOUTIL buffer is usually 1024 words long. The /MEM directive can be used to increase this size by increments of 512 words.

10.2.1 COPYF (Copy File) Directive

This directive copies the specified number of files from the indicated input logical unit to the given output logical unit(s). The directive has the general form

COPYF,*f*,*iu*,*im*,*irl*,*ou*(1),*om*,*orl*,*ou*(2),*ou*(3),...,*ou*(*n*)

where

f is the number of input files to be copied (must be 1 for RMD)

iu is the name or number of the input logical unit

im is 0 for binary, 1 for ASCII, 2 for BCD, or 3 for unformatted input files

irl is the number of words in each record of the input files. If a value of zero is specified then the record length is set to the maximum buffer size. Following the

INPUT/OUTPUT UTILITY PROGRAM

- ou(n)** is the name or number of an output logical unit
- om** is 0 for binary, 1 for ASCII, 2 for BCD, or 3 for unformatted output files
- orl** is the number of words in each record of the output files. If a value of zero is specified then the output record length is equal to the input record length.

Any RMD involved with copying files, whether as input or output medium, must have been previously positioned with a PFILE directive (section 10.2.9).

If a difference in record lengths **irl** and **orl** causes a partial record to remain when an end of file is encountered, the part-record is filled with blanks and thus transmitted to the output unit(s).

The following relation holds for input/output record lengths:

Input RCL	Output RCL	Output Format
fixed	fixed	As defined (blocked or unblocked)
random (0)	fixed	As defined (blocked or unblocked)
fixed	random (0)	Unblocked only
random (0)	random (0)	Unblocked only

Record lengths of zero are useful in copying mixed ASCII and binary data from cards to another media or vice versa. ASCII read must be specified for this operation.

Example: Copy three files containing 120-word records from the PI logical unit onto logical units LO, 50, and 51 in 40-word records.

```
COPYF, 3, PI, 1, 120, LO, 1, 40, 50, 51
```

10.2.2 COPYR (Copy Record) Directive

This directive copies the specified number of records from the indicated input logical unit to the given output logical unit(s). The directive has the general form

```
COPYR,r,iu,im,irl,ou(1),om,orl,ou(2),ou(3),...,ou(n)
```

where

- r** is the number of input records to be copied, or 0 if copying is to continue to the end of file

- iu** is the name or number of the input logical unit
- im** is 0 for binary, 1 for ASCII, 2 for BCD, or 3 for unformatted input records
- irl** is the number of words in each record of the input files. If a value of zero is specified then the record length is set to the maximum buffer size. Following the read the actual physical record length (word 5 of the RQBLK) is used as the input record length.
- each **ou(n)** is the name or number of an output logical unit
- om** is 0 for binary, 1 for ASCII, 2 for BCD, or 3 for unformatted output records
- orl** is the number of words in each record of the output files. If a value of zero is specified then the output record length is equal to the input record length.

Any RMD involved with copying records, whether as input or output medium, must have been previously positioned with a PFILE directive (section 10.2.9).

If a difference in record lengths **irl** and **orl** causes a part-record to remain when an end-of-file mark is encountered, the part-record is filled with blanks and thus transmitted to the output unit(s).

Example: Copy 25 unformatted records of 200 words each from the SS logical unit to the BO and PO units in binary format with 40 words per record.

```
COPYR, 25, SS, 3, 200, BO, 0, 40, PO
```

It may be necessary to copy from one file on an RMD partition to another file on the same partition. This can be accomplished by assigning two *different* logical units to this RMD partition, and then issuing two PFILE directives (section 10.2.9), positioning one logical unit to the beginning of one file and the second logical unit to the beginning of the other file. Additional positioning within the files can be specified by SREC directives (section 10.2.4).

The following relation holds for input/output record lengths:

Input RCL	Output RCL	Output Format
fixed	fixed	As defined (blocked or unblocked)
random (0)	fixed	As defined (blocked or unblocked)

Input RCL	Output RCL	Output Format
fixed	random (0)	Unblocked only
random (0)	random (0)	Unblocked only

Record lengths of zero are useful in copying mixed ASCII and binary data from cards to another media or vice versa. ASCII read must be specified for this operation.

Example: Copy the first ten records from file EDIT1 to record 11 through 20 of file EDIT2. Both files are on RMD partition D00K, have record lengths of 120 words, are in mode 1, and have no protection key (default value = 0). Assign the BI and BO logical units to the disc.

```

/ASSIGN, BI=D00K
/ASSIGN, BO=D00K
/IOUTIL
PFILE, BI, , 120, EDIT1
PFILE, BO, , 120, EDIT2
SREC, BO, 10
COPYR, 10, BI, 1, 120, BO, 1, 120
    
```

10.2.3 SFILE (Skip File) Directive

This directive, which applies only to magnetic-tape units and card readers, causes the specified logical unit to move the tape *forward* the designated number of end-of-file marks. The directive has the general form

SFILE,lun,neof

where

lun	is the name or number of the affected logical unit
neof	is the number of end-of-file marks to be skipped

If the end-of-tape mark is encountered before the required number of files has been skipped, IOUTIL outputs to the SO and LO logical units the error message IU05,nn, where nn is the number of files remaining to be skipped.

Example: Move tape on unit PI past three end-of-file marks.

```
SFILE, PI, 3
```

10.2.4 SREC (Skip Record) Directive

This directive, which applies only to magnetic-tape units, card readers and RMDs, causes the specified logical unit to skip *forward* the designated number of records. The directive has the general form

SREC,lun,nrec

where

lun	is the name or number of the affected logical unit
nrec	is the number of records to be skipped

Note that, unlike JCP directive /SREC (section 4.2.8), the IOUTIL directive SREC cannot skip records in reverse.

If lun designates an RMD partition, the device must have been previously positioned with a PFILE directive (section 10.2.9).

If a file mark, an end-of-tape mark, or an end-of-device mark is encountered before the required number of records has been skipped, IOUTIL outputs to the SO and LO logical units the error message IU05,nn, where nn is the number of records remaining to be skipped.

Example: Skip 40 records on the BI logical unit.

```
SREC, BI, 40
```

10.2.5 DUMP (Format and Dump) Directive

This directive copies the specified number of records from the indicated input logical unit, formats them for listing, and dumps the data onto the output unit in octal format, ten words per line, with one blank between words. The directive has the general form

DUMP,r,iu,im,irl,ou

where

r	is the number of input records to be dumped or is zero if dumping is to continue to an end-of-file
iu	is the name or number of the input logical unit
im	is 0 for binary, 1 for ASCII, 2 for BCD, or 3 for unformatted input records
irl	is the number of words in each record of the input
ou	is the name or number of the output unit, which cannot be an RMD partition

The first line of the dump contains the record number before word 1, but subsequent lines do not have the record number.

If ASCII mode is specified by im then an ASCII scan and dump will be made in addition to the octal dump. Printable

INPUT/OUTPUT UTILITY PROGRAM

character bytes will appear to the right of each line of the octal dump. Non-printable characters will appear as ASCII blanks. ASCII scan and dump is suppressed if dump is to a TY or CT device regardless of the mode.

Example: Dump 40 binary, 50-word records from the SW logical unit onto the LO unit.

```
DUMP, 40, SW, 0, 50, LO
```

10.2.6 PRNTF (Print File) Directive

This directive prints the specified number of files from the indicated input logical unit to the list output logical unit(s) specified. The directive has the general form

```
PRNTF,f,iu,ou(1),ou(2),...ou(n)
```

where

f is the number of files to be printed
iu is the name or number of the input logical unit

each **ou(n)** is the name or number of a list output logical unit

If an RMD is specified as the input logical unit, it must have been previously positioned with a PFILE directive (section 10.2.9) and only one file may be printed at a time (i.e., if it is greater than 1, it is defaulted to 1), because the end-of-file terminates printing.

This directive is designed to print list output files directed to devices other than a line printer (i.e., magnetic tape or disc). Therefore, the input file is read in ASCII mode (1), 132 characters, and the list output records are written also in ASCII mode.

Example: Print two (2) files on magnetic tape unit 18 on LO.

```
/IOUTIL  
REW, 18  
PRNTF, 2, 18, LO  
/ENDJOB
```

Example: Print an RMD file called SYSOUT in logical unit 25 to LO.

```
/IOUTIL  
PFILE, 25, , 120, SYSOUT  
PRNTF, 1, PI, LO  
/ENDJOB
```

10.2.7 WEOF (Write End of File) Directive

This directive writes an end-of-file mark on each logical unit specified. The directive has the general form

```
WEOF,lun,lun,...,lun
```

where each **lun** is the name or number of a logical unit upon which an end-of-file mark is to be written.

Example: Write an end-of-file mark on the BO logical unit and on the PO logical unit.

```
WEOF, BO, PO
```

10.2.8 REW (Rewind) Directive

This directive, which applies only to magnetic-tape units, causes the specified logical unit(s) to rewind to the beginning of tape. The directive has the general form

```
REW,lun,lun,...,lun
```

where each **lun** is the name or number of a logical unit to be rewound.

Example: Rewind the BI and PO logical units.

```
REW, BI, PO
```

10.2.9 PFILE (Position File) Directive

This directive, which applies only to rotating-memory devices, causes the specified logical unit to move to the beginning of the designated file, and opens the file. The directive has the general form

```
PFILE,lun,key,rec1,name
```

where

lun is the name or number of the affected logical unit

key is the protection code required to address **lun**

rec1 is the number of words in each record of the file

name is the name of the file to which the logical unit is to be positioned

Since IOUTIL has only six FCBs, there can be a maximum of six files open at any given time.

Example: Position the PI logical unit, using protection code Z, to the beginning of the file FILEXY, which contains 60-word records.

`PFILE, PI, Z, 60, FILEXY`

10.2.10 CFILE (Close File) Directive

This directive, which applies only to RMD partitions, closes the specified file. The directive has the general form

`CFILE,lun,key,name,add`

where

- lun** is the name or number of the logical unit containing the file to be closed
- key** is the protection code required to address **lun**
- name** is the name of the file to be closed
- add** is 0 (default value) if the current end-of-file address on the RMD file-directory is to remain unchanged, or 1 if it is to be replaced by the current record (i.e., actual) address

A PFILE directive (section 10.2.9) must have been used to position **lun** before the CFILE directive is issued. Closing a file frees the associated FCB for use with another file. Since IOUTIL has only six FCBs, there can be a maximum of six files open at any given time.

Example: Close the file WORK on the SW logical unit (protection code B) and update the file directory.

`CFILE, SW, B, WORK, 1`

10.2.11 PACKB (Pack Binary) Directive

This directive copies the specified number of files from the indicated input logical unit to the given output logical unit(s). It causes each new system binary program to start on a record boundary. The directive has the general form

`PACKB,f,iu,im,irl,ou(1),om,orl,ou(2),...ou(n)`

where

- f** is the number of input files to be copied
- iu** is the name or number of the input logical unit.

im is 0 for binary, 1 for ASCII, 2 for BCD, or 3 for unformatted input files.

irl is the number of words in each record of the input files. If a value of zero is specified then the record length is set to the maximum buffer size. Following the read the actual physical record length (word 5 of the RQBLK) is used as the input record length.

ou(n) is the name or number of an output logical unit.

om is 0 for binary, 1 for ASCII, 2 for BCD, or 3 for unformatted output files.

orl is the number of words in each record of the output files. If a value of zero is specified then the output record length is equal to the input record length.

The following relation holds for input/output record lengths:

Input RCL	Output RCL	Output Format
fixed	fixed	As defined (blocked or unblocked)
random (0)	fixed	As defined (blocked or unblocked)
fixed	random (0)	Unblocked only
random (0)	random (0)	Unblocked only

Any RMD used in this directive must have been previously positioned with a PFILE directive (section 10.2.9).

This directive can be used for any output media and any record length. It is primarily intended to be used for RMD output of 120 words. Use with non-RMD output may not produce the intended effect.

Example: Pack one binary file from the card reader onto a RMD file on logical unit 25 in 120 word blocks:

`PACKB, 1, CR, 0, 60, 25, 0, 120`

10.3 MULTI-VOLUME TAPE HANDLING (V\$RSW)

IOUTIL provides the operator with interfaces necessary for handling multi volume (i.e., multi-reel), magnetic tape files. The routine directs the operator to unload the current magnetic tape volume and mount a new one whenever end-of-tape is encountered.

INPUT/OUTPUT UTILITY PROGRAM

The magnetic tape unit to be unloaded is given a rewind directive and the following message is output to the operator:

```
IOUTIL: UNLOAD LUN nn  
IOUTIL: MOUNT NEXT VOLUME
```

where

nn is the logical unit number of the magnetic tape to unload

After the message for mounting a new magnetic tape has been output to the operator, the subroutine issues a suspend request. When the new volume has been successfully mounted, the operator can continue execution by keying in the following:

```
;RESUME, IOUTIL
```

If the mounting of a new magnetic tape volume is not needed, the operator will key in the message ;ABORT, IOUTIL on the OC device, which will return control to JCP.

SECTION 11

VSORT (SORT/MERGE)

The VORTEX Sort/Merge (VSORT) task constructs a sorted file in the order determined by fields selected by the user.

11.1 ORGANIZATION

VSORT is scheduled as a background task by the Job-Control Processor (JCP, section 4.2.19) upon input of the JCP directive

/LOAD, VSORT

Once activated, VSORT inputs the sort parameters from the SI logical unit. The maximum number of VSORT directives is five records. The directive ENDSORT terminates the input of VSORT directives within five records. Upon completion of the sort/merge, VSORT exits to JCP.

VSORT has a buffer area large enough for most sort/merge operations. To increase the size of the buffer, input a /MEM directive (see section 4.2.3) immediately preceding the /LOAD,VSORT directive.

Inputs to VSORT comprise

- a. VSORT directives (section 11.2) input through the SI logical unit
- b. File to be sorted, input through the INPUT logical unit
- c. Additional file to be sorted, input through the ALTIN logical unit.

Outputs from VSORT comprise

- a. Sorted file on the OUTPUT logical unit
- b. Listing of VSORT directives on the LO logical unit
- c. Listing of VSORT totals for the sort/merge on the LO logical unit
- d. Error messages, if any, on the LO logical unit

Error messages applicable to VSORT are given in Appendix A.11.

VSORT performs either a full-record sort or a tag sort. In a full-record sort the entire records are moved in central memory in order to accomplish the sort. In a tag sort, only the concatenated sorting control fields and the record numbers are manipulated in central memory. VSORT will perform the more efficient tag sort unless one of the following conditions occurs:

- a. INPUT file is not an RMD
- b. The file used for INPUT is also used for another file in the sort, either as a WORK or OUTPUT file
- c. A user input exit routine is specified (by the INEXIT directive)
- d. An alternate input has been defined

Workspace Requirements: Each work file must be large enough to contain a number of work records equal to the number of input records. For tag sorts, the length of the work records is equal to the sum of the length of the control fields plus one word. On full-record sorts, the sum of the control fields plus one input record length is needed.

Work records are blocked with a blocksize equal to a fourth or third of the central memory workspace for the merge phase.

Work space for the sort phase in central memory is allocated dynamically to overlay the initialization routine (about 2K), which occupies the highest memory locations of VSORT. Work space for the merge phase occupies an additional 1K in central memory. Additional work space may be allocated for a background sort by using the /MEM directive (JCP, 4.2.3).

11.2 VSORT DIRECTIVES

This section describes the VSORT directives.

a. Required Group

- | | |
|-----------|--------------------------------|
| • SORT | Sort directives follow |
| • INPUT | Define logical unit for input |
| • OUTPUT | Define logical unit for output |
| • WORK | Define work file(s) |
| • SORTKEY | Define sorting field(s) |
| • ENDSORT | Begin sorting |

b. Optional Group

- | | |
|----------|--|
| • ALTIN | Describe alternate sort input file |
| • ALTSEQ | Define collating sequence |
| • INCLF | Include this record by field comparison |
| • INCLC | Include this record by constant comparison |

VSORT (SORT/MERGE)

•	OMITF	Omit this record by field comparison	key	is the single character file protection key, as contained in the file directory for the file (required only if the filename is present and the RMD is protected)
•	OMITC	Omit this record by constant comparison		
•	MOVEF	Move Field		
•	MOVEC	Move Constant		
•	LOT	Output sorted tags (Record Numbers)	recordlength	is a 1- to 4-digit decimal number specifying the length in words of the logical records in the file.
•	INEXIT	Use input preprocessor		
•	OUTEXIT	Use output preprocessor	blocking-factor	is a 1-4 character decimal number specifying the number of logical records per physical record.

The general form of a VSORT directive is

name = p(1),p(2),...,p(n) terminator

where

- name** is one of the VSORT directives
- p(n)** is a parameter required by VSORT and defined below under the descriptions of the individual directives
- terminator** is a blank or right parenthesis

Example: Describe a sort input file on magnetic tape on logical unit 18, which has 200-word records.

INPUT=(18 , , , 200)

11.2.3 ALTIN Directive

This optional directive describes an alternative input which contains additional records to be stored. It has the form

ALTIN = (lun)

where

lun is a 1-3 character decimal number specifying the logical number of the file (cannot be an RMD).

11.2.1 SORT Directive

This directive signals the start of the sort directives. The general form is

SORT

The word SORT must be followed by at least one blank. The SORT directive must be the first directive on the first control record.

11.2.2 INPUT Directive

This directive describes the sort input file which contains the records to be sorted. It has the general form

INPUT = (lun,filename,key,recordlength,blocking factor)

where

- lun** is a 1- to 3-character decimal number specifying the logical unit of the file
- filename** is a 1- to 6-character name of the file as it exists on the RMD file directory (required for all RMD files)

Note: In this directive, the record length and blocking factor defaults to the input file read (refer to the INPUT directive).

Example: Describe an alternate input file on logical unit 13.

ALTIN = (13)

Handwritten notes:
 Input should be the same
 Output should be the same
 13. be the same

11.2.4 OUTPUT Directive

This directive describes the output file which will ultimately contain the sorted records. It has the general form

OUTPUT = (lun,filename,key,recordlength,blocking factor)

where lun, filename, key recordlength and blocking factor are the same as they are described in the INPUT directive (section 11.2.2).

Example: Describe a sort output file on a line printer logical unit 5, which has a 60-word record length.

OUTPUT=(5 , , 60)

Note: The record length does not include the printer line control character which is added by SORT.

11.2.5 WORK1-3 Directives

$$\text{WORK} \left\{ \begin{array}{l} 1 \\ 2 \\ 3 \end{array} \right\} = (\text{lun}, \text{filename}, \text{key})$$

where *lun*, *filename*, and *key* are the same as described for the INPUT directive (section 11.2.2).

Example: Describe intermediate sort files named W1, W2, and W3 on RMD logical unit 25. These files do not have protection keys.

WORK1=(25 , W1) , WORK2=(25 , W2) , WORK3=(25 , W3)

Note: The work files must be RMD files. Each file must have sufficient space to contain the intermediate work records equal to the number of records in the input file for the sort.

11.2.6 SORTKEY Directive

This directive describes one to 29 control fields to be used to sequence the records of the sort input file. It has the general form

SORTKEY = (sc(1),ec(1),order(1),...,sc(29),ec(29),order(6))

where each

- sc(n)** is a one- to four-digit decimal number specifying the starting character (or byte) position of the control field as it exists in the input record, or, if there positions are modified by an INEXIT routine, as they exist in the modified input record.
- ec(n)** is a one- to four-digit decimal number specifying the ending character (or byte) position of the control field. It must be greater than or equal to the preceding starting character position
- order(n)** is a single character A or D for ascending or descending sequence, respectively, for sorting the control field

At least one control field specification must be given. Each control field specification must have all three parameters specified.

Control fields may overlap.

Character positions are numbered starting with one.

The significance of a control field depends on its placement in the SORTKEY directive. The first control field defined is the most important (or major) control field. The next is the secondary (used in cases of matches in the first) control field. Similarly, until the last specification given is the least important.

Collating sequence: An algebraic collating sequence is used to sort the data. Each word (in numeric data) or each byte (in character data) is interpreted as an octal number having an algebraic sign. Thus, ASCII characters have the collating sequence from 0240 (low) to 0337 (high). If characters are other than ASCII, the sign bit (bit 7) of each 8-bit character must be the same for all the characters.

Word-boundary data are treated as signed octal numbers and have the collating sequence from 0100000 (low) to 077777 (high). Thus, FORTRAN variables of integer, real, complex or logical types may be sorted with SORT control fields. FORTRAN double-precision numbers cannot be sorted because the sign of the number is not in the first word.

Example: Describe two control fields, one is bytes 27 and 28 in ascending order, and the other is byte 1 through 4 to be sorted in descending order.

SORTKEY=(27 , 28 , A , 1 , 4 , D)

11.2.7 ALTSEQ Directive

This optional directive allows specification of alternate values from existing characters to be found in the file to be sorted, thus providing a direct means of specifying an alternate value for each character of concern when sorting. It has the form

ALTSEQ = (x)

where

x is a 64-character string

Example: Redefine the collating sequence where the vowels (a,e,i,o, and u) will be replaced with blanks for sorting purposes.

ALTSEQ = (! " # \$ % & ' () * , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? B C D F G H J K L M N P Q R S T V W X Y Z [\] ^ _ ` ,

11.2.8 INCLF Directive

This optional directive provides for including an input record by comparing one ASCII field against another ASCII field. It has the form

VSORT (SORT/MERGE)

INCLE = (fs1, fe1, rel, fs2)

where

- fs1** is the first byte of the first field.
- fe1** is the last byte of the first field.
- rel** is one of the following relation tests (required to include this record):

Relation Test	Include if
EQ	fs1 = fs2
NE	fs1 ≠ fs2
LT	fs1 < fs2
GT	fs1 > fs2
LE	fs1 ≤ fs2
GE	fs1 ≥ fs2

- fs2** is the first byte of the second field.

Example: Accept any input record where byte 11 and 12 are equal to bytes 26 and 27.

INCLF = (11, 12, EQ, 26)

11.2.9 INCLC Directive

This optional directive provides for including an input record by comparing an ASCII Field against an ASCII constant. It has the form

INCLC = (fs1, fe1, rel, con)

where

- fs1** is the first byte of the field.
- fe1** is the last byte of the field
- rel** is one of the following relation tests (required to include this record):

Relation Test	Include if
EQ	fs1 = con
NE	fs1 ≠ con
LT	fs1 < con
GT	fs1 > con

LE fs1 ≤ con
GE fs1 ≥ con

con is a constant of up to ten characters excluding comma and right parenthesis.

Example: Accept any record where byte 46 is not a 7.

INCLC = (46, 46, NE, 7)

11.2.10 OMITF Directive

This optional directive provides for rejecting an input record by comparing one ASCII field against another ASCII field. It has the form

OMITE = (fs1, fe1, rel, fs2)

where

- fs1** is the first byte of the first field.
- fe1** is the last byte of the first field.
- rel** is one of the following relation tests (required to reject this record):

Relation Test	Reject if
EQ	fs1 = fs2
NE	fs1 ≠ fs2
LT	fs1 < fs2
GT	fs1 > fs2
LE	fs1 ≤ fs2
GE	fs1 ≥ fs2

- fs2** is the first byte of the second field.

Example: Delete any input record where bytes 1 and 2 are equal to bytes 6 and 7.

OMITF = (1, 2, EQ, 6)

11.2.11 OMITC Directive

This optional directive provides for rejecting an input record by comparing an ASCII field against an ASCII constant. It has the form

OMITC (fs1, fe1, rel, con)

where

- fs1** is the first byte of the field.
- fe1** is the last byte of the field.
- rel** is one of the following relation tests (required to reject this record):

Relation Test	Reject if
EQ	fs1 = con
NE	fs1 ≠ con
LT	fs1 < con
GT	fs1 > con
LE	fs1 ≤ con
GE	fs1 ≥ con

- con** is a constant of up to ten characters excluding comma and right parenthesis.

Example: Delete any record where byte 26 is not a 2.

OMITC = (26 , 26 , NE , 2)

11.2.12 MOVEF Directive

This optional directive provides for moving a data field within a record to another field within the record. It has the form

MOVEF = (fs1, fs2, fe2)

where

- fs1** is the first byte of the from field.
- fs2** is the first byte of the to field.
- fe2** is the last byte of the to field.

Example: Move the data in bytes 30 through 40 to bytes 70 through 80.

MOVEF = (30 , 70 , 80)

11.2.13 MOVEC Directive

This optional directive provides for moving an ASCII constant to a field within the record. It has the form

MOVEC = (con, fs1, fe1)

where

- con** is a constant of up to ten characters, excluding comma and parenthesis.
- fs1** is the first byte of to field.
- fe1** is the last byte of the to field.

Example: Move the constant SORTED to bytes 13 through 18.

MOVEC = (SORTED , 13 , 18)

11.2.14 LOT Directive

This optional directive specifies that a list of sorted tags is to be output. It has the form

LOT

The directive is applicable when a tag sort is performed.

11.2.15 INEXIT Directive

This optional directive specifies whether a user-written input-exit routine is to be called at the time the input file is being read by the sort part of VSORT. The general form of the directive is

**INEXIT = { YES }
 { NO }**

The equal sign may be followed by a string of up to four alphabetic characters. Unless YES is specified, the default is NO (a user routine is not called). YES or NO must be followed by at least one blank.

11.2.16 OUTEXIT Directive

This optional directive specifies whether a user-written output exit routine is to be called at the time the final file output file is being created by the merge phase of VSORT. It has the general form

**OUTEXIT = { YES }
 { NO }**

VSORT (SORT/MERGE)

The meaning of YES and NO is the same as described for the INEXIT directive (section 11.2.15).

11.2.17 ENDSORT Directive

This directive signals the end of the sort directives. The general form of this directive is

ENDSORT

The word ENDSORT must be followed by at least one blank as the last directive on the last control record for VSORT.

11.3 USER EXITS

User exits provide for the insertion, deletion, or modification of input and output records by user-written routines. Exits are requested by the VSORT directives, INEXIT = YES and/or OUTEXIT = YES. The exit routines written by the user are added to VSORT at load-module generation time.

The input exit routine, if provided, is called for each input record before it enters the sort. Possible uses of the input exit are

- Add input records
- Delete input records
- Create part or all of the input file
- Change input records, such as control fields

The input record length may be changed to the output record length specified on the OUTPUT directive.

The output exit routine, if provided, is called for each output record before it is written on the output file. Possible uses for the output exit are

- Add output records, effectively merging one or more files with the sorted file
- Delete sorted output records, such as duplicates
- Change the sorted output records

If output records are added or changed, it's the user's responsibility to ensure that the control fields of the output records remain in sequence.

11.3.1 Calling Sequence

VSORT uses the following calling sequence for user exits:

Word 1	JMPM XITn
Word 2	input buffer address
Word 3	output buffer address
Word 4	flag

where

n is 1 for input exit and 2 for output exit

input buffer address is the address of input record passed to the user routine (INEXIT) or the address to which the user must move a record if it is to be inserted before the output record (or EOF) passed to the user routine (OUTEXIT)

output buffer address is the address of the output record passed to the user routine (OUTEXIT) or the address to which the user must move a record if it is to be inserted before the input record (or EOF) passed to the user routine (INEXIT)

flag is set by VSORT as 0 for an EOF encountered, 1 for INEXIT, or 2 for OUTEXIT; otherwise it is set by the user routine as follows

Bit 0 = 1 accept input record (INEXIT) or insert record in input buffer before output record (OUTEXIT)
= 0 is ignore the record in the input buffer

Bit 1 = 1 accept the output record (OUTEXIT) or insert record in the output buffer before the input record (INEXIT)

= 0 ignore the record in output buffer

After EOF notification has been given to the user input (output) exit routine, the user routine may continue to pass records to VSORT in the buffer, but the contents of the buffer are ignored.

11.3.2 Implementation

The exit routines written by the user must have the following external names

XIT1 User input exit entry point

XIT2 User output exit entry point



SECTION 12 DATAPLOT II

DATAPLOT II is a collection of FORTRAN callable subroutines that provide the user with interface to the Varian STATOS 31 and STATOS 33 electrostatic printer/plotters.

Using DATAPLOT II, the programmer can specify the desired graphic output at the functional level. For example, DATAPLOT II enables the STATOS printer/plotter to

- Draw a vector between two given points
- Produce a scaled set of axes for a given magnitude
- Produce a plot from a set of input data, using specified plot point markers

12.1 SYSTEM FLOW OUTLINE

DATAPLOT II consists of FORTRAN and DAS MR subroutines which permit STATOS 31 or STATOS 33 printer/plotters to draw lines, numbers, letters, symbols, and chart axes. Provision is also made for plotting lines from existing X-Y arrays and/or data from an external data base.

Figure 12-1 shows the relationship between the user and the DATAPLOT II Graphics System.

12.2 HARDWARE REQUIREMENTS

DATAPLOT subroutines can be linked to either foreground or background tasks under VORTEX (see VORTEX installation manual for memory requirements). DATAPLOT can be used with the following considerations:

The STATOS equipment that is supported under VORTEX is

Unit	Model	Width
STATOS 31	70-6602	14-7/8 inches
STATOS 31	70-6608	11 inches
STATOS 33	70-6611/21	8-1/2 inches
STATOS 33	70-6613/23	11 inches
STATOS 33	70-6615/25	14-7/8 inches
STATOS 33	70-6617/27	22 inches
STATOS 42	70-6661	8-1/2 inches
STATOS 42	70-6662	8-1/2 inches
STATOS 42	70-6663	11 inches
STATOS 42	70-6664	11 inches
STATOS 42	70-6665	14-7/8 inches
STATOS 42	70-6666	14-7/8 inches
STATOS 42	70-6667	22 inches
STATOS 42	70-6668	22 inches

Unit	Model	Width
STATOS 41	70-6651	8-1/2 inches
STATOS 41	70-6652	8-1/2 inches
STATOS 41	70-6653	11 inches
STATOS 41	70-6654	11 inches
STATOS 41	70-6655	14-7/8 inches
STATOS 41	70-6656	14-7/8 inches
STATOS 41	70-6657	22 inches
STATOS 41	70-6658	22 inches

- b. The STATOS unit must be operated under BIC control with PIM assigned interrupts. In addition, the STATOS 31 printer/plotters will support the single-line Input Buffer Option (Model 31-152); except, those without a hardware character generator.
- c. DATAPLOT II does not support any of the Hardware Character Generator options, the Simultaneous Print/Plot options, or the High Speed option.

12.3 GENERAL DESCRIPTION

12.3.1 DATAPLOT II Organization

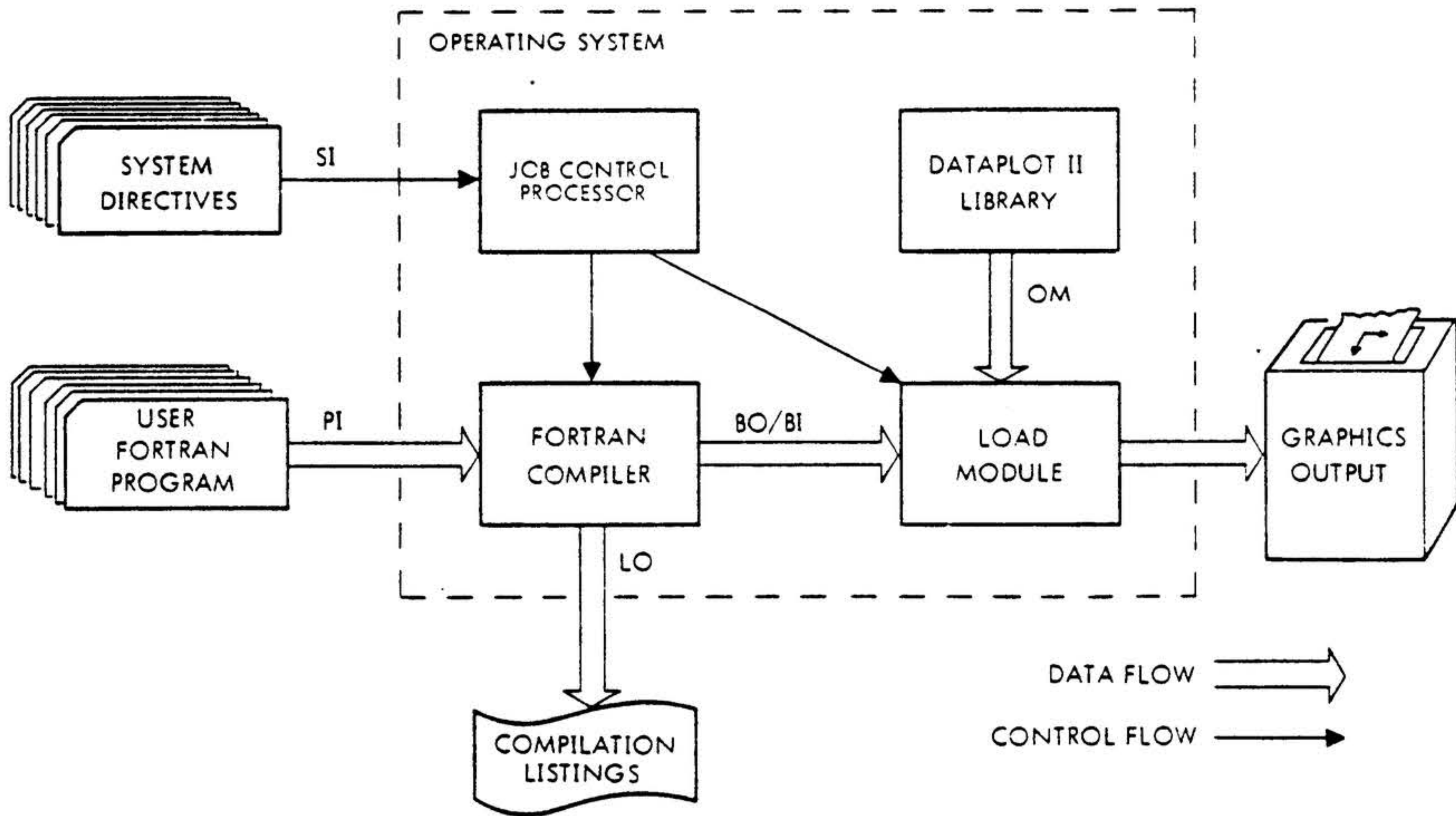
DATAPLOT II is organized into the following five logical operations:

- Defining the Plot File and Initialization
- Building the Plot File
- Sorting the Plot File
- STATOS Paper Control
- Outputting the Plot File in STATOS Raster Format

These are shown schematically in figure 12-2.

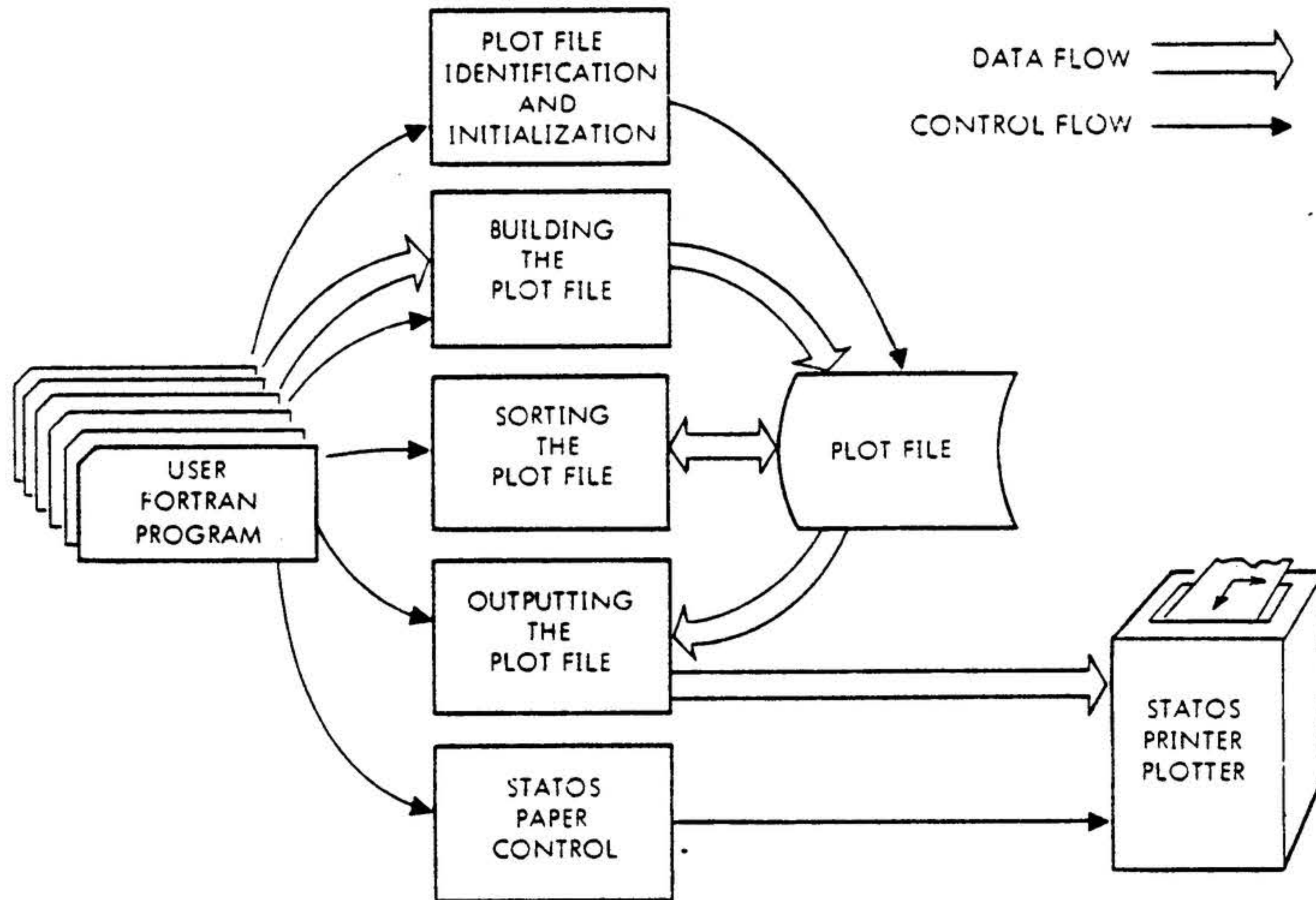
Defining the Plot File: Subroutine DPINIT defines which VORTEX logical unit will contain the Plot File, the logical size of the plot file records, and the block size of the output device for the plot data. If DPINIT is not called, the plot file will default to System Scratch (SS) with 120-word records, and plot data will be output in blocks of 88 words for the 14-7/8 inch STATOS. Subroutine DPINIT must be called when Dataplot is operating in a foreground mode to prevent a possible conflict with background programs which may use System Scratch.

DATAPLOT II



VTII-3225

Figure 12-1. DATAPLOT II Graphics System Data Flow



VTII-3224

Figure 12-2. DATAPLOT II Organization

Building the Plot File: If the plot file is to be built through calls to Dataplot subroutines ORIG, CHAR, PLOT, VECT, NUMBER, SCALE, AXIS, DATA, SYMBOL, APPEND, and LINE, the plot file must be assigned to an RMD device or the sort subroutine will not work.

STATOS Paper Control: Subroutine CUT, ENDCUT, and TOPFRM are auxiliary paper control subroutines. These subroutines issue FUNC commands to the output driver and will be processed as applicable to the driver.

Outputting the Plot File: Subroutine DPLOT outputs STATOS raster format data. DPLOT is called by subroutine PLOT when the plot is terminated.

12.3.2 System Considerations

DATAPLOT II is supplied as three groups of object module routines. The first group is the basic Dataplot Object Module (BDPOM). It contains the subroutines for initializing the plot file, drawing lines, sorting and outputting the plot file, and paper control. The second group is the VORTEX (pen-plotter compatible) Dataplot Object Module (VDPOM). It contains higher level routines for building the plot file. The third group is the MOS (compatible) Dataplot Object Module (MDPOM). It contains calls which are compatible to the MOS Dataplot II.

DATAPLOT II is put onto the object module library as a combination of either the BDPOM and VDPOM, or the BDPOM and MDPOM, depending on which set of higher level subroutines the user wishes to call. The VDPOM routines offer axes, character and number strings at any angle, while the MDPOM offers only two angles (0 degrees and 90 degrees). The MDPOM subroutines are provided for users who have already written MOS programs calling DATAPLOT II.

The MDPOM routines may be placed on an alternate object module library and the VDPOM routines may be placed on the standard OM library. Programs using the MDPOM routines may search the alternate library before the standard OM library, but this also prevents a load-and-go operation.

When converting programs written for MOS DATAPLOT II, a call to PLOTS must be substituted for the calls to OPEN, HOPEN, and DOPEN. The call CALL PLOT (0.0.0.0,999) must be substituted for calls to CLOSE, HCLOSE, and DCLOSE. There is a shift in the logical plot origin if the pseudo-pen encounters a plot boundary in VORTEX DATAPLOT II (incl MDPOM). There is no such shift in the MOS DATAPLOT II routines.

Users of STATOS 42 models 70-6661 through 70-6668 (200 stylii/inch) must include CALL DENSTY(200) prior to the call to PLOTS. Omission of the call to DENSTY for 200 stylii/inch models will result in a plot size one-half that expected from the program.

DATAPLOT II subroutines are listed below:

Dataplot II initialization

DPINIT	BDPOM	
PLOTS	BDPOM	
DENSTY	BDPOM	(STATOS 42 models)

Building the Plot File

PLOT	BDPOM
VECT	BDPOM
ORIG	BDPOM
FACTOR	BDPOM
WHERE	BDPOM
MLTPLE	BDPOM
APPEND	BDPOM
NUMBER	MDPOM
NUMBER	VDPOM
SCALE	MDPOM
SCALE	VDPOM
AXIS	MDPOM
AXIS	VDPOM
DATA	MDPOM
LINE	VDPOM
SYMBOL	MDPOM
SYMBOL	VDPOM
CHAR	MDPOM

Sort and Output

DPSORT	BDPOM
DPLOT	BDPOM

Paper Control

TOPFRM	BDPOM
CUT	BDPOM
ENDCUT	BDPOM

12.3.3 VORTEX Considerations

Plot File Assignment: The user must supply a secondary storage file sufficiently large enough to hold the plot file when the plot file is unsorted or generated by calls to DATAPLOT II subroutines ORIG, VECT, CHAR, NUMBER, SCALE, DATA, AXIS, LINE, PLOT, SYMBOL, or APPEND. Four 16-bit words are used for each vector or character to be plotted, and four 16-bit words are used for the end-of-plot indicator. An error (DP00) will be reported if the plot file is overflowed.

DATAPLOT II

The user may supply a sorted plot file in vector-end-point format. Sorted data may be plotted directly from the plot file by assigning the plot file to the logical unit containing the data during the call to DPINIT.

User-Supplied Central Memory Buffers: DATAPLOT II may use up to three types of buffers which the user must supply by a FORTRAN DIMENSION statement. The buffer types are:

- DATAPLOT II Working Buffer -- defined in call to PLOTS
- Append FILE I/O Buffer -- defined in call to APPEND
- Data Array Buffer(s) -- used by DATA and SCALE subroutines

DATAPLOT II Working Buffer: The DATAPLOT II Working Buffer is used in building, sorting, and outputting the plot file.

The algorithm for determining the size of the DATAPLOT II working buffer is:

$$22 + PFIO + RO + 6(VEC_{max})$$

where

- PFIO** is the size of the plot file I/O buffer
- RO** is the size of the raster (STATOS) output buffer
- VEC_{max}** is the maximum number of vectors or characters on any one STATOS scan line

The plot file I/O buffer size is a multiple of the physical record length of the plot file, and is specified in the call to DPINIT.

The raster output buffer size is determined by the width of the STATOS printer/plotter for which the plot is intended, as shown in the following table, and is specified in the call to DPINIT.

STATOS Model	Width	No. Styli/Line	Raster Buffer Size
70-6608	11 inches	1056	66
70-6602	14-7/8 inches	1408	88
70-6611, 70-6621, 70-6651, and 70-6652	8-1/2 inches	800	50
70-6613, 70-6623, 70-6653, and 70-6654	11 inches	1056	66
70-6615, 70-6625, 70-6655, and 70-6656	14-7/8 inches	1408	88

STATOS Model	Width	No. Styli/Line	Raster Buffer Size
70-6617, 70-6629, 70-6657, and 70-6658	22 inches	2048	132
70-6661 and 70-6662	8-1/2 inches	1600	100
70-6663 and 70-6664	11 inches	2112	132
70-6665 and 70-6666	14-7/8 inches	2816	176
70-6667 and 70-6668	22 inches	4096	264

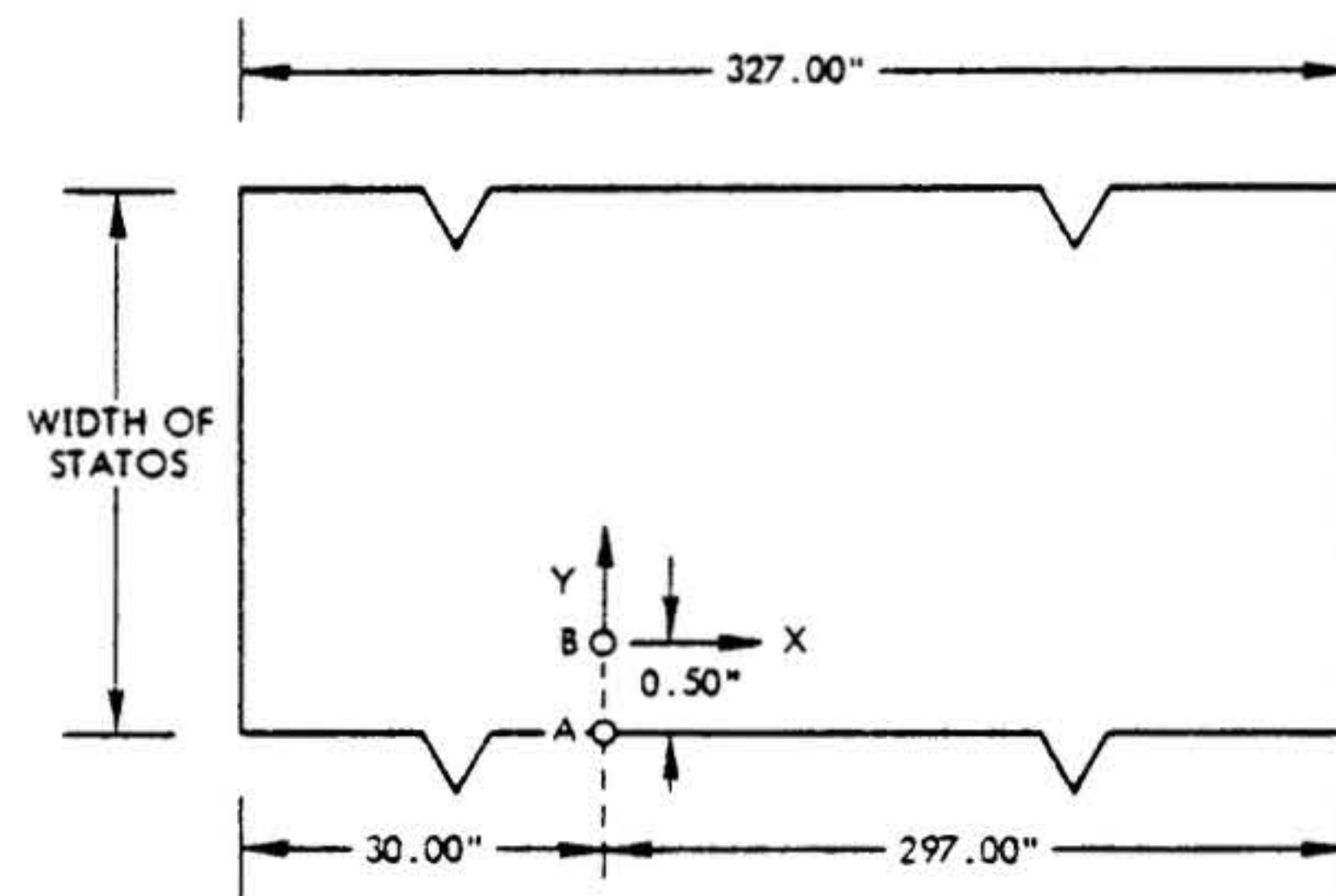
The buffer is also used to hold vectors and characters at the time they are being converted to STATOS raster format. A six-word entry will be placed in this buffer when the vector or character is first to appear on a STATOS scan line. The entry remains until the vector or character reaches its last STATOS scan line.

An error (DP01) will be reported if the concurrent vector buffer is overflowed.

Example: DATAPLOT II is going to plot from a plot file whose record length is 120, to a STATOS printer/plotter whose width is 14-7/8 inches. The maximum number of vectors or characters expected on any one raster line is 130. The length of the working buffer should be:

$$22 + 120 + 88 + 780 = 1010$$

Minimum and Maximum Plot Values: The minimum x value is -30.00 inches. The maximum x value is +297.00 inches. The maximum y value is determined by the width of the STATOS for which the plot is intended. These values are shown in figure 12-3.



A = Physical origin (0.0,0.0)

B = Starting logical origin (0.0,0.0) or (0.0,0.5) physical.

VTII-3083

Figure 12-3. Minimum and Maximum Plot Values

The logical origin may be moved by calling subroutine PLOT or ORIG. Subroutine PLOT will move the logical origin referenced to the last logical origin. Subroutine ORIG will move the logical origin referenced to the physical origin.

If the plot boundaries are encountered while building the plot file, the logical origin will be effectively shifted in a manner similar to a pen plotter. An error (DP04) will be reported.

12.4 DATAPLOT II SUBROUTINES

The general form of the DATAPLOT II functions is:

(statement number) CALL S (p(1),p(2),...p(n))

where:

(statement number) is the optional statement number.

S is the name of the subroutine.

p(1),...p(n) are the parameters, if any.

12.4.1 DPINIT (System File Initialization)

This function enables the user to specify certain initial conditions relating to the plot file and plot file I/O buffer. In the absence of this function, the default parameter values shown in the parameter description will exist.

The function has the general form

CALL DPINIT (lun,key,name,ipltbf,outsiz)*

*BDPOM

where

		Default	Error Conditions:
lun	is the number or variable of the plot file logical unit (Integer).	8	Condition: Work buffer size is too small Action: Incomplete Plot Message: DP01
key	is the protection key, if any.	None	Conditions: PLOTS not called Action: Abort Plot Message: DP05
name	RMD: is the six-character name of the plot file. It may be given as an array name or a Hollerith constant non-RMD: Not used.	SS (background scratch file)	
ipltbf	is the length, of the plot file I/O buffer. (Integer)	120	
outsiz	is the block size of the output plot data as given in section 12.3.3 (Integer).	88	

Error Conditions: None

Example: Select logical unit 25, file name PLTFIL, protection key Z, length 120 as the plot file. The output is to go to a STATOS, width 14-7/8 inches.

CALL DPINIT (25,2HbZ,6HPLTFIL,120,88)

12.4.2 PLOTS (Work Buffer Initialization)

The PLOTS function is used to initialize the DATAPLOT II work buffer. It must be called prior to any calls to the PLOT subroutine and prior to calls to higher level plot subroutines. STATOS models 70-6661 through 70-6668 are 200 stylii/inch models and hence require CALL DENSTY(200) prior to the call to PLOTS.

The function has the general form

CALL PLOTS (ibuf,nloc,lun)*

*BDPOM

where

ibuf is the name of the user-supplied storage area to be used as a work buffer by DATAPLOT II. This array should be dimensioned by the user in his FORTRAN program.

nloc is the number which identifies the size of the work buffer (ibuf). It will normally be the same number used in the DIMENSION statement. The size is determined by the algorithm supplied in section 12.3.3 (Integer).

lun is the logical unit number of the output device (Integer).

Error Conditions:

Condition: Work buffer size is too small
Action: Incomplete Plot
Message: DP01

Conditions: PLOTS not called
Action: Abort Plot
Message: DP05

Example:

DIMENSION IBUF (1500)

CALL PLOTS (IBUF,1500,5)

The above defines logical unit number 5 as the output device for the data in STATOS raster format. Buffer IBUF, of length 1500 words, will be used as a central memory work area by DATAPLOT II.

12.4.3 PLOT (Generate Plot)

The PLOT function is basic to the generation of graphic output. It may be used to draw lines between points, define new plot origins, sort plot data, cause the transfer of plot information to the output device and terminate plot generation.

The function has the general form

BDPOM** **CALL PLOT (x,y, ±idraw)

where

x,y are the x and y coordinates, in inches from the currently defined origin (Real).

± draw is an integer which determines whether or not a line is drawn from the "current" x,y coordinates to the coordinates defined in the call. It may also be used to define a new plot origin or to terminate the plot generation process and cause transfer of plot information to the output device.

If IDRAW = 2, a line is drawn from the current x,y coordinates to the coordinates defined in the call. The new coordinates then become the current x,y coordinates.

If IDRAW = 3, the coordinates in the call become the current x,y coordinates, but no line is drawn.

If IDRAW = -2 or -3, a new origin is defined at the call coordinates and the operation is completed as if IDRAW were positive. The current x and y coordinates are set to zero with respect to the new origin. If no call has been made to MLTPLE, or if the last call to MLTPLE was made with IND = 0, the current plot will be terminated and subsequent plotting will be defined with reference to a new origin on the paper. If the last call to MLTPLE was made with IND = 1, a redefinition of the origin will occur and subsequent plot definitions will be treated as belonging to the current plot.

If IDRAW = 999, the plot generation process will be terminated and all accumulated plot information will be transferred to the output device. Further calls to PLOT are not processed.

Error Conditions:

The normal pen plotter routines do not keep track of the actual location of the pen, but instead always assume that

the pen can be moved from the current location to the new location and that enough commands are output to accomplish this. If a mechanical stop is encountered during plotting, the motion in that direction is simply inhibited by the plotter. Because the mechanical stops are not precise, errors will be produced if a mechanical stop is encountered during plotting. However, this is sometimes done before initiating a plot in order to position the pen in a known location before beginning the actual plot.

DATAPLOT II routines have software stops contained internally and attempt to produce the same effect as a mechanical stop. If a plot boundary is encountered, an error (DP04) will be reported, the line will extend toward the boundary and follow the boundary to the final position, and the origin will be effectively shifted in a manner similar to the pen plotter.

Examples:

```
CALL PLOT ( 1.0, 2.0, 3)
CALL PLOT ( 2.0, 2.0, 2)
```

The above calls will draw a line between (1,2) and (2,2).

```
CALL MLTPLE ( 1)
CALL PLOT ( 1.0, 2.0, 3)
CALL PLOT ( 2.0, 3.0, -2)
CALL PLOT ( 1.0, 1.0, 2)
```

The above calls will draw a line in absolute coordinates from (1,2) to (3,4) and redefine the plot origin (0,0) to (2,3) in absolute coordinates.

12.4.4 SCALE (Generates Scale Factor)

This subroutine scales data by computing a scale factor and a displacement factor.

The subroutine has the general form

```
CALL SCALE (arr,npts,pgsz, +int)*
or
CALL SCALE (arr,pgsz,npts, ±int)**
*MDPOM
**VDPOM
```

where

arr is the name of the (real) array to be scaled.

npts is the number of points to be scaled in the array. Normally, all points are scaled (Integer).

pgsz is the size of the page (linear interval in inches) within which the data must fall. It must be greater than 1.0 inch (Real).

\pm int is the interval at which the array is to be sampled.

If INT is positive, the selected displacement approximates a minimum, and the scale factor is positive.

If INT is negative, the selected displacement approximates a maximum, and the scaling factor is negative (VORTEX call only).

The array must be dimensioned at least two elements larger than the actual number of data values it contains. The calculated displacement will be stored in ARR(NPTS + 1), and the calculated scale factor will be stored in ARR(NPTS + 2).

The subroutine scales data within the following constraints:

- The scale factor is 1., 2., 4., 5., or 8. times 10E(n).
- The displacement is an integral multiple of the scale factor.
- The displacement is .LE. the minimum value in the array.
- The displacement + the scale factor (units/inch)* axis length is .GE. the minimum value in the array.

Examples are shown in the sample programs (section 12.6).

Error Conditions: None

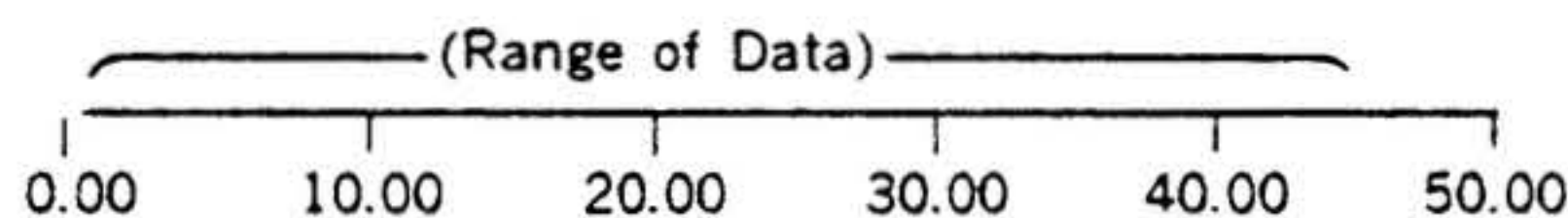
Examples:

- Given an array of 24 data values to be plotted over a 5-inch axis, assume the minimum value in the array is 1.00 and the maximum is 42.00. The statement CALL SCALE (ARR,5.0,24,+1) would give the following results:

$$\begin{aligned} \text{Units/inch} &= (42.00-1.00)/5.0 = 8.2 \\ \text{SF (scale factor)} &= 10.0 \\ \text{VLO (first value plotted)} &= 0.0 \end{aligned}$$

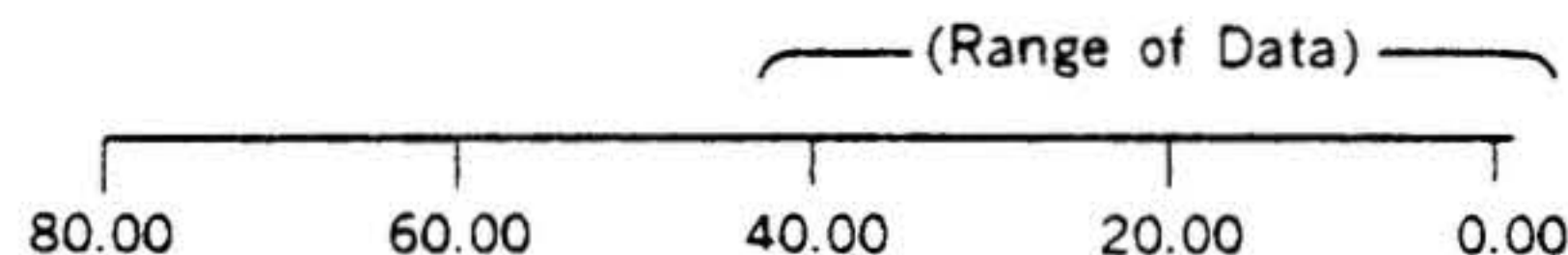
VLO value is stored in ARR(25)
SF value is stored in ARR(26)

Using these values, AXIS would draw the following axis line:



- Assume that the array of Example 1 is to be plotted on a 4 inch axis, from maximum to minimum. CALL SCALE (ARR,4.0,24,-1) would give these results:
 $\text{SF} = (1.00-42.00)/4.0 = -10.25$, which is adjusted to -20.
 Minimum multiple = 0.00; VLO = Minimum + (AXLEN * SF) = 80.00

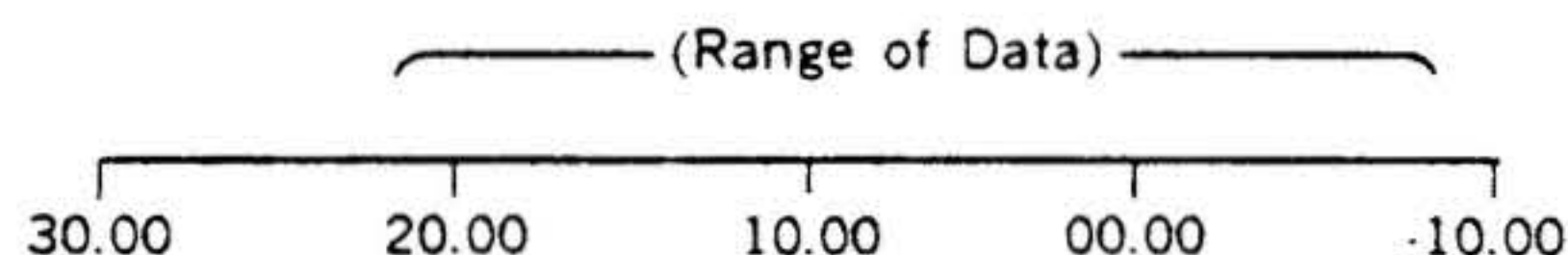
In this case the following axis would be drawn:



- Assume 100 points are to be plotted on a 4 inch axis from maximum (+22) to minimum (-9), using every other data value in the array. The DIMENSION statement should specify ARR(204), and the calling sequence is CALL SCALE (ARR, 4.0,100,-2).

Initial SF = $(-9 -22)/4 = -7.75$, adjusted to .8.
 Initial VLO = +16.00; last value on axis = -16.00.
 The axis range is inadequate for the data range, so SF is revised to the next higher interval.
 Revised SF = -10., stored in ARR(203).
 Revised VLO = 30.00, stored in ARR(201).

The resulting axis would appear as follows:



12.4.5 AXIS (Generate Segmental Axis)

Subroutine AXIS produces entries into the plot file for an axis with the markers every inch, an axis label and number labels for each tic mark, using the results of the SCALE subroutine if desired.

The subroutine is of the general form

```
CALL AXIS (x,y,axlh,idir,bcd, $\pm$ nchar,vlo,sf)*
or
CALL AXIS (x,y,bcd, $\pm$ nchar,axlh,angle,vlo,sf)**
```

*MDPOM
**VDPOM

where

- x,y** is the starting point on the page of the axis to be drawn (Real).
- axlh** is the length of the axis in inches. The value given will be truncated to the next smallest integer value (Real).
- idir** is the axis direction. Zero for x direction. Non-zero for y direction (Integer).
- bcd** is the first word address of a character string to be plotted as a label for the axis. If there is no label, use a dummy space.

\pm nchar

NCHAR is the number of letters contained in the character string to be plotted as a title (Integer).

If NCHAR < 0: the title, tic marks and interval labels will be plotted on the clockwise side of the axis.

If NCHAR \geq 0: the title, tic marks and interval labels will be plotted on the counter-clockwise side of the axis.

\pm nch

NCH is the number of letters contained in the character string to be plotted as a title (Integer).

If NCH \geq 0, the title, tic marks, and interval labels will be plotted on the clockwise side of the axis.

If NCH \geq 0, the title, tic marks, and interval labels will be plotted on the counter-clockwise side of the axis.

vlo

is the number to be plotted at the starting point of the axis (Real).

sf

is the scale factor (units/inch) to be used in labelling the 1-inch intervals. By making SF = ARR(NPTS + 2) (see SCALE routine), the axis and data will have the same scale factor (Real).

angle

is the angle at which the axis is to make with the x axis.

The interval labels will be scaled by powers of 10 if they are too large or too small to fit into two decimal place accuracy. Thus, assuming a scale factor of 1000./inch, 12000. would be printed 12.00 on the interval tic mark, but a note would be added to the axis label: "x10".

The SCALE routine should be used prior to using AXIS if SF = ARR(NPTS + 2).

Error Conditions: None

Example:

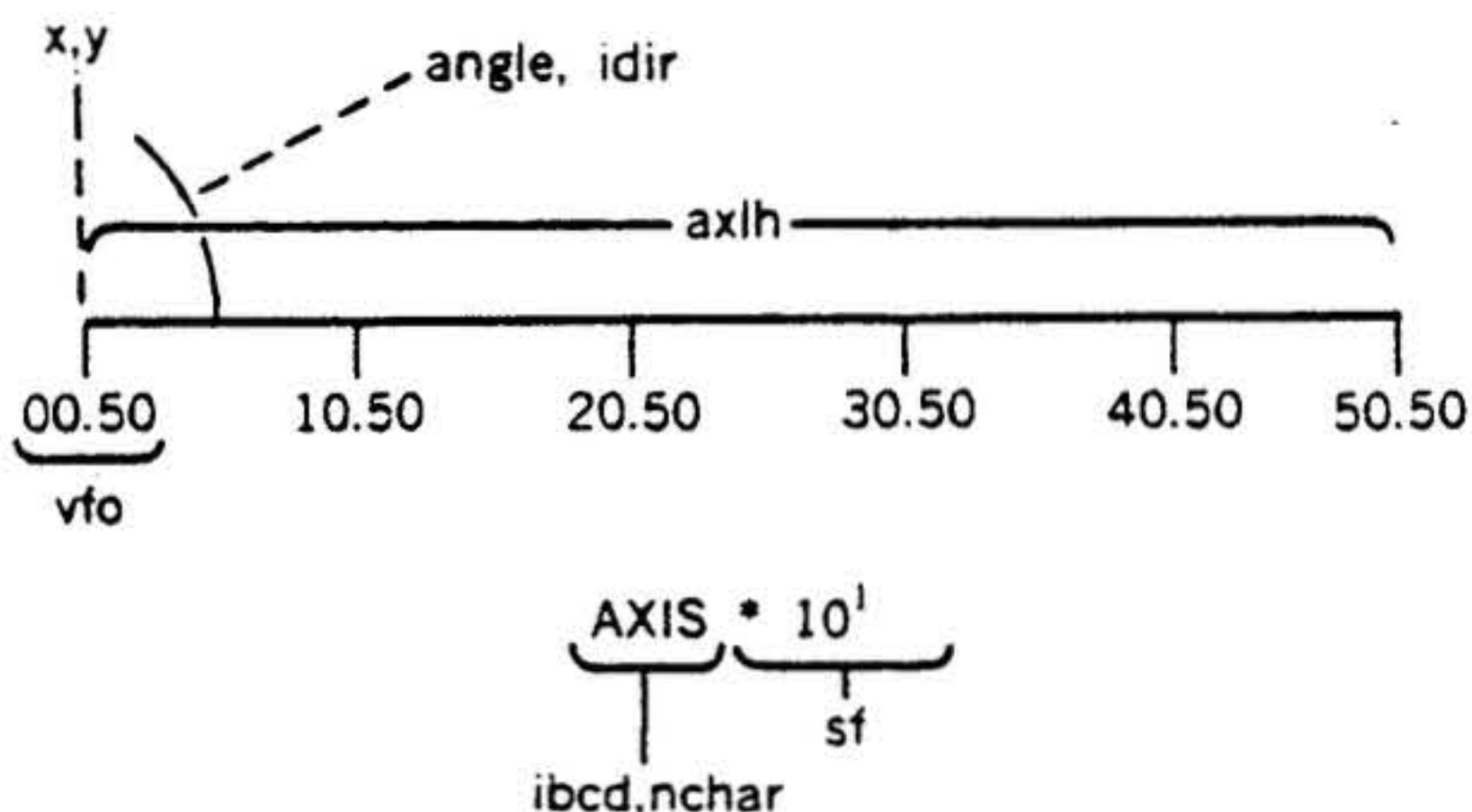
```
CALL AXIS (0.0,0.0,5.0,0,4HAXIS,
           4,5.0,100.0)*
```

```
CALL AXIS (0.0,0.0,4HAXIS,-4,5.0,
           0.0,5.0,100.0)**
```

* MDPOM

** VDPOM

The resulting axis would appear as follows:



12.4.6 SYMBOL (Generate Symbols)

This function generates plot file entries defining printable characters. Each entry contains an x and a y coordinate, a code which specifies that the entry is for a character, a code identifying the character and codes for size and orientation. The characters are software generated dot matrix characters in two sizes (5 x 7 and 10 x 14) and four orientations.

The function is of the general form

```
CALL CHAR (x,y,ibcd,isoar,+nchar,ispac)*
```

or

```
CALL SYMBOL (x,y,height,ibcd,angle, $\pm$ nchar)**
```

* MDPOM

** VDPOM

where

x,y are the x and y coordinates (in inches) of the first letter to be plotted. x will be the minimum x value of the character and y will be the minimum y value of the character (Real).

ibcd is the address of the first word containing the ASCII character string to be plotted. It can be given as an array name or a Hollerith constant.

isoar is the size and orientation:
 0 = small, +90 degrees rotation from x direction.
 1 = small, 0 degrees rotation from x direction.
 2 = large, +90 degrees rotation from x direction.
 3 = large, 0 degrees rotation from x direction.

height selects the character height. If height \leq 0.10, the characters will be 0.07 inches high. If height > 0.10, characters will be 0.14 inches high (Real).

angle is the angle, in degrees from the x-axis, at which the character string is to be plotted. The individual characters will be plotted at 0, 90, 180, or 270 degrees depending on the value of "angle" (Real).

ispac is the spacing constant in styli or scans from the starting coordinate of the previous character. A negative number causes default standard spacing (Integer).

nchar is the total number of characters to be plotted in the string (Integer).

if NCHAR = 0, one character will be plotted from the low order byte of the word containing the string. (VORTEX call only)

If NCHAR = -1, one symbol will be plotted. The symbol must be identified by setting IBCD to an integer (0 through 5). (VORTEX call only)

If NCHAR = -2 or less, one symbol will be plotted along with a vector from the previous current location to the symbol starting location. (VORTEX call only)

IBCD (when NCHAR 0)	Symbol
1	□
2	◇
3	○
4	■
5	●

Character Orientation and Coordinates:

Angle (in degrees)	-44 to 45	46 to 135	136 to 225	226 to 315
--------------------	-----------	-----------	------------	------------

Isaor	1,3			
VORTEX				
MOS				

The dot references the starting coordinate of the character.

Error Conditions: None

Example:

```

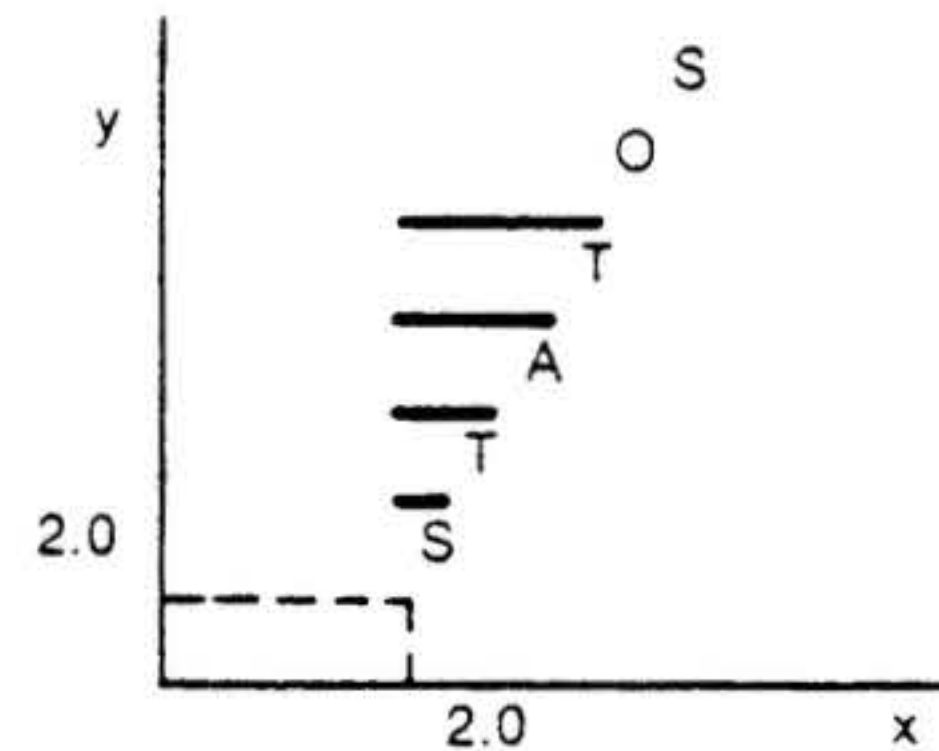
3 DIMENSION LABEL (3)
DATA LABEL/2HST,2HAT,2HOS/
17 CALL CHAR (5.0,5.0,6HSTATOS,2,6,-1)
20 CALL CHAR (5.0,5.0,LABEL,2,6,-1)
    
```

Statement 17 will place six entries for large letters, 90 rotation from the x axis, standard spacing, into the plot file. Statement 20 will do likewise. The characters "STATOS" will be printed starting at 5.0,5.0 from the last origin.

```

25 CALL SYMBOL (2.0,2.0,0.14,6HSTATOS,45.0,6)
    
```

Statement 25 will place six entries for large letters into the plot file. "STATOS" will be printed as follows:



12.4.7 NUMBER (Generate Number)

This function converts single precision real numbers to character codes and places corresponding entries into the plot file.

This function has the general form

```
CALL NUMBER (x,y,fpn,isaor,±ndec)*
```

or

```
CALL NUMBER (x,y,height,fpn,angle,±ndec)**
```

* MDPOM

** VDPOM

where

x,y are coordinates (in inches) of the first number in the string (Real).

fpn is the real number to be plotted. If negative, will be prefixed with a minus sign. Leading zeros will be suppressed, except the zero to the left of the decimal point. The real number is rounded by adding five to the digit to the right of the last digit to be plotted, then truncating the result (Real).

isaor is size and orientation:

0 = small, + 90 degrees rotation from x direction (Default).

DATAPLOT II

- 1 = small, 0 degrees rotation from x direction.
- 2 = large, +90 degrees rotation from x direction.
- 3 = large, 0 degrees rotation from y direction.

height selects the character height. If height = >0.10, the characters will be 0.07 inches high. If height = 0.10, characters will be 0.14 inches high (Real).

angle is the angle, in degrees from the x axis, at which the character string is to be plotted. The individual characters will be plotted at 9, 90, 180, or 270 degrees depending on the value of "angle" (Real).

ndec If this parameter is larger than zero, it defines the number of digits to be plotted to the right of the decimal point.

If NDEC = 0, the integer part will be plotted followed by a decimal point only.

If NDEC = -1, only the integer part will be plotted.

If NDEC is less than -1, (NDEC) - 1 digits are truncated from the integer part (Integer).

The following table illustrates the use of the NDEC parameter.

Suppose FPN = 123.4567; how the number actually will appear is a function of the parameter NDEC.

NDEC	Number Plotted	Comments
4	123.4567	
3	123.457	Note rounding action
2	123.46	
1	123.5	
0	123.	
-1	123	
-2	12	Note truncation action
-3	1	
-4		Nothing is plotted

Error Conditions: None

Example:

```
CALL NUMBER (1.0, 2.0, 12.3, 3, 1) *
CALL NUMBER (1.0, 2.0, 0.14, 12.3,
             0.0, 1) **
```

The above will produce the number 12.3 at location x = 1.0, y = 2.0 in 10 x 14 character matrix, zero degrees from the x axis.

* MDPOM ** VDPOM

12.4.8 LINE (Generate Graph Line)

Subroutines DATA and LINE produce a data line with one call. Prior to the call, the data must be placed in two arrays which have been dimensioned to provide two extra locations in each array. These must be placed at the end of the arrays and contain the displacement and scale factors in that order. The two arrays must be of equal size, one containing x values and the other y values.

The subroutine is of the general form

```
CALL DATA (xarr,yarr,npts,inc,±lty,ieq)*
or
CALL LINE (xarr,yarr,npts,inc,±lty,ieq)**
* MDPOM
** VDPOM
```

where

xarr is the name of the array from which x values are to be extracted.

yarr is the name of the array from which the y values are to be extracted.

npts is the number of data points to be plotted from each array to the end of the array (Integer).

inc is the increment at which the arrays are to be sampled. INC = 1 means every x,y pair is plotted. INC = 2 means every other pair, etc. (Integer).

±lty indicates the type of line desired (Integer).

LTY < 0: A symbol will be plotted at each selected point but no lines will connect the symbols.

LTY = 0: A line will be drawn connecting each selected point. No symbols will be drawn.

LTY > 0: A symbol will be plotted at each selected point and a line will connect all symbols.

ieq is the positive integer designating symbol to be produced (1,2,3,4, or 5).

If LTY = 0, IEQ has no meaning.

Plot values will be generated by the following algorithm:

Plot Value = array value—displacement
scale factor

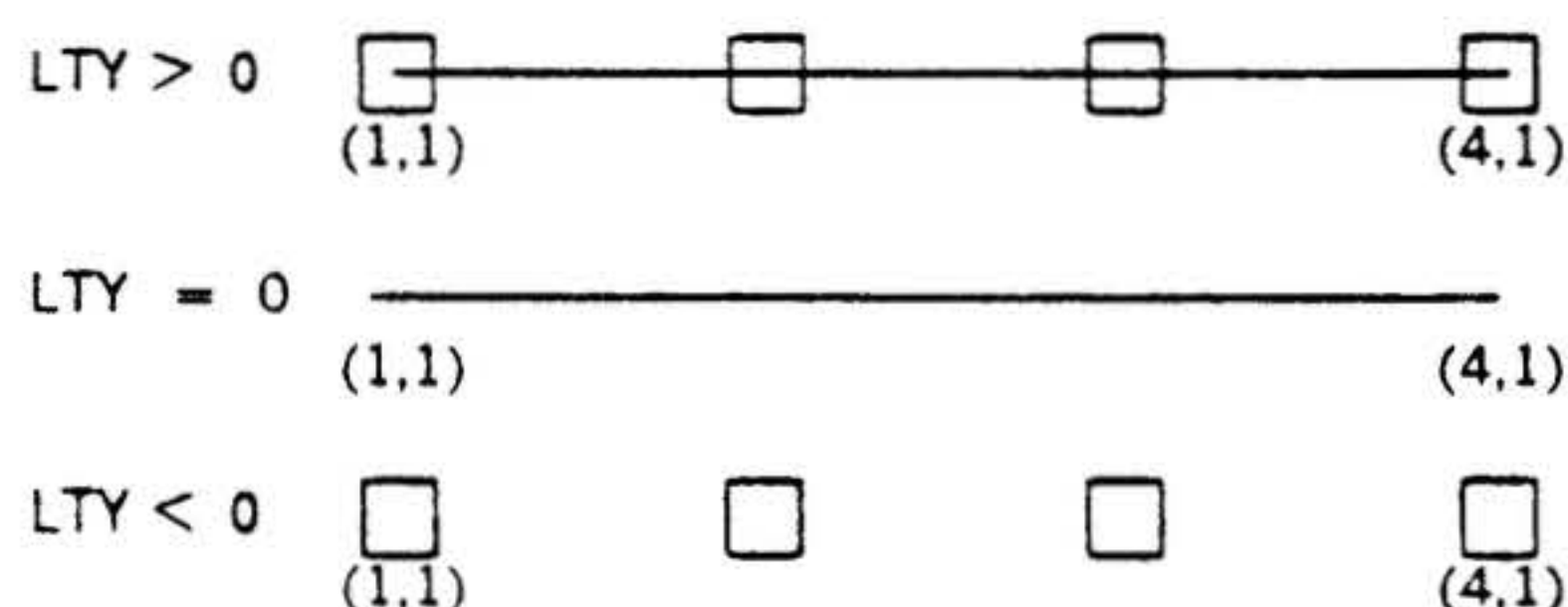
Error Conditions:

Condition: The scale factor in the data array = 0.0
 Action: Incomplete plot
 Message: ARITH OVFL

Examples:

```
DIMENSION XAR (6), YAR (6)
DATA XAR/1.0,2.0,3.0,4.0,1.0,1.0/
DATA YAR/1.0,1.0,1.0,1.0,1.0,1.0/
.
.
CALL DATA (XAR,YAR,4,1,LTY,1)
or
CALL LINE (XAR,YAR,4,1,LTY,1)
```

The above will produce the following plots:



12.4.9 MLTPLE (Multiple Plot)

The sign of the PLOT parameter IDRAW is used to indicate whether a new logical origin is to be defined. The MLTPLE call allows the user to change the origin without terminating his current plot definition. If no call has been made to MLTPLE, the PLOT origin change is treated as the completion of the current plot and the start of the new plot.

The subroutine is of the general form

```
CALL MLTPLE (ind)
```

*BDPOM
 where

ind +1 = on future calls to PLOT, a redefinition of the logical origin will not be treated as the end of the plot, and multiple logical plots will be treated as belonging to the same real plot.

0 = on future calls to PLOT, a redefinition of the logical origin will also be treated as the end of the plot.

-1 = Same as +1 except that the accumulated information from past PLOT calls defines a complete plot and it should be output. Note that the statement CALL MLTPLE (-1) is exactly equivalent to:

```
CALL WHERE (x,y, fact)
CALL MLTPLE (0)
CALL PLOT (0.0,0.0,-3)
CALL MLTPLE (+1)
CALL PLOT (x,y,+3)
```

Error Conditions: None

Examples:

```
CALL PLOT (1.0,2.0,3)
CALL PLOT (2.0,2.0,-2)
CALL PLOT (3.0,3.0,3)
CALL PLOT (4.0,4.0,2)
CALL PLOT (0.0,0.0,999)
```

The above sequence will output two physical plots of one line each.

```
CALL MLTPLE (1)
CALL PLOT (1.0,2.0,3)
CALL PLOT (2.0,,2.0,-2)
CALL PLOT (3.0,3.0,3)
CALL PLOT (4.0,4.0,2)
CALL PLOT (0.0,0.0,999)
```

The above sequence will output one physical plot with two lines on the plot.

12.4.10 FACTOR (Alter Plot Size)

This function is used to alter the overall size of the plot by changing the ratio of the desired plot size to the normal size.

The function is of the general form

```
CALL FACTOR (fact)
```

*BDPOM
 where

fact is the ratio of the desired plot size to normal plot size. If FACTOR is not called, fact = 1.0 (Real).

Error Conditions: None

Example: Make plot one-half normal size.

```
CALL FACTOR (0.5)
```

12.4.11 WHERE (Locate Coordinates)

This function returns information to the user. The three variables designated in the calling sequence are set to the current x and y coordinates and the current plot sizing factor.

DATAPLOT II

The function is of the general form

CALL WHERE (rx,ry,rfact)*

*BDPOM

where

rx is the variable which will be set to the current x coordinate.

ry is the variable which will be set to the current y coordinate.

rfact is the variable which will be set to the current plot sizing factor.

Error Conditions: None

Example:

```
CALL MLTPLE (1)
CALL FACTOR (2.5)
CALL PLOT (1.0,2.0,3)
CALL WHERE (XA,YA,F)
CALL PLOT (3.0,1.0,-2)
CALL WHERE (XB,YB,F)
```

The above sequence will set the variables as follows:

```
XA = 1.0
YA = 2.0
F = 2.5
XB = 0.0
YB = 0.0 new origin defined
```

12.4.12 APPEND (Append File)

Previously generated files in vector-end-point format may be added to the plot file and merged during the sort. A call to APPEND must be made after the call to PLOTS. If the file to be appended is not on an RMD device, it must be previously positioned.

The function is of the general form

CALL APPEND (lun,key,name,abuff,iabuff)*

*BDPOM

where

lun is the variable or number of the logical unit containing the file to be appended (Integer).

key is the protection key, if any.

name is the six-character name of the file to be appended. It may be given as an array name or a Hollerith constant.

abuff the name of the APPEND file input buffer.

iabuff is the length of abuff (Integer).

Error Conditions:

Condition: Wrong protection key
Action: Append call is ignored
Message: IO04,xxxxxx

Condition: File name not found
Action: Append call is ignored
Message: IO10,xxxxxx

xxxxxx is the task name.

Examples:

```
117 CALL APPEND (18,0,0,BUFF,1024)
136 CALL APPEND (132,2HbP,6HMAPbb,
                ABUFF,960)
```

Statement 117 will cause the file on logical unit 18 to be appended to the plot file. BUFF will be used as the input buffer. Statement 136 will cause the file named MAP on logical unit 132, with protection code P, to be appended to the plot file. ABUFF will be used as the input buffer. Data will be input in blocks of 960 words (8 sectors).

12.4.13 TOPFRM (Top-of-Form)

TOPFRM subroutine will advance the paper to the next TOP-OF-FORM mark or eleven inches, whichever occurs first (FUNC code = 0). A Top-of-Form command will be output to the output driver at the time the subroutine is called.

The subroutine is of the general form

CALL TOPFRM*

*BDPOM

Error Conditions: None

Example:

```
CALL TOPFRM (Outputs FUNC (0)
to the plot output device)
```

12.4.14 CUT (Cut Paper)

The CUT subroutine issues a cut command (FUNC code = 20) to the output driver when the subroutine is called.

The subroutine is of the general form

CALL CUT*

*BDPOM

Error Conditions:

Condition: Paper cutter option not connected.
Action: Command ignored
Message: none

Example:

CALL CUT

A cut command (FUNC (20)) is sent to the plot output device.

12.4.15 ENDCUT (Eject and Cut Paper)

The ENDCUT subroutine issues a FUNC code equal to 21 (cut command) to the output device and moves the paper approximately 34 inches.

The subroutine is of the general form

```

          CALL ENDCUT*
*BDPOM
Error Conditions:

Condition:   Output device not STATOS.
Action:     Command ignored
Message:    None
    
```

Example:

CALL ENDCUT

The above issues a cut and move paper command to the plot output device.

12.4.16 DPSORT (Sort Plot File)

This function sorts an RMD plot file. No sort is attempted if the plot file is not assigned to an RMD.

DPSORT is also called by subprogram DPPLLOT when IDRAW = 999, or when IDRAW = 1, or when MLTPLE is set 0.

The function is of the general form

```

          CALL DPSORT*
*BDPOM
Parameter Description:   None
    
```

Error Conditions:

```

Condition:   Data Plot working buffer too small.
Action:     Abort program
Message:    DP01

Condition:   Plot file not assigned to RMD.
Action:     Abort program
Message:    DP07
    
```

Example:

CALL DPSORT

12.4.17 DPPLLOT (Output File)

DPPLLOT subroutine converts the plot file to STATOS raster format and outputs the raster data to the output device specified in the call to PLOTS. DPPLLOT is called by subroutine PLOT when IDRAW = 999 or when IDRAW < 0, and MLTPLE = 0 or when MLTPLE is set < 0, to output the plot data.

This subroutine is of the general form

```

          CALL DPPLLOT*
*BDPOM
Parameter Description:   None
    
```

Error Conditions:

```

Condition:   Working buffer overflow
Action:     Incomplete plot
Message:    DP01

Condition:   Attempted to plot from unsorted File.
Action:     Abort plot
Message:    DP02

Condition:   End-of-plot indicator not detected.
Action:     Abort plot
Message:    DP03

Condition:   Min/Max x/y values exceeded.
Action:     Line will follow plot boundary,
            plot origin will be shifted.
Message:    DP04

Condition:   PLOTS not called.
Action:     Abort plot
Message:    DP05
    
```

Example:

```

DIMENSION IBUF (1200)
CALL PLOTS (IBUF,1200,5)
CALL DPINIT (107,2HbF,6HPLTFIL,
            120,88)
CALL DPSORT } or CALL PLOT
CALL DPPLLOT } (0.0,0.0,999)
    
```

The above program will output raster plot data to logical unit 5, block size 88, from an unsorted plot file residing on logical unit 107, protection code of F, name PLTFIL, block size of 120.

If the plot file is sorted, the call to DPSORT may be eliminated.

If the plot file is on system scratch (SS) and the STATOS is 14-7/8 inches wide, the call to DPINIT may be eliminated.

12.4.18 DPCLOS (Close Plot File)

DPCLOS subroutine closes and updates the plot file and writes an end-of-file if the plot file is on magnetic tape. The first three words of DPFCB (data plot file control block) are set to zero, and the plot file cannot be referenced until a call is made to DPINIT to restore DPFCB.

The subroutine is of the general form

CALL DPCLOS*

*BDPOM

Parameter Description: None

Error Conditions:

If the plot file is assigned to a device other than an RMD or magnetic tape, the close request will be ignored.

Example:

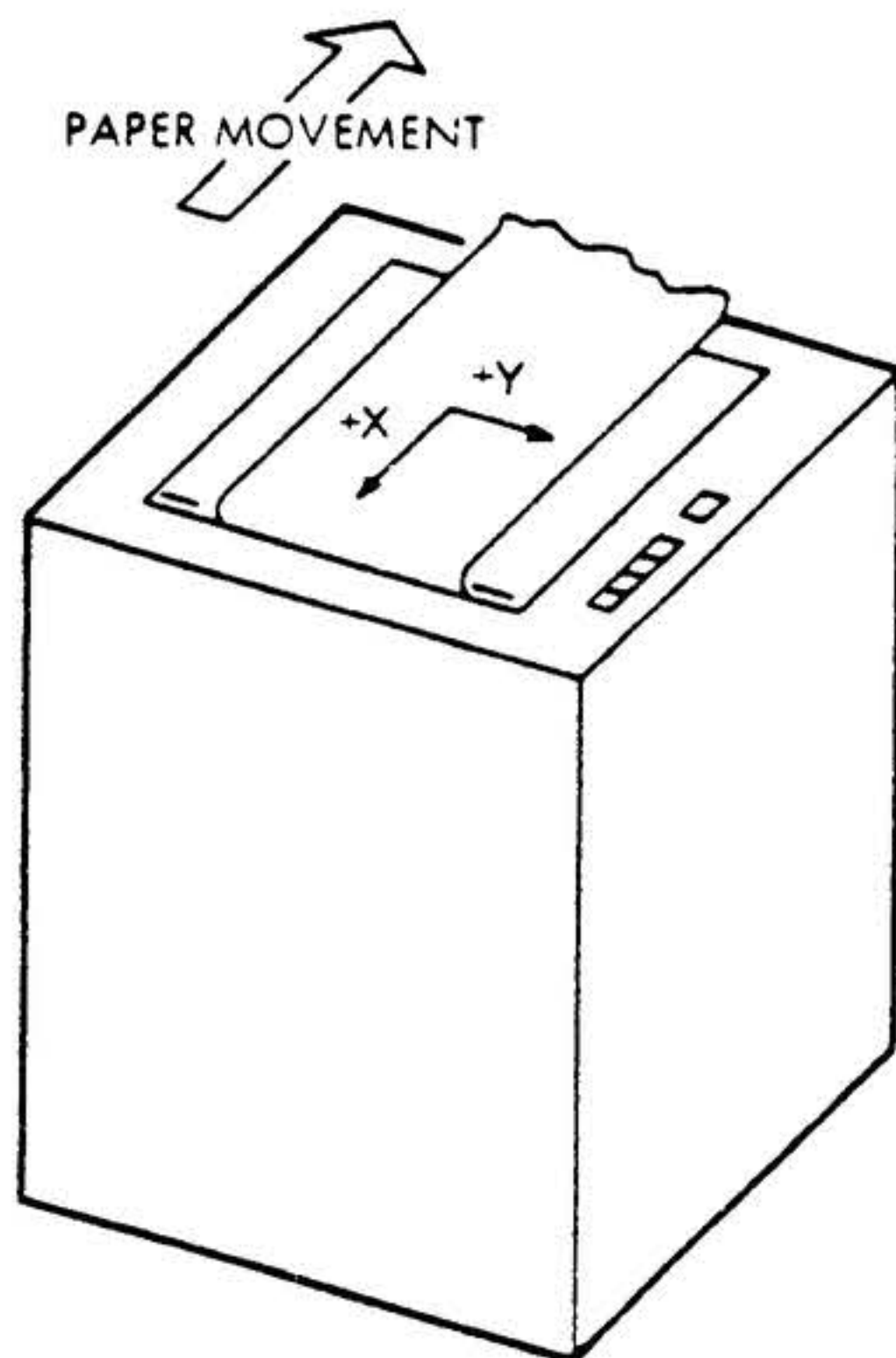
```
170 CALL DPCLOS
```

Statement 170 closes the plot file.

12.4.19 ORIG -- Offsetting the Origin Entry Point

This function offsets the origin entry point of the plot.

The origin of the plot is the lower left hand corner of the plot area, with the +y axis towards the right and the +x axis pointing into the plotter.



VTII-3087

Figure 12-4. +x Axis and +y Axis Relative to Paper Direction

The absolute y displacement may not go negative. If it is desired to offset the origin in order to allow (relative) negative numbers, or to allow large positive values to be plotted without wasting paper, it is possible to offset both x and y coordinates of the (relative) origin by the following call of the general form:

CALL ORIG (x,y)*

*BDPOM

where

x is the distance (in inches) along the x axis which the new (relative) origin will be offset (Real).

y is the distance (in inches) along the y axis which the new (relative) origin will be offset (Real).

The coordinates used in locating plot elements are always relative to the origin location.

Error Conditions: None

Example:

```
170 CALL ORIG (7.0,3.1)
```

Statement 170 offsets the origin 7.0 inches in the x direction and 3.1 inches in the y direction from the physical origin (0.0,0.0).

12.4.20 VECT -- Vector Entry Point

This subroutine generates plot file entries defining straight lines between two points. Four parameters define the points in the following order:

x1, y1, x2, y2. The parameters are single precision, real numbers representing inches from the origin. Provision is made for retaining the "current" (or last defined) point. When x1 = 999.0, a file entry is produced to generate a line between the "current" point and the point defined by x2 and y2.

The subroutine is of the general form

CALL VECT (x1,y1,x2,y2)*

*BDPOM

where

x1 is the starting x coordinate of line.

y1 is the starting y coordinate of line.

x2 is the ending x coordinate of line.

y2 is the ending y coordinate of line.

Error Conditions: The normal plotter routines do not keep track of the actual location of the pen, but instead always assume that the pen can be moved from the current location to the new location and that enough commands are output to accomplish this. If a mechanical stop is encountered during plotting, the motion in that direction is simply inhibited by the plotter. Because the mechanical stops are not precise, errors will be produced if a mechanical stop is encountered during plotting. However, this is sometimes done before initiating a plot to position the pen in a known location before beginning the actual plot.

DATAPLOT II routines have software stops contained internally in order to produce the same effect. If a plot boundary is encountered, an error (DP04) will be reported, the line will extend toward the boundary and follow the boundary to the final position, and the origin will be effectively shifted in a manner similar to the pen plotter.

Example: 5 CALL VECT (3.2,1.0,4.0,1.0)

Statement 5 will place an entry in the plot file for the vector x = 3.2 to 4.0 and y = 1.0.

12.4.21 Special SYMBOL Subroutine

Subroutine SYMBOL produces special symbols on the plot.

The subroutine is of the general form

CALL SYMBOL (x,y,ieq)*

* MDPOM

where

x,y are the x and y coordinates of the center of the symbol (Real).

ieq is the positive integer designating the symbol to be produced.

IEQ	SYMBOL
1	□
2	◇
3	○
4	■
5	●

Error Conditions: None

Example:

CALL SYMBOL (1.0,2.0,4)

The above will place a filled in square (■) at location x = 1.0, y = 2.0.

12.4.22 DENSTY (Alter Stylii/Inch)

This function is used to set the system stylii/inch constants to 200 for STATOS 42 models 70-6661 through 70-6668. The function is of the specific form

CALL DENSTY(200)

*BDPOM

where

200 is the number of stylii/inch. If DENSTY is not called or if any raster density other than integer 200 is specified, the system will default to 100 stylii/inch.

Error Conditions:

- Condition--resultant plots will be one-half the size expected from the program.
- Action--include or correct the call to DENSTY.

12.5 PLOT FILE DATA FORMAT

12.5.1 Vectors

X values represent distances from the beginning of the plot in the opposite direction of paper movement. A unit of x corresponds to one step of paper movement in the machine.

Y values represent stylus numbers.

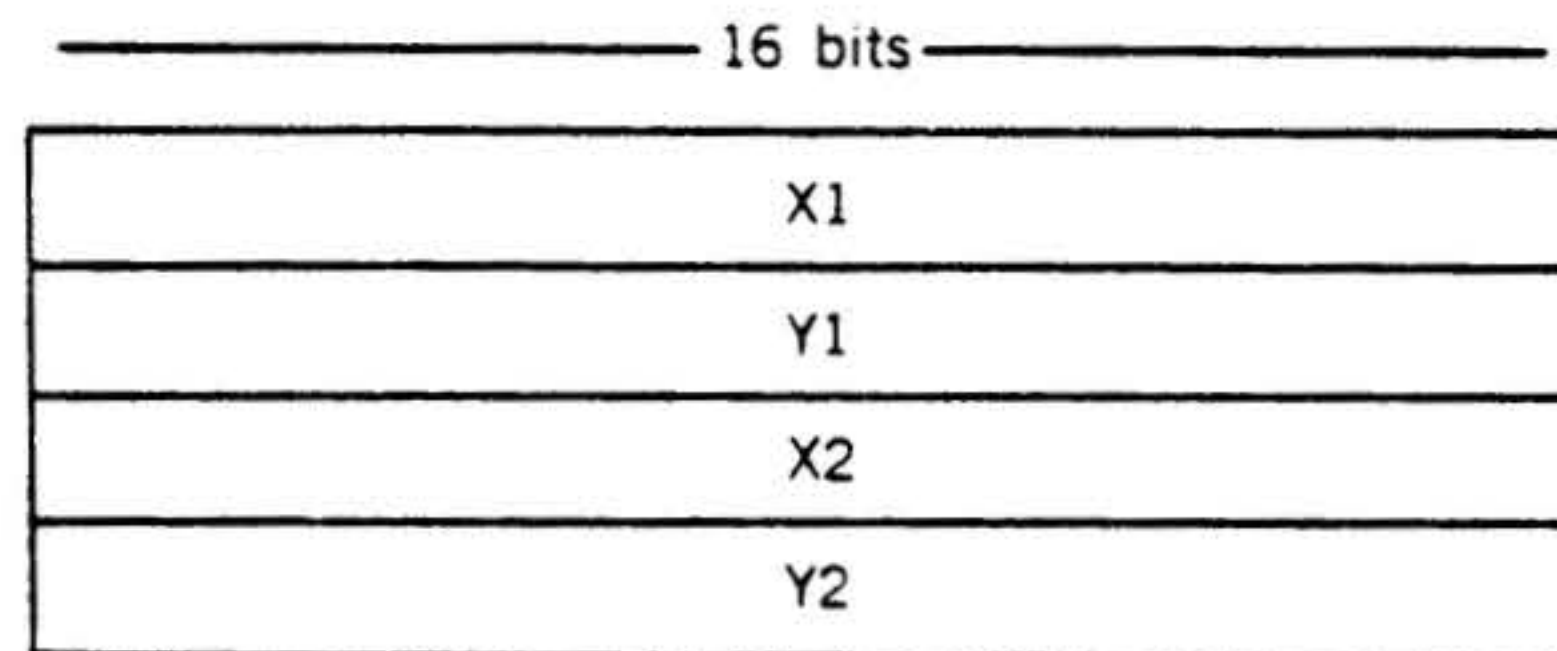


Figure 12-5. Vector-Data Format

where

$X2 \leq X1 \leq 32,700$

Y1 and Y2 number of STATOS stylii

DATAPLOT II

12.5.2 Characters

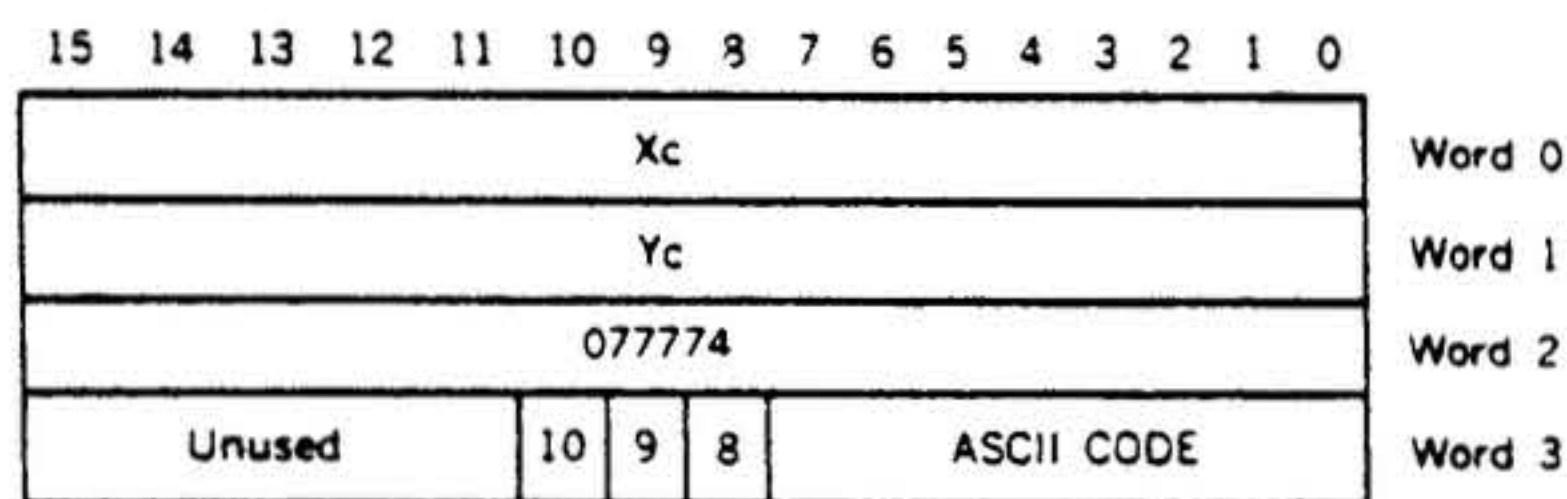
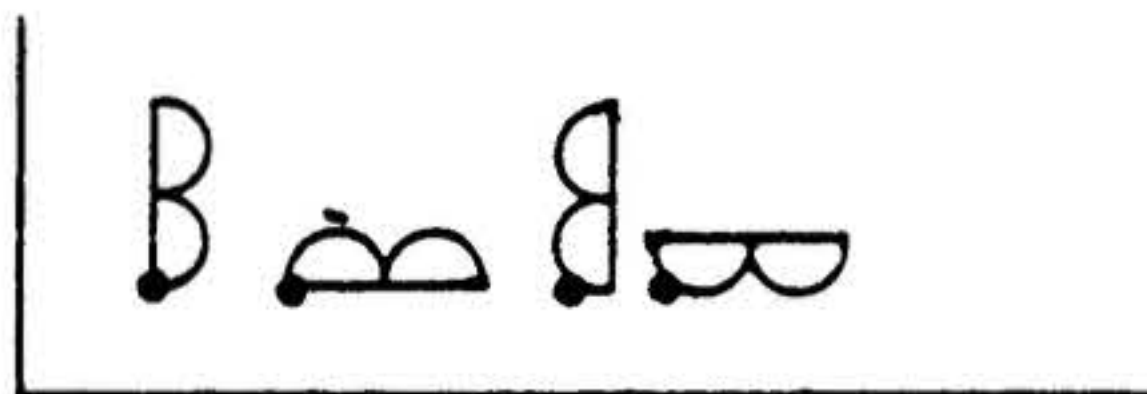


Figure 12-6. Character Data Format

Word 3, Bit 9 = 0 for small character (5x7)
 = 1 for large character (10x14)

Word 3, Bit 8 and 10 determine the character orientation.
 The x and y coordinates refer to the lower left-hand corner
 of the character.



Bit 8	1	0	1	0
Bit 10	0	0	1	1

Figure 12-7. Character Orientation Data Format

12.5.3 End-of-Plot Indicator

The end of the plot indicator is shown in figure 12-8.

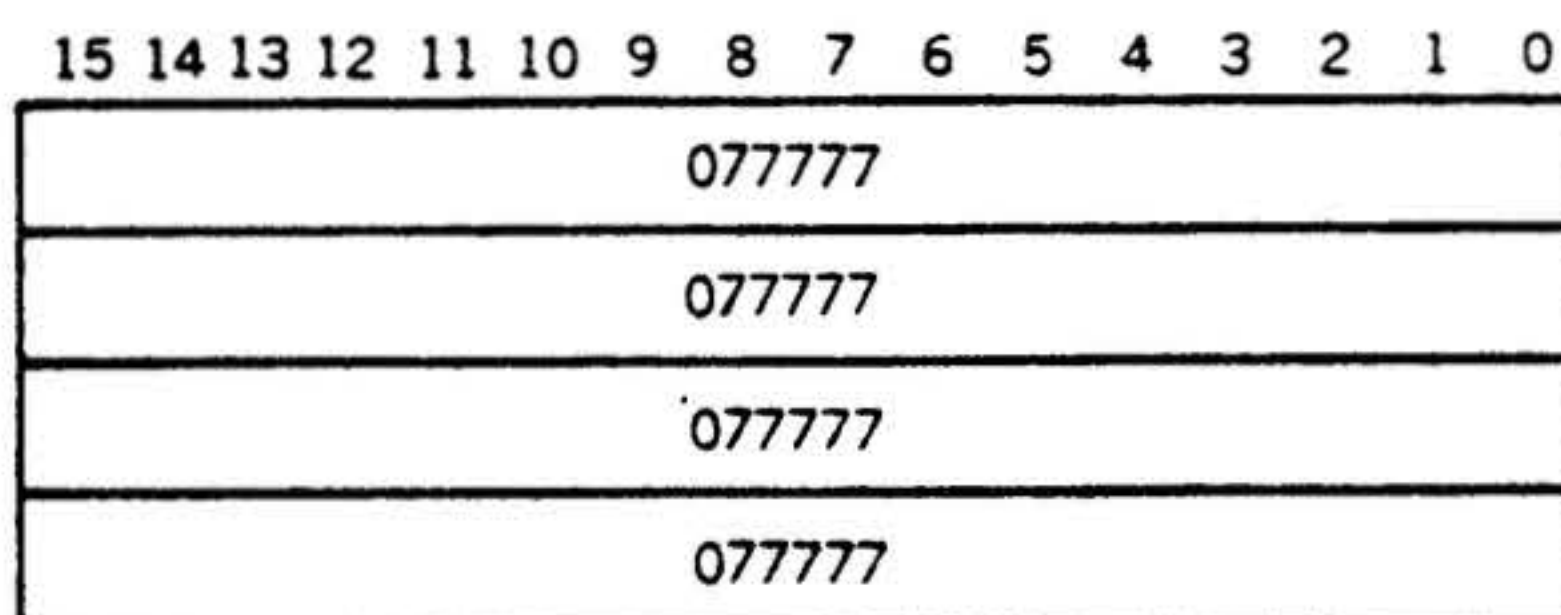


Figure 12-8. End-of-Plot Indicator

12.6 EXAMPLE OF APPLICATION OF DATAPLOT II

12.6.1 Program to Generate Sine Wave

```

C
C      SAMPLE PLOT (BDPOM/VDPOM CALLS)
C
      DIMENSION XAR (34),YAR(34),
      IBUFF(1000)
    
```

```

      XAR (33) = 0.0
      XAR (34) = 1.0
      YAR (33) = -100.0
      YAR (34) = 100.0
      CALL PLOTS (IBUFF,1000,5)
      CALL MLTPLE (1)
      CALL PLOT (1.0,1.0,-3)
      XVA = 0.0
      DO 200 I = 1,32
      XVA = XVA + 0.25
      XAR (I) = XVA
      200 YAR (I) = 100.0 + 200.0 * SIN(XVA)
C
C      PLOT AXES, DATA
C
      CALL AXIS (0.0,0.0,6HY-AXIS,
      6,4.0,90.0,YAR(33),YAR(34)
      CALL AXIS (0.0,0.0,6HX-AXIS,
      -6,8.0,0.0,XAR(33),XAR(34))
      CALL LINE (XAR,YAR,32,1,-1,1)
      CALL PLOT (0.0,0.0,999)
      CALL EXIT
      END
      (END-OF-FILE)
    
```

```

C
C      SAMPLE PLOT (BDPOM/MDPOM CALLS)
C
      DIMENSION XAR (34), YAR (34),
      IBUFF (1000)
      XAR (33) = 0.0
      XAR (34) = 1.0
      YAR (33) = -100.0
      YAR (34) = 100.0
      CALL PLOTS (IBUFF, 1000, 5)
      CALL ORIG (1.0, 1.0)
      XVA = 0.0
      DO 200 I = 1, 32
      XVA = XVA + 0.25
      XAR (I) = XVA
      200 YAR (I) = 100.0 + 200.0 * SIN (XVA)
C
C      PLOT AXES, DATA
C
      CALL AXIS (0.0, 0.0, 4.0, 1,
      6HY-AXIS, -6, YAR (33),
      YAR (34))
      CALL AXIS (0.0, 0.0, 8.0, 0,
      6HX-AXIS, 6, XAR (33),
      XAR (34))
      CALL DATA (XAR, YAR, 32, 1, -1, 1)
      CALL PLOT (0.0, 0.0, 999)
      CALL EXIT
      END
    
```

12.6.2 Program to Generate Communication Network

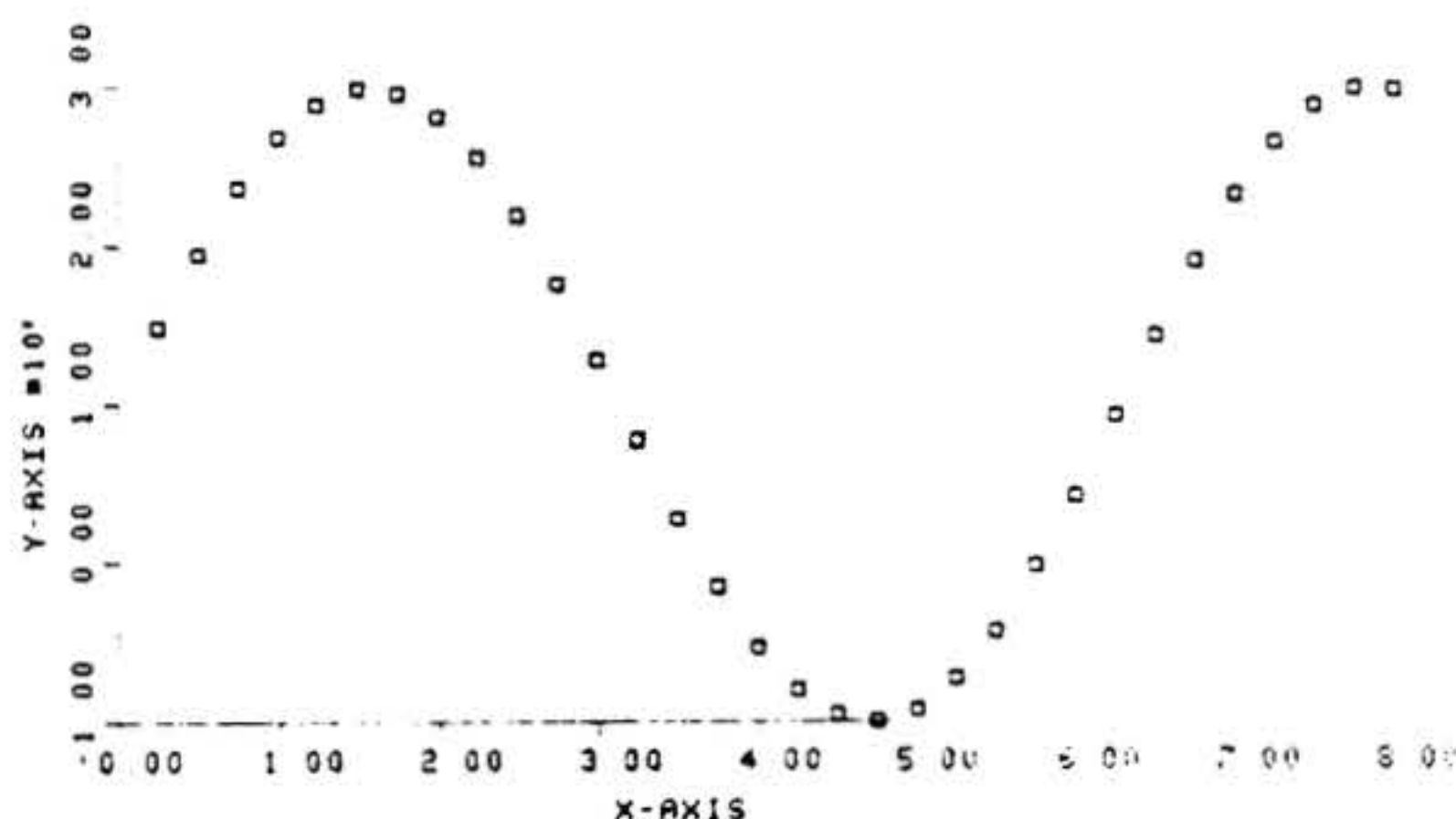
```

C
C      SAMPLE COMMUNICATIONS NETWORK
C
      DIMENSION IBUFF (1000),
      XAR(12),YAR(12)
      CALL PLOTS (IBUFF,1000,5)
    
```

```

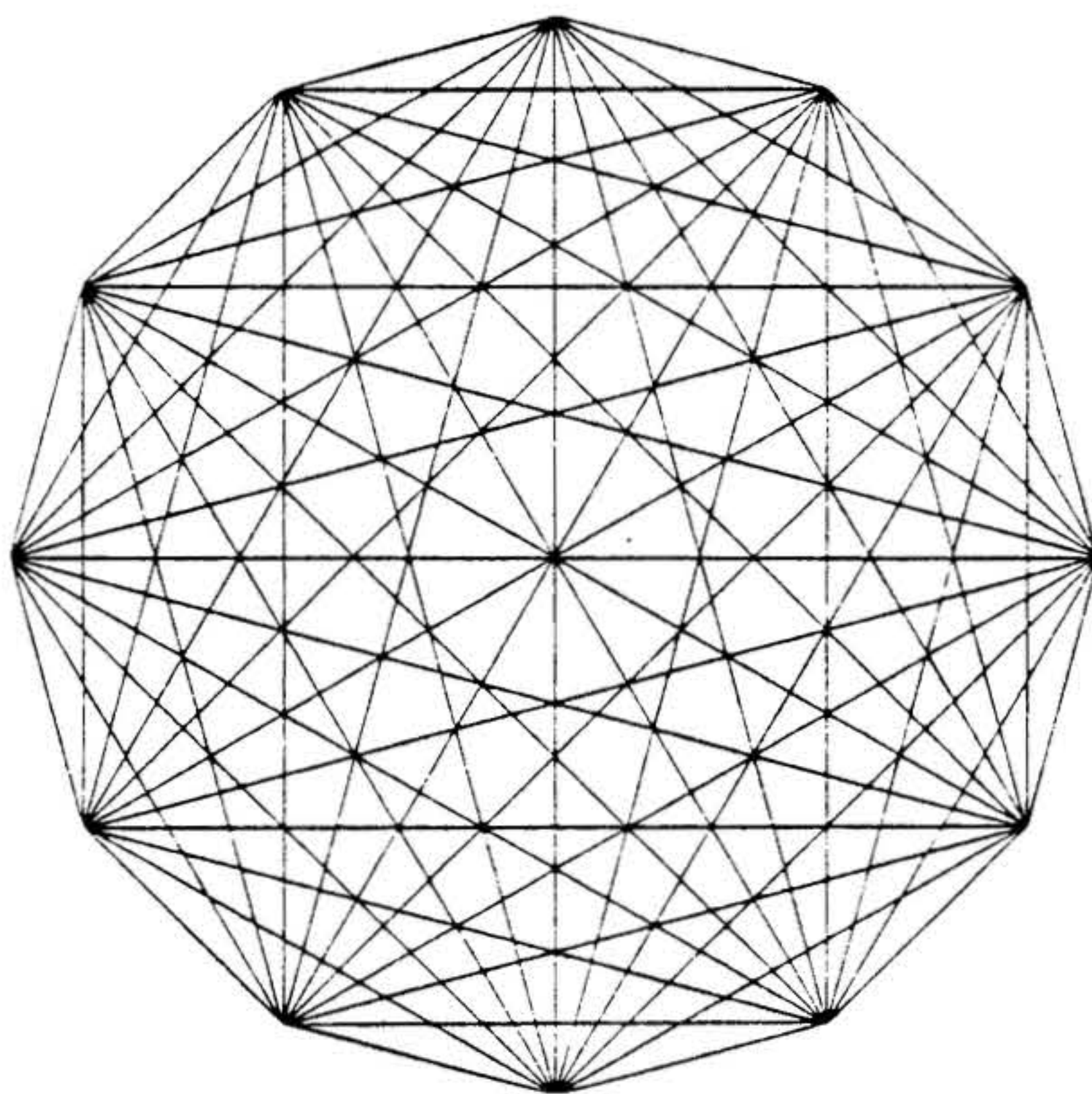
C   BUILD END-POINTS
      DO 10 I = 1, 12
      X = 6.283 * FLOAT (I)/12.0
      YAR(I) = 5.0 * SIN (X)+7.0
10  XAR(I) = 5.0 * COS(X)+7.0

C   DRAW THE LINES
      DO 30 I1 = 1, 11
      K = I1 + 1
      DO 30 I2 = K, 12
30  CALL VECT (XAR(I1),YAR (I1),
      XAR(I2),YAR(I2))
      CALL PLOT (0.0,0.0,-3)
      CALL EXIT
      END
( END-OF-FILE )
    
```



VT11-3095

Figure 12-9. Sine Wave Plot Generated by DATAPLOT II



VT11-3094

Figure 12-10. Communication Network Plot Generated by DATAPLOT II

12.7 OPERATING PROCEDURES AND ERROR MESSAGES

12.7.1 VORTEX Operating Procedures

Use of the DATAPLOT II plot generation routines requires the preparation of FORTRAN programs which make appropriate calls to the FORTRAN and SPERRY UNIVAC 70/620 assembly language programs.

The user may execute in a compile-and-go mode by ending his program with a call to PLOT (x,y,999) or PLOT (x,y,-i) and the plot output device assigned to the STATOS printer/plotter (Ref. paras 12.4.2).

12.7.2 Unsorted Plot Files

Unsorted plot files may be output by VORTEX DATAPLOT II by transferring the plot file to an RMD (if not already there) by IOUTIL or the APPEND subroutine, and calling the following subroutines:

```

DIMENSION
CALL DPINT      (          ) if necessary
CALL DENSTY    (          ) if necessary
CALL PLOTS     (          ) if necessary
CALL DPSORT
CALL DPLOT
CALL EXIT
END
    
```

12.7.3 Presorted Plot Files

Files which have been presorted may be in physical records whose length is any multiple of four 16-bit words. There is no restriction on the number of records which may be processed, other than the physical capacity of the peripheral device. The file must have been sorted on the numerical value of the X1's, in descending order. Each X1 must be greater than or equal to its associated X2. An end-of-plot indicator (four words containing 077777) must appear at the end of the significant data in the last record.

Presorted plot files may be output by VORTEX DATAPLOT II by assigning the plot file to the physical unit containing the plot file (DPINIT) and calling the following routines:

```

DIMENSION
CALL DPINIT    (          ) if necessary
CALL DENSTY    (          ) if necessary
CALL PLOTS     (          ) if necessary
CALL DPLOT
CALL EXIT
END
    
```

12.7.4 VORTEX Special Procedures

The VORTEX DATAPLOT II package may be executed in one, two, or three sections. No special modifications are necessary to build, sort, and output the plot file in one module.

Sorting and outputting the plot file may be separated from building the plot file by supplying dummy sorting and outputting routines. For example, this method may be used if it is desired to build the plot file in the background and output the plot file from the foreground. Subroutine PLOTS must be included in each section or an error (DP05) will be output. For STATOS models 70-6661 through 70-6668, CALL DENSTY(200) must precede the call to PLOTS or the resultant plot will be half the expected size.

Example:

```

/FORT,B,L,M
C   BUILD THE PLOT FILE
    DIMENSION Ibuff (142)
    CALL DPINIT (25,2HbK,6HFILEbb,
    120,88)
    CALL PLOTS (IBUFF,124,27)
    CALL AXIS (1.0,1.0,4HAXIS,4,5.0,
    0.0,0.0,1.0)
    CALL PLOT (0.0,0.0,999)
    CALL EXIT
    END
    
```

```

C   DUMMY SUBROUTINES
    SUBROUTINE DPSORT
    RETURN
    END
    SUBROUTINE DPLOT
    RETURN
    END

/FORT,B,L,M
C   SORT AND OUTPUT THE PLOT FILE
    DIMENSION Ibuff (1000)
    CALL DPINIT (25,2HbK,6HFILEbb,
    120,88)
    CALL PLOTS (IBUFF,1000,27)
    CALL DPSORT
    CALL DPLOT
    CALL EXIT
    END
(END-OF-FILE)
    
```

The above programs referenced the plot file named FILE on logical unit number 25, protection code K.

The Ibuff in the first program only needs to be the plot file record size (120) plus 22. The size of Ibuff in the second program may be increased to provide faster sorting when large plot files are generated.

SECTION 13

SUPPORT LIBRARY

The VORTEX system has a comprehensive subroutine library directly available to the user. The library contains mathematical subroutines to support the execution of a program, plus many commonly used utility subroutines. To use the library, merely code the proper call in the program, or, for the standard FORTRAN IV functions, implicitly reference the subroutine (e.g., $A = \text{SQRT}(B)$ generates a `CALL SQRT(B)`). All calls generate a reference to the required routine, and the load-module generator brings the subroutine into memory and links it to the calling program.

The performance of several routines in the support library is improved through the use of the V70 series Floating Point Firmware on V70 series systems having Writable Control Store (WCS). The necessary firmware and library routines which call the firmware are added to the Object Module Library (OM) by executing the supplemental WCS job stream supplied with the System Generation Library.

13.1 CALLING SEQUENCE

The subroutines in the support library are called through DAS MR or FORTRAN IV.

DAS MR: *General form:*

```
label CALL S,p(1),p(2),...,p(n)
```

Expansion:

```
label      JMPM      S
           DATA    p(1)
           DATA    p(2)
           .
           .
           .
           DATA    p(n)
```

Single-Precision Floating-Point Numbers

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
n)	s	-----Exponent-----						----High Mantissa----								
n+1)	0	-----Low Mantissa-----														

Double-precision floating-point numbers use four consecutive 16-bit words. The exponent (in excess 0200 form) is in bits 7 to 0 of the first word. The mantissa of a positive number is in the second, third, and fourth words. Bit 15 of the second, third and fourth words and bits 15 to 8 of the first word are zero. The negative of this number is created by one's complementing the second word. Any real number in the range $10^{\pm 13}$ can be stored as a double-precision floating-point number having a precision of more than 13 decimal digits.

Double-Precision Floating-Point Numbers

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
n)	0	0	0	0	0	0	0	-----Exponent-----								
n+1)	s	-----High Mantissa-----														
n+2)	0	-----Mid Mantissa-----														
n+3)	0	-----Low Mantissa-----														

FORTRAN IV: *General form:*

```
statement number CALL S(p(1),p(2),...,p(n))
```

Generated code:

```
JMPM      S
DATA      q(1)
DATA      q(2)
.
.
.
DATA      q(n)
```

Where $q(i) = p(i)$ if $p(i)$ is a single variable or array name. Otherwise, $q(i) =$ address containing $p(i)$.

13.2 NUMBER TYPES AND FORMATS

Integers use one 16-bit word. A negative number is in two's complement form. An integer in the range - 32,767 to + 32,767 can be stored as an integer.

Real numbers use two consecutive 16-bit words. For a positive real number, the exponent (in excess 0200 form) is in bits 14 to 7 of the first word. The mantissa is in bits 6 to 0 of the first word and bits 14 to 0 of the second word. The sign bit of the second word is zero. The negative of this number is created by one's complementing the first word. Any real number in the range $10^{\pm 13}$ can be stored as a single-precision floating-point number having a precision of more than six decimal digits.

13.3 SUBROUTINE DESCRIPTIONS

The following definitions and notation apply to the subroutine descriptions given in this section:

Notation	Meaning
AB	Hardware A and B registers
AC	Four-word software accumulator for double-precision numbers
ACCZ	Four-word accumulator for complex numbers (the real part is in AB and the imaginary part is in a temporary cell in subroutine V\$8G)
d	Address of a double-precision number
f	Address of a two-word, fixed-point number
i	Address of an integer

r	Address of a real number
S	A six-character ASCII string
X	Hardware X register
z	Address of a complex number
**	Exponentiation

An additional name in parentheses indicates a replacement by standard firmware. For example, \$SE(FSE) indicates the firmware routine FSE replaces \$SE on 70 series systems using standard firmware. Section 20.2 describes standard firmware.

The external references in table 13-3 refer to items in tables 13-1 and 13-2. A subroutine with more than one name is indicated by multiple calls under Calling Sequence.

Table 13-1. DAS Coded Subroutines

Name	Function	Calling Sequence	External References
\$HE	Given: A contains i_1 , in A compute $i_1^{*i_2}$.	CALL \$HE, i_2	\$SE(FSE), \$HM
\$PE	Given: AB contains r , in AB, compute r^{*i} .	CALL \$PE, i	\$SE(FSE), \$QM, \$QN
\$QE	Given: AB contains r_1 , in AB, compute $r_1^{*r_2}$.	CALL \$QE, r_2	ALOG, \$QM, EXP, \$SE(FSE)
ALOG	In AB, compute $\ln r$. If $r = 0$, output message FUNC ARG and exit with $A = B = 0$ and overflow = 1.	CALL ALOG, r	\$EE, \$QK(FAD), \$QM, XDMU, XDAD, \$NML, XDDI, XDSU, \$SE(FSE), \$PC, \$QL(FSB), \$QN
EXP	In AB, compute e^{*r} . If there is underflow, $AB = 0$. If overflow, $AB =$ maximum real number and the message FUNC ARG is output. In both cases, overflow = 1.	CALL EXP, r	XDMU, \$QK(FAD), \$NML, \$EE, \$QM, \$QN, \$SE(FSE)
ATAN	In AB, compute arctan r	CALL ATAN, r	\$QM, \$QL(FSB), \$QN, \$QK(FAD), \$SE(FSE)
SINCOS	In AB, compute $\cos r$ with COS, or $\sin r$ with SIN	CALL COS, r CALL SIN, r	\$QK(FAD), \$QL(FSB), \$QM, \$QN, \$SE(FSE)
SQRT	In AB, compute square root of r	CALL SQRT, r	XDDI, \$FSM, \$SE(FSE)
FMULDIV	Given: AB contains r_1 , in AB, compute $r_1^{*r_2}$ with \$QM, or r_1/r_2 with \$QN. If there is underflow, $AB = 0$. If overflow, $AB =$ maximum value and the message ARITH OVFL is output. In both cases, overflow = 1.	CALL \$QM, r_2 CALL \$QN, r_2	XDMU, \$FMS, XDDI, \$SE(FSE), \$EE, \$NML

Table 13-1. DAS Coded Subroutines (continued)

Name	Function	Calling Sequence	External References
FADDSUB	Given: AB contains r_1 , in AB, compute $r_1 + r_2$ with \$QK, or $r_1 - r_2$ with \$QL. If there is underflow, $AB = 0$. If overflow, $AB =$ maximum value and the message ARITH OVFL is output. In both cases, $overflow = 1$.	CALL \$QK,r2 CALL \$QL,r2	\$SE(FSE), \$FSM, \$NML, \$EE
SEPMANTI	Separate mantissa and characteristic of r into AB and X, respectively	CALL \$FMS CALL \$FSM	None
FNORMAL	In AB, normalize r	CALL \$NML	XDCO
XDDIV	In AB, compute f_1/f_2	CALL XDDI,f2	XDSU, XDCO
XDMULT	In AB, compute $f_1 * f_2$	CALL XDMU,f2	XDAD, XDCO
XDADD	In AB, compute $f_1 + f_2$	CALL XDAD,f2	None
XDSUB	In AB, compute $f_1 - f_2$	CALL XDSU,f2	None
XDCOMP	In AB, compute negative of f	CALL XDCO	None
\$FLOAT	In AB, convert the i in A to floating-point and, for \$QS, store result in r	CALL \$PC CALL \$QS,r	\$SE(FSE)
\$IFIX	In A, convert the r in AB to i and, for \$HS, store result in i	CALL \$IC CALL \$HS,i	\$SE(FSE), \$EE
IABS	In A, compute absolute i	CALL IABS,i	\$SE(FSE)
ABS	In AB, compute absolute r	CALL ABS,r	\$SE(FSE)
ISIGN	Set the sign of i_1 , in A, equal to that of i_2	CALL ISIGN,i2	\$SE(FSE)
SIGN	Set the sign of r_1 , in AB, equal to that of r_2	CALL SIGN,r2	\$SE(FSE)
\$HN	Given: A holds i_1 , in A, compute i_1/i_2	CALL \$HN,i2	\$SE(FSE), \$EE
\$HM	Given: A holds i_1 , in A compute $i_1 * i_2$	CALL \$HM,i2	\$SE(FSE), \$EE
DSINCOS	In AC, compute $\sin d$ or $\cos d$	CALL \$DSI,d CALL \$DSIN,d CALL \$DCO,d CALL \$DCOS,d	\$STO,\$DNO, \$ZC, \$ZK, \$ZL, \$SE(FSE), \$ZM, \$ZN, AC \$DLO
DATAN	In AC, compute $\arctan d$	CALL \$DAN CALL DATAN,d	\$DLO, \$STO, \$DAD, \$DSU, IF, \$SE(FSE), AC, \$DMP, \$DDI, POLY

SUPPORT LIBRARY

Table 13-1. DAS Coded Subroutines (continued)

Name	Function	Calling Sequence	External References
DEXP	In AC, compute exponential d	CALL \$DEX CALL DEXP,d	\$DLO, \$STO, \$SE(FSE), AC, \$DNO, \$EE, \$ZC, \$ZK, \$ZL, \$ZM, \$ZN
DLOG	In AC, compute $\ln d$	CALL DLOG,d CALL \$DLN	\$DLO, \$STO, \$DNO, \$EE \$SE(FSE), \$ZK, \$ZL, \$ZM, \$ZN
POLY	In AC, compute double-precision polynomial with t terms, coefficient list starting at address c, and argument at address y	CALL POLY,t,c,y	\$DLO, \$DAD, \$DMP
CHEB	In AC, compute shifted Chebyshev polynomial series with t + 1 terms and coefficient list starting at address c	CALL CHEB,t,c	\$DLO, \$STO, \$DAD, \$DSU, \$DMP
DSQRT	In AC, compute square root of d	CALL \$DSQ,d CALL DSQR,d	\$DLO, \$STO, \$DNO, \$DAD, \$DMP, \$DDI, \$SE(FSE), AC
\$DFR	In AC, compute fractional part of d	CALL \$DFR,d	\$DLO, \$DNO, \$DSU, \$DIT, AC, \$SE(FSE)
IDINT	In AC, compute integral part of d	CALL \$DIT,d CALL IDINT,d	\$DNO, \$SE(FSE)
DMULT	In AC, compute $d1 \cdot d2$	CALL \$DMP,d2 CALL \$ZM,d2	\$DLO, \$STO, \$DNO, \$DAD, AC, \$SE(FSE)
DDIVIDE	In AC, compute $d1/d2$	CALL \$DDI,d2 CALL \$ZN,d2	\$DLO, \$STO, \$DNO, \$DSU, AC, \$SE(FSE)
DADDSUB	In AC, compute $d1 + d2$ with \$DAD, or $d1 - d2$ with \$DSU	CALL \$DAD,d2 CAL \$DSU,d2 CALL \$ZK,d2 CALL \$ZL,d2	\$STO, \$DLO, \$DNO, AC, \$SE(FSE), \$EE
DNORMAL	In AC, normalize d	CALL \$DNO	\$SE(FSE)
DLOADAC	Load AC with d	CALL \$DLO,d CALL \$ZF,d	AC, \$SE(FSE)
DSTOREAC	Store AC in d	CALL \$STO,d CALL \$ZS,d	AC, \$SE(FSE)
RLOADAC	Load A with double-precision mantissa sign word from AC	CALL \$ZI	AC
SINGLE	In AB, convert the d in AC to r	CALL \$RC	AC
DOUBLE	In AC, convert the r in AB to d	CALL \$YC	AC
DBLECOMP	In AC, compute negative of the d in AC	CALL \$ZC	AC
\$3S	Store AB in memory address m	CALL \$3S,m	\$SE(FSE)

Table 13-1. DAS Coded Subroutines (continued)

Name	Function	Calling Sequence	External References and Return Conditions
A2MT	Translate in memory a character string of length <i>n</i> starting at <i>s</i> and ending at <i>e</i> from eight-bit ASCII to six-bit magnetic tape BCD code <i>s</i> is the start of the ASCII block and <i>e</i> is the start of the BCD block.	CALL A2MT, <i>n,s,e</i>	None
MT2A	Translate in memory a character string of length <i>n</i> starting at <i>s</i> and ending at <i>e</i> from six-bit magnetic tape BCD code to eight-bit ASCII <i>s</i> is the start of the BCD block and <i>e</i> is the start of the ASCII block.	CALL MT2A, <i>n,s,e</i>	None
EXIT	Formats and executes an RTE EXIT macro	CALL EXIT	V\$EXEC
SUSPND	Formats and executes an RTE SUSPND macro with parameter <i>i</i> .	CALL SUSPND(<i>i</i>)	V\$EXEC
RESUME	Formats and executes an RTE RESUME macro to resume task <i>s</i> .	CALL RESUME(<i>s</i>)	V\$EXEC, \$RTENM
ABORT	Formats and executes an RTE ABORT macro to abort task <i>s</i> .	CALL ABORT(<i>s</i>)	V\$EXEC, \$RTENM
ALOC	Formats and executes an RTE ALOC macro to call reentrant subroutine <i>s</i> .	CALL ALOC(<i>s</i>)	V\$EXEC
PMSK	Formats and executes an RTE PMSK macro to operate on PIM <i>i1</i> with line mask <i>i2</i> and enable/disable flag <i>i3</i> .	CALL PMSK(<i>i1</i> , <i>i2,i3</i>)	V\$EXEC
DELAY	Formats and executes an RTE DELAY macro with the 5-millisecond count in <i>i1</i> , the minute count in <i>i2</i> , and delay mode in <i>i3</i> .	CALL DELAY(<i>i1</i> , <i>i2,i3</i>)	V\$EXEC
LDELAY	Formats and executes an RTE DELAY type 1 with additional parameters to specify the LUN from which the task (lun in <i>i4</i> key in <i>i5</i>) is to be reloaded.	CALL LDELAY (<i>i1,i2,i3</i> , <i>i4, i5</i>)	V\$EXEC
TIME	Formats and executes an RTE TIME macro with the minute count in <i>i1</i> , and 5-millisecond count in <i>i2</i> .	CALL TIME(<i>i1,i2</i>)	V\$EXEC
OVLAY	Formats and executes an RTE OVLAY macro with <i>i1</i> = 0 to execute, <i>i2</i> = 0 to load, and <i>s</i> is the overlay name.	CALL OVLAY(<i>i1</i> , <i>i2,s</i>)	V\$EXEC, \$RTENM

Table 13-1. DAS Coded Subroutines (continued)

Name	Function	Calling Sequence	External References and Return Conditions
SCHED	Formats and executes an RTE SCHED macro with i1 = priority, i2 = wait flag, i3 = logical-unit number, s1 = key and s2 = task name.	CALL SCHED(i1, i2, i3,s1,s2)	V\$EXEC, \$RTENM
\$RTENM	Moves the six-character name from X to B	CALL \$RTENM	None
\$EE	Outputs error messages on the SQ device.	CALL \$EE	V\$IIOC, V\$IOST, V\$EXEC
\$SE	Fetches n parameters from a subroutine call	CALL \$SE, n BSS n	None
V\$RSW	Handles multi-reel volume files and information.	LDA = LUN to unload. LDX < 0 for no mount. LDX = 0 for mount next volume. LDX > 0 addr. of filename for mount. B = next volume number if X > 0 CALL V\$RSW	A = Restored B = Restored X = Restored
V\$HDR	To format a standard VORTEX header.	CALL V\$HDR DATA page number address DATA program name address DATA program title address (= 0 if not used)	A,B,X restored Header in 38 word external buffer V\$HBUF
\$TC	Converts a DP integer to a real value.	CALL \$TC	none
\$TE	Raises a real value to its DP integer power.	CALL \$TE,i2	none
DIADDSUB	Adds or subtracts double word integers. \$6K is add, \$6L is subtract.	CALL \$6K,i2 CALL \$6L,i2	FSE
DIMULDIV	Multiplies or divides two double word integers. \$6M is multiply, \$6N is divide, \$6NX is divide and return remainder only.	CALL \$6M,i2 CALL \$6N,i2 CALL \$6NX,i2	FSE

Table 13-2. OM Library Subroutines

Name	Function	Calling Sequence	External References
CB2A	Covert a 16-bit binary value (positive or negative) to an ASCII character string (octal or decimal) with leading zeros suppressed and right justified minus sign on negative decimal values.	LDA = 0 for octal conversion = 0 for decimal conversion JSR CB2A,X DATA Address of binary value	(A) = Address of ASCII string (B) = Restored
CA2B	Convert a decimal or octal ASCII number (positive or negative decimal) to a 16-bit binary value.	JSR CA2B,X DATA ASCII string address (compl = left byte, pos = right byte) DATA Address of termination character block	(A) = Binary value (B) = Next byte address OVFL = Set if an illegal character encountered
<p>The termination block format is</p> <p>DATA Legal termination character (right justified)</p> <p>DATA Legal termination character (right justified)</p> <p>.</p> <p>.</p> <p>.</p> <p>DATA 0 (end of block)</p>			
MOVE	Move a block of n words from address f to address t. If an overlap move, then; move in reverse.	JSR MOVE,X DATA n (word count) DATA f (from address) DATA t (to address)	(A) = Restored (B) = Restored
CTIME	Convert the time of day to an ASCII string of the form: HH:MM:SS:III where III is milliseconds	JSR CTIME,X	(A) = Address of ASCII string (B) = Restored

SUPPORT LIBRARY

Table 13-3. FORTRAN IV Coded Subroutines

Name	Function	Calling Sequence	External References
\$9E	Compute $ACCZ^{r \cdot i}$	CALL \$9E(i)	\$SE(FSE), IABS, \$8F, \$8M, \$8N, \$8S
CCOS	In ACCZ, compute $\cos z$	CALL CCOS(z)	\$SE(FSE), CSIN, \$8F, \$8K, \$8S
CSIN	In ACCZ, compute $\sin z$	CALL CSIN(z)	\$SE(FSE), EXP, \$QN, SIN, \$QK(FAD), \$QM, COS, \$QL(FSB), \$8F
CLOG	In ACCZ, compute $\ln z$	CALL CLOG(z)	\$SE(FSE), ALOG, \$QM, \$QK(FAD), \$QN, ATAN2, \$8F
CEXP	In ACCZ, compute exponential z	CALL CEXP(z)	\$SE(FSE), EXP, COS, \$QM, SIN, \$8F
CSQRT	In ACCZ, compute square root of z	CALL CSQRT(z)	\$SE(FSE), SQRT, CABS, \$QK, \$QN, \$8F
CABS	In AB, compute absolute z	CALL CABS(z)	\$SE(FSE), SQRT, \$QM, \$QK(FAD)
CONJG	In ACCZ, compute conjugate of z	CALL CONJG(z)	\$SE(FSE), \$8F
\$AK	Add r to real part of ACCZ	CALL \$AK(r)	\$SE(FSE), \$8S, \$QK(FAD), \$8F
\$AL	Subtract r from the real part of ACCZ	CALL \$AL(r)	\$SE(FSE), \$8S, \$QL(FSB), \$8F
\$AM	Multiply ACCZ by r	CALL \$AM(r)	\$SE(FSE), \$8S, \$QM, \$8F
\$AN	Divide ACCZ by r	CALL \$AN(r)	\$SE(FSE), \$8S, \$QM, \$8F
\$AC	Convert AC to z and store in ACCZ	CALL \$AC	\$3S, CMLPX
CMLPX	Load ACCZ with a value having a real part $r1$ and an imaginary part $r2$	CALL CMLPX(r1,r2)	\$SE(FSE), \$8F
\$8K	Add z to ACCZ	CALL \$8K(z)	\$SE(FSE), \$8S, \$QK(FAD), \$8F
\$8L	Subtract z from ACCZ	CALL \$8L(z)	\$SE(FSE), \$8S, \$QL(FSB), \$8F
\$8M	Multiply ACCZ by z	CALL \$8M(z)	\$SE(FSE), \$8S, \$QM, \$QL(FSB), \$QK(FAD), \$8F

Table 13-3. FORTRAN IV Coded Subroutines (continued)

Name	Function	Calling Sequence	External References
\$8N	Divide ACCZ by z	CALL \$8N(z)	\$SE(FSE), \$8S, \$QM, \$QK(FAD), \$QN, \$QL(FSB), \$8F
\$ZD	Compute negative of z	CALL \$ZD	\$8S, \$8F
AIMAG	Load AB with the imaginary part of z	CALL AIMAG(z)	\$SE(FSE)
\$OC	Load AB with the real part of ACCZ	CALL \$OC	\$8S
REAL	Load AB with the real part of z	CALL REAL(z)	\$SE(FSE)
\$8F	Load ACCZ with z	CALL \$8F(z)	\$SE(FSE)
\$8S	Store ACCZ in z	CALL \$8S(z)	\$SE(FSE), \$3S
\$XE	Compute $d^{\circ}i$ where d is in AC	CALL \$XE(i)	\$SE(FSE), \$ZF, MOD, \$ZM, \$HN, \$ZN, \$ZS
\$YE	Compute $d^{\circ}r$ where d is in AC	CALL \$YE(r)	\$SE(FSE), \$ZS, DBLE, \$ZE, \$ZF
\$ZE	Compute $d1^{\circ}d2$ where d1 is in AC	CALL \$ZE(d2)	\$SE(FSE), \$ZS, DEXP, DLOG, \$ZM
DATAN2	In AC, compute arctan (d1/d2)	CALL DATAN2(d1,d2)	\$SE(FSE), \$ZF, \$ZS, \$ZI, \$ER, \$ZN, \$ZL, \$ZK, DATAN
DLOG10	In AC, compute log d	CALL DLOG10(d)	\$SE(FSE), DLOG, \$ZM
DMOD	In AC, compute d1 modulo d2	CALL DMOD(d1,d2)	\$SE(FSE), DINT, \$ZF, \$ZN, \$ZS, \$ZM, \$ZL, \$ZC
DINT	In AC, compute integer portion of d	CALL DINT(d)	\$SE(FSE), \$ZF, \$JC, \$XC
DABS	In AC, compute absolute d	CALL DABS(d)	\$SE(FSE), \$ZF, \$ZI, \$ZC
DMAX1	In AC, select the maximum value in the set d1, d2, ..., dn	CALL DMAX1(d1,d2 ...,dn,0)	\$SE(FSE), \$ZF, \$ZS, I\$FA, \$ZL, \$ZI
DMIN1	In AC, select the minimum value in the set d1, d2, ..., dn	CALL DMIN1(d1,d2 ...,dn,0)	\$SE(FSE), \$ZF, \$ZS, I\$FA, \$ZL, \$ZI
DSIGN	Set the sign of d1 equal to that of d2	CALL DSIGN(d1,d2)	\$SE(FSE), \$ZF, \$ZI, \$ZN
\$YK	Add r to AC	CALL \$YK(r)	\$SE(FSE), \$ZS, DBLE, \$ZK
\$YL	Subtract r from AC	CALL \$YL(r)	\$SE(FSE), \$ZS, DBLE, \$ZL, \$ZC
\$YM	Multiply AC by r	CALL \$YM(r)	\$SE(FSE), \$ZS, DBLE, \$ZM

Table 13-3. FORTRAN IV Coded Subroutines (continued)

Name	Function	Calling Sequence	External References
\$YN	Divide AC by r	CALL \$YN(r)	\$SE(FSE), \$ZS, DBLE, \$ZF, \$ZN
DBLE	In AC, convert r to d	CALL DBLE(r)	\$SE(FSE), \$YC
\$XC	In AC, convert i to d where i is in A	CALL \$XC	\$PC, \$YC
TANH	In AB, compute tanh r	CALL TANH(r)	\$SE(FSE), \$QK(FAD), EXP, \$QL(FSB), \$QN
ATAN2	In AB, compute arctan (r1/r2)	CALL ATAN2(r1,r2)	\$SE(FSE), \$ER, ATAN, \$QK(FAD), \$QL(FSB), \$QN
ALOG10	In AB, compute log r	CALL ALOG10(r)	\$SE(FSE), ALOG, \$QM
AMOD	In AB, compute r1 modulo r2	CALL AMOD(r1,r2)	\$SE(FSE), AINT, \$QN, \$QM, \$QL(FSB)
AINT	In AB, truncate r	CALL AINT(r)	\$SE(FSE), \$IC, \$PC
AMAX1	In AB, select the maximum value in the set r1,r2,...,rn	CALL AMAX1(r1,r2, ...,rn,0)	\$SE(FSE), I\$FA, \$QL(FSB)
AMIN1	In AB, select the minimum value in the set r1, r2,...,rn	CALL AMIN1(r1,r2, ...,rn,0)	\$SE(FSE), I\$FA, \$QL(FSB)
AMAX0	In AB, select the maximum value in the set i1,i2,...,in and convert to r	CALL AMAX0(i1,i2, ...,in,0)	\$SE(FSE), I\$FA, FLOAT
AMIN0	In AB, select the minimum value in the set i1,i2,...,in and convert to r	CALL AMIN0(i1,i2, ...,in,0)	\$SE(FSE), I\$FA, FLOAT
DIM	In AB, compute the positive difference between r1 and r2	CALL DIM(r1,r2)	\$SE(FSE), \$QL(FSB)
FLOAT	In AB, convert i to r	CALL FLOAT(i)	\$SE(FSE), \$PC
SNGL	In AB, convert d to r	CALL SNGL(d)	\$SE(FSE), \$ZF, \$RC
MAX0	In A, select the maximum value in the set i1,i2,...,in	CALL MAX0(i1,i2, ...,in,0)	\$SE(FSE), I\$FA
MIN0	In A, select the minimum value in the set i1,i2,...,in	CALL MIN0(i1,i2, ...,in,0)	\$SE(FSE), I\$FA
MAX1	In A, select the maximum value in the set r1,r2,...,rn and convert to i	CALL MAX1(r1,r2, ...,rn,0)	\$SE(FSE), I\$FA, \$QL(FSB), IFIX
MIN1	In A, select the minimum value in the set r1,r2,...,rn and convert to i	CALL MIN1(r1,r2, ...,rn,0)	\$SE(FSE), I\$FA, \$QL(FSB), IFIX
MOD	In A, compute i1 modulo i2	CALL MOD(i1,i2)	\$SE(FSE), \$HN, \$HM

Table 13-3. FORTRAN IV Coded Subroutines (continued)

Name	Function	Calling Sequence	External References
INT	In A, truncate r and convert to i	CALL INT(r)	\$SE(FSE), \$IC
IDIM	In A, compute the positive difference between i1 and i2	CALL IDIM(i1,i2)	\$SE(FSE)
IFIX	In A, convert r to i	CALL IFIX(r)	\$SE(FSE), \$IC
\$JC	In AC, convert d to i and store result in A	CALL \$JC	\$RC, \$IC
IOR	In A, compute the inclusive or of m and n (m.OR.n)	CALL IOR(m,n)	\$SE
IAND	In A, compute the logical product of m and n (m.AND.n)	CALL IAND(m,n)	\$SE
NOT	In A, compute the logical inversion of m (.NOT.m)	CALL NOT(m)	\$SE
IEOR	In A, compute the exclusive or (logical difference) of m and n (m.XOR.n)	CALL IEO(m,n)	\$SE
ISHFT	In A, compute either zero, if the absolute value of n is greater than or equal to 16, or $m \cdot 2^{**}n$, otherwise (negative n value indicates a right shift, positive n value indicates a left shift)	CALL ISHFT(m,n)	\$SE



Table 13-3. FORTRAN IV Coded Subroutines (continued)

Name	Function	Calling Sequence	External References
INT	In A, truncate r and convert to i	CALL INT(r)	\$SE(FSE), \$IC
IDIM	In A, compute the positive difference between i1 and i2	CALL IDIM(i1,i2)	\$SE(FSE)
IFIX	In A, convert r to i	CALL IFIX(r)	\$SE(FSE), \$IC
\$JC	In AC, convert d to i and store result in A	CALL \$JC	\$RC, \$IC

13.4 DECIMAL SUBROUTINE

The decimal subroutine performs requested decimal operations (add, subtract, multiply, divide, move, or compare). Besides operand addresses and sizes, the user may specify pre-shifting of operands and post-shifting and rounding of result. Note that pre-shifting is decimal alignment and does not imply physical shifting. Operands may be signed or unsigned.

Decimal compare sets the user result condition word as follows:

= 0	if operand A < operand B
= 1	if operand A = operand B
= 2	if operand A > operand B

Decimal compare arithmetically compares two decimal operands.

On entry register R0(A) contains the address of an 85 word temporary storage block available to firmware, R1(B) contains the address of the user result condition word, and R2(X) contains the address of the users descriptive parameter block. Decimal math may be accessed either via

JMPM **V\$DECM**

or

JMP **C\$DECM**

If C\$DECM is used, return will be made to user supplied location VC\$RTN. If V\$DECM is used, the user must still define VC\$RTN.

Parameter Block

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 0	CODE			P	LA	UB	LA			LB						
1	BN _A				displacement _A											
2	BN _B				displacement _B											
3				Q	DA	DB	SA			SB						
4				R	UC	DC	LC			SC						
5	BN _C				displacement _C											

SUPPORT LIBRARY

Parameter Description:

CODE represents operation to be performed:

- 0 = opA + opB
- 1 = opA - opB
- 2 = compare opA: opB
- 3 = move opA to opB
- 4 = opA * opB
- 5 = opA/opB

P = 1 for presence of word 3.
 = 0 for absence of word 3.

UA = 1 if operand A is unsigned.
 = 0 if operand A is signed.

UB = 1 if operand B is unsigned.
 = 0 if operand B is signed.

LA = length of A in digits (1 to 31).

LB = length of B in digits (1 to 31).

BN_A = main storage base register number of operand A.

BN_B = main storage register number of operand B.

Q = 1 if returned in third operand (words 4 and 5 present).
 = 0 if third operand not present (words 4 and 5 absent).

DA = 1 pre shift operand A left
 = 0 pre shift operand A right

DB = 1 pre shift operand B left
 = 0 pre shift operand B right

SA = Operand A shift amount

SB = Operand B shift amount

R = 1 if rounding to be applied to result (only if result returned in third operand)
 = 0 if rounding not applied to result

UC = 2 if result unsigned
 = 0 if result signed

DC = 1 to shift result left
 = 0 to shift result right

LC = length of result field

SC = result shift amount

BN_C = main storage base register number of result

Displacement A, B, or C = Byte count used to calculate byte address of decimal operands.

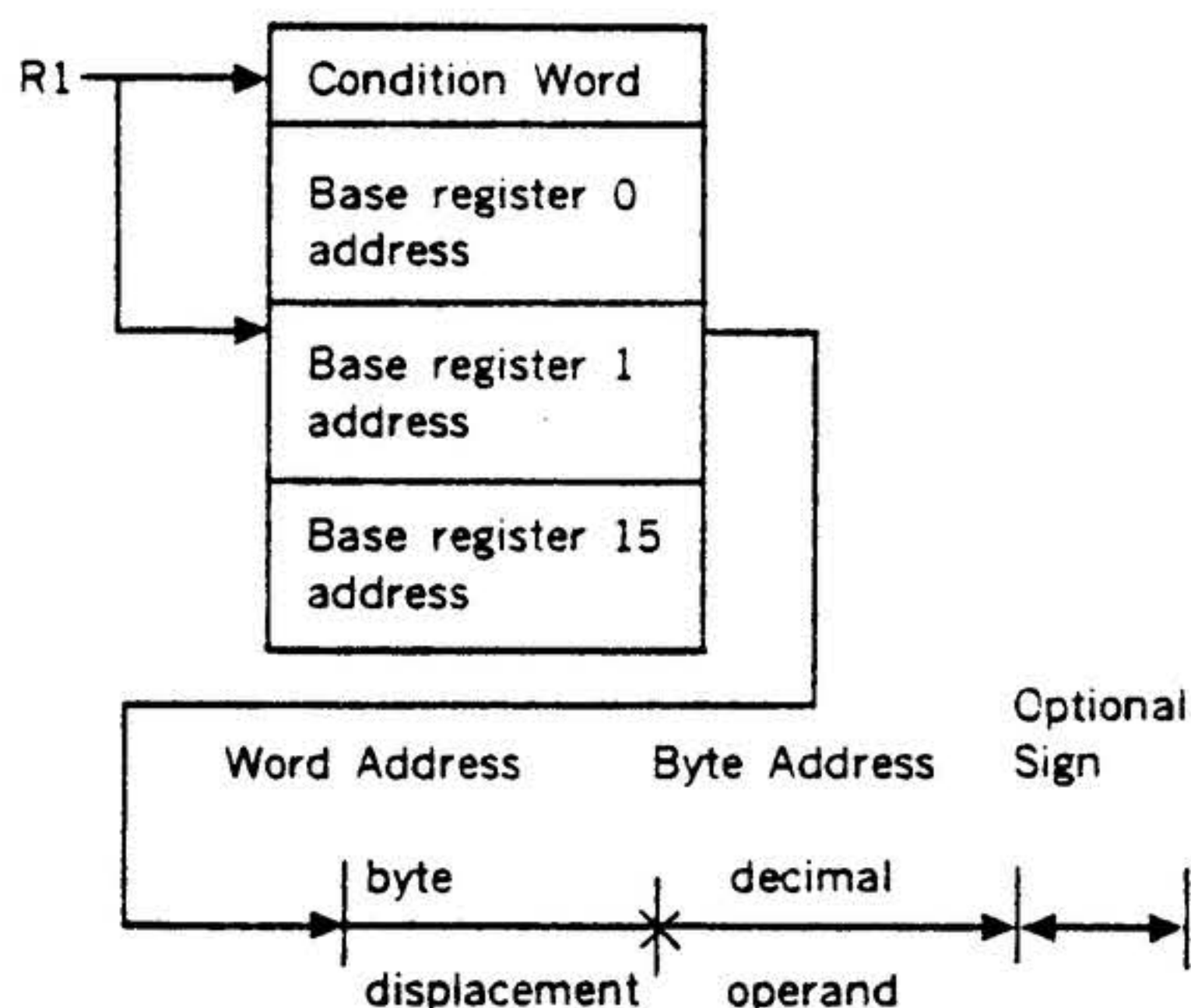
- b. Invalid digit - if the number portion of a digit (bits $2^3 - 2^0$) contains a value other than $0 - 9_{10}$ or the zone portion (bits $2^7 - 2^4$) contains a value other than 11_{10} , the conditions result word will be set to the value 4. This is also true of values specified as signed having signs other than blank (octal 240), minus (octal 255), or plus (octal 253).
- c. If the base word related to respective BN field is zero then the condition result word CONDIT will be set to 5.
- d. Attempted division by zero results in CONDIT being set to 3.

Notes

If operand C is not specified, the result will be returned in operand A, except for move. Decimal move moves operand A to operand B. Note that for a decimal move, the parameter block may be a maximum of 4 words. In this case, the Q bit is used to specify rounding, rather than a third operand.

Parameter byte addresses are calculated as follows: $(R1 + 1 + BN) * 2 + displacement = \text{byte address of least significant byte of decimal operand.}$

This represented pictorially as follows:



When pre-shifting is specified, this does not imply physical shifting of operands. Only the operand designated for result is modified by a decimal operation.

When the operation is complete, only the integrity of register R2 and R1 are maintained. R2 will be incremented to the address of the next word following the parameter block.

This is meant to imply all other V75 registers are volatile. The user must save and restore any registers R3 through R7 he requires to be maintained when executing the decimal operation.

Error Conditions:

(Note that on an error, register R2 will be incremented past the parameter block, and results will be unreliable.)

- a. Result operand overflow - if the result operand has an inadequate number of digits to contain the result, the condition result word (CONDIT) will be set to the value 3.

Examples:

Note: The following may be used to create decimal parameter blocks:

FOLLOWING ARE FORMS OF DECIMAL INSTRUCTION.

DWORD0 FORM 3, 1, 11, 5, 5
 DWORD1 FORM 4, 12
 DWORD2 FORM 4, 12
 DWORD3 FORM 3, 1, 1, 1, 5, 5
 DWORD4 FORM 3, 1, 1, 1, 5, 5
 DWORDS5 FORM 4, 12

DECIMAL OPERATION MACRO (DECIMAL PARAMETER BLOCK)

DECOP	MAC		
	IFT	P(12)-P(13)-P(5)-P(6)+P(14)	Select appropriate Word 0
	GOTO	DECWD1	(Note no third, fourth,
	DWORD0	P(7), 0, P(1), P(3), P(4), P(11)	or fifth word)
	GOTO	DECWD2	
DECWD1	COUNT		
	DWORD0	P(7), 1, P(1), P(8), P(4), P(11)	(Parameter block includes
DECWD2	CONT		at least word 3)
	DWORD1	P(2), P(3)	
	DWORD2	P(9), P(10)	
	IFF	P(12)+P(13)+P(5)+P(6)+P(14)	
	GOTO	DECWD3	(Terminate if no word 3)
	DWORD3	0, P(14)P(5), P(12), P(6), P(13)	
	IFF	P(14)	
	GOTO	DECWD3	(Terminate if no third
	DWORD4	0, P(15), P(16), P(20), P(19), P(21)	operand words 4 and 5)
	DWORDS5	P(17), P(18)	
DECWD3	CONT		
	EMAC		

INTERPRETIVE PARAMETER BLOCK DEFINED AS FOLLOWS:

P(01)	OP1	SIGNED (S) OR UNSIGNED (U)
P(02)	OP1	REG
P(03)	OP1	DISPLACEMENT
P(04)	OP1	LENGTH
P(05)	OP1	SHIFT LEFT (L) OR RIGHT (R)
P(06)	OP1	SHIFT AMOUNT
P(07)		OPERATION (DADD, DSUB, SMULL, DDIV, DMOV, DCMP)
P(08)	OP2	SIGNED (S) OR UNSIGNED (U)
P(09)	OP2	REG
P(10)	OP2	DISPLACEMENT
P(11)	OP2	LENGTH
P(12)	OP2	SHIFT LEFT (L) OR RIGHT (R)
P(13)	OP2	SHIFT AMOUNT
P(14)	=EQ	IF RESULT IN THIRD OPERAND
P(15)	F	FOR ROUNDING
P(16)	OP3	SIGNED (S) OR UNSIGNED (U)
P(17)	OP3	REG
P(18)	OP3	DISPLACEMENT
P(19)	OP3	LENGTH
P(20)	OP3	SHIFT LEFT (L) OR RIGHT (R)
P(21)	OP3	SHIFT AMOUNT

SUPPORT LIBRARY

Following are equates to be used with the above macro:

BN0	EQU	0	BASE NUMBER 0
BN1	EQU	1	BASE NUMBER 1
BN2	EQU	2	BASE NUMBER 2
BN3	EQU	3	BASE NUMBER 3
BN4	EQU	4	BASE NUMBER 4
BN5	EQU	5	BASE NUMBER 5
BN6	EQU	6	BASE NUMBER 6
BN7	EQU	7	BASE NUMBER 7
BN8	EQU	8	BASE NUMBER 8
BN9	EQU	9	BASE NUMBER 9
BNA	EQU	10	BASE NUMBER 10
BNB	EQU	11	BASE NUMBER 11
BNC	EQU	12	BASE NUMBER 12
BND	EQU	13	BASE NUMBER 13
BNE	EQU	14	BASE NUMBER 14
BNF	EQU	15	BASE NUMBER 15
DADD	EQU	0	DECIMAL ADD
DSUB	EQU	1	DECIMAL SUBTRACT
DCMP	EQU	2	DECIMAL COMPARE
DMOV	EQU	3	DECIMAL MOVE
DMUL	EQU	4	DECIMAL MULTIPLY
DDIV	EQU	5	DECIMAL DIVIDE
EQ	EQU	1	RESULT RETURNED IN C
F	EQU	1	ROUND (ADJUST)
R	EQU	0	SHIFT RIGHT
L	EQU	1	SHIFT LEFT
S	EQU	0	SIGNED
U	EQU	1	UNSIGNED

The above macro may be used as follows:

1. DECOP U, BN1, 2, 4, R, 1, DAD, U, BN2, 0, 4, L 1

generates four word parameter block

16204
10002
20000
02041

Explanation: Operand A is an unsigned decimal string residing in memory accumulator 1. It begins (most significant digit) two bytes into accumulator 1 with a length of four bytes. Operand A will be logically reshifted right one digit. Operand B is an unsigned decimal string beginning in memory accumulator 2 with a length of four bytes. Operand B will be logically pre-shifted left one digit. The result of addition will be returned in operand A. If operand A = 4310 and operand B = 0129, result of the above operation would be 1721.

Note following register settings:

	Before Operation	After Operation
R0(A)	1016	1016
R1(B)	3100	3100
R2(X)	4102	4106

2. DECOP U, BN5, 0, 4, , , DMUL, S, BNE, 0, 3, , ,
EQ, F, U, BN1, 0, 7, R, 1

generates six word parameter block

114203
050000
160000
010000
014341
010000

Explanation: An unsigned 4 digit decimal string in memory accumulator 5 is multiplied by a signed 3 digit decimal string in memory accumulator 14. The result will be right shifted one digit position, rounded, and stored in memory accumulator 1 (note maximum resulting digit string length is 7). If operand A = 0321 and operand B = 987 + result of above operation would be 0003168.

Note following register settings:

	Before Operation	After Operation
R0(A)	1200	1200
R1(B)	1105	1105
R2(X)	3506	3514

3. DECOP S, BNC, 0, 3, , , DCMP, S, BN1, 0, 4

generates three word parameter block

040144
150000
010000

Example 3 compares decimal digit string in memory accumulator D with decimal digit string in memory accumulator 1. If operand A = 123 + and operand B = 9871-, condition word pointed to by R1(B) would be set to 20.

Note following register settings:

	Before Operation	After Operation
R0(A)	13012	13012
R1(B)	6512	6512
R2(X)	1234	1237

SECTION 14

REAL-TIME PROGRAMMING

VORTEX real-time applications allow the user to interface directly with special devices, develop software that is interrupt-driven, and utilize reentrant subroutines. Four areas are covered in this section:

- Interrupts
- Task-scheduling
- Coding reentrant subroutines
- Coding I/O drivers

14.1 INTERRUPTS

14.1.1 External Interrupts

Priority interrupt module (PIM) hardware: A PIM comprises a group of eight interrupt lines and an eight-bit register. The register holds a mask where each set bit disarms a line. VORTEX allows up to eight PIMs for a maximum of 64 lines. The system of PIMs and lines is called the *external interrupt system*.

The processing of external interrupts is controlled by the programmed status of the line. The lines are continuously hardware-scanned, regardless of the status.

If more than one interrupt is detected on a single scan, the highest-priority line is acknowledged, and, if the PIM is enabled and the line armed, the interrupt is taken. If no conflict occurs, the lines are acknowledged on a first-in/first-out basis. If a signal is received on a disabled PIM, it is stored by the PIM, and causes an interrupt when the PIM is enabled.

Disabling the external interrupt system prevents any interrupt from entering the computer. Enabling the entire system allows acknowledgement of all interrupts. Enable/disable selection on a PIM basis allows for more selected control of the system. Individual line selection prevents receiving a second interrupt while a line is still processing the first.

Program setting of PIM registers causes the PIM to ignore interrupts received on lines that are busy processing an interrupt or held off because of priority.

All PIMs and interrupt lines to be used in VORTEX are specified at system-generation time and their status specified when VORTEX is loaded and initialized. VORTEX does not disable any line unless so directed by RTE service request PMSK (section 2.1.6).

When a PIM interrupt signal is acknowledged and the interrupt taken, the computer executes the instruction in a

selected memory location. Under VORTEX, PIM addresses are from 0100 to 0277. Linkage to VORTEX interrupt-processing routines is accomplished by a jump-and-mark instruction in the interrupt location. Unspecified lines are preset in VORTEX with no-operation instructions that ignore unspecified or spurious interrupts.

Since VORTEX always includes memory protection, certain instruction sequences cannot be interrupted and acknowledgement is delayed until they are complete. These include the instruction following an external control, halt, execution, or any instruction manually executed in step mode.

VORTEX interrupt line handlers: At system-generation time, a user specifies all interrupt-driver tasks. These include those that allow VORTEX to service the interrupt, as well as those that are directly connected and service the interrupt themselves. Then, VORTEX constructs a line-handler for each interrupt in the system (figure 14.1).

Directly connected routines preempt VORTEX and are thus used only when response time demands it. Section 14.4.5 describes directly connected interrupt handlers in detail.

Common interrupt handler: The common interrupt handler is the interface between PIM interrupts (via the line handlers) and system or user interrupt-processing tasks. Upon entry, the contents of the volatile registers are saved and the interrupt event word is inclusively ORed into the event word of the specified TIDB. A check then determines whether to return to the interrupted task or to enter the interrupt-processing task, depending upon priority. All interrupts are enabled upon leaving the common interrupt handler.

Interrupt-processing tasks: A task is activated by an interrupt when: (1) task's TIDB interrupt-expected status bit is set, (2) the interrupt event word contains a nonzero, and (3) the task is suspended.

The interrupt-processing task can be memory-resident or RMD-resident. In either case, the processing task clears the event word. The event word distinguishes different interrupt lines that could activate the same task. The dispatcher clears the interrupt expected bit and time delay activity for all tasks except TTY and CRT drivers.

An interrupt-processing task can exit with one of the following options:

- a. Issue a suspend RTE (type 1 or 2) service call that suspends the task and sets the interrupt-expected status bit. Upon receiving the external interrupt or simulated interrupt (TBEVNT word in TIDB is set to 1) caused by IOC or I/O completion events (type 2 only), the task continues execution following the request.

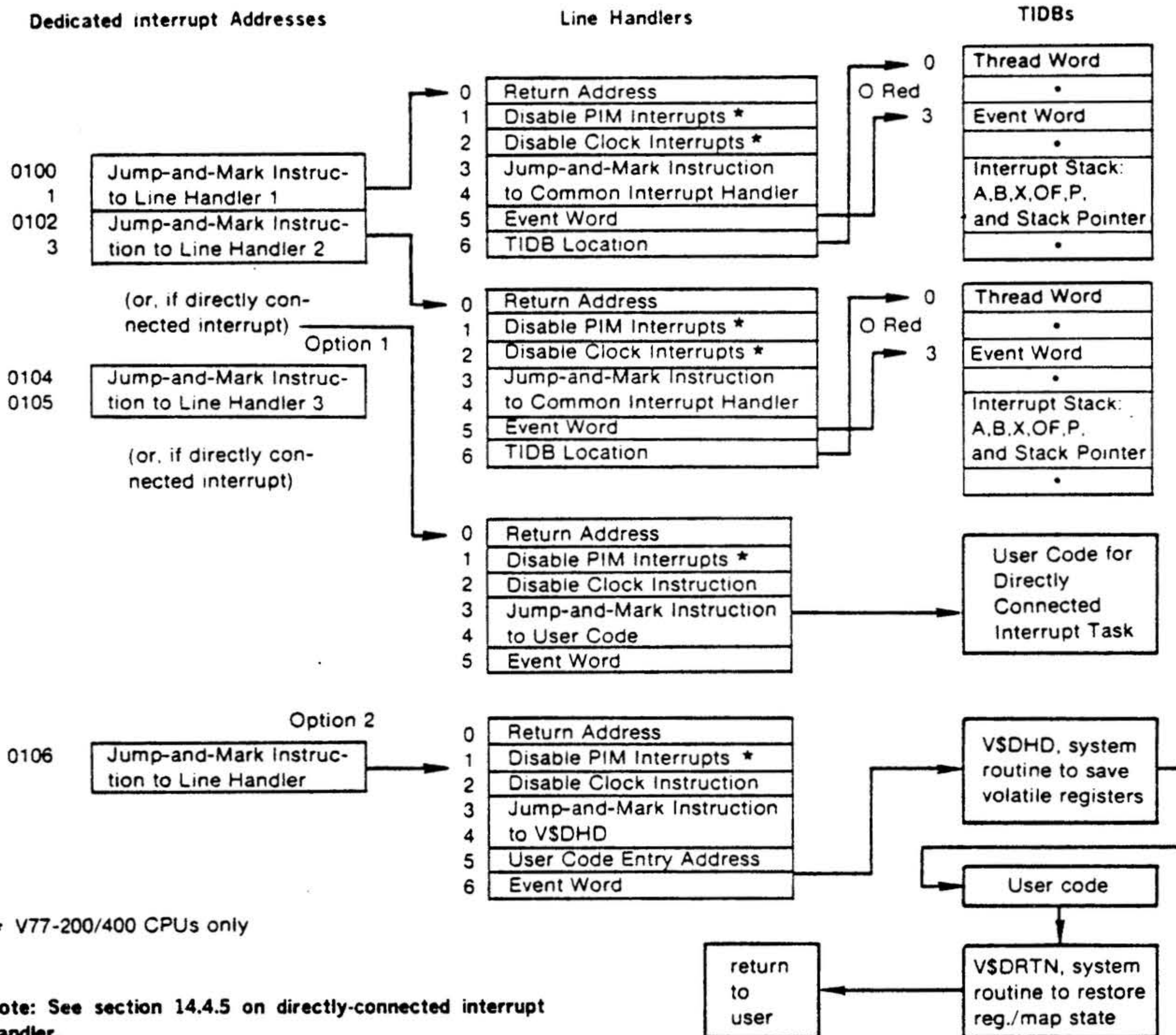


Figure 14-1. Interrupt Line Handlers

b. Issue a delay RTE (type 2 or 3) service call that suspends the task and sets the interrupt-expected and time-delay active status bits. The task is reactivated when time-delay expires or upon receipt of external interrupt or a simulated interrupt caused by IOC or I/O completions (type 3 only). Upon entry, the event word non-zero indicates interrupt activation by external or simulated interrupt (1). Since IOC set the TIDB event word to a 1, the event word in line handlers for external interrupts should be set to something other than 1 if a type 3 delay is to be used. The word also clears the time-delay status bit upon reactivation. It should also be noted that for suspend (type 2) and delay (type 3) service calls, bit 6 of TBPL word of task's TIDB is set to cause IOC to set TBEVNT word to 1 on I/O completion events. This bit is reset whenever a suspend or delay service call of a type other than the ones mentioned above.

c. If RMD-resident, set the interrupt-expected status bit and call EXIT to release space. (TIDB must be resident.)

Timing Considerations: The time necessary to process an interrupt through the common interrupt handler depends on when the interrupt occurred:

- If a task is interrupted and the interrupt-processing task has a lower priority, the interrupt is posted, and VORTEX returns control to the interrupted task in approximately 56 cycles.
- If a task is interrupted and the interrupt-processing task has a higher priority, the interrupt is posted, and VORTEX transfers control to the dispatcher (section 14.2.3) to start the higher-priority interrupt-processing task (if all its conditions are met). The posting time is 66 cycles, approximately.

- c. If an interrupt occurs during a dispatcher scan, the posting time is about 32 cycles. VORTEX returns to the dispatcher to restart the scan.
- d. If the real-time clock interrupts the interrupt handler, the RTC interrupt handler posts the interrupt and the common interrupt handler returns to the clock processor in approximately 40 cycles.

14.1.2 Internal Interrupts

VORTEX recognizes and services internal interrupts related to various hardware components. The processing routines are all directly connected and are the highest-priority tasks in the system.

Memory protection interrupt: Memory protection interrupts are generated when a task attempts to execute a privileged instruction such as external control or halt, or attempts to violate the access mode. The memory protection routines process all protection violation interrupts which are the highest priority interrupts in the system. When the interrupt occurs, the system is forced to the executive mode, state 0 (see table 1-1). Section 1.3 describes the memory map concept and the access modes which can be assigned to each virtual page.

VORTEX uses the memory protection interrupt for switching from the user mode to the executive mode when an I/O (section 3) or RTE (section 2) request is made.

The memory protection interrupt addresses for the various violations are shown in table 14-1.

Table 14-1. Memory Protection Interrupt Addresses

Error	Interrupt Address	Map Active Access Control Status
HALT	020	Attempt was made to execute HALT instruction.
I/O	022	A map number other than 0 attempted to execute an I/O instruction.
WRITE	024	Attempt was made to write into read-only or execute-only location.
JUMP	026	Attempt was made to jump into read operand only location.
UNASSIGNED	030	Attempt was made to read or write into unassigned location.
INSTRUCTION FETCH	032	Attempt was made to fetch instruction from read operand only location.

Power failure/restart interrupt: An interrupt occurs when the system detects a power failure. The VORTEX power failure processor saves the contents of volatile registers and the status of the overflow indicator, sets a power failure flag, and halts with the I register set to 077.

Following the power-up sequence, the PF/R hardware generates an interrupt. Upon entry to the VORTEX power-up processor, the power-failure flag is checked. A power-down sequence must have occurred or else a fatal error condition is assumed to have occurred and VORTEX halts with the I register set to 077.

Hooks are provided to the user to supply additional power failure/restart functions. These hooks consist of user supplied (in place of provided dummy routines) nucleus routines. The power-down routine entry must be V\$PFDN and the power-up routine entry must be V\$PFUP. Both routines must be callable by a JSR, by X register instructions, and should save and restore registers.

If a power-down sequence had occurred, the power-failure flag is cleared, the PIM mask registers are set, the real-time clock's variable interrupt interval is set, the saved volatile registers are restored, the clock and PIMs are enabled (if enabled upon interrupt), and control is returned to the location before the interrupt. Any input or output data transfers in operation at the time of the power failure result in the loss of data.

For peripheral devices such as magnetic tapes and RMDs, the I/O operation is automatically retried.

For other peripheral devices, such as the card reader, paper-tape system, card punch and lineprinter, a retry is not attempted.

The error message posted depends upon the error detected by the respective I/O driver, such as abnormal BIC stop, parity error, interrupt time-out, etc. Data losses on the RMD due to power failure could cause VORTEX to malfunction, but other devices which are not system-resident are recoverable.

The power failure-restart routines operate at the second-highest priority level in the system, which has memory protection at the highest priority level.

The power-up routine reloads the volatile memory map registers by scanning the TIDB thread and outputting the map image for each task which has an assigned, non-checkpointed map. Each task's map key number is contained in TBKEY and the map image address contained in TBMING.

The power-up routine also automatically reloads the writable control store for systems with WCS. Sections 20.1.3 and 20.1.4 describe the manner in which the microutility task saves the WCS image in the OM library file named WCSIMG and how the WCS reload task, WCSRLD, utilizes the file to restore the WCS content. The power-up routine checks location 017 to determine if WCS has been

REAL-TIME PROGRAMMING

loaded. A zero value indicates no WCS. A non-zero value is assumed to be the WCSRLD TIDB address. The FL library logical unit number and protect key are stored in TBRSTS and the WCSRLD TIDB (resident TIDB, non-resident task) is set active.

Real-time clock interrupt: The real-time clock interrupt provides the basis for timekeeping in VORTEX. It can be set to a minimum resolution of 5 milliseconds. However, a value greater than 5 milliseconds (i.e., 10-20 milliseconds) reduces overhead when the system does not have high-resolution timekeeping requirements. Upon receipt of an interrupt, the time-of-day is updated and the TIDBs are scanned for any time-driven task requiring activation. PIMs are disabled for approximately 18 cycles during real-time clock interrupt-processing. The clock routine is the third-highest priority interrupt in VORTEX.

14.1.3 Interrupt-Processing Task Installation

To install an interrupt-processing task that is not directly connected, at system-generation time provide line handlers and resident TIDBs by using a PIM directive (section 15.5.11) with `s(n)` zero and a TDF directive (section 15.6.2) using the same task name in both directives. Additional dummy TIDBs can be added during system generation. (Once a TIDB is in the system, OPCOM directive `;ATTACH` can be used to connect different interrupt-processing tasks to an interrupt line.)

Then, code the interrupt-processing task and add the task via system generation to the VORTEX nucleus as a resident task.

Then, use the `;ATTACH` directive to link the resident task to the interrupt line (if PIM directive not used).

14.1.4 Interrupt State

When a memory-protection, real-time (RT) clock or PIM interrupt occurs, the system is forced to the executive mode, state 0. The interrupts are enabled or disabled as follows:

- a. Memory-Protection Interrupt
 1. RT clock is unaffected and remains in the enabled state.
 2. Memory protection is disabled and is enabled prior to exiting the memory-protection processing routine (EXC 0646).
 3. PIMs are disabled when the JMPM instruction is executed and PIMs are enabled prior to exiting (EXC 0244).

- b. PIM Interrupt
 1. RT clock is unaffected and remains in the enabled state. The common interrupt line handler routine disables and enables the RT clock. The clock is not enabled if the PIM interrupted out of the RT clock processor (see section 14.4.5 for directly connected interrupt handlers).
 2. Memory protection is unaffected and remains in the enabled state.
 3. PIMs are disabled when the JMPM instruction is executed. The common interrupt line handler routine enables the PIMs upon exiting.
- c. RT Clock Interrupt
 1. The RT clock processor disables and reenables the RT clock.
 2. Memory protection is unaffected and remains in the enabled state.
 3. The PIMs are disabled when the JMPM instruction is executed. The RT clock processor enables the PIMs.

14.2 SCHEDULING

14.2.1 System Flow

VORTEX is designed around the TIDB (table 14-1). This block contains all of the information about a task during its execution. The setting and clearing of status bits in the TIDB causes a task to flow through the system. Two register stacks are saved within the TIDB: a reentrant (suspend register) stack, and an interrupt stack.

The dispatcher (section 14.3) is the prime mover of tasks through the system. When any function has reached a termination point or has to wait for an I/O operation, the task gives control to the dispatcher, which then finds another task to execute. A task maintains control until it gives control to the dispatcher, or to the interrupt task if the interrupt-processing task has a higher priority. The contents of the interrupted task's volatile registers are saved in its TIDB interrupt stack and control goes to the dispatcher, which searches for the highest-priority active task for execution.

Each TIDB is placed in sequence by priority level and threaded. Two stacks are maintained in the system: a busy stack and an unused stack. When a task is scheduled for execution, a TIDB is allocated from the unused stack and threaded onto the busy stack according to priority level.

The status word of each TIDB, starting with the highest-priority task, is scanned. Depending upon the setting of status bits, the task is activated, passed over, or made to activate a related system task.

Two resident system tasks are activated by the dispatcher to process functions relating to the execution of a task: (1) search, allocate, and load (SAL), and (2) common system errors (ERROR). SAL searches, allocates,

loads, and exits a scheduled task. ERROR posts common system error messages. These two tasks are not reentered once they start execution, so the dispatcher holds tasks requiring identical functions until they are completed. Then, the highest-priority waiting task is given control of the required function.

In VORTEX, SAL assigns a map (1-15) to each non-resident task scheduled to be executed. If a map is not available, SAL: (1) checkpoints any executing background task's map (memory is checkpointed as required only); (2) checkpoints a lower priority foreground task's map; or (3) checkpoints a higher priority foreground task's map (if TBST bit 8 is set); or (4) exits and does not execute the task until a map becomes available.

Each map defines a logical memory space of 32K words which is segmented into 512-word pages (see section 1.3). SAL sets each logical page to one of four access modes: unassigned, read only, read operand only, or read-write. Each logical page which is assigned an access mode other than unassigned is linked to a physical page of memory. If the access mode is violated by the executing task, a memory protect interrupt occurs. The memory protection interrupt processing is described in section 14.1.2. Page 0 (logical addresses 0-0777) is always assigned to physical page 0, which is the system data region as defined in table 14-1.

Each task, foreground or background, executes within its own logical memory space. The amount of logical memory space available to a task is reduced by: (1) page 0 for system data; and (2) the VORTEX nucleus module accessed by the task and mapped into its logical memory (see section 2.2). If none of the VORTEX nucleus module is accessed, the task has available all but one page (page 0) of the 32K logical memory space. Each task is loaded and executed from logical address 01000. Section 1.3 describes in greater detail available logical memory space.

SAL allocates physical memory by pages. SAL maintains a table designating the allocatability of each physical page within the system as defined during system generation.

If space is not available and the background is in operation, the background task is checkpointed on the RMD checkpoint file and its space allocated to foreground. Upon release of this space by the foreground tasks, the background is read in from the RMD and reactivated.

If space is required to load a program and the background has already been checkpointed, the task waits for a currently running task to exit and release memory.

A task may dynamically request more memory space via the ALOCPG and MAPIN RTE requests. Sections 2.1.15 and 2.1.17 further describe these RTE requests.

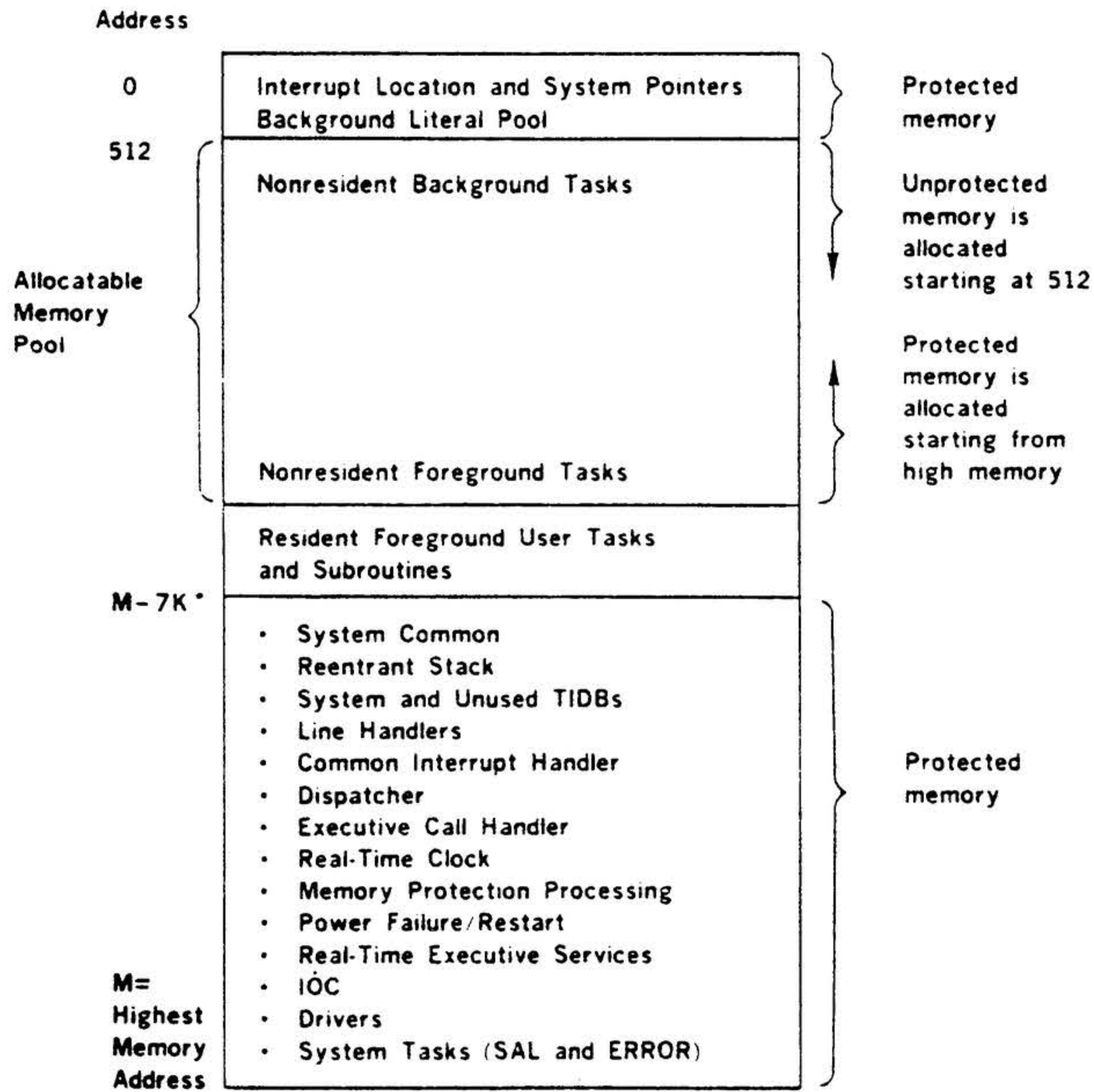
The background memory allocation depends on the size of the background task being loaded. Only the amount needed is so allocated automatically, although the JCP/MEN directive can allocate extra memory for a background task. Figure 14-2 is a VORTEX memory map of map 0, figure 14-3 shows the priority structure, table 14-2 is a description of a TIDB, and table 14-3 is a detailed description of lower memory.

14.2.2 Priorities

Thirty-two priority levels (0 through 31) are provided in the VORTEX system. Levels 2 to 31 are reserved for protected foreground usage. Level 26 is reserved for SAL2. Level 25 is reserved for the two VORTEX system tasks, SAL and ERROR. Levels 24 and 23 are reserved for I/O drivers. All other foreground levels are available to the user. More than one task per level can be scheduled.

Levels 1 and 0 are reserved for tasks running in the background allocatable memory and residing in the background library. Level 1 is reserved for VORTEX system protected tasks, e.g., the job-control processor, the load-module generator, the FORTRAN compiler, the DAS MR assembler, etc. These tasks run with memory protection disabled and can be checkpointed when their space is needed by a foreground task. Level 0 tasks cannot modify or destroy the system (figure 14-3).

Only one background task can be active and in memory at any given time. If other background tasks have been scheduled, the active background task must execute an EXIT service request before the scheduled task(s) can be loaded and executed. If a background task calls EXIT and no tasks are scheduled for the background area, and the requesting task is not the job-control processor, the JCP is scheduled. Otherwise, there is a normal exit.



If a configuration increases memory, the allocatable memory pool would increase and resident routines would reside in a higher position in memory.

* 7K is enough room for the minimum VORTEX nucleus components, plus four empty TIDB's and three I/O drivers. Users with more I/O devices or a greater number of TIDB's will need more than 8K.

Figure 14-2. VORTEX Memory Map

Priority Level		
Foreground Priority Levels	31 · · ·	
	26 System Task SAL2	
	25 VORTEX System Tasks SAL and ERROR	
	24 Driver Tasks (Low-Speed Devices)	
	23 Driver Tasks (High-Speed Devices)	
	22 · · · · · 11	
	10 Operator Communication Task	
	9 · · · 2	
	Background Priority Levels	1 VORTEX System Protected Tasks
		0 User Unprotected Tasks

Figure 14-3. VORTEX Priority Structure

REAL-TIME PROGRAMMING

Symbol	Word	Bits															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TBTRD	0	Task Thread															
TBST	1	Task Status															
TBPL	2	Task Status										Priority Level					
TBEVNT	3	Interrupt Event															
TBRSA	4	A Register (Reentrant and Suspension Stack)															
TBR SB	5	B Register (Reentrant and Suspension Stack)															
TBR SX	6	X Register (Reentrant and Suspension Stack)															
TBR SP	7	OF	P Register (Reentrant and Suspension Stack)														
TBR STS	8	Temporary Storage (Reentrant and Suspension Stack)															
TBENTY	9	Task Entry Address															
TBTMS	10	Time Counter - Clock Resolution Increments															
TBTMIN	11	Time Counter - Minute Increments															
TBISA	12	A Register (Interrupt Stack)															
TBISB	13	B Register (Interrupt Stack)															
TBISX	14	X Register (Interrupt Stack)															
TBISP	15	OF	P Register (Interrupt Stack)														
TBISRS	16	Reentrant Stack Address (Interrupt Stack)															
TBIO	17	No. of I/O Requests Threaded								No. of I/O Requests Active							
TBKN1	18	Task Name															
TBKN2	19	Task Name															
TBKN3	20	Task Name															
TBTLC	21	First Address in Allocatable Memory															
TBCPTH	22	Background Task Queue															
TBATSK	23	Address of Scheduling TIDB															
TBRSE	24	Task Error Code															
TBSIZ	25	Task Size								Unused							
TBNUCL	26	Nucleus Module Indicators								Unused				I T C		Map Key	
TBMING	27	Map Image Address															
TBIST	28	Interrupt Status															
*TBR SR3-TBR SR7	29-33	V75 Registers (reentrant and suspension stack)															
*TBIS R3-TBIS R7	34-38	V75 Registers (interrupt stack)															

*Words 29 through 38 are present only if the V75 flag was set at SYSGEN and the task had a long TIDB created.

Figure 14-4. TIDB Description

Table 14-1. TIDB Layout V77-800 (Continued)

Symbol	Word	Bits															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TBTRD	0	Task Thread															
TBST	1	Task Status															
TBPL	2	Task Status										Priority Level					
TBEVNT	3	Interrupt Event															
TBRSA	4	A Register (Reentrant and Suspension Stack)															
TBRSE	5	B Register (Reentrant and Suspension Stack)															
TBRSE	6	X Register (Reentrant and Suspension Stack)															
TBRSP	7	OF	P Register (Reentrant and Suspension Stack)														
TBRSTS	8	Temporary Storage (Reentrant and Suspension Stack)															
TBENTY	9	Task Entry Address (or RECOV)															
TBTMS	10	Time Counter - Clock Resolution Increments															
TBTMIN	11	Time Counter - Minute Increments															
TBISA	12	A Register (Interrupt Stack)															
TBISB	13	B Register (Interrupt Stack)															
TBISX	14	X Register (Interrupt Stack)															
TBISP	15	OF	P Register (Interrupt Stack)														
TBISRS	16	Reentrant Stack Address (Interrupt Stack)															
TBIO	17	No. of I/O Requests Threaded								No. of I/O Requests Active							
TBKN1	18	Task Name															
TBKN2	19	Task Name															
TBKN3	20	Task Name															

←

Table 14-1. TIDB Layout V77-800 (Continued)

→	TBTLC	21	First Address in Allocatable Memory			
	TBCPTH	22	Background Task Queue			
	TBATSK	23	Address of Scheduling TIDB			
	TBRSE	24	Task Error Code			
	TBSIZ	25	Task Size		Unused	
	TBNUCL	26	Nucleus Module Indicators	Un-used	R E C O V	I T C
	TBMING	27	Map Image Address			
	TBIST	28	Interrupt Status			
	TBRSR3-TBRSR7	29-33	V75 Registers (reentrant and suspension stack)			
	TBISR3-TBISR7	34-38	V75 Registers (interrupt stack)			
	TBABP	39	Shared Procedure Table Block Pointer			
	TBMSC1	40	Reserved for future use			
	TBTRCE	41	Trace mode word Bit 15 set if task in trace mode Bits 0-14 contains trace address			
	TBPRST	42	V77-800 processor status			
	TBRPSW	43	PSW IOC and serv storage			

Table 14-2. TIDB Description

Key:

Symbol	Word	Bits	Set =	Description
TBTRD	0	15-0	Task thread	Points to next TIDB in chain. V\$TB points to the highest-priority active task. Last TIDB on queue has zero in TBTRD.
TBST	1	15-0	Task status	See table 15-5.
TBPL	2	15	Task opened	Bit set when SAL has opened task but not loaded it (memory not available).
		14	Long TIDB	Bit set if V75 SYSGEN and task had a long TIDB created. Ten words are allocated at the end of TIDB to save extra registers.
		13	Load overlay	RTE overlay request made by task with overlay name in user request. 1 = overlay load.
		12	Background checkpoint I/O wait	Foreground task waiting for background I/O to complete so it can be checkpointed to make allocatable memory available. 1 = yes.
		11	Allocation override flag	Overrides bits 9 and 12 of TBPL and bit 5 of TBST. When FNIS routine of SAL releases memory and/or a TIDB, sets bit 11 for tasks having bits 9 and 12 of TBPL and bit 5 of TBST set. SAL then tries to allocate memory; nor scheduler, a TIDB. 1 = override.
		10	Background being checkpointed	Background task being written on checkpoint file. 1 = yes.
		9	TIDB not available	Schedule request made but no TIDBs available for allocation. The task is suspended until one becomes available. 1 = TIDB not available.
		8		Task waiting for available map. 1 = map has been assigned to task.

REAL-TIME PROGRAMMING

Table 14-2. TIDB Description (continued)

Symbol	Word	Bits	Set =	Description
		7		Task map checkpoint. 1 = task's map has been checkpointed.
		6	Delay type 3 request	Set by RTE when a delay, type 3 request is made. Cleared by IOC upon completion of I/O request.
		5-0	Task priority level	Specifies priority level (0-31) of task to be executed.
TBEVNT	3	15-0	Interrupt event	Matches bits in interrupt-handler calling sequence. Interrupt-handler event inclusively ORed into TIDB word 3 when processed by line handler. If a bit sets while status bits 3 and 14 are set, dispatcher activates task. Clear event word before exiting.
TBRSA	4	15-0	A register (reentrant and suspension stack)	IOC and RTE calls store volatile register contents in this stack (words 4-8).
TBRSE	5	15-0	B register (reentrant and suspension stack)	
TBRSE	6	15-0	X register (reentrant and suspension stack)	
TBRSP	7	15	OF (overflow) register (reentrant and suspension stack)	
		14-0	P register (reentrant and suspension stack)	
TBRSTS	8	15-0	Temporary storage (reentrant and suspension stack)	
TBENTY	9	15-0	Task entry	Absolute address of first executable data of a task.

Table 14-2. TIDB Description (continued)

Symbol	Word	Bits	Set =	Description
TBTMS	10	15-0	Time counter (clock resolution increments)	Words 10 and 11 indicate time left before execution. (Clock routine increments both words when bit 6 or 7 is set in status 1.)
TBTMIN	11	15-0	Time counter (minute increments)	
TBISA	12	15-0	A register (interrupt stack)	Words 12-16 store volatile register contents during interrupt by higher-priority task. (Upon reactivation, words 12-16, volatile register contents, and reentrant stack pointer are restored and execution is continued.)
TBISB	13	15-0	B register (interrupt stack)	
TBISX	14	15-0	X register (interrupt stack)	
TBISP	15	15	OF (overflow) register (interrupt stack)	
		14-0	P register (interrupt stack)	
TBISRS	16	15-0	Reentrant stack pointer (interrupt stack)	
TBIO	17	15-8	Number of I/O requests threaded	Incremented by IOC when I/O request is received, and decremented upon completion. (A task cannot exit or abort until counter is zero.)
		7-0	Number of active I/O requests	Incremented by IOC when it sets an I/O driver active, and decremented upon completion.
TBKN1	18	15-0	Task name	First two characters of six-character task name.

Table 14-2. TIDB Description (continued)

Symbol	Word	Bits	Set =	Description
TBKN2	19	15-0	Task name	Second two characters of six-character task name.
TBKN3	20	15-0	Task name	Final two characters of six-character task name.
TBTLC	21	15-0	First address in allocatable memory	Points to first address allocated for use by task. After a task has been loaded, SAL save the read-only page number and number of pages in TBTLC as described for TBNUCL, bit 12.
TBCPTH	22	15-0	Background task queue	Any background task waiting to be loaded in background allocatable memory is queued through this word. (A running background task can schedule other background tasks, but cannot load them until space is available.)
TBATSK	23	15-0	Address of scheduling task's TIDB	Stores this address, and upon EXIT or ABOFT (if bit 1 of TBST set) reactivates scheduling.
TBRSE	24	15-0	Task error	Upon error, system routines store error codes here and set error status bit (4 of TBST). ERROR routine decodes and prints message.
TBSIZ	25	15-10	Task size	Number of pages of memory to be allocated to task.
		9-0	Reserved for future use	
TBNUCL	26	15-8	Nucleus indicator	Bit 8 reserved for future VORTEX use. Bit 9 when set indicates map foreground blank common in task; read-write access mode. Bit 10 when set indicates map nucleus table module in task; priority 0 tasks are mapped with module set to read operand only. All other priority tasks are mapped with the module set to read-write access mode.

Table 14-2. TIDB Description (continued)

Symbol	Word	Bits	Set =	Description
				Bit 11 when set indicates map global FCB in task; this module is mapped read-write access mode. Bit 12 when set indicates map pages read-only specified by LMGEN. Read only pages have been designated during load module generation. The logical page number and the number of pages are set in the load module pseudo TIDB and temporarily stored in TBTMIN bits 15-8 and bits 7-0 respectively. After the task is loaded in memory, the page numbers are stored in TBTLC, SAL sets the specified pages to read-only access mode.
		7-5	Reserved for future VORTEX use	
		4	ITC	Task has active ITC request.
TBKEY	26	3-0	Key	Task map key. This is the map number (0-15) assigned to the task by SAL or SGEN.
TBMIMG	27	15-0	Map image	Address of task map image. This is the map 0 logical address of the task's map image. Normally it would be immediately following the task's TIDB.
TBIST	28	15-0	Interrupt status	Bit 15 is 0 if V\$KEY to be set to zero and is 1 if V\$KEY to be set to TBIST (bits 3-0). Bits 14-0 are the map status as input from hardware.
TBRSR3	29	15-0	V75 register 3 (reentrant and suspension stack)	IOC and RTE call store volatile register contents in this stack (words 29-34).
TBRSR4	30	15-0	V75 register 4	
TBRSR5	31	15-0	V75 register 5	
TBRSR6	32	15-0	V75 register 6	
TBRSR7	33	15-0	V75 register 7	

Table 14-2. TIDB Description (continued)

Symbol	Word	Bits	Set =	Description
TBISR3	34	15-0	V75 register 3	Words 31-35 store volatile register contents during interrupt by higher priority task (see description of TBISA).
TBISR4	35	15-0	V75 register 4	
TBISR5	36	15-0	V75 register 5	
TBISR6	37	15-0	V75 register 6	
TBISR7	38	15-0	V75 register 7	

Table 14-3. Map of Lowest Memory Sector

Address	Symbolic Name	Description
00-01		CPU interrupt code (preset to NOP)
02-015		Unassigned: available to the user
016		Unassigned. Reserved for future VORTEX II use
017		TIDB address for WCS reload task
020,021		Memory protection interrupt: halt (jump-and-mark to V\$MPER)
022,023		Memory protection interrupt: I/O (jump-and-mark to V\$MP3)
024,025		Memory protection interrupt: write (jump-and-mark to V\$MP2)
026,027		Memory protection interrupt: jump (jump-and-mark to V\$MAP2)
030,031		Memory protection interrupt: unassigned (jump-and-mark to V\$MAP1)
032,033		Memory protection interrupt: instruction fetch (jump-and-mark to V\$MAPE)
034,037		Reserved for future VORTEX II use. Jump-and-Mark to V\$MPI0 to ignore spurious interrupts
040,041		Power-down interrupt (jump-and-mark to V\$PFDN)

Table 14-3. Map of Lowest Memory Sector (continued)

Address	Symbolic Name	Description
042,043		Power-up interrupt (jump-and-mark to V\$PFUP)
044,045		Variable-interval interrupt address (jump-and-mark to V\$CLOCK)
046	V\$CRDM	Keypunch (0 = 026, 1 = 029): Bit 0 SGEN nominal keypunch Bit 1 Set to 1 (if V75 system) Bit 8 Current keypunch specified by JCP /KPMODE directive (/JOB, /FINI, or /ENDJOB resets the current value to nominal value)
047	V\$JCTM	JCP Temporary Storage
050-053	V\$JNAM	Eight-character job name
054	V\$LCNT	Line count (set by a JCP /FORM directive): used by DAS MR assembler and FORTRAN compiler to determine the number of lines printed before a top of form is issued.
055	V\$JCFG	JCP flags: Bits 15-10 Number of extra memory blocks to be allocated with background task (cleared after loading) Bits 9-6 Unused. Bit 5 JPDUMP active Bit 4 Dump flag if load and go Bit 3 Dump flag (if set, the background dumps after a normal EXIT or abortion) Bits 2-0 Load-and-go flags
056-057		Reserved for future use
060-061		Memory parity trap
062		V77 memory protect error address
063	V\$TCTL	Bit 0 VIDEO control Bits 1-15 Reserved
064		Reserved for future use
065		Bits 8-15 VOLA page Bits 0-7 VOLA size (in pages)
066		RPG II control flag
067		V\$VOLT address

Table 14-3. Map of Lowest Memory Sector (continued)

Address	Symbolic Name	Description
070-073	V\$DATE	Eight-character date set up by OPCOM directive ;DATE,mm/dd/yy
074	V\$PLCT	Permanent line count set up at system-generation time
075	V\$BGLB	Protection code and logical unit number of the BL unit
076-077		FPP (Floating-Point Processor) interrupt (jump and mark to V\$FPP)
0100-0117		PIM 0 jump-and-mark to individual line handlers. Unassigned lines are set to JMPM V\$MPI0 to ignore spurious interrupts
0120-0137		PIM 1* jump-and-mark to individual line handlers
0140-0157		PIM 2* jump-and-mark to individual line handlers
0160-0177		PIM 3* jump-and-mark to individual line handlers
0200-0217		PIM 4* jump-and-mark to individual line handlers
0220-0237		PIM 5* jump-and-mark to individual line handlers
0240-0257		PIM 6* jump-and-mark to individual line handlers
0260-0277		PIM 7* jump-and-mark to individual line handlers
0300	V\$CTL	Address of currently executing task TIDB (0177777 = dispatcher, 037, = real-time clock routine)
0301	V\$CPL	Priority level of currently executing task
0302	V\$CRS	Address of current reentrant stack (zero if the currently executing task is not executing a reentrant subroutine)
0303	V\$TB	Address of highest-priority TIDB in the active stack
0304	V\$UTB	Address of dynamically allocated page. If zero, no page yet allocated. This is the top of the thread for pages allocated for dynamic memory allocation as required for TIDB space, I/O request, etc.

Table 14-2. TIDB Description (continued)

Symbol	Word	Bits	Set =	Description
TBISR3	34	15-0	V75 register 3	Words 31-35 store volatile register contents during interrupt by higher priority task (see description of TBISA).
TBISR4	35	15-0	V75 register 4	
TBISR5	36	15-0	V75 register 5	
TBISR6	37	15-0	V75 register 6	
TBISR7	38	15-0	V75 register 7	
TBABP	39	15-0		Pointer to first shared procedure table block
(V77-400/600)				
TBEXTN	40	15-0		Pointer to first TIDB extension block (used by Executive Services) ←
TBORDS	41	15-8 7-0		Permanent VNO ordinal Current VNO ordinal
(V77-800)				
TBMSC1	40	15-0		Reserved
TBTRCE	41	15-0		Trace mode storage
TBPRST	42	15-0		Processor Status Word (inter.)
TBRPSW	43	15-0		Processor Status Word (Exec and I/O calls)
TBAFAU	44	15-14		Arithmetic fault enable
		13-6		Reserved
		5-0		Arithmetic fault flags
TBAEXT	45	15-0		Arithmetic fault service add
TBISFP	46-49	15-0		FPP register storage
TBEXTN	50	15-0		same as for V77-400/600
TBORDS	51	15-8		same as for V77-400/600
		7-0		same as for V77-400/600

Table 14-3. Map of Lowest Memory Sector

Address	Symbolic Name	Description
00-01		CPU interrupt code (preset to NOP)
02-015 016		Unassigned: available to the user Unassigned. Reserved for future VORTEX II use
017		TIDB address for WCS reload task
020,021		Memory protection interrupt: halt (jump-and-mark to V\$MPER) (V77-400/800 Non-jump memory protect)
022,023		Memory protection interrupt: I/O (jump-and-mark to V\$MP3) V77-800 Trace interrupt (jump-and-mark to V\$TRAC)
024,025		Memory protection interrupt: write (jump-and-mark to V\$MP2) (V77-800, 24 = Processor Status Word) (V77-800, 25=Cache Status Word)
026,027		Memory protection interrupt: jump (jump-and-mark to V\$MAP2)
030,031		Memory protection interrupt: unassigned (jump-and-mark to V\$MAP1) (V77-800, 30 = Arithmetic fault address) (V77-800, 31 = System PSW setting)
032,033		Memory protection interrupt: instruction fetch (jump-and-mark to V\$MAPE) 32 { Bit 15 = FPP available flag (V77-800) Bits 14-0 Reserved (V77-800) 33 V77-800 reserved
034,037		Reserved for future VORTEX II use. Jump-and-Mark to V\$MPI0 to ignore spurious interrupts
040,041		Power-down interrupt (jump-and-mark to V\$PFDN)

→

→

Table 14-3. Map of Lowest Memory Sector (continued)

Address	Symbolic Name	Description
042,043		Power-up interrupt (jump-and-mark to V\$PFUP)
044,045		Variable-interval interrupt address (jump-and-mark to V\$CLOK)
046	V\$CRDM	Keypunch (0 = 026, 1 = 029): Bit 0 SGEN nominal keypunch Bit 1 Set to 1 (if V75 system) Bit 4-2 =0 for V77-600 (software) = 1 for V77-400 = 2 for V77-600 (micro) = 3 for V77-800 (soft) = 4 for V77-800 (micro) Bit 8 Current keypunch specified by JCP /KPMODE directive (/JOB, /FINI, or /ENDJOB resets the current value to nominal value)
047	V\$JCTM	JCP Temporary Storage
050-053	V\$JNAM	Eight-character job name
054	V\$LCNT	Line count (set by a JCP /FORM directive): used by DAS MR assembler and FORTRAN compiler to determine the number of lines printed before a top of form is issued.
055	V\$JCFG	JCP flags: Bits 15-10 Number of extra memory blocks to be allocated with background task (cleared after loading) Bits 9-7 Unused. Bit 6 (V77-800 trace selected) Bit 5 JPDUMP active Bit 4 Dump flag if load and go Bit 3 Dump flag (if set, the background dumps after a normal EXIT or abortion) Bits 2-0 Load-and-go flags
056-057		Reserved for future use (Micro, 56 = Address of dispatcher) (Micro, 57 = Size of background task) (ERCC memory interrupt)
060-061		Memory parity trap
062		V77 memory protect error address
063	V\$TCTL	Bit 0 VIDEO Control Bits 1-14 Reserved Bit 15 Disable T\$RR (VIP time slicer)

Update B

↓
↑

Table 14-3. Map of Lowest Memory Sector (continued)

Address	Symbolic Name	Description
064		Reserved for future use
065		Bits 8-15 VOLA page Bits 0-7 VOLA size (in pages)
066		RPG II control flag
067		V\$VOLT address
070-073	V\$DATE	Eight-character date set up by OPCOM directive ;DATE,mm/dd/yy
074	V\$PLCT	Permanent line count set up at system-generation time
075	V\$BGLB	Protection code and logical unit number of the BL unit
076-077		FPP (Floating-Point Processor) interrupt (jump and mark to V\$FPP)
0100-0117		PIM 0 jump-and-mark to individual line handlers. Unassigned lines are set to JMPM V\$MPIO to ignore spurious interrupts
0120-0137		PIM 1* jump-and-mark to individual line handlers
0140-0157		PIM 2* jump-and-mark to individual line handlers
0160-0177		PIM 3* jump-and-mark to individual line handlers
0200-0217		PIM 4* jump-and-mark to individual line handlers
0220-0237		PIM 5* jump-and-mark to individual line handlers
0240-0257		PIM 6* jump-and-mark to individual line handlers
0260-0277		PIM 7* jump-and-mark to individual line handlers
0300	V\$CTL	Address of currently executing task TIDB (0177777 = dispatcher, 037, = real-time clock routine)
0301	V\$CPL	Priority level of currently executing task
0302	V\$CRS	Address of current reentrant stack (zero if the currently executing task is not executing a reentrant subroutine)

Table 14-3. Map of Lowest Memory Sector (continued)

Address	Symbolic Name	Description
0305	V\$PTVB	Address of next entry in reentrant stack
0306	V\$FLRS	Address of first location of reentrant stack
0307	V\$LRSK	Address of last location of reentrant stack + 1
0310	V\$CKPT	Checkpoint flag (set if background checkpointed)
0311	V\$OPCL	Address of TIDB for OPCOM task
0312	V\$LSAL	Address of TIDB for system SAL task
0313	V\$LER	Address of TIDB for system ERROR task
0314	V\$TJCP	Address of TIDB for JCP task
0315	V\$BTB	Address of current active background task TIDB (zero if no background task active)
0316	V\$NPAG	Number of available physical pages remaining in V\$PAGE for allocation
0317	V\$LLUP	Logical address specifying the end of the execution background tasks allocated memory space
0320	V\$IM	Interrupt mask for PIM 0 (0 = enable, 1 = disable) (bit 0 = line 0)
0321		Interrupt mask for PIM 1
0322		Interrupt mask for PIM 2
0323		Interrupt mask for PIM 3
0324		Interrupt mask for PIM 4
0325		Interrupt mask for PIM 5
0326		Interrupt mask for PIM 6
0327		Interrupt mask for PIM 7
0330	V\$MAP	Map key availability flag word. Bit 0 = map 0, bit 1 = map 1, etc. A zero indicates that the map is unavailable for assignment, a 1 = map is available for assignment
0331	V\$BTBM	Base address of nucleus table module. Top of nucleus table module defined by V\$GFCB

Table 14-3. Map of Lowest Memory Sector (continued)

Address	Symbolic Name	Description
0332	V\$GFCB	Base address of global FCBs
0333	V\$MIMG	Map 0 image address
0334-0337	V\$ST0, V\$ST1, V\$ST2, V\$ST3	FUNC1 word for executive mode states 0, 1, 2, 3. Used by map 0 tasks to switch executive mode states. See section 1.3 for description on the use of V\$ST0-V\$ST3. These words are set up by the dispatcher. Bits 0-3 are set to the map number in TBKEY. If the task has been interrupted, the map number in bits 0-3 of TBIST is used
0340	V\$KEY	VORTEX currently executing map key
0341	V\$CRDR	Address of resident directory. See section 14.4.8
0342	V\$TBGT	Top of thread of background tasks waiting for allocation
0343	V\$TMS	Time-of-day in 5-millisecond increments (fractions of a minute stored in this word; upon reaching 1-minute V\$TMN increments, V\$TMS resets). The range is 0 to 12000.
0344	V\$TMN	Time-of-day in minutes (full minutes up to 24 hours stored in this word; upon reaching 24 hours (24 x 60 minutes), V\$TMN resets). The range is 0 to 1440.
<u>0345</u>	<u>V\$LUNT</u>	Address of logical-unit name table
0346	V\$OPCF	OPCOM lockout flag (busy)
0347	V\$FGLB	Protection code and logical-unit number of the FL unit
0350	V\$FREE	Reserved for future VORTEX use
0351	V\$CTMS	Clock resolution in 5-millisecond increments (user-specified millisecond interrupt rate/5) specified at system-generation time
0352	V\$SCV	Selected clock count (1 to 4095) ([user-specified millisecond interrupt rate] x [1000/V\$CKB])
0353	V\$LPP	Pointer to last tested word in V\$PAGE
0354	V\$CRM	Clock resolution increments for fractions of a minute in 5-millisecond increments

Table 14-3. Map of Lowest Memory Sector (continued)

Address	Symbolic Name	Description
0355	V\$DSTB	Address of DST block
0356	V\$LIT	Last address in background literal pool
0357	V\$PGT	Address of V\$PAGE, physical page availability mask.
0360	V\$CTAD	Base address for controller address table
0361	V\$SCTL	Current controller in scan
0362	V\$NCTR	Number of controllers
0363-0372	V\$PIMN	External device address table for PIMs
0373-0374	JUMP V\$IOST	VORTEX II link for IOC STAT CALL
0375	V\$SLFG	System SAL task busy flag (1 = busy)
0376	V\$ERFG	Error task busy flag (1 = busy)
0377	V\$JOP	JCP operating flag (1 = busy)
0400	<u>V\$LUT1</u>	Starting address of logical-unit table for JCP/OPCOM-assignable logical units (1 - 100)
0401	<u>V\$LUT2</u>	Starting address of logical-unit table for unassignable logical units (101-179)
0402	<u>V\$LUT3</u>	Starting address of logical-unit table for OPCOM-assignable logical units (180-255)
0403	V\$1MIN	Clock constant set up by SGEN where $V\$1MIN = 32767 - (60000 / (5 * V\$CTMS)) + 1$
0404-0405	JUMP V\$IOC	VORTEX II link to IOC
0406,0407	JUMP V\$EXEC	VORTEX II link to RTE
0410	V\$IOA	I/O algorithm
0411	V\$CKIT	Clock interrupted PIM before it could be locked out (common interrupt handler and clock-processor flag)
0412	V\$JCB	Address of 41-word JCP buffer (all system background programs and JCP input directives into this system buffer)

REAL-TIME PROGRAMMING

Table 14-3. Map of Lowest Memory Sector (continued)

Address	Symbolic Name	Description
0413	V\$OCB	Address of 41-word OPCOM buffer (OPCOM reads operator key-in requests into this buffer; if JCP is not active and a slash record is read, OPCOM moves the directive to V\$JCB before scheduling JCP)
0414	V\$BVN	Bottom of VORTEX nucleus. SGEN sets to virtual address. Initializer sets to page number
0415	V\$BFC	Bottom of foreground blank common
0416	V\$TFC	Top of foreground blank common, top of VORTEX nucleus core
0417	V\$PST	Maximum RMD partitions per unit in system
0420	ZERO	Zero word
0421	BS0	Bit mask contents 0000001
0422	BS1	Bit mask contents 0000002
0423	BS2	Bit mask contents 0000004
0424	BS3	Bit mask contents 0000010
0425	BS4	Bit mask contents 0000020
0426	BS5	Bit mask contents 0000040
0427	BS6	Bit mask contents 0000100
0430	BS7	Bit mask contents 0000200
0431	BS8	Bit mask contents 0000400
0432	BS9	Bit mask contents 0001000
0433	BS10	Bit mask contents 0002000
0434	BS11	Bit mask contents 0004000
0435	BS12	Bit mask contents 0010000
0436	BS13	Bit mask contents 0020000
0437	BS14	Bit mask contents 0040000
0440	BS15	Bit mask contents 0100000
0441	BR0	Bit mask contents 0177776
0442	BR1	Bit mask contents 0177775
0443	BR2	Bit mask contents 0177773

Table 14-3. Map of Lowest Memory Sector (continued)

Address	Symbolic Name	Description
0444	BR3	Bit mask contents 0177767
0445	BR4	Bit mask contents 0177757
0446	BR5	Bit mask contents 0177737
0447	BR6	Bit mask contents 0177677
0450	BR7	Bit mask contents 0177577
0451	BR8	Bit mask contents 0177377
0452	BR9	Bit mask contents 0176777
0453	BR10	Bit mask contents 0175777
0454	BR11	Bit mask contents 0173777
0455	BR12	Bit mask contents 0167777
0456	BR13	Bit mask contents 0157777
0457	BR14	Bit mask contents 0137777
0460	BR15	Bit mask contents 0077777
0461	NEG	Bit mask contents 0177777
0462	LHW	Left-half word mask (0177400)
0463	RHW	Right-half word mask (0000377)
0464	THREE	Data word (000003)
0465	FIVE	Data word (000005)
0466	SIX	Data word (000006)
0467	SEVEN	Data word (000007)
0470	NINE	Data word (000011)
0471	TEN	Data word (000012)
0472	BM17	Bit mask word (000017)
0473	BM37	Bit mask word (000037)
0474	BM77	Bit mask word (000077)
0475	BM177	Bit mask word (000177)
0476	BM777	Bit mask word (000777)
0477	BM1777	Bit mask word (001777)
0500-0777		Background literals and pointers

14.2.3 Timing Considerations (Approximate)

Real-time clock interrupt processor: At each incrementation of the real-time clock, there is a TIDB service scan requiring

$$x + 8y + 7z \text{ cycles}$$

where

- x is 48 when the scan interrupts the dispatcher, or 63 when it interrupts a task and must establish a reentrant stack and store the contents of the volatile registers
- y is the number of TIDBs searched
- z is the number of tasks having time- or schedule-delay status bits set

The clock interrupt is disabled during the execution of the clock processor, and PIM interrupts are disabled for 26 cycles following the initial entry of the clock processor.

Dispatcher interrupt processor: The time required to begin execution of a task through the dispatcher is a function of the number of TIDBs searched before execution. The time required to begin execution of the *n*th task is

$$t + 14u + 17v + 12w + 18x + 25y + z$$

where

- t is 17 or 25, depending on the entry to the dispatcher
- u is the number of tasks with task-suspended bits (TBST bit 14) set
- v is the number of tasks with events expected but event word reset
- w is the number of tasks with error bits (TBST bit 4) set but error task busy
- x is the number of tasks with either task-aborted (TBST bit 13) or task-exited (TBST bit 12) set but I/O not completed
- y is the number of tasks active but not loaded
- z is one of the following values:
 107 to activate the ERROR task
 110 to activate the SAL task on aborting or exiting
 114 to activate a loaded task that is not suspended, or to activate the SAL task to load the requested task
 104 to activate an interrupted, suspended task
 62 to activate a task when the event word is set and the interrupt suspended

Search, allocate, and load:

Load processing requires, for a foreground task

$$852(k) + v(k) + w(k) + x + y + ny$$

where

- k is the cycle time
- v is the nucleus module required by the task and is $28 + A + B + C$ cycles

where

- A is $28 + 8$ times the size of common, in pages
- B is 81 cycles as an average for the nucleus table module
- C is $11 + 11$ times the number of specified read-only pages

- x is the time to process an OPEN request
- y is the time to read an RMD record (pseudo TIDB)
- ny is the time to read a task from RMD into memory (variable depending on RMD device and task size)
- w is the page allocation $45 + 35$ times the task size, in pages

For a background task, load processing requires

$$945(k) + v(k) + w(k) + x + y + ny$$

where

- k is the cycle time
- w is the page allocation and is $45 + 35$ times the task size, in pages
- v nucleus module required by task and is $28 + A + B + C$

where

- A is 53 cycles (global FCB module)
- B is 81 cycles (average, nucleus table module)
- C is $11 + 11$ times the number of specified read-only pages

x, y and ny are as defined for foreground task.

Resident task load processing requires

$$(533 + 9(x) + y)k$$

where

- k is the cycle time
- x is the task size, in pages

y is the nucleus module required by task
 $48 + A + B + C + D$

where

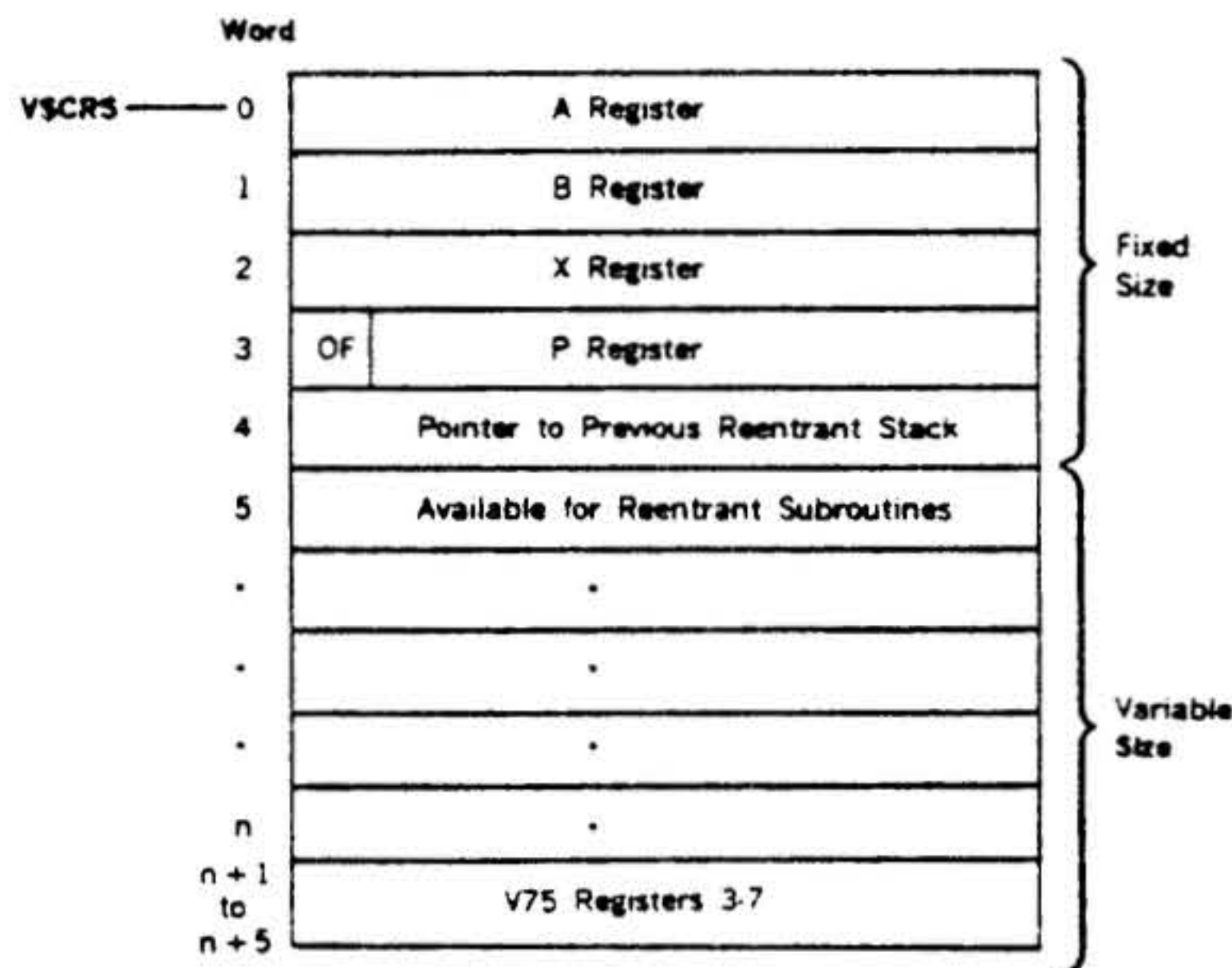
- A is $28 + 8$ times the size of common, in pages
- B is 53 cycles for global FCB
- C is 81 cycles for nucleus table module
- D is $11 + 11$ times the number of read-only pages

14.3 REENTRANT SUBROUTINES

The user can write a reentrant subroutine and add it to the VORTEX nucleus. RTE service requests ALOC and DEALOC interface between a task and a reentrant subroutine.

A task calls a reentrant subroutine via an ALOC request that allocates a variable-length push-down reentrant stack with the external name V\$CRS. The reentrant subroutine address is specified in the ALOC calling sequence. The first word of the reentrant subroutine contains the number of words to be allocated.

A reentrant stack generated by the ALOC request has the format:



When writing a reentrant subroutine, ensure that the entry location contains the number (≥ 5) of words to be allocated, execution starts at the address (entry address + 1), and that V\$CRS contains the reentrant-stack address. No IOC or RTE calls except DEALOC can be made while in a reentrant subroutine. The subroutine makes a DEALOC service request to return control to the calling task. DEALOC releases the reentrant stack, restores the A, B, and OF register contents, and returns control to the address following the ALOC request. No temporary storage is available for the reentrant subroutine except that allocated in the reentrant stack.

Parameters or pointers can be passed to the reentrant subroutine in the A and/or B (and V75 if present) registers, as well as in-line after the ALOC macro.

Two tasks make ALOC calls to RSUB. RSUB reserves six words of allocatable memory with the sixth word as temporary storage. The A register (reentrant stack) returns a value to the calling task. If task A is on priority level 5 and task B is on level 6, RSUB running on level 5 is interrupted and the level 6 task B executed. This, in turn, makes an ALOC request and executes RSUB. RSUB then executes to completion before RSUB on level 5 can be completed.

Example:

```

Task A
ALOC    RSUB
JAZ     ----
.
.
.
END
    
```

```

Task B
ALOC    RSUB
JAZ     ----
.
.
.
END
    
```

```

Reentrant Subroutine
V$CRS   NAME    RSUB
RSUB    EQU     0302
        DATA   6      Allocate six-word
        LDX    V$CRS   Stack (one temporary
                        location)
.
.
.
        STA    5, 1    Save A in temporary
                        storage
.
.
.
        LDA    5, 1    Get temporary storage
                        value
.
.
.
        STA    0, 1    Modify return in A
                        register
.
.
.
        DEALOC                Return to location
                                following ALOC call
.
.
.
END
    
```

14.4 CODING AN I/O DRIVER

The IOC (section 3) activates I/O drivers. When a user task makes an I/O request, it executes a JSR 0404,X instruction. IOC then makes validity checks on the parameters specified in the request block (RQBLK) that immediately follows the JSR instruction. IOC queues RQBLK to the I/O driver controller table (CTBL), and activates the corresponding controller-table TIDB. The TIDB contains the entry address for the I/O driver. To determine the proper CTBL and corresponding TIDB, IOC obtains the logical-unit number from RQBLK. By referring to the logical-unit table (LUT), IOC then finds the device assigned to that logical unit. Each device has a device specification table (DST) associated with it, and each DST has a corresponding controller table.

In VORTEX all RQBLKs are moved to map 0 dynamically allocable space. Upon completion of the I/O request, IOC moves the RQBLK to the requesting task's logical memory.

14.4.1 I/O Tables

Not all the data discussed in this section are required for coding every special-purpose driver, but it is presented to provide maximum flexibility in defining driver interfaces.

When an I/O driver is entered, it has the data, system pointers, and table address necessary for the I/O driver processing. At system-generation time, additional working storage space can be assigned to the I/O driver as an extension of the controller table. The data available are:

- a. V\$CTL (lower-memory system symbol defining the current TIDB) = address of TIDB associated with the I/O driver controller table.
- b. TBRSTS (word 8 of controller TIDB) = address of controller table CTBL.
- c. Within CTBL, the following:
 - (1) CTIDB (word 0) = controller TIDB address (V\$CTL)
 - (2) CTDST (word 3) = address of DST
 - (3) CTRQBK (word 4) = address of RQBLK to be processed
 - (4) CTDVAT (word 6) = controller device address
 - (5) CTSTAT (word 8) = temporary storage available for driver
 - (6) CTBICB (word 9) = address containing assigned BIC address (e.g., 020,022)
 - (7) CTFCB (word 10) = FCB or DCB address for I/O request specified in CTRQBK (word 4)
 - (8) CTWDS (word 11) = contains, upon exit, number of words transferred
 - (9) CTSTSZ (word 13) = number of words per RMD sector
 - (10) CTTKSZ (word 14) = number of sectors per RMD track
 - (11) CTPST0 (word 15) = base address of the RMD for unit 0 on this controller
 - (12) CTPST1, CTPST2, and CTPST3 (words 16, 17, and 18) = PST addresses for units 1, 2, and 3

note: 14 copies X

- d. Device specification table (DST):
 - (1) DSUNTN (bits 13 and 14 of word 2) = number (0-3) of this device on its controller
 - (2) DSPSTI (bits 6-10 of word 2) = RMD partition number (1-20) used to access the PST
- e. Request block (RQBLK): Contains user task I/O request information. The address of RQBLK is contained in CTRQBK (word 4 of the controller table). Word 1 of RQBLK contains the operation code in bits 8-11 and the mode specification in bits 12-14. Word 0 bits 5-14 contain the status.
- f. File control block (FCB): The FCB is used for RMD devices. CTFCB contains the address of FCB.
 - (1) FCRECL (word 0) = record length
 - (2) FCBUFF (word 1) = user buffer
 - (3) FCACM (word 2) = bits 8-15, access method, and bits 0-7, protection code
 - (4) FCCADR (word 3) = current record number (relative within file)
 - (5) FCCEOF (word 4) = current EOF record number (relative within partition)
 - (6) FCIFE (word 5) = beginning-of-file record number (relative within partition)
 - (7) FCFE (word 6) = end-of-file record number (relative within partition)
 - (8) FCNAM1, FCNAM2, and FCNAM3 (words 7, 8, and 9) = file names in ASCII
- g. Data control block (DCB): The DCB is used for non-RMD devices. CTDCB contains the address of DCB.
 - (1) DCRECL (word 0) = record length
 - (2) DCBUFF (word 1) = user buffer
 - (3) DCCNT (word 2) = function count
- h. V\$CTL, TIDB, CTBL, DST, and the RQBLK reside in map 0. The FCB and DCB reside in the user's logical memory and to access the data, the I/O drivers must switch to the proper executive mode state (see section 1.3).

14.4.2 I/O Driver System Functions

Each I/O driver under IOC performs certain system pre- and post-processing functions.

Pre-interrupt processing: The I/O driver must switch executive mode states to fetch or store data from user mode (see section 1.3). If the I/O driver uses a BIC, the driver calls V\$BIC with the X and A registers set to the initial and final buffer addresses respectively to build and execute the initial BIC transfer instruction. If the BIC is shared, the interrupt line handler is modified to the proper interrupt event word setting (TBEVNT) and TIDB address. V\$BIC performs this modification if the word immediately following the call (JSR V\$BIC,B) is nonzero, since this is assumed to be the interrupt event word setting. If it is zero, no line handler modification is performed. The I/O driver clears the interrupt event word (TBEVNT) in the controller TIDB immediately preceding a DELAY (type 2) call. To wait

for an interrupt, the I/O driver executes a DELAY (type 2) call with a time-out. The return to the driver, either from a time-out or interrupt is to the address immediately following the call. The contents of the X register is not restored following a DELAY call but the A and B registers are. Executing a TXA immediately preceding and a TAX following the DELAY call X restores the value in the X register.

Interrupt processing: The driver clears the time-delay flag (TBST bit 6) set by the DELAY call, and checks TBEVNT to determine if an interrupt occurred (TBEVNT = 0 indicates a time-out). Following the interrupt processing, the driver clears TBEVNT and calls DELAY (type 2) for the next instruction.

Post-interrupt processing (no errors): Upon the completion of interrupt processing, the driver sets the status bits (5-14) of RSTPR (word 0) in RQBLK, and enters the number of words transferred in CTWDS. The driver then relinquishes control and exits to IOC by executing JMP V\$FNR.

Post-interrupt processing (errors): If an error is encountered during interrupt processing, the driver sets the status bits (5-14) of RSTPR, according to the type of error. The driver then sets the A register to zero if the unit is not ready, negative if there is a parameter error, or positive if there is a hardware error. Finally, the driver exits to the IOC error routine by executing JMP V\$ERR.

14.4.3 Adding an I/O Driver to the System File

System-generation directives: The following directives are required for linkages to the controller table, controller TIDB, I/O driver entry location, DST, PST, and the PIM line handler (section 15):

Directive	Description
EQP	DSTs are generated by SGEN, one for each unit specified by the EQP directive. All DSTs generated for a controller point indirectly to the controller table specified by EQP. The pointer is to the entry name in the controller table assembly.
PIM	A PIM directive is required for each PIM line where an interrupt is expected. The PIM directive causes the system initializer to enable the mask for that line (except for the TTY or CRT output line, in which case it is initially disabled). If the driver processes both input and output interrupts, it may be advantageous for processing to set the interrupt event word for the input line to one value (e.g., 01) and the interrupt event word for the output line to another value (e.g., 02). The PIM directive also specifies if a directly connected interrupt handler is to be used (see section 14.4.5).

ASN This directive assigns logical units to physical units. If a new device is being added and it is necessary to assign that device to a logical unit when the system is initialized, an ASN is input. Otherwise, the JCP or OPCOM ASSIGN directive can be used. The logical-unit table is established by these directives.

PRT This directive for RMDs specifies the size and the mnemonic name of each partition. A PST and DST are created for each partition.

TDF This VORTEX nucleus-generation control record directive defines and builds the controller TIDB. It specifies the name of the driver, status word (TBST) setting, and priority level.

Adding controller tables: A controller table is assembled as a separate entity and added to the system-generation library (SGL) for loading at system-generation time. The controller table name is CT followed by the three- or four-character ASCII name of the controller, e.g., CTTY0A, CTMT0A, and CTD0B.

VORTEX Input/Output Control (IOC) assumes the first 13 words of all non-RMD controller tables to be identical, i.e., word 0 = CTIDB; word 1 = CTADNC, etc. For RMDs the first 18 words are assumed to be identical. Additional words may be added to the controller table by use by the individual I/O driver.

The controller table comprises parameters that are constant for a controller, and parameters that are variables for SGEN and can change with system configuration.

Constants are assembled as DATA statements. DATA statements can be added to the controller table to provide additional working space for an I/O driver.

The following standard items are required by IOC:

Word Item	Description
0	CTIDB = Name of the related controller TIDB (TB followed by the same three or four-character name used in the controller table e.g., TBD0B (for CTD0B). An EXT statement must specify the TIDB name as an external name. EXT TBD0B DATA TBD0B
1	CTADNC = This word is used by IOC as temporary storage.
2	CTOPM = The operation code mask specifying the type of I/O operation the driver is capable of processing 1 = driver is capable of processing.

REAL-TIME PROGRAMMING

Bit	Operation
0	Read
1	Write
2	Write EOF
3	Rewind
4	Skip record
5	Function
6	Open
7	Close
8-16	Reserved for future use

Example: DATA 037
For all operations excluding Function, Open, and Close.

- 3 **CTDST** = Set by IOC to DST address
Example: DATA 0
- 4 **CTRQBK** = Set by IOC to I/O request block being processed.
Example: DATA 0
- 5 **CTRTRY** = Error retry count. # T followed by the name of the controller.
Example: DATA #TTYOA
EXT #TTYOA
- 6 **CTDVAD** = Controller device address. # A followed by the name of the controller
Example: DATA #ATYOA
EXT #ATYOA
- 7 **CTIOA** = I/O algorithm. The ratio of device transfer rate to DMA transfer rate + 10 percent of the result times 32767. Zero for all non-BIC devices.
Example: when a disc transfer rate is 100K words per second and DMA rate is 300K words per second, the ratio is about .33. Set CTIOA to: DATA 030000
If ratio is .25 or 25 percent set CTIOA (DATA 020000); 50 percent set CTIOA (DATA 040000), etc.
To make CTIOA a SGEN selectable parameter (refer to section 15.5.2, EQP directive) assemble as an external e.g., EXT #D followed by the name of the controller:
EXT #DCIOA for process I/O
DATA #DCIOA
- 8 **CTSTAT** = DATA 0, for driver use.
- 9 **CTBICB** = Address of BIC flag table. B followed by the name of the name of controller,
Example: DATA BD0B
EXT BD0B
When the driver is entered the item points to a cell containing the BIC device address, 020, 022, 024, etc.
- 10 **CTFCB** = Set by IOC to the DCB or FCB address. Set to DATA 0

- 11 **CTWDS** = DATA 0. Driver use for number of words transferred.
- 12 **CTFRCT** = I/O algorithm frequency count. The number of retries to be attempted by IOC before suspending all subsequent I/O operations until the request in CTRQBK (word 4) is activated. DATA 0 for non-BIC devices.
- 13 **CTSTSZ** = RMD only. Number of words in an RMD sector.
Example: DATA 120
- 14 **CTTKSZ** = RMD only. Number of sectors in an RMD track
Example: DATA 48
- 15 **CTPST0** = RMD only. Base address of the PST for RMD unit 0 connect to this controller. P followed by the four character device name.
Example: DATA !PD00B
EXT !PD00B
- 16 **CTPST1** = RMD only. Base address of the PST for RMD unit 1.
Example: DATA !PD01B
EXT !PD01B
- 17 **CTPST2** = RMD only. Base address of PST for RMD unit 2.
Example: DATA !PD02B
EXT !PD02B
- 18 **CTPST3** = RMD only. Base address of PST for RMD unit 3.
Example: DATA !PD03B
EXT !PD03B

14.4.4 Enabling and Disabling PIM Interrupts

The disable and enable PIMs and RT clock instructions (EXC 0147, EXC 0747, EXC 0244, EXC 0444) are privileged instructions and cannot be executed in a user map (non-map 0) without creating a memory protect interrupt. The memory protect processor recognizes the interrupts caused by the disable/enable instructions and returns to the foreground task in the proper disabled or enabled state. The following restrictions apply:

- Only foreground tasks are permitted to execute the disable/enable PIMs and RT clock instructions. EX21 error message is output of a background task attempts to execute those instructions.
- The return to the foreground task is at location $n + 2$. In other words, both the disable PIMs and clock instructions (EXC 0747, EXC 0444 or vice versa) or enable PIMs and clock instructions (EXC 0147, EXC 0244 or vice versa) must be together. The second EXC instruction is not executed.

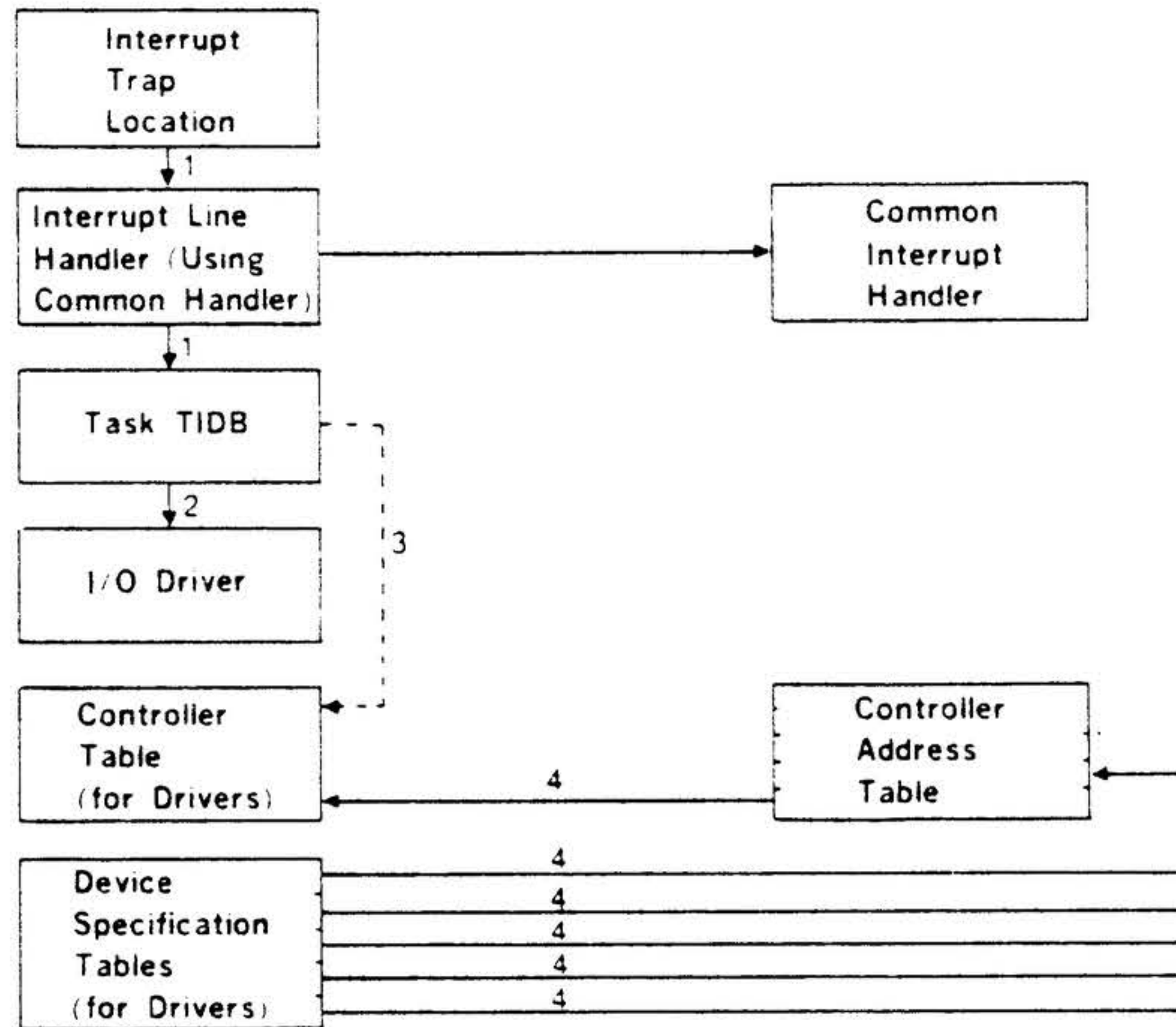
Example:

Location	Instruction	
n	EXC 0444	Disable RT clock instruction creates interrupt.
n + 1	EXC 0747	This instruction is not executed.
n + 2		Return location from the memory protect processor with PIMs and RT clock disabled.

EXC 0444 disables all PIM interrupts. EXC 0244 enables all PIM interrupts that are not masked. There is a PIM directive for each PIM line at system-generation time. The system initializer enables PIM lines. The mask is enabled unless the I/O driver specifically disables it. If a PIM directive is omitted, the linkage between the trap and the interrupt line handler cannot be established. If a PIM line mask is enabled or disabled by a driver, the system mask is updated to reflect the current status. The system mask configuration is given at low memory address V\$IM (0320 for PIM1, 0321 for PIM2, etc.).

EXC 0747 disables the real-time clock interrupt and EXC 0147 enables it.

Figure 14-5 shows the standard VORTEX driver interface.



KEY:

1. The trap address corresponding to the PIM number (from PIM directive) points to the SGEN-generated line handler. The line handler points to the TIDB (named in PIM directive), using the matching TIDB name (on TDF control record).
2. The TIDB name (on TDF control record) points to the task, using the entry name in the assembly of the task.
3. For OPCOM device drivers only. The task TIDB points to the device controller table name (on TDF control record), using the entry name in the controller table assembly.
4. The DSTs are generated by SGEN, one for each unit specified on the EQP directive. All DSTs generated for a controller point indirectly to the controller table (named in EQP directive), using the entry in the controller table assembly.

Figure 14-5. Driver Interface

14.4.5 Directly Connected Interrupt Handler

VORTEX provides a user two options of specifying directly connected interrupt handlers. The use of a directly connected interrupt handler, in lieu of the VORTEX common interrupt handler, is specified on the PIM directive during system generation (section 15.5.11). The interrupt handlers must be resident in executive mode, map 0.

Option 1 (specifying 1 as the s(n) parameter on the PIM directive) requires the user to:

- a. Save and restore the overflow indicator and all volatile registers used by the directly connected interrupt routine before returning to the interrupted task.
- b. Not allow IOC and RTE calls.
- c. Minimize execution time.
- d. Continue to lockout interrupts during processing, then enable the PIMs upon exiting. The RT clock is enabled in all cases except when the real time clock processor has been interrupted. Location 0300, V\$CTL, will contain 037 if the RT clock processor had been interrupted. The interrupt handler must provide a check for interruption out of the RT clock processor and enable or disable the RT clock accordingly.
- e. Restore the VORTEX system to the proper pre-interrupted state, executive or user mode. Any interrupt forces the system to executive mode, state 0 (see table 1-1). The interrupt handler must return to the proper state. V\$KEY, location 0340, contains the map key number of the interrupted task. If the interrupt task is the user mode ($1 \leq V\$KEY \leq 15$), the switch from "executive to user mode enable" instruction (EXC2 0246) must be executed. The "clear executive mode state mask" instruction (EXC2 0546) must also be executed.

Example:

```

.
.
.
LDB D5000
LDA 0300      Check location 0300
SUB 0473      System constant = 037
JAZ DIH10     Zero = interrupt out of
LDBI 0104546 RT clock
LDAI 0100147  Otherwise enable clock
JMP DIH10+1
DIH10 LDA D5000    = 5000
      STA DIH30     Enable clock instruction
      STB DIH30+1   Enable mask instruction
      ROP
      LDA ROV       Restore overflow
      JANZ **+3
      SOF
      LDB D5000     NOP instruction
      LDA 0340     V$KEY check interrupts
    
```

```

ANA 0472      Task map key
JAZ DIH20     0 = map 0
LDB 0104246   Switch to user map
DIH20 STB DIH30+2
      LDB RB       Now restore A, B, X
      LDX RX
      LDA RA
      EXC 0244     Enable PIM
DIH30 EXC 0147   Modified to enable clock
                        or NOP
      EXC2 0546    Modified to clear mask
      EXC2 0246    Modified to switch to
                        user map
      EXC2 0646    Enabled memory protect
      JMP *        Modified to return
                        address
D5000 DATA 05000
    
```

- f. Obtain the interrupted task return address. The directly connected interrupt line handler is entered via a JMPM instruction from the line handler (see figure 14-1) and as such the first word in the interrupt handler must be a mark location. The return address of the interrupted task is found in word 0 of the line handler, which is obtained by subtracting four from the contents of the interrupt handler's mark location.

Option 2 (specifying 2 as the s(n) parameter on the PIM directive) permits the user to use system routines to save (V\$DHD) the volatile registers and overflow indicator and restore (V\$DRTN) the volatile registers, overflow indicator, and reset the system to the proper pre-interrupted state as described above. Option 2 relieves the directly connected interrupt handler of the housekeeping chores. The A, B, X registers, overflow indicator are saved, PIM and clock interrupts are disabled prior to entering the user code (via JMPM), (see figure 14-1). The user code is entered with the A register set to the TBEVNT value and the X register set to the user code entry address.

Upon completion of processing, the directly connected interrupt handler exits to system routine, V\$DRTN.

Example:

TASK	NAME	TASK	
	ENTR		
	STA	EVNT	Save TBEVNT word
	.		Do processing
	.		.
	.		.
	EXT	V\$DRTN	
	JMP	V\$DRTN	Exit to common processor

where task must be specified on SGEN PIM directive, e.g., PIM,010,TASK,01,2.

14.4.6 VORTEX Use of BICs and BTCs

VORTEX supports a maximum of 15 BICs or BTCs. The practical system limit may be considerably less than ten depending on the availability of device addresses, the type

and number of peripherals, and other configuration considerations. The BIC or BTC transfer complete interrupts must be assigned by ascending BIC or BTC numbers (020, 022, 024, 026, 070, 072, etc.) starting with the first PIM and the first interrupt i.e., PIM 0, line 0 assigned to BIC 020; PIM 0, line 1 assigned to BIC 022, etc. The first BIC must have a device address of 020; the second, 022; the third, 024; the fourth, 026; the fifth, 070; the sixth, 072; etc. Unless the special DEF control directive is used.

I/O drivers utilizing BICs or BTC must call the common BIC routine V\$BIC. The X register is set to the initial buffer address and the A register set to the final buffer address. The call to V\$BIC is:

```
JSR    V$BIC, B
DATA          Interrupt event word or 0 if no
              line handler modification to be
              performed.
DATA          Map number
```

14.4.7 VORTEX II and VORTEX Compatibility

User programs written to operate under VORTEX will be operable under VORTEX II under the following conditions:

- a. Programs which contain any RTE service requests or Input/Output Control requests must be assembled by the VORTEX II version of DAS MR. Any program which builds these requests without the DAS MR macros must be modified so that the requests conform to the VORTEX II calling sequence.
- b. Any foreground task which executes hardware I/O instructions except disabling and/or enabling PIMs and RT clock, see section 14.4.4, must be included as part of the resident nucleus when the system is generated. Foreground library tasks which are made resident during system generation by use of the TSK directive are not considered nucleus tasks and therefore must not contain any hardware I/O instructions (see section 14.4.8 for discussion on resident tasks).
- c. Intertask communications can be accomplished: through the use of foreground blank common; by establishing named tables and buffers in the nucleus table module and referencing the named block by an external statement; by use of the RTE PASS request between a user map and map 0; by switching executive mode states (see section 1.3); by sharing the same physical pages utilizing the MAPIN and/or PAGNUM RTE requests.
- d. User tasks (except priority 1 system tasks) may not write into or execute instruction from the first physical page. This page is the VORTEX II low memory area. It is mapped as read-operand only into all user tasks (see figure 2-2), except priority 1 tasks where page 0 is mapped as read-write access mode.

- e. User tasks (non-nucleus) must not communicate with the nucleus except through the use of standard executive service and I/O requests or by referencing entry points which are contained in the core-resident library.
- f. A user task can request a transfer of a block of data from map 0 to the user map by executing a RTE PASS request.
- g. Direct connect interrupt handlers must restore the system to the pre-interrupted map state after servicing the interrupt. An alternative is to utilize the SGEN PIM directive, option 2, as described in section 14.4.5.
- h. I/O drivers written for VORTEX operation must be modified for VORTEX II as follows:
 1. The map number must be passed when calling V\$BIC, common BIC/BTC routine (see section 14.4.6).
 2. The I/O drivers must switch executive mode states (see section 1.3) to fetch/store data from a user map (DCB, FCB, buffer). RQBLK data are stored in map 0 by dynamic memory allocation.
 3. Rotating memory device (RMD) drivers must determine if a data transfer (read, write) I/O request is by SAL (search-allocated-load task). If it is a SAL request, the map number is obtained from TBEVNT of the TIDB for SAL. Otherwise, the requestor's map number is obtained from TBKEY. SAL is the RTE component which loads non-resident tasks into memory. The check may be accomplished as follows:

```

LDA    RTIDB, B    RTIDB = word 4 of RQBLK
SUB    V$LSAL      V$LSAL = location 0312 = SAL TIDB
JANZ   XXX        Jump if not SAL
LDB    V$LSAL      Yes SAL. Get map key
LDA    TBEVNT, B   From TBEVNT
JMP    YYY        Now common processing
XXX    LDB    RTIDB, B    I/O request not by SAL
YYY    LDA    TBKEY, B   Get map key from TBKEY
        ANA    BM17      Mask bits 4-0
```

4. Following a BIC transfer complete interrupt the I/O driver sense for a map memory protection I/O data transfer error:

```
SEN    0101+da, er
```

where da is the BIC device address (which is found in word 011 of the controller table), and er is the address of the error processing routine which must set up an IO46 error code prior to calling V\$ERR.

- i. If a user wants to fetch/store from the nucleus tables, the user must ensure that the nucleus table module is mapped into the user's logical memory. He does this through an external reference to a symbol, TIDB, controller table, etc., within the nucleus module. Example -- have an "EXT TBTYOA."

REAL-TIME PROGRAMMING

- j. TIDBs for non-resident tasks -- except JCP and OPCOM -- are dynamically allocated in map 0. Hence a foreground user task cannot load a register (B,X) from location 0300 (V\$CTL or an address from any other low-core location) and directly fetch the TIDB data. In VORTEX, it is possible; in VORTEX II, such an attempt would result in a memory protect interrupt. The foreground user can fetch the TIDB data by use of the PASS macro. Except for clearing the TBEVNT word, via the RTE TBEVNT request, a foreground user task cannot modify the TIDB.

14.4.8 Resident Tasks

The VORTEX II user may specify two types of resident tasks during system generation; user mode resident tasks; and executive mode map 0 resident tasks.

- a. User mode resident tasks. These tasks are foreground library tasks that are made resident via the SGEN TSK directive. These tasks execute as user mode tasks and cannot execute any I/O type instructions except enable/disable PIMs and RT clock. They reside in memory and may be scheduled via OPCOM or RTE SCHED requests specifying LUN = 0. As these tasks do not reside in map 0 virtual memory, the dynamically allocated space (see figure 1.2) is not reduced as it would be for the executive mode map 0 resident tasks. These resident tasks are defined in the resident directory specified by V\$CRDR (0341). Each entry in the directory is as follows:

Word/Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	Task Name, first two characters															
1	Task Name, second two characters															
2	Task Name, third two characters															
3	Entry Point															
4	Starting physical page number															
5	Number of pages															
6	Nucleus module indicator								Reserved for future use							

- b. Executive mode, map 0 resident tasks. These tasks reside in the nucleus program module in map 0. No special SGEN directive is required to include these tasks as part of the nucleus. The VORTEX II user specifies the generation of these resident tasks by adding the program object modules on the SGL between the CTL,21 and CTL,PART3 control records (see figure 15-2). The program name should not start with the characters "VZ--" as these are reserved for I/O drivers. SGEN processes I/O drivers selectively and ignores all I/O driver object modules unless a SGEN EQP directive specified the corresponding peripheral. These executive mode resident tasks: (1)

are permitted to execute I/O type instructions; (2) cannot normally be scheduled via the OPCOM or RTE SCHED request, but are activated by resetting bit 14 of the TIDB status word TBST (table 15-5) as are the I/O drivers and SAL; (3) must have a resident TIDB created by a SGEN TDF directive. An alternate means of executing these tasks is via an OPCOM RESUME request. However, caution must be exercised as the RESUME request activates the highest priority task with a matching name.

14.4.9 Purge Cache Command

Multi-processor systems which include shared memory, cache, and map hardware in their configurations need the ability to purge cache memory. This capability is provided by the PURGE CACHE function. The PURGE CACHE function code, which is 04000, is output to the map device by an OME instruction. This feature is restricted to foreground tasks only.

The overflow indicator determines the state of the CLOCK and PIM interrupts following the execution of a PURGE CACHE function. If the overflow indicator is reset, the CLOCK and PIM interrupts are enabled; and if the overflow indicator is set, the CLOCK and PIM interrupts are disabled following the execution of the OME instruction.

Example: Purge cache memory and leave the CLOCK and PIM interrupts enabled.

```
ROF                reset overflow to indicate
                   interrupts enabled.

OME , 046 , PCACHE  output purge cache function
                   code to map device.

.
.
.
PCACHE DATA 04000  purge cache function code
```

14.5 INTERTASK COMMUNICATION

The Intertask Communication Module (ITC) provides a means by which concurrently running tasks in a VORTEX II system may communicate with one another. The ITC module consists of a group of reentrant subroutines called through the VORTEX ALOC service request. The module, which resides in map 0, can only be used in systems having the extended instruction set.

The following functions are provided by the module:

MAILBOX Establishes the ownership of a mailbox entry in the mailbox list. This request must be issued by a receiving task before it can receive mail items.

FREEM	Releases ownership of a mailbox entry.
POST	Queues a mailbox item to a receiver's mailbox and conditionally activates the receiving task.
COPYM	Copies a message from a sender's map to a receiver's.
FIND	Returns the TIDB address of a specified task.
AWAKE	Set the TBEVNT word of a specified task.

The receiving task would provide a mailbox header and a list of empty mailbox items prior to establishing a mailbox entry. Sending tasks may then POST mail items to the receiver's mailbox. The receiving task may serially process the mail item, issuing COPYM or MAPIN requests to receive information from the receiving tasks.

14.5.1 ITC Module Operation

The ITC module has been given a driver-like name (VZITC) so it may be excluded from systems not needing this feature. In order to incorporate the ITC module into the nucleus, the user must include the following directive at system generation:

```
EQP,CTITOA,0,1,0,0
```

The entry names of the individual reentrant subroutines (VISMBX, VISFRE, VISPST, VISCPY, VISFND, VISAWK) will be placed in CL at system generation.

A receiving task will initialize its mailbox headers, queuing a sufficient number of empty mailbox items to the free list. It will then perform one or more MAILBOX calls to establish ownership of entries in the mailbox list. A sending task may then communicate with the receiver using the POST call, specifying a mailbox key. (If the receiver's TIDB address is used as the key, the sender may acquire this through the FIND call.) The POST will dequeue a mail item from the receiver's free list, insert the message control information into it and queue to the receiver's active list. If the receiving task is in a type 2 or type 3 delay, the POST will activate it by setting its TBEVNT word. (If this mechanism is used, the receiving task must reset the TBEVNT word and the interrupt expected bit by using the TBEVNT macro.) The POST will also logically-OR the activity mask into the activity flag so the receiving task may identify on which mailbox header the activity took place. The receiving task would then dequeue the mail item and use its contents to transfer a message by means of a MAPIN or COPYM call. The task would then return the processed mail item to the free list. If the sending task were suspending itself until the completion of the message transfer, the receiving task could activate it with an AWAKE call.

14.5.2 ITC Calling Sequences

The following are ITC calling sequences:

MAILBOX Establish mailbox list entry ownership

Call: R0 = mailbox key
R1 = mailbox header pointer

label **ALOC VISMBX**

Return: R0 = completion status

Where:

- 0 = Successful completion
- 1 = Memory not available for extending list
- 2 = Duplicate mailbox key found

The mailbox key may be any one-word non-zero value mutually agreed upon by the sending and receiving tasks. The receiving task's TIDB address may be used since this uniquely identifies any task in the system. A function (FIND), which is described below, has been provided to return the TIDB address of a named task. Since multiple copies of a task in the system, all bearing the same name, cannot be uniquely identified by the FIND request, the SCHED request has been modified to include a new parameter. The new form is:

SCHED level, wait, lun, key, 'xx', 'yy', 'zz', flag

Where flag = 1 the TIDB address of the scheduled task is returned in R0, and if 0 (or omitted) the original contents of R0 will be retained. This permits a task scheduling multiple copies of another task to uniquely identify each one.

Duplicate keys are not permitted and any request specifying one will be rejected.

There is a permanently allocated block of memory in the ITC module for the mailbox list which will support ten list entries. If more entries are required, a block of memory from the system's allocatable memory pool will be obtained for mailbox list extensions. The user is cautioned, however, that the mailbox list is manipulated with interrupts disabled and an inordinately long list may create system problems.

Tasks having mailbox entries allocated to them will be so identified by the system. The task should deallocate all mailbox entries it had established prior to exiting using the FREEM requests, however, if it fails to do so the system will deallocate.

FREEM Free mailbox list entry

Call: R0 = mailbox key of entry to be deleted or if zero, all entries associated with calling task.

label **ALOC VISFRE**

REAL-TIME PROGRAMMING

Return: R0 = completion date
 Where:
 0 = Successful completion
 1 = Specified key not found in list, or if found, does not belong in this task.

If the calling task specifies a mailbox key for which it does not have responsibility (i.e., it did not perform MAILBOX request which established it), the FREEM request will be ignored.

POST Post mail item to the receiver's mailbox.
 Call: R1 = pointer to 3-word message control block.
 label **ALOC VISPST**
 Return: R0 = completion status
 Where:
 0 = Successful completion
 1 = Mailbox key not found
 2 = Mail item free list empty
 3 = Map loading error

The message control block contains:

0	Mailbox key
1	Message address
2	Message size

Where the mailbox corresponds to the key used by the receiving task to establish a mailbox entry, the message address is a physical page number or the address of a message in the sender's map, and the message size is a page count or the number of words in the message.

If the receiving task is in a map-checkpointed state when the POST request is processed, the sender's map will be used to complete the posting and will be restored to the sending task before it is resumed.

The structure of the mailbox header in the receiver's map is as follows:

0	Free list front pointer
1	Free list rear pointer
2	Active list front pointer
3	Active list rear pointer
4	Activity flag pointer
5	Activity mask

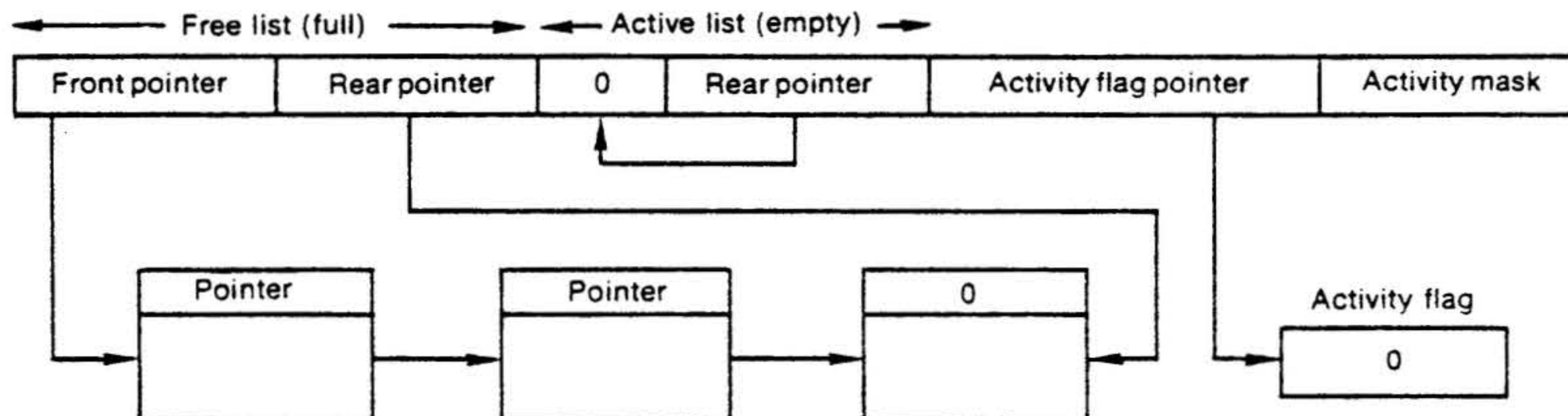
The mailbox header points to the list of empty mail items (each item consisting of a four-word block), to a list of active mail items, to an activity flag, and contains an activity mask. A POST will remove an item from the free list, insert message control information into it and queue it to the active list. The POST will also logically OR the activity mask contained in the mailbox header into the activity flag pointed to by the header. If the receiving task's interrupt expected bit is set when the POST is performed (denoting execution of a type 2 or type 3 delay), the POST will cause the receiving task's TBEVNT word to be set.

At the completion of a POST, a mail item in the active list would contain the following information:

0	Pointer to next mail item (0 if last item in list)
1	Sender's TIDB address
2	Page number or message address
3	Page count or word count

When the receiving task has acted upon the information contained in a mail item, it would dequeue it from the active list and requeue it to the free list. The receiving task is also responsible for resetting the activity flag if this is required. Note that when queueing and dequeuing an item, or when resetting the activity flag, system interrupt must be disabled to avoid possible contention with POST. See section 14.4.4 regarding disabling/enabling interrupts.

The initial state of a mailbox header would be:



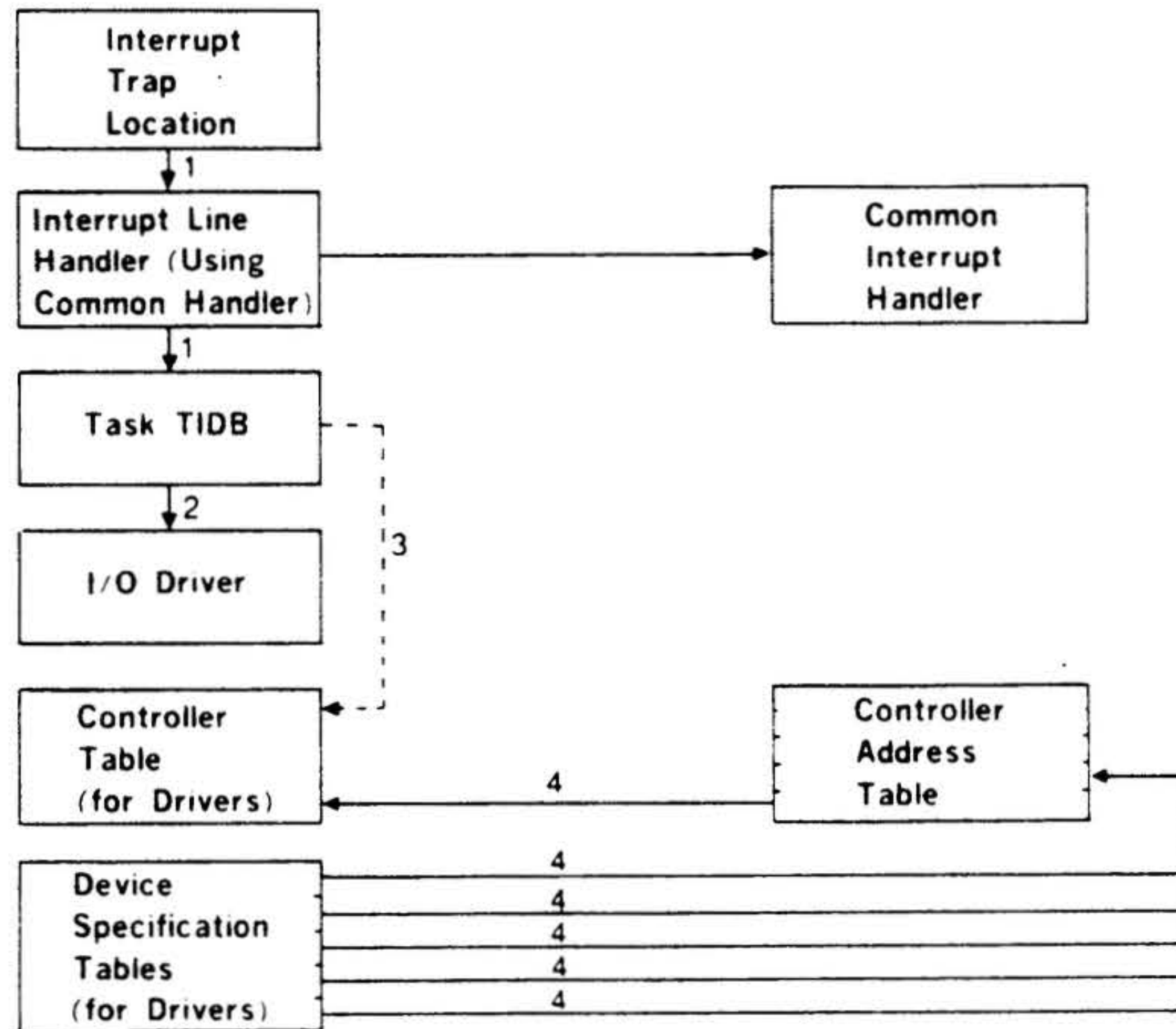
Example:

Location	Instruction	
n	EXC 0444	Disable RT clock instruction creates interrupt.
n + 1	EXC 0747	This instruction is not executed.
n + 2	.	Return location from the memory protect processor with PIMs and RT clock disabled.

EXC 0444 disables all PIM interrupts. EXC 0244 enables all PIM interrupts that are not masked. There is a PIM directive for each PIM line at system-generation time. The system initializer enables PIM lines. The mask is enabled unless the I/O driver specifically disables it. If a PIM directive is omitted, the linkage between the trap and the interrupt line handler cannot be established. If a PIM line mask is enabled or disabled by a driver, the system mask is updated to reflect the current status. The system mask configuration is given at low memory address V\$IM (0320 for PIM1, 0321 for PIM2, etc.).

EXC 0747 disables the real-time clock interrupt and EXC 0147 enables it.

Figure 14-3 shows the standard VORTEX driver interface.



KEY:

1. The trap address corresponding to the PIM number (from PIM directive) points to the SGEN-generated line handler. The line handler points to the TIDB (named in PIM directive), using the matching TIDB name (on TDF control record).
2. The TIDB name (on TDF control record) points to the task, using the entry name in the assembly of the task.
3. For OPCOM device drivers only. The task TIDB points to the device controller table name (on TDF control record), using the entry name in the controller table assembly.
4. The DSTs are generated by SGEN, one for each unit specified on the EQP directive. All DSTs generated for a controller point indirectly to the controller table (named in EQP directive), using the entry in the controller table assembly.

Figure 14-3. Driver Interface

14.4.5 Directly Connected Interrupt Handler

VORTEX provides a user two options of specifying directly connected interrupt handlers. The use of a directly connected interrupt handler, in lieu of the VORTEX common interrupt handler, is specified on the PIM directive during system generation (section 15.5.11). The interrupt handlers must be resident in executive mode, map 0.

Option 1 (specifying 1 as the s(n) parameter on the PIM directive) requires the user to:

- Save and restore the overflow indicator and all volatile registers used by the directly connected interrupt routine before returning to the interrupted task.
- Not allow IOC and RTE calls.
- Minimize execution time.
- Continue to lockout interrupts during processing, then enable the PIMs upon exiting. The RT clock is enabled in all cases except when the real time clock processor has been interrupted. Location 0300, V\$CTL, will contain 037 if the RT clock processor had been interrupted. The interrupt handler must provide a check for interruption out of the RT clock processor and enable or disable the RT clock accordingly.
- Restore the VORTEX system to the proper pre-interrupted state, executive or user mode. Any interrupt forces the system to executive mode, state 0 (see table 1-1). The interrupt handler must return to the proper state. V\$KEY, location 0340, contains the map key number of the interrupted task. If the interrupt task is the user mode ($1 \leq V\$KEY \leq 15$), the switch from "executive to user mode enable" instruction (EXC2 0246) must be executed. The "clear executive mode state mask" instruction (EXC2 0546) must also be executed.

Example:

```

.
.
.
LDB  D5000
LDA  0300      Check location 0300
SUB  0473      System constant = 037
JAZ  DIH10     Zero = interrupt out of
LDBI 0104546   RT clock
LDAI  0100147  Otherwise enable clock
JMP  DIH10+1
DIH10 LDA  D5000    = 5000
      STA  DIH40     Enable clock instruction
      STB  DIH30     Enable mask instruction
      ROF
      LDA  ROV      Restore overflow
      JANZ *+3
      SOF
      LDB  D5000     NOP instruction
      LDA  0340     V$KEY check interrupts
    
```

```

ANA  0472      Task map key
JAZ  DIH20     0 = map 0
LDB  0104246   Switch to user map
DIH20 STB  DIH30+1
      LDB  RB       Now restore A, B, X
      LDX  RX
      LDA  RA
DIH30 EXC2  0546   Modified to clear mask
      EXC2  0246   Modified to switch to
                          user map
      EXC2  0646   Enabled memory protect
      EXC  0244   Enable PIM
DIH40 EXC  0147   Modified to enable clock
                          or NOP
      JMP  *       Modified to return
                          address
D5000 DATA 05000
    
```

- Obtain the interrupted task return address. The directly connected interrupt line handler is entered via a JMPM instruction from the line handler (see figure 14-1) and as such the first word in the interrupt handler must be a mark location. The return address of the interrupted task is found in word 0 of the line handler, which is obtained by subtracting four from the contents of the interrupt handler's mark location.

Option 2 (specifying 2 as the s(n) parameter on the PIM directive) permits the user to use system routines to save (V\$DHD) the volatile registers and overflow indicator and restore (V\$DRTN) the volatile registers, overflow indicator, and reset the system to the proper pre-interrupted state as described above. Option 2 relieves the directly connected interrupt handler of the housekeeping chores. The A, B, X registers, overflow indicator are saved, PIM and clock interrupts are disabled prior to entering the user code (via JMPM), (see figure 14-1). The user code is entered with the A register set to the TBEVNT value and the X register set to the user code entry address.

Upon completion of processing, the directly connected interrupt handler exits to system routine, V\$DRTN.

Example:

TASK	NAME	TASK
	ENTR	
	STA	EVNT
		Save TBEVNT word
		Do processing
	EXT	V\$DRTN
	JMP	V\$DRTN
		Exit to common processor

where task must be specified on SGEN PIM directive, e.g., PIM,010,TASK,01,2.

14.4.6 VORTEX Use of BICs and BTCs

VORTEX supports a maximum of 15 BICs or BTCs. The practical system limit may be considerably less than ten depending on the availability of device addresses, the type

and number of peripherals, and other configuration considerations. The BIC or BTC transfer complete interrupts must be assigned by ascending BIC or BTC numbers (020, 022, 024, 026, 070, 072, etc.) starting with the first PIM and the first interrupt i.e., PIM 0, line 0 assigned to BIC 020; PIM 0, line 1 assigned to BIC 022, etc. The first BIC must have a device address of 020; the second, 022; the third, 024; the fourth, 026; the fifth, 070; the sixth, 072; etc. Unless the special DEF control directive is used.

I/O drivers utilizing BICs or BTC must call the common BIC routine V\$BIC. The X register is set to the initial buffer address and the A register set to the final buffer address. The call to V\$BIC is:

```

JSR    V$BIC, B
DATA          Interrupt event word or 0 if no
              line handler modification to be
              performed.
DATA          Map number
    
```

14.4.7 VORTEX II and VORTEX Compatibility

User programs written to operate under VORTEX will be operable under VORTEX II under the following conditions:

- a. Programs which contain any RTE service requests or Input/Output Control requests must be assembled by the VORTEX II version of DAS MR. Any program which builds these requests without the DAS MR macros must be modified so that the requests conform to the VORTEX II calling sequence.
- b. Any foreground task which executes hardware I/O instructions except disabling and/or enabling PIMs and RT clock, see section 14.4.4, must be included as part of the resident nucleus when the system is generated. Foreground library tasks which are made resident during system generation by use of the TSK directive are not considered nucleus tasks and therefore must not contain any hardware I/O instructions (see section 14.4.8 for discussion on resident tasks).
- c. Intertask communications can be accomplished: through the use of foreground blank common; by establishing named tables and buffers in the nucleus table module and referencing the named block by an external statement; by use of the RTE PASS request between a user map and map 0; by switching executive mode states (see section 1.3); by sharing the same physical pages utilizing the MAPIN and/or PAGNUM RTE requests.
- d. User tasks (except priority 1 system tasks) may not write into or execute instruction from the first physical page. This page is the VORTEX II low memory area. It is mapped as read-operand only into all user tasks (see figure 2-2), except priority 1 tasks where page 0 is mapped as read-write access mode.

- e. User tasks (non-nucleus) must not communicate with the nucleus except through the use of standard executive service and I/O requests or by referencing entry points which are contained in the core-resident library.
- f. A user task can request a transfer of a block of data from map 0 to the user may by executing a RTE PASS request.
- g. Direct connect interrupt handlers must restore the system to the pre-interrupted map state after servicing the interrupt. An alternative is to utilize the SGEN PIM directive, option 2, as described in section 14.4.5.
- h. I/O drivers written for VORTEX operation must be modified for VORTEX II as follows:
 1. The map number must be passed when calling V\$BIC, common BIC/BTC routine (see section 14.4.6).
 2. The I/O drivers must switch executive mode states (see section 1.3) to fetch/store data from a user map (DCB, FCB, buffer). RQBLK data are stored in map 0 by dynamic memory allocation.
 3. Rotating memory device (RMD) drivers must determine if a data transfer (read, write) I/O request is by SAL (search-allocated-load task). If it is a SAL request, the map number is obtained from TBEVNT of the TIDB for SAL. Otherwise, the requestor's map number is obtained from TBKEY. SAL is the RTE component which loads non-resident tasks into memory. The check may be accomplished as follows:

	LDA	RTIDB, B	RTIDB = word 4 of RQBLK
	SUB	V\$LSAL	V\$LSAL = location 0312 = SAL TIDB
	JANZ	XXX	Jump if not SAL
	LDB	V\$LSAL	Yes SAL Get map key
	LDA	TBEVNT, B	From TBEVNT
	JMP	YYY	Now common processing
XXX	LDB	RTIDB, B	I/O request not by SAL
	LDA	TBKEY, B	Get map key from TBKEY
YYY	ANA	BM17	Mask bits 4-0

- 4. Following a BIC transfer complete interrupt the I/O driver sense for a map memory protection I/O data transfer error:

```
SEN      0101+da, er
```

where da is the BIC device address (which is found in word 011 of the controller table), and er is the address of the error processing routine which must set up an IO46 error code prior to calling V\$ERR.

- i. If a user wants to fetch/store from the nucleus tables, the user must ensure that the nucleus table module is mapped into the user's logical memory. He does this through an external reference to a symbol, TIDB, controller table, etc., within the nucleus module. Example -- have an "EXT TBTYOA."

REAL-TIME PROGRAMMING

- j. TIDBs for non-resident tasks -- except JCP and OPCOM -- are dynamically allocated in map 0. Hence a foreground user task cannot load a register (B,X) from location 0300 (V\$CTL or an address from any other low-core location) and directly fetch the TIDB data. In VORTEX, it is possible; in VORTEX II, such an attempt would result in a memory protect interrupt. The foreground user can fetch the TIDB data by use of the PASS macro. Except for clearing the TBEVNT word, via the RTE TBEVNT request, a foreground user task cannot modify the TIDB.

14.4.8 MICRO-VORTEX (CPU-3) AND VORTEX II COMPATABILITY

The micro-VORTEX option (CPU-3) replaces certain VORTEX II nucleus modules with modules that have been recoded to make BCS (branch to control store) calls to a 2 page firmware set. These modules reduce system overhead by 20 to 40 percent (depending on the system configuration and usage) and reduce the nucleus space requirements by approximately 7 pages. The micro-VORTEX version of VORTEX remains fully user compatible at the RTE and IOC levels but does contain some internal changes that might affect the system level programmer. The following paragraphs describe these changes. The descriptions are not intended to provide full internal information but rather to provide a starting point for further investigation if warranted.

Micro-VORTEX contains unique versions of the following system tasks:

V\$LMEMBK

System Initializers (disc models C, D, E, F H only)

WCS Loading Tasks (replaces WCSRLD)

→ V\$SYTASK

V\$FUNC

V\$SERV

V\$IOC

Whereas these modules have been significantly recoded they still perform the same basic functions. Where full subroutines (i.e., V\$MALC) have been coded in microcode, a software "envelope" still exists that uses the same calling sequence to perform the function of the subroutine. The user may directly use the BCS for the function or he may continue to perform the function using a call to the software envelope. The firmware support for micro-VORTEX requires 2 pages of V77-600 WCS and may reside in either pages 1 and 2 or in pages 3 and 4 (user option). The firmware uses BCS entry points B and C and does not contain any user available micro space.

14.4.8.1 Functional Changes

The memory pool allocation scheme for nucleus dynamic memory has been changed so that dynamic memory fragmentation has been reduced. Nucleus dynamic memory is allocated from logical page 2, up with the thread header located in location 02000 and 02001 (not pointed to by V\$UTB). The thread scheme remains the same as for software VORTEX II. Dynamically allocated blocks of memory are allowed to cross a page boundary. If the block must reside within a physical page then the user should call a new routine V\$MALP to insure such a requirement. Also, physical pages are allocated out of V\$PAGE from the top of memory down instead of the bottom up as in software VORTEX II.

The TSK option and special SAL processing has been removed (from software VORTEX II as well). The Search-Allocate-Load code has been rewritten to be reentrant and runs at the priority of the task being loaded. Thus, a high priority task to be loaded does not have to wait for a low priority task to be loaded as in software VORTEX II. Because the core resident directory (TSK tasks) has been eliminated, the user can no longer schedule a foreground library task by specifying logical unit zero.

Logical page zero is set to read-operand-only mode for all priorities of tasks (it was set to read-write for priority one tasks). This eliminates the special mapping of the nucleus program region into priority one tasks but does prevent such a task from storing into page zero logical space directly. If the user must do this, then he should use the V\$STR subroutine in V\$FUNC.

The background job thread has been eliminated. The threading of TIDB by the TIDB allocation routine has been changed so that tasks of either priority zero or one are threaded after the active background task and the dispatcher prevents the activation of a background task if one is already loaded.

Checkpointing for the background is handled by a separate task. When it is determined that checkpointing is required, a separate resident TIDB is added to the top of the TIDB thread. This task takes care of writing the background to the checkpoint file and setting the proper values in the background TIDB. When it is finished this TIDB is moved to the bottom of the TIDB thread where it is activated to reload the background when memory becomes available. Upon completion of the reload, the checkpointing task exits.

JPDUMP (invoked by either /DUMP or /EXEC,D) runs as a background task after the current active background task has completed I/O and is ready for exiting.

Return:

R0 = Completion status, where

- 1 = ITE busy -- Go to sleep and try again later
- 0 = Successful completion
- 1 = ITE memory pool not initialized
- 2 = Invalid key
- 3 = Map loading error
- 4 = Buffer length too short
- 5 = Page list pending
- 6 = Mailbox queue is empty

R1 = Number of words required if R0 = 4

VI\$GTK

The entry point VI\$GTK is used by a task requesting a mailbox key. No entry parameters are used.

On Exit: R0 will contain the completion status

R0 = Completion status, where

- 1 = ITE busy -- Go to sleep and try later
- 0 = Successful completion
- 1 = Memory not available for extending list
- 2 = No more segments available for extending mailbox list
- 3 = Map loading errors
- 4 = No pages have been specified for the ITE memory pool
- 5 = No physical pages available for memory pool
- 6 = Dynamic memory required for initialization unavailable
- 7 = No space available in map 0 for the ITE memory pool

R1 = Mailbox key if R0 is zero

VI\$GTP

VI\$GTP is used by the receiving task to get the page or pages sent with a message. If the page(s) were deallocated from the sender's map, the pages will be allocated to the receiver (Bit 15 of the corresponding map image word will be set). If the pages were not deallocated by the sender, ITE will map the pages (Bit 14 of the map image word will be set.)

Calling sequence:

R0 = address of the request parameter block, consisting of three words:

- Word 1 = requestor's mailbox key
- Word 2 = logical address from which to begin linking the page or pages
- Word 3 = the number of pages to link

R1 = The number of pages to process

Return:

R0 = Completion status, where

- 1 = ITE is busy -- Go to sleep and try again later
- 0 = Successful completion
- 1 = ITE memory pool has not been initialized
- 2 = Invalid mailbox key
- 3 = Map loading error
- 4 = No page list is pending
- 5 = Logical address or number of pages invalid

R1 = Number of pages processed if R0 = 0, or
The logical address in error if R0 = 5.

VI\$DRP

VI\$DRP is used by the calling task to drop the page list of the current message. This function enables the user to proceed to the next message without first processing the pages associated with the prior message.

Calling sequence:

R0 = Mailbox key

Return:

R0 = completion status, where

- 1 = ITE is busy -- Go to sleep and try again later.
- 0 = Successful completion
- 1 = ITE memory pool not initialized
- 2 = Invalid mailbox key
- 3 = Map error
- 4 = No page list pending

VI\$RMB

VI\$RMB is used by the calling task to relinquish ownership of a mailbox.

Calling Sequence:

R0 = mailbox key of entry to be released or zero if all mailboxes belonging to the task are to be released.

Return:

R0 = Completion status, where

- 1 = ITE is busy -- Go to sleep and try again later
- 0 = Successful completion
- 1 = ITE memory pool has not been initialized
- 2 = Invalid key
- 3 = Map error
- 4 = No mailboxes assigned to calling task
- 5 = A mailbox with an unempty queue encountered

R1 = Key of the mailbox if R0 = 5

REAL-TIME PROGRAMMING

LIMITATIONS, RESTRICTIONS & CONSIDERATIONS

1. Any task establishing ownership of an ITE mailbox will have the responsibility of checking the mailbox for messages prior to exiting and going to sleep.
2. A task which issues an I/O without wait and then issues a delay type 3 will have to determine for itself what event caused to be activated.
3. If a task exits or aborts prior to emptying its mailbox, ITE will release the mailbox elements but no notification will be made to the senders that messages were thrown away.
4. Four possible cases can occur in the disposition of a physical page when a sender requests that the page be transferred by ITE.
 - a. The sender allocated the page and requests that the page be unlinked from his map.
 - b. The sender allocated the page and does not want it unlinked.
 - c. The sender mapped in the page and wants it unmapped.
 - d. The sender mapped in the page and does not want it unmapped.

In the first case, the ownership of the page would be transferred to the receiving task (i.e., the receiving task would be able to deallocate the page and release it to VORTEX). In all the other cases, the page would merely be mapped into the receiving task's logical memory. Therefore, in the event that an abort occurs in the receiving task before it was able to get its pages, ITE will release to VORTEX only those pages which would have been allocated to the receiver. (Case A.)

SYSTEM GENERATION REQUIREMENTS

The following DEF directives are required:

DEF,VISMXQ,n

where

- n Is the number of elements in the main box queue.
The value is placed in bits 9 - 15 and must be octal i.e., a value of 10 would be represented by DEF,VISMXQ,012000.

DEF,VISNPG,m

where

- m Is the number of physical pages to be used for ITE internal pool. This number of pages is made unavailable to VORTEX.

14.6 MEMORY PARITY CONSIDERATIONS

This paragraph describes VORTEX software handling of parity errors.

14.6.1 Memory Parity Considerations (V70/V77-600)

If it is desired to halt operations upon detection of a parity error in the nucleus, six words are provided, starting at V\$PERA, in which to place a halt patch.

If the parity routines are not desired, definitions for V\$PERR and PARANY must be made.

14.6.2 Memory Parity Considerations (V77-200/400)

Memory parity detection is a supported feature under VORTEX on the V77 model 200/400. A parity error is reported by an EX35 error message (see appendix 11.2). The map tracking register contains the address being fetched when the parity error occurred. Memory parity is a feature of the SYSGEN selection parameter.

14.6.3 V77-600 Parity Error Handling

Parity Error Handling software on V77-600s consists of the following components:

- A nucleus module named V\$PERR.
- When ERCC is used, library load modules: DMPPAR and INITPR on the background and PAROUT, PARCHK, and PERCHK on the foreground.
- • An error log file named PARITY on logical unit FL, containing 2 records each of which is 120 words long.

The nucleus module V\$PERR is created during SYSGEN. The library load modules DMPPAR, INITPR, PAROUT, PARCHK and PERCHK and the error log file PARITY can be loaded on disk by executing a segment of the post-SYSGEN job stream.

There are two types of parity errors associated with ERCC memory: single-bit (correctable) and double-bit (uncorrectable). Single-bit errors are transparent to the user except in the case of the log file overflow. Double-bit parity errors result in the current task being aborted and the operator being informed of an error.

Single-bit errors are ignored by the system, and automatically corrected by the hardware. The foreground task PARCHK will request the operator to input a testing interval, initialize the log file and schedule task PERCHK, which will do the actual testing and logging of single-bit errors. PARCHK (never PERCHK) should always be scheduled for single-bit error logging. The log file may be

reinitialized at any time by the execution of the background task INITPR. The file may be displayed at any time by the execution of another background task named DMPPAR. The output from DMPPAR is in the following format:

```

PARITY ERROR LOG      MM/DD/YY      HH/MM HOURS
ERROR COUNT = xxxxx

BOARD      MODULE      SYNDROME
  X                X          XXX
                .
    
```

The entries under BOARD and MODULE indicate the board number and module number where the parity error occurred. The entries under SYNDROME are a list of parity bit configurations which indicate the failing bit of a single bit error. For additional information, refer to Section 3 of the V77-600 128K Error Correction Memory Operation and Service Manual (UP-9008).

Whenever the log file overflows, a message appears on the console to inform the operator of the condition. When this message appears, the operator should (1) display the current log file, (2) reinitialize the log file, and (3) continue system operation with single-bit parity checking disabled (by aborting task PERCHK). Step (3) is desirable if single-bit errors occur in clusters. The PARCHK task might only be activated on a scheduled basis rather than continuously, to avoid unnecessary system overhead.

Double-bit errors or parity memory errors, if they occur during execution of either the dispatcher or the initializer, cause an error message to be displayed. Otherwise, the current task will be aborted, and a message of the following format will appear on the console:

```

DOUBLE BIT PARITY ERROR IN TASK XXXXXX

BOARD NUMBER = xx.  MODULE NUMBER = xx
    
```

Due to ERCC hardware design, the information in the above message may be incorrect if single-bit errors and double-bit errors occur together. The message, however, will only be displayed when double-bit errors occur, even if the contained information is incorrect.

Note: It is advisable to **always** schedule and run PARCHK to log single-bit errors, in order to take advantage of the ERCC hardware.

14.6.4 V77-800 Parity Error Handling

Parity-error-handling software on V77-800s consists of the following components:

- A nucleus module named V\$PERR.
- When ERCC is used, library load modules: DMPARR and INITPR on the background, and PAROUT, PARUP, and PARDWN on the foreground.
- An error log file named PARITY on logical unit 190, containing 2 records each of which is 120 words long.

V\$PERR is created during SYSGEN, and the other components can be loaded on disk by executing a segment of the post-SYSGEN job stream.

There are two types of parity errors: single-bit (correctable) and double-bit (uncorrectable). Single-bit errors are transparent to the user except in the case of the log file overflow. Double-bit parity errors result in the current task being aborted or the system being halted.

Upon entering any interrupt handler, single-bit parity errors are suppressed by hardware while double-bit errors will result in the machine going into step mode and the message "PE" appearing on the console.

For single-bit errors, the handler enters a one word information (containing the board number, the module number and syndrome pattern) in a system-resident queue and wakes up a resident task PARERR which, when executed, writes the information on the log file. The interrupt handler then resumes execution of the interrupted task. The log file may be reinitialized at any time by the execution of the background task INITPR. The file may be displayed at any time by the execution of another background task named DMPPAR. The output from DMPPAR is in the format

```

PARITY ERROR LOG      MM/DD/YY      HHMM HOURS
ERROR COUNT = XXXX

BOARD      MODULE      SYNDROME
  X                X          XXX
                .
                .
                .
    
```

The entries under BOARD and MODULE indicate the number of the board and module where the error occurred. The entry under SYNDROME is the error syndrome pattern for single bit errors. Refer to Section 4 of the V77-800 Memory System Functional Analysis and Servicing Manual (UP-8704) for additional information.

Whenever the log file overflows, a message appears on the console to inform the operator of the overflow condition, upon which he should 1) display the current log file, 2) reinitialize log file, and 3) continue system operation with single-bit parity disabled (by executing the foreground task PARDWN). Step (3) should be executed if single-bit errors occur in clusters. The foreground task PARUP is available for re-enabling single-bit parity.

Double-bit errors, if they occur during execution of either the dispatcher or the initializer, cause the system to halt. Otherwise, the current task will be aborted, and a message of the following format appears on the console:

```

DOUBLE-BIT PARITY ERROR IN TASK XXXX
BOARD NUMBER = XX.  MODULE NUMBER = XX
    
```



SECTION 15 SYSTEM GENERATION

The VORTEX **system-generation component (SGEN)** tailors the VORTEX operating system to specific user requirements. SGEN is a collection of program on magnetic tape, punched cards, or disc pack. It includes all programs (except the key-in loader, section 15.3) for generating an operating VORTEX system on an RMD.

Figure 15-1 is a block diagram of the data flow through SGEN.

15.1 ORGANIZATION

SGEN is a five-phase component comprising:

- I/O interrogation (section 15.4)
- SGEN directive processing (section 15.5)

- Building the VORTEX nucleus (section 15.6)
- Building the library (section 15.7)
- Resident-task configuration

I/O **interrogation** specifies the peripherals to:

- a. Input VORTEX system routines (LIB unit)
- b. Input user routines (ALT unit)
- c. Input SGEN directives (DIR unit)
- d. Output the VORTEX system generation (SYS unit)
- e. List special information and input user messages (LIS unit)

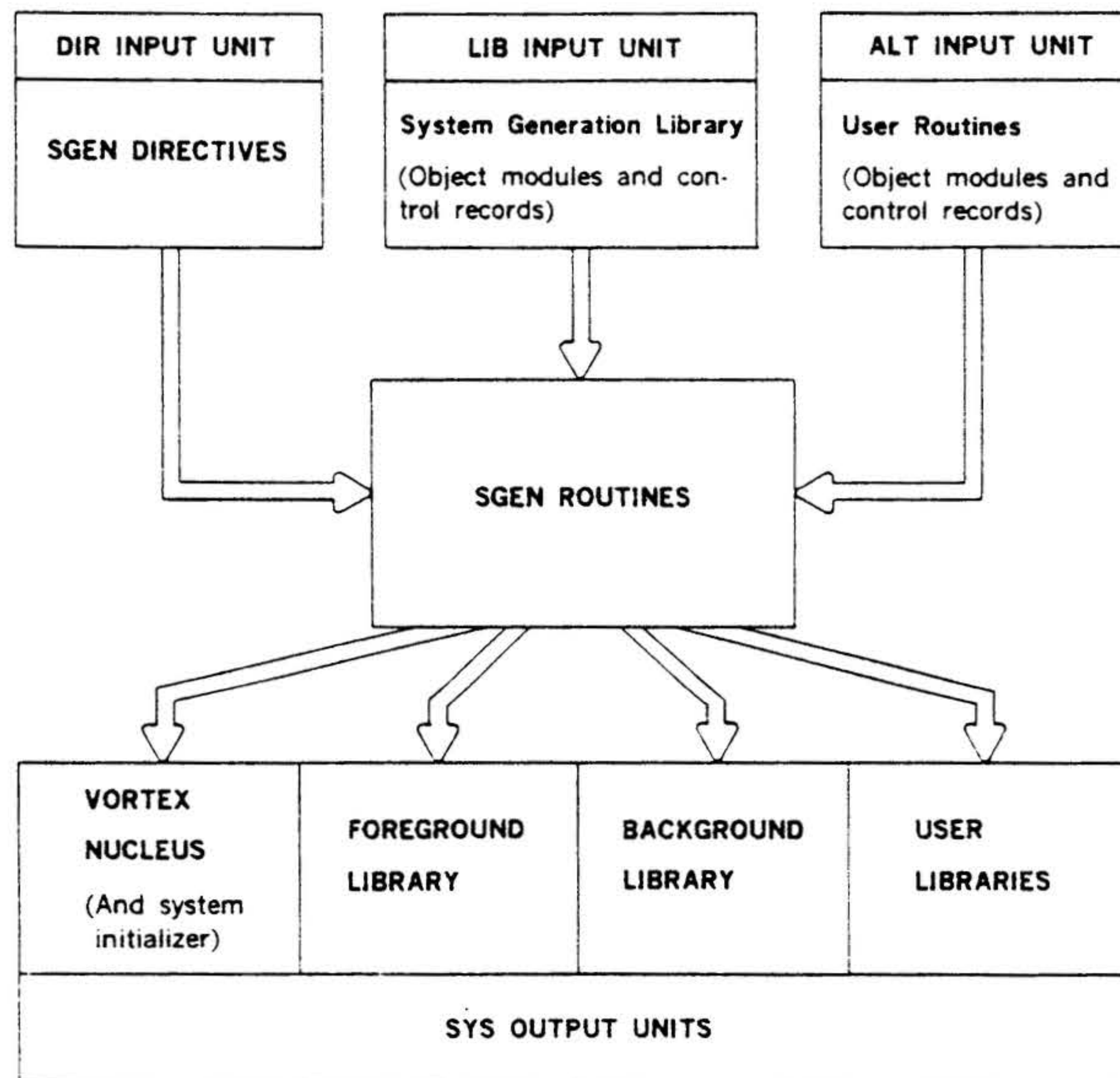


Figure 15-1. SGEN Data Flow

SYSTEM GENERATION

I/O interrogation also specifies that the Teletype on hardware address 01 is the OC unit. After these peripherals are assigned, appropriate drivers and I/O controls are loaded into memory.

Note: SGEN does not build an object-module library. To construct the VORTEX object-module library (OM) or any user object-module library, use the file-maintenance component (FMAIN, section 9).

SGEN directive processing specifies the architecture of the VORTEX system based on user-supplied information that is compiled and stored for later use in building the system. SGEN directives permit the design of systems covering the entire range of VORTEX applications.

Building the VORTEX nucleus consists of gathering object modules and control records from the system-generation library (SGL, section 15.2) and from user input, and constructing the VORTEX nucleus from these data. SGL items are input through the LIB input unit, and user items through the ALT unit according to rules set up by the SGEN directives.

Building the library and the resident-task configurator consists of generating load modules from the object modules and control records input from the SGL and user data. These load modules are then cataloged and entered into the foreground, background, and user libraries. During library building, load modules can be added, deleted, or replaced as required to tailor the library to any of a wide variety of formats. After the libraries are completed, designated load modules are copied into the VORTEX nucleus to become *resident tasks*. The resident-task configuration of SGEN can also be generated without regeneration of the VORTEX nucleus or libraries (section 15.7).

SGEN directive format requires that, unless otherwise indicated (e.g., section 15.5), the directives begin in column 1 and comprise sequences of character strings having no embedded blanks. The character strings are separated by commas (,), equal signs (=) or semi-colons (;). The directives are free-form and blanks are permitted between individual character strings, i.e., before and after commas, equal signs, or semi-colons. Although not required, a period (.) is a line terminator. Comments can be inserted after the period. For greater clarity in the descriptions of the directives, optional periods, optional blank separators between character strings, and the optional replacement of commas by equal signs are omitted. Section 14.4.8 describes resident tasks in greater detail.

Numerical data can be octal or decimal. Each octal number has a leading zero.

Error messages applicable to SGEN are given in Appendix A.15.

SGEN errors are divided into five categories according to type. The category of each error, as well as the specific error, is given by the error code. Recovery is automatic where manual intervention is not required. When manual intervention is necessary, the OC device expects a response after the error message is posted. The response can be either a corrected input statement (where the statement in error was an ASCII record) or the letter "C". In the latter case, the corrected input is expected on the input device where the error occurred, immediately after the "C" is input. If the input media is magnetic tape or disc pack, positioning to reread an input statement is also automatic.

15.2 SYSTEM-GENERATION LIBRARY

The **System-generation library (SGL)** is a collection of system programs (in object-module form) and control records (in alphanumeric form) from which a VORTEX system is constructed.

In the case of punched cards or of magnetic tape, the SGL occupies contiguous records, beginning with the first record of the medium.

In the case of disc pack, the SGL occupies contiguous records beginning with the second track. Track 0 contains the partition-specification table (PST, section 3.2) that specifies one partition extending from the second track (track 1) to the end of device.

The SGL and the VORTEX system cannot be on the same disc pack during system generation.

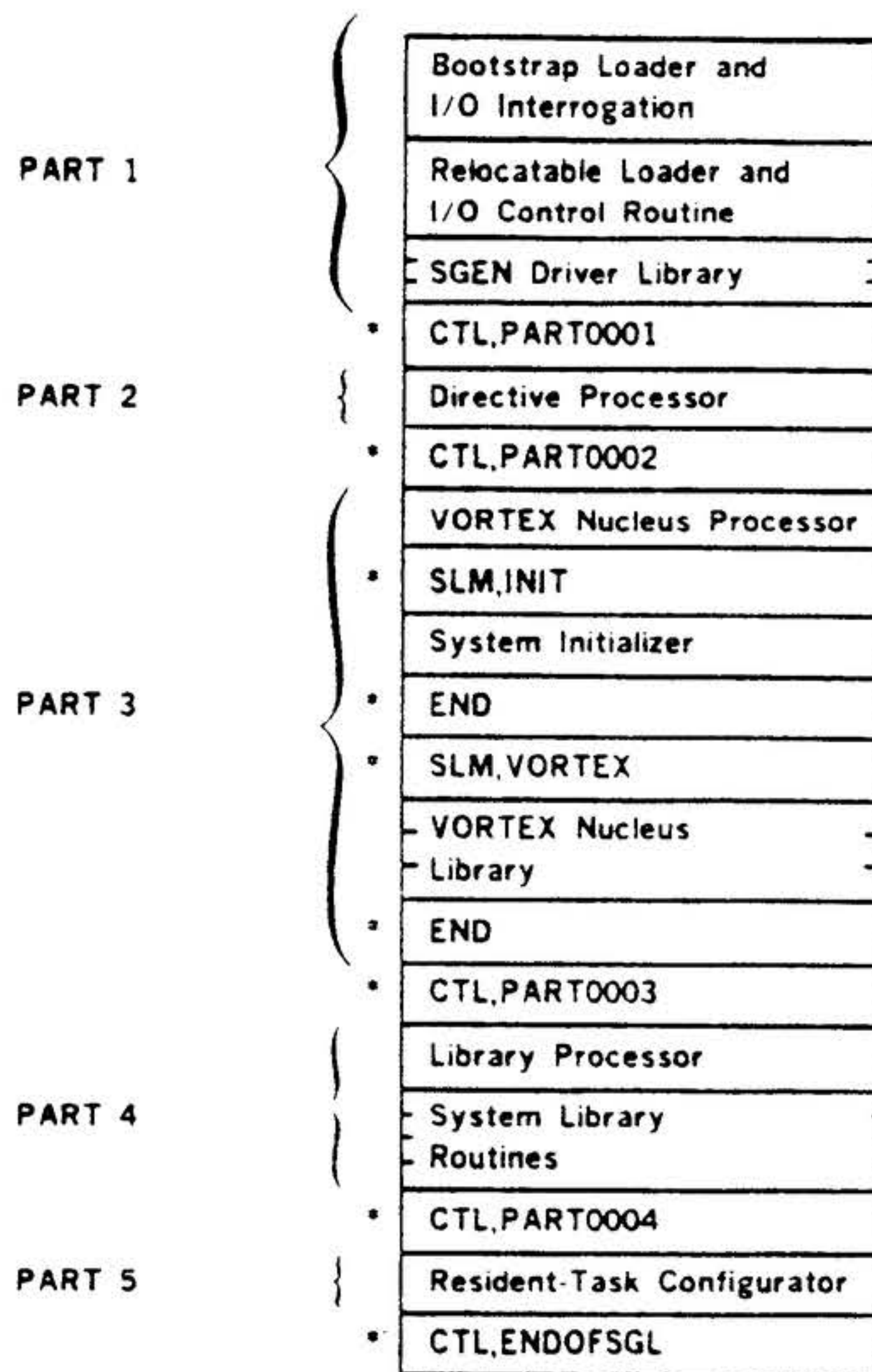
The SGL is divided into five functional parts, each separated by CTL control records (figure 15-2).

Part 1 of the SGL comprises a *VORTEX bootstrap loader* and an *I/O interrogation routine*. It also comprises the *SGEN relocatable loader*, the *basic I/O control routine*, and *library of peripheral drivers* for the use of SGEN. Part 1 consists entirely of object modules. It is loaded with device-sensitive key-in loader (section 15.3) that also serves the bootstrap loader as a read-next-record routine. The bootstrap-loader/interrogator is a core-image sequence of records generated by a VORTEX service routine. Because it calls the key-in loader to read records, the bootstrap-loader/interrogator is itself device-insensitive.

Control record CTL,PART0001 terminates part 1 of the SGL.

Part 2 of the SGL contains the *directive processor*. After being itself input, the directive processor obtains all input from the DIR and OC input devices. The system generation directives are to be placed between the directive processor and the CTL,PART0002 control record if the CIB and DIR input units are the same.

The control directive 'CPU.n' is used in conjunction with the SYSGEN CPU directive in selecting the proper mod-



NOTE:

- = Alphanumeric control record

Figure 15-2. System-Generation Library

ule where multiple versions exist. The 'n' specifies the number of versions of the module that exist. These versions must be contiguous. The value of 'n' is used to derive the module skip count before and after the selected module is processed.

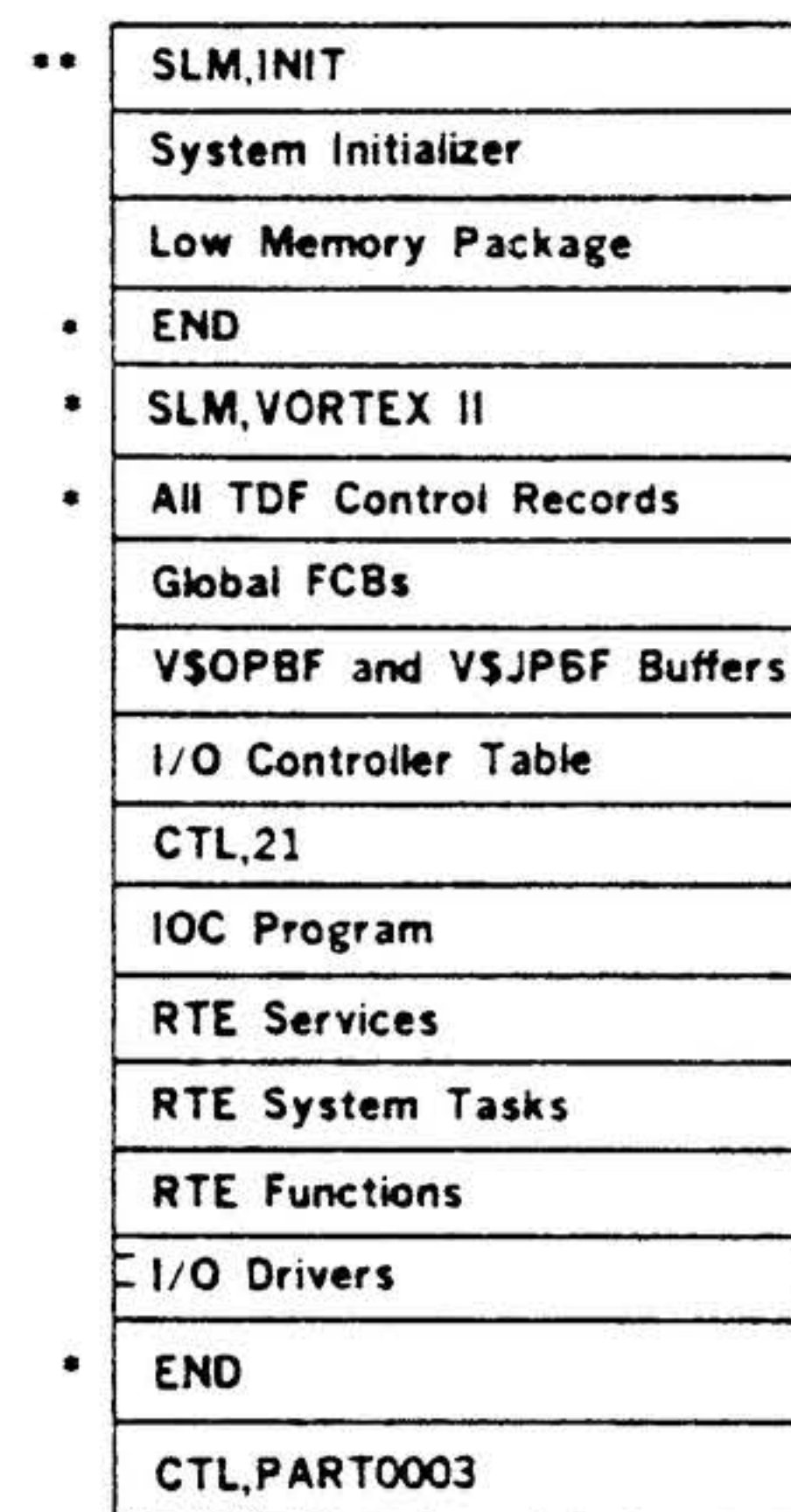
Note: The value of 'n' may be less than the value 'n' of the SYSGEN CPU directive. Such a case may be used to indicate that none of the versions of the module apply to the specified system.

Control record CTL,PART0002 terminates part 2 of the SGL.

Part 3 of the SGL comprises all system routines and control records required to build the VORTEX nucleus (figure 15-3):

- VORTEX nucleus processor -- the SGEN-processing portion
- SLM control record -- indicates the beginning of the system initializer portion
- System-initializer routines -- object modules to be converted into the system initializer

- END control record -- indicates the end of the system-initializer portion
- SLM control record -- indicates the beginning of the VORTEX nucleus portion
- VORTEX nucleus routines -- control records and object modules to be converted into the VORTEX nucleus
- END control record -- indicates the end of the VORTEX nucleus portion
- Control Record CTL,21 -- specifies the end of the nucleus table module. All user data and programs to be included in this module must precede the CTL,21 control record.
- All programs contained on the SGL between the CTL,21 and CTL,PART0003 control records are included in the nucleus program module



NOTE:

- = Alphanumeric control record

Figure 15-3. VORTEX Nucleus

Control record CTL,PART0003 terminates part 3 of the SGL.

Part 4 of the SGL comprises all system routines and control records required to build load-module libraries on the RMD. The library processor converts these inputs into load modules, catalogs them, and enters them into the foreground, background, and user libraries. The library processor is followed by groups of control records and object modules, with each group forming a load-module package (LMP).

SYSTEM GENERATION

**REQUIRED
(FOREGROUND)
SYSTEM
TASKS**

◦	SLM,FV\$OPC
◦	TID,V\$OPCM,2,8,106
	V\$OPCM Program
◦	ESB
◦	END
◦	SLM,FJCDUM
◦	TID,JCDUMP,2,0,106
	JCDUMP Program
◦	ESB
◦	END
◦	SLM,FRAZI
◦	TID,RAZI,2,0,106
	RAZI Program
◦	ESB
◦	END

◦	SLM,BFORT
◦	TID,FORT,1,0,105
	FORTRAN Compiler
◦	ESB
◦	END
◦	SLM,BCONC
◦	TID,CONC,1,0,105
	Concordance Program
◦	ESB
◦	END
◦	SLM,BIOUTI
◦	TID,IOUTIL,1,0,105
	I/O Utility Program
◦	ESB
◦	END

**REQUIRED
(BACKGROUND)
SYSTEM
TASKS**

◦	SLM,BJCP
◦	TID,JCP,1,0,105
	Job-Control Processor
◦	ESB
◦	END
◦	SLM,BLMGEN
◦	TID,LMGEN,1,0,105
	Load-Module Generator
◦	ESB
◦	END
◦	SLM,BFMAIN
◦	TID,FMAIN,1,0,105
	File Maintenance
◦	ESB
◦	END
◦	SLM,BSMAIN
◦	TID,SMAIN,1,0,105
	System Maintenance
◦	ESB
◦	END

◦	SLM,BSEEDIT
◦	TID,SEEDIT,1,0,105
	Source Editor
◦	ESB
◦	END
◦	SLM,BDASMR
◦	TID,DASMR,1,0,105
	DAS MR Assembler
◦	ESB
◦	END

NOTE:

- = Alphanumeric control record

Figure 15-4. Load-Module Library

Control record CTL,PART0004 terminates part 4 of the SGL.

Part 5 of the SGL contains the resident-task configurator portion of SGEN. The configurator copies specified load modules from the foreground library into the VORTEX nucleus, i.e., makes them resident tasks.

Control record CTL,ENDOFSGL terminates the SGL.

15.3 KEY-IN LOADER

SGEN is initiated on a new or initialized system by inputting the key-in loader through the CPU. The key-in

loader loads the VORTEX bootstrap loader (part 1 of the SGL). Key-in loaders are available for loading from magnetic tape, punched cards, or disc pack. The required key-in loader is input to memory through the CPU console and then executed to load the VORTEX bootstrap loader.

Automatic bootstrap loader (ABL): In systems equipped with an ABL, load the key-in loader from the input medium into memory starting with address 000000. To execute the key-in loader, clear the A, B, X, I, and P registers; then press RESET, set STEP/RUN to RUN, and press START.

See hardware handbook for details on manual loading.

Table 15-1. SGEN Key-In Loaders

Address	Magnetic Tape	Card Reader	RMD 70-76x0	RMD 70-76x3
000000	010030	010054	010064	010064
000001	001010	001010	140066	140066
000002	001114	001114	001010	001010
000003	040030	040054	001114	001114
000004	001000	001000	001000	001000
000005	000012	000012	000012	000012
000006	000000	000000	000000	000000
000007	006010	006010	006010	006010
000010	000300	000300	000300	000300
000011	050027	050053	050065	050065
000012	1041zz 22/0	1002zz	1004zz	1004zz
000013	1000zz 22/0	002000	1002zz	010063
000014	001000	000046	010063	110072
000015	000021	1025zz	110072	1031zz
000016	1025zz 22/0	002000	1031zz	1002zz
000017	057027	000046	101uzz	101dzz
000020	040027	1026zz	000023	000023
000021	1011zz 22/6	004044	001000	001000
000022	000016	004444	000017	000017
000023	1012zz 22/0	057053	1025zz	1025zz
000024	100006	005001	150071	150071
000025	001000	040053	001016	001016
000026	000021	004450	000012	000012
000027	000500	002000	1000yy	1000yy
000030	177742	000046	1003zz	5000
000031		1026zz	010064	010064
000032		004044	110072	110072
000033		004450	1031zz	1031zz
000034		002000	010065	010065
000035		000046	1031xx	1031xx
000036		1022zz	120070	120070
000037		057053	005012	005012
000040		040053	1031yy	1031yy
000041		067053	1000xx	1000xx
000042		040053	1000zz	1000zz
000043		001000	1014zz	1014zz
000044		000013	000043	000043
000045		1011zz	1025zz	1025zz
000046		000000	150071	150071
000047		1016zz	001016	001016
000050		100006	000012	000012
000051		001000	060065	060065
000052		000045	040064	040064
000053		000500	010064	010064

SYSTEM GENERATION

Table 15-1. SGEN Key-In Loaders (continued)

Address	Magnetic Tape	Card Reader	RMD 70-76x0	RMD 70-76x3
000054		177742	140067	140067
000055			001016	001016
000056			100006	100006
000057			050064	050064
000060			040063	040063
000061			001000	001000
000062			100006	100006
000063			000001	000001
000064			000001	000001
000065			000500	000500
000066			000037	000037
000067			000060	000060
000070			000074	000074
000071			007760	007760
000072			0v0000	ww0000

where

- xx = even BIC address
- yy = odd BIC address
- zz = device address
- u = RMD unit number in Sense Instruction
 - u = 0 for unit 0
 - u = 1 for unit 1

- v = RMD unit number in unit Select Instruction
 - v = 0 for unit 0
 - v = 4 for unit 1
- d = RMD drive number (0-3)
- ww = drive (bits 15-14) /platter (bit 13)
 - (i.e., platter 1 drive 0 = 02)

Note: To SGEN VORTEX II VNO from magnetic tape or cards, change bootstrap location 2 from 1106 to 1114; if from RMD, change location 3 from 1106 to 1114.

15.4 SGEN I/O INTERROGATION

Upon successful loading of the bootstrap loader and I/O interrogation, the OC unit outputs the message

IO INTERROGATION

after which the SGEN peripherals are specified by inputting on the OC unit the five I/O directives:

- DIR Specify SGEN directive input unit
- LIB Specify SGL input unit
- ALT Specify SGL modification input unit
- SYS Specify VORTEX system generation output unit
- LIS Specify user communication and list output unit

These directives can be input in any order. SGEN will continue to request I/O device assignments until valid ones have been made for all five functions.

SGEN drivers are loaded from the SGEN driver library according to the specifications of the SGEN I/O directives. Errors or problems with reading the drivers will cause the applicable error messages (Appendix A.15) to be output.

The general form of a SGEN I/O directive is

function = driver,device,bic

where

- function is one of the directive names given above
- driver is one of the driver names given below
- device is the hardware device address
- bic is the BIC address

Name*	Type of Device	Model Numbers
MTcuA	Magnetic-tape unit	70-7100
LPcuA	Line Printer	70-6701
LPcuB	Line Printer	
LPcuD	Status 31	70-6602
LPcuK	Status 33, 41 or 42	
CRcuA	Card reader	70-6200
CPcuA	Card punch	
PTcuA	Paper-tape read/punch	70-6320
TYcuA	Teletype or CRT	70-6100, 70-6104
DcuA1	Rotating memory	70-7702
DcuA2	Rotating memory	70-7703
DcuA5	Rotating memory	620-49
DcuB	Rotating memory	70-7600, 70-7610

15.4 PATCH IMAGE FILE FORMAT

The control directives `.DUMP`, `.APND`, and `.HIST` use the patch image file for building and listing patches. The patch image file must meet the following criteria:

- The file is supplied by the user prior to its first use and must reside in the system foreground library. The file name must be `PTCHIM`.
- The file must be large enough to hold all the patches that are to be made on an automatic basis. Each patch image record is 120 words in length and can hold 40 patch entries.

Each patch entry consists of a triplet with the following contents:

Word	Contents
1	Effective change address
2	Original contents of that address
3	New contents of that location

In order to indicate that a patch image file is valid, the first entry in the patch image file consists of a triplet containing a change address value and new contents value of `-2 (0177776)`. This entry is placed in the file by the execution of either the `.DUMP` or `.APND` control directives. The user does not supply this entry. The final triplet of the patch image file consists of a change address and new contents of `-1 (0177777)`. This entry is placed in the file by the termination of the `.DUMP` or `.APND` control directives. The user does not supply this entry.

Virtual nucleus overlay (VNO) control blocks may be embedded in the patch image file. The format of the triplet consists of word 1 = `-3`, word 2 = `0`, and word 3 = the TIDB address of the TIDB associated with the VNO task being patched. These VNO control triplets are not listed by the `.HIST` control directive and are not supplied by the user.

The second word of a patch image triplet is set by the execution of the foreground program `BTPTCH` and hence, will remain zero until `BTPTCH` is executed. This means that requesting a history of the patch image file prior to the execution of `BTPTCH` will produce a history output with invalid entries for the original contents of the effective location.

Note: Terminating `PATCH` while under the control of `.DUMP` or `.APND` with the `OPCOM` directive `;ABORT` may leave the patch image file without the proper terminator. Always terminate these directives with another `PATCH` control directive.

15.5 PATCH DIRECTIVE LOG FILE FORMAT

The `PATCH` directive log file must reside on the foreground library and have the name `"PDLOG"`. All permanent `PATCH` directives (both control and change) are logged to this file in "card image" format (one record per logical sector). Each entry

is also date and time stamped (ASCII format) in character positions 81 through 100. The last logged record is always followed by an EOF. Invocation of `PATCH` opens the log file (no rewind) and the `.EXIT` directive closes the file (with update). If the `PATCH` program is aborted instead of using the `.EXIT` command, the EOF pointer in the log file will not be updated and all patch activity logging for the `PATCH` session will be lost.

The `.HIST` directive may be used to list the contents of the log file. However, any ASCII list program (such as `IOUTIL`) may also be used. The contents of the `PATCH` log file may be used as inputs to the `PATCH` program for the target system or another identical system. The log file does not contain any patch directives that were rejected under `.SLCT` or `.MANL` control (except for the actual `.CNTL` directive), any temporary patches made under `.PTCH` control, any `.HIST`, `.EXIT`, or `.READ` directives, and thus is not a complete image of the entire "universal" patch directives.

The size of the `PATCH` directive log file, `PDLOG`, depends on the estimated patching of the target system. A minimum of 500 records is recommended. Using multiple change values on a single change directive will help conserve space on the log file. When a new file is created, it should be initialized with the directive `.RECD,INIT` prior to performing any patch activity on the system.

15.6 BTPTCH INTERFACE

The `PATCH` program has a companion program, `BTPTCH`, which utilizes the patch image file to perform patches without user intervention. This program resides on the system foreground library and can be scheduled in one of two ways.

- If `SSW1` is not set before or during the execution of the `VORTEX` system bootstrap, `BTPTCH` is set active when the system is activated.
- `BTPTCH` can be scheduled with the `OPCOM` `;SCHED` directive any time after the patch image file is created.

The first method allows a user to apply selected patches, without the need to enter patches with the `.PTCH` control directive, whenever the `VORTEX` system is loaded. The second method allows the user to select when the patches are to be made and make the patches without using the conversational mode of `PATCH`. The second method also allows the use of multiple patch image files by using `FMAIN` to rename the appropriate files, and then scheduling `BTPTCH`. When using `BTPTCH` by the second method, the user should ensure that the patches are not already in the nucleus. If the patches are in the nucleus, the second entry (original contents) of the history image will reflect the new value instead of the original value, since `BTPTCH` updates this entry every time it is executed.

`BTPTCH` does not have any effect on patches made to a load module file under control of the `.LIBR` directive.

`BTPTCH` opens and rewinds the patch image file, `PTCHIM`, checks for a valid file indicator, and performs the patches using

THE PATCH PROGRAM

the patch entry triplets until the end of image indicator is encountered. When patching is completed, the message

PATCHING COMPLETED

is output to the OC logical unit. If the patch image file is empty or does not contain a valid file indicator, no patching is done and the previous message is output.

If the file PTCHIM does not exist, the message

FILE PTCHIM NOT FOUND

is output to the OC logical unit and BTPTCH terminates. If the patch image file is invalid, BTPTCH terminates and does not output any message to the OC logical unit.

15.7 SYSTEM GENERATION INTERFACE

PATCH and its associated programs are included in the standard VORTEX II system generation libraries. If these programs are to be in the system, no user action is required. If these programs are not to be in the system, then they may be deleted by the SYSGEN directives:

**DEL,V\$STWD
LDE,PATCH,BTPTCH**

Note: The nucleus module of PATCH consists of less than 040 words. The PATCH program and its related programs may be required if a user requires a patch supplied by Sperry Univac. Hence, these modules should not be deleted unless it is absolutely necessary.

When patching the VORTEX nucleus, it may be necessary to jump out of in-line code to make a patch. Such changes require that the system have an area included within the nucleus region that may be used as a patch space. The SYSGEN DEF directive:

DEF,V\$PTSZ,n

where:

n

is the number of words to be reserved

↓
can be used to guarantee that a patch space of n words is available. The patch space is created at the bottom of the VORTEX nucleus, just above the VNO region. The first word is specified by the CL name V\$PSTR. The last word is specified by the CL name V\$PEND. Neither PATCH nor its related tasks ↑
limit the user to this area. However, the CL pointers are available to the user.

The word associated with V\$PSTR is always set (by system generation) to the bottom of a memory page + 1 and hence, the size of the reserved patch area may exceed the size requested with the DEF directive. If nucleus memory space is critical, care should be taken that the requested patch area size does not cause the nucleus to overflow into another memory page.

If the DEF directive is not used, the CL label V\$PSTR is still generated and a patch space is created down to the bottom of the current last page of the nucleus. This means that some patch space may exist even if the DEF directive is not used. Of course, the size of this space may vary from 0 to 511 words.

In order to have all patch activity logged, the file PDLOG must be created on the foreground library. If this file is not created, PATCH will automatically disable directive logging.

15.8 PATCHING CONSIDERATIONS

The following considerations should be made when using the PATCH program:

- Whenever possible, the CL directory name should be used instead of an actual numeric number. This allows a patch to function independently of a particular system generation.
- When making patches that cause a jump out of the system code, it is advisable to create the patch area first and then patch in the jump to the patch area.
- When patching in a jump, make sure that both the jump instruction and the jump address are entered in the same PATCH change directive.
- Make patches while the system is in as calm and predictable state as possible to avoid possible complications while entering a patch.
- Use the .BASE directive to set a base symbol value when making patches to an external subroutine of a load module file.
- When entering more than one patch that is making use of the patch space generated by system generation, make sure that the patch space contents do not overlap. PATCH does not handle this for the user.
- Special diagnostic patches can be kept on an alternate foreground library file which can be renamed. When these patches are needed, BTPTCH can be scheduled without using the conversational mode of PATCH.
- When an asterisk is used as a defined base symbol, it should be remembered that the asterisk will keep its base value until it is reset. This may affect other patches.
- When a VNO task is to be patched, its entry name should be used--never its address or other task labels.
- The PATCH directive stream is in "card image" format and thus can be modified or added to by using any suitable line editor. This allows the

user to revise patches and add new patches to the system patch stream. When adding new patches, be sure to include the appropriate .CNTL directives and terminate the patches with a .EXIT directive if the patch stream is to be used under control of .SLCT or .MANL.

If the load module to be patched contains an ORG statement, all addresses after the ORG value are absolute (A appears in the type field of the listing instead of R). When a relocatable address is specified to PATCH with a base symbol, the address provided is the address minus 01000. For example, to patch in an address that is absolute due to an ORG statement the user can enter

*+addr-1000

If this format is not followed, the PATCH program will generate too large a value for the address upon adding 01000 to the supplied value. The user can specify an absolute address by not using a base symbol; in this case, the PATCH program will not add a relocation value to the supplied address.

When a load module is to be patched, the user should note that load map addresses are absolute (they already include a relocation value). The PATCH program adds in a relocation value to any address that includes a base symbol and to address expressions that start a change line. When load map addresses are used, the appropriate relocation value must be subtracted out of the provided address (for instance, the address for start of overlays, size of the overlay directory, etc.)

15.9 ERROR MESSAGES

Error messages output by PATCH have the following format:

PTnn, error

where:

nn

Is the error number

error

Is a brief descriptor of the error

In addition to the PTnn error messages, VORTEX may also generate an IO10 PATCH diagnostic if a directory search error is caused.

All error messages and error recovery are directed to the SO logical unit. After posting the appropriate error message, PATCH outputs the prompt PT** and awaits the appropriate recovery. Two options exist for error recovery:

- Enter a C on the SO logical unit to indicate that the directive in error is to be reread from the DI logical unit. Repositioning is automatic for magnetic tape and RMD devices.
- Enter the entire corrected directive on the SO logical unit. Upon completion of processing of the directive entered from SO, subsequent directives are read from the DI logical unit.

An error recovery directive may be either a control directive or a change directive regardless of the type of directive that caused the error.

PATCH error messages are listed in Appendix A.15.

15.10 EXAMPLES

The following examples illustrate several methods of making a patch to an operational VORTEX system.

Example 1: Define (reserve) a patch area of 300 (decimal) words in the VORTEX nucleus region.

Enter the following system generation directive:

DEF,V\$PTSZ,300

Example 2: Change the nucleus locations 200,201,204, and 206 to contain 1,2,3, and 4 respectively.

Enter the following PATCH directives:

.PTCH
200,1,2,..3,..4

Example 3: Continue the patch address into the next change directive.

The change directives might be:

200,1,2,..3,..4
.5,6,7

(This would modify locations 200,201,204,206,207,210, and 211.)

Example 4: Place a patch in the nucleus program PROG1 to add a bit test after location 0123 relative to the start of the program. The code at location 0124 and 0125 must be saved and currently contains a LDA 4,X and a STA 9,B. Use the patch area created by system generation to hold the patch. The entry name PROG1 is located at the first loaded location of the program PROG1.

The PATCH directives might be:

.PTCH
V\$PSTR,6451,PROG1+145,15004,56011
1000,PROG1+126
PROG1+124,1000,V\$PSTR
.EXIT

THE PATCH PROGRAM

Example: Make the same patch as in example 4 but use the instruction mnemonics.

The change directives might be:

```
.PTCH
V$PSTR,BTA0+11,PROG1+145,LDAX+4
,STAB+11,JMP,PROG1+126
PROG1+124,JMP,V$PSTR
.EXIT
```

Example 6: Change locations 0245 and 0246 of the load module file on logical unit 25 (no protect key), filename TEST to a jump to program location 0310.

The PATCH directives might be:

```
.LIBR
25,,TEST
245,JMP,*+310
.EXIT
```

Example 7: Make another change to the program TEST in example 6 which places a jump at location 0276 to location 6 of the external subroutine SUBR that is linked with TEST on the load module file.

From the LMGEM map it is determined that subroutine SUBR is loaded at relative location 0544. The patch in example 6 has already been made.

The following PATCH directives might be used:

```
.LIBR
25,,TEST
276,JMP,*+544+6
```

or

```
.BASE
$=544
.LIBR
25,,TEST
276,JMP,$+6
```

Example 8: Display the current BASE value.

The PATCH directives would be:

```
.BASE
=
```

Example 9: Change location 0245 of the OPCOM overlay OPPRO to contain the relative address of 0247.

The PATCH directives might be:

```
.LIBR
FL,F,V$OPCM,OPPRO
245,*+247
.EXIT
```

Name*	Type of Device	Model Numbers
DcuC	Rotating memory**	70-7500
DcuD	Rotating memory**	70-7510
DcuE	Rotating memory**	70-7520, 70-7530
DcuF***	Rotating memory**	70-7603

* where c stands for the controller number (0, 1, 2, or 3), and u for the unit number (0, 1, 2, or 3).

**Always specify the first master unit of a particular device as being on controller 0, the second master unit on controller 1, etc. Regardless of the controller specifications in the EQP directives, different controller numbers must be used for each RMD type. (i.e., if using MT 1 on DA 12, specify MT00A). If the system has a 7600 and 7500 RMD, then specify D00B and D10C.

*** Unit number u is a value in the range 0 through 7.

15.4.1 DIR (Directive-Input Unit) Directive

This directive specifies the unit from which all SGEN directives (section 15.5) will be input (DIR unit). The directive has the general form

DIR = driver,device,bic

where

driver	is one of the driver names MTcum, TYcum, PTcum, or CRcum (m is a model code, as given in 15.4)
device	is the hardware device address
bic	is the BIC address (used only, and then optionally, for magnetic-tape units)

Example: Specify Teletype unit 0 having model code A and hardware device address 01 as the DIR unit.

DIR=TY00A, 01

15.4.2 LIB (Library-Input Unit) Directives

This directive specifies the unit from which the SGL will be input (LIB unit). The directive has the general form

LIB = driver,device,bic

where

driver	is one of the driver names MTcum, CRcum, or Dcum
device	is the hardware device address

bic is the BIC address (used only, and then optionally, for magnetic-tape units) mandatory for RMDs

Example: Specify magnetic-tape unit 0 having model code A and hardware device address 010 (no BIC) as the LIB unit.

LIB=MT00A, 010

15.4.3 ALT (Library-Modification Input Unit) Directive

This directive specifies the unit from which object modules that modify the SGL will be input (ALT unit). The directive has the general form

ALT = driver,device,bic

where

driver	is one of the driver names MTcum, PTcum or CRcum
device	is the hardware device address
bic	is the BIC address (used only, and then optionally, for magnetic-tape units)

Example: Specify card reader unit 0 having model code A and hardware device address 030 as the ALT unit.

ALT=CR00A, 030

15.4.4 SYS (System-Generation Output Unit) Directive

This directive specifies the RMD(s) onto which the VORTEX system will be generated, with the VORTEX nucleus on the first such device specified. Up to 16 RMDs can be specified. The directive has the general form

**SYS = driver1,device1,bic1;driver2,device2,
bic2;...;drivern,devicen,bicn**

where

driver	is an RMD driver name such as Dcum, where c = controller, u = unit, and m = model code
device	is the hardware device address of the corresponding driver
bic	is the mandatory address of the applicable BIC or BTC

All RMDs specified in the EQP directives (15.5.2) must be specified in the SYS directive. Subsequent SYS directives will overlay the previous directives. If all RMDs cannot be specified in a single line, then the directive must be

SYSTEM GENERATION

terminated with a colon. This will cause the next input line to be treated as a continuation of the previous SYS directive. The additional input lines begin with the driver parameter. The directive "SYS =" must not be used on additional SYS directive input lines.

Examples: Specify RMD 0 having model code B, hardware device address 016, and BIC address 020 as the SYS unit.

SYS=D00B,016,020

Specify two SYS units: RMD 0 with model code A2, hardware device address 014, and BIC address 020; and RMD 0 with model code B, hardware device address 015, and BIC address 022.

A system with 70-7500 (620-34) or 70-7510 (620-35) disc requires a special formatting program, described in section 18.4. This program formats disc packs and performs bad-track analysis.

SYS=D00A2,014,020;D10B,015,022

15.4.5 LIS Directive

This LIS (User-Communication and List Output Unit) directive specifies the unit that will be used for user communication and list output (LIS unit). The directive has the general form

LIS = driver,device

where

driver is one of the driver names TYcum or LPcum

device is the hardware device address

The following information appears on the LIS unit:

- a. Error messages
- b. Load map of each load module
- c. Directives input through the DIR unit (section 15.4.1)
- d. Partition table for each system RMD

To suppress listing during system generation set "map" to zero in EDR directive.

Example: Specify line printer 0 having model code A and hardware device address 035 as the LIS unit.

LIS=LP00A,035

15.5 SGEN Directive Processing

Upon successful loading of the SGEN directive processor, the OC and LIS (section 15.4.5) units output the message

INPUT DIRECTIVES

to indicate that SGEN is ready to accept SGEN directives from the DIR unit (section 15.4.1).

The SGEN directives described in this section can be input in any order, except for the EDR directive (section 15.5.14), which is input last to terminate SGEN directive input.

In cases of conflicting data, SGEN treats the last information input as the correct data.

Errors cause the output of the applicable error messages (Appendix A.15).

The general form of an SGEN directive is

aaaa,p(1)xp(2)x...xp(n)

where

aaaa is a three or four character SGEN directive name

each **p(n)** is a parameter as indicated in the specifications for the individual directives

each **x** is a punctuation mark as indicated in the specifications for the individual directives

In contrast to most VORTEX system directives, the punctuation in SGEN directives is exactly as defined in the specifications for the individual directives, although blanks are allowed between parameters, i.e., before or after punctuation marks. SGEN directives begin in column 1 and can contain up to 80 characters.

SGEN directives are listed on the OC and LIS units.

15.5.1 MRY (Memory) Directive

This directive specifies the memory-related parameters of SGEN. It has the general form

MRY,memory,common,size [V75] ,npg,lpg

where

memory is the extent of the memory area available to VORTEX (minimum 12K = 027777)

common is the extent (0 or positive value) of the foreground blank-common area

size is the total physical memory available to VORTEX in units of 1024 (K) words. The minimum is 32 and the maximum is 1024

V75 specifies that the system is to be generated for a V75 system

npg specifies the total number of 512-word physical pages required by all Virtual Nucleus Overlay tasks (see VOL directive). **npg** will be converted to a sector count and added to the nucleus image size on RMD. **npg** may affect the track number of the initial RMD partition.

lpg specifies the last or highest numbered physical page to be assigned to Virtual Nucleus Overlay tasks. VNO tasks normally occupy consecutively numbered physical pages ending with the page specified by **lpg**. However, the sequential assignment of physical pages to VNO tasks may be altered by the presence of unavailable pages (see the NMRY directive). If the page specified by **lpg** is within unavailable pages, then assignment of pages to VNO tasks will begin immediately below the unavailable pages. If the page specified by **lpg** is greater than the highest unavailable page, but insufficient to provide the number of consecutive pages specified by **npg**, then pages above and immediately below the unavailable pages will be assigned to VNO tasks. **lpg** must be $64 >$ and $\leq (2^{\text{size}} - 1)$.

Examples: Specify a 64K memory for VORTEX with a foreground blank common area of 0200 words. Save locations 075777 to 077777 of the first 32K memory for AID II. Fifteen physical pages are reserved for VNO tasks, with page 127 being the highest physical page to be occupied by the VNO tasks.

MRY, 075777, 0200, 64, , 15, 127

Specify an 18,000-word memory for a VORTEX V75 system with no foreground blank-common area.

MRY, 18000, 0, V75

15.5.2 EQP (Equipment) Directive

This directive defines the peripheral architecture of the system. It has the general form

EOP, name, address, number, bic, retry, alg, mul

where

name is the mnemonic for a peripheral controller

address is the controller device address (01 through 077 inclusive)

number is the number (1 through 4, inclusive) of peripheral units attached to the controller

bic is the BIC or BTC address (0 if no BIC applies)

retry is the number (0 to 99, inclusive) of retries to be attempted by the I/O driver when an error is encountered

alg is the I/O algorithm value ($0 \leq alg \leq 1$) as a decimal fraction (see section 14.4.3, word 7 for the calculation of this value). NOTE: this is an optional parameter and is not needed unless a change is desired in the algorithm value. If this parameter is to be used on non-process I/O controller tables, the subject controller table must contain CTIOA as an entry name

mul is the multiplexor address (this parameter applies only to process I/O drivers)

Acceptable mnemonics for **name** are:

- MTnm Magnetic-tape unit
- LPnm Line printer
- CRnm Card reader
- PTnm Paper-tape reader/punch
- TYnm Teletype
- CTnm CRT device
- CPnm Card Punch
- Dnm RMD
- Ci Process input
- CO Process output
- WCS Writable control store
- SPnm Spool Unit
- MXnm Communication Multiplexor
- TCnm Psuedo TCM

Where n is the controller number (0, 1, 2, or 3), and m is the model code (table 15-2).

Controller tables are arranged according to the priority levels of their task-identification blocks (TIDBs). On any given level, the tables are arranged in the input sequence of the corresponding EQP directives. Device-specification table (DST) entries are unsorted.

SYSTEM GENERATION

The following order is suggested for peripheral controllers:

- a. RMDs
- b. Operator-communication (OC) device (section 17)
- c. Magnetic-tape units
- d. Other units

For the 70-7603/7013 disc, a special DEF directive must be included for each EQP directive used for this model disc.

DEF,V\$DSKx,y

where

x is the controller number (0-3)

y is a bit pattern in bits 0-7. Bit(n) corresponds to platter(n). The bit is set if the corresponding platter is part of a dual platter driver.

Example: A system contains two 70-76x3 controllers with the following drives attached:

Controller 0 has 1 dual unit and 3 single units
Controller 1 has 2 dual units, and 1 single unit, and 1 dual unit

the corresponding directives would be:

```
EQP,D0F,016,5,020,5
DEF,V$DSK0,3
EQP,D1F,017,7,022,5
DEF,V$DSK1,0157
```

Table 15-2. Model Codes for VORTEX Peripherals

Code	Model Number	Description
TYnA	70-6104 (620-08)	ASR Teletype Model 33 ASR Teletype Model 35
CTnA	70-6401	CRT keyboard/display
CRnA	70-6200 (620-22, 620-25)	Card reader: 300 or 600 cards/minute
CPnA	70-6201 (620-27)	Card punch: 35 cards/minute
MTnA	70-7100 (620-30) (620-31A) (620-31B) (620-31C) 70-7102 (620-32) 70-7103 (620-32A)	Magnetic-tape: 9-track, 800 bpi, 25 ips Magnetic-tape: 7-track, 200-556 bpi Magnetic-tape: 7-track, 200-800 bpi Magnetic-tape: 7-track, 556-800 bpi Magnetic tape: 9-track, 800 bpi, 37 ips Slave unit with 620-32
MXnA	70-520X (520X) 70-521X	Data communications multiplexor
DnA	620-47,-48,-49 70-770X (620-43C,-43D)	Rotating memory Rotating memory
DnB	70-7600 (620-36) 70-7610 (620-37)	Rotating memory Rotating memory

Table 15-2. Model Codes for VORTEX Peripherals
(continued)

Code	Model Number	Description
DnC	70-7500 (620-35)	Rotating memory
DnD	70-7510 (620-34)	Rotating memory
DnE	70-7520 70-7530	Rotating memory
DnF	70-7603 70-7613	Rotating Memory
PTnA	70-6320 (620-55A) (620-51A)	Paper-tape reader/punch
LPnA	70-6701 (620-77)	Line Printer
LPnB	70-6701	Line printer with print buffer 24 characters
LPnD	70-6602	Statos-31 Printer/plotter
LPnE	70-6603 (620-76)	Statos-31, with 041 option
LPnG	70-6603 (42,51,71)	Statos-31 with 042 option
LPnH	70-7702	Statos-31 with 041, 051, 052 options
LPnJ	70-66xx	Statos-33
LPnK	---	Statos 41 with 145 character generator
LPnL	---	Statos 41 with 144 character generator
LPnM	---	Statos 42 with 146 character generator
LPnN	---	Statos 42 with 144 character generator
CInA	See sec. 19	Process I/O
COnA	See sec. 19	Process I/O
WCS	70-4002	Writable control store

Note: Other peripheral devices can be added to the system by creating an EQP directive with a unique physical-unit name for the device. A controller table with the same name is then added to the VORTEX nucleus by an ADD directive (section 15.5.5).

SYSTEM GENERATION

Example: Define a system containing one model B RMD, one model A magnetic-tape unit, one model A card reader, one model A line printer, one model A Teletype, one model A high-speed paper-tape reader/punch, one model A card punch, and a writable control store.

```
EQP,DOB,016,1,020,3
EQP,MT0A,010,1,022,5
EQP,CR0A,030,1,024,0
EQP,LP0A,035,1,024,0
EQP,TY0A,01,1,0,0
EQP,PT0A,037,1,0,0
EQP,CP0A,031,1,022,0
EQP,WCS,074,1,0,0
```

The paper width of each Status on the system must be defined through use of the SGEN DEF directive (see section 15.5.14). This directive has the form

DEF,V\$SWnm,c

where

- n** is the controller number (0, 1 or 2)
- m** is the Status model code (D,E,G,H,J,K,L, M. or N)
- c** is the width code, defined as

0 = 8-1/2-inch	4 = single buffer 8-1/2 inch
1 = 11-inch	5 = single buffer 11 inch
2 = 14-7/8-inch	6 = single buffer 14-7/8 inch
3 = 22-inch	7 = single buffer 22 inch

Example: Specify a SGEN directive for model G Status on controller 1 with 14-7/8-inch width paper

```
DEF,V$SW1G,2
```

15.5.3 PRT (Partition) Directive

This directive specifies the size of each partition on each RMD. It has the general form

```
PRT,Dcup(1),s(1),k(1);Dcup(2),s(2),k(2);...;
Dcup(n),s(n),k(n)
```

where

- Dcup(n)** is the name of the RMD partition with **c** being the number (0, 1, 2, or 3) of the controller, **u** the unit number (0, 1, 2, or 3), and **p** the partition letter (A through T, inclusive)
- s(n)** is the number (octal or decimal) of tracks in the partition. The maximum partition size on any RMD is 32,768 sectors
- k(n)** is the protection code (single alphanumeric character including \$) for the partition, or * if the partition is unprotected

At least six partitions are required for the system rotating memory. PRT directives are required for every partition on every RMD in the system. While the partition specifications can appear in any order, the set of partitions specified for each RMD must comprise a contiguous group, e.g., the sequence D00A, D00C, D00D, D00B is valid, but the sequence D00A, D00C, D00D, D00E constitutes an error.

NOTE: If the LIB unit is an RMD, the PRT directives for that RMD are ignored and the existing PST for the RMD is used. However, even though the PRT directives are ignored the RMD unit should have at least one PRT directive. RAZI may be used to partition the RMD unit after system generation. If the RMD SGL is to be saved, it must be replaced with a scratch pack prior to executing RAZI for that unit.

Logical units 101 through 106 inclusive have preassigned protection codes. Do not attempt to change these codes.

Preassigned Protection Codes

Unit Number	101	102	103	104	105	106
Code	S	B	C	D	E	F

Total number of tracks of all partitions and the capacity of VORTEX nucleus must not exceed rotating-memory track capacity. The nucleus size is equal to the memory size divided by the product of the number of sectors per track and 120. Tracks not included by a PRT directive are not accessible to the system.

Example: Specify the following partitions on two RMDs.

RMD No.	Partition	Tracks	Protection Code
0	A	2	C
0	B	20	F
0	C	25	E
0	D	40	D
0	E	8	S
0	F	18	B
0	G	18	None
0	H	66	None
1	A	40	None
1	B	60	R
1	C	50	None
1	D	52	X

```
PRT,D00A,2,C;D00B,20,F
PRT,D00C,25,E;D00D,40,D;D00E,8,S
PRT,D00F,18B;D00G,18,*;D00H,66,*
PRT,D01D,52,X;D01C,50,*
PRT,D01A,40,*;D01B,60,R
```

15.5.4 ASN (Assign) Directive

This directive assigns logical units to physical devices. It has the general form

```
ASN,lun(1)=dev(1),lun(2)=dev(2),...,lun(n)=dev(n)
```

where each

lun(n) is a logical unit number (1 through 100 or 107 through 255, inclusive) that can be followed optionally by a two-character logical unit name e.g., 107:Y7

dev(n) is a four-character physical-device name, e.g., TY00,D00G (table 17-1)

If a new assignment specifies the same logical unit as a previous assignment, the old one is replaced and is no longer valid. All logical units for which physical device assignments are not explicitly made are considered *dummy units*, except preassigned.

Restrictions: Any attempt to change one of the preset logical unit name:number or name:number:partition relationships given in table 15-3 will cause an error to be flagged. Table 15-4 indicates the permissible physical unit assignments for the first 12 logical units (with PO automatically set equal to SS for normal assembler operation).

Example: Specify physical device assignments for logical units 1-12, inclusive, 107 and 108, and 180 and 181, where the last two units have, in addition to their numbers, two-character names.

ASN, 1=TY00, 2=CR00, 3=TY01, 4=CR00
 ASN, 5=LP00, 6=MT00, 7=D00I, 8=D00G
 ASN, 9=D00H, 10=D00G, 11=TY00, 12=LP00
 ASN, 107=LP00, 108=CR00
 ASN, 180:S6=MT00, 181:S8=MT01

Table 15-3. Preset Logical-Unit Assignments

Preset logical-unit name/number relationships:

OC = 1	LO = 5	GO = 9	13 = RPG IV READ
SI = 2	BI = 6	PO = 10	14 = RPG IV PUNCH
SO = 3	BO = 7	DI = 11	15 = RPG IV PRINT
PI = 4	SS = 8	DO = 12	

Preset logical-unit/RMD-partition relationships:

Logical-Unit Name	Logical-Unit Number	Partition Name	Protection Key	Minimum VORTEX Sector Allocation
CL	103	D00A	C	025 (see note 5)
FL	106	D00B	F	0106
BL	105	D00C	E	01135
OM	104	D00D	D	0417
CU	101	D00E	S	0310 (See note 1)
SW	102	D00F	B	0310 (See note 2)

Optional logical-unit/RMD-partition relationships

GO	9	D00G	none	0310 (See note 3)
SS	8	D00H	none	varies
PO	10	D00H	none	0515 (See note 4)
BI	6	D00I	none	varies
BO	7	D00I	none	varies

1. CU file must be as large as background task's largest part in central memory at one time (24K assumed above).
2. SW file must be as large as the largest single task including overlays (24K assumed above).
3. GO file must be somewhat larger than the largest task run in load-and-go mode (24K assumed). If system is

foreground only or all tasks will be entered in libraries before execution, this partition may be eliminated.

4. PO file must be large enough for source images of the largest task to be assembled or compiled. Source images are stored 3 card images per sector (1000 cards assumed above). If this function is assigned to magnetic tape, this partition may be eliminated.
5. There are 12 entries per 2 sectors. Number of sectors equals numbers of entry + 6.

SYSTEM GENERATION

Table 15-4. Permissible Logical-Unit Assignments

Logical Units	Permissible Physical Units				
	Teletype or CRT	RMD or MT	Line Printer	Other Output (CP,PT)	Other Input (PT,CR)
1 (OC)	X				
2 (SI)	X	X			X
3 (SO)	X				
4 (PI)	X	X			X
5 (LO)	X	X	X	X	
6 (BI)		X			X
7 (BO)		X		X	
8 (SS)		X			
9 (GO)		X			
10 (PO)		X			
11 (DI)	X				X
12 (DO)	X		X		

15.5.5 ADD (SGL Addition) Directive

This directive specifies the SGL control records and object modules *after which* new control records and/or object modules are to be added during nucleus generation. It has the general form

ADD,p(1),p(2),....p(n)

where each p(n) is the name of a control record or an object module *after which* new items are to be added.

When the name of a specified item is read from the SGL, the program is processed and the message

**ADD AFTER p(n)
READY**

appears on the OC unit. User response on the OC unit is either

ALT*

if an item is to be added from the SGEN ALT input unit (section 15.4.3), or

LIB

if processing from the SGL is to continue. If the former response is used, SGEN reads an object module from the

ALT unit and adds it to the SGL, then prints on the OC unit the message

READY

to which the user again responds with either ALT or LIB on the OC unit.

Example: Specify that items are to be added during nucleus generation after control records or object modules named PROG1, PROG2, and PROG3.

ADD, PROG 1, PROG 2, PROG 3

15.5.6 REP (SGL Replacement) Directive

This directive specifies the SGL control records and object modules to be replaced with new control records and/or object modules during nucleus generation. It has the general form

REP,p(1),p(2),....p(n)

where each p(n) is the name of a control record or an object module to be replaced.

When the name of the specified item is read from the SGL, the item is skipped and the message

REPLACE p(n)
READY

appears on the OC unit. User response on the OC unit is either

ALT*

if an item is to be replaced by one on the SGEN ALT input unit (section 15.4.3), or

LIB

if processing from the SGL is to continue. If the former response is used, SGEN reads an object module from the ALT unit and replaces p(n) with it in the SGL, then prints on the OC unit the message

READY

to which the user again responds with either ALT or LIB on the OC unit.

Example: Specify that control records or object modules named PROGA and PROGB are to be replaced during nucleus generation.

REP, PROGA, PROGB

*ALT has a special form which allows searching the ALT device for a specified program. The form is

ALT, name

where

name is one to six alphanumeric characters representing the TITLE name of the model to be added

name can either specify an object module name or a TDF record name. When specified, ALT will search the alternate unit from its current position for the specified module. If an EOF is encountered prior to finding the module an SG08 diagnostic occurs. To cause the alternate unit to rewind prior to each search, set Sense Switch 1 prior to entering the ALT directive. If no module name is specified, ALT will load from its current position.

For example, to search for and load an object module named PGRM1, specify

ALT, PGRM1

To search for and load a TDF directive for TBLPOF, specify
ALT, TBLPOF

Systems using the line printer driver model code B require a DEF directive to define the size in characters of the first print buffer. The format is

DEF, V\$LPcB, x

where

c is the controller number
x is the printer buffer size (in characters)

15.5.7 DEL (SGL Deletion) Directive

This directive specifies the SGL control records and object modules that are to be deleted during nucleus generation. It has the general form

DEL, p(1), p(2), ..., p(n)

where each p(n) is the name of a control record or an object module to be deleted.

When the name of a specified item is read from the SGL, the item is skipped and processing continues with the following control record or object module.

Example: Delete, during nucleus generation, all control records and object modules named PROG1 and PROG2.

DEL, PROG1, PROG2

15.5.8 LAD (Library Addition) Directive

This directive specifies the SGL load-module package *after which* new load-module packages are to be added during library generation. It has the general form

LAD, p(1), p(2), ..., p(n)

where each p(n) is the name of a load-module package from an SLM control directive *after which* new items are to be added.

When the name of a specified load-module package is read from the SGL, the program is processed and the message

ADD AFTER p(n)
READY

appears on the OC unit. User response on the OC unit is either

ALT

if a load-module package is to be added from the SGEN ALT input unit (section 15.4.3), or

LIB

SYSTEM GENERATION

if processing from the SGL is to continue. If the former response is used, SGEN reads a module from the ALT unit and adds it to the library, then prints on the OC unit the message

READY

to which the user again responds with either ALT or LIB on the OC unit.

Example: Specify that items are to be added, during library generation, after load-module packages named PROG1, PROG2, and PROG3.

LAD, PROG1, PROG2, PROG3

15.5.9 LRE (Library Replacement) Directive

This directive specifies the SGL load-module package to be replaced with new load-module packages during library generation. It has the general form

LRE,p(1),p(2),...,p(n)

where each $p(n)$ is the name of a load-module package from an SLM control directive to be replaced.

When the name of the specified load-module package is read from the SGL, the program is skipped and the message

REPLACE p(n)
READY

appears on the OC unit. User response on the OC unit is either

ALT

if module is to be replaced by one on the SGEN ALT input unit (section 15.4.3), or

LIB

if processing from the SGL is to continue. If the former response is used, SGEN reads a module from the ALT unit and replaces $p(n)$ with it in the SGL, then prints on the OC unit the message

READY

to which the user again responds with either ALT or LIB on the OC unit.

Example: Specify that load-module packages named PROGA or PROGB are to be replaced during library generation.

LRE, PROGA, PROGB

15.5.10 LDE (Library Deletion) Directive

This directive specifies the SGL load-module packages that are to be deleted during library generation. It has the general form

LDE,p(1),p(2),...,p(n)

where each $p(n)$ is the name of a load-module package from an SLM control directive to be deleted.

When the name of a specified load-module package is read from the SGL, the load-module package is skipped and processing continues with the following load module.

Example: Delete, during library generation, all load-module packages named PROG1 and PROG2.

LDE, PROG1, PROG2

15.5.11 PIM (Priority Interrupt) Directive

This directive defines the interrupt-system architecture by specifying the number of priority interrupt modules (PIMs) in the system, the interrupt levels to be enabled at system-initialization time, and the interrupts to be manipulated by user-coded interrupt handlers. The PIM directive has the general form

**PIM,p(1),q(1),r(1),s(1);p(2),q(2),r(2),
s(2);...;p(n),q(n),r(n),s(n)**

where each

- | | |
|-------------|---|
| p(n) | is an interrupt line number comprising two octal digits with the first being the PIM number and the second the line number within the PIM. The two digits must be preceded by a zero, e.g., 002,011 |
| q(n) | is the name (1 to 6 characters) of the task handling the interrupt. The name format is TBxxx, where xxx is the hardware code name. For $s(n) = 2$, $q(n)$ is the interrupt processor entry name. |
| r(n) | is the content of the interrupt event word in octal notation (see appendix F for nonzero values for standard hardware) |
| s(n) | is 0 for an interrupt using the common interrupt-handler or 1 for a directly connected interrupt option 1, or 2 for directly connected interrupt option 2. (Described in section 14.4.5) |

If an interrupt line is to use the common interrupt handler, a TIDB is generated for the related interrupt-processing routine, which can be in the VORTEX nucleus or in the foreground library.

If an interrupt line is to have a direct connection, the interrupt-processing routine must be added to the VORTEX nucleus. Failure to do so results in an error message.

Example: Specify two interrupt lines, one handled by the common interrupt handler, the other directly connected, option 1.

```
PIM,002,TBMT0A,00001,0;003,TBLP0B,01,1
```

Note: The only interrupt used by the magnetic-tape I/O driver is the motion complete.

Note: The interrupt event word, r(n) for a Teletype or CRT (Teletype compatible) must be set to 01 for input interrupt on 02 for output interrupt.

15.5.12 CLK (Clock) Directive

This directive specifies the values of all parameters related to the operation of the real-time clock. It has the general form

```
CLK,clock,counter,interrupt
```

where

clock	is the number of microseconds in the basic clock interval
counter	is the number of microseconds in the free-running counter increment period. Stored in V\$FREE but not used in VORTEX II. Its nominal value is 100.
interrupt	is the number of milliseconds in the user interrupt interval. This value must be between 5 and 50.

The value of **interrupt**, when not a multiple of 5 milliseconds, is increased to the next multiple of 5 milliseconds; e.g., if **interrupt** is 31, the interrupt interval is 35 milliseconds.

Example: Specify a basic clock interval of 100 microseconds, a free-running counter rate of 100 microseconds, and a user interrupt interval of 20 milliseconds.

```
CLK,100,100,20
```

For models V77-200/400 running on 60 cycle power, the clock directives must be:

```
CLK,100,100,15
DEF,V$CKCT,3
```

For models V77-200/400 running on 50 cycle power:

```
CLK,100,100,20
DEF,V$CKCT,3
```

15.5.13 TSK (Foreground Task) Directive

This directive specifies the tasks in the foreground library that are to be made resident tasks. It has the general form

```
TSK,task(1),task(2),...,task(n)
```

where each **task(n)** is the name of an RMD foreground-library task that is to be made a resident task.

If this directive is input as part of a full system generation, the names are those of tasks that will be built on the foreground library during the library-building phase (section 15.7).

Resident TIDBs are not created for the tasks defined on the TSK directives to be resident tasks. A TIDB is created each time a resident task is specified on a SCHED call. A resident TIDB is created at system generation for each task specified on a TDF directive.

These tasks are treated as user mode tasks and are not executed in map 0. Hence, I/O instructions cannot be executed by these tasks. Resident map 0 tasks are added to the nucleus by adding the programs on the SGL between the CTL,21 and CTL,PART003 control records. Section 14.4.8 describes resident tasks.

Example: Specify that foreground-library tasks RTA, RTB, and RTC be made resident tasks.

```
TSK,RTA,RTB,RTC
```

15.5.14 DEF (Define External) Directive

This directive enters a name with a corresponding absolute value into the SGEN loader tables and the CL library. It has the general form

```
DEF,name(1),value(1);name(2),value(2);...;name(n)
value(n)
```

Modules processed by either SGEN or LMGGEN can reference any names defined by the DEF directive

Example: Use the DEF directive for the VTAM LCB address in CTMX0A. The entry in CTMX0A for the LCB address might be

```
EXT V$LCW0
DATA V$LCW0
```

Then, the following DEF directive would define the LCB to be at location 075000

```
DEF,V$LCW0,075000
```

SYSTEM GENERATION

15.5.15 EDR (End Redefinition) Directive

This directive, which must be the last SGEN directive, specifies all special system-parameters, or terminates SGEN directive input. If only a redefinition of resident tasks is required, the EDR directive is of the form

EDR,R

but if a full SGEN is necessary, the EDR directive has the general form

EDR,S,tidb,stack,part,list,kpun,map,analysis

where

tidb	is the number (01 through 0777, inclusive) of 25-word empty TIDBS allocated
stack	is the size (0 through 037777, inclusive) of the storage and reentry stack allocation, which is equal to the number of words per reentrant subroutine multiplied by the number of levels calling the subroutine summed overall subroutines
part	is the maximum number (6 through 20, inclusive) of partitions on an RMD in the system
list	is the number of lines per page for the list output, with typical values of 44 for the line printer and 61 for the Teletype
kpun	is 26 for 026 keypunch Hollerith code, or 29 for 029 code
map	is L if map information is to be listed, or 0 if it is to be suppressed
analysis	is 0 or blank if a complete bad track analysis is desired on all RMD's, or 1 if the bad track tables from the last SGEN are to be reused. If this parameter is omitted, a full analysis is performed. A value of 1 may be entered only when an analysis has been made on a previous SGEN effort. If SGL is on slave disc, bypass (SET 1) the bad track analysis.

Bad-track or RMD partitioning analysis is performed following input of the EDR directive. When that process is complete, the VORTEX nucleus or resident-task processor is loaded into main memory.

Examples: Specify redefinition of resident tasks only.

EDR,R

Specify full system generation with no stack area, a maximum of five partitions per RMD, 44 lines per page on

the list output, 026 keypunch mode, and a list map, and a new bad track analysis is wanted.

EDR,S,0,0,5,44,26,L

Specify full system generation with 0500 addresses in the stack area, a maximum of 20 partitions per RMD, 30 lines per page on the list output, 029 keypunch mode, and suppression of the list map. Assume bad track tables from the last SGEN are still good, and reuse them.

EDR,S,0,0500,20,30,29,0,1

15.5.16 VOL (Virtual Overlay Task Definition) Directive

This directive specifies the tasks which are to be made into Virtual Nucleus Overlays. VNO tasks are loaded into memory by the VORTEX system initializer. Just before a VNO task executes, the physical pages it occupies are output to the mapping hardware and the VNO task is then executed as other nucleus-resident tasks are. The directive form is

VOL,taskname(1),taskname(2),...,taskname(n)

where

taskname(i) is a 1 to 8 alphanumeric character name associated with a task. It must match the name in the first record of an object module on the SGL.

The following restrictions apply to VNO tasks:

- DCBs, FCBs, and I/O buffers used by VNO tasks may not be part of the task. They must reside in map zero logical memory during I/O transfer.
- Jumps to VNO tasks may not be performed without first having mapped the task into map zero logical memory.
- VNO tasks may not exceed 31 pages in size or the capacity of the nucleus.
- The task must be attached to a resident TIDB and the **taskname** attached to that TIDB (see the TDF directive).

Example: Specify the I/O drivers VZDC, V\$TYA, VZMTA, and VZLPA as VNO tasks.

VOL,VZDC,V\$TYA,VZMTA,VZLPA

15.5.17 NMRY (No Memory) Directive

This directive specifies the physical pages of memory which are made unavailable for use by the VORTEX II operating system. The general form is

NMPY,pgnu(1),nu(1);pgnu(2),nu(2);...;pgnu(n),nu(n)

where

pgnu(i) specifies the starting physical page number of a block of pages to be made unavailable to the operating system. **pgnu** must be ≥ 64 and \leq the highest numbered page in the system.

nu(i) specifies the number of consecutive pages starting at **pgnu(i)** which are to be made unavailable.

Example: Declare pages 64 through 127 and 192 through 255 as unavailable.

```
NMRY, 64, 64; 192, 64
```

15.5.18 Required Directives

VORTEX system including writable control store (WCS) must include an EQP,WCS...directive.

Systems without a WCS must delete certain WCS support software modules. In particular, the following directives should be included to delete the MIUTIL and WCSRLD tasks:

```
LDE, FMIUTI
LDE, FWCSRL
```

In addition, the following directives may optionally be used to delete the remaining microprogramming support modules. These modules may be used on systems without WCS, but their deletion will make extra space available in the background library. The following directives delete the microprogram assembler and the simulator:

```
LDE, BMIDAS
LDE, BMICSI
```

Systems including VTAM require a DEF directive to define the LcB address. The format is:

```
DEF, V$LcWn, aaaaaa
```

where *n* is the DCM number and *aaaaaa* is the LcB address for the DCM

Systems including a status printer/plotter require a DEF directive to define the bed width. The format is:

```
DEF, V$$Wcm,a
```

where *c* = controller number

m = model code

a = 0 for 8-1/2 inches 4 = with SLIB
 1 for 11 inches 5 = with SLIB
 2 for 14-5/8 inches 6 = with SLIB
 3 for 22 inches 7 = with SLIB

15.5.19 CPU Directive

This directive (optional) specifies the type of V70 series CPU (central) processing unit that the system contains. This directive is used in conjunction with the SGEN2 control directive of the same name to select the proper version of multiversion modules. The general form of the directive is.

```
CPU,n
```

where

n is defined as:

- 1 = (default) for all V71 through V76 and V77-600.
- 2 = for V77-200/400 CPUs.

Example: Specify the system as a V77-400

```
CPU, 2
```

15.6 BUILDING THE VORTEX NUCLEUS

If a full system generation has been requested by the S form of an EDR directive, the nucleus processor is loaded upon completion of directive processing. Once loaded, the nucleus processor reads the SGL routines and builds the VORTEX nucleus as specified by the routines and the SGEN control records.

There are three SGEN control records used in building the nucleus:

- SLM Start load module
- TDF Build task-identification block
- MEM Default extra memory pages
- END End of nucleus library

Normally these control records are used only to replace existing SGL control records.

VORTEX nucleus processing consists of the automatic reading of control records and object modules from the SGL, and, according to the specifications made by SGEN directives, either ignoring the item or incorporating it into the VORTEX nucleus. The only manual operations are the addition and replacement of object modules during system generation. In these cases, follow the procedures given in the ADD and REP directives respectively.

15.6.1 SLM (Start Load Module) Directive

This directive specifies the beginning of a load module. Its presence indicates the beginning of the system initializer or VORTEX nucleus. The directive has the general form

```
SLM,name
```

SYSTEM GENERATION

where **name** is the name of the load module that follows the directive.

Example: Indicate the beginning of the VORTEX nucleus.

SLM, VORTEX

15.6.2 TDF (Build Task-Identification Block) Directive

This directive specifies all parameters necessary to build a task-identification block in the VORTEX nucleus. It has the general form

TDF, name, exec, ctrl, stat, level [V75], taskname

where

- name** is the name (1 to 6 alphanumeric characters) given to the TIDB for linking purposes
- exec** is the name (1 to 6 alphanumeric characters) associated with the execution address of the task

- ctrl** is the name (1 to 6 alphanumeric characters) of the controller table required for Teletype and CRT processing tasks, or is 0 for any other task
- stat** is the 16-bit TIDB status word where the settings of the individual bits have the significance shown in table 15-5
- levl** is the priority level of the related tasks
- V75** if present, indicates a V75 system

Example: Define a foreground resident task PROG1 on priority level 10 to execute on boot.

TDF, TIDB, P1, PROG1, 0, 05400, 10, V75

The TDF directive causes a resident TIDB to be created for the specified task. The task itself may or may not be a resident task, as defined by the status word (**stat**). See the TSK directive for generation of resident tasks without resident TIDB.

If the **taskname** parameter is specified, then the VNO task and associated TIDB must be resident.

Table 15-5. TIDB Status-Word Bits

Bit	When Set Indicates	Explanation
15	Interrupt suspended	The task is suspended during the processing of a higher-priority task. The contents of volatile registers are stored in TIDB words 12-16 (interrupt stack).

14	Task suspended	The task is suspended because of I/O or because it is waiting to be activated by an interrupt, time delay, or another task. The task is activated whenever this bit is zero, or if TIDB word 3 has an interrupt pending and the task expects the interrupt.
13	Task aborted	The task is not activated. All stacked I/O is aborted, but currently active I/O is completed.
12	Task exited	The task is not activated. All stacked and currently active I/O is completed.

Table 15-5. TIDB Status-Word Bits (continued)

Bit	When Set Indicates	Explanation
11	TIDB resident	The TIDB (drivers, task-interrupt processors, resident tasks, and time-scheduled tasks) is resident and not released when the task is aborted or exited.
10	Task resident	The task is resident and not released when aborted or exited.
9	Foreground task	The task is in protected foreground.

8	Check-point flag	Set: may be check-pointed by a lower priority task. Reset: may not be check-pointed by a lower priority task.
7	Task scheduled by time increment	The task becomes nonsuspended when a specified time interval is reached.
6	Time delay active	The clock decrements the time counter that, upon reaching zero, clears bit 14.

5	Task checkpointed	The background task is check-pointed and suspended. I/O is not activated.
4	Error in task	The task contains an error that will cause an error message to be output.
3	Task interrupt expected	A task interrupt is expected.

2	Overlay task	The task contains overlays.
1	Task-schedule this task	The scheduling task is suspended until the scheduled task exits or aborts.
0	Task searched, allocated and loaded	The task is loaded in memory and is ready for execution.

15.6.3 END Directive

This directive indicates the end of the system initializer or the VORTEX nucleus. It has the form

END

Example: Indicate the end of the system initializer.

END

15.6.4 MEM Directive

This optional directive performs the same function as the same directive in LMGEN (see section 6.2.7). The directive has the general form

MEM,n

SYSTEM GENERATION

where

n is the number of extra pages desired.

This directive, if used, must appear after the last ESB directive and before the END directive.

15.6.5 Memory Parity Considerations (V70/V77-600)

Memory parity is not a supported feature under VORTEX on the V70 series and V77-600 processor. For those systems which require the use of memory parity, the user may write his own memory-parity service routine (see section 14) and add it to the system. The following are considerations when using memory parity.

- The memory parity interrupt trap must be an even modulo-8 address, e.g., 010, 0100, 0110, 0200, etc. The exact address depends upon the number of PIMs in the system. For example, a system with 3 PIMs can use any of the following addresses: 0160, 0170, 0200, 0230, 0240, 0250, 0260, 0270, or 010. If 4 PIMs are in the system, then any of the above addresses except for 0160 and 0170 may be used. In the case where all 8 PIMs are used, the only available address will be 010.
- For trap addresses between 0100 and 0277, the SGEN PIM directive, specifying the direct connect option, may be used to link up the trap address with the user's memory-parity routine. If a trap address of 010 is used, the PIM directive cannot be used. In this case, the easiest means of linking the trap address and the service routine would be to modify the "low-core" module (V\$LMEMBK) to specify an EXT to the user's interrupt service routine.
- No enable/disable memory parity instructions are required and hence no changes are required for the system initializer.

15.6.6 Memory Parity Considerations (V77-200/400)

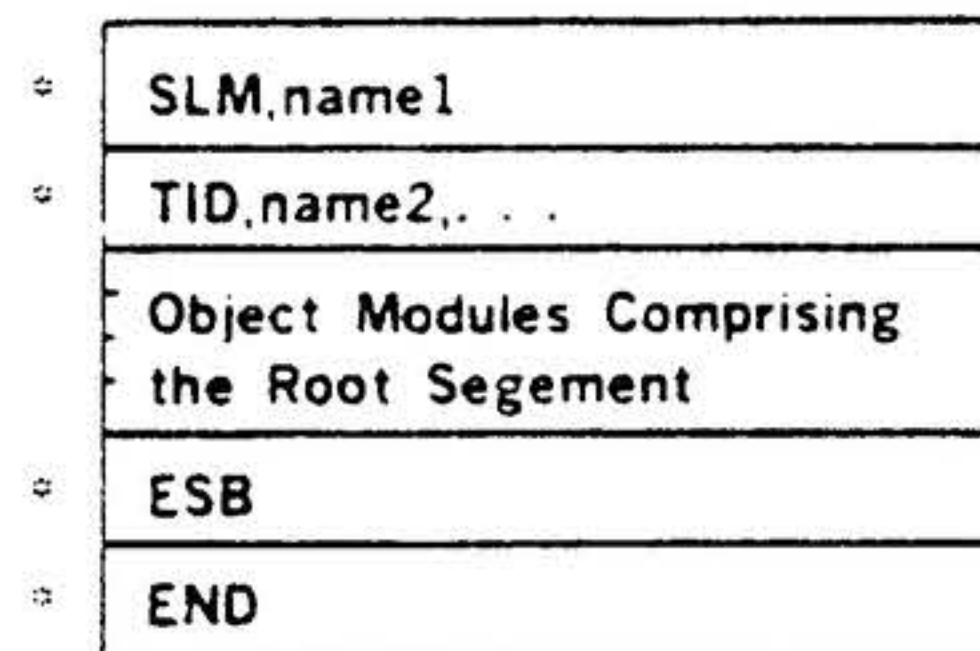
Memory parity detection is a supported feature under VORTEX on the V77 model 200/400. A parity error is reported by an EX35 error message (see appendix 11.2). The map tracking register contains the address being fetched when the parity error occurred. Memory parity is a feature of the SYSGEN selection parameter.

15.7 BUILDING THE SYSTEM LIBRARIES AND RESIDENT TASK CONFIGURATION

If a full system generation has been requested by the S form of an EDR directive (section 15.5.15), the library generator is loaded upon completion of nucleus processing. If only reconfiguration of resident tasks has been requested

(R form of the EDR directive), the resident task configurator is loaded immediately after directive processing.

A **load module** is a logically complete task or operation that can be executed by the VORTEX system in foreground or background. It resides in the foreground or background library, or in the user library. Load modules are constructed from sets of binary object modules interspersed with alphanumeric control records. The control records indicate the beginning and end of data for incorporation into each load module, and specify certain parameters to the load module. The group of object modules and control records used to construct a load module is called a **load-module package (LMP)**. Figure 15-5 shows an LMP for a load module without overlays, and figure 15-6 shows an LMP for a load module with overlays. Each LMP runs from a SLM control record to an END control record, and includes all modules and records between the SLM and END.



NOTE: * = Alphanumeric control record

Figure 15-5. Load Module Package for Module Without Overlays

There are five SGEN control records used in building the library:

- SLM Start load module
- TID Task-identification block specification
- OVL Overlay
- ESB End of segment
- END

Library processing consists of the automatic reading of control records and object modules from the SGL, and construction of the library from these inputs. The only manual operations are the addition and replacement of load modules. In these cases, follow the procedures given in sections 15.5.8 and 15.5.9, respectively.

Resident-task configuration takes place upon completion of library processing. All tasks specified by TSK directives (section 15.5.13) are copied from the foreground library into the VORTEX nucleus, thus becoming *resident tasks*. To change the resident-task configuration of a previously generated system, input the TSK directives followed by the R form of the EDR directive (section 15.5.15), thus bypassing nucleus and library processing and allowing the resident-task configurator to alter the existing system. **Note:** If a specified program is not found in the foreground library, configuration continues, but an appropriate message is output.

15.7.1 SLM (Start LMP) Directive

This directive indicates the start of an LMP. It has the general form

SLM,name

where **name** is the name of the LMP that begins with this directive.

Example: Indicate the start of the LMP named ABC.

SLM,ABC

15.7.2 TID (TIDB Specification) Directive

This directive contains the parameters necessary for the generation of the task-identification block required for each generated load module. The TID directive has the general form

TID,name,mode,ovly,lun

where

- name** is the name (one to six alphanumeric characters) of the task
- mode** is 1 if the task is a background task, or 2 if it is a foreground task
- ovly** is the number of overlay segments, or 0 if the task has no overlay segments, (note that the value 1 is invalid)
- lun** is the number of the logical unit onto which the task is to be cataloged

Once a TID directive is input and processed, object modules are input, processed, and output to the specified logical unit until the ESB directive (section 15.7.4) is found.

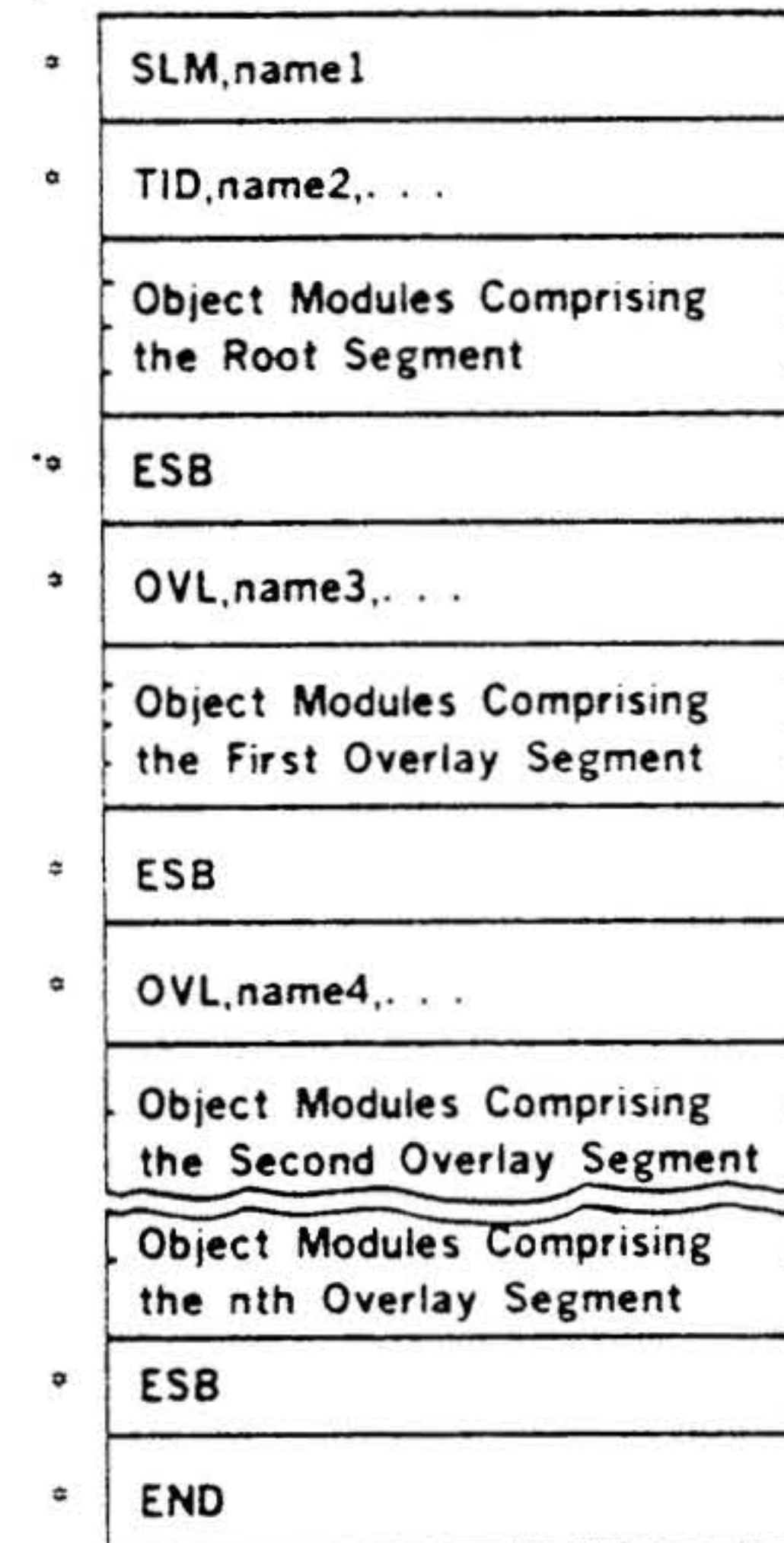
Examples: Specify a TIDB for a task PROG1 without overlays for cataloging on the BL unit (105).

TID,PROG1,1,0,105

Specify a TIDB for the task PROG2 with four overlay segments for cataloging on an FL unit (106).

TID,PROG2,2,4,106

Note: If a specified program is not found in the foreground library, configuration continues, but an appropriate message is output.



NOTE:

* = Alphanumeric control record

Figure 15-6. Load Module Package for Module With Overlays

15.7.3 OVL (Overlay) Directive

This directive indicates the beginning of an overlay segment. The OVL directive has the general form

OVL,segname

where **segname** is the name (one to six alphanumeric characters) of the overlay segment.

Example: Indicate the beginning of the overlay segment SINE.

OVL,SINE

SYSTEM GENERATION

15.7.4 ESB (End Segment) Directive

This directive indicates the end of a segment, i.e., that all object modules have been loaded and processed. The directive has the form

ESB

The ESB directive causes the searching of the CL library, which was generated during nucleus processing, to satisfy undefined externals.

The ESB directive concludes both root segments (following TID, section 15.7.2) and overlay segments (following OVL, section 15.7.3) of a load module.

Example: Indicate the end of a segment.

ESB

15.7.5 END (End Library) Directive

This directive indicates the end of load-module generation. It has the form

END

Example: Specify the end of load-module generation.

END

15.8 SYSTEM INITIALIZATION AND OUTPUT LISTINGS

Upon completion of load-module processing, SGEN outputs on the OC and LIS units the message

VORTEX SYSTEM READY

The system initializer and VORTEX nucleus are then loaded into memory, the initializer is executed to initialize the system, and the nucleus is executed to begin system operation. If writable control store is present in the system, the following messages will appear on the OC device at this time:

IO10,WCSRLD
FILE WCSIMG NOT FOUND
WCS RELOAD ABORTED

These messages are output by the WCS reload task. In WCS systems, this task is automatically scheduled upon loading the system in order to restore WCS contents. To do

this, it uses the contents for WCS which were saved on a disc file the last time WCS was loaded. At this point, however, WCS has not yet been loaded. Thus, the reload task cannot restore WCS and exits after outputting the above messages. At this time, the OM library should be loaded and build on the RMD using FMAIN.

The OM library is provided as job streams as the second through thirty-fifth files on the SGL. An EOF separates the SGL from the OM stream. A system generation leaves magnetic tape and card SGLs prior to this EOF, thus it must be skipped over before executing the OM job stream. For disc SGLs the OM library object modules are on the second partition of the disc pack (DcuB). Refer to the VORTEX/VORTEX II Installation Manual for details.

The VORTEX system is now operating with the peripherals in the status specified by TID control records.

If the EDR directive specified a listing, linking information is listed on the LIS unit during nucleus processing and library generation. Regardless of the EDR directive, RMD and resident-task information is listed during nucleus processing or resident-task configuration, respectively. Figures 15-7 through 15-10 show the listing formats of load maps for the VORTEX nucleus, the library processor, the RMD partitions, and the resident tasks.

CORE RESIDENT LIBRARY	
NAME	LOCATION
AAA	017285
BBB	021255
.	.
.	.
ZZZ	075777

NONSCHEDULED TASKS	
NAME	LOCATION
TBABC	072620
TBDEF	074640
.	.
.	.
TBXYZ	076400

Figure 15-7. VORTEX Nucleus Load Map

```

SLM,BGTSKI
TID,JCP,1,0,105
ESB
MOP      A      032556
QRS      R      000200
.        .
.        .
.        .
TUV      A      032501
SLM,FGTSKI
TID,V$OPCM,2,8,106
ESB
GHI      R      000010
JKL      R      000012
.        .
.        .
.        .
MNO      R      000077
    
```

Figure 15-8. Library Processor Load Map

RMD PARTITIONING			
NAME	FIRST TRACK	LAST TRACK	BAD TRACKS
D00A	0007	0008	0000
D00B	0009	0028	0000
D00C	0029	0053	0000
D00D	0054	0093	0000
D00E	0094	0101	0000
D00F	0102	0119	0000
D00G	0120	0137	0000
D00H	0138	0203	0000
D01A	0001	0039	0000
D01B	0040	0099	0000
D01C	0100	0149	0000
D01D	0150	0203	0000

Figure 15-9. RMD Partition Listing

MEMORY RESIDENT TASKS	
NAME	LOCATIONS
PROG1	014630
PROG2	014630
PROG3	NOT FOUND
PROG4	014500

Figure 15-10. Resident-Task Load Map

```

PAGES (OCTAL)      ALLOCATED TO
                    0 PAGE 0 SYSTEM DATA
                    1 - 50 UNALLOCATED
                    51 - 72 NUCLEUS PROGRAM MODULE
                    72 - 75 NUCLEUS TABLE MODULE
                        75 GLOBAL FCB PAGE
                        75 FOREGROUND BLANK COMMON
                    100 - 163 UNALLOCATED
                    164 - 177 VIRTUAL OVERLAY TASKS
                    200 - 277 UNAVAILABLE
VORTEX SYSTEM READY
    
```

Figure 15-11. Physical Memory Allocation

15.9 SYSTEM GENERATION EXAMPLES

EXAMPLE 1

Problem: Generate a VORTEX system using the following hardware:

- a. Computer with 32K main memory
- b. A model 70-7610 (620-37) disc unit with device address 016 on BIC 20
- c. Teletype keyboard/printer
- d. Card reader
- e. Two buffer interlace controllers (BICs) with device addresses 020 and 022
- f. One priority interrupt module (PIM) with device address 040
- g. No writable control store

and having the characteristics listed below:

- a. Foreground common size = 0200
- b. Storage/reentry stack area size = 0200
- c. Number of disc partitions = 9
- d. All eight interrupt lines connected through a common interrupt handler 0 = BIC1, 1 = BIC2, 2 = CR, 3 = Disc seek, 4 = TY read, 5 = TY write, 6-7 unassigned
- e. One user-coded task added to the resident module (PROG1)
- f. JCP replaced with a new version
- g. One user-coded load module added to the background library (after LMGEM) (PROG2)
- h. The system file listed after system generation

SYSTEM GENERATION

Procedure:

Step	User Action	SGEN Response
1	Load and execute the card reader loader (table 15-1)	Loads the I/O interrogation routine punched cards from the card reader, and outputs on the OC unit I/O INTERROGATION
2	On the OC unit, input DIR = TY00A,01 LIB = CR00A,030 ALT = CR00A,030 LIS = TY00A,01 SYS = D00B,016,020	Loads the SGEN drivers and directive processor, and outputs INPUT DIRECTIVES
3	On the Teletype (DIR unit), type CLK,100,100,20 MRY,757777,0200,32 EQP,D0B,016,1,020,3 EQP,TY0A,01,1,0,0 EQP,CR0A,030,1,022,0 PRT,D00A,2,C;D00B,20,F PRT,D00C,25,E;D00D,40,D PRT,D00E,8,S;D00F,18,B PRT,D00G,18,*;D00H,52,* PRT,D00I,14,* ASN,1 = TY00,2 = TY00,3 = TY00 ASN,4 = CR00,5 = TY00,6 = CR00 ASN,7 = D00I,8 = D00H,9 = D00G ASN,10 = D00H,11 = TY00,12 = TY00 ASN,180 = D00H,181 = D00I PIM,03,TBD0B,01,0;02,TBCR0A,01,0 PIM,03,TBD0B,01,0;04,TBTY0A,01,0 PIM,05,TBTY0A,02,0 TSK,PROG1 LRE,BJCP LAD,BLMGEN LDE,FMIUTI LDE,FMICSI LDE,FMIDAS LDE,FNCSRL EDR,S,20,0200,9,61,26,L	Processes the directives, partitions the disc, loads the nucleus processor and builds the nucleus, loads the library processor and builds the library until load module JCP is encountered, and outputs REPLACE BJCP READY
4	Load revised version of BJCP load module in the card reader, and on DIR type: ALT	Reads and processes the new load module, and outputs: READY
5	Load the remainder of the load module library in the card reader, and on DIR type LIB	Processes the load module library until the completion of LMGEN, and outputs ADD AFTER BLMGEN READY
6	Load the PROG1 load module in the card reader, and on DIR type	Reads and processes PROG1, and outputs

Procedure: (continued)

Step	User Action	SGEN Response
		READY
	ALT	
7	Load the PROG2 load module in the card reader, and on DIR type	Reads and processes PROG2, and outputs
	ALT	READY
8	Load the remainder of the load module library in the card reader, and on DIR type	Processes the remainder of the load module library, copies PROG1 from the FL unit to the VORTEX nucleus, lists the resident task information, and outputs on OC and LIS
	LIB	VORTEX SYSTEM READY
9	None	Loads and initializes the VORTEX nucleus

EXAMPLE 2

Problem: Replace the current resident tasks in the foreground library with the tasks listed below in an operational VORTEX system. Assume the SGL is on magnetic tape unit 0. The system has a line printer and a 620-48 RMD on DA014. ALT is on the slave MT.

PROG1
ABC
TEST
EFG

Procedure:

Step	User Action	SGEN Response
1	Load and execute the magnetic tape loader (table 15-1)	Loads the I/O interrogation routine from magnetic tape and outputs from the OC unit
		IO INTERROGATION
2	On the OC unit, input DIR = TY00A,01 LIB = MT00A,010 ALT = MT01A,010 LIS = LP00A,035 SYS = D00A2,014,020	Loads the SGEN drivers and directive processor, and outputs
		INPUT DIRECTIVES
3	On the Teletype (DIR unit), type TSK,PROG1,ABC TSK,TEST,EFG EDR,R	Processes the directives, loads the resident-task processor, enters the PROG1, ABC, TEST, and EFG load modules from FL, lists resident information, and outputs on OC and LIS
		VORTEX SYSTEM READY
4	None	Loads and initializes the VORTEX nucleus

SECTION 16 SYSTEM MAINTENANCE

The VORTEX **system-maintenance component (SMAIN)** is a background task that maintains the **system-generation library (SGL)**. The SGL (figure 15-2) comprises all object modules and their related control records required to generate a generalized VORTEX operating system.

16.1 ORGANIZATION

SMAIN is scheduled for execution by inputting the job-control-processor (JCP) directive /SMAIN (section 4.2.21).

Once SMAIN is so scheduled, loaded, and executed, SMAIN directives can be input from the SI logical unit to maintain the SGL. No processing of the SGL takes place before all SMAIN directives are input and processed. Then user-specified object modules and/or control records are added, deleted, or replaced to generate a new SGL.

SMAIN has a symbol-table area for 200 symbols at five words per symbol. To increase this, input a /MEM directive (section 4.2.5), where each 512-word block will increase the capacity of the table by 100 symbols.

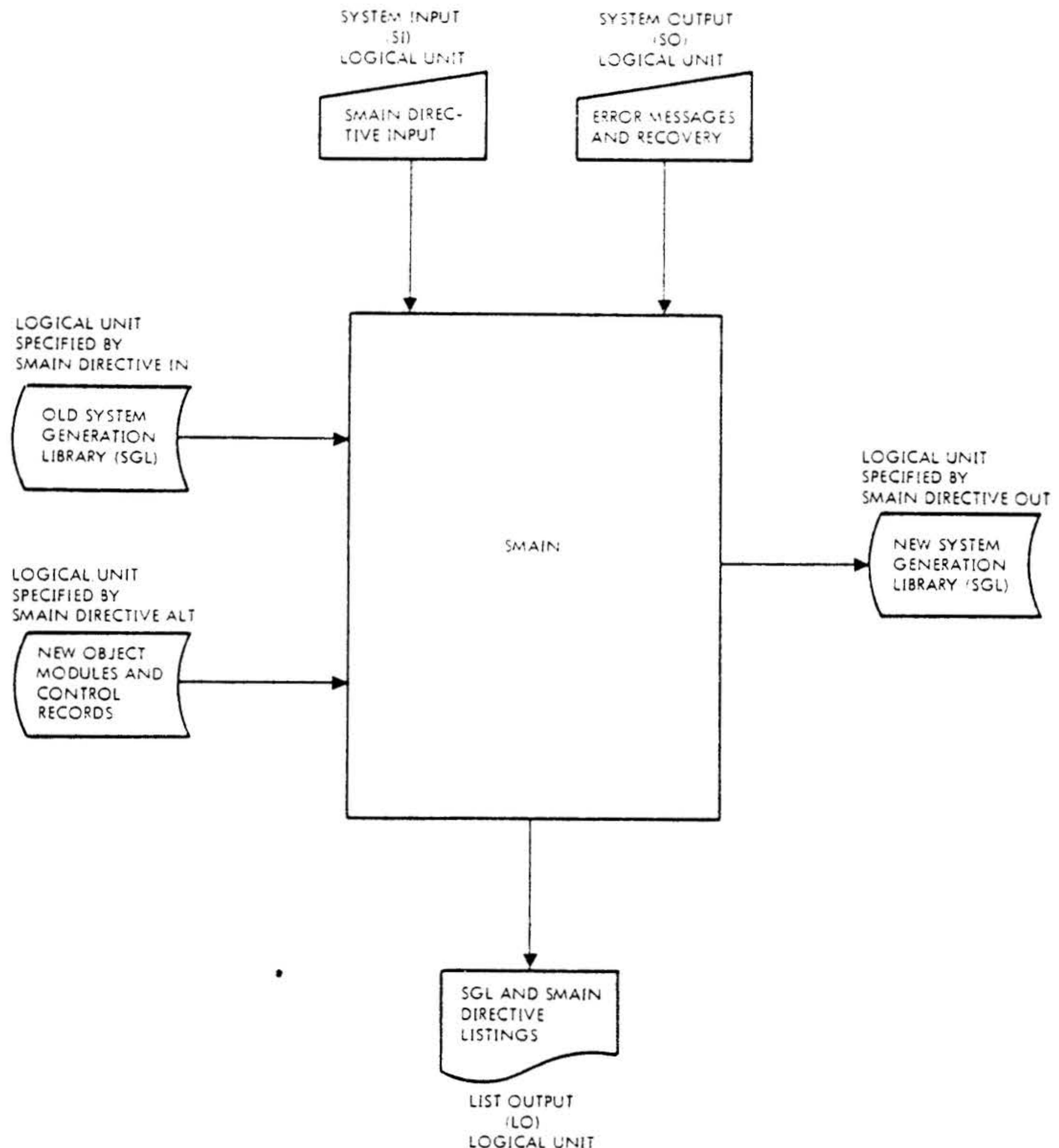


Figure 16-1. SMAIN Block Diagram

SYSTEM MAINTENANCE

INPUTS to the SMAIN comprise:

- a. *System-maintenance directives* (section 16.2) input through the SI logical unit.
- b. *The old SGL* input through the logical unit specified by the IN directive (section 16.2.1).
- c. *New or replacement object modules and/or control records* input through the logical unit specified by the ALT directive (section 16.2.3).
- d. *Error-recovery inputs* entered via the SO logical unit.

System-maintenance directives specify both the changes to be made in the SGL, and the logical units to be used in making these changes. The directives are input through the SI logical unit and listed, when specified, on the LO logical unit. If the SI logical unit is a Teletype or a CRT device, the message **SM**** is output to indicate that the SI unit is waiting for SMAIN input.

The old SGL contains three types of records: 1) control records and comments (ASCII), 2) the system-generation relocatable loader and BOOTLODR (the only SGL absolute core-image records), and 3) relocatable object modules such as are output by the DAS MR assembler and the FORTRAN compiler.

New or replacement object modules and/or control records have the same specifications as their equivalents in the old SGL.

Error-recovery inputs are entered by the operator on the SO logical unit to recover from errors in SMAIN operations. Error messages applicable to this component are given Appendix A.16. Recovery from the type of error represented by invalid directives or parameters is by either of the following:

- a. Input the character C on the SO unit, thus directing SMAIN to go to the SI unit for the next directive.
- b. Input the corrected directive on the SO unit for processing. The next SMAIN directive is then input from the SI unit.

Recovery from errors encountered while processing object modules and/or control records is by either of the following:

- a. Input the character R on the SO unit, thus directing a rereading and reprocessing of the last record.
- b. Input the character P on the SO unit, thus directing a rereading and reprocessing from the beginning of the current object module or control record.

In the last two cases, repositioning is automatic if the error involves a magnetic-tape unit or an RMD. Otherwise, such repositioning is manual.

If recovery is not desired, input a JCP directive (section 4.2) on the SO unit to abort the SMAIN task and schedule the JCP for execution.

OUTPUTS from the SMAIN comprise:

- a. *The new SGL*
- b. *Error messages*
- c. *The listing of the old SGL, if requested*
- d. *Directive images*

The new SGL contains object modules and control records. It is similar in structure to the old SGL.

Error messages applicable to SMAIN are output on the SO and on LO logical units. The individual messages, errors, and possible recovery actions are given in Appendix A.16.

The listing of the old SGL is output, if requested, on the LO unit. The output consists of a list of all control records and the contents of all object modules. At the top of each page, the standard VORTEX heading is output.

The image of an object module is represented by the identification name of the module, the date the module was generated, the size (in words) of the module (0 for a FORTRAN object module), and the external names referenced by the module, in the following format:

```
id-name    date    size    entry-names    external-names
```

Directive images are posted onto the LO unit, thus providing a hardcopy of the SMAIN directives for permanent reference.

16.1.1 Control Records

In SMAIN there are two types of control record:

- a. *SGL delimiters*
- b. *Object-module delimiters*

SGL delimiters divide the SGL into five parts. Each part is separated from the following part by a control record of the form

```
CTL, PART000n
```

where n is the number of the following part, and the SGL itself is terminated by a control record of the form

```
CTL, ENDOFSGL
```

Within SMAIN directives, these control records are referenced in the following format

```
PART000n
ENDOF SGL
```

Object-module delimiters precede and/or follow each group of object modules within the SGL. Each delimiter is of one of the forms

```
SLM, name
TID, name
OVL, name
TDF, name
ESB
END
```

The control records containing a name can be referenced by use of the name alone in SMAIN directives. These control records and their uses are described in the VORTEX II System Generation User Guide/Programmer Reference (UP-9083). ←

A set of object modules preceded by an SLM control record and followed by an END control record is known as a **load-module package (LMP)**. To add, delete, or replace an entire LMP, merely reference the name associated with the SLM control record. Thus, if the directive specifies deletion and includes the name associated with the SLM record, the entire LMP is deleted. Additions and replacements operate analogously.

16.1.2 Object Modules

Relocatable object-module outputs from the DAS MR assembler and the FORTRAN compiler are described in appendix G.

16.1.3 System-Generation Library

The SGL is a collection of system programs in binary-object form, and of control records in alphanumeric form, from which a VORTEX system is generated. The structure of the SGL is described in VORTEX II System Generation User Guide/Programmer Reference (UP-9083). ←

16.2 SYSTEM-MAINTENANCE DIRECTIVES

This section describes the SMAIN directives:

- IN Specify input logical unit
- OUT Specify output logical unit
- ALT Specify input logical unit for new SGL items
- ADD Add items to the SGL
- REP Replace SGL items
- DEL Delete items from the SGL
- LIST List the old SGL
- END End input of SMAIN directives

SMAIN directives begin in column 1 and comprise sequences of character strings having no embedded blanks. The character strings are separated by commas (,) or by equal signs (=). The directives are free-form and blanks are permitted between the individual character strings of the directive, i.e., before or after commas (or equal signs). Although not required, a period (.) is a line terminator. Comments can be inserted after the period.

The general form of an SMAIN directive is

```
name,p(1),p(2),...,p(n)
```

where

name is one of the directive names given above (any other character string produces an error)

each $p(n)$ is a parameter defined below under the descriptions of the individual directives

Numerical data can be octal or decimal. Each octal number has a leading zero.

For greater clarity in the descriptions of the directives, optional periods, optional blank separators between character strings, and the optional replacement of commas (,) by equal signs (=) are omitted.

Error messages applicable to SMAIN directives are given in Appendix A.16.

16.2.1 IN (Input Logical Unit) Directive

This directive specifies the logical unit from which the old SGL is to be input. It has the general form

```
IN,lun,key,filename
```

where

lun is the name or number of the logical unit to be used for the input of the old SGL

key is the protection code, if any, required to address **lun**

filename is the name of the input file only when **lun** is an RMD partition with a directory

There is no default value for **lun**. If it is not specified, any attempt at SGL processing will cause an error message output.

Once specified, the value of **lun** remains constant until changed by a subsequent IN directive. Each change of **lun** requires a new IN directive.

SYSTEM MAINTENANCE

If **lun** specifies an RMD partition and no filename is specified, the RMD is rewound to the first sector following the start of the partition before any processing takes place.

Examples: The old SGL resides on logical unit 4, the PI unit. Specify this unit to be the SGL input unit.

IN, 4

The old SGL resides on logical unit 107, which requires the protection code G. Specify this unit to be the SGL input unit. (This is a non-directoried partition.)

IN, 107, G

16.2.2 OUT (Output Logical Unit) Directive

This directive specifies the logical unit on which the new SGL is to be output. It has the general form

OUT,lun,key,filename

where

lun	is the name or number of the logical unit to be used for the output of the new SGL
key	is the protection code, if any, required to address lun
filename	is the name of the output file when lun is an RMD partition

The default value of **lun** is zero. When **lun** is zero by specification or by default, there is no output logical unit.

Once specified, the value of **lun** remains constant until changed by a subsequent OUT directive. Each change of **lun** requires a new OUT directive.

If **lun** specifies an RMD partition and no filename is specified, the RMD is rewound to the first sector following the PST before any processing takes place. The PST comprises one entry defining the entire RMD.

Examples: Specify the PO logical unit, unit 10, to be the output unit for the new SGL.

OUT, 10

Specify that there is to be no output logical unit.

OUT, 0

16.2.3 ALT (Alternate Logical Unit) Directive

This directive specifies the logical unit from which new object module(s) and/or control record(s) are to be input to the new SGL. It has the general form

ALT,lun,key,filename

where

lun	is the name or number of the logical unit to be used for the input of new items to the SGL
key	is the protection code, if any, required to address lun
filename	is the name of the input file when lun is an RMD partition

There is no default value for **lun**. If it is not specified, any attempt to input new object modules or control records to the SGL will cause an error message output.

Once specified, the value of **lun** remains constant until changed by a subsequent ALT directive. Each change of **lun** requires a new ALT directive.

Examples: Specify that new object modules and control records are to be input to the SGL from the BI logical unit only.

ALT, 6

Make the same specification where BI is an RMD partition without a protection code. Use file FILEX.

ALT, BI, , FILEX

Note: SMMAIN expects binary input on RMD to be packed (two 60 word binary records per sector).

16.2.4 ADD Directive

This directive permits the addition of object modules and/or control records during the generation of a new SGL, the additions being made immediately after each of the items specified by the parameters of the ADD directive. The directive has the general form

ADD,p(1),p(2),...,p(n)

where each **p(n)** is the name of an object module or control record **after which** additions are to be made.

Each **p(n)** has the option of selecting the occurrence of the module (they must be contiguous). The form is:

p(n)/#

where # is the selected occurrence of the module (pin) as handled. For example, to add a new module after the second occurrence of V\$IOC, enter:

```
ADD, V$IOC/2
```

SMAIN copies object modules and control records from the old SGL into the new SGL up to and including an item specified by one of the parameters, p(n), of the ADD directive. After this item is copied, the message

```
ADD AFTER p(n)
SM**
```

is output to indicate that SMAIN is waiting for a control character (Y or N) to be input on the SO logical unit.

If the control character input is Y, SMAIN adds the next object module or control record contained on the logical unit specified by the ALT directive (section 16.2.3), then repeats the message requesting another control character. This continues until the control character input is N.

If the control character input is N, SMAIN assumes the additions at this point are complete. It continues copying from the old SGL and outputs the message

```
END REPLACEMENTS
```

The entire process is repeated when the next item specified by one of the parameters, p(n), of the ADD directive is found. The items in the directive need not be in the same order as they appear on the old SGL.

Example: During generation of a new SGL, add object module(s) and/or control record(s) after the old SGL control record PART0001 and after the old SGL object module LMP, the added items to be input from the logical unit specified by the ALT directive. Input

```
ADD, PART0001, LMP
```

then, when the message

```
ADD AFTER PART0001
SM**
```

appears, input the control character Y. SMAIN then inputs the next item on the logical unit specified by the ALT directive, and again outputs the message

```
SM**
```

and awaits another control character. If more is to be added here, input Y. If no more additions are required at this point, input N. After receiving the N, SMAIN outputs the message

```
END REPLACEMENTS
```

and continues to read the old SGL and copy it into the new SGL up to and including the object module LMP. SMAIN then outputs the message

```
ADD AFTER LMP
SM**
```

at which time the process is repeated.

Note that PART0001 does not have to precede LMP in the old SGL. If the positions of the items are reversed relative to their order in the directive, the order of messages will be reversed. In any case, the items on the logical unit specified by ALT must be in the order in which they are to be added to the SGL.

16.2.5 REP (Replace) Directive

This directive permits the replacement of object modules and/or control records during generation of a new SGL. The directive has the general form

```
REP, p(1), p(2), ..., p(n)
```

where each p(n) is the name of an object module or control record that is to be replaced.

Each p(n) has the option of selecting the occurrence of the module (they must be contiguous). The form is:

```
p(n)/#
```

where # is the selected occurrence of the module p(n) as handled. For example, to replace all occurrences of V\$FUNC and the second occurrence of V\$IOC, enter:

```
REP, V$FUNC, V$IOC/2
```

SMAIN copies object modules and control records from the old SGL into the new SGL until it encounters one specified by one of the parameters, p(n), of the REP directive. SMAIN then reads the item to be replaced, but does not copy it into the new SGL. After this is completed, the message

```
REPLACE p(n)
SM**
```

is output to indicate that SMAIN is waiting for a control character (Y or N) to be input on the SO logical unit. These control characters operate just as in the ADD directive (section 16.2.4), allowing the addition (in this case, replacement, since the parameter item was not copied into the new SGL) of new items to the SGL. The items in the directive need not be in the same order as they appear in the old SGL.

Example: During generation of a new SGL, replace the old

SGL object module IOCTL with object modules and/or control records from the logical unit specified by an ALT directive (section 16.2.3). Input

```
REPLACE, IOCTL
```

SYSTEM MAINTENANCE

then, when the message

```
REP IOCTL  
SM**
```

appears, continue as for an ADD directive (section 16.2.4).

16.2.6 DEL (Delete) Directive

This directive permits the deletion of object modules and/or control records during generation of a new SGL. The directive has the general form

```
DEL,p(1),p(2),...,p(n)
```

where each $p(n)$ is the name of an object module or control record that is to be deleted.

Each $p(n)$ has the option of selecting the occurrence of the module (they must be contiguous). The form is:

```
p(n)/#
```

where # is the selected occurrence of the module $p(n)$ as handled. For example, delete all occurrences of V\$FUNC and the second occurrence of V\$IOC, enter:

```
DEL,V$FUNC,V$IOC/2
```

SMAIN copies object modules and control records from the old SGL into the new SGL until it encounters one specified by one of the parameters, $p(n)$, of the DEL directive. SMAIN then reads the item to be deleted, but does not copy it into the new SGL. The items in the DEL directive need not be in the same order as they appear on the old SGL.

If a listing of the old SGL is specified either by a LIST directive (section 16.2.7) or by the L parameter of an END directive (16.2.8), the deleted items are preceded on the listing by asterisks (*).

Example: During generation of a new SGL, delete the following old SGL items: object module IOST and control record LMGENCTL.

```
DEL,IOST,LMGENCTL
```

16.2.7 LIST Directive

This directive lists, on the LO logical unit, the old SGL as found on the logical unit specified by the SMAIN directive IN (section 16.2.1). The LIST directive has the form

```
LIST
```

Example: List the old SGL.

```
LIST
```

Figure 16-2 shows the format of output from this directive.

16.2.8 END Directive

This directive indicates that all ADD (section 16.2.4), REP (section 16.2.5), and DEL (section 16.2.6) directives have been input. END initiates the SGL maintenance process. The directive has the general form

```
END,L
```

where L, if present, specifies that the old SGL is to be listed.

Examples: After all ADD, REP, and DEL directives have been input, initiate SGL maintenance processing.

```
END
```

Initiate the SGL maintenance processing as above, but list the old SGL.

```
END,L
```

16.3 SYSTEM-MAINTENANCE OPERATION

The normal SMAIN operation consists of copying an existing SGL from the logical unit specified by the IN directive (section 16.2.1) to the logical unit specified by the OUT directive (section 16.2.2), making the modifications specified by the ADD (section 16.2.4), REP (section 16.2.5), and DEL (section 16.2.6) directives, and thus creating a new SGL.

Input of the END directive (section 16.2.8) initiates the copying process. All ADD, REP, and DEL directives, if any, must precede the END directive.

Modifications to the SGL are made through the logical unit specified by the ALT directive (section 16.2.3). Such modifications are in the form of additions and/or replacements of object modules and/or control records. (These items can also be deleted, but this process does not, of course, require input on the ALT unit.)

When an object module is input, SMAIN verifies that there is no error with respect to check-sum, record size, loader codes, sequence numbers, or structure.

16.4 PROGRAMMING EXAMPLES

Example 1: Schedule SMAIN, copy the old SGL from logical unit 4 onto logical unit 9 without listing the old SGL, and return to the JCP.

```
/SMAIN  
IN,4  
OUT,9  
END  
/ENDJOB
```

Example 2: Schedule SMAIN; copy the old SGL from logical unit 4 onto logical unit 9, listing the old SGL and

deleting object modules A, B, C, D, and E; and return to the JCP.

```

/SMAIN
IN, 4
OUT, 9
DEL, A
DEL, B, C, D, E
END, L
/ENDJOB
    
```

Example 3: Schedule SMAIN, list the contents the old SGL on logical unit 4, and return to the JCP.

```

/SMAIN
IN, 4
LIST
/ENDJOB
    
```

Example 4: Schedule SMAIN; copy the old SGL from logical unit 4 onto logical unit 9 without listing the old SGL; add object modules or control records from logical unit 6 after control record PART0002 and after object module A; replace load module LMGEM and control record JCPDEF; delete object modules B, C, D, and E; and return to the JCP.

```

/SMAIN
IN, 4
OUT, 9
ALT, 6
ADD, PART0002, A
REP, LMGEM
DEL, B, C, D, E
REP, JCPDEF
END
/ENDJOB
    
```

PAGE		11/13/72		VORTEX SMAIN			
		IN, M1					
		OUT, PU					
		LIST					
		ROUTLDDR					
		ID NAME	DATE	SIZE	ENTRY NAMES	EXTERNAL NAMES	
		VSSGENLD	10/02/72	1551	SGLDR	TPROG	SGIBUF
						BSTACK	SPUN
						SPUB	SLUN
						SLUB	
		ID NAME	DATE	SIZE	ENTRY NAMES	EXTERNAL NAMES	
		VSD00A1	02/24/72	36	000A1	DRWEOP	DRSTAT
						DRSKRD	DRSFIL
						DRRITE	DRREWD
						DRREAD	
		ID NAME	DATE	SIZE	ENTRY NAMES	EXTERNAL NAMES	
		VSD00A2	02/24/72	36	000A2	DRWEOP	DRSTAT
						DRSKRD	DRSFIL
						DRRITE	DRREWD
						DRREAD	
		ID NAME	DATE	SIZE	ENTRY NAMES	EXTERNAL NAMES	
		VSD00A5	02/24/72	36	000A5	DRWEOP	DRSTAT
						DRSKRD	DRSFIL
						DRRITE	DRREWD
						DRREAD	
		ID NAME	DATE	SIZE	ENTRY NAMES	EXTERNAL NAMES	
		VSD10A1	02/24/72	36	D10A1	DRWEOP	DRSTAT
						DRSKRD	DRSFIL
						DRRITE	DRREWD
						DRREAD	
		ID NAME	DATE	SIZE	ENTRY NAMES	EXTERNAL NAMES	
		VSD10A2	02/24/72	36	D10A2	DRWEOP	DRSTAT
						DRSKRD	DRSFIL
						DRRITE	DRREWD
						DRREAD	
		ID NAME	DATE	SIZE	ENTRY NAMES	EXTERNAL NAMES	
		VSD10A5	02/24/72	36	D10A5	DRWEOP	DRSTAT
						DRSKRD	DRSFIL
						DRRITE	DRREWD
						DRREAD	
		ID NAME	DATE	SIZE	ENTRY NAMES	EXTERNAL NAMES	
		VSD20A1	02/24/72	36	D20A1	DRWEOP	DRSTAT

Figure 16-2. SMAIN LIST Directive Listing

SECTION 17 OPERATOR COMMUNICATION

The operator communicates with the VORTEX system through the **operator communication component** by means of *operator key-in requests* input through the *operator communication (OC) logical unit*.

17.1 DEFINITIONS

An **operator key-in request** is a string of up to 80 characters beginning with a semicolon. The request is initiated by the operator and is input through the OC unit. An operator key-in request is independent of I/O requests via the IOC (section 3) and, hence, is known as an *unsolicited request*.

The **operator communication (OC) logical unit** is the logical unit through which the operator inputs key-in requests. There is only one OC unit in the VORTEX system. Initially, the OC unit is the first Teletype, but this assignment can be changed by use of the ;ASSIGN key-in request (section 17.2.9).

17.2 OPERATOR KEY-IN REQUESTS

This section describes the operator key-in requests:

- ;SCHED Schedule foreground task
- ;TSCHED Time-schedule foreground task
- ;ATTACH Attach foreground task to PIM line
- ;RESUME Resume task
- ;TIME Enter or display time-of-day
- ;DATE Enter date
- ;ABORT Abort task
- ;TSTAT Test task status
- ;ASSIGN Assign logical unit(s)
- ;DEVDN Device down
- ;DEVUP Device up
- ;IOLIST List logical-unit assignments

Operator key-in requests comprise sequences of character strings having no embedded blanks. The character strings are separated by commas (,) or by equal signs (=). However, the key-in requests are free-form and blanks are permitted between the individual character strings of the key-in request, i.e., before or after commas (or equal signs). Although not required, a period (.) is a line terminator. Comments can be inserted after the period. A carriage return is required to terminate any key-in request, however, regardless of whether it contains a period.

The general form of an operator key-in request is

;request,p(1),p(2),...,p(n)cr

where

- request** is one of the key-in requests listed above in capital letters
- each *p(n)* is a parameter defined under the descriptions of the individual key-in requests below
- cr* is the carriage return, which terminates all operator key-in requests

Each operator key-in request begins with a semicolon (;) and ends with a carriage return. Parameters are separated by commas. A backarrow (←) deletes the preceding character. A backslash (\) deletes the entire present key-in request.

Table 17-1 shows the system names of physical I/O devices as used in operator key-in requests.

Peripherals for data communication are not used in OPCOM request, but are controlled with the Network Control Module (NCM) described in the VTAM Reference Manual.

For greater clarity, optional blank separators between character strings, and the optional replacement of commas (,) by equal signs (=) are omitted from the descriptions of the key-in requests.

Error messages applicable to operator key-in requests are given in Appendix A.17.

Table 17-1. Physical I/O Devices

System Name	Physical Device
DUM	Dummy
CPcu	Card punch
CRcu	Card reader
CTcu	Cathode ray tube (CRT) device
Dcup	Rotating-memory device (RMD) (disc/drum)
LPcu	Line printer or Statos
MTcu	Magnetic tape unit
PTcu	High-speed paper tape reader/punch
TYcu	Teletype printer/keyboard
ClmA, COmA	Process I/O

OPERATOR COMMUNICATION

Table 17-1. Physical I/O Devices (continued)

System Name	Physical Device
MXcu	Communication Multiplexor
TCc0	Psuedo TCM
SPc0	Spool Unit

NOTES

c = Controller number. For each type of device, controllers are numbered from 0 as required.

u = Unit number. For each controller, units are numbered from 0 as required (within the capacity of the controller).

p = Partition letter. RMD partitions are lettered from A to T as required to refer to a partition on the specified device, e.g., D00A.

m = Multiplexor number

17.2.1 ;SCHED (Schedule Foreground Task) Key-In Request

This key-in request immediately schedules the specified foreground-library task for execution at the designated priority level. It has the general form

;SCHED,task,level,lun,key,trace

where

- task** is the name of the foreground task to be scheduled
- level** is the priority level (from 2 to 31) of the scheduled task
- lun** is the number or name of the foreground-library rotating-memory logical unit where the scheduled task resides
- key** is the protection code, if any, required to address lun
- trace** is T if the task to be scheduled is to be executed in TRACE mode. (Applies to V77-800 systems only) see paragraph 7.4.

A dump of the contents of a library can be obtained by use of the VORTEX file-maintenance component (section 9).

Operator key-in examples: Schedule on priority level 3 the foreground task DOTASK residing on the FL logical unit. Use F as the protection key. The task is to be executed in the TRACE mode.

;SCHED, DOTASK, 3, FL, F, T

Schedule on priority level 9 the resident foreground task COPYIO.

;SCHED, COPYIO, 9, 0

17.2.2 ;TSCHED (Time-Schedule Foreground Task) Key-In Request

This key-in request schedules the specified foreground-library task for execution at the designated time-of-day and priority level. It has the general form

;TSCHED,task,level,lun,key,time

where

- task** is the name of the foreground task to be scheduled
- level** is the priority level (from 2 to 31) of the scheduled task
- lun** is the number or name of the foreground-library rotating-memory logical unit where the scheduled task resides (0 for scheduling a resident foreground task)
- key** is the protection code, if any, required to address lun
- time** is the scheduled time in hours (from 00 to 23) and minutes (from 00 to 59), e.g., 1945 for 7:45 p.m.

Note: If a task has already been scheduled using a TSCHED command, a subsequent TSCHED request for the task replaces the first request.

Operator key-in examples. Schedule for execution at 11:30 p.m. on priority level 3 the foreground task DOTASK residing on the US logical unit. Use T as the protection key.

;TSCHED, DOTASK, 3, US, T, 2330

Schedule for execution at 8:30 a.m. on priority level 9 the resident foreground task TESTIO.

;TSCHED, TESTIO, 9, 0, 0830

17.2.3 ;ATTACH Key-In Request

This key-in request attaches the specified foreground task to the designated PIM (priority interrupt module) line. It has the general form

;ATTACH,task,line,iew,enable

where

task is the name of the foreground task to be attached to the PIM line

line is the two-digit number of the PIM line to which the task is to be attached, with the tens digit specifying the PIM number (0-7) and the units digit the line number (0-7) on that PIM

iew is the value (from 01 to 0177777) of the interrupt event word (section 14 or appendix F) and identifies the bit(s) to be set in the task TIDB when an interrupt occurs on **line**

enable is E (default value) to enable the line, or D to disable it

The **task** can be resident or nonresident. However, its TIDB must have been defined at system-generation time. ATTACH provides a flexible way of altering interrupt assignments without having to regenerate the system.

Operator key-in example: Connect task INTRPT to PIM 0, line 3. Use 020 as the interrupt event word value (i.e., set bit 4 of the interrupt event word in TIDB if INTRPT is scheduled due to an interrupt on PIM 0, line 3).

;ATTACH,INTRPT,03,020

A PIM directive with the PIM line to be attached must have been specified during system generation to set up the link to the interrupt line handler region.

Note: This directive detaches the PIM from a previous task.

17.2.4 ;RESUME Key-In Request

This key-in request reactivates the specified task for execution at its specified priority level. It has the general form

;RESUME,task

where **task** is the name of the task to be resumed

Operator key-in example: Resume the task DOTASK.

;RESUME,DOTASK

17.2.5 ;TIME Key-In Request

This key-in request enters the specified time, if any, as system time-of-day. If no time is specified in the key-in request, ;TIME displays the current time-of-day. The key-in request has the general form

;TIME,time

where **time** is the time-of-day in hours (from 00 to 23) and minutes (from 00 to 59), e.g., 1945 for 7:45 p.m.

The time-of-day output for a ;TIME request without **time** is of the form

T hhmm HRS

where **hhmm** is the time of day in hours and minutes.

Operator key-in example: Set the system time-of-day to 3:00 p.m.

;TIME,1500

17.2.6 ;DATE Key-In Request

This key-in request enters the specified date as the system date. It has the general form

;DATE,mm/dd/yy

where

mm is the month (01 to 12)

dd is the day (01 to 31)

yy is the year (00 to 99)

Note that since the entire date is considered one parameter, there are no commas other than the one immediately following DATE. The components of the date are, however, separated by slashes as shown. VORTEX does not support date roll-over.

Operator key-in example: Set the system date to 25 December 1971.

;DATE,12/25/71

OPERATOR COMMUNICATION

17.2.7 ;ABORT Key-In Request

This key-in request aborts the specified task. It has the general form

;ABORT,task

where **task** is the name of the task to be aborted

Operator key-in example: Abort the task DOTASK.

;ABORT, DOTASK

17.2.8 ;TSTAT (Task Status) Key-In Request

This key-in request outputs the status of the specified task, if any. If no task is specified, ;TSTAT outputs the status of all tasks queued on the active task identification block (TIDB) stack. This request is not applicable to tasks having no established TIDB. The request has the general form

;TSTAT,task

where **task** is the name of the task whose status is to be output.

The status-output for a ;TSTAT key-in request is of the form

task Plevel Sstatus TMmin TSmilli

where

task is the name of the task whose status is being output

level is the priority level (from 0 to 31) of the task

status is the status of the task as found in words 1 and 2 of the TIDB (table 17-2)

min is the value of the counter in TIDB word 11

milli is the value of the counter in TIDB word 10

The values of min and milli are printed only if bit 6 and/or 7 of TIDB word 1 (table 17-2) is set.

Table 17-2. Task Status (TIDB Words 1 and 2)

TIDB Word	Bit	Meaning of Set Bit
1	15	Suspend interrupt
1	14	Suspend task
1	13	Abort task

1	12	Exit from task
1	11	TIDB resident
1	10	Resident task
1	9	Foreground task
1	8	Protected task
1	7	Task scheduled by time-delay
1	6	Time-delay active
1	5	Task waiting to be loaded (check pointed)
1	4	Task error
1	3	Task interrupt expected
1	2	Overlay task
1	1	Scheduled task upon termination of active task
1	0	Task search-allocated-loaded
2	15	Task opened, but not loaded
2	14	Task loaded in background (checkpoint) area
2	13	Load overlay
2	12	Background checkpoint I/O wait
2	11	Allocation override flag
2	10	Background being checkpointed
2	9	TIDB not available
2	8	Unused
2	7	Unused
2	6	Delay type 3 request
2	5-0	Task priority level

Operator key-in examples: Request the output of the status of the task BIGJOB.

;TSTAT, BIGJOB

The output will be

BIGJOB P02 S000100, 000000 TM077777 TS077430

if the status BIGJOB is such that it is on priority level 2, contains a status of 0100 in TIDB words 1 and 2, with time counters (TIDB words 1 and 10) of 077777 and 077430, respectively. The latter two octal complement counters show zero minutes and 0347 5-millisecond increments.

Request the output of the status of all active tasks.

;TSTAT

and receive as a typical response

VZDB P24 S047401, 000000
V\$TYA P23 S047411, 000000
V\$TYA P23 S047411, 000000
VZLPA P22 S047401, 000000
VZCRA P22 S047401, 000000

```
VZMTA    P22      S047401,  000000
VZMTA    P22      S047401,  000000
V$OPCM   P10      S005405,  020000
PROG1    P05      S041501,  000000
JCP      P01      S044400,  000000
```

17.2.9 ;ASSIGN Key-In Request

This key-in request equates and assigns particular logical units to specific I/O devices. It has the general form

```
;ASSIGN,l(1) = r(1),l(2) = r(2),...,l(n) = r(n)
```

where

each *l(n)* is a logical-unit number (e.g., 12) or name (e.g., SI)

each *r(n)* is a logical-unit number or name, or a physical-device system name (e.g., TY00 or TY, table 17-1)

The logical unit to the left of the equal sign in each pair is assigned to the unit/device to the right.

An inoperable device, i.e., one declared down by ;DEVDN (section 17.2.10), cannot be assigned. A logical unit designated as unassignable (unit numbers 101 through 179) cannot be reassigned.

Operator key-in examples: Assign the card reader CR00 as the SI logical unit and the Teletype TY01 as the OC unit.

```
;ASSIGN,SI=CR00,OC=TY01
```

Assign a dummy device as the PI unit.

```
;ASSIGN,PI=DUM
```

17.2.10 ;DEVDN (Device Down) Key-In Request

This key-in request declares the specified physical device inoperable for system use. It is not applicable to the OC unit or to devices containing system libraries. The request has the general form

```
;DEVDN,device
```

where *device* is the system name of the physical device in four ASCII characters, e.g., LP00 (or LP), TY01, (table 17-1)

Operator key-in example: Declare TY01 inoperable for system use.

```
;DEVDN, TY01
```

17.2.11 ;DEVUP (Device Up) Key-In Request

This key-in request declares the specified physical device operational for system use. It has the general form

```
;DEVUP,device
```

where *device* is the system name of the physical device in four ASCII characters, e.g., LP00 (or LP), TY01 (table 17-1)

Operator key-in example: Declare TY02 operational for system use.

```
;DEVUP, TY02
```

17.2.12 ;IOLIST (List I/O) Key-In Request

This key-in request outputs a listing of the specified logical-unit assignments, if any. If no logical unit is specified, ;IOLIST outputs all logical-unit assignments with names. The key-in request has the general form

```
;IOLIST,lun(1),lun(2),...,lun(n)
```

where each *lun(n)* is the name or number of a logical unit, e.g., SI.5.

Where the ;IOLIST key-in request specifies a logical-unit name, the output is of the form

```
name (number) = device D
```

where

name is the name of the logical unit, e.g., LO

number is the number of that logical unit, e.g., 005

device is the name of the physical device assigned, e.g., LP00

D if present, indicates that the physical device has been declared down and is thus inoperable

If the key-in request specifies the number rather than the name of the logical unit, the output will repeat the number in both the **name** and **number** fields.

In a listing of all assignments, the output uses a name and number where applicable. Logical units without names assigned at system-generation time are not listed and must be individually specified by number.

OPERATOR COMMUNICATION

Operator key-in examples: Request the output of the logical-unit assignments for the BI and BO units. Input

```
; IOLIST, BI, BO
```

and receive as a typical response

```
BI (006) = CR00
BO (007) = CP00 D
```

Request the output of the logical-unit assignment for logical unit 180. Input

```
; IOLIST, 180
```

and receive as a typical response

```
180 (180) = D11H
```

Request the output of all logical-unit assignments. Input

```
; IOLIST
```

and receive as a typical response

```
OC (001) = TY00
SI (002) = TY00
SO (003) = TY00
PI (004) = CR00 D
LO (005) = LP00
BI (006) = CR00 D
BO (007) = PT00
SS (008) = D00H
PO (009) = D00H
CU (100) = D00E
GO (101) = D00G
SW (102) = D00F
CL (103) = D00A
OM (104) = D00D
BL (105) = D00C
FL (106) = D00B
```

17.3 BLDSKD

The standard VORTEX teletype drivers support the foreground scheduling task BLDSKD. BLDSKD allows the operator to define selected control characters (A-Z, excluding G and M) to cause scheduling of a selected task. Use of the defined control character as the input to an unsolicited request results in the same function as using the OPCOM; SCHED request. The operator may add, delete, or list control characters and their definitions. Using an undefined control character schedules BLDSKD, therefore it is suggested that one control character be reserved for BLDSKD scheduling. Entering a carriage return only once BLDSKD has been activated causes BLDSKD to exit.

17.3.1 Set-Up Requirements

Before attempting to use BLDSKD the operator must create a file called SCHEDAD on the foreground library

containing two 120 word records. This should be done with FMAIN. For example:

```
FMAIN
CREATE, FL, F, BLDSKD, 120, 2
```

17.3.2 TASK SCHEDULING

Typing a control character will schedule BLDSKD. If the particular control character has not previously been assigned to a task, BLDSKD will respond with

```
TY** ADD TASK, DELETE TASK, OR LIST? ENTER
      A, D, OR L
TY**
```

The operator then responds appropriately with either A, D, or L

'A' Command

```
TY** ENTER TASK NAME
TY**
```

Operator enters a 6 character (or less) alpha numeric task name he wishes to have scheduled, followed by (carriage return):

```
TY** ENTER PRIORITY LEVEL
TY**
```

Operator enters the priority level at which the task is to be scheduled.

```
TY** ENTER 1 FOR WAIT OR 0 FOR IMMEDIATE
TY**
```

Operator enters 0 or 1

```
TY** ENTER OCTAL LOGICAL UNIT NUMBER
TY**
```

Operator enters LUN where task to be scheduled resides.

```
TY** ENTER PROTECT KEY OR CARRIAGE RETURN
TY**
```

Operator enters LUN protect key or Carriage Return if unprotected.

```
TY** CHARACTER ADDED
TY** INITIALIZATION COMPLETE, EXITING TASK
      SCHEDULER
```

Task is now scheduled automatically. Henceforth, upon entering a control character, with an already assigned task, BLDSKD will immediately schedule the user task.

'D' Command

```
TY** ENTER ALPHA ONLY OF CHARACTER TO BE
      DELETED
TY**
```


After a control character has been assigned a task via BLDSKD 'A' command, it may at any time be deleted via the 'D' command. Upon entering the character to be deleted, BLDSKD responds with:

```
TY** CHARACTER DELETED
TY** EXIT
```

'L' Command

BLDSKD responds with:

```
a bbbbbb
where
a is the defined control character
bbbbbb is the task attached to character a
```


SECTION 18

OPERATION OF THE VORTEX SYSTEM

This section explains the operation of devices in the VORTEX system, the loading of the system bootstrap loading and initializing of writable control store and procedures for changing and initializing the disc pack during VORTEX operation.

18.1 DEVICE INITIALIZATION

18.1.1 Card Reader

(Model 70-6200)

- a. Turn on the card reader.
- b. Place the input deck in the card hopper.
- c. Press READY/ALERT.

18.1.2 Card Punch

(Model 70-6200)

- a. Turn on the card punch.
- b. Place blank cards in the card hopper.
- c. If the visual punch station is empty, insert a card into it as follows:
 - (1) Place a card in the auxiliary feed slot.
 - (2) Clear all registers.
 - (3) Set the instruction register I to 0100131.
 - (4) Set REPEAT.
 - (5) Press STEP. The card should move from the auxiliary feed slot to the visual punch station.
 - (6) Reset REPEAT.

18.1.3 Line Printer

(Model 70-6701)

- a. Turn on the line printer.
- b. Wait for the READY light to come on.
- c. Set the ON LINE/OFF LINE switch to ON LINE.
- d. For manual paper ejection set to OFF LINE, then press the TOP OF FORM switch.

18.1.4 Statos-31 (Model 70-6602 and -6603)

- a. Turn on plotter/printer
- b. Set the ON LINE/OFF LINE switch to ON LINE
- c. Select roll or z-fold paper switch for paper type used
- d. For manual form feed press FORM FEED

18.1.5 33/35 ASR Teletype

(Models 70-6200 and 6201)

- a. Turn on the Teletype.
- b. Set the Teletype in off-line mode and simultaneously press the CONTROL and D, then the CONTROL and T, finally the CONTROL and Q keys.
- c. Set the Teletype on-line.

18.1.6 High-Speed Paper-Tape Reader

(Model 70-6320)

- a. Turn on the paper-tape reader.
- b. Position the input paper tape in the reader with blank leader at the reading station and close the reading gate.
- c. Set the LOAD/RUN switch to RUN.

18.1.7 Magnetic-Tape Unit

(Models 70-7100, -7102, and 620-31)

- a. Turn on the magnetic-tape unit.
- b. Mount the input magnetic tape.
- c. Position the magnetic tape to the loading point.
- d. Press ON LINE.

18.1.8 Magnetic-Drum and Fixed-Head Disc Units

(Models 620-47 through 620-49, 70-7702 and 70-7703)

- a. Turn on the drum unit.
- b. Wait for the drum unit to reach operating speed.

18.1.9 Moving-Head Disc Units

(Models 70-7600 and 70-7610)

- a. Place the START/STOP switch in the STOP position.
- b. Press POWER ON button and wait for the SAFE light to come on.
- c. Mount the disc pack.
- d. Place the START/STOP switch in the START position.
- e. Wait for the disc unit to reach operating speed (READY indicator lights).

OPERATION OF THE VORTEX SYSTEM

f. Turn off WRITE PROTECT.

Table 18-1. Key-In Loader Programs

18.1.10 Moving-Head Disc Units (Model 70-7500)

- Mount the disc pack
- Press POWER-ON button and wait for unit to reach operating speed and for the heads to emerge
- Press on-line button.

18.1.11 Moving-Head Disc Units (Model 70-7510)

- Mount the disc pack(s).
- Turn power on and wait for the unit(s) to reach operating speed (unit-ready light comes on).

18.1.12 Moving-Head Disc Units (Models 70-7603, 70-7613)

- Mount disc pack.
- Press START button and wait for Ready light.

18.1.13 Moving Head Disc Units (Models 70-7520, 70-7530)

- Mount disc pack.
- Place in run.
- Wait for unit to reach operating speed (run light will stop blinking).

Address	Drum -48,49	Disc 70-7510 -7220, -7530 <i>ctrl comp</i>	Disc 70-7500	Disc 70-7600, -7610, -7603 or 7613
001130	1000yy	005302	005302	1004zz
001131	006020	006030	006030	1040zz
001132	000002	000005	177773	1002zz
001133	005001	005001	005001	005001
001134	1031xx	1000zz	1000zz	1031zz
001135	006120	1031zz	1031zz	1010zz
001136	001127	1005zz	1005zz	001141
001137	1031yy	1010zz	1010zz	001000
001140	1000xx	001143	001143	001135
001141	1000zz	001000	001000	1025zz
001142	1032zz	001137	001137	151167
001143	1010xx	1025zz	1025zz	001016
001144	000600	001016	001016	001130
001145	001000	001200	001130	1000yy
001146	001143	005123	005122	1003zz
001147		006120	005021	005102
001150		000167	006120	1032zz
001151		004460	000167	1031xx
001152		1000zz	004460	006010
001153		1000yy	1000zz	001130
001154		1031xx	1000yy	1031yy
001155		103?yy	1031xx	1000xx
001156		1000xx	1032yy	1000zz
001157		005041	1000xx	1014zz
001160		1031zz	005041	001157
001161		1004zz	006150	1025zz
001162		1014zz	000007	151167
001163		001166	1031zz	001016
001164		001000	1004zz	001130
001165		001162	1014zz	001000
001166		1025ZZ	001171	000600
001167		001016	001000	007760
001170		000120	001165	
001171		005145	1025ZZ	
001172		006140	001016	
001173		000012	001130	
001174		001002	005144	
001175		000600	001040	
001176		001000	000600	
001177		001146	001000	
001200		000000	001146	

where xx = even BIC address, yy = odd BIC address, and zz = device address.

18.2 SYSTEM BOOTSTRAP LOADER

System key-in loaders initiate loading of the VORTEX system from a drum or disc memory. The key-in loader loads the system initializer from the RMD to main memory (locations 000000 to 001127). The system initializer then loads and initializes the system. Table 18-1 contains the key-in loader programs.

18.2.1 Automatic Bootstrap Loader

Where the automatic bootstrap loader option is available, the appropriate key-in loader is loaded from the required medium (high-speed paper-tape or Teletype reader) into locations starting with 001130. If the system contains a V70 RMD ABL the boot program is automatically loaded and executed.

To initiate the loader: (1) clear the A, B, X, I, and P registers; (2) with the computer in STEP, press the RESET switch on the front panel; (3) place the STEP/RUN switch in the RUN position; and (4) press and release the LOAD switch.

18.2.2 Control Panel Loading

The appropriate key-in loader is entered through the computer control panel. Refer to the hardware handbook for details.

To initiate the bootstrap, clear the A, B, X, and I registers, and load 001130 into the P register. Then, press RESET, place the STEP/RUN switch in the RUN position, and press START. See section 15.8 and 20.1.4 for details as system initialization messages.

NOTE: To facilitate reloading, the key-in loader may be dumped out on paper tape and then loaded by the binary loader (BLD II).

18.3 DISC PACK HANDLING

VORTEX provides for dynamic mounting of disc packs during program execution by means of a system utility program called **rotating memory analysis and initialization (RAZI)**. RAZI handles:

- a. A disc pack not previously used with VORTEX that is replacing a disc pack presently in the system.
- b. A disc pack previously formatted under VORTEX that is replacing a disc pack presently in the system.

The normal RAZI operating procedure is:

- a. The task requiring the disc pack change issues an operator message directing him to switch packs.
- b. The task suspends itself.
- c. The operator makes the necessary pack changes.
- d. The operator schedules and executes RAZI.
- e. Upon completion of RAZI, the operator resumes the suspended task. The task can now perform I/O on the new pack.

RAZI is a foreground program residing in the foreground library (FL). It is scheduled by a request of the form:

```
;SCHED,RAZI,p,FL,F
```

where *p* is the priority level.

If the SI logical unit is a Teletype or a CRT device, the message RZ** is output to indicate that the Si unit is waiting for RAZI input.

Each directive is completely processed before the next is entered. All directives are output on the SO device. In addition, partitioning information is listed on the LO device when integration of the requested disc pack is complete.

OUTPUTS from the RAZI comprise:

- a. Error messages
- b. The listing of the RAZI directives on the SO unit
- c. Partition description listing

Error messages applicable to RAZI are output on the SO and LO logical units. The individual messages and errors are given in Appendix A.18.

The **partition description listing** is output on the LO device upon completing the integration of a new disc pack into the VORTEX system. After the VORTEX standard heading, there are three blank lines followed by the RAZI heading:

PARTITION NAME	FIRST TRACK	LAST TRACK	BAD TRACKS
-------------------	----------------	---------------	---------------

followed by one more blank line. Then the information concerning each partition of the device is output, one partition per line, as shown in the following example.

PARTITION NAME	FIRST TRACK	LAST TRACK	BAD TRACKS
D10A	0002	0019	0000
D10B	0020	0052	0001
D10C	0053	0082	0000
D10D	0083	0118	0000
D10E	0119	0126	0000
D10F	0127	0141	0000
D10G	0142	0156	0000
D10H	0157	0206	0002
D10I	0207	0242	0000
D10J	0243	0251	0000
D10K	0252	0256	0000

The RAZI directives are:

- PRT Partition
- FRM Format rotating memory
- INL Initialize
- EXIT Exit

RAZI directives begin in column 1 and comprise sequences of character strings having no embedded blanks. The character strings are separated by commas (,) or equal signs (=). The directives are free-form, and blanks are permitted between the individual character strings of the directive, i.e., before or after commas (or equal signs).

OPERATION OF THE VORTEX SYSTEM

The general format of a RAZI directive is

`name,p(1),p(2),...,p(n)`

where

`name` is one of the directive names given above

each `p(n)` is a parameter required by the directive and defined below under descriptions of the individual directives

Numerical data can be octal or decimal. Each octal number has a leading zero.

For greater clarity in the descriptions of the directives, optional periods, optional blank separators between character strings, and the optional replacement of commas (,) by equal signs (=) are omitted.

Note: The disc pack containing the VORTEX nucleus cannot be replaced.

18.3.1 PRT (Partition) Directive

This directive specifies the size and protection code for each RMD partition. It has the general form

`PRT,p(1),s(1),k(1),p(2),s(2),k(2),...,p(n),s(n),k(n)`

where

each `p(n)` is the RMD partition letter (A through T, inclusive)

`s(n)` is the number (octal or decimal) of tracks in the partition. This value must be greater than zero.

`k(n)` is the protection code, if any, required to address `p`, or * if the partition is unprotected

While the partition specifications can appear in any order, the set of partitions specified for each RMD must comprise a contiguous group, e.g., the sequence A, C, D, B is valid but, the sequence A, C, D, E constitutes an error.

Consecutive PRT directives redefine partitions, if `p(n)` has been specified, or adds partitions if `p(n)` is new partition letter.

Example: Define three partitions on an RMD. The first occupies ten tracks and uses protection code Q, the second two tracks and code S, and the third 48 tracks without protection.

`PRT,A,10,Q,B,2,S,C,060,*`

18.3.2 FRM (Format Rotating Memory) Directive

This directive causes RAZI to run a bad-track analysis on the specified RMD and build a new PST for it or accepts a previously constructed bad-track-table from the RMD and builds a new PST for it.* The directive has the general form

`FRM,lu,size,flag`

where

`lu` is the logical-unit name or number to which the subject RMD is assigned. This must be assigned to the first partition.

`size` is the number (octal or decimal) of tracks on the RMD

`flag` is 1 to perform a complete bad-track analysis, or 0 to accept a bad-track-table from the RMD

*FRM clears all PSTs and directories. It should not be used when a unit contains a good BIT and files as these will be destroyed.

Caution: When performing a bad-track analysis or accepting a bad-track table from an RMD the bad-track table is positioned adjacent to the resident foreground task area. Unless there already exists an active bad-track table for the prior RMD, the bad-track table for the new RMD will be overlaid, if the resident foreground area is increased by means of a partial SYSGEN. Thus if a partial SYSGEN is performed which increases the resident foreground size, another RAZI must be performed.

Examples: Clear the RMD assigned to PO, having 203 tracks, and build a PST for it according to previously defined partition information.

`FRM,PO,203,0`

Run a complete bad-track analysis on the RMD assigned to 25, having 128 tracks, and build a PST for it according to previously defined partition information.

`FRM,25,128,1`

620-35 and 620-34 discs in a system require the formatting program (describe in section 18.4) to format disc and analyze bad tracks.

18.3.3 INL (Initialize) Directive

This directive causes RAZI to incorporate a PST and a bad-track table from the named RMD into the VORTEX nucleus. It has the general form

where

lu	is the logical-unit name or number to which the subject RMD is assigned. This must be assigned to the first partition.
size	is the number (octal or decimal) of tracks on the RMD
flag	is 1 to perform a complete bad-track analysis, or 0 to accept a bad-track-table from the RMD

*FRM clears all PSTs and directories. It should not be used when a unit contains a good BIT and files as these will be destroyed.

Caution: When performing a bad-track analysis or accepting a bad-track table from an RMD the bad-track table is positioned adjacent to the resident foreground task area. Unless there already exists an active bad-track table for the prior RMD, the bad-track table for the new RMD will be overlaid, if the resident foreground area is increased by means of a partial SYSGEN. Thus if a partial SYSGEN is performed which increases the resident foreground size, another RAZI must be performed.

Examples: Clear the RMD assigned to PO, having 203 tracks, and build a PST for it according to previously defined partition information.

FRM, PO, 203, 0

Run a complete bad-track analysis on the RMD assigned to 25, having 128 tracks, and build a PST for it according to previously defined partition information.

FRM, 25, 128, 1

620-35 and 620-34 discs in a system require the formatting program (describe in section 18.4) to format disc and analyze bad tracks.

18.3.3 INL (Initialize) Directive

This directive causes RAZI to incorporate a PST and a bad-track table from the named RMD into the VORTEX nucleus. It has the general form

INL,lu,size

where **lu** and **size** have the same definition as in the FRM directive (section 18.3.2).

Example: Read the PST and bad-track table from the unit assigned to BO, having 128 tracks, and incorporate them into the VORTEX nucleus.

INL, BO, 128

18.3.4 EXIT Directive

This directive terminates RAZI. It has the general form

EXIT

Example: Terminate RAZI.

EXIT

18.4 WRITABLE CONTROL STORE (WCS)

The writable control store must be loaded with the appropriate firmware. The WCS is loaded by the V73 WCS Microprogram Utility (MIUTIL). MIUTIL is a foreground program scheduled by a request:

;SCHED,MIUTIL,p,FL,F

where **p** is the priority level. Use of the MIUTIL program is described in detail in the Microprogramming Guide.

If the optional V70 series Floating Point Firmware is to be used, it must be loaded into page 1 of WCS. The WCS microprogram is catalogued into the OM library under the name WCSFP, and must be transferred to the BI device for loading by MIUTIL. The WCS should be initialized through the use of MIUTIL prior to loading the floating-point microprograms.

Section 20 gives additional information about writable control store.

INL,lu,size

where **lu** and **size** have the same definition as in the FRM directive (section 18.3.2).

Example: Read the PST and bad-track table from the unit assigned to BO, having 128 tracks, and incorporate them into the VORTEX nucleus.

INL,BO,128

18.3.4 EXIT Directive

This directive terminates RAZI. It has the general form

EXIT

Example: Terminate RAZI.

EXIT

18.4 70-7500 (620-35) DISC PACK FORMATTING PROGRAM

Each 70-7500 (620-35) disc pack requires formatting before any input or output operation can be performed on it. Before VORTEX can be prepared on a 70-7500 disc pack or any 70-7500 discs can be used under VORTEX, disc packs must be formatted. The formatting program forms 120-word sectors, which are grouped 24 per track. The program also examines the disc pack for bad tracks.

The formatting program operates in a stand-alone mode. It may be loaded and executed with either AID or BLD. Execution begins at location 01354. Upon execution the formatting program requests some parameters to be input from the keyboard. The following requests are made. An inappropriate response causes the request to be repeated.

Request**INPUT BTC NUMBER**

Type a value and a carriage return. The acceptable values are octal 020, 022, 024, 026 and 070

INPUT DEVICE ADDRESS

Type a value in the range from octal 014 through 017 followed by a carriage return

INPUT VARIABLE SECTOR GAP

Type a value and carriage return. Acceptable values are 1, 2, 3, 4, 6, 8, 12, or their equivalent octal representations. This value determines the physical location on the disc pack of sequentially addressable sectors, as such sequential transfers may be accomplished without waiting for a

full revolution of the disc unit. Recommended setting is 3. Another setting may be more effective depending upon various application parameters such as number of tasks, frequency of disc transfers, and types of disc transfers.

INPUT UNIT NUMBER

Type unit number followed by a carriage return. Acceptable values are 0 through 3. Up to four units can be connected to a single controller.

In addition the formatting program performs bad-track analysis and creates and maintains a bad-track table, which is entered on each disc pack at the completion of its formatting. The bad-track table is located on sectors 0 through 2 of the first track. The table is 254 words long, starting at word 64 of sector 0. The first 64 words of sector 0 reserve the necessary space for the PST. The remaining unused words of sector 2 are filled with zeroes. Each disc I/O error will generate a ten-event retry sequence, which upon failure will set the bad-track flag within the track header. The program also sets the corresponding bit in the bad-track table. No alternate tracks are assigned.

If the first track is determined to be bad, the bad-track table may not be placed there. The program prints the error message,

FIRST TRACK BAD

and aborts formatting the current disc pack. The program returns to the keyboard interrogation routine. After the bad-track table has been written on the disc pack, the formatting program resumes the keyboard interrogation to obtain parameters for formatting the next disc. In this way, more than one disc pack can be formatted in the same session. The formatting program may be terminated at this point when no disc packs (except those with bad first tracks) remain unformatted. If an unsafe condition (SELECT LOCK light on) occurs, reload and execute the program. Formatting disc packs is not necessary before every VORTEX system generation.

18.5 70-7510 (620-34) DISC PACK FORMATTING PROGRAM

Each 620-34 disc pack requires formatting before any input or output operation can be performed on it. Before VORTEX can be prepared on a 620-34 disc pack or these disc can be used under VORTEX, the packs must be formatted. The formatting program forms 120-word sectors, which are grouped 24 per track. The program also examines the disc pack for bad tracks.

The formatting program operates without an operating system. It may be loaded and executed either with AID II or

OPERATION OF THE VORTEX SYSTEM

BLD II. Its execution begins at location 01354. Upon execution the formatting program requests some parameters to be input from the keyboard. An inappropriate response causes the request to be repeated. The following requests are made.

INPUT BTC NUMBER

Type a value and a carriage return. The acceptable values are octal 020, 022, 024, 026 and 070.

INPUT DEVICE ADDRESS

Type a value in the range from octal 014 through 017 followed by a carriage return.

INPUT VARIABLE SECTOR GAP

Type a value and a carriage return. Acceptable values are 1, 2, 3, 4, 6, 8, 12, or their equivalent octal representations. This value determines the physical location on the disc pack of sequentially addressable sectors, as such sequential transfers may be accomplished without waiting for a full revolution of the disc unit. Recommended setting is 3. Another setting may be more effective depending upon various application parameters such as number of tasks, frequency of disc transfers, and types of disc transfers.

INPUT UNIT NUMBER

Type unit number followed by a carriage return. Acceptable values are 0 through 3. Up to four units can be connected to a single controller.

In addition the formatting program performs bad-track analysis and creates and maintains a bad-track table, which is entered on each disc pack at the completion of its formatting. The bad-track table is located on sectors 0 through 4 of the first track. The table is 508 words long, starting at word 64 of sector 0. The first 64 words of sector 0 reserve the necessary space for the PST. The remaining unused words of sector 4 are filled with zeros. Each disc I/O error will generate a ten-event retry sequence, which upon failure will set the bad-track flag within the track header. The program also sets the corresponding bit in the bad-track table. No alternate tracks are assigned.

If the first track is determined to be bad, the bad-track table may not be placed there. The program prints the error message:

FIRST TRACK BAD

and aborts formatting the current disc pack. The program returns to the keyboard interrogation routine. After the bad-track table has been written on the disc pack, the formatting program resumes the keyboard interrogation to

obtain parameters for formatting the next disc. In this way, more than one disc pack can be formatted in the same session. The formatting program may be terminated at this point when no disc packs (except those with bad first tracks) remain unformatted. If an unsafe condition (SELECT LOCK light on) occurs, reload and execute the program. Formatting disc packs is not necessary before every VORTEX system generation.

18.6 70-7603/7613 DISC PACK FORMATTING PROGRAM

Each 70-7613/7613 disc pack requires formatting before any input or output operation can be performed on it. The formatter forms 120 word sectors which are grouped 48 per track. The program also performs a bad-track analysis.

The formatter (format F p/n 92A0205-030) operates under the MAINTAIN III executive. For instructions on loading from magnetic tape, cards or paper tape, see the MAINTAIN III Manual (98A9952-07x). Execution begins at location 500. Some parameters are requested from the keyboard. Inappropriate responses cause the request to be repeated. All inputs are terminated by periods.

INPUT BIC NUMBER

Enter an even value in the range octal 020 through 076.

INPUT DEVICE ADDRESS

Enter a value in the range octal 014 through 017.

INPUT UNIT

Enter a value in the range 0 through 7. This must be the physical unit number calculated as follows:

$UUP_{(2)}$

where

UU is unit number 0-3
P is platter 0 fixed
platter 1 removable
(Note: System RMD is always
000 regardless of which
platter.

INPUT KNOWN BAD TRACKS

Enter octal track numbers in the range 0 through 0625 separated by commas and terminated by a period. If there are no known bad tracks, input only a period.

In addition, the formatting program performs bad-track analysis and creates and maintains a bad-track table.

which is entered on each disc pack at the completion of its formatting. The bad-track table is located on sector 0 of the first track. The table is 26 words long, starting at word 64 of sector 0. The first 64 words of sector 0 reserve the necessary space for the PST. The remaining unused words of sector 0 are filled with zeros. Each disc I/O error will generate a five event retry sequence which, upon failure, will set the corresponding bit in the bad-track table. No alternate tracks are assigned.

If the first track is determined to be bad, the bad-track table may not be placed there. The program prints the error message,

FIRST TRACK BAD

and aborts formatting the current disc pack. The program returns to the keyboard interrogation routine. After the bad-track table has been written on the disc pack, the formatting program resumes the keyboard interrogation to obtain parameters for formatting the next disc. In this way, more than one disc pack can be formatted in the same session. The formatting program may be terminated at this point when no disc packs (except those with bad first tracks) remain unformatted. Formatting disc packs is not necessary before every VORTEX system generation.

18.7 70-7520/7530 DISC PACK FORMATTING PROGRAM

Each 70-7520/7530 disc pack requires formatting before any input or output operation can be performed on it. The formatters form 120 word sectors which are grouped 48 per track. The program also performs a bad track analysis.

The formatter operates under the MAINTAIN III executive. For instructions on loading from magnetic tape, cards, or paper tape, see the MAINTAIN III Manual (98 A 9952-07x). Execution begins at location 500. Some parameters are requested from the keyboard. Inappropriate responses cause the request to be repeated. All inputs are terminated by periods.

INPUT BTC NUMBER

Enter an even value in the range octal 020 through 076.

INPUT DEVICE ADDRESS

Enter a value in the range octal 014 through 017.

INPUT UNIT

Enter a value in the range 0 through 3.

INPUT KNOWN BAD TRACKS

Enter octal track numbers in the range 0 through 017667 separated by commas and terminated by a period. If there are no known bad tracks, input just a period.

In addition, the formatting program performs bad track analysis and creates and maintains a bad track table, which is entered on each disc pack at the completion of its formatting. The bad track table is located at sector 0 of the first track. The table is 508 words long, starting at word 64 of sector 0. The first 64 words of sector 0 reserve the necessary space for the PST. Each disc I/O error will generate a three event retry sequence, which upon failure will set the corresponding bit in the bad-track table. No alternate tracks are assigned.

To bypass a full bad-track analysis, set SSW1 on. This option should not be used on new packs which require a full verification.

If the first track is determined to be bad, the bad-track table may not be placed there. The program prints the error message

FIRST TRACK BAD

and aborts formatting the current disc pack. The program returns to the key-board interrogation routine. After the bad-track table has been written on the disc pack, the formatting program resumes the keyboard interrogation to obtain parameters for formatting the next disc. In this way, more than one disc pack can be formatted in the same session. The formatting program may be terminated at this point when no disc packs (except those with bad first tracks) remain unformatted. Formatting disc packs is not necessary before every VORTEX system generation.

18.8 WRITABLE CONTROL STORE (WCS)

The writable control store must be loaded with the appropriate firmware. The WCS is loaded by the V73 WCS Microprogram Utility (MIUTIL). MIUTIL is a foreground program scheduled by a request:

;SCHED,MIUTIL,p,FL,F

where p is the priority level. Use of the MIUTIL program is described in detail in the Microprogramming Guide.

If the optional V70 series Floating Point Firmware is to be used, it must be loaded into page 1 of WCS. The WCS microprogram is catalogued into the OM library under the name WCSFP, and must be transferred to the BI device for loading by MIUTIL. The WCS should be initialized through the use of MIUTIL prior to loading the floating-point microprograms.

Section 20 gives additional information about writable control store.



SECTION 19 PROCESS INPUT/OUTPUT

19.1 INTRODUCTION

To support process input/output, VORTEX uses special interface drivers and FORTRAN subroutines. This section describes the F2963 Data Acquisition and Control System (19.2), ISA FORTRAN Process Control Subroutines (19.3), and the IEEE Standard 488-1975 Driver (19.4).

19.2 F2963 DATA ACQUISITION AND CONTROL SYSTEM

This section contains a description of the VORTEX I/O driver for the Model F2963 Data Acquisition and Control System (DA/CS). All I/O operations to the DA/CS are performed by the I/O driver, thus, eliminating the necessity for the user to be concerned with the details of performing input and output from the DA/CS.

19.2.1 I/O Macros

I/O requests in assembly language programs are in the form of macro calls. The DASMR assembler provides a standard set of system macros which specify I/O operations. The macros provide uniformity and ease of coding I/O request. The following macros are available:

READ
WRITE

19.2.2 Required Hardware

Minimum hardware required for a Data Acquisition and Control System is one master chassis and controller, Sperry Univac model number F2963-xx (consisting of one Analogic AN5400 chassis and one MCO controller), plus applicable user selected printed circuit boards (PCBs) necessary for the user's unique application.

Figure 19-1 shows a typical system configuration for the DA/CS.

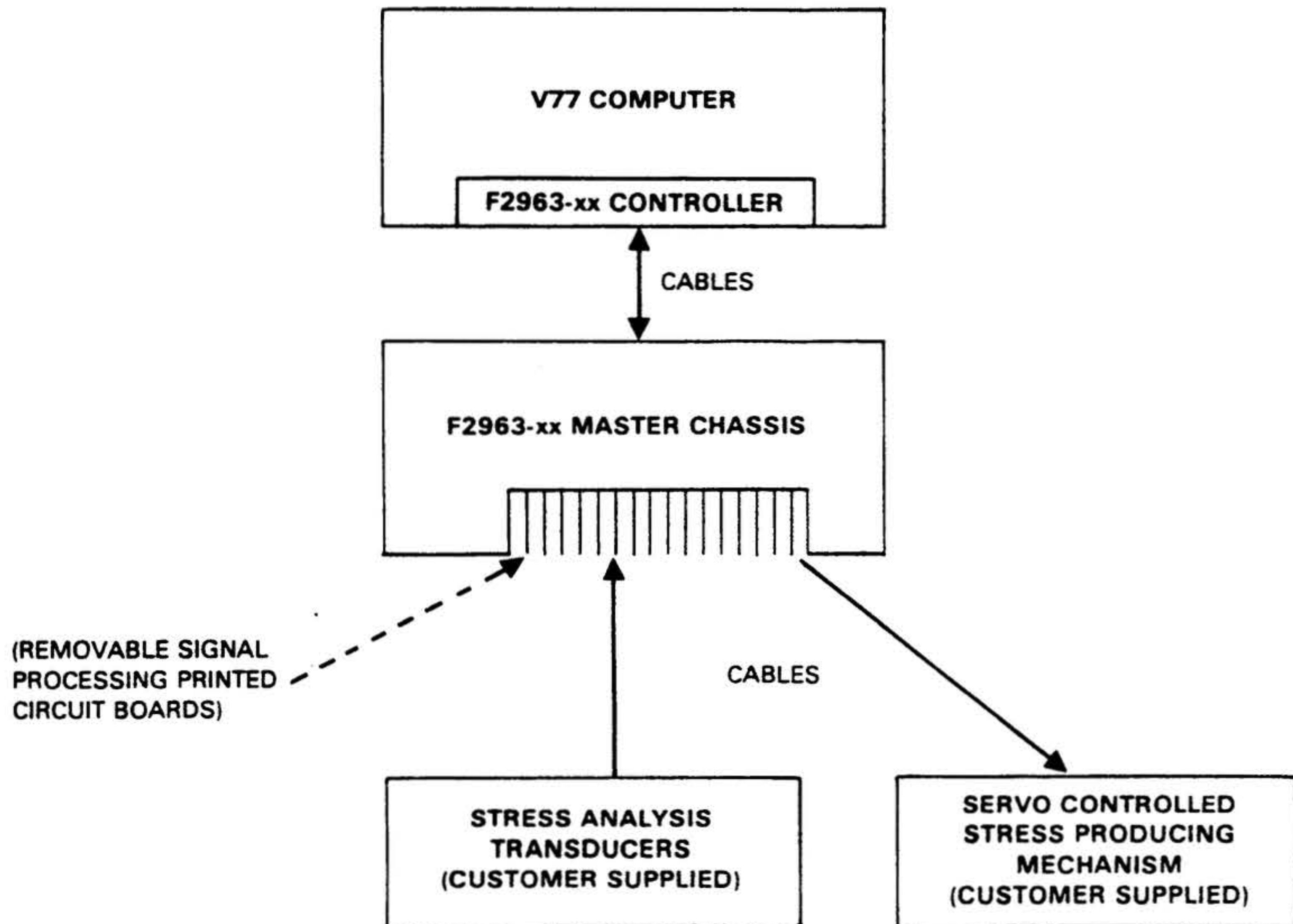


Figure 19-1. Typical System Configuration

PROCESS INPUT/OUTPUT

19.2.3 Hardware Description

The F2963 Data Acquisition and Control System is a general purpose, computer oriented, analog-to-digital (A/D)/digital-to-analog (D/A) data conversion system. Offering field add-on modular flexibility, and expandability, this system is designed for commercial, scientific, and industrial process monitoring and control and data-acquisition applications. The self-powered and self-contained F2963 can process and convert both multi-channel analog input signals and digital input and output signals within a single chassis. This feature, combined with a variety of plug-in signal-translation PCBs, offers the user freedom to configure inputs and outputs in any order or manner to meet his particular requirements.

Based on a single master chassis that holds up to 16 user-selected multiplexer/signal conditioning input PCBs, the F2963 system is expanded by adding up to seven expander chassis for a total of eight chassis. Consequently, up to 128 (16 x 8) user-selected PCBs can accommodate up to 4096 high-level single-ended, or 2048 high-level differential, multiplexer data channels. In any single chassis, differential multiplexing can be provided.

Standard multiplexer PCBs provide a variety of input formats, signal levels, isolation, transducer and thermocouple accommodations, and, when needed, cold-junction compensation. A single chassis can also contain up to 64 simultaneous sample and holds and up to 64 D/A converter outputs. Standard D/A converters provide 10- to 16-bit resolution, and various speeds, accuracies, and stabilities. Standard analog-to-digital converters provide from 8 to 16 bit resolution, moderate to very high speeds, moderate to very high stabilities and bipolar or unipolar input voltage ranges. There are also standard digital input/output PCBs to choose from.

19.2.4 Software Addressing

19.2.4.1 Device Address

Four device addresses are available for Sperry Univac DA/CS controllers. They are 050, 051, 052, and 053 (octal). An individual call to the VORTEX driver can read or write data from one controller only.

19.2.4.2 Unit Address

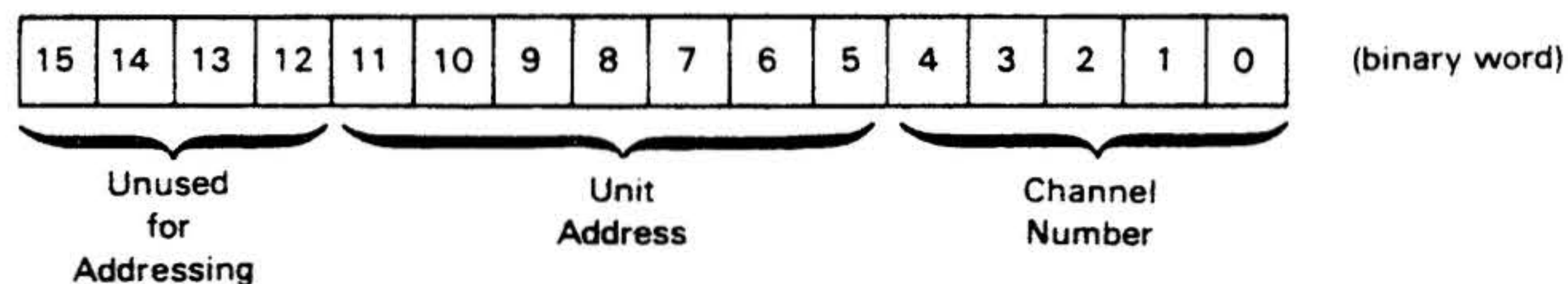
There are 16 slots for individual signal processing printed circuit boards on each chassis. One F2963-xx master chassis and one to seven F2963-03 expansion chassis can be connected to a Sperry Univac controller at an individual device address. Each individual slot within each chassis is referred to as a unit. Therefore, a controller with one chassis connected to it could have as valid unit addresses 0 through 15 (0 through 017). A controller with 8 chassis connected to it would have valid unit addresses 0 through 127 (0 through 0177).

19.2.4.3 Channel Address

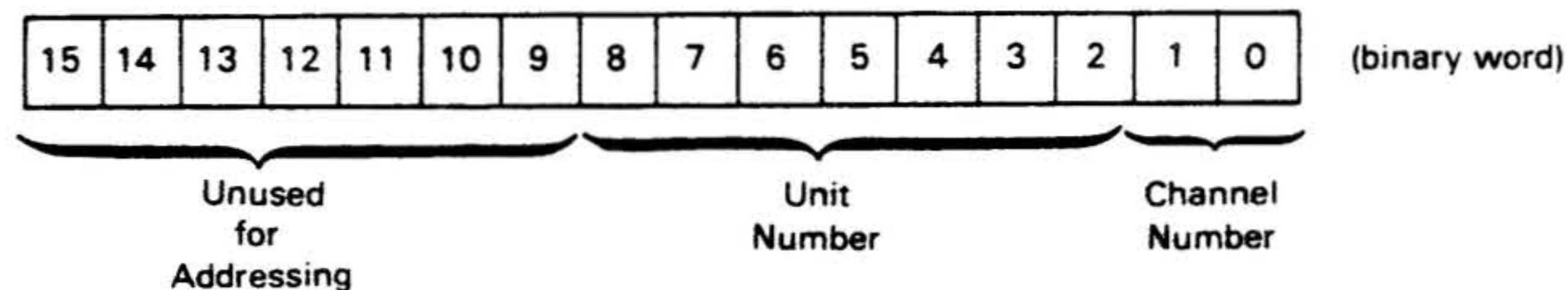
Each unit (individual signal processing printed circuit board) has 1, 2, 4, 16 or 32 individual sample points, depending upon the function of the individual unit. Each sample point is called a channel. A 4 sample point unit will have valid channel addresses 0, 010, 020, and 030. A 16 sample point unit will have valid channel addresses 0, 02, 04, 06, 010, 012, 014, 016, 020, 022, 024, 026, 030, 032, 034, and 036.

19.2.4.4 Addressing Format

The format for addressing input units is



The format for addressing output units is



Note: Only 2 bits are required for specifying output channel numbers, since output units can have a maximum of four channels per unit.

19.2.5 Input

19.2.5.1 READ Macro

Input to the DA/CS is by use of the IOC 'READ' macro. The macro call has the format

READ dcb,lun,wait,mode

where

- dcb is the name of data control block (DCB)
- lun is the logical unit number
- wait is the wait flag
- mode is the data mode (ignored)

Data is always input directly, without modification, so the data mode is effectively system binary.

19.2.5.2 Data Control Block

The READ macro Data control block (DCB) format is:

Input Buffer Count	Word 0
Input Buffer Address	Word 1
Opcode	Word 2
Status Word Address	Word 3
Unit/Channel List Address	Word 4
Trace Area Begin Address	Word 5
Trace Area End Address	Word 6

Opcode (Word 2)

0	0	0	0	0	0	0	0	0	0	0	0	0	V	W	X	Y	Z
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

- Z = 1 Digital input data
- Z = 0 Analog input data
- Y = 1 Sequential Addresses
- Y = 0 Random Addresses
- X = 1 Place amplifiers in hold mode before input.
- X = 0 This is not a sample and hold operation.

- W = 1 Only one input address specified. Repeat same address until input buffer is full.
- W = 0 Number of addresses specified is identical to number of words in input buffer.
- V = 0 Reserved for special application. Must be zero.

Analog/Digital - If input data is digital, all other bits in the opcode should be set to 0.

Sample and Hold - If X=1 in the opcode, then the amplifiers in all sample and hold type units will be simultaneously placed in the 'hold' state. After this is performed, all channels listed in the unit/channel address list (word 4) will be input. When input is complete, all units will be reset to 'sample' state.

Random Addresses - In this mode, the unit/channel address list may specify one or several addresses. Upon completion of the sampling of the first address, the next address in the list will be sampled. The operation terminates when the count specified in the input buffer count (Word 0) is reached.

Sequential Addresses - In this mode, the unit/channel address list specifies only one address. After inputting data from the specified channel, the address is incremented to the address of the next input channel in sequence. This is normally valid only for analog input cards with 32 channels per card. The operation terminates when the count specified in word 0 is reached.

Input Buffer Count (Word 0) - This is the octal number of words in the input buffer. This must not be less than the number of words contained in the unit/channel address list (Word 4) or a loss of data will occur.

Input Buffer Address (Word 1) - This is the address of the input buffer area.

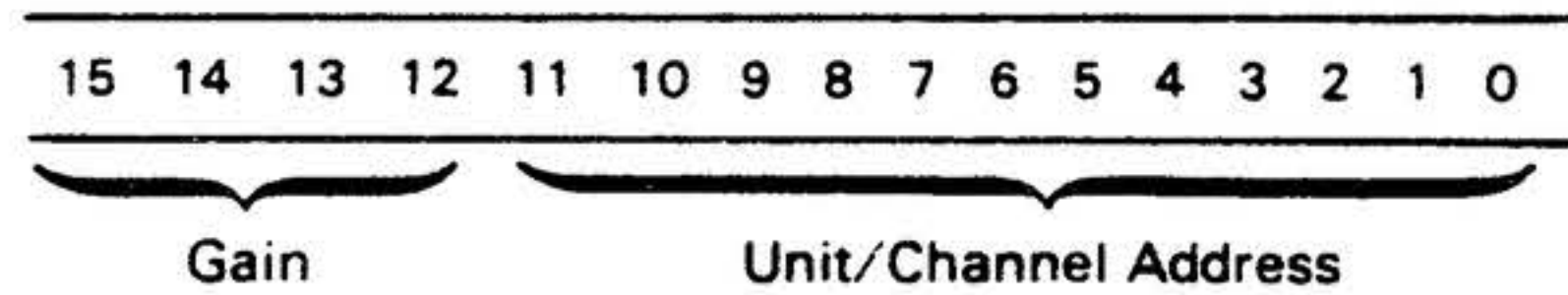
Status Word Address (Word 3) - The status word is a word in the calling program in which status of the IOC call is maintained. This parameter contains the address of that word. Status is an octal number returned in the Data Control Block. Its contents give the status of the call as follows:

- 1 I/O correctly completed
- 2 I/O in execution

PROCESS INPUT/OUTPUT

- 3 Cable between V70 and DA/CS is not connected (hardware malfunction)
- 4 DA/CS is in local mode. Put in remote
- 5 "DONE" line from control is in incorrect state (hardware malfunction)
- 6 BIC transfer error (hardware malfunction)
- 7 Unsuccessful attempt to place amplifiers in hold state (hardware malfunction)

Unit/Channel List Address (Word 4) - This is the address of a list which contains the gain setting and the unit and channel numbers to be input. The format is as follows:



Gain Field Portion of Unit/Channel List

GAIN =	000	Unity Gain	}	Single-Ended Amplifier
	0001	Gain of 2		
	0010	Gain of 4		
	0011	Gain of 8		
	1000	Unity Gain	}	Differential Amplifier
	1001	Gain of 2		
	1010	Gain of 4		
	1011	Gain of 8		

Any values not specified are invalid. For a signal processor card without programmable gain (i.e., F2963-07), the gain field is used only to determine whether the amplifier is single-ended or differential.

Unit/Channel Address Portion of Unit/Channel List - This is as specified in section 26.3.

TRACE Area Begin Address (Word 5) - This is the address of the beginning of a memory area in which a trace of all I/O instructions to and from the DA/CS can be stored. If tracing is not desired, then this field must be zero. Since tracing is performed only during debugging, this field will normally be zero.

TRACE Area end Address (Word 6) - This is the end address of the trace area. If more I/O operations are traced than can be contained within this area, tracing will "wrap around" and begin at the address specified in "Trace Area Begin Address". If tracing is not desired, this field must be zero.

19.2.5.3 Programming Examples

Example 1: A DASMR program is to sample single-ended analog input channels 0 through 31 on unit number 7 in sequential mode. Do not return until I/O complete. Do not set gain. Logical unit number is 22.

```

READ          DCB1,22,00,0
.
.
DCB1          DATA      32          (length of 32)
              DATA      INBUF      (input buffer is "INBUF")
              DATA      02          (OP code is sequential)
              DATA      STAT        (status is at address "STAT")
              DATA      CHLIST      (channel number list is at address
              "CHLIST")
              DATA      BEGTR       (TRACE area begins at address
              "BEGTR")
              DATA      ENDTR       (TRACE area ends at address "ENDTR")
    
```


Example 1:

A DAS MR program is to sample an input channel 100 times at a rate of 1 input/50 microsecond . The channel is number 5 on device address 062, which is assigned to logical unit number 22, and the data is to be input into buffer IBUF. Do not return till I/O complete.

```

      .
      .
      .
      READ      PCB1, 22, 0, 0
      .
      .
PCB1      DATA      100
          DATA      IBUF
          DATA      CHNO
          DATA      0
          DATA      1
          DATA      50
      .
      .
CHNO      DATA      02005
    
```

Example 2: (see section 19.4)

A FORTRAN program is to input sequentially from channels 04001, 04002, and 04003, which are assigned to logical unit number 35, storing the input values into IBUF. Do not return till I/O complete. Set the input rate to 1 word/20 microsecond. The device address to which the input module is assigned is seen to be 064 (52 in decimal digits, and the decimal equivalent of 04000 is 2048).

```

      .
      .
      INTEGER STAT, PTLIST
      DIMENSION IBUF(3)
      DATA PTLIST/2049/
      .
      .
      CALL V$OPIO (52, 35, 20, STAT)
      .
      .
      CALL AISQW(3, PTLIST, IBUF, STAT)
      .
      .
    
```

19.3.4 Low-Level Multiplexor Gain Control

Control of the low-level multiplexor amplifier gains is accomplished through the use of the IOC FUNC macro. FORTRAN source programs can set amplifier gains by calling one of the subroutines described in section 19.4.1, which will construct and execute such a macro.

The macro call has the general form (see section 3.5.8).

FUNC	dcb,lun,wait
where:	
dcb	the address of the data control block.
lun	the number of the logical unit (ADCM) being manipulated.
wait	unused.

The DCB macro has the general form

DCB	rl, buff, fun
where:	
rl	is the number of channels for which the gain will be set.
buff	address of the channel table.
fun	is the function code.

- 0 = Set gains on sequential channels to a fixed value, delay 5 milliseconds.
- 1 = Set gains on random channels through a table, delay 5 milliseconds.
- 2 = Set gains on sequential channels to a fixed value, immediate return.
- 3 = Set gains on random channels through a table, immediate return.

The format of the channel table when fun = 0 or 2 is:

STARTING CHANNEL ADDRESS	Word 0
GAIN OF CHANNELS	Word 1

The format of the channel tables when fun = 1 or 3 is:

- Word
- 0 = ADDRESS OF CANNEL a
 - 1 = GAIN CODE FOR CHANNEL a
 - 2 = ADDRESS OF CHANNEL b
 - 3 = GAIN CODE FOR CHANNEL b
 - 4 = ADDRESS OF CHANNEL c
 - etc.

PROCESS INPUT/OUTPUT

19.2.6 Output

19.2.6.1 WRITE Macro

Output to an AN5400 is by use of the IOC 'WRITE' macro. The macro call has the format:

WRITE dcb,lun,wait,mode

where

- dcb is the name of Data Control Block (DCB)
- lun is the logical unit number
- wait is the wait flag
- mode is the data mode (ignored)

Data is always output directly, without modification, so the Data Mode is effectively System Binary.

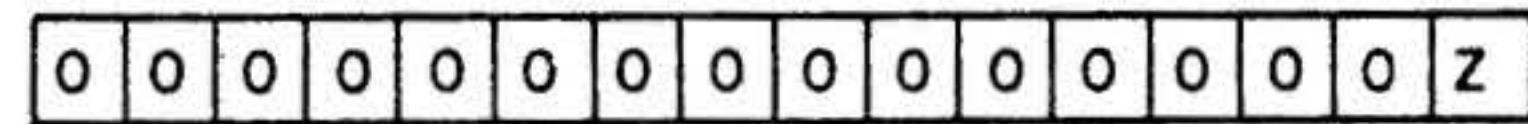
Data Control Block

Output Buffer Count	Word 0
Output Buffer Address	Word 1
Opcode	Word 2
Status Word Address	Word 3
Unit/Channel List Address	Word 4
Trace Area Begin Address	Word 5
Trace Area End Address	Word 6
Mask Word List Address	Word 7
Pulse Width Word	Word 8

Output Word Count (Word 0) - This is the octal number of words of data to be output. This must be the number of words in both the output buffer and the unit/channel list, and, if specified, the Mask Word List.

Output Buffer Address (Word 1) - This is the address of the output data buffer. Data must be in a position in the buffer corresponding to the position of the desired channel in the unit/channel list.

Opcode (Word 2)



- Z = 0 output data is analog
- Z = 1 output data is digital

Status Word Address(Word 3) - The Status Word is a word in the calling program in which the status of the IOC call is maintained. This parameter contains the address of that word. See Paragraph 26.4.2 for format of this word.

Unit/Channel List Address (Word 4) - This is the address of a list that shows to which unit/channel addresses to output the data.

Mask Word List Address (Word 5) - The mask word is used to select specific bits within a digital output word which need to be changed. Bits set to one in the mask words flag bits that are to be updated. The list must contain the same number of entries as the output buffer and the unit/channel list. In order to use this function, all units referenced in the unit/channel list must be able to perform readback. The VORTEX driver will read back the previous setting of the channel, update only those bits selected by the mask word, then output to the channel the previous setting of any bits not selected in the mask word, plus the updated setting of any bits selected in the mask word. If the mask word list address is zero, the entire word in the output buffer is output and latched, with the mask function being ignored.

Pulse Width Word (Word 6) - The pulse width word gives the length of time in VORTEX basic cycles (5 milliseconds) that digital output points are to remain set. At the conclusion of this waiting period, the VORTEX driver will output a word of zeros to every channel in the unit/channel list.

19.2.6.2 Examples

Example 1: A DASMR source program is to output the first 3 words from buffer OBUF to channel 1 on unit number 3 and channels 0 and 1, on unit number 2. Data is analog. Logical unit number is 17. Do not wait for completion.

```

WRITE          DCB1,17,0,0
.
.
.
DCB1          DATA      3          (length of 3)
              DATA      OBUF       (output buffer is "OBUF")
              DATA      0          (OP code is analog output)
              DATA      STAT       (status is at address "STAT")
              DATA      CHLIST     (channel number list is at address "CHLIST")
              DATA      0          (do not TRACE)
              DATA      0          (do not TRACE)
.
.
.
CHLIST        DATA      0141      (binary equivalent = 0000000001100001)
              DATA      0100      (binary equivalent = 0000000001000000)
              DATA      0101      (binary equivalent = 0000000001000001)
.
.
.
STAT          BSS        1
.
.
.
OBUF          DATA      01,02,03

```

Example 2: A DASMR source program is to output the first 3 words from buffer OBUF to channels 1 on unit number 3, and channels 0 and 1 on unit number 2, logical unit number is 22. Data is digital. The first word in the output buffer is to update only the low order 2 bits, the next two words will update the entire word. Leave output latched indefinitely. Do not wait for completion. Report Status in word STAT.

```

WRITE          DCB2,22,0,0
.
.
.
DCB2          DATA      3          (length of 3)
              DATA      OBUF       (output buffer is "OBUF")
              DATA      01         (OP code is digital output)
              DATA      STAT       (status is at address "STAT")
              DATA      CHLIST     (channel number list is at address "CHLIST")
              DATA      BEGTR      (begin TRACE at address "BEGTR")
              DATA      ENDTR      (end TRACE at address "ENDTR")
              DATA      MASK       (mask is at address "mask")
              DATA      0          (leave output latched indefinitely)

```

PROCESS INPUT/OUTPUT

CHLIST	DATA	0141
	DATA	0100
	DATA	0101
MASK	DATA	03
	DATA	0177777
	DATA	0177777
STAT	BSS	0
OBUF	DATA	01,02,03
BEGTR	BSS	500
ENDTR	BSS	0

19.2.7 Throughput Considerations

19.2.7.1 Sequential Input

If one range (gain) is applicable to all channels to be sampled and they are in sequential order in contiguous

addresses (or the same channel is to be sampled repeatedly), the F2963-xx will be able to automatically sequence through the channels. The VORTEX driver will setup the operation, start the BIC, and will not be interrupted until all points have been sampled. The following table represents throughput rates for the different signal processors and A/D converters.

A/D THROUGHPUT MATRIX FOR SEQUENTIAL INPUT

A/D Converter Module	With F2963-07 (AC262) Signal Processor Module	With 2963-06 (AC260) Signal Processor Module
F2963-09 (AC2710)	50,000	50,000
F2963-10 (AC2712)	50,000	50,000
F2963-11 (AC2713)	50,000	27,000
F2963-12 (AC2714)	50,000	27,000
F2963-13 (AC8016)	50,000	15,000

Note: Figures represent number of Analog to Digital conversion per second. All timings were performed on V77/600 with 660 ns. semiconductor memory with no cache.

19.2.7.2 Random Input

Throughput for any operation which samples more than one channel and requires the range (gain) to be set and/or requires addresses to be specified in any given order will require VORTEX I/O driver setup and handling for each

individual channel. The BIC will not be used. The I/O driver will operate in a programmed I/O mode. Timings may vary between adjacent channels and these figures are maximum under ideal conditions (no other tasks running concurrently).

A/D THROUGHPUT MATRIX FOR RANDOM INPUT

A/D Converter Module	With F2963-07 (AC262) Signal Processor Module	With F2963-06 (AC260) Signal Processor Module
F2963-09 (AC2710)	6,000	6,000
F2963-10 (AC2712)	6,000	6,000
F2963-11 (AC2713)	6,000	6,000
F2963-12 (AC2714)	6,000	6,000
F2963-13 (AC8016)	6,000	6,000

Note: Figures represent number of Analog to Digital conversions per second. All timings were performed on a V77/600 with 660 ns. semiconductor memory with no cache.

19.3 FORTRAN PROCESS CONTROL SUBROUTINES

The Instrument Society of America (ISA) has defined as standards a number of FORTRAN subprogram calls useful in process input/output applications. VORTEX includes the following subroutines modeled after the ISA standards.

call	name
AISQ(W)(H)	Analog Input Sequential
AIRD(W)(H)	Analog Input Random
AIRP(W)(H)	Analog Input Repeat
AO(W)	Analog Output
DI(W)	Digital Input
DOM(W)	Digital Output-Momentary
DOL(W)	Digital Output-Latching

The (W) option for each of these subroutines selects a wait mode; that is, it specifies that the return is not to be made from the subroutine until the I/O is finished, either normally or by an error condition.

The (H) option for some subroutines selects a sample and hold mode; that is, it specifies that the amplifiers in all sample and hold type units are simultaneously placed in a hold state. Then all channels listed in the unit/channel address list are input. When input is complete, all units are reset to the sample state.

The function performed by each subprogram is described in 19.3.2.

19.3.1 PROCESS I/O PARAMETERS

The following list indicates the parameters used with process I/O FORTRAN subroutine calls. Not all parameters are used with all calls.

parameter	description
lun	is the logical unit number of the DA/CS system.
count	is the number of words in the input buffer 'ibuf' or the output buffer 'obuf'.

PROCESS INPUT/OUTPUT

↓

ptlist is the point list. For AISQ and AIRP calls (with or without the W and H options), this is a single point address. For all other calls, 'ptlist' is an array containing 'count' point addresses. For the output calls AO, DOM, and DOL (with or without the W option), each point address is a word with the unit number in bits 2-8 and the channel number in bits 0-1. For AIRD and DI calls (with or without options), each point address is a word with a channel number in bits 0-4, a unit address in bits 5-11, and a gain value in bits 12-15.

ibuf is an array of (at least) size 'count' into which data is read.

obuf is an array of (at least) size 'count' out of which data is written.

time is a number indicating a time interval. This parameter is used only with the DOM and DOMW calls. If 'time' is not equal to zero, then 5*'time' milliseconds after output a word of zeros is output to every channel indicated in 'ptlist'; that is, the random digit channels are reset.

mask is a mask word used only with calls DOL and DOLW. Bits set to 1 in 'mask' indicate those bits that the device driver program changes in the most recently output word from 'obuf'.

stat is a status word whose value indicates the status of the process I/O call:

value	status
1	I/O correctly completed
2	I/O in execution
3-7	Error condition (see 19.2.5.2)

19.3.2 PROCESS I/O Calls

The formats for process I/O FORTRAN subroutine calls are presented below. Following each instruction format is a brief description of the functions performed by the subprogram.

CALL $\left\{ \begin{array}{l} \text{AISQ} \\ \text{AISQW} \\ \text{AISQH} \\ \text{AISQWH} \end{array} \right\}$ (lun, count, ptlist, ibuf, stat)

↑

↓

This subprogram reads 'count' analog inputs into the buffer 'ibuf'. The first channel read is channel 0x001, where 'ptlist' contains the word 0xyyy. Channels are then read sequentially.

CALL $\left\{ \begin{array}{l} \text{AIRD} \\ \text{AIRDW} \\ \text{AIRDH} \\ \text{AIRDWH} \end{array} \right\}$ (lun, count, ptlist, ibuf, stat)

This subprogram reads 'count' analog inputs into the buffer 'ibuf'. The inputs are taken from the 'ptlist' list of random points.

CALL $\left\{ \begin{array}{l} \text{AIRP} \\ \text{AIRPW} \\ \text{AIRPH} \\ \text{AIRPWH} \end{array} \right\}$ (lun, count, ptlist, ibuf, stat)

This subprogram reads 'count' analog inputs into the buffer 'ibuf'. All inputs are taken from the single channel given in 'ptlist'.

CALL $\left\{ \begin{array}{l} \text{AO} \\ \text{AOW} \end{array} \right\}$ (lun, count, ptlist, obuf, stat)

This subprogram outputs 'count' analog values from the buffer 'obuf'. These values are output to the list of random points 'ptlist'.

CALL $\left\{ \begin{array}{l} \text{DI} \\ \text{DIW} \end{array} \right\}$ (lun, count, ptlist, ibuf, stat)

This subprogram reads 'count' words of digital input into the buffer 'ibuf'. These words are input from the list of random digital channels 'ptlist'.

CALL $\left\{ \begin{array}{l} \text{DOM} \\ \text{DOMW} \end{array} \right\}$ (lun, count, ptlist, ibuf, time, stat)

This subprogram outputs 'count' words of digital output from the buffer 'obuf'. These words are output according to the list of random digital channels 'ptlist'. If 'time' is zero this completes the operation; otherwise, after 5*'time' in milliseconds a word of zeros is output to every channel in 'ptlist', thus resetting all channels.

CALL $\left\{ \begin{array}{l} \text{DOL} \\ \text{DOLW} \end{array} \right\}$ (lun, count, ptlist, obuf, mask, stat)

This subprogram outputs 'count' words of digital output from the buffer 'obuf'. These words are output according to the list of random digital channels 'ptlist'. The device driver program saves the previous word output to each channel and changes only those bits specified by 1-bits in 'mask'. The 'mask' is an integer array parallel to 'obuf' and 'ptlist'.

↑

19.4 IEEE STD 488-1975 DRIVER

19.4.1 Description

The IEEE STD 488-1975 driver is designed to provide a means for managing information interchange between a V77 minicomputer and devices connected to the IEEE Standard Bus within a VORTEX environment. This will enable various programmable instruments to transfer digital data among themselves directly, and not necessitate intermediate processing by an intervening controller unit. The user directs the information management through the execution of various VORTEX I/O macros and a series of general interface management commands.

19.4.2 User Interface

19.4.2.1 General Capabilities

The IEEE - 488 VORTEX driver uses a BIC to pass a buffer to/from the interface bus. This buffer contains an ASCII string which represents command instructions as well as data input for programming a device. These instructions and data enable a device to be addressed then listen and transmit the buffer to the device under control; address a device to talk and transmit its measurement reading to the buffer; or command a device or devices to perform a certain function or to assume a specific state.

19.4.2.2 Program Interface

The IEEE - 488 interface driver is initiated by the user through the execution of VORTEX I/O control macros READ and WRITE. With these two macros, the user is able to send data to a device or receive measurement readings from a device. Also available to the user, is the FUNC macro which provides the ability to execute universal commands of the IEEE controller.

READ Macro:

This macro is used to receive data from a bus instrument. This involves addressing a device to talk, receiving the instrument's data, and un-addressing the device after receiving all its data. Only one device may be addressed to talk at any given time. The form of the I/O calling sequence is:

READ DCB, LUN, WAIT, MODE

where

DCB	is the address of the data control block (see section 2.2.1).
LUN	is the number of the logical unit to which to IEEE controller has been assigned.
WAIT	specifies the type of return, and should be coded 0 for a return suspended until I/O is complete.
MODE	specifies the I/O data mode and should always be coded as 1 for ASCII.

WRITE Macro

This macro will transmit data from a user buffer area to a bus instrument. This includes addressing the device to listen, sending data or programming instructions, and un-addressing the device after it has accepted all data. More than one device may be addressed to listen at any given time. The form of the I/O calling sequence is:

WRITE DCB, LUN, WAIT, MODE

where the parameters have the same definitions and take on the same values as in the READ macro.

FUNC Macro

This macro allows the user to perform various universal command functions of the IEEE - 488 controller. The form of the FUNC macro is:

FUNC DCB, LUN, WAIT

Where the parameters have the same definitions and take on the same values as in the READ macro.

The particular action which is initiated by the FUNC call is dependent on the function code FUN in the data control block. Possible actions are:

PROCESS INPUT/OUTPUT

Function Code	Universal Command	Operational Function
0	DCL	Device Clear. Initializes all devices to a pre-defined state. The particular state is device dependent and should be described in the operating manual.
1	LLO	Local Lockout. Disables the 'Return to Local' button on all bus instruments. While in remote mode this prevents a return to local mode via local controls.
2	TCT	Take Control. This allows a bus instrument other than the V77 to act as a controller device.
3	SPE	Serial Poll Enable. This allows a serial poll to be taken to determine the source and nature of a pending service request.

NOTE:

Function 2 is in an addressed command which requires a device be addressed to talk.

Data Control Block

The Data Control Block is required for all three macro requests READ, WRITE and FUNC. The general form of this macro is:

DCB RL, BUFF, FUN

where

- RL is the record length, in words, of the record to be transmitted or received.
- BUFF is the address of a buffer containing user parameters and a pointer to the data buffer to be transmitted (or in the case of a READ, the buffer into which data will be received).
- FUN is the function code of the FUNC request, and is unused for other requests.

DCB Buffer Description for READ

The BUFF parameter of the READ DCB contains a pointer address to a buffer of user data. The structure of that user data buffer is:

Word 1 Mask Talk Address
 Word 2 Pointer to Buffer to Receive Data

Word 1
 The lower byte of the first parameter word provides the talk address of a selected device.

The upper byte is an eight bit mask which is applied to received data before it is stored in the user's buffer. A "1" in any mask position will cause the corresponding bit of a received data byte to be set to "1" before being sent to the user. The primary purpose of this mask is to force a set parity for the ASCII data received.

Word 2
 The second parameter word is an address pointer to the user buffer which will receive the data returned from the talk device.

DCB Buffer Description for WRITE

The BUFF parameter of the WRITE DCB contains a pointer address to a buffer of user data. The contents of that user buffer may be a number of listen addresses, programming commands, universal or addressed commands, or device dependent data. A great deal of flexibility is provided for the

user to set up a buffer designed for his own specific purposes. The contents of the buffer will control the transmission of data to listening devices and also handle certain system status lines (i.e., Remote Enable).

The universal commands issued via a FUNC request may be placed into the WRITE buffer if desired. There is no distinction between a FUNC O Device Clear and placing the code for a Universal Device Clear in the WRITE buffer. Their inclusion as FUNC requests are for user convenience only. The only exception is Function Code 3 which performs an entire serial polling routine to determine the source and nature of a service request. A Serial Poll Enable in the WRITE buffer will set the system in a mode ready to do a serial poll, but will not perform the actual polling sequence.

DCB Buffer Description for FUNC

The BUFF parameter of the data control block for a FUNC macro contains an address pointer to a buffer location of user data. This buffer is only used for Function Codes 2 and 3. The structure of the user data area is:

Word	# of Devices	Talk Address 1
Word 2		Talk Address 2
Word 3		Talk Address 3
.	.	.
.	.	.
.	.	.
Word n		Talk Address n
Word n + 1	Status Byte	Talk Address of SRQ

Word 1
The lower byte of the first parameter word for Function Code 3 is the TALK address of a device to be polled for a service request. For Function Code 2, it is the TALK address of the device which is to take control.

The upper byte of the first parameter word represents the number of devices to be serially polled for a pending service request. The upper byte is only used for Function Code 3.

Word 2 thru n
The lower byte of these words is used only for Function Code 3 and contain the TALK address of all devices which are to be polled for a service request.

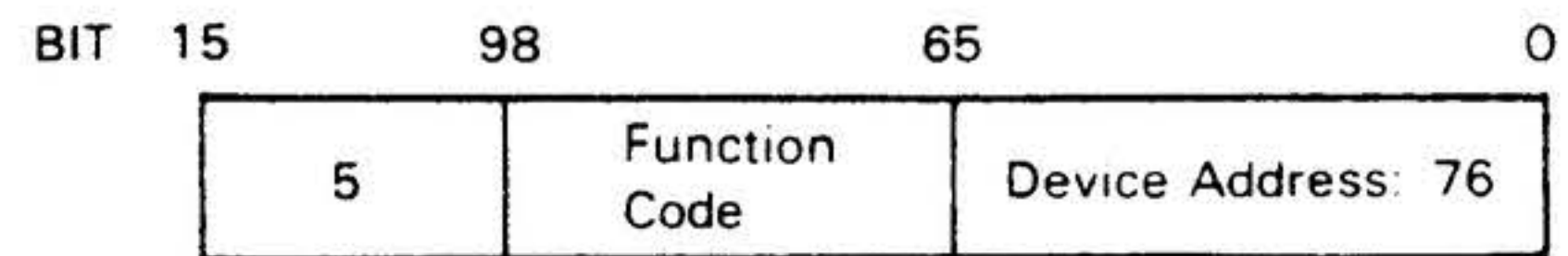
The upper byte for Words 2 thru n are ignored for FUNC requests.

Word n + 1
The IEEE - 48 interface driver will store in the lower byte of this word the TALK address of the device making a service request.

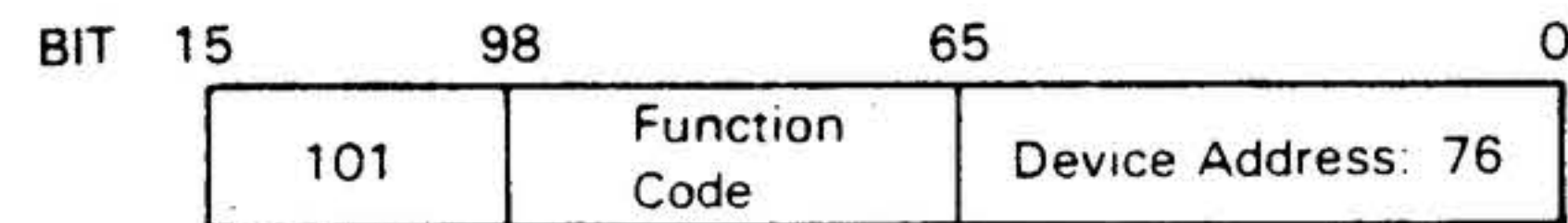
The driver will place the status byte of the device into the upper byte of Word n + 1. This word is only used for a FUNC 3 request.

19.4.3 Data and Instruction Formats

The user has the capability to incorporate into his WRITE buffer a series of programming commands (EXC, EXC2). In order to accommodate the use of a BIC for the transference of data through the bus, it became necessary to change the format of the EXC and EXC2 instructions. The EXC format is:



the EXC2 format is:



When Bits 9 and 11 are set, the word is interpreted as an EXC instruction. Bits 9 and 15 indicate an EXC2 instruction.

Programming commands, listen addresses and other device dependent messages should be placed in the buffer in the lower byte of each buffer word. Measurement readings returned to the application program will be stored in the lower byte of each buffer word.

Programming commands which are available to the user:

Command	Function
EXC 076	Program Identify True
EXC 176	Program Identify False
EXC 276	Program Remote Enable True
EXC 376	Program Remote Enable False
EXC 576	Program Attention False
EXC 676	Program Attention True
EXC 776	Issue an Interface Clear
EXC2 076	Program Take Control Synchronously True

PROCESS INPUT/OUTPUT

19.4.3.1 Identify

The Program Identify True or False is used strictly for a parallel polling operation. Polling occurs when both Attention and Identify are true and stops when either ATN and IDY is False.

19.4.3.2 Remote

Setting Remote Enable true will allow devices to go to the remote state when their LISTEN address is transmitted. Programming this line False places all bus instruments into local state.

19.4.3.3 Interface Clear

Issuing an Interface Clear will initialize the system by placing the system controller in charge and clearing the bus.

19.4.3.4 Take Control Synchronously

There are times when the controller may take control of the bus by a synchronous interruption of an action TALKER. To prevent the loss of data which is currently being transmitted, the Take Control Synchronously (TCS) interface function is executed. This does not have to be set for each transmission performed, but need only be executed once. It is reset by an Interface Clear or System Reset.

19.4.3.5 Attention

Messages which are sent through the BUS may be interpreted in one of two ways depending on the state of the Attention line. by placing an EXC 676 instruction in the WRITE buffer, the line is set true. This places the bus in command mode where all data in the buffer following the EXC command is read as universal or addressed commands.

In a similar manner, the Attention line is programmed false via an EXC 576 instruction. This will put the bus in the data mode in which the following data is interpreted as device dependent messages.

Once the controller has been programmed to the data or command mode, it retains that state until it is reprogrammed to the alternate mode.

19.4.4 System and User Requirements

19.4.4.1 System Generation Considerations

The environment for which the IEEE - 488 Interface Driver has been designed to perform in should have the following characteristics:

- System generated in conformance with the standard procedure for a VORTEX system generation
- The inclusion of the V75 extended instruction set is not necessary
- A dedicated BIC for the IEEE controller

19.4.4.2 User Program Considerations

There are times when an instrument on the IEEE bus will generate a service request interrupt. The driver will note the presence of the service request by placing a byte of all 1's (i.e., 0377) in event word TBEVNT of the TIDB. It is the responsibility of the application program to periodically check TBEVNT to see if a service request is pending, and, if so, to handle it accordingly.* The application program should respond to all pending service requests before issuing an I/O request.

To determine the source and nature of a service request, the user must perform a serial poll. This is done by issuing a FUNC 3 command. The result of this request will be the TALK address of the device with a pending service request and a status byte. Bits 0-6 of this byte will represent the specific nature of the request. The application program must interpret bits 0-6 (interpretation is device dependent) and perform whatever actions are warranted by the request. The event word TBEVNT should be cleared after a serial poll is performed.*

* **Note:** The RTE "TBEVNT" request should be used to examine and clear TBEVNT (Section 2.1.15).

SECTION 20

WRITABLE CONTROL STORE AND FLOATING-POINT PROCESSOR

The **Writable Control Store (WCS)** option extends the SPERRY UNIVAC 70 series processor's read-only control store to permit the addition of new instructions, development of microdiagnostics, and optimal tailoring of the computer system to its application. Unlike the read-only control store, which contains the 70 series standard instruction set and cannot be altered, the WCS can be loaded from main memory under control of certain I/O instructions. The capabilities of WCS give the user more complete access to the resources of the 70 series computer system.

20.1 MICROPROGRAMMING SOFTWARE

Supporting software for the WCS includes the following:

- Microprogram assembler MIDAS
- Microprogram simulator MICSIM microprogram
- Microprogram utility loader and diagnostic MIUTIL
- WCS reload task

All software for microprogram development operates under VORTEX. The capabilities and use of WCS and its supporting software are described in the SPERRY UNIVAC Microprogramming Guide.

20.1.1 Microprogram Assembler

The MIDAS program allows the user to prepare microprograms for SPERRY UNIVAC 70 series WCS, using operation mnemonics, symbolic addressing, address-field calculations, macro definitions, error detection and automatic program documentation.

Under VORTEX, MIDAS is scheduled from the background library at level 0 by

```
/LOAD, MIDAS
```

20.1.2 Microprogram Simulator

The SPERRY UNIVAC microprogram simulator (MICSIM) helps the programmer to verify and optimize microprograms. MICSIM runs the output from MIDAS within the system's main memory. At selected times, conditions and the contents of data locations can be examined and changed. MICSIM is scheduled from the background library at level 0 by

```
/LOAD, MICSIM
```

20.1.3 Microprogram Utility

Loading the control store with the assembled and tested microcode is performed by microprogram utility, MIUTIL.

In addition, on-line debugging directives are available through the utility on a special configuration. The MIUTIL program operates as a foreground program at priority level set by the user. The program is scheduled by operator input over the OC device. For example,

```
; SCHED, MIUTIL, 3, FL, F
```

The microprogram utility is also responsible for maintaining an up-to-date image of the contents of the WCS on an RMD file, named WCSIMG on the OM library, see section 15.8. This image is then used by the WCS reload task, WCSRLD, to restore the WCS following a power failure/restart and VORTEX reload. The RMD file image is updated each time the R directive is used to exit from the utility.

If the update is completed successfully, the message:

```
WCS SAVED
```

is output on the OC and LO devices before the utility exits. If the RMD file for saving the WCS is not present on the OM library the OM library, the system outputs

```
IO10, MIUTIL  
FILE WCSIMG NOT FOUND  
WCS SAVE ABORTED
```

I/O errors which may occur during the save operation result in outputting messages

```
IOxx, MIUTIL  
WCS SAVE ABORTED
```

If the restoration of WCS is completed successfully, the message WCS RELOADED will be output to the OC and LO devices before the reload task exits.

To exit from the microprogram utility without updating the RMD file, the operator may issue the directive.

```
; ABORT, MIUTIL
```

20.1.4 WCS Reload Task, WCSRLD

This task, WCSRLD, reinitializes the WCS to the contents specified by the RMD file image of WCS, WCSIMG on the OM library. It is automatically scheduled on power failure/restart or upon the reloading of the VORTEX system. In this way, WCS contents are preserved through any periods without power.

Though usually scheduled automatically by the system, the reload task may also be scheduled manually by the operator. For example, the following directive schedules the reload task at priority level 15:

```
; SCHED, WCSRLD, 15, FL, F
```

20.2 STANDARD FIRMWARE

Standard firmware is available on the 70 series computers to provide faster and more compact code. The executable code which uses the firmware, or microprograms, is automatically generated by the VORTEX FORTRAN IV compiler when the option F is specified (in the JCP directive /FORT, see section 4.2.15). The firmware also extends the capabilities of the user's assembly language programs and the support library (see section 13).

Standard firmware includes routines which are loaded into the system's WCS for the following categories of operations:

- Arithmetic for two-word fixed-point and integer numbers
- Arithmetic for real (floating-point) numbers
- Transfer of two-word values, such as a memory to memory move
- FORTRAN oriented routines
- Byte manipulation
- Stack manipulation

Executing a branch-to-control-store (BCS) instruction causes a transfer of control from the system's read-only memory to the WCS at the address specified in the BCS instruction. The MIUTIL program (see section 20.1.3) loads the standard firmware as well as any extensions to the instruction set the user may write. To execute firmware, the program must use a BCS instruction with the appropriate entry address and calling sequence for passing parameters.

A FORTRAN IV program specifies the option F on its request for compilation, and then BCS instructions are generated. The FORTRAN IV programs use this firmware without any changes to the FORTRAN IV statements.

Due to size constraints, some firmware is unavailable under certain hardware configurations. Table 20-1 shows these restrictions.

Table 20-1. Firmware Availability

Firmware Routine	Hardware Configurations	
	without FPP	with FPP
XAD,XSB	YES	YES
XMU,XDV	YES	NO
IMU,IDV	NO	YES
FAD,FSB,FMU,FDV	YES	NO
FSQ	NO	YES
FLD,FST,FMV	YES	YES
FSE,FDO,FDO1	YES	YES
FTNE,FTEQ,....,FTGT	NO	YES
FJNE,FJEQ,....,FJGT	NO	YES
FAIF,FIOP	NO	YES
FRSC,FRSR,FJAG	NO	YES
Byte Firmware	YES	YES
Stack Firmware	YES	YES
FIMPY	NO	YES
NOX120	NO	YES

20.2.1 Fixed-Point Arithmetic Firmware

Two-word fixed-point and integer numbers use the following arithmetic firmware:

Mnemonic	Function	BCS Call
XAD	Fixed-point and integer add	0105334
XSB	Fixed-point and integer subtract	0105374
XMU	Fixed-point multiply	0105274
XDV	Fixed-point divide	0105234
→ IMU	Integer multiply	0105227
IDV	Integer divide	0105067

These operations are performed on the hardware A and B registers (AB), using the number specified by the second word of the respective BCS call. If overflow occurs, AB is set to the maximum number with the proper sign and the overflow flag (OVFL) is set.

For two-word fixed-point numbers, the decimal point is assumed to be to the left of bit 15 of the most significant word. For two-word integer numbers, the decimal point is assumed to be to the right of bit 0 of the least significant word. As a result, rounding and overflow conditions are different for multiply and divide. For example, multiplying two double-word numbers produces a logical four-word result. The fixed-point function returns the high order two-words and drops the lower two. The integer multiply returns the lower two-words of the logical result and sets overflow if either of the two higher words are non-zero.

20.2.1.1 Fixed-Point Multiply (FIMPY)

The following firmware is used for single word fixed-point and integer numbers.

FIMPY Fixed-point (one word) multiply

This operation is performed on an operand in the A register and a value located at an address which has been provided as a parameter, and returns its result in the A register. Overflow is not checked for. The parameter address may be indexed by the X register.

The BCS call is

0105015 if no indexing desired,
0105055 if indexing is desired.

20.2.2 Floating-Point Arithmetic Firmware

The addition, subtraction, multiplication, and division of single-precision real, or floating-point, numbers can be performed with the following firmware.

Mnemonic	Function	BCS Call
FAD	Floating-point add	0105134
FSB	Floating-point subtract	0105174
FMU	Floating-point multiply	0105074
FDV	Floating-point divide	0105034
FSQ	Floating-point square root	0105127

A floating-point arithmetic operation is performed on AB using the floating-point number specified by the second word of the BCS call. If underflow occurs, AB is set to zero. If overflow occurs, AB is set to the maximum floating-point number with a proper sign. Taking square root of a negative number results in the overflow being set and AB set to zero.

20.2.3 Data Transfer Firmware

The data transfer firmware routines load AB from memory, store AB in memory, and move the contents of two contiguous memory locations to another place in memory.

Mnemonic	Function	BCS Call
FLD	Load AB with two words from memory	0105032
FST	Store AB into memory	0105033
FMV	Memory-to-memory move of two words	0105037

20.2.4 FORTRAN-Oriented Firmware

These microprograms are oriented toward FORTRAN IV operations. However, they have a similar utility to assembly-language programs.

Mnemonic	Use	BCS Call
FINE	Test for not equal	0105024
FTEQ	Test for equal	0105064
FTLT	Test for less than	0105124
FTGE	Test for greater than or equal	0105164
FTLE	Test for less than or equal	0105324
FTGT	Test for greater than	0105364
FJNE	Jump if not equal	0105026
FJEQ	Jump if equal	0105066
FJLT	Jump if less than	0105126
FJGE	Jump if greater than or equal	0105166
FJLE	Jump if less than or equal	0105326
FJGT	Jump if greater than	0105366
FAIF	Arithmetic IF processor	0105226
FIOP	Indexed operand processor	0105167
FRSC	Reentrant subroutine call	0105025
FRSR	Reentrant subroutine return	0105065
FJAG	Jump if A register greater	0105125
FSE	Pass parameters between subroutines	0105036
FDO	Terminate DO loop	0105035
FDO1	Terminate DO loop (1 increment)	0105027
NOX120	General indexed array processor	varies (see macro)

For FSE, the calling routine would use the following sequence:

CALL	SUB	
DATA	P 1	Address of first data to be moved
.		
.		
DATA	Pn	Address of last data to be moved

In the subroutine being called, the following sequence is necessary to receive the data or data address:

WRITABLE CONTROL STORE AND FLOATING-POINT PROCESSOR

SUB BSS 1
 DATA 0105036 BCS transfer for FSE
 DATA n Number of parameters
 BSS m Number of parameters

The second instruction, FDO to control a DO loop, uses the following calling sequence:

 DATA 0105035 BCS transfer to FDO
 DATA P1 Address of DO
 increment

 DATA P2 Address of DO loop
 counter

 DATA P3 Address of DO loop
 limit

 DATA P4 Address for jump if
 the counter is not
 greater than the
 limit

The third instruction, FDO1 to control a DO loop with increment of 1 uses the following calling sequence.

 DATA 0105027 BCS transfer to FDO1
 DATA P1 Address of DO loop
 counter

 DATA P2 Address of DO loop
 limit

 DATA P3 Address for jump
 if the counter is
 not greater than the
 limit

The DO loop is incremented and tested against the DO loop limit. If the loop counter is less than the limit, execution continues at the address specified by the BCS call word 5. If the value of the loop counter is equal to or greater than the value represented by the limit, execution continues at the instruction following this calling sequence.

The calling sequence for all the relational test (FT--) and jump (FJ--) instructions are as follows:

BCS
 DATA Address of first number
 DATA Address of second number
 DATA Jump address

These routines compare the two single precision floating-point numbers pointed to be the words following the BCS. The A register is set to minus one or zero, depending on the specified relation being met or not met, respectively. For the jump instructions, FJ--, the branch address is taken only when the condition is met, (i.e., when the A register equals minus one). Note that the specified relation is that of the first number to the second. For example, FTGT tests for the first number greater than the second.

The calling sequence for the arithmetic IF processor (FAIF), is as follows:

BCS
 DATA Address of first number
 DATA Address of second number
 DATA Branch address if less than
 DATA Branch address if equal
 DATA Branch address if greater than

This BCS also compares two single precision floating-point numbers. It determines if the first number is less than, equal to, or greater than the second number, and then takes the appropriate branch address.

The indexed operand processor is used to compute the effective address of an element in a FORTRAN real array. It has the following call sequence:

BCS
 DATA Address of index value
 DATA Base address

The effective address is computed by subtracting one from the index value, multiplying the result by two, and then adding in the base address. This allows for an array with two-word entries and induces from one to 'n'. The effective address is stored in the second word of the following instruction.

The reentrant subroutine call, FRSC, has the following call sequence:

BCS
 DATA Subroutine address

The B register points to a memory location which is used as a stack pointer. This memory location is decremented and the resulting value used as the address where the return address is stored.

Control is then transferred to the subroutine. Note that the subroutine address should be that of the first instruction of the subroutine.

The reentrant subroutine return, FRSR, has a calling sequence consisting of just the BCS without parameters. The return address is popped off the stack using the B register and the memory stack pointer as in the subroutine call. Note that no limit checks are made on the stack by either the call or the return. Also, the stack pointer format is not consistent with that of the general stack firmware.

The BCS calling sequence for FJAG (jump if A register greater than zero) is as follows:

BCS
DATA Jump address

The jump address is taken only if the A register is strictly greater than (and not equal to) zero.

The General Indexed Array Processor is used to compute the effective address of an element in a one or two dimensional FORTRAN array, where an array element is one, two, or four words.

The calling sequences are:

For one-dimensional arrays A(I) dimensioned
A(m)--

BCS word
address of index value (I)
address of array base address
(next instruction, word 1)
(next instruction, word 2)

For two-dimensional arrays A(I,J) dimensioned
A(m,n)--

BCS word
address of index value (I)
address at dimension (m)
address of index value (J)
address of array base address
(next instruction, word 1)
(next instruction, word 2)

All operand addresses may be indirect. The array base address must not reflect the offset due to base zero, base one addressing. The actual (direct) address of the desired element of the array will be placed in the second word of the instruction following the BCS call.

The table below gives the BCS word for all combinations of 1, 2, and 4 words; one or two dimensional arrays.

BCS Word Contents (Octal)

	one dimension	two dimensions
one word/element	105216	105016
two words/element	105356	105156
four words/element	105316	105116

The effect address is computed by the algorithm
$$(((J - 1) * m) + m) + (I - 1) * L + B$$

where

I,J,m are as defined above
L is words/element
B is the array base address
(i.e., the address of
element (1.) or (1))

For a one dimensional array, J is assumed to be one in the above formula.

20.2.5 Byte Manipulation Firmware

The byte instructions use a byte pointer address where bits 15-1 specify the word number and bit 0 is 0 for the left byte and 1 for the right byte. The byte-oriented instructions implemented in firmware are:

Mnemonic	Function	BCS Call
CBS	Compare byte strings	0105030
MBS	Move byte string	0105070

In the first microprogram sequence, the CBS instruction requires that the second word contain the address to which control is returned if the strings are not equal. The B register contains the byte starting address of the first string, the X register is the byte starting address of the second string, and the A register specifies the number of bytes to be compared.

The second byte-oriented microprogram sequence, the MBS instruction, moves the number of bytes specified in the A register from the location specified by the B register to the location specified by the X register.

Both share a common BCS entry point, and this may be extended for six more instructions.

20.2.6 Stack Firmware

A stack is kept in memory for use for return addresses, temporary storage or arithmetic operations. The base and limit of the stack (see figure 20-1) are defined by the user. The stack control block is indicated by a pointer in the second word of the calling sequence. Figure 20-2 is the format of the stack control block.

The following BCS instructions correspond with each of the stack operations:

Operation	BCS	Operation	BCS
Add	0105031	Push	0105231
Subtract	0105071	Pop	0105331
Multiply	0105131	Push double	0105271
Divide	0105171	Pop double	0105371

Eight stack instructions transfer to the same initial entry point in the WCS, where the decoder determines the specific instruction to be executed.

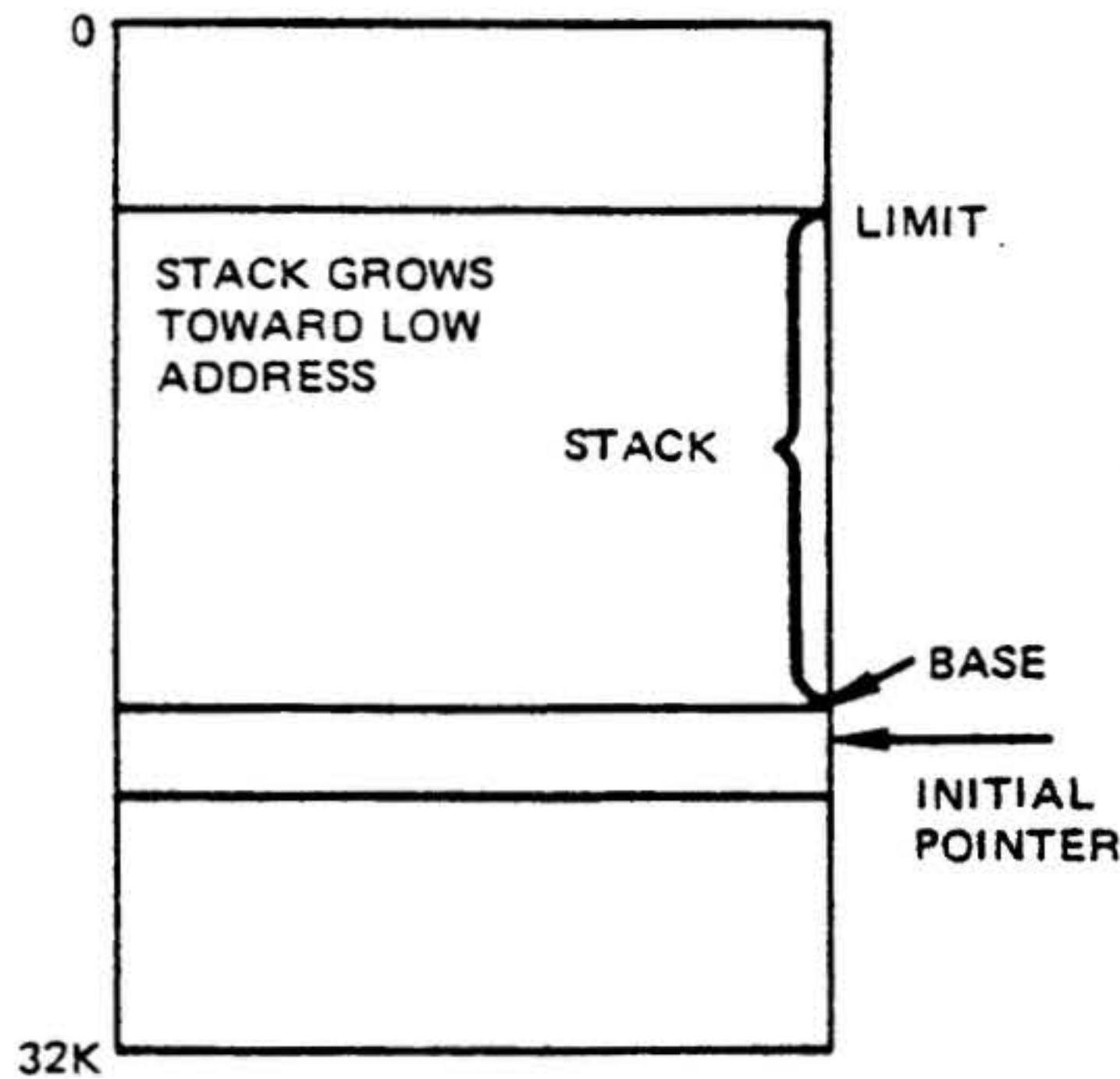


Figure 20-1. Base and Limit of Stack

On all stack operations, if the top-of-stack pointer (PTR) ever exceeds the boundaries of the stack (as the user defined them in the stack control block), no further processing takes place and a JMPM is made to the fourth word in the stack control block.

Single-Precision Integer Stack Arithmetic

Add: adds the top two words of the stack, increments the pointer and replaces the new topmost word. If the result exceeds the maximum positive number (077777), the overflow indicator (OF) and the sign in bit 15 are set to one. For example, adding 000002 to 077777 sets OF to one and the result to 100001.

Subtract: subtracts the next stack word from the top of stack word (by adding the top word to the two's complement of the next stack word), increments the top-of-stack pointer, and stores the remainder in the new top-of-stack word. If the result exceeds the maximum negative number, it sets the overflow indicator and resets the sign.

Multiply: multiplies the two words at the top of the stack and replaces them by their 32-bit product (see figure 20-3). The most significant part of the product is placed in the top word, and the least significant portion will be placed in the next word. The sign bit of the top word gives the sign of the product, and the sign of the next word is set to zero. The overflow indicator (OF) is not set.

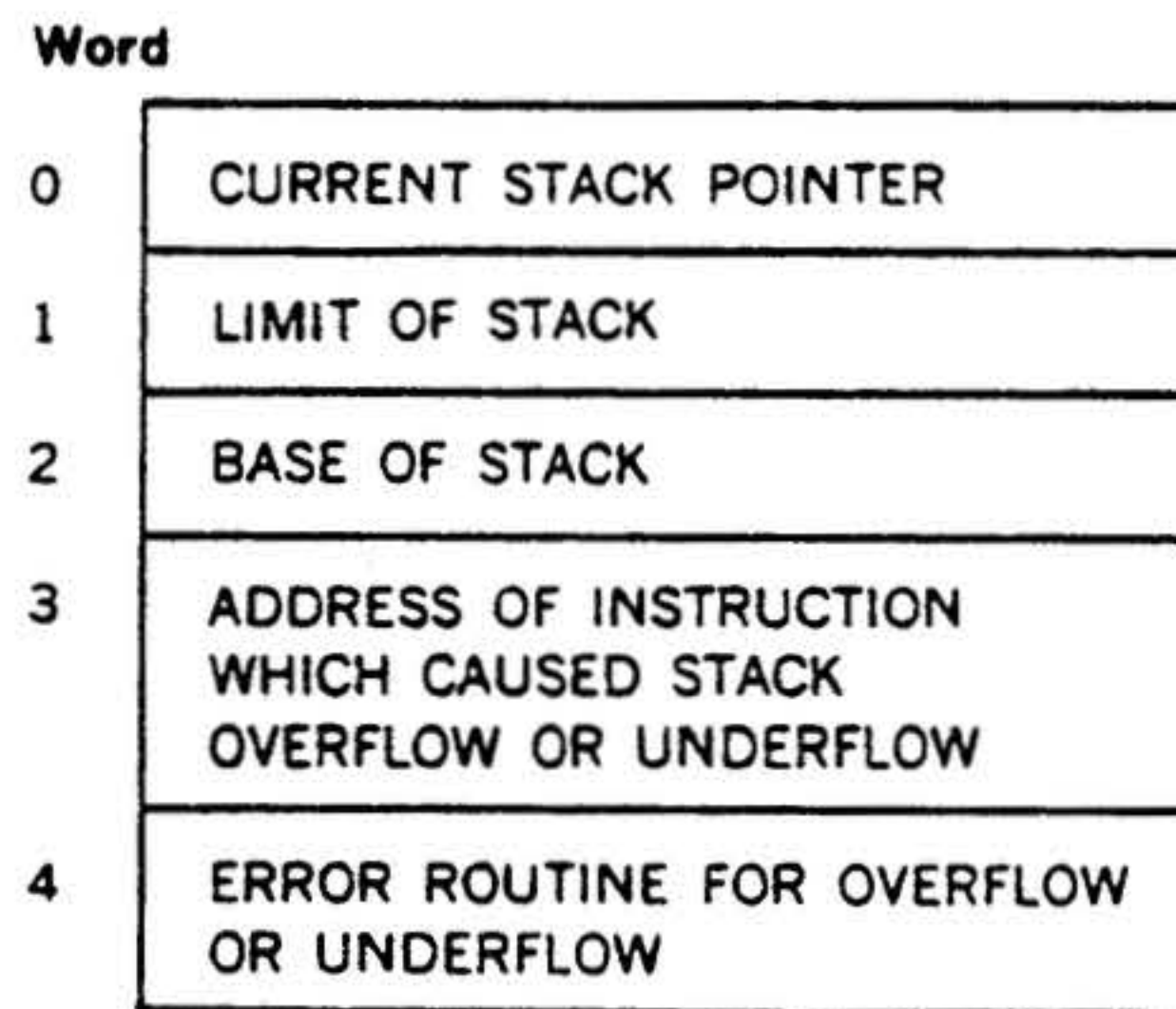


Figure 20-2. Stack Control Block

Divide: divides the top stack word into the following two words. The top-of-stack pointer (PTR) is incremented and the single-precision quotient with the sign of the dividend is stored in the new top-of-stack location. The remainder is stored in the next stack location (see figure 20.4).

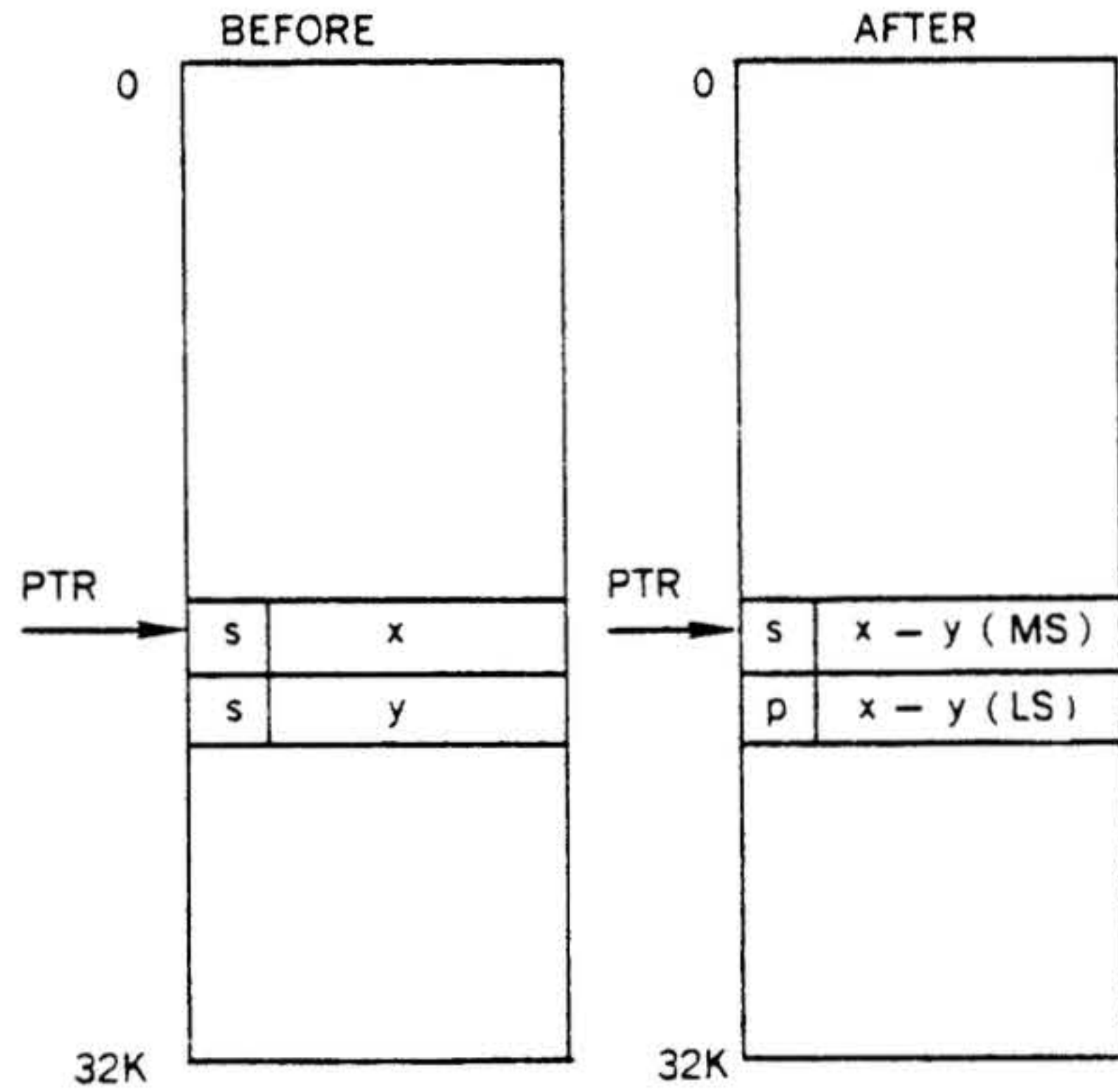


Figure 20-3. Stack Multiply

Stack operators: these operators also require a stack control block as in figure 20-2.

Push (SPUSH): the A register (R0) is placed on the stack at the location addressed by the decremented top-of-stack pointer (see figure 20-5.)

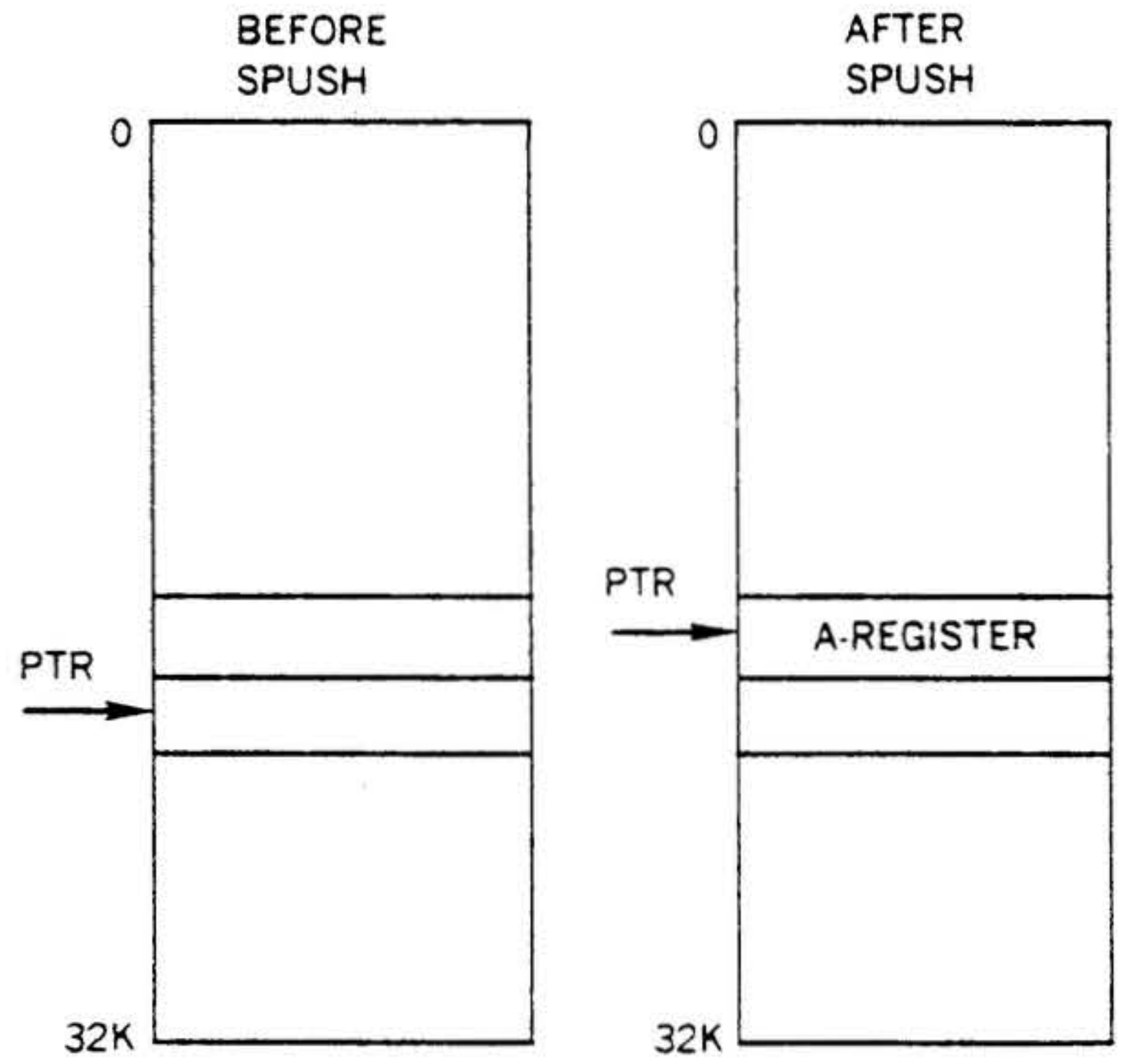


Figure 20-5. Stack Push

If the quotient overflows, the contents are unpredictable, and control is returned with the overflow indicator set (OF).

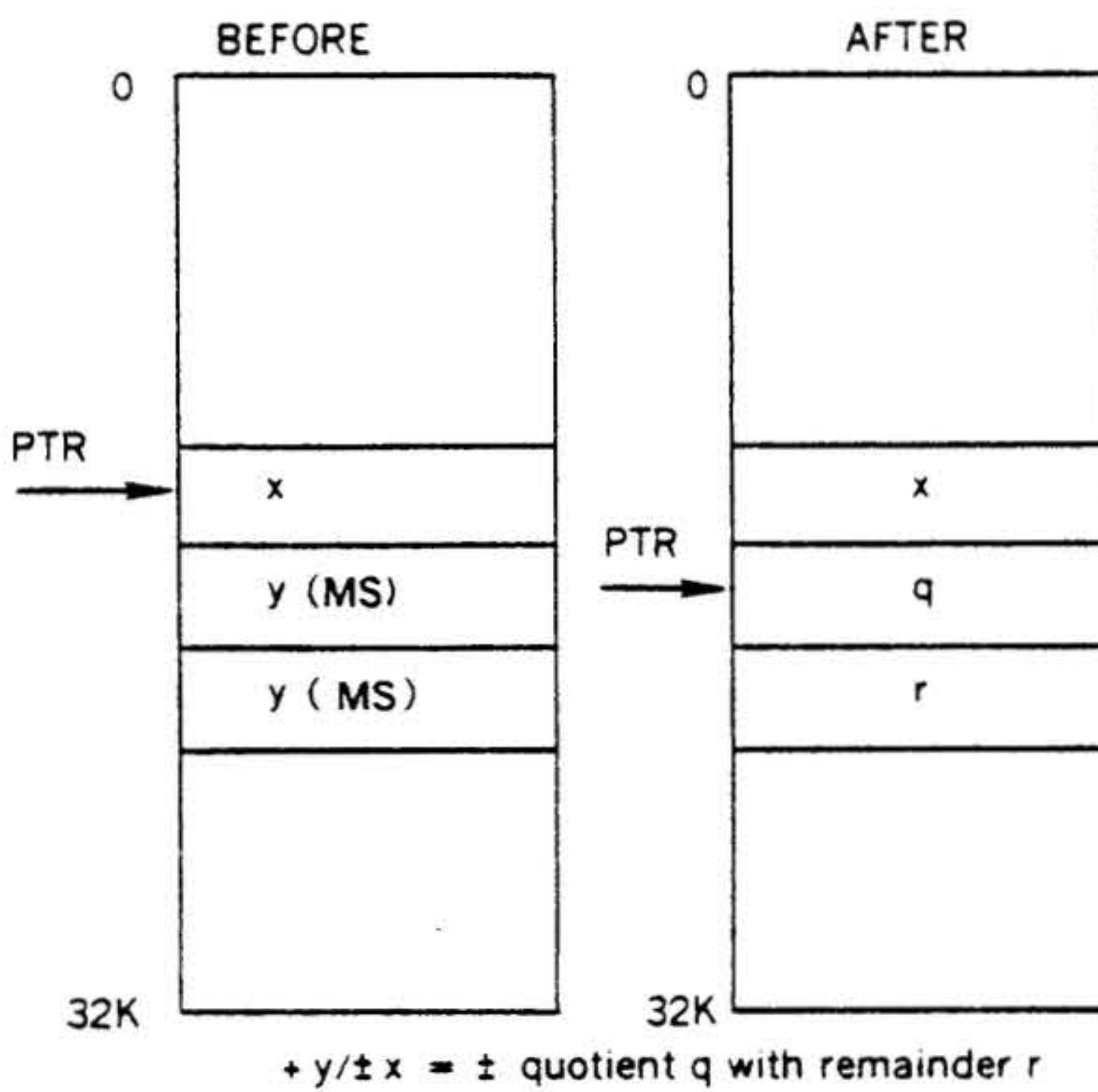


Figure 20-4. Stack Divide

Pop (SPOP): the A-register (R0) is loaded from the top stack word and the stack pointer is incremented (see figure 20-6).

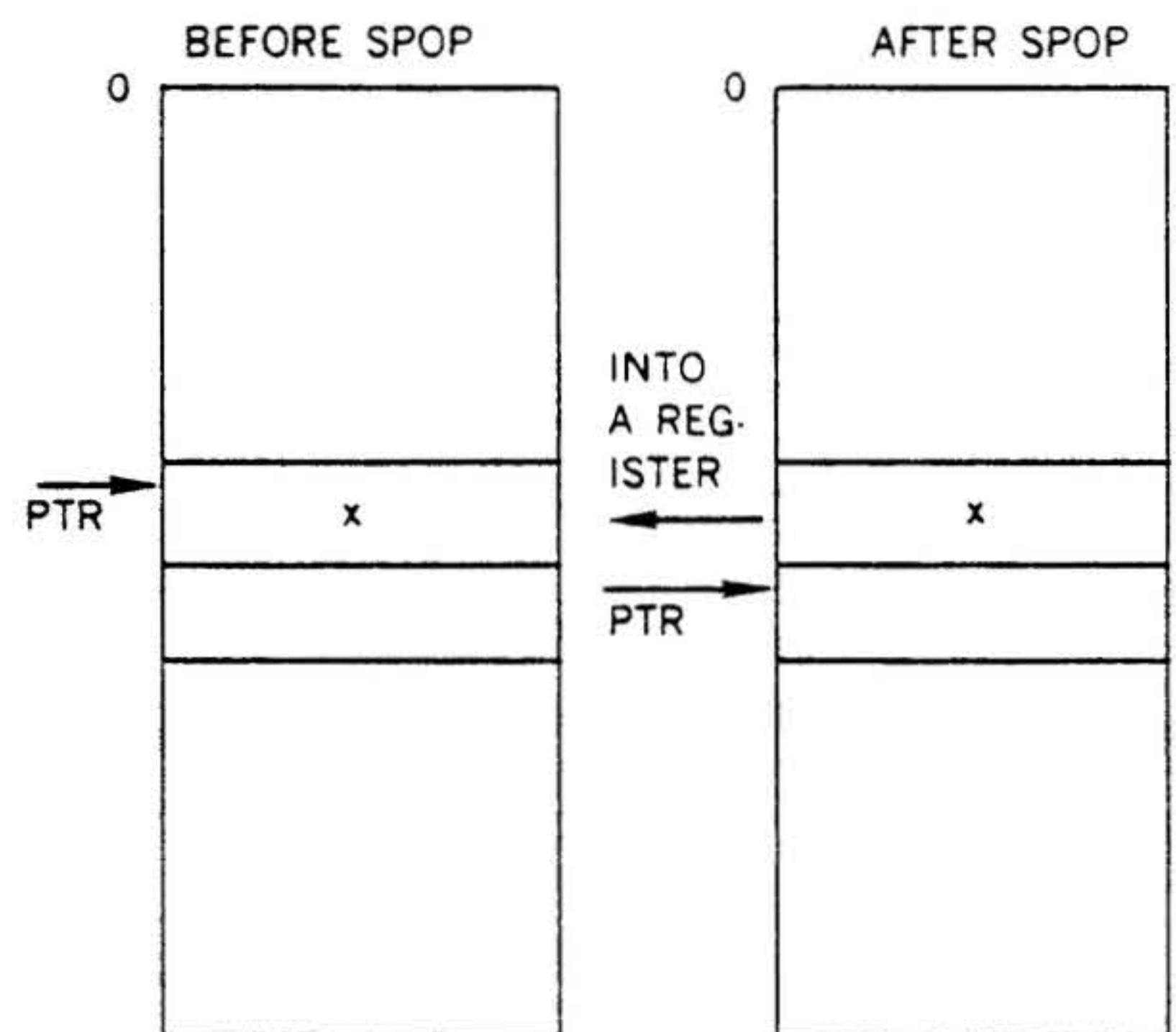


Figure 20-6. Stack Pop

Push Double (PUSHD): decrements the stack pointer and stores the B register (R1), and then decrements the pointer and stores the A register (R0) (see figure 20-7).

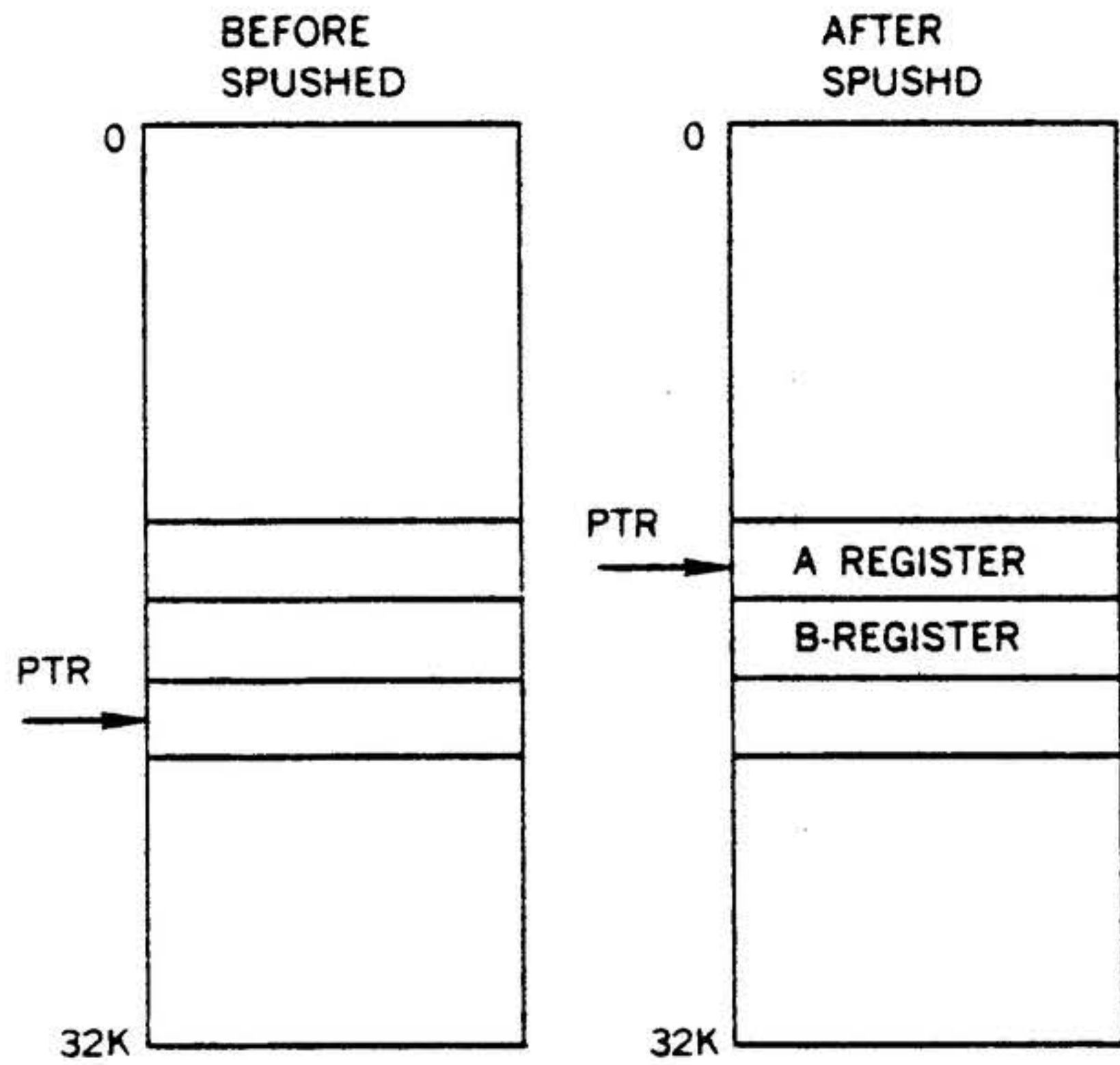


Figure 20-7. Stack Double Push

Pop Double (POPD): loads the A register (R0) with the word addressed by the top-of-stack pointer and then increments the top-of-stack pointer; loads the B register (R1) with the word addressed by the new value of the top-of-stack register and then increments the top-of-stack pointer again (see figure 20-8).

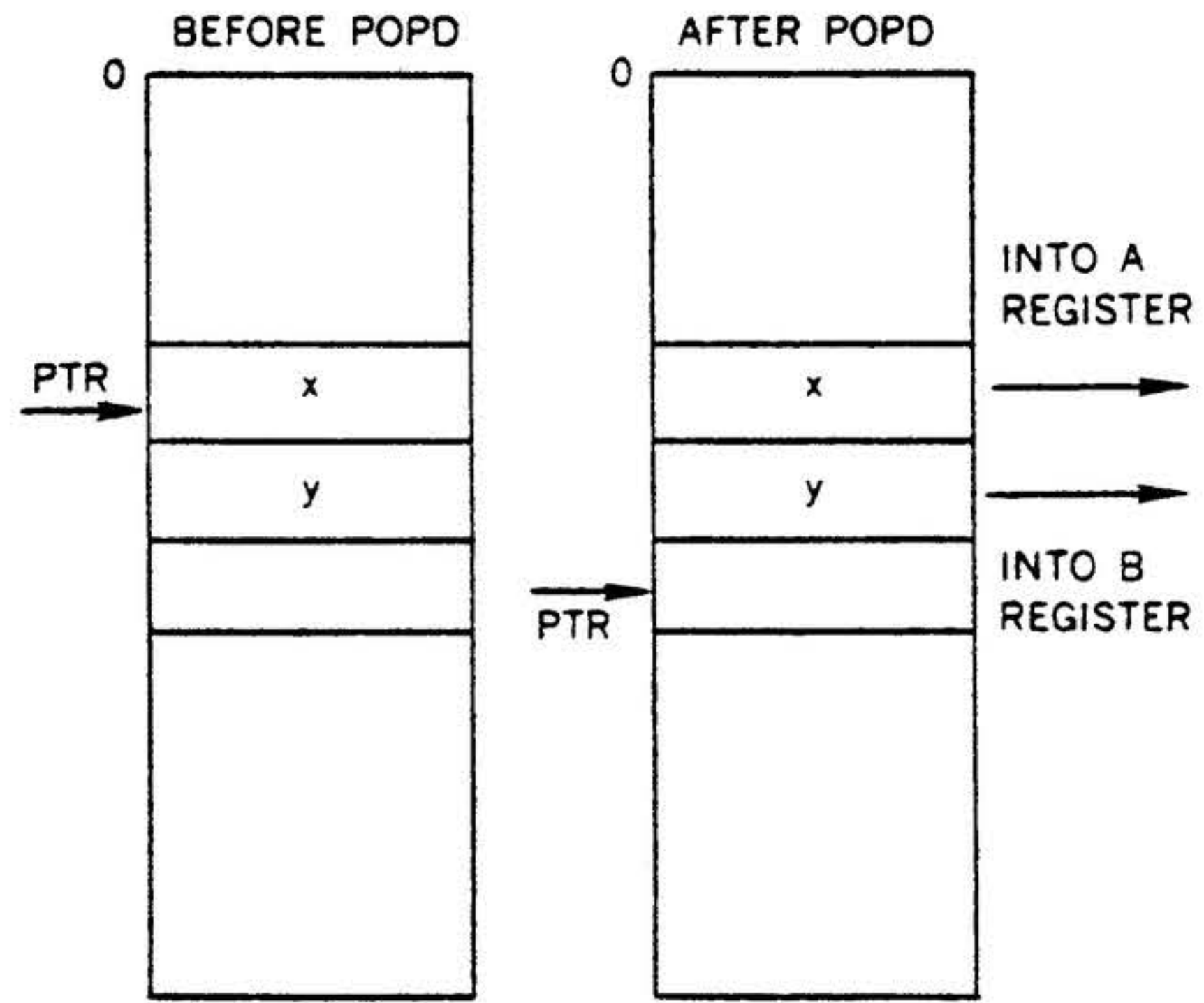


Figure 20-8. Stack Double Pop

20.2.7 Firmware Macros

The mnemonics given are not supported by the DAS MR assembler. The assembly-language programmer must supply his own macros in order to use any of these mnemonics. The following are examples and possible use of the required macros.

Macro			Use
Fixed point add:			
XAD	MAC DATA EMAC	0105334,P(1)	XAD address
Fixed point subtract:			
XSB	MAC DATA EMAC	0105374,P(1)	XSB address
Fixed point multiply:			
XMU	MAC DATA EMAC	0105274,P(1)	XMU address
Fixed point divide:			
XDV	MAC DATA EMAC	0105234,P(1)	XDV address
Integer multiply:			
IMU	MAC DATA EMAC	0105027,P(1)	IMU address
FIMPY	MAC DATA DATA EMAC	0105015 + P(1) P(2)	FMPY ,address or FMPY 040,address
Integer divide:			
IDV	MAC DATA EMAC	0105067,P(1)	IDV address
and, immediately following the macros for floating point divide, add:			
Floating square root:			
FSQ	MAC DATA EMAC	0105127,P(1)	FSQ address

WRITABLE CONTROL STORE AND FLOATING-POINT PROCESSOR

Floating point add:

FAD	MAC		FAD	address
	DATA	0105134,P(1)		
	EMAC			

Floating point subtract:

FSB	MAC		FSB	address
	DATA	0105174,P(1)		
	EMAC			

Floating point multiply:

FMU	MAC		FMU	address
	DATA	0105074,P(1)		
	EMAC			

Floating point divide:

FDV	MAC		FDV	address
	DATA	0105034,P(1)		
	EMAC			

Load AB:

FLD	MAC		FLD	address
	DATA	0105032,P(1)		
	EMAC			

Store AB:

FST	MAC		FST	address
	DATA	0105033,P(1)		
	EMAC			

Memory to memory:

FMV	MAC		FMV	from address to address
	DATA	0105037,P(1),P(1)		
	EMAC			

Pass parameters:

FSE	MAC		FSE	#params
	DATA	0105036,P(1)		
	BSS	P(1)		
	EMAC			

DO loop:

FDO	MAC		FDO	inc addr, count addr,
	DATA	0105035,P(1),P(2),		lim addr, loop addr
	EMAC	P(3),P(4)		

DO loop (one increment):

FDO1	MAC		FDO1	count addr, lim addr,
	DATA	0105027,P(1),P(2),P(3)		loop addr
	EMAC			

WRITABLE CONTROL STORE AND FLOATING-POINT PROCESSOR

Test for not equal:

FTNE	MAC		FTNE	OP address, OP address
	DATA	0105024,P(1),P(2)		
	EMAC			

(Typical relational test form).

Jump if not equal:

FJNE	DATA	0105026,P(1),P(2),P(3)	FJNE	OP address, OP address jump address
------	------	------------------------	------	--

(Typical relational Jump form).

Arithmetic IF processor:

FAIF	MAC		FAIF	OP address, OP address, LT address, EQ address, GT address
	DATA	0105226,P(1),P(2),P(3),P(4),P(5)		
	EMAC			

Index operand processor:

FIOP	MAC		FIOP	index address, base address
	DATA	0105167,P(1),P(2)		
	EMAC			

Reentrant subroutine call:

FRSC	MAC		FRSC	sub address
	DATA	0105025,P(1)		
	EMAC			

Reentrant subroutine return:

FRSR	MAC		FRSR	
	DATA	0105065		
	EMAC			

Jump if A register greater:

FJAG	MAC		FJAG	jump address
	DATA	0105125,P(1)		
	EMAC			

General subscripting:

NOX120	MAC	
	DATA	0105016 + ((2-P(1)) * 128) + (((4 - P(2))/2) - ((4 - P(2)/2)/4) * 4) * 128)
	DATA	P(3)
	IFP	P(1), , 1
	GOTO	1
	DATA	P(4)
	DATA	P(5)
	DATA	P(6)
	GOTO	2
1	DATA	P(4)
2	CONT	

WRITABLE CONTROL STORE AND FLOATING-POINT PROCESSOR

EMAC

NOX120 should have the format

number dimensions, words/element, first subscript address, address of base

or

number dimensions, words/element, first subscript address, first dimension address, second subscript address, address of base

Compare string:

CBS	MAC		CBS	non compare addr
	DATA	0105030,P(1)		
	EMAC			

Move string:

MBS	MAC		MBS	
	DATA	0105070		
	EMAC			

Stack add:

SADD	MAC		SADD	stack addr
	DATA	0105031,P(1)		
	EMAC			

Stack subtract:

SSUB	MAC		SSUB	stack addr
	DATA	0105071,P(1)		
	EMAC			

Stack multiply:

SMUL	MAC		SMUL	stack addr
	DATA	0105131,P(1)		
	EMAC			

Stack divide:

SDIV	MAC		SDIV	stack addr
	DATA	0105171,P(1)		
	EMAC			

Stack push:

SPUSH	MAC		SPUSH	stack addr
	DATA	0105231,P(1)		
	EMAC			

Stack pop:

SPOP	MAC		SPOP	stack addr
	DATA	0105331,P(1)		
	EMAC			

Stack push double:

SPUSHD	MAC		SPUSHD	stack addr
	DATA	0105271,P(1)		
	EMAC			

Stack pop double:

SPOPD	MAC		SPOPD	stack addr
	DATA	0105371,P(1)		
	EMAC			

The Floating Point Processor has the following OP codes.

Mnemonic	Opcode	Operation
FLD	0105420	Floating load single
FLDD	0105522	Floating load double
FAD	0105410	Floating add single
FADD	0105503	Floating add double
FSB	0105450	Floating subtract single
FSBD	0105543	Floating subtract double
FMU	0105416	Floating multiply single
FMUD	0105506	Floating multiply double
FDV	0105401	Floating divide single
FDVD	0105535	Floating divide double
FLT	0105425	Fix to float
FIX	0105621	Float to fix
FST	0105600	Floating store single
FSTD	0105710	Floating store double

Load or Float interrupts are locked out until a store or fix.
EX34, -- as time out.

An interrupt after a store may change floating-point registers. User should restore their contents.

Mnemonics for floating-point operations are not supported by DAS MR. The following are possible macros which must be included by the user to define the mnemonics:

	Macro		Use	
FLD	MAC		FLD	address
	DATA	0105420,P(1)		
	EMAC			
FLDD	MAC		FLDD	address
	DATA	0105522,P(1)		
	EMAC			
FAD	MAC		FAD	address
	DATA	0105410,P(1)		
	EMAC			
FADD	MAC		FADD	address
	DATA	0105503,P(1)		
	EMAC			

WRITABLE CONTROL STORE AND FLOATING-POINT PROCESSOR

FSB	MAC DATA EMAC	0105450,P(1)	FSB	address
FSBD	MAC DATA EMAC	0105543,P(1)	FSBD	address
FMU	MAC DATA EMAC	0105416,P(1)	FMU	address
FMUD	MAC DATA EMAC	0105506,P(1)	FMUD	address
FDV	MAC DATA EMAC	0105401,P(1)	FDV	address
FDVD	MAC DATA EMAC	0105535,P(1)	FDVD	address
FLT	MAC DATA EMAC	0105425,P(1)	FLT	address
FIX	MAC DATA EMAC	0105621,P(1)	FIX	address
FST	MAC DATA EMAC	0105600,P(1)	FST	address
FSTD	MAC DATA EMAC	0105710,P(1)	FSTD	address

20.2.8 Commercial Firmware

Commercial firmware is available on the 70 series computers for supporting VORTEX, COBOL, and TOTAL.

Commercial firmware includes the following operations:

- Load/Store multiple registers
- 32 bit unsigned math

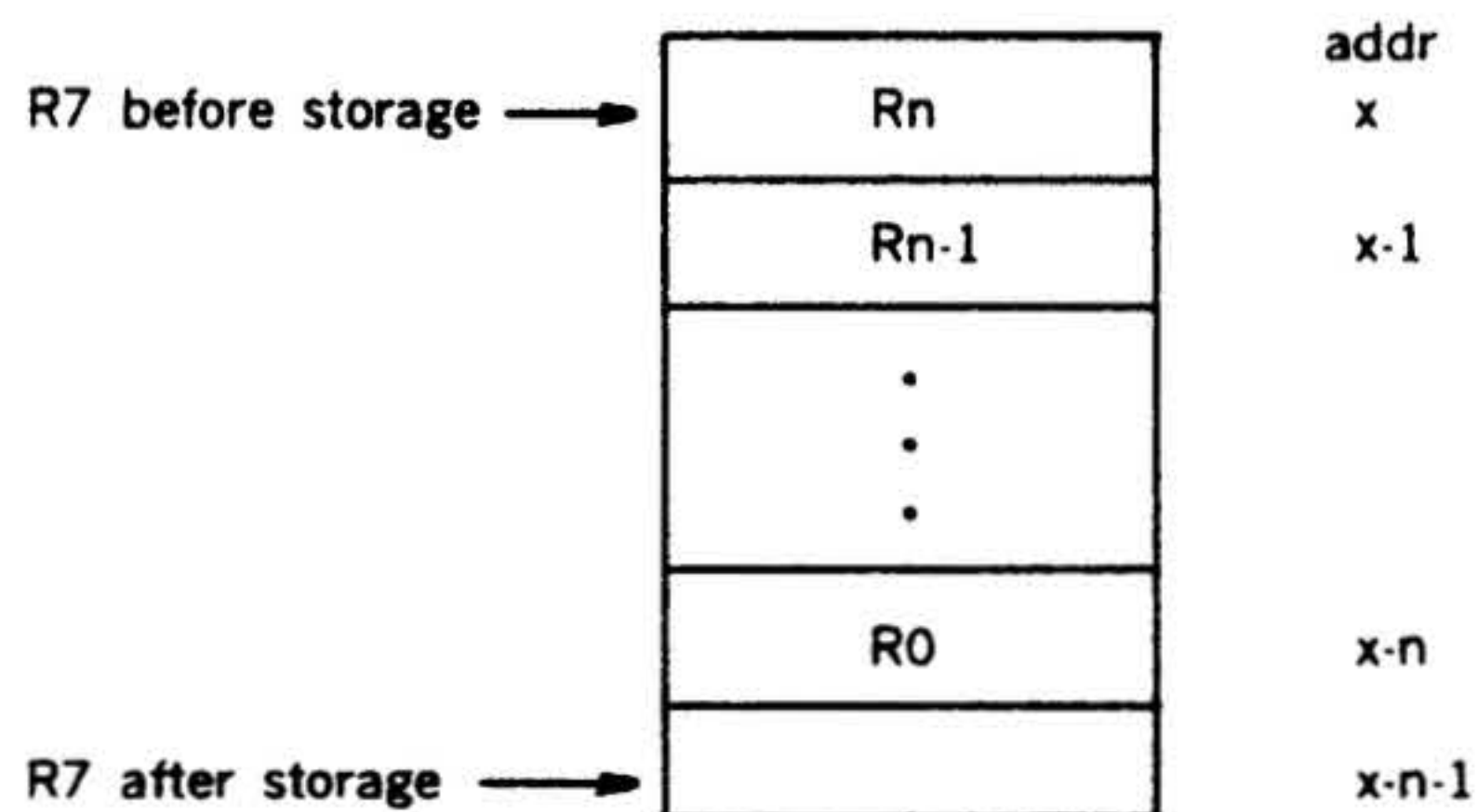
Load/Store Registers

Multiple register loading or storing is performed by the following BCS instructions:

Registers loaded/stored			
DATA	0105020	load	R0
	0105060	↑	R0,R1
	0105120	↑	R0,...,R2
	0105160	↑	R0,...,R3
	0105220	↑	R0,...,R4
	0105260	↑	R0,...,R5
	0105320	↑	R0,...,R6
	0105360	load	R0,...,R7
DATA	0105017	store	R0
	0105057	↑	R0,R1
	0105117	↑	R0,...,R2
	0105157	↑	R0,...,R3
	0105217	↑	R0,...,R4
	0105257	↑	R0,...,R5
	0105317	↑	R0,...,R6
	0105357	store	R0,...,R7

WRITABLE CONTROL STORE AND FLOATING-POINT PROCESSOR

R7 contains the main storage address for loading or storing registers. Register contents are stored in main storage as follows:



R7 is decremented to the location following the contents of R0. For load registers, R7 initially points to the word following R0. After loading is complete, R7 will point to the last register loaded.

32 Bit Integer Math

These routines perform the operations add, subtract, multiply, and divide on 32 bit unsigned integer operands. Register R0(A) contains the four word parameter block address. The four word parameter block contains the two operands and received the results as follows:

add	Operand two is replaced by the sum of the two operands.
subtract	Operand two is replaced by operand one minus operand two.
multiply	Both operands are replaced by the 4 word product of the two operands.
divide	Operand one receives the quotient of operand one divided by operand two; operand two is replaced by the remainder.

The hardware overflow flag is set when any of the following occur:

- carry out of the most significant bit during an add.
- subtracting a larger number from a smaller one.
- dividing by zero.

The calling routine uses one of the following instructions:

Add	DATA 0105023
Subtract	DATA 0105063
Multiply	DATA 0105123
Divide	DATA 0105163



SECTION 21

FILE MAINTENANCE UTILITY

21.1 INTRODUCTION

The File Maintenance Utility program, (FMUTIL) is a background task which performs the following functions:

- Copies files, file directories, and/or partitions from one device to another
- Loads files, file directories, and/or partitions onto a device.
- Manipulates files and records
- Formats files and records which are to be printed or displayed.
- Manages filename directories
- Manages space allocation for files.

The following items should be noted when using FMUTIL.

- Only files assigned to disk devices can be referenced by name.
- Filespace allocated by FMUTIL is allocated contiguously within a partition, skipping bad tracks.

21.2 ORGANIZATION

FMUTIL is scheduled for execution by the JCP directive /FMUTIL. If the SI logical unit is a teletypewriter or a CRT device, the message FL** is output to indicate that the SI unit is waiting for FMUTIL input. Once activated, FMUTIL accepts directives from the SI unit until another JCP directive (first character is a slash) is input, or the exit directive, E is input.

In either case, FMUTIL terminates and JCP is scheduled.

If there is an error, one of the error messages given in Appendix A is output on logical unit SO, and a record is input from the SO unit to the JCP buffer. If the first character of this record is /, FMUTIL exits via the EXIT request. If the first character is C, FMUTIL continues. If the first character is neither / or C, the record is processed as a normal FMUTIL directive.

21.3 OUTPUT LISTINGS

FMUTIL outputs the following two types of listings to the LO logical unit:

- **Directive Listing** - lists, without modification, all FMUTIL directives entered from SI logical unit.

- **Directory Listing** - lists file names from a logical unit filename directory in response to the FMUTIL P,D,T,C and L directives.

All FMUTIL listings begin with the standard headings.

21.4 FMUTIL - OVERVIEW

The FMUTIL directives are listed and described in Table 21-1. The list order corresponds to the order of the subsections that follow, in which the directives are discussed in detail.

Table 21-1. FMUTIL Directives

Directive	Function
D	Dump individual files or entire partitions to a magnetic tape.
L	Load individual files or entire partitions from a magnetic tape.
R	Rewind magnetic tape.
E	Exit from FMUTIL or write end-of-file (EOF) to magnetic tape.
S	Search for specified partitions on magnetic tape.
P	Print contents of a partition in alphabetical order.
U	Release unused space in a specified file.
T	List contents of magnetic tape created by a dump.
X	Set expiration date for use in conjunction with the L or C directive.
I	Initialize disk partition.
C	Copy individual files or an entire partition to another disk partition.

File maintenance utility directives consist of sequences of character strings having no embedded blanks. The characters strings are separated by commas (,) or equal signs (=). Although not required, a period (.) is a line terminator. Comments can be inserted after the period.

The general form of a file maintenance utility directive is:

directive,p(1),p(2),...,p(n)

where

directive Is one of the directive names given in Table 21-1.

p(n) Is a parameter.

FILE MAINTENANCE UTILITY

Numerical data can be octal or decimal. Each octal number must have a leading zero.

For greater clarity in the descriptions of the directives, optional blank separators between character strings, and the optional replacement of commas (,) by equal signs (=) are omitted from the descriptions.

Error messages applicable to file maintenance utility directives are given in Appendix A.

21.5 D DIRECTIVE

The D directive dumps information contained in files, partitions, and/or directories onto magnetic tape. Once this information is on magnetic tape, it can be reloaded onto disk, or stored for later use. There are three types of D directives; dump file, dump partitions, and dump directories. Each D directive is described in the following paragraphs.

21.5.1 Dump File

The directive for dumping a file has the general form:

D,lun.key,file,tapelun

where

- lun** Is the number or name of the input logical unit.
- key** Is the partition protection code.
- file** Is the name of the file being dumped.
- tapelun** Is the number or name of the output logical unit (magnetic tape only). Note that **tapelun** cannot be equal to 50.

When individual files are dumped, the tape format consists of a file header, "EOF.", text record, file image (in 5760 word blocks, except for last block), and an "EOF." text record.

Note: Magnetic tapes created by pre-VORTEX G1 FMUTIL have EOF tape marks in place of the "EOF." text record.

File information is listed to the LO device as the file is dumped in the following format:

file name, file type, extension, sectors used, sectors unused, total sectors, first sector number, last sector number, create date, access date.

All values are expressed in decimal.

Note: Dumping individual files only, does not terminate the output tape properly for all input functions (S and T). The E command must be used twice to properly terminate the output tape if the last operation to the tape was an individual file dump. No logical unit is kept in the individual file header when this directive is used. This means that the dumped file can be loaded to a different LUN than from which it was dumped.

Example: Dump the file DEBUG from the OM library with a protection code D to logical unit 18.

D,OM,D.DEBUG,18

A sample listing produced by this directive is shown in Figure 21-1.

VTAM	DATA	0	2221	1279	3500	6909	10429	05/18/79	06/18/79
↑	↑	↑	↑	↑	↑	↑	↑	↑	↑
filename	file type	extension	sectors used	sectors unused	total sectors	first sector number	last sector number	create date	access date

Figure 21-1. Sample Output Produced by Dump File Directive

SECTION 21 FILE MAINTENANCE UTILITY

The File Maintenance Utility program (FMUTIL) is a background task for copying and/or loading files, file directories and/or partitions from one device onto another, for manipulating files and records, for formatting files and records which are to be displayed or printed, and for managing filename directories and space allocations of the files.

Only files assigned to rotating memory devices (disc or drum) can be referenced by name.

File space is allocated contiguously within a partition, skipping bad tracks.

21.1 ORGANIZATION

FMUTIL is scheduled for execution by inputting the JCP directive /FMUTIL. If the SI logical unit is a teletype or a CRT device, the message FU** is output to indicate that the SI unit is waiting for FMUTIL input. Once activated, FMUTIL accepts directives from the SI unit until:

- a. Another JCP directive (first character is a slash) is input, or
- b. The exit directive, E, is input.

In either case, FMUTIL terminates and JCP is scheduled.

If there is an error, one of the error messages given in appendix A is output on the SO logical unit, and a record is input from the SO unit to the JCP buffer. If the first character of this record is /, FMUTIL exits via the EXIT request. If the first character is C, FMUTIL continues. If the first character is neither / or C, the record is processed as a normal FMUTIL directive.

21.2 PARTITION SPECIFICATION TABLE

For a description of the Partition Specification Table (PST) and File Name Directory, refer to section 9.1.

21.3 OUTPUT LISTINGS

FMUTIL outputs the following two types of listings to the LO logical unit:

- a. Directive Listing lists, without modification, all FMUTIL directives entered from SI logical unit.
- b. Directory Listing, lists file names from a logical unit filename directory in response to the FMUTIL,P,D, and L directives.

All FMUTIL listings begin with the standard VORTEX headings.

21.4 FILE MAINTENANCE UTILITY DIRECTIVES

The following file maintenance utility functions are supported by FMUTIL:

- D Dumps RMD files, partitions, and file name directories to magnetic tape.
- L Loads RMD files, partitions, and file name directories from magnetic tape.
- R Rewinds magnetic tape.
- E Writes end-of-file on magnetic tape.
- S Searches for RMD files, partitions, and file name directories on magnetic tape.
- P Prints a listing of file names contained on each directory.
- U Releases all unused space in each file.
- E Exits from FMUTIL.

File maintenance utility directives comprise sequences of character strings having no embedded blanks. The character strings are separated by commas (,) or key equal signs (=). Although not required, a period (.) is a line terminator. Comments can be inserted after the period.

The general form of a file maintenance utility directive is

directive, p(1),1(2),...,p(n)

where

directive is one of the directive names given above.

p(1) is a parameter

Numerical data can be octal or decimal. Each octal number has a leading zero.

For greater clarity in the descriptions of the directives, optional blank separators between character strings, and the optional replacement of commas (,) by equal signs (=) are omitted.

FILE MAINTENANCE UTILITY

Error messages applicable to file maintenance utility directives are given in appendix A.

21.5 D DIRECTIVE

This directive dumps information contained in files, partitions, and/or directories onto magnetic tape where this information can be later re-loaded onto the RMD, or stored for later use. There are three types of D directives; one for file, one for partitions, and one for directories.

21.5.1 Dump File

The directive for dumping a file has the following general form

D,lun,key,file,tapelun

where

- lun** is the number of name of the input logical unit.
- key** is the partition protection code.
- file** is the name of the file being dumped.
- tapelun** is the number or name of the output logical unit. (magnetic tape only)

When a file is dumped to magnetic tape, it is organized with a header record, end-of-file, n file records, and terminates with a double end-of-file. The file, after the dump with the header record, is formatted as follows:

Each n file record has 5,760 words, except for the last which has the remaining number of words in the file. In other words, the last record may have less than 5,760 words.

On a dump file directive a listing is output. The listing output format is as follows:

PAGE XXXX XX/XX/XX XX:XX:XX VORTEX PNTLCX PNTUIL

D, 22, X, COBINT, 18
COBINT 141 0 141

The top heading line consists of:

- a. One blank
- b. The word PAGE
- c. Four character positions that contain the decimal page number
- d. Two blanks
- e. Eight character positions that contain the current data obtained from the VORTEX resident constant V\$DATE.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
Word 0								'F'								.				'I'
Word 1								'L'											'E'	
Word 2	FCB																			
Word 3																				
Word 4																				
Word 5																				
Word 6																				
Word 7																				
Word 8																				
Word 9																				
Word 10																				
Word 11																				
end-of-file																				

5760 word data record
.
.
.
5760 word data record
≤ 5760 word last data record
end-of-file
end-of-file

- f. Two blanks
- g. Eight character positions that contain the current time
HR: MN: SC.
- h. Two blanks
- i. Name of run-time operating system.
- j. Two blanks
- k. The /JOB name of which the system is executing
- l. Two blanks
- m. Eight character positions that contain the job processor
name, FMUTIL
- n. Blanks through the 120th character position.

Beginning with the first character position, the next line (after 2 blank lines) contains the list of the input directives.

Beginning with the first character position the next line contains: the name of the file, number of sectors used, number of sectors unused, and the number of total sectors allocated to the file.

Example: Dump the file COBINT contained on logical unit 22, whose protection code is X, onto magnetic tape unit 18.

D, 22, X, COBINT, 18

21.5.2 Dump Partition

The directive for dumping a partition has the following general form

D,lun,key,ALL,tapelun

where

lun is the number or name of the input logical unit.

key is the protection code required to address lun.

tapelun is the output logical unit (magnetic tape only).

ALL keyword specifying partition dump.

All partitions dumped onto magnetic tape are organized with a header record, n files record, and terminated by an end-of-file.

The header record is formatted as follows:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Word 0							'P'									'A'	
Word 1							'R'									'T'	
Word 2	number of file entries																
Word 3	logical unit number																
Word 4	all zeros																
Word 5																	
Word 6																	
Word 7																	
end-of-file																	

FILE MAINTENANCE UTILITY

An alternate name record has the format shown below:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 0	'E'						'N'									
Word 1	'T'						'R'									
Word 2	Entry Name															
Word 3																
Word 4																
Word 5	Original Name															
Word 6																
Word 7																
Word 8	file size															
	end-of-file															

A partition dump directive produces a listing. This listing output format has the following FMUTIL heading, a one line heading as shown below:

FILENAME USED UNUSED TOTAL LOGICAL UNIT-XXXX

The heading line consists of:

- One blank
- The word FILENAME that shows an alphabetical list of all the file located on a particular partition.
- Four blanks
- The word USED shows many sectors, of each file, contain information.
- Four blanks
- The word UNUSED shows how many sectors contain blanks.
- Five blanks
- The word TOTAL shows the total number of sectors allocated to each file.
- Forty spaces
- The words LOGICAL UNIT shows what logical unit the files are located on.
- Four character positions that contain the logical unit number.

Example: Dump the partition contained on logical unit OM, protection code D, onto magnetic tape unit 18.

D, OM, D, AL, 18

21.5.3 Dump File-Name Directory

The directive for dumping a directory has the following general form

D,lun,key,DIR,tapelun

where

- lun** is the number or name of the input logical unit.
- key** is the protection code required to address lun.
- tapelun** is the number or name of the output logical unit. (magnetic tape only.)
- DIR** keyword specifying directory dump.

A filename directory dumped to magnetic tape is organized into a header record, directory record, and double end-of-file. The header record is formatted as follows:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 0	'D'								'I'							
Word 1	'R'								blank							
Word 2	all zeros															
Word 3	logical unit number															
Word 4	all zeros															
Word 5																
Word 6																
Word 7																
end-of-file																

The directory record has the following format:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 0	Directory Sector Addr															
1-120	120 word directory block															
121	Directory Sector Addr															
122-241	120 word directory block															
	.															
	.															
	.															
5639	Directory Sector Addr															
5640 5759	120 word directory block															
end-of-file																
end-of-file																

FILE MAINTENANCE UTILITY

Example: Dump directories for partition contained on logical unit OM, protection code D, onto magnetic tape unit 18.

```
D, OM, D, DIR, 18
```

21.6 L DIRECTIVE

This directive loads information into RMD files, partitions, and/or directives from magnetic tape.

There are three types of L directives, one for files, one for partitions, and one for directories.

21.6.1 Load File

The directive for loading a file has the following general form

```
L, lun, key, file, tapelun
```

where

lun	is the number or name of the output logical unit.
key	is the partition protection code.
file	is the name of the file being loaded.
tapelun	is the number or name of the input magnetic tape unit.

When a file is being loaded from magnetic tape, a search is made for that file. After the search, the tape is positioned in front of the file within the correct partition dump. The search stops if a double end-of-file is encountered and an error message is output. After the file is located, an attempt is made to create the file space. If the file already exists the existing file is used. If the existing file is too small, an error message is output.

When creating a file for loading, the file size of the created file will include all of the original extent of the file, including the unused portion.

When a file already exists, the only check made is to see if there is enough space for the used portion of the file as on the tape, and the original extent of the file is ignored.

On a load file directive a listing is output. The listing output format is the same as the D directive when files are called. The only change would be the directive shown on the listing.

Example: Load the file COBINT contained on magnetic tape unit 18 onto RMD logical unit 22, protection code is X.

```
L, 22, X, COBINT, 18
```

21.6.2 Load Partition

The directive for loading a partition has the following general form

```
L, lun, key, ALL, tapelun
```

where

lun	is the number or name of the output logical unit.
key	is the partition protection code.
tapelun	is the number or name of the input magnetic tape unit.
ALL	keyword specifying partition load.

When a partition is loaded, from magnetic tape, a search is made for it as specified by the logical unit number parameter. After the search tape is positioned in front of the required partition dump, the search stops if a triple end-of-file is encountered and an error message is output.

When the partition is found, the files are loaded as indicated key file loading in the order in which they appear on the tape. If any non-previous record names are encountered, an entry is made in the directory for them.

During the loading of a partition, space for the directory is allocated at the beginning of the partition. After loading, however, there is no embedded unused space in the partition. All unused space is at the end of the partition.

On a partition load directive, a listing is output. The listing output has the following FMUTIL heading, a one-line heading as shown below:

```
FILENAME USED UNUSED TOTAL START END LOGICAL UNIT-XXXX
```

The heading line consists of:

- One blank
- The word FILENAME that gives a list of all filenames now contained in the partition.
- Four blanks
- The word USED shows how many sectors per filename contain valid information.
- Four blanks
- The word UNUSED shows how many sectors per filename contain blanks.
- Five blanks

Whenever a file header is encountered it is listed to LO in the format

FILE: FCB = file name , file BOD , file EOD

Whenever an entry header is encountered it is listed to LO in the format:

ENTR: ENTRY NAME = entry name , ORG NAME = root name

Actual file contents and control records "EOF.", "EOP.", and "EOT" are not listed.

Example: Display the contents of the FMUTIL tape on logical unit number 18.

T,18

21.13 X DIRECTIVE

This directive is used to set the expiration date to be used in conjunction with the "L ALLNEW", "D ALL", "C ALL", or "C ALLNEW" directives. Files (except load modules) which have not been accessed since the specified date will not be reloaded onto a partition during the performance of a L or C directive. The X directive is useful for removing obsolete files.

The X directive has the format

X,MM/DD/YY

or

X,T

where

MM is a two-digit month number.

DD is a two-digit day number.

YY is a two-digit year number.

If X,T is used then the current system date is the expiration date. This form of the directive can be used to dump only the files accessed that day. The only way to revoke an expiration date is to provide a new date with another X directive.

Example: Set an expiration date of January 1, 1979.

X,01/01/79

21.14 I DIRECTIVE

This directive is used to initialize a partition directory. The initialized partition is provided with the long directory format including access dates, creation dates, and file types. The I directive has the format

I,lun,key

where

lun is the number or name of the input logical unit.

key is the protection code, if any, required for addressing the lun.

Example: Initialize the directory for logical unit 30, protection code M.

I,30,M

21.15 C DIRECTIVE

This directive is used to copy a single file or entire partition from one disk partition to another. It is not used for loading files from or dumping files to magnetic tape. The functioning of this directive is similar to that of the L directive. The C directive has the format

C,lun1,key1,filename1,lun2,key2,filename2

where

lun1 is the input logical unit name or number.

key1 is the protection code, if any, required for addressing lun1.

filename1 is the name of the file to be copied or one of the keywords ALL or ALLNEW.

lun2 is the output logical unit name or number.

key2 is the protection code, if any, required for addressing lun2.

filename2 is the optional output file name. This parameter is not used if filename1 is a keyword, ALL or ALLNEW.

FILE MAINTENANCE UTILITY

↓ If filename1 is a specific file name, then that file and its alias names will be copied to the output logical unit. If filename1 is either ALL or ALLNEW, then all files on the input logical unit are copied to the output logical unit under the same conditions as for the L directive (see 21.6.2).

If filename2 is omitted and filename1 is a specific file name, then filename1 is used as the output file name. If the specified output file name already exists on the output logical unit, the input file will be copied into that file, provided that the existing file on the output unit contains sufficient space.

Note: The output file extent must be as large, or larger than, the input file extent--including unused space, i.e. that beyond the current EOF. The C directive copies the entire input file, including that information beyond the current EOF. If this is not desired, then the FMUTIL directive U can be used to remove unused space before the copying procedure is begun.

If the input file name is either of the keywords ALL or ALLNEW, then the output file name is not used. When partitions are copied, the output file partition directory is not initialized. The user should use the I directive before the C directive if the output logical unit is not to keep its current files.

The C directive does not perform its own file directory handling; it uses VZFMA for its file creation needs. If the partition to be copied contains a large number of files, the C directive operation will exhibit progressively slower processing per file as the number of entries in the output partition increases. The C directive has a limit of 1600 file names for the input logical unit; if this limit is attained, attempts to copy that partition will result in the message

CAPACITY EXCEEDED

The user can use the FMUTIL directives D and L in this case.

↑ **Note:** If the C directive is used to copy an entire partition into the Background Library using the ALL parameter and FMUTIL

↓ is one of the files being copied, the Background Library copy of FMUTIL should not be used because of the overlaid structure of FMUTIL. If this operation is required, the user should execute the copy of FMUTIL on the input partition instead of the copy on the Background Library.

Example 1: Copy the file FMAIN from partition BL, protection code E, to logical unit number 23.

C,BL,E,FMAIN,23

Example 2: As above except that the output file name is to be FMAINC.

C,BL,E,FMAIN,23,,FMAINC

Example 3: Copy the FL partition to logical unit number 25 and then back to FL in order to remove fragmentation.

I,25
C,FL,F,ALL,25
I,FL,F
C,25,,ALL,FL,F

Example 4: Remove files on logical unit number 195, protection code T, that have not been accessed since January 15, 1980. Use partition 25 as the intermediate scratch partition.

X,01/15/80
I,25
C,195,T,ALL,25
I,195,T
C,25,,ALL,295.T

Example 5: Copy all files on logical unit 30, no protection code, to partition 31, protection code A, that are not already on partition 31.

C,30,,ALLNEW,31,A

- h. The word TOTAL shows how many sectors have been allocated to each file.
- i. Ten blanks
- j. The word START shows the beginning sector number
- k. Seven blanks
- l. The word END shows the ending sector numbers.
- m. Fifteen blanks
- n. The word LOGICAL UNIT shows on which logical unit (partition) these files are contained.
- o. Four character positions that contain the logical unit number.

Example: Load the partition contained on magnetic tape, which is on logical unit 18, onto RMD logical unit name OM, protection code.

L, OM, D, ALL, 18

21.6.3 Load Directory

The directive for loading filename directories has the following general form

L, lun, key, DIR, tapelun

where

- lun** is the number or name of the output logical unit.
- key** is the protection code required to address lun.
- tapelun** is the number or name of the input magnetic tape unit.
- DIR** keyword specifying directory load.

When a directory is being loaded, a search is made for it on the input magnetic tape, after the search tape is positioned in front of the required partition directory.

If the directory is found its sectors are loaded onto their former recorded sectors. No reorganization takes place.

If the directory is not found or if a triple end-of-file is encountered, an error message is output, and the search stops.

Example: Load directory for partition contained on magnetic tape, on magnetic tape unit 18, onto RMD logical unit OM, protection code is D.

L, OM, D, DIR, 18

21.7 R DIRECTIVE

This directive rewinds a magnetic tape to the beginning of tape. The directive has the general form

R, lun

where

- lun** is the number or name of the input or output magnetic tape unit.

Example: Rewind magnetic tape located on logical unit 18.

R, 18

21.8 E. DIRECTIVE

This directive writes an end-of-file on a magnetic tape. The directive has the general form

E, lun

where

- lun** is the number or name of the output magnetic tape unit.

This directive should be used after writing a series of files onto magnetic tape instance:

Header Record | EOF | Series of Partition Files | EOF | EOF | EOF *

*The E directive is used to write the third end-of-file.

E, 18

21.9 S DIRECTIVE

This directive searches for files, partitions, and directories located on magnetic tapes. The directive has the general form

S, lun, key, tapelun

FILE MAINTENANCE UTILITY

where

- lun** is the number or name of the RMD's logical unit.
- key** is the protection code required for addressing lun.
- tapelun** is the number or name of the input magnetic tape unit.

After the search, the tape will be positioned after the partition or file identification record, and is now ready for the loading of individual files.

Example: Search for the partition, file or directory named OM, protection code D, located on logical unit 18.

S,OM,D,18

21.10 P DIRECTIVE

This directive prints out a listing of the file directory on the LO for each partition specified. The directive has the general form

P,lun,key

where

- lun** is the number or name of the input logical unit.
- key** is the protection code required for addressing lun.

Files are listed in alphabetical order. The output listing has, following the FMUTIL heading, a one-line heading as shown below:

FILENAME USED UNUSED TOTAL START END LOGICAL UNIT-XXXX

The heading line consists of:

- One blank
- The word **FILENAME** that gives a list of all filenames contained in a partition.
- Four blanks
- The word **USED** shows how many sectors per filename contain information.
- Four blanks
- The word **UNUSED** shows how many sectors per filename contain blanks.

- Five blanks
- The word **TOTAL** shows how many sectors have been allocated for each file.
- Ten blanks
- The word **START** shows the beginning sector number.
- Seven blanks
- The word **END** shows the ending sector number.
- Fifteen blanks
- The words **LOGICAL UNIT**, one character, a dash (-), shows upon which logical unit (partition) these files are contained.
- Four character positions that contain the logical unit number.

Example: Print a listing of OM, protection code D.

P,OM,D

21.11 U DIRECTIVE

This directive releases unused space from files, after they have been written on the RMD. The directive has the general form

U,lun,key,file

where

- lun** is the number or name of the logical unit where space to be released is located in the protection code
- key** is the protection code required for addressing lun.
- file** is the name of the file where the unused space is located.

Example: Release unused space located in file COBINT, on partition 22, protection code X.

U,22,X,COBINT

21.12 EXIT DIRECTIVE

This directive exits from FMUTIL. The directive has the general form

E

where

E keyword specifying EXIT from
FMUTIL

Example: Exit from FMUTIL

E

SECTION 22 COMPRESSION/EDIT SYSTEM (COMSY)

COMSY is a source record COMpression and edit SYstem. It is a background task that constructs files of programs in a compressed format for later updating and decompression. It has provision for maintaining these files as sequential files on magnetic tape and RMD or as random accessed files on RMD.

Figure 22-1 is a block diagram of the general data flow through COMSY.

22.1 ORGANIZATION

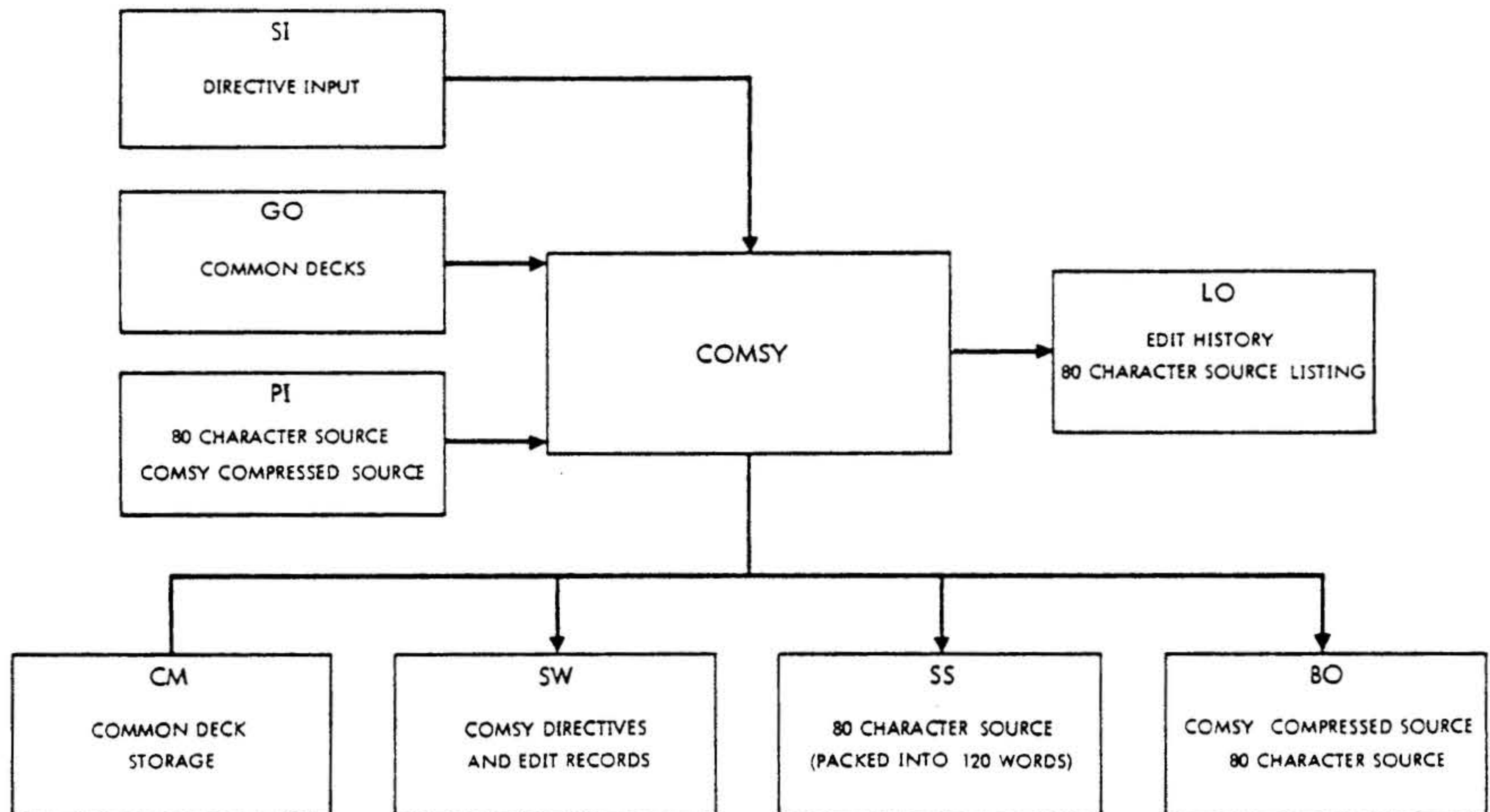
COMSY is scheduled by the job-control processor (JCP) directive/COMSY. Once activated, COMSY inputs and executes directives from the SI logical unit. COMSY directives specify both the action to be taken and the logical units and files to be used.

22.1.1 COMSY Compression

COMSY compresses 80 character ASCII records into modules called decks. A COMSY deck consists of an ASCII deck identification record and any number of 60-word binary records. The deck identification record is described in section 22.3.15.

COMSY binary records consist of a sequence count in word 0, a checksum in word 1, and compressed ASCII text in words 2 through 59. The last record of a deck contains its sequence number as a negative number. The checksum is a value which is obtained by summing the 116 8 bit bytes contained in words 2 through 59 in an unpacked array with each byte right justified in a word with the remainder of the word zeros.

COMSY compresses the ASCII text by reducing two or more imbedded blanks to a two character sequence the first of



VTII-3537

Figure 22-1. COMSY Data Flow

COMPRESSION/EDIT SYSTEM (COMSY)

which is an ASCII NUL character (200) and the second of which is the count of the number of blanks imbedded minus two. During compression, characters 73 through 80 are ignored and any trailing blanks are dropped and replaced by an end of record character represented by an ASCII rub-out (377). The last compressed record of a deck is followed by an ASCII EOT character (204).

22.1.2 Sequential Files

A sequential COMSY file is a file of COMSY decks which contain unpacked records. The last deck is followed by a .FILE directive and an end-of-file. Sequential files may be recorded on magnetic tape or RMD. Although COMSY will allow input of decks from a card reader, it is not programmed to consider the handling of files from cards.

22.1.3 Random Files

A random COMSY file is an RMD file which contains a deck directory and COMSY decks. COMSY decks are recorded in 120 word blocks of two 60 word records per block. This provides faster access to specific COMSY decks that can be obtained on a sequential file.

22.1.4 Common Files

A COMSY common file is a file containing up to nineteen decks which may be inserted into COMSY decks as updates. Each common file contains a directory which allows random accessing of the decks which it contains. Common decks may be entered into a common file by transferring an existing COMSY deck or by input of 80 character source records. Common decks are stored on an RMD in uncompressed form to allow for speedy insertion into other decks.

When initialized, COMSY assumes that the common file is assigned to unit CM with the default logical unit as lun 9 (GO File). A different common file may be utilized by assigning it to unit CM with a UNIT directive.

22.1.5 Sequence and Edition Numbers

During the compression of 80 character source records, COMSY truncates characters 73 through 80. Any identification or sequence numbers contained in these characters is lost. In order that different versions of the same deck may be identified, COMSY maintains a deck edition number. A decks edition number appears in its identification records.

Within a deck, COMSY identifies records by their relative positions in the deck. The first record has a sequence number of one, the second, two, etc. COMSY updating directives require the use of these sequence numbers to specify the location of insertions and deletions.

When 80 character source records are output, COMSY inserts the deck edition number in character positions 73 through 74 and the record sequence number in positions 75 through 80 of each record. When a new COMSY deck is not being output, the edition number used is the edition number of the input and the sequence numbers refer to each records position in the input deck. Inserted records are denoted by the insertion of asterisks in place of edition and sequence numbers. If a new COMSY deck is being output, the edition number used is the edition number of the new deck and the sequence numbers refer to each records position in the new deck. In this case, inserted records will have an edition and sequence number.

22.2 INPUT/OUTPUT

COMSY utilizes seven logical units, some of which are reassignable by use of the .ASSIGN and .UNIT directives. Table 22-1 contains the logical unit names, default assignments and usage.

Table 22-1. Default VORTEX

COMSY Name	LUN	Default Unit	Usage
SI	2	SI	Directive input. Source record input. COMSY deck input.
PI	4	PI	Source record input. COMSY deck input. COMSY file input.
BO	7	BO	Unblocked decompressed output. COMSY deck output. COMSY file output.
LO	5	LO	List output.
SS	8	SS	Block decompressed output.
SW	102	SW	Temporary update storage.
CM	9	GO	Common deck storage.

Note: SS, SW and CM must be on RMD.

22.3 COMSY DIRECTIVES

This section describes the COMSY directives:

- a. I/O assignment and option selection directives

ASSIGN Assign non-RMD logical unit

UNIT Assign and open RMD file (with rewind)

- SET** Set COMSY options
- GANG** Select and specify out of identification field

These directives are used to replace the default logical units assigned by COMSY with user specified logical units and to select user options.

b. Deck creation, copying and checking directives

- DECK** Build COMSY deck from source input
- COMDECK** Build a common deck
- COPY** Copy decks or files
- RANDOM** Build a random file
- APPEND** Append to a random file
- EDIT** Edit a random file
- LIST** List decknames in a file
- CHECK** Check sequence and checksums

These directives are used to create, copy and check the validity of COMSY decks and files.

c. Updating directives

- INSERT (ADD)** Record insertion
- REPLACE (DELETE)** Record deletion and replacement
- COMMON** Common deck insertion
- COMSY** Deck decompression

These directives are used to update an existing COMSY deck and to cause decompression when required. The updating directives INSERT, ADD, REPLACE, DELETE and COMMON must directly precede as a group the COMSY directive which specifies the deck to be processed. All other directives required to produce a desired result must precede the updating directives. Sequence numbers must always be in ascending order (note: Equal is not considered ascending).

d. End-of-file and exit directives

- FILE** Logical end-of-file
- END** Exit COMSY

These directives are used to specify a logical end-of-file and to exit from COMSY.

COMSY directives must begin with a period as the first character of the record and must contain no imbedded blanks. Directives are terminated at the first blank with the

exception of the .COMSY record which is the first record of a COMSY deck. Comments may appear after the terminating blank.

The general form of a COMSY directive is

`.name,p(1),p(2)...p(n)`

where

- name** is one of the directives names given above
- each p(n)** is a parameter defined below under the descriptions of the individual directives.

22.3.1 ASSIGN Directive

This directive specifies a logical unit assignment for a COMSY reassignable unit. This directive cannot be used for an RMD logical unit. It has the form

`.ASSIGN,unit,lun,R`

where

- unit** is the name of a COMSY reassignable unit. Allowable unit may be SI, PI, BO, LO, and SS.
- lun** is the two character name or the logical unit number of the VORTEX logical unit to be assigned.
- R** is the character R which along with its preceding comma is optional. If present, it indicates the unit is to be rewound prior to use.

If the result of the assignment is a reassignment of unit BO or the logical unit, lun, currently assigned to BO, COMSY checks to see if any COMSY output had been written on BO since the last assignment of BO. If so, a .FILE directive and an end-of-file are output to BO prior to making the assignment. Additionally, if the current assignment of BO is to an RMD file (see section 22.3.2), the file is closed with update.

If the logical unit, lun, being assigned is currently assigned to unit PI and the current assignment is to an RMD file, the file is closed without update prior to making the assignment.

Reassignment of a lun to the same unit as is currently assigned is permitted and should be used to rewind units when necessary.

Example: Assign M0 as PI and logical unit 25 as BO specifying rewind of BO.

`.ASSIGN,PI,M0`
`.ASSIGN,BO,25,R`

COMPRESSION/EDIT SYSTEM (COMSY)

22.3.2 UNIT Directive

This directive specifies a logical unit and file assignment for a COMSY reassignable unit on RMD. It has the form

`.UNIT,unit,lun,file,key`

where

- unit** is the name of a COMSY reassignable unit which may be assigned to RMD. Allowable units may be PI, BO or CM.
- lun** is the two character name or the logical unit number of the RMD partition containing the file to be assigned.
- file** is the name of the file to be assigned and opened.
- key** is the one character key for the assigned partition, lun. This parameter along with its preceding comma may be omitted when the partition does not require a key.

If the result of the assignment is a reassignment of unit BO or the logical unit, lun, currently assigned to BO, COMSY checks to see if any COMSY output had been written on BO since the last assignment of BO. If so, a .FILE directive and an end-of-file are output to BO prior to making the assignment. Additionally, if the current assignment of BO is to an RMD file, the file is closed with update.

Since COMSY compares only the partition logical unit numbers and ignores file names, it is not possible to have two files referenced with the same logical unit number. If this is required, an alternate logical unit number should be assigned to the partition outside of COMSY. The normal logical unit number is then used for one file and the alternate may be used for the other.

Reassignment of a lun and file to the same unit as is currently assigned is permitted and should be used to reposition to beginning of file when necessary.

Example: Assign file OFILE on logical unit 25, key equal X, to PI.

`.UNIT,PI,25,OFILE,X`

Assign files OFILE to PI and NFILE to BO. Both files are on logical unit 20. The partition has no key.

```
/ASSIGN,25,20
/COMSY
.UNIT,PI,20,OFILE
.UNIT,BO,25,NFILE
```

22.3.3 SET Directive

This directive is used to turn on selected user options. It has the general form

`.SET,P(1),P(2)...P(n)`

where

- each $P(i)$ is one of the parameters listed in the table below.

The appearance of a parameter in the list turns the selected option on. All options whose parameters do not appear in the list are turned off. Any options which are previously set on and which are to remain on must appear in the list. The resulting option setting remains in effect until another SET directive is encountered.

If no parameters appear, the standard default options as indicated in the table below will be set. The standard default options are automatically set when COMSY is initialized. The acceptable parameters are:

Parameter	Default	Option
A	on	Addition/deletion listing
C	on	Compile file output
E	off	End-of-file insertion
I	off	Input source records from PI
L	off	List output
N	off	New decks to be output
S	off	Source record output
Vn	off	VORTEX option switch
Y	off	Copy option

Examples: Set the options to input source records from PI and output a new COMSY deck.

`.SET,I,N`

Set the standard default options which are to output to the compile file and list all additions/deletions.

`.SET`

The following describes the characteristics of the SET options:

Addition/Deletion List Option (A)

When this option is turned on, all records which are added or deleted are listed on the LO unit. Records which are deleted are preceded on the line by *D*. Records which are added or inserted are preceded by *A*. The update directive which caused deletions and/or additions is listed preceding the deleted or added records.

Compile File Option (C)

When this option is turned on, it indicates that 80 character source records which are decompressed are to be output to SS for submission to FORTRAN or DASMR. Records are packed three to a sector. The last sector will be blank filled when necessary. Records from successive decompression of different COMSY decks may be concatenated on the compile file, however, the last RMD sector occupied by records from a deck may contain up to two trailing blank records. If each deck is a separate subprogram, the last record is an END. FORTRAN and DASMR will ignore the trailing blank records and begin processing with the first record of the next sector. COMSY closes the file with update after each deck insertion.

Source Record Output Option (S)

When this option is turned on, it indicates that 80 character source records which are decompressed are to be output a record at a time to unit BO.

End-of-file Option (E)

When this option is turned on and the source record output option is also on (S), an end-of-file is output on BO after the last record of each deck is output.

New Deck Option (N)

When this option is turned on, it indicates that a new COMSY deck is to be output to BO for each COMSY deck or source deck input. Any updates which are applied to a COMSY deck input will be included in the deck output. The new deck retains the same name and date of origination, however, the edition number is incremented by one and the date of last update is set to the current date.

Copy Option (Y)

When this option is turned on, it has the same effect as the new deck option (N), with the following addition. In addition to outputting a new deck for each COMSY deck or source deck processed, any COMSY decks which are passed over during a search for a specific COMSY deck will be copied without modification to BO.

Input Source from PI Option (I)

When a DECK directive is encountered, this option will cause COMSY to input source records from PI until a FILE directive or an end-of-file is encountered. When this option is turned off source records are input from SI.

List Output Option (L)

When this option is turned on, 80 character source records will be listed on unit LO.

VORTEX Switch Option (Vn)

This option is used to control the conditional assembly of programs. When the option is turned on, it causes COMSY to examine the first source record of each COMSY deck which is being decompressed. If the first record is a DASMR SET directive of the form:

1	8	16
VORTEX	SET	C

where

C is any character

The character in position 16 is replaced by character n. Character n may be omitted, in which case a 2 is placed in position 16.

Examples:

A DASMR SET directive of the following form appears as the first record of a deck.

VORTEX SET 1

A COMSY SET directive of the form:

.SET, V4,

would cause the above record to be changed to:

VORTEX SET 4

and a COMSY SET directive of the form:

.SET, V,

would cause the above record to be changed to:

VORTEX SET 2

Note: If the first record of a deck does not contain a SET directive in the form indicated above, the option has no effect on that deck.

22.3.4 GANG Directive

This directive specifies a three character identification code which is to be inserted into the identification field, character positions 73 through 75, of all 80 character source records which are output as a result of the source record output option (S) is set. The identification code

COMPRESSION/EDIT SYSTEM (COMSY)

replaces the deck edition number which is normally inserted in each record. The GANG directive has the general form

```
.GANG,xxx
```

where

xxx is any three ASCII characters, including blank.

If the parameter is omitted, the comma is absent, the normal edition number insertion made is reinstated. The GANG directive has no effect on other forms of COMSY output.

Example: Output a COMSY deck in 80 character source record mode, with the identification field set to COM.

```
.GANG,COM  
.SET,S
```

22.3.5 DECK Directive

This directive is used to specify the name of a deck and to direct COMSY to input 80 character source records from unit SI or PI. The form of the directive is

```
.DECK,deckname
```

where

deckname is a one to eight ASCII character name to be assigned to the deck.

If the input from PI option is on (I), input is from unit, PI; otherwise, input is from unit SI. Records are input until a FILE directive or an end-of-file condition is encountered on the input unit. Output created as a result of this directive is controlled by the on or off conditions of the user options as specified by the last SET directive encountered.

Example: Input source records from SI and output a new COMSY deck with the deckname SOURCE, listing the records on the printer.

```
.SET,N,L  
.DECK,SOURCE  
  n 80 character source  
  records  
.FILE
```

Example: Input source records from logical unit M0 and output a new COMSY deck with the deckname ALPHA on logical unit 25.

```
.ASSIGN,PI,M0  
.ASSIGN,BO,25
```

```
.SET,I,N  
.DECK,ALPHA
```

22.3.6 COMDECK Directive

This directive is used to specify the name of a common deck and to cause COMSY to transfer the deck to the common file (see section 22.1.4). A special form of the directive is used to open an existing common file. The directive has the form:

```
.COMDECK,deckname,S
```

where

deckname is a one to eight ASCII character name to be assigned to the deck.

S is the optional character S, which when present, causes COMSY to input 80 character source records from PI until a .FILE or end-of-file is encountered.

If no parameter appears, COMSY will input a COMSY deck from SI using the name of the COMSY deck input as the common deck name. If parameter S and its preceding comma are absent, COMSY will search the COMSY file on PI for the named COMSY deck and transfer it to the common file.

Common decks are transferred to a common file as uncompressed records packed three records per RMD sector. Decks may be added to an existing common file up to a maximum of nineteen decks, after which an error will be indicated. Common decks which are to be used in any one update of a COMSY deck must reside in the same common file as there is no provision for changing common file once an update is started.

Upon initialization, the common file is defaulted to the VORTEX GO file. The UNIT directive may be utilized to assign CM to a different user file. A special COMDECK directive of the form:

```
.COMDECK,*
```

will cause COMSY to open the file currently assigned to CM, assuming that the file already contains a directory and existing common decks. If this directive does not appear, COMSY will assume that the file assigned to CM does not contain valid common information. It should be noted that a common file which resides in the VORTEX GO file will not be retained between COMSY executions within the same job.

Example: Assign an existing common file, CMFILE, on logical unit 25 as the common file and add COMSY deck COMMON from the file on PI into it.

```
. UNIT, CM, 25, CMFILE
. COMDECK, *
. COMDECK, COMMON
```

Example: Input 80 character source records from PI and transfer them to the common file with a common deck named COMF.

```
. COMDECK, COMF, S
```

22.3.7 COPY Directive

This directive is used to copy COMSY decks on unit PI to a file on unit BO. It has the form

```
.COPY,first,last
```

where

first is the optional deckname of the first deck to be copied. COMSY will search the file on PI for the named deck. If the parameter is absent, COMSY will copy decks from the current position on PI.

last is the optional deckname of the last deck to be copied. If the parameter is absent, COMSY will copy decks until a FILE directive or end-of-file is encountered on PI.

Input to COPY is always in the form of COMSY decks. PI may be an existing random file in which case decks are copied in the order in which they were placed in the file. Output from COPY is controlled by the settings of the user selectable output options. If the new deck (N) or copy (Y) options are set, output is in the form of COMSY decks. Otherwise, output is in the form selected by the remaining user options (S,E, C and L). The output file is always sequential. Random files may be copied by utilizing the RANDOM directive.

Example: Copy decks from the current position on PI until the deck named ADECK is copied. Output is to be COMSY decks.

```
. SET, N
. COPY, , ADECK
```

Copy decks starting from deck FIRST to end-of-file

```
. COPY, FIRST
```

Copy the deck named MYDECK only

```
. COPY, MYDECK, MYDECK
```

22.3.8 RANDOM Directive

This directive is used to copy COMSY decks from an existing sequential or random COMSY file into a new random file. It has the form

```
.RANDOM,first,last
```

where

first is the optional deckname of the first deck to be copied. COMSY will search the file on PI for the named deck. If the parameter is absent, COMSY will copy decks from the current position on PI.

last is the optional deckname of the last deck to be copied. If the parameter is absent, COMSY will copy decks until a FILE directive or end-of-file is encountered on PI.

During the process of copying, COMSY constructs a directory which will allow random accessing of the file. The input file may be an existing random file. If so, any deleted decks (decks which contain the deckname *DELETED) will be omitted from the output file.

At the completion of the copy, the output file is closed and must be reopened for subsequent use. Once a random file is created, additions to the file are made by use of the APPEND directive.

Example Build a random file called RFILE on logical unit 22, key X, from a COMSY file on tape unit M0.

```
. ASSIGN, PI, M0
. UNIT, BO, 22, RFILE, X
. RANDOM
```

In the above example, include only those decks which are between and include DECKA and DECKZ.

```
. RANDOM, DECKA, DECKZ
```

22.3.9 APPEND Directive

This directive is used to copy COMSY decks from an existing sequential or random COMSY file into an existing random file. It has the form

```
.APPEND,first,last
```

COMPRESSION/EDIT SYSTEM (COMSY)

where

first is the optional deckname of the first deck to be copied. COMSY will search the file on PI for the named deck. If the parameter is absent, COMSY will copy decks from the current position on PI.

last is the optional deckname of the last deck to be copied. If the parameter is absent, COMSY will copy decks until a FILE directive or end-of-file is encountered on PI.

Decks which are copied are appended to the end of the existing random file. At the completion of the copy, the output file is closed and must be reopened for subsequent use.

Example: Append decks FDECK to and including LDECK from a tape on logical unit 18 to an existing random file, RFILE, on logical unit 22, key X.

```
.ASSIGN,PI,18,R
.UNIT,BO,22,RFILE,X
.APPEND,FDECK,LDECK
```

Append decks from FDECK until the end of the input file.

```
.APPEND,FDECK
```

Append all decks from current position to the end-of-file

```
.APPEND
```

22.3.10 EDIT Directive

This directive used to edit existing random files. It may be used to delete a deck, rename a deck, or change a deck edition number. It has the form

```
.EDIT,op,deckname,newname,newedition
```

where

op is DEL to delete deckname or REN to rename deckname and / or change the edition number

deckname is the current name of the effected deck.

newname is the optional new deckname for the specified deck.

newedition is the optional new edition number for the specified deck.

COMSY does not remove the entry for a deleted deck from the directory or is the deck removed from the file. The deckname is replaced by the name *DELETED in both the directory and in the deck's COMSY record. Deleted decks may be dropped by copying a random file to a new file utilizing the RANDOM directive.

Examples: Delete deck named ALPHA

```
.EDIT,DEL,ALPHA
```

Rename deck currently named as BETA with new name GAMMA and edition 01

```
.EDIT,REN,BETA,GAMMA,01
```

22.3.11 LIST Directive

This directive is used to list the deck information from the COMSY records of all the decks contained in the file assigned to PI. It has the form

```
.LIST
```

Listing starts from the current position on PI, with no rewind, and continues until a FILE directive or end-of-file is detected. The listing contains deck position, deckname, edition, original COMSY date, last update date, deck size, and file accumulated size. Deck size is the number of records in the deck including the initial COMSY record.

22.3.12 CHECK Directive

This directive is used to verify the contents of a COMSY file on PI to list the deck information from the COMSY records of the decks contained in the file. It has the form

```
.CHECK
```

Verification starts from the current position on PI, with no rewind, and continues until a FILE directive or end-of-file is detected. A listing is produced with the same format as that produced by the LIST directive. All records contained within the decks are checked for checksum and sequence. When an error is detected, an error message is output prior to the deck information for the erroneous deck.

22.3.13 INSERT (ADD) Directive

This directive is used to insert new records after the record with the sequence number specified. It has the form

```
.INSERT,seqno
```

or

```
.ADD,seqno
```


where

seqno is the sequence number (in the COMSY deck to be processed), of the record after which new records are to be inserted. Records directly following the INSERT directive are inserted until another directive is encountered. When an addition/deletion listing is checked (see SET directive) added records are preceded by *A*.

Example: Insert new records after record number 8 and after record number 15.

```
.INSERT, 8
  ABX=10
  ABY=ABX*25
.INSERT, 15
  READ(5,100)IBUF
.COMSY,HTEST
```

22.3.14 REPLACE (DELETE) Directive

This directive is used to delete old records and to optionally replace the deleted records with new records. It has the form

```
.REPLACE,first,/ast
```

or

```
.DELETE,first,/ast
```

where

first is the sequence number of the first record to be deleted.

last is the optional sequence number of the last record to be deleted. If omitted, only the record specified by *first* is deleted.

REPLACE and DELETE both operate in the same manner and are interchangeable. Records are first deleted from output and then replaced by new records which directly follow the directive until another directive is encountered. New records may be omitted to cause deletion only. When an addition/deletion listing is created (see SET directive), deleted records are preceded by *D* and added records by *A*.

Example: Replace records 15 through 19 with new records and delete record 24.

```
.REPLACE 15,19
  LDA      TEMP
  STAE     ALPHA
.DELETE,24
.COMSY      (process next
             deck from PI)
```

22.3.15 COMMON Directive

This directive is used to insert a deck from the common file after a specified input record, or in place of specified input records. It has the form

```
.COMMON,name,action,first,/ast
```

where

name is the name of the deck to be inserted from the common file currently open.

action is the directive INSERT or the directive REPLACE to specify designed action. ADD or DELETE are not acceptable.

first is the sequence number of the input record after which the common deck is to be inserted, or is the first record to be replaced.

last is the optional sequence number of the last record to be replaced. If omitted, only the record specified by *first* is replaced.

This directive essentially operates in the same manner as INSERT and REPLACE, with the exception that the new records are in the common file rather than following the directive.

Example: Place decks COM1 and COM2 in the common file and then insert COM1 at record 2 and replace records 25 through 32 with COM2.

```
.COMDECK,COM1
.COMDECK,COM2
.COMMON,COM1,INSERT,2
.REPLACE,16
  LDXI 5
.COMMON,COM2,REPLACE,25,32
.
.
.COMSY,PROGA
```

COMPRESSION/EDIT SYSTEM (COMSY)

22.3.16 COMSY Directive

This directive specifies the COMSY deck which is to be processed. Updates directly preceding the COMSY directive will be applied during processing. It has the following two forms

form 1: `.COMSY,deckname,newname,newedition`

or

form 2: `.COMSY,deckname,edition,odate,update`

where in form 1

deckname is the optional name of the deck to be processed

newname is the optional new name to be assigned to the deck if a new COMSY deck is to be output

newedition is the optional new edition number to be assigned to the deck if a new COMSY deck is to be output

where in form 2

deckname is the eight character deckname (including blanks)

edition is the two character edition number (00-99)

odate is the date the original COMSY edition was created

update is the date the deck was updated

When **deckname** is omitted in form (1), the deck to be processed is the next deck on unit PI. When **deckname** is present, it is the name of the deck to be searched for in the COMSY file assigned to PI. The input file may be in sequential or random format. During the search on a sequential file, decks passed over will be copied to BO if the copy option is selected (see SET directive).

The second form of the COMSY directive is the COMSY record which is output by COMSY as the first record of a COMSY deck. When a COMSY deck is to be processed from SI, form (1) must be omitted. Although COMSY originates form (2) it may be modified or replaced by the user; however, character positions within the record are fixed, therefore, field sizes must not be changed.

Example: Process deck ABLE changing its name to BAKER and edition to 09.

```
.COMSY,ABLE,BAKER,09
```

22.3.17 FILE Directive

This directive is used to specify a logical end-of-file. It may replace or be replaced by a physical end-of-file or a disc end-of-file. It has the form

```
.FILE
```

22.3.18 END Directive

This directive causes COMSY to exit. It may optionally specify that a series of JCP directives be executed. It has the form

```
.END,NN,directive
```

where

NN is an optional numeric parameter specifying the number of pages to be allocated by a JCP/MEM directive. If omitted, the preceding comma is omitted.

directive is optionally, any legal JCP directive without a slash (/).

Prior to exiting, a check is made to determine if any new COMSY decks have been output on BO. If so, a FILE directive and an end-of-file are output to BO. If BO is assigned to a file on RMD, the file is closed with update.

When the first parameter is present, COMSY performs the JCP/MEM function as if a /MEM,NN directive had been processed by JCP. If the second parameter is present, the parameter, preceded by a slash (/), is transferred into the JCP input buffer; and, if the compile file output option is on, the JCP functions

```
/ASSIGN,PI=SS,PO=DUM  
/PFILE,PI,,SS
```

are performed as if they had been input to JCP.

Example:

```
/COMSY  
.SET,N,C  
.DELETE,5  
.COMSY,PROG1  
.END,3,DASHR,B
```

The above directives will cause program PROG1 to be input from PI, record 5 will be deleted, a new COMSY deck will be

output to BO, assembler input will be blocked and output to SS, and COMSY will cause the following JCP functions to be performed:

```

/ASSIGN,PI=SS,PO=DUM
/PFILE,PI,,SS
/MEM,3
/DASMR,B
    
```

22.4 COMSY LOAD MODULE GENERATION

COMSY is normally executed as a priority 1 task from the VORTEX Background Library (BL). It may also be executed from an alternate library.

COMSY contains eighteen overlays. During load module generation, the error code LG16 will be output to SO for each overlay. This diagnostic is normal and should be ignored. COMSY requires 10K (20 pages) of memory for execution.

22.5 COMSY EXECUTION

COMSY is executed by input on the SI logical unit of a directive of the form

```
/COMSY
```

There are no parameters.

22.6 ERROR PROCESSING

When COMSY detects an error, a diagnostic message is printed on the LO logical unit and processing is terminated. COMSY exits by executing a FORTRAN STOP statement

which contains the error number of the error detected. FORTRAN displays the STOP statement in the form

```
COMSY STOP n
```

where n is the error number

22.7 COMSY EXAMPLE

This example shows how to convert VORTEX COMSY source into a file containing trailing blanks by removing the COMSY line numbers. Such a file is then more efficiently stored as a VIP text file.

Example:

Decompress the deck LOGOUT in the file C_{PSRCE} and use the SEDIT utility to remove the COMSY line numbers from columns 73-80 of each record. The stripped file is to be copied to logical unit 25 with the file name LOGOUT. FMUTIL is then to be used to release any unused space in the file.

```

/JOB
/COMSY
.SET,C,E
.UNIT,PI,189.CPSRCE
.COMSY,LOGOUT
/FMAIN
.CREATE,25,LOGOUT,120,5000
/SEdit
AS,IN,SS,,SS
AS,OU,25,LOGOUT
GA,(73,80),'b b b b b b b b'
FC
/FMUTIL
U,25,LOGOUT
/FINI
    
```


Upon completion of processing of a spool file status or spool file delete request, S\$SPC writes a prompt message to the operator console and waits for the next operator request to be entered. After initiation of communication to a spool print task, S\$SPC exits from the VORETX system.

23.4.1 EX (Spool Print Task Exit)

End of communication and exiting of either S\$SPC or a spool print task may be requested by use of the EX command.

The format of the EX command is:

EX

23.4.2 SG (Spool Print Task Go)

End of communication with a spool print task may be requested by use of SG, the spool print task go command. A spool print task upon recognition of the SG command, resumes processing at the point of interruption to communicate with the operator console.

The format of the SG command is:

SG.nn

where

nn is a two-digit number in the range 01-14. This number specifies the spool print task with which communication is desired.

Example: Initiate communication with the spool print task S\$SP01. The spool communication task S\$SPC, is to be scheduled at priority level 9. The task resides on the foreground library whose protection code is F.

```
;SCHED,S$SPC,9,FL,F
S$SPC,SP**
SB,01
S$SPC,SP05,01
```

After S\$SPC is scheduled, communication is initiated with the spool print task S\$SP01. The communication initiated message (S\$SPC, SP05,01) is written to the operator console and S\$SPC exits from the system.

23.5 INITIATION OF COMMUNICATION WITH A SPOOL PRINT TASK

After scheduling S\$SPC, the spool print break command SB is entered from the operator console to initiate

communication to a spool print task. If communication is initiated to the spool print task specified in the spool print break command, S\$SPC outputs a communication initiated message on the operator console and exits from the VORETX system. If the spool break command is not valid or communication cannot be initiated to the spool print task specified in the break command, S\$SPC outputs an error message on the operator console. In this case, if no further attempt to communicate with a spool print task is to be initiated, the spool communication exit command EX may be entered from the operator console to request S\$SPC to exit from the VORETX system.

The format of the SB command is:

SB,nn

23.6 SPOOL FILE STATUS DISPLAY

The status of spool files is obtained by use of the display status (DS) command. The status of the requested spool files is displayed on the operator console in the order of the spool files in the spool file directory. The format of the display status command is

DS { .ALL
.printclass.jobname }

where

ALL is a constant which specifies a display of the status of all spool files in the spool file directory.

printclass is a single alphabetic character (A through Z) used to relate individual spool files. This relationship or grouping together of individual spool files can be for any desired reason, but most often the grouping is based on the similarity of information in the grouped files.

jobname is a one- to eight-character job name; this parameter is applicable only to spool file output from background tasks.

Example: Obtain the status of all spool files within print class A

```
;SCHED,S$SPC,9FL,F
S$SPC,SP**
DS,A
```

MULTITASK SPOOLER

JOB NAME	TASK NAME	SPOOL NAME	PRINT CLASS	LUN NO.	DST STATUS	
JOBONE	TASKA	S00036	A	5	43	S\$SP01
JOBONE	TASKB	S00037	A	5	43	0
JOBONE	TASKC	S00038	A	5	43	S\$SPO

S\$SPC,SP05,04

After S\$SPC is scheduled, operator console display of the three spool files associated to class A is initiated. The display is followed by the status display completion message (S\$SPC,SP05,04)

A spool file may have one of four possible status settings

Status	Indication Status Display
open for write	S\$SPO
available for print	0
open for print	S\$SPnn
printed and held	nnnnn

where

S\$SPO	is the name of the spool output task
0	indicates that the spool file is complete and has not been printed
S\$SPnn	is the name of the spool print task and nn is a two digit number in the range 01-14
nnnnn	is a count of the number of times a file has been printed using the PD command.

23.7 DD (SPOOL FILE DELETE)

A Multitask Spool System command for deleting spool files is available. The spool file delete command, DD, removes the specified spool file from the spool file directory and deletes the specified spool file from the directory of the RMD partition on which the spool file resides. A spool file to be deleted must have a status of closed or held.

The format of the DD command is:

DD, print class, jobname, filename

where

print class	is one alphabetic character (A through Z)
jobname	is a one- to eight-character VORTEX jobname
filename	is a 6-character spool file name

↓
Example: Delete the closed spool file named S00038 in print class A.

```
;SCHED,S$SPC,9,FL,F
S$SPC,SP**
DD,A,,S00038
S$SPC,SP05,05
```

↑
 After S\$SPC is scheduled, the spool file S00038 is removed from the spool file directory, deleted from the RMD partition directory on which it resided and the delete complete message (S\$SPC,SP05,05) is written to the operator console.

23.8 PA, PH, PD (SPOOL FILE PRINT)

Three Multitask Spool system commands are available to print spool files. The three commands are:

Command Code	Description
PA	Spool File Print and Align - If the first record of the specified spool file is a forms control record, it is displayed on the operator console then the number of records specified in the PA command are written to the line printer specified in the PA command. Specifying the line printer lun as a dummy lun allows a forms control record to be displayed with no printing.
PH	Spool Print and Hold - Executes the print request and does not delete the spool file after completion of print. If the first record of a spool file is a forms control record, it is displayed on the operator console. The spool records are then written to the line printer specified in the PH command. If the print is of spool files in a print class and all spool files in the print class are printed, then the print task waits for a time interval to lapse before searching the spool file directory for additional spool files to print.
PD	Spool Print and Delete - Executes the print request and deletes each spool file after completion of print. As with PA and PH, a forms control record, if present for a spool file, is displayed on the operator console.

The format of the PA,PH, and PD commands is

$\left. \begin{array}{c} \text{PA} \\ \text{PH} \\ \text{PD} \end{array} \right\} .\text{printer lun,print class,filename,record number}$

where

- printer lun** is a one- to three-digit logical unit number assigned to a printer or dummy device.
- print class** is one alphabetic character (A through Z).
- filename** is a 6-character spool file name.
- record number** is a one- to nine-digit number. For the PA command this parameter is required and it specifies the number of records to be printed. For the PH and PD commands the parameter may be included only if a spool file name ('filename') has been specified; in this case, the record number parameter specifies the record number within the specified spool file at which printing is to begin.

Example 1: SPOOL FILE PRINT AND ALIGN - The spool print task S\$SP01 is to print the first 10 records of the spool file S00037 from the print class A. Printing is on the line printer to which the logical unit number 15 is assigned.

```

;SCHED,S$SPC,9,FL,F
S$SPC,SP**
SB,01
S$SPC,SP05,01
S$SP01,SP**
PA,15,A,S00037,11
S$SP02,SP05,06
    
```

The first record of the spool file S00037, if it is a forms control record, is displayed on the operator console. The first 10 print records of the spool file are written to the line printer to which the lun 15 is assigned and the print and align complete message (S\$SP01,SP05,06) is written to the operator console.

Example 2: SPOOL PRINT AND HOLD - The spool print task S\$SP01 is to print all spool files generated for print class B. Printing is on the line printer to which the logical unit number 15 is assigned.

```

;SCHED,S$SPC,9,FL,F
S$SPC,SP**
SB,01
S$SPC,SP05,01
S$SP01,SP**
PH,15,B
    
```

After S\$SPC is scheduled and communication is established with the spool point task S\$SP01, each spool file in print class B that is available for printing (closed) is written to the line printer to which lun 15 is assigned. When printing of all spool files in print class B is complete, the spool print task S\$SP01 waits for a time interval to elapse and then searches the spool file directory for additional print class B spool files to print.

23.9 SC (Spool Print Cancellation)

The SC command directs the system to cancel the spool print task currently in process. When the spool print task is cancelled, the spool file record number of the last printed line is displayed on the operator console.

The format of the SC command is

SC

Example: Cancel the spool print in progress of print class A by the spool print task S\$SP01.

```

;SCHED,S$SPC,9,FL,F
S$SPC,SP**
SB,01
S$SPC,SP05,01
S$SP01,SP**
SC
S00118,0000000047
S$SP01,SP05,02
S$SP01,SP**
    
```

The spool print in progress by the spool print task S\$SP01 of spool file S00118 within print class A is cancelled after the printing of record number 47. The spool print task S\$SP01 is now waiting for a command to be entered by the console operator as indicated by the spool print prompt message (S\$SP01,SP**).

23.10 ST (Spool Print Termination)

A spool print in progress is terminated by use of the ST command. When the ST command is entered, the spool file print continues to spool to file end. If no spool file print is in progress when the ST command is entered, the spool print task outputs a prompt message to the operator console. The format of the ST command is

ST

MULTITASK SPOOLER

Example: Terminate the spool file print currently in progress by the print task S\$SP01

```
;SCHED,S$SPC,9,FL,F
S$SPC,SP**
SB,01
S$SPC,SP05
S$SP01,SP**
ST
```

After S\$SPC is scheduled and communication is established with print task S\$SP01, the ST command is entered. The spool print task S\$SP01 completes the spool file print in progress, then waits for initiation of communication via S\$SPC.

23.11 SPOOL FILE ALLOCATION

Spool file records are 80 words in length. Word 1-66 of a spool file record contain forms control or print and words 67-80 contain spool file control information.

Spool files are created only on RMD partitions on the disk pack mounted on the drive to which the spool directory logical unit number 107 is assigned.

Spool files are created with a primary space allocation as specified in the SPOPN subroutine call or with the default space allocation of 100 sectors. When the primary space allocation of a spool file is exhausted, the spool file is extended by the secondary space allocation specified in the SPOPN subroutine call or by the default space allocation of 100 sectors.

The search for space to allocate a spool file begins on the disk partition on which the spool file directory resides. The search for space continues on each succeeding partition until the end of the disk or until either the required amount of space is located, or space allocation has been unsuccessfully attempted on each partition available. Note that partitions with a protection key are not used for allocation of spool files. If the space allocation search was not successful and the search was for an amount of requested space greater than the default space allocation, then a search to allocate the default space amount is performed in the same manner as the previous search for space.

23.12 COMPONENT DESCRIPTION

The VORTEX II Multitask Spool system consists of six components:

- VZSDA - spool dummy driver; nucleus, map 0, resident task
- SPOPN - spool file open subroutine. Resides in object module library

- SPCLS - spool file close subroutine. Resides in object module library
- S\$SPO - spool output task. Foreground task with a resident TIDB
- S\$SPC - spool print communication task. Foreground task with a non-resident TIDB
- S\$SP01 - spool print task. Foreground task with a non-resident TIDB

VZSDA, the spool dummy driver, accepts open (spool file create), write, and close requests from either background or foreground tasks for logical units assigned to VZSDA. These requests are passed by VZSDA to S\$SPO, the spool output task. VZSDA, upon indication by S\$SPO of request processing completion, posts an I/O completion status to the initiating task.

A call to SPOPN, the spool file open subroutine, requests the creation of a spool file. In addition, SPOPN performs the following functions: specifies spool file disk space allocation, specifies the logical unit used in the requesting task's WRITE statement which directs output to a spool file, and specifies a print class to be associated with the spool file.

A print class is a single alphabetic character (A through Z) used to relate individual spool files. This relationship or grouping together of individual spool files may be for any desired reason but most often is based on the importance or similarity of information in the associated spool files. All spool files associated with a given print class may be printed by the computer operator by means of a print request that designates the print class to be printed and the line printer lun to which the print output is to be directed. This facility allows printing of spool files with no knowledge of either the names of tasks that generated the information in the spool files or names of the spool files associated to a print class.

A spool file may also be created without the use of the SPOPN call. Upon each WRITE from a background or foreground task to a logical unit assigned to VZSDA, the spool output task, S\$SPO, checks to see if a spool file has been created (opened) for both the task issuing the WRITE and the logical unit number to which the WRITE is directed. If a spool file is not found to be open, then the spool output task creates a spool file with a default disk space allocation of 100 lines and a default print class of A for a background task and B for a foreground task.

A call to SPCLS, the spool file close subroutine, is a request to close a spool file for the task issuing the SPCLS call and for the logical unit number specified in the call. A spool file may also be closed without use of the SPCLS call. At least once every 500 milliseconds S\$SPO, the spool output task, scans a table containing an entry for each open spool file to determine if the task that requested opening of the spool file

remains active within the system. Open spool files for which the associated task is not active are closed and removed from the table of open spool files.

S\$SPO, the spool output task, processes spool file created (open), write, and close requests from VZSDA, the spool dummy driver. S\$SPO also performs spool file close processing based on a core table of open spool files as previously described in the discussion of the SPCLS subroutine. After processing an open, write or close request, S\$SPO informs VZSDA of the result of request processing and waits for either another request to process from VZSDA or a time interval to lapse before resuming execution.

S\$SPC, the spool print communication task, is scheduled by the computer operator for spool file status display, spool file deletion or to initiate communication with a spool print task.

The Multitask Spool system allows from 1 to 14 spool print tasks to be concurrently active within a system. Spool print tasks are named S\$SP01 through S\$SP14. One spool print task, S\$SP01, is assigned at system generation time. Other spool print tasks may be created using the procedures described in this manual.

A spool print task is scheduled by entry of the SCHED directive from the operator console. Once a spool print task is scheduled, the task remains active until terminated by entry of the EX, spool print task exit command, from the operator console. Use of the ABORT directive to terminate a spool print task may result in unpredictable M\$SPOOL system operation.

The following Multitask Spool system actions are controlled by a spool print task and are available from each scheduled spool print task.

- Spool File Print
- Spool File Print Cancellation
- Spool file Print Termination
- Spool Print Task Exit

When a spool print task is scheduled or when a scheduled spool print task recognizes a communication request from S\$SPC, the spool print communication task, it writes a prompt message to the operator console and waits for an operator entry. If the operator entry is a valid command for a spool print task then the action directed by the entered command is taken otherwise an error message and then the prompt message are written to the operator console and the spool print task again waits for an operator entry.

23.13 SYSTEM GENERATION

To incorporate the Multitask Spool sub-system into a VORTEX II system, include the following EQP directive at SYSGEN time:

```
EQP,SD0A,0,1,0,0
```

This will incorporate the dummy driver front-end to spool. Include from 1 to 15 ASN directives for each spool logical unit desired. For example, if it is desired to incorporate three spool channels using logical unit numbers 6, 40 and 41, include:

```
ASN,6=SD00
ASN,40=SD00
ASN,41=SD00
```

Include an ASN directive for logical unit number 107; it must be assigned to an **unprotected RMD partition**. For example,

```
ASN,107=D00J
```

Include an ASN directive for logical unit numbers 108 and 109; they must be assigned to a dummy device as follows:

```
ASN,108=DUM
ASN,109=DUM
```

Include an ASN directive for logical unit numbers 110 through 123 as needed. 110 is used by print task number one (S\$SP01), 111 is used by print task number two (S\$SP02), etc. They must be assigned to a dummy device. For example, for a system using spool print tasks S\$SP01, S\$SP02 and S\$SP04, include:

```
ASN,110=DUM
ASN,111=DUM
ASN,112=DUM
```

Spool print tasks in addition to S\$SP01 are obtained by use of the File Maintenance ENTER directive. For example, to reference also the spool print task S\$SP01 as S\$SP02 and S\$SP04 from the foreground library with logical unit number 106 and protection code F, use the File Maintenance ENTER directive as follows:

```
ENTER,106,F,S$SP01,S$SP02
ENTER,106,F,S$SP01,S$SP04
```

CONSOLE LOGGING PROGRAM

25.4.2 OFF Command

The OFF command terminates console logging. By including one of the optional parameters, LOG1, LOG2, or ALL, the operator may also cause the specified file to be printed. Console logging may be activated at any subsequent time by rescheduling CONLOG, via the SCHED command, and entering the ON command. If the operator does not want any printout, he enters only the command OFF. The files may be printed at a later time by rescheduling CONLOG with the SCHED macro, and then entering the print command.

The general form of the OFF command is:

$$\text{OFF } \left\{ \begin{array}{l} \text{,LOG1} \\ \text{,LOG2} \\ \text{,ALL} \end{array} \right\}$$

Example: Terminate console logging and print both LOG1 and LOG2

OFF,ALL

25.4.3 PRINT Command

The PRINT command outputs the specified file to the list output (LO) device (lun = 5). LOG1, LOG2 or ALL may be specified. If the optional parameter is not included the default is ALL.

The log files printed with the PRINT command are initialized (rewound) after printing unless the optional parameter ,S is included.

The PRINT command has the format

$$\text{PRINT } \left\{ \begin{array}{l} \text{,LOG1} \\ \text{,LOG2} \\ \text{,ALL} \end{array} \right\} \text{,S}$$

Example: Print file LOG1. Initialize LOG1 after printing.

PRINT,LOG1

25.4.4 SWITCH Command

The SWITCH command causes the CONLOG program to output information the alternate device. If console messages were being placed on LOG1 and a SWITCH command entered, the console logging would then commence output to LOG2 to LOG2 and vice versa.

The form of the SWITCH command is:

SWITCH

25.4.5 HELP Command

The HELP command is included as an aid to the operator in reminding him of the commands and their format. If the HELP command is entered after CONLOG has been scheduled and the operator is still in the conversational mode with CONLOG, each of the commands, along with its optional parameters, will be output to the OPCOM device.

The form of the HELP command is:

HELP

25.4.6 EXIT Command

The EXIT command causes CONLOG to leave the conversational mode. The operator enters an EXIT command at the point when all the required control commands have been issued. After this command is issued, CONLOG will still log or print messages as specified by the operator, but will no longer expect additional commands from the operator.

The form of the EXIT command is:

EXIT

25.4.7 STATUS Command

The STATUS command causes the CONLOG program to output a status list in the form:

```
CONSOLE LOGGING ON/OFF
PRINT MODE AUTO/MAN
CURRENT LOGFILE LOG1/LOG2
LOGFILE LOG1/LOG2 FULL
```

The form of the STATUS command is:

STATUS

↓

25.5 STATUS WORDS

The status word CLSTAT is maintained by both the teletypewriter device and the CONLOG program. The conditions indicated by each bit of this word are described in Table 25-1.

The status word CLSTA1 is reserved for future use.

↑

Table 25.1. Status Word (CLSTAT)

bit	
0	0 = console logging inactive; 1 = console logging active
1	0 = printing must be requested; 1 = files automatically printed when full
2	0 = first half of buffer to be filled; 1 = second half of buffer to be filled
3	0 = do not initialize files; 1 = initialize files
4	0 = LOG1 is current logging file; 1 = LOG2 is current logging file
5	0 = LOG1 not full; 1 = LOG1 is full
6	0 = LOG2 not full; 1 = LOG2 is full
7	0 = do not print LOG1; 1 = print LOG1
8	0 = do not print LOG2; 1 = print LOG2
9	1 = Disk I/O error in TTYLOG
10	1 = No memory available for allocating 123-word buffer
11	1 = Logging terminated by files full condition or OFF command (deallocate 123-word buffer)
12	1 = TTYLOG has called CONLOG via the SCHED macro
13	1 = Open file 1
14	1 = Open file 2
15	1 = CONLOG scheduled by operator (the busy flag)

25.6 OUTPUT FORMAT

Each message will be preceded by the time of day in HH:MM:SS format. In addition, the date, logical unit number of the sending/receiving device, and the name of the corresponding task will be appended to the message. Output consist of 120 word blocks of data. Each block of data will thus contain two messages. At the time the output goes to the logging device, a three word block of all ones is added to the buffer and a 123 word WRITE command is issued. This results in a 120 word record being written to one sector and a three word record of negative ones being written to the succeeding sector. The three word record serves as an end of file indicator and as a current record location. When coming back up after a system crash, the program can search for this last record written indication and continue output at that location in that file. Each time a new output occurs, the previous three word record is written over. This action is performed until the buffer becomes full. The output format is as follows:

Buffer	
Word	Contents
1	HH
2	:M
3	M:
4	SS
5	blanks
6	message
...	...
...	...
...	...
45	...
46	blanks
47	mm
48	/d
49	d/
50	YY
51	blanks
52	lun
53	...
54	blanks
55	task
56	name
57	...
58	unused
59	...
60	...

61	HH
62	:M
63	M:
64	SS

CONSOLE LOGGING PROGRAM

65	message
.	...
.	...
.	...
105	...
106	blanks
107	mm
108	/d
109	d/
110	yy
111	blanks
112	lun
113	...
114	blanks
115	task
116	name
117	...
118	unused
119	...
120	...

121	-1
122	-1
123	-1

↓ 25.7 TELETYPEWRITER DRIVER (V\$TYB) INTERFACE

If logging is active and a message is sent from hardware device address 1, but not to logical unit number 5, V\$TYB will move the message to the TTYLOG buffer and activate TTYLOG.

The ;SCHED command sets the A register to 0143152 before the requested program is called, so CONLOG uses the A register as an indicator which shows whether it was called from TTYLOG or from the operator SCHED request. Only one copy of CONLOG can be scheduled by the operator.

25.8 TTYLOG

TTYLOG collects messages two at a time and then writes them to the log file. If a log file becomes full or if an error occurs, TTYLOG will schedule CONLOG to print an appropriate message.

↑

APPENDIX A

ERROR MESSAGES

This appendix comprises a directory of VORTEX operating system error messages, arranged by VORTEX component.

A.1 ERROR MESSAGE INDEX

Except for the language processors (section 5), VORTEX error messages each begin with two letters that indicate the corresponding component:

Messages beginning with:	Are from component:	Listed in subsections:
CN	Concordance program	A.5.3
DG	Debugging program	A.7
DM	DSYSTEM/DSPMEM	A.7
DP	Dataplot II	A.12
EX	Real-time executive	A.2
FH	Disk Formatter	A.18.1
FM	File maintenance	A.9
IO	I/O control	A.3
IU	I/O utility	A.10
JC	Job-control processor	A.4

LG	Load-module generator	A.6
MS	Microprogram simulator	A.20.2
MU	Microprogram utility	A.20.3
NC	VTAM Network control	A.21
OC	Operator communication	A.17
PT	Patch Program	A.15
RL	RELINK	A.6.1
RP	RPG IV Compiler	A.5.3
RZ	RMD Init and Analysis	A.18
SE	Source editor	A.8
SM	System maintenance	A.16
ST	VSORT	A.11
*	DAS MR assembler	A.5.1

Section A.25 gives explanations of error codes listed under "Possible User Action" in the last column of the following sections.

Error messages from the following components do not have identifying prefixes.

Component	Subsection
COMSY-Compression/Edit	A.23
FMUTIL-File Maintenance Utility	A.22
Parity Errors	A.24

A.2 REAL-TIME EXECUTIVE

Message	Condition	Action	Possible User Action
EX01,xxxxxx	Invalid RTE service request by task xxxxxx	Abort task xxxxxx	D01,D02,P01
EX02,xxxxxx	Scheduled task xxxxxx name not in specified load-module library	Abort task xxxxxx	D01,D03
EX03,xxxxxx	Task xxxxxx made RESUME request but requested task not found	Continue scheduling task	D01,D03
EX04,xxxxxx	Task xxxxxx made ABORT request but requested task not found	Task xxxxxx continues	D01,D03
EX05,xxxxxx	Background task xxxxxx larger than allocatable	Task xxxxxx not loaded	M01,M02,M03 M04,P02
EX06,xxxxxx	Not enough allocatable space available for ALOC request	Abort task xxxxxx	M06
EX07,xxxxxx	OVLAY requests a segment not in library	Abort task xxxxxx	D01,D03

ERROR MESSAGES

EX10,xxxxxx	Scheduled request has a library task priority conflict (task priority 0 from foreground library, task priority 2 from background library). Scheduled request specifies a foreground task to be executed at priority 0 or 1	Schedule request ignored, scheduling task continues	D04,D02,P01
EX11,xxxxxx,n	Memory protection violation at address n, math firmware generated (i.e., attempt to divide by zero)	Abort task xxxxxx	P03
EX12,xxxxxx	I/O link error (foreground task making request, or incorrect logical unit number)	Abort task xxxxxx	P01
EX13,xxxxxx	Attempted to load map registers and a sense-DMA-error stop condition occurred	Abort task xxxxxx	H05
EX14,xxxxxx	Lack allocable TIDB memory space for task xxxxxx attempted to be scheduled	If an OPCOM request, OPCOM is aborted. If the schedule is not an OPCOM, the request is reattempted	M02
EX15,xxxxxx	Foreground common specified by background task	Abort task xxxxxx	P01
EX16,xxxxxx	PASS macro specified zero or negative word count	Abort task xxxxxx	P01
EX17,xxxxxx	RMD I/O error detected when SAL attempted to load scheduled task, xxxxxx. Also pseudo TIDB data assumed bad, execution address less than 01000	Abort task xxxxxx	H06,P01
EX20,xxxxxx,n,p 1 2	Map memory-protection HALT violation at virtual address n in task xxxxxx	Abort task xxxxxx	P17

Note: xxxxxx is the name of a task.

EX21,xxxxxx,p 1 2	Map memory-protection I/O violation at virtual address n in task xxxxxx. User attempted to execute I/O command in a map other than map 0	Abort task xxxxxx	P17
EX22,xxxxxx,n,p 1 2	Map memory-protection WRITE violation at virtual address n in task xxxxxx. User attempted to write/store into read-only or read-operand-only location	Abort task xxxxxx	P17
EX23,xxxxxx,n,m,p 1 2	Map memory-protection JUMP violation at virtual address n in task xxxxxx. User attempted to jump into read-operand-only location m + 2	Abort task xxxxxx	P17
EX24,xxxxxx,n,m,p 1 2	Map memory-protection UNASSIGNED violation at virtual address n in task xxxxxx. User attempted to read or write into unassigned location m	Abort task xxxxxx	P17
EX25,xxxxxx,n,p 1 2	Map memory-protection instruction-fetch violation at virtual address n in task xxxxxx. User attempted to fetch an instruction from read-operand-only location	Abort task xxxxxx	
EX26,xxxxxx,m 1	Firmware floating point or stack overflow or underflow occurred at logical address or in task xxxxxx.	Task is continued at location n + 2	None
EX27,xxxxxx	ALOCPG request error. Parameter error or pages not available for allocation.	Program continues execution at specified reject address	P01

¹ The instruction which generated the memory-protection violation and the contents of the A, B, and X (and V75) registers are also posted. Note that on non-micro-VORTEX, if errors occur in two tasks almost simultaneously, the register values for the first task may incorrectly be the same as those for the second task.

² Where p is the physical page associated with the logical address of the instruction.

ERROR MESSAGES

EX30,xxxxxx	DEALPG request error. Parameter error. Program continues execution at specified reject address	Program continues execution at specified reject address	P01
EX31,xxxxxx	MAPIN request error.	Program continues execution at specified reject address	P01
EX32,xxxxxx	Attempted to schedule a task from a non-RMD unit	Directive ignored	D02,P01
EX33,xxxxxx	Floating-point processor, FPP, error For V77-800: indicates arithmetic overflow	Program continues at the address following the FPP store instruction	None
EX34,xxxxxx	Floating-point processor, FPP, timeout	Program continues at interrupted instruction	None
EX35,xxxxxx,n,m	Memory parity error. n is the P register. m is the tracking register. For V77-800: indicates arithmetic underflow		
EX36,xxxxxx	REALPG request error.	Program continues execution at specified reject address	P01
EX37,xxxxxx	Attempt to schedule a file that is not a load module. Error during RTE SELECT request.	Task not scheduled	P01
EX40,xxxxxx	Attempt to schedule a task with TRACE while TRACE is already active.	Task not scheduled	P01

Note: xxxxxx is the name of a task.

A.3 I/O CONTROL

A device status code word is displayed following the task name for some I/O messages. For information about the meaning of this 16-bit word, see Appendix H.

IO00,xxxxxx	Unit not ready, or unit file protected.	Repeats message until condition is corrected	H01,H03,H07	↓
IO01,xxxxxx	Device declared down	Repeats message until condition is corrected	H04,D19	↑
IO02,xxxxxx	Invalid LUN specified	Abort task or request	D02,P01	
IO03,xxxxxx	FCB/DCB parameter error	Abort task or request	P04	
IO04,xxxxxx	Invalid protection code	Abort task or request	D01,D02,P01	
IO05,xxxxxx	Request block, FCB, or buffer extends into unassigned memory.	Abort task or request	P01	↓
IO06,xxxxxx	I/O request error, e.g., I/O-complete bit not set, prior request may be queued	Abort task or request	P01	↑
IO07,xxxxxx	Attempt to read from a write-only device, or vice versa	Abort task or request	D02,P01	
IO10,xxxxxx	File name specified in OPEN or CLOSE not found	Abort task or request	D01,D03,P01,D29	
IO11,xxxxxx	Invalid file extent, record number, address or skip parameter, file already closed	Abort task or request	P04,P01	
IO12,xxxxxx	RMD OPEN/CLOSE error, or bad directory thread, seek or read error on OPEN request.	Abort task or request	H05,D03	

ERROR MESSAGES

IO13,xxxxxx	Level 0 program read a JCP (/) directive	Task xxxxxx is aborted.	None
		directive passed to JCP buffer	
IO14,xxxxxx	Interrupt timed out or no cylinder-search-complete interrupt	Abort task or request	H05,D05
IO15,xxxxxx	Disc cylinder-search or malfunction error	Abort task or request	H05
IO16,xxxxxx	Disc read/write timing error	Abort task or request	H05
IO17,xxxxxx	Disc end-of-track error	Abort task or request	H05
IO20,xxxx	BIC: abnormal stop, not ready, or time out error on device xxxx	Abort task or request	D05,H05
IO21,xxxxxxx	Memory parity error	Abort task or request	H05
IO22,xxxxxxx	Map error	Abort task or request	H05
IO23,xxxxxxx	Memory timeout error	Abort task or request	H05
IO24,xxxxxxx	Rate error	Abort task or request	H05
IO25,xxxxxxx	Memory/data bus verification error	Abort task or request	H05
IO26,xxxxxxx	Memory access error	Abort task or request	H05
IO27,xxxxxxx	Alternate sector partition full	Abort task or request	H05
IO30,xxxxxx	Parity error	Abort task or request	H05,D02
IO31,xxxxxx	Reader or tape error	Abort task or request	H05,P19

Note: xxxxxx is the name of a task or device.

ERROR MESSAGES

I032,xxxxxx	Odd-length record error	Abort task or request	H05,P12
I033,xxxxxx	Invalid terminal identifier or logical line number	Request ignored	D27
I034,xxxxxx	Line or terminal not opened	Request ignored	D28
I035,xxxxxx	Line or terminal down	Request ignored	D28
I036,xxxxxx	Line or terminal already open	Request ignored	D28
I037,xxxxxx	Request still pending	Request ignored	None
I040,xxxxxx	Action on terminal not opened	Request ignored	D28
I042,xxxxxx	Invalid physical line address	Request ignored	D27
I043,xxxxxx	Invalid TCM type	Request ignored	D27
I044,xxxxxx	No temporary storage available	Request ignored	None

Note: xxxxxx is the name of a task or device.



ERROR MESSAGES

I045,xxxxxx	RMD error. Format, end-of-file or head selection error	Abort task or request	H05,D13
I046,xxxxxx	Map memory protection I/O data transfer error	Abort task or request	H05
I047,xxxxxx	User write specified word count > 73	Record is truncated	P04
I050,xxxxxx	RMD read error	Data is used	H08 ←
I05x,xxxxxx	RMD read error on spool stream X. Specified stream is last digit of error number	The data is used	H06
I060,xxxxxx	RMD file full	The program waits until space is available on the file. The message is repeated every 200 times the condition occurs	D08
I061,xxxxxx	User parameter error in request	Request is ignored	P01
I062,xxxxxx	RMD write error	The bad sector is skipped. This is likely to cause an I05x error later, but no data will be lost	H06
I063,xxxxxx	Buffer unavailable for spooler	Spooler waits until buffer is available	None
I065,xxxxxx	Alternate sector directory record read error	Abort task or request	H05
I066,xxxxxx	Alternate sector directory record write error	Abort task or request	H05

Note: xxxxxx is the name of a task or device.

ERROR MESSAGES

A.4 JOB-CONTROL PROCESSOR

Message	Condition	Action	Possible User Action
JC01	Invalid JCP directive	Ignore directive	D01,D02
JC02	Invalid or missing parameter in a JCP directive; or illegal separator or terminator	Ignore directive	D01,D02
JC03	Specified physical device cannot perform the functions of the assigned logical unit	Ignore directive	D07,H06
JC04	Invalid protection code or file name in a JCP directive	Ignore directive	D01,D02
JC05,nn	End of tape before the number of files specified by an /SFILE directive has been skipped; or end of tape, beginning of tape, or file mark before the number of records specified by an /SREC directive has been skipped where nn is the number of files (or records) remaining to be skipped	SFILE, SREC terminates upon error condition	P07
JC06	An irrecoverable I/O error while compiling or assembling; or an error during a load/go operation; or insufficient symbol table memory (insufficient /MEM directive), or an EOF was encountered before an END statement	Job flushed to next /JOB directive	P07,M01,P06
JC07	Invalid or illegal logical/physical-unit referenced in JCP directive	Ignore directive	D01,D02,H06
JC08	I/O error encountered while performing operation	Terminates directive operation	H05,H06, D01

↓

↑

*NS	Nested DUP statements
*NR	Symbol table full
*TF	Tag error (undefined or illegal index register specifications)
*SZ	Expression value too large for the size of the subfield, or a DUP statement specifying more than three statements to be duplicated (m parameter)
*UD	Undefined digit in an arithmetic expression
*SE	The symbol in the label field has, during pass 2, a value different than that in pass 1
*E	Syntax error (source statement incorrectly formed)
*R	Relocation error (relocatable item encountered where an absolute item was expected)
*MQ	Missing right quotation mark in character string
* =	Invalid use of literal
*II	Implicit indirect reference when I parameter is present on the /DASMR directive

A.5.2 FORTRAN IV Compiler and Runtime Compiler

During compilation, source statements are checked for such items as validity, syntax, and usage. When an error is detected, it is posted on the LO usually beneath the source statement. The errors marked T terminate binary output.

All error messages are of the form

ERR xx c(1)-c(16)

where xx is a number from 0 to 18 (notification error), or T followed by a number from 0 to 9 (terminating error); and c(1)-c(16) is the last character string (up to 16) encountered in the statement being processed. The right-most character indicates the point of error and the @ indicates the end of the statement. The possible error messages are:

Notification

Error	Definition
0	Illegal character input
1	Construction error
2	Usage error
3	Mode error

Notification

Error

Definition

4	Illegal DO termination
5	Improper statement number
6	Common base lowered
7	Illegal equivalence group
8	Reference to nonexecutable statement
9	No path to this statement
10	Multiply defined statement number
11	Invalid format construction
12	Spelling error
13	Format statement with no statement number
14	Function not used as variable
15	Truncated value
16	Statement out of order
17	More than 29 named common regions
18	Noncommon data
19	Illegal name
20	DO index not referenced
21	Name is dummy
22	Array name previously declared
23	Exponent underflow or overflow
24	Undefined statement number

ERROR MESSAGES

Terminating

Error	Definition
T0	I/O error
T1	Construction error
T2	Usage error
T3	Data pool overflow
T4	Illegal statement
T5	Improper use
T6	Improper statement number
T7	Mode error
T8	Constant too large
T9	Improper DO nesting
T10	DO not parenthesized
T11	Item not operand
T12	Item not function
T13	Invalid unary + ,
T14	Invalid hierarchy
T15	Invalid =
T16	Illegal operator
T17	Function statement without parameters
T18	Logical If follows logical If
T19	Invalid dimensions
T20	Operand is not a name
T21	Too many numeric characters
T22	Non-numeric exponent
T23	Terminator not
T24	Illegal terminator
T25	Not statement end
T26	Invalid common type
T27	Target statement precedes DO
T28	Subscript variable not dummy
T29	Not first statement (Title statement)
T30	First two characters not DO
T31	Not in subprogram
T32	Subscript not integer constant

Note: due to optimization, the error message may appear on the next labeled statement and not on the actual statement error.

RUNTIME

When an error is detected during runtime execution of a program, a message is posted on the LO device of the form:

taskname message

Fatal errors cause the job to be aborted; execution continues for non-fatal errors. The messages and their definitions are:

Message	Cause
ARITH OVFL	Arithmetic overflow
GO TO RANGE	Computed GO TO out of range*
FUNC ARG	Invalid function argument (e.g., square root of negative number)
FORMAT	Error in FORMAT statement*
MODE	Mode error (e.g., outputting real array with I format)*
DATA	Invalid input data (e.g., inputting a real number from external medium with I format)*
I/O	I/O error (e.g., parity, EOF)*

* indicates fatal error; all others non-fatal

A.5.3 RPG IV Compiler and Runtime Compiler

During compilation, source statements are checked for such items as validity, syntax and usage. When an error is detected an arrow is printed pointing to the discrepancy in the source statement and an error message is output on the LO device. Detailed descriptions can be found in the RPG IV User's Manual (98 A 9947 03X). The possible error messages are:

Messages

Indicator	Name
Invalid	Relational
Label	Size
Literal	Syntax

If an I/O error occurs during compilation one of the following messages is posted on Logical Unit 15 and compilation is terminated:

ERROR MESSAGES

Message	Condition	Action	Possible User Action
RP01,nnn	I/O error	Compilation terminated	H06
RP02,nnn	End of file error	Compilation terminated	P07
RP03,nnn	End of device error	Compilation terminated	P07
RP04	End card error (End card encountered before procedure card)	Compilation terminated	P07
RP05	Available memory exceeded	Compilation terminated	M01,M03,M04

where nnn is the logical unit number on which the error occurred.

RPG Runtime/loader during the loading or executing of an RPG IV object program in the background any of the following conditions will cause an error. The message is posted on Logical Unit 15 and the task aborted:

Message	Condition	Action	Possible User Action
RT01,nnn	I/O error	Task aborted	H06
RT02,nnn	End of file error	Task aborted	P07
RT03,nnn	End of device error	Task aborted	P07
RT04	Program too big	Task aborted	P07
RT05	Invalid object record	Task aborted	P08
RT06	Checksum error	Task aborted	P08
RT07	Sequence error	Task aborted	P08
RT08	Program not executable	Task aborted	P08
RT09	Work list overflow	Task aborted	M01,M02,M03 M04
RT10,xxxxxx	Invalid call to sub-routine or missing sub-routine where xxxxx = subroutine name	Task aborted	P08

ERROR MESSAGES

Concordance Program:

Message	Condition	Action	Possible User Action
CN01	Symbol table full	Partial concordance output, then next segment is processed	M01

A.6 LOAD-MODULE GENERATOR

Message	Condition	Action	Possible User Action
LG01	Invalid LMGEN directive	Ignore directive	D01,D02
LG02	Invalid or missing parameter in an LGMEN directive	Ignore directive	D01,D02
LG03	Check-sum error in object module	Abort loading	P08,D02
LG04	READ error in object module	Abort loading	P08,H06
LG05	WRITE error in load module loading	Abort loading	P08,H06
LG06	Cataloging error, name already in library, library full, or reference, subroutine without 'NAME' statement	Abort loading	D03,H06
LG07	Loader code error in object module	Abort loading	P08
LG08	Sequence error in object module	Abort loading	P08
LG09	Structure error in object module (i.e., non-binary record)	Abort loading	P08
LG10	Literal pool overflow or use of literal or indirect by foreground program	Abort loading	P08,P09
LG11	Invalid redefinition of common-block size during load-module generation	Abort loading	P08

Concordance Program:

Message	Condition	Action	Possible User Action
CN01	Symbol table full	Partial concordance output, then next segment is processed	M01

A.6 LOAD-MODULE GENERATOR

Message	Condition	Action	Possible User Action
LG01	Invalid LMGEN directive	Ignore directive	D01,D02
LG02	Invalid or missing parameter in an LGMEN directive	Ignore directive	D01,D02
LG03	Check-sum error in object module	Abort loading	P08,D02
LG04	READ error in object module	Abort loading	P08,H06
LG05	WRITE error in load module loading	Abort loading	P08,H06
LG06	Cataloging error, name already in library, library full, or reference, subroutine without 'NAME' statement	Abort loading	D03,H06
LG07	Loader code error in object module	Abort loading	P08
LG08	Sequence error in object module	Abort loading	P08
LG09	Structure error in object module (i.e., non-binary record), or directory structure error.	Abort loading	P08
LG10	Literal pool overflow or use of literal or indirect by foreground program	Abort loading	P08,P09
LG11	Invalid redefinition of common-block size during load-module generation	Abort loading	P08

ERROR MESSAGES

	LG12	Load-module size exceeds available memory or SW file size	Abort loading	P02,D34
	LG13	LMGE internal tables exceed available memory	Abort loading	M01
	LG14	Number of overlay segments input not equal to that specified in TIDB	Abort loading	D01,D02
	LG15	Undefined externals	Loading continues	P10
→	LG16	No program execution address	Loading continues. Address defaults to the first location of the program	P11
	LG17	Attempt to load protected task on background library or unprotected task on foreground library	Abort loading	D01,D02,D33
	LG18	No load module to catalog	Abort cataloging	P08

A.6.1 RELINK

Message	Condition	Action	Possible User Action
RL01	Input error	Ignore input	correct syntax
RL02 xxxxxx	Load module xxxxxx in error	Ignore	Check validity of load module
RL03 xxxxxx	Invalid CRST item in module xxxxxx	Ignore module	Load module not relinkable, re-LMGEN
RL04 xxxxxx	Nucleus pointer name xxxxxxx not found in CL	Abort relinking	Load module not relinkable on this system
RL05 xxxxxx	Invalid displ. found in CRST	Abort relinking	CRST is invalid, Re-LMGEN

ERROR MESSAGES

FM10	Loader code error in object module	FMAIN process terminated	P08
FM11	Sequence error in object module	FMAIN process terminated	P08
FM12	Non-binary record in object module	FMAIN process terminated	P12
FM13	Number of input logical unit not specified by INPUT	FMAIN process terminated	D01,D02
FM14	Insufficient space in memory	FMAIN process terminated	M01

* Messages FM07 through FM14 apply only to the processing of object modules. The occurrence of any of these errors requires that the processing of the object module be restarted after the error condition is removed.

A.10 I/O UTILITY

Message	Condition	Action	Possible User Action
IU01	Invalid IOUTIL directive	Directive ignored	D01,D02
IU02	Invalid or missing parameter in IOUTIL directive	Directive ignored	D01,D02
IU03	PFILE directive not used to open an RMD file	Directive ignored	D02
IU04	I/O error	IOUTIL process terminated	H06
IU05,nn	END-OF-FILE before the specified number or records skipped. When nn = the number of records remaining when the END-OF-FILE or END-OF-DEVICE (on RMD only) occurred. END-OF-TAPE outputs MSG where operator has option to ;RESUME or ABORT. Note: nn is module 0 to 100.	SFILE, SREC terminates upon error condition	P07

ERROR MESSAGES

A.11 SORT ERROR MESSAGES

Message	Condition	Action	
ST01,xxxxxxx	Invalid or missing parameter or control word for the SORT control word xxxxxxxx	Abort job	D01
ST02	Alternate input on RMD	Abort SORT	D01
ST03	SORT control field ending character position is less than start character position, or character position is past end of SORT record	Abort SORT	D01
ST04	Insufficient memory available for work space.	Abort SORT	M01
ST05,xxxxxx	OPEN error on file xxxxxx	Abort SORT	D01,H06
ST06,xxxxxx	I/O error on file xxxxxx	Abort SORT	H06
ST07,xxxxxx	Attempt to write past end-of-file xxxxxx. (Work file or output file too small.)	Abort SORT	D32
ST08	Last of tags (LOT) requested, but tag SORT cannot be performed.	Abort SORT	D01
ST09,XXXXXX	Syntax error on control word INCL, OMIT, or MOVE XXXXXX	Abort SORT	D01

A.12 DATAPLOT

Message	Condition	Action	Possible User Action
DP00,xxxxxx	Plot file overflow	Incomplete plot	D30
DP01,xxxxxx	Buffer overflow	Incomplete plot	M05
DP02,xxxxxx	Attempted to plot from unsorted plot file	Abort plot	P20
DP03,xxxxxx	End-of-file detected before end-of-plot indicator	Incomplete plot	P07

ERROR MESSAGES

FM04	Insufficient space for entry	Module not added	D07,D08,D09 D37
FM05	I/O error	FMAIN process terminated	H06
FM06	Directory structure error, including writing over the directory by direct addressing of an RMD partition	FMAIN process terminated	H06
FM07	Check-sum error in object module	FMAIN process terminated	P08
FM08	No entry name in object module	FMAIN process terminated	P08
FM09	Record-size error in object module	FMAIN process terminated	P12
FM10	Loader code error in object module	FMAIN process terminated	P08
FM11	Sequence error in object module	FMAIN process terminated	P08
FM12	Non-binary record in object module	FMAIN process terminated	P12
FM13	Number of input logical unit not specified by INPUT	FMAIN process terminated	D01,D02
FM14	Insufficient space in memory	FMAIN process terminated	M01
FM15	Released unused request made to an empty file.	Space not released	D36

* Messages **FM07** through **FM14** apply only to the processing of object modules. The occurrence of any of these errors requires that the processing of the object module be restarted after the error condition is removed.

ERROR MESSAGES

A.10 I/O UTILITY

Message	Condition	Action	Possible User Action
IU01	Invalid IOUTIL directive	Directive ignored	D01,D02
IU02	Invalid or missing parameter in IOUTIL directive	Directive ignored	D01,D02
IU03	PFILE directive not used to open an RMD file	Directive ignored	D02
IU04	I/O error	IOUTIL process terminated	H06
IU05,nn	END-OF-FILE before the specified number or records skipped. When nn = the number of records remaining when the END-OF-FILE or END-OF-DEVICE (on RMD only) occurred. END-OF-TAPE outputs MSG where operator has option to ;RESUME or ABORT. Note: nn is modulo 100 (i.e. 0-99)	SFILE, SREC terminates upon error condition	P07
→ IU06	IOUTIL buffer is too small for record size	Directive ignored	M01

DP04,xxxxxx	Minimum/maximum x or y value exceeded	Line will follow plot boundary, origin will be shifted	P21
DP05,xxxxxx	PLOTS not called	Abort plot	P22
DP06,xxxxxx	Data Plot I/O error	Abort task xxxxxx	H06,H05
DP07,xxxxxx	Attempted to sort from a non-RMD media	Abort task	D31

where xxxxxx is the task name.

A.13 SUPPORT LIBRARY

There are no error messages unique to this section of the manual.

A.14 REAL-TIME PROGRAMMING

There are no error messages unique to this section of the manual.

A.15 SYSTEM GENERATION

RECORD-INPUT ERRORS: Errors in input record found before processing.

Message	Condition	Action	Possible User Action
SG00	Read error (I/O)	Waits for corrected input	P19,D11
SG01	Syntax error in SGEN directive	Waits for corrected input	D01,D11
SG02	Invalid or missing parameter in SGEN directive	Waits for corrected input	D01,D11
SG03	Syntax error in control record	Waits for corrected input	D11
SG04	Invalid or missing parameter in control record	Waits for corrected input	D01,D11
SG05	Binary-object check-sum error	Waits for corrected input	P08,D11

ERROR MESSAGES

SG06	Binary-object sequence error	Waits for corrected input	P08,D11
SG07	Binary-object record code error	Waits for corrected input	P08,D11
SG08	Unexpected end of file, end of device, or beginning of device	Waits for corrected input	P07,D11
SG09	Improper ordering of load-module-package control records	Waits for corrected input	D11

OUTPUT ERRORS: Errors in the attempt to perform I/O on an RMD or listing unit.

Message	Condition	Action	Possible User Action
SG10	RMD I/O error in directive processor	Waits for indicated corrective action	D12
SG11	RMD I/O error in nucleus processor	Waits for indicated corrective action	D12
SG12	RMD I/O error during library generation	Waits for indicated corrective action	D12
SG13	RMD I/O error during resident-task generation	Waits for indicated corrective action	D12
SG14	First track on RMD bad (unable to write PST/bad-track table)	Waits for indicated corrective action	D12
SG15	Write error on listing device	Waits for indicated corrective action	None

SYSTEM-GENERATOR PROCESSING ERRORS: Errors preventing the correct functioning of the system generator.

Message	Condition	Action	Possible User Action
SG20	Requested SGEN driver not available	System halts	M05,D22,D18,D15

ERROR MESSAGES

SG21	Loading error in directive processor	Waits for indicated corrective action	D12
SG22	Loading error in nucleus processor	Waits for indicated corrective action	D12
SG23	Loading error in library processor/ resident-task configurator	Waits for indicated corrective action	D12
SG24	Stacks exceed available memory	Waits for indicated corrective action	M03,D12
SG25	In complete system definition (missing directive or conflicting ASN definitions)	Waits for indicated corrective action	D01,D12
SG26	RMD error (too many sectors allocated, or nonsequential partition assignments)	Waits for indicated corrective action	D01,D25,D12
SG27	Error while loading SGEN loader, I/O control, or drivers. Driver not found in SGL	System halts	D15
SG28,xx	Error while loading SGEN component xx = 05 - checksum 06 - sequence 07 - record 21 - other in SGEN1 22 - other in SGEN2 23 - other in SGEN3 24 - other in SGEN4	Waits for indicated corrective action	P08,D12
SG29	Last partition listed begins on a bad track	Waits for indicated corrective action	P25
MEMORY ERRORS: Errors of compatibility between allocated memory and a portion of the VORTEX system.			
SG30	Size of nucleus larger than that of defined foreground area	Waits for indicated corrective action	M03,D12

ERROR MESSAGES

Message	Condition	Action	Possible User Action
SG31	Load-module literal pool overflow	Current load module processing terminated, system continues	P09,D17
SG32	Size of load module larger than defined memory area	Current load module processing terminated, system continues	M03,P02,D17
SG33	Invalid definition of common during load-module generation	Current load module processing terminated, system continues	M03,D17
SG34	Number of overlays input not the same as specified by TID control record	Current load module processing terminated, system continues	D01,D17
SG35	Insufficient space in two largest RMD partitions to contain VNO task object modules	Nucleus processing terminated	D14,D34
SG50	Task TIDB set for VNO, but no VOL directive for task	Task loaded as non-VNO	D35

SYSTEM LOADING AND LINKING ERRORS: Errors that prevent normal loading or linking of system components.

Message	Condition	Action	Possible User Action
SG40	Loader code error in library processor	Current load module processing terminated, system continues	P08,D17
SG41	Loaded program contains no entry name	Current load module processing terminated, system continues	P08,D17

ERROR MESSAGES

Message	Condition	Action	Possible User Action
SG42	Unsatisfied external in library processor	Current load module processing terminated, system continues	P10,D17
SG43	No execution address found in root segment or overlay	Processing continues. Address defaults to the first location of the program	P11
SG44	Loader code error in nucleus processor (i.e., indirect or literal in foreground task)	Waits for indicated corrective action	P08,D12,
SG45	Unsatisfied external in nucleus processor	Waits for indicated corrective action	P10,D12
SG46	System peripheral assigned to more than one logical-unit class	Waits for indicated corrective action	D12

A.16 SYSTEM MAINTENANCE

Message	Condition	Action	Possible User Action
SM01	Invalid SMAIN directive	Ignore directive	D01,D02
SM02	Record not recognized	Ignore directive	P19,D10
SM03	Check-sum error in object module	Waits for indicated corrective action	P08,D10
SM04	Incorrect size of object-module record (correct: 120 words for RMD input, otherwise 60 words)	Waits for indicated corrective action	P12,D10
SM05	Loader code error in object module	Waits for indicated corrective action	P08,D10

ERROR MESSAGES

SM06	Sequence error in object module	Waits for indicated corrective action	P08,D10
SM07	Object module contains non-object-module text record	Waits for indicated corrective action	P12,D10
SM08	Error or end of device received after reading operation	Waits for indicated corrective action	P07,D10
SM09	Error or end of device received after writing operation	Waits for indicated corrective action	P07,D10
SM10	Stack area full	Waits for indicated corrective action	M01
SM11	Invalid control record	Waits for indicated corrective action	P19,D10

A.17 OPERATOR COMMUNICATION

Message	Condition	Action	Possible User Action
OC01	Request type error	Ignore directive	D01,D02
OC02	Parameter limits exceeded	Ignore directive	D01,D02
OC03	Missing parameter	Ignore directive	D01,D02
OC04	Unknown or undefined parameter	Ignore directive	D01,D02
OC05	Attempt to schedule or time schedule OPCOM task	Ignore directive	D01,D02
OC06	Attempt to declare OC device or system resident unit down	Ignore directive	D01,D02
OC07	Task specified in TSTAT key-in has no established TIDB, task currently not active	Ignore directive	D01,D02

ERROR MESSAGES

RZ05	Named device cannot be replaced (system RMD or device busy)	Ignore directive	D01,D11
RZ06	Irrecoverable I/O error on designated RMD	Ignore directive	H06,D11
RZ07	First track of disc pack bad (pack unusable)	Ignore directive	D24
RZ08	Directive incompatible with specified RMD	Ignore directive	D25,D23
RZ09	Irrecoverable I/O error on system RMD (VORTEX nucleus)	Ignore directive	H06,D11
RZ10	I/O error on LO device	Ignore directive	D11,H06
RZ11	I/O error on SI device	Ignore directive	D11,H06
RZ12	No memory available to allocate for new bad-track table	RAZI aborted	M02
RZ13	Total number of tracks specified in PRT directive exceeds size of the device or is incompatible with the FRM directive	Ignore directive	D25,D11

A.19 PROCESS INPUT/OUTPUT

There are no error messages unique to this section of the manual.

A.20 WRITABLE CONTROL STORE

A.20.1 Microprogram Assembler

During assembly the symbolic statements are checked for syntactic errors. In addition, a condition may occur where the assembler is unable to determine the correct meaning of the symbolic source statements.

Either case is indicated as an error and up to eight error codes will be output beneath the source statement incorrectly constructed.

NR, LC and IO errors terminate the assembly.

ERROR MESSAGES

Each error code with the exception of IO is followed by a space and two decimal digits indicating the character position the assembler was scanning when the error was detected.

The error codes and their meanings are listed below:

Error Code	Meaning	Error Code	Meaning
AD	Address expression or associated fields in error	LC	Program location counter setting exceeds the maximum WCS page size (512 words)
CC	Continuation not expected	MF	Duplicate field reference
CE	Numeric conversion error	NR	No memory available for addition of an entry to assembler's tables
DD	Illegal redefinition of a symbol	NS	No symbol in the label field where required
ER	Syntax error	OP	Operation field undefined
EX	An expression contained an illegal construction	SE	Symbol in label field has a value during pass 2 that is different from the value determined in pass 1
FN	Field number inconsistent with format	Sy	Undefined symbol. A value of zero is assumed
IO	I/O error	SZ	A value too large for the size of a field, or the fields defined in a format statement do not equal 64 bits

A.20.2 Microprogram Simulator

Message	Condition	Action	Possible User Action
MS01	Input could not be interpreted as a valid command	Directive ignored; input recovery*	D01,D02
MS02	A non-hex character was encountered when hex expected	Directive ignored; input recovery*	D02,D02
MS03	Insufficient common area to contain specified number of pages	Request for highest page repeated	M01,D26
MS04	The selected page number was not valid	Directive ignored; input recovery*	D26
MS05	An attempt was made to jump to an unavailable WCS page	Simulation halted	P13

OC10	Attempt to assign unit declared down or assign an unassignable logical unit/device	Ignore directive	D19,H04
OC11	Attempt to allocate TIDB unsuccessful for TSCHED request	Ignore directive	M02

A.18 RMD ANALYSIS AND INITIALIZATION

Message	Condition	Action	Possible User Action
RZ01	Invalid RAZI directive or illegal separator or terminator	Ignore directive	D01,D11
RZ02	Invalid parameter in a RAZI directive	Ignore directive	D01,D11
RZ03	Insufficient or conflicting directive information	Ignore directive	D01,D11
RZ04	New PST incompatible with the system	Ignore directive	D20,D21,D22, D11
RZ05	Named device cannot be replaced (system RMD or device busy)	Ignore directive	D01,D11
RZ06	Irrecoverable I/O error on designated RMD	Ignore directive	H06,D11
RZ07	First track of disc pack bad (pack unusable)	Ignore directive	D24
RZ08	Directive incompatible with specified RMD	Ignore directive	D25,D23
RZ09	Irrecoverable I/O error on system RMD (VORTEX nucleus)	Ignore directive	H06,D11
RZ10	I/O error on LO device	Ignore directive	D11,H06
RZ11	I/O error on SI device	Ignore directive	D11,H06
RZ12	No memory available to allocate for new bad-track table	RAZI aborted	M02
RZ13	Total number of tracks specified in PRT directive exceeds size of the device or is incompatible with the FRM directive	Ignore directive	D25,D11

ERROR MESSAGES

A.19 PROCESS INPUT/OUTPUT

There are no error messages unique to this section of the manual.

A.20 WRITABLE CONTROL STORE

A.20.1 Microprogram Assembler

During assembly the symbolic statements are checked for syntactic errors. In addition, a condition may occur where the assembler is unable to determine the correct meaning of the symbolic source statements.

Either case is indicated as an error and up to eight error codes will be output beneath the source statement incorrectly constructed.

NR, LC and IO errors terminate the assembly.

Each error code with the exception of IO is followed by a space and two decimal digits indicating the character position the assembler was scanning when the error was detected.

The error codes and their meanings are listed below:

Error Code	Meaning
AD	Address expression or associated fields in error
CC	Continuation not expected

CE	Numeric conversion error
DD	Illegal redefinition of a symbol
ER	Syntax error
EX	An expression contained an illegal construction
FN	Field number inconsistent with format
IO	I/O error
LC	Program location counter setting exceeds the maximum WCS page size (512 words)
MF	Duplicate field reference
NR	No memory available for addition of an entry to assembler's tables
NS	No symbol in the label field where required
OP	Operation field undefined
SE	Symbol in label field has a value during pass 2 that is different from the value determined in pass 1
Sy	Undefined symbol. A value of zero is assumed
SZ	A value too large for the size of a field, or the fields defined in a format statement do not equal 64 bits

A.20.2 Microprogram Simulator

Message	Condition	Action	Possible User Action
MS01	Input could not be interpreted as a valid command	Directive ignored; input recovery*	D01,D02
MS02	A non-hex character was encountered when hex expected	Directive ignored; input recovery*	D02,D02
MS03	Insufficient common area to contain specified number of pages	Request for highest page repeated	M01,D26
MS04	The selected page number was not valid	Directive ignored; input recovery*	D26

ERROR MESSAGES

MS05	An attempt was made to jump to an unavailable WCS page	Simulation halted	P13
MS06	A BCS instruction was encountered when WCS page 1 is unavailable	Simulation halted	D26,P13
MS07	Read error on BI device	Loading aborted	H06
MS08	EOF encountered before load complete	Loading aborted	P07
MS09	EOD/BEOD encountered before load complete	Loading aborted	P08
MS10	Sequence error on BI	Loading aborted	P08
MS11	Invalid loader code	Loading aborted	P08
MS12	Checksum error	Loading aborted	P08
MS13	Undefined macro opcode	Simulation continues	P15
MS14	Attempted to write to memory outside defined main memory	Simulation continues	P16
MS15	Attempted to load outside main memory	Loading aborted	P23
MS16	Invalid field name	Remainder of directive ignored	D01
MS17	Invalid field value	Remainder of directive ignored	D01

* Input recovery message or corrected directive from SO device.

A.20.3 Microprogram Utility

Message	Condition	Action	Possible User Action
MU01	Input could not be interpreted as a valid command	Directive ignored; input recovery*	D01,D02

ERROR MESSAGES

MU02	A non-hex character was encountered when hex expected	Directive ignored; input recovery*	D01,D02
MU03	EOF detected on SI	Microprogram utility aborted	P07
MU04	The selected page number was not valid	Directive ignored; input recovery*	D01,D02
MU05	Unable to access WCS: WCS is busy	Directive ignored	H05
MU06	Unable to access WCS: BIC load in progress	Directive ignored	H05
MU07	Read error on BID device	Loading aborted	H06
MU08	EOF encountered before load complete	Loading aborted	P07
MU09	EOD/BOD encountered before load complete	Loading aborted	P08
MU10	Sequence error on BI	Loading aborted	P08
MU11	Invalid loader code	Loading aborted	P08
MU12	Checksum error	Loading aborted	P08

* Input recovery message or corrected directive from SO device.

A.21 VTAM NETWORK CONTROL MODULE

The VTAM network control module (NCM) generates the following error messages:

Message	Condition	Action	Possible User Action
NC01	Syntax error	Ignore directive	D01,D02
NC02	Undefined line	Ignore directive	D27,D02
NC03	Undefined TUID	Ignore directive	D27,D02
NC04	I/O error on file VT\$DFL	Ignore directive	H06,D02

NC05	I/O error on file VT\$DFT	Ignore directive	H06,D02
NC06	Undefined CCM number	Ignore directive	D27,D02

A.22 FILE MAINTENANCE UTILITY (FMUTIL) ERRORS

Message	Condition	Action	Possible User Action
END-OF-FILE	This is an ERROR MSG meaning an END-OF-FILE was encountered before the specified request could be completed.	FMUTIL Process Terminated	D01,P07
A DIRECTORY STRUCTURE ERROR-LUN lun SECTOR-sector num	A = blanks lun = 4 digits giving logical unit number sector num = 7 digits giving the sector number in error. This is an ERROR MSG. Meaning there is a structure error in the object module.	FMUTIL Process Terminated	H06
FILENAME ERROR	INVALID filename or filename not found	No action taken error output and ignored goes to next entry.	D01,D02, D03
. DIRECTORY ERROR ERROR - - - Beg. - - end - - eof - - current - - end - - eof.	Directory error shows writing over the directory by direct addressing of an RMD partition. - = blanks Beg = 2 digits showing beginning sector addr. end = 2 digits showing the ending sector addr. eof = 2 digits showing end-of-file addr. current = 7 digits showing current beg. addr. end = 7 digits showing ending addr. of current sector. eof = 7 digits showing current eof. .	FMUTIL Process Terminated	P17
TAPE INPUT ERROR	READ ERROR (file Header not found)	Outputs error tries again.	D01,D07, D11
PARTITION OVERFLOW	Insufficient space for entry into partition.	Module not added, outputs last directory sector.	D07,D09, D01,D03

ERROR MESSAGES

Message	Condition	Action	Possible User Action
INSUFFICIENT SPACE IN PARTITION	Insufficient space for entry	File not added, FMUTIL process terminated	H06,M01
FMAIN ERROR	4 blanks and 1 digit reference to FMAIN ERROR indicated required I/O error.	Outputs msg. FMUTIL process terminated, depending upon error mentioned.	H06
CAPACITY EXCEEDED	Insufficient space for entry to Directory.	Sorts entries in alphabetical order, and outputs listing.	M01
PARTITION SIZE size - - - - - num ARE UNASSIGNED	Partition size and sectors as stated in error message have not been assigned. - = blanks size = 7 digits showing size of partition. num = 5 digits showing number of sectors unassigned.	Returns to try again.	P17,H06
↓			
ROOT FILE xxxxxx NOT FOUND, NAME yyyyyy NOT ENTERED	Entry name yyyyyy cannot be entered because root name xxxxxx was not found	Processing continues next file.	---
FILE EXTENT ERROR, FILE xxxxxx SKIPPED	File xxxxxx had invalid extents	Processing continues with next file.	---
INSUFFICIENT SPACE IN LUN yyy FOR FILE xxxxxx	File xxxxxx too large to fit in logical unit yyy	Processing continues with next file.	---
↑			

A.23 COMSY ERROR MESSAGES

The following are the COMSY error numbers and associated types of errors detected:

Error Definition

1 Directive not understood.

2 Missing directive.

3 Input was not .COMSY or .FILE when searching for a named COMSY deck on PI.

4 Record sequence error on binary COMSY input.

5 Record checksum error on binary COMSY input.

6 Parameter list in error.

7 Missing .COMSY directive on PI.

8 Updates were not terminated by a .COMSY directive.

9 Sequence number greater than 99999 on an update directive.

10 Update sequence numbers not ascending.

11 .COMSY deck specified, not on COMSY file on PI.

12 Incorrect unit.

14 Common decks limited to 19.

15 Common deck not found.

16 Update directive not understood.

17 I/O error.

18 Erroneous end-of-file condition.

19 Directory error on a random file.

A.24 PARITY ERROR MESSAGES

Message	Description
DOUBLE BIT PARITY ERROR IN TASK XXXXXX BOARD NUMBER = XX, MODULE = XX, SYNDROME = XX	Start routine DUMPPR to print parity error log. If double bit error occurs in nucleus or is repeated, place the system in "HALT" mode and run hardware tests."

A.25 ERROR CODES

A.25.1 Errors Related to Directives

- D01 Check spelling, delimiters, and parameters.
- D02 Enter corrected request from OC or SO.
- D03 Check specified library for module name (FMAIN list).
- D04 Correct task priority.
- D05 Check PIM directives used at system generation.
- D06 Use a global logical unit in directive.
- D07 Use an alternate library or unit.
- D08 Increase library size with RAZI or during SGEN.
- D09 Delete unused modules from library.
- D10 Reposition record if PT or CR (for MT or RMD positioning is automatic and enter on SO:
 - R@ to reread the record or where @ is a
 - P@ to reread the program or carriage return
 - /SMAIN@ to restart SMAIN
- D11 Correct input record by entering it on SO or indicate that it is positioned for rereading by entering C on SO.
- D12 Restart component by entering C on SO. (Repositioning is automatic for MT and RMD, for cards reload the entire deck and SGEN will find component.)
- D13 SGEN requesting bad track analysis for unformatted RMDs or reformat formatted RMDs.
- D14 Restart SGEN from beginning.
- D15 Check spelling, delimiters, etc. of IO INTEROGATION.
- D16 Correct appropriate SGEN directives as indicated.
- D17 Correct indicated module for next SGEN or add corrected module with LMGEM after SGEN completes.
- D18 Check that all RMDs are included in the SYS directive that are indicated by the EQUIP directives.
- D19 Use OPCOM IOLIST for unit to check unit status (up or down) and unit's logical group.
- D20 Check PRT directive.
- D21 Check if maximum number of partitions specified in EDR directive has been exceeded.
- D22 Check for conflicts in controller/unit relations.
- D23 Check logical unit in directive, must be assigned to first partition of the subject RMD unit.
- D24 The specified RMD pack cannot contain a bad track table due to the first track being bad, use another pack.
- D25 Check FRM directive and total number of tracks specified in PRT directive. The following table gives the track capacity for the standard RMDs:

70-75XX	4060 tracks
70-76X0	203 tracks
70-76X3	406 tracks
70-7701	128 tracks
70-7702	256 tracks
70-7703	512 tracks
- D26 Check response to the highest page number requested.
- D27 Check NDM definition or use LIST directive of NCM.
- D28 Use NCM module to check line/terminal status.
- D29 Check that all subject logical units assigned to RMD have been positioned with a PFILE.
- D30 Use a larger file for the plot file.
- D31 Check for proper logical unit (i.e., IOLIST).
- D32 Increase work file xxxxxx size.
- D33 Check type parameter on TIDB directive
- D34 Increase track allocation for the two largest RMD partitions in PRT directives
- D35 (1) Specify task using VOL directive
(2) Remove taskname from TDF directive
(3) Ignore warning message and leave as non-VNO task
- D36 Ignore or delete empty file from partition.

ERROR MESSAGES

D37 Continue to process of adding object modules by entering "ADD,lun,key" from SO

A.25.2 Errors Related to Programs

- P01 Correct request in requesting task and re-execute.
- P02 Recode task using overlays.
- P03 Check for privileged or illegal instruction at specified location. Check listings or check memory by requesting a dump.
- P04 Check FCB or DCB entries.
- P05 Check for proper read mode, packed or unpacked.
- P06 Check for needed global files such as PO, SS, GO, SW. Note: the diagnostic gives the task name and not necessarily the missing file name.
- P07 Check source for an erroneous EOF, END directive, etc.
- P08 Check module for the indicated error;
sequence number--word 1, bits 0-7
checksum value--word 2
Note: binary records can be listed using the DUMP directive of IOUTIL.
- P09 Check \$LIT and \$IAP values from the load module map.
- P10 Examine map for missing externals and make necessary program changes.
- P11 Check for an execution label on the END statement of the source. Note: this is a normal diagnostic for FORTRAN overlays.
- P12 Check for a non-binary record or a short or long record in the module. The record length can be found in word 5 of the request block upon completion of I/O.
- P13 Check code and continue after making corrections as indicated.
- P14 Check requested page number.
- P15 Check opcode for valid instruction.
- P16 Check memory address, store request is ignored.
- P17 Check for specified instruction or operation at location indicated in error message. Note: the address indicated refers to the instruction causing the error and not the violated address.
- P18 Check the page status: read/write, read only, fetch operand only, or unassigned.

P19 Check for illegal data under current mode, i.e., binary in ASCII record, non-binary in binary record.

P20 Sort the plot file.

P21 This may be an intentional message. Plot continues.

P22 Call PLOTS.

P23 Check memory address, check ORG value and load range

P24 Recode into multi tasks or use fewer overlays

P25 Modify PRT directive(s) so bad track is not first one in partition.

A.25.3 Errors Related to Memory Size

- M01 If background, adjust MEM directive as needed.
- M02 Wait for foreground tasks to release memory or TIDB space.
- M03 If MEM request OK or cannot be increased then cut back on foreground common, empty TIDBs, reentry stack size, peripheral drivers, etc. by re-SGEN.
- M04 If sharing blank common and VTAM LCB area, check that a program has not used part of the LCB area.
- M05 Increase buffer area with BSS or dimension commands.
- M06 Increase reentry stack size in SGEN EDR directive.

A.25.4 Errors Related to Hardware

- H01 Make indicated unit ready.
- H02 Clear the protection of the unit. (Disc write protection or write ring in MT).
- H03 ABORT task, reassign SI if necessary, and then declare device down through OPCOM, do not forget to declare it back up again.
- H04 ABORT task and assign alternate device or declare device back up.
- H05 Check hardware for indicated problem.
- H06 Check the OC device for an IO error message, i.e., IOxx.
- H07 Peripheral-related Error. Refer to hardware documentation for the affected device.
- H08 Use back-up media, if available, to recreate file.

APPENDIX B I/O DEVICE RELATIONSHIPS

Function	Allowable Functions by I/O Device Type						
	RMD	MT	PT	CR	CP	LP	TY or CRT
Read binary record	X	X	X	X			X ⁴
Read alphanumeric record	X ¹	X	X	X			X
Read BCD record	X ¹	X	X ²	X ²			X ⁴
Read unformatted record	X ¹	X ¹	X	X			X ⁴
Write binary record	X	X	X		X	X ⁹	X ⁴
Write alphanumeric record	X ¹	X	X		X ³	X	X
Write BCD record	X ¹	X	X ²		X ²	X ⁹	X ⁴
Write unformatted record	X ¹	X ¹	X		X	X ^{9,10}	X ⁴
Write end of file		X	X		X		X ⁸
Rewind unit	X	X	X ³	X			
Skip one record forward	X	X					
Skip one record backward	X	X					
Perform function zero			X ³		X ³	X ⁵	X ⁵
Perform function one						X ⁶	X ⁶
Perform function two						X ⁷	X ⁷
Open a file with rewind option	X	X					
Open a file with leave option	X	X					
Close a file with leave option	X	X					
Close a file with update option	X	X					

NOTES

- (1) All modes are read/written in binary mode.
- (2) BCD mode is handled like unformatted mode.
- (3) Punch 256 frames of leader on paper tape or eject one blank card on card punch.
- (4) All modes are written in alphanumeric mode.
- (5) Advances paper to top of form on line

- printer, or causes carriage return and feeds three lines on Teletype or CRT.
- (6) Advances paper one line.
- (7) Advances paper two lines.
- (8) Rings bell on Teletype or beeps on CRT.
- (9) 620-77 line printer -- All modes are treated as alphanumeric.
- (10) 620-76 printer/plotter -- Unformatted records are transmitted without interpretation as plot data.

I/O DEVICE RELATIONSHIPS

I/O Errors by I/O Device Type

Code	Description	I/O Device						TY or CRT
		RMD	MT	PT	CR	CP	LP	
000	Unit not ready	X	X	X	X	X	X	X
001	Device down	O	O	O	O	O	O	X
002	Illegal LUN specified	O	O	O	O	O	O	O
003	FCB/DCB parameter error	O	O	O	O	O	O	O
004	Level 0 program references a protected partition	O	O	O	O	O	O	O
005	Level 0 program references protected memory	O	O	O	O	O	O	O
006	I/O request error	O	O	O	O	O	O	O
007	Read request to write-only device, or vise versa				O	O	O	
010	File name not found	X						
011	File extent error	X						
012	RMD directory error	X						
013	Level 0 program read a JCP (/) directive on SI	O	O	O	O			
014	Interrupt time out	X	X	X				
015	RMD cylinder-search or malfunction error	X						
016	RMD read/write timing error	X						
017	RMD address error	X						
02n	BICn error	X	X		X	X	X	
030	Parity error	X	X					
031	Reading error by card reader or paper tape device			X	X			
032	Odd-length record error		X					

X = Error reported by I/O drivers.

O = Error reported by I/O control processor.

APPENDIX C DATA FORMATS

This appendix explains the formats and symbols used by VORTEX for storing information on paper tape, cards, and magnetic tape.

C.1 PAPER TAPE

Information stored on paper tape is binary, alphanumeric, or unformatted. It is separated into records (blocks of words) by three blank frames. The last frame of each record contains an end-of-record mark (1-3-4-8 punch).

C.1.1 Binary Mode

Binary information is stored with three frames per computer word (figure C-1). Note that channels 6 and 7 are always punched.

C.1.2 Alphanumeric Mode

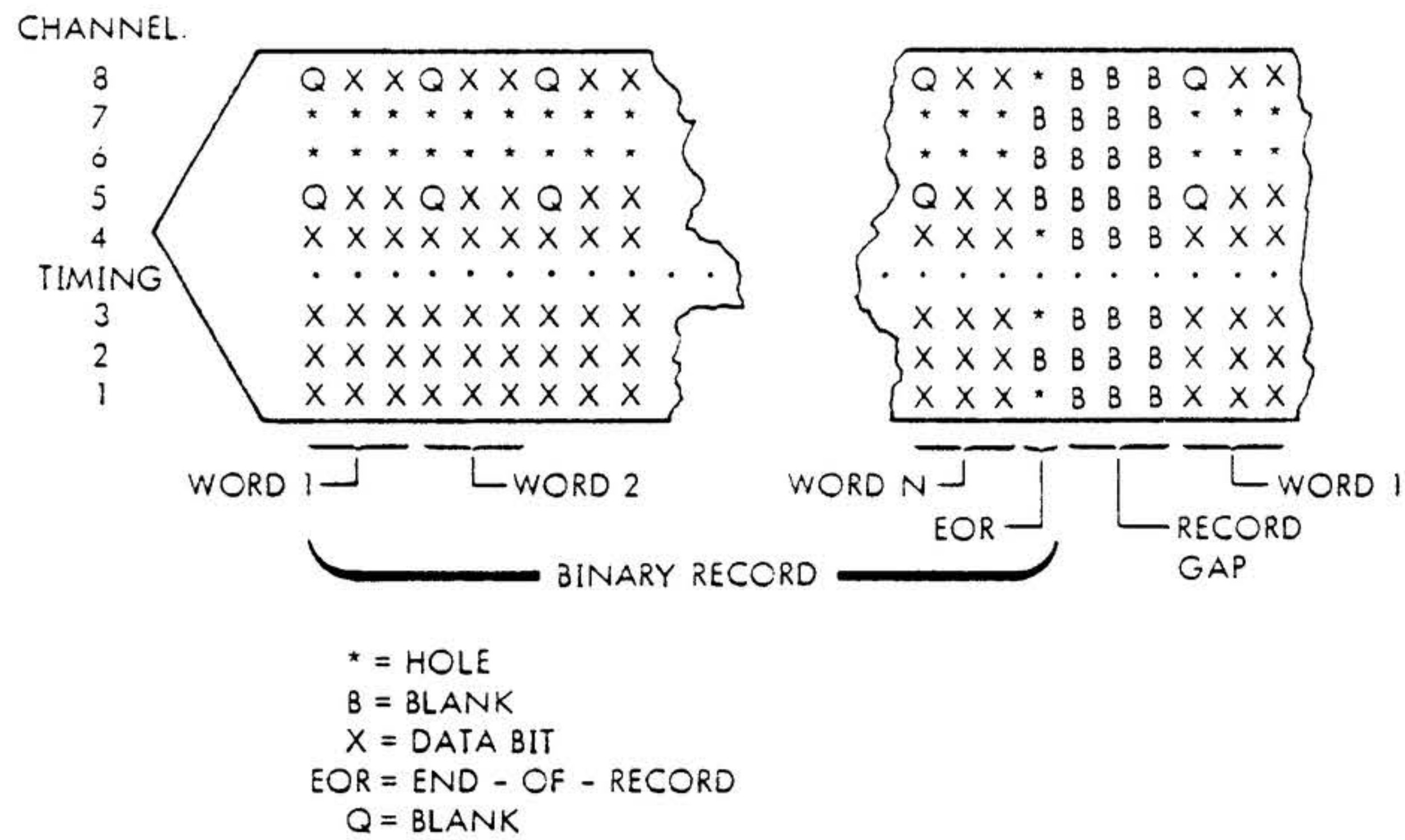
Alphanumeric information is stored with one frame per character (figure C-2). Standard ASCII-8 punch levels are used.

C.1.3 Unformatted Mode

The tape is handled as for alphanumeric mode, but without validity-checking.

C.1.4 Special Characters

An end of file is represented by the ASCII-8 BELL character (1-2-3-8 punch).



VTII-1374

Figure C-1. Paper Tape Binary Record Format

DATA FORMATS

When paper tape is punched on a Teletype, the ASCII-8 ERROR character flags erroneous frames punched by the Teletype when it is turned on or off. This notifies the Teletype and paper-tape reader drivers to ignore the next frame.

When alphanumeric input tapes are punched off-line on a Teletype, there is no means of spacing the three blank frames after every record. The following procedure gives a tape that can be read by the paper-tape reader driver:

- a. Punch the alphanumeric statement.
- b. Punch an end of record (RETURN on the Teletype keyboard).
- c. Punch three or more frames containing any of the following characters:

Press CONTROL and:	ASCII-8 Equivalent
@	DCO
LINE FEED	LINE FEED
WRU	WRU
EOT	EOT
RU	RU
VT	VTAB
TAB	HTAB
HERE IS (33 ASR only)	NULL

NOTE

Any of these characters can also be used for leader and trailer.

- d. Punch the next alphanumeric statement. Return to step b.

C.2 CARDS

Information stored on cards in binary, alphanumeric, or unformatted. Each card holds one record of information. Hence, there is no end-of-record character for cards.

C.2.1 Binary Mode

Binary information is stored with sixty 16-bit words per card. The information is serial with bit 15 of the first word in row 12 of column 1, bit 14 in row 11, etc. (figure C-3). Any 11-0 punch in column 1 is treated as binary.

C.2.2 Alphanumeric Mode

Alphanumeric information is stored one character per card column (figure C-4) using the standard punch patterns.

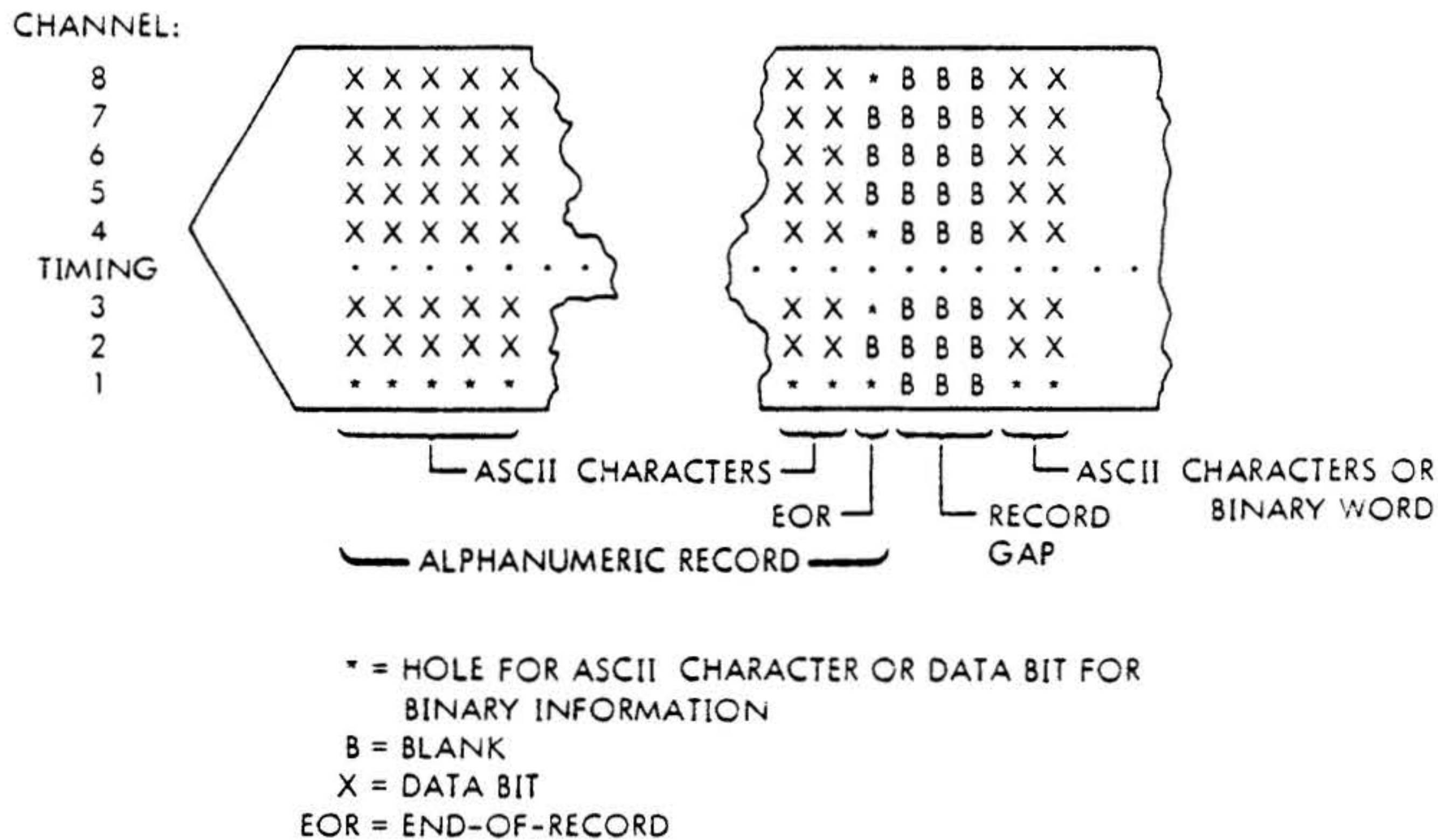
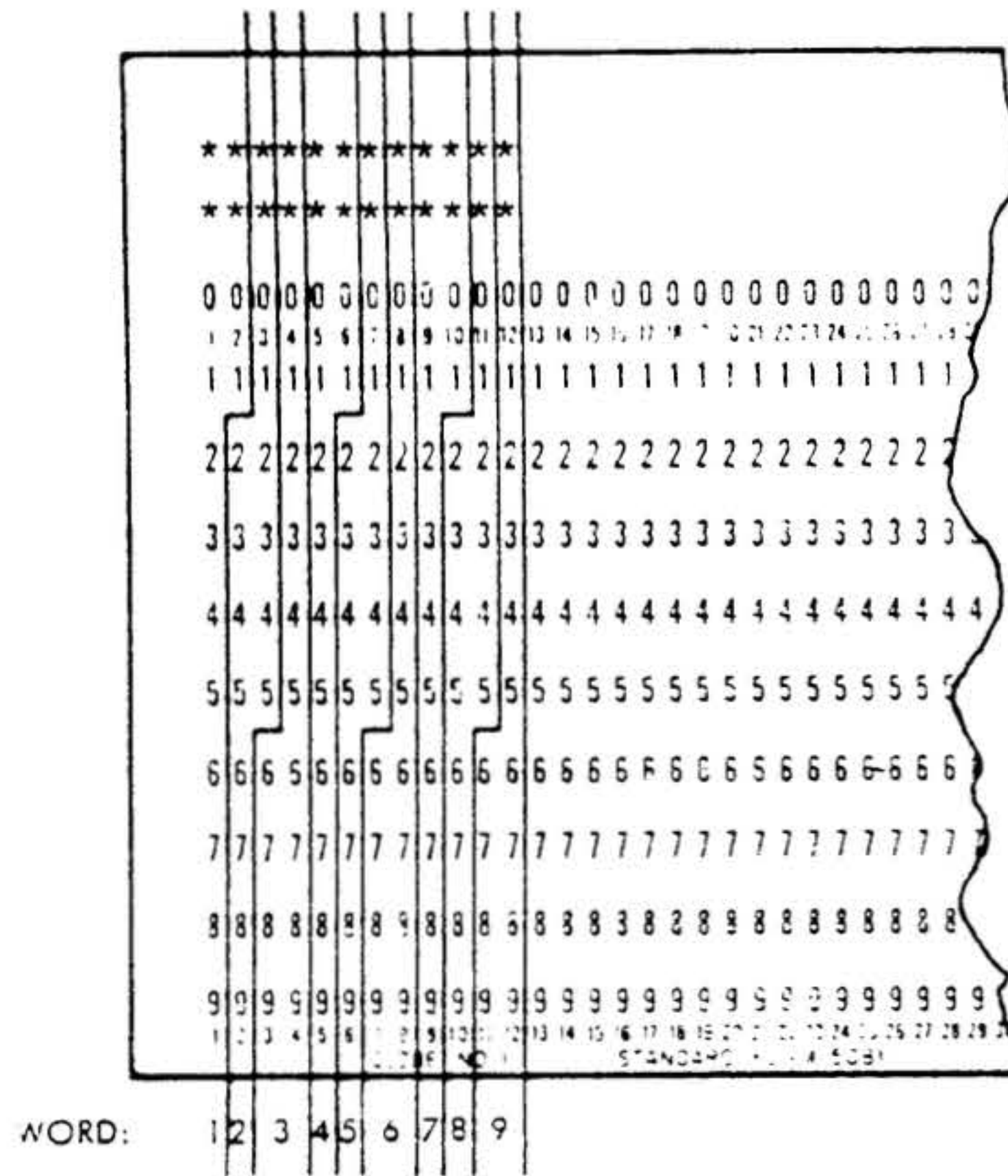
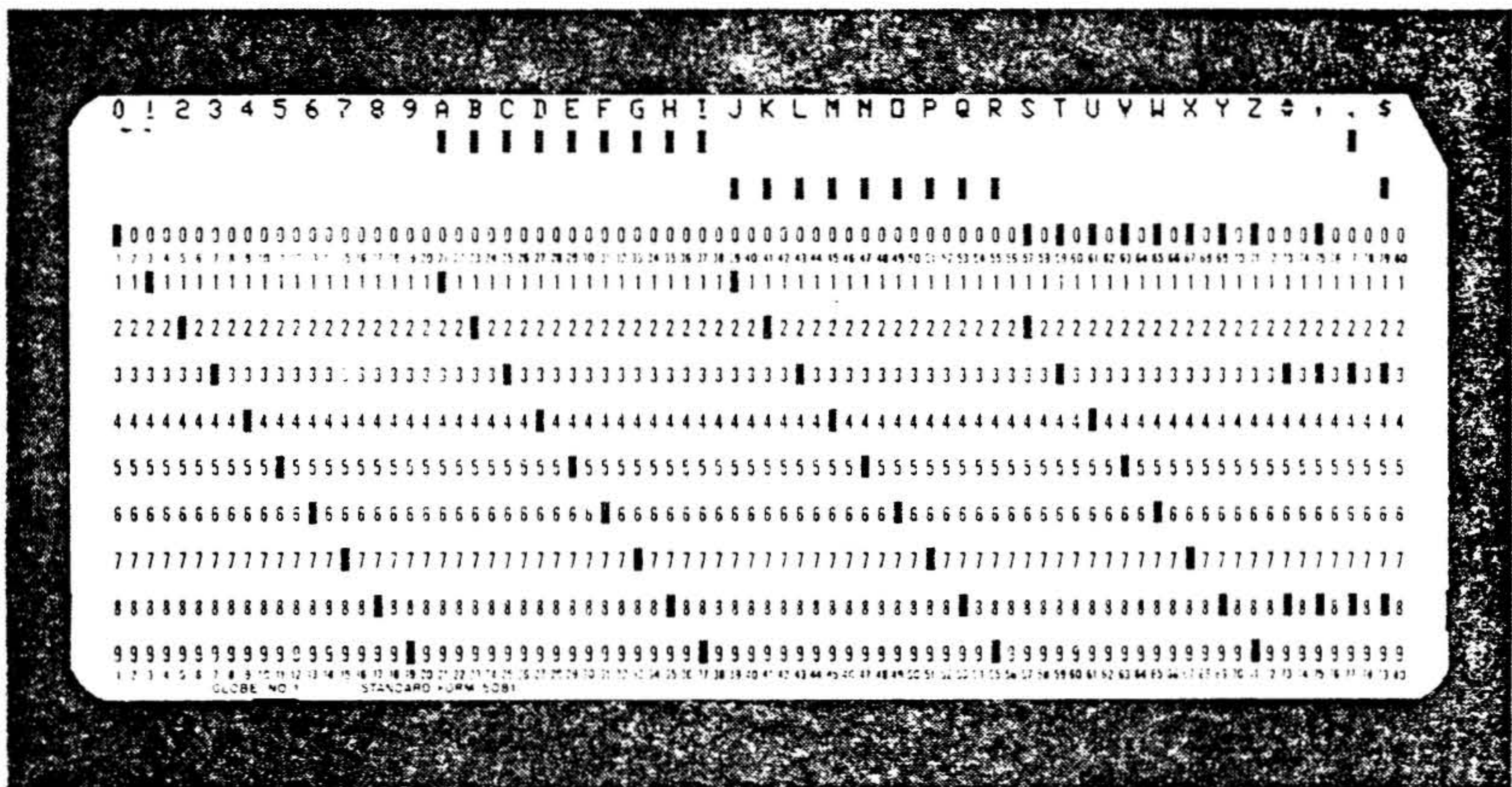


Figure C-2. Paper Tape Alphanumeric Record Format



VTII-1376

Figure C-3. Card Binary Record Format



VTII-0957

Figure C-4. Card Alphanumeric Record Format (IBM 026)

DATA FORMATS

C.2.3 Unformatted Mode

The data are handled, one column per computer word, right-justified, and without validity-checking.

C.2.4 Special Character

An end of file is represented on cards by a 2-7-8-9 punch in column 1 of an otherwise blank card.

C.3 MAGNETIC TAPE

Information stored on seven-track magnetic tape is either binary or BCD. On nine-track tape, information is always binary.

C.3.1 Seven-Track

For system-binary, ASCII, and unformatted modes, the first frame is read into bits 15-12 of the word, the second frame into bits 11-6, and the third into bits 5-0. For BCD mode, the first frame is read into bits 11-6 and the second into bits 5-0.

C.3.2 Nine-Track

In all modes, the first frame is read into bits 15-8 of the word, and the second frame into bits 7-0.

C.4 STATOS PRINTER/PLOTTER

Information may be output to the Statos printer/plotter in alphanumeric and unformatted modes.

C.4.1 Alphanumeric Mode

Information output in alphanumeric mode is assumed to be ASCII characters packed two to a word. Each character is converted to a dot matrix and the print line is transmitted to the device. Characters may be printed in two sizes. The normal print size consists of a 7 by 11 dot matrix and allows 140 characters per line. The large size print consists of a 14 by 22 dot matrix and allows 70 characters per line. Excess characters will be truncated.

C.4.2 Unformatted Mode

Information output in unformatted mode is assumed to be plot data. The information is truncated after n words and transmitted to the device without conversion. Each 1 bit transmitted will cause a dot to be printed on the output line. The most significant bit of the first word is transmitted to represent the left-hand dot position on the line.

" n " depends on the bed width of the plotter. See section 20.3.3 for specific value.

APPENDIX D

STANDARD CHARACTER CODES

IBM 026 Punch			IBM 029 Punch		
Symbol	ASCII	Hollerith	ASCII	Symbol	
!	375	11-0	242	"	
>	276	6-8	275	=	
:	272	5-8	247	'	
,	247	4-8	300	@	
=	275	3-8	243	#	
-	337	2-8	272	:	
9	271	9	271	9	
8	270	8	270	8	
7	267	7	267	7	
6	266	6	266	6	
5	265	5	265	5	
4	264	4	264	4	
3	263	3	263	3	
2	262	2	262	2	
1	261	1	261	1	
(blank)	240	(blank)	240	(blank)	
&	246	11-0	375	!	
<	274	12-6-8	253	+	
[333	12-5-8	250	(
)	251	12-4-8	274	<	
.	256	12-3-8	256	.	
	277	12-2-8	333	[
I	311	12-9	311	I	
H	310	12-8	310	H	
G	307	12-7	307	G	
F	306	12-6	306	F	
E	305	12-5	305	E	
D	304	12-4	304	D	
C	303	12-3	303	C	
B	302	12-2	302	B	
A	301	12-1	301	A	
+	253	12	246	&	
	245	11-7-8	334	\	
:	273	11-6-8	273	:	
]	335	11-5-8	251)	
*	252	11-4-8	252	*	
\$	244	11-3-8	244	\$	
	241	11-2-8	241	!	
R	322	11-9	322	R	
Q	321	11-8	321	Q	
P	320	11-7	320	P	
O	317	11-6	317	O	
N	316	11-5	316	N	
M	315	11-4	315	M	
L	314	11-3	314	L	
K	313	11-2	313	K	
J	312	11-1	312	J	
-	255	11	255	-	
#	243	0-7-8	277	?	
\	334	0-6-8	276	>	
"	242	0-5-8	337	←	
(250	0-4-8	245	%	

STANDARD CHARACTER CODES

IBM 026 Punch			IBM 029 Punch		
Symbol	ASCII	Hollerith	ASCII	Symbol	
.	254	0-3-8	254	.	
@	300	0-2-8	335]	
Z	332	0-9	332	Z	
Y	331	0-8	331	Y	
X	330	0-7	330	X	
W	327	0-6	327	W	
V	326	0-5	326	V	
U	325	0-4	325	U	
T	324	0-3	324	T	
S	323	0-2	323	S	
/	257	0-1	257	/	
0	260	0	260	0	

APPENDIX E ASCII CHARACTER CODES

Character	Internal ASCII	Character	Internal ASCII
0	260	R	322
1	261	S	323
2	262	T	324
3	263	U	325
4	264	V	326
5	265	W	327
6	266	X	330
7	267	Y	331
8	270	Z	332
9	271	(blank)	240
A	301	"	241
B	302	"	242
C	303	=	243
D	304	\$	244
E	305		245
F	306	&	246
G	307	'	247
H	310	(250
I	311)	251
J	312	*	252
K	313	+	253
L	314	,	254
M	315	-	255
N	316	.	256
O	317	/	257
P	320	:	272
Q	321	;	273
<	274	FORM	214
=	275	RETURN	215
>	276	SO	216
	277	SI	217
@	300	DCO	220
....	333	X-ON	221
....	334	TAPE AUX
....	335	ON	222
!	375	X-OFF	223
-	337	TAPE OFF
RUBOUT	377	AUX	224
NUL	200	ERROR	225
SOM	201	SYNC	226
EOA	202	LEM	227
EOM	203	S0	230
EOT	204	S1	231
WRU	205	S2	232
RU	206	S3	233
BEL	207	S4	234
FE	210	S5	235
H TAB	211	S6	236
LINE FEED	212	S7	237
V TAB	213		



APPENDIX F VORTEX HARDWARE CONFIGURATIONS

Device	Device Address	Interrupt	Interrupt Address	BIC	Comments
73-3300 Memory Map	046	MP halt error	020	n/a	Wired as system priority 1
		MP I/O error	022	n/a	
		MP write error	024	n/a	
		MP jump error	026	n/a	
		MP unassigned error	030	n/a	
		MP instruction fetch error	032	n/a	
		MP write and overflow error	034	n/a	
		MP jump and overflow error	036	n/a	
Power Failure/ Restart	---	Power failure	040	n/a	Wired as system priority 2
		Power restart	042	n/a	
Real-Time Clock	047	RTC variable interval	044	n/a	Wired as system priority 4 Base timer inter- val rate is 100 microseconds; free-running clock rate is 100 micro- seconds
		RTC overflow	046	n/a	
Priority Interrupt Module (PIM)	040-043		0100-0277	n/a	Wired as system priority 5; assign- ments should be from fastest to slowest Addresses 064- 067 available for special use
Special PIM Instruction	044		n/a	n/a	PIMs modified to enable/disable with EXC 044
Buffer Interlace Controller (BIC) or Block Transfer Controller (BTC)	020-027 070-073	BIC complete	0100-0277	n/a	All wired as sys- tem priority 3 Addresses 070- 073 available for BIC5 and BIC6 others created for spe- cial use

VORTEX HARDWARE CONFIGURATIONS

Device	Device Address	Interrupt	Interrupt Address	BIC	Comments	
Disc Memory	70-7702 70-7703	620-47 -48,-49 Drum -43C, D Disc Memory	014	BIC complete	0100-0277 Yes	RMD assigned to Highest system BIC (no other devices can be so assigned)
Disc Memory	70-7600 70-7610	620-37, -36 Disc Memory	016-017	BIC complete Cylinder- search com- plete	0100-0277 0100-0277 Yes	RMD assigned to highest system BIC (no other devices can be so assigned)
	70-7603 70-7613	Model F Disc Memory	015-017	BIC complete Cylinder- search com- plete	0100-0277 0100-0277 Yes	RMD assigned to highest system BIC (no other devices can be so assigned)
	70-7500	620-35 Disc Memory	015	BIC complete Cylinder- search com- plete	0100-0277 0100-0277 Yes	RMD assigned to highest system BTC (no other devices can be so assigned)
	70-7510	620-34 Disc Memory	015-017	BIC complete Cylinder- search com- plete	0100-0277 0100-0277 Yes	RMD assigned to highest system BTC (no other devices can be so assigned)
Magnetic Tape	70-7100	620-30 -31A, -31B, or -31C, -32 Magnetic Tape Unit		Tape motion complete	0100-0277 0100-0277 Yes	
Card Reader	70-6200	620-25 Card Reader	030	BIC complete	0100-0277 Yes	
Printer/ Plotter	70-6602	620-75 Statos Printer/ plotter	035-036	BIC complete PC not busy	0100-0277 Yes	
		70-7702 70-660x Statos Printer/ Plotter	035-036	BIC complete PC not busy Statos not busy	0100-1077 0100-0277 0100-0277 Yes	Interrupt event words should be 01 for BIC, 02 for Statos, and 04 for PC
Line Printer		620-77 Line Printer	035-036	BIC complete	0100-0277 Yes	

VORTEX HARDWARE CONFIGURATIONS

Device	Device Address	Interrupt	Interrupt Address	BIC	Comments	
Card Punch	70-6201	620-27 Card Punch	031	BIC complete	0100-0277 Yes	
Paper-tape System	70-6320	620-55, -55A Paper Tape System	037,034	Character ready	0100-0277 No	
Teletype	70-6100	620-6, -7, -8 Teletype	001-007	Read buffer ready	0100-0277 No	Event 1 = READ Event 2 = WRITE
	70-6104			Write buffer ready	0100-0277	
	70-6400	CRT with E-2184 Controller	---	Read buffer ready	0100-0277 No	Compatible with Teletype (Event 1 = READ, Event 2 = WRITE)
	Front Panel	---	Write buffer ready	0100-0277		
					00-01 No	Wired as system priority 6; not used by VORTEX
WCS	73-4000, -4001, -4002	070-074	n/a	n/a	No	Only one device is used in a given system. Multiple (512 word) WCS pages use the same device address.

NOTES

(1) The priority look-ahead option is required if there are more than eight priority devices in the system.

(2) PIM assignments are arranged from the fastest devices to the slowest.



APPENDIX G OBJECT MODULE FORMAT

Object modules generated by the VORTEX language processors result from assembly or compilation. The modules are input by the load-module generator and are bound together into a load module.

The first record of the module contains the size of the program, an eight-character identification, and an eight-character date. Entry name addresses, if any, appear as the first data field items of the object module.

G.1 RECORD STRUCTURE

Object-module records have a fixed length of sixty 16-bit words. Word 1 is the record control word. Word 2 contains the exclusive-OR check-sum of word 1 and words 3 to 60. Words 3 to 11 can contain a program identification block (optional). Words 12 to 60 (or 3 to 60 if there is no program identification block) contain data fields.

Table G-1 illustrates record control word formats.

G.2 PROGRAM IDENTIFICATION BLOCK

The program identification (ID) block appears in words 3 to 11 of the starting record of each module. Word 3 contains the program size, words 4 to 7 contain an ASCII eight-character program identification, from the TITLE statement, and words 8 to 11 contain an ASCII eight-character date.

G.3 DATA FIELD FORMATS

Data fields contain one-, two-, three-, or four-word entries. One-word entries consist of a control word; two-word

entries consist of a control word and a data word; three-word entries consist of a control word and two data words; and four-word entries consist of a control word, two name words, and a data word. Data words can contain instructions, constants, chain addresses, entry addresses, and address offset values.

Table G-1. Record Control Word Format

Bit	Binary Value	Meaning
15	0	Verify check-sum
	1	Suppress check-sum
13-14	11	Binary record
	00-10	Nonbinary record
12	0	First record of module
	1	Not the first record
11	0	Last record of module
	1	Not the last record
10	0	
9	0	
8	0	Not a relocatable module (absolute)
	1	Relocatable module
0-7		Sequence number (modulo 256)

G.4 LOADER CODES

Loader codes, which have the following format, are among the data in an object module.

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0			

Code Subcode Pointer Name			
Code Values	Meaning		
00	Refer to subcode for specific action.		
01	Undefined.		
02	Add the value of the selected pointer to the data word before loading.		
03	Add the value of the selected pointer to the first data word (literal value) and enter the sum in the direct literal pool if bit 11 of the second data word is zero. Otherwise, enter it in the indirect literal pool. Add the address of the literal to the second data word before loading.		

OBJECT MODULE FORMAT

Code Values	Meaning
04	Load the data word(s) absolute. Bits 12 through 0 indicate the number of words minus one (n-1) to load.
05-07	Undefined.
Subcode Values	Meaning
00	Ignore this entry (one word only).
01	Set the loading address counter to the sum of the specified pointer plus the data word.
02	Chain the current loading address counter value to the chain whose last address is given by the sum of the selected pointer plus the data word. Stop chaining when an absolute zero address is encountered.
03	Complete the postprogram references by adding to each address the sum of the selected pointer plus the data word.
04-06	Undefined.
07	Set the program execution address to the sum of the values of the selected pointer plus the data word.
010	Define the entry name with entry location as equal to the value of the selected pointer plus the data word.
011	Define a region for the pointer whose size is given in the data word. If the entry name is not blank, define the entry point as the base of the region.
012	Enter a load request for the external name. The chain address is given by the sum of the selected pointer plus the data word.
013	Enter the loading address of the external name in the indirect literal pool. Add the address of the literal plus the value of the selected pointer to the data word (command) before loading.
014-017	Undefined.
Pointer Values	Meaning
00	Program region.
01	Postprogram region.
02	Blank common region.
03-036	Labelled COMMON regions.
037	Absolute (no relocation).

Name Format

Names are one to six (six-bit) characters, starting in bit 3 of the control word and ending with bit 0 of the second

name word. Only the right 16 bits of the two name words are used.

G.5 EXAMPLE

The following is a sample background program with the description of the object module format after the assembly and the core image after loading.

G.5.1 Source Module

	NAME	SUBR
	EXT	BBEN
SUBR	ENTR	
	LDA*	SUBR
	CALL	BBEN
	STA	TIME
	JAN	DONG
	LDA	=2
	CALL	BBEN
DONG	INR	SUBR
	JMP*	SUBR
TIME	BSS	1
	END	

G.5.2 Object Module

060400 Record control word (first and last record, verify check-sum sequence number 0)

157631 Check-sum word.

(Begin program ID block)

000016 Program size (exclusive of FORTRAN COMMON, literals, and indirect address pointers).

142730 Identification in ASCII (assume this program was labeled
140715 EXAMPLE).
150314
142640

131263 Date of creation in ASCII (assume assembled 03-10-69)
126661
130255
133271

(End program ID block)

010000 Define entry name SUBR at relative 0 (code 0, subcode 010,
000647 pointer 0, name SUBR, and data word 0).
054262
000000

100000 Enter absolute data word 0 in memory at relative 0.
000000

060000 Enter literal (indirectly addressed relative 0) in indirect
100000 pointer pool, add address of pointer to load 017000 and en-
017000 ter memory at relative 1.

100000 Enter absolute data word 02000 in memory at relative 2.
002000

OBJECT MODULE FORMAT

100000 000000	Enter absolute data word 000000 in memory at relative 3.
100000 054010	Enter absolute data word 054010 in memory at relative 4.
100000 001004	Enter absolute data word 01004 in memory at relative 5.
040000 000012	Enter relative data word 012 in memory at relative 6.
060760 000002 010000	Enter literal (absolute 2) into literal pool, add address of literal to load command 010000, and enter in memory at relative 7.
100000 002000	Enter absolute data word 02000 in memory at relative 010.
040000 000003	Enter relative data word 03 in memory at relative 011.
060000 000000 047000	Enter literal (relative 0) into indirect pointer pool, add address of literal to increment command 047000, and enter in memory at relative 012.
100000 001000	Enter absolute data word 01000 in memory at relative 013.
040000 100000	Enter relative data word 0100000 in memory at relative 014.
001000	Set loading location for next command, if any, to relative 016.
012003 000212 024556 000011	Enter load request for external name BBEN and chain entry address to relative 011.

(The remaining words of this record contain zero).

G.5.3 Core Image

Assume the program originates at 01000, the literal pool limits are 0500-0777, and BBEN is loaded at 01016.

0500	101000	DATA	*01000
0501	001000	DATA	1000
.			
.			
0777	000002	DATA	2
.			
.			
01000	000000	ENTR	0
01001	017500	LDA*	0500
01002	002000	JMPM	
01003	001016		01016
01004	054010	STA	01015
01005	001004	JAN	
01006	001012		01012
01007	010777	LDA	0777
01010	002000	JMPM	
01011	001016		01016
01012	047501	INR*	0501
01013	001000	JMP	
01014	101000		* 01000
01015		BSS	1
01016		BSS	1

The following six-bit codes are used by the load-module generator in building load modules. The codes define names created by NAME, TITLE, and EXT directives.

Character	Octal	Character	Octal	Character	Octal
@	40	V	66	+	13
A	41	W	67	,	14
B	42	X	70	-	15
C	43	Y	71	.	16
D	44	Z	72	/	17
E	45	[73	0	20
F	46	\	74	1	21
G	47]	75	2	22
H	50		76	3	23
I	51	-	77	4	24
J	52	(blank)	00	5	25
K	53	!	01	6	26
L	54	"	02	7	27
M	55	#	03	8	30
N	56	\$	04	9	31
O	57	%	05	:	32
P	60	&	06	;	33
Q	61	'	07	<	34
R	62	(10	=	35
S	63)	11	>	36
T	64	*	12	?	37
U	65				

OBJECT MODULE FORMAT

G.6 END LOAD RECORD

An end-load-module record is used to terminate one or more object modules which comprise a root or sequent of a load module. This record is processed similarly to an end-of-file indication by LMGEM, however, more than one end-load-module record may be present on an RMD file.

The form of an end-load-module record is a binary record in which the first word has the value 077000 and all other words are zero.

H.5.2 Secondary Status

Word	Bit	Field	Definition
0	15-0	C	Cylinder address of the last operation
1	15-8	T	Track address of the last operation
1	7-0	S	Sector address of the last operation
2	15-10	ZERO	All zeros
2	9-0	SF	Secondary status field

9 = Correctable ECC error detected
 8 = Write protect error
 7 = Sector flagged bad
 6 = Data synchronization fail
 5 = Header synchronization fail
 4 = Sector search
 3 = Track select error
 2 = Cylinder seek error
 1 = Data check error
 0 = Header CRC error

RMD STATUS WORDS

H.6 F3064 FLEXIBLE DISKETTE

Bit	Meaning	Description
15-14	Undefined	
13	Track analysis recommended	Following a format operation, indicates that at least one track required rewriting. Otherwise, indicates that more than one retry was required to complete the last operation.
12	Selected unit not ready	Selected drive is not ready.
11	Write protect violation	WRITE operation attempted on protected diskette.
10	Track overflow	BIC buffer length exceeded number of sectors remaining on the track.
9	Data CRC error	Data field CRC error occurred.
8	Header CRC error	Header field CRC error occurred.
7	Transfer timing error	Data lost or data overwrite detected during transfer operation.
6	Seek error	Desired track cannot be found (verified).
5	Bad sector indicator	Bad sector flag detected during READ operation.
4	Illegal disk address	Range error for sector or track address was detected by firmware.
3	Record search error	Header for desired record cannot be found.
2-1	Undefined	
0	Context error	Firmware command is not preceded or followed by necessary supportive commands.



H.7 VZLPC LINE PRINTER DRIVER (SUL/ACORN PRINTER)

A bit value 1 indicates an error condition as described below.

Bit	Meaning	Description
0	Stacker Full	Indicates Forms Pullout Stacker is full
1	Band Error	Indicates an error in the Band Drive mechanism
2	Parity Error	Indicates a parity error in the Printer Controller
3	Actuator Error	Indicates an error in the hammer drive system
4	Temperature Error	Indicates an error in the Printer Overtemp Circuitry
5	Forms Runaway	Indicates paper movement for more than 1.25 seconds



RMD STATUS WORDS

Bit	Meaning	Description
6	Paper Feed Motion Error	Indicates an error in the Paper Drive System
7	Forms Jam	Indicates no paper motion due to jam
8	Paper Check	Indicates a forms low condition
9	Not used	
10	VFU Error	Indicates one of the following: <ul style="list-style-type: none"> . An attempt to fill VFU memory beyond its limit . An attempt to slew on an undefined VFU channel . VFU definitions count of zero in VFU initialization request
11	Power	Indicates a loss of power to the printer.
12	Invalid Transfer	Indicates an attempt to transfer a character that was not in a printer code buffer
13	BIC Error	Indicates an abnormal stop or BIC not ready condition, set by driver only
14	Data Parity Error	Indicates that the data on the Printer Data Bus does not have even parity
15	Not used	





APPENDIX I VORTEX II 8-BIT ASCII SUPPORT

I.1 GENERAL

Some VORTEX modules support the use of standard 8-bit ASCII code consisting of eight variable bits in the binary words. Characters in the 8-bit ASCII code and their corresponding octal representations are presented in Table I-1.

The 8-bit ASCII code is different from VORTEX internal ASCII code (Appendix E). VORTEX internal ASCII code is called 7-bit

ASCII code because the most significant (eighth) bit is always set to one. Thus the number of bits which can be changed to signify different characters is seven. The 8-bit ASCII code includes the same characters as the VORTEX internal ASCII code and provides the capacity to include additional special characters. For example, Table I-2 indicates Japanese language Katakana characters and their corresponding octal values.

Table I-1. 8-Bit ASCII Character Set

Octal	Char	Octal	Char	Octal	Char	Octal	Char
000	NUL	040	SP	100	@	140	`
001	SOH	041	!	101	A	141	a
002	STX	042	"	102	B	142	b
003	ETX	043	#	103	C	143	c
004	EOT	044	\$	104	D	144	d
005	ENQ	045	%	105	E	145	e
006	ACK	046	&	106	F	146	f
007	BEL	047	'	107	G	147	g
010	BS	050	(110	H	150	h
011	HT	051)	111	I	151	i
012	LF	052	*	112	J	152	j
013	VT	053	+	113	K	153	k
014	FF	054	,	114	L	154	l
015	CR	055	-	115	M	155	m
016	SO	056	.	116	N	156	n
017	SI	057	/	117	O	157	o
020	DLE	060	0	120	P	160	p
021	DC1	061	1	121	Q	161	q
022	DC2	062	2	122	R	162	r
023	DC3	063	3	123	S	163	s
024	DC4	064	4	124	T	164	t
025	NAK	065	5	125	U	165	u
026	SYN	066	6	126	V	166	v
027	ETB	067	7	127	W	167	w
030	CAN	070	8	130	X	170	x
031	EM	071	9	131	Y	171	y
032	SUB	072	:	132	Z	172	z
033	ESC	073	;	133	[173	{
034	FS	074	<	134	\	174	
035	GS	075	=	135]	175	}
036	RS	076	>	136	^	176	~
037	US	077	?	137	—	177	DEL

Note: Octal values 200 through 377 can be used to represent special characters.

Table I-2. 8-Bit Special Characters, Katakana

Octal	Char	Octal	Char	Octal	Char	Octal	Char
240	(not used)	260	-	300	ヲ	320	ミ
241	。	261	ア	301	チ	321	ム
242	「	262	イ	302	ツ	322	メ
243	」	263	ウ	303	テ	323	モ
244	、	264	エ	304	ト	324	ヤ
245	・	265	オ	305	ナ	325	ユ
246	ヲ	266	カ	306	ニ	326	ヨ
247	ヅ	267	キ	307	ヌ	327	ラ
250	イ	270	ク	310	ネ	330	リ
251	ウ	271	ケ	311	ノ	331	ル
252	エ	272	コ	312	ハ	332	レ
253	オ	273	サ	313	ヒ	333	ロ
254	カ	274	シ	314	フ	334	ワ
255	ユ	275	ス	315	ヘ	335	ン
256	ョ	276	セ	316	ホ	336	ハ
257	ッ	277	ソ	317	マ	337	。

Modifications to certain VORTEX software modules permit the use of the supplementary 8-bit ASCII code for input/output functions, in text strings, or in comments. The affected modules include:

- Input/Output Control (IOC)
- Input/Output Utility (IOUTIL)
- Source Editor (SEDIT)
- DASMR assembler

The sections of this appendix provide detailed descriptions of the enhancements to individual directives which permit the use of 8-bit ASCII code. Existing documentation in this manual is not superseded by the information presented here. Each enhancement description includes a reference to the appropriate section of this manual for complete information on the directive.

1.2 IOC MODIFICATIONS

To use the 8-bit ASCII Character Set with the Input/Output Control module, the user specifies `mode = 6` in the IOC macros `READ` or `WRITE` (Section 3) as follows:

READ/WRITE `cb,lun,wait,6`

Eight-bit ASCII data can be input from a card reader or output to a card punch. When the JCP `KPMODE` (Section 4) is set to 1, 8-bit ASCII data including special characters is input. When the `KPMODE` is 0, 8-bit ASCII data excluding special characters is input. Regardless of the `KPMODE` setting, 2-7-8-9 code punched in the first column of a card indicates an end-of-file. This is indicated in the user's buffer area by bits 15

through 8 of the first word, which are set to the value 0207. The rest of the first word and the remainder of the user's buffer area contain the 8-bit ASCII data. An 11-0 punch (with or without another punch) in the first column of a card does not change the I/O mode from 6 to 0, and this card image is written to the user's buffer area as 8-bit ASCII code.

Eight-bit ASCII data can be written to line printer type 0786 or F3361. The control character specified in the first byte of the first word must be in 8-bit ASCII code.

If the `READ` or `WRITE` macro specifies a `lun` assigned to a magnetic tape or RMD device, 8-bit ASCII code is input or output by the code-transparent driver.

1.3 IOUTIL MODIFICATIONS

Mode 6 for input and/or output can be specified in the IOUTIL directives `COPYF`, `COPYR`, `PRINTF`, and `DUMP` (Section 10) to permit the manipulation of 8-bit ASCII coded data and directives. The option `S` can be included in the JCP directive `/IOUTIL` (Section 4) to provide conversion functions from/to 7-bit ASCII code.

1.3.1 OPTION S

The form of the enhanced JCP directive is:

`/IOUTIL S`

If any parameter other than `S` is used in this directive, the `JC02` error message is output and VORTEX continues processing as if no parameter had been used.

For the COPYF and COPYR directives, mode 6 can be specified for **im** (input mode) and/or **om** (output mode). Conversions performed by the enhanced IOUTIL are shown in Table I-3.

Mode 6 can be specified in the DUMP directive. Eight-bit ASCII characters are output and JCP directives in mixed files are converted from 7-bit code to 8-bit code before the ASCII characters are printed. However, the accompanying octal listing presents those records exactly as they are stored on the file; i.e., JCP directives are listed in octal 7-bit ASCII form.

When the PRINTF directive is used after option S has been specified, both input and output are performed for 8-bit ASCII code.

Table I-3. IOUTIL COPY Directive Conversions, Option S

		Output Mode 1		Output Mode 6	
		PRT	RMD/MT	PRT	RMD/MT
Input Mode 1	CR	No Conversion	No Conversion	Conversion to 8-Bit ASCII	Conversion to 8-Bit ASCII
	RMD/MT	No Conversion	No Conversion	Conversion to 8-Bit ASCII	Conversion to 8-Bit ASCII
Input Mode 6	CR	Conversion to 7-Bit ASCII	Conversion to 7-Bit ASCII	No Conversion	No Conversion (See Note)
	RMD/MT	Conversion to 7-Bit ASCII	Conversion to 7-Bit ASCII	No Conversion (See Note)	No Conversion (See Note)

NOTE: For files copied to RMD/MT using output mode 6, input records containing 057 (8-bit ASCII "/") as the value for the first character are converted to 7-bit ASCII code. These records are JCP directives; VORTEX correctly interprets JCP directives only if they are in 7-bit ASCII code. When such mixed files are printed (mode 6), the 7-bit ASCII records are converted to 8-bit ASCII code.

Table I-4. IOUTIL COPY Directive Output, Without Option S

		Output Mode 1		Output Mode 6	
		PRT	RMD/MT	PRT	RMD/MT
Input Mode 1	CR	7-Bit ASCII	7-Bit ASCII	Unpredictable Output	7-Bit ASCII
	RMD/MT	7-Bit ASCII	7-Bit ASCII	Output as 8-Bit ASCII	No Conversion
Input Mode 6	CR	Unpredictable Output	8-Bit ASCII	8-Bit ASCII	8-Bit ASCII
	RMD/MT	8-Bit ASCII	No Conversion	8-Bit ASCII	8-Bit ASCII

I.3.2 NO OPTION

If the option S is not specified in the /IOUTIL directive, specifying mode 6 for input/output results in the output form indicated by Table I-4

5. When the option S and mode 6 for input/output are specified, only one output device can be specified (COPYR and COPYF directives).

I.3.3 NOTES AND GUIDELINES FOR THE USER

1. If the option S is specified and the IOUTIL directive COPYF or COPYR with input mode 6 is used, records whose first character is 057 are converted to 7-bit ASCII form. The "/" character, octal 057, indicates a JCP directive.
2. When the option S is specified and the DUMP directive (mode 6) is used to process an 8-bit ASCII coded file, records whose first character is 0257 are JCP directives and characters are output in 7-bit ASCII form. The octal record value printed is 0257.
3. A JCP jobstream file to be stored on MT or RMD cannot include mixed 7-bit and 8-bit data. For example, the following jobstream cannot be constructed by IOUTIL:

```

/FMAIN          7-bit data
/SEDIT,S        8-bit data
/FINI
    
```

4. IOUTIL directives to be stored on MT/RMD must be recorded in 7-bit ASCII form.

I.4 SEDIT MODIFICATIONS

Enhancements to SEDIT (Section 8) to accommodate 8-bit ASCII code can be utilized by specifying the options V or S in the JCP /SEDIT directive (Section 4). When either of these options is specified, the SEDIT output (OUT and print file) is prepared in 8-bit ASCII code form.

The format of the enhanced JCP directive is

```

/SEDIT          S
                V
    
```

The V option directs SEDIT to test the IN and ALT files to determine whether each is composed of 7-bit ASCII records or 8-bit ASCII words. Each assignment (via the AS directive) is followed by a checking procedure in which the first record is read: a value of 0177 or less in the first record indicates a file in 8-bit ASCII format; a value of 0200 or more in the first record indicates a file in 7-bit ASCII format. Conversion to 8-bit ASCII code is performed if necessary. This option is useful for merging files coded differently so that the merged source program can be assembled by the enhanced DASMR (see I.5).

The S option directs SEDIT to handle the IN and ALT files as 8-bit ASCII code. No conversion is performed. Output is also in 8-bit ASCII code format.

Notes:

1. An option other than S or V on the /SEdit directive generates a JC02 error message. The directive is then processed as if no option was specified.
2. If files containing JCP directive records are edited, these records must be reconverted (using IOUTIL) to 7-bit ASCII coded form because VORTEX does not correctly interpret JCP directives in 8-bit ASCII coded form.
3. Eight-bit ASCII coded SEdit directives and data that are input
 - (a) from the OC device cannot include special characters (octal value 200 or greater) in strings;
 - (b) from the card reader device can include the special characters if KPMODE is set to 1;
 - (c) from RMD or MT must have been stored on the device in 8-bit ASCII form (using IOUTIL).

I.5 DASMR ASSEMBLER MODIFICATIONS

Enhancements to the DASMR Assembler permit the assembly of programs written using 8-bit ASCII code and the use of character strings written in 8-bit ASCII code. The parameter S can be included in the JCP directive /DASMR (Section 4). Also, particular strings within a DASMR language program (Section 5) can be generated in 8-bit ASCII code.

The enhanced JCP directive has the form:

/DASMR,S

When the option S is specified, the DASMR assembler assumes that input from RMD, magnetic tape, or the card reader is in 8-bit ASCII code. Printed output is produced in 8-bit ASCII code.

Within a DASMR source program, individual character strings can be generated in 8-bit ASCII code by specifying the character S immediately before the first apostrophe that delimits the string.

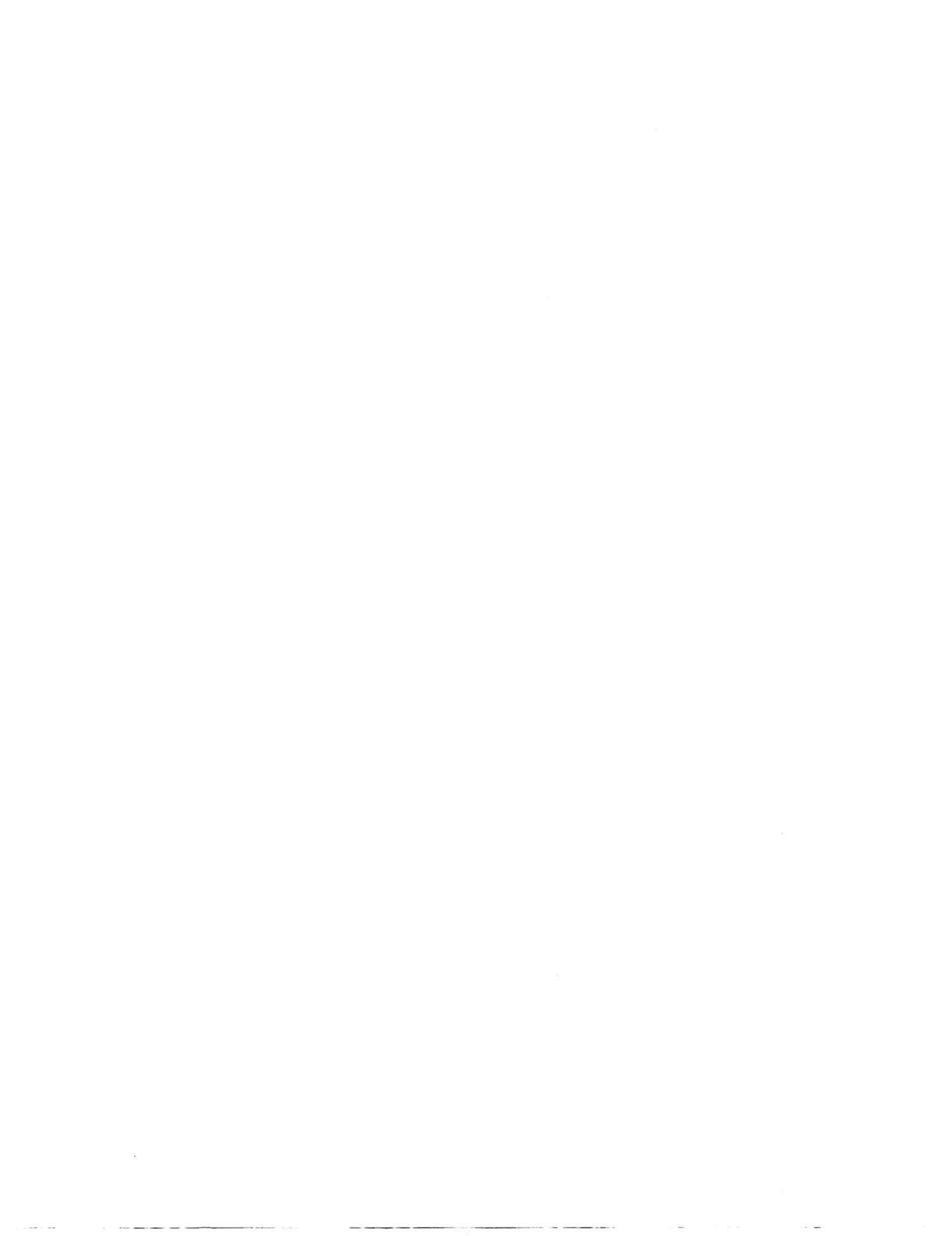
Examples:

```
DATA S'AB', 'CD', S'XYZ'
ADD = S'3'
LDA = S'A'
LDI = S'TJ'
```

In the DATA statement only the first and third strings are specified to be generated in 8-bit ASCII code.

NOTES:

1. If the option S is not specified, the assembler assumes that input and output consist of 7-bit ASCII code.
2. Special characters (8-bit octal value 200 or greater) can be specified only in character strings or comments.
3. The output of pass-1 of the DASMR assembler to the SS logical unit cannot be used as input for any job, other than for pass-2 of the assembler.
4. If the option S is specified, PI and SS should not be assigned to the same logical unit. If this is done, output from the Concordance program can be incorrect.



INDEX

A

ABL Automatic Bootstrap Loader, 18-2
ABORT (OPCOM), 17-4
ABORT (RTE), 2-8
ABORT procedure, 2-16
accelerator (FORTRAN firmware), 20-2, 13-1
access method (I/O), 3-10
access modes, 1-5, 2-13
AD (SEdit), 8-3
ADD (FMAIN), 9-6
ADD (SMAIN), 16-4
add records (SEdit), 8-3
add string (SEdit), 8-3
adding an I/O driver, 14-31
adding controller tables, 14-31
AFAULT (RTE), 2-12
allocate memory pages, 2-9
allocate, stack, 2-6
ALOC (RTE), 2-6
ALOCPG, 2-9
ALOCPG (RTE), 2-9
alphanumeric mode, Status, C-4
alphanumeric mode, cards, C-2
alphanumeric mode, paper tape, C-1
ALT (SMAIN), 16-4
alternate logical unit (SMAIN), 16-4
alternate sector partition, 3-17, 3-18
alternate sector processing, 3-18
ALTIN (VSORT), 11-2
ALTLIB (JCP), 4-8
analog input system, 19-3
APND (PATCH), 15-2
arithmetic fault setup, 2-12
AS (SEdit), 8-2
ASCII character codes, E-1
ASR Teletype, 18-1
assembler, DAS MR, 5-1
assembly listing format, 5-10
ASSIGN (JCP), 4-3
ASSIGN (OPCOM), 17-5
assign logical units (SEdit), 8-2
ATTACH (OPCOM), 17-3
automatic bootstrap loader, 18-2
auxiliary group directives, 8-2

B

background processing, 1-2
background tasks, 2-14, 1-5
bad-track table, 3-4
bad sector table, 3-4
BASE (PATCH), 15-2

bibliography, 1-6
BCB macro (I/O control), 3-21, 3-23
BI: Binary input, 3-2
BIC flag table, 14-32
BIC (Buffer Interface Controller), 14-34
binary mode, cards, C-2
binary mode, paper tape, C-1
binary records (COMSY), 22-1
blank common, 6-3
bootstrap loader, 18-2
BTC (Block Transfer Controller), 14-34
BTPTCH program (PATCH), 15-2, 15-7
byte manipulation firmware, 20-5

C

C (JCP), 4-2
calling sequence, 11-6
card data modes, C-2
card punch (initializing), 18-1
card reader (initializing), 18-1
CFILE (IOUTIL), 10-5
CFILE (JCP), 4-9
change directives, 15-1, 15-3, 15-5
change directive parameter, reference to address, 15-5
change directive parameter, reference to instruction, 15-5
character codes, standard, D-1
checkpoint file, 1-4
checkpointing, 2-14
CLD directive, 6-6
CLOSE (IOC), 3-11
close file (IOUTIL), 10-5
CNTL (PATCH), 15-5
CO (SEdit), 8-7
codes, ASCII character, E-1
codes, standard character, D-1
COMDECK, 22-6
COMSY, 22-1, 22-10
COMSY binary records, 22-1
COMSY directives, 22-2
COMSY error messages, A-28
COMSY error processing, 22-11
COMSY execution, 22-11
COMSY load module generation, 22-11
comment (JCP), 4-2
common, 6-3
common files (COMSY), 22-2
common interrupt handler, 14-1
common module, foreground blank, 1-4
compare inputs (SEdit), 8-7

INDEX

compatibility-micro-VORTEX and VORTEX II, 14-36
compatibility-VORTEX II and VORTEX I, 14-35
Compression/Edit System (COMSY), 22-1
compilers: language processors, 5-1
CONC (JCP), 4-6
concordance program, 5-11
configurations, hardware, F-1
console logging, 25-1
control directives (PATCH), 15-1
control records (SMAIN), 16-2
controller device address, 14-32
copy file (IOUTIL), 10-1
copy file (SEdit), 8-5
copy record (IOUTIL), 10-2
COPYF (IOUTIL), 10-1
copying group, 8-1
COPYR (IOUTIL), 10-2
core resident symbol table, 6-7
CREATE (FMAIN), 9-4
CTADNC, 14-31
CTBICB, 14-30
CTDST, 14-30
CTDVAD, 14-30
CTFCB, 14-30
CTFRCT, 14-30
CTIDB, 14-31
CTIOA, 14-30
CTOPM, 14-31
CTPSTO, 14-30
CTPSTI, 14-30
CTPST2, 14-30
CTPST3, 14-30
CTRQBK, 14-30
CTRTRY, 14-30
CTSTAT, 14-30
CTSTSZ, 14-30
CTTKSZ, 14-30
CTWDS, 14-30

D

DASMR (JCP), 4-5
DASMR assembler, 5-1, 4-5
DASMR assembler, error messages, A-9
data control block (IOC), 3-14
data formats, C-1
DATAPLOT II, 12-1
data record blocking and deblocking, 3-18
data sets, 24-2, 24-7
data set label, 24-1
data transfer firmware, 20-3
DATE (OPCOM), 17-3
DBGEN, 4-9
DCB (IOC), 3-14

DE (SEdit), 8-4
deallocate pages of memory, 2-10
deallocate reentrant stack, 2-7
DEALLOC (RTE), 2-7
DEALPG (RTE), 2-10
DEBUG, 7-1
debugging aids, error messages, A-15
DECK SETUPS (JCP), 4-11
DEL (SMAIN), 16-6
DELAY (RTE), 2-3
DELETE (FMAIN), 9-3
delete (SMAIN), 16-5
delete records (SEdit), 8-4
delete string (SEdit), 8-5
DEMAND macro (I/O control), 3-20, 3-21, 3-24
DEVDN (OPCOM), 17-5
device down (OPCOM), 17-5
device initialization, 18-1
device specification table (DST), 3-4
DEVUP (OPCOM), 17-5
digital-to-analog converter, 26-2
direct access, 3-10
directive (COMSY), 22-10
directive input unit (SGEN), 15-7
directives, DASMR assembler, 5-1
directives, FMUTIL, 21-1
directly connected interrupt handler, 14-34
disk, 18-1
disk access methods, 3-19
disk device I/O, 3-17
disk, key-in loader programs, 18-2
disk pack handling, 18-3
dispatcher interrupt processor, 14-28
DMEMRY, 7-12
driver interface, 14-33
drum, key-in loader programs for, 18-2
DSPMEN directive, 7-5
DSPMEN program, 7-5
DUMMY logical unit, 3-1
DUMP (FMUTIL), 21-2
DUMP (IOUTIL), 10-3
DUMP (JCP), 4-8
DUMP (PATCH), 15-1
dump directory (FMUTIL), 21-3
dump partition (FMUTIL), 21-3

E

edition numbers (COMSY), 22-2
eight-bit ASCII support, I-1
END (COMSY), 22-10
END (DEBUG), 7-8
END (LMGEN), 6-5
ENDSORT, 11-6

END JOB (JCP), 4-2
 ENTER (FMAIN), 9-5
 ERROR task, 14-4
 error retry count (CTRTRY), 14-30
 error processing (COMSY), 22-11
 error messages (COMSY), A-28
 error messages (PATCH), 15-9, A-20
 event word, 2-9
 EXEC (JCP), 4-7
 execute (COMSY), 22-11
 execute (JCP), 4-7
 executive mode, 1-5, 2-13, 14-4
 execution-time I/O unit (FORTRAN), 5-18
 EXIT (FMUTIL), 21-8
 EXIT (PATCH), 15-3
 EXIT (RAZI), 18-5
 EXIT (RTE), 2-7
 exists (VSORT), 11-6
 extension number, 9-2
 external interrupts, 14-1

F

F2963 Data Acquisition and Control System, 19-1
 FC (SEdit), 8-5
 FCB (IOC), 3-15
 FCB macro (I/O control), 3-21, 3-23
 FCB module, global, 1-4
 FIL (image file), 7-5
 file-control block, 3-5
 file-control block (IOC), 3-15
 file extension, 3-5
 file maintenance (JCP), 4-6
 file maintenance utility (FMUTIL), 21-1
 file maintenance utility (FMUTIL) error messages, A-27
 file maintenance, error messages, A-16
 file-name directory, 3-4, 9-2
 FINI (JCP), 4-2
 FIMPY (fixed-point multiply), 20-3
 firmware, 20-2
 firmware, F option, 4-6
 firmware macros, 20-9
 fixed-point arithmetic firmware, 20-2
 floating-point arithmetic firmware, 13-1, 20-3
 flow, system, 1-2
 FMAIN (JCP), 4-6
 FMAIN: file maintenance, 9-1
 FMUTIL, 21-1, 6-7
 FMUTIL directives, 21-1
 FMUTIL directory, 21-5, 21-6
 FMUTIL, dump file, 21-3
 FMUTIL, dump partition, 21-3
 FMUTIL error messages, A-27
 FMUTIL, load file, 21-3
 FMUTIL, load partition, 21-4
 foreground blank common module, 1-4
 foreground tasks, 2-14

FORM (JCP), 4-4
 format and dump (IOUTIL), 10-3
 format rotating memory (RAZI), 18-4
 FORT (JCP), 4-5
 FORTRAN IV compiler, 5-13
 FORTRAN IV compiler, error messages, A-10
 FORTRAN IV level-G enhancements, 5-13
 FORTRAN IV functions, 13-1
 FORTRAN compiler (JCP), 4-5
 FORTRAN-oriented firmware, 20-3
 FORTRAN program input/output operation, 1-5
 FORTRAN subprogram calls for process I/O, 19-9
 FRM (RAZI), 18-4
 FUNC (IOC), 3-14
 function (IOC), 3-4

G

GA (SEdit), 8-6
 gang-load all records (SEdit), 8-6
 generate communication network, 12-16
 global file control blocks, 4-9
 global FCB module, 1-4
 GO file, 1-4

H

handlers, 14-1
 hardware configurations, F-1
 hardware, minimum, 1-1
 HIST (PATCH), 15-3
 hooks, 14-3

I

identification buffer, 3-18
 IEEE STD 488-1975 Driver, 19-8
 INCLC directive (VSort), 11-4
 INCLF directive (VSort), 11-3
 INEXIT directive (VSort), 11-5
 initialize (FMAIN), 9-5
 initialize (RAZI), 18-4
 initialize background pointers, 4-2
 initialize memory (DEBUG), 7-1
 initialize peripheral devices, 18-1
 IN (SMAIN), 16-3
 INIT (FMAIN), 9-5
 INL (RAZI), 18-4
 INPUT (FMAIN), 9-6
 input logical unit (SMAIN), 16-3
 instruction mnemonics included in PATCH, 15-6
 integer math (32 bit), 20-16
 integers, storing, 13-1
 interface (PATCH), 16-1, 15-7

INDEX

Intermap DEBUG Program, 7-9
Internal mailbox element description, 14-45
interrupt handler, directly connected, 14-34
interrupt-processing task, exit, 14-2
interrupt-processing tasks, 14-1
Intertask Communication Module, 14-42
invoking a dump, 7-9, 7-11
IOC: input/output control, 3-1
IOLINK (RTE), 2-8
IOLIST (OPCOM), 17-5
IOUTIL: input/output utility program, 10-1, 6-7
IOUTIL, scheduling (JCP), 4-7
I/O algorithm, 14-28
I/O control, error messages, A-4
I/O devices, physical, 17-1, B-1
I/O driver, 14-30, 14-31
I/O errors by I/O device type, B-2
I/O interrupts, 3-5
I/O linkage, 2-8
I/O tables, 14-31
I/O utility (JCP), 4-7
I/O utility, error messages, A-18
ISA FORTRAN process control, 19-9
ITE, 14-42
ITE system generation requirement, 14-54

J

JCP: job-control processor, 4-1
JCP (ASSIGN), 4-3
JCP (COMSY), 22-1
JOB (JCP), 4-2
job-control processor, error messages, A-8
JCP (FMUTIL), 21-1
JCP (JOB), 4-2
JCP (LMGEN), 4-6
JCP (LOAD), 4-8
JCP (MEM), 4-3
JCP (PFILE), 4-4
JCP (position file), 4-4
JCP (REW), 4-4
JCP (rewind), 4-4
JCP (SFILE), 4-3
JCP (SMAIN), 4-7
JCP(SREC), 4-3

K

key-in requests, 17-1
keypunch mode (JCP), 4-4
KPMODE (JCP), 4-4

L

language processors, 5-1
LD (LMGEN), 6-4

LDBR (PATCH), 15-3
LI (SEDT), 8-6
LIB (LMGEN), 6-5
library (LMGEN), 6-5
line printer, 18-1
linkage, I/O with RTE, 2-8
LIST (FMAIN), 9-5
LIST (SMAIN), 16-6
list I/O (OPCOM), 17-5
list records (SEDT), 8-6
listing format (DASMR), 5-10
LMGEN (JCP), 4-6
LMGEN: load-module generator, 6-1
LMP: load module package, 16-3
LOAD (JCP), 4-8
load (LMGEN), 6-4
load directory (FMUTIL), 21-5
load file (FMUTIL), 21-3
load partition (FMUTIL), 21-4
load-module generation (COMSY), 22-11
load-module generator (JCP), 4-6
load-module generator directives, 6-4
load-module generator, error messages, A-12
loader, bootstrap, 18-2
lock bit, 9-3
logical memory, 1-4
logical record, 3-10
logical unit, 3-1, 8-2
LOT directive (VSORT), 1-5

M

magnetic-drum, (see RMD), 18-1
magnetic-tape modes, C-4
magnetic-tape, 18-1
mailbox entry description, 14-44
mailbox list description, 14-43
MANL (PATCH), 15-4
map memory, 1-4, 14-5
map 0 allocable memory, 1-4
map 0 nucleus, 1-3, 1-4
MAPIN, 14-5
MAPIN (RTE), 2-10
mask, 2-5
memory, 1-2
memory (JCP), 4-3
memory dump (DMEMRY), 7-12
memory map, 1-4, 1-5
memory protection interrupt, 14-3, 1-5
MEM (JCP), 4-3
message control block, 14-46, 14-48
microprogram assembler (MIDAS), error messages, A-23
microprogram simulator (MICSIM), error messages, A-24
microprogram utility (MIUTIL), error messages, A-25
micro-VORTEX and VORTEX II compatibility, 14-36
MICSIM, 20-1

MIDAS, 20-1
 MIUTIL, 20-1
 MO (SEDT), 8-5
 move records (SDEIT), 8-5
 MOVEC (VSORT), 11-5
 MOVEF (VSORT), 11-5
 moving-head disk, (see RMD), 18-1, 18-2
 multitask spool system, 23-1
 multitask spool system command summary, 23-1
 multi-volume tape having (V\$RSW), 10-5

N

named common, 6-3
 nine-track magnetic tape, C-4
 nucleus image, effect of PATCH on, 15-2
 nucleus, map 0, 1-3
 nucleus modules, 1-5
 nucleus pointers, 6-7
 nucleus programs module, 1-4
 nucleus table module, 1-4

O

object module formats, G-1
 OMITIC (VSORT), 11-4
 OMITF (VSORT), 11-4
 (OPCOM) operator communication, 17-1
 OPEN (IOC), 3-10
 operator communication, error messages, A-22
 OUT (SMAIN), 16-4
 OUTEXIT (VSORT), 11-5
 output logical unit (SMAIN), 16-4
 OV (LMGEN), 6-5
 overlay (RTE), 2-5
 overlays, 6-3
 OVLAY (RTE), 2-5

P

page 0, 1-4, 1-5
 pages, 1-5
 PAGNUM (RTE), 2-11
 paper-tape modes, C-1
 paper-tape reader, 18-1
 parity errors (V70/V77-600), 14-54
 parity errors (V77-200/400), 14-54
 parity errors (V77-600), 14-55
 parity errors (V77-800), 14-55
 partition, 3-4
 partition (FMUTIL), 21-3, 21-4
 partition (RAZI), 18-4

partition description listing, 18-3
 partition protection bit, 3-4
 partition specification table, 3-4
 partition specification table (PST), 3-4, 9-1
 partitions, 1-4
 partitions, 9-1
 partitions (RMD), 3-4
 PASS (RTE), 2-9
 pass buffer parameters, 2-8
 PATCH directive log file, clearing of (PATCH), 15-4
 PATCH directive log file (PATCH), 15-4, 15-5, 15-7
 patch image file, 15-1, 15-2, 15-7
 PATCH program, 15-1
 patching considerations, 15-8
 PFILE (IOUTIL), 10-4
 PFILE (JCP), 4-4
 physical I/O devices, 17-1
 physical memory, 1-5
 physical record, 3-10
 PLOT (generate plot), 12-6
 PIM interrupts, 2-5, 14-26
 PMSK (RTE), 2-5
 position file (IOUTIL), 10-4
 position file (JCP), 4-4
 post-interrupt processing, 14-31
 post SYSGEN requirements, 7-8
 power failure/restart interrupt, 14-3
 power-down, 14-3
 power-up, 14-3
 pre-interrupt processing, 14-30
 print contents of dump tape FMUTIL, 21-6
 print file (IOUTIL), 10-4
 print file directive FMUTIL, 21-6
 printer, line, 18-1
 priorities, 14-5
 priority 1 tasks, 1-5
 priority interrupt module (PIM), 14-1
 priority levels, 2-1
 PRNTF (IOUTIL), 10-4,
 programs module, nucleus, 1-4
 PRT (RAZI), 18-4
 pseudoregisters (DEBUG), 7-1

R

Random files (COMSY), 22-2
 RAZI: rotating memory analysis and initialization, 18-3
 READ (IOC), 3-12
 READ (F2963 DACS), 26-1
 read-only pages, 2-14
 real numbers, 13-1
 real-time clock, 14-28
 real-time clock interrupt, 14-4
 real-time clock interrupt processor, 14-22
 real-time executive, error messages, A-1

4/10/1914