VORTEX

REFERENCE MANUAL

Specifications are subject to change without notice. Address comments regarding this document to Varian Data Machines, Publications Department, 2722 Michelson Drive, Irvine, California, 92664.

varian data machines / a varian subsidiary
2722 michelson drive/p.o. box e/irvine/california/92664
 printed in USA

98 A 9952 103

SEPTEMBER 1974

FOREWORD

SECTION 1 INTRODUCTION

SECTION 2 REAL-TIME EXECUTIVE SERVICES

SECTION 3 INPUT/OUTPUT CONTROL

SECTION 4 JOB-CONTROL PROCESSOR

SECTION 5 LANGUAGE PROCESSORS

SECTION 6 LOAD-MODULE GENERATOR

SECTION 7 DEBUGGING AIDS

SECTION 8 SOURCE EDITOR

SECTION 9 FILE MAINTENANCE

SECTION 10 INPUT/OUTPUT UTILITY PROGRAM

SECTION 11 VSORT (SORT/MERGE)

SECTION 12 DATAPLOT II

SECTION 13 SUPPORT LIBRARY

SECTION 14 REAL-TIME PROGRAMMING

SECTION 15 SYSTEM GENERATION

SECTION 16 SYSTEM MAINTENANCE

SECTION 17 OPERATOR COMMUNICATION

SECTION 18 OPERATION OF THE VORTEX SYSTEM

SECTION 19 VORTEX PROCESS INPUT/OUTPUT

SECTION 20 WRITABLE CONTROL STORE AND FLOATING-POINT PROCESSOR

This manual explains the **Varian Omnitask Real-Time Executive (VORTEX)** and its use, but it is not intended for a beginning audience. Prerequisite to an understanding of this manual is a knowledge of general programming concepts, and preferably some Varian Data Machines 620 series or V70 series computer system is desirable.

NOTATION IN THIS MANUAL

In the directive formats given in this manual:

- **Boldface type** indicates an obligatory parameter.
- *Italic type* indicates an optional parameter.
- Upper case type indicates that the parameter is to be entered exactly as written.
- Lower case type indicates a variable and shows where the user is to enter a legal value for that variable.

a(1),a(2),...,a(n).

Indicates a series of elements separated by commas repeated and terminated with a period.

If at least one element is required the first element is given in bold. The parentheses are only part of the format description.

For example

a(1),a(2),...,a(n).

where
each a(i) is a single alphabetic character
allows

A,B,C,F,G,H.
or
Z,Y,X.
or
V.
as valid in this position.

A number with a leading zero is octal, one without a leading zero is decimal, and a number in binary is specifically indicated as such.

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 1-3
1.2.3 Secondary Storage 1-3
1.3 BIBLIOGRAPHY 1-4

SECTION 2
REAL-TIME EXECUTIVE SERVICES

2.1 REAL-TIME EXECUTIVE MACROS 2-1
2.1.1 SCHED (Schedule) Macro 2-1
2.1.2 SUSPND (Suspend) Macro 2-2
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 2-4
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 2-7
2.1.13 IOLINK (I/O Linkage) Macro 2-8
2.1.14 TBEVNT (Set or Fetch TBEVNT) Macro 2-8
2.2 ABORT PROCEDURE 2-9

SECTION 3
INPUT/OUTPUT CONTROL

3.1 LOGICAL UNITS 3-1
3.2 RMD FILE STRUCTURE 3-4
3.3 I/O INTERRUPTS 3-5
3.4 SIMULTANEOUS PERIPHERAL OUTPUT OVERLAP (SPOOL) 3-5
3.4.1 SPOOL Operation 3-6
3.4.2 SPOOL Files 3-6
3.5 I/O-CONTROL MACROS 3-7
3.5.1 OPEN Macro 3-9
3.5.2 CLOSE Macro 3-10

SECTION 3
INPUT/OUTPUT CONTROL *(continued)*

3.5.3 READ Macro 3-10
3.5.4 WRITE Macro 3-11
3.5.5 REW (Rewind) Macro 3-11
3.5.6 WEOF (Write End of File) Macro 3-12
3.5.7 SREC (Skip Record) Macro 3-12
3.5.8 FUNC (Function) Macro 3-12
3.5.9 STAT (Status) Macro 3-13
3.5.10 DCB (Data Control Block) Macro 3-14
3.5.11 FCB (File Control Block) Macro 3-14

SECTION 4
JOB-CONTROL PROCESSOR

4.1 ORGANIZATION 4-1
4.2 JOB-CONTROL PROCESSOR DIRECTIVES 4-1
4.2.1 /JOB Directive 4-2
4.2.2 /ENDJOB Directive 4-2
4.2.3 /FINI (Finish) Directive 4-2
4.2.4 /C (Comment) Directive 4-2
4.2.5 /MEM (Memory) Directive 4-2
4.2.6 /ASSIGN Directive 4-2
4.2.7 /SFILE (Skip File) Directive 4-3
4.2.8 /SREC (Skip Record) Directive 4-3
4.2.9 /WEOF (Write End of File) Directive 4-3
4.2.10 /REW (Rewind) Directive 4-3
4.2.11 /PFILE (Position File) Directive 4-3
4.2.12 /FORM Directive 4-4
4.2.13 /KPMODE (Keypunch mode) Directive 4-4
4.2.14 /DASMR (DAS MR Assembler) Directive 4-4
4.2.15 /FORT (FORTRAN Compiler) Directive 4-5
4.2.16 /CONC (System Concordance) Directive 4-5
4.2.17 /SEDIT (Source Editor) Directive 4-5
4.2.18 /FMAIN (File Maintenance) Directive 4-5
4.2.19 /LMGEN (Load-Module Generator) Directive 4-6
4.2.20 /IOUTIL (I/O Utility) Directive 4-6
4.2.21 /SMAIN (System Maintenance) Directive 4-6

SECTION 4
JOB-CONTROL PROCESSOR *(continued)*

4.2.22 /EXEC (Execute) Directive 4-6
4.2.23 /LOAD Directive 4-6
4.2.24 /ALTLIB (Alternate Library) Directive 4-7
4.2.25 /DUMP Directive 4-7
4.3 SAMPLE DECK SETUPS 4-7

SECTION 5
LANGUAGE PROCESSORS

5.1 DAS MR Assembler 5-1
5.1.1 TITLE Directive 5-1
5.1.2 VORTEX Macros 5-2
5.1.3 Assembly Listing Format 5-7
5.2 CONCORDANCE PROGRAM 5-8
5.2.1 Input 5-9
5.2.2 Output 5-9
5.3 FORTRAN IV COMPILER 5-10
5.3.1 TITLE Statement 5-10
5.3.2 Execution-Time I/O Units 5-11
5.3.3 Encode/Decode Functions 5-14
5.3.4 Runtime I/O Exceptions 5-14
5.3.5 Reentrant Runtime I/O 5-15
5.4 RPG IV COMPILER 5-15
5.4.1 Introduction 5-15
5.4.2 RPG IV I/O Units 5-15
5.4.3 Compiler and Runtime Execution 5-15

SECTION 6
LOAD-MODULE GENERATOR

6.1 ORGANIZATION 6-1
6.1.1 Overlays 6-3
6.1.2 Common 6-3
6.2 LOAD-MODULE GENERATOR DIRECTIVES 6-3
6.2.1 TIDB (Task-Identification Block) Directive 6-4
6.2.2 LD (Load) Directive 6-4
6.2.3 OV (Overlay) Directive 6-4
6.2.4 LIB (Library) Directive 6-4
6.2.5 END Directive 6-5
6.3 SAMPLE DECKS FOR LMGEN OPERATIONS 6-5

SECTION 7
DEBUGGING AIDS

7.1 DEBUGGING PROGRAM 7-1
7.2 SNAPSHOT DUMP PROGRAM 7-2

SECTION 8
SOURCE EDITOR

8.1 ORGANIZATION 8-1
8.2 SOURCE-EDITOR DIRECTIVES 8-2
8.2.1 AS (Assign Logical Units) Directive 8-2
8.2.2 AD (Add Records) Directive 8-3
8.2.3 SA (Add String) Directive 8-3
8.2.4 REPL (Replace Records) Directive 8-4
8.2.5 SR (Replace String) Directive 8-4
8.2.6 DE (Delete Records) Directive 8-4
8.2.7 SD (Delete String) Directive 8-5
8.2.8 MO (Move Records) Directive 8-5
8.2.9 FC (Copy File) Directive 8-5
8.2.10 SE (Sequence Records) Directive 8-6
8.2.11 LI (List Records) Directive 8-6
8.2.12 GA (Gang-Load All Records) Directive 8-6
8.2.13 WE (Write End of File) Directive 8-7
8.2.14 REWI (Rewind) Directive 8-7
8.2.15 CO (Compare Inputs) Directive 8-7
8.3 EXAMPLE OF EDITING A FILE 8-7

SECTION 9
FILE MAINTENANCE

9.1 ORGANIZATION 9-1
9.1.1 Partition Specification Table 9-1
9.1.2 File-Name Directory 9-1
9.1.3 Relocatable Object Modules 9-2
9.1.4 Output Listings 9-2
9.2 FILE-MAINTENANCE DIRECTIVES 9-2
9.2.1 CREATE Directive 9-3
9.2.2 DELETE Directive 9-3
9.2.3 RENAME Directive 9-4
9.2.4 ENTER Directive 9-4
9.2.5 LIST Directive 9-4
9.2.6 INIT (Initialize) Directive 9-4
9.2.7 INPUT Directive 9-5
9.2.8 ADD Directive 9-5

SECTION 10
INPUT/OUTPUT UTILITY PROGRAM

10.1 ORGANIZATION 10-1
10.2 I/O UTILITY DIRECTIVES 10-1
10.2.1 COPYF (Copy File) Directive 10-1
10.2.2 COPYR (Copy Record) Directive 10-2
10.2.3 SFILE (Skip File) Directive 10-3
10.2.4 SREC (Skip Record) Directive 10-3
10.2.5 DUMP (Format and Dump) Directive 10-3
10.2.6 PRNTF (Print File) Directive 10-4
10.2.7 WEOF (Write End of File) Directive 10-4
10.2.8 REW (Rewind) Directive 10-4
10.2.9 PFILE (Position File) Directive 10-4
10.2.10 CFILE (Close File) Directive 10-5
10.2.11 PACKB (Pack Binary) Directive 10-5

SECTION 11
VSORT (SORT/MERGE)

11.1 ORGANIZATION 11-1
11.2 VSORT DIRECTIVES 11-1
11.2.1 SORT Directive 11-2
11.2.2 INPUT Directive 11-2
11.2.3 OUTPUT Directive 11-2
11.2.4 WORK1,WORK2,WORK3, Directives 11-2
11.2.5 SORTKEY Directive 11-2
11.2.6 INEXIT Directive 11-3
11.2.7 OUTEXIT Directive 11-3
11.2.8 ENDSORT Directive 11-3
11.3 USER EXITS 11-3
11.3.1 Calling Sequence 11-3
11.3.2 Implementation 11-4
11.4 VSORT MESSAGES 11-4

SECTION 12
DATAPLOT II

12.1 SYSTEM FLOW OUTLINE 12-1
12.2 HARDWARE REQUIREMENTS 12-1
12.3 GENERAL DESCRIPTION 12-1
12.3.1 DATAPLOT II Organization 12-1
12.3.2 System Considerations 12-3
12.3.3 VORTEX Considerations 12-3

SECTION 12
DATAPLOT II *(continued)*

12.4 DATAPLOT II FUNCTIONS 12-4
12.4.1 DPINIT (System File Initialization) 12-5
12.4.2 PLOTS (Work Buffer Initialization) 12-5
12.4.3 PLOT (Generate Plot) 12-5
12.4.4 SCALE (Generates Scale Factor) 12-6
12.4.5 AXIS (Generate Segmental Axis) 12-7
12.4.6 SYMBOL (Generate Symbols) 12-8
12.4.7 NUMBER (Generate Number) 12-9
12.4.8 LINE (Generate Graph Line) 12-10
12.4.9 MLTPLE (Multiple Plot) 12-11
12.4.10 FACTOR (Alter Plot Size) 12-11
12.4.11 WHERE (Locate Coordinates) 12-11
12.4.12 APPEND (Append File) 12-12
12.4.13 TOPFRM (Top-of-Form) 12-12
12.4.14 CUT (Cut Paper) 12-12
12.4.15 ENDCUT (Eject and Cut Paper) 12-13
12.4.16 DPSORT (Sort Plot File) 12-13
12.4.17 DPPLOT (Output File) 12-13
12.4.18 DPCLOS (Close Plot File) 12-14
12.4.19 ORIG -- Offsetting the Origin Entry Point 12-14
12.4.20 VECT -- Vector Entry Point 12-14
12.4.21 Special SYMBOL Subroutine 12-15
12.5 PLOT FILE DATA FORMAT 12-15
12.5.1 Vectors 12-15
12.5.2 Characters 12-15
12.5.3 End-of-Plot Indicator 12-16
12.6 EXAMPLE OF APPLICATION OF DATAPLOT II 12-16
12.6.1 Program to Generate Sine Wave 12-16
12.6.2 Program to Generate Communication Network 12-16
12.7 OPERATING PROCEDURES AND ERROR MESSAGES 12-17
12.7.1 VORTEX Operating Procedures 12-17
12.7.2 Unsorted Plot Files 12-17
12.7.3 Presorted Plot Files 12-17
12.7.4 VORTEX Special Procedures 12-17

SECTION 13
SUPPORT LIBRARY

13.1 CALLING SEQUENCE 13-1
13.2 NUMBER TYPES AND FORMATS 13-1
13.3 SUBROUTINE DESCRIPTIONS 13-2

SECTION 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 Task Installation 14-3
14.2 SCHEDULING 14-4
14.2.1 System Flow 14-4
14.2.2 Priorities 14-4
14.2.3 Timing Considerations (Approximate) 14-19
14.3 REENTRANT SUBROUTINES 14-20
14.4 CODING AN I/O DRIVER 14-21
14.4.1 I/O Tables 14-21
14.4.2 I/O Driver System Functions 14-21
14.4.3 Adding an I/O Driver to the System File 14-22
14.4.4 Enabling and Disabling PIM Interrupts 14-24
14.4.5 Directly Connected Interrupt Handler 14-25
14.4.6 VORTEX use of BICs 14-25

SECTION 15
SYSTEM GENERATION

15.1 ORGANIZATION 15-1
15.2 SYSTEM-GENERATION LIBRARY 15-2
15.3 KEY-IN LOADER 15-5
15.4 SGEN I/O INTERROGATION 15-6
15.4.1 DIR (Directive-Input Unit) Directive 15-6
15.4.2 LIB (Library-Input Unit) Directives 15-6
15.4.3 ALT (Library-Modification Input Unit) Directive 15-6
15.4.4 SYS (System-Generation Output Unit) Directive 15-7
15.4.5 LIS Directive 15-7
15.5 SGEN Directive Processing 15-7
15.5.1 MRY (Memory) Directive 15-8
15.5.2 EQP (Equipment) Directive 15-8
15.5.3 PRT (Partition) Directive 15-10
15.5.4 ASN (Assign) Directive 15-10
15.5.5 ADD (SGL Addition) Directive 15-12
15.5.6 REP (SGL Replacement) Directive 15-12
15.5.7 DEL (SGL Deletion) Directive 15-13
15.5.8 LAD (Library Addition) Directive 15-13
15.5.9 LRE (Library Replacement) Directive 15-13
15.5.10 LDE (Library Deletion) Directive 15-14
15.5.11 PIM (Priority Interrupt) Directive 15-14

SECTION 15
SYSTEM GENERATION *(continued)*

15.5.12 CLK (Clock) Directive 15-14
15.5.13 TSK (Foreground Task) Directive 15-15
15.5.14 DEF (Define External) Directive 15-15
15.5.15 EDR (End Redefinition) Directive 15-15
15.5.16 Required Directives 15-16
15.6 BUILDING THE VORTEX NUCLEUS 15-16
15.6.1 SLM (Start Load Module) Directive 15-16
15.6.2 TDF (Build Task-Identification Block) Directive 15-16
15.6.3 END Directive 15-17
15.6.4 Memory Parity Considerations 15-17
15.7 BUILDING THE LIBRARY CONFIGURATOR 15-17
15.7.1 SLM (Start LMP) Directive 15-19
15.7.2 TID (TIDB Specification) Directive 15-19
15.7.3 OVL (Overlay) Directive 15-19
15.7.4 ESB (End Segment) Directive 15-20
15.7.5 END (End Library) Directive 15-20
15.8 SYSTEM INITIALIZATION AND OUTPUT LISTINGS 15-20
15.9 SYSTEM GENERATION EXAMPLES 15-21

SECTION 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 16-4
16.2.4 ADD Directive 16-4
16.2.5 REP (Replace) Directive 16-5
16.2.6 DEL (Delete) Directive 16-5
16.2.7 LIST Directive 16-6
16.2.8 END Directive 16-7
16.3 SYSTEM-MAINTENANCE OPERATION 16-7
16.4 PROGRAMMING EXAMPLES 16-7

SECTION 17
OPERATOR COMMUNICATION

17.1 DEFINITIONS 17-1
17.2 OPERATOR KEY-IN REQUESTS 17-1
17.2.1 ;SCHED (Schedule Foreground Task) Key-In Request 17-2
17.2.2 ;TSCHED (Time-Schedule Foreground Task) Key-In Request 17-2
17.2.3 ;ATTACH Key-In Request 17-2
17.2.4 ;RESUME Key-In Request 17-3
17.2.5 ;TIME Key-In Request 17-3
17.2.6 ;DATE Key-In Request 17-3
17.2.7 ;ABORT Key-In Request 17-3
17.2.8 ;TSTAT (Task Status) Key-In Request 17-3
17.2.9 ;ASSIGN Key-In Request 17-4
17.2.10 ;DEVDN (Device Down) Key-In Request 17-5
17.2.11 ;DEVUP (Device Up) Key-In Request 17-5
17.2.12 ;IOLIST (List I/O Key-In Request 17-5

SECTION 18
OPERATION OF THE VORTEX SYSTEM

18.1 DEVICE INITIALIZATION 18-1
18.1.1 Card Reader 18-1
18.1.2 Card Punch 18-1
18.1.3 Line Printer 18-1
18.1.4 Statos-31 *(Model 70-66XX)* 18-1
18.1.5 33/35 ASR Teletype 18-1
18.1.6 High-Speed Paper-TApe Reader 18-1
18.1.7 Magnetic-Tape Unit 18-1
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
18.1.11 Moving-Head Disc Units 18-2
18.2 SYSTEM BOOTSTRAP LOADER 18-2
18.2.1 Automatic Bootstrap Loader 18-2
18.2.2 Control Panel Loading 18-2
18.3 DISC PACK HANDLING 18-3
18.3.1 PRT (Partition) Directive 18-4
18.3.2 FRM (Format Rotating Memory) Directive 18-4
18.3.3 INL (Initialize) Directive 18-4
18.3.4 EXIT Directive 18-4
18.4 70-7500 (620-35) DISC PACK FORMATTING PROGRAM 18-5

SECTION 18
OPERATION OF THE VORTEX SYSTEM *(continued)*

18.5 70-7510 (620-34) DISC PACK FORMATTING PROGRAM 18-5
18.6 WRITABLE CONTROL STORE (WCS) 18-6

SECTION 19
PROCESS INPUT/OUTPUT

19.1 INTRODUCTION 19-1
19.2 PROCESS OUTPUT 19-1
19.2.1 Hardware 19-1
19.2.2 SGEN Operations 19-1
19.2.3 Output Calls 19-2
19.3 PROCESS INPUT 19-3
19.3.1 Hardware 19-3
19.3.2 SGEN Operations 19-3
19.3.3 Input Calls 19-4
19.3.4 Low-Level Multiplexor Gain Control 19-5
19.4 ISA FORTRAN PROCESS CONTROL SUBROUTINES 19-6
19.4.1 Input/Output Calls 19-6
19.4.2 Bit String Operations 19-8
19.5 ERRORS 19-8
19.6 EXTENSIONS 19-8

SECTION 20
WRITABLE CONTROL STORE AND FLOATING-POINT PROCESSOR

20.1 MICROPROGRAMMING SOFTWARE 20-1
20.1.1 Microprogram Assembler 20-1
20.1.2 Microprogram Simulator 20-1
20.1.3 Microprogram Utility 20-1
20.1.4 WCS Reload Task, WCSRLD 20-2
20.2 STANDARD FIRMWARE 20-2
20.2.1 Fixed-Point Arithmetic Firmware 20-2
20.2.2 Floating-Point Arithmetic Firmware 20-2
20.2.3 Data Transfer Firmware 20-2
20.2.4 FORTRAN-Oriented Firmware 20-2
20.2.5 Byte Manipulation Firmware 20-3
20.2.6 Stack Firmware 20-3
20.2.7 Firmware Macros 20-6

APPENDIX A
ERROR MESSAGES

A.1 ERROR MESSAGE INDEX A-1
A.2 REAL-TIME EXECUTIVE A-1
A.3 I/O CONTROL A-3
A.4 JOB-CONTROL PROCESSOR A-6
A.5 LANGUAGE PROCESSORS A-6
A.5.1 DAS MR Assembler A-6
A.5.2 FORTRAN IV Compiler and Runtime Compiler A-8
A.5.3 RPG IV Compiler and Runtime Compiler A-8
A.6 LOAD-MODULE GENERATOR A-10
A.7 DEBUGGING PROGRAM A-11
A.8 SOURCE EDITOR A-11
A.9 FILE MAINTEANCE A-12
A.10 I/O UTILITY A-13
A.11 SORT ERROR MESSAGES A-13
A.12 DATAPLOT A-14
A.13 SUPPORT LIBRARY A-14
A.14 REAL-TIME PROGRAMMING A-14
A.15 SYSTEM GENERATION A-15
A.16 SYSTEM MAINTENANCE A-19
A.17 OPERATOR COMMUNICATION A-20
A.18 RMD ANALYSIS AND INITIALIZATION A-20
A.18.1 Microprogram Simulator A-21
A.18.2 Microprogram Utility A-22
A.19 PROCESS INPUT/OUTPUT A-23
A.20 VTAM NETWORK CONTROL MODULE A-23
A.21 ERROR CODES A-24
A.21.1 Errors Related to Directives A-24
A.21.2 Errors Related to Programs A-24
A.21.3 Errors Related to Memory Size A-25
A.21.4 Errors Related to Hardware A-25

APPENDIX B
I/O DEVICE RELATIONSHIPS

APPENDIX C
DATA FORMATS

C.1 PAPER TAPE C-1
C.1.1 Binary Mode C-1
C.1.2 Alphanumeric Mode C-1
C.1.3 Unformatted Mode C-1
C.1.4 Special Characters C-1
C.2 Cards C-2

APPENDIX C
DATA FORMATS *(continued)*

C.2.1 Binary Mode C-2
C.2.2 Alphanumberic Mode C-2
C.2.3 Unformatted Mode C-4
C.2.4 Special Character C-4
C.3 MAGNETIC TAPE C-4
C.3.1 Seven-Track C-4
C.3.2 Nine-Track C-4
C.4 STATOS PRINTER/PLOTTER C-4
C.4.1 Alphanumeric Mode C-4
C.4.2 Unformatted Mode C-4

APPENDIX D
STANDARD CHARACTER CODES

APPENDIX E
ASCII CHARACTER CODES

APPENDIX F
VORTEX HARDWARE CONFIGURATIONS

APPENDIX G
OBJECT MODULE FORMAT

G.1 RECORD STRUCTURE G-1
G.2 PROGRAM IDENTIFICATION BLOCK G-1
G.3 DATA FIELD FORMATS G-1
G.4 LOADER CODES G-1
G.5 EXAMPLE G-3
G.5.1 Source Module G-3
G.5.2 Object Module G-3
G.5.3 Core Image G-5

LIST OF ILLUSTRATIONS

Figure 1-1. VORTEX System Flow 1-2
Figure 1-2. VORTEX Computer Memory Map 1-3
Figure 1-3. VORTEX RMD Storage Map 1-3
Figure 3-1. Spooling Subsystem Flow 3-6
Figure 5-4. FORTRAN I/O Execution Sequences 5-1
Figure 6-1. Load-Module Overlay Structure 6-2
Figure 12-1. DATAPLOT II Graphics System Data Flow 12-2
Figure 12-2. DATAPLOT II Organization 12-2
Figure 12-3. Minimum and Maximum Plot Values 12-4
Figure 12-4. +x Axis and +y Axis Relative to Paper Direction 12-14
Figure 12-5. Vector-Data Format 12-15
Figure 12-6. Character Data Format 12-15
Figure 12-7. Character Orientation Data Format 12-15
Figure 12-8. End-of-Plot Indicator 12-16
Figure 12-9. Sine Wave Plot Generated by DATAPLOT II 12-16
Figure 12-10. Communication Network Plot Generated by DATAPLOT II 12-17
Figure 14-1. Interrupt Line Handlers 14-2
Figure 14-2. VORTEX Memory Map 14-5
Figure 14-3. VORTEX Priority Structure 14-6
Figure 14-4. TIDB Description 14-7
Figure 14-5. Driver Interface 14-24
Figure 15-1. SGEN Data Flow 15-1
Figure 15-2. System-Generation Library 15-3
Figure 15-3. VORTEX Nucleus 15-3
Figure 15-4. Load-Module Library 15-4
Figure 15-5. Load Module Package for Module Without Overlays 15-17
Figure 15-6. Load Module Package for Module With Overlays 15-19
Figure 15-7. VORTEX Nucleus Load Map 15-20
Figure 15-8. Library Processor Load Map 15-21
Figure 15-9. RMD Partition Listing 15-21
Figure 15-10. Resident-Task Load Map 15-21
Figure 16-1. SMAIN Block Diagram 16-1
Figure 16-2. SMAIN LIST Directive Listing 16-6
Figure 20-1. Base and Limit of Stack 20-3
Figure 20-2. Stack Control Block 20-4
Figure 20-3. Stack Multiply 20-4
Figure 20-4. Stack Divide 20-4
Figure 20-5. Stack Push 20-5
Figure 20-5. Stack Pop 20-5
Figure 20-5. Stack Double Push 20-5
Figure 20-8. Stack Double Pop 20-5
Figure C-1. Paper Tape Binary Record Format C-1
Figure C-2. Paper Tape Alphanumeric Record Format C-2
Figure C-3. Card Binary Record Format C-3
Figure C-4. Card Alphanumeric Record Format (IBM 026) C-3

LIST OF TABLES

Table 2-1. RTE Service Request Macros 2-1
Table 3-1. VORTEX Logical-Unit Assignments 3-1
Table 3-1. VORTEX Logical-Unit Assignments *(continued)* 3-2
Table 3-1. VORTEX Logical-Unit Assignments *(continued)* 3-3
Table 3-2. Valid Logical-Unit Assignments 3-3
Table 3-3. FCB Words Under I/O Macro Control 3-14
Table 5-1. Directives Recognized by the DAS MR Assembler 5-1
Table 5-2. RTE Macros Available Through FORTRAN IV 5-10
Table 7-1. DEBUG Directives 7-1
Table 13-1. DAS Coded Subroutines 13-2
Table 13-1. DAS Coded Subroutines *(continued)* 13-3
Table 13-1. DAS Coded Subroutines *(continued)* 13-4
Table 13-1. DAS Coded Subroutines *(continued)* 13-5
Table 13-1. DAS Coded Subroutines *(continued)* 13-6
Table 13-2. FORTRAN IV Coded Subroutines 13-6
Table 13-2. FORTRAN IV Coded Subroutines *(continued)* 13-8
Table 13-2. FORTRAN IV Coded Subroutines *(continued)* 13-9
Table 14-1. TIDB Description 14-8
Table 14-1. TIDB Description *(continued)* 14-9
Table 14-1. TIDB Description *(continued)* 14-10
Table 14-1. TIDB Description *(continued)* 14-11
Table 14-2. Map of Lowest Memory Sector 14-11
Table 14-2. Map of Lowest Memory Sector *(continued)* 14-12
Table 14-2. Map of Lowest Memory Sector *(continued)* 14-13
Table 14-2. Map of Lowest Memory Sector *(continued)* 14-14
Table 14-2. Map of Lowest Memory Sector *(continued)* 14-15
Table 14-2. Map of Lowest Memory Sector *(continued)* 14-16
Table 14-2. Map of Lowest Memory Sector *(continued)* 14-17
Table 14-2. Map of Lowest Memory Sector *(continued)* 14-18
Task A 14-20
Task B 14-20
Table 15-1. SGEN Key-In Loaders 15-5
Table 15-2. Model Codes for VORTEX Peripherals 15-8
Table 15-2. Model Codes for VORTEX Peripherals *(continued)* 15-9
Table 15-3. Preset Logical-Unit Assignments 15-11
Table 15-4. Permissible Logical-Unit Assignments 15-12
Table 15-5. TIDB Status-Word Bits 15-18
Table 17-1. Physical I/O Devices 17-1
Table 17-2. Task Status (TIDB Words 1 and 2) 17-4
Table 18-1. Key-In Loader Programs 18-2

SECTION 1
INTRODUCTION

The **V**arian **O**mnitask **R**eal-**T**ime **EX**ecutive (**VORTEX**) is a modular software operating system for controlling, scheduling, and monitoring tasks in real-time multiprogramming environment. VORTEX also provides for background operations such as compilation, assembly, debugging, or execution of tasks not associated with the real-time functions of the system. Thus, the basic features of VORTEX comprise:

- 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

1.1 SYSTEM REQUIREMENTS

VORTEX requires the following minimum hardware configuration:

a. Varian 620/f, 620/f-100 or V70 series computers with 16K read/write memory (24K for foreground and background usage)

b. 33/35 ASR Teletype on a priority interrupt module

c. Real-time clock (standard on V70 series computers)

d. Memory protection (standard on V70 series computers)

e. Power failure/restart (standard on V70 series computers)

f. Priority Interrupt Module (PIM)

g. Rotating memory device (RMD) on a PIM with either a buffer interlace controller (BIC) or priority memory access (PMA)

h. 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

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

a. Additional main memory (up to 32K) and/or rotating memory

b. Additional rotating memory devices

c. Automatic bootstrap loader

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

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 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 in this manual.

1.2.1 Computer Memory

The VORTEX operating system divides computer (main) memory into five areas (figure 1-2):

a. Real-time executive area

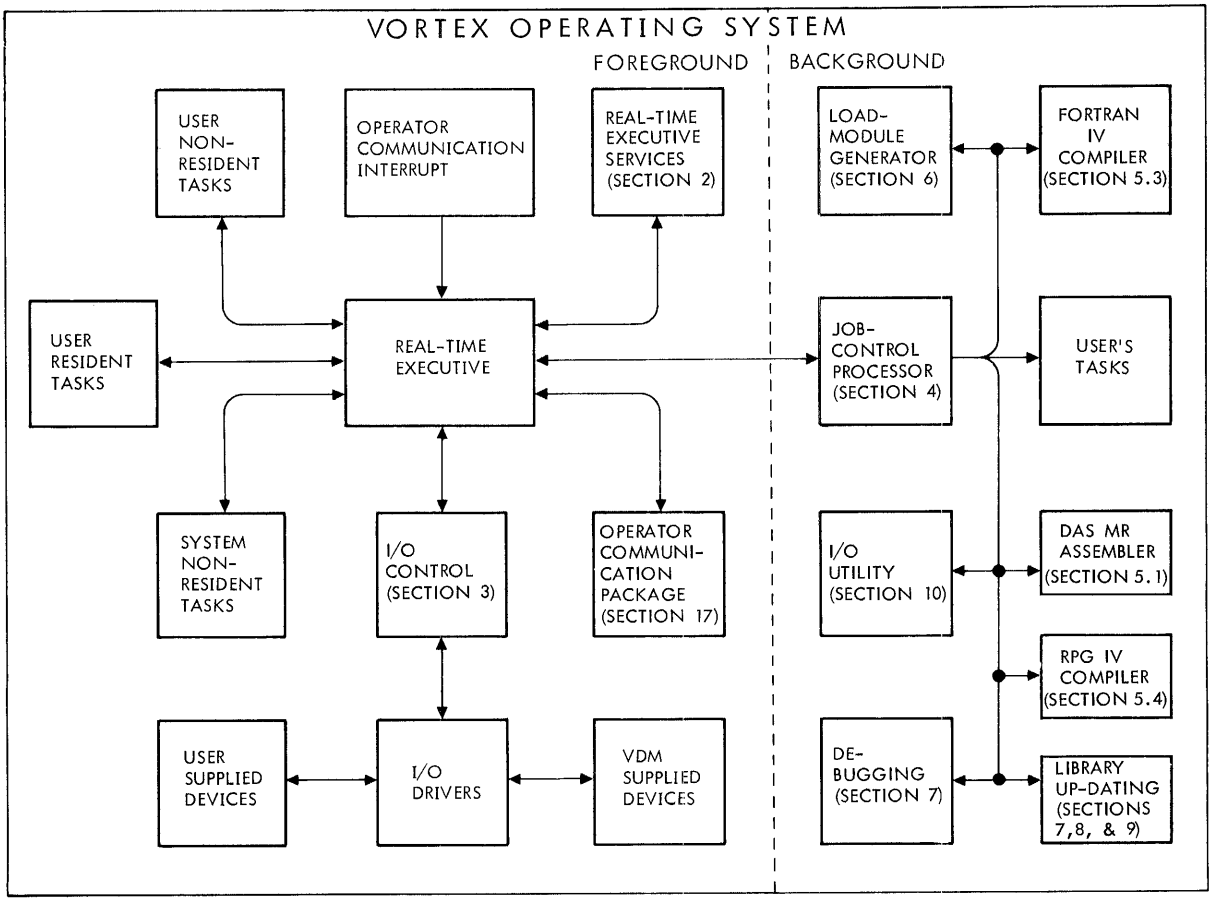
b. User's resident task and subroutine area

c. User's nonresident task allocation area

d. Background task area

e. Low-memory block area

The **real-time executive area** is the highest segment of memory. It contains the real-time executive, the I/O control component, I/O drivers, the load-module loader, interrupt processors, and the foreground blank common (section 6). All subroutines that reside in this area must be declared at system-generation time because no modification of the area is possible at run time. (Maintenance of the foreground blank common is a user responsibility. The



VTII-1314 A

Figure 1-1. VORTEX System Flow

VORTEX system provides blank-common pointers for use by the load-module generator.)

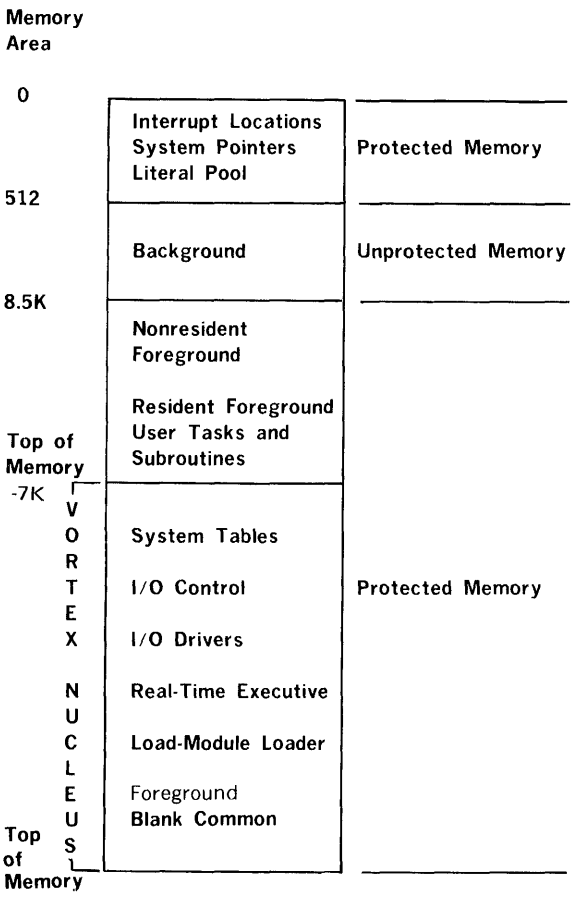


Figure 1-2. VORTEX Computer Memory Map

The **user's resident task and subroutine area** is adjacent to the real-time executive area. All resident foreground subroutines must be declared at system-generation time because no modification of the area is possible at run time.

The **user's nonresident task allocation area** is for the execution of tasks that reside on the RMD in the form of load modules, i.e., fully link-edited, but relocatable. When such a task is to be executed, it is loaded into this area and activated. If no nonresident foreground area is available for loading this task, background area is used, the background task being suspended and stored on the RMD. When the background area is again free, the background task is reloaded and resumed.

The **background task area** is for the execution of tasks that are less time-critical, such as compilers, assemblers, editors, and other general-purpose tasks. *Note that this area is the only unprotected area of memory.* Tasks executing in this area cannot modify the system, i.e., this area is suitable for the execution of undebugged tasks.

The **low-memory block area** contains system pointers and tables, interrupt addresses, and the background literal pool.

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 (section 15).

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

1.2.3 Secondary Storage

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

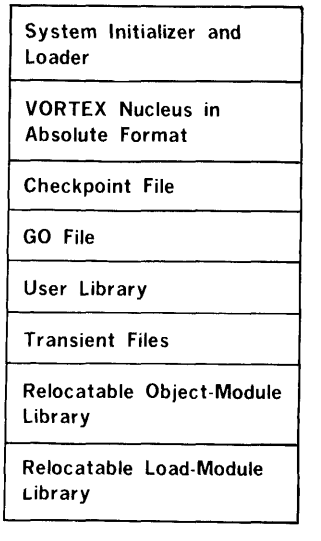


Figure 1-3. VORTEX RMD Storage Map

1.3 BIBLIOGRAPHY

The following gives the stock numbers of Varian manuals pertinent to the use of VORTEX and the V70/620 computers:

Title	Document Number
V72 Handbook	98 A 9906 20x
V73 Handbook	98 A 9906 01x
620-100 Computer Handbook	98 A 9905 00x
FORTRAN IV Reference Manual	98 A 9902 03x
RPG IV User's Manual	98 A 9947 03x
VTAM Reference Manual	98 A 9952 22x
HASP/RJE Operator's Manual	09 A 9952 21x
Microprogramming Guide	98 A 9952 21x
Vortex Installation Manual	98 A 9906 07x

Where x is a revision level number subject to change.

Maintenance information is in the following VORTEX Software Performance Specifications:

Document Number	Title
89A0156	System Overview
89A0203	External Specification
89A0231	Internal Specification, Vol. I
89A0232	Internal Specification, Vol. II
89A0233	Internal Specification, Vol. III
89A0225	DAS MR Assembler Internal
89A0214	FORTRAN IV Compiler Internal
89A0211	FORTRAN IV Library Internal
89A0246	FORTRAN IV Runtime I/O Internal
89A0234	RPG IV Runtime/Loader Internal
89A0184	RPG IV Compiler Internal

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.

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.

Background and foreground RTE service requests are similar. However, a level 0 background RTE request causes a memory-protection interrupt and the request is checked for validity. If there is an error, the system prints the error message **EX11** with the name of the task and the location of the violation of memory protection. The background task is aborted.

Table 2-1. RTE Service Request Macros

Mnemonic	Description	Level 0	FORTRAN
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
DEALOC	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

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

There are 12 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)

```
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 the

two values in the A and B registers to the scheduled task. The macro has the general form

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

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

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

xxyyzz 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',''

The FORTRAN calling sequence for this macro is

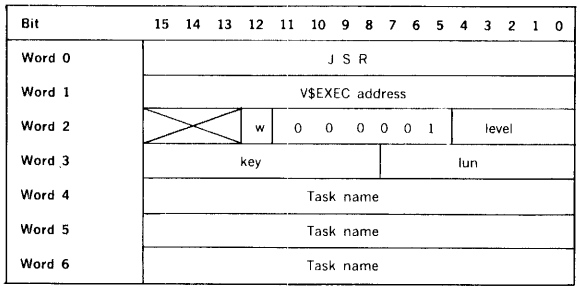
CALL SCHED(level,*wait*,**lib,***key*,**name)**

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 containing the values 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 task, a foreground task, 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).



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(3),N2(3)
DATA N1(1)/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 interrupt or a RESUME (section 2.1.4) macro. The macro has the general form

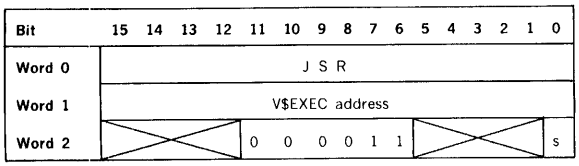
label **SUSPND susp**

where **susp** is 0 if the task is to be resumed by RESUME, or 1 if the task is to be resumed by interrupt.

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

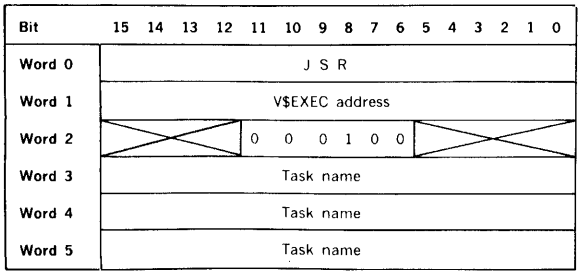
The RTE searches for the named task and activates it when found. The task will execute when it becomes the task with the highest active priority. If the priority of the specified task is higher than that of the task making the request, the specified task executes before the requesting task and immediately 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, after the specified delay be automatically rescheduled from the foreground library in a time-of-day 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

The FORTRAN calling sequence for this macro is

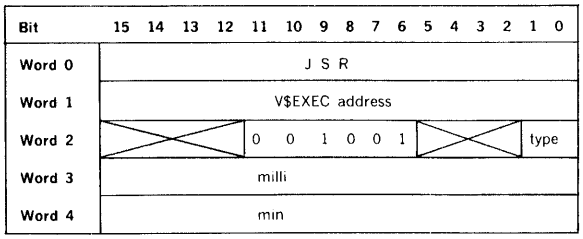
CALL DELAY(milli,min,type)

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 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 milleseconds. 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. Time is kept on a 24-hour clock.

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
```

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

key is the protection code for the logical unit

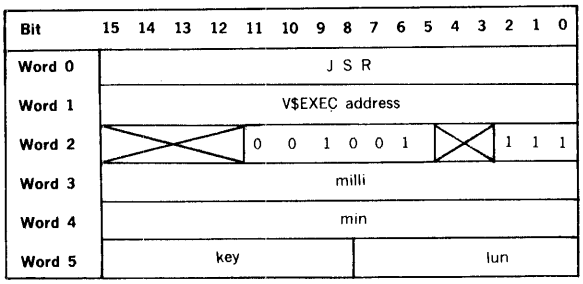
The FORTRAN calling sequence for this macro is

CALL LDELAY (milli,min,lun,key)

where the integer-mode parameters have the definitions given in the assembly-language form of the call.

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)

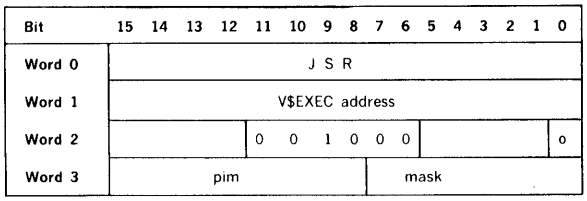
where the integer-mode parameters have the definitions given above.

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.

Expansion: The *opt* flag is bit 0 of word 2 (o).



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

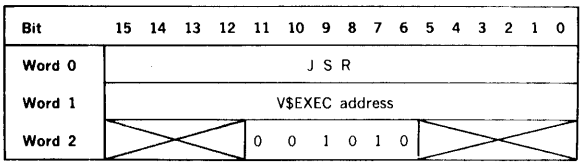
label **TIME**

The FORTRAN calling sequence for this macro is

CALL TIME(min,milli)

where **min** is the hours and minutes in 1-minute integer increments, and **milli** is the seconds in 5-millisecond integer increments.

Expansion: The *opt* flag is bit 0 of word 2 (o).



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

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

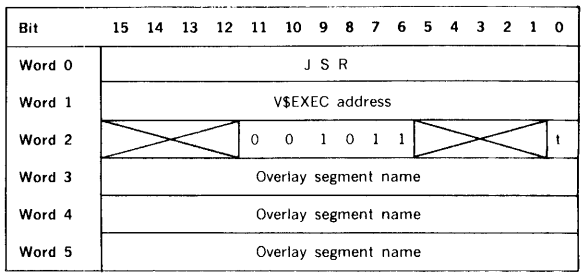
The FORTRAN calling sequence for this macro is

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 a 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$OPEN for CALL V$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 ALOC(subr)

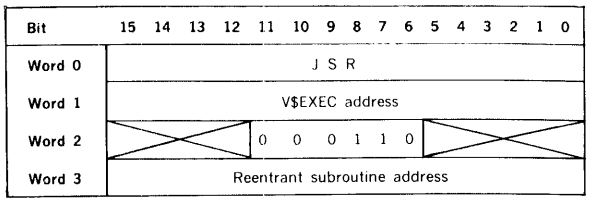
where **subr** is the name of the DAS MR assembly language subroutine.

The first location of the LIFO stack is V$LOC, and that of the current position in the stack is V$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 2.1.8) 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.

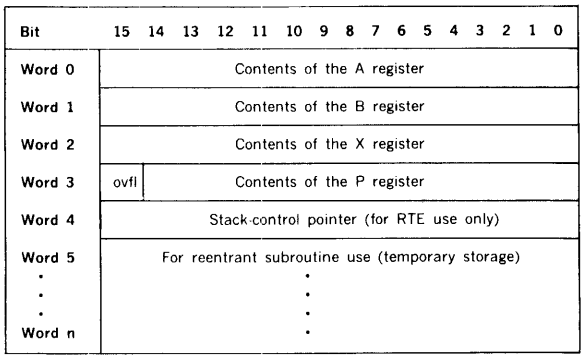
Expansion:



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:



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     SUB1
        ALOC    SUB1
                (Return Control)
        .
        .
        NAME    SUB1
SUB1    DATA    6
        .
        .
        .
        DEALOC
        END
```

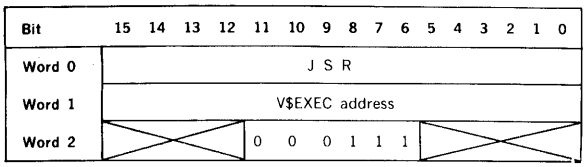
Each time SUB1 is called, six words are reserved in the reentrant stack. Each time the reentrant subroutine makes a DEALOC request (section 2.1.8), six words are deallocated from the reentrant stack.

2.1.10 DEALOC (Deallocate) Macro

This macro deallocates the current reentrant stack, restores the contents of the A and B 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 2.1.7). The macro has the form

label **DEALOC**

Expansion:



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

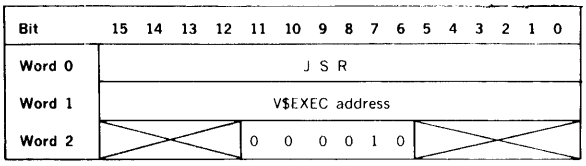
label **EXIT**

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:



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

```
        .
        .
        .
        EXIT            (No return)
```

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'**

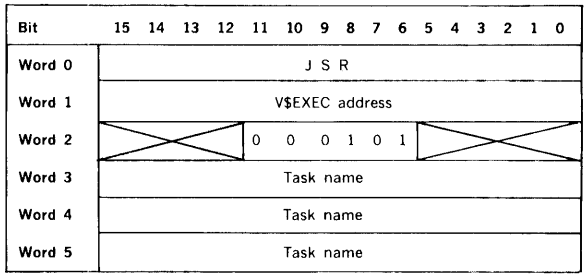
where **xxyyzz** is the name of the task being aborted, coded as in the SCHED macro (section 2.1.1).

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)

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 4.2.12) 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.

Expansion:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 0	J S R															
Word 1	V$EXEC address															
Word 2						0	0	1	1	0	0	lungsd				
Word 3	bufloc															
Word 4	bufsiz															

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 TBEVNT (Set or Fetch TBEVNT) Macro

This macro fetches or sets the requesting task's event word, TBEVNT, word 3 of the TIDB. It can also be used to change other words in the TIDB. However, most changes to entries in the TIDB could cause irrecoverable errors, so the TBEVNT macro should be used only with caution. Section 14 gives information about the format and contents of the TIDB.

This macro has the general form

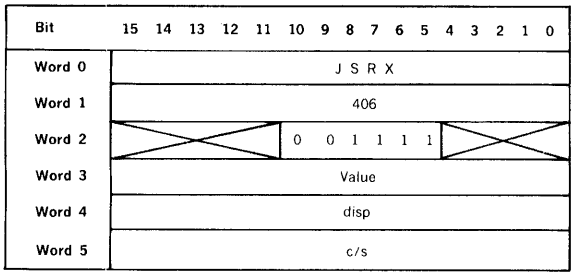
label **TBEVNT** **value,***disp*,c/s

where

value is a value or bit mask for the specified TIDB word. If *disp* is 0, value 0 - 0177776 changes the TBEVNT word and a value of 0177777 fetches the TBEVNT contents into the A register. If *disp* is not zero, it sets or resets (depending on c/s) the word specified by *disp*

disp is the displacement of the word in the TIDB to be set or reset, or 0 for TBEVNT (word 3). The default is 0.

c/s is the clear or set indicator, if *disp* is not 0. c/s = 0 for clear (the zero bits of the **value** indicate the bits of the specified word to be set to 0) and 1 for clear (the one bits in **value** indicate the bits to be set to 1). Default is 0



Default values: disp = 0 c/s = 0

Example: Save the value of TBEVNT in TEMP then set TBEVNT to 02.

```
START    TBEVNT   0177777
         STA      TEMP     (Save TBEVNT)
         TBEVNT   02       (Set TBEVNT= 2)
         .
         .
         .
TEMP     BSS      1
```

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

```
TBEVNT   0177377,2,0
TBEVNT   0400,2,1
```

2.2 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.

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.

VORTEX logical-unit assignments for all systems are specified in table 3-1. All logical-unit numbers that are not listed are available to the reassignability scheme above.

Table 17-1 shows the scheme of system names for physical devices. Table 3-2 shows the possible logical-unit assignments.

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; Teletype or CRT only
4	PI	Processor input	For input of source statements from all operating system language processors

(continued)

Table 3-1. VORTEX Logical-Unit Assignments

(continued)

Number	Name	Description	Function
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 only
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
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

Table 3-1. VORTEX Logical-Unit Assignments
(continued)

Number	Name	Description	Function
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	**OC**	**SI**	**SO**	**PI**	**LO**	**BI**	**BO**	**SS**	**GO**
Unit No.	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**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	**PO**	**DI**	**DO**	**CU**	**SW**	**CL**	**OM**	**BL**	**FL**
Unit No.	**10**	**11**	**12**	**101**	**102**	**103**	**104**	**105**	**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						

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															
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.

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.

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 throughout 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.

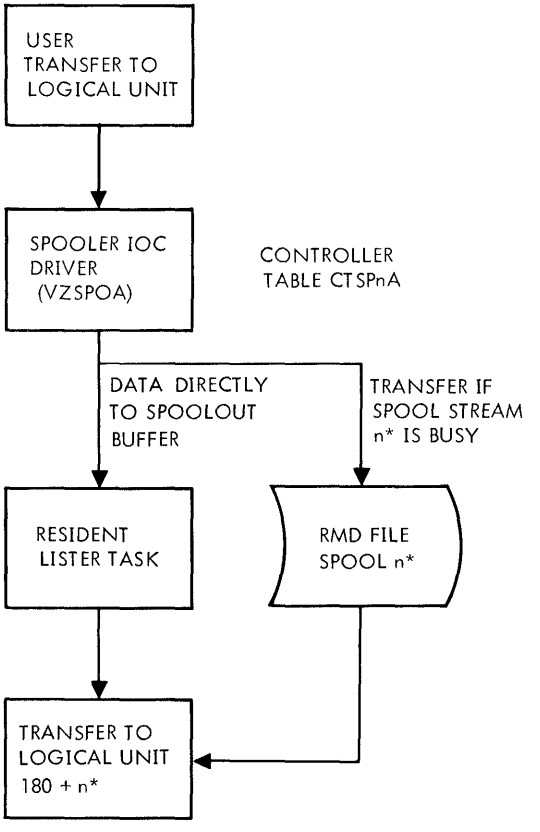
Components

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 then to the appropriate output device.

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

VTII-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, SP0A and 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 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, of 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 queued directly to the output device, up to 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. Withtout 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 I060 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 system's 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 system with a large amount of SPOOL throughout, it is recommended that four buffers be specified for controller tables, otherwise two-buffer tables should be sufficient. For systems without SPOOL, the DEL, V$SPLC and DEL, V$SPRM SGEN directive should be input to delete those resident tasks from the nucleus.

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.

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															
Word 1	V$IOC address															
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:

a. Bit 15 indicates whether the I/O request is complete.

b. Bits 14 through 9 contain one of the error-message status codes described in Appendix B.2.

c. Bit 8 indicates an irrecoverable I/O error.

d. 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.

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

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.

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															
Word 1	V$IOC address															
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 tape and bbb is the address of the end of the tape or end of device.

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.

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 initializating 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

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.

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)
```

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

fcb 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-
                         tape unit)
        .
        .
        .
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.

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
          .
          .
```

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 x
	1	Advance paper one line
	2	Advance paper two lines
Statos 31	7	Advance paper to bottom of form
	8	Normal print size*
	9	Large print size*

*Only for software character generator.

I/O Driver	Function Code	Function
Statos 31/42	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
	16	Large/ 180 degrees
	17	Large/ −90 degrees
	20	Cut paper
	21	End cut

Plot data may be transmitted to the Statos 31 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 left-most dot position.

Statos 31/42 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.

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     ,,CNT
```

3.5.9 STAT (Status) Macro

This macro examines 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-
                        tape unit)
IM      EQU     1       (Immediate return)
        .
        .
REW     REW     ,MT,IM  (DCB can be omitted
                        for REW)
        .
        .
        .
BUSY    STAT    REW,ERR,BOT,EQT,BUSY
        .
        .
        .
BOT
        .
        .
        .
ERR
```

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

rl is the length, in words, of the record to be transmitted

buff is the address of the user's data area

fun 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

rl is the length, in words, of the record to be transmitted

buff is the address of the user's data block

acc specifies the access method and is 0 (default value) for the direct access by logical record, 1 for sequential 1 access by logical record, 2 for direct access using the relative sector number (beginning with 1) within the file, or 3 for sequential access using the relative sector number within the file

key 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).

xxyyzz 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	Set to position of file on directory by mode chosen	Current record set to one or beginning address of logical unit	
4	Set to current position of file as noted on directory	Checks end of file	No action	Checks end of file	No action	Set to ending address of logical unit	
5	Set to beginning of file address put in this word	No action	No action	No action	No action	Set to beginning address of logical unit	Skip first directory sector
6	Set to end of file address	No action	No action	No action	No action	Set to ending address of logical unit	
			Direct-Access Method				
3	Set to position of current record by mode chosen	No action	No action	No action	Set to position of file on directory by mode chosen	Current record set to one or beginning address of logical unit	
4	Set to current position of file as noted on directory	No action	No action	No action	No action	Set to ending address of logical unit	
5	Set to beginning of file address	No action	No action	No action	No action	Set to beginning address of logical unit	Skip first directory sector
6	Set to end of file address	No action	No action	No action	No action	Set to ending address of logical unit	

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 (section 17.2) 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.

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
/MEM	Allocate extra memory for background task

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

/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

c. Language-Processor directives:

/DASMR	Schedule DAS MR assembler
/FORT	Schedule FORTRAN compiler

d. Utility directives:

/CONC	Schedule system-concordance program
/SEDIT	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

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
/DUMP	Dump background at completion of task execution

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)*

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.

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

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 (section 17).

Example: Terminate JCP operations.

```
/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 15-1)

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.

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, 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.

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.

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, ccauses 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, 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 2.1.12). 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 (default value) 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

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
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
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 up to 7 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

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,SS=MT00
/REW,SS
/DASMR
/PFILE,SS,,SS
/CONC
```

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*

where *M*, if present, suppresses the output of a memory 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.

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

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 *p(n)* (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
```

4.3 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.

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.

Details of the DAS MR assembly language are given in the Varian 620/f Computer Handbook (document 98 A 9908 00x), 620-100 Computer Handbook (98 A 9905 o3x), and 73 System Handbook (98 A 9906 01x). These references include descriptions of the directives recognized by the assembler (table 5-1), except for the directive title, which is discussed below.

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
```

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 3.4) macros. Figure 5-1 illustrates these definitions.

```
*
M1          MAC
            EXT         V$IOC
            JSR         V$IOC,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$IOST
        JSR     V$IOST, 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)*

```
*          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$EXEC
          JSR       V$EXEC,1
          EMAC
*
*
*          VORTEX SCHEDULE MACRO DEFINITION
*          SCHED     PL,W,LUN,KEY,'N1','N2','N3'
*                         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
SCHED     MAC
          M2
F         FORM      3,1,6,1,5
          F         0,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 INCREMENT
*                              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)*

```
*
*          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)
F          FORM      8,8
           F         P(4),P(3)
           EMAC
*
*          VORTEX TIME REQUEST MACRO DEFINITION
*          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 SET/FETCH EVENT WORD MACRO DEFINITION
*          TBEVNT VALUE,DISP,C/S
*                         WHERE VALUE = BIT MASK
*                                DISP = DISPLACEMENT OF TIDB WORD
*                                 C/S = 0 FOR CLEAR
*                                     = 1 FOR SET
TBEVNT     MAC
           M2
           DATA  01700
           DATA  P(1),P(2),P(3)
           EMAC
```

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

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   JCPR1     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 symblolic 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

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-statment 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   0DATE     1180   1182   1190
 1937   0NUM       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	

Excepting the STOP and PAUSE statement, compilation and execution with the VORTEX operating system is the same as with the MOS system described in Varian 620 FORTRAN IV Reference Manual (document 98 A 9902 03x). STOP and PAUSE statements output the message

```
taskname STOP (or PAUSE) n
```

With VORTEX, the PAUSE statement generates a SUSPND call to the VORTEX executive.

To resume the suspended task, input operator-communication key-in request ;RESUME (section 17.2.4).

FORTRAN-compiled programs can execute either in foreground or background.

Details of the FORTRAN IV compiler language are given in the Varian FORTRAN IV Reference Manual, except for the TITLE statement, which is discussed in section 5.3.1.

5.3.1 TITLE Statement

This FORTRAN 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

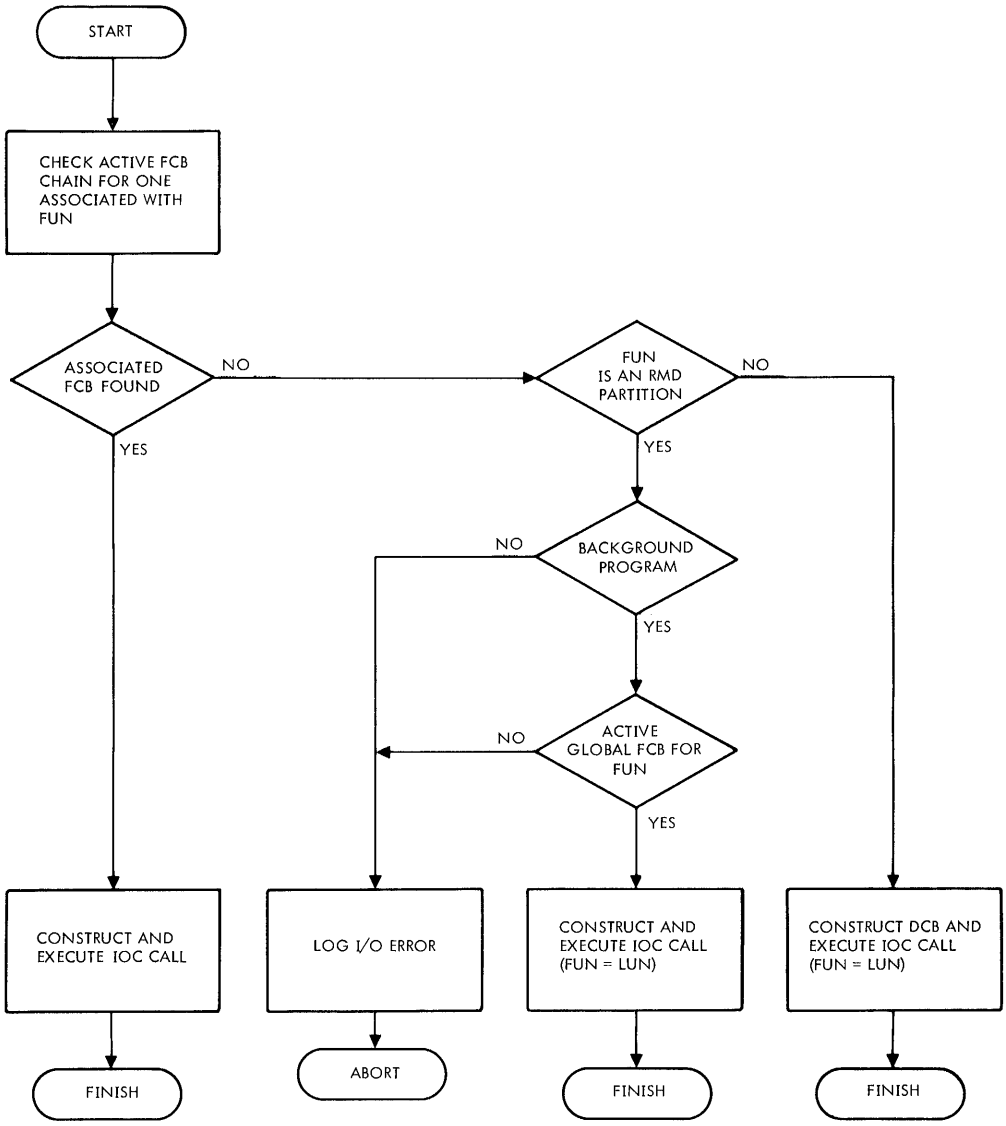
where **name** is the title to be output. The title contains up to eight characters, and is output in the object text as the name by which the program is to be referenced by SMAIN.

If a TITLE statement is used, it must be the first source statement. A TITLE statement forces a page eject on the LO listing.

5.3.2 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



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.

VTII-1445

Figure 5-4. FORTRAN I/O Execution Sequences

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:

```
.
.
.
/ASSIGN,PI=50
/PFILE,4,X,FILE1
/FORT
.
.
.
READ (4,...
.
.
.
```

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, defined in a DATA statement or Hollerith character string

lun is the name or number of the file logical unit, defined in a DATA statement or Hollerith character string

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 FCB 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 is defined in a DATA statement or Hollerith character string

lun is the name or number of the logical unit, which is defined in a DATA statement or Hollerith character string

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 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
$\neq$ 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 replace ment 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 1. 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)

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 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 number.

5.3.3 Encode/Decode Functions

Using the FORTRAN blocking/deblocking feature with the logical unit set = 0 (dummy) provides the capability to transfer data from memory to memory using formatted I/O statements in FORTRAN. This feature is often implemented in other systems by use of ENCODE, and DECODE statements, but in VORTEX it is an intrinsic capability of the blocked I/O feature.

Not all the capabilities of the ENCODE and DECODE statements, which specify a character count so that "new record" format specifiers slash (/) and right parenthesis ( )) position to the end of this count, are available.

5.3.4 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 positive non-zero value indicates an error. A negative non-zero value indicates EOF or EOD.

label(1) is a statement label to which control is transferred, if and 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, 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.

5.3.5 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 the 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 language, and its compilation and execution under VORTEX is described in the Varian 620 RPG IV User's Manual (98 A 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, 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.

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.

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.

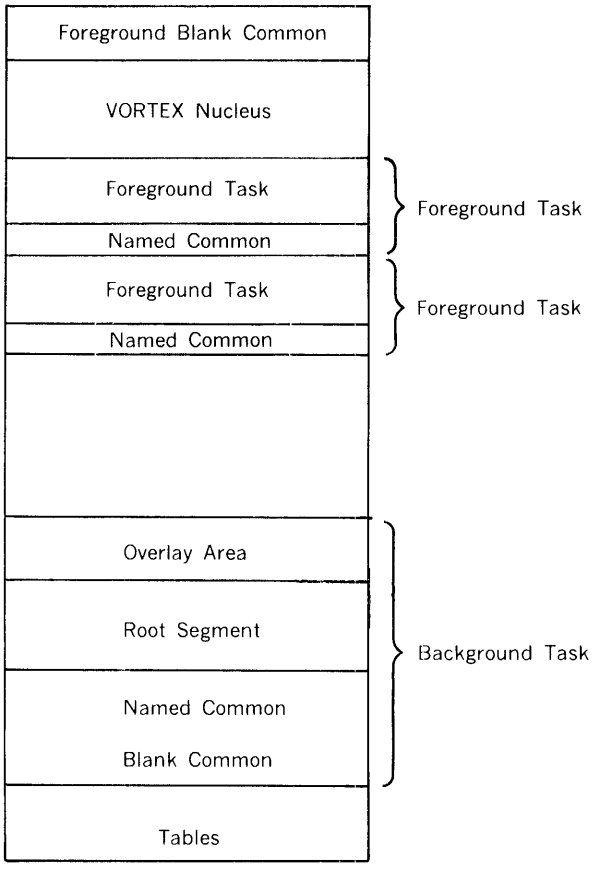
b. Input the corrected directive on the SO unit for processing. The next LMGEN directive is then input from the SI unit.

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.



All foreground tasks share the foreground blank common area but may have their own named common area.

Figure 6-1. Load-Module Overlay Structure

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, section A.6.

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
	item	*b*	*x*	*b*	*location*

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 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).

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.

6.2 LOAD-MODULE GENERATOR DIRECTIVES

- TIDB Create task-identification block
- LD Load relocatable object modules
- OV Overlay
- LIB Library search
- 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*

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

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).

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, LMGEN 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 is encountered.

Successive LD directives permit the loading of object modules that reside on different logical units.

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. (LMGEN 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 LMGEN 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 requires a protection code. The directive has the general form

END,*lun*,*key*

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 catalogued 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
```

6.3 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
```

SECTION 7
DEBUGGING AIDS

The VORTEX system contains two debugging aids: the *debugging program (DEBUG)* and the *snapshot dump program (SNAP)*.

7.1 DEBUGGING PROGRAM

The 816-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 if the root segment of the task is an overlay load module. The load-module generator sets the load-module execution address equal to that of DEBUG.

If the load module has been cataloged, DEBUG executes when the module is scheduled. Otherwise, JCP directive /EXEC (section 4.2.22) is used to schedule the module and DEBUG.

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.

When debugging is complete, the input of any job-control directive (section 6.2) 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 if 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.

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.

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 psuedoregister
B	Display and change the contents of the B pseudoregister
Bx	Change, but do not display, the contents of the B pseudoregister
Cx	Display and change the contents of memory address 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 PI unit until EOF
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
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

Numerical data are always interpreted as octal by DEBUG. Negative numbers are accepted, but they are converted to their two's complements by DEBUG.

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 and the carriage return is represented by the at sign (@). Other entries, in *italics*, are program responses to the directives.

Display the contents of a pseudoregister:

A@
(001200)@

Display and change the contents of a pseudoregister:

B@
(001200) **010406@**

Change, but do not display, the contents of a pseudoregister:

X02050@

Display, but do not change, the status of the overflow indicator:

O@
(000001)@

Display and change the status of the overflow indicator:

O@
(000000) **000001@**

Display, but do not change, the contents of memory address 002050:

C002050@
(102401)@

Display and change the contents of memory address 002050:

C002050@
(102401) **001234@**

Display and change the contents of memory address 002050, then display the contents of the next sequential location:

C002050@
(102401) **001234,@**
(000067)@

Display, but do not change, the contents of memory address 002050, then display the contents of the next location:

C002050@
(102401) ,@
(000067)@

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:

1000200,000210,077777@

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 000001@

Display the contents of memory address 001234:

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 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
```

Using the same data, display, in addition, the contents of memory addresses 001002 through 001025, inclusive and request a dump of the active TIDB.

```
 SN**  SW      000500
 001023  000000  000000  001023  000000
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  065604  055075  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
```

SECTION 8
SOURCE EDITOR

The VORTEX operating system **source editor** (**SEDIT**) is a background task that constructs sequenced or listed output files by selectively copying sequences of records from one or more input files. SEDIT operates on the principle of forward-merging of subfiles and has file-positioning capability. The output file can be sequenced and/or listed.

8.1 ORGANIZATION

SEDIT is scheduled by the job-control processor (JCP, section 4.2.17) upon input of the JCP directive /SEDIT. Once activated, SEDIT inputs and executes directives from the SI logical unit until another JCP directive (first character = /) is input, at which time SEDIT terminates and the JCP is again scheduled.

SEDIT has a buffer area for 100 source records in MOVE operations (section 8.2.8). To increase this, input a /MEM directive (section 4.2.5), immediately preceding the /SEDIT directive, where each 512-word block will increase the capacity of the buffer area by 12 source records.

INPUTS to SEDIT comprise:

a. *Source-editor directives* (section 8.2) input through the SI logical unit.

b. *Old source records* input through the IN logical unit.

c. *New or replacement source records* input through the ALT logical unit.

d. *Error-recovery inputs* entered via the SO logical unit.

Source-editor directives specify both the changes to be made in the source records, and the logical units to be used in making these changes. The directives are input through the SI logical unit and listed as read on the LO logical unit, with the VORTEX standard heading at the top of each page. If the SI logical unit is a Teletype or a CRT device, the message **SE**** is output to it before directive input to indicate that the SI unit is waiting for SEDIT input.

There are two groups of source-editor directives: the copying group and the auxiliary group. **The copying group directives** copy or delete source records input on the IN logical unit, merge them with new or replacement source records input on the ALT unit, and output the results on the OUT unit. Copying-group directives must appear in sequence according to their positioning-record number since there is no reverse positioning. If the remainder of the source records on the IN unit are to be copied after all editing is completed, this must be explicitly stated by an FC directive, (section 8.2.9). Ends of file are output only when specified by FC or WE directives (sections 8.2.9 and 8.2.13). The processing of string-editing directives is different from that of record-editing directives. A string-editing directive affects a specified record, where source records on the IN unit are copied onto the OUT unit until the specified record is found and read into memory from the IN unit. After editing, this record remains in memory and is not yet copied onto the OUT unit. This makes possible multiple field-editing operations on a single source record. **The auxiliary group directives** are those used for special I/O or control functions.

All source records, whether old, new, or replacement records, are arranged in blocks of three 40-word records per VORTEX RMD physical record. Any unused portion of the last physical record of an RMD file on the IN unit should be padded with blanks. When necessary, SEDIT will pad the last RMD record on the OUT unit. When the OUT file will contain more than one source module for input to a language processor, the user should insert two blank records after each END statement to insure that each source module starts on a physical record boundary. Record numbers start with 1 and have a maximum of 9999. Sequence numbers start at any value less than the maximum 9999, and can be increased by any integral increment. These specifications for sequence numbers are given by the SE directive (section 8.2.10).

Error-recovery inputs are entered by the operator on the SO logical unit to recover from errors in SEDIT operations. Error messages applicable to this component are given in Appendix A.8. Recovery is by either of the following:

a. Input the character C on the SO unit, thus directing SEDIT to go to the SI unit for the next directive.

b. Input the corrected directive on the SO unit for processing. The next SEDIT 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 SEDIT task and schedule the JCP for execution. (Note: If there is an I/O control error on the SO unit, SEDIT is terminated automatically.)

OUTPUTS from the SEDIT comprise:

a. *Edited source-record sequences* output on the OUT logical unit.

b. *Error messages.*

c. *The listing of the SEDIT directives* on the LO logical unit.

d. *Comparison outputs* (compare-inputs directive, section 8.2.15).

e. Listing of source records on the LO logical unit when specified by the LIST directive (section 3.2.1).

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.

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 device.

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 now 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.

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

first is the first character position of the sequence name field

last is the last character position of the sequence number field, where the default value of *first,last* is 76,80

initial is the initial number to be used as a sequence number (default value = 10)

increment 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

first is the first character position of the field to be gang-loaded

last is the last character position of the field to be gang-loaded, where the default value of *first,last* is 73,75

string is the string of characters to be gang-loaded into character positions *first* through *last*, 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

first is the first character position of the field to be compared

last is the last character position of the field to be compared, where the default value of *first,last* 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

```
SEDIT 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

```
SEDIT 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
/SEDIT
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
```

The result of running the preceding source editor example would be the following:

Input File

```
 1 *
 2 *        CATALOG ROUTINE
 3 *
 4 A$3      EQU     6
 5 B$3      EQU     9
 6 *
 7 CATLOG   DATA    0
 8          LDA     TMX
 9          LDB     TMY
10          JBZM    ODER
11          ADD     PARM6
12          ANAI    0770
13          STA     TBL+2
14          LRLA    6
15          STA     TBL+4
16          TZB
17          JMP*    CATLOG
```

Output File

```
 1 *
 2 *        CATALOG ROUTINE
 3 *
 4 A$3      EQU     6
 5 *
 6 CATLOG   DATA    0
 7          LDA     TEMP
 8          ADD     PARM6
 9          ANAI    0770
10          STA     TBL+2
11          LRLA    6
12          STA     TBL+4
13          TZB
14          JMP*    CATLOG
15 TBL      BSS     5
```

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.

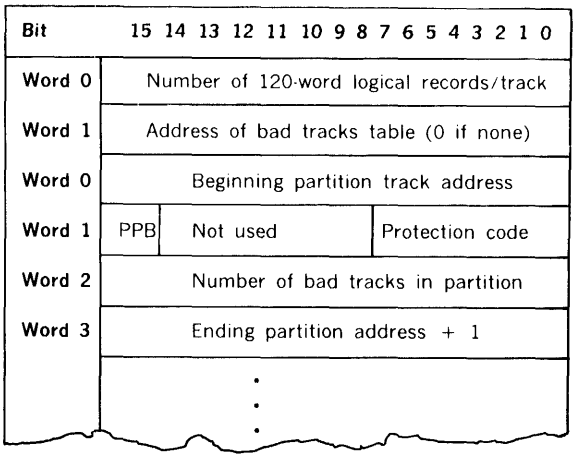
If the SI logical unit is a Teletype or a CRT device, the message **FM**** is output on it before input to indicate that the SI unit is waiting for FMAIN input.

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:



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

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 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, section 3.2) 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 **words**/120 is 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)

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).

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 number 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 (word 2 of the PST entry), and both the CURRENT and END columns contains the last address + 1 (word 3 of the PST entry).

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**

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.

Examples: Specify logical unit 6 as the device from which object modules are to be input.

```
INPUT,6
```

Open and rewind the file ARCTAN on logical unit 104, which has protection code D.

```
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 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
```

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.

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
- PRINTF 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

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.

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 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 visa versa. ASCII read must be specified for this operation.

Example: Copy three files containing 120-word records from the SW logical unit onto logical units LO, 50, and 51 in 40-word records.

```
COPYF,3,SW,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)
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 visa 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 task.

```
/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 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 direct (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.

```
/ASSIGN,PI,25
/IOUTIL
PFILE,PI,,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,recl,name

where

lun	is the name or number of the affected logical unit
key	is the protection code required to address **lun**
recl	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 of 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.

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 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 10.2.8).

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
```

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

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 perform 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)

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

•	INEXIT	Use input preprocessor
•	OUTEXIT	Use output preprocessor

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

11.2.1 SORT Directive

This directive starts the series of directives. The general for 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)**

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)

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

recordlength is a 1- to 4-digit decimal number specifying the length in words of the records in the file.

Example: Describe a sort input file on magnetic tape on logical unit 18, which has 200-word records.

```
INPUT=(18,,,200)
```

11.2.3 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)**

where **lun,** *filename, key* and **recordlength** 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 (120-character) record.

```
OUTPUT=(5,,,60)
```

11.2.4 WORK1,WORK2,WORK3 Directives

These directives describe the intermediate work files for the sort. They have the general form

WORK$\left\{\begin{matrix}1\\2\\3\end{matrix}\right\}$**= (lun,filename,***key***)**

where **lun, filename,** and *key* are the same as described for the INPUT directive (section 11.2.2).

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.

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)
```

11.2.5 SORTKEY Directive

This directive describes one to six 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(6),ec(6),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)
```

11.2.6 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

$$\text{INEXIT} = \left\{ \begin{matrix} \text{YES} \\ \text{NO} \end{matrix} \right\}$$

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.

11.2.7 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

$$\text{OUTEXIT} = \left\{ \begin{matrix} \text{YES} \\ \text{NO} \end{matrix} \right\}$$

The meaning of YES and NO is the same as described for the INEXIT directive (section 11.2.6).

11.2.8 ENDSORT Directive

This directive signals the end of the sort directives. 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

To build a load module using user exits, place the user exit modules in front of the VSORT object module and proceed to generate a single load module.

11.4 VSORT MESSAGES

In addition to listing the VSORT directives, VSORT outputs the following totals:

a. End of sort phase totals

```
SORT PHASE COMPLETE,TOTAL MERGE
RECORDS=XXXXX

INPUT XXXXX ACCEPTED=XXXXX
INSERTED=XXXXX DELETED=XXXXX
```

b. End of merge phase totals

```
SORT COMPLETE,OUTPUT RECORDS
COUNT=XXXXX

MERGE=XXXXX ACCEPTED=XXXXX
INSERTED=XXXXX DELETED=XXXXX
```

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 requitements). DATAPLOT can be used with the following considerations:

a. 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

b. The STATOS unit must be operated under BIC control with PIM assigned interrupts. In addition, the STATOS 31 printer/plotters must be supported by the Single-line Input Buffer Option (Model 31-151).

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:

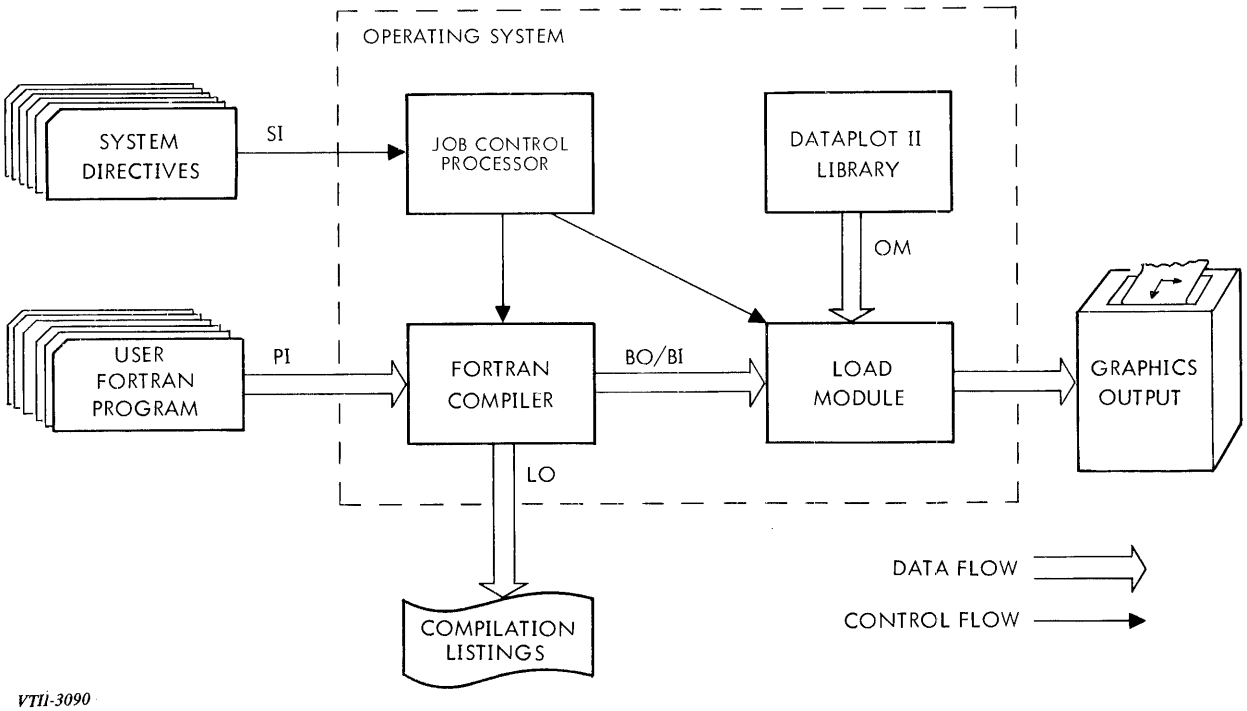
- Defining the Plot File and Initialization
- Building the Plot File
- Sorting the Plot File
- STATOS Paper Control
- Outputting the Plot File in STATOS Raster Format

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.

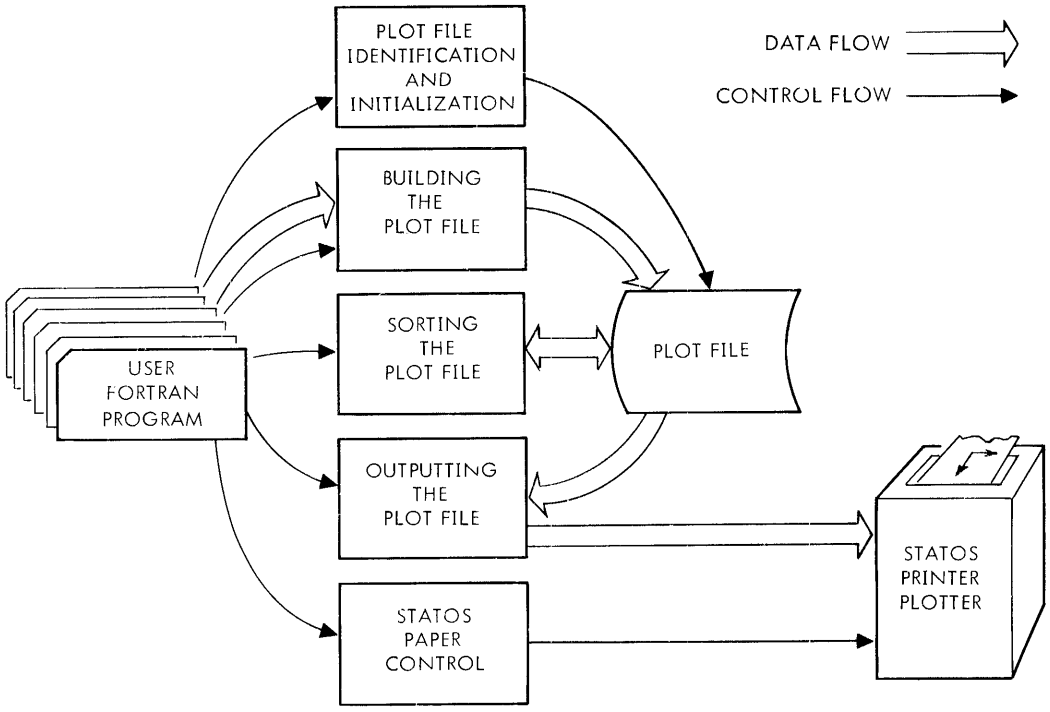
Building the Plot File: If the plot file is to be built through calls to Dataplot subroutines ORIG, CHAR, PLOT, VECT, NUMBER, SCALE, AXIS, DATA, SYMBOL, APPEND, and LINE, the plot file must be assigned to an RMD device or the sort subroutine will not work.

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.



VTII-3090

Figure 12-1. DATAPLOT II Graphics System Data Flow



VTII-3089

Figure 12-2. DATAPLOT II Organization

Outputting the Plot File: Subroutine DPPLOT outputs STATOS raster format data. DPPLOT 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 (MPBOM). 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.

DATAPLOT II subroutines are listed below:

Dataplot II initialization

DPINIT	BDPOM
PLOTS	BDPOM

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
DPPLOT	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.

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. Stylii/Line	Raster Buffer Size
70-6608	11 inches	1056	66
70-6602	14-7/8 inches	1408	88
70-6611 and 70-6621	8-1/2 inches	800	50
70-6613 and 70-6623	11 inches	1056	66
70-6615 and 70-6625	14-7/8 inches	1408	88
70-6617 and 70-6629	22 inches	2048	132

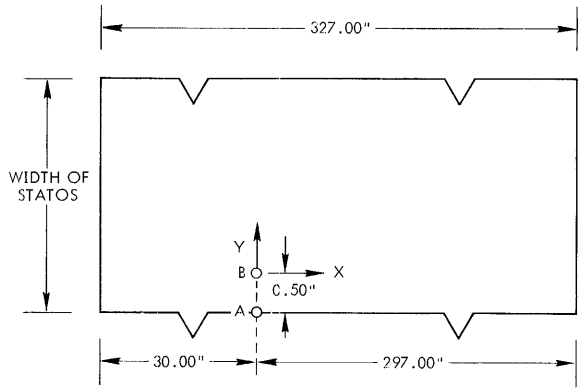
The buffer is also used to hold vectors and characters at the time they are being converted to STATOS raster format. A six-word entry will be placed in this buffer when the vector or character is first to appear on a STATOS scan line. The entry remains until the vector or character reaches its last STATOS 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 160. 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-3088

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 subroutine call 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
lun	is the number or variable of the plot file logical unit (Integer).	8
key	is the protection key, if any.	None
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.

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

CALL PLOT (x,y,±idraw)*

***BDPOM**

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).

± 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:

a. The scale factors is 1., 2., 4., 5., or 8. times 10E(n).

b. The displacement is an integral multiple of the scale factor.

c. The displacement is .LE. the minimum value in the array.

d. 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

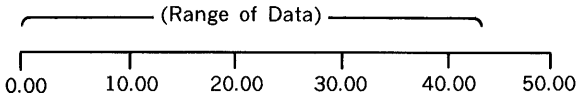
Examples:

1. 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:

   Units/inch = (42.00-1.00)/5.0 = 8.2
   SF (scale factor) = 10.0
   VLO (first value plotted) = 0.0

   VLO value is stored in ARR(25)
   SF value is stored in ARR(26)

Using these values, AXIS would draw the following axis line:

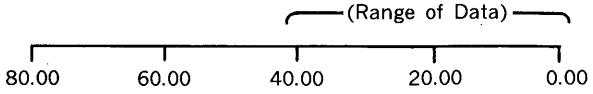


2. Assume that the array of Example 1 is to be plotted on a 4-inch axis, from maximum to minimum. CALL SCALE (ARR,4.0,24,-1) would give these results:

   SF = (1.00-42.00)/4.0 = -10.25, which is adjusted to -20.

   Minimum multiple = 0.00; VLO = Minimum + (AXLEN * SF) = 80.00

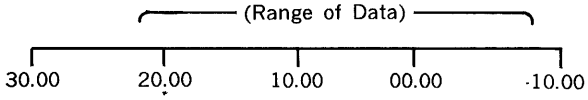
In this case the following axis would be drawn:



3. 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 tic 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,±nch,vlo,sf)*

or

CALL AXIS (x,y,bcd,±nchar,axlh,angle,vlo,sf)**

* MDPOM

** BDPOM

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.

±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≥0: the title, tic marks and interval labels will be plotted on the counter-clockwise side of the axis.

±nch NCH is the number of letters contained in the character string to be plotted as a title (Integer).

If NCH≥0, the title, tic marks, and interval labels will be plotted on the clockwise side of the axis.

If NCH≥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: "x10³."

The SCALE routine should be used prior to using AXIS if SF = ARR(NPTS + 2).

Error Conditions: **None**

Example:

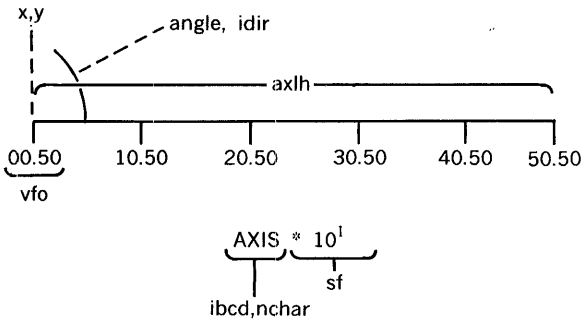
```
CALL AXIS (0.0,0.0,5.0,0,4HAXIS,
           4,5.0,100.0)*

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,±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.

isaor 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≤ 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

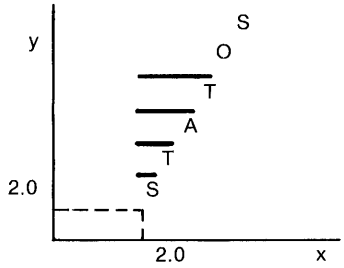
Example:

```
      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).

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

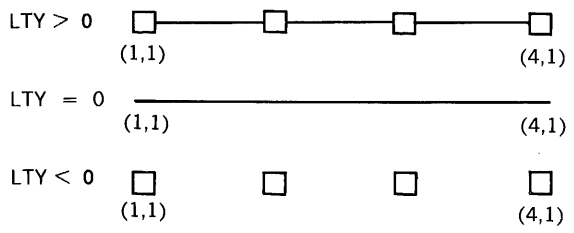
Error Conditions:

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.

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:	I004,xxxxxx
Condition:	File name not found
Action:	Append call is ignored
Message:	I010,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 ENCUT*

*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 DPPLOT 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 DPPLOT (Output File)

DPPLOT subroutine converts the plot file to STATOS raster format and outputs the raster data to the output device specified in the call to PLOTS. DPPLOT 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 DPPLOT*

*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 DPPLOT } (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:

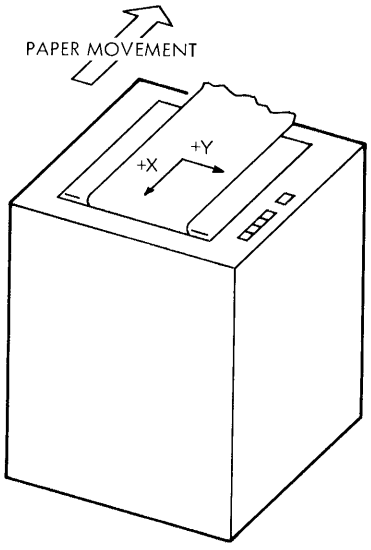
```
170    CALL DPCLOS
```

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 (7.1,3.1)
```

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.

12.5 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.

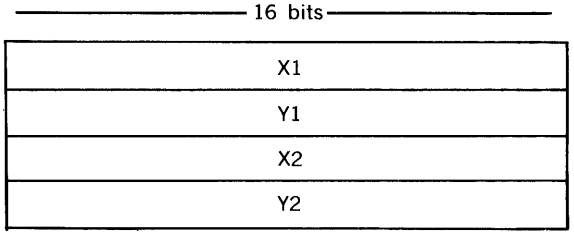


Figure 12-5. Vector-Data Format

where

$X2 \leq X1 \leq 32,700$

Y1 and Y2 number of STATOS stylii

12.5.2 Characters

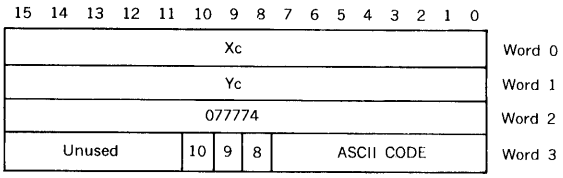
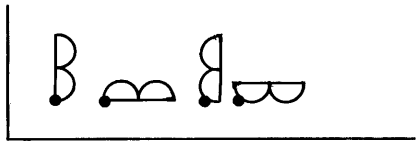


Figure 12-6. Character Data Format

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.

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
077777
077777
077777
077777

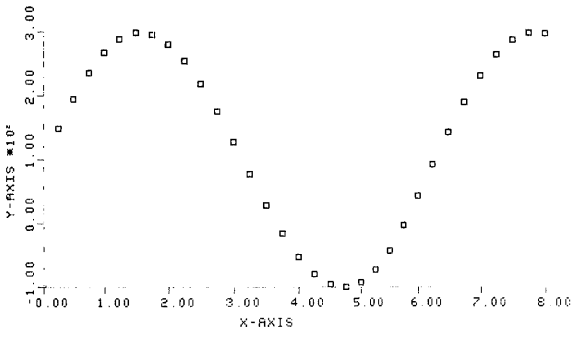
Figure 12-8. End-of-Plot Indicator

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(54)
            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
```



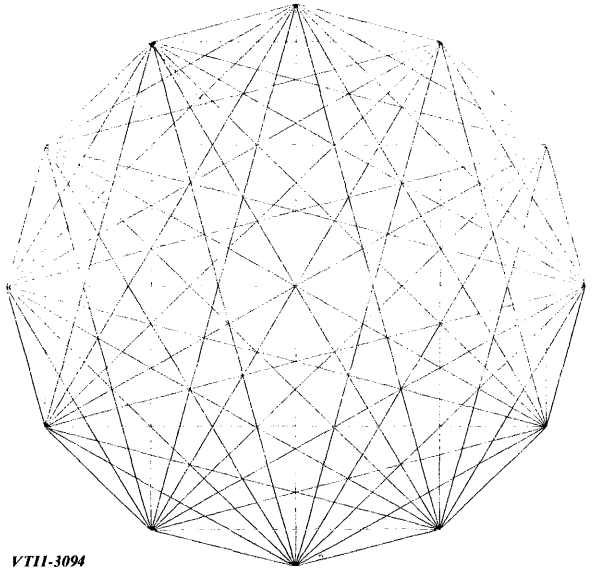
VTII-3095

Figure 12-9. Sine Wave Plot Generated by DATAPLOT II

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)
```



VTII-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 VDM 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 DPINIT (       ) if necessary
CALL PLOTS  (       )
CALL DPSORT
CALL DPPLOT
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 PLOTS  (       )
CALL DPPLOT
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.

Example:

```
/FORT,B,L,M
C     BUILD THE PLOT FILE
      DIMENSION IBUFF (130)
      CALL DPINIT (25,2H6K,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 DPPLOT
      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 DPPLOT
      CALL EXIT
      END
```

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 = 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)
```

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(1).

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 $10^{\pm 38}$ can be stored as a single-precision floating-point number having a precision of more than six decimal digits.

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 $10^{\pm 38}$ 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   0  --------Exponent--------
n+1)  s    ----------------High Mantissa-----------------
n+2)  0    ----------------Mid Mantissa------------------
n+3)  0    ----------------Low Mantissa------------------
```

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	A double-precision number
f	Two-word, fixed-point number
i	An integer
r	A real number
S	A six-character ASCII string
X	Hardware X register
z	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 11-2 refer to items in tables 11-1 and 11-2. 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 i1, in A compute i1**i2.	CALL $HE,i2	$SE(FSE), $HM
$PE	Given: AB contains r, in AB, compute r**i.	CALL $PE,i	$SE(FSE), $QM, $QN
$QE	Given: AB contains r1, in AB, compute r1**r2.	CALL $QE,r2	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, $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 r1, in AB, compute r1*r2 with $QM, or r1/r2 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,r2 CALL $QN,r2	XDMU, $FMS, XDDI, $SE(FSE), $EE, $NML

Table 13-1. DAS Coded Subroutines *(continued)*

Name	Function	Calling Sequence	External References
FADDSUB	Given: AB contains r1, in AB, compute r1 + r2 with $QK, or r1 − r2 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 f1/f2	CALL XDDI,f2	XDSU, XDCO
XDMULT	In AB, compute f1*f2	CALL XDMU,f2	XDAD, XDCO
XDADD	In AB, compute f1 + f2	CALL XDAD,f2	None
XDSUB	In AB, compute f1 − f2	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 i1, in A, equal to that of i2	CALL ISIGN,i2	$SE(FSE)
SIGN	Set the sign of r1, in AB, equal to that of r2	CALL SIGN,r2	$SE(FSE)
$HN	Given: A holds i1, in A, compute i1/i2	CALL $HN,i2	$SE(FSE), $EE
$HM	Given: A holds i1, in A compute i1*i2	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

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*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
A2MT	Translate in memory a character string of length n starting at s and ending at e from eight-bit ASCII to six-bit magnetic tape BCD code s is the start of the ASCII block and e is the start of the BCD block.	CALL A2MT,n,s,e	None
MT2A	Translate in memory a character string of length n starting at s and ending at e from six-bit magnetic tape BCD code to eight-bit ASCII s is the start of the BCD block and e is the start of the ASCII block.	CALL MT2A,n,s,e	None
EXIT	Formats and executes an RTE EXIT macro	CALL EXIT	V$EXEC
SUSPND	Formats and executes an RTE SUSPND macro with parameter i.	CALL SUSPND(i)	V$EXEC
RESUME	Formats and executes an RTE RESUME macro to resume task s.	CALL RESUME(s)	V$EXEC, $RTENM
ABORT	Formats and executes an RTE ABORT macro to abort task s.	CALL ABORT(s)	V$EXEC, $RTENM
ALOC	Formats and executes an RTE ALOC macro to call reentrant subroutine s.	CALL ALOC(s)	V$EXEC
PMSK	Formats and executes an RTE PMSK macro to operate on PIM i1 with line mask i2 and enable/disable flag i3.	CALL PMSK(i1, i2,i3)	V$EXEC
DELAY	Formats and executes an RTE DELAY macro with the 5-millisecond count in i1, the minute count in i2, and delay mode in i3.	CALL DELAY(i1, i2,i3)	V$EXEC
TIME	Formats and executes an RTE TIME macro with the minute count in i1 and delay mode in i2.	CALL TIME(i1,i2)	V$EXEC
OVLAY	Formats and executes an RTE OVLAY macro with i1 = 0 to execute, i2 = 0 to load, and s is the overlay name.	CALL OVLAY(i1, i2,s)	V$EXEC, $RTENM
SCHED	Formats and executes an RTE SCHED macro with i1 = priority, i2 = wait flag, i3 = logical-unit number, s1 = key and s2 = task name.	CALL SCHED(i1, i2, i3,s1,s2)	V$EXEC, $RTENM

Table 13-1. DAS Coded Subroutines *(continued)*

$RTENM	Moves the six-character name from X to B	CALL $RTENM	None
$EE	Outputs error messages on the SO device.	CALL $EE	VIOC, VIOST, V$EXEC
$SE	Fetches n parameters from a subroutine call	CALL $SE, n BSS n	None

Table 13-2. FORTRAN IV Coded Subroutines

Name	Function	Calling Sequence	External References
$9E	Compute ACCZ**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, CMPLX
CMPLX	Load ACCZ with a value having a real part r1 and an imaginary part r2	CALL CMPLX(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 13-2. 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**i where d is in AC	CALL $XE(i)	$SE(FSE), $ZF, MOD, $ZM, $HN, $ZN, $ZS
$YE	Compute d**r where d is in AC	CALL $YE(r)	$SE(FSE), $ZS, DBLE, $ZE, $ZF
$ZE	Compute d1**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, IFA, $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, IFA, $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 13-2. 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 13-2. **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

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-clearing 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. The rules for the use of directly connected routines are:

a. All volatile registers used by the routine are restored before returning to the interrupted task.

b. Interrupts remain disabled during processing.

c. IOC and RTE calls are not allowed.

d. Execution time is minimal.

e. PIM interrupts are enabled before returning to the interrupted task through word 0 of the line handler. The real time clock (RT clock) is enabled only if the task is not the VORTEX RT clock processor (location 0300, V$CTL, contains 037 if the VORTEX RT clock processor is interrupted).

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 tasks 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

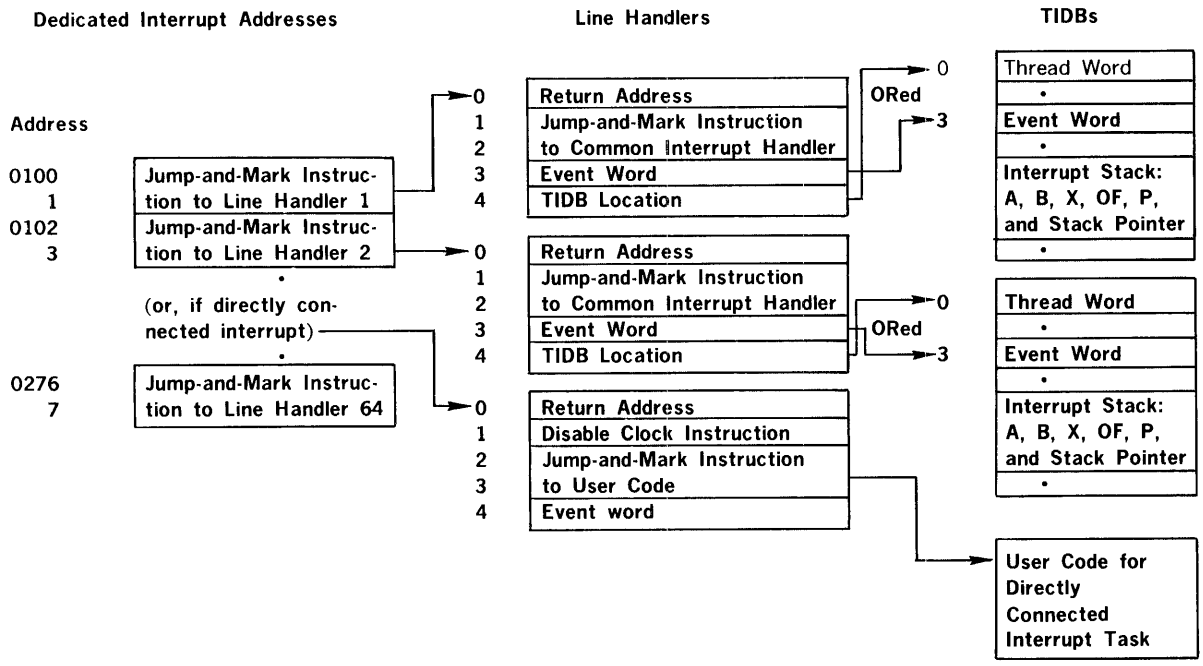


Figure 14-1. Interrupt Line Handlers

event word and the interrupt-expected status bit to lock out further interrupts until processing is complete. The event word distinguishes different interrupt lines that could activate the same task.

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.

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 supspend (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:

a. 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.

b. 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 processor interrupts the interrupt handler, the common interrupt handler posts the interrupt and 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: When the background area is active, it is run as an unprotected area of memory with the rest of the system protected. In such a situation, memory protection interrupts are generated when the background task attempts to execute a "privileged" instruction such as external control or halt, or attempts to jump into, write into, or perform I/O on protected memory. The memory protection routine processes all protection violation interrupts and is the highest-priority interrupt in the system.

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 procesor, 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.

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, map registers are reloaded, 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 adddress 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 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, one 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 **r(n)** and **s(n)** both 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.

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 allocates memory in 512-word blocks starting with location 512 for background, or the first 512-word block below the resident task directory for foreground tasks. A foreground task is allocated into the first such available area. 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.

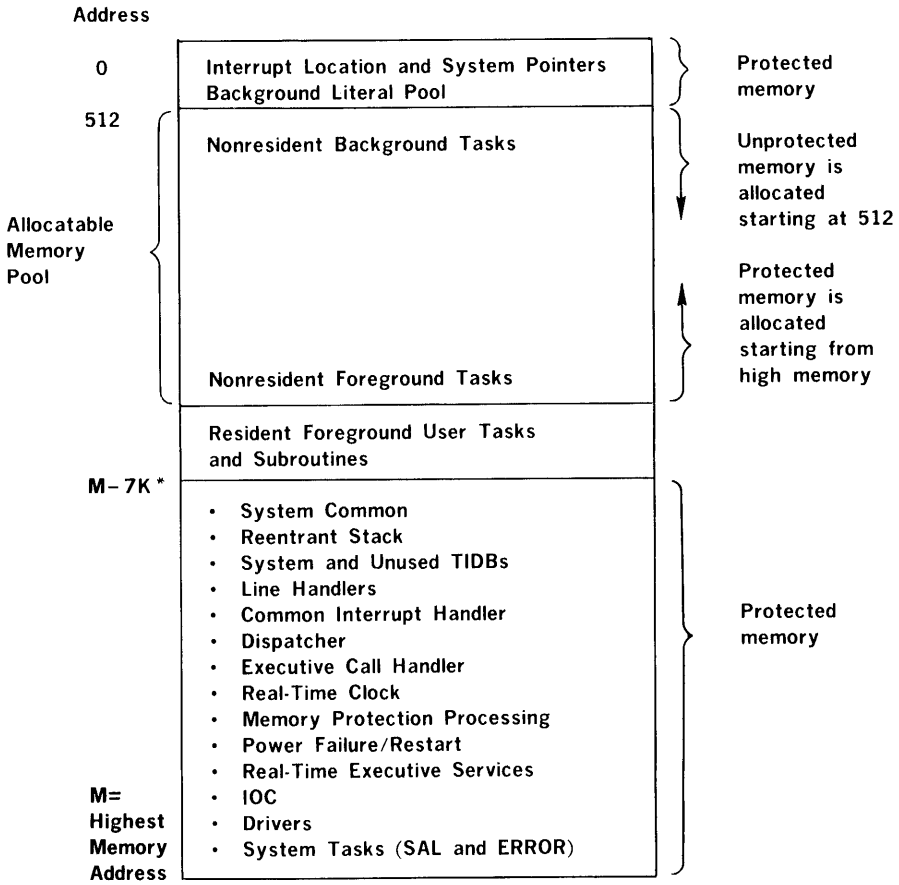
The background memory allocation depends on the size of the background task being loaded. Only the amount needed is so allocated automatically, although the JCP /MEM directive can allocate extra memory for a background task. Figure 14-2 is a VORTEX memory map, figure 14-3 shows the priority structure, table 14-1 is a description of a TIDB, and table 14-2 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 usuage. 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 7K.

Figure 14-2. VORTEX Memory Map

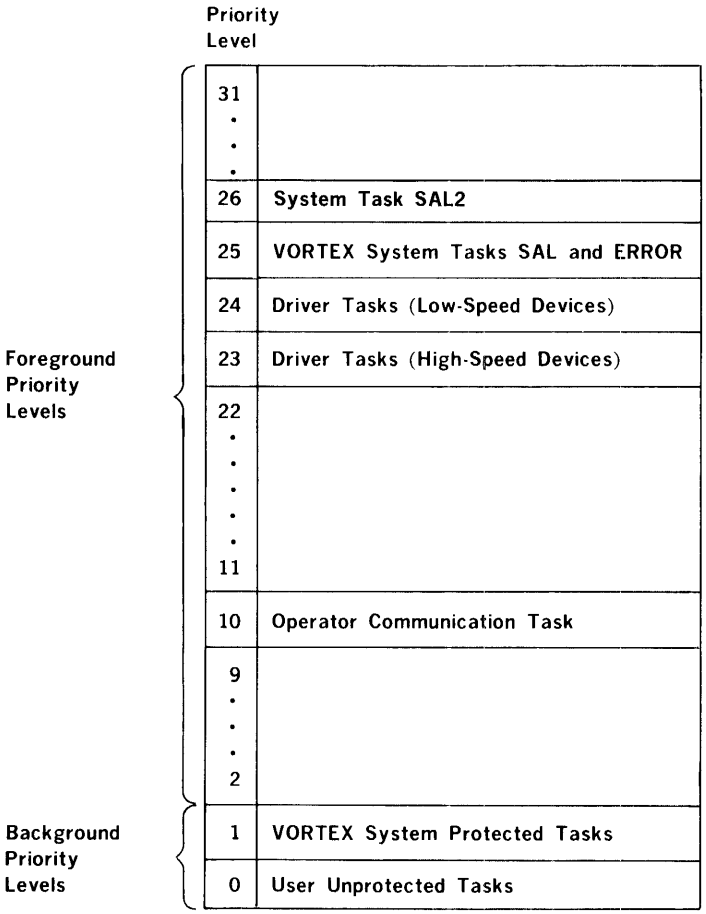


Figure 14-3. VORTEX Priority Structure

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)
TBRSB	5	B Register (Reentrant and Suspension Stack)
TBRSX	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
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

Figure 14-4. TIDB Description

Table 14-1. TIDB Description

KEY:

Symbol	Word	Bits	Set =	Description
TBTRD	0	15-0	Task thread	Points to next TIDB in chain. Two queues are maintained in the system: active and inactive. V$TB points to the highest-priority active task. V$UTB points to next available inactive TIDB space. 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	Unused	
		13	Load overlay	RTE overlay request made by task with overlay name in user request.
		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.
		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; or scheduler, a TIDB
		10	Background being checkpointed	Background task being written on checkpoint file.
		9	TIDB not available	Schedule request made but no TIDBs available for allocation.
		8	Unused	
		7	Unused	

Table 14-1. TIDB Description *(continued)*

Symbol	Word	Bits	Set =	Description
TBPL *(continued)*		6	Unused	
		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. Clears 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).
TBRSB	5	15-0	B register (reentrant and suspension stack)	
TBRSX	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.
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.)

Table 14-1. TIDB Description *(continued)*

Symbol	Word	Bits	Set =	Description
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	15-10	Block allocation size	Number of 512-word block for execution of task.
		9-5	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.)
		4-0	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.
TBKN2	19	15-0	Task name	Second two characters of six-character task name.

Table 14-1. TIDB Description *(continued)*

Symbol	Word	Bits	Set =	Description
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.
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 ABORT (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.

Table 14-2. Map of Lowest Memory Sector

Address	Symbolic Name	Description
00-01		CPU interrupt code (preset to NOP)
02-016		Unassigned: available to the user
017		TIDB address for WCS reload task
020,021		Memory protection interrupt: halt (jump-and-mark to V$MPER)
022,023		Memory protection interrupt: I/O (jump-and-mark to V$MPER)
024,025		Memory protection interrupt: write (jump-and-mark to V$MPER)
026,027		Memory protection interrupt: jump (jump-and-mark to V$MPJP)
030,031		Memory protection interrupt: overflow (jump-and-mark to V$MPER)
032,033		Memory protection interrupt: I/O overflow (jump-and-mark to V$MPER)

Table 14-2. Map of Lowest Memory Sector *(continued)*

Address	Symbolic Name	Description
034,035		Memory protection interrupt: write overflow (jump-and-mark to V$MPER)
036,037		Memory protection interrupt: jump overflow (jump-and-mark to V$MPER)
040,041		Power-down interrupt (jump-and-mark to V$PFDN)
042,043		Power-up interrupt (jump-and-mark to V$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 9 Current keypunch specified by JCP /KPMODE directive (/JOB, /FINI, or /ENDJOB resets 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 9-5 Unused. 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
056-067	V$BIC1	BIC in sequence (maximum 8)
070-073	V$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

Table 14-2. Map of Lowest Memory Sector *(continued)*

Address	Symbolic Name	Description
076-077		FPP (Floating-Point Processor) interrupt (jump and mark to V$FPP)
0100-0117		PIM 0 jump-and-mark to individual line handlers
0120-0137		PIM 1* jump-and-mark to individual line handlers
0140-0157		PIM 2* jump-and-mark to individual line handlers
0160-0177		PIM 3* jump-and-mark to individual line handlers
0200-0217		PIM 4* jump-and-mark to individual line handlers
0220-0237		PIM 5* jump-and-mark to individual line handlers
0240-0257		PIM 6* jump-and-mark to individual line handlers
0260-0277		PIM 7* jump-and-mark to individual line handlers
0300	V$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 unused TIDB stack (zero if no TIDB are available to be allocated)
0305	V$PTVB	Address of next entry in reentrant stack
0306	V$FLRS	Address of first location of re-entrant stack
0307	V$LRSK	Address of last location of re-entrant stack + 1
0310	V$CKPT	Checkpoint flag (set if background checkpointed)
0311	V$OPCL	Address of TIDB for OPCOM task

* If PIM is not present, the space is available to the user.

Table 14-2. Map of Lowest Memory Sector *(continued)*

Address	Symbolic Name	Description
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$LUP	Address of first unprotected word (memory address 01000)
0317	V$LLUP	Address of last unprotected word (depends upon size of background executing task)
0320	V$IM	Interrupt mask for PIM 0 (0 = enable, 1 = disable)
0321		Interrupt mask for PIM 1
0322		Interrupt mask for PIM 2
0323		Interrupt mask for PIM 3
0324		Interrupt mask for PIM 4
0325		Interrupt mask for PIM 5
0326		Interrupt mask for PIM 6
0327		Interrupt mask for PIM 7
0330-0333	V$MPM	Memory protection mask (4 words), 0 = unprotected, 1 = protected (words initially set to 0177777)
0334-0337	V$CAM	Core allocation mask (4 words), 0 = 512-word block available for allocation, 1 = 512-word block in use and not available for allocation (SGEN generates initial mask)
0340		Reserved for future VORTEX use
0341	V$CRDR	Address of resident directory
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)

Table 14-2. Map of Lowest Memory Sector *(continued)*

Address	Symbolic Name	Description
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)
0345	V$LUNT	Address of logical-unit name table
0346	V$OPCF	OPCOM lockout flag
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$CKB])
0353	V$CKB	Basic clock interrupt rate in milliseconds
0354	V$CRM	Clock resolution increments for fractions of a minute in 5-millisecond increments
0355	V$DSTB	Address of DST block
0356	V$LIT	Last address in background literal pool
0357		Reserved for future VORTEX use
0360	V$CTAD	Base address for controller address table
0361	V$SCTL	Current controller in scan
0362	V$NCTR	Number of controllers
0363-0372	V$PIMN	External device address table for PIMs
0373-0374		Reserved for future VORTEX use
0375	V$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	V$LUT1	Starting address of logical-unit table for JCP/OPCOM-assignable logical units

Table 14-2. Map of Lowest Memory Sector *(continued)*

Address	Symbolic Name	Description
0401	V$LUT2	Starting address of logical-unit table for unreassignable logical units
0402	V$LUT3	Starting address of logical-unit table for OPCOM-assignable logical units
0403	V$1MIN	Clock constant set up by SGEN where V$1MIN = 32767 - (60000/(5*V$CTMS)) + 1
0404-0407		Reserved for future VORTEX use
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$JCB	Address of 41-word JCP buffer (all system background programs and JCP input directives into this system buffer)
0413	V$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
0415	V$BFC	Top of foreground area, bottom of foreground blank common
0416	V$TFC	Top of foreground blank common, top of VORTEX nucleus core
0417	V$PST	Maximum RMD partitions 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

Table 14-2. Map of Lowest Memory Sector *(continued)*

Address	Symbolic Name	Description
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
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)

Table 14-2. Map of Lowest Memory Sector *(continued)*

Address	Symbolic Name	Description
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 **nth** task is

$$t + 14u + 17v + 12w + 18x + 25y + z \text{ cycles}$$

where

- t — is 9 or 11, 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:

  48 to activate the ERROR task
  56 to activate the SAL task on aborting or exiting
  56 to activate a loaded task that is not suspended, or to activate the SAL task to load the requested task
  61 to activate an interrupted, suspended task
  65 to activate a task when the event word is set and the interrupt suspended

Example of dispatcher interrupt timing: Calculate the time needed to begin execution of a task under the following conditions:

a. ten tasks have task-suspended bits set
b. two tasks have event-expected bits set but event words reset
c. two tasks have error bits set but error task busy
d. one task has exited (TBST bit 12) but its I/O is not complete
e. no tasks are aborted with incomplete I/O
f. four tasks are active but not loaded
g. the task to be started was interrupted and suspended

Also, assume the entry to the dispatcher sets the value t to 9.

These conditions set the following parameters of the timing formula:

$u = 10$ $\quad x = 1$

$v = 2$ $\quad y = 4$

$w = 2$ $\quad z = 61$

Substitution of these in the formula gives the following:

$$9 + 14(10) + 17(2) + 12(2) + 18(1) + 25(4) + 61 =$$

$$9 + 140 + 34 + 24 + 18 + 100 + 61 =$$

386 cycles

Search, allocate, and load: Foreground task load processing requires

$$550k + x + y + ny + my + 270kz + wk$$

where

- k — is the cycle time
- x — is the time required for an RMD OPEN request
- y — is the time required to read one RMD record (pseudo TIDB)
- ny — is the time required to read the task into memory (n is a variable task size)
- my — is the time required to read allocation data (m is the variable number of relocation records)
- z — is the number of 16-bit relocation words
- wk — is the memory allocation time (in the best case, the value of w is 35 times the numbers of blocks to be allocated + 142)

Foreground task load processing

Example: Calculate the time required to load a foreground task from disc which has an average read time of 80 millisecond per record. Assume the cycle time is 330 nanoseconds. The average time needed for an RMD OPEN will be one access if the number of files in the partition is 19 or fewer and assuming that only one access is needed an estimate of the time would be 80. For this example, the task is 1200 words in 10 records and its number of relocation words is 480 or 4 records. In the best case of two blocks, w would be calculated as 212.

$550 \times 330 \ (\times 10^{-6}) + 80 + 800 + 320 +$

$270 \cdot 330 \cdot 400 \times 10^{-6} + 212 \times 330 \times 10^{-6}$ or 1280 milliseconds

Background task load processing requires

$$575k + x + y + ny + wk$$

where

k, x, y, and ny are defined above

wk is the background memory allocation (the best case, the value of w is 20 times the number of blocks to be allocated + 240)

Resident task load processing requires

$$(61 + 16x)\ k$$

where

k is the cycle time

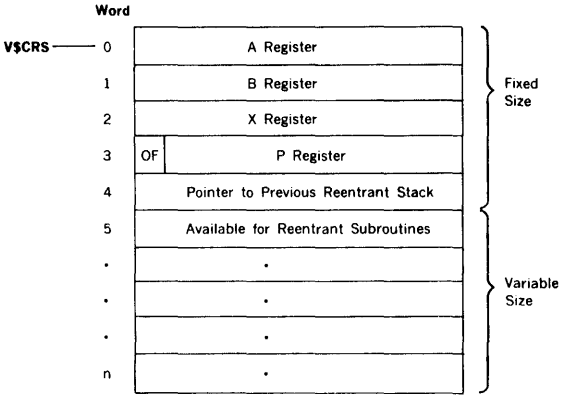
x is the number of entries searched in the resident directory before the required task name is found.

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 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

```
        NAME    RSUB
V$CRS   EQU     0302
RSUB    DATA    6          Allocate six-word
        LDX     V$CRS      stack (one temporary
                           location)
        .
        .
        STA     6,1        Save A in temporary
                           storage
        .
        .
        LDA     6,1        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 V$IOC,X instruction with V$IOC containing the IOC entry address. 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.

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

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) FCEFE (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. CTFCB contains the address of DCB.
   (1) DCRECL (word 0) = record length
   (2) DCBUFF (word 1) = user buffer
   (3) DCCNT (word 2) = function count

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: 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 RSTPE (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.

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 (or CTD0B). An EXT statement must specify the TIDB name as an external name.

```
EXT   TBD0B
DATA  TBD0B
```

1 **CTADNC** = End of table + 1
DATA CTEND
where CTEND is the end of the controller table + 1.

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.

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 #TTY0A

Word Item Description

6 **CTDVAD** = Controller device address. #A followed by the name of the controller
Example: DATA #ATY0A

7 **CTIOA =** I/O algorithm. The ratio of device transfer rate to DMA transfer rate + 10 percent. 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 the controller,
Example: DATA !BD0B
When the driver is entered the item contains 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 retires 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 sections 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. DATA !PD00B

16 **CTPST1** = RMD only. Base address of the PST for RMD unit 1.
Example: DATA !PD01B

17 **CTPST2** = RMD only. Base address of PST for RMD unit 2.
Example: DATA !PD02B

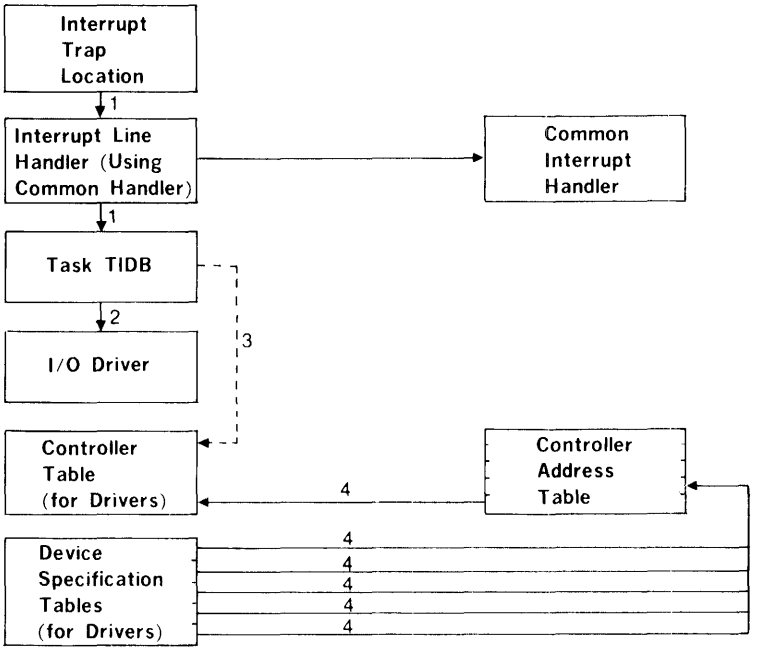
18 **CTPST3** = RMD only. Base address of PST for RMD unit 3.
Example: DATA !PD03B

14.4.4 Enabling and Disabling PIM Interrupts

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

The use of a directly connected interrupt handler (see section 14.4.1) in lieu of the VORTEX common interrupt handler is specified in the PIM directive during system generation. The directly connected interrupt handler is entered with a jump-and-mark instruction (see figure 14.1). The first word in the interrupt handler must be a mark location. When entered, both the real-time (RT) clock and PIMs are disabled. Before exiting, the interrupt handler must enable the PIMs (EXC 0244). The RT clock must also be enabled (EXC 0147) in all cases except when the RT clock processor has been interrupted. Location 0300, V$CTL, will contain 037 if the RT clock processor had been interrupted. The directly connected interrupt handler must provide a check for interruption out of the RT clock processor and enable or disable the RT clock accordingly. The interrupted task's return address is found in word 0 of the line handler. The address of word 0 is obtained by subtracting four from the contents of the directly connected interrupt handler mark location.

14.4.6 VORTEX use of BICs

VORTEX supports a maximum of ten BICs. 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-transfer-complete interrupts must be assigned by ascending BIC numbers (020, 022, 024, 026, 070, 072, etc) starting with the first PIM and the first interrupt line; 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.

I/O drivers utilizing BICs must call a common BIC routine V$BIC as described in section 14.4.2.

SECTION 15
SYSTEM GENERATION

The VORTEX **system-generation component** (**SGEN**) tailors the VORTEX operating system to specific user requirements. SGL 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 four-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 and the resident-task configurator (section 15.7)

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)

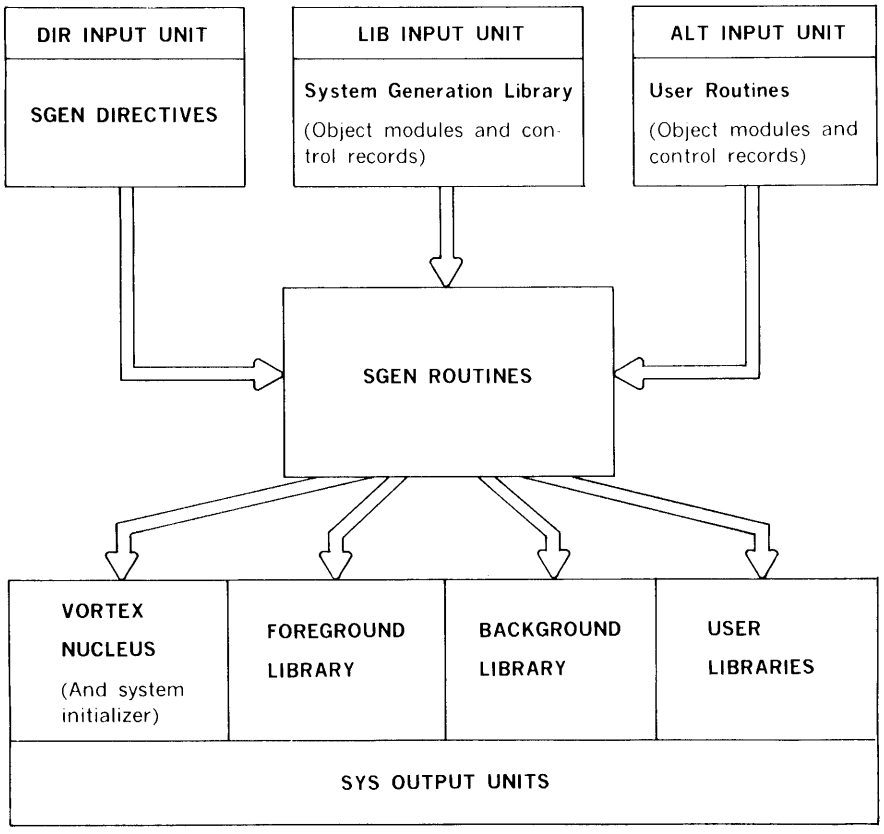


Figure 15-1. SGEN Data Flow

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 (,) or by equal signs (=). The directives are free-form and blanks are permitted between individual character strings, i.e., before and after commas (or equal signs). 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.

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.

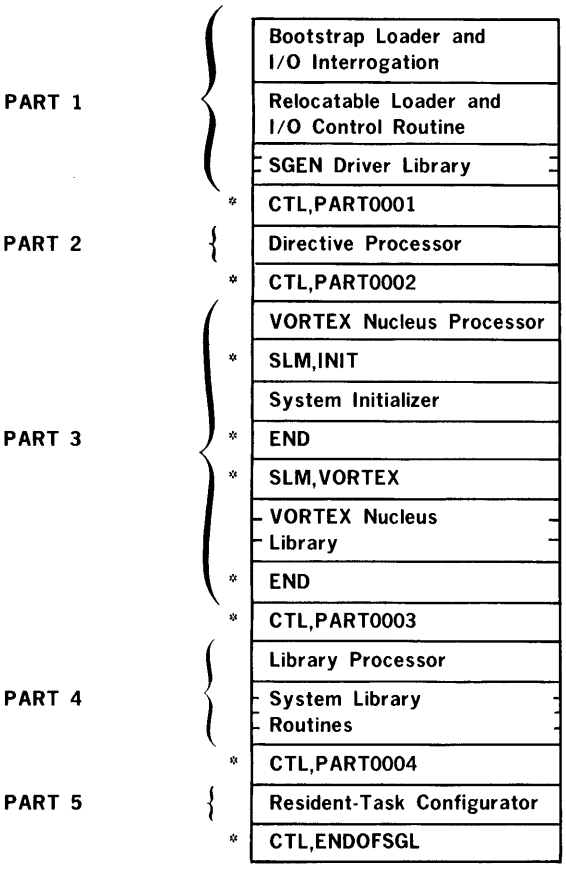
The SGL is divided into five functional parts, each separated by CTL control records (figure 15-2).

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.

Control record CTL,PART0002 terminates part 2 of the SGL.



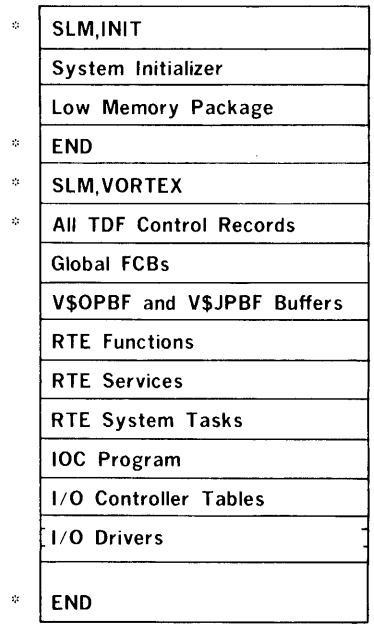
NOTE:

* = Alphanumeric control record

Figure 15-2. System-Generation Library

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



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 (figure 15-4) 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)*.

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.

REQUIRED (FOREGROUND) SYSTEM TASKS

*	SLM,FGTSK1
*	TID,V$OPCM,2,8,106
	V$OPCM Program
*	ESB
*	END
*	SLM,FGTSK2
*	TID,JCDUMP,2,0,106
	JCDUMP Program
*	ESB
*	END
*	SLM,FGTSK3
*	TID,RAZI,2,0,106
	RAZI Program
*	ESB
*	END

*	SLM,BGTSK5
*	TID,FORT,1,0,105
	FORTRAN Compiler
*	ESB
*	END
*	SLM,BGTSK6
*	TID,CONC,1,0,105
	Concordance Program
*	ESB
*	END
*	SLM,BGTSK7
*	TID,IOUTIL,1,0,105
	I/O Utility Program
*	ESB
*	END

REQUIRED (BACKGROUND) SYSTEM TASKS

*	SLM,BGTSK1
*	TID,JCP,1,0,105
	Job-Control Processor
*	ESB
*	END
*	SLM,BGTSK2
*	TID,LMGEN,1,0,105
	Load-Module Generator
*	ESB
*	END
*	SLM,BGTSK3
*	TID,FMAIN,1,0,105
	File Maintenance
*	ESB
*	END
*	SLM,BGTSK4
*	TID,SMAIN,1,0,105
	System Maintenance
*	ESB
*	END

*	SLM,BGTSK8
*	TID,SEDIT,1,0,105
	Source Editor
*	ESB
*	END
*	SLM,BGTSK9
*	TID,DASMR,1,0,105
	DAS MR Assembler
*	ESB
*	END

NOTE:

* = Alphanumeric control record

Figure 15-4. Load-Module Library

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.

Manual loading through the CPU front panel: The key-in loader can be entered manually as follows using the appropriate loader given in table 15-1.

a. Press REPEAT.

b. Enter a STA instruction (054000) in the I register.

c. Clear the P register.

d. Enter a key-in loader instruction in the A register.

e. Press STEP.

f. Clear the A register.

g. Repeat steps (d), (e), and (f) for each key-in loader instruction.

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.

Table 15-1. SGEN Key-In Loaders

Address	Magnetic Tape	Card Reader	RMD
000000	010030	010054	010064
000001	001010	001010	140066
000002	001106	001106	001010
000003	040030	040054	001106
000004	001000	001000	001000
000005	000012	000012	000012
000006	000000	000000	000000
000007	006010	006010	006010
000010	000300	000300	000300
000011	050027	050053	050065
000012	1041zz	1002zz	1004zz
000013	1000zz	002000	1002zz
000014	001000	000046	010063
000015	000021	1025zz	110072
000016	1025zz	002000	1031zz
000017	057027	000046	101uzz
000020	040027	1026zz	000023

(continued)

Table 15-1. SGEN Key-In Loaders *continued*

Address	Magnetic Tape	Card Reader	RMD
000021	1011zz	004044	001000
000022	000016	004444	000017
000023	1012zz	057053	1025zz
000024	100006	005001	150071
000025	001000	040053	001016
000026	000021	004450	000012
000027	000500	002000	1000yy
000030	177742	000046	1003zz
000031		1026zz	010064
000032		004044	110072
000033		004450	1031zz
000034		002000	010065
000035		000046	1031xx
000036		1022zz	120070
000037		057053	005012
000040		040053	1031yy
000041		067053	1000xx
000042		040053	1000zz
000043		001000	1014zz
000044		000013	000043
000045		1011zz	1025zz
000046		000000	150071
000047		1016zz	001016
000050		100006	000012
000051		001000	060065
000052		000045	040064
000053		000500	010064
000054		177742	140067
000055			001016
000056			100006
000057			050064
000060			040063
000061			001000
000062			100006
000063			000001
000064			000001
000065			000500
000065			000037
000067			000060
000070			000074
000071			007760
000072			0v0000

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

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
LPcuD	All Statos models***	70-6602 70-6603
CRcuA	Card reader	70-6200
CPcuA	Card punch	70-6320
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
DcuC	Rotating memory**	70-7500
DcuD	Rotating memory**	70-7510

* where c stands for the controller number (0, 1, 2, or 3), and u for the unit number (0, 1, 2, or).

** this disc must be formatted first (see section 18.4).

*** Statos 33 is not supported during system generation.

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) Directive

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)

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 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 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,016,022
```

A system with 620-35 disc requires a special formatting program, described in section 16.4. This program formats disc packs and performs bad-track analysis.

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 15.4.2) 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 13.5.14), 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

aaa,p(1)*xp(2)x...xp(n)*

where

aaa is a three-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

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

Examples: Specify a 16K memory for VORTEX with a foreground blank-common area from 037600 to 037777.

```
MRY,037777,0200
```

Specify an 18,000-word memory for VORTEX with no foreground blank-common area.

```
MRY,18000,0
```

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

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.

The following order is suggested for peripheral controllers:

a. RMDs

b. Operator-communication (OC) device (section 17)

c. Magnetic-tape units

d. Other units

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	E2250	CRT keyboard/display
CRnA	70-6200 (620-22, 620-25)	Card reader: 300 or 600 cards/minute

(continued)

Table 15-2. Model Codes for VORTEX Peripherals *(continued)*

Code	Model Number	Description
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
DnC	70-7500 (620-35)	Rotating memory
DnD	70-7510 (620-34)	Rotating memory
PTnA	70-6320 (620-55A)	Paper-tape reader/punch
LPnA	70-6701 (620-77)	Line Printer
LPnE	70-6603 (620-76)	Statos-31,-41 Printer/plotter
LPnG	70-6603 (42,51,71)	Statos-31/42 Printer/plotter
LPnH	70-7702	Statos-31 (-41,-51,-52)
LPnJ	70-660	Statos-33
CInA	See sec. 19	Process I/O
COnA	See sec. 19	

Note: Other peripheral devices can be added to the system by creating an EQP directive with a unique phsyical-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).

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, and one model A Teletype.

```
EQP,D0B,016,020,3
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
```

The paper width of each Statos 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 Statos model code (D,E,G,H, or J)

c is the width code, defined as

0 = 8-1/2-inch
1 = 11-inch
2 = 14-7/8-inch
3 = 22-inch

Example: Specify a SGEN directive for model G Statos 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 seven 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.

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	53	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,53,X;D01C,50,*
PRT,D01A,40,*;D018,60,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

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).

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
SI = 2	BI = 6	PO = 10
SO = 3	BO = 7	DI = 11
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
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. 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.

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 a load 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,PROG1,PROG2,PROG3
```

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 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 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 a load 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
```

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
```

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 package 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 TBxxxx, where xxxx is the hardware code 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

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.

```
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.

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

interrupt is the number of milliseconds in the user interrupt interval

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 151, the interrupt interval is 155 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
```

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 (paragraph 15.6.2).

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 LMGEN 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
```

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

part is the maximum number (1 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

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 empty TIDBs, 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 no bad track analysis is wanted.

```
EDR,S,0,0,5,44,26,L
```

Specify full system generation with 100 empty TIDBs, 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,100,0500,20,30,29,0,1
```

15.5.16 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 statos printer/plotter require a DEF directive to define the bed width. The format is:

DEF, V$SWcm, a
where c = controller number
m = model code
a = 0 for 8½ inches
1 for 11 inches
2 for 14⅞ inches
3 for 22 inches

15.6 BUILDING THE VORTEX NUCLEUS

If a full system generation has been requested by the S form of an EDR directive (section 15.5.15), 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**
- **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 section 15.5.5 and 15.5.6, 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

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,levl

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

Example: Define a foreground resident task PROG1 on priority level 10.

```
TDF,TIDPR1,PROG1,0,07401,10
```

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 section 15.5.13 for generation of resident tasks without resident TIDB.

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 Memory Parity Considerations

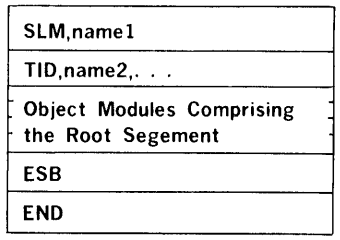
Memory parity is not a supported feature under VORTEX. For those systems which require the use of memory parity, the user may write his own memory-parity service routine (see section 14) and add it to the system. The following are considerations when using memory parity:

- 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$LMEMBLK) 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.

15.7 BUILDING THE LIBRARY CONFIGURATOR

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 library generator 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 15.5.5 and 15.5.6, 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.

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.
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. A background task is protected only if bit 8 is set.
8	Protected task	The task is protected.
7	Task scheduled by time increment	The task becomes nonsuspended when a specified time interval is reached. Prerequisite: Resident TIDB (bit 11).
6	Time delay active	The clock decrements the time counter that, upon reaching zero, clears bit 14.
5	Task checkpointed	The background task is checkpointed 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.

15.7.1 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

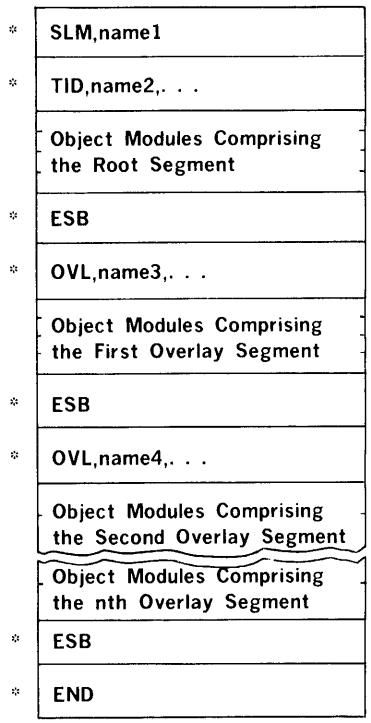
Once a TID directive is input and processed, object modules are input, processed, and output to the specified logical unit until the ESB directive (section 15.7.4) is found.

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,PROG2,1,4,106
```



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
```

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. At this time, the OM library should be loaded and built 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 job stream. A system generation leaves the SGL positioned just prior to this EOF. For card and magnetic tape SGLs this EOF 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.

If the supplemental Writable Control Store (WCS) material is to be added to the Object Module Library, its job stream should be executed at this time. The library routines which use WCS replace their non-WCS equivalents.

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             000100
 .               .
 .               .
 .               .
ZZZ             025863

NONSCHEDULED TASKS

NAME            LOCATION

ABC             022620
DEF             014640
 .               .
 .               .
 .               .
XYZ             011400
```

Figure 15-7. VORTEX Nucleus Load Map

```
LOAD MODULE:    ABC

CATALOGED ON:   D00H

NAME            LOCATION

MOP         A   032556
QRS         R   000200
 .          .      .
 .          .      .
 .          .      .
TUV         A   032501

LOAD MODULE:    CDE

CATALOGED ON:   D10A

NAME            LOCATION

GHI         R   000010
JKL         R   000012
 .          .      .
 .          .      .
 .          .      .
MNO         R   000077
```

Figure 15-8. Library Processor Load Map

```
RMD PARTITIONING

NAME    FIRST       LAST        BAD
        TRACK       TRACK       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

```
CORE RESIDENT TASKS

NAME            LOCATIONS

PROG1           014630
PROG2           014630
PROG3           NOT FOUND
PROG4           014500
```

Figure 15-10. Resident-Task Load Map

15.9 SYSTEM GENERATION EXAMPLES

EXAMPLE 1

Problem: Generate a VORTEX system using the following hardware:

a. Computer with 16K main memory

b. A model 70-7610 disc unit with device address 016

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

and having the characteristics listed below:

a. Foreground common size = 0200

b. Storage/reentry stack area size = 0200

c. Number of empty TIDBs = 20

d. Number of disc partitions = 9

e. All eight interrupt lines connected through a common interrupt handler

f. One user-coded program added to the resident module (PROG1)

g. JCP replaced with a new version

h. One user-coded load module added to the background library (after LMGEN) (PROG2)

i. The system file listed after 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,037777,0200** **EQP,D0B,016,1,020,10** **EQP,TY0A,01,1,0,0** **EQP,CR0A,030,1,0,0** **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, = 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,BGTSK1** **LAD,BGTSK2** **EDR,S,20,0200,9,44,26,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 JCP** **READY**
4	Load revised version of BGTSK1 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 BGTSK2** **READY**
6	Load the PROG1 load module in the card reader, and on DIR type **ALT**	Reads and processes PROG1, and outputs **READY**

Procedure: *(continued)*

Step	User Action	SGEN Response
7	Load the PROG2 load module in the card reader, and on DIR type **ALT**	Reads and processes PROG2, and outputs **READY**
8	Load the remainder of the load module library in the card reader, and on DIR type **LIB**	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 **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.

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

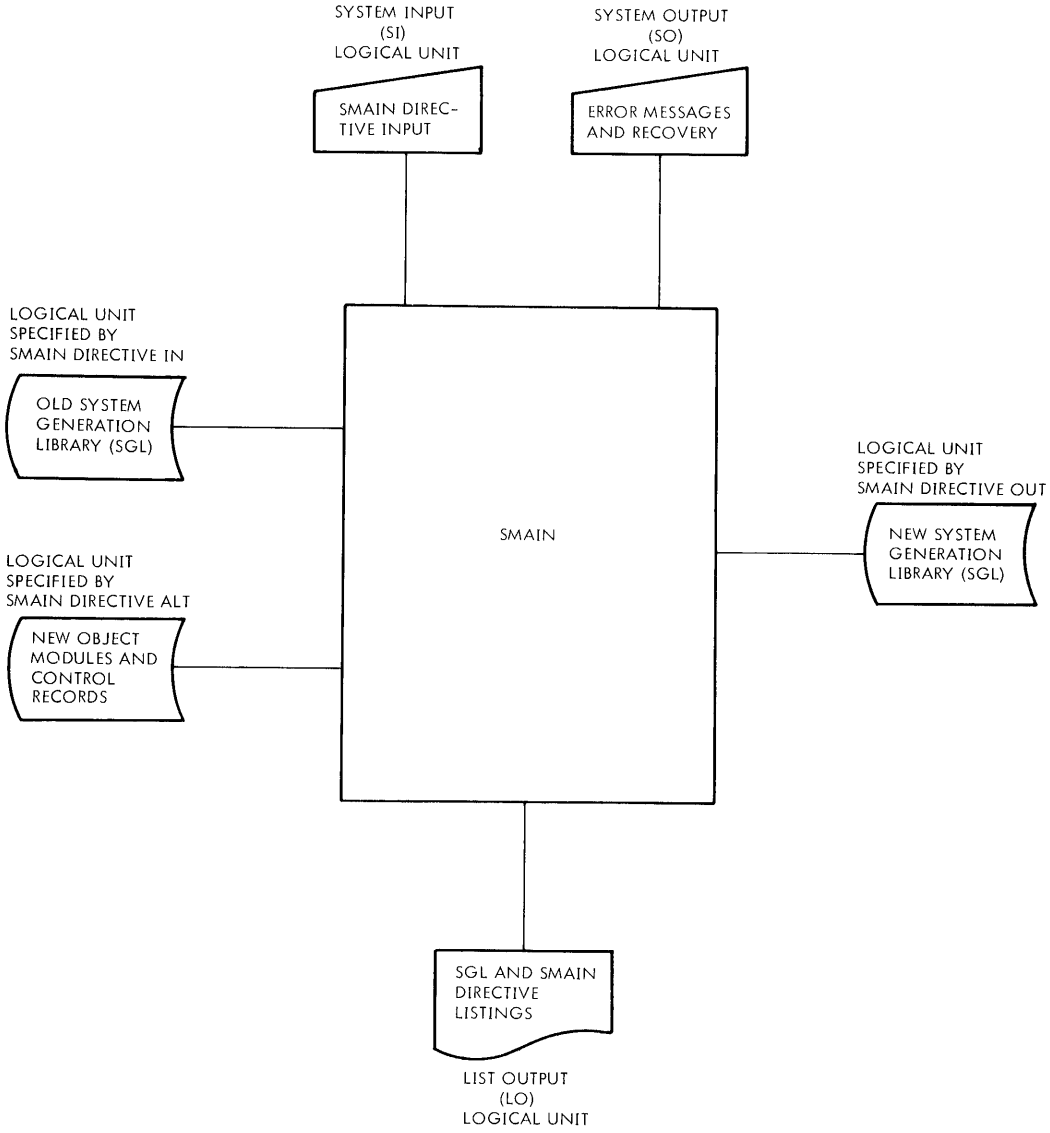
The VORTEX **system-maintenance component (SMAIN)** is a background task that maintains the **system-generation library (SGL)**. The SGL (figure 14-1) comprises all object modules and their related control records required to generate a generalized VORTEX operating system.

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.



VTII-1364

Figure 16-1. SMAIN Block Diagram

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 record: 1) control records and comments (ASCII), 2) the system-generation relocatable loader (the only SGL absolute core-image record), 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
ENDOFSGL
```

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 section on the system-generator component (section 13).

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 section 15.

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 when **lun** is an RMD partition

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.

If **lun** specifies an RMD partition, the RMD is rewound to the first sector following the partition specification table (PST, section 3.2) before any processing takes place. The PST comprises one entry defining the entire RMD.

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.

```
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, 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
```

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.

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 "SM**" 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.

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 14.2.4), 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 14.2.3). Input

```
REP,IOCTL
SM**
```

then, when the message

```
REP IOCTL
```

appears, continue as for an ADD directive (section 14.2.4).

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.

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.

```
PAGE    1  11/13//2              VORTEX  SMAIN

 IN,M1
 OUT,PU
 LIST
 BOUTLODR
 ID NAME     DATE      SIZE      ENTRY NAMES      EXTERNAL NAMES
 V$SGENLD  10/02/72     1551    SGLDR              TPROG   SGIBUF
                                                   BSTACK  $PUN
                                                   $PUB    $LUN
                                                   $LUB
 ID NAME     DATE      SIZE      ENTRY NAMES      EXTERNAL NAMES
 V$D00A1   02/24/72       36    D00A1              DRWEOF  DRSTAT
                                                   DRSKRD  DRSFIL
                                                   DRRITE  DRREWD
                                                   DRREAD
 ID NAME     DATE      SIZE      ENTRY NAMES      EXTERNAL NAMES
 V$D00A2   02/24/72       36    D00A2              DRWEOF  DRSTAT
                                                   DRSKRD  DRSFIL
                                                   DRRITE  DRREWD
                                                   DRREAD
 ID NAME     DATE      SIZE      ENTRY NAMES      EXTERNAL NAMES
 V$D00A5   02/24/72       36    D00A5              DRWEOF  DRSTAT
                                                   DRSKRD  DRSFIL
                                                   DRRITE  DRREWD
                                                   DRREAD
 ID NAME     DATE      SIZE      ENTRY NAMES      EXTERNAL NAMES
 V$D10A1   02/24/72       36    D10A1              DRWEOF  DRSTAT
                                                   DRSKRD  DRSFIL
                                                   DRRITE  DRREWD
                                                   DRREAD
 ID NAME     DATE      SIZE      ENTRY NAMES      EXTERNAL NAMES
 V$D10A2   02/24/72       36    D10A2              DRWEOF  DRSTAT
                                                   DRSKRD  DRSFIL
                                                   DRRITE  DRREWD
                                                   DRREAD
 ID NAME     DATE      SIZE      ENTRY NAMES      EXTERNAL NAMES
 V$D10A5   02/24/72       36    D10A5              DRWEOF  DRSTAT
                                                   DRSKRD  DRSFIL
                                                   DRRITE  DRREWD
                                                   DRREAD
 ID NAME     DATE      SIZE      ENTRY NAMES      EXTERNAL NAMES
 V$D20A1   02/24/72       36    D20A1              DRWEOF  DRSTAT
```

Figure 16-2. SMAIN LIST Directive Listing

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 LMGEN 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,LMGEN
DEL,B,C,D,E
REP,JCPDEF
END
/ENDJOB
```

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.

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-31
MTcu	Magnetic tape unit
PTcu	High-speed paper tape reader/
TYcu	Teletype printer/keyboard
CIma, COma	Process I/O

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).

cu can be omitted to specify unit 0 controller 0, e.g., CR00 or CR.

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.

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*

where

task is the name of the foreground task to be scheduled

level is the priority level (from 2 to 3) 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**

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.

```
;SCHED,DOTASK,3,FL,F
```

Schedule on priority level 9 the resident foreground task COPYIO.

```
;SCHED,COPYIO,9,0,0
```

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.

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,9,0,0,0830
```

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 (1-8) 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 17) 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 1, 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 1, line 3).

```
;ATTACH,INTRPT,13,020
```

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.

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 (00 to 12)

dd is the day (00 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.

Operator key-in example: Set the system date to 25 December 1971.

```
;DATE,12/25/71
```

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 2 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 0 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
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
2	14	Task loaded in background (checkpoint) area
2	13	Load overlay
2	12-0	Unused

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 of 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 10 and 11) of 077777 and 077430, respectively. The latter two octal complement counters show zero minutes and 0340 5-millisecond increments.

Request the output of the status of all foreground tasks inputs.

```
;TSTAT
```

and receive as a typical response

```
VZDB      P24 S047401, 000000 TM077311 TS071000
V$TYA     P23 S047411, 000000 TM077005 TS071011
V$TYA     P23 S047411, 000000 TM077200 TS076000
VZLPA     P22 S047401, 000000 TM077002 TS022000
VZCRA     P22 S047401, 000000 TM077000 TS070221
VZMTA     P22 S047401, 000000 TM077200 TS071000
VZMTA     P22 S047401, 000000 TM077200 TS071000
V$OPCM    P10 S005405, 020000 TM077020 TS077033
JCP       P01 S044400, 000000 TM077000 TS070005
```

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 15-1)

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 15-1)

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 15-1)

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. 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, and the repeated number where no name is assigned to the logical unit. Logical units without names assigned at system-generation time are not listed and must be individually specified by number.

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
```

SECTION 18
OPERATION OF THE VORTEX SYSTEM

This section explains the operation of devices in the VORTEX system, the loading of the system bootstrap 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 70-6201)*

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 70-66XX)*

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).

f. Turn off WRITE PROTECT.

18.1.10 Moving-Head Disc Units
(Model 70-7500)

a. Mount the disc pack

b. Press POWER-ON button and wait for unit to reach operating speed and for the heads to emerge

c. Press on-line button.

18.1.11 Moving-Head Disc Units
(Model 70-7510)

a. Mount the disc pack(s).

b. Turn power on and wait for the unit(s) to reach operating speed (unit-ready light comes on).

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.

Table 18-1. Key-In Loader Programs

Address	Drum -48,49	Disc 70-7510	Disc 70-7500	Disc 70-7600 or -7610
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		1032yy	1031xx	1000xx
001156		1000xx	1032yy	1000zz

Table 18-1. Key-In Loader Programs *(continued)*

Address	Drum -48,49	Disc 70-7510	Disc 70-7500	Disc 70-7600 or -7610
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	102515	
001172		006140	001016	
001173		000012	001130	
001174		001002	005144	
001175		000600	001040	
001176		001000	000600	
001177		001146	001000	
001200		000000	101146	

where xx = even BIC address, yy = odd BIC address, and zz = device address.

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.

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 as follows:

a. Press REPEAT.

b. Load an STA instruction (054000) into the I register.

c. Load 001130 into the P register.

d. Load a key-in loader instruction into the A register.

e. Lift the STEP/RUN switch to STEP.

f. Clear the A register.

g. Repeat steps (d), (e), and (f) for each bootstrap instruction.

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.

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     FIRST     LAST      BAD
NAME          TRACK     TRACK     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     FIRST     LAST      BAD
NAME          TRACK     TRACK     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).

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 parition 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, 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

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.

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 overlayed, 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 70-7500 (620-35) DISC PACK FORMATTING PROGRAM

Each 70-7500 (620-35) disc pack requries 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 II 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 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 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 correpsonding 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. Head crashes generally indicate formatting should be done again.

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, 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 without an operating system. It may be loaded and executed either with AID II or 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. Head crashes generally indicate formatting should be done again.

18.6 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

VORTEX supports a number of VDM devices which are used in industrial applications for a wide range of monitor and control purposes. These devices are called 'Process Input/Output' devices and are listed below:

VDM Model	Description
70-8310 and -8311 (620-830A/B)	Digital Output Module User's Guide (98 A 9968 100)
70-8410 and -8411 (620-831A/B)	Digital Input Module User's Guide (98 A 9968 110)
70-800x and 70-801x (620-850/851)	Analog-to-Digital User's Guide (98 A 9968 060)
70-8020 and -8021 (620-860/860/A 70-8022 and -8023 (620-861/861A)	Converter/Multiplexor User's Guide (98 A 9968 070)
70-821x,8220,8221 (620-870/1/2/ 3/4/5, 620-870A/B, 620-871A/B, 620/872A/B)	Digital-to-Analog Module User's Guide (98 A 9968 050)
70-811x,812x (620-855xx)	Low Level Multiplexor User's Guide (98 A 9968 130)

VORTEX configurations which include Process Input/Output devices differ from others in that each is, to some degree, 'tailor-made', even though they are composed of the standard products listed above. This requires the VORTEX user to operate with VORTEX I/O features at a more fundamental level than with most other devices. For this reason, the operation of Process Input/Output devices under VORTEX will be presented in considerable detail in the following sections.

The VORTEX Support Library includes a number of subroutines (section 19.4) with FORTRAN calling sequences defined by the Instrument Society of America (ISA), which are useful for input, output, and manipulation of process data.

19.2 PROCESS OUTPUT

19.2.1 Hardware

VORTEX supports combinations of the 70-8310 (620-830A) Digital Output Module and the 70-8311 (620-830B) Digital Output Expansion Module. VORTEX also supports combinations of the following DAC (Digital-to-Analog Converter) modules and expansion modules: (620-870,-870A,-870B,-871,-871A, -871B,-872,-872A,-872B,-873,-874,-875).

Eight device addresses (050-057) are available for these modules. Each address can hold up to four modules, each module containing two digital output registers or DAC's for a maximum of 64 registers of DACs.

For VORTEX operation, a device is defined as the collection of modules at a single device address, and the word 'device' will have this meaning for the remainder of this section. The word 'channel' will be used to mean either a digital output register or a DAC.

Software capabilities for referencing channels directly by number are provided. For this purpose, channels are assigned an (octal) number mn, where:

m = (device address–050)
n = hardware channel number (0-7) within device.

thus, for example, the channel selected by the command

```
EXC2  0352
```

would be called channel number 023.

Process output is totally under control of software (BICs, interrupts, or SENs are used). Therefore, no ready, complete, or error information is provided by the hardware.

19.2.2 SGEN Operations

The following SGEN operations must be performed to include Process Output capabilities in a VORTEX system:

a. Add EQP directives to SGEN directive input file.

b. Add ASN directives to SGEN directive input file.

Note: the SGL contains four input controller tables, four output controller tables, input and output drivers, and TDF records.

In the examples in the following discussions, the symbols 'm' and 'n' refer to register number mn.

The EQP Directive

Each device must have an EQP directive in the SGEN directive file, with the following format:

```
EQP,COmA,050+m,1,0,0
```

For example, the device at address 053 will require the directive:

```
EQP, CO3A, 053, 1,0,0,alg,mul
```

The ASN Directive

Each device must be assigned to a logical unit number by any ASN directive of the following format:

```
ASN, lun = COm0
```

For example, assigning the device at address 053 to logical unit 24 will require the directive:

```
ASN, 24 = CO30
```

19.2.3 Output Calls

Output to a Process Output device is by use of the IOC 'WRITE' macro. FORTRAN source programs can request output by calling one of the ISA process control subroutines described in section 19.4, which will construct and execute such a macro.

The macro call has the format (see section 3.4.4):

```
WRITE   pcb,lun,wait,mode
```

where:

pcb = Name of Process Control Block (PCB)
lun = Logical Unit Number
wait = Wait Flag
mode = Data Mode (ignored)

Data is always output directly, without modification, so the Data Mode is effectively System Binary.

PCB format is:

Output Word Count C	Word 0
Output Buffer Address	Word 1
Address of Channel Number List	Word 2
Status Word Address (0 if none)	Word 3
Mask Word Address (0 if none)	Word 4
Pulse Width Word Address (0 if none)	Word 5

The Channel Number List is a sequential list of channel numbers m(i)n(i) (i = 1,C), where M(i) = m(1) for all i, and the device address to which the logical unit number is assigned is 050 + m(i). Thus, a single WRITE call can only reference those channels assigned to a single device address.

The Status Word is a word in the calling program in which status of the IOC call is maintained. This is required by the ISA subroutines of section 19.4.

The Mask Word is used by the ISA 'Latching' subroutines DOL and DOLW. 1-bits in this word flag bits that are to be updated. The device controller table will contain the previous setting of all bits in the output word and the output buffer will contain the new settings.

An error I003 is reported if the Channel Number List contains a channel mn where m is not in range 0-7, or if m does not correspond to the device address defined by the ASN directive at SGEN time.

The Pulse Width Word is used by the ISA 'Momentary' subroutines DOM and DOMW. It gives the time in VORTEX basic cycles (5-millisecond) that output points are to remain set.

Example 1:

A DASMR source program is to output the first 3 words from buffer OBUF to channels 023, 027, and 021 in a group of Digital Output Modules which are assigned to logical unit number 24.

Note that channels 023, 027, and 021 are all assigned to the module at device address 052 by the channel numbering convention.

```
          .
          WRITE    PCB1,24,0,0
          .
          .
PCB1      DATA     3
          DATA     OBUF
          DATA     PTLIST
          DATA     0,0,0
          .
          .
PTLIST    DATA     023,027,021
          .
          .
```

Example 2:

A FORTRAN program is to output the first 3 words of OBUF to analog channels 49, 50, and 53, which are assigned to logical unit 17. The octal equivalents of these channel numbers are 061, 062, and 065, so the device address of the output module is 056 (46 in decimal digits).

```
          .
          .
          INTEGER STAT, PTLIST, OBUF
          DIMENSION OBUF (3), PTLIST (3)
          DATA PTLIST/49, 50, 53/
          .
          .
          CALL V$OPIO (46, 17, 0, STAT)
          .
          .
          CALL AO (3, PTLIST, OBUF, STAT)
          .
          .
```

19.3 PROCESS INPUT

19.3.1 Hardware

VORTEX supports combinations of the 70-8410 (620-831A) Digital Input Module and the 70-8411 (620-831B) Digital Input Expansion Module. VORTEX also supports combinations of the 70-8010 (620-850) and the 70-8011 (620-851) Analog Input System, the 70-8020 (620-860) and 70-8022 (620-861) High-Level Multiplexor Modules and the 70-8021 (620-860A) and the 70-8023 (620-861A) High-Level Multiplexor Expansion Modules, and the 70-811x (620-855x) Low-Level Analog Input System and the 70-812x Low-Level Multiplexor Expansion Modules. These provide from 1 to 2,048 digital or analog input channels.

Eight device addresses (060 to 067) are available for these modules. Each address can handle, through multiplexing, up to 256 digital channels. To each of these device addresses will correspond a multiplexor attached to a different device address in the range (040-077). All Process Input requires a Buffer Interlace Controller (BIC).

Software capabilities are provided for referencing channels directly by number. Each channel is assigned an (octal) number mn by the following rules:

m = (device address – 060)
n = channel number (0-0400) within device. n is a 3-digit octal number

Thus, for example, channel number 01003 would be selected by outputting a 3 as the select code to the multiplexor which is connected to the Analog-to-Digital convert whose address is 061.

A BIC will be used for all input and all input will end with a BIC complete interrupt. The BIC will operate with the programmable timer.

19.3.2 SGEN Operations

The following SGEN operations must be performed to include Process Input capabilities in a VORTEX system:

a. Add EQP directives to SGEN directive input file.

b. Add ASN directive to SGEN directive input file.

c. Add PIM directive to SGEN directive input file.

In the example in the following discussions, the symbols 'm' and 'n' refer to register number mn.

The EQP Directive

Each device must have an EQP directive in the SGEN directive file, with the following format:

```
EQP,CImA,060+m,1,b,0,ioa,ma
[b = BIC device address]
[ioa = I/O algorithm as decimal
fraction, see section 14.4.3]
[ma = multiplexor address]
```

For example, the device at address 063 using the BIC at address 020 with I/O algorithm value of .5 and multiplexor address 072 will require the directive:

```
EQP,CI3A,063,1,020,0,.5,072
```

The ASN Directive

Each device must be assigned to a logical unit number by an ASN directive of the following format:

```
ASN,lun=CIm0
```

For example, assigning the device at address 063 to logical unit number 21 will require the directive:

```
ASN,21=CI30
```

The PIM Directive

Linkage must be established between the BIC and its Priority Interrupt Module (PIM) by a PIM directive of the format:

```
PIM,pl,TBCImA,1, 0
```

where: p = PIM number (single octal digit)
l = line number (single octal digit)

I/O Algorithm

The I/O algorithm value must be set for the highest transfer rate (smallest PCB Timer Count) that will be used in the system.

1.10 x (BIC RATE*/DEVICE RATE)

Rates are in microseconds.

* BIC rate represents the maximum trap-in, trap-out timing sequence on the E-bus.

19.3.3 Input Calls

Input to a Process Input device is by use of the IOC 'READ' macro. FORTRAN source programs can request input by calling one of the ISA process control subroutines described in section 19.4, which will construct and execute such a macro.

The macro call has the format (see section 3.5.3)

```
READ    pcb,lun,wait,mode
```

where:

pcb = Name of Process Control Block (PCB)
lun = Logical Unit Number
wait = Wait Flag
mode = Data Mode (ignored)

Data is always input directly, without modification, so the Data Mode is effectively System Binary.

PCB format is:

Input Word Count C	Word 0
Input Buffer Address	Word 1
Address of Channel Number	Word 2
Status Word Address (0 if none)	Word 3
Op Code	Word 4
Timer Count	Word 5

The Status Word is a word in the calling program in which status of the IOC call is maintained. This required by the ISA subroutines of section 19.4.

The Op Code (OP) is defined thus:

OP = 0:

Sequential Mode. Let m00n be the channel number specified by word 2. Data is repeatedly input from channels m001-m00n, till the input word count C (Word 0) is satisfied.

OP = 1:

Random Mode. Channel mn is repeatedly input the number of times specified in word 0.

The Timer Count (Word 5) is the desired time, in microseconds, between inputs. This value is output to the programmable timer, which will control the BIC input rate.

An error (I003) is reported if m is not in range 0-7, if n (or C, if in sequential mode) is not in range 0-255, or if m does not correspond to the device address defined by the ASN directive at SGEN time.

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:

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/2051/
      .
      .
      .
      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 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 is:

STARTING CHANNEL ADDRESS	Word 0
GAIN OF CHANNELS	Word 1

The format of the channel tables when fun = 1 is:

Word
0 = ADDRESS OF CHANNEL a
1 = GAIN CODE FOR CHANNEL a
2 = ADDRESS OF CHANNEL b
3 = GAIN CODE FOR CHANNEL b
4 = ADDRESS OF CHANNEL c
etc.

The gain is internally referenced by the following table

Gain parameter	Actual MUX Gain
0	8
1	16
2	32
3	64
4	128
5	256
6	512
7	1024

Therefore the gain parameter must be in the range of 0 through 7.

An error (IO03) is reported if the gain is not in the proper range.

Example: In a DAS MR program, set the gain to 256 (gain code 5) on 27 contiguous channel (starting from 04001), which are assigned to logical unit 36.

Delay 5 milliseconds after the gains have been set to give the amplifier time to settle.

```
          .
          .
          FUNC     LDCB,36,0
          .
          .
          .
LLDCB     DCB      27,TABLE,0
          .
          .
          .
TABLE     DATA     04001,5
          .
          .
          .
```

Example 2: A DAS MR program is to set the gain of 3 random channels which are assigned to logical unit 37. Return after the gains have been set. The gain of channel 04001 will be set to 64 (gain code 3), the gain of channel 04031 will be set to 512, and the gain of 04007 to 8.

```
          .
          .
          .
          FUNC     LLDCB,37,0
          .
          .
          .
LLDCB     DCB      3,TABLE,3
          .
          .
          .
TABLE     DATA     04001,3,04031,6,04007,0
          .
          .
          .
```

19.4 ISA 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 of this group:

Input/Output Calls

AISQ(W): Analog Input Sequential
AIRD(W): Analog Input Random
AO(W): Analog Output
DI(W): Digital Input
DOM(W): Digital Output-Momentary
DOL(W): Digital Output-Latching

The (W) option with each of these subroutine names selects a 'wait' mode, that is, it specifies that return is not be made from the subroutine until the I/O is finished, either normally or erroneously.

Bit String Manipulation

IOR: Inclusive OR (logical add)
IAND: AND (logical multiply)
NOT: NOT (logical invert)
IEOR: Exclusive OR (logical subtract)
ISHFT: Logical Shift

VORTEX also provides two FORTRAN subprogram calls to set the amplifier gains on the Low-Level Multiplexors. The gain control calls are not ISA standard calls.

Low Level Gain Calls

SGNF(D): Set gain on sequential channels
SGNT(D): Set gains through a table

The (D) option of each of these routines cause a 5 millisecond delay after the last gain control has been issued, to give the amplifiers time to settle.

19.4.1 Input/Output Calls

The parameter 'stat' appears in all the following I/O calls. Its contents give the status of the call, as follows:

stat =
1: I/O correctly completed
2: I/O in execution
3: Invalid channel number
4: BIC timeout error
5: Invalid parameter value

VORTEX provides a FORTRAN call which establishes execution-time association between channel numbers and logical unit numbers, and sets the timer for data input rate. The format is:

```
CALL V$OPIO (da, lun, time, stat)
```

where:

da	=	device address
lun	=	logical unit number
time	=	time, in microseconds, between input. This is loaded into device programmable timer, which controls BIC rate. It is ignored on output. Parameters may be redefined by successive calls to V$OPIO.

Read Analog Input Sequential

```
CALL AISQ (count, ptlist, ibuf, stat)
```

or

```
CALL AISQW (count, ptlist, ibuf, stat)
```

This call reads *count* analog inputs into buffer *ibuf*, starting with channel 0X001, where *ptlist* contains 0XYYY, and reading channels sequentially.

Read Analog Input Random

```
CALL AIRD (count, ptlist, ibux, stat)
```

or

```
CALL AIRDW (count, ptlist, ibuf, stat)
```

This call reads *count* analog inputs into buffer *ibuf*, inputting from the list of random points *ptlist*.

Perform Analog Output

```
CALL AO (count, ptlist, obuf, stat)
```

or

```
CALL AOW (count, ptlist, obuf, stat)
```

This call outputs *count* analog values from buffer *obuf*, outputting to the list of random points *ptlist*.

Read Digital Input

```
CALL DI (count, ptlist, ibuf, stat)
```

or

```
CALL DIW (count, ptlist, ibuf, stat)
```

This call reads *count* words of digital input into buffer *ibuf*, inputting from the list of random digital channels *ptlist*.

Perform Digital Output - Momentary

```
CALL DOM (count, ptlist, obuf,
time, stat)
```

or

```
CALL DOMW (count, ptlist, obuf,
time, stat)
```

This call outputs *count* words of digital output from buffer *obuf*, outputting from the list of random digital channels *ptlist*. If *time* = 0 this completes the operation. Otherwise, after 5**time* in milliseconds a word of zeros will be output to every channel in *ptlist*, thus resetting all channels.

Perform Digital Output - Latching

```
CALL DOL (count, ptlist, obuf,
mask, stat)
```

or

```
CALL DOLW (count, ptlist, obuf,
mask, stat)
```

This call outputs *count* words of digital output from buffer *obuf*, outputting from the list of random digital channels *ptlist*. The device driver program will save the previous word output to each channel, and change only those bits specified by 1-bits in *mask*, which is an integer array parallel to *obuf* and *ptlist*.

Perform Gain Selection or Sequential Channels

```
CALL SGNF (chntbl,nochnl)
```

or

```
CALL SGNFD (chntbl,nochnl)
```

This call selects the gain on *nochnl* sequential low level input channels. *Chntbl is the name of a two word control table. The first word contains* the address of the first low level channel. The second word contains the gain parameter (0-7).

Perform Gain Selection on Channels through a Table

```
CALL SGNT (chntbl,nochnl)
```

or

```
CALL SGNTD (chntbl,nochnl)
```

This call selects gains on *nochnl* low level channels. *Chutbl* is the name of a table which contains a pair of words for control for each low level channel. The first word of each pair contains the address of the low level channel. The second word of each pair contains the gain parameter (0-7).

19.4.2 Bit String Operations

All these subprograms are defined as Integer Function Subprograms. In the following descriptions, m and n are integer mode expressions.

IOR(m, n) = m.OR.n	Inclusive OR (logical sum)
IAND(m, n) = m.AND.n	AND (logical product)
NOT(m) = NOT.m	NOT (logical invert)
IEOR(m, n) = n.XOR.n	Exclusive OR (logical difference)
ISHFT(m,n) = 0	If the absolute value of m ≥ 16
m*2**n	Otherwise

19.5 ERRORS

Process Output

IO03	INVALID CHANNEL NUMBER

Process Input

IO03	INVALID CHANNEL NUMBER
IO2X	BIC TIMEOUT ERROR

19.6 EXTENSIONS

Other process control devices besides those in the table of section 19.1 may be brought into the VORTEX system at some future time. The procedure for entering a new process control device is as given for the currently supported devices: one codes a driver program and controller tables and enters them into the VORTEX Nucleus at SGEN time, remembering to increment the one-character suffix on all names (all names herein end in 'A'; the next type of DAC, say, would be tagged with 'B'). The controller table can be extended by as many words as desired, to store flags and fixed device parameters. For variable parameters, say a gain parameter on an analog input device, the PCB table can be extended to hold the new parameter. In the FORTRAN I/O calls, the array PTLIST can be made 2-dimensional if gain or other parameter information is to be transferred with each point or channel number.

SECTION 20
WRITABLE CONTROL STORE AND FLOATING-POINT PROCESSOR

The Writable Control Store (WCS) option extends the Varian 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 Varian 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 Varian 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 Varian Microprogramming Guide.

20.1.1 Microprogram Assembler

The MIDAS program allows the user to prepare microprograms for Varian 70 series WCS. Through the use of operation mnemonics, symbolic addressing, address-field calculations, macro definitions, error detection and automatic program documentation, MIDAS makes writing microprograms easier.

Under VORTEX, MIDAS is scheduled from the background library at level 0 by

```
/LOAD,MIDAS
```

20.1.2 Microprogram Simulator

The Varian 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 the 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 integers
- 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 the two-word integer and real arithmetic, the user's program uses the CALL statement, in the format used for other support library functions described in section 13. To execute other 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.

20.2.1 Fixed-Point Arithmetic Firmware

Two-word integers use the following arithmetic firmware:

Mnemonic	Function	BCS Call
XAD	Fixed-point add	0105334
XSB	Fixed-point subtract	0105374
XMU	Fixed-point multiply	0105274
XDV	Fixed-point divide	0105234

These operations are performed on the hardware A and B registers. AB, using the integer specified by the second word of the respective BCS call. If overflow occurs, AB is set to the maximum integer with the proper sign and the overflow flag (OVFL) is set.

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	BSC Call
FAD	Floating-point add	0105134
FSB	Floating-poing subtract	0105174
FMU	Floating-point multiply	0105074
FDV	Floating-point divide	0105034

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.

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
FSE	Pass parameters between subroutines	0105036
FDO	Terminate DO loop	0105035
FDO1	Terminate DO loop (1 increment)	0105027

For FSE, the calling routine would use the following sequence:

```
        CALL    SUB
        DATA    P1          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:

```
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.

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.

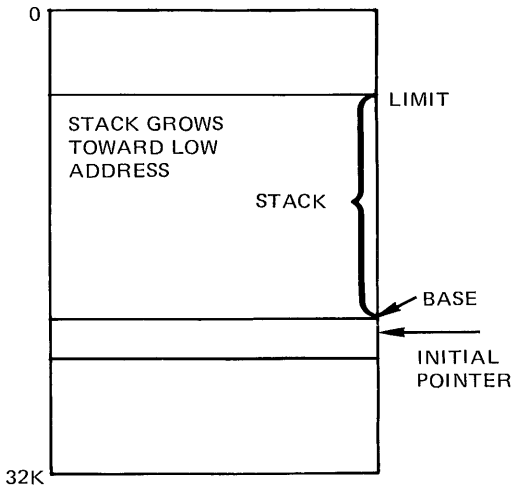


Figure 20-1. Base and Limit of Stack

WORD	
0	CURRENT STACK POINTER
1	LIMIT OF STACK
2	BASE OF STACK
3	ADDRESS OF INSTRUCTION WHICH CAUSED OVERFLOW OR UNDERFLOW
4	ERROR ROUTINE FOR OVERFLOW OR UNDERFLOW

Figure 20-2. 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.

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 replaced them by their 32-bit product (see figure 20-3). The most signficant 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.

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).

If the divided is greater than the divisor, the quotient is unpredictable, and control is returned with the overflow inicator set (OF). If the top-of-stack pointer (PTR) ever exceeds the boundaries of the stack, a JMPM is made to the fourth word in the stack control block.

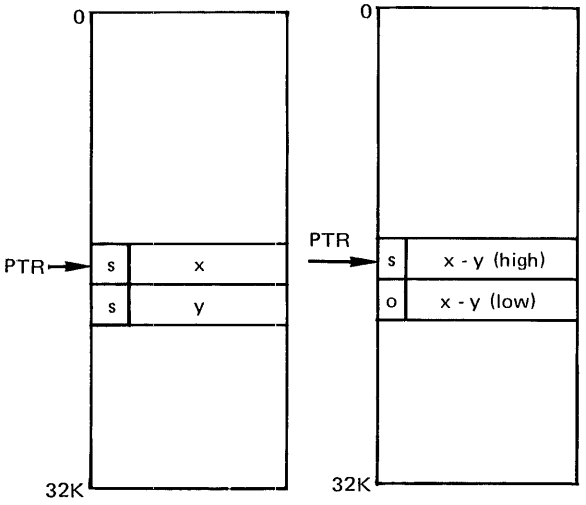


Figure 20-3. Stack Multiply

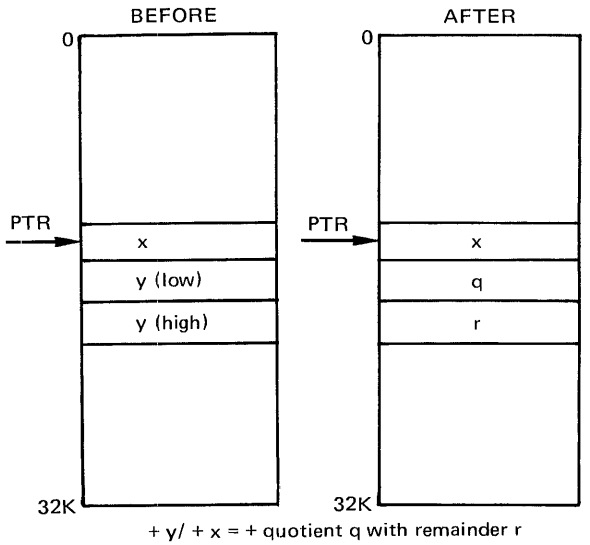


Figure 20-4. Stack Divide

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.)

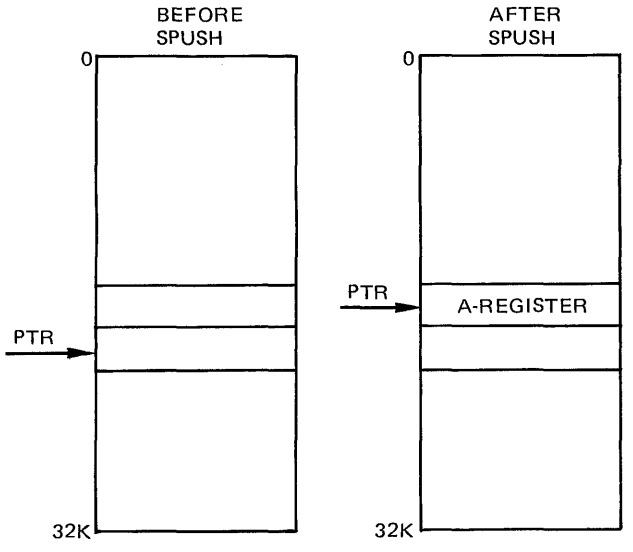


Figure 20-5. Stack Push

Pop (SPOP): the A-register (R0) from the top stack word and increments the stack pointer (see figure 20-6).

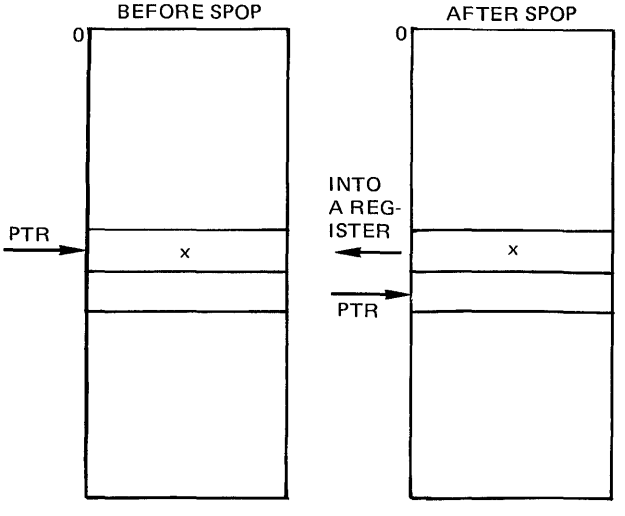


Figure 20-6. Stack Pop

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).

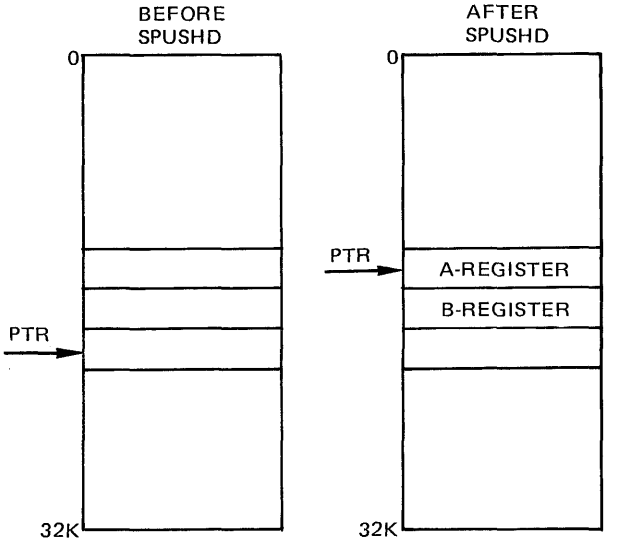


Figure 20-7. Stack Double Push

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).

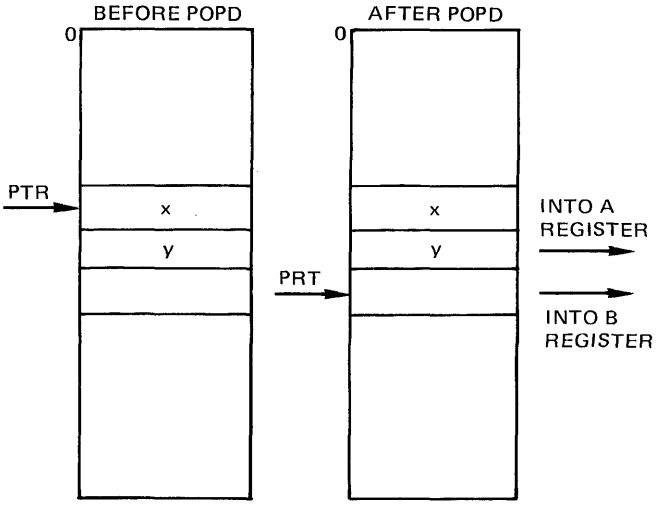


Figure 20-8. Stack Double Pop

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                          XAD     address
        DATA    0105334,P(1)
        EMAC
```

Fixed point subtract:

```
XSB     MAC                          XSB     address
        DATA    0105374,(P1)
        EMAC
```

Fixed point multiply:

```
XMU     MAC                          XMU     address
        DATA    0105274,P(1)
        EMAC
```

Fixed point divide:

```
XDV     MAC                          XDV     address
        DATA    0105234,P(1)
        EMAC
```

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     address,address
        DATA    0105037,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,
                                                lim addr, loop addr
        DATA    0105035,P(1),P(2),
                P(3),P(4)
        EMAC
```

DO loop (one increment):

```
FDO1    MAC                             FDO1    count addr, lim addr,
                                                loop addr
        DATA    0105027,P(1),P(2),P(3),
        EMAC
```

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:

```
SPUPD   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			
FSB	MAC		FSB	address
	DATA	0105450,P(1)		
	EMAC			
FSBD	MAC		FSBD	address
	DATA	0105543,P(1)		
	EMAC			
FMU	MAC		FMU	address
	DATA	0105416,P(1)		
	EMAC			
FMUD	MAC		FMUD	address
	DATA	0105506,P(1)		
	EMAC			
FDV	MAC		FDV	address
	DATA	0105401,P(1)		
	EMAC			
FDVD	MAC		FDVD	address
	DATA	0105535,P(1)		
	EMAC			
FLT	MAC		FLT	address
	DATA	0105425,P(1)		
	EMAC			
FIX	MAC		FIX	address
	DATA	0105621,P(1)		
	EMAC			
FST	MAC		FST	address
	DATA	0105600,P(1)		
	EMAC			
FSTD	MAC		FSTD	address
	DATA	0105710,P(1)		
	EMAC			

APPENDIX A
ERROR MESSAGES

This appendix comprises a directory of VORTEX operating system error messages, arranged by VORTEX component. For easy reference, the number of the subsection containing the error messages for a component ends with a number corresponding to that of the section that covers the component itself, e.g., the file-maintenance error messages are listed in subsection A.9 because the file-maintenance component itself is discussed in section 9.

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:
CM	Concordance program	A.5
DG	Debugging program	A.7
DP	Dataplot II	A.12
EX	Real-time executive	A.2
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.18.1
MU	Microprogram utility	A.18.2
NC	VTAM Network control	A.20
OC	Operator communication	A.17
RP	RPG IV Compiler	A.3
RT	RPG IV Runtime/Loader	A.5.3
SE	Source editor	A.8
SG	System generator	A.15
SM	System maintenance	A.16
ST	VSORT	A.11
*	DAS MR assembler	A.5

Section A.21 gives explanations of error codes listed under "Possible User Action" in the last column of the following sections.

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	M01,M02,M03 M04
EX07,xxxxxx	OVLAY requests a segment not in library	Abort task xxxxxx	D01,D03

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	Abort task xxxxxx	P03
EX12,xxxxxx	I/O link error (foreground task making request, or incorrect logical unit number)	Abort task xxxxxx	P01
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	M06,P01
EX32,xxxxxx	Attempted to schedule a task from a non-RMD unit	Directive ignored	D02,P01
EX33,xxxxxx	Floating-point processor fault, FPP, error	Program continues at the address following the FPP store instruction	None
EX34,xxxxxx	Floating-point processor timeout	Program continues at interrupted instruction	None

Note: **xxxxxx** is the name of a task.

A.3 I/O CONTROL

Message	Condition	Action	Possible User Action
I000,xxxxxx	Unit not ready, or unit file protected	Repeats message until condition is corrected	H01,H03
I001,xxxxxx	Device declared down	Repeats message until condition is corrected	H04,D19
I002,xxxxxx	Invalid LUN specified	Abort task or request	D02,P01
I003,xxxxxx	FDB/DCB parameter error	Abort task or request	P04
I004,xxxxxx	Invalid protection code	Abort task or request	D01,D02,P01
I005,xxxxxx	Protected partition specified by unprotected task	Abort task or request	P01
I006,xxxxxx	I/O request error, e.g., I/O-complete bit not set, prior request may be queued	Abort task or request	H05
I007,xxxxxx	Attempt to read from a write-only device, or vice versa	Abort task or request	D02,P01
I010,xxxxxx	File name specified in OPEN or CLOSE not found	Abort task or request	D01,D03,P01, D29
I011,xxxxxx	Invalid file extent, record number, address or skip parameter	Abort task or request	P04,P01
I012,xxxxxx	RMD OPEN/CLOSE error, or bad directory thread	Abort task or request	H05,D03
I013,xxxxxx	Level 0 program read a JCP (/) directive	Task xxxxxx is aborted, directive passed to JCP buffer	None
I014,xxxxxx	Interrupt timed out or no cylinder-search-complete interrupt	Abort task or request	H05,D05

I015,xxxxxx	Disc cylinder-search or malfunction error	Abort task or request	H05
I016,xxxxxx	Disc read/write timing error	Abort task or request	H05
I017,xxxxxx	Disc end-of-track error	Abort task or request	H05
I020,xxxxxx	BIC1: abnormal stop, not ready, or time out error	Abort task or request	D05,H05
I021,xxxxxx	BIC2: abnormal stop, not ready, or time out error	Abort task or request	D05,H05
I022,xxxxxx	BIC3: abnormal stop, not ready, or time out error	Abort task or request	D05,H05
I023,xxxxxx	BIC4: abnormal stop, not ready, or time out error	Abort task or request	D05,H05
I024,xxxxxx	BIC5: abnormal stop, not ready, or time out error	Abort task or request	D05,H05
I025,xxxxxx	BIC6: abnormal stop, not ready, or time out error	Abort task or request	D05,H05
I026,xxxxxx	BIC7: abnormal stop, not ready, or time out error	Abort task or request	D05,H05
I027,xxxxxx	BIC8: abnormal stop, not ready, or time out error	Abort task or request	D05,H05
I030,xxxxxx	Parity error	Abort task or request	H05,D02
I031,xxxxxx	Reader or tape error	Abort task or request	H05,P19
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
I045,xxxxxx	RMD error. Format, end-of-file or head selection error	Abort task or request	H05,D13
I047,xxxxxx	User write specified word count >73	Record is truncated	P04
I05x,xxxxxx	RMD read error on stream X, specified 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

Note: **xxxxxx** is the name of a task or device.

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 specfied 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 insufficent 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

A.5 LANGUAGE PROCESSORS

A.5.1 DAS MR Assembler

During assembly, the source statements are checked for syntax errors and usage. In addition, errors can occur

where the program cannot determine the correct meaning of the source statement.

When an error is detected, the assembler outputs an error code following the source statement containing the error, on the LO unit, and continues to the next statement.

The assembler error messages are:

Message	Condition
*IL	First nonblank character of the source statement invalid (statement is not processed)
*OP	Instruction field undefined (two no-operation (NOP) instructions are generated in the object module)
*SY	Expression contains undefined symbol
*EX	Expression contains two consecutive arithmetic operators
*AD	Address expression error
*FA	Floating-point number format error
*DC	An 8 or 9 in an octal constant
*DD	Invalid redefinition of a symbol or the location counter
*VF	Instruction contains variable subfields either missing or inconsistent with the instruction type
*MA	Inconsistent use of indexing and indirect addressing
*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 symbolic source statements to be assembled
*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

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 form 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
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

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

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:

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 subroutine or missing subroutine where xxxxxx is the subroutine name	Task aborted	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 loading	Abort loading	P08,H06
LG06	Cataloging error, name already in library, library full	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	Abort loading	P08
LG10	Literal pool overflow or use of literal by foreground program	Abort loading	P08,P09
LG11	Invalid redefinition of common-block size during load-module generation	Abort loading	P08
	Load-module size exceeds available memory	Abort loading	P02
LG13	LMGEN 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	P17
LG17	Attempt to load protected task on background library or unprotected task on foreground library	Abort loading	D01,D02

A.7 DEBUGGING PROGRAM

Message	Condition	Action	Possible User Action
DG01	Invalid DEBUG directive	Ignore directive	D01,D02
DG02	Invalid or undefined parameter in DEBUG directive	Ignore directive	D01,D02

A.8 SOURCE EDITOR

Message	Condition	Action	Possible User Action
SE01	Invalid SEDIT directive	Directive ignored	D01,D02
SE02	Invalid or missing parameter in SEDIT directive	Directive ignored	D01,D02
SE03	Error reported by IOC call	Edit terminated	H06
SE04	Invalid end of file	Edit terminated	P07

A.9 FILE MAINTEANCE

Message	Condition	Action	Possible User Action
FM01	Invalid FMAIN directive	Ignore directive	D01,D02
FM02	Name already in directory	Module not added	D03,D01,D02, D07
FM03	Name not in directory	Module not deleted	D03,D01,D02
FM04	Insufficient space for entry	Module not added	D07,D08,D09
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

* 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,nnnn	End of file or end of tape before the specified number or records skipped, or end of tape before specified number of files skipped. When nn = the number of records remaining when the end-of-file or end-of-device occurred. Note: nn is modulo 100.	SFILE, SREC terminates upon error condition	P07

A.11 SORT ERROR MESSAGES

Message	Condition	Action	
ST01,xxxxxxxx	Invalid or missing parameter or control word for the SORT control word xxxxxxxx	Abort job	D01
ST02	Record lengths for INPUT and OUTPUT unequal and no user exit specified.	Abort job	D01
ST03	Store control field ending character position is less than start character position, or character position is past end of sort record	Abort job	D01
ST04	Insufficient memory available for work space.	Abort job	M01
ST05,xxxxxx	OPEN error on file xxxxxx	Abort job	D01,H06

ST06,xxxxxx	I/O error on file xxxxxx	Abort job	H06
ST07,xxxxxx	Attempt to write past end-of-file xxxxxx. (Work file or output file too small.)	Abort job	D32

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
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
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	D24
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
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	Incomplete system definition (missing directives)	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

MEMORY ERRORS: Errors of compatibility between allocated memory and a portion of the VORTEX system.

Message	Condition	Action	Possible User Action
SG30	Size of nucleus larger than that of defined foreground area	Waits for indicated corrective action	M03,D12
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 OVL control record	Current load module processing terminated, system continues	D01,D17

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
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	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
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
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

A.18.1 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*	D01,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
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

* Input recovery message or corrected directive from SO device.

A.18.2 Microprogram Utility

Message	Condition	Action	Possible User Action
MU01	Input could not be interpreted as a valid command	Directive ignored; input recovery*	D01,D02
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.19 PROCESS INPUT/OUTPUT

There are no error messages unique to this section of the manual.

A.20 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.21 ERROR CODES

A.21.1 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
P@ to reread the program or
/SMAIN@ to restart SMAIN

where @ is a carriage return

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 SYSGEN will find component.)

D14 Restart SGEN from beginning.

D15 Check spelling, delimiters, etc. of IO INTERROGATION.

D16 Correct appropriate SGEN directives as indicated.

D17 Correct indicated module for next SGEN or add corrected module with LMGEN 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
70-76XX	203 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.

A.21.2 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, bit 0-7
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

A.21.3 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, retry 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.

A.21.4 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 indicate problem.

H06 Check the OC device for an IO error message, i.e., IOxx.

APPENDIX B
I/O DEVICE RELATIONSHIPS

Allowable Functions by I/O Device Type

Function	RMD	MT	PT	CR	CP	LP	TY or CRT
Read binary record	X	X	X	X	X	X[4]	X[4]
Read alphanumeric record	X[1]	X	X	X	X	X	X
Read BCD record	X[1]	X	X[2]	X[2]	X[2]	X[4]	X[4]
Read unformatted record	X[1]	X[1]	X	X	X	X[8]	X[4]
Write binary record	X	X	X	X	X	X[5]	X[4]
Write alphanumeric record	X[1]	X	X		X[3]	X[6]	X
Write BCD record	X[1]	X	X[2]			X[7]	X[4]
Write unformatted record	X[1]	X[1]	X				X[4]
Write end of file	X	X	X				X[8]
Rewind unit	X	X	X[3]				X[5]
Skip one record forward	X	X		X			X[6]
Skip one record backward	X	X					X[7]
Perform function zero	X	X					
Perform function one	X	X					
Perform function two	X	X					
Open a file with rewind option		X					
Open a file with leave option							
Close a file with leave option							
Close a file with update option							

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 Errors by I/O Device Type

Code	Description	I/O Device RMD	MT	PT	CR	CP	LP	TY or CRT
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				
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

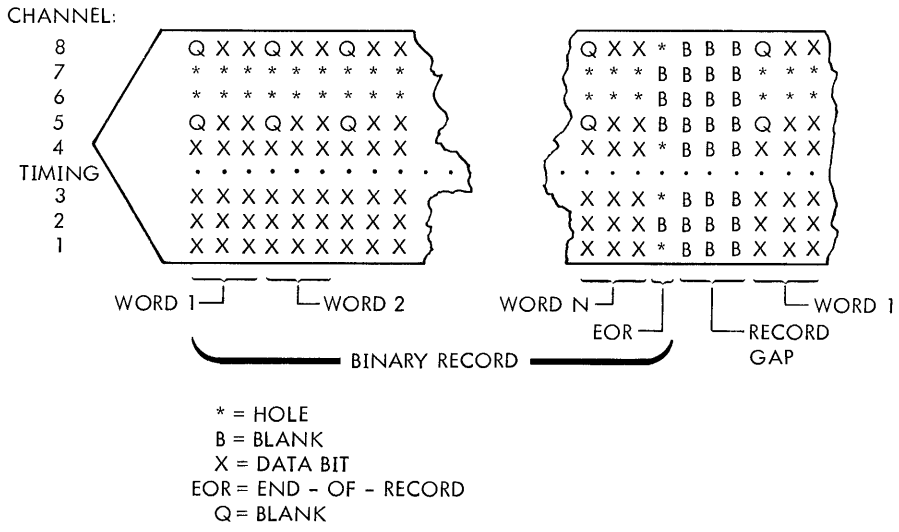
Alphanumeric information is stored with one frame per character (figure C-2). Standard ASCII-8 punch levels are used.

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

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

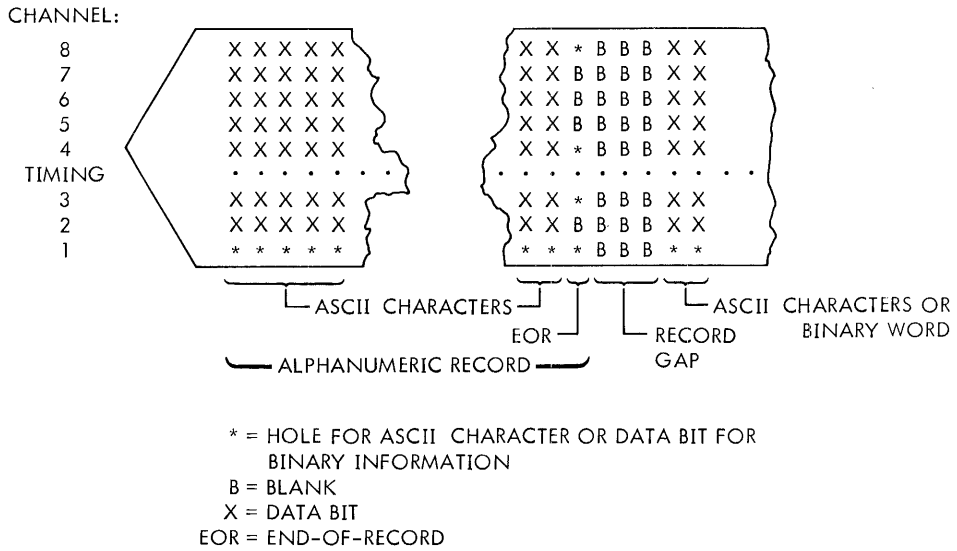
Information stored on cards is binary, alphaumeric, or unformatted. Each card holds one record of information. Hence, there is no end-of-record character for cards.

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).

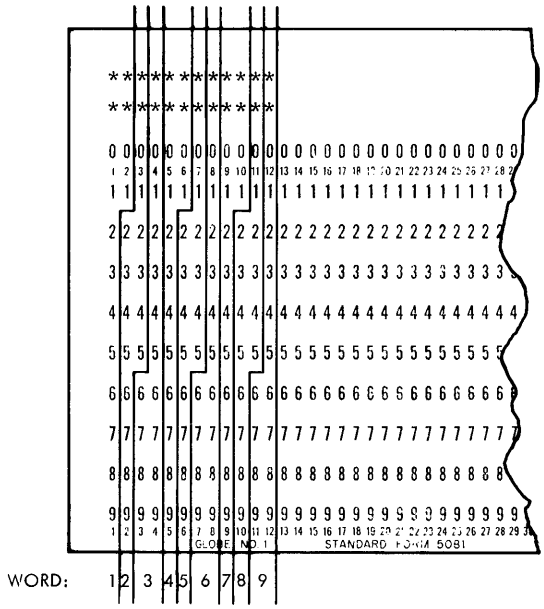
C.2.2 Alphanumberic Mode

Alphanumeric information is stored one character per card column (figure C-4) using the standard punch patterns.



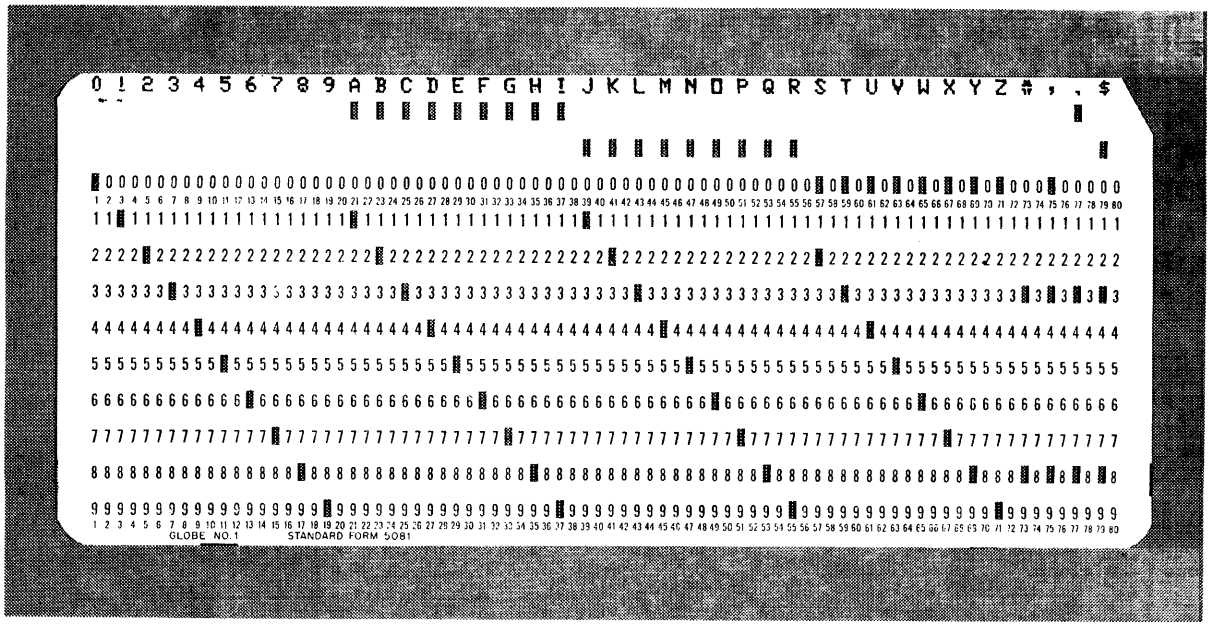
VTII-1375

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)

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 88 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.

APPENDIX D
STANDARD CHARACTER CODES

IBM 026 Punch Symbol	ASCII	Hollerith	IBM 029 Punch ASCII	IBM 029 Punch Symbol
↑	336	7-8	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	12-7-8	336	↑
<	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 Symbol	ASCII	Hollerith	IBM 029 Punch 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
↑	336	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 Protection	045	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 overflow error	030	n/a	
		MP I/O and overflow 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
		RTC overflow	046	n/a	Base timer interval rate is 100 microseconds; free-running clock rate is 100 microseconds
Priority Interrupt Module (PIM)	040-043		0100-0277	n/a	Wired as system priority 5; assignments 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)	020-026 070-073	BIC complete	0100-0277	n/a	All wired as system priority 3 Addresses 070-073 available for BIC5 and BIC6 others created for special use

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 complete	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 complete	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 complete	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	
Card Punch	70-6201	620-27 Card Punch	031	BIC complete	0100-0277	Yes	

Device			Device Address	Interrupt	Interrupt Address	BIC	Comments
Paper-tape System	70-6320	620-55, -55A Paper Tape System	037,034	Character ready	0100-0277	No	
Teletype	70-6100 70-6104	620-6, -7, -8 Teletype	001-007	Read buffer ready Write buffer ready	0100-0277 0100-0277	No	Event 1 = READ Event 2 = WRITE
	70-6400	(E-2250) CRT with E-2184 Controller	---	Read buffer ready Write buffer ready	0100-0277 0100-0277	No	Compatible with Teletype (Event 1 = READ, Event 2 = WRITE)
		Front Panel	---		00-01	No	Wired as system priority 6; not used by VORTEX

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.

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 140715 150314 142640	Identification in ASCII (assume this program was labeled EXAMPLE).
131263 126661 130255 133271	Date of creation in ASCII (assume assembled 03-10-69)
	(End program ID block)
010000 000647 054262 000000	Define entry name SUBR at relative 0 (code 0, subcode 010, pointer 0, name SUBR, and data word 0).
100000 000000	Enter absolute data word 0 in memory at relative 0.
060000 100000 017000	Enter literal (indirectly addressed relative 0) in indirect pointer pool, add address of pointer to load 017000 and enter memory at relative 1.
100000 002000	Enter absolute data word 02000 in memory at relative 2.

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	Enter load request for external name BBEN and chain entry address to relative 011. 000011
•	
•	
•	
•	
•	
	(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        100500      DATA        0500
0501        000500      DATA        0500
 .
 .
 .
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              *   0500
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				