SPERRY RAND

UNIVAC 494

REAL-TIME SYSTEM

OPERATING SYSTEM

PROGRAMMER
REFERENCE

UP-7504 Rev. 2

PRINTED IN U.S.A.

PAGE STATUS SUMMARY

ISSUE: Update Package A to UP-7504 Rev. 2

Section	Page Number	Update Level
Cover/Disclaimer		A
PSS	1	A
Contents	1, 2	A
	3	Orig.
	4 thru 6	A
	7	Orig.
	8	A
	9	Orig.
	10	A
	11	A*
1	1 thru 3	Orig.
2	1 thru 3	Orig.
	4	A
	5 thru 18	Orig.
	19, 20	A
	20a, 20b	A*
	21	A
	22 thru 42	Orig.
	43, 44	A
	44a	A*
	45 thru 61	Orig.
	62	A
	63 thru 80	Orig.
3	1 thru 8	Orig.
	9	A
	10, 11	Orig.
	12 thru 15	A
	16 thru 18	Orig.
	19	A
	20 thru 36	Orig.
	37	A
	38 thru 44	Orig.
	45	A
	46	Orig.
	47	A
	48, 49	Orig.
	50	A
	51, 52	Orig.
	53, 54	A
	55 thru 66	Orig.
	67	A
	68 thru 88	Orig.
4	1 thru 7	Orig.
	8	A
	9 thru 11	Orig.
	12, 13	A
	14 thru 19	Orig.
	20	A
	20a	A*
	21	Orig.
	22 thru 24	A
	24a	A*
4 (cont.)	25 thru 33	Orig.
	34	A
	35 thru 39	Orig.
	40	A
	41	Orig.
	42	A
	43 thru 53	Orig.
5	1 thru 5	Orig.
6	1 thru 21	Orig.
7	1, 2	Orig.
8	1 thru 16	Orig.
	17	A
	18 thru 29	Orig.
9	1 thru 9	Orig.
	10	A
	10a	A*
	11 thru 17	Orig.
	18	A
	19 thru 61	Orig.
10	1 thru 5	Orig.
Appendix A	1 thru 33	Orig.
Appendix B	1 thru 4	A*
User Comment Sheet		

*Original at update level indicated.

CONTENTS

PAGE STATUS SUMMARY

CONTENTS

1. INTRODUCTION

1.1. GENERAL 1-1

1.2. BASIC FEATURES 1-1
1.2.1. Real Time/Online Processing 1-2
1.2.2. Batch Processing 1-2
1.2.3. Multiprogramming Operation 1-2
1.2.4. Program Development 1-2
1.2.5. Automatic Operation 1-3
1.2.6. Integrity 1-3
1.2.7. Modularity 1-3

2. SYSTEM CONTROL

2.1. GENERAL 2-1
2.1.1. Job Control 2-1
2.1.2. Task Control 2-1

2.2. THE EXECUTIVE CONTROL LANGUAGE 2-2
2.2.1. Control Statement Format 2-3
2.2.2. Control Statement Types 2-3

2.3. ORGANIZATIONAL CONTROL STATEMENTS 2-5
2.3.1. The JOB Statement 2-5
2.3.2. The START Statement 2-7
2.3.3. The Correction (COR) Statement 2-8
2.3.4. The Parameter (PRAM) Statement 2-11
2.3.5. The LOG Statement 2-13
2.3.6. The DUMP Statement 2-15
2.3.7. The END Statement 2-18
2.3.8. The FIN Statement 2-18
2.3.9. The READY Statement 2-18
2.3.10. The CALL Statement 2-19
2.3.11. The HDG Statement 2-20 ←
2.3.12. The LST Statement 2-20

→

2.4. TASK ACTIVATION STATEMENTS 2-20a
2.4.1. The GO Statement 2-20a
2.4.2. Systems Processor Control Statements 2-21

2.5. SERVICE CONTROL 2-21
2.5.1. Summary of OMEGA Service Requests 2-22
2.5.2. Internal Control Statements 2-24
2.5.2.1. METHOD OF SUBMISSION 2-24
2.5.3. Segment Call (LOAD$, LOADA$) Service Requests 2-26
2.5.4. Subroutine Load (FETCH$) Service Request 2-27
2.5.5. Locate Table of Contents (FETCHL$) Service Request 2-28
2.5.6. Common Subroutines 2-30
2.5.6.1. OPERATING ENVIRONMENT 2-30
2.5.6.2. ENTRANCE CONDITIONS 2-31
2.5.6.3. SPECIAL DATA TRANSFER (CMOVE$) 2-31
2.5.6.4. EXIT PROCEDURES (CEXIT$, CSWCT$, CABORT$) 2-32
2.5.6.5. NAME CONVENTIONS 2-33
2.5.6.6. COMMON SUBROUTINE REQUESTS 2-33
2.5.7. Send/Receive 2-33
2.5.7.1. THE (SEND$) SERVICE REQUEST 2-34
2.5.7.2. THE (RECEIVE$) SERVICE REQUEST 2-35
2.5.8. DATE and TIME Operations 2-36
2.5.8.1. THE DELAY (DELAY$) SERVICE REQUEST 2-36
2.5.8.2. TIME OF DAY (TIMED$) SERVICE REQUEST 2-37
2.5.8.3. ELAPSED CENTRAL PROCESSOR TIME (TIMEL$) SERVICE REQUEST 2-37
2.5.8.4. DATE AND TIME (DATIM$) SERVICE REQUEST 2-38
2.5.8.5. DATE AND QUANTUM TIME (TIMEQ$) SERVICE REQUEST 2-38
2.5.8.6. DATE AND TIME OF YEAR (TIMEY$) SERVICE REQUEST 2-39
2.5.9. Logical Switches 2-39
2.5.9.1. SET OFF (XOFF$) REQUEST 2-39
2.5.9.2. SET ON (XON$) REQUEST 2-40
2.5.9.3. TEST SWITCHES (XTEST$) REQUEST 2-40
2.5.10. Checkpoint/Restart Mechanism 2-40
2.5.10.1. CHECKPOINT (CKPT$) SERVICE REQUEST 2-41
2.5.10.2. CHECKPOINT PARAMETER TABLE 2-41
2.5.10.3. CHECKPOINT STATUS CODES 2-42
2.5.10.4. DIAGNOSTIC MESSAGES 2-43
2.5.10.5. RESTART ROUTINE 2-43
2.5.10.6. RESTART STATUS MESSAGES 2-44
2.5.10.7. TAPE MOUNTING DIRECTIVES 2-44
2.5.10.8. TAPE CHANGING DIRECTIVE 2-44

→

2.6. ACTIVITY CONTROL 2-44a
2.6.1. Standard Activity Registration (REG$) Service Request 2-45
2.6.2. Queue Process Activity Registration and Activation 2-48
2.6.2.1. QUEUE PROCESS ACTIVITY REGISTRATION (REGQ$) SERVICE REQUEST 2-48
2.6.2.2. ACTIVATE QUEUE PROCESS ACTIVITY REGISTRATION (QREF$) SERVICE REQUEST 2-50
2.6.3. Fork and Join Mechanism 2-54
2.6.3.1. FORK (FORK$) SERVICE REQUEST 2-54
2.6.3.2. JOIN (JOIN$) SERVICE REQUEST 2-55
2.6.4. Termination Service Requests 2-57
2.6.4.1. THE RETURN (RETURN$) SERVICE REQUEST 2-57
2.6.4.2. THE ABORT (ABORT$) SERVICE REQUEST 2-58
2.6.4.3. THE ERROR (ERROR$) SERVICE REQUEST 2-58
2.6.4.4. THE RETURN 1 (RETURN1$) SERVICE REQUEST 2-58

2.7. CENTRAL PROCESSOR CONTROL 2-58
2.7.1. The Dispatcher and the Central Processor Queue 2-59
2.7.2. Operating Priority 2-60

2.8. PRIMARY STORAGE ALLOCATION 2-61
2.8.1. Task Primary Storage Allocation 2-61
2.8.2. Task Primary Storage Extensions 2-62
2.8.2.1. EXTERNAL PRIMARY STORAGE EXTENSIONS (CORE STATEMENT) 2-62
2.8.2.2. INTERNAL PRIMARY STORAGE EXTENSIONS (MADD$) 2-63
2.8.2.3. INTERNAL CORE RELEASE (MREL) 2-63
2.8.2.4. TEST PRIMARY STORAGE AREA/LIMITS (TCORE) 2-64
2.8.3. Worker Program Considerations for Primary Storage Compaction 2-66
2.8.3.1. INHIBITING COMPACTION (TCORE1$) 2-66
2.8.3.2. PACKET GENERATOR 2-66
2.8.4. Register Index Modes 2-69
2.8.4.1. SET 15-BIT B REGISTERS (SET15$) 2-70
2.8.4.2. SET 17-BIT B REGISTERS (SET17$) 2-70
2.8.4.3. SET DUAL INDEX MODE (SETD$) 2-70
2.8.4.4. CLEAR DUAL INDEX MODE (CLD$) 2-71
2.8.5. Program Considerations 2-71
2.8.5.1. THE RELATIVE INDEX REGISTER (RIR) 2-71
2.8.5.2. THE PROGRAM LOCK-IN REGISTER (PLR) 2-72
2.8.5.3. PRIMARY STORAGE ADDRESSING 2-73

2.9. PROGRAM CONTINGENCIES 2-75
2.9.1. Contingency Types and Interrupts 2-76
2.9.2. Error Routine Specification (ERRADD$, FOFADD$, FUFADD$) Service Requests 2-76
2.9.3. Error Routine Operation 2-77
2.9.3.1. ENTRY 2-77
2.9.3.2. EXIT (RETURN$, RETURN1$, ERROR$, ABORT$) 2-78
2.9.4. Default Procedures 2-79
2.9.4.1. FAULT CONTINGENCIES 2-79
2.9.4.2. FLOATING POINT (TESTFOF$, TESTFUF$) 2-79

3. DATA MANAGEMENT SYSTEM

3.1. GENERAL 3-1
3.1.1. Assignment 3-1
3.1.2. Duration of Assignments 3-2
3.1.3. Data Access Methods 3-2
3.1.4. Maintenance Functions 3-3

3.2. ASSIGNMENT (ASG) STATEMENTS 3-3
3.2.1. The Assignment (ASG) Statement 3-3
3.2.2. Peripheral Name 3-4
3.2.3. File Code 3-6
3.2.4. Direct Access Storage Assignment 3-7
3.2.4.1. FEATURES 3-8
3.2.4.2. DIRECT ACCESS STORAGE ASSIGNMENT STATEMENT AND MNEMONICS 3-9
3.2.4.3. METHOD OF SUBMISSION 3-13

3.2.5. UNISERVO Tape Assignment 3-13
3.2.5.1. METHOD OF SUBMISSION 3-15
3.2.5.2. BLOCK NUMBERING 3-16
3.2.5.3. NOISE CONSTANT 3-17
3.2.5.4. TRANSLATE FEATURE 3-17
3.2.5.5. 9-TRACK FORMAT FEATURE 3-17
3.2.6. Card Device Assignment 3-19
3.2.6.1. METHOD OF SUBMISSION 3-20
3.2.7. Printer Assignment 3-20
3.2.7.1. METHOD OF SUBMISSION 3-21
3.2.8. Paper Tape Device Assignment 3-21
3.2.8.1. METHOD OF SUBMISSION 3-22
3.2.9. The FREE Statement 3-23
3.2.9.1. METHOD OF SUBMISSION 3-23
3.2.10. The SWITCH Statement 3-24
3.2.10.1. METHOD OF SUBMISSION 3-25
3.2.10.2. EXAMPLES 3-25
3.2.11. Service Requests 3-25
3.2.11.1. TEST FILE CODE (TFC$) 3-25
3.2.11.2. PARTIAL FILE RELEASE (FREL$ AND FRELA$) 3-27
3.2.11.3. FILE EXTENSION (FADD$ AND FADDA$) 3-28
3.2.11.4. USER DECLARATION OF "BAD" FILE AREA (FRPL$ AND FRPLI$) 3-29
3.2.11.5. EXAMPLES 3-31

3.3. DEVICE CONTROL 3-34
3.3.1. Data Format Considerations 3-34
3.3.2. File Codes 3-34
3.3.3. Device Control Macros 3-35
3.3.4. Status Codes 3-37
3.3.5. Drum Macros 3-40
3.3.5.1. READ$ 3-40
3.3.5.2. WRITE$ 3-40
3.3.5.3. SEARCH$ 3-41
3.3.5.4. BLOCK READ (BLOCKR$) 3-41
3.3.5.5. BLOCK SEARCH (BLOCKS$) 3-41
3.3.5.6. SCATTER READ (SREAD$) 3-42
3.3.5.7. GATHER WRITE (GWRITE$) 3-43
3.3.5.8. MULTIPLE READ (MREAD$) 3-44
3.3.5.9. MULTIPLE WRITE (MWRITE$) 3-44
3.3.5.10. READ LOCK (READL$) 3-45
3.3.5.11. WRITE RELEASE (WRITER$) 3-45
3.3.5.12. SEARCH LOCK (SEARCHL$) 3-46
3.3.5.13. LOCK READ (LOCKR$) 3-46
3.3.5.14. WRITE RELEASE (WRELS$) 3-47
3.3.5.15. MULTIPLE READ LOCK (MREADL$) 3-47
3.3.5.16. MULTIPLE WRITE RELEASE (MWRITER$) 3-48
3.3.5.17. LOCK DEFINITION (LOCKA$) 3-49
3.3.6. Direct Access Storage Image Filing 3-50
3.3.6.1. IMAGE FILE MACROS (LRASL$, RDTS$, AND WRTS$) 3-50
3.3.6.2. EXAMPLES OF IMAGE FILE CONTROL 3-54
3.3.7. Magnetic Tape Macros 3-57
3.3.7.1. READ$ 3-58
3.3.7.2. WRITE$ 3-59
3.3.7.3. SEARCH$ 3-59

3.3.7.4. SCATTER READ (SREAD$) 3-60
3.3.7.5. GATHER WRITE (GWRITE$) 3-62
3.3.7.6. MULTIPLE READ (MREAD$) 3-63
3.3.7.7. READ BACK (READB$) 3-64
3.3.7.8. REWIND$ 3-66
3.3.7.9. REWIND WITH INTERLOCK (REWINDI$) 3-66
3.3.7.10. WRITE END-OF-FILE (WRTEOF$) 3-67 ←
3.3.7.11. ERASE (ERASE$) 3-67
3.3.8. Unit Record Macros 3-68
3.3.8.1. CARD OPERATIONS 3-68
3.3.8.2. PRINTER OPERATION 3-70
3.3.8.3. PAPER TAPE OPERATIONS 3-72

3.4. COOPERATIVE CONTROL 3-76
3.4.1. Input Unit Record Routine 3-76
3.4.2. Input/Output (I/O) Cooperative Control 3-76
3.4.3. Output Unit Record Routines 3-78
3.4.4. Service Requests 3-78
3.4.4.1. PRIMARY INPUT 3-78
3.4.4.2. PRIMARY OUTPUT 3-79
3.4.4.3. SECONDARY OUTPUT 3-80
3.4.5. Method of Operation 3-80
3.4.6. Error Diagnostics 3-81
3.4.6.1. PRIMARY INPUT 3-81
3.4.6.2. PRIMARY OUTPUT OVERFLOW 3-82
3.4.6.3. SECONDARY OUTPUT OVERFLOW 3-82

3.5. MASTER FILE DIRECTORY (MFD) 3-82
3.5.1. The MFD Statement 3-82
3.5.2. Examples 3-85
3.5.3. Error Diagnostics and Messages 3-86

4. LANGUAGE PROCESSOR CONTROL

4.1. GENERAL 4-1

4.2. METHOD OF OPERATION 4-1
4.2.1. Preparation of Input 4-2
4.2.2. General Functions 4-2
4.2.3. Loader 4-2
4.2.4. Program Library Editor 4-2

4.3. PROGRAM LIBRARIES 4-3
4.3.1. Element Types 4-3
4.3.1.1. SOURCE ELEMENT 4-3
4.3.1.2. RELOCATABLE (RB) ELEMENT 4-4
4.3.1.3. LOAD ELEMENT 4-4
4.3.2. Libraries 4-4
4.3.2.1. JOB LIBRARY 4-4
4.3.2.2. GROUP LIBRARY 4-5
4.3.2.3. SYSTEM LIBRARY 4-5

4.4. LANGUAGE PROCESSOR CONTROL STATEMENTS 4-5
4.4.1. Statement Format 4-6
4.4.2. Language Processor Facilities 4-7
4.4.3. SPURT Statement 4-7
4.4.4. ASM Statement 4-9
4.4.5. FOR Statement 4-11
4.4.6. COB Statement 4-13

4.5. SOURCE CODE FORMAT 4-14
4.5.1. SOURCE Routine – Primary Function 4-15
4.5.2. SOURCE Statement 4-15
4.5.3. Error Messages 4-18
4.5.4. Source Language Correction 4-19
4.5.4.1. CORRECTION STATEMENTS 4-19
4.5.4.2. DELETIONS 4-20
4.5.4.3. INSERTIONS 4-20
4.5.5. Examples. 4-20

4.6. LOADER 4-24
4.6.1. Control Statement Format 4-24a
4.6.2. The LOAD Statement 4-24a
4.6.3. Secondary Control Statements 4-26
4.6.3.1. SEGMENT STATEMENT 4-27
4.6.3.2. INCLUDE STATEMENT 4-29
4.6.3.3. MAP STATEMENT 4-29
4.6.3.4. EXCLUDE STATEMENT 4-30
4.6.3.5. ENTRY STATEMENT 4-30
4.6.3.6. EQUALS STATEMENT 4-30
4.6.3.7. MISCELLANEOUS 4-31
4.6.4. Loader Requirements 4-32
4.6.5. MAP Elements 4-32
4.6.6. Error Diagnostics 4-33
4.6.7. Element Selection and Placement Algorithm 4-35
4.6.8. Loader List Option 4-38

4.7. PROGRAM LIBRARY EDITOR 4-39
4.7.1. IN Statement 4-39
4.7.2. OUT Statement 4-41
4.7.3. PRT Statement 4-44
4.7.4. DEL Statement 4-45
4.7.5. LINK Statement 4-46
4.7.6. Error Messages 4-48

4.8. INTERLANGUAGE COMPATIBILITY 4-49
4.8.1. Common Area 4-50
4.8.2. Data Types 4-50
4.8.3. Transfer of Data 4-51
4.8.4. Interlanguage Transfer Techniques 4-52

5. OPERATOR COMMUNICATIONS

5.1. GENERAL 5-1

5.2. THE MSG STATEMENT 5-1

5.3. MAGNETIC TAPE STATEMENTS 5-2
5.3.1. MOUNT$ 5-2
5.3.2. DEMOUNT$ 5-3
5.3.3. CHANGE$ 5-4

5.4. UNSOLICITED TASK INPUT MESSAGES 5-4

6. TEST SYSTEM

6.1. GENERAL 6-1

6.2. CONDITIONAL PROCEDURES 6-1

6.3. TEST SYSTEM CALL 6-2

6.4. SECONDARY CONTROL LANGUAGE 6-2
6.4.1. The AT Statement 6-3
6.4.2. The IF Statement 6-3
6.4.3. The DUMP Statement 6-5
6.4.4. The TRACE Statement 6-6
6.4.5. The TRAP Statement 6-6
6.4.6. The EXIT Statement 6-7
6.4.7. The SET Statement 6-7
6.4.8. The END Statement 6-8
6.4.9. Examples of Secondary Control Language Statements 6-9

6.5. LOGICAL SWITCHES 6-14

6.6. DIAGNOSTIC MESSAGES FROM TEST SYSTEM 6-16

6.7. REQUIREMENTS AND METHOD OF OPERATION 6-17
6.7.1. Requirements 6-17
6.7.2. Service Requests 6-17
6.7.3. Segment Control 6-17

6.8. SAMPLE PROGRAM 6-18

7. SYSTEM GENERATION

7.1. SYSTEMS GENERATION AND INITIALIZATION 7-1

7.2. SYSTEMS GENERATION 7-1

7.3. SYSTEMS INITIALIZATION 7-2

8. UTILITY SERVICE ROUTINES

8.1. GENERAL 8-1

8.2. THE REXECUTOR 8-1
8.2.1. Restrictions 8-1

8.2.2. Control Cards 8-3
8.2.3. The MEANS Statement 8-4
8.2.4. The PROG Statement 8-6
8.2.5. A Card 8-7
8.2.6. B and C Cards 8-7
8.2.6.1. B CARD FORMAT 8-7
8.2.6.2. C CARD FORMAT 8-8
8.2.7. The PS Statement 8-8
8.2.8. Control Stream Examples 8-9

8.3. THE UTILITY PROCESSOR 8-14
8.3.1. The Utility (UTL) Control Statement 8-15
8.3.2. Secondary Control Statements 8-15
8.3.2.1. GENERAL SPECIFICATIONS AND OPTIONS 8-16
→ 8.3.2.2. ACTION STATEMENTS (READ, BLKRD, SEARCH, WRITE, REWIND, WRTEOF, MOVE, GO TO, DO) 8-17
8.3.2.3. TEST STATEMENTS 8-21
8.3.2.4. USER OWN CODE ROUTINES 8-22
8.3.3. Applicable OMEGA Control Statements 8-25
8.3.4. Data Conventions 8-25
8.3.5. Utility Processor Examples 8-26
8.3.6. Error Messages 8-28

9. REMOTE DEVICE CONTROL

9.1. GENERAL 9-1
9.1.1. Device Control Elements 9-1
9.1.2. Levels of Interface 9-3
9.1.2.1. LEVEL 1 INTERFACE 9-4
9.1.2.2. LEVEL 2 INTERFACE 9-4
9.1.3. Summary of Service Requests and Control Statements 9-6

9.2. REMOTE FACILITY ASSIGNMENT 9-7
9.2.1. The LASG Statement 9-8
9.2.2. The LACQ Service Request 9-12
9.2.3. The LFREE Statement 9-14
9.2.4. Miscellaneous Remote Facility Assignment Requests 9-16
9.2.4.1. THE CTMFREE SERVICE REQUEST 9-16
9.2.4.2. REMOTE FACILITY UPDATE 9-16

9.3. REMOTE DATA ACCESS SERVICE REQUESTS 9-18
9.3.1. Message Queue Control 9-18
9.3.2. Read/Write Service Requests 9-19
9.3.2.1. READ MESSAGE (READM$, READMW$ AND LOOKM$) 9-19
9.3.2.2. WRITE MESSAGE (WRITEM$) 9-23
9.3.3. Get/Put Service Requests 9-24
9.3.3.1. GET MESSAGE (GETM$ AND GETMW$) 9-24
9.3.3.2. PUT MESSAGE (PUTM$) 9-26
9.3.3.3. PUT MESSAGE AFTER PURGE (PUTMP$) 9-27
9.3.4. Message Buffer Format 9-28
9.3.5. Own Code Routines 9-29

9.4. REMOTE SYSTEMS 9-30
9.4.1. The Remote UNIVAC 1004 and UNIVAC 9200/9300 Systems 9-30
9.4.2. The Remote DCT 2000 System 9-32
9.4.3. The Remote UNISCOPE 300 System 9-34
9.4.4. Remote UNISCOPE 100/DCT 1000 System 9-40
9.4.4.1. INPUT MESSAGE BUFFER 9-41
9.4.4.2. OUTPUT MESSAGE BUFFER 9-44
9.4.4.3. UNISCOPE 100 OUTPUT EDITING FUNCTIONS 9-45
9.4.4.4. DCT 1000 OUTPUT EDITING FUNCTIONS 9-50
9.4.5. Remote DCT 500 and Teletypewriter System 9-51
9.4.5.1. INPUT/OUTPUT MESSAGE BUFFER 9-51
9.4.5.2. CONSTRAINTS 9-52

9.5. REMOTE SCHEDULER 9-52
9.5.1. Operation of Remote Scheduler 9-53
9.5.2. SCHEDULE Entry 9-55
9.5.3. TIME Entry 9-56
9.5.4. PHONE Entry 9-56
9.5.5. STOP Entry 9-57
9.5.6. HOLD Entry 9-58
9.5.7. TERM Entry 9-58
9.5.8. Error Conditions 9-59
9.5.8.1. ERROR CODES 9-59
9.5.8.2. CONNECTION MESSAGE 9-60
9.5.9. Examples 9-60

10. BASIC OPERATIONAL DESCRIPTION

10.1. JOB SUBMISSION 10-1

10.2. JOB SCHEDULING 10-1
10.2.1. Preselection 10-2
10.2.2. Control 10-2
10.2.3. Selection 10-3

10.3. ALLOCATION AND MAINTENANCE OF PRIMARY STORAGE 10-3

APPENDIXES

A. ELEMENT FORMAT AND DESCRIPTION

A.1. RELATIVE BINARY (RB) ELEMENT (ELEMENT TYPE 1) A-1
A.1.1. Element Organization A-2
A.1.1.1. TOC (TABLE OF CONTENTS) A-3
A.1.1.2. PREAMBLE A-5
A.1.1.2.1. Entry Definition (EDEF) A-5
A.1.1.2.2. External Reference (XREF) A-6
A.1.1.2.3. Control Counter (CC) A-6
A.1.1.2.4. Common Area A-6
A.1.1.2.5. Symbol Definition (SDEF) A-7
A.1.1.2.6. Preamble Length A-7
A.1.1.3. TEXT A-9

A.1.2. Tape/Drum Output Format A-10
A.1.2.1. HEADER BLOCK A-10
A.1.2.2. DATA BLOCK A-13
A.1.2.3. END SENTINEL BLOCKS A-16
A.1.3. Card Output Format A-16
A.1.4. Instruction Modification A-17

A.2. LOAD ELEMENT (ELEMENT TYPE 2) A-19
A.2.1. Element Organization A-20
A.2.1.1. TOC (TABLE OF CONTENTS) A-20
A.2.1.2. CONTROL STATEMENTS A-22
A.2.1.3. INSTRUCTIONS A-23
A.2.1.4. MODIFICATION-M OPTION A-25
A.2.1.5. SYMBOL DEFINITIONS (SDEF'S) A-26
A.2.2. Tape Output Format A-26
A.2.2.1. DATA BLOCK A-27
A.2.3. Card Output Format A-27
A.2.4. Sample Listings of Load Element Output A-27

A.3. SOURCE ELEMENT (ELEMENT TYPE 3) A-28
A.3.1. Element Organization A-30
A.3.1.1. TABLE OF CONTENTS (TOC) A-30
A.3.1.2. TEXT A-31
A.3.2. Tape Output Format A-31
A.3.2.1. DATA BLOCK A-31
A.3.3. Card Output Format A-32

B. TAPE LABELLING

B.1. GENERAL B–1
B.1.1. Tape Labelling and Facility Assignment B–2
B.1.2. Tape Labelling and the IN Processor B–2
B.1.3. Tape Labelling and the OUT Processor B–3
B.1.4. Tape Labelling and the LINK Processor B–3

B.2. OPERATOR COMMUNICATION B-3

B.3. EXAMPLE JOB STREAM B-5

B.4. ERROR MESSAGES B-6

USER COMMENT SHEET

FIGURES

2–1. Task Control 2-2

3–1. Cooperative Mechanism 3-77

4–1. Data Word Types 4-51

9–1. OMEGA Remote Device Control Elements 9-3
9–2. Direct Access Queue 9-19
9–3. Direction of Message Flow 9-20

A–1. Sample Source Code for Relative Binary Text A-18
A–2. Sample Relative Binary Text – 16_8-Word Item A-19
A–3. Sample Listing – Load Element A-28

TABLES

2–1. General Service Requests 2-22

3–1. Block Sizes 3-11
3–2. Standard Magnetic Tape Character Conversion 3-18
3–3. Summary of Device Level Input/Output Macros 3-39
3–4. Printer Codes 3-74
3–5. Card Conversion 3-75

9–1. UNISCOPE 100 Output Editing Functions 9-45
9–2. DCT 1000 Output Editing Functions 9-50

I. INTRODUCTION

1.1. GENERAL

The UNIVAC 494 Operating System has been designed and implemented to establish and function within the efficient multiprogramming environment needed for utilizing the full capabilities of the UNIVAC 494 Real-Time System. The Operating System comprises the entire system software programs and routines, including the executive routine, OMEGA, and the system utility processor, language processors, applications processors, and libraries. To take maximum advantage of the capabilities of this advanced system and to make effective use of a given configuration, a complex internal operating environment, with a master control, must be created. The environment must allow for the concurrent operation of many programs, allow the system to react immediately to the inquiries, requests, and demands of many different users at local and remote stations, allow for the demands of real time application, be able to store file, retrieve, and protect large blocks of data, and it must make optimum use of all available facilities, while minimizing job turnaround time.

Only through central control of all activities can the system be fully established and maintained to satisfy the requirements of all applications. The responsibility for efficient, flexible, centralized control is borne by OMEGA. By presenting a relatively simple interface to the programmer, OMEGA allows him to use the system with relative ease, while relieving him of concern for the internal interaction between his program and other coexistent programs.

The capabilities of OMEGA span a broad spectrum of data processing activities. The design of OMEGA is such that no penalties of inefficiency are imposed upon any one of these activities by the support provided for the other activities. An installation which is not interested in utilization of specific capabilities may eliminate them at system generation time.

Emphasis has been placed upon ease of use by the programmer or user of the system. Operator intervention, decision requirements, and work to be performed by the system are described on control cards to minimize job turnaround time. The user may construct any logical combination of programs for a particular job by inserting the proper control cards in his job deck.

Job decks may be collected and entered into the system from many sources, either remote or at the central site. Once the job deck is entered into the system, OMEGA controls the loading, allocation, and execution of the described programs. Jobs which cannot be completed due to program error are automatically deallocated and purged from the system with appropriate diagnostic information.

1.2. BASIC FEATURES

A summary of the major or outstanding components of OMEGA is given in the following paragraphs. Particular functional capabilities and utilization procedures are presented in subsequent sections.

1.2.1. Real Time/Online Processing

The most critical requirement of the system is the ability to respond efficiently to the demands of real time processing, and to give preference to the operational needs of a real time program. Executive services appropriate to the construction and execution of real time programs are provided. These services allow a real time program to exercise critical control over system service. The contingencies of real time are supported by nonstop operation with procedures such as roll-out of conflicting user programs. Roll-out involves the suspension, intermediate storage, and possible relocation of a program prior to re-initiation.

1.2.2. Batch Processing

Design emphasis has been placed upon facilitating job preparation and submission with minimization of job turnaround time. A priority specification provides preferential service for batch runs submitted by remote operation or where turnaround time is critical.

All jobs entering the system are described by a control language. The user has the ability to specify preferred service for certain jobs, but has no responsibility in planning schedules to achieve machine optimization. Job descriptions are accepted from any specified source and may be preregistered as a convenience in referencing standard jobs. The system provides for automatic job-to-job transition, communication within and between jobs, and associated services such as logging and accounting.

1.2.3. Multiprogramming Operation

Computer utilization is maximized by the technique of multiprogramming. OMEGA schedules and executes a combination of independent, real time, and batch jobs in a concurrent mode, and balances utilization of system facilities.

Multiprogramming involves the cyclic execution of independent jobs ordered by input/output utilization and response requirements. Multiprogramming is effected by queuing and servicing requests for time-shared elements, and rotating control between competing programs. The system elements which provide these services use other elements of the system to perform their functions.

1.2.4. Program Development

To enhance the modularity of the Operating System, a set of standard routines is provided which interfaces the language processors with OMEGA. Standardization provides the following features not otherwise obtainable:

- The ability to form programs as a combination of elements produced by various language processors such as COBOL, FORTRAN, etc.

- The ability to incorporate additional systems or language processors into the operating system with little or no change to system components.

- The elimination of redundant elements used in the control of individual processors; thereby simplifying usage and standardizing the individual compiler or assembler control elements.

The system provides standardization of common functions by eliminating duplication of these functions in separate user programs and by establishing a common program and operator interface. Standardization contributes to installation efficiency by accommodating changes in machine configuration and/or operating procedures without direct impact on user programs. Changes in one user program which have impact on other user programs are similarly minimized.

A test system provides the user with complete control over programs in the debugging process and allows run time information extraction and display. The test system provides an object time source-level debugging mechanism common to all programs and eliminates the need for source time planning of debugging strategy. This system significantly reduces the time and expense associated with program checkout.

1.2.5. Automatic Operation

System operation is defined through control languages which provide efficient and flexible user direction. The control language is a format description of the functions preparatory to execution of a program. Operator participation is explicitly defined and minimized as far as possible.

Utilization of direct access storage is the primary method of eliminating the delays and errors inherent in operator intervention and for increasing overall system efficiency. Direct access storage is used as a system buffer for the job backlog accepted from system input devices and the resultant images for system output devices shared among the executed jobs. This buffering allows operation of the system independent of these essential low speed peripheral devices. All executable programs are obtained from direct access storage. To facilitate automatic operation, temporary intermediate files required in operation of a program are generally assigned to direct access storage instead of tape storage.

A catalog of direct access files (the Master File Directory) which transcend a particular job is maintained by the system. The Master File Directory facilitates automatic operation and provides permanent storage to a collection of individual and independent users.

1.2.6. Integrity

Complete system integrity is effected through memory lockout and guard mode and system validation of service requests. An errant program is unable to destroy either the system or other programs in the multiprogram environment. Comprehensive contingency procedures facilitate recovery from error conditions, and provide restart from unrecoverable conditions.

System control is enforced at the central site by a System Controller through options exercised at system generation time or during operation through the operator console. Programmers, users, and engineers are essentially prevented from exercising basic options in influencing operation of the total system.

1.2.7. Modularity

OMEGA is explicitly modular in design to facilitate future extensions, expansions of particular functions, or selection of available variants of a basic function. Modularity can be exercised during generation of a system so that each user can create versions of the system which will operate more efficiently for his system needs and configuration.

2. SYSTEM CONTROL

2.1. GENERAL

System Control consists of those elements of OMEGA which are concerned with the introduction, allocation, and control of work to be performed by the system.

2.1.1. Job Control

The job control level provides a formal means for describing work to be performed by the system. Through use of the executive control language, the user forms a job deck which describes one or more tasks to be selected and executed sequentially to accomplish an explicit goal. As an example, a programmer desiring to assemble and test an element of a specific program would have a job deck composed of the following tasks:

- Assemble the element in question.
- Ready the program for execution.
- Execute the program in a test mode.
- Perform post-run processing.

Each job deck is recognized and introduced into a job stack via the primary input cooperatives on a first in, first out basis subject to the priorities allowed for standard production, rush, remotely originated jobs, and available facilities. The job stack is processed by a selection routine that determines which job will be next introduced into the multiprogram environment and which initiates preparatory functions for the execution of the first task. Preparatory functions include allocation of necessary input/output peripheral units and operator intervention such as tape mounting. Subsequent to operator setup, the required primary storage is allocated and the first task is initiated.

When an active task completes processing, control is returned through termination to selection. As each task terminates, the facilities which do not transcend the task are released, and the selection cycle is repeated for the next task described by the job deck. When all tasks of a job are completed, post-job processes are initiated, including an accounting information summary, release of facilities, and operator communications.

2.1.2. Task Control

All system functions are performed to effect the actual execution of a user or system task as grouped within the job deck. Each task is the execution of a specific system or user program and requires executive control statements to describe the task to the system.

A task is activated and controlled through a task addendum which is used by all system elements in performing functions for the task. The operating task may define parts of itself to be executed concurrently in the multiprogram environment. Such a part is called an activity. An activity is controlled by an activity addendum which is linked to the task addendum. Although the activity is in large part independent, the activity shares peripheral allocation, file reference, and other requirements through a link to the task addendum. An activity may be executed synchronously or asynchronously through OMEGA, or may be synchronized by the task itself. Declaration of this concurrency is an essential aspect of most commerical real time control programs. It is this type of definition which attains the advantages of multiprogramming within a task on a processor.

Figure 2–1 illustrates the basic relationship between the job deck and selection/execution of a task with subsequent registration of activities.

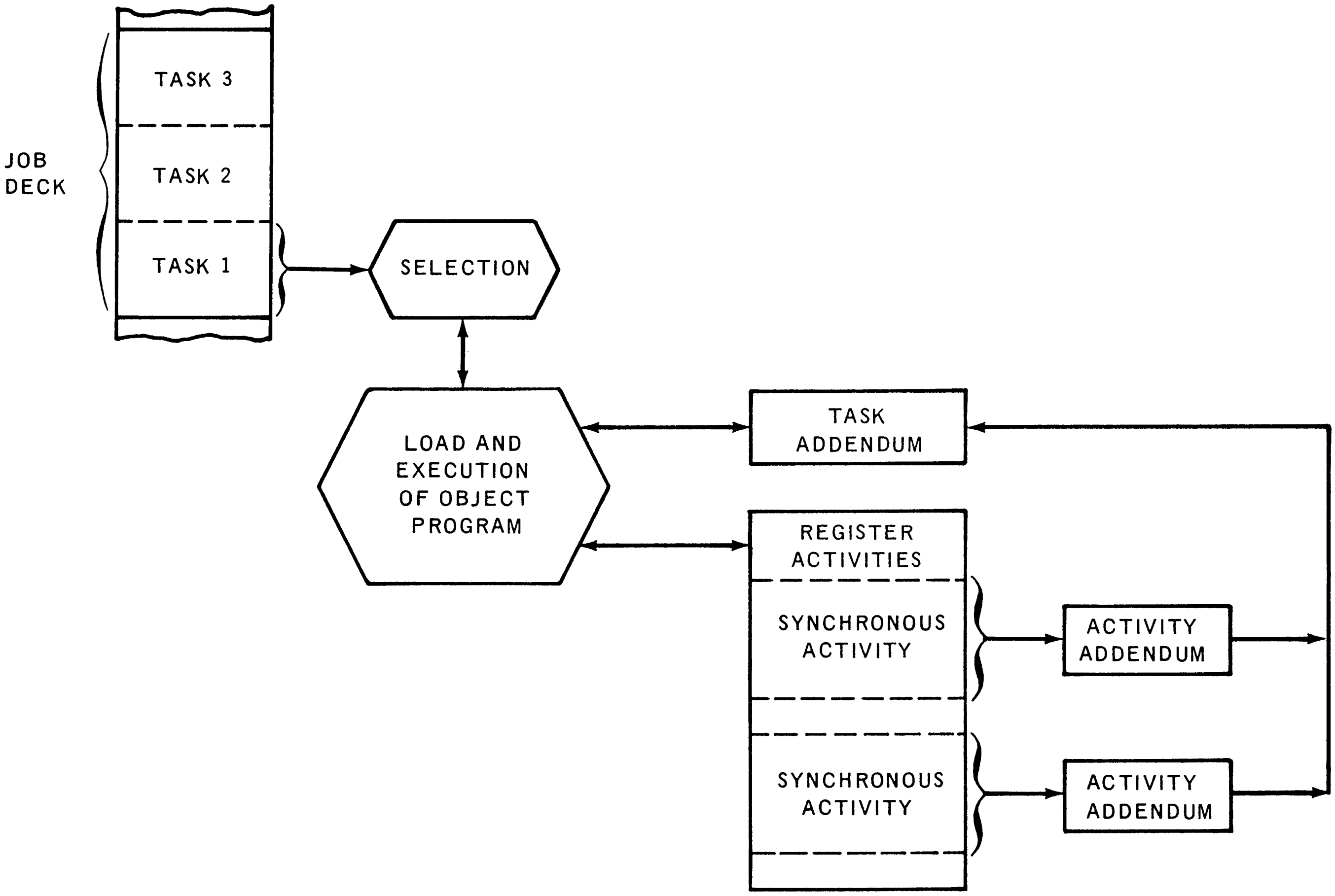


Figure 2–1. Task Control

2.2. THE EXECUTIVE CONTROL LANGUAGE

The Executive Control Language provides the means for a user to direct OMEGA in its execution of the individual tasks of a job and to relay operational information concerning a job to the executive routine. The language is open ended and easily expanded, so that features and functions may be added as the specific needs of different installations dictate.

The language is made up of control statements which are of two types. Primary control statements, which are discussed here, are used for organisational control, input/output control, task activation, and system processor control. Secondary control statements perform various functions for certain routines.

The construction of a job deck is performed by the user and may include supplementary cards representing data, source code, or object code with control cards.

2.2.1. Control Statement Format

The basic format of the control statement is quite simple and is amenable to a large number of input devices. Statements are in card-image format. Each primary control statement consists of a leading character (#) for recognition purposes, a function which categorizes the statement, options as desired, and a variable number of specifications. The secondary statement has the same format except that a blank occurs in column 1. Normally, the end of a statement is signified by the end of a card for card input, or by a carriage return or its equivalent for other types of input. The control statement has the following general form:

#functionƀoption(s)ƀspecification, specification...

The function field must always be present to determine the basic operation. For certain control statements an appendage to the function field is recognized and called the option field. The option field is separated from the function field by a blank (ƀ) and is composed of a string of alphabetic characters, in any sequence. The option field is terminated by one or more blanks (ƀ); if no options are selected, the function field and specifications field must be separated by two or more blanks. Specifications are interpreted by the element named by the function code and are separated by commas. The absence of a specification in an ordered list must be indicated by a comma.

Unless a given specification is explicitly stated as being optional, the specification is required for completion of the function in accordance with the system's design.

The contents and number of specifications are dependent upon the named element. Specification fields may contain subfields that are separated by a slash (/).

All numeric fields may be specified as decimal or octal. Numeric fields specified as decimal must have a D immediately following the low order numeric. Absence of D on numeric quantity fields implies that the expressed value is in octal notation.

For operations requiring the specification of a name/version, the name is from 1 to 10 alphanumeric characters, and the version is from 1 to 5 alphanumeric characters.

2.2.2. Control Statement Types

The primary control statements are divided into four general categories as noted:

- Organization control statements are used to activate and control a job stream. In general, this category of statements describes the job decks to be activated; supplies informational data to the system, the operational personnel, and the program; or in some manner activates a systems process not constituted as a task. These statements are processed upon their occurrence in the job deck. The statements are summarized as follows:

  JOB Delineates an independent ordered sequence of one or more tasks to be executed serially.

  START Schedules the execution of a job stream.

  COR Enters corrections to a collected program.

  SOURCE Introduces supplementary source code, control statements, or data into the job stream.

PRAM Conveys operational parameters to a task.

MSG Conveys instructions to console operator.

LOG Records user-specified literal on systems log.

DUMP Submits diagnostic printouts of direct access storage, primary storage or tape storage to the primary output routine.

DEL Deletes elements from job library.

END Marks the end of the primary input stream for any one task.

FIN Marks the physical end of a control stream from any one device.

READY Identifies a remote terminal requesting service.

CALL Schedules nonstandard output cooperative action.

→ HDG Modifies the heading line on each page of primary output.

- Input/Output Control statements are used to assign and release peripheral devices and direct access storage to a task.

ASG Associates an input/output device or direct access storage unit to a task.

FREE Releases a particular peripheral device for use by other tasks.

SWITCH Changes the file code reference.

MFD Catalogs, assigns, and releases files in connection with the Master File Directory (MFD).

LASG Assigns a communications line to a task.

LFREE Releases a file code and closes output queues on a communications line.

- Task Activation statements are used to call for the selection and activation of a user program. (System Processor Control statements are also Task Activation Control Statements).

GO Calls for loading and execution of absolute program elements produced by the Loader.

- System Processor Control statements call for the loading and execution of a processor contained in the systems library or user job library. Each system processor is selected as a task, and in general, recognizes a secondary control language and/or source code. A detailed description of each is contained in subsequent sections. Processor calls currently available are as follows:

IN,OUT,
PRT,
LINK, } Structure and access internal libraries.

TEST Calls the test system. A secondary control language defines the debugging procedures to be employed.

LOAD Calls for the collection and allocation of an absolute program by the loader from relocatable binary elements produced by the assemblers and/or compilers. A secondary control defines the collection parameters.

UTL Calls the utility system. A secondary control language allows distribution or collection of data as an adjunct to test runs or general utility purposes.

ELM Calls element library maintenance. A secondary control language defines manipulation of external libraries and systems generation procedures.

REX Calls and activates the UNIVAC 490 REXecutor. A secondary control language describes the UNIVAC 490 program to be loaded and executed.

SPURT Calls and activates the UNIVAC 494 SPURT Assembler.

ASM Calls and activates the UNIVAC 494 ASM Assembler.

FOR Calls and activates the FORTRAN compiler.

COB Calls and activates the COBOL compiler.

REPORT Calls and activates the UNIVAC 494 Report Writer.

2.3. ORGANIZATIONAL CONTROL STATEMENTS

Organizational control statements are discussed in the following paragraphs.

2.3.1. The JOB Statement

The JOB Statement identifies the beginning of a job deck and furnishes certain parameters necessary for scheduling and accounting purposes.

- Format:

  The JOB statement has the following general form:

  # JOBb̸optionsb̸identity/individual,account,priority,running time, output/output

- Options:

  A – E Logical switches maintained by the system for each job. The switches can be referenced by each task within a job, affording simple on/off parameters accessible to, and controllable by, the task, and initially set external to the program.

  F Break in primary output is requested. When the primary output device (printer) is not being used to full capacity by the present job, the output device may be released temporarily to another job.

  G Break in secondary output is requested. When not needed by the present job, the secondary output device (card punch) can be released temporarily to another job.

  I Inhibit printing of page headers on primary output.

  L Accounting information will be submitted for this job even if accounting is inhibited in general.

  P Print primary output immediately, i.e., at execution time.

  Q Applies to remote site job entry only. Any output for job is to be on-site rather than at the remote site.

R The job stream describes a set of RT/COMM (REAL TIME/COMMUNICATIONS) tasks.

T All information on the job is broken into separate task reports. Used with L option.

- Specifications:

Identity/individual specifies the project, department or other identification, and the individual responsibile for the job. This information is used by the installation for routine purposes upon job completion.

Account supplies a charge number used in logging charges, and must be specified.

Priority specifies job priority and is designated by a single alphabetic character:

A – E This priority is used by scheduling routines for selection purposes and is the initial service priority assigned to each task at execution time. The priority designators have the following meaning:

A The job deck is a high priority run for real time or mandatory programs, which will be processed at the expense of other jobs within the system. The option may require the suspension and roll-out of an operating program to obtain the necessary primary storage.

B The job deck is rush, and is used for remotely originated, high priority production or for jobs having critical turnaround constraints.

C The job deck comprises normal batch programs submitted on site and requiring no abnormal priority designation.

D The job comprises program development work on programs which are to be used as background to production jobs.

E The job has lowest priority (used normally for online maintenance routines).

Running Time is optional and specifies the amount of CPU time in minutes required for the job, as estimated by the user. If this time is exceeded, the operator may optionally terminate the job, as guided by the operating policy of the installation, and his own CPU time estimate. If the specification is omitted, running time is determined by the installation option that has been set at systems generation time. If the field is marked with a C, the job is indicated as continuous, as is the case with some RT/COMM programs or long input/output bound batch programs.

As part of the post processing, the total amount of CPU time utilized by each task within the job stream is noted to the user as an aid in making future running time estimates.

Output/Output is optional and, in the first output field, provides the system with a page-number estimate of the amount of primary output that the user is expecting. The second output specification is the estimated number of card images that will be submitted to the secondary output stream. If the field is omitted, the number set at systems generation time is assumed. If the field is marked with a C, the output is continuous; no estimate is supplied.

- Example:

```
# JOBƀABCƀACCOUNTING/SMITH,312642,C,5,C/0
```

The above card directs the system to assign normal batch program priority, set logical switches A, B, and C to the "on" condition, and set switches D and E to the "off" condition. The department is Accounting, and the individual submitting the job is Smith. All costs for running the job will be logged to account number 312642. The estimated

CPU time required to run tasks described by the job is five minutes. The estimated number of pages of primary output is C, indicating that the described tasks are continuous. (Normally this option is used for tasks which have been debugged.) The amount of output is data dependent and varies radically. No secondary output is anticipated.

2.3.2. The START Statement

The START statement identifies a job description which is to be scheduled immediately or at a set time of day. The statement may identify a job for processing a generated data file, initiating a parallel operation, or initiating jobs stored with the system as a data file. This is of particular advantage in scheduling jobs from remote terminals to avoid re-entry and transmission of the required control statements.

■ Format:

#STARTƀoptionƀname/version,library,day clock time,date

■ Options:

D Delete the job entry from the START list.

N Start the named job now.

R Register the named job for reactivation on the RESTART command.

X Abort the element and remainder of the current job (i.e., skip to the next JOB card) if a find is not made.

■ Specifications:

Name/Version is the symbolic name, and version designator, if any, assigned as the identity of the source element containing the control stream which is to be located and scheduled. The source element must contain a complete job stream.*

Library identifies the location of the control stream on mass storage. The following specifications are valid:

SYS specifies that the named control stream element is contained in the systems library.

JOB specifies that the named control stream element is contained in the user's job library.

xxxxx Group library number, which may be up to five digits, specifies the file number by which the group library is identified. Decimal numbers may range from 0 to 9999, and must be indicated by a D following; e.g., 9989D. Octal numbers may range from 0 to 77777.

Day Clock Time specifies the time of day, on a 24 hour clock, at which the job deck will be scheduled. The time format is HH/MM where HH is 0 through 23 to indicate the hour of day and MM is 0 through 59 to indicate the minute within the hour. Absence of the specification implies that the job deck will be scheduled immediately.

Date specifies the date upon which the job will be initiated. The date has the form YYDDD, where YY represents the last two digits of the year (range 00–99) and DDD represents the day of the year (range 001–366). If the date field is omitted the current date will be used.

START statements are processed upon their occurrence in the primary input stream. Each statement will cause the entry of an independent job deck from the described direct access storage library to be processed in the multiprogrammed environment. Once the job stream is activated, no form of direct communication exists between the requesting job and the requested job. All files which are to be conveyed to the activated job stream from the requester must be previously registered and stored in the Master File Directory.

**The SOURCE statement (see 4.5.1) provides a method for creating and/or updating this type of element.*

2.3.3. The Correction (COR) Statement

The correction (COR) statement is a means for correcting an absolute element within the library complex or for applying corrections to absolute memory locations. Corrections applied to the absolute element will remain until the library is altered or the system is reinitialized. Corrections or modifications to locations in memory are subject to program modification and are not necessarily permanent. Modifications to memory locations must be made with extreme caution to prevent inadvertent destruction of vital locations. COR statements may be submitted through the primary input stream or as internal control statements.

- Format:

For application to an absolute element, the COR statement has the following general form:

#CORƀoptionsƀlibrary specification/element identification, relative correction address/segment number, corrections, etc.

For application to memory locations, the COR statement has the following general form:

#CORƀoptionsƀrelative correction address/corrections, etc.

- Options:

A For application to memory location correction statement only. Corrections will be applied to the indicated memory locations. The library specification/element identification field is not required. The entire range of primary storage locations is accessible. The relative correction address is, in this case, the location relative to the system base, which is normally zero.

P All secondary executive routines which are not active are purged from memory so that corrected routines will be used when required.

X An ABORT$ condition is caused if an error is encountered while processing COR statements. OMEGA terminates the current job.

Z An ERROR$ condition is caused if an error is encountered. OMEGA terminates the current task, but continues the job.

Blank If no options are present, all of the fields described for correcting an element are required.

- Specifications:

Library specification identifies the library file upon which the element that will be corrected can be found:

JOB indicates that the element is in the job library.

SYS indicates that the element is in the systems library.

xxxxx Group library number which may be four decimal or five octal digits, ranging from 0–9999 $_{10}$ or 0–77777$_8$, specifies the file number by which the group library is identified. The letter D must follow all decimal numbers.

Element identification is the symbolic name, and version designator, if any, assigned as identification of the element which will be corrected.

Relative correction address (absolute element) is the octal position within the segment, relative to the segment base, to which the correction will be applied. For nonsegmented programs, the correction address is relative to the element base. The relative correction address is, in all cases, followed by a slash (/).

Relative correction address (memory locations) is the absolute address, relative to zero, to which the correction or modification will be applied. One or more corrections may immediately follow the relative correction address.

Segment number is the number of the segment within the collection to which corrections will be applied. For nonsegmented programs, the segment number is zero.

Corrections may be any numeric instructions or data desired. Numeric values which are decimal must be followed by the letter D. Leading zeros are not required. Consecutive locations may be changed by a series of correction fields separated by commas. A new correction address need not start on a new card. If nore than one card is required, continuation cards may be used, with continuation being indicated by a 12–7–8 punch (#) as the terminating character on a card.

- Examples:

  – Correction of an absolute element in the job (JOB) library:

  #CORƀZƀJOB/FCA,67/2, 65000 01002, 15130 00131, 165/4, 16035 00001, 65000 00100#
  (continuation card) 14030 00131, 61000 01172, 060710, 3715

  This COR statement causes the absolute element FCA to be found in the job library, and changes locations 67–70 relative to the base of segment 2 and locations 165–172 relative to the base of segment 4.

  – Correction of an absolute element in the system (SYS) library:

  #CORƀXƀSYS/LOAD, 323/2, 16070 02710, 12000 00000, 12000 00000

  This COR statement causes the absolute element LOAD in the system library to be found, and changes locations 323–325 relative to the base of segment 2.

  – Correction of a location in memory:

  #CORƀAXƀ6541/11430 07621, 61000 05462, 7231/1, 17, 6

  This COR statement changes the contents of memory locations 6541–6542 and 7231–7233.

- Method of Operation:

The COR routine locates the absolute element by using the library specification and the element identification to retrieve the table of contents (TOC) for the element which locates the element in the system. If the element is segmented, as shown in the TOC, the COR routine reads the segment descriptors appended to the element by the Loader during collection, which locate the segment in the element.

The corrections for the first determined relative correction address, and for all consecutive locations, are held in the system until a second address encountered. At this point, the corrections being held are deposited at the first and consecutive addresses. The corrections for the second address will be held until a third address is determined, etc. This mechanism permits access to a maximum number of locations through a minimum number of specifications. There is no limit to the number of corrections which can be made through the COR statement.

When correcting memory locations under the A option, the COR routine deposits the corrections at the address specified. The deposit address is incremented after each correction or until replaced by a new deposit address. Any number of corrections may be applied.

- Diagnostic Messages

The following diagnostic messages reflect error conditions which may be encountered while processing the COR statement. These messages appear in the primary output stream. Error status codes, if applicable, are returned in the A register.

DRUM ERROR

An unrecoverable drum error has occurred during processing of the COR statement. The correction routine will continue processing.

ERROR ON CARD READ

An error has occurred on the attempt to read a continuation card. The card in error will be printed, and the next card will be read.

CARD FORMAT ERROR

The format of the correction card being read is in error. The field(s) in error are passed over, and processing is continued.

ERROR, X OPTION, ABORT JOB

An error condition has occurred during processing of the COR statement and the validity of subsequent tasks depends upon the present corrections being error free. The job is terminated.*

ERROR, Z OPTION, TERM TASK

An error condition has occurred during processing of the COR statement and the validity of the task depends upon the present corrections being error free. The task is terminated, but processing continues for the remainder of the job.*

SEARCH FOR ELT PRODUCED ERROR

The element named on the COR statement cannot be located. The correction routine will terminate according to specified options.

ERROR STATUS FROM UNSTRINGER

An error condition has occurred during processing of the UST$ request. The routine will continue or terminate according to specified options.

ERROR STATUS FROM CONVERSION

An error condition has occurred during processing of the CVT$ request. The routine will continue or terminate according to specified options.

**The cause of the error conditions for the X and Z options will be specified on the basis of the other defined error message.*

2.3.4. The Parameter (PRAM) Statement

The parameter (PRAM) statement provides the user with a method for submitting operational parameters to a task at execution time. These statements are submitted externally to the program through the primary input stream or internally as a service request.

- Format:

The PRAM statement has the following general form:

#PRAMb̸optionsb̸parameter1,parameter2,...

- Options:

A Parameters contained in the specification field are in alphanumeric form, with each containing one or more characters from the groups A through Z and 0 through 9. No special characters are allowed in the specification field.

In the absence of the option, parameters contained in the specification field are considered to be numeric values. Decimal numbers are indicated by a character D at the end of the number. Octal numbers are recognized as containing 1 to 10 octal characters from the number set 0 through 7. Decimal numbers are converted to the binary equivalent. The conversion process limits the magnitude of the number to 30 bits. For octal numbers, characters are converted to binary and are right justified in a 30-bit word. Negative values in numeric fields are indicated by a leading minus sign (-); positive values are indicated by a leading plus sign (+) or by the absence of a sign.

C Parameters sent with this option are considered as 'literals'. All alphanumeric and special characters are valid with the exception of delta (b̸ or 11–7–8 punch) which will terminate the literal.

X Abort the task and the remainder of the job (i.e., skip to the next JOB card) if errors are detected on the PRAM statement.

Y Accept the next task although errors are detected on the PRAM statement.

Z Abort the next task but continue the remainder of the task if errors are detected on PRAM statement.

- Specifications:

Each parameter contained in the specification field will be translated or converted to a 30-bit word(s) as indicated by the inclusion or omission of the A option. When the A option is specified, all parameters on any one statement are mapped as alphanumeric fields, leading blanks are ignored, and the fields are left justified with interim and trailing blank characters. When the A option is omitted, all parameters on any one statement are mapped as numerics and converted to equivalent binary form. A parameter field which consists of blank(s) only will be ignored for both alphanumeric and numeric number fields. The maximum number of fields on any one statement is limited to 24_{10} words.

- Termination Control:

Termination control may be used to indicate the end of the parameters on the PRAM statement. This control is useful for indicating the necessary blanks in the last alphanumeric field. A period (.or 12–3–8 punch) or a delta (b̸ or 11–7–8 punch) is used for this purpose.

NOTE: The period is not a valid terminating character when the C option is used; in this case, the delta is the terminating character.

- Method of Operation:

Each parameter statement encountered during the processing of the primary input stream is unstrung and the parameters contained are mapped into 30-bit words. The number of words is determined by the number of parameters contained in the statement.

The words are arranged sequentially, as indicated by their left-to-right order on the statement, and transferred to executive storage linked to the task addendum. Subsequent statements will be handled in the same manner.

An operating task acquires parameters through the use of the SEND/RECEIVE service request mechanism which transfers data to the 'requester's' deposit area under control of OMEGA (see 2.5.6). The RECEIVE operator specifies the base primary storage address and the number of 30-bit parameters desired. Words will be consecutively transferred to the requester on a first in, first out basis until the specified number is attained or the supply submitted through the PRAM is exhausted. Identification 77777 will be specified by the RECEIVE operator.

- Status Codes:

Upon return of program control from the internal PRAM request, the following status information is conveyed to the requesting task in the A register.

Normal Completion: The A register is set to binary 0 which indicates that parameters on the statement have been transferred through the SEND operator to the request with 77777 identification.

Abnormal Completion: The A register contains one of the following values, dependent upon the cause. The parameters specified by the request will not be entered into parameter storage.

4000000000	Deposit area has overflowed.
4100000000	Deposit area is not within the lock limits of requester.
4100000010	No parameter given on PRAM statement.
4200000000	Invalid option specified or SEND is abnormal.
4200000001	Nonoctal value specified.
4200000002	Octal number greater than 7777777777 used.
4200000004	Non-numeric character used.
4200000010	Decimal number greater than 536870911 used.
4300000000	Exceeds maximum number of fields (24 decimal).
4400000000	No primary storage available for this request.

- Examples:

The following sample indicates various ways in which parameters may be submitted:

```
#PRAM Y  1277   ,  12777   ,  2048D
#PRAM Y 123,,256,     ,123.
#PRAM AY   ABCD,      ,FGHUJKL,, ZXCVBNƀƀƀΔƀ
#PRAM Y   255D, + 256D   ,   753,+753,+77775.
#PRAM Y   256D, –256D  ,  753,–753,–456321.
#PRAM YC DATA TRANSMITTED INTACT, WITH PUNCTUATION Δ
```

Assume the execution of:

```
RECEIVE$ TOM, 32D,77777
```

The following will be the contents of 32 words after the location TOM:

```
TOM    0000001277
       0000012777
       0000004000
       0000000123
       0000000256
       0000000123
       0607101105
       1314153217
       2021050505
       3735103307
       2305050505
       0000000377
       0000000400
       0000000753
       0000000753
       0000077775
       0000000400
       7777777377
       0000000753
       7777777024
       7777321456
       1106310605
       3127062330
       2216313112
       1105162331
       0610315605
       3416311505
       2532231031
       3206311624
       2305050505
```

Remarks:

1. Leading blanks are ignored for alphanumeric fields.
2. Interim and trailing blanks are saved for alphanumeric fields.
3. Leading and trailing blanks have no meaning for numeric number fields.
4. Wholly blank fields are ignored.

2.3.5. The LOG Statement

The LOG statement provides the user with a means for entering information into the systems log. This statement may be submitted through the control stream as a primary input statement or as an internal control statement.

- Format:

  The LOG statement has the following general form:

  #LOGƀoptionsƀcode number,literal

- Options:

A–K Specify the nature or source of the LOG message contained in the code and the literal, and are conventionalized as follows:

A	Miscellaneous
B	User LOG Messages
C	Console Messages
D	System Information
E	Job Accounting
F	Input/Output Diagnostics
G	CPU and Core Diagnostics
H	Program Contingency Diagnostics
I	Used by OMEGA For Logging Control
J	(Unused)
K	Output To Exec Primary Output Stream

Only one of the option letters A through K may be specified per request. Absence of A through K options in this case implies the D option.

- Specifications:

Code number is an octal value which specifies a particular message within a group. Groups are specified by option letters A through K. The range of codes is 000 through 777.

Literals are the messages to be contained on the Systems Log and displayed for user interpolation through the accounting routine. Literals can be of variable length containing both binary and alphanumeric fields. All literals to be displayed by the accounting routine must be specified previously by the user through the code word, with the exception of literals submitted with B option, for proper interpolation and processing. The maximum size of a literal is 276_8 words.

- Method of Operation:

Log statements may be submitted externally through the control stream and/or collected with the task element or they may be submitted dynamically during execution of the task. Upon return of program control from a dynamically submitted log statement, the following status information is conveyed in the A register:

Normal Completion	The register is set to binary zero.
Abnormal Completion	The register contains one of the following values dependent upon the cause. No logging is performed.
4200000000	Incorrect parameter: log statement could not be interpolated.
4300000000	An unrecoverable error was encountered during the operation.

Registers B1 through B6 and Q will contain the original values held prior to execution. B7 contains the address of the log statement.

2.3.6. The DUMP Statement

The DUMP statement may be used to obtain a postmortem dump of all, or part of the primary storage used for the execution of a task; or selected areas of direct access storage or selected tape blocks. The statement calls for a reentrant secondary executive routine, DUMP, and can be submitted through the primary input stream or during execution of the task through service requests. When submitted through the control stream, the DUMP statements must immediately follow the end-of-task statement to which they pertain.

Upon submission of a service request or termination of a task, the specified areas are extracted from primary storage, direct-access storage, or tape storage. The areas are edited into a readable format and submitted to the primary output stream.

- Format:

  The DUMP statement has the following general form:

  #DUMPƀoptionsƀ v_0, v_1, v_2, v_3

- Options:

  O, A, D, F — Specify format to which each extracted word will be converted before submission to the primary output stream, as follows:

  O — Each word is 10 octal characters.

  A — The extracted data is in Fieldata alphanumeric form and needs no conversion.

  D — Each 30-bit word will be converted from binary to decimal and will be represented as a signed decimal value.

  F — The extracted data is a two-word floating point number and will be converted and represented as a signed decimal number.

  P — Specifications v_0, v_1, v_2, and v_3 describe a peripheral device or direct access storage.

  E — Perform the indicated dump only if the task is terminated by an ERROR or ABORT condition.

  C — Space the paper to a new page before printing.

  L — Include a listing of the operating task addendum, activity addendum of the requesting activity and all storage modules linked from the activity addendum.

- Specifications:

  v_0 is the alphabetic character(s) of the file code to which peripheral or direct access storage is assigned.

  v_1 is the number of words of primary storage, or direct access storage, to be dumped. Whenever v_0 specifies a tape, v_1 will specify the number of tape blocks to be dumped. A maximum of 6200_8 words will be dumped per block. V_1 may not exceed a value of 77770_8.

  v_2 has meaning based upon the task, and may specify:

  the starting address of primary storage relative to the PLR, or the task base of the requestor, the logical increment in the case of direct access storage, or

the number of tape blocks to be bypassed at the beginning of the tape storage.

v_3 is an optional parameter specifying the number of lines to space before each print operation. If this parameter is not used one line is assumed. This parameter may be used for skipping blocks when dumping from tape.

- Method of Operation:

When the DUMP statement is submitted through a service request, the executive entry instruction is used (see 2.5).

Two calls are recognized by the DUMP routine:

EXRN*	20741

Action: Requests the DUMP routine to unstring a card image and to perform the specified display operations.

User: Unrestricted.

Where used:

– The #DUMP card may be entered into the primary input stream, where it is detected and submitted by the Control Card Interpreter (CCI) routine. Usually this card is placed after the #END card of the operating task's input stream, causing postmortem dump. The card may also be entered among the data parameters, where it will be detected by the CCI routine and processed immediately without affecting the operation activity.

– A ƀDUMP card may be read as a parameter and submitted by the operating task. B7 must be set (relative to the task base) to the buffer base with an EXRN*20741 given.

– The #DUMP card (starting in column 18) may be inserted in a SPURT source deck. SPURT recognizes the #sign and generates the necessary packet.

On receiving the 20741 call, the DUMP routine takes the task base from the operating task addendum, and all increments are added to this base. The primary storage limits are also taken from the task addendum and no dumping is permitted outside these limits.

EXRN*	20740

Action: Requests the DUMP routine to process an internal packet.

User: Unrestricted.

Where used:

– Must be sumbitted as an internal packet, with the following format:

<table>
<tr><td></td><td colspan="2">EBJP*B7</td><td colspan="2">N</td></tr>
<tr><td></td><td colspan="2">FILE CODE (v_0)</td><td colspan="2">NO. OF WORDS (v_1)</td></tr>
<tr><td></td><td colspan="4">INCREMENT (v_2)</td></tr>
<tr><td></td><td colspan="3">29 4
OPTIONS</td><td>3 0
SP(v_3)</td></tr>
<tr><td>N →</td><td colspan="2">EXRN</td><td colspan="2">20740</td></tr>
</table>

The parameters within the packet correspond to the indicated v-fields on the #DUMP statement. The option field is a master bit designation of the desired alphabetic character beginning with bit 29 for A and ending with bit 4 for Z (e.g., for the D and C options, set bits 26 and 27, respectively).

B7 must be set relative to the lower lock of the requesting activity. The DUMP routine takes the starting increment and primary storage limits from the storage module of the requesting activity. No data will be dumped outside of the lock limits set in the storage module.

- Examples:

#DUMPƀOƀ,70000,0,1 (For primary storage assignment)

DUMP 70000 words of primary storage, in octal format, starting at the task base, and space one line between print images. If 70000 is larger than the primary storage assigned to the task, only that primary storage assigned to the task will be dumped.

#DUMPƀDEƀ,100D,400,2 (For primary storage assignment)

The E option indicates that the dump is to be performed only if the task is terminated with an ERROR or ABORT service request. The D option implies that each data word is to be converted to its signed decimal value; 100D, 400 specifies a dump of 100_{10} words beginning at the primary storage area relative address at 400_8; the 2 requests that printing be double spaced.

#DUMPƀPAƀC,5,0 (For tape assignment)

The A option indicates that the described data is in Fieldata or printable form, ready for submission to primary output as ten word lines. Specification of C indicates file code C; 5 requests printing of five consecutive blocks; 0 requests rewind of tape and bypass of zero blocks.

#DUMPƀPOƀA,400D,300,1 (For direct access storage assignment)

The O option indicates that the described data will be represented on the dump in its octal form. Specification A indicates file code A; 300 specifies the starting logical increment from File A of direct access storage; 1 specifies a one line space before printing; 400D specifies a dump of 400_{10} consecutive words.

2.3.7. The END Statement

The END statement is required to mark the end of the primary input stream of a task.

- Format:

  The END statement has the following form:

  #END

  The statement is nonoperational; therefore, no options or specifications are required. The end-of-task statement is required to separate the end of one task from the beginning of a subsequent task within the job deck. The only control statements pertaining to the current task which can follow the END statement are the postmortem DUMP statements.

2.3.8. The FIN Statement

The FIN statement is a termination indicator used in several situations for delineating end-of-input stream conditions, such as end-of-tape or end-of-transmission from remote terminals, and to avoid interlocks or hangups on card equipment. The statement is not used to mark the end of a job deck.

- Format:

  The FIN statement has the following form:

  #FIN

  Since the statement is basically nonoperational, no options or specifications are required.

2.3.9. The READY Statement

The READY statement precedes each run of jobs or data submitted from a remote terminal. The statement is transmitted only once, when connection is made with the central site.

- Format:

  The READY statement has the following form:

  #READYƀoptionsƀstation

- Options:

  Options are variable depending upon the remote unit (see individual documents for remote peripheral systems).

- Specifications:

  Station identifies the remote terminal. A list of communicating terminals is created by systems generation which is used to verify legality, and to determine station priority and output line, and other unit capacities.

2.3.10. The CALL Statement

The CALL statement is used to override normal system action in routine primary or secondary output. The normal system action is set at systems generation time to the high speed printer and card punch.

- Format:

  The CALL statement has the following general form:

  #CALLƀoptionsƀname/version, library, type, file ID

- Options:

  Options are passed to the named unit record routine for its interpretation. The only option recognized by OMEGA itself is the X option.

  X Abort the job if the called for unit record routine cannot be located or activated.

  Absence of the X option implies use of normal output unit record routine if the called for routine cannot be located or activated and the CALL statement is ignored.

- Specifications:

  Name/Version is the symbolic name and version, assigned as identification of the called for unit record routine. At the time of the call the named unit record routine must be in absolute form with all required peripheral facilities collected with it in the form of the ASG statement.

  Library identifies the random access program library file in which the called for program element is contained. The following specifications apply:

  Type:

  P Primary Output Routine

  S Secondary Output Routine

  SYS indicates that the named absolute element is contained in the systems library.

  JOB indicates that the named absolute element is contained in the user's job library.

  xxxxxGroup library number, which may be four decimal or five octal digits, specifies the file number by which the group library is identified. Decimal numbers may range from 0 to 9999, and must be indicated by a D following, e.g., 9989D. Octal numbers may range from 0 to 77777.

  Absence of a specification implies a search of the user's job library.

  File is used to specify the primary and/or secondary output stream which the called unit record routine is responsible for processing. P indicates primary output stream; S indicates secondary output stream.

  File ID is an optional parameter that may be specified for tape primary output routines. If present, the tape unit record will submit a CHANGE$ request to inform the operator that ID file is to be used.

  Because of the unpredictability of the time at which output unit record routines will be activated, CALL statements should be positioned toward the beginning of the job deck, immediately following the JOB statement. For example, upon encountering the CALL statement in the primary input stream, the system will store the statement as an override for normal unit record call. The statement will be processed when the system requires primary and/or secondary output in the normal manner. (See 3.4.3 for determining when output unit record routines are activated.)

2.3.11. The HDG Statement

The HDG statement is used to modify the heading line normally printed at the top of every page of primary output.

- Format:

  The HDG statement has the following general form:

  #HDGb̸optionsb̸text

- Options:

  N Revert to the normal page heading from a revised heading.

  R Re-set the page count on the revised heading to one.

- Specification:

  Text specifies the data that will be printed, left justified, in the central 70_{10} print positions of the heading line of the following pages. The JOB NUMBER and PAGE NUMBER information is not affected.

  The text is limited to 70_{10} characters and spaces. There are no restrictions on special characters.

- Methods of operation:

  The HDG statement may be used in one of three ways.

  – The HDG card may be entered into the primary control stream where it is detected and submitted to the Control Card Interpreter where it is processed to provide information for the primary output routines. Usually, the card is placed after the #END card of the operating task's input stream, causing the header information on the following pages to be updated. The card may also be submitted as part of a data deck, in which case it will be processed immediately without affecting normal processing.

  – The HDG card may be submitted via an EXRN call on CCI (EXRN 20406). If the HDG card (starting in column 18) is submitted in a SPURT source deck, SPURT will recognize the '#' sign and generate the necessary instructions and packet.

  – The HDG card may be submitted to the LOADER as a secondary control card with a space in COL. 1 instead of a '#'. The LOADER will accept the card, generate necessary code and, upon execution of the absolute element, submit the HDG card for evaluation.

2.3.12. The LST Statement

The LST statement is used to control the production of primary output. Upon detection of a #LST card, the system will ignore all PRINT requests, thereby effectively terminating all primary output, until a subsequent #LST card is detected, at which time normal operation will be resumed. There is no limit to the number of times that the LST card may be used.

- Format:

  The LST statement has the following format:

  #LST

- Options:

  There are no options for this statement.

- Method of Operation

  The LST statement can be submitted for operation in one of two ways:

  1. Via the primary input stream as part of a job control sequence.

  2. As part of the object code of an element. If the LST card is submitted to the SPURT Assembler, starting in column 18, the Assembler will generate the necessary code for execution at program run time.

2.4. TASK ACTIVATION CONTROL STATEMENTS

Task Activation Control Statements are the primary control statements which describe an absolute program produced by the Loader. These include statements which are selected on the basis of priority and available facilities; statements readied for execution by assigning peripheral devices, direct access storage, and primary storage area; and statements activated by loading into primary storage and by registering the initial activity addendum on CPU queue (see 10.2).

Two forms of task activation statement are recognized by OMEGA. The first calls for the activation of a systems processor or utility routine which may require special preprocessing during the allocation phase. The second is the GO statement which is the normal method of activating user-developed programs.

2.4.1. The GO Statement

The GO statement is used to initiate the execution of an absolute program prepared by the Loader routine:

- Format:

  The GO statement has the following general form:

  #GOƀoptionsƀname/version, library, time estimate

- Options:

  A–E Identify logical test switches maintained by the system for each job. The switches are specified on the JOB card as applicable throughout the job. When the switches are defined or being reset with the GO card, the R option must be used.

  P (FORTRAN only) Indicates 90-column card input.

  R Reset or clear logical switches as defined by A–E option letters.

  H,M,L Set service and selection priority (high, medium, low) for execution of the task.

  W Specifies that timing by task is required.

  X Abort the element and the remainder of the job (i.e., skip to the next JOB card) if error occurs during loading or execution.

  Y Accept the absolute element for execution although the element is marked as containing an error.

- Specifications:

Name/Version is the symbolic name and version, if any, of the absolute element which will be entered into memory and initiated.

Library identifies the library in which the absolute element is contained on direct access storage. Any of the following specifications are valid:

SYS specifies that the absolute element is contained in the system's library.

JOB specifies that the named element is contained in the user's job library or in a group library previously linked to the user's job library.

xxxxx Group library number, which may be four decimal or five octal digits, specifies the file number by which the group library is identified. Decimal numbers may range from 0 to 9999, and must be indicated by a D following; e.g., 9989D. Octal numbers may range from 0 to 77777.

Absence of a library specification implies a search of the user's job library.

Time Estimate – If the W option is present this field is interpreted as the number of tenths of a second to be allowed for this task. If the estimate is exceeded, CP OFLW process will occur in the same manner as when JOB time estimate is exceeded.

2.4.2. Systems Processor Control Statements

Systems processor control statements call for the execution of a task whose name/version and processing requirements are known to the system. In general, all specifications and options contained in the control statement are passed to the implied processor. The processor is specified by the function code in the control statement. TEST, used to call the Test system, and FOR, used to call and activate the FORTRAN compiler, are examples of processors called in this way. A detailed description of applicable statements and secondary language is contained in subsequent sections.

Implied processors, other than the Program Library Editor, may be contained in the systems library, job library, or group library. When locating the called processor, the selection process will search the systems library and then the job library. Because of the nature of the Program Library Editor, it must be contained in the systems library.

A list of system processors and utility routines activated in this manner is given on the following pages. Detailed description of control statements and secondary language is contained in subsequent sections.

2.5. SERVICE CONTROL

Service control is the interface by which an operating task communicates with and requests services from OMEGA. An operating task requests service by a sequence of instructions which submits a parameter packet appropriate to the request with interrupts to OMEGA.

Since hardware guard mode is enforced against operating tasks, the special executive entry instruction (EXRN) is used to submit a request. This instruction causes an interrupt and includes a 15-bit field identifying the function requested. In many cases, operational registers A, Q, and B7 are also used in communication of parameters for the request, and the status or condition upon completion. Registers B1 through B6 are always preserved. The calling sequences are consistent with reentrant programs since they are restricted to the operational registers.

The routines which perform the requested service are logically executed as an extension of the requester activity. This maintains CPU service at the priority level assigned to the requester and provides for direct logging of CPU time used in the execution of the function to the requesting task. An organization is also achieved which maintains interruptability for the extent of the function. Service routines which are not permanently resident will be called and controlled by OMEGA. Some functions will be performed by reentrant routines which will service simultaneous requests for several user or systems elements, such as input/output handlers and the segment loader.

Since a service request is executed as an extension of the calling activity, the caller, under the current activity addendum, will be delayed until completion of the request. If the task has previously fragmented itself (see 2.6), the task may be eligible for program control under another activity addendum associated with the task.

Essentially, all OMEGA functions called upon during execution of a task are service requests, such as activity control, contingency control, and input/output. The service requests are explained in detail in other sections of this manual. The service requests which are contained in this section are the miscellaneous functions which are not covered in subsequent chapters.

2.5.1. Summary of OMEGA Service Requests

Table 2–1 presents a summary of general service requests to OMEGA which can be performed by a batch or real time communications program. Some of the requests are restricted to use by OMEGA only. The summary is broken into general service requests as shown below, input/output service requests (see 3.4.4), and optional real time/communications requests (see Section 9).

GENERAL SERVICE REQUESTS			
MNEMONIC	DESCRIPTION	RESTRICTION	SECTION
ABORT$	Purge job from system	No	2.6.4.2
CABORT$	Deallocate current activity	1	2.5.6.4
CEXIT$	Common subroutine exit	1	2.5.6.4
CHANGE$	Direct operator to change tape reel	No	5.3.3
CKPT$	Records environment as part of program interruption	No	2.5.10.1
CLD$	Clear dual index mode	No	2.8.4.4
CMOVE$	Perform data transfer	1	2.5.6.3
CSWCT$	Switches control between routines	1	2.5.6.4
DATIM$	Supply current date and time	No	2.5.8.3
DELAY$	Delay activity for a given time period	No	2.5.8.4
DEMOUNT$	Direct operator to dismount tape reel	No	5.3.2
ERRADD$	Establish fault routine address	No	2.9.2
ERROR$	Purge task from system	No	2.6.4.3
FADD$	Expand direct access storage	No	3.2.11.3
FADDA$	Expand direct access storage	No	3.2.11.3
FETCH$	Subroutine load	No	2.5.4
FETCHL$	Locate Table of Contents	No	2.5.5
FOFADD$	Establish floating point overflow address	No	2.9.2
FORK$	Split program control	No	2.6.3.1
FREL$	Release direct access storage	No	3.2.11.2
FRELA$	Release direct access storage	No	3.2.11.2
FRPL$	Replace direct access storage	No	3.2.11.4
FRPLI$	Replace direct access storage	No	3.2.11.4
FUFADD$	Establish floating point underflow address	No	2.9.2
JOIN$	Close split program control path	No	2.6.3.2
LOAD$	Load Segment	No	2.5.3
LOADA$	Load and activate segment	No	2.5.3

Table 2–1. General Service Requests (Part 1 of 2)

GENERAL SERVICE REQUESTS			
MNEMONIC	DESCRIPTION	RESTRICTION	SECTION
MADD$	Extent core assigned	No	2.8.2.2
MOUNT$	Direct operator to mount tape reel	No	5.3.1
MREL$	Release core assigned	No	2.8.2.3
QREF$	Activate queue process activity	No	2.6.2.2
RECEIVE$	Receive parameters and/or data	No	2.5.7.2
REG$	Standard activity registration	No	2.6.1.1
REGQ$	Queue process activity registration	No	2.6.2.1
RETURN$	Terminate activity (conditional)	No	2.6.4.1
RETURN1$	Terminate activity (unconditional)	No	2.6.4.4
SEND$	Send parameters and/or data	No	2.5.7.1
SETD$	Set B registers to dual index mode	No	2.8.4.3
SET15$	Set B registers to 15-bit mode	No	2.8.4.1
SET17$	Set B registers to 17-bit mode	No	2.8.4.2
TCORE$	Supply primary storage limits of task	No	2.8.2.4
TCORE1$	Request that a task be not subject to compaction and supply primary storage limits of task.	No	2.8.3.1
TESTFL$	Test floating point overflow/underflow	No	2.9.4.2
TESTFOF$	Test floating point overflow	No	2.9.4.2
TESTFUF$	Test floating point underflow	No	2.9.4.2
TFC$	Test file code for type device	No	3.2.11.1
TIMED$	Supply edited time of day	No	2.5.8.1
TIMEL$	Supply elapsed central processor time for task	No	2.5.8.2
TIMEQ$	Supply current quantum time	No	2.5.8.5
TIMEY$	Supply current time of year	No	2.5.8.6
UNSOL$	Unsolicited operator messages	No	5.4
XOFF$	Set logical switches off	No	2.5.9.1
XON$	Set logical switches on	No	2.5.9.2
XTEST$	Supply current logical switches	No	2.5.9.3
#	Submit control statement internally	No	2.5.2

① Can only be used by routines that operate under control of the content supervisor (i.e., common S/R or non-resident exec. service routines).

Table 2–1. General Service Requests (Part 2 of 2)

2.5.2. Internal Control Statements

OMEGA recognizes a subset of external control statements which can be submitted and processed as internal control statements during execution of the task, thereby providing the user a degree of dynamic control over the task/job environment. Essentially all nontask activation control statements may be submitted in this manner by either of the two methods illustrated below, which include the following set:

CONTROL STATEMENT OPERATOR	DESCRIPTION	NOTE
START	Schedule the execution of a related job deck	Will be queued for scheduling
PRAM	Convey operational parameters to a task	Processed immediately
COR	Enter corrections to user program	Processed immediately
MSG	Convey abnormal operating instructions to console op.	Processed immediately
LOG	Record user specified literal on systems log file	Processed immediately
DUMP	Perform diagnostic printouts of direct access storage, primary storage or tape	Processed immediately
ASG	Assign input/output device or direct access storage to a task	Processed immediately
FREE	Release input/output device or direct access storage assigned to the task	Processed immediately
SWITCH	Change input/output reference codes	Processed immediately
DEL	Delete elements from job library	Processed immediately

Details of the function and operation of the above statements may be found in appropriate sections of this manual.

2.5.2.1. METHOD OF SUBMISSION

Two methods are provided for submitting control statements during task execution. The first method involves the use of the control statement contained in the task code entered there through parameter mechanism or generated by an assembler. The second method involves the submission of statements through the job control stream. In either case, the operating task/activity dynamically submits the control statement to OMEGA by a service request. The format of the service request is as follows:

	ENT*Q	NUMBER OF WORDS
	EBJP*B7	N
	CONTROL STATEMENT	
N →	EXRN	20406

Number of words specifies the length of the control statement literal.

Control Statement is a literal containing the complete statement in Fieldata, left justified in the field. The statement format must be identical to that of the externally submitted control statements, with one exception: the # symbol in position one (1) of the statement is present at execution time. Otherwise, all fields are present and follow normal conventions described in 2.2.1.

Upon submission of a service request, OMEGA will perform the following functions:

- Interpret and unstring the literal address by B7 and switch to the OMEGA routine responsible for performing functions as determined by the function code.
- Process or queue the processing of the request, if possible, dependent upon the requested function.
- Return program control to the instruction following EXRN.

Upon return of program control from the requester, registers B1 through B6 will contain original values held prior to the request, and the Q register will contain the length of the literal submitted. In general, the A register will contain one of the following status indications:

0000000000 Normal completion of request.

4100000000 Inappropriate function: the literal contains a control statement such as GO, SOURCE, LOAD, etc., other than the subset allowed by the system.

4200000000 Incorrect parameter: the submitted literal could not be unstrung properly, or incorrect function code, options, or specifications have been included.

4300000000 Unrecoverable subsystem error is encountered during processing of the requested function.

5000000000 Function is not performed due to lack of facilities or other system requirements.

The above register settings and status codes may be further modified depending upon the function requested.

The method of submitting control statements during execution of the task through use of the job control stream is detailed in 3.4.4.

2.5.3. Segment Call (LOAD$, LOADA$) Service Requests

Segmentation is the normal method of overlay utilized by the system. Segments or overlays are defined during the collection of a program by the Loader (see 4.6.4.1).

Segment calls can be performed by direct or indirect methods. The LOAD service request is the direct method for utilizing a segment. Upon submission of the LOAD request, an original copy of the segment is read from direct access storage. The indirect method provides for automatic loading of a segment referenced by a jump type command. For an indirect reference, the segment may be entered directly since the segment is in primary storage from a previous reference; when not in primary storage, the original copy is brought from direct access storage. Since segments exist in absolute form, a segment call involves only a single direct access reference with no modification of the code after it is read in.

- Format:

The format of direct segment call service requests is as follows:

Load (LOAD) requests that a referenced segment be loaded at this time with control returned immediately following the request:

LOAD$ƀlabel

where label is an external reference which is defined by an entry definition in the called segments.

The generated request parameter is as follows:

ENT*B7	LABEL
EXRN	20004

Load and activate (LOADA) requests that a referenced segment be loaded at this time with control given to the segment:

LOADA$ƀtag

where tag is any external reference which is defined by an entry definition in the called segment and to which control will be transferred.

The generated request parameter is as follows:

ENT*B7	LABEL
EXRN	20005

When a LOAD or LOADA request is made, all operational registers are preserved unless an error condition arises. If a direct access storage error occurs, control is returned following the request, and the direct access storage error status is shown in the A register. If the LOAD or LOADA cannot be completed because of erroneous information, a status of 4200000000 will be returned in the A register. When the indirect method is used, all operational registers are also preserved; if an error occurs, the routine will be aborted.

2.5.4. Subroutine Load (FETCH$) Service Request

A subroutine load requests the loading of a named absolute library program at a specified location. This function permits an operating task to control operation of absolute programs as an alternate to formal segmentation. The requester must have an existing allocation of primary storage to accommodate the absolute element. The subroutine will not be activated until a fragmentation request is made. The operating base and memory lockout protection associated with the fetched subroutine is a subset defined by the requester through activity registration.

- Format

The format of the call is as follows:

FETCH$b̸base address, name/version,library

- Specifications:

Base address is a primary storage address relative to the lower lock register in which the element will be loaded. Base address may be specified as a label, tag, or constant. However, the generated address or binary value must be a multiple of 100_8 from an assumed base of 0 to facilitate its registration.

Name/Version is the symbolic name and version, if any, in alphanumerics assigned as identification for the called element.

Library identifies the library in which the absolute element is contained on random access storage. The following specifications are valid:

SYS specifies that the absolute element is contained in the systems library.

JOB specifies that the absolute element is contained in the job library.

xxxxx Group library number, which may be four decimal or five octal digits, specifies the file number by which the group library is identified. Decimal numbers may range from 0 to 9999, and must be indicated by a D following; e.g., 9989D. Octal numbers may range from 0 to 77777.

Absence of specification implies that a search for the element will start with the job library and continue through any group libraries and the systems library or until the element is found.

The generated request parameter is as follows:

	EBJP*B7	N	
		BASE ADDRESS	
	NAME		Left justified in Fieldata form
	NAME		
	VERSION		
	LIBRARY		
N →	EXRN	20320	

- Method of Operation

Upon submission of the FETCH service request, OMEGA performs the following functions:

– Interprets the packet addressed in B7.

– Locates and reads into primary storage the named element from the specified library.

– Returns program control to the instruction following EXRN.

Upon return of program control to the requester, B1 through B6 will contain the original values held prior to the request. B7 will contain the address of the packet. The A register will contain one of the following status indications:

0000000000	Normal completion
4200000000	Incorrect parameters, address outside task area of packet unrecognizable. In this case, the requested function was not performed.
4300000000	Unrecoverable direct access storage error was encountered in performing the request. The function may or may not be complete.

The contents of the Q register will be destroyed.

2.5.5. Locate Table of Contents (FETCHL$) Service Request

This request is used to locate the table of contents (TOC) of any source, relocatable binary or absolute program on the system, job or group library. This function is used when a requester wishes to have information about a particular library program.

- Format

  The format of the call is as follows:

  FETCHL$b̸base address,name/version,library

- Specifications

  Base address is a primary storage address relative to the lower lock register of the nine word area where the TOC is to be deposited. Base address may be specified as a label, tag, or constant.

  Type of TOC indicates whether the TOC is for source code, relocatable binary, or an absolute element.

  Name/version is the symbolic name and version, if any, in alphanumerics assigned as identification to the element whose TOC is to be read.

  Library identifies the library in which the TOC is contained on random access storage. The following specifications are valid:

  SYS specifies that the TOC is contained in the systems library.

  JOB specifies that the TOC is contained in the job library.

  xxxxx Group library number, which may be four decimal or five octal digits, specifies the file number by which the group library is identified. Decimal numbers may range from 0 to 9999, and must be indicated by a D following; e.g., 9989D. Octal numbers may range from 0 to 77777.

  Absence of specification implies that a search for the TOC will start with the job library and continue through any group libraries and the systems library or until the TOC is found.

The generated request parameter is as follows:

	EBJP*B7	N	
	TYPE OF TOC	BASE ADDRESS	
	NAME		Left justified in Fieldata form
	NAME		
	VERSION		
	LIBRARY		
N →	EXRN	20322	

Type of TOC field; if this field is 0 it is assumed that a TOC to an absolute element is wanted, otherwise there has to be a specification. If the field is 01, TOC is for RB element; if 02, TOC is for absolute element (same as 00); if 03, TOC is for a source element.

- Method of Operation

Upon submission of the FETCHL service request, OMEGA performs the following functions:

– Interprets the packet addressed in B7.

– Locates and reads into primary storage the name TOC from the specified library.

– Returns program control to the instruction following EXRN.

Upon return of program control to the requester, B1 through B6 will contain the original values held prior to the request. B7 will contain the address of the packet. The Q register will contain the file code of the library where the TOC was found. The A register will contain one of the following status indications:

0000000000	Normal completion
4200000000	Incorrect parameter, specified TOC not in library.
4300000000	Unrecoverable system error was encountered in performing the request.

2.5.6. Common Subroutines

A common subroutine is an element that is cataloged, maintained, and controlled by OMEGA so that the element may be referenced by any operating task through the Executive Return (EXRN) service request mechanism. The use of common subroutines can increase the throughput of a multiprogrammed system by conserving both primary storage and direct access storage.

All requests for common subroutines are channeled through a resident element that established and maintains the linkages required to allow multiple tasks to utilize the same code. The common subroutines, along with a directory, are contained within the systems library on direct access storage; and a specific common subroutine is established in primary storage only when its services are required. Once a common subroutine has been loaded, the subroutine will be maintained as a resident element until such time as it is inactive and the primary storage assigned to it is required for another purpose.

When a common subroutine call is performed, the requesting activity will be suspended until control has been returned.

2.5.6.1. OPERATING ENVIRONMENT

The design and coding of a common subroutine must include consideration of the operating environment which will be established when the routine is activated.

The operating environment which is needed by a particular common subroutine is specified at systems generation time and will consist of the combination of conditions, one out of each set, that have been selected from the following:

- RE-ENTRANT

  NON RE-ENTRANT

- 15-BIT B REGISTERS

  18-BIT B REGISTERS

- NORMAL INDEXING

  DUAL INDEXING

- ROUTINE LOCK

  REQUESTER LOCK

  OPEN LOCK

2.5.6.2. ENTRANCE CONDITIONS

A common subroutine is supplied with necessary parameters contained by the values in the operational registers at the time that the EXRN service request is performed.

The manner in which a common subroutine accesses the parameter values is dependent upon the operating environment specified for the routine. A routine that operates with open lock and dual indexing must refer to the parameters through a Storage Module (SMOD) that is addressed by register B4. The values of the registers B1 through to B7, A and Q are contained in consecutive locations, in the order indicated, beginning at the word addressed by B4+5 through to B4+15. A routine that operates with lock limits will be activated with the parameters contained in the registers.

2.5.6.3. SPECIAL DATA TRANSFER (CMOVE$)

A CMOVE$ service request is provided which permits a common subroutine with lock limits to perform data transfers either to or from a requester.

- Format:

  The CMOVE request has the following general form:

  CMOVE$ƀ$v_0$,$v_1$

- Specifications:

  v_0 is the address of the data area within the common subroutine.

  v_1 is the size of the data area (if bit 14 is set to one, the transfer will be made from the common subroutine; if bit 14 is zero, the transfer will be made to the common subroutine).

  The generated service request is as follows:

ENT*B7	v_0
ENT*Q	v_1
EXRN	20007

- Method of Operation:

  This function requires that the data address with the requester program be contained in register B7 at the time that the common subroutine service request is performed.

  When program control is returned to the requester, the B and Q registers will be undisturbed, and the A register will contain one of the following status indicators:

  00000XXXXX Normal completion: XXXXX is the number of words transferred.

  4200000000 Transfer has not been performed because the transfer would have violated lock limitations of either or both elements.

  4400000000 Parameter error in the number of words to transfer

2.5.6.4. EXIT PROCEDURES (CEXIT$, CSWCT$, CABORT$)

When a common subroutine has completed the processing of a request, the subroutine must relinquish control in a manner which allows the system to update linkage information that is created when the request is initiated.

When a common subroutine exits, the A, Q, and B7 registers should contain any values that are to be conveyed to the requester. The system automatically transfers these registers to the original requester at the time that the system is reactivated.

The following is a description of the service requests and the resultant returns that are permitted for the common subroutines:

- CEXIT$

  The CEXIT operator is the normal means for exit.

  – Format:

    The CEXIT request has the following form:

    CEXIT$

    The generated service request is as follows:

ENT*B1	0
EXRN	05004

This return indicates that processing is completed and control should be returned to the requester.

- CSWCT$

  The CSWCT operator is a means for routing control to a point other than that of the normal exit.

  – Format:

    The CSWCT request has the following general form:

    CSWCT\$ƀv_0

  – Specifications:

    v_0 is the service call value of another common subroutine to which control will be switched. The generated service request is as follows:

ENT*B1	1
ENT*B2	v_0
EXRN	05004

This return allows requests to be channeled through a series of common routines without requiring the worker task to make a series of requests. When exiting from a common subroutine by the CSWCT operator to another common subroutine, the register settings which are set when entering the first common subroutine are the same register settings used when entering the second common subroutine.

- CABORT$

The CABORT operator is used to deallocate the current activity.

— Format:

The CABORT request has the following form:

CABORT$

The generated service request is as follows:

ENT*B1	2
EXRN	05004

This return is required by system elements which are controlled by the content supervisor and which cannot issue an ABORT CONTROL THREAD request directly.

2.5.6.5. NAME CONVENTIONS

The name of each common subroutine that is defined within the system is used to form an entry definition (EDEF) item within the system library. The EDEF permits referencing of a common subroutine by its symbolic name without concern for the system-created call value.

A naming convention has been established to eliminate conflicts between symbolic EDEF's which must be maintained within the system. A set of two characters has been reserved for use as the initial portion of names which are required as EDEF's in a particular group of elements. The characters which have been reserved for common subroutine names are D$.

2.5.6.6. COMMON SUBROUTINE REQUESTS

A common subroutine is requested by an Executive Return (EXRN) instruction that contains a service call value which uniquely identifies the particular routine.

The service call value for a specific common subroutine consists of the library number which was assigned at systems generation time plus the constant 50040.

In symbolic coding, the common subroutine name may be used as an operand on the EXRN instruction. An external reference (XREF) will be created at assembly time which will be satisfied by a system-created EDEF when the absolute program is formed by the system load routine.

Any parameters that are required by the common subroutine must be contained in the proper operational registers when the EXRN service call is executed.

2.5.7. SEND/RECEIVE

The SEND/RECEIVE mechanism provides the user with a means for storing and transferring limited data sets between independent tasks and/or activities. The mechanism also provides communication between consecutively executed tasks within a job stream or between asynchronous activities within a task.

2.5.7.1. THE (SEND$) SERVICE REQUEST

The SEND$ operator transfers data from a specified core area to executive mass storage. The executive mass storage is provided by OMEGA from the Input/Output Cooperative Library and is linked to the requesting job. The RECEIVE operator reverses this process, and transfers the stored data from executive mass storage to the requesting task/activity. Once data is transferred to the requesting task/activity, the data is purged from executive mass storage and cannot be received again. The RECEIVE operator can also be used by an operating task/activity to obtain data submitted by the PRAM control statement.

A 15-bit binary identity, supplied by the requester, is attached to each parameter that is stored in executive storage. This mechanism makes it possible for the user to establish different sets of parameters that can be received, to the exclusion of other parameter or data sets. The user must establish conventions for the use of identity numbers for communicating between different tasks/activities. The one binary identity that has special meaning is 77777. If a data set has any other identity, only that amount of data transmitted by one SEND operation can be received per RECEIVE request on a first in, first out basis. For example, if three SEND operations send 20 words each, all having the binary identity of 12345, the first RECEIVE operation having a binary identity of 12345 will get only the first 20 word data set that is sent and will not receive all of the 60 words. The 77777 identity specifies that all data sets with this 77777 identity will be received regardless of the number of SEND operators used to transmit the data. In the example above, all 60 words would have been transmitted by the RECEIVE operator, if the binary identity were 77777.

The maximum number of words which can be sent by any one SEND operation is 17777_8.

- Format:

  The SEND request has the following general form:

  SEND$ƀ$v_0$, v_1, v_2

- Specifications:

  v_0 is the base address of the data area, relative to the lower lock.

  v_1 is the number of consecutive core locations which are to be transferred to executive storage.

  v_2 is a binary identity number which is to be used when transferred data will be separated into different sets. The identity is limited to 15 bits.

  The generated service request is as follows:

ENT*B7	v_0
ENT*Q	v_1
ENT*A	v_2
EXRN	20304

  Operands v_1 and v_2 may be specified as constants or tags with k-designator and B register modification.

- Method of Operation

  Upon submission of a SEND service request, OMEGA performs the following functions:

  – Interprets and verifies parameters submitted through registers A, Q, and B7.

– Acquires sufficient direct access storage from the I/O Cooperative Library to fulfill the request, and transfers data to the acquired storage.

– Returns program control to the instruction following the EXRN instruction.

Upon return of program control, registers B1 through B6 will contain their original values; values in registers B7 and Q will be destroyed. The A register will contain one of the following status indications:

0000000000	Normal completion
4200000000	Incorrect parameters: all or part of the data to be stored lies outside of the primary storage limits assigned to task.

2.5.7.2. THE (RECEIVE$) SERVICE REQUEST

The RECEIVE$ operator transfers data from executive direct access storage to the requestor. This data must be sent previously to executive storage by SEND operators initiated through an independent or asynchronously executed activity or previously executed task. Each set of data transmitted by the SEND operator has a 15-bit binary identity making possible the selection of particular data or parameter sets to the exclusion of all other sets having different binary identities. The one binary identity with special meaning is 77777. Any single RECEIVE request for data with a binary identity other than 77777 will cause only that data transmitted by one SEND operation to be received. This will be on a first in, first out basis. Only data with the 15-bit binary identity contained in the A register at the time of the RECEIVE request will be transferred from executive storage. In order for data to be received at all, the corresponding binary identity must be attached by the SEND operator to the data set.

Data will be purged from executive storage as it is received, and the data cannot be received again. Upon return of control to the user, the A register will show the number of words actually received. If the receiving field is too small, the A register will be set to negative. The Q register will contain a count of the number of words of data left to be received. Data which is already received will be purged from executive storage and the next request for data having this binary identity will cause the remainder of the data to be received. If the binary identity in the A register at the time of the request is 77777, all of the data having this identity will be received, regardless of the number of SEND operations which have transmitted data to executive storage. This identity is used by OMEGA; worker programs should use the identity with caution. Data with this identity is purged between tasks. Binary identity 77776 is reserved for system usage.

- Format:

  The RECEIVE request has the following general form:

  RECEIVE$ƀ$v_0$, v_1, v_2

- Specifications:

  v_0 is the base address of the area where the data is to be deposited, and is relative to the lower lock.

  v_1 is the maximum number of consecutive primary storage locations reserved for the deposit area.

  v_2 is an identity number associated with the requested data, and attached by the SEND operator.

  The generated service request is as follows:

ENT*B7	v_0
ENT*Q	v_1
ENT*A	v_2
EXRN	20305

Operand v_0 may be specified as a label, tag, or constant with k-designator and B register modification. Operand v_1 and v_2 may be specified as constants or tags with k-designator and B register modification.

- Method of Operation:

Upon submission of the RECEIVE service request, OMEGA performs the following functions:

- Interprets and verifies parameters submitted through registers A, Q, and B7.
- Locates, by binary identity, the data to be transferred, and transmits data to the requesters deposit area. The transferred data is deleted from the system.
- Returns program control to the instruction immedately following the EXRN.

Upon return of program control to the register registers B1 through B6 will contain the values held prior to the RECEIVE request. The B7 register contents will be destroyed. The A register will contain one of the following status codes:

00000XXXXX	Normal completion of request: XXXXX represents the number of words actually received.
4200000000	Incorrect parameter: all or part of the deposit area is outside the primary storage limits assigned to the task.
4300000000	An unrecoverable subsystem error has been encountered in transferring data from executive storage to the requester.
44000XXXXX	Overflow condition: the requesters deposit area is not large enough to hold all of the data that should have been received for this binary identity. As much data as could be received was transferred and purged from executive storage. The lower 15-bits of the A register will show the number of words that are actually received. Subsequent RECEIVE requests will cause receipt of the remainder of the data. The Q register will show the number of words remaining to be received.
4500000000	No data is in executive storage for this particular binary identity.

2.5.8. DATE and TIME Operations

OMEGA provides date and time operations for both internal and external uses. Timing functions for activity control and for running programs at specific time of day are also provided.

2.5.8.1. THE DELAY (DELAY$) SERVICE REQUEST

The DELAY$ service request provides a method for deactivation of activities and task fragments for a specific time period. Upon completion of this delay time period, the activity or task fragment is then eligible for program control. The period, specified by the user, may range from one millisecond to a maximum of 24 hours. If the delay period is five minutes or more, the delay will be established on the day clock time chain, and reactivation will occur at the closest six-second interval to the time specified. For delays under five minutes, the real time clock chain will be used.

The operator for the delay service request is:

DELAY$b̸$v_0$

where v_0 is the delay counter, a number (octal, or decimal followed by a D) in one millisecond units.

The packet generated is:

ENT*Q	v_0
EXRN	04001

The delaying routine requesting the delay will be eligible for control only upon completion of the delay time. Subsequent return of control may be further delayed by higher priority activities.

Program control will be returned to the instruction immediately following the packet. The Q register will contain the time of day at which the activity was eligible for control and the time at which control was returned.

The A register and registers B1 through B7 will remain unaltered.

2.5.8.2. TIME OF DAY (TIMED$) SERVICE REQUEST

The TIMED$ service request supplies an edited time for external representation, such as the system log or console output, which time is supplied as ten Fieldata characters in the AQ register. The format is:

HH:MM:SSƀƀ

All characters will be Fieldata, including the two space characters (05).

The operator for the time of day request is:

TIMED$

The generated packet is:

EXRN	04002

Upon return of control to the instruction following the packet, the A register will contain five Fieldata characters in the form HH:MM, for use where seconds are not needed and the Q register will contain: SSƀƀ. Registers B1 through B7 will remain unaltered.

2.5.8.3. ELAPSED CENTRAL PROCESSOR TIME (TIMEL$) SERVICE REQUEST

The TIMEL$ service request produces an account of the amount of CPU time used by this task and previous tasks of this job, and the allowable time that the job stream has remaining, as specified originally by the job control statement.

The operator for the elapsed time request is:

TIMEL$

The packet generated is:

EXRN	04003

Upon return of control to the instruction following the packet, the A register will contain in binary form the number of real time clock updates that have been used by the job stream.

The Q register will contain the maximum number of updates allowed the job stream. Registers B1 through B7 will remain unaltered.

If time specified on the job control statement is exceeded, the console operator will be given the option of terminating the job stream.

2.5.8.4. DATE AND TIME (DATIM$) SERVICE REQUEST

The DATIM$ request will supply the current date and the time of day. The date supplied will be in Fieldata characters in the form YYDDD; for external representation the time of day will be a binary representation of the number of real time clock updates that have occurred since the previous midnight.

The operator for the date and time request is:

DATIM$

The packet generated is:

EXRN	05013

Upon return of control to the instruction following the packet, the A register will contain five Fieldata numeric characters in the form YYDDD representing the year and date. YY will be the last two digits of the year (range 00–99) and DDD will be the day of the year (range 001 through 366). The Q register will contain the time of day. Registers B1 through B7 will remain unaltered.

2.5.8.5. DATE AND QUANTUM TIME (TIMEQ$) SERVICE REQUEST

The TIMEQ$ request will supply the current date in the form YYDDD, and the current time in terms of quantums. A quantum time interval is defined at tape generation time, and is equal to a multiple of real time clock increments, or updates. The multiple must be a whole number.

The operator for the date and quantum time service request is:

TIMEQ$

The packet generated is:

EXRN	04005

Upon return of control to the instruction following the packet, the A register will contain five Fieldata numeric characters in the form YYDDD representing the year and date. YY will be the last two digits of the year and DDD will be the day numbered 001 through 366. The Q register will contain the time of day. Time of day will be the binary number of quantum units elapsed from the previous midnight. Registers B1 through B7 will remain unaltered.

2.5.8.6. DATE AND TIME OF YEAR (TIMEY$) SERVICE REQUEST

The TIMEY$ request will supply the current date and the time of year. The date supplied will be in the form YYDDD. The time of the year will be a binary representation of the number of 100 millisecond units since the start of the new year.

The operator for the date and time of year service request is:

TIMEY$

The packet generated is:

EXRN	04006

Upon return of control to the instruction following the packet, the A register will contain five Fieldata numeric characters in the form YYDDD representinf the year and date. YY will be the last two digits of the year and DDD will be the day numbered 001 through 366. The Q register will contain the time of year. Registers B1 through B7 will remain unaltered.

2.5.9. Logical Switches

A set of logical switches is maintained by the system for each job entering the system and the switches are referenced by each task within a job deck. This provides the user with a simple on/off parameter to be conveyed to each task, which can be set external to the program. The switches A through E are initially set by the JOB control statement, from which point they may be changed or tested dynamically by a task through service requests. The switches may be reset or altered by the GO control statement. The on/off condition of the switches is represented by binary 1 for on and binary 0 for off.

2.5.9.1. SET OFF (XOFF$) REQUEST

This operation will set all or any combinations of logical switches A through E to the off condition (binary 0) as established by the JOB or GO statement.

The operator for the set off request is as follows:

XOFF$ƀswitches

Switches are any combination of alphabetic Fieldata characters, A through E, with each letter representing the respective switch which is to be turned off.

The generated service request is:

ENT*Q*W($+1)*SKIP	
FIELDATA SWITCHES	
EXRN	20104

The Q register at request time contains the logical switch names, in Fieldata form, which are to be turned off. The switch letters A–E may be in any order. Registers B1 through B7 are unaltered upon execution of the XOFF service request, while A register will always be zero.

2.5.9.2. SET ON (XON$) REQUEST

The operation sets all or any combination of logical switches A through E to the on condition (binary 1) as established by the JOB or GO control statements.

The operator for the set on request is as follows:

XON$ƀswitches

Switches are any combination of alphabetic characters A through E, with each letter representing the respective switch which is to be turned on.

The generated service request is:

<table>
<tr><td colspan="2">ENT*Q*W($+1)*SKIP</td></tr>
<tr><td colspan="2">FIELDATA SWITCHES</td></tr>
<tr><td>EXRN</td><td>20105</td></tr>
</table>

The Q register, at execution time, contains the logical switch name, in Fieldata form, of switches which are to be turned on. The switch letters A–E may be in any order. Registers B1 through B7 are unaltered upon execution of the XON service request, while A registers will always be zero.

2.5.9.3. TEST SWITCHES (XTEST$) REQUEST

This operation supplies the current condition (on/off) of logical switches A through E.

The operator for the Test Switches is as follows:

XTEST$

The generated service request is:

EXRN	20106

Upon return of program control, registers B1 through B7, and Q will remain unaltered. The A register will contain the current state of switches A through E, in binary form, in bit positions 29 through 25, respectively. Bit positions 24 through 0 will be cleared to binary 0. For example, if switches B, D, and E are on, and switches A and C are off, bit positions 29 through 25 will be as follows: 01011.

2.5.10. Checkpoint/Restart Mechanism

The Checkpoint Restart mechanism of OMEGA provides the user with the ability to record and utilize rerun dumps as the means of recovery for batch programs. Checkpoint/Restart is a segmented secondary executive routine that is responsible for processing two service requests. The checkpoint function allows a batch program to record a complete description of the operational environment existing at a given point during execution. The restart function permits the user to re-initiate a run at any previously recorded checkpoint.

Checkpoint dump information is recorded on a tape file assigned by the user. This file may be either an independent file reserved for checkpoint use, or an output data file. The checkpoint routine does not reposition the output unit prior to recording the rerun information, and standard bypass sentinel items are used as header and trailer blocks. This permits recording of multiple rerun dumps on a single file and enables the user to detect rerun information that is interspersed with other data. A unique identity is included within the header block of each rerun dump so that the particular dump can be specified at restart time.

The checkpoint routine automatically records most of the information needed to reconstruct the operational environment at a particular time. The notable exceptions and the reasons for them, are presented in the following paragraphs.

The contents of temporary direct access storage files that are to be recorded must be specified by descriptor items with the checkpoint parameter table. In many cases the pertinent data areas are only a small percentage of the total area assigned. Automatic recording of all temporary direct access storage files would use significant amounts of time to transfer meaningless data.

The contents of direct access storage files that are cataloged within the Master File Directory (MFD) are never dumped. The structure of these files remains intact, and, therefore, re-assignment is all that is required at restart time.

No provision is made for set up of unit devices, such as card or paper tape readers and punches, and printers. The assignments are recorded, but the user is responsible for repositioning the input or output device.

2.5.10.1. CHECKPOINT (CKPT$) SERVICE REQUEST

Checkpoint is the secondary executive service function that is responsible for forming rerun dumps for batch programs.

- Format:

  The checkpoint operator is:

  CKPT\$ƀv_0

  The generated packet is:

ENT*B7	v_0
EXRN	21241

- Specification:

  v_0 – is the address of the user constructed parameter table to be used in conjunction with this request. The format and content of this table is presented in the following section.

2.5.10.2. CHECKPOINT PARAMETER TABLE

The checkpoint parameter table is of variable length. The minimum requirement is two words. The user must append an additional two word descriptor for each direct access storage area that is to be recorded.

The initial two words have the following format:

	29 15	14 0
0	0--------0	FC
1	ND	RA

Word 0 Upper — Zero

0 Lower — The file code of the tape unit upon which the checkpoint dump is to be recorded.

Word 1 Upper — The number of the two word mass storage descriptor items that are appended to this table.

1 Lower — The address to which control is to be transferred upon a restart from this checkpoint. If this location contains 77777, the restart address is the same as the checkpoint completion address.

Each mass storage area descriptor has the following form:

	29	15	14 0
0		FC	NW
1	LA		

Word 0 bit 29 — An indicator that can be set to 1 to specify that the file is optional. If the file is not assigned or if the file is an MFD file, the descriptor is bypassed without causing an error diagnostic.

bit 28 – bit 15 — The file code of a storage area that is assigned to the requester.

bit 14 – bit 0 — The number of words in the storage area. A value of zero indicates that this descriptor is to be bypassed. A value of 77777 indicates that all words up to the end-of-file are to be dumped.

Word 1 — The logical address of the first word of the data to be dumped.

2.5.10.3. CHECKPOINT STATUS CODES

When control is returned following a checkpoint request, the A register contains a value indicating the status.

VALUE	MEANING
0000000XXX	Normal completion. XXX represents the job number of the requester.
4100000000	Inappropriate function. This status is returned if the requester has an open PLR setting, or if communications devices are assigned.
4200000000	Parameter error. This status is returned if the checkpoint file is unassigned or assigned to a device other than a magnetic tape unit, or when the user supplied parameter table cannot be interpreted.
4300000000	Subsystem error. An unrecoverable hardware error was encountered by the checkpoint routine.
4400000000	Direct access storage I/O error on worker file.

2.5.10.4. DIAGNOSTIC MESSAGES

One of the following messages is submitted to the primary output stream and to the console printer for each checkpoint request (xxxxx represents the identity assigned to the checkpoint):

CHECKPOINT xxxxx COMPLETED

CHECKPOINT ABORTED DUE TO BAD REQUEST PACKET

CHECKPOINT xxxxx ABORTED DUE TO RANDOM STORAGE I/O ERROR

CHECKPOINT xxxxx ABORTED DUE TO END OF TAPE ERROR ON CHECKPOINT TAPE

2.5.10.5. RESTART ROUTINE

Restart is the secondary executive service function that is responsible for processing rerun dumps to re-initiate batch programs at the point where a checkpoint was taken.

The restart routine is activated by an unsolicited console entry.

- Format:

  The restart request is:

  RRb̸v_0b̸v_1b̸v_2 Ⓢ

- Specification:

  v_0 – is the identity of the checkpoint dump that is to be used. (The identity that was assigned by OMEGA at the time that the checkpoint was taken).

  v_1 – is the peripheral name that is used to assign a tape unit for the checkpoint file.

  v_2 – (optional) either U or N may be used to specify, respectively that the tape is in unnumbered or numbered block format. If no specification is given, numbered block format is assumed.

  H, M, or L specifies high, medium, or low density recording.

  X, Y, or Z specifies A, B, or C format to be used with UNISERVO 12/16 tape units.

  E specifies that errata cards are to be read in at restart.

2.5.10.6. RESTART STATUS MESSAGES

One of the following messages is printed on the console printer in response to a restart typein:

RESTART FROM CHECKPOINT xxxxx COMPLETED

RR TYPEIN IS INCORRECT

TAPE NOT AVAILABLE FOR RESTART

RESTART TAPE FORMAT ERROR

RESTART TAPE UNREADABLE

RESTART FROM CHECKPOINT xxxxx ABORTED DUE TO RANDOM STORAGE I/O ERROR

RESTART FROM CHECKPOINT xxxxx ABORTED DUE TO JOB LIBRARY OVERFLOW

RESTART FROM CHECKPOINT xxxxx ABORTED DUE TO TAPE FORMAT ERROR ON CHECKPOINT TAPE

RESTART FROM CHECKPOINT xxxxx ABORTED DUE TO SUBSYSTEM ERROR ON CHECKPOINT TAPE

RESTART FROM CHECKPOINT xxxxx ABORTED DUE TO END OF FILE ERROR ON CHECKPOINT TAPE

RESTART FROM CHECKPOINT xxxxx ABORTED DUE TO SUBSYSTEM ERROR ON TAPE FILE POSITIONING

RESTART FROM CHECKPOINT xxxxx ABORTED DUE TO SEND/RECEIVE ERROR

RESTART FROM CHECKPOINT xxxxx ABORTED DUE TO ERROR IN RESTORING CORE

xxxxx represents the identity assigned to the checkpoint.

2.5.10.7. TAPE MOUNTING DIRECTIVES

Unless a tape file has been cataloged in the Master File Directory, the file is referred to by its file code in the tape mounting directive.

2.5.10.8. TAPE CHANGING DIRECTIVES

If in the course of recording a checkpoint, the end of tape is reached, then the checkpoint is aborted, the tape is re-wound, and the following message is displayed on the console:

LBL FC C/U CHECKPOINT TAPE + MNT NEW TAPE

When the operator has replaced the tape, the checkpoint will be recorded on the new tape.

2.6. ACTIVITY CONTROL

An activity is established by definition of an operating task or another activity. The function allows a dynamic declaration of parallel parts of a task, thereby achieving a multiprogram environment within the task. Activity control is particularly appropriate and has been used in real time processing where activities are selected on the basis of the data (transaction) being processed. The batch program(s) may utilize activities to regain CPU control during input/output waits for instruction executions, thereby decreasing task turnaround time and utilizing available CPU time more efficiently.

Activities are established at execution time through the use of fragmentation service requests by the operating task. Three forms of fragmentation requests are provided by OMEGA.

- Standard activity is normally used to register and activate an independent program or subprogram. Once the program is activated, little or no communication or synchronization exists between the requester and the requested activity.

- Queue process activity provides a method for controlling access to an independent subprogram or process. This form of fragmentation request is generally useful where the subprogram or process is to be performed serially or is non re-entrant due to complexity of code or data (e.g. tables, files, buffers). Only one point in a queue processing activity is eligible for control although other queued references exist.

- Fork and Join – A FORK service request provides the user with a method for establishing two points of program control, and the ability to synchronize a completion point of the two, through use of the JOIN service request.

A fragmentation service request implicitly requires the allocation of additional primary storage to serve as the activity addendum necessary to execute the requested fragment. Once established, an activity or fork may make the same service requests as a task, and will share with the task operational identity, primary co-operative streams, facility allocation, logging and accounting.

2.6.1. Standard Activity Registration (REG$) Service Request

The REG$ service request defines a point of program control within a task which is to be registered with the OMEGA dispatcher. Once the service request is executed by the task, the registered activity is eligible for program control. Optional parameters may specify the data area to be locked in, the response priority, and the abnormal index register setting. The activity may voluntarily terminate through use of the RETURN$, RETURN1$, ERROR$, or ABORT$ service requests. At this time, the point of program control and associated activity addendum will be deallocated and purged from the system.

- Format:

  The standard activity registration operator has the following general form:

  REG$b̸ $v_0, v_1, v_2, v_3, v_4, v_5$

- Specifications:

v_0 defines the address of the activity in primary storage relative to the lower lock or RIR setting of the requester, dependent upon v_1. Specification v_0 is the implicit starting point of the activity when initially given program control. At the initiation time of the registered activity, all operational registers are set to the address of the requester at registration time.

v_1 is the activity mode indicator:

one (1) mode specifies that the activity is an entity and contains all referenced data exclusive of the declared data area, and that the activity has been independently compiled and collected so that the first instruction is relative to address 0. In this case, the RIR will be set to v_0, which is assumed relative to the lower lock.

Zero (0) mode indicates that the activity is integral to the requester (collected as part of the requester task through the Loader) and that the RIR is set to the address of the requesting activity. Specification v_0 is assumed relative to the RIR.

v_2 is the relative response priority (0 through 17_8) which may be declared to attain a differentiation with respect to other activities within the task (see 2.7.2). When priority is not specified, or is specified as 0, service priority of the requesting task is used. Priority can be changed to any value between 5 and 37 for a batch job; all priorities are legal for a real time job.

v_3 specifies the starting primary storage address of the data area which is to be utilized by the activity, and is optional. The address is relative to the lower lock limit of the requester.

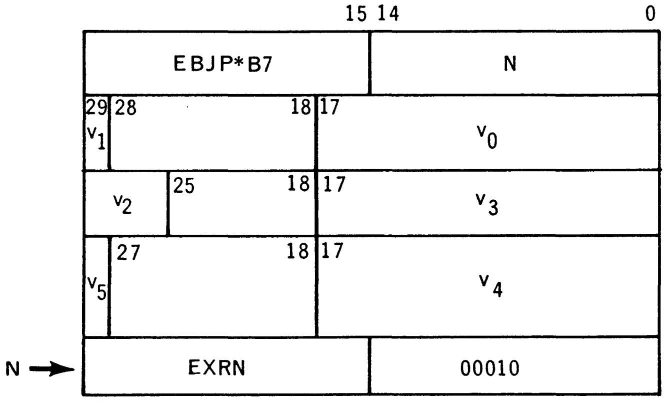
v_4 specifies length of data area and is required when v_3 is given.

v_5 is the data area mode. Bits 29 and 28 will cause the generation of the desired IFR setting.

v_5	15B	15/17B	RIR	RIR/PLR	GUARD	WRITE	READ	U(IFR)
00*	X		X		X	X	X	02100
10*		X		X	X	X		16300
01		X	X		X	X	X	06100
11	X			X	X	X		12300

*SPURT will generate these options in the REG$ operator

The packet generated is:



Operands v_0, v_3, and v_4 may be specified as labels, tags, or constants. However, generated addresses or binary values must be multiples of 100_8 from the assumed base of 0 in order to set the hardware address properly. Operands v_1, v_2 and v_5 are specified as binary constants or as tags, defining constants.

- Method of Operation

Upon submission of the REG$ service request, OMEGA performs the following functions:

– Interprets the packet address by B7 and forms the activity addendum for the requested fragment; sets the program register as determined by the packet.

– Places the formed activity addendum on CPU queue, at which point the activity addendum is eligible for program control if the requesting activity is interrupted because of activity time out or because of high priority activity regaining program control upon occurrence of an I/O interrupt.

– Returns program control to the instruction following EXRN under the activity addendum of the requesting activity.

Upon return of program control to the requesting activity, registers B1 through B6, and the Q register will contain original values held prior to the request. B7 will contain the address of the registration packet. The A register will contain one of the following status indications:

0000000000	Normal completion.
4200000000	Incorrect parameters: address outside task area or packet unrecognizable.

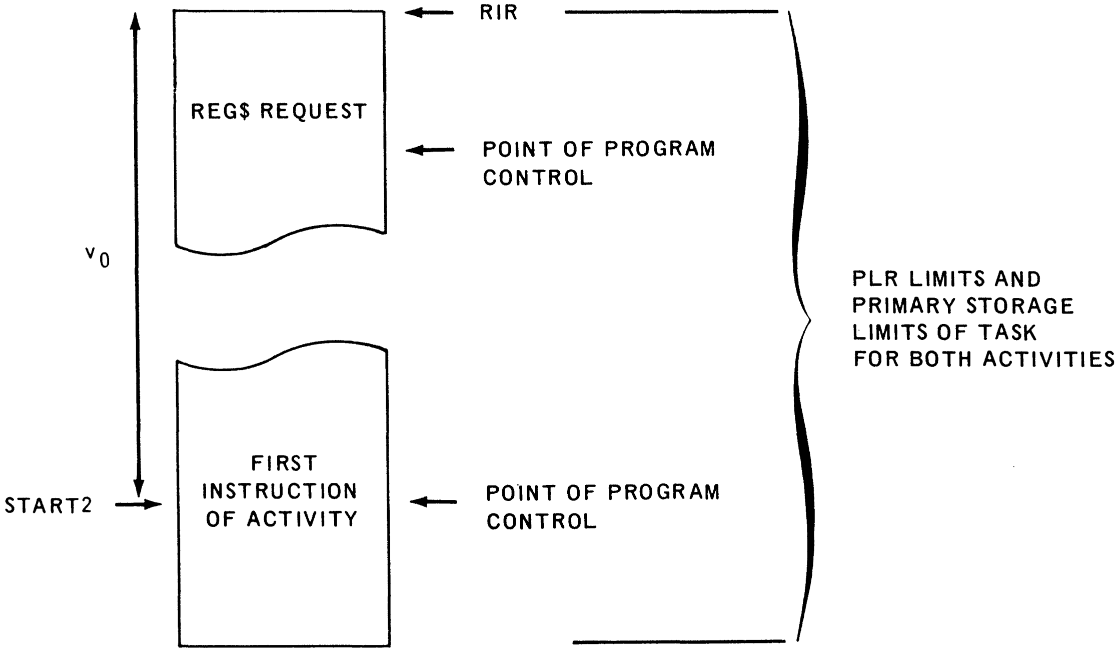
When a task in initially activated by selection, the task is under standard activity addendum with: the RIR set to the base of the task primary storage assigned, the PLR set to allow access to all of primary storage assigned to the task, and B registers set to 15-bit mode relative to the RIR.

- Examples:

Example 1

REG$ƀSTART2

This operator will cause the activation of a second activity at location START2. Lack of specifications v_1 through v_5 indicates use of the PLR and RIR of the requester with no abnormal constraints on the registered activity. The following illustration of task primary storage assignment shows eligible points of program control and program register settings:


RIR
REG$ REQUEST
POINT OF PROGRAM CONTROL
v_0
PLR LIMITS AND PRIMARY STORAGE LIMITS OF TASK FOR BOTH ACTIVITIES
START2
FIRST INSTRUCTION OF ACTIVITY
POINT OF PROGRAM CONTROL

Both activities are free to access all primary storage assigned to the task or to make service requests, utilizing input/output or other assignment modes, to the task. Once an activity is registered, it is unpredictable (dependent upon dispatcher queue) as to which activity may have program control at any one point in time. Therefore, access to common data areas or code must be controlled by the task code through design (truly independent subprograms in which parameters are contained in registers), or by the use of test and set instructions to control access to common code or data.

Example 2

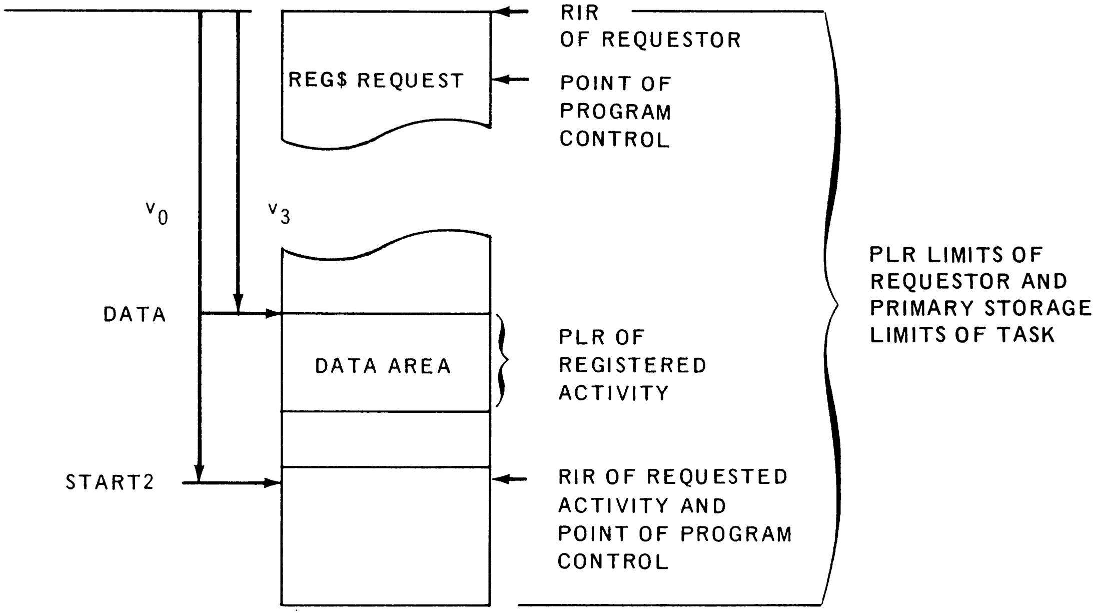
REG$ƀSTART2, 1,0,DATA, 100,1

This operator will cause the activation of a second activity whose code has the following constraints:

- The activity must be individually compiled and collected as an absolute element relative to 0, and brought into primary storage by the FETCH$ service request or some similar function.

- The activity must be a re-entrant subroutine which will not write or store within the program code. Data area used to store or write will be reached through use of registers B4 through B7 in an 18 bit mode relative to the PLR lower.

The following illustration of task primary storage assignment shows eligible points of program control and register settings:



2.6.2. Queue Process Activity Registration and Activation

Queue processing of an activity is a means for utilizing a task-permanent activity to respond to a series of events. Transactions are accepted and queued by the system, and the activity is executed whenever a queue entry exists and the activity is dormant. The activity signals completion for a given transaction by return of control through the return operator. OMEGA re-activates the activity if transactions remain on the queue.

Use of this function allows the scheduling of events at the time of occurrence. The function is appropriate where no advantage can be gained from registration of concurrent executions by re-entrant code. Two operators are associated with use of this function. The first defines the activity; the second supplies the data to be queued, and causes activation of the fragment.

2.6.2.1. QUEUE PROCESS ACTIVITY REGISTRATION (REGQ$) SERVICE REQUEST

The REGQ$ service request defines the activity to be registered.

- Format

The queue process registration operator has the following general form:

REGQ$b v_0, v_1, v_2, v_3

- Specifications

v_0 defines the address of the activity in primary storage relative to the lower lock or RIR setting of the requesting activity. v_0 is the implicit starting point of the activity when activated through QREF.

v_1 is the activity mode indicator:

One (1) mode specifies that the activity is an entity, and has been independently compiled and collected so that the first instruction is relative to address 0. In this case, the RIR will be set to v_0 each time that a QREF activates the registered processing routine.

Zero (0) mode indicates that the activity is integral to the requester collected as part of the requester task (through the Loader) and that the RIR is set to that of the requesting activity. In either case, mode 1 or 0, the program lock register and B register mode will be set to the mode of the activity making the QREF and not to that of the activity performing the registration.

v_2 is the relative response priority (0 through 17_8) which may be declared to attain a differentiation with respect to other activities within the task (see 2.7.2). When priority is not specified or is specified as 0, the current service priority of the task is used. (The priority created by this parameter cannot be 0 – 4 for a batch job, all priorities are legal for real time jobs).

v_3 contains a binary identity of the activity to be used on all references to the established activity.

The packet generated is:

	EBJP*B7		N
	29 v_1	28 18	17 v_0
	v_2	25 12	11 v_3
N →	EXRN		00012

Operand v_0 may be specified as a label, tag, or constant. The generated address must be a multiple of 100_8 from an assumed base of 0 if v_1 is 1 in order to set the RIR properly; v_1, v_2, and v_3 are specified as binary constants or as tags defining constants; v_3 is limited to four octal characters (0–7777).

Upon submission of the REGQ$ service request, OMEGA performs the following functions:

– Interprets the packet address by B7 and forms the activity addendum for the requested fragments. Set up of RIR, identity, and priority are determined by the packet.

– Links the formed activity addendum to the task addendum as a latent activity to be activated upon QREF.

– Returns program control to the instruction following EXRN under the activity addendum of the registering activity.

Upon return of program control to the requesting activity, registers B1 through B6, and the Q register, will contain original values held prior to the request. B7 will contain the address of the registration packet. The A register will contain one of the following status indications:

0000000000 Normal completion.

4200000000 Incorrect parameter: v_0 is outside the task area or not a multiple of 100_8; the binary identity has been previously assigned; or the packet is unreadable, or an illegal priority was generated by the value specified in v_2.

The registered activity will remain dormant until referenced by a QREF request at which time the activity will be placed on the CPU queue and become eligible for program control. Registration of queue process activity will remain task permanent until: all activity addendums have executed a RETURN service request; an activity has given up program control through an ABORT or ERROR service request; or, an explicit request to deallocate queue process activity is given by the RETURN1 service request.

2.6.2.2. ACTIVATE QUEUE PROCESS ACTIVITY REGISTRATION (QREF$) SERVICE REQUEST

The QREF$ service request causes a specified queue process activity to be activated.

- Format:

The queue process activity reference operator has the following form:

QREF$ƀ$v_0$

- Specifications:

v_0 contains binary identity of the queued activity as defined by the REGQ operator (v_3).

Queue activity reference represents the mechanism for queuing requests to previously defined activities registered by the REGQ operator. All parameters to be conveyed to the queued activity will be contained in the operational registers A, Q, and B1 through B6, at the time of reference. The lock register and B register mode will be set to that of the requester when the registered activity is initiated. B7 will contain the binary identity.

The packet generated is:

ENT *B7	v_0
EXRN	00002

Upon submission of the QREF service request, OMEGA performs the following functions:

– Locates the activity addendum specified by the binary identity.

– Forms a queue packet for the activity containing the requester's register values B1 through B7, A, and Q, together with the requester's PLR and index register mode. The packet is queued to the addendum on a first in, first out basis. If the queue process activity is dormant, the packet will be registered on the CPU queue for activation.

– Returns program control to the instruction following EXRN under the activity addendum submitting the QREF. Control will be returned to the requesting activity before control is given to the referenced activity unless the referenced activity is of a higher priority.

Upon return of program control to the referencing activity, registers B1 through B6, and Q will contain the original values held prior to the request; B7 will contain the binary identity. The A register will contain one of the following status indicators:

0000000000	Normal completion
4200000000	Incorrect parameter: v_0 specifies binary identity which is not registered or has been terminated.

- Example

The following sample coding illustrates a tape to drum routine utilizing two queue process activities as follows:

Activity 1 reads data from magnetic tape in the form of blocks and activates Activity 2 to process data through use of QREF. Activity 1 assumes that a tape was assigned to file code A and has been mounted. Upon each activation, B1 contains the address of a buffer adequate to hold the largest tape block.

Activity 2 records data submitted by Activity 1 on direct access storage and returns the buffer to Activity 1 upon completion through use of QREF. Activity 2 assumes an assignment of direct access storage to file code B and will request extensions when exhausted. Upon each activation, B1 contains the address of the buffer in which data is contained, B2 shows the number of words of data.

The routine utilizes two buffers in order to achieve overlap of reading from tape and recording on drum. Note the processing activities code would be the same if one or N buffers were used.

For simplicity of description L(ERR) is set to nonzero when nonrecoverable error is reached. U(ERR) is set to nonzero when end-of-file or end-of-reel is reached on tape, or additional direct access storage cannot be attained.

Sample Code

Initialisation Routine

```
START     REGQ$ƀACT1,0,0.1     → Register tape process routine

          REGQ$ƀACT2,0,0,2     → Register drum process routine

          ENT*B1*BUFF1         → Activate tape process routine

          QREF$ƀ1              → With buffer

          ENT*B1*BUFF2         → Queue second buffer to tape

          QREF$ƀ1              → Process routine

          RETURN$              → End of initialization

BUFF1     RESERVE*10000        →

BUFF2     RESERVE*10000        →
```

Tape Process Routine

```
ACT 1       ENT*A*W(ERR)*AZERO    → Abnormal completion of end of processing

            RETURN$               → Yes

            STR*B1*L($+3)         → Store buffer base in packet


            EBJP*B7*$+3
            06*LENGTH             → Tape packet

            0*0

            EXRN*READ

            ENT*B2*A              → Save number of words

            JP*ABNORM*ANEG        → Abnormal completion

            QREF$ƀ2               → Queue drum routine

            RETURN$

ABNORM      LSH*A*6               → Check status code

            SEL*XL*X77770

            SUB*A*4*AZERO         → End-of-file or end-or-reel

            JP*END

            PRINT$ƀERR1,2,1       → Submit error message to primary

            STR*B0*CPL(ERR)       → Output

            ERROR$

END         STR*B0*CPU(ERR)

            RETURN$

ERR1        FD*2*TAPE ERROR

ERR         0*0                   → Abnormal indicator

READ        EQUALS*10001

WRITE       EQUALS*10002
```

Drum Process Routine

```
ACT 2       ENT*A*W(ERR)*AZERO        → Abnormal end condition

            RETURN$                   → Yes

            STR*B1*L($+4)             → Store buffer base in packet

            STR*B2"L($+2)             → Store number of words in packet

WRT         EBJP*B7*$+4

            07*0                      → Drum packet

            0*0

LINCR       0*0

            EXRN*WRITE

            JP*ABNORM1*ANEG           → Abnormal completion

            RPL*A+Y*W(LINCR)          → No update drum increment

            QREF$*1                   → Queue read routine

            RETURN$

ABNORM1     LSH*A*6                   → Check status

            SEL*CL*X77744*ANOT        → End of drum assignment

            JP*ASG                    → Yes

            PRINT$ƀERR2,2,1           → Submit error message to primary

            STR*B0*CPL(ERR)           → Output

            ERROR$

ASG         ASGƀMƀDRUM,B,5000/10000

            JP*WRT*APOS               → Additional direct access storage assigned

            PRINT$ƀERR3,4,1           → No

            STR*B0*CPU(ERR)

            ERROR$

ERR2        FD*2*DRUM ERROR

ERR3        FD*4*INSUFFICIENT DRUMƀƀƀ →
```

2.6.3. Fork and Join Mechanism

The FORK and JOIN service requests provide a method of activity registration in which OMEGA correlates all fragments from a given level. Completion of the asynchronous activities from a level can be tested through the use of the JOIN function.

2.6.3.1. FORK (FORK$) SERVICE REQUEST

The FORK operator is more applicable to general batch processing programs than the real time transactions, since the operator provides a higher level interface, and synchronization is on a gross basis. An activity established by a form may, in turn, establish other forks which provide additional levels of controlling parallel paths. All FORK activities are considered integral to the requester. They attain the same RIR, lock register value, and index mode of the requester. A queue processing activity may not execute a fork.

- Format:

  The FORK operator has the following general form:

  `FORK$ƀ`v_0

- Specification:

  v_0 is the starting address of the requested activity relative to the lower lock of the requesting activity and may be specified as a label, tag, constant, or B register modification with any of the read k-designators.

  The generated packet is:

EN T*B7	v_0
EXRN	00013

Upon return of program control to the requesting activity, registers B1 through B6 and the Q register will contain original values held prior to the request. B7 will contain the address of the forked activity. The A register will contain one of the following status indicators:

0000000000	Normal completion.
4200000000	Incorrect parameter: Fork address is outside program limits; forking attempted from a queue processing activity.

Upon submission of a FORK service request, OMEGA performs the following functions:

– Forms an activity addendum for the requested fragment. Operational registers B1 through B6, A, Q, PLR, RIR, and the index mode of the forked activity are set to those of the requester.

– Places the forked activity addendum on the CPU queue, at which time the activity is eligible for program control under the following conditions: (a) if the requesting activity is interrupted because of self-invoked activity time-out, and (b) if a high priority activity regains program control due to input/output interrupt.

– Returns program control to the instruction following EXRN under the activity addendum of the requesting activity.

2.6.3.2. JOIN (JOIN$) SERVICE REQUEST

The JOIN entry requests a wait until all asynchronous activities previously established by FORK have been completed as indicated by the task completion operator RETURN. A JOIN requests completion of all activities directly emanating from the requesting activity. A JOIN request given by the originating task activity, itself, will wait on all fork requests outstanding within the task since the original task activity is the base of all forking. A JOIN request by subsequent activities will wait on only those activities established either directly by the activity or indirectly by forks from activities which are themselves direct forks from the requester. No parameters are required for this operation.

- Format:

The JOIN operator has the following general form:

JOIN$

The generated packet is:

EXRN	00014

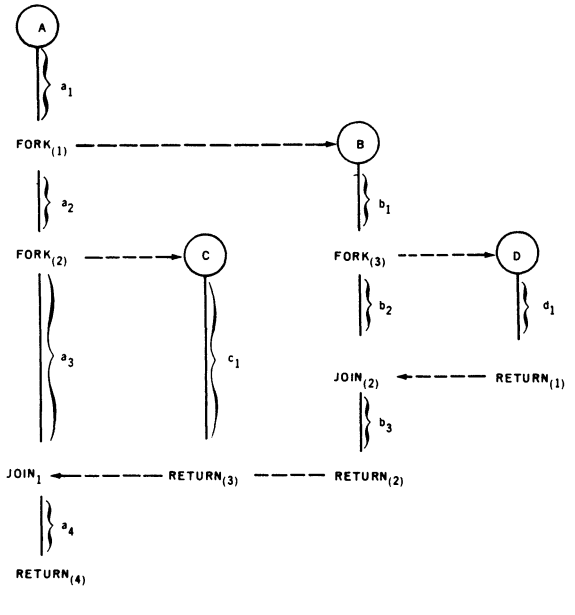
Upon submission of the JOIN operator, OMEGA checks for any activity addenda which still exist and which were formed by the FORK operator under current activity addendum. If none exists, program control will be returned to the activity submitting the JOIN request.

If activity addenda do exist, the current activity will be held dormant until all linked activities have been completed. In this case, a check for reactivation of dormant activity will be made each time that an activity under current task makes a RETURN service request.

Upon return of program control to the requesting activity, registers B1 through B7, A, and Q will contain original values held prior to the JOIN request. Program control will be given to the primary storage cell immediately following the JOIN instruction.

- Example

The following instruction shows the sequence of three forks executed from a base activity A causing the activation of activities B, C, and D, and the subsequent points of synchronization. The code within activities is described by $a_1, a_2 \ldots n_m$, and the point at which the code is eligible for execution is shown by (- - -)



In the preceding illustration, coding contained in a_4 will not be executed until coding contained in a_1, a_3 is completed, and activities B, C, and D have executed a RETURN.

As can be seen, OMEGA exercises synchronization for completion of activities formed through the FORK operator; but, by definition, each activity, once on the CPU queue, may operate in parallel with other activities within the task. Therefore, no form of synchronization is imposed by OMEGA on an activity once it is activated. Each activity can access input/output devices and primary storage assigned to the task or perform any OMEGA service requests. Therefore, the sequence for execution of the FORM, JOIN, or RETURN service requests varies between tasks, and could vary between executions of a particular task. The factors causing variance are as follows:

– The amount of CPU time required to complete the code in (n_m), including input/output waits and input/output error recovery procedures.

– The number of tasks currently operating in the multiprogram environment, causing rotation of activities and a switch of control to higher priority programs.

The following chart illustrates an assumed sequence of service requests emanating from the illustrated task and the code eligible for program control:

SEQUENCE IN TIME (EVENTS)	SERVICE REQUEST	REQUESTING ACTIVITY	CODE ELIGIBLE FOR CONTROL
1	task initiated	OMEGA	a_1
2	$\text{FORK}_{(1)}$	A	a_2, b_1
3	$\text{FORK}_{(2)}$	A	a_2, b_1, c_1
4	$\text{FORK}_{(3)}$	B	a_3, b_2, c_1, d_1
5	$\text{JOIN}_{(1)}$	A	b_2, c_1, d_1
6	$\text{JOIN}_{(2)}$	B	c_1, d_1
7	$\text{RETURN}_{(1)}$	D	b_3, c_1
8	$\text{RETURN}_{(2)}$	B	c_1
9	$\text{RETURN}_{(3)}$	C	a_4
10	$\text{RETURN}_{(4)}$	A	task terminated

The preceding is an assumed sequence, and, dependent upon the variance factors, the sequence could have been as follows:

SEQUENCE IN TIME (EVENTS)	SERVICE REQUEST	REQUESTING ACTIVITY	CODE ELIGIBLE FOR CONTROL
1	task initiated	OMEGA	a_1
2	FORK$_{(1)}$	A	a_2, b_1
4	FORK$_{(3)}$	B	a_2, b_2, d_1
7	RETURN$_{(1)}$	D	a_2, b_2
6	JOIN$_{(2)}$	B	a_2, b_3
8	RETURN$_{(2)}$	B	a_2
3	FORK$_{(2)}$	A	a_3, c_1
9	RETURN$_{(3)}$	C	a_3
5	JOIN$_{(3)}$	A	a_4
10	RETURN$_{(4)}$	A	task terminated

The two charts show that events 1, 2 and 10 are fixed. However, once 2 is executed, either 3 or 4 are eligible; once 4 is executed, either 7, 6 or 3 is eligible etc.; and the user should not rely upon the sequence.

2.6.4. Termination Service Requests

The following service requests are the formal means for termination of an activity, task, or job.

2.6.4.1. THE RETURN (RETURN$) SERVICE REQUEST

The Return is used to relinquish control to the executive (RETURN$) Service at the conclusion of an activity or task. Since the request may have no outstanding business which would cause reactivation (with the exception of queued activity registration), the associated task is checked. If no outstanding activities, forks, incomplete hardware level I/O requests or latent time-of-day restart requests exist, the task is terminated. Subsequent to task termination, facilities are deallocated and the next job task is sequenced, or the job is terminated if no tasks remain; otherwise, control is switched to some other task/activity active.in the multiprogram environment. No parameters are required for the RETURN operator.

- Format

The operator has the following form:

RETURN$

The generated request is:

EXRN	00005

2.6.4.2. THE ABORT (ABORT$) SERVICE REQUEST

The ABORT$ Service Request causes voluntary release of the CPU by the operating task/activity. The system purges all references to the task, including outstanding I/O and service requests, with the exception of primary and secondary output which will be processed in the normal manner to the point of ABORT. The entry is an indication of abnormal operation and implies that the entire job be terminated. No parameters are required for the ABORT operator.

- Format:

  The operator has the following form:

  ABORT$

  The generated request is:

EXRN	20502

2.6.4.3. THE ERROR (ERROR$) SERVICE REQUEST

The ERROR$ Service Request is the same as ABORT except that only the associated task is terminated, and processing will continue with the next task in the input stream. No parameters are required for the ERROR operator.

- Format:

  The operator has the following form:

  ERROR$

  The generated request is:

EXRN	20503

2.6.4.4. THE RETURN 1 (RETURN1$) SERVICE REQUEST

The RETURN1$ Service Request is the same as RETURN except that, given by an activity that was registered as a queue process activity, the request will cause the activity's deallocation along with all outstanding QREF's associated with the activity. No parameters are required for the RETURN 1 operator.

- Format:

  The operator has the following form:

  RETURN1$

  The generated request is:

EXRN	00004

2.7. CENTRAL PROCESSOR CONTROL

Central processor control consists of those elements of OMEGA which affect and control the multiprogram system environment of the UNIVAC 494. The elements determine which activity registered with the system will be given program control. Once determined, program control is returned to the activity at the point of previous interrupt. Central processor control is composed of the dispatcher and the CPU queue function.

RESPONSE PRIORITY	OPERATING PRIORITY	ACTIVITY FUNCTION
17	17–17 = 0	TELPAK* buffer clearing
16	17–16 = 1	Staging TELPAK data to drum
10	17–10 = 7	Low speed buffer clearing triggered by communications interrupt
7	17–7 = 10	Staging low speed data to drum
6	17–6 = 11	Output transmission control
0	17–0 = 17	Processing functions

*Registered Trademark of American Telephone and Telegraph Corporation.

Response priorities may range from 0–17_8.

2.8. PRIMARY STORAGE ALLOCATION

The function of primary storage allocation control is to maintain the availability and status of all assignable primary storage associated with the UNIVAC 494 system. To perform this function, OMEGA utilizes chaining techniques, and allocates or releases primary storage upon demand via a service request.

OMEGA maintains chains of available primary storage for four distinct purposes (see 10.3). Description and links of available core storage within any one chain is held within the free storage which the chain describes.

2.8.1. Task Primary Storage Allocation

Primary storage allocated for any one task is always contiguous. The relative index register (RIR) allows the program to operate in any contiguous area in a 262K memory. All primary storage assigned to a task element and extensions are in multiple units of 100_8 words, thereby allowing the Program Lock-in Register (PLR) to be effective for the total primary storage assigned to one task. Although restrictions are placed on the instruction execution area (as explained in 2.8.4), the total primary storage assigned to a task is not restricted to $32{,}768_{10}$ words.

The initial primary storage limits for a task program are determined at selection time of the task. These limits are specified in the preamble of the task element and in any optional CORE statements used to extend the initial allocation. The preamble primary storage limits of the task code are determined by the Loader at collection time. The optional CORE statement may be collected with the task element or the statement may be contained in the control stream.

Once activated, the task program may dynamically expand or contract primary storage assigned to the program by a service request to the operating system. Additional allocations of primary storage are always added to and/or released from the end of the task element. Since primary storage allocation for an operating task is always contiguous, dynamic expansion requests should be minimized, as either compaction of primary storage or roll-out of a low priority task is often required to maintain the contiguous area.

The RIR makes it feasible for the operating system to dynamically relocate program elements. By a contiguous area of physical core storage can be made available for selection of a task or expansion of an operating task compacting or re-ordering of programs. Compacting is only performed when necessary, since it requires that the task(s) to be moved is temporarily stable. Stability implies that all I/O transfers into the program code, common subroutine, or service requests are complete.

2.8.2. Task Primary Storage Extensions

Four Executive services are provided by OMEGA for control of additional assignments of contiguous primary storage to the task element. External primary storage extensions may be made through the CORE statement submitted at task selection time to obtain optional primary storage without invoking compaction procedures. Internal primary storage extensions may be made through the memory add (MADD) service request, which permits dynamic expansion of primary storage assigned to the task. An internal primary storage release mechanism, using the memory release (MREL) service request, permits dynamic contraction of the assigned primary storage. The test primary storage limits (TCORE) service request allows the operating task to obtain its current primary storage assignment as made by the foregoing requests.

2.8.2.1. EXTERNAL PRIMARY STORAGE EXTENSIONS (CORE STATEMENT)

External primary storage extensions are the preferred ways for allocating additional primary storage to the task; and may be obtained by either or both of two methods. The first method is the inclusion of the CORE statement with the Loader's secondary language. The first character control symbol (#) must be changed to a blank (see 4.6).

The Loader will include the image with the absolute element, and the image will be processed by OMEGA. The second method is inclusion of the assign image as part of the primary input control language. When the CORE statement is part of the input control language, the control symbol # is the first character. The statement is included in the input job stream at any place after the END statement for the preceding task (after the JOB statement if there is no preceding task), and before the activation statement.

If both methods are used, i.e. an assignment image is included with the collection and an image is included in the job stream, OMEGA will add the minimum/maximum fields to compute the total additional primary storage required for assignment to the task.

- Format:

  The CORE statement has the following general form:

  #COREƀoptionƀminimum/maximum

- Options:

  R – This usage of #CORE is the declaration of an expansion area for a real time job. The primary storage is not to be assigned to the task; use of the storage will be assigned to tasks which may be rolled out to satisfy a MADD$ request.

  → L – Total core assignment from all sources should not exceed 32K.

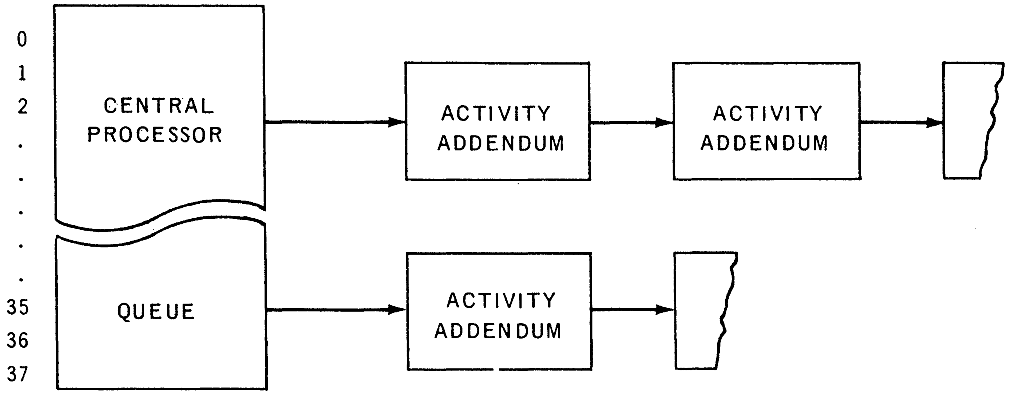
- Specifications:

  Minimum/maximum specifies the minimum and maximum number of words by which primary storage assigned to the task element will be extended. The values may be given as an octal or decimal pair of numbers (XXXX/YYYY implies octal and XXXXD/YYYYD implies decimal). Minimum specifies the smallest number of words acceptable for an extension which satisfies the request; if this value is not available at selection time, no extension will be made. Maximum is the largest number of words desired to satisfy the request. The maximum, or a part thereof, will be assigned at request time, if available, in multiples of 100_8 words. When a fixed amount of primary storage is desired, only the minimum field need be specified. If the R option is present minimum specifies the number of words by which the real time task may extend task primary storage limits. The maximum specification is not used with the R option.

2.7.1. The Dispatcher and the Central Processor Queue

The dispatcher is the basic OMEGA element responsible for controlling the CPU queue which contains activity addendums eligible for return of program control. The dispatcher allocates CPU time among the active task/activities in the multiprogram environment. A simple, efficient algorithm provides for ordering of activities by potential input/output dependence; that is, compute-limited activities operate within the wait of I/O limited activities. In this way, both the peripheral devices and the CPU are efficiently utilized. This algorithm may be altered by two additional factors: (1) the operating priority of the activity and, (2) the rotation mechanism imposed upon compute-bound activities.

The CPU queue is essentially a table of 40_8 positions used by the dispatcher to determine which activity will gain program control. A position is reserved in the table for each of the operating priorities. Activity addendums eligible for program control are linked by chaining techniques to the appropriate position in the table as determined by the operating priority. CPU queue can be illustrated as follows:



Through use of chaining, any number of activities may be registered on the CPU queue for return of program control.

The dispatcher, when taking an activity off the queue for return of program control, will determine the highest priority position containing an entry on the CPU queue (0 through 37 represents priorities high to low, respectively), and with return program control to the first activity chained to the position. When an activity is placed on queue, activity is entered at the end of the chain on any one priority position. This assures a first in, first out sequence for program control within any one priority level.

Switching between activities or linking of an activity to the CPU queue occurs in the following four cases:

- An activity addendum is formed by OMEGA to initiate a task; or by the user through the FORK and REG fragmentation service requests; or as the result of a QREF to an activity which is currently dormant. In the latter case, the activity addendum is linked to the CPU queue and switching does not occur.

- On occurrence of an input/output interrupt, a latent activity is entered on the CPU queue at the appropriate priority. Program control is normally returned to the interrupted activity. Since a switch is never made to operate an activity of equal priority, a real time activity of the highest priority is assured of a logically noninterruptable cycle which experiences only minimum physical interruption.

- On rotation, since input/output dependence for a given task and associated activities may change irregularly. A rotation is imposed by the system on any one activity utilizing the CPU for one time quantum. Rotation effects an automatic switch whereby the current activity, if having program control for more than one time quantum, will be requeued to the CPU queue and the dispatcher will locate a new activity to which control may be returned. The rotation value may be altered at system's generation time. (MAX = 200 millisecs; MIN = 200 microsecs.)

- On the occurrence of a POP service routine by an OMEGA control element. The routine generally occurs as the result of a service request which could not be performed due to the requested OMEGA element being non-re-entrant and currently processing a request, or requiring the use of a common list or table which is currently in use. When conflicts of this type occur, OMEGA places the activity and its associated addendum on a dormant chain through the PUSH service request until such time as the list or routine is free. The POP is the inverse of PUSH and places the dormant activity addendum back on the CPU queue when the element or list is free.

2.7.2. Operating Priority

Communications equipment processing may require an arbitrary precedence over all other processing. Similarly, nonproductive programs should be suppressed to the lowest priority to avoid conflict with productive processing. The operating priority defines the basic position of each activity scanned by the dispatcher.

The operating priority has a range of 0 to 37_8 positions in which a given activity can be registered. Zero is the highest operating priority which an activity can attain, indicating logical nonstop processing. The lowest priority level which an activity can attain is 37_8 operating priority, and is used normally for nonproductive routines.

Operating priority of an activity is a combination of service priority, high, medium, or low, as specified on the task control card, and of optional response prioirity given through activity registration.

Service priority as originally specified or implied on the task control statement is used as operational priority during the select/initiate phase and the start of the task. The value is conveyed to selection by options letters H, M, and L contained on the GO statement. All implied task activation statements, such as SPURT, COB, and LOAD, are assumed to contain an M option. Therefore, service priority is both selection priority (in case of equal task candidates during select phase) and the initial operating priority (when the task is given program control). High, medium, or low service priority is normally used for the following:

- High (operational priority 17_8) is used for real time processing routines where throughput or turnaround time constraints are necessary due to equipment type or application.
- Medium (operational priority 27_8) is the normal service priority assigned to batch programs and system routines (LOAD, SPURT, FOR, and others).
- Low (operational priority 37_8) is used for nonproductive routines and background jobs.

Response priority to balance utilization and control of peripheral devices may require a differential priority for activities within a task. Response priority simply allows a task to spread its activities over a continuum of priority whose endpoint or low priority is the service priority. The submission of an I/O request by an I/O cooperative, or buffer clearing for communications input, are typical uses of response priority.

When response priority is specified at activity registration time, the priority specification is considered as a negative increment to the service priority associated with the task. A real time task with service priority of 17_8, for example, may establish several levels of response priority.

The following chart illustrates priority levels in the system:

2.8.2.2. INTERNAL PRIMARY STORAGE EXTENSIONS (MADD$)

The (MADD$) service request allows the operating task to dynamically expand primary storage assigned to, and addressable by, the task. If primary storage is not available, contiguous to the end of the requesting task element, compaction and roll-out procedures will be invoked only for real time tasks which declared an expansion area via #CORE with the R option. Compaction and roll-out procedures, action causes temporary suspension of the task and possibly other concurrent tasks. The service request will not be honoured if primary storage is not available in the configuration or if obtaining the primary storage would cause suspension of a higher priority program.

The internal request for additional primary storage must be used with care by the task that has many activities registered. The additional primary storage assigned will be added to the task limits; and the PLR for the requesting activity will be modified, if possible, to enable addressing the additional primary storage. All other activities that have been registered by the task and which have PLR limits equal to that of the requesting activity will be modified to reflect the additonal primary storage.

- Format:

  The service request has the following general form:

  MADD$ƀmaximum/minimum

- Specifications:

  Maximum, minimum specify the maximum and minimum number of words of storage to add to the current primary storage assigned to the task. Additional allocations will be made only in multiples of 100_8.

  The generated service request is:

ENT*Q	MAXIMUM
ENT*A	MINIMUM
EXRN	20703

Upon completion of the requested extension, program control will be returned to the requesting activity with one of the following status indications in the A register:

00000 00000 Successful completion. The Q register will indicate the number of words of primary storage acquired.

41000 00000 Unsuccessful completion. The number of words is not a multiple of 100_8, or primary storage from the end of the task to the end of the program chain is insufficient for satisfying the request.

42000 00000 Incorrect parameter. The requesting activity does not have access to all primary storage assigned to the task.

50000 00000 Primary storage is not available. This status is returned to a batch program if the extension would require the roll out of some other task. The status is returned to a real time program if the request still cannot be satisfied with roll-out and compaction procedures.

2.8.2.3. INTERNAL CORE RELEASE (MREL)

The memory release (MREL) service request allows the operating task to dynamically release core assigned to it. The core is released from the upper end (high addresses) of the task allocation and is returned to the program chain for use in running other operating tasks.

- Format:

The MREL service request has the following general form:

MREL$b̸number of words

- Specifications:

Number of words specifies the quantity of words to be released from the end of primary storage currently assigned to the task element. Primary storage will be released in multiples of 100_8 only.

The generated service request is:

ENT*Q	NUMBER OF WORDS
EXRN	20704

After verifying that the release request is valid, OMEGA will modify the program lock limits of all activities registered under the requester's task to reflect the change in program limits.

Upon completion of the requested release, program control will be returned to the requesting activity with one of the following status indications in the A register.

0000000000 Successful completion: The requested amount of primary storage has been released from the end of the task element and the PLR has been adjusted to exclude the released primary storage.

4100000000 Inappropriate function (normally a program error): the requested amount of primary storage to be released exceeds the amount of primary storage assigned; the MREL service request is being executed from the primary storage being released. Some activity registered under the task has its upper lock limit within the area being released; the activity making the request does not have access to the primary storage being released, or some activities have outstanding service requests that will return program control to a point within the released primary storage.

2.8.2.4. TEST PRIMARY STORAGE AREA/LIMITS (TCORE)

The Test Core limits (TCORE) service request supplies the requester with information concerning the amount of primary storage, including task coding and supplementary allotments, that has been assigned to the task at run time. The TCORE request provides the requesting activity with its location in primary storage (RIR value), the task program's lower lock limit (PLR value), and the number of contiguous 100_8 – or 64_{10} – word groups assigned to the task.

- Format:

The TCORE operator has the following form:

TCORE$

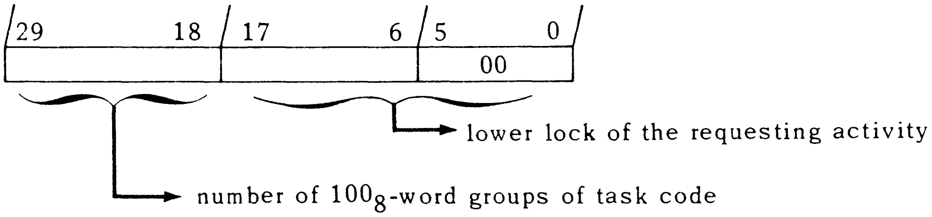
No operands are required for the TCORE request.

The generated service request is:

EXRN	20117

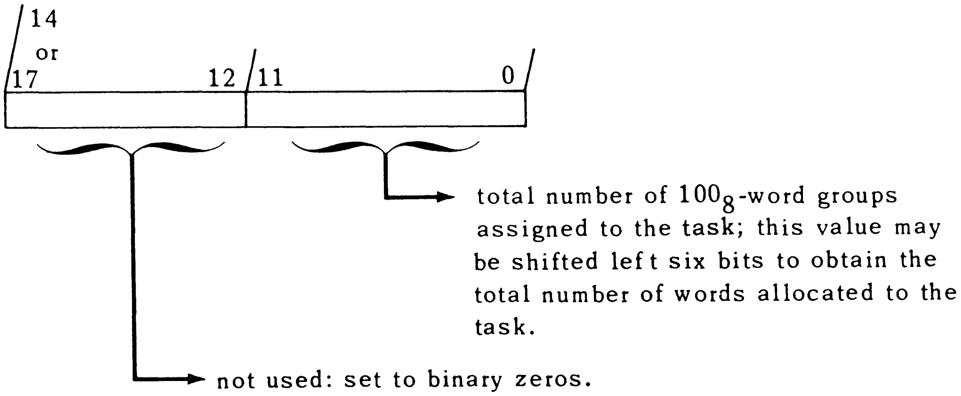
Upon return of program control to the requesting task, the following values will be contained in registers A, Q, and B7.

A will contain the lower lock limit (PLR value) of the requesting activity and the number of 100_8 – word groups of task code. This length of task code does not include the primary storage assigned from CORE statements and internal primary storage extensions. The lower lock is always a multiple of 100_8; hence, the lower six bits of this value will be zero. The format of the A register is:


29
18
17
6
5
0
00
lower lock of the requesting activity
number of 100_8-word groups of task code

Q will contain the RIR value of the requesting activity. The format of the RIR value is the same as that for the lower lock.

B7 will contain the total number of contiguous 100_8- word groups assigned to the task. This value includes the length of the task code plus the primary storage assigned from any CORE statement and/or internal primary storage extensions. The register format is:


14
or
17
12
11
0
total number of 100_8-word groups assigned to the task; this value may be shifted left six bits to obtain the total number of words allocated to the task.
not used: set to binary zeros.

Registers B1 through B6 will contain the original values held prior to the request.

Because of the ability of a task to fragment itself into multiple activities with modified primary storage limits, the primary storage limits supplied are those of the requesting activity. Hence, primary storage limits of the task should be requested before any fragmentation is done.

2.8.3. Worker Program Considerations for Primary Storage Compaction

To make efficient use of program primary storage, the primary storage compaction routine relocates programs at the discretion of OMEGA. Through compaction, elements of a program are established in contiguous areas of primary storage by elimination of blank areas and removal of extraneous elements from between the desired program elements. For compaction, temporary suspension of concurrent tasks is required, and a worker program may be rolled out. The task suspension and resumption, and the rolling out and rolling in of programs, are normal functions of OMEGA, and usually occur without concern to or knowledge by the programmer.

Some worker programs may be designed to pick up and save program RIR values in order to modify packets for input/output and other functions, in which cases the orderly compaction process by OMEGA is interfered with, and system operations may be affected. Programs of this kind may be used with either of alternative procedures. The first procedure is to inhibit compaction of the task/program through the TCORE1$ service request; the other procedure is to have OMEGA make the modification for the task, based upon the RIR bias at the time of the service request.

2.8.3.1. INHIBITING COMPACTION (TCORE1$)

The TCORE1$ service request may be used by the programmer to specify that a program/task be made not subject to compaction. The service request will also furnish the primary storage limits for the task.

The TCORE1 service request has the following form:

TCORE1$

No specifications are required for the service request.

The generated packet is:

EXRN	20134

Upon return of program control to the requesting task, the following values will be contained in registers A, Q, and B7:

A	Bits 29–18	Number of 100_8, word groups of task code
	Bits 17–0	Lower lock of the requesting activity
Q		RIR value of the requesting activity
B7		Total number of continuous 100_8 word groups assigned to the task

In addition, a bit is set in the task addendum which inhibits compaction. Caution should be exercised in use of TCORE1$ since compaction may terminate the task in the event that the task interferes with a MADD$ request by a real time job.

2.8.3.2. PACKET GENERATOR

The packet generator is activated through an EXRN*60000 call, using the A register, or an EXRN*60001 call, using the Q register. The generator function will cause OMEGA to modify parameters in the storage module based upon information in the requester's packet. After modification, the request is processed in the same manner as a normal service request.

- Parameter area format:

The packet for the generator is as follows:

A_0	0 0 0 X X			EXRN value
A_1	RIR/PLR	F.C.	B_b	Y
A_2	RIR/PLR	F.C.	B_b	Y
A_3	RIR /PLR	F.C.	B_b	Y
A_n	RIR/PLR	F.C.	B_b	Y

Word A_0 Bits 29 – 21 Set to 0.

Bits 20–15 XX represents the number of parameter words exclusive of A_0.

Bits 14–0 EXRN value represents the number of the EXRN which is to be simulated, that is the number of the EXRN call which is to be made upon completion of the generation function.

Words A_1–A_n Bits 29–27 RIR/PLR represents the modification indicator:

0 Y + B_b is to be modified by the RIR value from the storage module of the requester.

1 Y + B_b is to be modified by the PLR value from the storage module of the requester.

2 Y + B_b needs no modification by the RIR or PLR values.

Bits 26–21 F.C. represents the function code:

00 Y + B_b + RIR/PLR is formed and saved in storage.

01 The word in working storage is transferred to Y + B_b + RIR/PLR as defined in the current parameter word.

02 The B register indicated in the parameter word is modified by RIR/PLR.

03 The word in working storage is transferred to the indicated B field in the requester's storage module.

Bits 20–15 B_b represents the register specification that is to be used:

01–07 B registers, B_1–B_7, respectively.

10 A register

11 Q register.

Bits 14–0 Y represents the address (operand) portion of the instruction, assumed relative to the RIR.

- Generator call format:

The call for the packet generator has the following form:

ENT*A	v_0
EXRN	60000

or

ENT*Q	v_0
EXRN	60001

v_0 represents the address of the parameter packet:

$v_0 = A_0$ if relative to the RIR.

$v_0 = 40000*A_0$ if relative to the PLR.

Upon completion of the generation function, control is returned to the requester, with the status indicated in the A register:

4200000000 Bad information is found, for example, illegal function code or B register field, or violation of PLR limits.

XXXXXXXXXX Any other return that may come from the specified EXRN.

- Example

The following sample coding shows usage of the generator function, with input/output by a routine with open lock limits:

```
ENT*B7*IOP    'Unmodified packet address

ENT*A*PP      'Parameter packet address relative to the RIR

EXRN*60000

    :

IOP06*306     'File A, 306 words

0*BUFF        'Buffer address

  714         'Logical increment

PP3*10002     '3 word packet, write call

  0*BUFF      'BUFF + RIR to working storage

  100*IOP + 1 'Working storage to IOP + 1 + RIR

 207*0        'Modify B7 in SMOD (storage module) by RIR
```

In the sample, the packet generator will modify the address and buffer base of the I/O packet by the RIR value of the job, and the write function is submitted to OMEGA.

2.8.4. Register Index Modes

The minimum memory primary storage size available on the UNIVAC 494 is $65,536_{10}$ words. The memory size is expandable in 65K increments to $262,144_{10}$ words maximum. With 15-bit index registers, only $32,768_{10}$ words of continuous primary storage can be addressed. A 17-bit address field is necessary for accessing all of the locations in the 131K primary storage range. An 18-bit address value is needed for accessing the maximum primary storage range, i.e., beyond 131K. Since the address portion of an instruction is 15 bits, register modes have been defined which permit all of the B registers to operate as 15-bit index registers, or some B registers to operate as 17 or 18-bit registers while the others continue in the 15-bit mode. The index register mode is controlled by bit 26, the B length designator, of the Internal Function Register (IFR). When the B length designator is set to 1, registers B4, B5, B6 and B7 of the executive register set or the worker register set, whichever is active, will operate as 17 or 18-bit registers, providing access to all of primary storage. Registers B1, B2, and B3 will continue to operate as 15-bit registers. When the B length designator is set to 0, all of the registers operate as 15-bit registers.

On each primary storage reference, the UNIVAC 494 normally adds the contents of the RIR to the operand of the instruction being executed (see 2.8.5.1 for details of relative indexing). However, another indexing mode, dual indexing, is provided which uses the lower lock limit of the PLR as the index base in lieu of the RIR value. Through activity registration (as explained in 2.6 Activity Control), a worker program might fragment its code and assign modified program lock limits to different activities. With the dual index mode, registers B4, B5, B6 and B7 provide modifications based on the lower limit of the PLR while B0 (no modification), B1, B2, and B3 continue to provide modification based on the RIR. The dual index mode is used in executing drum-based OMEGA routines in which a base is selected from free primary storage and used as the RIR, and the lower limit of the PLR is set to 0 to allow communication between the drum-based routines and the resident OMEGA system. The dual index mode is controlled by bit 27, the lower lock designator, of the IFR. When the lower lock designator is set to 1, registers B4, B5, B6, and B7 of the active set operate relative to the base of lower lock register value while B0, B1, B2 and B3 continue to operative relative to the RIR.

OMEGA provides four operators, SET15$, SET17$, SETD$, and CLD$, in addition to the options available with activity registration, which define and set the operational mode of the index registers. These operators allow the worker program to specify and change the operational mode for the requesting activity.

2.8.4.1. SET 15-BIT B REGISTERS (SET15$)

The SET15$ operator sets all index registers to the 15-bit mode.

- Format:

The SET15 operator has the following form:

SET15$

No operands are required by the SET15 request.

The generated service request is:

EXRN	20120

This request will set bit 26 of the IFR to 0 in the requesting activity's storage module (SMOD). Upon return of control to the requesting activity, all B registers will operate in the 15-bit mode. If this operation is used in a queue processing activity, only the operation of the current processing mode will be changed.

2.8.4.2. SET 17-BIT B REGISTERS (SET17$)

The SET 17 operator sets registers B4, B5, B6, and B7 to the 17-bit mode, or, if greater than primary storage is used, to the 18-bit mode. Registers B1, B2, and B3 continue in the 15-bit mode.

- Format:

The SET17 operator has the following form:

SET17$

No operands are required for the SET17 statement.

The generated service request is:

EXRN	20121

This request will set bit 26 in the IFR to 1 in the requesting activity's SMOD. Upon return of control to the requesting activity, registers B4, B5, B6 and B7 will operate in the 17 or 18 bit mode. If the operator is used in a queue processing activity, only the operating mode of the current QREF will be affected.

2.8.4.3. SET DUAL INDEX MODE (SETD$)

The SETD$ operator sets the dual index mode for the requesting activity. All primary storage references with operand modification by B4, B5, B6 or B7 will be relative to the lower lock limit of the PLR. This operator will cause no change in B length.

- Format:

The SETD operator has the following form:

SETD$

No operands are required for the SETD statement.

The generated service request is:

EXRN	20127

This request will set bit 27 of the IFR in the requesting activity's SMOD. Upon return of control, the requesting activity will operate in the dual index mode, that is primary storage references with modifications of B4, B5, B6 or B7 will be relative to the lower lock limit of the PLR. If the operator is used in a queue processing activity, only the operating mode of the current QREF will be affected.

2.8.4.4. CLEAR DUAL INDEX MODE (CLD$)

The CLD$ operator clears the dual index mode for the requesting activity. All primary storage references will be relative to the RIR. The operator will cause no change in B length.

- Format:

The CLD operator has the following form:

CLD$

No operands are required with the CLD statement.

The generated service request is:

EXRN	20130

This request will clear bit 27 of the IFR in the requesting activity's SMOD. Upon return of control, primary storage referencd will be relative to the RIR. If this operator is used by a queue processing activity, only the operating mode of the current QREF will be affected.

2.8.5. Program Considerations

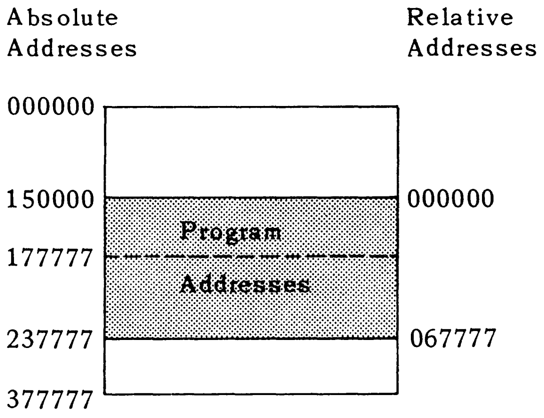
This section discusses the functions of the Relative Index Register (RIR), Program Lock-in Register (PLR) and B registers, and their contribution to memory addressing.

2.8.5.1. THE RELATIVE INDEX REGISTER (RIR)

The primary function of the Relative Index Register (RIR) in the UNIVAC 494 system is to provide OMEGA with the ability to load or relocate any absolute program in primary storage without instruction modification. When all of the separate elements of a program have been collected and allocated (relative to zero), the program may be loaded and executed at any primary storage address ending in 00. This eliminates the need for modification at load time.

The RIR also permits OMEGA to move programs already loaded and to compact primary storage, making larger contiguous areas available for use. A complex multiprogram environment, with tasks being selected and terminated, creates continuously changing demands on primary storage usage, and many small areas of discontinuous available primary storage may be formed. The RIR allows OMEGA to compact and make the most efficient use of the available primary storage.

As the worker program executes its instructions, which have been collected and allocated relative to zero, the RIR acts as a bias for all primary storage references. Although worker programs are restricted to $32{,}768_{10}$ continuous instructions, the programs are not restricted to location in one storage bank. The RIR permits a program to cross the program boundaries and to operate as though primary storage were loaded at absolute address zero. The example below shows a worker program that is $70{,}000_8$ words in length which has been loaded at address $150{,}000_8$. The program has absolute address 150,000 through 237,777 and relative addresses 000000 through 067,777.



- Format:

The RIR is a hardware register which uses 17 or 18-bits as shown below:

29 18	17 6	5 0
NOT USED	RELATIVE INDEX	000000

The lower six bits (0–5) are equal to 00; the succeeding 12 bits (6–17) specify the address; and the upper 12 bits (18–29) are not used.

The RIR is loaded by the ERIR *(Enter RIR)* instruction or, in conjunction with loading the IFR, by using the EIFR *(Enter IFR)* instruction. Both instructions are privileged and are used only by OMEGA.

2.8.5.2. THE PROGRAM LOCK-IN REGISTER (PLR)

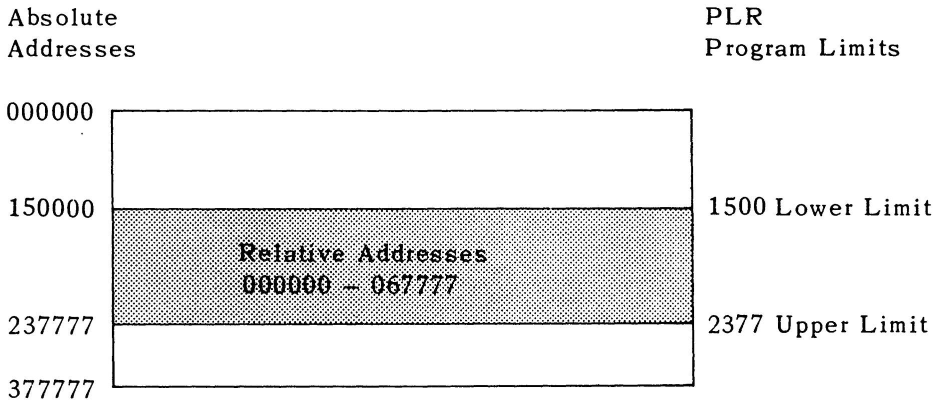
The primary function of the Program Lock-in Register (PLR) is to provide OMEGA with the ability to set limits for worker and system programs beyond which the programs are prevented from reading or storing information. The PLR protects OMEGA from destruction by untested worker programs and prevents unwanted interference among the concurrently operating worker programs. Through the PLR, and by software verification of service requests, untested programs may be included in the mix of real time and batch programs.

- Format:

The PLR is a hardware register which uses 22 bits as shown below:

29 26	25 15	14 11	10 0
0000	UPPER LIMIT	0000	LOWER LIMIT

The 11-bit fields specify the limits of the program in increments of 100_8. The lower limit is specified in bits 0–10 and the upper limit is specified in bits 15–25. As shown in the example below, the program of $70,000_8$ words in length which is loaded at location $150,000_8$ (see 2.8.5.1) would have a lower limit set at 1500 and an upper limit set at 2377, allowing the program to reference 150,000 as the lowest absolute address and 237,777 as the highest absolute address.



The PLR is loaded by the EPLR *(Enter PLR)* instruction, which is privileged, and is used only by OMEGA. Before giving control to any program OMEGA will load the PLR. Any attempt to violate the limits will be captured by a fault interrupt.

2.8.5.3. PRIMARY STORAGE ADDRESSING

A variety of index register lengths and index modes are available on the UNIVAC 494 for primary storage addressing. The following discussion will clarify some aspects of primary storage access. For the discussion, the following definitions are made:

- B_b is the index register (B register) specified in bits 15 – 17 of an instruction word.
- y is the lower 15 bits (0 – 14) of an instruction word.
- $\overline{y}$ is the sum formed by the addition of y to the contents of an index register ($\overline{y} = y + B_b$).
- Y is an absolute primary storage address consisting of y plus a relative index value.

In the normal single index mode of operation, primary storage is referenced using the contents of the RIR. The UNIVAC 494 first forms a relative index by adding y and B_b. The absolute address is then formed by the addition of $y + B_b$ + RIR. The addition forming $y + B_b$ is either a 15-, 17-, or 18-bit addition with an end-around carry: overflow from the highest order bit is brought around and added to the lowest order bit of the partial sum. The y-designator of the instruction word is always 15-bits, obtained from the lower half of the word. B_b is either 15, 17 or 18 bits depending upon the B register selected in the instruction word and the B length designator, bit 26, in the Internal Function Register (IFR). If bit 26 is 0, all B registers are 15 bits in length. If the bit is 1, registers B1, B2 and B3 are 15 bits in length while B4, B5, B6, and B7 are 17 or 18 bits, dependent upon the primary storage range.

When B_b is 15 bits, the addition of $y + B_b$ is the same as in the UNIVAC 490: address 00000 cannot be generated unless y and B_b are both 00000. When y and B_b are complements of each other, the sum of the addition is 77777. Some examples (in octal notation) of 15 bit $y + B_b$ addition follow:

y	12345	y	00005	y	00001	y	77774
B_b	35610	B_b	77772	B_b	77777	B_b	00005
$\bar{y}$	50155	$\bar{y}$	77777	$\bar{y}$	00001	$\bar{y}$	00002

When B_b is 17 or 18 bits, the addition of $y + B_b$ adds a 15-bit and a 17– or 18-bit number; thus y, the 15-bit number, can never appear as negative. Some examples (in octal notation) of 17-bit $y + B_b$ addition are:

y	00005	y	00001	y	77774
B_b	377772	B_b	377777	B_b	000005
$\bar{y}$	377777	$\bar{y}$	000001	$\bar{y}$	100001

After $y + B_b$ is formed, a second addition is used to form $y + B_b$ + RIR. The RIR value is a 17 or 18-bit value with the lower six bits always equal to 00; so, the lower six bits of the absolute address are determined by $y + B_b$. The forming of RIR + B_b + y, or RIR + $\bar{y}$, is an additive process with no end-around carry.

Some examples (in octal notation) of RIR + y addition are:

RIR	000700	RIR	377700	RIR	200000
$\bar{y}$	300105	$\bar{y}$	000100	$\bar{y}$	377777
Y	301005	Y	000000	Y	177777

A number of programs require that the instructions be located in one area of primary storage and that data be located in a different area. An example is the common subroutine concept which permits the subroutine to be located in primary storage only once, but with the ability to operate on data in all parts of primary storage. For handling this type of program more efficiently, two relative index values are needed: one value for instruction references and another for operand (or data) references. The dual index mode is a means for obtaining the relative index values, using the contents of the RIR for biasing the instructions and the lower limit (LL) of the PLR, because the PLR is set to enclose the data area, for references to the data area. When using the LL, the absolute address for an operand, Y, is formed by $y + B_b$ + LL.

Bit 27 in the IFR activates the dual relative index mode. When the bit is 0, all primary storage references will be made relative to the RIR. When the bit is 1, primary storage references will be made relative to the RIR or the LL according to the following rules:

B_b	Y
0	y + RIR
1	y + B1 + RIR
2	y + B2 + RIR
3	y + B3 + RIR
4	y + B4 + LL
5	y + B5 + LL
6	y + B6 + LL
7	y + B7 + LL

NOTE: Transfers of data into and out of 17- or 18-bit index registers may be either full words or half words. For full word transfer, the lower 17 or 18 bit value is transferred out of the register into the location. When half words are transferred into the 17 or 18 bit register, the upper two or three bits are zero. Half word transfers out of a 17 or 18 bit register will result in the loss of the upper two or three bits.

The *STORE B* worker instruction uses full words of memory; that is, the complete 30-bit location is altered, although only 15-bit registers may be involved.

When entering a register with an operand having a k-designator of 4, the sign extension is performed of bit 14 of y + B_b regardless of the length of B_b. For example, if B6 is a 17 bit register, and contains the value 301010, the instruction 11046·00000 will enter the A register with the value 00000·01010.

When operating, the UNIVAC 494 in the dual index mode, and the RIR differs from the lower limit of the PLR, care must be taken in use of the *Enter B and Jump* instruction with index registers B4 – B7. The relative address captured by this instruction is, in all cases, the absolute primary storage address minus the value of the RIR. Therefore, mnemonic statements (such as READ$, WRITE$, etc.) which cause the generation of the *Enter B7 and Jump* instruction may not be used since the packet address captured is relative to the RIR and not to the PLR lower limit. Also, the exit from a subroutine entered through an *Enter B (4 – 7) and Jump* instruction cannot be made to the point of entry by executing a *Jump to B (4 – 7)* since the return point is relative to the RIR, and the exit jump is relative to the PLR lower limit.

2.9. PROGRAM CONTINGENCIES

A program contingency is defined for this system as an occurrence within an operating program which makes the continued operation of that program impossible without exception processing. The contingency occurs as the result of improper instruction execution and usually indicates an error or bug in the operating program.

OMEGA permits an operating task to specify routines for processing program contingencies by establishing the starting point of an error routine as the address to which program control is to be returned if a program contingency occurs. In the absence of error routine specification by the user, OMEGA will take certain predetermined actions of its own.

The user error routines are given information concerning the contingency. The routines may then inspect the circumstances and take the action desired, such as, return to the point of interrupt, terminate the activity causing the interrupt, terminate the task, or terminate the entire job.

2.9.1. Contingency Types and Interrupts

Four hardware interrupts are currently classified in OMEGA as program contingency interrupts. These are: illegal instruction/guard mode, memory protect/timeout, floating point overflow, and floating point underflow.

- Illegal instruction/guard mode interrupt occurs when a program attempts to execute an instruction with an octal function code of 00 or 7700, and when a program operating in guard mode attempts to execute an instruction with a function of 13, 17, 62, 63, 66, 67, 73 through 76, 7760 through 7767, 7770, and 7772 through 7777. All user programs operate in guard mode.

- Memory protect/timeout interrupt occurs when a program attempts to read, write, or jump outside the area defined by the PLR, and when a program, operating in guard mode, locks out I/O interrupts for a period of approximately 100 microseconds by executing a *Set Interrupt Lockout* instruction that is not followed by a *Release Interrupt Lockout* instruction within the time limit.

- Floating point overflow interrupt occurs during the execution of a floating point add, subtract, multiply, divide, or pack instruction when the magnitude of the biased characteristic of the answer exceeds the capacity of the characteristic portion of the register (i.e., the biased characteristic exceeds 3777_8 or the unbiased characteristic exceeds 1023_{10}). An overflow interrupt also occurs when a floating point division by 0 is attempted.

- Floating point underflow interrupt occurs during the execution of a floating point add, subtract, multiply, divide, or pack instruction when the magnitude of the biased characteristic of the answer is less than zero (i e. the unbiased characteristic is less than -1024_{10})

Overflow or underflow will not be generated when a mantissa of zero results from a floating point operation. In this case, the instruction proceeds to normal completion with a zero mantissa and a zero characteristic.

When overflow or underflow occurs, the arithmetic operation proceeds to completion; the mantissa portion of the AQ register is arithmetically correct. The magnitude of the characteristic will be one beyond the limit value, and the sign bit will be the complement of the correct sign.

2.9.2. Error Routine Specification (ERRADD$, FOFADD$, FUFADD$ Service Requests)

The user may establish a recovery or error routine for fault and floating point contingency interrupts by specifying the starting address of the recovery routine relative to the base of the primary activity of the task. The primary activity is defined as the activity established as the result of the processing of the task card in the job stream. For example, the task card #GOƀƀTEST would cause the coding TEST to be activated as the primary activity of the task. The primary activity may activate other activities through the use of the REG$, REGQ$, and FORK$ service requests. These activities, essentially formed by the primary activity, are called secondary activities.

A specified error routine must handle all errors of the same type for all activities of the task, both primary and secondary. Since secondary activities may have different RIR and PLR values from those of the primary activity, a convention has been set that the error routines are activated with the RIR and PLR of the primary activity. Thus, the activity establishing the error routines must be the primary activity. If in addition, the program is segmented, the error routine must be contained within the control segment.

The error routines are specified through the use of a menomonic operator.

■ Format:

The error routines have the following general form:

operator$ ƀv_0

- Specification:

v_0 is the beginning address, relative to the RIR of the primary activity, of the error routine, and may be any valid read class operand.

The coding generated is:

ENT*B7	v_0
EXRN	Executive Call

The operators and executive calls for the error conditions are as follows:

ROUTINE	OPERATOR	EXECUTIVE CALL
Fault	ERRADD$	20112
Floating point overflow	FOFADD$	20113
Floating point underflow	FUFADD$	20114

Control is returned to the user following the EXRN instruction with one of the following status codes in the A register:

0000000000 Successful completion: The error routine has been successfully established.

4100000010 An error address for this type of contingency has already been established by the task. The new address is not established.

4200000000 Parameter error: The address specified is outside the task limits or the activity attempting to establish the error address has an RIR value different from that of the primary activity. The error address is not established.

2.9.3. Error Routine Operation

As given, the error routine must handle all errors of that type for the entire task, when a task consists of more than one activity, the error routine must be re-entrant to provide for the possibility of simultaneous errors in the multiple activities.

2.9.3.1. ENTRY

When a contingency interrupt occurs, the appropriate error routine is activiated by returning control to the established error address. The routine operates under the addendum of the activity causing the contingency interrupt, but with the RIR and PLR of the primary activity. Thus, the error routine may access any area of the task. If a program contingency occurs within any of the program contingency error routines, the task is aborted by the system.

At activation time, parameters concerning the contingency are given to the error routine. The parameters and their location in various registers are as follows:

- Fault

  B7 contains the address of the illegal instruction relative to the RIR of the error routine primary activity.

  B6 contains the address field of the Internal Function Register (IFR) at interrupt time. For nonjump instructions, this field contains the address of the last referenced memory location. For jump instructions, the field contains the address of the memory location following the jump instruction. The captured address will be modified relative to the RIR of the primary activity.

  B5 contains the RIR value of the activity causing the contingency. The RIR value will be relative to the RIR of the error routine.

  B4 contains the PLR lower limit of the activity causing the contingency. The value given will be relative to the RIR of the error routine.

  B1 contains a code defining the type of program contingency:

  B1 = 0 indicates an illegal instruction/guard mode interrupt

  B1 = 1 indicates a memory protect/timeout interrupt

- Floating point overflow/underflow

  B7 contains the address, relative to the RIR of the error routine, of the floating point instruction causing the overflow/underflow.

  B6 contains the address, relative to the RIR of the error routine, of the operand used in the floating point instruction causing the overflow/underflow.

  B5 contains the RIR value of the activity causing the overflow/underflow. The RIR value will be relative to the RIR of the error routine.

  B4 contains the PLR lower limit value of the activity causing the contingency. The value will be given relative to the RIR of the error routine.

  AQ will contain the result of the floating point operation. The mantissa will be arithmetically correct. The magnitude of the characteristic will be one beyond the limit value, and the sign bit will be the complement of the correct sign.

The error routine may now inspect the circumstances surrounding the contingency, and access or change any area of the task. OMEGA service requests may also be issued by the error routines.

2.9.3.2. EXIT (RETURN$, RETURN1$, ERROR$, ABORT$)

After performing its function, the error routine exits to OMEGA. The exits available, and the action taken by OMEGA, are as follows:

- RETURN$ Returns control to the instruction following the instruction which caused the contingency interrupt. All operational registers are restored to the values held at the time that the contingency occurred. For floating point overflow and underflow, the AQ register value of the contingency routine will be returned, thereby allowing the contingency routine to alter the result of the floating point operation.

- RETURN1$ Terminates the activity causing the contingency. If the activity to be terminated is a queue processing activity, all QREF's associated with the activity will be deallocated.
- ERROR$ Terminates the task, and continues the rest of the job.
- ABORT$ Terminates the entire job.

2.9.4. Default Procedures

When a program contingency occurs, and the user has not specified a routine for processing the error condition, OMEGA processes the contingency interrupt as outlined in the following paragraphs.

2.9.4.1. FAULT CONTINGENCIES

In the absence of a fault routine specification, the occurrence of an illegal instruction/guard mode or memory protect/timeout will cause OMEGA to terminate the offending task. The message FAULT CONDITION, TASK TERMINATED is submitted to the primary output stream of the task together with the values in the operational registers of the program at fault time. The following values are displayed:

RIR	(absolute memory address)
RELP	(P register value at interrupt, relative to the RIR)
LLOCK	(lower lock limit from the PLR)
ULOCK	(upper lock limit from the PLR)
B1 – B7	(contents of program's index registers)
A,Q	(contents of the A and Q registers)
P – 1	(instruction causing the contingency interrupt)
LMR	(last memory reference, bits 0 – 17 of the IFR)

2.9.4.2. FLOATING POINT (TESTFOF$, TESTFUF$, TESTFL$ SERVICE REQUESTS)

In the absence of floating point contingency routines, OMEGA will adjust the result of the floating point operation, set the appropriate point switch in the task addendum, and return control to the program. For floating point overflow interrupts, the result of the operation in the AQ register is returned, either as the maximum possible positive floating point number ($3777{:}7777777777777777_8$) or as the minimum possible negative floating point number ($4000{:}0000000000000000_8$), depending upon the direction of the overflow. For floating point underflow interrupts, the result in the AQ register is returned as positive 0 or 0000:0000000000000000.

When a floating point overflow/underflow interrupt occurs, and an error routine has not been established for the contingency, the floating point overflow or underflow switch is set. These switches are contained in the task addendum and, therefore, apply to all activities of the task. The appropriate switch is set on the occurrence of the interrupt and is not cleared until the corresponding test is made. The switches may be tested by using the operators shown as follows:

FUNCTION	OPERATOR	EXECUTIVE CALL
Testing floating point overflow	TESTFOF$	20107
Test floating point underflow	TESTFUF$	20110
Test overflow/underflow	TESTFL$	20111

TESTFOF Provides a test of the overflow switch. If the switch is not set, control is returned to the requester with the A register cleared. If the switch is set, control is returned with the A register set to 1: all other registers are undisturbed. The switch is then turned off.

TESTFUF Provides a test of the underflow switch. If the switch is not set, control is returned to the requester, with the A register cleared. If the switch is set, control is returned with the A register set to 1: all other registers are undisturbed. The switch is then turned off.

TESTFL Provides a test of both the overflow switch and the underflow switch. The floating point overflow switch in the task addendum is tested first. If the overflow is set, the A register is set to 1 and control is returned to the requester. If the overflow switch is not set, the underflow switch is tested. If the underflow switch is set, the A register is set –1 (7777777776) If neither switch is set, the A register is set to 0. Both switches are turned off, regardless of the results of the test. If both switches are set, only the overflow switch will be reported. All registers other than the A register will be undisturbed.

3. DATA MANAGEMENT SYSTEM

3.1. GENERAL

The Data Management System of OMEGA exercises centralized control over all peripheral resources available on the UNIVAC 494: their assignment, usage, and access. This centralized control of facilities is the basis for establishing an efficient multiprogram and multiactivity environment. In addition, centralization establishes the procedures necessary for providing programmers and operational personnel with the tools necessary for storage, retrieval, and manipulation of the large volume of data and programs involved in the utilization of a computing system.

To effectively utilize hardware resources in a multiprogram and multiactivity environment, the Data Management System performs three major roles: Assignment, File Access, and File Manipulation.

3.1.1. Assignment

One of the major functions of OMEGA is the efficient assignment of the system resources. The assignment is performed so that a relatively simple interface is provided between the user and the hardware while a high degree of device independence is achieved. Device independence permits flexibility in the choice of peripheral devices assigned to the program at execution time without need for changing the program.

References to direct access storage or to peripheral devices within a user program are symbolic so that the program may be compiled, collected, entered and rolled out, independent of assignments. The association of physical device or area with symbolic reference is made when the task programs are set up, without modifying the text of the program.

The type of device or direct access storage desired is expressed to the system by the ASG. The ASG statement can be submitted to OMEGA by any one of or a combination of three methods:

- The normal method for submitting ASG statements is through the primary input stream, immediately preceding the task activation card to which the statements pertain.

- The second method for submission of ASG statements is through the Loader, at the time when relative binary (RB) elements are collected into an absolute program. The Loader recognizes an option to collect control statements as part of the preamble of the collected program. This description will allow the selection mechanism to consider the allocation requirements of the program and to relieve the user of providing the ASG statements each time that the program is run. All systems processors, utility routines, assemblers, and compilers use this method of facility assignment.

- A third method for ASG statement submission involves the active task program which may submit statements dynamically during execution of the program. Caution must be used in the submission of dynamic requests. In a multiprogram environment, selection elements of OMEGA determine which tasks can be entered into the system from the external definitions of the task's primary storage and facility requirements. Therefore, requested peripheral device or large allocations or direct access storage may not be available during execution of a task.

3.1.2. Duration of Assignments

Assignments of direct access storage or peripheral devices are made at the task level at the time when the task is selected for execution, and are automatically released back to OMEGA upon termination of the task. Two notable exceptions to this rule are provided to allow intratask and intrajob control over assignment of generated data files:

- Intratask Assignment

  As an option on the ASG statement, a hold specification is provided, directing the system to maintain the described assignment for the duration of the job or until explicitly released by the FREE statement. This feature provides the user with an ability to employ program chaining techniques such as generation of a file with one task, processing with another, and disposing of a processed file with a third.

  The function is applicable to the assignment of tape units or direct access storage, and should not be used for unit record devices such as card equipment, high speed printers, etc.

- Intrajob Assignment

  The Master File Directory (MFD) allows the user to permanently hold data files or their descriptors which transcend jobs. The MFD is applicable to direct access storage where the installation data base is contained, or for maintaining semipermanent data files. A second application of the MFD is for holding description of magnetic tape reels containing files originally recorded on magnetic tape or which were relocated from direct access storage because of low frequency usage or expiration date.

Allocation of primary storage and storage extensions to a task are performed at activation time and do not transcend tasks within the job deck. Any data in primary storage which is to be saved between successive tasks of a job; for example, blank commom blocks utilized by FORTRAN object code, may utilize the SEND and RECEIVE operators. The SEND operator requests the system to transfer described primary storage data to executive direct access storage until a requesting task within the job accepts the data by a RECEIVE operator.

3.1.3. Data Access Methods

OMEGA provides the centralized control over data access in a multiprogram and multiactivity environment necessary to coordinate utilization by concurrent users. This control includes three levels of user interface: device control, cooperative control, and file control. The user of either device control or file control must acquire assignment of required peripherals or direct access storage prior to access. Assignment associates the physical device or the area of direct access storage with a unique alphabetic character file code as the symbolic reference for each data source contained in the program. The file code, in turn, is presented with each data access either explicitly through device control or implicitly through file control, and establishes the link between data and device or area.

- Device Control

  The device control level of interface provides for functional control of a particular type of device or direct access storage area. The user submits parameter packets describing the functions to be performed and assumes the responsibility for buffer, item, and device strategies.

  Packet requests are formatted for execution by common subroutines referred to as input/output handlers. The basic handlers required by the system are permanent residents of the systems primary storage. Infrequently used handlers are maintained in the systems library and called into primary storage only when an assignment for the type of associated device is made.

  All OMEGA system elements and processors utilize this level of data access.

- Cooperative Control

  Cooperative control is inherent in the OMEGA executive routine and is available for use through direct service request. Functions performed are requests for primary input images, normally cards, which were included in the job stream; and submission of print images and card images for processing by system output routines. No assignment is required for utilization of cooperative control by the user. Data requested or conveyed to cooperative control is buffered to direct access storage, and is collected and distributed to system allocated peripheral units, and determined at systems generation time.

- File Control

  File control involves a group of standard elements providing data handling operations at the block or item level. These elements provide a high level device-independent interface which manages blocking and buffering while utilizing and augmenting the system facility for storing and retrieving data. File control utilizes device level input/output to perform its functions.

3.1.4. Maintenance Functions

The OMEGA executive routine contains elements used in the maintenance and manipulation of data files or program libraries, and elements used to perform general utility functions. These elements are provided with the operating system, but are not an intrinsic part of OMEGA. In general, they are system processors and utility routines activated by control card or service request to perform an explicit function with minimal interface with OMEGA.

Data management maintenance routines fall into three categories: maintenance of program libraries, maintenance of program libraries, maintenance of the file directory, and routines used for utility functions. Later sections of this manual give a detailed explanation of their use and functions performed.

3.2. ASSIGNMENT STATEMENTS

The following paragraphs discuss assignment statements, peripheral names and file codes used for requesting system facilities, and methods for effectively utilizing the statements in the system.

3.2.1. The Assignment (ASG) Statement

The function of the assignment elements within OMEGA is to maintain the status and availability of assignable peripheral subsystems attached to the UNIVAC 494. To perform this function, facility assignment maintains the peripheral and direct access storage requirements of all active tasks and OMEGA elements by responding to their static and dynamic, peripheral, and direct access storage assignment requests.

The association of peripheral device or area of direct access storage to a task is specified by the ASG statement. The statement contains specifications and options for selecting and initializing a specific device, or a device from a general class of subsystems, dependent upon availability. In addition to describing the desired subsystem, the ASG statement specifies the symbolic link which the operating task will use at access time. The ASG statement is of the normal primary control statement format and is amendable to many types of peripheral subsystems. The variety of input/output devices available for the UNIVAC 494 makes it necessary to describe the ASG statement according to the class of subsystem, e.g., direct access storage, tape units, unit record peripherals, and remote devices. However, the general form of the statement will be fairly standard as follows (subsequent sections describe specific formats by the class of subsystem):

- Format: #

  #ASGƀoptionsƀperipheral name, file code, assignment specifications.

  Options, in general, specify the grade of subsystem desired and supply initialization parameters.

  Peripheral name is a one to five character alphanumeric name of the desired device or class of devices.

  File code is one- or two-alphabetic character(s) used as the symbolic link between an active task and a physical assignment.

  Assignment specifications are peculiar to the type of subsystem and are discussed in detail in subsequent sections of this manual.

3.2.2. Peripheral Name

The peripheral name is the mnemonic name of the requested peripheral unit or direct access storage assignment, e.g., TAPE, UN6C, FH880, etc. The permissible mnemonics for this field are determined by the names applied by the installation at systems generation time to represent a particular configuration or order of assignment. The choice of mnemonics is completely open-ended. Several names may describe a single peripheral subsystem or a unique name may specify a particular unit on a specific peripheral subsystem. Omega assignment provides a mapping function of mnemonic names whereby a specific name may contain a number of alternate choices for assignment in a preset order.

To illustrate the use of peripheral name, mnemonic names and assignment dropout rules, assume the following hypothetical configuration with each unit being assigned a specific peripheral name and alternates:

TYPE OF DEVICE	PHYSICAL CHANNEL/UNIT	NAME(1)	NAME(2)	NAME(3)	NAME(4)
UNISERVO VI C	13/0	SEQ	TAPE	UN6C	UN6CA
UNISERVO VI C	13/1	SEQ	TAPE	UN6C	UN6CA
UNISERVO VI C	13/2	SEQ	TAPE	UN6C	UN6CA
UNISERVO VI C	14/0	SEQ	TAPE	UN6C	UN6CB
UNISERVO VI C	14/1	SEQ	TAPE	UN6C	UN6CB
UNISERVO VI C	14/2	SPT	–	–	–
UNISERVO VI C	14/3	SPT	–	–	–
UNISERVO VIII C	16/0	SEQ	TAPE	UN8C	–
UNISERVO VIII C	16/1	SEQ	TAPE	UN8C	–
UNISERVO VIII C	16/2	SEQ	TAPE	UN8C	–
FH-880 Drum	17	SEQ	RAN	FH880	–

Note: SPT is a mnemonic for a special tape unit not available for general assignment; with two units eligible for SPT, both would contain functionally the same information.

As can be seen from the above example, a request for a sequential file (SEQ) is the most general assignment which could be requested and may be satisfied by either a tape unit or an allocation of direct access storage. In contrast, a request for a tape file (TAPE) limits the choice to UNISERVO tape units. Through the specification of additional mnemonic names, the generality of assignment may be further limited to a specific type of tape unit (UNISERVO VI or VIII), to the request for a unit from a specific channel (UN6CA or UN6CB), or to the request for a specific unit on a channel (SPT).

In the above example, the assignment mapping element of OMEGA could have been adjusted at systems generation time to allow the use of alternates to satisfy the request. Alternates could have specified that units from the UN8C and/or UN6CB group could be assigned if units in the UN6CA group were not available.

The following chart illustrates the choices which could be used as alternates from the above example:

NAME	1st CHOICE	2nd CHOICE	3rd CHOICE	4th CHOICE	5th CHOICE	6th CHOICE	7th CHOICE	8th CHOICE	9th CHOICE
SEQ	17	16/0	16/1	16/2	13/0	13/1	13/2	14/0	14/1
TAPE	16/0	16/1	16/2	13/0	13/1	13/2	14/0	14/1	–
RAN	17	– –	–	–	–	–	–	–	–
UN6C	13/0	13/1	13/2	14/0	14/1	16/0*	16/1*	16/2*	–
UN8C	16/0	16/1	16/2	13/0*	13/1*	13/2*	14/0*	14/1*	–
FH880	17	– –	–	–	–	–	–	–	–
UN6CA	13/0	13/1	13/2	16/0*	16/1*	16/2*	14/0*	14/1*	–
UN6CB	14/0	14/1	16/0*	16/1*	16/2*	13/0*	13/1*	13/2*	–
SPT	14/2	14/3	–	–	–	–	–	–	–

*Indicates optional alternates to the original assignment. The alternate may be specified at systems generation time as desired by the installation.

- I/O Handler

In addition to specifying a physical device, the peripheral name also implies the device handler which will be used in processing I/O requests by the operating task to the assignment. Input/output handlers, for infrequently used devices or nonstandard I/O processing, are entered into primary storage and are initialized only when the device is assigned or when nonstandard processing is to take place. The major use of this specification is for devices such as paper tape and card subsystems which are essential to common subroutines. This provides the user with the following features:

– The ability to conserve primary storage during periods when a particular input/output handler is not required because of lack of assignment.

– The ability to utilize nonstandard input/output handlers for the shared usage of the system or a particular channel on the system. That is, two or more tape or direct access storage handlers may utilize a physical channel concurrently. This is particularly advantageous to the installation which requires special or additional processing for a class of input/output access, for example, double queueing of file updates, audit trails, and other operations. This also permits the integrated subsystem test to operate concurrently with production tasks, with little or no impact upon their functional requirements.

3.2.3. File Code

File code is the symbolic bridge by which an operating task accesses or references the physical device or area assigned to it. Once the choice is made by the ASG statement, the user conveys the file code with each device level I/O request or other reference to the assigned peripheral device or area of direct access storage. This establishes a mapping arrangement whereby the task code does not require modification at execution for I/O access. File code also affords the user a procedure whereby a task cannot inadvertently access a unit or area of direct access storage which has not been previously equated to the task through the ASG statement.

File codes are established at the task level. That is, each task currently operating within the system has a complete set of file codes eligible for the task's use. Therefore, no programming conventions are required by the user for specifying file codes, other than those conventions required for intratask control of assignments. However, all activities emanating from the task have shared usage of all facilities assigned to the task.

Each task addendum is provided with a basic set of 25 file codes to which the user may assign a peripheral device or areas of direct access storage. The codes are symbolically referenced by an alphabetic character from the set A through Y. In cases where 25 file codes are inadequate for the task, a user may specify that a designated file code be fragmented into an additional set of 26 file codes which have the same characteristics as the original set. For example, if the file code B were fragmented, the new set would be referred to as BA, BB . . . BZ.

Fragmentation of a particular file code is implied when an ASG statement is submitted which specifies a two letter file code. A file code from the basic set A through Y which is to be fragmented may not be used for assignment. Essentially only one peripheral device, file of direct access storage, or fragment of file codes may be associated with any one file code from the basic set, and only one peripheral device or area of direct access storage may be associated with any one file code of the extended set.

The file code Z from the basic set is reserved for OMEGA and installation control program routines. The Z file code is automatically fragmented to access the following files or devices:

FILE CODE	RESERVED USAGE
ZA	Unit record routine (primary input)
ZB	Unit record routine (primary output)
ZC	Unit record routine (secondary output)
ZD	Cooperative library assignment
ZE	Systems library assignment
ZF	Job library assignment
ZG	Source routine

FILE CODE	RESERVED USAGE
ZH	Systems scratch file
ZI	Systems scratch file
ZJ	Systems compaction file

3.2.4. Direct Access Storage Assignment

A request for the assignment of direct access storage can be satisfied on any one of several available subsystems, including the FH series drums, the FASTRAND series mass storage, and the 8400 series discs. Each assignment request is satisfied by a multiple of the block size used to map the subsystem. The block size for any one type of subsystem is a practical minimum number of words consistent with the characteristics of the device. On FASTRAND mass storage, for example, the block size is a multiple of a track.

The physical description of an assigned direct access file may consist of discrete noncontiguous areas across available blocks of direct access storage. The noncontiguous areas may cross drums, channels, and even types of systems. As an example, a file may be composed of blocks from FH-880 drum and FASTRAND mass storage.

This may occur either through planned positioning of a file or as the result of successive extensions to a sequential (SEQ) or random (RAN) file. However, to the operating task, all direct access storage assigned to a file code is word addressable and logically continuous.

OMEGA file routines and direct access storage handler provide the user with a word-addressable interface for all types of direct access storage through the use of file code and logical increment. The logical increment is essentially a pseudo drum address relative to the file code to which the assignment was made. At execution time, the task code submits the logical increment of the file segment being accessed along with the desired I/O function. The OMEGA Random Storage File Handler then maps the logical increment to the physical address to perform the I/O function. The following example illustrates the relationship of physical address to logical increment:

Assume a file composed of three discrete noncontiguous areas of direct access storage. The first two areas are FASTRAND mass storage assignments, split because of nonavailability of a contiguous area to satisfy the original request or as the result of an original assignment and subsequent extension to the assignment. The third area is FH-880 drum storage assigned dynamically as an extension to the original file. The relationship between the physical address and the logical increment would be as follows:

Beginning Sector Address 1400		Beginning Logical Increment 0
	Three Contiguous Tracks of FASTRAND Mass Storage Assigned	
Ending Sector Address 1591		Ending Logical Increment 6,335
Beginning Sector Address 300		Beginning Logical Increment 6,336
	Three Contiguous Tracks of FASTRAND Mass Storage Assigned	
Ending Sector Address 427		Ending Logical Increment 10,559
Beginning Drum Address 1,000		Beginning Logical Increment 10,560
	Two 256 Word Blocks of FH-880 Drum Assigned	
Ending Drum Address 1,511		Ending Logical Increment 11,071

3.2.4.1. FEATURES

OMEGA direct access assignment and data access elements provide the user with the following important features necessary in a multiprogram environment:

- File protection

  This is the ability to protect direct access storage files assigned to a task from advertent destruction by concurrently operating tasks. At the same time, through use of the file directory, two or more concurrently operating tasks may share usage of common files and interlock records in process of update (see file directory and logical lock I/O requests).

- File expansion

  This is the logical increment in conjunction with the file code which allows direct access storage files to be dynamically expanded or contracted without the need for compaction, to develop a physically contiguous area.

- Device independence

  This logical incrementing allows the program to be assigned word-addressable drum, sector-addressable FASTRAND mass storage, or record-addressable disc storage without impact upon code logic. However, in cases where the possibility exists that the file will be assigned to sector or record addressable storage, the user should orient read and write requests to multiples of 33 words from the base of the file to minimize latency.

- Partitioned files

  These are the portions of a particular file which are frequently accessed and which may be assigned to drum; portions of the file which are infrequently accessed may be assigned to FASTRAND or disc storage, thereby minimizing task turnaround time within the constraints of the configuration.

3.2.4.2. DIRECT ACCESS STORAGE ASSIGNMENT STATEMENT AND MNEMONICS

The direct access storage ASG statement specifies the type and quantity of direct access storage to be assigned as a file or the type and quantity by which an existing file is to be extended. The ASG statement also specifies the creation of image or duplex files; these are two direct access storage areas, or two collections of direct access storage areas, for which, normally, the total of one area equals the total storage of the other area, and in which the data is identical for both areas. This means that a read function addressed to either of the Random Access Storage Lists (RASL) which describe the two files will produce identical data as a read function addressed to the other. Image filing is discussed in 3.3.6.

Format:

The ASG statement has the following general form:

```
#ASGƀoptionsƀperipheral name,   file code,   estimate,   packname,   packname,   etc.
```

Options:

A Assign storage beginning at the logical address specified in the area field.

C Inhibit the automatic recovery on the secondary file when the storage handler detects an error condition in the primary file.

D Inhibit the automatic recovery on the primary file when the storage handler detects an error condition in the secondary file.

G Assign storage even if the subsystem is marked as being down.

J Hold assignment for duration of the job unless explicitly released by the FREE statement. Absence of the J option implies that the assignment will be released upon termination of the task.

K Assign storage to the primary copy of an image file. (Used in conjunction with the Q option). Absence of the K option implies assignment to the secondary copy of an image file.

Q Assign storage to only one copy of an image file. (Used in conjunction with the K option.)

R Satisfy the storage request from one subsystem only.

S Negate the printing of the assignment message to primary output.

V Assignment is optional for execution of the task. If storage is not available at selection time, do not assign a file, but continue to execution phase.

Specifications

Peripheral name is a one to five character mnemonic used to name the direct access storage which is to be assigned. See peripheral name (3.2.2) and systems generation (7.2) for additional use and specifications.

The peripheral name field may consist of two subfields each containing a peripheral name mnemonic, and separated by an oblique (/) character, when assigning storage to primary and secondary image file copies, respectively.

The list of standard mnemonics contained in the distributed system appears below:

MNEMONIC	ACTION TO BE TAKEN
DRUM	Request to be satisfied by assignment of FH-432, FH-1782, or FH-880 drum, respectively, dependent upon availability.
F432 or FH432	Request to be satisfied by assignment of an area on FH-432 drum only.
F880 or FH880	Request to be satisfied by assignment of an area on FH-880 drum only.
F1782	Request to be satisfied by assignment of an area on FH-1782 drum only.
FAST	Request to be satisfied by assignment of area on FASTRAND drum only.
FBAN	Request to be satisfied by assignment of Fastrand area on FASTRAND drum only.
DISC or DISK	Request to be satisfied by assignment of area on 8400 series disc only.
RAN	Request to be satisfied by assignment of an area on FH-432, FH-1782, FH-880, Fastband or FASTRAND respectively, dependent upon availability of storage.
SEQ	Request to be satisfied by assignment of an area on FH-432, FH-1782, FH-880, Fastband, FASTRAND or tape unit, respectively, dependent upon availability of storage. (See 3.2.2 for special program considerations when using this mnemonic.)

File Code is a one or two alphabetic character code used to link the assignment to task calls (see 3.2.3 and 3.3.2 for detailed explanation).

The logical address is relative to the base of the system specified by the peripheral name field, and must be divisible by the sub-system's module size. An A-option assignment must be wholly satisfied on a single sub-system.

NUMBER OF UNITS ON CHANNEL	FH-432	FH-1782	FH-880	FASTRAND*	FASTBANDS
1	132	528	264	2,112	
2	264	528	264	2,112	
3	264	1,056	528	2,112	
4	264	1,056	528		
5	264	1,056	528		
6	264	2,112	528		
7	528	2,112	528		
8	528	2,112	528		
9	528	-	-	-	-

* NOTE: FASTRAND BLOCKS are in multiples of a track.

Table 3–1. Block Sizes

From the above chart the user is able to determine the optimum size for a direct access storage request. For example, if a file required approximately $100{,}000_{10}$ words of direct access storage, and if the request were for a two-drum FH-880 channel, the number of words assigned would be $100{,}056_{10}$. However, if the request were submitted for a one-unit FASTRAND channel, $101{,}376_{10}$ words would be assigned.

Estimate, in the absence of an A option, is interpreted as follows:

MIN/MAX/RS

MIN defines the minimum amount of storage required.

MAX defines the maximum amount of storage required.

RS defines the record size of the file on disc subsystems, and is ignored for all other subsystems.

If available, the maximum area of storage, or any amount above the minimum specification is assigned. If the minimum amount of storage is not available, a non-assignment status is returned.

The MIN/MAX or area specification has a special interpretation if equal to zero, and if the assignment is made to an existing simplex file. Under these circumstances, the simplex file will be assigned a corresponding duplex or image file.

On disc subsystems, the entire area assigned is prepped at assignment time to the specified record size.

When associated with an A option, the estimate field, for non-disc devices, is interpreted as follows:

WDS/LAP/LAS

WDS defines the amount of storage requested.

LAP defines the beginning logical address for the primary file.

LAS defines the beginning logical address for the secondary file.

The ability to specify the minimum acceptable area, distinct from the maximum requirement, is not provided.

For disc devices only, the estimate field, in association with the A option, is interpreted as follows:

WDS/LA/RS

WDS defines the amount of storage requested

LA defines the beginning logical address of the file

RS defines the record size of the disc storage

In this case, only simplex assignments may be made. In order to create a duplex disc file, when using the A option, the user must submit a zero WDS request, to add an image leg to the simplex file.

As with other direct access storage devices, the logical address must be divisible by the subsystem's module size, which for disc devices is one track. Unlike other devices, the logical address is relative to the base of the disc pack on which the assignment is to be made, and not to the base of the disc subsystem.

The number of words in a track on a disc device are a function of the record size of the assignment, since the more interrecord gaps that are present, the smaller the area that may be devoted to data storage. The following chart contains some examples of track capacities on 8414 subsystems.

NUMBER OF WORDS PER RECORD	1	33	66	99	132	165	198	200	1945
NUMBER OF WORDS PER TRACK	70	1056	1320	1485	1584	1485	1584	1600	1945

Users may calculate 8414 or 8424 track capacities, at any record size, using the following algorithm:

$$\text{Records per track} = \frac{(7294 - 15W//4) \times 2048}{206848 + 32055W//4} + 1$$

where W is the number of words per record, and // expresses the operation of division, and rounding up to the next integer, when there is a remainder.

Packname is a one to ten character alphanumeric field, and describes the disc pack on which the assignment is to be made. Two packnames in two subfields separated by an oblique (/) character may be specified when assigning an image file.

As many packname fields may be specified, as can be contained on a single ASG statement card. Area is assigned on the specified packs in such a way as to minimize fragmentation: if the request cannot be assigned in one area, it will be assigned in the largest available areas. If the area is not available on the specified packs, *or the request can only be satisfied by creation of ten or more fragments on a single pack, no assignment is* made.

When no packname is specified, the file will be assigned on any available pack in the system.

3.2.4.3. METHOD OF SUBMISSION

Direct access storage ASG statements may be submitted externally through the control stream or collected with the task element, or they can be submitted internally during execution of the task. All ASG statements contained in the control stream should precede the task activation card to which the statements pertain.

ASG statements submitted internally during execution of the task are normally used to extend an existing file. The original assignments for the existing file should be formed previously by an external file should be formed previously by an external ASG statement or should be acquired from the file directory.

Any number of file extension requests may be performed during the execution of a task. Each may request the same peripheral name or a name different from that assigned to the original file. Upon return of program control from an internal ASG request, the following status information is conveyed in registers A and Q:

Normal Completion: A is set to 00LLLLLLLL where L is the starting logical increment relative to the file base of the extension. Q contains the number of words assigned.

Abnormal Completion: No assignment is made and the A register contains one of the following values, dependent upon the cause:

5000000000	No assignment made: Requested direct access storage was not available.
4200000000	Incorrect parameter: ASG statement could not be interpreted by the system.
4100000000	Inappropriate function: Peripheral name required storage which is not in the configuration.
430000000	Subsystem error occurred while processing the ASG statement.

The Q-register will be negative (all 7's), if the error condition was detected on the primary file, but will be positive (all 0's), if the error was detected on the secondary file.

3.2.5. UNISERVO Tape Assignment

A request for the assignment of a tape unit can be made for any type of magnetic tape subsystem available on the UNIVAC 494 Real-Time System. These subsystems include the compatible UNISERVO units III C, IV C, VI C, and VIII C, and the UNISERVO II A and III A, and the UNISERVO 12/16 units.

The request for tape unit assignment is submitted through the ASG statement. The ASG statement contains the necessary specifications required to ready the unit for operation. These include: parity and density in which the unit is to be used, file code of assignment, and tape reel mounting instructions for the operator.

- Format:

  The ASG statement has the following general form:

  #ASGƀoptionsƀperipheral name, file code, filename

- Options:

X	A format on UNISERVO 12/16
Y	B format on UNISERVO 12/16
Z	C format on UNISERVO 12/16

  no X, Y, or Z assumes B format on UNISERVO 12/16

H,M,L Specifies the density in which the tape is to be recorded or read at access time. The option letters have the following connotations:

H (High)	Set mode at 800 fpi.
M (Medium)	Set mode at 556 fpi.
L (Low)	Set mode at 200 fpi.

Absence of H, M, or L option implies ttat the mode is set at medium density, 556 fpi.

Density options are ignored if the peripheral name is selected for a UNISERVO VIII C or 12/16 unit with the 9-track format feature. The fixed density for all 9-track NRZI recordings and readings is 800 fpi, and for all 9-track phase-encoding operations, 1600 fpi.

E Set parity even. This option is normally used only when reading tapes which are going to be read by other computer systems. Lack of an E option implies use of odd parity. All tapes recorded by OMEGA and its processors use the odd parity mode unless explicitly overridden. UNISERVO VIII C subsystems with the 9-track format feature transfer data in odd parity mode only, regardless of the selected option.

F Specifies the fixed or compatible mode for UNISERVO II A units only. When this option is used, write buffers must be 144_{10} words in length and write end-of-file functions are illegal. If the L option is used with F, data will be written in blockette. With no density specified, or with the M or H options specified, data is recorded in blocks of 720_{10} characters.

I Rewind with interlock will be performed when the tape unit is released.

J Hold assignment for duration of job unless explicitly released through the FREE statement. Absence of the J option implies that the assignment will be released upon termination of the task.

N Inhibit the noise record constant defined at systems generation time.

P Perform a closing routine when the peripheral unit is released. Tape units will be rewound without interlock.

R Rewind the assigned unit without interlock after operator mounting of tapes.

T Translate to Fieldata (FD) from binary coded decimal (BCD) on read requests, and from FD to BCD on write requests. This option is applicable to compatible tape units having the translate option only and normally requires that the E option be present.

U Inhibit automatic block-numbering feature (see 3.2.5.2 for further details).

V Assignment is optional for execution of the task. If the unit is available at selection time, assign it; otherwise, continue to task execution phase.

W Wait for operator response after the file name is printed on the console printer.

- Specifications:

Peripheral Name is a one- to five-character mnemonic used to name the type of unit to be assigned. See peripheral name (3.2.2) and systems generation (7.2) for additional use and specifications.

The list of standard mnemonics contained in the distributed system is as follows:

MNEMONIC	ACTION TO BE TAKEN
TAPE	Assign any available tape unit.
UN3C	Assign only a UNISERVO III C tape unit
UN4C	Assign only a UNISERVO IV C tape unit. Essentially, UNISERVO III C and IV C are functionally equivalent; hence, this distinction is normally not necessary.
UN6C	Assign only a UNISERVO VI C unit in normal 7-track mode.
UN8C	Assign only a UNISERVO VIII C tape unit in normal 7-track mode.
UN2A	Assign only a UNISERVO II A tape unit.
UN3A	Assign only a UNISERVO III A tape unit.
UN9C	Assign only a UNISERVO VI C or VIII C tape unit which has a 9-track compatible option.
U12	Assign only a UNISERVO 12 unit.
U16	Assign only a UNISERVO 16 unit.
COMP	Assign UNISERVO VI C or VIII C with 9-track option. ←
NOTE:	See 3.2.5.5. for 9-track format programming considerations.

File Code is a one- or two-alphabetic character code used to link assignment task calls (see 3.2.3 for details). Only one unit may be assigned to a particular file code.

File Name is used for operator tape mounting instructions, and is a 1- to 15-alphanumeric character name which identifies the physical tape reel to be mounted. At assignment time, this field will be typed on the console printer for directing the operator (see Section 5 for console format).

The console operator has the option of premounting tapes and indicating their presence to the operating system. This is accomplished by mounting the tape on an available unit and specifying, through console type-in, the file name which will be contained on the subsequent ASG statement, and the physical channel/unit on which the tape is mounted (see Section 5). When this option is used, OMEGA bypasses the peripheral name specified on the ASG statement, and assigns the unit indicated by the console operator. In the case of premounted magnetic tapes, no console type-out is performed. File names SCRATCH and BLANK having meaning to the system, and apply to tapes which are normally premounted by the operator. SCRATCH implies a temporary assignment which will be used for the duration of the task or job. BLANK implies that the file(s) which will be recorded on the tape will be saved for further use.

3.2.5.1. METHOD OF SUBMISSION

Tape unit assignment statements are normally submitted by the ASG statement externally through the control stream and collected with the task element to be scheduled. All ASG statements contained in the control stream must precede the task activation card to which the statements pertain.

The tape ASG request can also be submitted internally, during execution of the task. However, care must be used when making internal facility requests, especially with regard to unit assignments of any type. The OMEGA algorithms for multiprogramming scheduling will select tasks to be run on the basis of facilities expressed externally to the task. Hence, during execution, additional facilities may not be available because of their assignment to a concurrently operating task. In such cases, the internal request will not be satisfied.

Upon return of program control from an internal ASG request, the following status information is conveyed to the task in the A register:

Normal completion: The register is set positive, indicating that the unit is successfully assigned. The lower portion contains the peripheral type code, and the upper portion contains the subsystem number.

Abnormal completion: No assignment is made and the register contains one of the following values, dependent upon the cause:

5000000000	Requested unit not available.
4200000000	Incorrect parameter: #ASG statement could not be interpreted by the system.
4100000000	Inappropriate function: peripheral code required a unit not in the configuration.
4300000000	Subsystem error occurred while processing the #ASG statement.

3.2.5.2. BLOCK NUMBERING

A feature of the OMEGA magnetic tape handler is automatic block numbering. Each block of data recorded on a magnetic tape, load point being 0, is affixed with a sequential octal number as its first word if the operational mode is 7-track, or with a four word Fieldata/octal description if the mode is 9-track. Block numbering can be used with either even or odd parity setting.

The block numbering is performed without assistance from the user task. No allowance for increase d buffer specification is required to accommodate the block number, with the exception of a read backward request in 9-track mode. The OMEGA tape handler effects the reading and writing of each tape through scatter read and gather write techniques to separate block number from data. On each read request, the block number is unstrung and returned to the requesting activity in its binary form by the Q register (see 3.3.7).

When the read backward operation in the 9-track mode is performed, the block number is not separated from the data. This creates problems in data reconstruction if the block does not contain a multiple of four words. Read backward operations in 9-track mode with block numbering should be avoided. If the option is unavoidable, the receiving buffer should be large enough to accommodate the block number and data words. The entire contents of the block should then be read into the buffer. If abnormal frame status is received, the data can be reconstructed with a series of character shifts (refer to the discussion of device control in 3.3). With no abnormal frame status, data can be reconstructed by reversing the word order.

The block numbering option is considered the normal mode of operation for all system processors and user programs operating under control of OMEGA. When tapes which have been recorded on other computers or which will be read under operating systems other than OMEGA, are processed by routines under control of OMEGA, the block numbering option should be turned off by the U option of ASG statements. This is because the block number is the first physical word on each tape block. Otherwise, it would require that the source and/or object computer be cognizant of the block number feature and compensate for it through programming. When reading backward, the block numbers are not checked by the system. The block number word(s) will appear as the last word(s) of the data when the tape block is less than or equal to the buffer size (see 3.3.7.7).

Block numbering provides the ability to verify block continuity on a tape, thereby eliminating the possibility of undetected data loss during process functions. In addition, block numbering simplifies error recovery and provides for a more efficient checkpoint and restart procedure.

3.2.5.3. NOISE CONSTANT

Noise constant is a programming convention used to define the minimum size in 30-bit words for any one magnetic tape block which will be recognized by the system as containing data during read options. If a parity error has occurred and the block is smaller than this value, the block read operations will be ignored and regarded as noise during read operations.

The noise constant value is set at systems generation time. The value contained in the distributed version is set at three words and four words in 7-track and 9-track mode, respectively.

This check can be disabled through the N option contained on the ASG statement when the possibility exists that the tape contains data blocks of less than three words.

3.2.5.4. TRANSLATE FEATURE

The translate feature is an optional hardware feature on the UNISERVO VI C and VIII C control units. This feature provides for hardware translation of complete data blocks transferred from tape to primary storage (read operations) or to tape (write operations). The translation is from one six-bit code set to another six-bit code set, normally Fieldata to/from binary coded decimal (BCD).

The translate option is activated through the T option on the ASG statement. Once assignment is set for translation, all I/O requests directed to the file code will undergo translation.

The conversion table is contained in·the tape control unit (see Table 3-2 for the standard character conversion table wired at installation time). The table can be modified at the installation for variations of code conversion.

3.2.5.5. 9-TRACK FORMAT FEATURE

The optional 9-track format feature for the UNISERVO VI C and VIII C tape units provides the ability to read and/or write tapes prepared on or for the UNIVAC 9200/9300 computers or the IBM* 2400 series magnetic tape subsystem. The main difference between the normal 7-track format tape and a 9-track format tape is in the read/write head. The 7-track unit writes seven bits in each frame. The nine bits in each frame consist of eight data bits plus a parity bit which provides odd lateral parity for a frame.

Specification of peripheral name, UN9C, requests OMEGA to assign a unit with the 9-track option. Once assigned to a file code, the unit will remain in the 9-track mode until release. All I/O commands given at access time are of the normal format, detailed in subsequent sections of this chapter. However, the following program considerations must be noted when using the 9-track option:

- All read/write operations must be initialized to high density, H option, 800 fpi.
- Tapes recorded on a unit with the 9-track option must be read by a unit with the 9-track option or 9-track heads.
- Eight data bits per frame is not a multiple of the 30-bit computer words.

*Registered Trademark of International Business Machines Corporation.

TAPE TO CORE CONVERSION					
TAPE CODE	494 CODE	HSP SYMBOL	TAPE CODE	494 CODE	HSP SYMBOL
00	46	&	40	41	–
01	61	1	41	17	J
02	62	2	42	20	K
03	63	3	43	21	L
04	64	4	44	22	M
05	65	5	45	23	N
06	66	6	46	24	O
07	67	7	47	25	P
10	70	8	50	26	Q
11	71	9	51	27	R
12	60	0	52	55	!
13	44	=	53	47	$
14	72	'	54	50	*
15	53	:	55	02	]
16	45	>	56	73	;
17	00	@	57	04	Δ
20	05	Space	60	42	+
21	74	/	61	06	A
22	30	S	62	07	B
23	31	T	63	10	C
24	32	U	64	11	D
25	33	V	65	12	E
26	34	W	66	13	F
27	35	X	07	14	G
30	36	Y	70	15	H
31	37	Z	71	16	I
32	77	≠	72	54	?
33	56	'	73	75	.
34	51	(	74	40	)
35	52	%	75	01	[
36	57	\	76	45	>
37	76	¤	77	03	#

CORE TO TAPE CONVERSION					
494 CODE	TAPE CODE	HSP SYMBOL	494 CODE	TAPE CODE	HSP SYMBOL
00	17	@	40	74	)
01	74	[	41	40	–
02	55	]	42	60	+
03	77	#	43	76	<
04	57	b̸	44	13	=
05	20	Space	45	16	>
06	61	A	46	00[1]	&
07	62	B	47	53	$
10	63	C	50	54	*
11	64	D	51	34	(
12	65	E	52	35	%
13	66	F	53	15	:
14	67	G	54	72	?
15	70	H	55	52	!
16	71	I	56	33	,
17	41	J	57	36	\
20	42	K	60	12	0
21	43	L	61	01	1
22	44	M	62	02	2
23	45	N	63	03	3
24	46	O	64	04	4
25	47	P	65	05	5
26	50	Q	66	06	6
27	51	R	67	07	7
30	22	S	70	10	8
31	23	T	71	11	0
32	34	U	72	14	'
23	25	V	73	56	;
34	26	W	74	21	/
35	27	X	75	73	.
36	30	Y	76	37	¤
37	31	Z	77	32	≠

[1] The data code 00 cannot be written on tape in even parity.

Table 3–2. Standard Magnetic Tape Character Conversion

3.2.6. Card Device Assignment

A request for assignment of a card device can be satisfied on either the UNIVAC 494 card subsystem, the UNIVAC 9300 subsystem or the UNIVAC 1004 subsystem. The request for card device assignment is normally submitted by the ASG statement. The ASG statement contains the specifications required to ready the unit for operation.

- Format: The ASG statement has the following general form:

  #ASGƀoptionsƀperipheral name,file code, file name

- Options:

  B Read or punch cards in column binary mode.

  J Hold assignment for duration of job unless specifically released through the FREE statement. Absence of the J option implies that the assignment will be released upon termination of the task.

  P Perform a closing routine when the peripheral device is released (punch three blank cards).

  T Translate card read image from XS–3 or 90-column code to Fieldata. Translate card punch image from Fieldata to XS–3 or 90-column code.

  Absence of B and T options implies that the card image will be translated.

  V Assignment is optional for execution of the task. If the unit is available at selection time, assign it; otherwise, continue to task execution phase.

  W Wait for operator response after the file name is printed on the console printer.

- Specifications:

  Peripheral Code is a one- to five-character mnemonic used to name the type of unit to be assigned. See peripheral name (3.2.2) and systems generation (7.2) for additional use and specifications.

  File name is a 1 to 15-character identifier, which may be used for directing the operator or for the dedication of a unit through the MT (mount tape) function wherein the system directs the use of a unit on which the operator has premounted a tape reel.

  The list of standard mnemonics contained in the distribution system is as follows:

MNEMONIC	ACTION TO BE TAKEN
CARD	Assign any available card reader.
CIN8	Assign any 80-column card reader other than primary input.
CIN9	Assign any 90-column card reader other than primary input.
COUT8	Assign any 80-column card punch other than secondary output.
COUT9	Assign any 90-column card punch other than secondary output.

File Code is a one- or two-alphabetic character code used to link the assignment to task calls. Only one unit may be assigned to a particular file code.

3.2.6.1. METHOD OF SUBMISSION

Card device assignment statements are normally submitted by the ASG statement externally through the control stream and/or collected with the task element to be scheduled. All ASG statements contained in the control stream should precede the task activation card to which the statements pertain.

Card device ASG requests can also be submitted internally during execution of the task. However, care must be used when making internal facility requests, especially with regard to the unit assignments of any type. The OMEGA algorithms for multiprogramming scheduling will select tasks to be run on the basis of facilities expressed externally to the task. Hence, during execution, additional facilities may not be available due to their assignment to a concurrently operating task. In such a case, the internal request will not be satisfied.

Upon return of program control from an internal ASG request, the following status information is conveyed to the task in the A register:

Normal Completion: The register is set to positive, indicating that the unit is successfully assigned. The lower portion contains the peripheral type code, and the upper portion contains the subsystem number.

Abnormal Completion: No assignment is made and the register contains one of the following values:

5000000000	Requested unit is not available.
4200000000	Incorrect parameter: ASG statement could not be interpreted by the system.
4100000000	Inappropriate function: Peripheral code required a unit which is not in the configuration.
4300000000	Subsystem error occurred while processing the ASG statement.

3.2.7. Printer Assignment

A request for assignment of a printer can be satisfied on either the UNIVAC 494 high speed printer, the UNIVAC 9300 subsystem, or a UNIVAC 1004 printer. The request for printer assignment is normally submitted by the ASG statement. The ASG statement contains the specifications required to ready the unit for operation.

- Format

  The ASG statement has the following general form:

  #ASGƀoptionsƀperipheral name, file code, estimate, file name

- Options:

  J Hold assignment for duration of job unless explicitly released through the FREE statement. Absence of the J option implies that the assignment will be released upon termination of the task.

  V Assignment is optional for execution of the task. If the unit is available at selection time, assign it; otherwise, continue to task execution phase.

  W Wait for operator response after file name is printed on console printer.

- Specifications:

  Peripheral Name is a one- to five-character menmonic used to name the unit to be assigned. See peripheral name (3.2.2) and systems generation (7.2) for additional use and specifications.

The standard mnemonics contained in the distributed system are:

MNEMONIC	ACTION TO BE TAKEN
PRINT	Assign any available printer.
HSP	Assign only a High Speed Printer.
PRT	Assign only a 1004 printer.
93PT	Assign only a 9300 subsystem printer.

File Code is a one- or two-alphabetic character code used to link assignment to task calls. Only one unit may be assigned to a particular file code.

Estimate defines the upper margin (UM), bottom margin (BM), and number of printable lines per page (PL) in the format UM/BM/PL. The values may be given in octal or decimal. If this parameter is absent, the parameter system format will be assumed. If PL = 0, the system assumes that the user will perform his own form control; the system assumes that the paper is in home position when an ASG statement is encountered.

File Name is an identifier which may be used for direction of the operator or for dedication of a unit by the MT function.

3.2.7.1. METHOD OF SUBMISSION

Printer assignment statements are normally submitted by the ASG statement externally through the control stream and/or collected with the task element to be scheduled. All ASG statements contained in the control stream should precede the task activation card to which they pertain.

Printer ASG requests can also be submitted internally during execution of the task. However, care must be used when making internal facility requests especially with regard to unit assignments of any type. The OMEGA multiprogramming scheduling algorithms select tasks to be run on the basis of facilities expressed externally to the task. Therefore, during execution, additional facilities may not be available because of their assignment to a concurrently operating task. In such a case, the internal request will not be satisfied.

Upon return of program control from an internal ASG request, the following status information is conveyed to the task in the A register:

Normal Completion: The register is set to positive, indicating that the unit has been successfully assigned. The lower portion contains the peripheral type code, and the upper portion contains the subsystem number.

Abnormal Completion: No assignment is made and the register contains one of the following values:

5000000000	Requested unit is not available.
4200000000	Incorrect parameter: ASG statement could not be interpreted by the system.
4100000000	Inappropriate function: peripheral code required a unit which is not in the configuration.
4300000000	Subsystem error occurred while processing the ASG statement.

3.2.8. Paper Tape Device Assignment

A request for assignment of a paper tape device can be satisfied by either the Standard Paper Tape Subsystem or the UNIVAC 1004 Subsystem. The request for paper tape device assignment is normally submitted by the ASG statement. The ASG statement contains the specifications required to ready the unit for operation.

- Format:

The ASG statement has the following general form:

#ASGƀoptionsƀperipheral name, file code, file name

- Options:

J Hold assignment for duration of job unless explicitly released by the FREE statement. Absence of the J option implies that the assignment will be released upon termination of the task.

V Implies that the assignment is optional for execution of the task. If the unit is available at selection time, assign it; otherwise, continue to task execution phase.

W Wait for operator response after file name is printed on console printer.

- Specifications:

Peripheral Name is a one- to five-character mnemonic used to name the type of unit to be assigned. See peripheral name (3.2.2) and systems generation (7.2) for additional use and specifications.

The standard mnemonics contained in the distributed system are:

MNEMONIC	ACTION TO BE TAKEN
PTIN	Assign a paper tape reader.
PTOUT	Assign a paper tape punch.

File Code is a one- to two-alphabetic character code used to link the assignment to task calls. Only one unit may be assigned to a particular file code.

File Name is an identifier which may be used for direction of the operator or for dedication of unit by the MT function.

3.2.8.1. METHOD OF SUBMISSION

Paper tape device assignment statements are normally submitted externally by the ASG statement through the control stream and/or collected with the task element to be scheduled. All ASG statements contained in the control stream should precede the task activation card to which they pertain.

Paper tape device requests can also be submitted internally during execution of the task. However, care must be used when making internal facility requests, especially with regard to unit assignments of any type. The OMEGA multiprogram scheduling algorithms select the task to be run on the basis of facilities expressed externally to the task. Hence, during execution, additional facilities may not be available due to their assignment to a concurrently operating task. In such a case, the internal request will not be satisfied.

Upon return of program control from an internal ASG request, the following status information is conveyed to the task in the A register:

Normal Completion: The register is set to positive, indicating that the unit is successfully assigned. The lower portion contains the peripheral type code, and the upper portion contains the subsystems number.

Abnormal Completion: No assignment is made and the register contains one of the following values:

5000000000 Requested unit is not available.

42000000000 Incorrect parameter: ASG statement could not be interpreted by the system.

41000000000 Inappropriate function: peripheral code required a unit which is not in the configuration.

43000000000 Subsystem error occurred while processing the ASG statement.

3.2.9. The FREE Statement

The FREE statement is used to release a peripheral device or allocated random access storage from the current task. The released facility is returned to the general facility pool. The FREE statement will override the hold option, J, used at the assignment time. A FREE statement will not release a unit or a direct access storage area registered with the Master File Directory.

The FREE statement is required only when the hold option was given on the original statement or it is desired to release a facility prior to task termination.

■ Format:

The FREE statement has the following general form:

#FREEb̸optionsb̸file code, file identifier

■ Options:

I Rewind the assigned tape unit with interlock before release.

L Rewind the assigned tape unit with interlock, and submit a message to the console printer instructing the operator to label the tape before release.

P .Perform the appropriate closing routine by either rewinding the tape unit or by punching three blank cards to clear the card punch.

■ Specifications:

File Code is a one- or two alphabetic character code used to define the unit or area of direct access storage being released (see 3.3.2). Once the release function is complete, the specified file code is available for subsequent assignments.

File Identifier is the name/version which will be recorded on the physical tape label by the console operator when the L option is specified. The maximum field length for the identifier is 15 characters including the slash (/).

3.2.9.1. METHOD OF SUBMISSION

The FREE statement may bs submitted dynamically during execution of the task or may be contained externally in the control stream. FREE statements contained in the control stream normally follow the END statement of the task being released and precede the control statements of the subsequent task. All FREE statements encountered between the task activation card and the END statement are processed upon occurrences when requests are made for primary input.

Upon return of program control to the requester the following status information is conveyed to the requesting task in the A register:

Normal Completion: The register is set to binary 0 which indicates that the assignment release was successfully completed.

Abnormal Completion: No release function was performed and the register contains one of the following values dependent upon the cause:

4100000000	Inappropriate function: The file code was not found.
4200000000	Incorrect parameter: The FREE statement could not be interpreted by facility assignment, or the file was registered with the Master File Directory.
4300000000	Subsystem error occurred while processing the FREE statement.

3.2.9.2. EXAMPLES

The following are examples of the use of the FREE statement:

```
#FREEƀƀC
#FREEƀIƀFF
#FREEƀLƀN, PERMANENT/FILES
```

In the first example, the unit or storage area with the file code C is released. In the second example, the tape unit with the file code FF is rewound with interlock and released. In the third example, the tape unit with file code N is rewound with interlock and released, and the reel is labeled PERMANENT/FILES by the operator.

3.2.10. The SWITCH Statement

The SWITCH statement is used to change the logical file code reference used by the task to the physical assignment. This feature allows for establishing an equivalence between dispatcher file code references for successive tasks utilizing the same physical assignment.

The switch is performed by interchanging assignments of two distinct file codes.

- Format:

  The SWITCH statement has the following general form:

  #SWITCHƀoptionsƀfile code, file code

- Options:

  R Rewind without interlock the tape units associated with either or both file codes.

- Specifications:

  File Code comprises two fields which contain, one code each, the one- or two-alphabetic character file codes being exchanged. Both file codes may be single letter assignment, both may be double letter assignment, or one single letter and one double letter code may be used. Only one file need be previously assigned.

3.2.10.1. METHOD OF SUBMISSION

The SWITCH statement may be submitted externally by the control stream and/or collected with the task element, or may be submitted dynamically during execution of the task. The SWITCH statements contained in the control stream normally precede the task activation card to which they pertain.

Upon return of program control to the requester the following status information is conveyed to the task in the A register:

Normal Completion: The register is set to binary 0 which indicates that the exchange of file codes is successfully completed.

Abnormal Completion: The requested exchange was not made and the register contains:

4200000000	Incorrect parameter: the SWITCH statement could not be interpreted by facility assignment. Neither file code was assigned, or one code was assigned to communications.

3.2.10.2. EXAMPLES

The following are examples of file code interchange using the SWITCH statement:

```
    #SWITCHƀƀA, Y
    #SWITCHƀRƀWF, J
    #SWITCHƀƀBA, BZ
```

In each example, exchange of physical assignments will be made for both file codes. In the second example, the tape units assigned to either or both file codes will also be rewound without interlock.

3.2.11. Service Requests

Facility Assignment service requests allow the user to test file codes, release or replace areas of a direct access storage file, or expand a direct access storage file dynamically during task execution. The direct access storage file must be previously established by an ASG statement, or by the MFD statement (see 3.5.1). If the file is registered with the Master File Directory, facility assignment will automatically recatalog the file. The following sections provide a description of each service request and include examples of their use.

3.2.11.1. TEST FILE CODE (TFC$)

The Test File Code (TFC$) service request is used to determine the type of peripheral device which has been assigned to a specific file code.

The operator for the Test File Code service request is as follows:

TFC\$ƀv_0

v_0 is the file code to be tested.

The coding generated for the TFC$ service request is as follows:

ENT*B7	v_0
EXRN	20116

Upon completion of the request, control is returned to the requesting activity with the status and information contained in the A register:

00000000XX Normal Completion: The lower six bits of the register contain the type code of the peripheral unit assigned to the file code as given below.

If the assignment has been made to direct access storage, the Q register contains the number of words assigned to the file.

5000000000 No assignment: The file code referenced has not been assigned.
For 8414 peripheral types, the record size of the file will be returned in B7.

The peripheral type codes returned in response to this device request are as follows:

Direct Access Subsystems:

00	FH-432 Magnetic Drum
01	FH-880 Magnetic Drum
02	FH-1782 Magnetic Drum
03	8414 Disc
04	Fastband II Mass Storage
05	FASTRAND II Mass Storage
06	Fastband III Mass Storage
07	FASTRAND III Mass Storage

Magnetic Tape Units:

10	UNISERVO III C Magnetic Tape
11	UNISERVO IV C Magnetic Tape
12	UNISERVO VI C Magnetic Tape
13	UNISERVO VIII C Magnetic Tape
14	UNISERVO III A Magnetic Tape
15	UNISERVO II A Magnetic Tape
16	UNISERVO VIII C in 9-track mode
17	UNISERVO 16 Magnetic Tape

Unit Record Devices:

20	High Speed Card Reader
21	High Speed Card Punch
22	Multiple High Speed Printer
23	High Speed Printer
24	UNIVAC 1004 Card Reader
25	UNIVAC 1004 Card Punch
26	UNIVAC 1004 Printer
27	Unassigned
30	High Speed Paper Tape Reader
31	High Speed Paper Tape Punch
32	UNIVAC 1004 Paper Tape Reader
33	UNIVAC 1004 Paper Tape Punch
34	UNIVAC 9300 Card Reader
35	UNIVAC 9300 Column Card Punch
36	UNIVAC 9300 Printer
37–39	Unassigned
40	UNIVAC 9300 Row Card Punch
41	UNIVAC 9300 High Speed Printer
42	UNIVAC 9300 Paper Tape Reader
43	UNIVAC 9300 Paper Tape Punch

3.2.11.2. PARTIAL FILE RELEASE (FREL$ AND FRELA$) .

Areas of a direct access storage file may be released dynamically during task execution by means of the following macros:

$\mathrm{FREL\$}\cancel{b}v_0,v_1,v_2$

v_0 is the file code under which the file has been assigned.

v_1 is the number of words that are to be released.

v_2 is the logical increment of the first word to be released.

The coding generated in response to the FREL$ macro is as follows:

ENT*B7	v_0
ENT*Q	v_1
ENT*A	v_2
EXRN	20262

$\mathrm{FRELA\$}\cancel{b}v_0,v_1,v_2$

v_0 is the file code under which the file has been assigned.

v_1 is the number of words that are to be released.

v_2 is the logical increment of the first word to be released.

The coding generated in response to the FRELA$ macro is as follows:

ENT*B7	v_0
ENT*Q	v_1
ENT*A	v_2
EXRN	20263

When the file area to be released is not at the end of the file, the FREL$ operator will create an inactive or hidden file segment for the area released. The logical addressing scheme for the portion of the file following the released area will remain unchanged. Any attempt to reference this released area will result in a "hidden file segment" status (see 3.3.4).

The operation of FRELA$ is similar to FREL$ with one exception. FRELA$ does not build a hidden file segment for the area that has been released. The result is that areas of the file following the area released by FRELA$ are now referenced with a logical increment equal to the old logical increment minus the number of words released. See 3.2.11.5 for examples of FREL$ and FRELA$.

Upon completion of the operations, control is returned to the requester with the following status information conveyed in the A register:

Normal Completion: The register is set to binary 0 which indicates the partial release has been successfully completed.

Abnormal Completion: The partial release was not performed and the register contains one of the following values:

4100000000	Inappropriate function: The file was not assigned to direct access storage, or the file had write protection.
42000000000	Incorrect parameter: The file code was not in the eligible set of file codes, or the file code was not assigned.
43000000000	Subsystem error: An unrecoverable subsystem error has occurred during the processing of the file.
44000000000	End-of-file: The logical increment specified was not contained in the file.

3.2.11.3. FILE EXTENSION (FADD$ AND FADDA$)

A mass storage file previously established by means of the ASG or MFD statements may be dynamically expanded during task execution by the following macros:

FADD$ƀ$v_0,v_1$

v_0 is the file code under which the file has been assigned.

v_1 is the number of words that ate to be added to the file.

The coding generated in response to the FADD$ macro is as follows:

ENT*B7	v_0
ENT*Q	v_1
EXRN	20260

FADDA$ƀ$v_0,v_1$

v_0 is the file code under which the file has been assigned.

v_1 is the number of words that are to be assigned to the file.

The coding generated in response to the FADDA$ macro is as follows:

ENT*B7	v_0
ENT*Q	v_1
EXRN	20261

Any area previously released within the file by the FREL$ service request (see 3.2.11.2) will be searched first to determine if any such area will satisfy the expansion request. If there is such an area it will be reassigned even though it may be larger than the requested area. If a previously released area will not satisfy the request, the required area is added to the end of the file. (See 3.2.11.5 for examples of FADD$.)

The operation of the FADDA$ request is similar to the FADD$ request, with one exception. The FADD$ request will obtain additional direct access storage from the same subsystem as the previous Random Access Storage List (RASL) entry. The FADDA$ request obtains additional direct access storage from the same subsystem, if possible, or from a subsystem with an equal or greater peripheral type code (see 3.2.11.1) than that of the previous RASL entry. (See 3.2.11.5 for examples of FADD$.)

Upon completion of the operation, control is returned to the requester with the following status information conveyed in the A and Q registers:

Normal Completion: A is set to the logical increment relative to the file base of the extension. Q contains the number of words assinged.

Abnormal Completion: No assignment is made and A register contains one of the following values, dependent upon the cause:

4100000000	Inappropriate function: The file was not assigned to random access storage, or the file had write protection.
4200000000	Incorrect parameter: The file code was not in the eligible set of file codes, or file code was not assigned.
4300000000	Subsystem error: An unrecoverable subsystem error has occurred during processing of the file.
5000000000	The direct access storage required to replace the *bad* area was not available.

If the request was initiated on an image (duplex) file, the Q register is negative if the error was detected on the primary file (file 1); the Q register will be positive if the error condition was detected on the secondary file (file 2).

3.2.11.4. USER DECLARATION OF "BAD" FILE AREA (FRPL$ AND FRPLI$)

If a user receives an unrecoverable subsystem error status on a primary file (file 1) FASL, the file area involved can be returned to the system to be held from further assignment. The following macro is used:

FRPL$b̸$v_0$,$v_1$,$v_2$

v_0 is the file code under which the file has been assigned.

v_1 is the number of words found to be bad.

v_2 is the logical increment for the beginning of the bad area.

The coding generated in response to the FRPL$ macro is as follows:

ENT*B7	v_0
ENT*Q	v_1
ENT*A	v_2
EXRN	20264

If a user receives an unrecoverable subsystem error status on a secondary file (file 2) RASL, the file area involved can be returned to the system to be held from further assignment. The following macro is used:

FRPL1\$b̸v_0,v_1,v_2

v_0 is the file code under which the file has been assigned.

v_1 is the number of words found to be bad.

v_2 is the logical increment of the beginning of the bad area.

The coding generated in response to the FRPL1\$ macro is as follows:

ENT*B7	v_0
ENT*Q	v_1
ENT*A	v_2
EXRN	20265

The File area that has been declared bad will be replaced with a different area of direct access storage from the same subsystem, if possible, as the declared bad area or from an area with an equal or greater peripheral type code. Since direct access storage is assigned by a multiple of the block size used to map the subsystem (see 3.2.4), only multiples of the block size can be replaced. Thus, area adjacent to the bad area may be replaced with a different area of mass storage. The actual area replaced is conveyed to the requester when the operation is completed. (see 3.2.11.5 for an example of the FRPL\$ service request.)

It is the responsibility of the requester, then, to be able to re-create the file area that will be replaced.

Upon completion of the operation, control is returned to the requester with the following status information conveyed in the A and Q registers:

Normal Completion: A is set to the logical increment relative to the file base of the actual area replaced. Q contains the number of actual words replaced.

Abnormal Completion: The operation was not performed and the A register contains one of the following values:

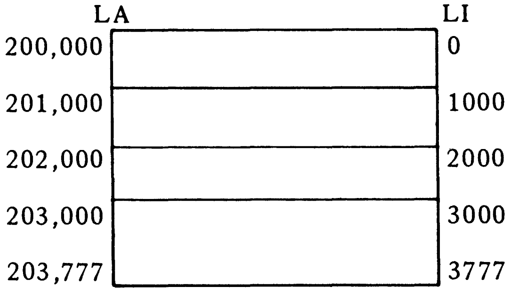
4100000000	Inappropriate function: The file was not assigned to direct access storage, or the file had write protection.
4200000000	Incorrect parameter: The file code was not in the eligible set of file codes, or the file code was not assigned.
4300000000	Subsystem error: An unrecoverable subsystem error has occurred during processing of the file.
4400000000	End-of-File: The logical increment specified was not contained in the file.
5000000000	The direct access storage required to replace the bad area was not available.
5200000000	Hidden file segment: The request specified a hidden file area.

3.2.11.5. EXAMPLES

This section provides examples of the service requests honored by facility assignments: FREL$, FRELA$, FADD$, and FRPL$. In all of the examples, a module size of 1000 words is inferred.

EXAMPLE 1 – FREL$

Assume a direct access storage file, FILE 1, assigned to file code A with the following logical address (LA) and logical increment (LI) scheme:

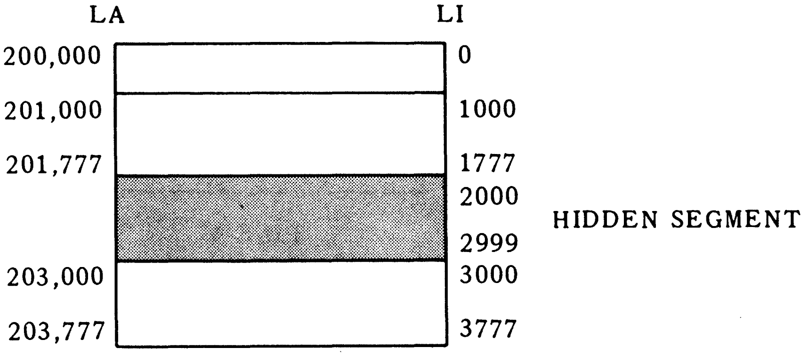


FILE 1

The service request

FREL$ A, 1000, 2000

will generate the following file structure:



FILE 2

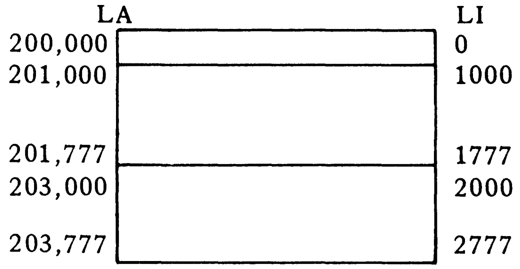
EXAMPLE 2 – FRELA$

Assume the direct access storage file, FILE 1 from Example 1, assigned to file code A.

The service request

FRELA$ A, 1000, 2000

will generate the following file structure:



FILE 3

NOTE: With FRELA$ no hidden segment is created and logical addressing scheme has changed.

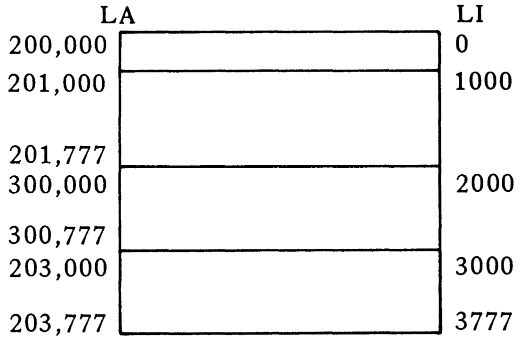
EXAMPLE 3 – FADD$ Using Hidden Segment

Assume the direct storage file, FILE 2 from Example 1, assigned to file code A.

The service request

FADD$ A, 1000

will generate the following file structure:



FILE 4

The status returned to the requester in the A and Q registers would be 2000 and 1000, respectively, showing an addition of 1000 words at Logical Increment 2000. The area previously hidden was equal to the request, and, therefore, another module was obtained (LA=300,000 and LI=2000).

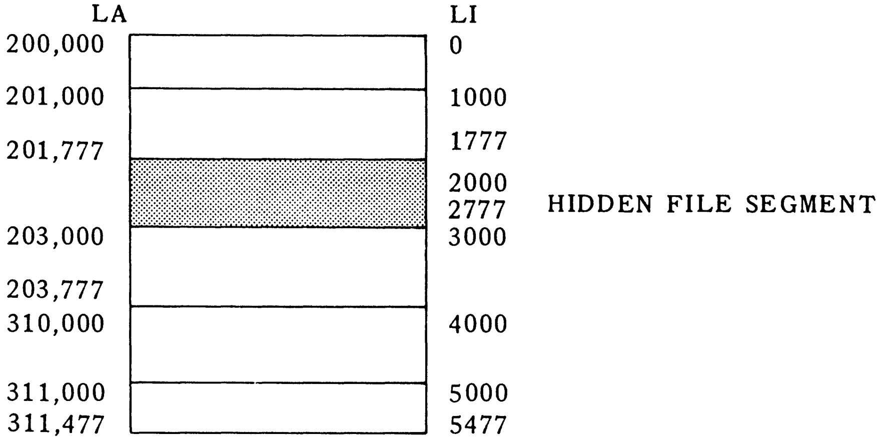
EXAMPLE 4 – FADD$

Assume the direct access storage file, FILE 2 from Example 1, assigned to file code A.

The service request

FADD$ A, 1500

will generate the following file structure:



FILE 5

The status returned to the requester in the A and Q registers would be 4000 and 1500, respectively, showing an addition of 1500 words at logical increment 4000.

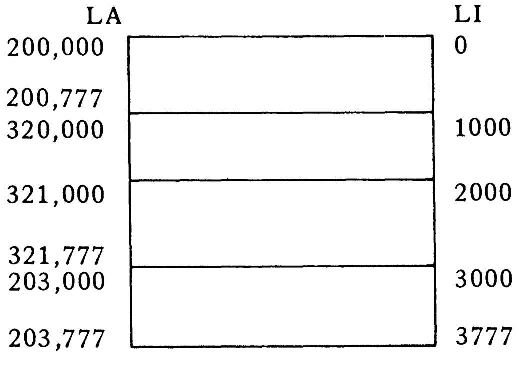
EXAMPLE 5 – FRPL$

Assume the direct access storage file, FILE 1 from Example 1, assigned to file code A. In attempting to read or write 200 words at logical increment 1700, an unrecoverable subsystem error status has been returned. A call is now made to facility assignment to replace the bad area with a new area and to keep the bad area from further assignment.

The service request

FRPL$ A, 200, 1700

will generate the following file structure:



FILE 6

The requester would receive a status in the A and Q registers showing 2000 words replaced at logical increment 1000 since the area to be released extended over two modules. The requester must now re-create the data originally contained in the replaced area.

3.3. DEVICE CONTROL

Device Control is a basic access method applicable to both randomly and sequentially arranged data. Physical characteristics of specific peripheral equipment may be recognized with this method, for more efficient utilization than is generally possible with higher level access methods. In addition, it is possible to maintain a degree of device independence, the extent of which is determined by the programmer. The programmer does not require detailed knowledge of specific input/output devices.

Data access using the device control method is accomplished by specifying certain macro instructions within the user program. The macros cause the generation of a parameter list and a function code which completely define the input/output operation to be performed. The parameter list may be changed dynamically by the user to effect a variation in blocking, buffering, and other operations.

The input/output operation is scheduled by the operating system at the time that the request is received from the user program. The parameter list is checked for validity by the system, and if all is found to be in order, the operation is performed. At the completion of the operation, control is returned to the requester with an indication of the success or failure of the operation. Supplementary information may also be available at this time, depending upon the function performed.

The system will attempt to recover from all error conditions encountered during the performance of an input/output request. The data block will be reread (or rewritten) a number of times in an attempt to perform the operation without error. Only when all system recovery attempts fail to produce a normal response from the device will an error condition be reported to the requester. The user may then take any action desired (i.e., attempt his own recovery, use the data block in spite of the error, ignore the block, abort, etc.)

3.3.1. Data Format Considerations

The device control method of data access deals with physical data blocks rather than with individual logical records. The data blocks may consist of any number of fixed or variable size records. The data blocks themselves may be of fixed or variable size. Initial data block size is specified by the user through use of the macro instructions: the block size may, however, be varied dynamically. For output operations, the block size represents the number of words to be transferred to the output device. For input operations, the block size represents the maximum number of words to be accepted from the input device. The user is informed by the system of the number of words transferred so that the *short blocks* may be successfully processed.

Buffer location must also be specified by the user through the macro instructions. The buffer areas must be contained within the PLR limits of the requester in order to avoid conflict when the data area is referenced. Data is transferred directly to and from the buffer area; thus, the buffer may also serve as a work area.

3.3.2. File Codes

The data set which is to be referenced by a particular input/output request is specified by a file code contained in the parameter list generated by the input/output macro instruction. This file code defines a unique unit or area of direct access storage which has been assigned to the task. The type of secondary storage assigned to the file code may be specified in the job control stream; therefore, the peripheral type may be varied for different executions of the same task.

Reference may be made to the peripheral unit only by file code, never by channel and unit. This procedure makes it impossible for a task to reference a peripheral unit which has not been assigned validly to the task through the operating system. Direct access storage is referenced through a file code and also through a logical address. The file code refers to an area (or areas) of storage that has been assigned to the task, and the logical address refers to the position within the assigned area. Thus, two different tasks operating in multiprogrammed environment may be using the same file code and logical address, and yet be referencing two completely distinct areas.

Two activities which operate asynchronously in the same task may reference the same secondary storage medium by using identical file codes. This occurrence, however, is under the control of the programmer, since the programmer has established the asynchronous activities. The file code does protect a task's secondary storage from destruction by other tasks.

3.3.3. Device Control Macros

The device control macros are source statements placed within the program by the user. From the information contained the macro statement, a parameter list and the necessary control instructions are generated to perform the input/output operation specified. The general format of the macro statement contains an operator and a specification list. The operator defines the function to be performed, and the specification list contains the parameters necessary to perform the function. The specification list includes such items as file code, buffer location and size, logical address, and search identifier. The general format is given in the following paragraphs.

- Format

  The device control macro has the following general form:

  operators\$ƀv_0,v_1,v_2,v_3,v_4

The operator specifies the function to be performed such as read, write, or search. The operator field always terminates with the control character \$. The specification list is defined by its component operands, v_0,v_1,v_2,v_3,v_4 which must be separated by commas. The operator and the operands of the specification list are separated by a space (ƀ).

- Specifications

  v_0 File Code is a one- or two-character alphabetic code which determines the location of the data set to be referenced. The location of the data must be defined previously by the ASG statement.

  v_1 Buffer size may be iether an octal or decimal constant, or a tag which is equated to an octal or decimal constant by an EQUALS statement. The buffer size is limited to a maximum of fifteen bits.

  v_2 Beginning of buffer area is specified by a tag which also must appear at a label in the coding or as an entry definition (EDEF) at collection time. The rag may be modified by an octal or decimal constant but *not* by an index register.

  v_3 Logical increment is used for direct access storage requests to indicate the increment, from the base of the file, defining the address at which the operation is to be performed. This operand is specified by an octal or decimal constant or by a tag which is equated to an octal or decimal constant by an EQUALS statement. If this operand is missing, the logical increment will be generated as 0.

  v_4 Search identifier is a numeric value consisting of ten octal characters which is the value that is to be matched in a search operation. The operand may be omitted if the operation is not a search.

  See Table 3–3 for a summary fo device label input/output macros.

The general format of the service request and parameter list generated in response to the menmonic is as follows:

	EBJP*B7	14 N 0
	v_0	14 v_1 0
		17 v_2 0
	29 v_3 (if specified) 0	
	29 v_4 (if specified) 0	
N →	EXRN	FUNCTION CODE

The Enter B and Jump (EBJP) instruction brings the address of the parameter list into the B7 register and then jumps to the Executive Return (EXRN) instruction. The address entered into B7 by the hardware is relative to the value of the RIR at the time of execution. File code (v_0) is Fieldata coded and right justified in the upper halfword. Buffer size (v_1) is specified with fifteen bits; buffers larger than the hardware maximum of 4096 words will be automatically handled by the system as scatter/gather operations. The buffer base (v_2) is considered an eighteen-bit value relative to the value of the PLR lower at the time of submission. The logical increment (v_3) position in the list is always present even if the operand is missing in the macro, in which case the value will be set to 0. The search identifier (v_4) position will not be generated if the operand is missing from the macro. The lower half of the EXRN instruction determines the operation to be performed. The EXRN value depends upon the operator used in the macro.

The buffer which is to be used in the operation is completely specified by the v_1 and v_2 operands. The entire buffer must be contained within the upper and lower limits of the program as defined by the PLR value at the time of submission. Violation of this rule constitutes an error condition, and the requested operation will not be performed.

An exception to the above packet description is the use of V_4 to specify a parameter for mass storage logical lock requests. In this case, the packet is generated as follows:

ENT*A	14 v_4 0
EBJP*B7	14 N 0
v_0	14 v_1 0
	17 v_2 0
29 v_3 0	
EXRN	FUNCTION CODE

Otherwise, the packet is interpreted as explained previously.

The location of the parameter list, as contained in B7 at the time of submission, is treated by the system as relative to the lower limit of the PLR. When executing the Enter B and Jump instruction, however, the hardware enters the value relative to the RIR. This means that common subroutines and other such programs which operate with different values in the RIR and the PLR lower may not use this type of *inline* packet generated by the macro instructions. In this contingency, the parameter list may be *remote* to the control instructions. This method may be implemented by coding the parameter list as shown:

<table>
<tr><td>LABEL</td><td colspan="2">v_0</td><td>v_1</td></tr>
<tr><td></td><td></td><td colspan="2">v_2</td></tr>
<tr><td></td><td colspan="3">v_3</td></tr>
<tr><td></td><td colspan="3">v_4 (if required)</td></tr>
</table>

The execution of the operation may be effected through the following instructions:

ENT*B7	LABEL
EXRN	FUNCTION CODE

Where LABEL is the address, relative to the PLR lower, of the parameter list, and the function code determines the operation to be performed.

3.3.4. Status Codes

At the completion of each input/output operation, control is returned to the requester at the line of coding following the Executive Entry instruction of the macro call. At this time, the results of the operation are presented in the form of a status code in the A register. The number of words transferred will also be indicated in the A register. The Q register will contain supplementary information dependent upon the type of peripheral referenced. The information returned is given for each function in the following paragraph. If there is no supplementary information to be returned, the Q register will be 0. B7 will be altered on completion of an I/O operation and will contain the absolute address of the parameter list. ←

For all operations, the number of words transferred wwill be indicated in bit positions 17 through 0 of the A register. An abnormal status condition will be flagged by a bit in position 29 of the A register. The status code, when position 29 is set, will be contained in bit positions 28 through 24 of the A register. If the operation has been completed successfully, bit position 29 as well as bit positions 28–24 of the A register will be 0. The following illustrates the format of the A register:

29	28 24		17 0
	STATUS CODE		NUMBER OF WORDS TRANSFERRED

NOTE: i= abnormal indicator

The abnormal status code contained in bit position 28 through 24 will be one of the following:

- Inappropriate function (01)

  The function code is not applicable to the file (e.g., READ$ on a printer file). The function has not been executed; the number of words transferred will be 0. The Q register will also be 0.

- Incorrect parameter (02)

  A parameter error has been detected in the specification of the operation (e.g., buffer outside of the PLR limits, undefined function code, etc.). The operation has not been performed; the number of words transferred will be 0. The Q register will also be 0.

- Unrecoverable subsystem error (03)

  The requested operation has been attempted, but cannot be completed successfully because of a subsystem error (e.g., parity error, sequence error, etc.). System recovery procedures have been attempted, but have proved unsuccessful. The number of words successfully transferred is given in the lower 18 bits.

- End-of-file (04)

  An end-of-file mark has been detected on a magnetic tape read operation, or the end of allocated area has been reached on a direct access file read or write operation. The number of words transferred will be 0 for magnetic tape and will include the last word of the file for direct access file requests.

- End-of-tape (05)

  A write operation has been successfully completed beyond the tape limit mark, or the load point has been encountered during a READB operation. The number of words transferred will be indicated.

- Unsuccessful search (06)

  The search identifier could not be located within the specified area of the file. The number of words transferred will be indicated.

- Unsuccessful search (06)

  The search identifier could not be located within the specified area of the file. The number of words transferred will be 0.

- Illegal character (07)

  An illegal combination of punches (any combination of holes in rows 1, 2, 3, 4, 5, 6, 7, or 9) has been detected in a card column (translate mode only). The illegal character will appear as a 00 code in the buffer.

- No assignment (10)

  The file code referenced in the parameter list has not been assigned to a unit (or area) by the task. The function cannot be executed.

- Interlock (11)

  The operator has responded to the system interlock message indicating that the interlock condition cannot be corrected.

- Hidden file segment or block number sequence error (12)

  The packet parameters have defined a data transfer into or from an inactive file segment on direct access storage, or a data transfer from a tape file for which the next block sequence number cannot be located. The inactive segment is called hidden (the segment is bounded by active file segments), and is the result of a program releasing one or more words of storage within a file. The block sequence error occurs when numbering has been specified for a tape block and numbering has not occurred, or when some portion of the tape may have been destroyed. The A register will contain the number of words transferred, and will include; the word preceding the hidden segment, when applicable.

FUNCTION CODE	OPERATOR	V-OPERANDS REQUIRED FOR DIRECT ACCESS	V-OPERANDS REQUIRED FOR MAGNETIC TAPE	V-OPERANDS REQUIRED FOR UNIT RECORD	NOTES
10001	READ$	0, 1, 2, 3	0, 1, 2	0, 1, 2	
10002	WRITE$	0, 1, 2, 3	0, 1, 2	0, 1, 2*	
10003	BLOCKR$	0, 1, 2, 3	N/A	N/A	Drum Subsystem only
20044	SEARCHT$	0, 1, 2, 3, 4	N/A	N/A	FASTRAND only
20045	SEARCHP$	0, 1, 2, 3, 4	N/A	N/A	FASTRAND only
20046	SEARCH$	0, 1, 2, 3, 4	0, 1, 2, 3, 4	N/A	
20047	BLOCKS$	0, 1, 2, 3, 4	N/A	N/A	Drum only
10010	READB$	N/A	0, 1, 2	N/A	No action on direct access (successful status)
10011	READL$	0, 1, 2, 3	N/A	N/A	
10012	WRITER$	0, 1, 2, 3	N/A	N/A	
10013	LOCKR$	0, 1, 2, 3, 4	N/A	N/A	Drum only
10014	WRELS$	0, 1, 2, 3	N/A	N/A	Drum only
10015	RDTS$	0, 1, 2, 3, 4	N/A	N/A	Drum only
10016	WRTS$	0, 1, 2, 3, 4	N/A	N/A	Drum only
20056	SEARCHL$	0, 1, 2, 3, 4	N/A	N/A	
10020	WRTEOF$	N/A	0	N/A	No action on direct access
10021	REWIND$	N/A	0	N/A	No action on direct access
10022	REWINDI$	N/A	0	N/A	No action on direct access
10023	ERASE$	N/A	0	N/A	No action on direct access
10024	SREAD$	LISTA$	LISTA$	N/A	
10025	GWRITE$	LISTA$	LISTA$	N/A	
10026	MREAD$	LISTB$	LISTA$	LISTA$	N/A on paper tape
10027	MWRITE$	LISTB$	N/A	N/A	Drum only
10030	MREADL$	LISTB$	N/A	N/A	Lock as with LOCKR$
10031	MWRITER$	LISTB$	N/A	N/A	Lock as with WRELS$
10300	LRASL$	0, 1, 2, 3, 4	N/A	N/A	
10302	LOCKA$	0, 1, 2, 3, 4	N/A	N/A	

N/A = Not Applicable

*For printer subsystems, the v_1 operand is the number of lines to space before printing.

Table 3–3. Summary of Device Level Input/Output Macros

For magnetic tape read functions, an abnormal frame count is not considered an error condition, that is, bit positions 29 through 24 of the A register will be 0. The magnitude of frame count error will always be displayed in bit positions 29 through 24 of the Q register (00 indicates no frame count error, 01 indicates that one extra frame is significant, etc.) The number of words transferred as contained in the A register will include the partial word.

In addition, a number of status codes are returned only by the mass storage handler. For a list of these special codes, see 3.3.6.3 in *UNIVAC 494 Real-Time System Random Storage File Handler, UP-7633* (current version).

3.3.5. Drum Macros

The following functions are available to users for input/output on drum subsystems (FH-880, FH-1782, FH-432, FH-432/1782, FASTRAND II Mass Storage and FASTRAND III Mass Storage).

3.3.5.1. READ$

Data will be read from the drum beginning at the address determined by the file code and logical increment. The data will be placed in consecutive words of primary storage beginning at the buffer base specified. Data transfer will continue until the specified number of words have been transferred.

The macro call and generated code are:

READ$ƀfile code, number of words, buffer base, logical increment

	EBJP*B7	N	
	FILE CODE	NO. OF WORDS	
		BUFFER BASE	
	LOGICAL INCREMENT		
N →	EXRN	10001	Executive Entry Instruction

3.3.5.2. WRITE$

The contents of the specified buffer will be transferred to the drum area determined by the file code and the logical increment.

The macro call and generated code are:

WRITE$ƀfile code, number of words, buffer base, logical increment

	EBJP*B7	N
	FILE CODE	NO. OF WORDS
		BUFFER BASE
	LOGICAL INCREMENT	
N →	EXRN	10002

3.3.5.3. SEARCH$

Each word of the file, beginning at the logical increment specified, is compared with the search identifier. When a find is made, a read operation is initiated beginning with the find address.

In order to prevent prolonged subsystem lockout when a drum search is initiated, the system simulates the search function through the use of repetitive read commands. The buffer specified in the search parameter list is used by the system; thus, the efficiency of the search will depend upon the buffer supplied by the user.

The macro call and generated code are:

SEARCH$ƀfile code, number of words, buffer base, logical increment, search identifier

	EBJP*B7	N	
	FILE CODE	NO. OF WORDS	
		BUFFER BASE	
	LOGICAL INCREMENT		
	SEARCH IDENTIFIER		
N →	EXRN	20046	Executive Entry

3.3.5.4. BLOCK READ (BLOCKR$)

Data will be read from the drum beginning at the specified address and will continue until either the buffer is filled, an end-of-block (EOB) sentinel (a word of all 1 bits) is encountered, or an abnormal termination is indicated. This function should be used only with continuous, permanent files. If the area of the file is discontinuous, the situation could arise where the EOB sentinel is the last word of a continuous segment of the file. In this case, the overflow word obtained from the hardware would not be the overflow word written by the user.

It is also recommended that the input buffer be of sufficient length to include the overflow word. The external interrupt containing the overflow may not be presented to the subsystem if a monitor interrupt is generated when the EOB statement sentinel is accepted.

The recommended values for overflow words are 1 through 77777777_8. The macro call and generated code are as follows:

BLOCKR$ƀfile code, number of words, buffer base, logical increment

	EBJP*B7	N
	FILE CODE	NO. OF WORDS
		BUFFER BASE
	LOGICAL INCREMENT	
N →	EXRN	10003

3.3.5.5. BLOCK SEARCH (BLOCKS$)

Beginning at the logical increment specified, each consecutive word of the file is compared with the search identifier. The action taken when a find is made will depend upon the buffer size specified in the parameter list. If the buffer size (number of words) is 0, the operation will be a locate; the find address will be returned, but no data will be transferred. If the buffer size is nonzero, the operation will be a search read: data will be transferred and the overflow word may be returned, but the address of the find is unavailable.

To prevent transferring data from an unknown file, this function is executed in two parts, SEARCH and BLOCK READ, when the buffer size is nonzero.

This function should only be used with files that occupy one continuous area of the drum. The overflow word provided by the hardware may be incorrect if the file is segmented.

The macro call and generated code for the block search function are:

BLOCKS$ƀfile code, number of words, buffer base, logical increment, search word

	EBJP*B7	N
	FILE CODE	NO. OF WORDS
		BUFFER BASE
	LOGICAL INCREMENT	
	SEARCH IDENTIFIER	
N→	EXRN	20047

3.3.5.6. SCATTER READ (SREAD$)

Data will be read from the drum, beginning at the logical increment specified, and be placed into the noncontinuous primary storage buffers indicated by the user. The first buffer specified will be initiated; when this buffer terminates, the next buffer will be established by the system. This process will continue until the last buffer has been filled. An unrecoverable parity error will terminate the operation at that point; the total number of words read successfully will be returned.

In order to prevent skipping drum revolutions when using this function, small buffers (except the last buffer) should be avoided. The minimum buffer size that may be used without skipping revolutions is variable, depending upon the transfer rate of the drum and upon the system load at execution time.

Since multiple buffers may be given with this function, the parameter list is generated separately from the control statements.

The macro and generated code for the control statement are:

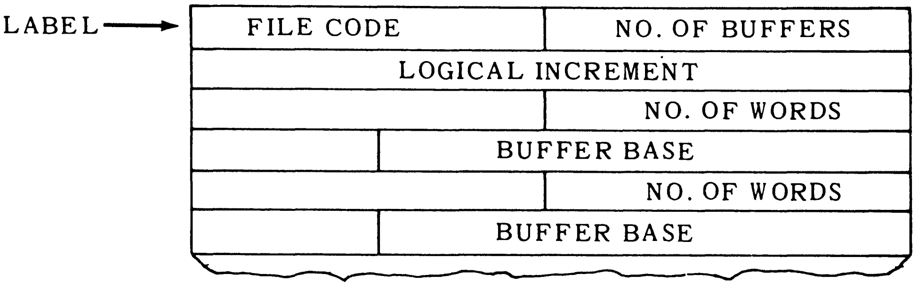
SREAD$ƀ$v_0$

ENT*B7	v_0
EXRN	10024

v_0 is the label of the parameter list.

The parameter list may be generated by using the LISTA operator. This macro call and generated code are as follows:

LABEL→ LISTA$ƀfile code, number of buffers, logical increment, number of words/buffer base, number of words/ buffer base, etc.



LABEL →

FILE CODE	NO. OF BUFFERS
LOGICAL INCREMENT	
	NO. OF WORDS
	BUFFER BASE
	NO. OF WORDS
	BUFFER BASE

3.3.5.7. GATHER WRITE (GWRITE$)

Data from discontinuous primary storage buffers will be placed on the drum in one logically continuous area. The first buffer specified in the parameter list will be written at the logical increment given; when this buffer terminates, the next buffer will be established and written at the next drum address. This will continue until the last buffer is emptied. An unrecoverable subsystem error will terminate the operation at that point; the total number of words successfully written will be returned to the requester.

In order to avoid a complete drum revolution between the establishment of the buffers of this operation, small buffers (except the last buffer) should not be specified. The minimum buffer size is variable, depending upon the transfer rate of the subsystem, and upon the system load at execution time. The macrocall and generated code for the control statement are:

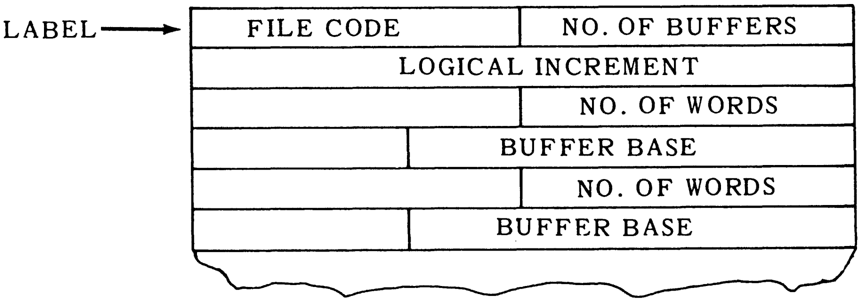
GWRITE$ƀ$v_0$

ENT*B7	v_0
EXRN	10025

v_0 is the label of the LISTA$ operator.

The parameter list for this operation is generated from the LISTA$ operator. This macro call and the code generated are as follows:

LABEL LISTA$ƀfile code, number of buffers, logical increment, number of words/buffer base, number of words/buffer base, etc.



LABEL →

FILE CODE	NO. OF BUFFERS
LOGICAL INCREMENT	
	NO. OF WORDS
	BUFFER BASE
	NO. OF WORDS
	BUFFER BASE

3.3.5.8. MULTIPLE READ (MREAD$)

The multiple read function will transfer data from discontinuous logical file areas into distinct primary storage buffers. A logical increment must be supplied for each area of the file to be read, and a primary storage buffer must be associated with each logical increment specified. Data is read beginning at the first logical increment and continues until the first buffer has terminated; the next read operation will begin at the second logical increment, and data is transferred into the second buffer, and the process continues to the end of the operation.

The macro call and the code generated for the control statement are:

MREAD$ƀ$v_0$

ENT*B7	v_0
EXRN	10026

v_0 is the label of the LISTB$ operator.

The parameter list for this function is specified with the LISTB$ operator. The macro call and code generated are:

LABEL → LISTB$ƀfile code, number of buffers, number of words/buffer base/logical increment,
number of words/buffer base/logical increment, etc.

The parameter packet generated is as follows:

LABEL →

FILE CODE	NO.OF BUFFERS
	NO. OF WORDS
	BUFFER BASE
LOGICAL INCREMENT	
	NO. OF WORDS
	BUFFER BASE
LOGICAL INCREMENT	

3.3.5.9. MULTIPLE WRITE (MWRITE$)

The multiple write function will transfer data from distinct primary storage buffers into discontinuous logical file areas. A logical increment must be supplied for each file area which is to be written, and a primary storage buffer must be associated with each logical increment specified. Data will be trans- ferred from the first buffer, beginning at the first logical increment, until the buffer has terminated; the second write operation will transfer data from the second specified buffer, beginning at the second logical increment; and the process continues to the end of the write operation.

The macro and the coding generated are as follows:

MWRITE$ƀ$v_0$

ENT*B7	v_0
EXRN	10027

v_0 is the label of the LISTB$ operator.

The parameter list for the function is specified with the LISTB$ operator. The macro call and packet generated are as follows:

LABEL → LISTB$b̸file code, number of buffers, number of words/buffer base/logical increment, number of words/buffer base/logical increment, etc.

LABEL →

<table>
<tr><td colspan="2">FILE CODE</td><td>NO. OF BUFFERS</td></tr>
<tr><td colspan="2"></td><td>NO. OF WORDS</td></tr>
<tr><td></td><td colspan="2">BUFFER BASE</td></tr>
<tr><td colspan="3">LOGICAL INCREMENT</td></tr>
<tr><td colspan="2"></td><td>NO. OF WORDS</td></tr>
<tr><td></td><td colspan="2">BUFFER BASE</td></tr>
<tr><td colspan="3">LOGICAL INCREMENT</td></tr>
</table>

3.3.5.10. *READ LOCK (READL$)*

The READL$ function is identical to the read function except that a lock list is interrogated before the read operation is performed. If any part of the file area specified by the logical increment and buffer size of the parameter list appears on the lock list, the request will be delayed until the area has been removed from the lock list. If the file area specified by the read lock function does not appear on the lock list, the request will be performed without delay, and the area is placed on the lock list. This means that once an area has been read with the lock list, the area may not be read with the lock list again until the area has been released. However, the area may be read with the normal read function, READ$, at any time.

The lock lists are maintained in such a manner that a physical drum area. is locked rather than a logical file area. This. means that an area which may be referenced by different file codes and/or logical increments (overlapping files) will be locked to all requests when it is locked by one request.

The macro call and the code generated by the control statement are:

READL$b̸file code, number of words, buffer base, logical address

<table>
<tr><td colspan="2">EBJP*B7</td><td>N</td></tr>
<tr><td colspan="2">FILE CODE</td><td>NO. OF WORDS</td></tr>
<tr><td></td><td colspan="2">BUFFER BASE</td></tr>
<tr><td colspan="3">LOGICAL INCREMENT</td></tr>
<tr><td colspan="2">N → EXRN</td><td>1 0 0 1 1</td></tr>
</table>

3.3.5.11. *WRITE RELEASE (WRITER$)*

The WRITER$ function is identical to the normal function (WRITE$) except that the area which is written is removed from the lock list. When any part of the area of the drum defined by the logical address and buffer size falls within an area on the lock list, that area is removed from the list after the write operation has taken place. A request that has been delayed because of the locked area now may be executed. A locked area may be released without writing by specifying a buffer size of 0 and a logical address anywhere within the locked area.

A locked area may be written into any time without removing the lock by using the normal WRITE$ function. If the WRITE$ function is specified, and the area does not appear on the lock list, a normal write operation will take place.

The macro call and the coding generated by the control statement are:

WRITER$ƀfile code, number of words, buffer base, logical increment

	EBJP*B7	N
	FILE CODE	NO. OF WORDS
		BUFFER BASE
	LOGICAL INCREMENT	
N →	EXRN	10012

3.3.5.12. SEARCH LOCK (SEARCHL$)

The SEARCHL$ function is identical to the normal SEARCH$ function except that the area which is read is placed on the lock list. The area begins at the find increment, and the area length is equal to the buffer size specified in the parameter list.

In order to prevent prolonged subsystem lockout when a hardware search is initiated, the system simulates the search function through the use of repetitive read commands. The buffer specified in the search parameter list is used by the system; thus, the efficiency of the search will depend upon the buffer supplied by the user.

The macro call and the coding generated by the control statement are:

SEARCHL$ƀfile code, number of words, buffer base, logical increment, search identifier

	EBJP*B7	N
	FILE CODE	NO. OF WORDS
		BUFFER BASE
	LOGICAL INCREMENT	
	SEARCH IDENTIFIER	
N →	EXRN	20056

3.3.5.13. LOCK READ (LOCKR$)

The LOCKR$ function is identical to a normal READ$ operation except that the area to be read is placed on a lock list to prevent the area from being read or written upon by another activity before being released by the originating activity. Any activitity making a prohibited request is delayed until the lock is released.

Lock read is distinguished from READL$ in that READL$ delays only those activities making READL$ or WRITER$ requests; LOCKR$ delays all requests specifying a read or write operation.

The macro call and code generated for this request are as follows:

LOCR$ƀfile code, number of words, buffer base, logical increment, v_0

ENT*A	v_0
EBJP*B7	$+4
FILE CODE	NO. OF WORDS
	BUFFER BASE
LOGICAL INCREMENT	
EXRN	10013

v_0 is a parameter constant specifying which functions are being locked with this request:

If v_0 = 1, the area is locked to all WRITE$, WRITER$, WRELS$, MWRITE$, and MWRITEL$ requests.

If v_0 = 2, the area is locked to all READ$, READL$, LOCKR$, MREAD$, and MREADL$ requests.

If v_0 = 3, the area is locked to all of the above requests.

3.3.5.14. WRITE RELEASE (WRELS$)

Write Release function WRELS$ causes a normal write (WRITE$) operation while releasing any lock(s) created by the activity in control, and included in the area specified.

If the number of words specified is zero, the area is released, but no write operation is performed.

This request releases only locks created by READL$, LOCKA$, and MREADL$ requests.

The macro call and code generated by this request are as follows:

WRELS$b̸file code,number of words,buffer base,logical increment

EBJP*B7	$+4
FILE CODE	NO. OF WORDS
	BUFFER BASE
LOGICAL INCREMENT	
EXRN	10014

3.3.5.15. MULTIPLE READ LOCK (MREADL$)

The multiple read lock MREADL$ function is identical to a normal MREAD$ operation except that all areas which are read are placed on a lock list as with LOCKR$. If several levels of multiple read operations combine to form a contiguous lock area, this entire area may be released as though it had been locked by a single LOCKR$ request.

The locking procedure is performed before the read operations, so that if any specified areas have not been locked by another activity, the entire request is delayed, and no read functions are performed until the prohibiting lock is released.

The macro call and code generated for this request are as follows:

MREADL$ƀ$v_0$,$v_1$

ENT*B7	v_0
ENT*A	v_1
EXRN	10030

v_0 is the label of the LISTB$ operator.

The parameter list is specified with the LISTB$ operator as follows:

LABEL → LIST$Bƀfile code, number of buffers, number of words/base/logical increment, number of words/base/logical increment, etc.

The packet generated is as follows:

LABEL →	FILE CODE	NO. OF BUFFERS
		NO. OF WORDS
		BUFFER BASE
	LOGICAL INCREMENT	
		NO. OF WORDS
		BUFFER BASE
	LOGICAL INCREMENT	

v_1 is a parameter constant specifying which activities are to be locked by this request:

If v_1 = 1, the area is locked to all WRITE$, WRITER$, WRELS$, and MWRITER$ requests.

If v_1 = 2, the area is locked to all READ$, READL$, LOCKR$, and MREAD$ requests.

If v_1 = 3, the area is locked to all the above requests.

3.3.5.16. MULTIPLE WRITE RELEASE (MWRITER$)

Multiple write release function is identical to a normal multiple write (MWRITE$) operation except that any areas to be written are removed from the activity's lock list so that they may again be accessed by other activities. Only locks originally created by LOCKR$, LOCKA$, or MREADL$ requests are released.

The lock list is checked before any write function is performed, so that if any area to be written has been locked by another activity, the entire request is delayed, and no write operations are performed until the prohibiting lock is released.

The macro call and code generated for this request are as follows:

MWRITER$b̸$v_0$

ENT*B7	v_0
EXRN	10031

v_0 is the label of the LISTB$ operator.

The parameter packet for this request is generated using the LISTB$ operator as follows:

LABEL → LISTB$b̸ file code, number of buffers, numbers of words/buffer base/logical increment, number of words/ buffer base/logical increment etc.

The packet generated is as follows:

LABEL →

<table>
<tr><td colspan="2">FILE CODE</td><td>NO. OF BUFFERS</td></tr>
<tr><td colspan="2"></td><td>NO. OF WORDS</td></tr>
<tr><td></td><td colspan="2">BUFFER BASE</td></tr>
<tr><td colspan="3">LOGICAL INCREMENT</td></tr>
<tr><td colspan="2"></td><td>NO. OF WORDS</td></tr>
<tr><td></td><td colspan="2">BUFFER BASE</td></tr>
<tr><td colspan="3">LOGICAL INCREMENT</td></tr>
<tr><td colspan="3"></td></tr>
</table>

3.3.5.17. LOCK DEFINITION (LOCKA$)

The lock definition function is used to define or to redefine locks on partial files without performing a read or write operation. The function redefines any locks created by LOCKR$ or MREADL$ requests in this activity. Locks may be implemented for read, write, or both operations specified by a parameter constant. An attempt to relock an existing lock or to release an area that is already released has no effect. The macro call and code generated for this request are as follows:

LOCKA$b̸file code, number of words, buffer base, logical increment, v_0

<table>
<tr><td>ENT*A</td><td>v_0</td></tr>
<tr><td>EBJP*B7</td><td>$+4</td></tr>
<tr><td>FILE CODE</td><td>NO. OF WORDS</td></tr>
<tr><td></td><td>BUFFER BASE</td></tr>
<tr><td colspan="2">LOGICAL INCREMENT</td></tr>
<tr><td>EXRN</td><td>10302</td></tr>
</table>

Buffer base is specified to allow re-use of the packet for further I/O requests. In a normal LOCKA$ call, the buffer base will be specified as zero.

v_0 is the parameter constant specifying which functions are to be locked by this request:

If v_0 = 0, release area to all subsequent read and write operations.

If $v_{1,}$ = 1, lock area to all subsequent read operations, and release area to subsequent write operations.

If v_0 = 2, lock area to all subsequent write operations, and release area to all subsequent read operations.

If v_0 = 3, lock area to all subsequent read and write operations.

3.3.6. Direct Access Storage Image Filing

Image or duplex filing is available on all drum subsystems (FH-432, FH-880, FH-1782, FASTRAND II mass storage, and FASTRAND III mass storage), providing two copies of the same file on the same or different subsystems. The copies may be referred to as primary and secondary copies or images, or as file 1 and file 2, or by other distinctive names. The advantages of image filing are:

- Speed:

  While a write operation will take place at the speed of the slower subsystem, a read operation is executed at the speed of the faster subsystem. In addition, if a request to the faster subsystem would result in a long wait due to a long queue of requests, the read operations take place on the alternate subsystem, thereby providing shorter overall execution time for the request.

- File Protection:

  In the event of a subsystem failure, the alternate file image is available to restore valuable files. If a hardware error prevents the reading of a file, the system switches automatically to the alternate image, executes the read operation, and, if successful, rewrites and verifies the bad file image. Status is returned to the user to indicate the condition of the files.

Image files are assigned using a variation of the #ASG control card (see 3.2.4.5).

→ *3.3.6.1. IMAGE FILE MACROS (LRASL$, RDTS$, AND WRTS$)*

All drum macros described in section 3.3.5 are available for use on image files. In addition, several macros are available for image file maintenance.

Image File Control (LRASL$)

The LRASL$ function controls the use of image files. The function may be used to inhibit read and/or write operations on either or both copies of an image file. In addition, the request may be used to test for the existence of an image file and to test which specific inhibit functions are in force.

The macro call and code generated for this request are as follows:

LRASL$b̸file code v_0

ENT*A	v0
ENT*B7	FILE CODE
EXRN	10300

v_0 is a 15-bit parameter constant indicating which inhibit functions or tests are being requested, as follows:

Bit 0 = 1: Inhibit read function on primary image.

Bit 1 = 1: Inhibit write function on primary image.

Bit 2 = 1: Inhibit read function on secondary image.

Bit 3 = 1: Inhibit write function on secondary image.

Bits 4 and 5 are not used.

Bit 6 = 1: Release inhibit read function on primary image.

Bit 7 = 1: Release inhibit write function on primary image.

Bit 8 = 1: Release inhibit read function on secondary image.

Bit 9 = 1: Release inhibit write function on secondary image.

Bits 10 and 11 are not used.

Bit 12 = 1: Test the existence of a primary image of the file.

Bit 13 = 1: Test the existence of a secondary image of the file.

Bit 14 = 1: Use bits 0 through 3 to test the specific inhibit conditions, instead of the operation indicated.

For example, the call:

LRASL$ƀD, 11

sets bits 0 and 3, inhibiting write operations on the secondary image and read operations on the primary image of the file assigned to file code D.

Immediately following this, the call:

LRASL$D, 40017

tests inhibit functions to determine which functions were in force, and, if the first example was executed correctly, returns the value II in the A register, indicating that read operations are inhibited on the primary image, and write operations are inhibited on the secondary image.

Upon completion of the request, control is returned to the requester with the A register indicating the status:

0 Normal completion

4200000000 Illegal parameter

If A is negative, the bit configuration indicates the inhibit conditions as followsl;

Bit 0 = 1: Inhibit read function on primary image was not applied.

Bit 1 = 1: Inhibit write function on primary image was not applied.

Bit 2 = 1: Inhibit read function on secondary image was not applied.

Bit 3 = 1: Inhibit write function on secondary image was not applied.

Bits 4 and 5 are not used.

Bit 6 = 1: Release inhibit read function on primary image was not applied.

Bit 7 = 1: Release inhibit write function on primary image was not applied.

Bit 8 = 1: Release inhibit read function on secondary image was not applied.

Bit 9 = 1: Release inhibit write function on secondary image was not applied.

Bits 10 and 11 are not used.

Bit 12 = 1: Primary image does not exist or is not available.

Bit 13 = 1: Secondary image does not exist or is not available.

Any 40xxxxxxxx return indicates that the specific inhibit or release condition already exists, or the specific file image did not exist.

As an example, consider the first example above. If this call is repeated, the return in the A register is:

A: 4000000011

indicating that the inhibit functions requested were not applied, due in this case to existing inhibit conditions.

Read, Time Lock, and Select Image (RDTS$):

The RDTS$ function reads an assigned drum file specifying which copy of an image file is to be used (optional), and creates a full read/write lock on the specified area of the file (optional). This lock differs from the other locks only in that it is purged after a parameter-specified time in the system.

The macro call and code generated for this request are as follows:

RDTS$b̸file code, number of words, buffer base, logical increment, v_0. The packet generated is:

ENT*A	v_0
EBJP*B7	$+4
FILE CODE	NO. OF WORDS
	BUFFER BASE
LOGICAL INCREMENT	
EXRN	10015

v_0 is a literal parameter specifying the image selection and the maximum elapsed time that the lock is to be allowed by the system:

Bit $2^{15} = 1$ Read specified area only on primary copy of image file.

Bit $2^{16} = 1$ Read specified area only on secondary copy of image file.

Bits 2^{15} and $2^{16} = 1$ System can select the image to read.

Bits 2^{15} and $2^{16} = 0$ Do not perform I/O.

Bits $2^{14} - 2^{0} \neq 0$ Area is to be locked for this period of time (expressed in tenths of a second, minimum 100 milliseconds, maximum 54 minutes) unless released by any of the lock release calls, as LOCKA$, WRELS$ or MWRITER$. If the specified time is exceeded, 63000*00000 is returned without further action to all requesters pushed on the lock.

Bits $2^{14} - 2^{0} = 0$ No lock is to be established on the specified area.

Write and Select Image (WRT$):

This function writes to an assigned file, specifying which copy of an image file is to be used.

The macro call and the packet generated for this request are as follows:

WRTS$b̸file code, number of words, buffer base, logical increment, v_0

ENT*A	v_0
EBJP*B7	$+4
FILE CODE	NO. OF WORDS
BUFFER BASE	
LOGICAL INCREMENT	
EXRN	10016

v_0 is a literal parameter specifying which image is to be used as follows:

Bit $2^{15} = 1$: Write specified area only on primary copy of image file.

Bit $2^{16} = 1$: Write only on secondary image.

Bits 2^{15} and $2^{16} = 1$: Write to both images.

Bits 2^{15} and $2^{16} = 0$: Do not perform I/O.

NOTE: To perform a normal write operation (WRITE$), both bits 2^{15} and 2^{16} must be set to 1.

3.3.6.2. EXAMPLES OF IMAGE FILE CONTROL

The following are included as typical examples of how a user might assign and maintain image files.

Example 1:

An image file is assigned using the #ASG control statement (see 3.2.4.5):

#ASGƀƀ F432/FAST, D, 1000

The statement assigns one image to FH-432 drum and one image to FASTRAND mass storage both areas being 1000 words long. The request:

READ$ D, 100, BUFFER, 0

causes a read operation to take place on the FH-432 image. In the event that a long queue of requests (8 times the length of that on the FASTRAND image) exists on the FH-432 drum, the read operation automatically takes place on FASTRAND mass storage.

Returning from the above READ$ request, the status might be:

A = 20000 00100
Q = 00000 00010 (See image file status returns, section 3.2.4.5)

indicating that:

1. The primary image FH-432 drum had failed.

2. The read was executed correctly on the secondary image.

3. The primary image was rewritten successfully.

Both images of the file are now intact, and the requested data has been transferred to the user's buffer.

Example 2:

On repeating the same read request, as in example 1, the status returned might be:

A = 22000 00100
Q = 00000 00010

This indicates the same condition as in example 1, except that recovery was not successful on the primary image.

To restore his file, the user might employ the following sequence:

1. FRPL$ D,1000,0 (See 3.2.11.4) to replace the primary image of the file.

2. LRASL$ D,0,0,0,5 (See image file control, 3.3.6.1) to allow a write operation only to the primary image and read operation only from the secondary image.

3. LOCKR$ D,1000,SCRATCH,0,3
   WRELS$ D,1000,SCRATCH,0 to copy the secondary image to the new primary image, locking the file to other users during the process.

4. LRASL$ D,0,0,0,1100 to release inhibit conditions and restore 138K file to full use.

Both images of the file are now intact.

Example 3:

A simplex (single) file might be initially assigned as follows:

#ASGƀƀF432,E,1000/1000

If image filing becomes desirable, the following sequence might be used:

1. ASGƀƀF432/FAST,E,0 to assign a FASTRAND secondary image creating the image file;
2. RDTS$ E, 1000, BUFFER, 0, 100200 to temporarily lock file and read data from primary image;
3. WRTS$ E, 1000, BUFFER, 0, 1000000 to copy data to newly assigned secondary image;
4. LOCKA$ E, 1000, 0, 0, 0 to release file lock.

The file represented by file code E is now an image file, having two identical copies of the data.

3.3.6.3. STATUS RETURNS

Upon completion of any direct access I/O or lock request, control is returned with a status code in the A register. Information may also be contained in the Q register. Interpretation of the status codes is shown below:

A register

0000xxxxxx Normal completion. The lower 18 bits indicate the number of words transferred.

20000xxxxx While processing the specific request, the primary image of an image file had a permanent failure. However, the data was completely accommodated by the secondary image. The contents of the lower portion of the A register indicates the number of words accommodated by the secondary image. The contents of the lower portion of the Q register indicate the number of words successfully accommodated by the primary image.

21000xxxxx The secondary image of an image file had a permanent failure. However, data was accommodated completely by the primary image. The contents of the lower portion of the A register indicate the number of words accommodated by the primary image.

The contents of the lower portion of the Q register indicate the number of words successfully accommodated by the secondary image.

x22000xxxxx Identical to 20000xxxxx except that the primary image was rewritten from the secondary image, and the primary image could not accommodate the data.

23000xxxxx Identical to 21000xxxxx except that the secondary image was rewritten from the primary image, and the secondary image could not accommodate the data.

A register

24000xxxxx A declared bad area was encountered in the primary image of an image file while processing the request. Data was transferred correctly to or from the secondary image. xxxxx represent the number of words accommodated by the secondary image. The contents of the lower portion of the Q register indicate the number of words accommodated correctly by the primary image.

25000xxxxx A declared bad area was encountered in the secondary image of an image file. Data was transferred correctly to or from the primary image. The contents of the lower portion of the A register indicate the number of words accommodated by the primary image. The contents of the lower portion of the Q register indicate the number of words accommodated correctly by the secondary image.

26000xxxxx The primary image of an image file accommodated the data completely; the secondary image was inhibited to the function. The contents of the lower portion of the A register indicate the number of words accommodated by the primary image.

27000xxxxx The secondary image of an image file accommodated the data correctly; the primary image was inhibited to the function. The contents of the lower portion of the A register indicate the number of words successfully accommodated by the secondary image.

40000xxxxx Control conditions from a LRASL$ call. See Image File Control Request.

4100000000 Inappropriate request.

A register

42000 00000 Incorrect parameter.

43000 xxxxx An unrecoverable subsystem error has occurred. The contents of lower portion of the A register indicate the number of words successfully transferred. In the case of lock request, the entire area specified has been locked or released as requested.

44000xxxxx During the processing of the request, the end of the area allocated to the file code has been reached. The contents of the lower portion of the A register indicate the number of words transferred, including the last word of the file. In the case of a lock request, the locked area will not extend past the end of the file.

46000 00000 Unsuccessful SEARCH$ or BLOCKS$ operation. The search identifier could not be found before an end of file or end of block. The area will not be locked on a SEARCHL$ call.

51000 00000 Interlock, channel down, or unit down.

52000xxxxx A transfer of data has been attempted to or from an inactive file segment. The segment is hidden (bounded by active file segments) as a result of releasing storage within a file. The contents of the lower portion the A register indicate the number of words successfully transferred. In the case of a lock request, the entire area specified has been locked or released as requested.

53000 xxxxx Primary image of an image file has failed or encountered a declared bad area, and the secondary image is inhibited to the function. The contents of the lower portion of the A register indicate the number of words successfully accommodated by the primary image.

54000xxxxx The secondary image of an image file has failed or encountered a declared bad area, and the primary image is inhibited to the function. The contents of the lower portion of the A register indicate the number of words successfully accommodated by the secondary image.

55000xxxxx The primary image of an image file has failed, and a declared bad area has been encountered in the secondary image. The contents of the lower portion of the A register indicate the number of words successfully accommodated by the primary image. The contents of the lower portion of the Q register indicate the number of words successfully accommodated by the secondary image.

56000xxxxx The secondary image of an image file has failed, and a declared bad area was encountered in the primary image. The contents of the lower portion of the A register indicate the number of words successfully accommodated by the secondary image. The contents of the lower portion of the Q register indicate the number of words accommodated by the primary image.

57000xxxxx Image hardware error. Both images of an image file have encountered hardware errors preventing the correct transfer of data. For example, one image has encountered an unrecoverable subsystem error, and the other image has found an interlock condition; or both images have encountered permanent errors. The contents of the lower portion of the A register indicate the number of words transferred to or from the secondary image.

60000xxxxx The request cannot be processed because the request requires a delay of this activity as a result of another activity's lock; however, the other activity is now being delayed by a lock created by the activity now in control. The request can only be implemented if the indicated release is made by the activity in control. The contents of the lower portion of the A register indicate the logical increment of the file. The contents of the lower portion of the Q register indicate the number of words in the lock, and register B7 indicates the file code which must be released.

6100000000 Request has not been implemented because the lock specified as a part of this request exceeds the permitted number of locks.

6200000000 Programmer file construction error. This status is returned in a number of special cases not covered by the above status returns, and indicates programmer error; for example:

a. One image of an image file is inhibited to the function and other image has reached an end-of-file or hidden segment.

b. One image of an image file has encountered a declared bad area, and the other image has reached an end-of-file or hidden segment.

c. Both images of an image file have encountered declared bad areas.

The contents of the lower portion of the A register indicate the number of words transferred to or from the primary image. The contents of the lower portion of the Q register indicate the number of words transferred to or from the secondary image.

3.3.7. Magnetic Tape Macros

The magnetic tape macros that follow may be used to effect input/output on UNISERVO magnetic tape subsystems. The descriptions apply to track subsystems and to subsystems with optional 9-track format feature.

Block numbering should be avoided in loaded system environments where many tasks are making requests to the I/O initiator. The execution time required to transfer four identification words plus data words leads to undesirable system tie-ups, because of the number of instructions between the block number transfer and the preparation of the data transfer.

Certain macros are scatter/gather requests which initiate data transfers to or from several primary storage buffers in succession. A series of steps involving several microseconds are required to activate each buffer when the previous buffer has been terminated, and interrupts are locked out during the period. In a busy system where the system I/O initiator must respond to several tasks, the delay, which may be as long as 42 microseconds, could be critical to the successful execution of a task, and such requests should be avoided in this environment.

3.3.7.1. READ$

A read operation in the forward direction will be performed on the unit determined by the file code. Data from one tape block will be placed in primary storage beginning at the buffer base specified. Data transfer will continue until either the input buffer is filled or the end of the tape block is reached.

If the buffer size in the parameter list is 0, the function will be interpreted as a move operation. A read command will be issued to the subsystem, but no data will be transferred. Abnormal frame counts and parity errors will be ignored.

The macro call and the code generated from the control statement are:

READ$ƀfile code, number of words, buffer base, logical increment

	EBJP*B7	N
	FILE CODE	NO. OF WORDS
		BUFFER BASE
	LOGICAL INCREMENT	
N →	EXRN	1 0 0 0 1

The logical increment is ignored by the operating system when the request is directed to magnetic tape. The logical field may be omitted from the macro call; however, the logical increment position in the parameter list is retained and is set to 0.

At completion of the read operation, control will be returned following the executive entry instruction, with the status of the operation and the number of words transferred being in the A register. The following codes may be received at the completion of this function:

NOTE: Only abnormal conditions encountered during the execution of the function are listed in this section.

0000XXXXXX The read operation has been completed successfully. The number of words transferred will be indicated in the lower 18 bits. Bit positions 23 through 0 of the Q register will contain the binary block if this option is selected (if not, these bits will be 0). Bit positions 29 through 24 of the Q register will contain the magnitude of the frame count error; the value is zero if there is no frame count error. If a frame count error has occurred, the number of words transferred will include the partial word(s); bit position 29 will be set to 1.

4300000000 An unrecoverable subsystem error has occurred. System recovery procedures have been unsuccessful. The number of words transferred appears as 0 since the number of words successfully read cannot be determined. If the block number option has been selected, the Q register will contain the binary number of the block from which data would have been transferred by the successful execution. The tape read head will be positioned in back of the block in error. The status will also be reported when the numbered block option is in use and the next sequential block cannot be located.

4400000000 An end-of-file mark has been detected by the subsystem. Since the block does not contain data, the number of words transferred will be 0. The Q register will contain the binary identity of the last data block read (if the option is selected). The tape head will be positioned in back of the EOF mark. A READB$ operation at this point will result in another EOF status.

5100000000 Nonrecoverable interlock status. The operator has answered NO, or six tries have been attempted to recover from interlock without success.

5200000000 Tape cannot be put back into sequence. No block numbers were present on the blocks when block numbering was designated. If the condition occurs during the first read type function, the operator must verify that the tape has been written with the numbered option. If this is not the case, interlock noise is greater than the noise constant and the tape is bad.

3.3.7.2. WRITE$

The contents of the specified buffer will be transferred to the subsystem and written as one block on the tape unit allocated to the file code specified. The macro call and the resultant code generated for this function are as follows:

WRITE$ƀfile code, number of words, buffer base, logical increment

	EBJP*B7	N
	FILE CODE	NO. OF WORDS
		BUFFER BASE
	LOGICAL INCREMENT	
N→	EXRN	1 0 0 0 2

The logical increment field of the macro call is not necessary and may be omitted when using magnetic tape. However, the position of the logical increment field in the generated parameter list is retained and is set to 0.

Upon completion of the operation, the status is returned in the A register.

0000XXXXXX The operation has been successfully completed. The number of words written is completed. The number of words written is contained in the lower 18 bits. The Q register contains the binary number assigned to the block if the option has been selected on the ASG statement.

4500XXXXXX The write operation has been successfully completed; however, the end-of-tape mark was detected during the operation. The number of words written is indicated. The Q register contains the binary block number assigned if the option has been selected.

4300000000 An unrecoverable error, such as a parity error, has occurred. The system has performed its recovery procedures but is unable to complete the operation successfully. The number of words transferred is considered to be 0. The Q register will contain the binary identity of the last data block which has been written successfully. The tape is positioned beyond the block in error.

4700000000 (UNISERVO II A only) A buffer of less than 144 words has been specified in the FIXED BLOCK mode or a buffer of less than 208 words has been specified in the VARIABLE BLOCK mode.

5100000000 Nonrecoverable interlock status. The operator has answered NO, or six tries have been attempted to recover from interlock without success.

3.3.7.3. SEARCH$

The first word of each successive data block is compared with the search indentifier contained in the parameter list of the operation. When a find is made, a read operation is initiated of the data in the block found.

The search is a software search performed by the system. Repetitive READ$ commands are given, using the buffer specified by the user in the parameter list of the operation. The search is terminated when a find is made, an error condition occurs, or end-of-file is reached.

The macro call and generated code for the operation are as follows:

SEARCH$b̸file code, number of words, buffer base, logical increment, search identifier

	EBJP*B7	N
	FILE CODE	NO. OF WORDS
		BUFFER BASE
	LOGICAL INCREMENT	
	SEARCH IDENTIFIER	
N →	EXRN	2 0 0 4 6

The logical increment field of the macro call must be present, although the logical increment is not used. The field may be specified as 0.

Upon completion of the operation, the A register contains the status code and number of words transferred as shown below. The Q register will contain the magnitude of frame count error in bit positions 27 through 24 and the binary number of the last block to be successfully read in bit positions 23 through 0 (if the block number option has been selected on the ASG statement).

0000XXXXXX	A find was made, and a successful read operation accomplished.
4300000000	A subsystem error has occurred or the next sequentially numbered block cannot be located. Systems recovery procedures have been attempted, but have been unsuccessful. The tape is positioned beyond the block in error.
4600000000	Unsuccessful search. The search identifier could not be located before an end-of-file mark was encountered. The tape is positioned beyond the end-of-file mark.
5100000000	Nonrecoverable interlock status. The operator has answered NO, or six tries have been attempted to recover from interlock without success.
5200000000	Tape cannot be put back into sequence. No block numbers were present on the blocks when block numbering was designated. If the condition occurs during the first read type function, the operator must verify that the tape has been written with the numbered option. If this is not the case, interlock noise is greater than the noise constant and the tape is bad.

3.3.7.4. SCATTER READ (SREAD$)

With SREAD$ function, data from one tape block will be placed in multiple primary storage buffers. Initially, the first buffer specified will be established. When the buffer terminates, the next buffer will be activated. The process continues until either the last buffer has terminated or the end of the tape block has been reached. Any buffer whose size is specified as 0 is ignored.

When using the UNISERVO II A subsystem in the variable block mode and end-of-file marks are being passed over, the first nonzero buffer must be greater than three words for end-of-file recognition.

Since multiple buffers may be given with the function, the parameter list is generated separately from the control statements. The macro call and generated code for the control statement are:

ENT*B7	$^{v}0$
EXRN	1 0 0 2 4

V_0 is the label of the parameter list.

The parameter list may be generated by using the LISTA operator. This macro call and generated code are as follows:

LABEL→LISTA$b̸file code, number of buffers, logical increment, number of words/buffer base, number of words/base, etc.

<table>
<tr><td>LABEL →</td><td colspan="2">FILE CODE</td><td>NO.OF BUFFERS</td></tr>
<tr><td></td><td colspan="3">LOGICAL INCREMENT</td></tr>
<tr><td></td><td colspan="2"></td><td>NO. OF WORDS</td></tr>
<tr><td></td><td></td><td colspan="2">BUFFER BASE</td></tr>
<tr><td></td><td colspan="2"></td><td>NO. OF WORDS</td></tr>
<tr><td></td><td></td><td colspan="2">BUFFER BASE</td></tr>
</table>

The logical increment field of the macro call must be present even though it is not used. It may be specified as 0.

Upon completion of the operation, the A register contains the status code and number of words transferred as shown below. The Q register will contain the magnitude of frame count error in bit positions 27 through 24 and the binary number of the last block to be successfully read in bit positions 23 through 0 (if the block number option has been selected on the ASG statement).

0000XXXXXX	A successful operation has been accomplished.
4300000000	A subsystem error has occurred or the next sequential block cannot be located (numbered block option). System recovery procedures have been unsuccessful. The tape is positioned beyond the block in error.
4400000000	An end-of-file mark has been detected by the subsystem. The tape is positioned beyond the end-of-file mark.
5100000000	Nonrecoverable interlock status. The operator has answered NO, or six tries have been attempted to recover from interlock without success.
5200000000	The tape cannot be put back into sequence. No block numbers were present on the blocks when block numbering was designated. If the condition occurs during the first read type function the operator must verify that the tape has been written with the numbered option. If this is not the case interlock noise is greater than the noise constant and the tape is bad.

3.3.7.5. GATHER WRITE (GWRITE$)

Data will be transferred from multiple primary storage buffers and written as one block on magnetic tape. The first buffer specified in the parameter list is established first, and when it has terminated, the second buffer is established and so on. Any buffer whose size is specified as 0 is ignored.

The macro call and the code generated from the control statement are:

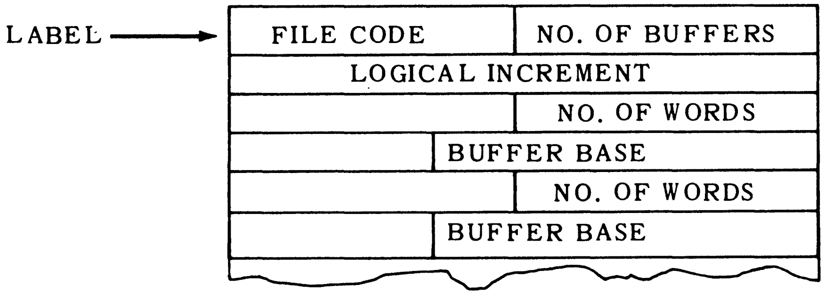
GWRITE$ƀ$v_0$

ENT*B7	v_0
EXRN	1 0 0 2 5

v_0 is the label of the LISTA$ operator.

The parameter list for this operation is generated from the LISTA$ operator. This macro call and code generated are as follows:

LABEL→LISTA$ƀfile code, number of buffers, logical increment, number of words/buffer base, number of words/buffer base, etc.



LABEL →	FILE CODE	NO. OF BUFFERS
	LOGICAL INCREMENT	
		NO. OF WORDS
	BUFFER BASE	
		NO. OF WORDS
	BUFFER BASE	

The logical increment must be specifed in the macro call.

Upon completion of the operation, the A register contains the status code and the number of words transferred as shown below. The Q register contains the binary number of the last block successfully written in bit positions 23 through 0 (if this option has been selected).

0000XXXXXX	A successful write operation has been accomplished.
4300000000	An unrecoverable subsystem error has occurred. The tape is positioned beyond the error block.
4500XXXXXX	The write operation has been successfully completed beyond the tape limit mark.
4700000000	(UNISERVO II A only) A buffer of less than 144_{10} words has been sapecified in the fixed block mode, or a buffer less than 20_8 words has been specified in the variable block mode.
5100000000	Nonrecoverable interlock status. The operator has answered NO, or six tries have been attempted to recover from interlock without success.

3.3.7.6. MULTIPLE READ (MREAD$)

The MREAD$ function will transfer data from successive tape blocks into the buffers specified in the parameter list. Each tape block will be read into a distinct buffer (i.e., the first block will be read into the first buffer, the second block will be read into the next buffer, etc.)

If the size of any buffer is specified as 0, the result will be a move over the block corresponding to the zero buffer. Thus, with one function, the user may move a number of blocks and then read. The entire operation will be a multiple move when the size of each buffer is specified as 0.

Since multiple buffers may be given with this function, the parameter list is generated separately from the request statements. The macro and generated code are as follows:

MREAD$ƀbv$_0$

ENT*B7	v_0
EXRN	10026

v_0 is the label of the parameter list.

The parameter list may be generated by using the LISTA operator. This macro call and generated code are as follows:

LABEL↳LISTA$ƀfile code, number of buffers, logical increment, number of words/buffer base, number of words/ buffer base, etc.

LABEL →

<table>
<tr><td colspan="2">FILE CODE</td><td>NO.OF BUFFERS</td></tr>
<tr><td colspan="3">LOGICAL INCREMENT</td></tr>
<tr><td colspan="2"></td><td>NO. OF WORDS</td></tr>
<tr><td></td><td colspan="2">BUFFER BASE</td></tr>
<tr><td colspan="2"></td><td>NO. OF WORDS</td></tr>
<tr><td></td><td colspan="2">BUFFER BASE</td></tr>
</table>

The logical increment field must be included in the macro call.

Upon completion of the operation the status is given in the A register as below:

0000XXXXXX A successful multiblock read operation has been accomplished. The total number of words transferred is given in bit positions 17 through 0. The number of words transferred from each tape block cannot be indicated. An abnormal frame count on any one of the blocks read will cause the magnitude of frame count error to be placed in the bits 27–24 of the Q register. A subsequent frame count error will cause the new magnitude to overlay the previous value.

Thus, at completion, the value in bits 27–24 of the Q register indicate the magnitude of the frame count error of the last block on which a frame count error occurred. This does not indicate how many blocks had frame count errors (a 0 value indicates that no frame count errors occurred on the entire read operation). If the block number option has been selected, bit positions 23 through 0 of the Q register will contain the binary identity of the last block read.

4300XXXXXX An unrecoverable subsystem error has occurred. The total number of words transferred from the block read operation before the error block was encountered, is given in the lower 18 bits.

The Q register contains the identity of the last block from which a read attempt was made. The final tape position is at the next block to be read. For example, if a four block multiread operation is requested from a tape unit positioned at the load point, and an unrecoverable parity error has occurred when reading the third block, the number of words transferred would be the total number transferred from blocks one and two. The Q register would indicate the third block as being the last block from which a read attempt was made: The tape is now in position to read block four.

4400XXXXXX An end-of-file mark has been detected during the multiread operation. The total number of words transferred is indicated. The Q register contains the identity of the last block read before the EOF mark was detected. The final tape position is beyond the EOF mark.

5100000000 Nonrecoverable interlock status: The operator has answered NO, or six tries have been attempted to recover from interlock without success.

5200000000 Tape cannot be put back into sequence: No block numbers were present on the blocks when block numbering was designated. If the condition occurs during the first read tape function, the operator must verify that the tape has been written with the numbered option. If this is not the case, interlock noise is greater than the noise constant, and the tape is bad.

3.3.7.7. READ BACK (READB$)

The READB$ function will cause the tape to be moved in a backward direction. The data from one tape block will be placed in the buffer specified, but will be reversed; that is, the last word of the block written will be placed into the first word of the buffer when the block is read in a backward direction. The position of bits within a word will remain the same when read backward as when written.

If the buffer size, as specified in the parameter list, is 0, this function will be treated as a move backward operation: no data will be transferred. For tape subsystems on which the hardware does not recognize a read backward instruction, this function will be treated as a move backward operation.

Tapes written to the 9-track options, and contains blocks with abnormal frame counts, that have been read according to the method described in section 3.2.5.2 can have blocks reconstructed as follows:

(1) Set BX equal to the abnormal frame count.

(2) Enter By with the number of words processed, as returned in the A register. Descrease the value by one.

(3) Enter Bz with the first word address of the deposit area.

(4) Execute the coding that follows. BASE is the buffer base specified in the read backward request. TABLE is the octal equivalent of the data bit value opposite the frame count received, as shown in the table. Lower (L) refers to the actual data bit value. Upper (U) refers to its complement, also shown in the table.

```
CL*A
ENT*A*W(BASE+By)
BJP*By*$+2
LSH*AQ*L(TABLE+Bx)
BSK*BO*By
ENT*Q*W(BASE+By)*SKIP
CL*Q
LSH*AQ*U(TABLE+Bx)
STR*A*W(Bz)
BSK*Bz*7777
CL*A
BJP*By*$—12
```

The following table shows the relationship between the contents of the frame count field in Q and the number of data bits in the last word assembled.

Frame Count Field (Binary)	Data Bits Assembled (Base 10)	
	Actual	Complement
0000	––	––
0001	8	22
0010	16	14
0011	24	6
0100	2	28
0101	10	20
0110	18	12
0111	26	4
1000	4	26
1001	12	18
1010	20	10
1011	28	2
1100	6	24
1101	14	16
1110	22	8

Care must be taken during a READB$ operation to ensure that the buffer being read is larger than the block being transferred, as a frame count error is also indicated when all the words have not been transferred.

The macro call and generated code are as follows:

READB$ƀfile code, number of words, buffer base, logical increment

	EBJP*B7	N
	FILE CODE	NO. OF WORDS
		BUFFER BASE
	LOGICAL INCREMENT	
N ⟶	EXRN	1 0 0 1 0

The logical increment field may be omitted from the macro call; however, its position in parameter will be retained and set to 0.

Upon completion, the status and number of words transferred will be contained in the A register. Bit positions 23 through 0 of the Q register will contain the block number of the last data block passed over (if the option has been selected).

0000XXXXXX The operation has been completed successfully. The total number of words transferred will be given. The magnitude of frame count error will be contained in bits 27–24 of the Q register.

For a move back operation, the number of words transferred will be 0. The Q register also will be 0.

4500000000 The load point was encountered during the operation. No data transferred. The Q register will be 0.

4300000000 An unrecoverable error has been encountered; system recovery has been attempted. The Q register will be 0. This status will not be received on a move backward operation. If the inhibit noise record constant operation has been selected on the ASG statement.

3.3.7.8. REWIND$

The REWIND$ function will cause the selected tape unit to be positioned at load point. The only parameter required for this operation is the file code. The macro call and generated code are as follows:

REWIND$ƀfile code

ENT * B7	FILE CODE
EXRN	1 0 0 2 1

At completion of the operation, the status is returned in the A register.

0000000000 The rewind operation has been successfully initiated. The Q register will be 0.

3.3.7.9. REWIND WITH INTERLOCK (REWINDI$)

The REWINDI$ function will cause the selected tape unit to be positioned at load point and interlocked so that the unit may not be used again without operator intervention. The macro call and generated code are as follows:

REWINDI$ƀfile code

ENT * B7	FILE CODE
EXRN	1 0 0 2 2

At completion of the operation, the status is returned in the A register.

0000000000 The rewind with interlock operation has been successfully initiated. The Q register will be 0.

3.3.7.10. WRITE END-OF-FILE (WRTEOF$)

The WRTEOF$ will cause a mark to be written on the tape which will be recognized by the subsystem as an end-of-file (EOF) mark when encountered during a read operation. When the function is used to mark the end of a tape reel, there should be as many EOF marks written as there are standby buffers or block reads in the MREAD$ operator of the routine to read the tape. The file code is the only parameter required. This operation will be ignored when directed to a UNISERVO II A. The macro call and generated code are as follows:

WRTEOF$ƀfile code

ENT * B7	FILE CODE
EXRN	1 0 0 2 0

At completion of the operation, the status is returned in the A register as shown below. The Q register contains the block number of the last data block written, when the option has been selected.

0000000000 The end-of-file mark has been successfully written.

4500000000 The end-of-file mark has been successfully written and the end-of-tape mark has been detected during the write operation.

4300000000 An unrecoverable subsystem error has occurred during the write operation. The tape is positioned beyond the block in error.

4700000000 (UNISERVO II A only) The unit has been assigned in the fixed block mode. The operation is inappropriate.

3.3.7.11. ERASE (ERASE$)

The erase function will cause 3.25 inches of tape to be erased for editing purposes. No data will be written on the tape. This operation will be ignored when directed to a UNISERVO II A subsystem is referenced, a contingency write operation will be performed. The macro call and generated code are as follows:

ERASE$ƀfile code

ENT * B7	FILE CODE
EXRN	1 0 0 2 3

At completion of the operation, the status is returned in the A register as shown below. The Q register contains the binary identity of the last data block written when the option has been selected.

0000000000 The operation has been successfully completed.

4500000000 The end-of-tape mark has been detected during execution of the function. The operation has been successfully completed.

4300000000 An unrecoverable subsystem error has occurred during execution.

3.3.8. Unit Record Macros

Device level control of unit record equipment may be exercised by the user, provided that the peripheral unit has been duly assigned to the user by the system. Assignment cannot be made if the unit is being used for cooperative input and/or output as described in 3.4.2.

For the purpose of device control, unit record equipment is classified on the basis of data rather than peripheral type; that is, card input may be specified and the peripheral assigned may be either an online UNIVAC 1004 or UNIVAC 9300 Subsystem, or a Card Subsystem, whichever is available. The classifications are card, printer, and paper tape.

3.3.8.1. CARD OPERATIONS

Card operations may take place on either the Card Subsystem or an online UNIVAC 1004 or UNIVAC 9300 subsystem. 80-column cards may be read and punched in either translate mode (card code to Fieldata) or in column binary mode. If a 90-column (or 80/90 column) UNIVAC 1004 or UNIVAC 9300 subsystem is available, 90-column cards may be read or punched in either a translate (80-column code to Fieldata) or binary (90-column code) mode.

Table 3–2 (see 3.3.8.4) demonstrates the Fieldata conversion for both 80- and 90-column cards. The table may be used for both reading and punching.

The card macros are as follows:

- Read$

  One card image is read in the mode specified by the ASG statement and is placed in the buffer specified. The minimum buffer size is: 80-column translate mode, 20_8 words; column binary mode, 40_8 words; and 90-column mode, 22_8 words. A buffer, smaller than the minimum for the effective mode, will constitute a parameter error.

  The read macro call and generated code are:

  READ$b̸file code, number of words, buffer base, logical increment

	EBJP*B7	N
	FILE CODE	NO. OF WORDS
		BUFFER BASE
	LOGICAL INCREMENT	
N →	EXRN	10001

  The logical increment field of the macro call is not used and may be omitted; the position of the field in the parameter list, however, is retained.

  Upon completion of the operation, the status and number of words transferred are returned in the A register as below. The Q register will be 0.

  0000XXXXXX The operation has been successfully completed.

  4700XXXXXX An illegal combination of punches (any combination of holes in rows 1, 2, 3, 4, 5, 6, 7, or 9) has been detected by the hardware in a single card column (translate mode only). The illegal character will appear as 00 in code in the buffer.

4300000000 An unrecoverable subsystem error has occurred.

4100000000 The requester has submitted an invalid function code.

4200000000 The requester has specified an input size which is less than the minimum size required for the type of image which will be read.

5100000000 The requested function cannot be executed because of an interlock condition that cannot be corrected.

- WRITE$

The WRITE$ function will cause one card to be punched in the mode selected on the ASG statement. The same minimum buffer size requirements as for the card read are maintained for the card write operation. The macro call and generated code are:

WRITE$b̸file code, number of words, buffer base, logical increment

	EBJP*B7	N
	FILE CODE	NO. OF WORDS
		BUFFER BASE
	LOGICAL INCREMENT	
N →	EXRN	10002

The logical increment field of the macro call is not used and may be omitted.

At completion of the operation, the status code and number of words transferred are returned in the A register. The Q register will be 0.

0000XXXXXX The operation has been completed successfully.

4300000000 An unrecoverable subsystem error has occurred.

4100000000 The requester has submitted an invalid function code.

5100000000 The requested function cannot be executed because of an interlock condition that cannot be corrected.

- Multiple Read (MREAD$)

The MREAD$ function will cause a number of cards to be read and transferred. The number of cards to be read is determined by the number of buffers specified. Each card is read in the same mode, and the minimum buffer requirements of that mode must be observed for each buffer specified.

The parameter list for this operation is generated from the LISTA$ operator. The macro call and generated code are as follows:

LABEL→LISTA$b̸file code, number of buffers, logical increment, number of words/buffer base, number of words/buffer base, etc.

LABEL →

FILE CODE	NO. OF BUFFERS
LOGICAL INCREMENT	
	NO. OF WORDS
	BUFFER BASE
	NO. OF WORDS
	BUFFER BASE

The logical increment field of the macro call must be present.

The execution statements for this function may be generated from the following macro:

MREAD\$ƀv_0

v_0 is the label of the LISTA\$ operator. The code generated from this macro is:

ENT*B7	v_0
EXRN	10026

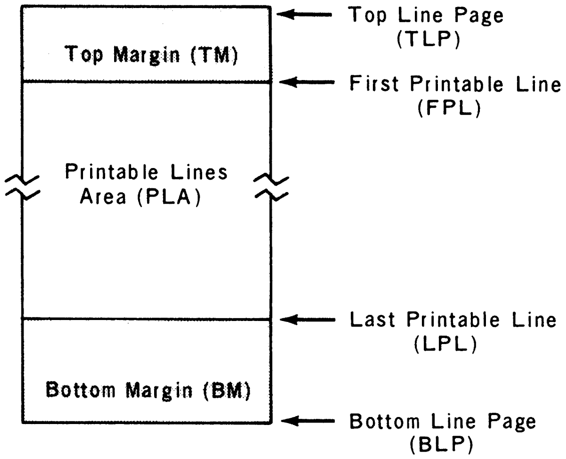
Upon completion of the operation, the status and total number of words transferred are returned in the A register as below. The Q register will be 0.

0000XXXXXX	The operation has been completed successfully.
4700XXXXXX	An illegal combination of punches (any combination of holes in rows 1, 2, 3, 4, 5, 6, 7, or 9) has been detected by the hardware in a single card column (translate mode only). The illegal character will appear as a 00 code in the buffer.
4300000000	An unrecoverable subsystem error has occurred.
4100000000	Requester has submitted an invalid function code.
4200000000	Requester has specified an input buffer size as less than the minimum required size for the type of image to be read.
5100000000	The requested function cannot be executed because of an interlock condition that cannot be corrected.

3.3.8.2. PRINTER OPERATION

Print requests may be directed to any high speed printer subsystem or to an online UNIVAC 1004 or UNIVAC 9300 subsystem. The system routines handling the UNIVAC 1004 or UNIVAC 9300 printer have been designed to duplicate the function of the standard printer subsystem.

Vertical form control is specified on the ASG statement in the form of a top margin, a bottom margin, and the number of printable lines per page, the information is used to exercise basic forms control during subsequent print applications. The following illustration characterizes the page format.


Top Margin (TM)
Printable Lines Area (PLA)
Bottom Margin (BM)
Top Line Page (TLP)
First Printable Line (FPL)
Last Printable Line (LPL)
Bottom Line Page (BLP)

The Top Margin constitutes the number of lines (single spaces) contained from TLP to FPL exclusive of FPL. Similarly, the Bottom Margin is BLP to LPL exclusive to LPL. The intervening area, the Printable Lines Area (PLA), is the number of lines reserved for printing. Thus:

TM + BM + PLA = Total lines per page

Once defined and established, the page format will remain in effect until the facility is released. The user may wish to regulate his own margins. In this case, the number of printable lines per page is specified as 0, and the print function will be performed as received, without regard to margin spacing. Failure to specify form control on the ASG statement will cause a standard page format to be supplied.

With a page format in effect, requested print operations will be performed within the PLA only. Form-to-form skipping (LPL to FPL next page) will be accomplished by the system without user intervention. Vertical spacing within the PLA, however, must be regulated by the user.

Spacing is an additional function of the print operation; the number of lines to be spaced is indicated in the parameter list. Spacing precedes printing of the line. If no spacing is indicated, printing will occur on the same line as that of the previous print operation.

Each print operation causes the printing of one 132-character line. The characters to be printed must be Fieldata coded. The correspondence between the Fieldata code and the printed symbol is shown in Table 3–4.

The UNIVAC 9300 handler spaces paper only by the use of a form control loop. The loop is set up according to system parameters in eleven levels. The first ten are punched 1 through 6, and the last level is 1 through 5 and then 7. The seventh is the top of the form. The user specifies the total lines per page by TM + BM + PLA = total. The absence of these parameters indicate a standard form control, and the handler supplies the standard system format.

The 77 code is recognized as a stop code; when encountered in a print buffer, the 77 code and the remainder of the 132 print positions will print as spaces.

The macro call and generated code for the print operation are as follows:

WRITE$b̸file code, spacing, buffer base

	EBJP*B7	N
	FILE CODE	SPACING
		BUFFER BASE
N →	EXRN	10002

The spacing in the macro call and parameter list is the number of lines to space before printing and may be specified in the macro as an octal or decimal constant. A value of 0 will cause overprinting. When bit position 14 of the spacing field is set, the paper will be advanced to the home position, the first physical line of the next page, and printing will not occur. Advancement of paper to the home position is permissible only if a page format is in effect. Otherwise, an invalid function code status will be returned to the requester.

With a page format in effect, spacing to the middle of the next page is accomplished as follows: a print is required with spacing that exceeds the remaining PLA, e.g. 77_8, and with a print buffer consisting of a stop code (77) as the first character of the print line. This operation will space the paper to the FPL of the next page.

The desired spacing within the PLA and the line to be printed after the spacing are then specified in the next print operation.

If the user is employing the UNIVAC 9300 subsystem, and is using his own form control loop, the A register must contain the channel number from the skip code 1 through 7. The specification causes the handler to issue a command to space to channel number and print. To move the paper only, the A register must have bit 29 set to 1, and must specify the channel number from the code 1 through 7. The A register must be set up prior to the WRITE$ packet; unused space in the packet is ignored. The user may also specify a channel number with the value of 0, which causes an overprint.

The data to be printed begins at the buffer base and ends when a stop code (77) is detected or when 132 characters (26.4 words) have been transferred.

Upon completion of the print operation, the status and number of words transferred will be contained in the A register. The Q register will be 0.

0000000033	A successful print operation has been completed. The number of words transferred is assumed to be 33_8 even though a stop code may have been detected.
4300000000	An unrecoverable subsystem error has occurred.
4100000000	The requester has submitted an invalid function code.
5100000000	The requested function cannot be executed because of an interlock condition that cannot be corrected.

3.3.8.3. PAPER TAPE OPERATIONS

Paper tape read and punch requests may be directed either to a standard paper tape Subsystem or to an online UNIVAC 1004 subsystem containing the appropriate paper tape equipment.

Paper tape data will not be packed; that is, each buffer word will contain only one tape frame. The tape frame may contain 5, 6, 7, or 8 levels of information. Any code may be read or punched by the user; the system does not perform code conversion on paper tape data. Parity, if desired, is the responsibility of the user. Control characters are not recognized by the system.

- READ$

  The READ$ function will cause paper tape to be read from the unit specified by the file code. Data from one tape frame will be placed in bit positions n to 0 of each buffer word (n = 4, 5, 6, or 7) depending upon the level of tape read. The user must have knowledge of the level of tape being read should clear the unused bits (the UNIVAC 1004 tape reader will generate a bit in position 7 when seven-level tape is being read, and bits in positions 6 and 7 when six-level tape is being read, etc.). The user must also determine if, and what level, parity is contained on the tape.

Each tape being read should contain a trailer of sufficient length to prevent the reader from running out of tape. The minimum trailer length, in inches, may be calculated by dividing the maximum program buffer size by ten.

The macro call and generated code for the paper tape read request are as follows:

READ$ƀfile code, number of words, buffer base, logical increment

	EBJP*B7	N
	FILE CODE	NO. OF WORDS
		BUFFER BASE
	LOGICAL INCREMENT	
N →	EXRN	10001

The logical increment field of the macro call may be omitted; however, the position is retained and set to 0.

At completion of the operation, the status code and the number of frames transferred are returned in the A register:

0000XXXXXX	A successful read operation has been accomplished.
4300000000	A subsystem error has occurred.
4100000000	The requester has submitted an invalid function code.
4500XXXXXX	The requester has specified an input buffer which is larger than the number of frames remaining on paper tape. XXXXXX= number of frames read successfully.
5100000000	The requested function cannot be executed because of an interlock condition that cannot be corrected.

- WRITE$

WRITE$ function will cause the data contained in the specified buffer to be punched on the assigned paper tape unit. Bit positions 7 through 0 of each buffer word will be punched. Thus, if tape of less than eight levels is being used, the user must zero fill the remaining bit positions. For example, when punching six-level tapes, bit positions 6 and 7 of each buffer word must be clear. Parity* may be added to each character if desired. The leader and trailer may be punched (with or without parity) in the same manner as data.

The macro call and generated code for the paper tape punch request are as follows:

WRITE$ƀfile code, number of words, buffer base, logical increment

	EBJP*B7	N
	FILE CODE	NO. OF WORDS
		BUFFER BASE
	LOGICAL INCREMENT	
N →	EXRN	10002

*_The parity feature is not available on the UNIVAC 1004 paper tape punch._

The logical increment field of the macro call may be omitted; however, the position is retained and set to zero.

At completion of the operation, a status code and the number of frames punched are returned in the A register below:

0000XXXXXX The punch operation has been completed successfully.

4300000000 A subsystem error has occurred.

4100000000 The requester has submitted an invalid function code.

5100000000 The requested function cannot be executed because of an interlock condition that cannot be corrected.

The following tables show conversion codes: Table 3–4 shows conversion for the printer and Table 3–5 shows conversion for card codes.

OCTAL CODE	CHAR.	OCTAL CODE	CHAR.
00	@	40	)
01	[	41	–
02	]	42	+
03	#	43	<
04	△	44	=
05	Space	45	>
06	A	46	&
07	B	47	$
10	C	50	*
11	D	51	(
12	E	52	%
13	F	53	:
14	G	54	?
15	H	55	!
16	I	56	, (comma)
17	J	57	\
20	K	60	0
21	L	61	1
22	M	62	2
23	N	63	3
24	O	64	4
25	P	65	5
26	Q	66	6
27	R	67	7
30	S	70	8
31	T	71	9
32	U	72	' (apos.)
33	V	73	;
34	W	74	/
35	X	75	.
36	Y	76	¤
37	Z	77	Stop Code

Table 3–4. Printer Codes

CARD CONVERSION				
80-COLUMN CARD CODE	FIELDATA CODE	CHARACTER	90-COLUMN CARD CODE	90-COLUMN OCTAL
12-1	06	A	1-5-9	25
12-2	07	B	1-5	24
12-3	10	C	0-7	42
12-4	11	D	0-3-5	54
12-5	12	E	0-3	50
12-6	13	F	1-7-9	23
12-7	14	G	5-7	06
12-8	15	H	3-7	12
12-9	16	I	3-5	14
11-1	17	J	1-3-5	34
11-2	20	K	3-5-9	15
11-3	21	L	0-9	41
11-4	22	M	0-5	44
11-5	23	N	0-5-9	45
11-6	24	O	1-3	30
11-7	25	P	1-3-7	32
11-8	26	Q	3-5-7	16
11-9	27	R	1-7	22
0-2	30	S	1-5-7	26
0-3	31	T	3-7-9	13
0-4	32	U	0-5-7	46
0-5	33	V	0-3-9	51
0-6	34	W	0-3-7	52
0-7	35	X	0-7-9	43
0-8	36	Y	1-3-9	31
0-9	37	Z	5-7-9	07
0	60	0	0	40
1	61	1	1	20
2	62	2	1-9	21
3	63	3	3	10
4	64	4	3-9	11
5	65	5	5	04
6	66	6	5-9	05
7	67	7	7	02
8	70	8	7-9	03
9	71	9	9	01
12	42	+	1-5-7-9	27
11	41	- (minus)	0-3-5-7	56
12-0	54	?	0-1-3	70
11-0	55	! (exclam.)	0-3-7-9	53
0-1	74	/	3-5-7-9	17
2-8	46	&	0-1-3-5-7	76
3-8	44	=	0-1-3-5-7-9	77
4-8	72	' (apos.)	0-1-5-7-9	67
5-8	53	: (colon)	1-3-7-9	33
6-8	45	>	0-3-5-7-9	57
7-8	00	@	0-1-3-7	72
12-3-8	75	(period)	1-3-5-9	35
12-4-8	40	)	0-1-3-5	74
12-5-8	01	[	0-5-7-9	47
12-6-8	43	<	0-1-5-9	65
12-7-8	03	#	0-1-5-7	66
11-3-8	47	$	0-1-3-5-9	75
11-4-8	50	*	0-1	60
11-5-8	02	]	1-3-5-7	36
11-6-8	73	; (semicolon)	1-3-5-7-9	37
11-7-8	04	△	0-1-7	62
0-2-8	77	≠	0-1-7-9	63
0-3-8	56	, (comma)	0-3-5-9	55
0-4-8	51	(	0-1-9	61
0-5-8	52	%	0-1-5	64
0-6-8	57	\	0-1-3-7-9	73
0-7-8	76	¤	0-1-3-9	71
BLANK	05	Space	BLANK	00

Table 3–5. Card Conversion

3.4. COOPERATIVE CONTROL

Cooperative control involves collection of systems elements by which OMEGA retrieves all scheduling information, in the form of control and submits accounting and actions taken by OMEGA as the result of processing schedule parameters.

The cooperative control mechanism because of its scheduling role, is inherent to OMEGA and is responsible for controlling three data streams defined as follows:

- Primary Input is used to contain system schedule parameters, limited user data, source code to system compilers and assemblers, program parameters and so on.

- Primary Output is printed copy of program scheduling results, listings from assemblers and compilers, limited data, accounting and so on.

- Secondary Output is an optional stream which can be used to contain assembler and compiler object code, and limited data.

The cooperative mechanism is composed of input unit record routines responsible for accepting primary input streams from systems input devices; input/output (I/O) cooperative control which performs staging for all three streams; and output unit record routines responsible for submitting primary and/or secondary output streams to designated systems devices. Figure 3–1 depicts the relationship and interaction betwen these three sets of elements.

3.4.1. Input Unit Record Routine

The input record elements are individual routines which accept the primary input streams from their assigned devices and submit the obtained streams to I/O cooperative control for staging. Any number or type of unit record routines may operate concurrently under control of OMEGA, allowing multiple streams to enter the system. Each unit record scans the input stream to identify job descriptions, which are in turn queued to OMEGA for selection and activation.

Access to primary input data is by service request through I/O cooperative control from the active task. Supplementary streams may be introduced from a specified auxiliary source. This feature may be used to merge and correct source code, or may be extended by an installation to serve the need of user programs. The feature may be used also to enter supplementary control statements for job descriptions which are resident to the system.

A standard set of unit record routines is provided with the system. However, it must be noted, because of the system's modularity and relatively simple interface, any device is an eligible candidate as a unit record device.

3.4.2. Input/Output (I/O) Cooperative Control

Input/Output (I/O) cooperative control is a basic system element which is responsible for the staging of input/output to and from direct access storage as presented or requested by unit records routines. In performing this function, it must recognize and control overflow conditions which may occur. A second function of I/O cooperative control is to process service requests from operating tasks or systems elements for primary input or for the submission of primary or secondary output.

Cooperative control ultilizes direct access storage as a staging area for the above streams, providing the system and user with the following advantages and features:

- The staging of low speed input/output data to direct access storage to balance intermittent system ultilization with slow rate of peripheral devices. Staging allows the device to operate at full capacity within the controlled constraints of the staging area. The buffering to direct access storage permits the parallel utilization of low speed devices by operating tasks in a multiprogram environment.

- Provides OMEGA compilers, assemblers, system processors, and user programs with a consistent mechanism, independent of device characteristics, to obtain and submit data. The feature purges system elements of redundant code required to assign, recognize, and handle varied devices.

- The staging area cooperative library is maintained by cooperative control, allowing multiple streams of primary and secondary data to utilize the pooled library. Data for any one stream is threaded by chaining techniques to the job. Cooperative control expands and contracts the cooperative library as required to maintain the system. In addition to expansion of direct access storage, cooperative control will invoke temporary suspension and/or roll in, roll out procedures to control overflow conditions.

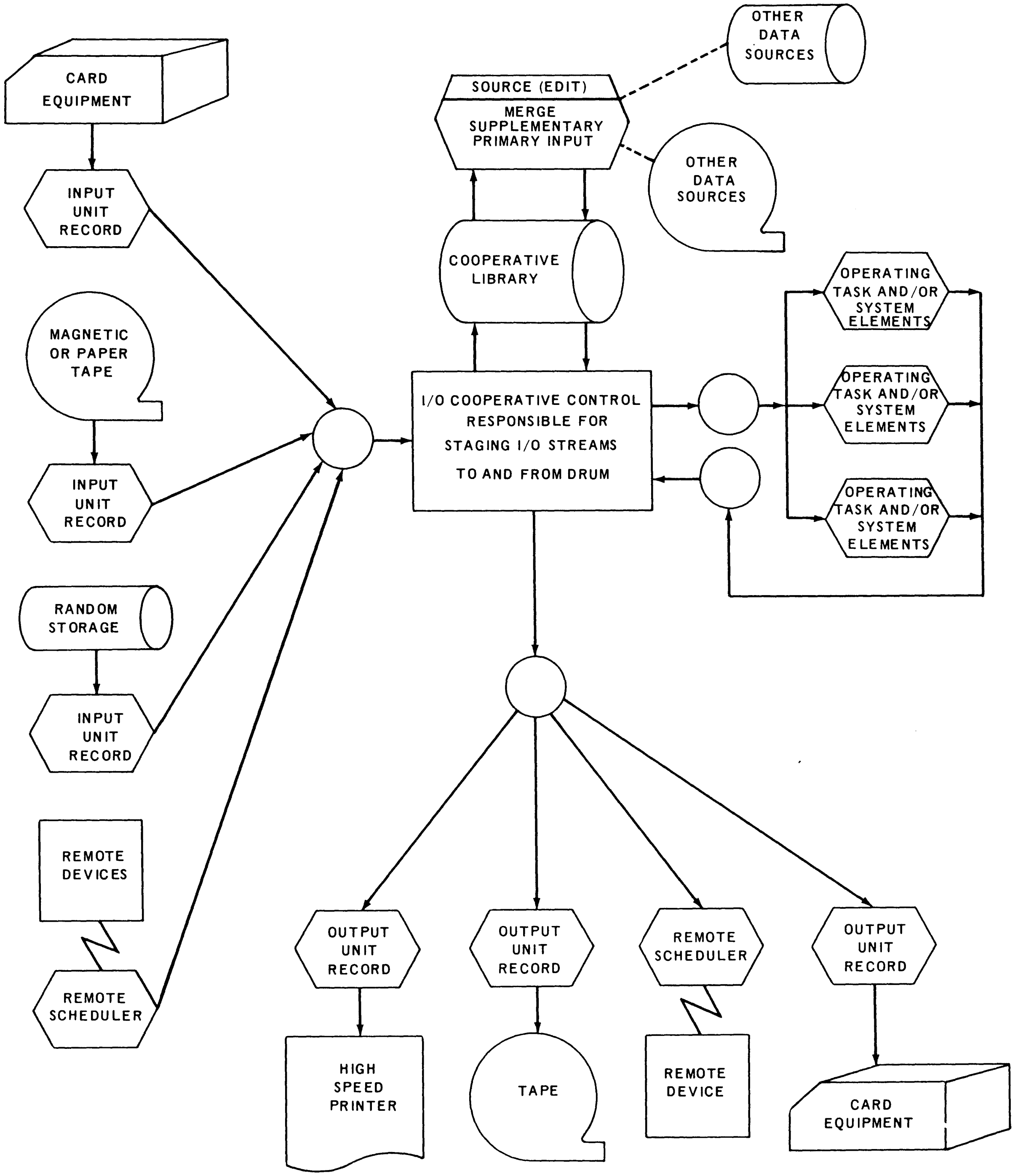


Figure 3–1. Cooperative Mechanism

3.4.3. Output Unit Record Routines

Primary and secondary output unit record routines are individual system elements which punch, print, record, or otherwise process primary and secondary output streams. Each routine is designed to handle a particular device and is activated and controlled by OMEGA. Output unit record routines are activated when a task or job terminates, or an overflow condition in a particular stream is reached.

3.4.4. Service Requests

The following paragraphs discuss the general usage of service requests.

3.4.4.1. PRIMARY INPUT

The user may obtain card image input to his program through the cooperative control mechanism. The data must be contained in the primary input stream for this job and be located at the correct point within the stream. For example, the images in the control stream must follow the control card which starts the user program, e.g., the GO card. If the user data precedes the GO card, the images are discarded; when the GO statement is finally reached, the user data is no longer available.

The image presented to the user has a maximum length of 24_{10} words, and is Fieldata coded. The first character of each image may not be a Fieldata 03 (#) since this character is recognized as a system control image. Certain control images within the user's image set are legal; these are processed by the system, and the next image is retrieved and presented to the user. The set of control images which may be present within the user image set is listed in 2.5.2.1.

The end of the user's image set is determined by a unique image which the user recognizes as a signal to stop requesting input, or the user continues requesting images until an end-of-file or end-of-input status is received as described below.

The user may request an image from the primary input stream through the following macro call:

CARD\$b̸v_0

v_0 is the base address, relative to the lower limit of the PLR, of the buffer to which the image is to be transferred. v_0^0 may be any valid read class operand.

The following coding is generated in response to the CARD call:

ENT*B7	v_0
EXRN	3 0 0 0 1

Upon completion of the operation, control is returned to the requester with the status of the operation in the A register.

0000XXXXXX	Successful completion: The lower 18 bits contain the image length. Full words of trailing space codes are deleted from the image and are not reflected in the image length.
4200000000	The buffer (base plus image length) lies outside the PLR limits of the requester.
4300000000	An unrecoverable error was encountered in attempting to read the image from direct access storage or the orginating read device.

4400000000 The next image in the input stream is a control image which is invalid when located within the user's image set. This code may be recognized by the user to determine the end of his image set.

4500000000 The user's primary input job stream has been exhausted (i.e., the FIN image has been encountered). This code may also be recognized by the user to determine the end of his image set.

3.4.4.2. PRIMARY OUTPUT

The primary output stream is a series of print images containing information pertaining to each job. The system uses the primary output stream for items such as assembler output, accounting, control cards, test mode dumps and traces, etc. This method of printing is also available to the user. An image submitted to primary output is placed in the stream as the image.

The image submitted by the user may have a maximum length of 27_{10} words and must be Fieldata coded. The user also provides the number of lines to space before the image is to be printed. If the spacing is indicated as 77_8 or greater, it is recognized as a skip to the next page.

Full words of trailing spaces are deleted from the image by the system. The various unit record routines will create a complete print line by adding the necessary number of trailing spaces to the image. The user need not specify a full print line when submitting an image to the primary output stream; the remainder of the line will be space filled by the unit record routine.

An image may be submitted to the primary output stream through the following macro call:

`PRINT$ƀ`v_0,v_1,v_2

v_0 is the base address, relative to the lower limit of the PLR, of the buffer which contains the image.

v_1 is the size (number of words) of the image.

v_2 is the number of lines to space before printing the image.

v_0,v_1, and v_2 may be any read class operand.

The following coding is generated:

ENT*B7	v_0
ENT*Q	v_1
ENT*A	v_2
EXRN	3 0 0 0 2

Upon completion of the operation, the status and number of words transferred are contained in the A register as shown below:

0000XXXXXX Successful completion: The image has been placed in the primary output stream. The number of words placed in the stream is contained in the lower 18 bits.

4200000000 Illegal parameter: The buffer, base address plus the length, does not lie within the program lock limits of the requester.

3.4.4.3. SECONDARY OUTPUT

The secondary output stream is an optional output stream which may be employed for user data, assembler/compiler output, and other library output. An image submitted to secondary output is placed in the stream in the same manner as for primary output.

The image submitted by the user may have a maximum length of 16_{10} words and must be Fieldata coded. The image is in the form of a card to be punched; however, the ultimate destination of the image is determined by the output unit record routine chosen.

Full words of trailing spaces are deleted from the image by the system. The various unit record routines will create a complete card image, if required, by adding the necessary number of trailing spaces. The user need not submit a complete card image to the secondary output stream; the remainder of the image will be space filled by the system when, and if, required.

An image may be submitted to the secondary output stream through the following macro call:

PUNCH$b̸$v_0$,$v_1$

v_0 is the base address, relative to the lower limit of the PLR, of the image to be placed in the secondary output stream.

v_1 is the size (number of words) of the image.

v_0 and v_1 may be any valid read class operands.

The following coding is generated.

ENT*B7	v_0
ENT*Q	v_1
EXRN	3 0 0 0 3

Upon completion of the operation, control will be returned to the requester with the A register containing the status of the operation and the number of words transferred. The number of words transferred will be contained in the lower 18 bits.

0000XXXXXX	Successful completion: The image has been placed in the secondary output stream.
4200000000	Parameter error: The image, base address plus image length, does not lie within the program lock limits of the requester.

3.4.5. Method of Operation

Primary input is entered into the system by the input unit record routines from a variety of devices: 80- or 90-column cards, paper tape, magnetic tape, direct access storage (prestored job streams), etc.

The appropriate unit record routine is activated in response to a keyboard entry by the computer operator as follows:

URb̸name/version

Name/Version identifies the routine to be activated, which depends on the data source and format. For example, different magnetic tape routines would handle different blocking arrangements. The unit record routine, when loaded, requests the facility of the appropriate type; a particular unit (e.g., Tape Unit 4) cannot be preselected. The facility is released to the available pool when the FIN statement is encountered.

The input unit record routine begins reading images from its assigned facility. When a JOB statement is encountered, a task addendum for the job is set up. Subsequent images for this job are then staged to direct access storage by the cooperative control mechanism and linked to the job through the task addendum. When another JOB statement is encountered, the previous input stream is closed off and another task addendum is created for the new job. This process continues until a FIN statement is reached. At this time, the input stream is closed off, the unit record is terminated, and the facility is released.

As soon as a task addendum ahs been created for a job the task is eligible for selection although its complete job stream has not yet entered the system. When the job has been selected, its primary input stream is retrieved from storage, an image at a time, by the input/output cooperative mechanism. The system then performs the functions specified by the control images in the order determined by the sequential arrangement of the images. At this time, the primary (and possibly secondary) output stream is initiated.

The primary output images generated for a job are staged to direct access storage by the input/output cooperative and linked through the job's task addendum. Any secondary output generated by the job is staged and linked in the same manner as primary output.

Primary (and secondary) output images are allowed to accumulate on direct access storage until any of three contingencies occur, as described below at this point, the appropriate output unit record routine is activated to retrieve the output stream, via cooperative control, and to output the images to the chosen device (e.g., printer, magnetic tape, paper tape, etc.). When the output unit record routine senses that the end of an output stream has been reached, a search is made to find another output stream which is ready to be processed. If another stream for this unit record routine is not ready, the routine is deallocated and its facility is released.

The three contingencies which trigger the activation of an output unit record routine are:

- The complete job has terminated.
- The maximum allowable output for the job has been exceeded.
- An individual task of a job has terminated and the cooperative library contains excessive output for the task (the point at which the output becomes excessive is an installation parameter). In this instance, the output stream for the task is depleted by the unit record routine before the next successive task of the job is scheduled. Thus, the output stream of a different job may be handled by the unit record routine between the output of successive tasks of the same job.

3.4.6. Error Diagnostics

The system will detect certain error conditions in connection with the user input/output through cooperative control. In these cases, error messages will be submitted to the primary output stream and, in certain instances, to the operator's console. The circumstances causing these diagnostic messages and the resulting action by the system, are delineated below.

3.4.6.1. PRIMARY INPUT

A status code of 4400000000 or 4500000000 indicates that user data can no longer be obtained from the input stream. If the user persists in requesting an image the appropriate status will be returned a total of three times. If a fourth request is made, the following diagnostic message is submitted to the primary output stream:

END OF PRIMARY INPUT FILE LOOP

The task or job will then be aborted in accordance with the X option of the current task control statement.

3.4.6.2. PRIMARY OUTPUT OVERFLOW

A count of the number of pages of primary output for each job is maintained by the system. When this number exceeds the maximum allowed for the job (the maximum is specified either as an installation parameter or on the JOB statement), the following diagnostic message is placed in the primary output stream:

PRIMARY OUTPUT STREAM OVERFLOW

A message to the operator is then printed on the console. The message informs the operator of the primary output overflow condition. The operator decides whether to continue or to abort the task and enters his choice through the console keyboard. The formats of the printed message and operator response are described in Section 5.

If the decision is made to continue the task, the estimate of primary output is doubled and processing continues. If the decision is made to abort, the task is aborted in accordance with the X option contained on the last task control card processed.

3.4.6.3. SECONDARY OUTPUT OVERFLOW

The diagnostic message is similar to that for primary output. A count of the number of images of secondary output is maintained for each job. If this count reaches the maximum allowed for the job, the following message is submitted to the job's primary output stream.

SECONDARY OUTPUT STREAM OVERFLOW

The operator then receives a message through the console, informing him of the overflow condition and soliciting his decision whether to continue or to abort the task. A decision to continue will cause the original estimate to be doubled; another overflow condition will occur if the new estimate is exceeded. Processing will be continued. If the abort option is chosen the task will be aborted according to the X option of the task control card currently in effect.

3.5. MASTER FILE DIRECTORY (MFD)

The Master File Directory (MFD) is a direct access storage area which holds descriptors of users' permanent or temporary data files. The descriptors (or qualifiers) contain pertinent information, read/write keys and/or REELID, describing certain direct access storage areas or magnetic tape reels. The original or described storage areas and tape units must be obtained by means of the #ASG statement, and cataloging of the assignment places the necessary information into the descriptor. Any assignment or release of the cataloged area or reel must be done with the #MFD statement. The MFD is not applicable to unit record devices, e.g., high speed printer and card reader.

3.5.1. The MFD Statement

Files in the MFD are cataloged, assigned and released by means of the MFD statement.

■ Format:

For cataloging, recataloging, and assigning of files, the MFD statement has the following general form:

#MFDƀoptionsƀfile code, identifier, protection keys, retention period, REELID.

For delinking, purging, and releasing of files, the MFD statement has the following form:

#MFDƀoptionsƀfile code.

- Option:

A Assign the file specified in the identifier field to the given file code.

C Re-catalog the file with the MFD as a permanent file. Re-catalog the file by changing the protection keys and retention period for direct access storage area or by changing the REELID for tape reel.

D Delink the file assignment before termination of the task, but do not release the area or unit associated with the data file.

E Do not assign the specified file to another job during the time that the current job has control of the file. If the file is in use at the present time, the request is suspended.

I Rewind the tape unit with interlock when the assignment is released.

J Hold the file code assignment for the duration of the job.

L Lock the file until it has been established.

P Purge the file from the directory, but do not release the direct access storage area.

Q Do not submit the MFD statement to the primary output stream (used to protect read/write keys).

R Release the file from the MFD, and release the direct access storage associated with the file.

S Allow the assignment of the file when the read/write keys do not match. Set the read and write protection bits.

T Catalog the file with the MFD as a temporary file.

U Unlock a file which has been locked previously with the L option.

V Execute the task whether or not the assignment or catalog request is satisfied.

W Specify write enable on a tape file.

- Specifications:

File code is a one or two character alphabetic code used as a symbolic link to reference the file during the execution of the job. Permissible file codes are A – Y and AA – AZ.....YA – YZ.

Identifier is the external means by which jobs reference a file cataloged with the MFD, and has the following form:

user number/file number

with the user number and the file number being one to five digits each. A decimal number is indicated by D following the number. The largest allowable value is 77777_8 or 32767_{10}. The actual maximum user number and file number is a system generation parameter.

Protection keys are safeguards which may be specified when a file is cataloged, requiring subsequent assignment requests for the file to match the keys before read or write permission will be granted. The format of the protection key field is:

read key/write key

with each key subfield being up to five alphanumeric characters in length.

If the protection key field is not provided for a cataloged file, a match is automatically assumed upon any subsequent file assignment request. If the read key, on an assignment request, does not match the read key specified when the file was cataloged, and the S option was not specified, no assignment is made and an incorrect parameter status is returned to the requester. If a match is made on the read key, and the write key does not match, the file is assigned, but the requester is prevented from writing on the file because the write protection bit is set. A file cannot be recataloged or released from the MFD if the write key was not matched when the file was assigned.

Retention period specifies the number of days that the file will be held in the MFD.

REELID is the label by which a tape reel is referenced, and must be present in the MFD statement when a tape file is to be cataloged. The RELID will be used as an operator directive for tape mounting on subsequent file assignments.

- Method of Submission

The MFD statement may be submitted externally through the control stream or dynamically during task execution. Upon return of program control from an internal MFD request, the status information is conveyed to the task in the A register and some information is carried in the Q register.

Normal completion: The A register is set to binary zero indicating that the statement was successfully processed. The Q register is set to 0, indicating that the image was processed correctly; to 1, indicating that the primary image could not be assigned, but the secondary image was assigned; or to 2, indicating that the secondary image could not be assigned but the primary image was assigned.

Abnormal completion: The MFD request was not performed, and the register contains one of the following values. An appropriate error message will appear in the primary output stream.

4100000000	Invalid operation: the task has requested an operation that cannot be performed, such as the assignment of a file which has not been cataloged.
4200000000	Incorrect parameter: the MFD statement could not be interpreted by the system.
4300000000	A system error has occurred during processing of the MFD statement.
5000000000	The required tape unit was not available, the operator was unable to mount the tape, the direct access storage file could not be assigned because of the E option exclusive user, or a subsystem error occurred during reading of the descriptors.
5100000000	Unrecoverable interlock: the operator gave a negative response to the tape mount message.

3.5.2. Examples

The following examples show the use of the MFD statement.

Example 1:

Initial generation of a master file on direct access storage.

```
#JOBƀƀTEST/OMEGA,12345,C,C,C/C
#ASGƀoptionsƀRAN,D,50000/10000
———
———
——— Tasks necessary to create a data program file on to file code D.
———
———
#MFDƀCƀD,10/001,,999D
```

The file associated with file code D is entered into the MFD. The user number and file numbers are respectively 10 and 001; no read or write keys are supplied, and the file is to be retained for a period of 999 days.

Example 2:

Subsequent reference to the above file in another job.

```
#JOBƀoptionsƀONE/OMEGA,12300,C,C,C/C
#MFDƀAJƀC,10/001
```

The file code "C" is associated with the file cataloged in the previous example. The function of the card is the same as for an ASG card. Read, write permission is granted. Any program is allowed full access to the file within the job.

Example 3:

Addition of protection keys to the file during the above job. The following control statement is submitted:

```
#MFDƀCƀC,10/001,ABC3X/ABC3Y
```

In the following assignments of the file, the protection keys have to be specified correctly to read or write to the file. If the read key in a subsequent assignment does not match, no assignment is made; and a negative status is returned to the requester together with an appropriate error message submitted to primary output. If the read key matches but the write key does not match, the file is assigned with write protection; and the requester is not allowed to write on the file, re-catalog the file, or release the file from the MFD.

Example 4:

Once a file is assigned by a task, the file remains with that task until one of the following requests occurs, assuming that the assignment was made by the following statement:

```
#MFDƀAƀB,1/1,READP/WRITP
```

(a) A release request is issued by the worker program.

```
#MFDƀRƀB
```

This causes the linkage to the descriptor to be cleared, the actual release of the direct access storage occupied by the file back into the facility pool, and the file code to be released from the task.

(b) A delink request is issued by the worker program.

```
#MFDƀDƀB
```

This causes the release of the file code from the task. No actual release of the direct access storage or clearing of the linkage to the descriptor is made.

(c) The termination of the task to which the file is assigned. Upon the termination of a task, all files acquired through the MFD are de-linked as in (b) above, unless the J option is provided. If the J option is present on the assignment, the file is held until the job is to be terminated.

(d) A purge request is issued by the worker program.

```
#MFDƀPƀB
```

A purge request clears the linkage to the descriptor, but does not release the file code from the task or its related direct access storage, i.e. the file code is still assigned.

Example 5:

To expand a file, the following statements are required:

```
#MFDƀAJƀYA,10/1,ABC3X/ABC3Y
#ASGƀJƀRAN,YA,10000/20000
---
--- Operations necessary to operate on expanded portion of file
---
#MFDƀCƀD,10/001,,999D
```

The first MFD card assigns the file with read and write keys, The ASG card expands the file, and the second MFD card re-catalogs the expanded file.

NOTE: Once a file is cataloged, the file cannot be re-cataloged under a different identifier. The example above assigns user number 10 and file number 1, and has to re-catalogue the file under the same numbers.

3.5.3. Error Diagnostics and Messages

Whenever facility assignment cannot process a control statement or service request because of an error condition, an appropriate error diagnostic is submitted to the primary output stream, primary storage for working storage, and Unstring Parameter Table (UPT) is released, and the status is returned to the requester. This section provides a list of the error diagnostics and a description of the error(s) producing the diagnostic.

PARAMETERS NOT INTERPRETABLE

The parameters on the control statement or service request could not be interpreted by facility assignment. Status of 4200000000 is returned to the requester.

INVALID FILE CODE

The specified file code is not included in the eligible set. Status of 4200000000 is returned to the requester.

ILLEGAL USER NUMBER

The specified user number was larger than the maximum user number in the MFD. Status of 4200000000 is returned to the requester.

ILLEGAL FILE NUMBER

The specified file number was larger than the maximum file number for the given user number. Status of 4200000000 is returned to the requester.

INCORRECT READ KEY

The read key specified in the file assignment request does not match the read key specified in the catalog. Status of 4200000000 is returned to the requester.

FILE CODE NOT ASSIGNED

The specified file code was not assigned to an on-line peripheral unit. Status of 4200000000 is returned to the requester.

FILE CODE ALREADY ASSIGNED

The file code specified on the ASG or MFD statement was currently assigned. Status of 4200000000 is returned to the requester.

REELID NOT SPECIFIED ON TAPE FILE CATALOG

A request was made for an MFD catalog of a tape file, and the REELID (label) of the tape was not provided. Status of 4200000000 is returned to the requester.

WRITE ENABLE ON FILE WITH WRITE PROTECTION

An MFD Assignment statement for a tape file requested write enable, but the file was approved with write protection. Status of 4100000000 is returned to the requester.

FILE HAS NOT BEEN CATALOGED

An MFD assignment request was made for a file that has not been cataloged. Status of 4100000000 is returned to the requester.

REQUIRED PARAMETERS NOT AVAILABLE

Parameters necessary to perform the requested function were not provided. Status of 4200000000 is returned to the requester.

WRITE PROTECTION ON THIS FILE

A function was requested which is not allowed when a file has write protection. These functions include: FADD$, FADDA$, FREL$, FRELA$, FRPLI$, FRPL$, MFD release from the directory and MFD re-catalog. Status of 4100000000 is returned to the requester.

INAPPROPRIATE PERIPHERAL NAME

Peripheral name specified on the ASG statement could not be located in the facility map. Status of 4100000000 is returned to the requester.

SUBSYSTEM ERROR

An unrecoverable error occurred while the MFD statement was being processed. Status of 4300000000 is returned to the requester.

NECESSARY TAPE UNIT NOT AVAILABLE

An MFD tape file assignment request was submitted, but the necessary tape unit was not available. Status of 5000000000 is returned to the requester.

OPERATOR UNABLE TO MOUNT TAPE FILE

An MFD tape file assignment request was made, and the operator answered in the negative to the tape mount message. Status of 5000000000 is returned to the requester.

LOGICAL INCREMENT NOT IN FILE

An FRPLI$, FRPL$, FREL$, or FRESA$ service request specified a logical increment which is not contained in the direct access storage file. Status of 4400000000 is returned to the requester.

HIDDEN AREA REFERENCED

An FRPLI$, FRPL$ service request specified the replacement of an area previously marked as hidden by an FREL$ request. Status of 5000000000 is returned to the requester.

U/F NUMBER HAS BEEN PREVIOUSLY CATALOGED

An attempt was made to catalog an MFD file under a different user/file number. Status of 4100000000 is returned to the requester.

FILE CANNOT BE CATALOGED UNDER TWO U/F NUMBERS

An attempt was made to catalog a non-MFD file under a user/file number already cataloged. Status of 4100000000 is returned to the requester.

PERIPHERAL NOT AVAILABLE

The requested unit device or area of direct access storage was not available. Status of 5000000000 is returned to the requester.

4. LANGUAGE PROCESSOR CONTROL

4.1. GENERAL

Language processors are system components which translate programming languages into machine usable form. From source language statements, the language processors produce intermediate output codes, or relocatable binary (RB) elements, which may be collected and allocated with other program elements by the Loader Processor, prior to execution, to form absolute object or executable programs.

Language processors for the UNIVAC 494 Real-Time System include FORTRAN IV, COBOL, UNIVAC 494 Assembler, and UNIVAC 494 SPURT. The source language input, print output, and RB code output are co-ordinated and made compatible through the storage, handling, and retrieval functions of OMEGA and its co-operative action with the language processors. Source language, relative binary and object code elements can be stored, accessed, updated, and linked within various libraries of the system. Thus, users are provided with a flexible and efficient means for program development and maintenance.

4.2. METHOD OF OPERATION

Programs are composed of one or more relocatable binary (RB) elements. Each element is treated by a language processor as a separate entity, having its own distinct source code input, call for translation, and subsequent RB output and print listings. An RB element is not executable as such, and may be thought of as a subprogram requiring collection with other subprograms before the executable program is obtained. The element or subprogram may contain references (external references or XREF's) to other subprograms, and/or may contain definitions (entry definitions or EDEF's) which are referenced by other subprograms. The object program is formed by the Loader Processor which collects the subprograms and resolves all references. The resulting program is called a load or absolute program, this being its final, executable form.

At collection time, subprograms may be contained in any of the direct access storage libraries recognized by the system, placed there as a result of language translation or entered by the Program Library Editor. The joining together of subprograms provides the user with two important aids to program development:

(1) the capability for compiling or assembling only those parts of the program which are in error or which require updating, and

(2) the ability to develop a program which is composed of subprograms produced by various translators, such as COBOL, FORTRAN IV, and SPURT.

The following is a brief description of the steps, or processors, involved in the compilation and collection of programs.

4.2.1. Preparation of Input

The user prepares the source Language deck which will be conveyed through the primary input stream to the selected language processor. The source deck may contain all of the source statements for the translation, or the deck may be composed of corrections and/or updates to a source element contained in a direct access library.

Each source deck is then prefaced by the primary control statement used to call and activate the selected language processor, and is followed by an END statement. The language processor Call, source deck, and END statement are placed in the job deck as a task. Any number or combination of individual assemblies and/or compilations may be contained in the job deck. Each will be recognized by the system as an individual task causing selection and activation of the language processor specified by the primary control statement.

4.2.2. General Functions

In general, upon activation, each language processor, compiler and assembler, performs the following functions, which differ in form and application according to the processor:

- Retrieves source statements from the primary input stream. The end-of-stream for one translation is designated by the END statement.

- Translates source code into an RB element, or into RB elements, in the case of COBOL.

- Requests OMEGA to retrieve and store the resultant RB elements in the user's job library.

- Submits all printer listings and diagnostic information to the primary output stream.

The RB elements produced by the language processors can be transferred subsequently to permanent storage by means of the Program Library Editor and/or used in subsequent collection processes by the Loader. A call to the Program Library Editor for permanent storage of elements is recognized by OMEGA as a task separate from production and collection of elements, and must be contained in the current job deck if produced RB elements are to be saved upon termination of the job stream.

4.2.3. Loader

The Loader is a systems processor which is responsible for the collection and allocation of RB elements into an executable program. A secondary control language describes all overlay or segmentation techniques to be used in the formation of the program. Using the program libraries and secondary control language, the Loader retrieves all required subprograms, resolves cross references, modifies RB code to absolute, and stores the formed absolute program in the job library.

Upon completion of the load process, the absolute program element can be activated and run by the GO control statement or recorded in a permanent file for later activation. Once the load process has been completed and the absolute element has been retained, loading need not be repeated unless a correction or update is required.

4.2.4. Program Library Editor

The Program Library Editor is a collection of file maintenance routines activated by control statement or service request, and is responsible for maintenance and manipulation of the user job library. The editor routines also co-ordinate and control elements in the group and system libraries which are used by the job library. The explicit functions of the Program Library Editor are:

- Control of the loading of program elements, source, RB, and absolute, into direct access job libraries from an external medium, normally magnetic tape.
- Control of the recording of program elements, source, RB, and absolute, onto an external medium, normally magnetic tape, from the direct access program libraries.
- Linking of permanent or semi-permanent direct access storage program libraries to the user's job library.
- Printing of table of contents and description of elements contained in the program libraries.
- Deletion from the job library of those program elements which are no longer needed for the execution of the job stream so that storage may be conserved.
- Addition of elements produced by system processors to the job library.

4.3. PROGRAM LIBRARIES

The library concept is basic to an understanding of the system. The smallest logical unit of information which may be entered into the system is an element. A collection of elements is called an element library. Three logical levels of libraries are referenced: job, group, and system. These levels are referenced in the stated order so that override may be controlled and predicted.

To identify and locate the elements within the library, a table of contents (TOC) is created and maintained for each element within the library by the system. The TOC identifies the element by name, version, and type, and by control information unique to each particular type. The name is symbolic and is associated with the element at time of creation. A version may be associated with each relocatable element to differentiate between adaptations of the same element when checking out large programs. A library may contain alternate versions of the same program, with one version being outdated but workable, and the other being in a test stage.

The element name may comprise one to ten alphanumeric characters. The version designator, which is optional, may comprise up to five alphanumeric characters. When the name/version specification is given to the system through the control stream to identify an element, both name and version, separately, will be left justified with trailing space codes inserted before a search is made. Name and version must identify each element uniquely, apart from all elements of a particular type within the user's program library structure: job, group, or system. Elements of different types, e.g. source and RB, may have the same name and version.

Elements within the library complex are manipulated through the Program Library Editor Control statements (IN, OUT, etc.).

4.3.1. Element Types

The following three types of elements may be contained in libraries recognized and processed by the system (see Appendix A for formats).

4.3.1.1. SOURCE ELEMENT

A source element may be input data to a language processor. In addition to the programming languages processed by COBOL, 494 SPURT, 494 Assembler, and FORTRAN IV, the control languages processed by OMEGA may be stored as an element. This includes any set of statements which may appear in the primary input stream. Thus, job control statements, secondary control statements, and limited data sets may be included as source elements.

Each source element is composed of a table of contents (TOC), which identifies and describes the element, and a text, which is composed of source language statements carrying the information and making up the body of the element. The source element input into the system is in the form of variable length images in Fieldata code, each image having been edited to delete full words of trailing blank columns in order to conserve direct access storage space.

Source elements may be entered into the libraries through the Program Library Editor or the SOURCE processor. Once the elements are in the library, they can be inserted or merged into the current input stream, also through the SOURCE statement. Source elements containing secondary control statements to system processors, e.g., REXecutor, Loader, etc., can be specified on the primary control statement activating the processor.

4.3.1.2. RELOCATABLE BINARY (RB) ELEMENT

Relocatable binary (RB) elements are produced by the language processors, COBOL, FORTRAN IV, 494 SPURT, and 494 Assembler, and represent processed source language programs or subprograms which may be complete or which may be dependent upon collection with other RB elements before utilization. An RB element must be processed into a load element before it is executable as a program.

Each RB element is composed of a table of contents (TOC), which identifies and describes the element; a preamble which gives declarative information concerning the element, its references, and cross references; and a text which comprises the machine codes generated by the language processors, together with any necessary codes for instruction modification at collection time.

4.3.1.3. LOAD ELEMENT

Absolute and relative load elements are produced by the loader through the collection and loading process which combines RB elements. An absolute element is an entity with all external references connected and interconnected, cross references resolved, and relative locations assigned. The absolute element can be entered into any contiguous primary storage area by an essentially direct read operation from direct access storage. The absolute element is directly executable on machines operating in the UNIVAC 494 mode under OMEGA. For execution on machines operating in the UNIVAC 490 mode under REX494, a relative load element may be produced so that the instructions may be modified, as necessary, at execution time. Each load element is composed of the following parts:

- table of contents (TOC), which identifies and describes the element;
- control statements, if desired, for processing at load time;
- instructions and modifications for collected elements, as necessary;
- symbol definitions (SDEF's), if desired, for use in program testing and debugging;
- and the collected RB elements making up the program.

4.3.2. Libraries

The following is a brief description of libraries used in program control and their relationship to the system.

4.3.2.1. JOB LIBRARY

The job library is a collection of elements created for each job. The library is built as the job is executed, and is interrogated only to satisfy requests associated with the job. This library supports the continuity between tasks within the job.

A job library is established by a library maintenance function (IN), or by elements generated or referenced during job execution. Control language statements, therefore, explicitly and implicitly contribute to a job library. A given job library is transient and remains only as long as the job is active because one job library exists for each active job. Elements which are to be preserved must be output from a job library by a library maintenance function (OUT) as part of the job.

Since the job library exists for the entire span of time that a job is within the system, it is advantageous to the user to purge elements which are no longer required, thereby releasing storage to new elements produced. An element or the entire job library may be purged using the DEL statement (see 4.7.4).

4.3.2.2. GROUP LIBRARY

The group library is a bridge between the impermanence of a job library and the permanence of the system library. In effect, the group library is an independent file established in the system through the LINK statement and the Master File Directory (see 3.5). A group library has the same general format as the systems library. For example, a particular programming or applications group at an installation may develop a set of elements unique to the group. Rather than repeatedly introduce this set of elements into job libraries, the user merely establishes a group library within the system. Since the group library need exist only once for the multiprogram execution of a series of jobs, both time and storage are conserved. A group library may be registered with the system, either permanently or immediately, before the group's machine utilization. The Master File Directory provides a convenient mechanism for registering group libraries for availability to the system.

Once a group library is established by the LINK statement, it will remain within the library complex until it is formally released from the Master File Directory (see 3.5).

The elements within a group library are permanent; they can not be changed or deleted by the Program Library Editor. Functions such as OUT and PRT, which attempt no change to elements within the library, may be used as normal.

4.3.2.3. SYSTEM LIBRARY

The system library, which is resident on direct access storage, is an inherent and permanent part of the operating system. The system library contains standard RB elements, such as mathematical, utility, input/output, and editing routines, which may contribute to construction of a program; standard absolute programs, including system processors, transient elements of the operating system and internally registered job streams. The system library cannot be altered by the library maintenance procedures employed through the control language. Any function which does not cause alteration of the referenced element, such as OUT and PRT, is permissible for use with the library.

4.4. LANGUAGE PROCESSOR CONTROL STATEMENTS

System compilers and assemblers are recognized as individual tasks to be selected and activated as normal batch programs in a multiprogram environment. Selection of the task is activated when the primary control specifying the desired language processors is sensed in the job deck as the next sequential task to be executed. The processor control statement also conveys the necessary parameters and options for effecting the translation.

4.4.1. Statement Format

The control statement for each called language processor is essentially the same:

#Translatorƀoptionsƀname/version

- Translator:

  The translator field contains an abbreviation or acronym of the called processor: COB for COBOL, FOR for FORTRAN IV, ASM for 494 Assembler, and SPURT for 494 SPURT.

- Options:

X Abort the task and the remainder of the job stream if errors are detected during processing of the source code. This option means that an error encountered will affect the validity of the element, and that tasks following are dependent upon an error free condition. When an error occurs, all requested listings of the processing will be submitted to the primary output stream, if possible, as will the error diagnostics. No RB element will be produced. The processor will terminate with an ABORT$ service request which causes all current assignments and subsequent tasks within the job deck to be purged from the system.

Y Accept the translated RB element as correct although errors are detected during processing. This option means that some noncritical errors are known or expected to be present which will not affect the validity of the element and which should be ignored. The language processors will submit the element to the user's job library as being error free. If a critical error occurs which will inhibit the validity of the element, the Y option is ignored; and the element is marked as being in error when submitted to the job library.

Z Abort the task, but continue execution of the remainder of the job stream if errors occur during processing. This option means that an error condition will affect the validity of the element, making the element useless, but that the tasks following are not dependent upon the element. All requested listings and error diagnostics will be submitted to the primary output stream; however, no RB element will be produced. The task will be terminated with an ERROR$ service request, and all subsequent tasks within the job deck will be performed in the normal manner.

Absence of X, Y or Z options implies that the element will be generated and submitted to the user's library in the normal manner. Elements will be marked as being in error if errors occur; and termination of translation is by normal RETURN$ service request. Attempts to collect or execute a generated element in subsequent tasks can be controlled by the LOAD and GO statement options.

L Produce a complete listing of the generated program or subprogram being translated. The list will consist of all pertinent data, dependent upon the language processor. Any summarization and error listings will also be given. All listings are submitted to primary output.

N Suppress the source and object code listing implied by the L option. No information will be submitted to primary output other than error diagnostic messages.

T Search the job, then the group libraries for a version of the processor before going to the system library.

Additional options are also available for some processors. These will be discussed, along with variations in the above options as they apply to different processors, in the related paragraphs.

- Specifications:

  Name/Version is the symbolic name, and version designator, if any, assigned as the identity of the element being generated. If omitted, the name of the element is supplied automatically by the processor involved (FORTRAN IV, for example, supplies the subroutine or function name).

4.4.2. Language Processor Facilities

All system processors and utility routines, including language processors, are normal batch program elements which follow the rules of batch program composition, selection, activation, and termination. Their object code is in the form of an absolute element normally contained in the systems library. However, the language processors and Loader may be contained in a group library. This requires the user to link the group library to the job stream before a call to a given processor is made, but allows the direct access storage area used to hold the processors to be free when programs are not being developed.

Facilities required by the system processors for primary storage extensions, direct access storage, or magnetic tape scratch files are expressed by normal ASG cards. These cards are attached as part of the preamble of the processors at collection time. The facility assignments expressed are set for average source code translations. These can be modified for the needs of a particular re-collection of the translator or can be superseded by ASG statements preceding the processor call for a particular task at object time (see Loader and ASG override for details). The explicit requirements of each processor are given with explanations of their respective call statements.

4.4.3. SPURT Statement

The SPURT statement is used to call the 494 SPURT assembler into primary storage from the library. The assembler is responsible for translating a set of source code statements written in the SPURT language into relocatable binary (RB) elements which may be collected by the Loader to form absolute elements that are capable of being executed.

The elements produced by the assembler are automatically entered into the user's job library where they may be accessed by other system routines. Listings containing the source code, generated code, and error diagnostics are also produced in the primary output stream unless specifically inhibited by the use of an option.

The input data for the assembly must be contained in the primary input stream following the SPURT statement. This is automatic if the data and the SPURT statement originate from the same media. Data originating from other sources must be entered by the SOURCE statement (see 4.5.1).

- Primary Storage Requirements:

  The SPURT assembler requires $13{,}500_8$ consecutive primary storage locations for instructions, and an additional $14{,}000_8$ (minimum) to $40{,}000_8$ (maximum) consecutive primary storage locations for tables.

- Peripheral Requirements:

  The SPURT assembler requires the use of two files: File Code ZH, which specifies a file on direct access storage media, and File Code ZI, which specifies a sequential file on direct access storage or tape media. The ASG statements necessary for acquisition of the usual peripheral requirements are contained in the SPURT assembler in the library. These assignments may be overridden by the user through assignment of other parameters for the requested files. Such assignment statements must precede the SPURT control statement in the job deck.

The assignment statements incorporated with the SPURT assembler are:

```
ƀASGƀƀRAN,ZH,53240/71600,RANDOM FILE

ƀASGƀƀ,RAN,ZI,53240/71600,SEQUENTIAL FILE

ƀCOREƀLƀ11400/40400
```

If the ASG statements are not overridden, the assembler will obtain a minimum of $53,240_8$ words of mass storage for the random file, and it may obtain a maximum of $71,600_8$ words of mass storage for the file if such storage is available. For satisfying the sequential file request, $53,240_8$ to $71,600_8$ words of mass storage will be assigned. The CORE statement, which assigns primary storage to the assembler, cannot be overridden; it can only be supplemented.

■ Format:

The SPURT statement has the following general form:

```
#SPURTƀoptionsƀname/version
```

■ Options:

X Abort the element and the remainder of the job if assembly errors are detected. Produce the requested listings of the processing, if possible, and the error diagnostics.

Y Accept the assembly although nonfatal errors are detected or elements marked as if in error are included. Produce the RB element as being error free. If fatal errors occur during processing, the option will be ignored.

Z Abort the element if assembly errors are detected. Continue processing the remainder of the job. Produce the requested listings of the processing and the error diagnostics. If fatal errors occur during processing, the option will be ignored.

Absence of X, Y, or Z options implies that the element will be generated and submitted to the user's job library in the normal manner.

A Sort all labels alphabetically. This option is used as an addition to the S or L options to provide an alphabetic label table.

C Provide combination output replacing A, I, M, P, R, and U options.

D Delete card columns 1–6 from side-by-side listings.

E Assign all extra primary storage to table expansion.

H Suppress printing of any line which contains object code only.

I Produce side-by-side listing of instructions generated by a MACRO call.

J Prevent expansion of MACRO, EDEF, and XREF tables to allow a maximum number of labels.

L Produce a listing of the source and object codes, and all program labels, associated values, and control counters.

M List MACRO names and EDEF's with control counter values.

N Suppress listing with the exception of error diagnostics.

P Produce a listing of all external references (XREF's) and the references to them.

Q Dictate operation of QIF*v_0*v_1 statement (skip v_0 instructions and assemble v_1 if Q option is present; otherwise, assemble v_0 instructions and skip v_1. See section 7.5., *UNIVAC 494 Real-Time System SPURT Programmer Reference, UP-4090* (current version)).

R Produce an alphabetically sorted listing of labels, and the addresses of all references to them.

S Place symbol definitions (SDEF's) with the RB element.

T Execute assembler version in job library/group library.

U Provide a summary of assignment and usage of internal tables (EDEF, XREF, MACRO, LABEL).

- Specifications:

Name/Version specifies the name, or name and version, which will be assigned to the RB output element produced by SPURT. The name may comprise up to ten alphanumeric characters. The version, if specified, may comprise up to five alphanumeric characters and is separated from the name by a slash (/). The entry in the name/version field should be used in all future references to the element.

- Example:

```
#SPURTbbTEST
```

The source code provided in the primary input stream, TEST, will be assembled; an RB element, TEST, will be produced and will be placed in the job library automatically.

4.4.4. ASM Statement

The ASM statement is used to call the 494 Assembler into primary storage from the library. The assembler is responsible for translating a set of source code statements written in the ASM language into relocatable binary (RB) elements which may be collected by the Loader to form absolute elements that are capable of being executed.

The elements produced by the assembler are automatically entered into the user's job library where they may be accessed by other system routines. Listings containing the source code, generated code, and error diagnostics are also produced in the primary output stream unless specifically inhibited by the use of an option.

The input data for the assembly must be contained in the primary input stream following the ASM statement. This is automatic if the data and the ASM statement originate from the same media. Data originating from other sources must be entered by the SOURCE statement (see 4.5.1).

- Primary Storage Requirements:

The ASM assembler requires $50{,}000_8$ consecutive primary storage locations.

- Peripheral Requirements:

The ASM assembler requires the use of three files, File Code, ZH (for holding source images), File Code ZI (for holding RB information), and File Code ZJ (for holding PROC source images and cross reference information), all of which specify random access storage on mass storage media. The ASG statements necessary for acquisition of the usual peripheral requirements are contained in the ASM assembler in the library. These assignments may be overridden by the user through assignment of other parameters for the requested files. Such assignment statements must precede the ASM control statement in the job deck.

The assignment statements incorporated with the ASM assembler are:

ƀASGƀƀRAN, ZH,100000

ƀASGƀƀRAN, ZI,100000

ƀASGƀƀRAN, ZJ,130000

If the above statements are not overridden, the assembler will obtain a minimum of $100,000_8$ words of mass storage for the File Codes ZH and ZI and $130,000_8$ words of mass storage for File Code ZJ.

- Format:

The ASM statement has the following general form:

#ASMƀoptionsƀname/version

- Options:

X Abort the element and the remainder of the job if assembly errors are detected. Produce the requested listings of the processing, if possible, and the error diagnostics.

Y Accept the assembly although nonfatal errors are detected or elements marked as if in error are included. Produce the RB element as being error free. If fatal errors occur during processing, the option will be ignored.

Z Abort the assembly if errors are detected or elements marked as if in error are included. Continue processing the remainder of the job. Produce the requested listings of the processing and the error diagnostics. If fatal errors occur during processing, the option will be ignored.

Absence of X, Y, or Z options implies that the element will be generated and submitted to the user's job library in the normal manner. Errors will inhibit assembly. The requested listings will be produced, if possible, as will the error diagnostics.

B Specify 17-bit index mode processing (gives a warning flag if 15-bit B register is used i.e., B1–B3).

C Check sequence numbers of source images.

D Inhibit sequence numbers from listing.

H Reverse hierarchy between instructions and PROC's so that PROC's have higher priority.

J Produce ASM page headings for listing.

L Produce a complete listing of the source and object codes, and all pertinent data, including summarizations and error diagnostics.

N Suppress all listings with the exception of error diagnostics.

R Produce a listing of labels with all references to the labels sorted alphabetically.

S Produce a listing of symbol definitions (SDEF's) as a by-product of the RB output.

T Execute the assembler version in Job Library/Group Library.

- Specifications:

  Name/Version specifies the name, or name and version, which will be assigned to the RB output element produced by ASM. The name may comprise up to ten alphanumeric characters. The version, if specified, may comprise up to five alphanumeric characters and is separated from the name by a slash (/). The entry in the name/version field should be used in all future references to the element.

- Example:

  #ASMƀƀTEST

  The source code for the element, TEST, will be assembled; an RB element will be produced and will be placed in the job library automatically.

4.4.5. FOR Statement

The FOR statement is used to call the FORTRAN IV compiler into primary storage from the library and to transfer control to the compiler. The primary function of the compiler is the translation of a set of source code statements written in the FORTRAN IV language into relocatable binary (RB) elements which may be collected by the Loader to form absolute elements that are capable of being executed.

The elements produced by the compiler are automatically entered into the user's job library where they may be accessed by other system routines. Listings containing the source code, generated code, and error diagnostics are also produced in the primary output stream unless specifically inhibited by the use of an option.

The input data for the compilation must be contained in the primary input stream following the FORTRAN IV statement. This is automatic if the data and the FORTRAN IV statement originate from the same media. Data originating from other sources must be entered by the SOURCE statement (see 4.5.1).

- Primary Storage Requirements:

  The FORTRAN IV compiler requires a minimum of $14{,}000_{10}$ and a maximum of $32{,}000_{10}$ consecutive primary storage locations.

- Peripheral Requirements:

  The FORTRAN IV compiler requires the use of two files: File Code ZH and File Code ZI; both are random access.

  The FORTRAN IV compiler, in the library, contains the ASG statements necessary for acquisition of the usual primary storage and peripheral requirements. These assignments may be overridden by the user through specification of other parameters. Such assignment statements must precede the FOR control statement.

  The ASG statements incorporated with the FORTRAN IV compiler are as follows:

  ƀCOREƀƀ14000/14000

  ƀASGƀƀRAN,ZH,54000D/60000D, RANDOM FILE

  ƀASGƀƀRANƀZI,6000D/10000D, RANDOM FILE

  The CORE statement, which applies primary storage to the compiler, cannot be overridden; it may only be supplemented. For satisfying the request for random file ZH, $54{,}000_{10}$ to $60{,}000_{10}$ words of mass storage will be assigned; for file ZI,$6{,}000_{10}$ to $10{,}000_{10}$ words of mass storage will be assigned. The primary storage assignment is more than adequate for most compilations.

- Format:

The FOR statement has the following general form:

#FORƀoptionsƀname/version

- Options:

X Abort the element if errors are detected during processing. Continue processing the remainder of the job. Produce the requested listings of the processing and the error diagnostics. If fatal errors are encountered, ignore the option.

Y Accept the compilation although nonfatal errors are detected or elements marked as if in error are included. Produce the RB element as being error free. If fatal errors are encountered, ignore the option.

Z Abort the element if errors are detected during processing. Continue processing the remainder of the job. Produce the requested listings of the processing and the error diagnostics. If fatal errors are encountered, ignore the option.

Absence of X, Y, or Z options implies that the element will be generated and submitted to the user's library in the normal manner. Errors will inhibit compilation. The requested listings of the processing will be produced, if possible, as will the error diagnostics.

C Print compilation time.

I Produce the output listing, single-spaced.

J Produce in line position the number of each card image presented to the compiler and the relative P of the first machine code instruction associated with each image.

L Produce a complete listing of the source and object codes, and all pertinent data, including summarizations and error diagnostics.

N Suppress all listings with the exception of error diagnostics.

P Input cards are 90-column format.

R Produce a listing of all reference subroutines, and of all variable and array names with their relative locations:
non-common – relative to control counter 1
blank common – relative to control counter 2
1st labeled common – relative to control counter 3
2nd labeled common – relative to control counter 4 etc.

T Execute the compiler version in the job library/group library.

- Specifications:

Name/Version specifies the name, and version if any, which will be assigned to the RB element produced. The name may comprise up to ten alphanumeric characters; the version may comprise up to five alphanumeric characters. The version, if used, must be separated from the name by a slash (/). The entry in the name/version field should be used in all future references to the element.

- Example:

#FORƀƀPROGTEST

The source code for the element, PROGTEST, will be compiled, and the RB element will be produced and placed in the job library automatically.

4.4.6. COB Statement

The COB statement is used to call the COBOL compiler into primary storage from the library and to transfer control to the compiler. The primary functions of the compiler is the translation of a set of source code statements written in the COBOL language into relocatable binary (RB) elements which may be collected by the Loader to form absolute elements that are capable of being executed. The initial output of the compiler is a source or MAP element which will direct the Loader in the proper collection of all RB elements produced by the compilation.

The compiled elements are entered automatically into the user's job library where they may be accessed by other system routines. Listings containing the source code and error diagnostics are also produced unless specifically inhibited by the use of an option.

The input data for the compilation must be contained in the primary input stream following the COBOL statement. This is automatic if the data and the COBOL statement originate from the same media. Data originating from other sources must be entered by the SOURCE statement (see 4.5.1).

- Core Requirements:

  The COBOL compiler requires a minimum of $47{,}000_8$ and a maximum of $77{,}700_8$ consecutive primary storage locations.

- Peripheral Requirements:

  The COBOL compiler requires a minimum of $42{,}500_8$ words for random access files, File Code ZH. More storage is acquired by the compiler as needed.

  The COBOL compiler, in the library, contains ASG statements necessary for the acquisition of the usual primary storage and peripheral requirements.

- Format:

  The COB statement has the following general form:

  #COBƀoptionsƀname/version

- Options:

  Absence of X, Y, or Z options implies that the element will be generated and submitted to the user's job library in the normal manner. Errors will inhibit compilation. The requested listings of the processing will be produced, if possible, as will the error diagnostics.

  A Assign sequence numbers to source images.

  B Inhibit sequence check on source images.

  C Indicate checksum tape block.

  D Delete precautionary diagnostics.

  E List diagnostics. ←

  L List source and diagnostics – overrides N and S options, and is assumed if R, N and S options are not present.

  M Inhibits Library listing under Library Copy.

N Inhibits listings – overrides L, R, and S options

R List data and procedure names with locations following source listing.

S Produce a listing of source coding.

T Execute compiler in job library/group library if present, otherwise, execute compiler in systems library.

U Inhibit sequence numbers from source images.

V Inhibit generation of start address in Procedure Division RB element.

W Generate cross reference listing following data and procedure name list if present.

- Specifications:

  Name/Version specifies the name, and version if any, which will be assigned to the source (MAP) output element produced by the compiler. The name may comprise up to ten alphanumeric characters; the version may comprise up to five alphanumeric characters. The version, if used, is separated from the name by a slash (/). In all compilations, the name/version field must contain a minimum of eight alphanumeric characters. The entry in this field should be used in all future references to the compiler output. Since the first eight characters of name will be used (in part) in naming the RB elements produced by the compilation, MAP elements (for which RB elements may occupy the same library) should have names in which the first eight characters are unique.

- Example:

  #COBƀƀTESTPROG

  The source code for the element, TESTPROG, will be compiled, and the source element and RB elements will be produced and placed in the job library automatically.

  NOTE: The collection of the RB elements into an absolute element is effected by a call to the Loader naming the source elements. The K option indicates the named element is a map element and must be present on the LOAD statement; for example:

  #LOADƀKƀTESTPROG, TESTABS/FEB67

  results in the collection of the element TESTABS/FEB67.

4.5. SOURCE CODE FORMAT

Source code input to the language processor is brought in from the primary input stream in the form of variable length images or source statements in Fieldata code. Length variations depend upon the following considerations:

- Language conventions of the individual processor which restrict the format of statements to a maximum length.

- Size of the item read unit record device used for entering images into the primary input stream. For example, and 80-column card reader restricts the image length to 16 words; a 90-column card reader permits a length of 18 words; and a paper tape unit or magnetic tape unit allows virtually unrestricted length.

- Deletion by the unit record routines, of all full words of trailing blank columns, from each image as it is read. This is performed automatically as the image is transferred into the primary input stream so that direct access storage buffering area is conserved.

The source code deck for any one translation is delimited in the job deck as comprising all statements following the processor call and preceding the end-of-task (END) statement. The source deck can be composed of statements submitted directly into the primary input stream, calls for source elements into the primary input stream from the job library, or a combination of both.

4.5.1. SOURCE Routine – Primary Function

The source routine is a program for maintaining and updating source language programs on any medium, and for entering source programs into the primary input stream. The routine provides the user with a means for reducing the size of his program decks. On one preliminary run, large programs or data decks can be formed into source elements by the routine and can be saved through the OUT function on media such as magnetic tape. On succeeding runs, the user need only supply corrections to the element residing on the selected medium, again through the source routine. If corrections become voluminous, such corrections can be formed into a source element, again reducing the amount of necessary input.

4.5.2. SOURCE Statement

The source routine is called, and source elements are maintained, through the SOURCE statement.

- Format:

  The SOURCE statement has the following general form:

  #SOURCEƀoptionsƀname/version1,library,name/version2,begin/increment/column/length,ID/column/length

- Options:

  C Corrections to name/version 1 follow the SOURCE card in the primary input stream. Absence of this option implies that no corrections follow the SOURCE card. No images from the primary input stream are processed if neither the C nor the J option is specified.

  D Delete the source element, name/version 1, from the job library after completion of processing. This option is ignored if the library specification is the systems library/group library.

  E List the update cards. The listed image is limited to 128D characters, including the sequence number. When this option is used, the P option on the source card should not be used.

  J Create a new source element from the images that follow in the primary input stream, and place the new element in the job library. The entire images – beginning with column 1 – will be used to build a new source element called name/version 2. The D and P options have no meaning. The input images may not have #punches in the first column. Input images which are control cards must be blank in column one. The # punch is inserted by the source routine. When the R and D options are used with the J option, the user should be careful not to overlay useful information with these fields.

O Submit the updated source element to secondary output.

P The images from name/version 1 are corrected by image number. Images from name/version 1 are counted decimally from 000100 in increments of 000100. These numbers do not appear on any of the processed images. The first column after the space following the sequence number of the correction image becomes column one of the new source image except in the case of a delete card. When the P option is used, the C option must also be used.

R Resequence the element name/version 1. The new sequence numbers are decimal. See the field description of begin/increment/column/length.

S Do not put the updated source element name/version 2 into the primary input stream. This option must be accompanied by an O and/or U option. Absence of this option implies that the updated element is to be inserted into the primary input stream.

U Enter the updated source element name/version 2 into the job library.

V The option may be used in conjunction with the J option. The presence of the V in addition to the J causes source to accept input images with # punches in the first column. The # punch is still inserted in column one where needed.

NOTE: an #END card is no longer sufficient to halt processing. The source routine continues processing until a #HALT card is found (or the end of primary input). The system tests a #HALT card as an #END card.

JOB cards which are intended to be part of the new source element must still be blank in column one. This is because the input unit record routine sets up a new job upon encountering a #JOB card.

X Abort the job and the remainder of the job stream if errors are encountered.

Y Continue processing although errors are detected in the correction image. The image in error is printed and ignored. Do not mark the element as being in error.

Absence of both the X and Y options indicates that processing is to be terminated if errors are detected. The next task in the job stream is activated. Fatal errors, of course, override the Y option.

If any task control images are being inserted into the primary input stream (no S option) along with source images, the X option is automatically set.

Name/version 1 specifies the name and version, if any, of the source element, contained in the library, which will be processed with or without additional images from the input stream. Name may comprise up to ten alphanumeric characters and version 1 up to five alphanumeric characters. Images within the element are limited to 120D characters.

■ Specifications

SYS indicates the system's library

JOB indicates the user's job library or a group library attached to the job.

xxxxx Group library number, which may be four decimal or five octal digits, specifying the file number by which the group library is identified. Decimal numbers may range from 0 to 9999, and must be indicated by a D following, e.g. 9989D. Octal numbers may range from 0 through 77777.

Name/version 2 specifies the name and version, if any, to be given to the updated source element. Name/version 2 must be given when the U option is used. Name/version 2 may be name/version 1.

begin/increment/column/length. This field applies to the R option as follows:

begin – The sequence number (decimal) to be given to the first image of name/version 2.

Default value: 000000

increment – The decimal increment between successive sequence numbers.

Default value: 000100

column – The column number of the final image where the sequence number is to begin. The column indicates blanks in columns one through six if the column number is specified. This is, if the sequence number is to begin in column 64, the column specification is 64D or 100. The maximum specification is 115D. Some processors truncate their input images.

Default value: 1

length – The number of columns to be occupied by the sequence number. The maximum specification is 10D. The column plus the length cannot exceed 121D.

Default value: 6

ID/column/length

ID – An alphanumeric identification field to be added to each image of name/version 2. This field is added to the final image after the resequence field; thus, if both are specified to begin in the same column, the ID overlays the sequence number.

Default value: ƀƀƀƀƀƀƀƀ

column – The column number (of the final image where the ID is to begin).

Default value: 73D

length – The number of columns that the ID field is to occupy. This is set to 8 if the specification is greater than 10D. The column number plus the length cannot exceed 127D.

Default value: 8

Specifications exceeding any of the above limits causes task termination and the message:

#SOURCE STATEMENT FORMAT ERROR.

Illegal option and/or field specification combinations which produce the same action and error message as shown.

4.5.3. Error Messages

The following error diagnostics may be returned by the source routine:

#SOURCE STATEMENT FORMAT ERROR

This error causes the current task to be terminated. This message is printed if the SOURCE statement contains any of the following illegal option and/or field specifications.

– J, P

– J, C

– J, name/version 1

– No J, No C, no name/version 1

– S, No U, No O

– a field terminator other than , or /

– a resequence column specification of 116D or greater

– a resequence width specification of 11D or greater

– the sum of the resequence column and width exceeds 121D.

– the sum of ID column and width exceeds 128D.

TASK TERMINATED BY #SOURCE

This message follows fatal errors. It is preceded by a message indicating the type of error.

SYSTEM ERROR

This message indicates that an unrecoverable system error has occurred. The task is terminated.

PART OF DRUM LOST TO SYSTEM

This message is accompanied by SYSTEM ERROR.

ELEMENT CANNOT BE LOCATED

The element name/version 1 cannot be found. The task is terminated.

SEQUENCE ERROR

This message is printed when any two successive correction images do not have strictly increasing sequence numbers. This message follows the offending image. The error causes task termination unless the options include S and Y and exclude X.

INVALID DELETE

This message is printed if the initial sequence number specified on a delete card from the input stream does not exist in name/version 1. The message follows the offending image. The error causes task termination unless the options include S and Y and exclude X.

FORMAT ERROR

This message is caused by such things as: a correction image which has fewer characters than the number of characters expected in the sequence number, or a nonnumeric number of images specified on a delete card. The message follows the offending image. The error causes task termination unless the options include S and Y and exclude X.

SYSTEM ERROR IN SECONDARY OUTPUT FOR ABOVE IMAGE

This message indicates that an unrecoverable system error has occurred. This error causes task termination unless the options include S and Y and exclude X.

ILLEGAL IMAGE

A #FIN image was detected when the source routine was submitting the image to the primary input stream. This is a fatal error.

4.5.4. Source Language Correction

Any source element within the job or system library may be corrected or modified by the source image routine. This processing may include correction of the source element, submitting the source element to the primary input stream, constructing a new updated source element, etc. All corrections to the source element are done by means of correction statements which immediately follow the SOURCE image. Any number of corrections may be made.

4.5.4.1. CORRECTION STATEMENTS

The general format of the correction statement is as follows:

nnnniib̸correction

nnnn represents a base number used to specify the positions within the source element at which the correction is to be inserted. Generally, this number is four characters in length, but it may be shorter.

ii represents an insert number which fixes the correction as an insertion between two statements. Generally, the number is two characters in length, but it may be one.

correction is the message which will be inserted. This may be actual data or an instruction.

An alternate format for the correction statement is:

nnnn.iib̸correction

If the period is used, the "nnnn" portion will be right justified with leading zeros and the "ii" portion will be left justified with trailing zeros. (Examples: 32.8 becomes 003280; 1719 becomes 171900; .32 becomes 000032). If the period is not used, the entire number is right justified and is filled with leading zeros. (Examples: 328 becomes 000328; 1719 becomes 001719; 32 becomes 000032; 3280 becomes 003280; 132269 becomes 132269.)

The nnnnii field may contain alphabetic characters.

4.5.4.2. DELETIONS

The deletion of statements within source elements is accomplished by a correction statement containing the word DELETE.

The general format of the delete correction statement is as follows:

nnnnnnƀDELETEƀxxxx

nnnnnn specifies the sequence number (no P option) or image number (P option) of the first image which is to be deleted.

xxxx specifies the number of statements which are to be deleted. The number is octal unless D is appended.

4.5.4.3. INSERTIONS

The insertion of statements within source elements is accomplished by a correction statement containing the word INSERT.

The general format of the insert correction statement is as follows:

nnnnnnƀINSERTƀxxxx

nnnnnn specifies the sequence number (no P option) or image number (P option) after which the statements indicated by the INSERT statement are to be inserted.

xxxx specifies the number of statements which are to be inserted

Note that the statements which are inserted are not changed unless they are control statements having a space in column 1 in which case a # will replace the space.

4.5.5. Examples

The following examples show use of the source routine.

Example 1:

Form a source element from unsequenced source images.

```
#SOURCEƀJUSƀ,,EX1
      DIMENSION ARRAY(10)
      DO 10 I = 1,10
      PRINT 11
11    FORMAT (5x, 6H**HI**)
10    CONTINUE
#END
```

Example 2:

Form a source element from sequenced source images.

```
#SOURCEƀJUSƀ,,EX2
005100                  ENT*A*1
005500                  ENT*Q*1
010000       TAG1       ENT*B1*1
#END
```

In example 1 and example 2 above, the new source element consists of exactly those images appearing between the #SOURCE control card and the #END card.

Example 3:

Form a source element from unsequenced source images.

```
#SOURCEƀJUSVƀ,,EX3
ƀJOBƀƀTEST/CENTER,O,C,C,C/C
#ASGƀJUMRWƀTAPE,A,,OOPS
#INƀRNƀA
#END
#PRTƀIƀJOB
#END
ƀJOBƀƀTEST2/CENTER,O,C,C,C/C
#MSGƀƀHIƀTHERE
#HALT
#ASGƀJUMRWƀTAPE,A,,OOPS
#INƀRNƀA
#END
#PRTƀTƀJOB
#END
#FIN
```

Example 4:

Form a source element from sequenced source images.

```
#SOURCEƀJUSRƀ,,EX4
005100                      ENT*A*1
005200                      ENT*Q*1
010000         TAG1         ENT*B1*1
#END
```

The source elements formed in the examples 3 and 4 are respectively:

```
#JOBƀƀTEST/CENTER,O,C,C,C/C
#ASGƀJUMRWƀTAPE,A,,OOPS
#INƀRNƀA
#END
#PRTƀTƀJOB
#END
#JOBƀƀTEST2/CENTER,O,C,C,C/C
#MSGƀƀHIƀTHERE
000000                      ENT*A*1
000100                      ENT*Q*1
000200         TAG1         ENT*B1*1
```

In the following examples, corrections are applied to the source elements produced by examples 1 through 4.

Example 5:

Correct and compile the source element from example 1.

```
#FORƀJYƀEX5
#SOURCEƀCUDPƀEX1,,EX5
000100ƀDELETEƀ1
000200ƀƀƀƀƀƀƀƀDO 10 I = 1,15
000550ƀƀƀƀƀƀƀƀSTOP
000675ƀƀƀƀƀƀƀƀEND
#END
```

The corrected source element will be:

```
        DO 10 J = 1,15
        PRINT 11
11      FORMAT (5X, 6H**HI**)
10      continue
        STOP
        END
```

Example 6:

Correct and assemble the source element from example 2.

```
#SPURTƀRPƀEX6
#SOURCEƀCUDƀEX2,,EX6
002000             ENT*B2*5
005100             ENT*A*2
005500ƀdeleteƀ2    BJP*B2* $–1
010101
#END
```

The corrected source element will be:

```
002000             ENT*B2*5
005100             ENT*A*2
010101             BJP*B2* $–1
```

Example 7:

Correct the source element from example 3.

```
#SOURCEƀCUDSPƀEX3,,EX7
000500ƀ#SOURCEƀCUDSƀTEST,,TEST
000600ƀ000100ƀTAG1                  ENT*A*1
000650ƀ#END
000700ƀdeleteƀ2
#END
```

The corrected source element will be:

```
#JOBƀƀTEST/CENTER,O,C,C,C/C
#ASGƀJUMRWƀTAPE,A,,OOPS
#INƀRNƀA
#END
#SOURCEƀCUDSƀTEST,,TEST
000100ƀTAG1                         ENT*A*1
#END
```

Example 8:

Update two source elements, combine the two elements into one long element, and obtain a SPURT assembly of the resultant element.

```
#SPURTƀRPƀEXAMPLE
#SOURCEƀCUDSƀELT/1,,ELT/1              ƀcard1
[sequenced corrections to element 1
#SOURCEƀCURDSƀELT/2,,ELT/2,100000      ƀcard2
[sequenced corrections to element 2
#SOURCEƀCUPƀELT/2                      ƀcard3
000001ƀ#SOURCEƀCURƀELT/1,,ELT/3         ƀcard4
#END
```

Card 1 and the corrections that follow update ELT/1, and put the updated version into the job library. Card 2 and the corrections that follow update ELT/2, and resequence the element. The important consideration is to resequence ELT/2 beginning at some number larger than the largest sequence number of ELT/1. Card 3 inserts card 4 into ELT/2 as the new first image, and because card 3 has no S option, the updated version is put back into primary input stream. Thus, the next control card that will be found is:

```
#SOURCEƀCURƀELT/1,,ELT/3               ƀcard4
```

The rest of ELT/2 follows card 4 in the input stream, and is inserted as corrections to the end of ELT/1. The resequenced version is put into the job library as ELT/3 and also is submitted to primary input to be assembled by SPURT.

Example 9:

Obtain a listing of a job deck. This can be accomplished by inserting a card such as the following as the second card of the deck – and proceeding to send in the deck as usual.

```
#SOURCEƀJUSVEƀ,,EXAMPLE
```

NOTE: Any blank cards in the deck are not listed because the input unit record routine ignores them.

Example 10:

Use of the INSERT facility without the P option on the source statement

```
#SOURCEƀSUCDƀELT/1,,ELT/2
000500ƀINSERTƀ4D
ƀASGƀJUMRINƀTAPE,A,,FORTRAN
ƀINƀLƀA
ƀDEL
ƀEND
#END
```

In this example the four statements following the INSERT statement will be inserted immediately after the card with the sequence no. 0 0 0 5 0 0 in element ELT/1. The cards inserted will be unchanged except that a # will replace the space in column one.

Example 11:

Use of the INSERT facility with the P option on the source statement

```
#SOURCEƀPSUCDƀELT/1,,ELT/2
000500ƀINSERTƀ3D
010200           ENT* B2*5
010220           ENT* A*2
010230           ENT* Q*6
#END
```

In this example the three statements following the INSERT statement will be inserted immediately after the fifth card image of ELT/1. No changes will be made to these three statements.

4.6. LOADER

The Loader is a processor which provides a flexible and efficient means for collecting independent relocatable binary (RB) elements to produce a load or object program for execution as a task under control of OMEGA. An RB element is an intermediate output code and is not executable. The Loader joins RB elements generated from source statements expressed in FORTRAN IV, COBOL, 494 SPURT, and 494 Assembler languages, and so on. Language differences are resolved, references and cross references are satisfied, and disparate components are integrated in the joining process, called collection. The Loader does not actually load a program into memory for execution; rather, the Loader constructs the entity which may be read and executed. This organization facilitates compiling and debugging of small or select parts of a total program, and combinations of these individual parts for execution, without recompiling the entire set of individual parts. The absolute program formed by the Loader is base relative for machines working in the UNIVAC 494 mode under OMEGA; that is, the program can be read into any memory area for execution without modification of instructions. The absolute program is an entity, with no unresolved references, which might be read in and executed. The program's relocatability is inherent with regard to system references. The load program may also be formed with base absolute for machines working in the 490 mode, in which case the instruction may be modified by the UNIVAC 490 system at execution time.

Separate elements existing in the job, group, or system libraries are collected in constructing an object program. Elements are collected on the basis of an external reference, or XREF, in the one element which can be satisfied by an ENTRY definition, or EDEF, within a second element. The Loader may be directed to include or exclude specific elements by secondary control statements.

The basic output of the Loader is the absolute object program which is entered into the job library with the name specified by the user. Optional output includes a list of labels and tags contained in the program for utilization in testing procedures. Error messages or a storage layout listing may be obtained as a hard copy record of the collection process. The Loader can also transfer the secondary control language as a job library element for subsequent reference.

The order of user specified elements within a program segment will be maintained as specifically named by INCLUDE statements. All elements included by a library search are located in the highest level segment from which the segment can be accessed by all elements that refer to it. A starting point in the program is determined during text modification or as indicated by an ENTRY control statement. Any element may specify a starting address; the Loader will accept the first address encountered. The ENTRY statement will take precedence over a start address.

4.6.1. Control Statement Format

The control statements used by the Loader can be made in primary and secondary control language. The primary control statement is the LOAD statement. Secondary control statements used are MAP, INCLUDE, EXCLUDE, SEGMENT, ENTRY, and EQUALS. The general format for all statements follow system conventions.

Secondary control statements have the same general format as primary statements with the exception that the first position in the secondary statement must be blank.

In the element specification on Loader control statements, the version name must be separated from the element name by a slash (/). The name is limited to ten alphanumeric characters, and the version is limited to five alphanumeric characters.

4.6.2. The LOAD Statement

The Loader is scheduled and activated in response to a LOAD control statement in the input job stream which calls for the collection of an RB element for processing. Information on the #LOAD card is comprehensive enough to direct the collection and loading of most programs.

- Format:

  The LOAD statement has the following general form:

  #LOADb̸optionsb̸field 1, field 2, field 3

- Options:

  F Inhibit implicit inclusion of elements within the job library. Implicit inclusion of elements within the group and system libraries are not inhibited by this option. Only those elements specified by field 1 of the LOAD statement and by the operands of INCLUDE secondary control statements are collected. Elements within the group or system libraries may still be collected implicitly by an unsatisfied XREF, but not elements within the job library.

  K The element in Field 1 is a MAP element.

  U Save the secondary language element in the job library. The third field of the specifications list is given as the element name. Implicit inclusion becomes explicit inclusion, by this process.

  M Append modification to the LOAD element. This option should be used when the Loader output will be executed on a machine without a Relative Index Register (UNIVAC 490 mode).

X Abort the task and the remainder of the job (i.e., skip to next JOB card) if errors are detected. Produce the requested listing, if possible, and the error diagnostics.

Y Accept the program for execution although errors are detected during collection, or elements marked as in error are included. If the errors prevent the production of an absolute program, the Y option will be ignored.

Z Abort the task if errors occur during the collection. The listings requested will be produced, if possible, with the error diagnostics. No absolute element will be produced for the task. This option differs from the X option in that a Z option will result in the current load function being aborted only, with processing continuing on the next control statement.

If an X, Y, or Z option is not given, an error will inhibit execution of the program but the remainder of the job will be produced.

L Produce a complete listing summarizing the memory space used by each element included in the program. The XREF's are listed with the VALUE of the corresponding EDEF's.

N Produce no listing. The N option will be overridden if error diagnostics are output.

If neither an N or L option is indicated, only summary information will be printed.

S Produce a list of symbol definitions (SDEF's) which may be used by the system diagnostic routine.

- Specifications:

Field 1 in the LOAD statement is optional. Normally the field specifies the name and version of the basic element scheduled for collection, which may be an RB element to be included in the resident portion of the collection, or which may be the name and version of a MAP element (see 4.6.5) containing control statements necessary for directing the collection process. If this field is left blank, it is assumed that the collection will be directed by the secondary control statements following the LOAD statement.

Field 2 is required, and contains the name and version which will be given to the absolute element produced by the Loader. This name and version must be used in any reference to the absolute element being created.

Field 3 is optional. If the field is present together with a U option, the name and version in the field will be assigned as the name and version of the MAP element produced (see 4.6.5).

- Example:

A simple collection may be performed by the inclusion of a single LOAD statement, or a more complex collection may be performed if the basic element specified is a MAP element. The following examples will show possible methods for using the LOAD statement alone; however, the statement could be used in conjunction with secondary control statements.

Example 1:

Perform the following collection:

```
#LOADƀLYUƀELT/RB,CELT/ABS,CELT/MAP
```

This statement causes the Loader to locate the element with the name/version, ELT/RB. If the element is an RB element, the element is included in the collection with any other element which, as the basic element, it references.

An absolute element with the name/version, CELT/ABS, is formed from the collected RB elements and placed in the user's job library. Because of the Y option, the absolute element is not marked as being in error, although some noncritical errors may have occurred in the collection.

A compressed source element, referred to as a MAP, is placed in the user's job library. The MAP consists of control statements necessary to recreate the collection with the use of INCLUDE and other secondary control statements.

If name/version, ELT/RB, had referred to a MAP element, under the K option, the control statements contained in that element would have been processed and used to guide the collection process.

A printed output of information concerning the collection is obtained because the L option was specified.

Example 2:

Perform the following collection:

#LOADƀƀ,CELT/ABS

The statement illustrates the condition in which no basic element is specified in the LOAD statement. This implies that the LOAD statement is followed by secondary control statements directing the collection process.

Secondary control statements may be necessary even if a basic element is called for on the LOAD statement.

The absolute element is formed as prescribed by the available control statements and is entered into the user's job library with the name/version, CELT/ABS.

A MAP element is not produced because of the absence of the name/version field for the MAP element and the absence of the U option.

Because no list options are indicated, a partial listing is obtained with the absolute element. Errors which occurred during the collection are indicated and are checked on subsequent control statements referring to the element (e.g., #GO).

4.6.3. Secondary Control Statements

Construction of segmented programs, or of particular collections, is described by secondary control language which normally follows the LOAD statement. The Loader can be directed, however, to place a set of these control statements into a job library as an element (see 4.6.4). Subsequent execution of the Loader may utilize this element as directed by the LOAD control statement.

The secondary control language recognized by the Loader permits description of the most complex programs. The user can enter these control statements with the input stream for each collection, or, the user can specify a library element of statements to control the collection.

The control statements recognized by the Loader are MAP, INCLUDE, EXCLUDE, SEGMENT, ENTRY, and EQUALS. All secondary control language statements are blank in column 1 to distinguish them from primary control language statements.

4.6.3.1. SEGMENT STATEMENT

The Loader provides a straightforward means for constructing overlay segments. For each segment, the user prepares MAP, INCLUDE, and/or EXCLUDE control statements specifying which relocatable elements will be contained within the segment. These statements are preceded by a SEGMENT statement indicating the name of the segment and its logical origin.

When a segmented program is called for execution, only the main segment is initially loaded. There are two methods by which other segments may be loaded. The direct method is used whenever a direct call is made to the overlay supervisor, specifying the segment to be loaded and the location to which control will be transferred. The indirect method provides automatic loading of a segment referenced by a JP or RJP instruction (or their equivalents in another language) whenever the segment is not in primary storage. The mechanics for such loading are set up by the Loader and carried out by the overlay supervisor. The Loader replaces the address of an entry vector which, in turn, jumps to the location of the externally defined symbol. If the overlay supervisor is entered, the required segment is loaded, and control is transferred in the same way that the original jump intended. All registers are preserved by the process, and all necessary entry vectors are reset at the loading of any segment. Segmented programs should not exceed a total length of 77700 if automatic segment loading is employed.

Any reference to a secondary segment from outside that segment will be directed to the vector table created to allow jump and return jump operators to be used in communicating between segments. Reference to an element within a secondary segment from outside that segment, other than a jump or return jump, will have an address within the vector table as the effective address, resulting in an incorrect reference.

The SEGMENT statement is used to declare the beginning of a new segment.

- Format:

  The SEGMENT statement has the following general form:

  ƀSEGMENTƀsegname1,segname2

  or

  ƀSEGMENTƀsegname1, (segname2)

- Specifications:

  Segname1 contains the name of the segment, which must always be specified.

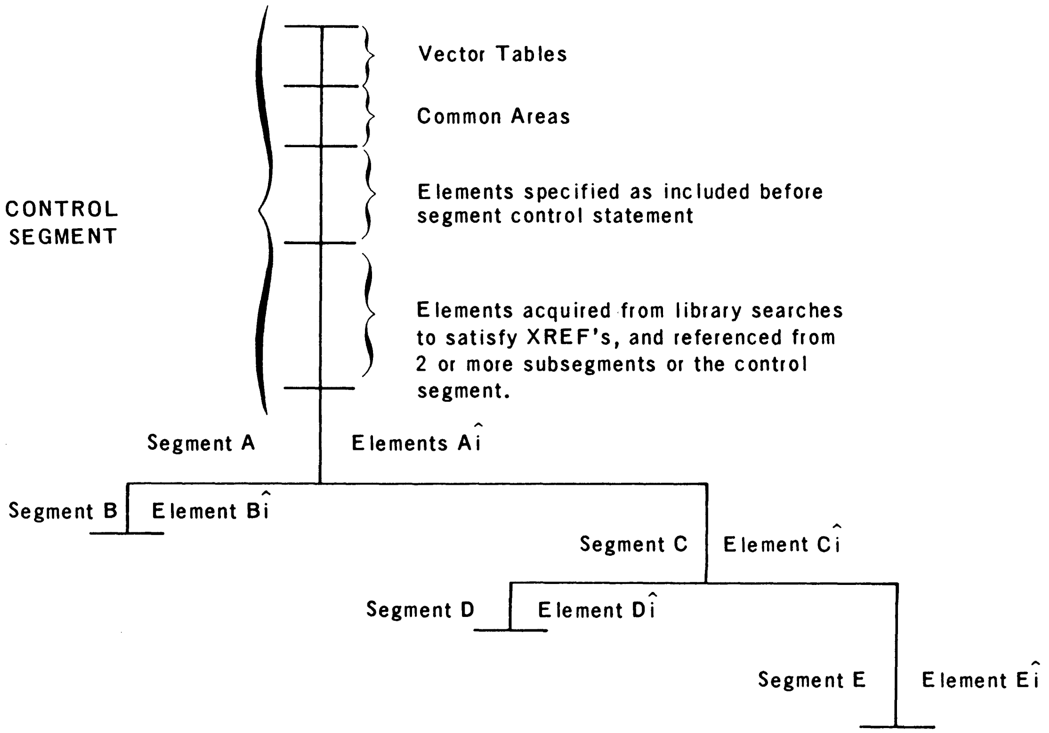
  Segname2 without parentheses specifies that the segment identified by segname1 starts immediately after segname2. If the field contains a series of segment names enclosed in parentheses and separated by commas, the Loader starts the segment identified by segname1 immediately following the highest location occupied by any of these. If the field is void, the segment identified by segname1 has its origin immediately following the preceding segment.

- Example:

  The following example illustrates the control statements and resultant memory layout for segmentation (see 4.6.3.2 for INCLUDE):

INSTRUCTION	COMMENT
#LOAD P,P1	Element P is automatically included in the resident portion of the collection. The output will be assigned the name, P1.
SEGMENT A INCLUDE A1,A2,A3	The segment, A, is originated immediately following the preceding segment (RESIDENT).
SEGMENT B, (A) INCLUDE B1,B2	Identifies specific elements to be collected. The segment, B, is originated immediately following segment A.
SEGMENT C, B INCLUDE C1,C2,C3	The segment, C, is originated at the same position as segment B.
SEGMENT D, (B,C) INCLUDE D1,D2,D3	The segment, D, is originated following the longest of of segments B and C.
SEGMENT E, D INCLUDE E1,E2,E3,E4	The segment, E, is originated at the same position as segment D.
#END #GOƀƀP1	

The set of control statements would result in the following memory structure:


Vector Tables
Common Areas
CONTROL SEGMENT
Elements specified as included before segment control statement
Elements acquired from library searches to satisfy XREF's, and referenced from 2 or more subsegments or the control segment.
Segment A
Elements Aî
Segment B
Element Bî
Segment C
Element Cî
Segment D
Element Dî
Segment E
Element Eî

4.6.3.2. INCLUDE STATEMENT

The INCLUDE statement specifies that the elements named on the statement should be included in the collection. Elements collected will be included in the order in which they appear on the statement. At least one INCLUDE or MAP statement must follow a SEGMENT statement.

- Format:

  The INCLUDE statement has the following general form:

  ƀINCLUDEƀname/version,name/version...

- Specifications:

  Name/Version refers to an RB element which is contained in one of the libraries and which is to be included in the collection. More than one name/version may be specified in one statement by separating the fields with commas.

- Example:

  ƀINCLUDEƀELT1/RB,ELT2/RB

  The elements ELT1/RB and ELT2/RB will be included in the collection process within the segment in which the INCLUDE statement is located. If elements are referenced by an included element, but are not specifically stated as being included, they will be placed in the resident portion of the collected element from which they can be referenced freely by any element within any segment.

4.6.3.3. MAP STATEMENT

The MAP statement specifies a MAP element(s) (see 4.6.5) which may contain part or all of the secondary control statements necessary for the collection. The secondary control statement contained in the MAP element are added to the secondary control stream of the Loader at the point occupied by the MAP statement.

- Format:

  The MAP statement has the following general form:

  ƀMAPƀname/version,name/version,...

- Specifications:

  Name/Version refers to a MAP element which exists within the library complex. More than one MAP element may be specified on a MAP statement by separating the fields with commas.

- Example:

  ƀMAPƀSOURCE/V1,MAP/VX

  A search is initiated through the library complex for the MAP element, SOURCE/V1. The search begins in the job library, goes through group libraries linked to this job, and finally checks the system library. When the MAP element is located, the secondary control statements contained in the element are processed. The MAP element, MAP/VX, is processed in the same way as SOURCE/V1. An error message is printed if the element is not found. When the processing of the MAP element is complete, the Loader continues with the processing of any following secondary control statements contained in the primary input stream.

4.6.3.4. EXCLUDE STATEMENT

The EXCLUDE statement, which is essentially the inverse of the INCLUDE statement, allows the user to state explicitly which elements should not be included in a collection. All RB elements implied by the collection, other than those specifically excluded, are accepted.

- Format:

  The EXCLUDE statement has the following general form:

```
ƀEXCLUDEƀname/version,name/version...
```

- Specifications:

  Name/version specifies an element which is to be excluded from the collection. More than one element may be specified by separating the name/version fields with commas.

  NOTE: An element cannot appear in both an INCLUDE statement and an EXCLUDE statement within the same segment.

- Example:

```
#LOADƀYLƀELTA,ELTB
ƀEXCLUDEƀELTXREF
```

  The example implies that during the collection of the element, ELTA, and those elements referenced by ELTA, and collected elements, an element, ELTXREF, may be referenced. The EXCLUDE statement will prevent the ELTXREF element from being included in the collection.

4.6.3.5. ENTRY STATEMENT

The ENTRY statement provides the capability for overriding the starting address specified by the assembler or through the entrance to a main program generated by FORTRAN IV or COBOL.

- Format:

  The ENTRY statement has the following general form:

```
ƀENTRYƀname
```

- Specifications:

  Name is an externally defined (EDEF) symbol of some element within the library complex which will be included in the collection. The starting point may be anywhere within the absolute element. If the starting point is within a subsegment, the segment will be loaded automatically upon the initiation of the routine.

4.6.3.6. EQUALS STATEMENT

The EQUALS statement provides a capability for assigning a value to an external reference (XREF) at collection time. The EQUALS statement takes precedence over EDEF's in satisfaction of XREF's during collection. The user may find need for the EQUALS statement as the result of an error in compilation/assembly or an error detected in a previous collection. Any unsatisfied XREF's will be listed by the Loader.

- Format:

The EQUALS statement has the following general form:

ƀEQUALSƀname/value,name/value

- Specifications:

Name/Value specifies a symbol to be defined and the value to be used. More than one symbol can be specified by separating the name/value fields with commas. The value may be an octal or decimal integer, or it may be a symbol with or without an integer. Such symbols must be externally defined (EDEF) by one of the collected elements.

- Example:

ƀEQUALSƀNAMEIT/value	NAMEIT is the XREF, and value is assigned by the user.
ƀEQUALSƀTAGAB/00165	XREF TAGAB will now have an absolute value of 165.
ƀEQUALSƀNUTAG/OLDTAG+6	NUTAG will be assigned the value of OLDTAG plus 6. OLDTAG must be an EDEF in the collection.
ƀEQUALSƀROOT/24680D	ROOT will be assigned the octal equivalent of the decimal number specified.

4.6.3.7. MISCELLANEOUS

Primary control statements such as ASG and LOG can be submitted as part of the Loader's secondary language. These statements will conform to the normal primary control language format except that a blank must be in column 1 of the statement card. The Loader will accept such cards and form them into a block of statements associated with the absolute program. The statements will thereafter be utilized by OMEGA whenever the absolute program is called for execution. By this means, the ASG statements necessary for selection and execution of the program can be permanently declared and associated with the program. The statements may be placed anywhere within the Loader's secondary control language.

- Example:

```
#LOADƀUXLƀ,ABSELT,MAPELT
ƀINCLUDEƀELT1,ELT2
ƀSEGMENTƀA
ƀINCLUDEƀECTS1
ƀSEGMENTƀ,B,A
ƀINCLUDEƀELTS2
ƀASGƀRƀPNAME,A,1000,DRUM
ƀLOGƀ000START,ABSELT
#END
#GOƀƀABSELT
#END
```

The ASG and LOG statements would be attached to the absolute element (ABSELT) produced, and would be processed upon the execution of the element (#GO).

4.6.4. Loader Requirements

The Loader requires both primary storage and direct access storage for the collection process.

- Primary Storage Requirements:

  The Loader requires a minimum of 4096_{10} and a maximum of 8192_{10} primary storage locations. The efficiency of the Loader is directly proportional to the amount of primary storage which it receives. If a minimum of primary storage locations is received, however, no unreasonable restrictions are made upon the size or number of elements involved in the collection of the program.

  A system restriction imposed is that no collected routine may occupy more than 77777 octal consecutive primary storage locations. Overlays due to segmentation are not counted in the restriction.

- Mass Storage Requirements:

  The Loader requests from $65{,}536_{10}$ mass storage locations for the collection process. This storage is used for various tables and lists required for the collection as well as for the establishment of a temporary storage area where the absolute routine will be built.

4.6.5. MAP Elements

A MAP element is a source language element contained in the program libraries and is composed of control statements that act as a secondary control language for directing a collection.

A MAP element may be specified in the LOAD secondary control language and will be processed when encountered. MAP elements may not be nested; that is, a MAP statement may not occur within a MAP element.

The MAP element may be produced by the user or be produced as a secondary output of the Loader (see 4.6.2, LOAD statement). If it is produced by the Loader, the element consists of all control statements necessary to recreate the collection just performed. This secondary input (MAP) may be output by way of the Program Library Editor, using the OUT statement, and re-entered by way of the Program Library Editor, using the IN statement to direct any further collection of the element.

- Example:

```
#LOAD K MAPELT, ABSELT
#END
#GO                ABSELT
#END
```

or

```
#LOADƀƀ,ABSELT
ƀMAPƀMAPELT
#END
#GOƀƀABSELT
#END
```

NOTE: If more than one MAP element is to be specified, the MAP statement must be used.

4.6.6. Error Diagnostics

During the collection of an absolute element, the Loader may encounter certain conditions which will cause an error message to be submitted to the primary output. This aids in pinpointing the circumstances of the error condition. The action taken by the Loader is dependent upon the options provided on the LOAD statement and upon the nature of the error. The applicable options (see 4.6.2) may be summarized as follows:

X Abort the current collection and the remainder of the job after sending the appropriate error message if errors occur.

Y Complete the collection after sending the appropriate error message if non-critical errors occur.

Z Abort the current collection, but continue processing the remainder of the job after sending the appropriate error message if errors occur.

If X, Y, or Z options are not present, the action will be that of the Y option, with the exception that error flags will be attached to any absolute element produced.

- Error Group:

  The error conditions are divided into three groups on the basis of degree of severity:

  Group 1 contains those error conditions which are not necessarily critical, although the possibility exists that the error conditions may cause an invalid collection.

  Group 2 consists of those error conditions which will probably cause some invalidity in the absolute element produced, but of such nature that the element may still be executable under certain conditions.

  Group 3 consists of those errors which will undoubtedly cause an invalid collection. When errors of this kind occur, the absolute element will be marked as being in error regardless of the options in the LOAD statement.

- Error Messages:

  When an error concerns a control statement, the error message will immediately follow the control statement on the primary output media. Messages concerning particular elements or XREF's will contain some identification of the element affected.

  The following is a list of possible error messages submitted by the Loader, with interpretations:

  – Group 1

    MULTIPLE ENTRY STATEMENTS

    More than one ENTRY statement has been encountered in the secondary control language. The first ENTRY encountered takes precedence.

    MULTIPLE START ADDRESS

    More than one start address has been encountered during the processing of the text for the included elements. The first start address encountered takes precedence.

– Group 2

CONTROL STATEMENT FORMAT ERROR

The control statement being processed contains some error in the data in the statement. If the options are such that processing is to continue, the Loader will make an assumption, if possible, as to the meaning of the data, and will continue with the collection.

name version DUPLICATE N/V

An element named has been requested for inclusion in the same segment twice, or an element appearing in more than one subsegment (INCLUDEd) has been referenced from some segment not containing the element referenced. The error message will be submitted, and the reference will be made to the current element N/V.

SEGMENT NAME NOT DEFINED

A segment name specified on the SEGMENT statement, giving the relative position of the segment, has not been encountered. The segment being defined will be positioned immediately following the last segment defined.

name version CANNOT BE FOUND

The element named was specified as being included but could not be located within the libraries. All references to points within this element will be set as references to 77777.

INCLUDED ELEMENT CANNOT BE EXCLUDED, name version

An element has been requested to be included through an INCLUDE statement and also to be excluded through an EXCLUDE statement within the same segment. The element will be included in the collection.

XREF NAME IS AN UNSATISFIED XREF

A reference has been made to an entry point which is not defined within the libraries. The reference will be set to 77777.

– Group 3

DEPOSIT ADDRESS OUTSIDE SEGMENT LIMITS

In the bit stream of the text modification, a deposit address has been encountered which falls outside of the segment limits.

– INVALID REFERENCE NUMBER IN name/version

In text modification, for name/version, a reference is made to a control counter or external reference that is nonexistent.

TABLE OVERFLOW

The primary storage area assigned to the Loader has been depleted although its tables and lists are dynamically expanded. The collection will be aborted.

XREF MODIFIED BY INFO CC IN name/version

In text modification for name/version, the control counter for modifying an external reference has been assigned as a common storage control counter. Fatal error-collection aborted.

ERROR ON DRUM OPERATION IN COLLECTOR

Data required by the Loader for a collection cannot be successfully transferred to or from the mass storage devices available. The collection will be aborted.

START ADDR OR ENTRY REQUIRED

A start address or an ENTRY statement is necessary in a segmented collection. Without one of these, the absolute element cannot be executed.

SEGMENT TABLE FULL. COLLECTION DISCONTINUED.

The internal table kept by the Loader for its Segment Map has been filled because too many SEGMENT statements have been used. The absolute maximum number of SEGMENTS allowed is 100_8. However, this will be dynamically reduced if complex segments are defined.

4.6.7. Element Selection and Placement Algorithm

In the collection process, the selection and placement of the elements involved are predictable according to the following rules:

- Selection:

  When the Loader is searching for an element to include in the collection, the first element found with the appropriate name and version will be selected. The libraries are searched in the following order: job, group, and system. Because of the manner in which elements are added to the libraries, this method will insure that the last element entered will be the first element checked. This means that an element could be overridden by the later addition of the same element in an updated form.

  In the satisfaction of external references (XREF's) with the entry definitions (EDEF's) of other elements, the first EDEF which satisfies the XREF will be assumed as the proper definition, providing that the EDEF and XREF are in the same segment.

- Placement:

  The placement of collected elements within the collection of elements is determined by the order of inclusion. Those elements specified on INCLUDE or MAP statements will be placed within the specified segment, in the order in which the elements appeared on the statements.

  Elements which are not specifically named in an INCLUDE statement, but which are included because of an XREF, will be positioned following the last element in the segment.

  The general rule for the placement of elements which are not specified as being included is:

  Elements referenced from an element within a segment will be placed in that segment in the order in which they are referenced.

  The exceptions to this rule are:

  If an element specifically included in another segment is referenced, the reference will be made to the element specifically included in the other segment. The element will not be placed within the segment from which the reference came. If an element, not included, is referenced by more than one segment, the element will be placed in the resident portion where it can be freely referenced from all subsegments.

- Example:

The following illustrates the selection and placement algorithm:

```
#INƀCƀELTA,ELTB,ELTC,ELTA1,ELTA2,ELTA3,ELTA4,CONTROL
   RB card decks for elements named on previous IN
   function

#INƀCƀELTB1,ELTB2,ELTB3,ELTB4,ELTC1,ELTC2,ELTC3
   RB card decks for elements named on previous IN
   function

#LOADƀYLƀ     CONTROL, LOADELT
ƀSEGMENT      SEG1
ƀINCLUDE      ELTA
ƀSEGMENT      SEG2, SEG1
ƀINCLUDE      ELTB
ƀSEGMENT      SEG3, SEG2
ƀINCLUDE      ELTC
#END
```

Assuming that the element contains the references shown below, the primary storage layout and placement of the resultant LOAD element can be shown by the following diagram.

	Reference		Reference		Reference	
ELTA	→	ELTA1				
	→	ELTA2	→	ELTA3	→	ELTA4
	→	CONTROL				
	→	ELTB				
	→	ELTC				
ELTB	→	ELTB1	→	ELTB2	→	ELTB4
	→	ELTB3				
	→	CONTROL				
	→	ELTC				
ELTC	→	ELTC1				
	→	ELTC2				

UNIVAC 494 Real Time System
Operating System Programmer Reference
UP-7504 Rev. 2

May 1973

UPDATING PACKAGE A

File pages as specified below

SECTION	DESTROY FORMER PAGES NUMBERED	FILE NEW PAGES NUMBERED
Front Cover & Disclaimer	†	†
Page Status Summary	PSS-1	PSS-1A
Contents	1 and 2	1A and 2A
	3 and 4	3 and 4A
	5 and 6	5A and 6A
	7 and 8	7 and 8A
	9 and 10	9 and 10A
	N. A.	11A**
Section 2	3 and 4	3 and 4A
	19 and 20	19A and 20A
	N. A.	20aA** and 20bA**
	21 and 22	21A and 22
	43 and 44	43A and 44A
	N. A.	44aA**
	61 and 62	61 and 62A
Section 3	9 and 10	9A and 10
	11 and 12	11 and 12A
	13 and 14	13A and 14A
	15 and 16	15A and 16
	19 and 20	19A and 20
	37 and 38	37A and 38
	45 and 46	45A and 46
	47 and 48	47A and 48
	49 and 50	49 and 50A
	53 and 54	53A and 54A
	67 and 68	67A and 68
Section 4	7 and 8	7 and 8A
	11 and 12	11 and 12A
	13 and 14	13A and 14
	19 and 20	19 and 20A
	N. A.	20aA**
	21 and 22	21 and 22A
	23 and 24	23A and 24A
	N. A.	24aA**
	33 and 34	33 and 34A
	39 and 40	39 and 40A
	41 and 42	41 and 42A
Section 8	17 and 18	17A and 18
Section 9	9 and 10	9 and 10A
	N. A.	10aA**
	17 and 18	17 and 18A

UNIVAC 494 Real Time System
Operating System Programmer Reference
UP-7504 Rev. 2

May 1973

UPDATING PACKAGE A (Cont'd)

File pages as specified below

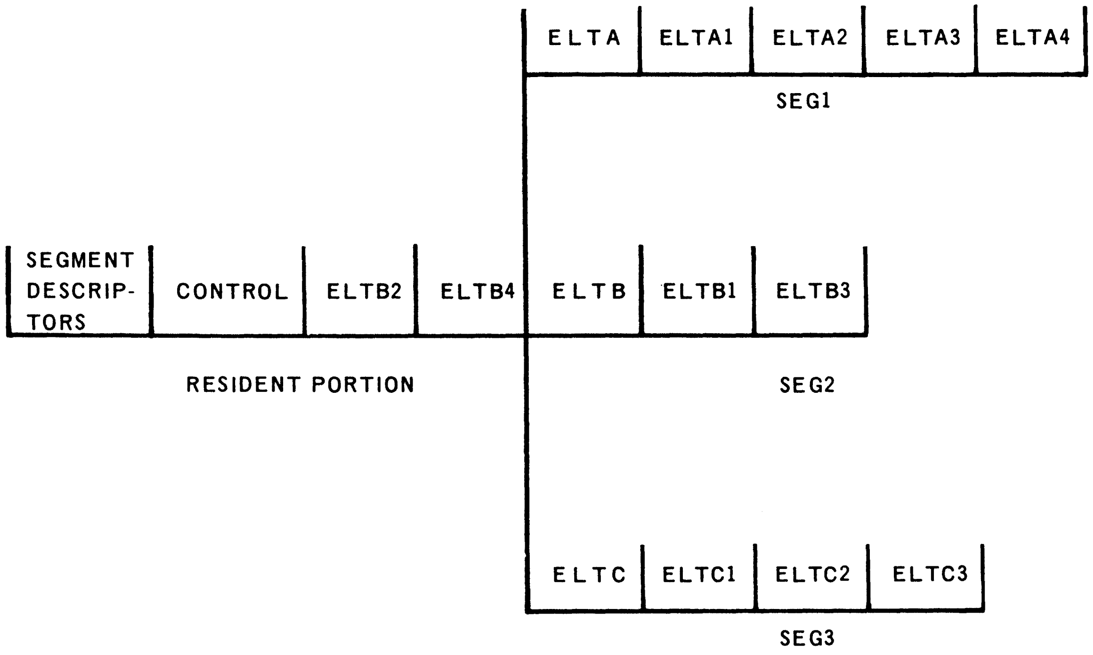
SECTION	DESTROY FORMER PAGES NUMBERED	FILE NEW PAGES NUMBERED
Appendix B	N. A.	1A** thru 7A**

†Destroy old cover and file new cover.

**These are new pages

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

	Reference		Reference	Reference
ELTC	→	ELTC3		
	→	CONTROL		
	→	ELTA		
	→	ELTB2		



An explanation of the positioning of the elements in the diagram follows:

Resident	Segment Descriptors	Segmentation Descriptors Produced by the Loader
	CONTROL	Presence on the LOAD statement implies inclusion in the resident portion
	ELTB2	Referenced from SEG2 and SEG3, therefore located in the resident portion
	ELTB4	Referenced by ELTB2 which is in the resident portion
SEG1	ELTA	Specified on the INCLUDE statement.
	ELTA1–ELTA4	Included because of XREF's from within the statement.
SEG2	ELTB	Specified on INCLUDE statement
	ELTB1,ELTB3	Included because of XREF's from an element within the segment. ELTB2 is not included in this segment because it was referenced from more than one subsegment and was not specified on an INCLUDE statement.
SEG3	ELTC	Specified on an INCLUDE statement
	ELTC1–ELTC3	Included because of XREF's

4.6.8. Loader List Option

The Loader provides the user with the capability for obtaining a hard copy summary of the collection being performed. If an L option is present on the LOAD statement, a detailed description is produced and submitted to the primary output. If the N option is present, no descriptive information is given, with the exception of error diagnostics. If neither L nor N is present, a partial description is provided. The description of the collection may be used to obtain information such as the relative location of elements and references within the collection.

In the listings provided, the headings have the meanings given below:

- Segment Header Line

  SEGMENT NAME — The name of the segment specified on the SEGMENT control statement.

  NUMBER — The number assigned to the named segment for identification. In referring to the segment, this number will be used.

  BASE — The base address of the segment relative to the base address of the total collection.

  LENGTH — The combined length of all elements and other information within the segment.

- Element Header Line

  ELEMENT NAME VERSION
  The name and version of the element which is included within the segment.

  LENGTH
  Length in words, octal, of the element.

  BASE
  The relative address assigned to the first word of the data under the specified control counter.

  CC
  The control counter number assigned to a mass of data.

  COMMON NAME
  The name of a common area defined by the element

  VALUE
  The base assigned to the named common element.

- Satisfied External Reference Line

  SXREF
  The name of the satisfied external reference.

  VALUE
  The relative address of the point which is referenced.

  ELEMENT N/V
  The element which contained the external definition (EDEF) which satisfied the named XREF.

  SEG
  The identifying number of the segment within which the referenced element is contained.

4.7. PROGRAM LIBRARY EDITOR

The Program Library Editor consists of a set of routines through which the library complex may be maintained. Most of the services provided by the Program Library Editor may be obtained through the control statements within the primary input stream.

The Program Library Editor can transfer or access any type of element within any of the libraries: job, group, or system. Certain restrictions are imposed upon specific operations concerning a group library or the system library. These libraries may not be altered in any way by the Program Library Editor. If it is necessary to alter or change a routine in either library, the Element Library Maintenance (ELM) routine must be used. This prevents the inadvertent change or destruction of an element within a library having multiple users.

Program Library Editor control statements include IN, OUT, PRT, and DEL. In any operation of the Program Library Editor, a general rule concerning the order in which elements are accessed in the library is: the last element brought IN is the first element brought OUT.

4.7.1. IN Statement

The IN control statement is used to transfer elements from some media (primary input, tape, etc.) to the job library from which the elements may be utilized. The elements transferred may be of any type: source, RB, or load. Any RB or load element will have been obtained previously as an output of a language processor or the Loader. A source element may be set up manually and brought in through the IN statement.

When a number of elements are specified on an IN statement, the order of specification does not have to correspond to the order in which the elements will be encountered during the input process. If the name only is supplied in the specification of an element, no check is made on the version although a version may be present. If the name and version are both supplied, they will both be checked.

- Format:

The IN statement has the following general form:

#INb̸optionsb̸file code, name/version,...

- Options:

C The elements which will be transferred to the job library are contained in the primary input stream following the IN statement. If this option is used, the file code field should be omitted from the statement.

NOTE: A source element in the primary input stream may not be entered through a # IN statement. The #SOURCE statement must be used for this purpose.

R If the input medium is tape, the tape is rewound prior to the location and input of the elements specified. When the input medium is a direct access storage file, a read control word is maintained within the initial header item of the file. The control word is updated upon completion of an IN function. The control word is accessed at the beginning of each IN function to determine the next logical address to be read. The R option can be used to initiate reading from logical address O rather than at the address indicated in the control word.

L The relative binary elements in the external media are to be included in the library, and other elements are to be excluded.

M Only the absolute or load elements are to be included in the library.

N Only the source element(s) are to be included in the library.

P Print the name, version, and type of each element transferred to the job library.

S Skip one tape file before locating required elements.

X The current task and the remainder of the job will be aborted if errors occur during the checking and transferring of the specified elements. This option should be used when tasks following the IN statement are dependent upon a correct copy of the elements being available.

Y The elements specified will be transferred although some noncritical error conditions are encountered during the operation. The option implies that such errors are known to the user and that the elements will be transferred as error free. The option will be overridden if a critical error occurs.

Z The current task will be aborted if an error condition occurs. Processing will continue with the next task execution control statement in the primary input stream. This option should be used when tasks following the IN function are not dependent upon the output of the function.

The absence of X, Y, and Z options implies that processing will continue.

- Specifications:

File code is used only when the origin of the input is other than the primary input stream (C option), and should be omitted when the C option is used. When present, the file code is alphabetic, A–YZ, and is the means by which the input medium may be referenced. The assignment for the peripheral used must be made prior to entering the IN statement.

Name/Version specifies the name and version, if any, of the element which will be transferred to the job library. The name may comprise up to ten alphanumeric characters; the version may comprise up to five alphanumeric characters. The version, if used, must be separated from the name by a slash (/). The identifications for individual elements are separated by commas. A maximum of 66 decimal elements may be specified on one IN statement with continuation images.

If the name/version field is omitted, all elements encountered within the input are to be established in the job library.

- Examples:

The following illustrates the format of the IN statement and the use of the different fields within the statement.

```
#INƀCƀNAME1/VER1,NAME2
(Element)
#END
```

The above statement would result in the data for the two elements, NAME1/VER1 and NAME2, being obtained from the primary input stream immediately following the IN statement and both being transferred to the job library.

```
#ASGƀƀTAPE, D
#INƀRƀD,NAME1/VER1, NAME2, NAME3
#END
```

This statement causes the element NAME1/VER, NAME2, and NAME3 to be obtained from the first file on the tape assigned to file code D, and establishes them in the job library.

```
#ASGƀƀUN8C,D
#INƀRƀD
#END
```

This statement would cause all elements contained in the first file on the peripheral unit defined by file code D to be brought in and established in the job library.

4.7.2. OUT Statement

The OUT control statement is used to transfer elements from within the job, group, and system libraries, to some external media. The elements transferred may be source, RB, or load.

When a series of elements is specified on an OUT statement, the elements are put in the order in which they are located in the library. In all cases the job library is constructed on a last in, first out basis.

- Format:

The OUT statement has the following general form:

```
#OUTƀoptionsƀfile code, library/identity,name/version, etc.
```

- Options:

A Perform FREL$ function for any unused direct access storage (used only if output medium is direct access storage).

C The elements specified on the OUT statement will be submitted to the secondary output stream defined for the job.

F An end-of-file tape mark will be written on the output file after the last element is transferred to the output file.

L The OUT operation will buffer out only relocatable binary elements.

M Only load elements will be included in the output.

N Only source elements will be included on the specified output medium.

P Print the name, version and type of each element transferred from the library system.

R If the output medium is tape, the tape will be rewound prior to processing. This option must also be used on the initial #OUT function to a direct access storage file to ensure proper initialization of control words used by the processors for the #OUT and #IN functions. The control words are maintained within the initial header item of the file. When the R option is not specified, the #OUT control word is read to determine the next available location for writing on the file. At the completion of an #OUT function, the control word is updated so that the word will reflect the current end-of-data condition.

S A system or group library format will be generated by inclusion of elements from the given library.

X The current task and remainder of the job will be aborted if errors occur during the checking and transferral of the specified elements. The option should be used when tasks following the OUT statement are dependent upon a correct copy of the elements being available.

Y The elements specified on the OUT statement will be transferred even though some error condition is encountered during the operation. The presence of the Y option implies that the error is known to the user and represents a noncritical error condition. If critical errors occur, the option will be overridden.

Z The remainder of the task will be aborted if an error condition is encountered. Processing will continue with the next task execution control statement in the primary input stream. This option should be used when tasks following the OUT function are not dependent upon the output of the function.

The absence of X, Y, and Z options implies that processing will continue.

- Specifications:

Library indicates the location in the library file where the element which will be transferred may be accessed:

SYS The elements specified are located in the system library.

JOB The elements specified are located in the job library.

xxxxx Group library number, which may be four decimal or five octal digits, specifying the file number by which the group library is identified. Decimal numbers may range from 0 to 9999, and must be indicated by a D following, e.g., 9989D. Octal numbers may range from 0 to 77777.

Identity specifies the group library file number which will be assigned to the output element(s), and is used only with the S option. The number has the same format as the input group library number.

File code is used only in the absence of a C option. When present, the field contains the alphabetic file code defining the output media to which the elements are to be transferred.

Name/Version specifies the name and version, if any, of each element which is to be transferred from the library. The name may comprise up to ten alphanumeric characters and the version may comprise up to five characters. The version, if present, is separated from the name by a slash (/). The identifications for individual elements are separated by commas. A maximum of 66 decimal elements may be specified on any one OUT statement with continuation images. If the name/version field is omitted, all elements encountered within the specified library will be put out on the defined media.

- Examples:

The following illustrate the use of the OUT statement.

```
#OUTƀCƀJOB, NAME1/VER1,NAME2

#END
```

This statement would cause the elements, NAME1/VER1 and NAME2, to be located in the job library and to be submitted to the secondary output stream in a card image format.

NOTE: the file code is not used with the C option.

```
#OUTƀCƀSYS, NAME

#END
```

The statement would cause the element, NAME, to be located in the system library and to be submitted to the secondary output stream in a card image format.

```
#ASGƀRƀTAPE, D

#OUTƀƀD,JOB,NAME,NAME1

#END
```

The statement would cause the elements, NAME and NAME1, to be located in the job library and to be put out on the file defined by file code D.

```
#ASGƀƀTAPE,D

#OUTƀƀD, JOB

#END
```

The statement would cause all elements within the job library to be put out onto the file defined by file code D.

```
#ASGƀƀTAPE,D

#OUTƀSƀD,JOB/1, NAME,NAME1

#END
```

The statement would cause the named elements in the job library to be synthesized into a group library input file with the identity of 00001.

4.7.3. PRT Statement

The PRT control statement provides a printed summary as description of elements within the library complex. The print images formed are submitted to the primary output stream defined for the job:

- Format:

  The PRT statement has the following general form:

  #PRTƀoptionsƀlibrary, name/version, etc.

- Options:

  I Print sequential statement number that may be used for element correction through #source/function.

  L Process relocatable binary (RB) elements only.

  M Process load elements only.

  N Process source elements only.

  O Print only element name(s) and version(s).

  T Only the table of contents (TOC) of the elements specified will be printed. If a T option is not used all pertinent data for the elements concerned will be printed. The pertinent data printed in the absence of a T option is as follows:

  RB Element: TOC and preamble will be printed.

  Load Element: TOC and control statements will be printed.

  Source Element: TOC and source image will be printed.

  X The task and the remainder of the job will be aborted if errors occur during processing of the specified element.

  Y The element(s) specified on the PRT statement will be processed even though some error condition is encountered during the operation. The presence of the Y option implies that the error is known to the user and represents a noncritical condition. If critical errors occur, the option will be overridden.

  Z The remainder of the task will be aborted if an error condition is encountered. Processing will continue with the next task execution control statement in the primary input stream.

  The absence of X, Y, and Z options implies that processing will continue.

- Specifications:

  Library indicates the library file upon which the elements scheduled for processing can be located:

  SYS The elements specified are located in the system library.

  JOB The elements specified are located in the job library.

  xxxxx Group library number, which may be four decimal or five octal digits, specifying the file number by which the group library is identified. Decimal numbers may range from 0 to 9999, and must be indicated by a D following, e.g., 9989D. Octal numbers may range from 0 to 77777.

Name/Version specifies the name and version, if any, of each element which is to be printed. The name may comprise up to ten alphanumeric characters, and version may comprise up to five characters. The version, if present, is separated from the name by a slash (/). The identification for individual elements are separated by commas. A maximum of 66 decimal elements may be specified on any one PRT statement with continuation images. If the name/version field is omitted, all elements encountered within the specified library are to be printed.

- Examples:

The following illustrate the use of the PRT statement:

```
#PRTƀTƀJOB,NAME,NAME2

#END
```

The statement will cause the information contained within the table of contents (TOC) of the elements, NAME and NAME2, specified as being in the job library, to be submitted to the primary output stream.

```
#PRTƀƀJOB

#END
```

The statement will cause all elements within the job library to be printed.

```
#PRTƀƀ198D,NAME0,NAME1

#END
```

The statement will cause the elements NAME0 and NAME1, located in the group library, 198D, to be printed.

4.7.4. DEL Statement

The DEL statement is used for purging nonuseful routines from the job library. Since a job library remains intact through the execution of all tasks within a job, removal of unnecessary elements will free direct access storage for the addition of more elements as desired. Deletions of this type can be made only from the job library, since a group or system library may have more than one user, and a routine which may be unnecessary for one job may be necessary to another.

The DEL statement may also be utilized by a current user to disable his access to specific group libraries which were established for him through the LINK statement.

- Format:

The DEL statement has the following general form:

```
#DELƀoptionsƀelement or file specification, etc.
```

- Options:

G Group libraries will be freed and, for this reason, only library numbers may be specified on the control statement.

L Process relocatable binary (RB) elements only.

M Process load elements only.

N Process source elements only.

X The task and the remainder of the job will be aborted if errors occur during the deletion of the specified elements. This option should be used when tasks following DEL statement are dependent upon a correct copy of the elements being available.

Y The elements specified on the DEL statement will be deleted even though some error condition was encountered during the operation. The presence of the Y option implies that the error is known to the user and represents a noncritical error condition. If critical errors occur, the option will be overridden.

Z The remainder of the task will be aborted if an error condition is encountered. Processing will continue with the next task execution control statement in the primary input stream. This option should be used when tasks following the DEL function are not dependent upon the output of the function.

The absence of X, Y, and Z options implies that processing will continue.

- Specifications:

Element or file specification designates the name and version, if any, of each element which will be deleted from the job library or designates the group library file which will be freed. The name may comprise up to ten alphanumeric characters and the version may comprise up to five characters. The version, if present, is separated from the name by a slash (/). The identification for individual elements is separated by commas. A maximum of 66_{10} elements may be specified on any one DEL statement with continuation images.

If the specification field is omitted, all elements encountered within the job library are to be deleted.

- Examples:

The following illustrate the use and format of the DEL statement.

#DELƀƀNAME, NAME1

The statement causes the elements, NAME and NAME1, to be deleted from the job library. The direct access storage used by the elements is released.

#DELƀƀ

The statement causes all elements within the job library to be purged.

#DELƀGƀ1298D, 012

The statement causes group libraries 1298D and 012 to be freed. Note that the G option is used in the statement.

4.7.5. LINK Statement

The LINK statement is used to establish the linkage from a job to a specified group library through a file code given on the statement. If the group library does not currently reside within the system, LINK has the ability to establish a group library from the input media specified.

All group libraries are cataloged with the Master File Directory (see 3.4.7) enabling the library to have multiple users and to transcend jobs within the system. Group libraries have file numbers corresponding to the identifier provided on the LINK statement.

The input media from which a group library is established is normally a tape which contains element(s) desired in the library. This input is produced by the OUT function with an S option. The identity of the group library which will be established, is assigned at this time by the specification of the identity (file number) on the OUT statement (see 4.7.2). The corresponding identity must be used on the LINK statement referring to that particular library.

- Format:

  The LINK statement has the following general form:

  #LINKƀoptionsƀidentity/library p-name,library file code, input p-name, input file code, input file name

- Options:

  A The input media from which the group library is to be established has been assigned prior to the initiation of LINK. If the group library has been established, no internal assignment will be issued by LINK.

  R If the group library is to be established from the input media, a rewind function will be issued by LINK prior to any attempt to read the data.

  X Abort the job if an error is encountered during the LINK process.

  Additional Assignment Options:

  Any options exclusive of the above options will be utilized on the internal assignment for the input media necessary to establish the group library. These options should include density specifications, numbered or unnumbered blocks, etc.

- Specifications:

  Identity is the number by which the group library is identified, and may consist of four decimal or five octal digits, or less, as applicable over the range, 0 – 9999_{10} or 0 – 77777_8. A decimal number should be followed by the letter D.

  Library p-name is the peripheral name which is to be used in the assignment of the direct access storage to contain the group library. If the field is unfilled, a default value of RAN will be used.

  Library file code is the one- or two-character file code, in the range A – YZ, to which the group library is linked. The file code is associated with specific areas of direct access storage, and is initialized by the LINK statement for assigning direct access storage or for cataloging the library through the Master File Directory as a temporary file. Service routines will use the file code to reference a specific group library.

  Input name represents the peripheral name, corresponding to the peripheral name field on an ASG statement, which will be used in the internal assignment for the input media.

  Input file code represents the one- or two-character alphabetic file code, in the range A–YZ, to which the input media is already assigned.

  Input file name represents the ID which will appear on the operators console in relation to the input tape when used with the W option.

- Examples:

  (1) #JOBƀƀID/NAME, 12345, C,C,C/C

  (2) #ASGƀUMƀUN8C, D

```
(3)   #INƀRƀD, TEST, TEST1            △INPUT ELTS

(4)   #END

(5)   #ASGƀJUMƀUN8C,A

(6)   #OUTƀRSFƀA,JOB/1, TEST,TEST1
       △CREATE GROUP LIB INPUT FILE

(7)   #END

(8)   #LINKƀRAUMƀ1,B,UN8C,A

(9)   #END

(10)  #GOƀƀTEST,1

(11)  #END

(12)  #FIN
```

NOTE: If the J option were not present on the line 5, the LINK function would request a tape unit for the group library input file.

4.7.6. Error Messages

The following messages describe the error conditions which may be encountered during the processing of the library editor functions. The action taken is dependent upon the nature of the error and the options in the control statement.

NO ENTRY IN LIBRARY

No elements are present in a library. If an X option is present, the job will be aborted.

COULD NOT LOCATE "Name"

An element named on the control statement is not within the library. The name of the control statement is not within the library. The name of the offending element is printed with the message.

#OUT△WRITE ERROR "cc"

An unrecoverable tape write error has occurred. The options are checked, the error message is submitted with the appropriate error code (see end of this subsection) and the applicable error exit is executed.

DRUM ERROR "cc"

An unrecoverable drum error has occurred. The options are checked, the error message is submitted to the primary output stream, and the appropriate error exit is executed.

FIELD EXCEEDS 10 CHARACTERS

A name field on a control statement exceeds the maximum of 10 characters.

DATA INCONSISTENT WITH TOC "Name"

An element exceeds the data described by the element TOC. The message is submitted to the primary output stream with the associated data.

SEQUENCE ERROR "Name"

An image is received for an element, contained on cards, for which the four-character sequence number is not in the correct sequence. The error option is checked and the appropriate action is taken. The name of the incorrect element is printed with the message.

READ ERROR "cc"

An error status is returned from a tape input. Depending upon the option present, the appropriate error exit will be taken.

INADEQUATE MASS STORAGE

An unsuccessful attempt was made to expand the job library direct access storage area while processing the #IN statement.

CARD ERROR "cc"

A primary input error has been encountered. The job will be aborted if the X option is present.

"cc"Codes

The following codes are pertinent for one or more of the error messages which involve status codes.

41 or 01	Inappropriate function
42 or 02	Incorrect parameter
43 or 03	Unrecoverable error
44 or 04	End-of-file
45 or 05	End-of-stream or end-of-tape
50 or 10	No assignment made for the referenced file code
51 or 11	Interlock
52 or 12	Block numbers not present when numbering has been specified.

4.8. INTERLANGUAGE COMPATIBILITY

Interlanguage compatibility is defined as the ability to mix, at object time, subroutines which may have been written in any set of languages. For the UNIVAC 494 Real-Time System, the languages currently available are the COBOL and FORTRAN IV compilers and the 494 SPURT and 494 ASM assemblers. These have, of course, many differences in terminology and functions, these differences being passed on into the generated subroutines.

Basic compatibility between object subroutines is provided by the design of the executive system. The OMEGA Loader (collector) routines are capable of integrating any routines or subroutines in the various libraries as long as the library elements are in relative binary form. Since all of the languages produce the requisite RB elements, the basic problem of incompatibility is solved automatically.

Using a called program in one language with a calling program written in another, usually requires the transfer of data from one routine to the other. When transferring data between assembly language routines, the programmer has complete and direct control over the methods and conventions employed. The programmer can specify a calling sequence between the routines, or he can allocate memory in any manner which he desires and make reference to items directly, using his knowledge of the correct locations. In the case of compiler generated subprograms, the programmer has only an indirect control over the machine instructions generated. Therefore, the programmer must be aware of the conventions followed by the compilers if he is to transmit and receive data successfully.

4.8.1. Common Area

A common area, by definition, is an area allocated independently of any particular subprogram, but which is capable of being referenced by all executable subprograms within the collected program. There are two types of common areas or blocks: labeled common and blank common.

A labeled common block has a name or label of its own, comprising up to ten alphanumeric characters, and has a group number of 0. The group number is specified in the INFO item supplied by the language processor to OMEGA in the preamble of an RB element.

A blank common block has a label consisting of ten blank characters and has a group number of 1.

The programmer can specify either labeled or blank common blocks in all languages except COBOL. In COBOL, the programmer is given a COMMON STORAGE statement which can be used in place of, or in addition to, WORKING STORAGE. If COMMON STORAGE is used, the COBOL processor will generate a blank common area.

4.8.2. Data Types

The various languages of the UNIVAC 494 Real Time System provide several types of data which are named differently for each language. The names and formats are given in Figure 4–1. It is the responsibility of the programmer to assure that any data from more than one subroutine has been properly described in each subroutine so that the data will be correctly handled. For example, if the FORTRAN IV square root function, SQRT(X), is employed by a SPURT program, the programmer must make sure that X is a floating point number. If X is a binary integer, an error will result.

ASM	SPURT	COBOL	FORTRAN	U494 MACHINE REPRESENTATION
LITERAL	LITERAL	DISPLAY (NUMERIC, ALPHABETIC)	HOLLERITH	6-BIT FIELDATA
SINGLE PRECISION	INTEGER CONSTANTS	*COMPUTATIONAL (single word)	INTEGER	30-BIT BINARY WORDS
DOUBLE LENGTH INTEGERS	DOUBLE PRECISION INTEGERS	*COMPUTATIONAL (two word)		2 EACH 30-BIT BINARY WORDS
**	**	*COMPUTATIONAL (three word)		3 EACH 30-BIT BINARY WORDS
DOUBLE LENGTH INTERNAL DECIMAL	DOUBLE LENGTH INTERNAL DECIMAL			2 EACH INTERNAL DECIMAL WORD FORMAT
FLOATING POINT	FLOATING POINT		REAL OR DOUBLE PRECISION	2 EACH 30-BIT BINARY WORDS FLOATING POINT FORMAT 29 28–18 17–0 29–0 S | CHAR | MAN | TISSA

* A COMPUTATIONAL item in COBOL may be stored in one, two or three words depending upon the SIZE. An integer item in FORTRAN is only a one-word item. Compatibility is attainable only when the COMPUTATIONAL item does not exceed one word.

** These can be defined and used by the programmer within his own coding.

Figure 4–1. Data Word Types

4.8.3. Transfer of Data

At present, several methods are available for data transfer between subprograms. In addition to these, the programmer employing a routine generated by an assembler, ASM, or SPURT, can devise his own techniques.

In the following discussion, a calling subprogram is considered to be the subprogram which jumps to a called subprogram to accomplish some specific function. The calling subprogram may or may not supply data (arguments) to the called subprogram; the called subprogram may or may not return data to the calling subprogram.

- Method 1. Common Area

  The calling subprogram stores all arguments in a common area in a preplanned sequence, and then executes a jump to the called subprogram. In turn, the called subprogram supplies all results to the calling subprogram, also in a common area.

- Method 2. Use of $AT

The $AT routine was created for the FORTRAN library. The function of this routine is to physically move the addresses of arguments from a calling subprogram to a called subprogram. The technique is illustrated by the following code:

Calling Subprogram

```
RJP*subroutine
NO-OP*Argument 1 address
NO-OP*Argument 2 address
ENTRY
ENT*A*Number of Arguments (in this case, 2)
RJP*$AT
NO-OP*0
NO-OP*0
```

$AT then moves the argument addresses from the calling subprogram to the called subprogram.

- Method 3. SEND$, RECEIVE$

The SEND/RECEIVE operators are available in OMEGA for sending parameters from one routine, SEND$, and receiving the parameters in another routine, RECEIVE$. This method is used mostly for information transfer between independent tasks, but can also be used between subprograms within a specific task. The SEND/RECEIVE operators are utilized to save blank common area between tasks. However, since executive logic is employed, this method is slower than the other methods available and is not recommended.

4.8.4. Interlanguage Transfer Techniques

- Assembler-Generated Routine to/from Assembler-Generated Routine

Any of the methods shown in the preceding paragraphs, or any method devised by the programmer, may be used.

- COBOL to/from Assembler

COBOL generates only a blank common block. The programmer can effect an RJP* subroutine in a COBOL program by using the ENTER verb (e.g., ENTER SQUARE). No arguments can be specified in the ENTER statement; therefore, both the COBOL elements and the assembler elements must reference blank common for sending or receiving data. To pass addresses of COBOL data-names to the Assembler routine, ENTER routine-name, WITH data-name can be used.

- FORTRAN to Assembler Routine

The following routines are used to transfer information from FORTRAN:

  – Method 1.

  A FORTRAN program can specify blank or labeled common. The programmer can utilize the same common blanks in the assembler element.

– Method 2.

The FORTRAN statements CALL S or CALL S (a,a ,...), where S is a subroutine name, will generate an RJP*S followed by the argument list as shown under Method 2. A 'function' call in FORTRAN does the same thing. The receiving program can employ either $AT to accept the arguments or be written to accept them in some other manner.

– Method 3.

Combination of Methods 1 and 2.

■ Assembler Routine to FORTRAN Routine

The following routines are used to transfer information from FORTRAN:

– Method 1.

A FORTRAN program can specify blank or labeled common. The programmer can utilize the same common blocks in the assembler element.

– Method 2.

A called FORTRAN subprogram, which expects an argument list, will always go to $AT. The assembler routine must have a calling sequence as shown under Method 2 in FORTRAN to Assembler routine described above.

– Method 3.

Combination of Method 1 and 2.

■ COBOL to/from FORTRAN

The data for the COBOL and FORTRAN programs must be equivalently ordered in each program so that the code generated for each is correct. Since COBOL can neither send nor receive data through an argument list, all communication between programs is carried out through the blank common block in which the data must be equivalently arranged. The COBOL statement, ENTER S, and the FORTRAN statement, CALL S, are used to generate the correct linkages.

5. OPERATOR COMMUNICATIONS

5.1. GENERAL

OMEGA has been designed to minimize the extent to which the computer operator must communicate with the system. However, it is recognized that certain operational information must be made known to the operator, while another class of information must be made known to the operating task by the computer operator.

The console operator device may be used by both OMEGA and by operating activities either to inform the computer operator of an event or to solicit a reply from the operator as to the completion of a requested action.

5.2. THE MSG STATEMENT

The standard method of communicating with the computer operator is through the use of the MSG statement. An option may be specified to allow the computer operator to respond to the message. This statement can be given either internally by an operating program or externally by the primary input stream. When given externally, the statement will be processed upon its occurrence in the stream. That is, the statement will not be related by the system to control statements preceding or following the statement.

- External format:

  The MSG statement has the following general form:

  #MSGƀoptionsƀliteral

- Options:

  R Assign response number and wait for operator response. When this option is used, the requesting activity will be suspended until the computer operator originates an input message preceded by the assigned response number. The absence of the R option will cause the message to be queued for output and the requesting activity will be reactivated without any acknowledgment on the part of the computer operator.

  Literal is the text of the message to the computer operator. The format of the text will vary, depending upon whether the system is using the standard operators' display console (cathode ray tube/PAGEWRITER printer) or the operator's alternate console (teletypewriter/printer), as follows:

Operators' display console – The text is limited to 174_{10} characters for a message which does not require a response, or 169_{10} characters for a message which does require a response. LINE formatting will be supplied by the operating system, although the user may format a message if desired, providing that the following conventions are adhered to:

– The text portion may contain a maximum of two new line codes (octal 04), thus defining a maximum of three lines.

– The first line is limited to a maximum of 49_{10} characters if the message requires a response, or 54_{10} characters if the message does not require a response. The remaining lines may contain up to 60_{10} characters each.

Operators' alternate console: The literal is limited to a maximum of 132_{10} characters. Trailing space codes are deleted from the literal before the literal is printed.

When control is returned to the requester the A register contains a status code. If the option specified is not correct, or if the literal is not in the correct format, the message will be rejected and the A register will contain the incorrect parameter status code (4200000000). If the message has been accepted, the status will indicate successful completion (00).

The message printed in response to the MSG operator has the following format:

x: literal

When the R option has been specified, the format is:

*z: x: literal

where: z is the response number assigned

x is the job number of the requester (three digit maximum)

When the R option has been specified, the computer operator is required to respond before control is returned to the requester. The additional information from operator response will be returned in the Q register. The maximum number of characters returned will be five and they will be left justified, Fieldata characters. If less than five characters are entered, the remaining character positions of the Q register will contain master spaces (00). The format for the computer operator response may be found in Appendix B.

5.3. MAGNETIC TAPE STATEMENTS

This class of messages directs the computer operator to perform some activity at the specified magnetic tape unit. Each message has a fixed format, and operator response is always required. These messages direct the operator to mount, demount, and change certain tape files. The mnemonic operator and format for each message in this class is given below.

5.3.1. MOUNT$

The MOUNT$ request is used to direct the computer operator to mount a file on a particular unit. The MOUNT$ request has the following form:

MOUNT$ƀfile code, file identifier

File code is a one- or two-character alphabetic code which determines the physical unit. If the file code has not been assigned, the request will be rejected and an incorrect parameter status will be returned to the requester.

File identifier is an alphanumeric identification of the file to be mounted. The maximum length of this field is 15 characters. The file identifier is printed on the console as it appears in this field with all space codes deleted. The identifier may specify a class of tapes such as BLANK or SCRATCH.

The format of the printed message will be:

*z: MT xx...x

A normal operator response will initiate a further message of the form:

*z: fc c/u xx...x

whereas a negative operator response will initiate a message as follows:

*z: RDY fc c/u xx...x

where: z is the response number assigned

fc is a one or two-character file code.

c/u is the channel and unit specified by the file code.

xx...x is the file identifier

When the computer operator has mounted the appropriate file and responded with an input message, control is returned to the requester with the A register equal to 0 and the Q register containing the characters entered by the computer operator (maximum of five).

5.3.2. DEMOUNT$

The DEMOUNT$ request informs the computer operator that a file is to be demounted and labeled. The DEMOUNT$ request has the following form:

DEMOUNT$b̸file code, file identifier.

The file code identifies the channel and unit, whereas the file identifier is the name to be placed on the file label. When the operator has properly labeled the file, the system must be informed by the response input. File code, file identifier and status code have the same format as in the MOUNT$ operator.

The output message format is:

*z: DMNT + LBL fc c/u xx...x

where: z is the response number (two digit maximum)

fc is a one or two character file code

c/u is the physical channel and unit location.

xx...x is the name to be placed on the file label.

5.3.3. CHANGE$

The CHANGE$ request provides a method for notifying the computer operator that a particular file is to be demounted, labeled, and an alternate file mounted in its place.

The CHANGE$ request is of the following form:

CHANGE$b̸file code, file identifier 1, file identifier 2

File code identifies the channel and unit.

File identifier 1 is the label to be placed on the demounted file.

File identifier 2 is the name of the new file to be placed on the unit.

When the computer operator has performed the actions requested, he informs the system by the response input. File code, file identifiers and status codes follow the same format as in the MOUNT$ operator.

The message format is:

*z: LBL/fc c/u/xx...x + MNT yy...y

where: z is the response number assigned (two digit maximum)

fc is the one or two character file code

c/u is the channel and unit

xx...x is the name to be placed on the label of the demounted file

yy...y is the label of the new file to be mounted.

5.4. UNSOLICITED TASK INPUT MESSAGES

Unsolicited task input messages are those entries for which no direct request has been made. This type of input will be accepted by the system only when the task referenced is currently operational and has made provision for such input. This input is intended primarily for continuous jobs such as real time/ communications jobs, although the input may be used by batch programs.

A task may allow up to forty input characters by specifying the input area through the UNSOL$ statement. This statement has the following format:

UNSOL$b̸label

The generated coding is:

ENT*B7	LABEL
EXRN	20122

Label defines the buffer area for the input message. The buffer area must be constructed as shown below.

Word 0	0	No. of Words (N)
1		
2		
.		
.		
.		
N		

The upper portion of word 0 is a control/indicator location. The lower portion of word 0 contains the number of words reserved for input. Words 1 through N are the input area. *No. of words* need not be greater than 8 since the maximum input message allowed is 40 characters.

The buffer must be completely within the primary storage limits of the task. If this is not the case, the *incorrect parameter status* is returned. If the buffer is within the primary storage limits, the *successful completion* status is returned.

The input characters will be Fieldata coded and packed left justified in the buffer. The maximum number of characters placed in the buffer will be determined by the *No. of words* specified or by the absolute maximum of 40 characters, whichever is less. When the number of characters entered is not a multiple of a computer word, the unused character positions of the last word will be cleared to master spaces (00). All characters within the text entered by the console operator, become part of the input message, with the following exceptions: carriage return (04), line feed (03), master space (00), and erase (77). The end of the input message may be determined from either the number of characters placed in the upper position of word 0 or by the position of the stop character (57) in the buffer.

The upper portion of word 0 is used as a flag to inform the task of the presence of an unsolicited message in the buffer. When this control location is found to be nonzero by the task, the buffer contains an input message. The nonzero value will be the number of characters entered. (The location is set to nonzero only when a complete message has been entered.) An unsolicited entry will be placed in the buffer only when the control location is 0. Thus, the task must clear the control location in order to allow a second unsolicited entry.

The method of operation, then, would be for a task to periodically check the control location for a nonzero condition. When the location is found to be nonzero, the task knows that a complete message is contained in the buffer. The message may then be processed and possibly acknowledged by the task.

When the task has finished with the message, the control location may be cleared to allow the entry of another unsolicited message.

6. TEST SYSTEM

6.1. GENERAL

The test system routine is a utility function designed to provide the programmer with a basic set of procedures to aid program development and testing.

Through use of control statements and symbol tables, the package interpretively executes programs placed in a test mode and provides a flexible means of dynamic control of test procedures. Symbol tables generated by assemblers are accepted to provide symbolic reference of test points and data areas. Relative reference is also provided for testing collected programs. Either means of reference allows definition of test procedures, external to the program being tested. Test procedures can, therefore, be employed at object time in contrast to source level, allowing the programmer to vary test strategies without costly recompilation and collection.

Test procedures fall into two general categories: conditional procedures, which regulate logical switches that are used to control activation and frequency of functional procedures; and functional procedures which are used to perform diagnostics. Functional procedures provided in the basic package are snapshot dumps of primary storage or peripheral areas; store statements used either for patches or to set test values; printing trace; trap; and exit used for early termination.

Programs initiated with the test system are interpretively executed until an end condition occurs. End conditions are:

(1) The program discontinues execution.

(2) The program attempts to store or jump to a memory location outside of its assigned limits.

(3) An indicated time limit on program execution is reached.

The test system will automatically dump operational registers and descriptive diagnostics.

6.2. CONDITIONAL PROCEDURES

The test system contains a set of 26 logical switches, A through Z, which may be set to on or off. A conditional procedure logically operates the state of a particular switch (see 6.5). The effect of successive conditional procedures is cumulative. The purpose of conditional procedures and the logical switches is to establish whether or not a functional procedure, contingent upon the state of the switches, will be executed. This allows the user to employ debugging techniques which are responsive to the dynamic execution of a program. Through use of multiple switches the programmer may nest procedures to provide additional flexibility.

6.3. TEST SYSTEM CALL

A call for the loading and activation of the test system, used to interpretively execute a program, is effected by the TEST control statement:

- Format:

  The TEST control statement is of the following form:

  #TESTƀoptionsƀname/version, library

- Options:

  A–E Identify logical test switches maintained by the system for each job. The switches are specified on the JOB card as applicable throughout job. When the test switches are defined or are being reset with the TEST card, the R option must be used. These switches should not be confused with Test System switches used for conditional procedures.

  R Reset or clear logical switches as per A–E option letters.

- Specifications:

  Name/Version identifies the absolute element to be placed in test. If symbolic references are used in secondary control statements, a symbol table must be included with the element. (The use of the 'S' option, when assembling and collecting, will provide this.)

  Library identifies the library in which the absolute element is contained on direct access storage. Any one of the following specifications is valid:

  SYS The absolute element is contained in the system library.

  JOB The absolute element is contained in the user's job library.

  xxxxx Group library number, which may be four decimal or five octal digits, specifying the file number by which the group library is identified. Decimal numbers may range from 0 to 9999, and must be indicated by a D following, e.g. 9989D. Octal numbers may range from 0 to 77777.

  In the absence of a library specification, job library is assumed by the system.

6.4. SECONDARY CONTROL LANGUAGE

The secondary control language describes functions to be performed by the system and immediately follows the TEST call card. All secondary control statements are blank in column one for differentiation from primary control languages, and are of the following general format:

ƀFunctionƀoptionsƀspecification, specification,...

The statements follow essentially the same rules as those of the executive control language. Operands within the specification field may, in general, refer to labels and tags within the object program which are retained along with their relative location through use of the symbol table produced by the assemblers, or optionally provided by relative address. When operands reference symbol tables, the user may apply + or – increments to the symbol. Increments may be given in decimal or octal notation and are identified by the presence or absence of a trailing D.

LABEL + 30D is a decimal increment of 30

LABEL + 030 is an octal increment of 30

6.4.1. The AT Statement

The AT procedure maintains a counter to provide conditionality for a switch which is usetul in controlling iterations of a loop or program. Through use of the AT statement, a logical switch may be set every nth iteration to a predetermined maximum number of times. No options are defined for this statement.

- Format:

The AT statement has the following general form:

ƀATƀƀP/switch$_1$, v_0/v_1,switch$_2$

- Specifications:

P/switch$_1$ is the program location, and conditional indicator if any, at which the AT procedure will be performed. P may be expressed as a symbol, + or – an increment or address relative to the RIR value of the program. Switch$_1$ is an optional operand used for nested procedures, and when given, the AT procedure will be executed only when the indicated switch is set.

v_0 is the value which the counter must attain before a logical switch is set.

v_1 is the maximum number of times that the switch will be turned on before the procedure is to be ignored. Lack of specification implies no maximum.

Switch$_2$ indicates the logical switch A through Z which is to be turned on if the counter attains v_0.

At load time a counter is initialized to 0 and is incremented by 1 each time instruction P is executed and switch$_1$ is on. Upon incrementation, the counter is compared with v_0; if equal, the switch is turned on, the counter is reset to 0, and v_1 is decremented. If v_1 is 0, the AT procedure will be bypassed on subsequent execution of P and the indicated logical switch will be turned off. If, upon comparing the count with v_0, an unequal condition exists, the indicated switch is turned off.

Example of AT statement:

ƀATƀƀLOOP1+6/B,10/20,A

6.4.2. The IF Statement

The IF procedure is used to set an indicated switch on the basis of one or more relational expressions that are joined to form a logical statement. The complete statement must be true before the switch is turned on. The IF statement may be continued on subsequent cards by placing a # sign after the previous operator (see coded examples at the end of this section). No options are defined for this statement.

- Format:

The IF statement has the following general form:

ƀIFƀƀP/switch$_1$,logical statement, switch$_2$

- Specifications:

P/switch$_1$ is the program location and conditional indicator if any, at which the IF procedure will be performed. P may be expressed as a symbol + or - an increment or address relative to the RIR of the program. Switch$_1$ is an optional operand used for nested procedures, and when given, the IF procedure will be executed only when the indicated switch is set.

Logical statement is composed of one or more relational expressions strung together by OR or AND conditions which, evaluated from left to right, determines the true or false state of the total expression. If true, the indicated switch will be set to on; if false, the switch will be set to off. The general form of the logical statement is as follows, where r_i are relational expressions.

r_1 *OR *r_2 *OR *r_3 *AND *etc.

where r_i are separated from the OR or AND operators by *.

Each r_i is of the format:

x_i/rel/y_i

The rel field specifies the relation between the two operands to be tested. Allowable codes for this field are:

Code	Meaning
GT	Greater than
LT	Less than
EQ	Equal to
GE	Greater than or equal to
LE	Less than or equal to
NE	Not equal

The x_i, y_i, operands may specify the content of the upper or lower portion, or the whole of a 30-bit word, or a constant given externally for comparison purposes. Specification of internal primary storage locations is expressed as follows:

k(address)

where k-designator may be U, L, or W to specify the upper or lower portion of, or the whole of, a primary storage location. Address can be expressed as a tag or label + or - an increment, or as a relative address. Comparative constants are expressed as a numeric value without k-designator.

Switch$_2$ indicates the logical switch from the set A through Z which is to be set to on if the logical statement is true or to off if the statement is untrue.

- Example of IF statement:

```
ƀIFƀƀSTART+10/B,L(XLIST+5)/EQ/L(YLIST +5)*OR*L(XLIST +5)/LT/6*OR*,A
```

The IF statement is to be performed at location START +10 provided that switch B is set to on. If either of the two relational expressions is true, switch A will be turned on. If neither is true, switch A will be turned off.

Each relational expression must be followed by its operator.

Continuation cards are indicated by placing a # sign after the previous operator. This is the only breaking point.

Example:

Card 1: ƀIFƀƀSTART + 10/B, r_1 * OR * r_2 OR * r_3 *AND* #

Card 2: ƀr_4 *AND*,B

6.4.3. The DUMP Statement

The DUMP statement describes data in primary storage or on peripheral files which will be formed into a printer image and submitted to primary output. The snapshot dump technique permits the programmer to extract data at various points during execution and under control of conditional procedures. The extracted data is edited into a readable format and submitted to the primary output stream for subsequent printing. Dumps of primary storage or peripheral files have edited format options for octal, decimal, Fieldata, or floating point values.

- Format:

The DUMP statement has the following general form:

ƀDUMPƀoptionsƀP/switch,v_0,v_1,v_2,v_3

- Options:

O,A,D,F Type of format for word conversion to:

O octal representation

A fieldata

D decimal representation

F decimal representation (from a double precision floating point number (two words))

C Space the printer to a new page before printing

L List the task control thread.

P v_0,v_1,v_2,v_3 describe a peripheral device, either direct access storage or tape, which will be dumped.

- Specifications:

P/Switch is the program location, and conditional indicator, if any, at which the dump will be performed during execution of the program. P may be expressed as a symbol + or - an increment or address relative to the RIR of the program. Switch is an optional operand which provides conditionality to the dump. When the switch is specified, the dump will be performed only if the switch is set.

v_0 is the file code; the field is blank if primary storage is requested.

v_1 is the number of blocks to be dumped or the number of words, if direct access storage or primary storage.

v_2 is the logical address if direct access storage;
the number of blocks to be skipped at the beginning of the tape;
the beginning address if primary storage:
the dump always starts at the preceding even ten location; for example, a call for 467 would include information beginning at 460.

$^{v}2$ cannot be a tag or label.

v_3 is the number of print lines to be spaced between print images. If the number is omitted, one line will be assumed.

Examples:

ƀDUMPƀDƀSTART+20/D, 500D, 1000,2

An octal primary storage dump will be performed at START+20, if switch D is set to on. 500_{10} words will be dumped starting at RIR+1000. Two lines will be spaced between print images.

ƀDUMPƀPAƀLOOP/K,R,100D,200

If file R is direct access storage, 200 words will be listed in Fieldata format, starting at logical increment 100_{10}. If file R is tape, the unit will be rewound, the first 100 blocks will be bypassed, the 200 blocks will be listed in Fieldata.

6.4.4. The TRACE Statement

The TRACE statement specifies areas of a program in which a printing trace is to be performed for each instruction executed within the bounds of the given areas. The display will show an instruction and all register settings at execution time.

- Format:

The TRACE statement is of the following form:

ƀTRACEƀoptionsƀswitch, start-of-area, end-of-area

- Options:

J Display only jump type instructions enclosed in the area.

- Specifications

Switch is optional, and is used to control the trace on the basis of the indicated logical switch being set to on.

Start-of-area and end-of-area program addresses delimit the area of the program in which the trace should be performed. Addresses may be expressed as symbols, + or - an increment, or as an address relative to the RIR. For a segmented program, addressing must be via symbols which must be within the same segment.

6.4.5. The TRAP Statement

The TRAP statement specifies a location which is to be displayed each time that it is accessed on the storage cycle of an instruction. The display will contain the register setting, the instruction responsible for alteration, and the contents before modification.

- Format:

The TRAP statement is of the following form:

ƀTRAPƀoptionsƀlocation/switch

- Options:

O,A,D,F Type of format:

O	Convert word to its 10 characters in octal representation.
A	Word is assumed to be five characters in Fieldata code.
D	Convert word to its decimal equivalent.
F	Convert a floating point number (two words) to its decimal representation.

- Specifications:

Location is a symbol + or - an increment or relative address of the cell to be monitored.

Switch is optional. It is used to control the printing of the trapped location.

NOTE: In both the printing trace and the trap operations, registers will be displayed only when the register contents change.

6.4.6. The EXIT Statement

The EXIT statement provides a mechanism for terminating a program before the program has completed its cycle. No options are defined for this statement.

- Format:

The EXIT statement is of the following form:

ƀEXITƀƀP/switch

- Specifications:

P/switch is the program location, and conditional indicator, if any, at which the exit will be performed. P may be expressed as a symbol + or - increment or address relative to the RIR of the program. Switch is an optional operand which provides conditionality to the exit mechanism; when this specification is given, termination will occur only if the indicated switch is set to on.

6.4.7. The SET Statement

The SET statement provides a method for entering dynamic or static values to change program execution. The statement may be used to introduce patched instructions or to establish values in variable storage cells to arrive at a particular test condition. No options are defined for this statement.

- Format:

The SET statement is of the following form:

ƀSETƀƀP/switch, store address, value, value,etc.

- Specifications:

  P/Switch is the program location and conditional indicator, if any, which is used to reset values within the program dynamically during execution. P is the program address at which point the indicated values are to be stored during execution of program. P may be expressed as a symbol, + or - an increment or an address relative to the RIR of the program. Switch is an optional operand which provides conditionality to the SET procedure, and when given, values will be stored only if the indicated switch is set.

  Store address specifies the starting primary storage location relative to the RIR in which the specified values are to be stored. Store address may be a symbol, + or - an increment or a relative address.

  Value is used to form a 30-bit octal value to be entered into primary storage at a specified location, and may be a symbol, + or - an increment, or an octal or decimal constant in any combination to indicate upper and lower word portions. Examples of values are as follows:

  LIST+020/LIST+021,40D/60D,61000/LOOP, 100D,

  The first entry requests that two addresses be formed and stored in the upper and lower portions of a word, respectively. The second requests that two decimal constants be stored in the upper and lower portions of a word. The third requests a jump to LOOP, and the fourth is a constant which will be entered right justified.

- Example:

  (Card 1) ƀSETƀƀSTART/B,DOG+2,CAT/CAT+10,6100D DOG,40D/27,100D,#

  (Card 2) ƀDOG+7,100/D,12000 00000

  When the operating system reaches START, and switch B is turned on, the six words will be stored, beginning at location DOG+2. Al! continuation cards have a #sign immediately following the last comma of the preceding card. The store address is always assumed to be the first field of each continuation card. The subsequent cards need not be stored in sequential locations.

- Application:

  In a routine where data is read from larger master files, it may be desired to test certain error procedures. To arrive at this test condition, enter a SET statement just after the data is read; the SET operation can then alter the data to arrive at the particular test condition. Thus, it may not be necessary to generate special data files to test the routine. The SET statement will be performed each time that a record is read. The operation may be turned on or off by the use of the logical switches. Any SET statement may place values only in the same segment which contains its P-value.

6.4.8. The END Statement

The END statement, which is nonoperational, is used to signal the end of secondary control language, and to activate the loading and execution of the program.

- Format:

  The format of the END statement is as follows:

  ƀEND

  No options or specifications are required.

6.4.9. Examples of Secondary Control Language Statements

The OMEGA Test System operates at the task level and does not affect the rest of the job. Several examples are presented to illustrate both the simple and more involved use of the OMEGA Test package. The simple case is presented first.

- Example 1:

In this form the OMEGA Test package requires very little effort to use. The illustration is of the user wishing to trace a nonsegmented routine:

```
#GOƀƀNAME/VERSION,LIBRARY
  ≀
Optional user parameters
  ≀
#END
```

To run this task under the OMEGA Test package, the user would substitute for, GOƀƀNAME/VERSION,LIBRARY, the card

```
#TESTƀƀNAME/VERSION,LIBRARY
```

and follow immediately with a ƀTRACE card. Then the ƀEND card follows, which is a nonoperational sentinel and serves only to signify the end of the TEST system cards. The sample would then appear as:

```
#TESTƀƀNAME/VERSION,LIBRARY

ƀTRACEƀƀ,0,1000

ƀEND
  ≀
Optional user parameters
  ≀
#END
```

When this task is selected by the OMEGA JOB input control mechanism, the #TEST card will cause the OMEGA Test System to be loaded from the system library. The Test System will load the NAME/VERSION,LIBRARY, and interpretive execution will begin. Each time that an instruction is performed within the relative program limits 0—1000, this instruction, together with all appropriate registers, will be submitted to the primary output stream.

A more complicated example would involve the use of several TRACE cards instead of one, thus obtaining a more selective trace. The package might look like the following:

```
#TESTƀƀNAME/VERSION,LIBRARY

ƀTRACEƀƀ,0,100

ƀTRACEƀƀ,1000,2000

ƀTRACEƀƀ,6000,10000
```

```
ƀEND
 ≀
Optional user parameters
 ≀
#END
```

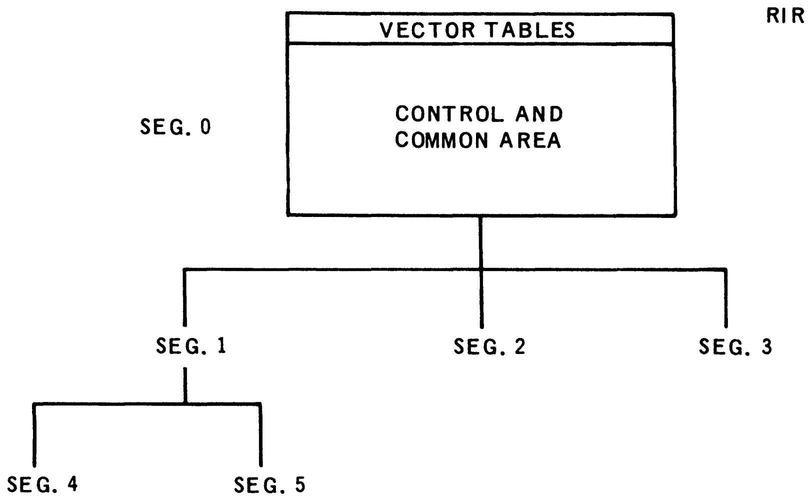
Under the Test System, the complete routine is monitored; however, only those instructions performed within the limits specified by the three TRACE cards will be submitted, along with the appropriate registers, to the primary output stream for subsequent printing. A further example of this type involves the combination of the trace snapshot dump and trap operation:

```
#TESTƀƀNAME/VERSION,LIBRARY


ƀTRACEƀƀ,0,1000

ƀDUMPƀƀ25/,,100,0,1

ƀDUMPƀƀ50/,,500,50,1

ƀTRAPƀƀ565

ƀTRAPƀƀ732

ƀTRACEƀƀ,2000,4000

ƀEND
 ≀
Optional user parameters
 ≀
#END

#DUMPƀOEƀ,50000,0,1

#FIN
```

In this example, the user is able to trace the control portion of his routine, receive dynamic snapshot dumps of his buffer areas and trap locations in his routine, all in one run. In addition, should the Test System find it necessary to terminate the task because of error, the OMEGA #DUMP card with the E option will give a complete dump of the primary storage assigned to the routine under test.

As the level of program sophistication increases, it becomes slightly more difficult to make efficient use of the Test System. In the case of segmented programs, the user may wish to receive some diagnostics in a segment other than the control segment. A typical segmentation scheme might look like the following:

- Example 2:



In this example, assume that the user wishes to trace a limited area of segment 3. In some test systems, it is necessary to trace the corresponding areas of segments 1 and 2 to obtain the trace of segment 3. The OMEGA Test System provides the means for specifying the segment number as well as the relative primary storage limits. In all segmented routines, the Test System always checks for the proper segment number before any TEST procedure statement will be performed. This requires that the user, when testing a segmented program specify the desired segment number. This is done through the Symbol Definition (SDEF) Tables. Every tag in the user's source code is a potential SDEF. The SDEF are established at essembly time by the S option. The assembler retains the tags and passes them on to the OMEGA Loader, which computes the relative addresses and proper segment numbers.

Example of an SDEF

Word	0	BALAN	
	1	CE---	
	2	Seg#	Relative Address

The SDEF tables are placed in the library after the absolute element and are available to the Test System. The Test System searches the SDEF for a specified tag starting with segment 0 and proceeding sequentially until the first such tag is found. This is assumed to be the proper tag.

NOTE: It is possible to have duplicate tags, as in cases where the elements are independently compiled. The first such tag encountered will always be the tag used by the Test System.

The following is an example in which the programmer wishes to trace a portion of the third segment starting with the tag BALANCE, illustrating the complete JOB input stream:

- Example 3:

```
          #JOBƀƀ494/EN,RSV002,C,4,50/100

          #MSGƀƀMOUNT TAPES 1067 AND 52

TASK 1    #SPURTƀSƀPAYMENT10A

          Source code for control segment
             ≀
          #END

TASK 2    #SPURTƀSƀPAY10B

          Source code for Segment 1
             ≀
          #END

TASK 3    #SPURTƀSƀPAY10C

          Source code for Segment 2
             ≀
          #END

TASK 4    #SPURTƀSƀPAY10D

          Source code for Segment 3
             ≀
          #END

TASK 5    #SPURTƀSƀPAY10E

          Source code for Segment 4
             ≀
          #END

TASK 6    #SPURTƀSƀPAY10F

          Source code for Segment 5
             ≀
          #END

TASK 7    #LOADƀSƀPAYMENT10A, PAY/10A

          ENTRY START

          SEGMENT ONE

          INCLUDE PAY10B
          SEGMENT TWO, ONE

          INCLUDE PAY10C

          SEGMENT THREE,ONE

          INCLUDE PAY10D

          SEGMENT FOUR,(ONE,TWO,THREE)

          INCLUDE PAY10E

          SEGMENT FIVE,FOUR

          INCLUDE PAY10F

          #END

          #ASGƀƀUN8C,D

          #ASGƀƀUN8C,E

          #COREƀƀ20000

TASK 8    #TESTƀƀPAY/10A,JOB

          TRACEƀƀ,BALANCE,BALANCE + 259D

          END

          Optional user parameters
            ≀
          #END

          #DUMPƀOEƀ, 10000,0,1

          #FIN
```

The Test System will perform a printing trace of all instructions executed within the limits of BALANCE and BALANCE + 259.

In this same routine the user might have wished to do more extensive checking. Starting with those cards associated with TASK 8, the following might be obtained:

```
#ASGƀƀUN8C,D

#ASGƀƀUN8D,E

#COREƀƀ20000

#TESTƀƀPAY 10A,JOB

ƀTRACEƀƀ,BALANCE,BALANCE + 259D
```

```
ƀTRAPƀFƀ,TSEG1

ƀTRAPƀDƀ,TSEG5

ƀDUMPƀOƀTSEG4/,,100,850D,1

ƀEND
```

6.5. LOGICAL SWITCHES

In the process of testing intricate routines, it is frequently desirable to turn a debugging procedure on and off intermittently, as in the case where the routine makes numerous iterations through a particular path. To provide this flexibility, the logical switches are introduced into the OMEGA Test System.

There are 26 logical switches defined for the Test System, A – Z. These switches are internal to the Test System and should not be confused with the five logical switches defined for OMEGA JOB control. The Test System switches are initially set to off when the task is activated and any conditional functional procedure will not be performed until that procedure's switch is turned on. All functional procedures become conditional procedures whenever the (optional) logical switch is specified. Example 1 is an unconditional dump:

- Example 1:

```
ƀDUMPƀOƀSTART/,,1000, 0,1
```

This DUMP statement will always be performed whenever control of the routine under TEST reaches relative location START.

Example 2 is the same DUMP statement made conditional by including the logical switch A.

- Example 2:

```
ƀDUMPƀOƀSTART/A,,1000, 0,1
```

This conditional procedure will be performed only when control of the routine under TEST reaches location START and logical switch A has been turned on. This same conditionality may be applied to all functional statements; that is, to the following:

DUMP

TRAP

TRACE

SET

EXIT

A means for operating the logical switches is use of the AT statement which enables the user to predetermine every nth iteration at which he wishes a functional statement to be performed. On every other iteration, the AT procedure will turn the logical switch off and the conditional functional procedure will not be performed. The following example uses the AT statement to control a conditional dump.

- Example 3:

```
#TESTƀƀNAME/VERSION,LIBRARY

ƀDUMPƀOƀSTART+10/A,,1000,2000,1

ƀATƀƀSTART+7/,10/3,A

ƀEND
```

When the control reaches START+7 on the tenth iteration, switch A will be turned on. At START+10, the conditional dump will be performed. On all other iterations switch A will be turned off. After the 30th iteration, the switch will not be turned on again, and no subsequent dumps will be performed.

In the next example, the user wishes to trace a particular path for the first three times through and then ignore the path on all subsequent passes:

- Example 4:

```
#TESTƀƀNAME/VERSION,LIBRARY

ƀTRACEƀƀ,READ,READ+200

ƀATƀƀREAD/,1/3,A

ƀEND
```

In checking out routines, it is sometimes desirable to allow the input data or dynamic values, internal to the routine being tested, to determine when or IF a debug procedure should be performed. The IF statement is introduced into the OMEGA Test System to provide this capability. The sole function of the IF procedure is to turn a logical switch on or off, depending upon its true or false value, and thus to indirectly control the performance of conditional functional procedures:

- Example 5:

```
#TEST ƀƀNAME/VERSION,LIBRARY

ƀIFƀƀSTART+20/,W(BALANCE)/LE/20*OR*W(BALANCE)/GE/1000D*OR*,A

ƀDUMPƀOƀSTART+22/A,,1000,200,1

ƀEXITƀƀSTART+23/A

ƀEND
```

The DUMP and EXIT statements are conditional to switch A, which will be turned on by the IF statement only when the value at BALANCE moves outside of the limits of 10 and 10000.

6.6. DIAGNOSTIC MESSAGES FROM TEST SYSTEM

All error diagnostics are submitted to the primary output stream:

- THE PREVIOUS CARD IS NOT A VALID CONTROL STATEMENT

  Reason: One of the secondary control cards is in error.

  Action: This card is bypassed and processing continues.

- NO VALID PARAMETER STATEMENTS WERE PROCESSED

  Reason: an END or #END card is detected before any valid secondary control cards have been processed.

  Action: The run is terminated.

- NAME/VERSION, LIBRARY ERROR

  Reason: The routine which is to be run under test cannot be found in the library.

  Action: The run is terminated.

- QREF ACTIVITY CANNOT BE FOUND

  Reason: An activity is referenced with an improper identity.

  Action: The run is terminated.

- NOT ENOUGH CORE ASSIGNED

  Reason: Insufficient primary storage was assigned for the Test System to load the routine, or there have been too many fragmentation requests, and all available primary storage has been used up.

  Action: The run is terminated. The user should increase the amount of primary storage assigned and resubmit the task.

- LOGIC ERROR IN IF STMT

  Reason: The IF statement cannot logically be processed, usually because a field is missing.

  Action: The IF statement is bypassed and processing continues.

- OUTSIDE OF PROGRAM LIMITS

  Reason: The routine attempts to access primary storage which is outside of the routine's limits.

  Action: The run is error terminated. The incorrect instruction is listed together with the contents of all registers.

- ILLEGAL OP CODE

  Reason: The routine attempts to perform an illegal operation.

  Action: The run is error terminated. The incorrect instruction is listed together with the contents of all registers.

6.7. REQUIREMENTS AND METHOD OF OPERATION

The Test System permits the user to specify optional amounts of primary storage and primary output, to simulate certain service requests, and to display selective areas of the program.

6.7.1. Requirements

The Test System requires about 10,000 octal words of primary storage and primary output. Any additional primary storage needed by the routine under test must be assigned with an #ASG card preceding the #TEST card or a #CORE card must be in the element.

The Test System occupies the same block of primary storage as the program under test and has the same RIR and PLR settings. There are two separate routines in the test package. The first is a small routine which determines the amount of primary storage assigned and floats the processing routine to the upper limit of this primary storage. The second routine then processes the secondary control cards, loads, and interpretively executes the program to be tested.

6.7.2. Service Requests

OMEGA service requests are recognized by the Test System and, where possible, are passed on to OMEGA. Control is then picked up when the service request is completed. Certain types of service requests are changed or simulated by the Test System.

In general, these alterations are hidden from the routine under test and the results are the same as they would be if the Test System were not present.

Activity registration of any kind is simulated by the Test System. When a fragmentation request is detected by the Test System, a storage module is built and control is returned to the activity making the request. Control is then rotated when a service request is made. Timeout may also cause rotation after approximately 2000 worker instructions have been performed.

6.7.3. Segment Control

The Test System is designed to permit the user to be very selective of the areas when he wishes displayed. This is accomplished by any or all of the following:

- Logical switches determined by the user, by the AT statement, or by control of the program under test, with the IF statement.

- Primary storage limits which are defined by the secondary control statements.

- Segmentation, wherein the user may specify any particular segment by referencing a unique tag (or the first such tag encountered) in that segment by the SDEF's included with the collection.

The Test System monitors the loading and activation of all segments. The number of the current operating segment is maintained at all times. All action points specified by the secondary control statements are checked for, including primary storage limits, appropriate switches, and the proper segment number. All three conditions must be satisifed before any action will be taken. Since all segment loading is done with information contained in the vector tables, the Test System considers the tables to be out of bounds for the program under test. Any attempt by the user to access these tables, for reasons other than segment loads, is considered an error and the run will be terminated. Free access between the control segment and any current loaded segment is permitted, although these cross references do not go through the vector tables. This includes the transfer of controls between any loaded sgments (there may be more than one, as where two or more nonoverlapping segments are in primary storage at one time). The primary storage area of each segment is mapped, and when control moves from any segment to another nonoverlapping segment, the Test System switches segment numbers. Any other method of switching from one segment to another may cause logic problems in the test package. For example, the user knows that a newly loaded segment overlays only part of the previous segment, and tries to take advantage of some of his coding in the previous segment, thereby causing a logic error.

6.8. SAMPLE PROBLEM

This problem is intended only to illustrate the proper use of the Test System and has no other function.

- Statement of Problem:

  File R is the input for the run; File RK is the updated output file.

  The debugging procedures are:

  (1) Patch segments of the routine.

  (2) Perform a conditional trace of a small portion of the program.

  (3) Always trap and list the value at BAL in decimal representation.

  (4) Exit if the value at BAL goes to 10 or less.

  (5) Rewind the output file RK, bypass 20_8 blocks, and list 10_8 blocks in Fieldata code, if the run goes t
  completion.

- Cards:

  (1) #TESTƀƀPAY/2,JOB

  (2) ƀSETƀƀSTART/ƀ,HIT+4,61000/PATCH,#

  ƀPATCH,2700000306, 1201000000,61000/START+5

  (3) ƀATƀƀHIT/,5/500,B

  (4) ƀTRACEƀƀB,PROCRD,PROCRD+20,

  (5) ƀTRAPƀDƀBAL/ƀ,

  (6) ƀIFƀƀPROCRD+3/ƀW(BAL)/LE/(10)*AND*,K

  (7) ƀEXITƀƀPROCRD+4/K

  (8) ƀDUMPƀPAƀEND/ƀ,RK,20,10

  (9) ƀEND

- Explanation:

Card 1 is the call for the Test System. PAY is the name of the program to be tested; 2 is the version; and the program is on the job library. All primary storage assignment cards must precede the call for test.

Card 2 patches the program with one instruction at HIT+4; and the continuation card is used to insert three instructions at location PATCH.

Card 3 sets an AT statement at location HIT. Every 5th time that the program reaches the location HIT, switch B will be turned. At all other times, the switch will be turned off. The operation may occur a maximum of 500 times before the AT statement is bypassed.

Card 4 is the TRACE statement which is conditional to the setting at switch B. Only when switch B is set to on will the printing trace be performed. The printing trace will start at location PROCRD and will end with location PROCRD+20.

Card 5 is an unconditional trap of location BAL. Location BAL will be listed in decimal format each time that an instruction addresses BAL on a store cycle.

Card 6 is an IF statement which keeps an unconditional check on location BAL. When the value of BAL goes to 10 or less, switch K will be turned on; otherwise, switch K is always turned off. The IF statement is used in conjunction with the EXIT statement of Card 7.

Card 7 is the EXIT statement. The exit will be performed only when the routine reaches the instruction located at PROCRD+4 and switch K is set to on.

Card 8 calls for peripheral dump of the output file RK. The unit will be rewound and bypassed 20_8 blocks. The next 10_8 blocks will be submitted to primary output in Fieldata format.

Card 9 is the nonoperative END statement which signals the end of the secondary control stream. This card will cause the routine PAY to be loaded and activated.

- Program:

```
START   CARD$ƀPCARD
        JP*CRDERR*ANEG
        REWIND$ƀR                 RWD IPT
        JP*TERR*ANEG
        REWIND$ƀRK                RWD OPT
        JP*TERR*ANEG
        CVT$ƀPCARD+4,10D          CVD FD TO BINARY
        SUB*Q*O*QNOT
        JP*CRDERR                 CARD ERROR
        STR*Q*W(BAL)
TRD     READ$ƀR,100D,TRDA,O       TAPE READ
        JP*TERR*ANEG
        ENT*A*W(PCARD)
        SUB*A*W(TRDA+4)*AZERO
                                  FOUND RECORD
        JP*TRD                    GET NEXT RECORD
```

```
PROCRD      RPL*Y+1*L(COUNT)
HIT         ENT*Q*W(PCARD+1)
            CL*B1
            CL*A
            ENT*B2*10
            LSH*AQ*6                    ID TO A REG
PROCRD2     ENT*B1*B1+1
            COM*A*U(TAB+B1)*YLESS       TEST FOR TYPE
            JP*PROCRD4
            BJP*B2*PROCRD2
            PRINT$ƀTRDA,20,5            PRINT
            PRINT$ƀCON1,3,5             ERROR MSG
            JP*TRD
PROCRD4     ENT*Q*L(TAB+B1)             PICK UP FACTOR
            STR*Q*W(PCRD+2)
            MUL*W(PCARD+10)
            RPL*Y–Q*W(BAL)
            ENT*A*0505050505
            STR*A*W(PCRD+3)
            STR*A*W(PCRD+7)
            DPENT*TRDA
            DPSTR*PCRD+10
            DPENT*PCARD+11
            DPSTR*PCRD+13
            ENT*A*L(COUNT)
            LSH*Q*15D
            ENT*B3*4
            ENT*A*6060606060
PROCRD6     LSH*A*3
            LSH*AQ*6
            BJP*B3*PROCRD6
            STR*Q*W(PCRD+15)
OPUT1       PUNCH$ƀPCRD,16D             PUNCH CARD
            JP*CRDERR*ANEG
OPUT2       PRINT$ƀPCRD,16D,2           PRINT CARD
WRTFILE     WRITE$ƀRK,100D,TRDA,0       WRITE OPT

            JP*TRD*APOS
TERR        SEL*CL*4477777777*AZERO     DIAGNOSTICS

            JP*TERR2                    UNRECOVERABLE ERROR
            REWIND$ƀR                   RWD IPT
            WRITEOF$ƀRK                 WRITE OEF OPT
            REWIND$ƀRK                  RWD OPT
END         EXRN*5                      EXIT
TERR2       PRINT$ƀCON2,2,77            PRINT MESSAGE
            JP*END
CRDERR      ENT*B7*PCARD
            JP*TERR2+1
PCRD        RRSERVE*16D
PCARD       RESERVE*16D
TRDA        RESERVE*100D
BAL         0
COUNT       0
```

```
TAB             0
                15*25
                20*30
                17*21
                16*11
                33*14
                31*27
                34*67
                27*15
CON1            2712102427                RECORD ERROR
                1105122727
                2427050505
CON2            1006271105                CARD ERROR
                1227272427
PATCH           RESERVE*20
```

7. SYSTEM GENERATION

7.1. SYSTEMS GENERATION AND INITIALIZATION

Building and maintenance of OMEGA and related processors comprise two distinct relatively independent procedures: the generation and collection of OMEGA elements, and the initialization of OMEGA.

7.2. SYSTEMS GENERATION

Systems generation may be defined as the procedures required to build an executable OMEGA tape which can be used to bootstrap the operating system in the UNIVAC 494 System. To build the bootstrap tape, the following steps will be required, in one form or another:

(1) Assemble from source code, and collect the generated RB elements into executable form all system library elements. These include all programs or routines which are to be contained in the system library, e.g. secondary executive elements, input/output handlers, common subroutines, systems processors, and user programs. System library programs or routines have no special distinction as to code or format; they are assembled and collected with normal system processors, e.g. 494 SPURT, 494 Assembler Loader, (ASM) etc. Each element which will be entered into the system library must have a unique name/version.

The secondary executive elements, common subroutines, and input/output handlers have special programming conventions which must be used to effect their call and activation.

(2) Generate OMEGA maps and tables which are as follows:

- System table links, a table to be collected with the interrupt processors and used to contain: channel control blocks; installation variables for the operating system; base addresses for all chains, primary storage push/pop, addendums, etc.; the OMEGA task addendum; Externally Specified Index (ESI) and Buffer Control Word (BCW) positions; and, miscellaneous values used to control OMEGA.

- Remote facility map describes remote devices, name/version of handler, and mnemonics used to assign remote device and handlers.

- Facility map, describes on-site peripherals, name/version of handlers, and mnemonics used to assign remote devices and handler.

- Mass storage summary, describes direct access storage on each channel.

- Master File Directory (MFD) describes files which will be contained in the system along with system requirements.

(3) Assemble and collect the basic executive block which will contain: interrupt processors, dispatcher, system table links, primary storage allocation, task control routines, content supervisor, input/output director, common subroutine linkage, input/output cooperative control, communication director, optional user real time control and dummy tape handler used to load bootstrap block(s).

(4) Assemble and collect elements of the systems initialization routine used to set up the OMEGA routine at bootstrap time.

(5) Record the elements produced from Steps 1–4, along with any drum-stored job streams on magnetic tape through the Program Library Editor to form an OMEGA system tape. There is a prescribed order in which the elements are to be entered on the tape, which in general, is as follows:

Basic executive routine

Initialization routine

Secondary executive routines, maps, and tables

Input/output handlers and common subroutines

Systems processors, user programs, and job control streams

7.3. SYSTEMS INITIALIZATION

Systems initialization is a routine which is activated upon completion of the bootstrap function. The routine is composed of a number of independent elements collected into a string, each routine of which is responsible for initializing a particular portion or routine in OMEGA. These include elements to initialize input/output cooperative control, content supervision, primary storage allocation, reading and storing the system library, etc. The last one or two of the initialization elements will establish job control streams to activate user routines, remote communication initialization, etc.

8. UTILITY SERVICE ROUTINES

8.1. GENERAL

OMEGA utility routines are software components which are designed to perform specialized program conversion and management tasks. Utility routines include the REXecutor and Utility Processor, which are discussed in this section, and the Basic File Handler, Random Storage File Handler, Report Writer, and CONVAID, which are described in UNIVAC 494 Real-Time System library publications.

8.2. THE REXECUTOR

The REXecutor is designed to load and execute REX 490 Real-Time Executive Routine oriented programs under the UNIVAC 494 Executive OMEGA System. The REXecutor will load and execute programs which have been assembled by SPURTFD with simple relative (SPURT output 321) or complex relative (SPURT output 322) object coding and standard REX request packets.

The REXecutor is composed of two secondary executive routines, one of which is re-entrant and the other is nonreentrant, and a 400_8 word interface which is attached to each task. This organization allows concurrent REXecutor execution of several independent jobs.

The REX 490 worker program is loaded into primary storage by the REXecutor. Once the worker program is loaded and initiated, it is allowed to run free. The REXecutor is not an interpretive executor. The REXecutor is activated by the SILRJP instruction to the jump table at location 140–146 (the RIL operation is performed immediately by the REXecutor to prevent interference with a real time environment). The request is interpreted and either reformatted for submission to OMEGA or executed by the REXecutor itself, whichever is apporpriate. Checkstats (status checking) and takeovers are executed in the same way as in REX. Proper REX status words will be returned. The return points are the same as they would be under REX.

8.2.1. Restrictions

While the REXecutor is able to operate most worker programs, some restrictions are imposed:

- The REX request packets must conform to standard 490 REX specifications. Any packet written to conform to special site modifications of REX will not be executed properly, if at all.
- The worker program should be debugged. A program which has not been debugged may destroy the 400_8 word interface which is vital to the REXecutor.
- The REXecutor is not designed to execute real time communications programs.
- Execution of hardware level input/output instructions (privileged instructions) will cause a fault.

- Input/output facilities cannot be acquired during execution time. The facilities must be assigned prior to the selection of the REXecutor task. They will be held by the REXecutor until requested by a REX internal facility request. This implies that once a facility is released, it cannot be acquired again.

- The maximum number of characters which can be taken by an ACCEPT function from the console is five; additional characters are ignored.

- The UNISERVO IIA PWRITE command will be treated as a write function, without regard to density. An end-of-file (EOF) mark will be written if, following a UNISERVO IIA write command (WRITE or PWRITE), an input/output command, which will move the tape backwards, is issued (REWIND, RWI, LOCATEB, SEARCHB, READB, or MOVEB). The tape will then be positioned in front of the EOF mark before execution of the command for moving the tape in a reverse direction. This procedure is necessitated by the automatic EOF (erased tape) following a UNISERVO IIA write operation. Writing over old files is not recommended.

- Initialization is a function of the #ASG statement. PIN functions, FASTRAND head positioning, and UNISERVO IIIC, card reader, and card punch initialization packets will be ignored and marked as completed by the REXecutor.

- Disc and communications facility requests or packets will not be honoured.

- The binary time at which the REXecutor was intiated will be in word 147. This time is in UNIVAC 494 format and will not be updated by the REXecutor. The date is in word 135.

- Only five input/output operations, with or without CKSTAT, are allowed to be outstanding at one time. Requests in excess of this limit will be treated as a REX addendum overflow.

- During the load process, no modifications of channel/unit positions in REX I/0 packets, T-TAG or D-TAG are performed. Therefore, original channel/unit numbers specified by MEANS, FACIL, and ASSIGN SPURT directives will be contained in the packets at execution time and must conform to channel/unit numbers on used MEANS control statements.

- REXexecutor will not return some of the status codes provided by REX, e.g. drum down, buffer filled before EOB, or rewinding. These are either handled between OMEGA and the operator, or are not included in OMEGA.

- Due to the disparities between the REX addendum and REXecutor's program interface, the UNIVAC 490 program should not contain instructions which modify its own addendum.

- The following REX loader requests will be ignored and marked as done by the REXecutor. Where possible, the request will be marked as not having been fulfilled.

REX LOADER FUNCTION CODE	REX SERVICE REQUEST
13	Real time initialization
14	Start slave
15	Rerun dump
20	Subroutine load
20	Real time extension
21	Internal load request
25	Core release
26	Random storage release

- The presence of the following REX loader requests will be considered as an error. The REXecuted program will be terminated.

REX LOADER FUNCTION CODE	REX SERVICE REQUEST
00–10	Unassigned
11	Site Utility
12	Unassigned
27–7777	Unassigned

In addition, the exchange request will be considered as an error and cause the program to be terminated.

- Logical lockout of magnetic tape units will be ignored.

- Any subroutine employed by a REX 490 program must be defined by a PROG statement and loaded at the same time as the primary program. The subroutines must be activated by a jump or return jump. As noted below, subroutine load packets will be ignored. This limitation does not apply to the loading of secondary segments.

- There is a slight difference in handling the STOPRUN condition. If the STOPRUN is caused by the execution of REX*STOPRUN, the following message will be typed on the console: REX STOPRUN P = XXXXX PLEASE ANSWER (XXXXX is the address of the STOPRUN). There are five possible responses.

* (asterisk)	Start program following the STOPRUN packet.
B	Start program from beginning
XXXXX	Start program at the specified five-character address
T	Terminate program by REX*TERMRUN
A	Abort program

If the STOPRUN is caused by an error condition arising from the execution of an I/O packet and no error address in the CKSTAT, the following messages will be typed on the console: I/O ERROR, NO ERROR ADDRESS followed by REX STOPRUN P = 77777 PLEASE ANSWER. This condition is handled the same as the normal STOPRUN condition except that an asterisk (*) is an illegal response.

8.2.2. Control Cards

Execution of a program under the REXecutor is very similar to execution of any other normal task under OMEGA. The REXecutor is activated by a #REX statement.

- Format:

  The REX statement is of the following form:

  #REXƀoptionsƀminimum core/maximum core

- Options:

  X Abort the task and the entire job if any fatal errors are encountered.

  Y Continue processing, if possible, although errors were detected during the interpretation of control cards or the loading of the worker program.

  Absence of an X or Y option implies that this task should be terminated and the next task within the job stream initiated, if fatal errors are detected during execution.

- Specifications:

  Minimum core/Maximum core is the total amount of primary storage required to hold all programs, including subroutines, and the extra scratch primary storage during the execution of the entire task. OMEGA will automatically increase this amount by 400_8 words to allow for inclusion of the REXecutor interface. Maximum primary storage specifications plus the 400_8 work interface cannot exceed 32K. The amount of primary storage requested should be divisible by 100_8.

  When the task is selected and the primary storage is allocated, OMEGA will insert the 400_8 word interface into the beginning of the primary storage area. This interface is for the exclusive use of the REXecutor and must not be changed in any way by the worker program. All worker programs will be loaded above the interface. Control will be given to the interface, which in turn calls in the REXecutor proper.

  When the REXecutor is activated, the routine begins reading secondary control cards through primary input. There are five different types of secondary control cards:

  MEANS for equating the REX channel/unit with an OMEGA file code

PROG	for loading the worker programs
A	for absolute errata
B or C	for parameters
PS	for activating the worker program

8.2.3. The MEANS Statement

All peripheral and direct access storage assignments must be assigned previously to the task by means of #ASG statements. There will be no load time allocation of peripherals as under REX. This OMEGA assignment must be equated to a specific peripheral device in the worker program. The peripheral device is a unique channel/unit as defined by the SPURT MEANS, ASSIGN, and FACIL statements. Since the channel and unit are not altered when the worker program has been loaded, the channel/unit specification must be exactly as it appears in the SPURT MEANS and ASSIGN statements. The REXecutor does not examine the contents of the previously submitted #ASG statement. This allows some degree of flexibility.

A UNIVAC 490 program written for UNISERVO IIA tapes, for example, can now be run on a UNIVAC 494 System with UNISERVO VIII C tapes without alteration or reprogramming. The UNISERVO IIA tape packet will be properly submitted for UNISERVO VIII C tapes by the REXecutor.

- Format:

  The MEANS statement has the following form:

  MEANSƀoptionsƀfile code, channel/unit, peripheral type, drum base/length

- Options:

A The described unit will be used to satisfy an internal REX facility request. The channel/unit specification may contain any arbitrary but unique channel and unit number (from 0–16_8) which is not used in any of the UNIVAC 490 worker programs.

- Specifications:

File Code is the alphabetic file code to which this channel/unit is being equated. This file code is the code given on the previously declared #ASG statement for this peripheral device.

Channel/Unit is the numeric channel/unit as defined by the SPURT MEANS and ASSIGN statements. The channel number can range from 0–16_8; the unit number can range from 0–16_8. In the case of hand coded packets, the channel/unit used in the packet must be specified. The channel/unit will be equated at execution time to an OMEGA file code and not to a physical UNIVAC 494 channel/unit. When a unit is not specified, unit 0 will be assumed if a unit number is necessary.

Peripheral Type is a mnemonic for the REX peripheral unit type in the original REX input/output packet, or in the internal facility request if the A option is used. Only the following are valid entries:

MNEMONIC	SUBSYSTEM
FH880	FH–880 Drum
FAST	FASTRAND
UN2A	UNISERVO II A
UN3A	UNISERVO III A
UN3C	UNISERVO III C
UN6C	UNISERVO VI C
UN8C	UNISERVO VIII C
CRIN	Card Reader
CROUT	Card Punch
PRINT	High Speed Printer
PTIN	Paper Tape Reader
PTOUT	Paper Tape Punch
PIN	Submit card read request to primary input
POUT	Submit high speed printer request to primary output
SOUT	Submit card punch request to secondary output

The CRIN, CROUT, and PRINT entries can be used only if these devices have been assigned to the REXecutor by means of a previously submitted #ASG statement. If it is desired that a card printer input/output request be submitted to the primary input, or primary or secondary output streams, the PIN, POUT or SOUT entries are used. If the POUT entry is used, no form control such as that allowed by the #ASG or SPURT PIN statement is possible. If the PIN, POUT, or SOUT entries are used, the file code field on this statement must be left blank and the #ASG statement is not required.

Drum Base/Length is used to map a location from a REX mass storage address to an OMEGA logical increment. It is used only where the peripheral type specification is FH880 or FAST. It is required for those programs that use absolute drum addresses. For each specific direct access file assigned to the REXecutor by a distinct #ASG statement, the absolute base address and length of the file for the related UNIVAC 490 direct access file must be specified. If the drum base is not specified, a base of 0 is assumed. This will be correct only if the UNIVAC 490 program uses relocatable drum storage. If the UNIVAC 490 program uses absolute drum addresses, the FH-880 drum or FASTRAND sector address used as the base of the file must be specified. The length specification is the number of words or sectors in the file. Absence of a specification implies that the entire drum or FASTRAND mass storage area is required by the UNIVAC 490 worker program.

8.2.4. The PROG Statement

The PROG statement is used to load the REX 321 or REX 322 program into primary storage. Any number of PROG statements may be submitted to the REXecutor, one for each program to be loaded. Any subroutine must be loaded before the primary program is activated. Only one program, the primary program, can be segmented. The total amount of primary storage required for all programs and the 400_8 word interface cannot exceed 32K.

It is important that all MEANS statements which pertain to the program described by a PROG statement be submitted to the REXecutor before the program is loaded by a PROG statement.

- Format:

  The PROG statement has the following form:

  PROGƀoptionsƀfile code, program ID/name, base, logical increment

- Options:

  P refers to the primary program, and the only program capable of being segmented and receiving REX parameters. The program must have an Executive Information Region (EXIR). One program must be described by the P option; however, there cannot be more than one program with a P option.

  L Rewind the load tape before searching for the program.

  R Free the peripheral used to load the program (described by the file code specification) after the program is loaded.

- Specifications:

  File Code is the alphabetic file code for the peripheral unit where the object code (321 or 322) resides. If the assignment is for a direct access device, the logical increment specification will be required to define the location of the object code. This file code must be defined previously by a #ASG statement.

  Program ID/Name is used to identify the object code which will be loaded and executed. The program ID is a ten-character searchword (e.g., 74747 library number) placed in the header block by SPURT. This specification is mandatory. The program name is an optional field which is used where the program ID is not unique. The program name is the label attached to the SPURT program statement and is placed in the object code header block by the assembler.

Base is an optional field that specifies the location, relative to the base of the execution area (i.e., relative to 0 or to the RIR), at which the program will be loaded. If this specification is used, the value must be greater than 400_8. If no specification is made, each program will be loaded immediately following the previous program, beginning at 400_8. Once a base is given, all subsequent programs will be loaded following this base unless the subsequent programs also have bases. This specification is normally used to insure that a subroutine is loaded at a specific place.

Logical Increment is the increment to the base of the element which describes the beginning address of the object code. The increment is required only if the peripheral device where the object program resides is direct access storage.

8.2.5. A Card

The A card is used for errata in the same way that it is used in REX and the format is identical to the format used for the A card in REX. The addresses will be relative to the task base. This includes the 400_8 word interface area. For example, if a program is loaded at 1500_8 and an instruction at 203_8 (relative to the program base) is to be changed to 6100000570 (the 570 is also relative to the program base), the A card would be Ab̸b̸b̸b̸01703610000 2270. The A card must follow the PROG card which refers to the program being modified.

- Format:

CARD COLUMN	CONTENTS
1	A
2–3	Segment number (Blank implies a control segment.)
4–5	Blank
6–10	Address of correction, relative to the task base (RIR or relative zero). The value must be greater than 400_8.
11–80	Seven 10-character fields used to contain octal errata. Errata words will be loaded at consecutive locations starting withtthe address given in columns 6–10. A blank correction word will terminate the card.

8.2.6. B and C Cards

The B and C cards are used for parameters exactly as they are used in REX. Parameters will be stored beginning at the address specified in the upper portion of word 4 of the EXIR. The REXector will supply the count of the number of parameter words in the lower portion of word 4. The B and C cards must follow the PROG card which loaded the program that is receiving the parameters.

8.2.6.1. B CARD FORMAT

The B card is used for numeric parameters. The format is as follows:

CARD COLUMN	CONTENTS
1	B
2–3	Number of words of parameters. A blank implies seven words, the maximum.
4–10	Blank
11–80	A maximum of seven 10-character octal numeric parameters. Each 10-character field will be converted from Fieldata to binary. Leading blanks will become zeros. An entirely blank field will be converted into ten zeros. Parameters will be stored from left to right until the number of words specified in columns 2–3 have been stored.

8.2.6.2. C CARD FORMAT

The C card is used for alphanumeric Fieldata parameters. The format is as follows:

CARD COLUMN	CONTENTS
1	C
2–3	An octal number specifying the number of parameters on this card. A blank implies 16_8, the maximum.
4–10	Blank
11–80	A maximum of 16_8, five-character alphanumeric parameters. These characters will be stored in five-character groups regardless of their content. Parameters will be stored from left to right until the number of words specified in columns 2–3 have been stored.

8.2.7. The PS Statement

This statement is used to activate the worker program. All images in the primary input stream between this statement and the #END statement will be considered to be data for the worker program. REXecutor control statements which follow this statement will not be recognized.

- Format:

  PSƀoptionsƀP,A,Q,B1/B2/B3/B4/B5/B6/B7

- Options:

  D Perform an octal dump of the worker area upon termination of the program, regardless of the reason (normal or abnormal).

  E Perform an octal dump of the worker area upon termination of the program only if an error occurs during execution.

  P Perform an octal dump of the worker area before activating the worker program.

- Specifications:

P is the start address of the worker program relative to the task base (relative to 0 or the RIR). This address must be greater than 400_8. Lack of a specification implies that the start address in the upper of word 0 of the EXIR of the primary program (the program whose PROG statement had the P option) is to be used.

A, Q and B1–B7 are optional fields representing registers. The value in the location for the specified register will be inserted into that register. Lack of a specification indicates that the register will be set to 0 when the worker program is activated.

8.2.8. Control Stream Examples

Example 1:

```
(1)  #JOB##EXAMPLE1/REX, 12345,C,20,40/0

     #GO##SOMEPROG,SYS

     #END

(2)  #ASG##RAN,A,100000,SCRATCH

(3)  #ASG RU TAPE,B,,INPUT

(4)  #ASG R TAPE,C,,SCRATCH

(5)  #ASG RU TAPE,D,,OUTPUT

(6)  #ASG RUW TAPE,E,,PROGRAM

(7)  #REX X 2000/2400

(8)  MEANS A,15,FH880,0/100000

(9)  MEANS B,3/0,UN3C

(10) MEANS C,3/1,UN3C

(11) MEANS D,3/2,UN3C

(12) MEANS ,6,PIN

(13) MEANS ,10,POUT

(14) PROG PR E,7474760213

(15) A    007231103101033

(16) A01 014076100006375

(17) B01   ,,,,,,, 36   176

(18) C03    ALPHA PARAMETER
```

```
(19) PS E ,,16//8D//100

(20) DATA CARDS

(21) #END

(22) #LOAD XL MOREPROG,MOREPROG/ABS

     #END

     #GO MOREPROG/ABS

     #END

     #FIN
```

(1) Job stream, up to the REXecutor parameters.

(2) Assignment for direct access storage. This statement matches the MEANS control statement at (8).

(3) Assignment for tape. This statement matches the MEANS control statement at (9).

(4) Assignment for tape. This statement matches the MEANS control at (10).

(5) Assignment for tape. This statement matches the MEANS control statement at (11).

(6) Assignment for tape. This is the tape unit which contains the object coding. The statement matches the PROG control statement at (14).

(7) REXecutor task control statement, which states that the minimum primary storage required is 2000_8 words and that the maximum primary storage needed is 2400_8 words. This does not include the 400_8 word interface. If the maximum primary storage request is granted, the total amount of primary storage that will be allocated is 3000_8 words.

(8) Informs the REXecutor that file code A maps FH–880 drum requests to channel 15 in the object program. The SPURT packet was set up for relocatable drum, hence the drum base of 0. The file is $100{,}000_8$ words long. The SPURT MEANS and FACIL statements could have been:

```
DRUM          MEANS*C15

DRUM          FACIL*DRUM*R*100000
```

Any DRUM-AREA and D-TAG statements have no effect upon the REXecutor MEANS statement.

(9) Informs the REXecutor that file code B maps to unit 0 for UNISERVO III C requests to channel 3. It makes no difference that the UNIVAC 494 peripheral might be a UNISERVO VI C or VIII C.

(10) Informs the REXecutor that any UNISERVO III C packet for channel 3, unit 1, will be executed on file code C.

(11) Informs the REXecutor that any UNISERVO III C packet for channel 3, unit 2, will be executed on file code D.

(12) The REX 490 program contains card reader packets for channel 6. The REXecutor will obtain the cards from the primary input stream. The comma in the file code field is mandatory.

(13) The REX 490 program contains high speed printer packets for channel 10. The REXecutor will submit the printer images to primary output. The comma in the file code field is mandatory.

(14) This statement loads the object program whose identification on the header block is 7474760213. This program will be found on the peripheral unit whose file code is E. The program must have the P option, since it is the only program being loaded and one program must be designated as a primary program. The first instructions of the program will be loaded at 400$_8$.

(15) Corrects primary storage location 00723. This is the instruction that is at 00323 on the 110 listing.

(16) Corrects primary storage location 01407 (01007 relative to program base) whenever segment 1 is loaded.

(17) Two numeric parameters, 0000000036 and 0000000176, will be inserted into the location specified in the EXIR.

(18) The three word alphabetic parameter will be inserted behind the two numeric parameters.

(19) The worker program will start at the address specified in the EXIR of the worker program. The operating registers will be initialized as follows:

A=0, Q=16$_8$, B1=0, B2=0, B3=10$_8$, B4=0, B5=100$_8$, B6=0, B7=0.

(20) Data cards to be read by REXecutor when a card read packet is executed.

(21) This ends the input stream for the REXecutor.

(22) The remainder of the primary input stream.

Example 2:

```
(1)   #JOB##EXAMPLE2/REX,54321,C,30D,50/0

      #GO##SOMEPROG/SYS

      #END

(2)   #ASG MU UN8C,M,,SPURTFD

(3)   #ASG##UN8C,E,,SCRATCH

      #ASG##UN8C,F,,SCRATCH

      #ASG##UN8C,G,,SCRATCH

(4)   #ASG J UN8C,H,,322 OUTPUT

(5)   #ASG W UN8C,J,,DUMMY

(6)   #ASG##RAN,A,5000

(7)   #REX X 24000/30000

(8)   MEANS A J,1/3,UN3C
```

```
(9)   MEANS A E,1/10,UN3C

      MEANS A F,1/11,UN3C

      MEANS A G,1/12,UN3C

(10)  MEANS A H,1/13,UN3C

(11)  MEANS ,4,POUT

      MEANS ,3,PIN

(12)  MEANS M,1/0,UN3C

(13)  MEANS A,7,FH880,0/5000

(14)  PROG PL M,7474710020

(15)  A10 03076641200014205000000001

(16)  PS E ,,,20

(17)  SPURT SOURCE CARDS

                  OUTPUTS 322

      /////

(18)  #END

(19)  #ASG##UN8C,A,,SCRATCH

(20)  #ASG##RAN,E,410000

(21)  #ASG U UN8C,F,,PROGRAMS

(22)  #ASG W UN8C,G,,OUTPUT

(23)  #REX X 15000

(24)  MEANS A A,1/0,UN3C

(25)  MEANS G,2/0,UN2A

(26)  MEANS E,11,FAST,0/10000

(27)  MEANS ,4,POUT

(28)  MEANS ,3,PIN

(29)  PROG PL F,7474720471/PROG1

(30)  PROG F,7474720471/PROG2

(31)  PROG LR F,7474710011,10000
```

(32) PROG RL H,7474700322

(33) A 023771200000000

(34) PS E 10000

(35) DATA CARDS

(36) #END

(37) #FIN

(1) Job stream, up to the REXecutor parameters.

(2) Assignment statement for the tape unit containing the object program. SPURTFDA is being used as an example of an object program. This set of parameter cards will produce a 322 output which, in turn can be executed by the REXecutor.

(3) The three scratch tapes required by SPURT.

(4) The output tape which will hold the 322 object program.

(5) A dummy unit required by SPURT. As part of its initialization SPURT acquires, then releases, a tape unit. When the peripheral release is performed by the REXecutor, the unit is returned to OMEGA. Thus, a dummy unit is required.

(6) SPURT stores macro information on direct access storage. If source coding has macros, an assignment of this type is required. If there are no macros, this assignment is not required.

(7) Activates the REXecutor. SPURT requires a minimum of 24000_8 and a maximum of 30000_8 words of primary storage.

(8) The MEANS statement for the dummy unit mentioned in (5). This MEANS statement must be in this position in job deck. All tape units are acquired by SPURT by means of an internal facility request.

(9) The MEANS statement for the three scratch tapes used by SPURT.

(10) The MEANS statement for the 322 output tape.

(11) The MEANS statement for card and printer packets.

(12) A MEANS statement for the load tape unit as required for SPURT.

(13) A MEANS statement for the direct access storage for macros. This statement is not required if there are no macros.

(14) Loads SPURTFDA from file code M.

(15) Inserts a correction at 2476, relative to program base, every time that segment 10 is loaded. This correction causes SPURT to come to a REX TERMRUN.

(16) Activates SPURTFDA, SPURT requires that the channel/unit of the load tape unit be in B1.

(17) The SPURT source program.

(18) The end of the input stream to the REXecutor.

(19) Begins a new REXecutor run. This assignment matches the MEANS statement at (24).

(20) Assigns 410,000$_8$ words of direct access storage. This assignment matches the MEANS statement at (26). Note that the MEANS statement calls for FASTRAND Mass storage. The length of the file is stated in sectors. The facility assignment must be stated in words.

(21) The facility assignment for a tape unit holding previously assembled 321 or 322 object coding. This statement matches the PROG statements at (29), (30) and (31).

(22) This facility assignment matches the MEANS statement at (25).

(23) Activates the REXecutor. The total amount of primary storage needed by the program is 15,000$_8$ words.

(24) The MEANS statement for UNISERVO III C, channel 1, unit 0, packets.

(25) The MEANS statement for UNISERVO II A channel 2, unit 0, packets. Note that both the UNISERVO II A and UNISERVO III C commands will be executed on UNISERVO VIII C tape units. The REXecutor will adjust the OMEGA requests as required to allow this intermixing.

(26) The MEANS statement for FASTRAND packets. Note that while the length of the file on the MEANS statement is in sectors, the facility statement requires sectors to be converted to words.

(27) The MEANS statement for high speed printer packets.

(28) The MEANS statement for card packets.

(29) Loads the primary program at 400$_8$. The program length is assumed to be 4200$_8$ words.

(30) Loads a subroutine. Note that the program ID fields for (29) and (30) are identical. The name field is used to distinguish the two object programs. This program will be loaded immediately behind (29) or at 4600$_8$.

(31) Loads another subroutine at 10,000$_8$. Since this program is located somewhere between the load point and (30), the L option is used to save unnecessary searching. The program is assumed to be 3117$_8$ words in length.

(32) Loads the program assembled by the previously executed SPURTFDA run behind (31) at 13117$_8$.

(33) Absolute errata.

(34) Activates the worker program at 10,000$_8$.

(35) Data cards for the worker program.

(36) Terminates the input stream for the REXecutor. If the worker attempts to read this card, a 06 status code will be returned.

(37) End of primary input stream.

8.3. THE UTILITY PROCESSOR

The OMEGA Utility Processor offers a flexible and device-independent method for generating and manipulating data files. By means of the Utility Processor, data may be transferred into primary storage from a file, from primary storage to a file, or from one file to another file.

The output of the Utility Processor is a load (absolute) element which is capable of performing the operations described by secondary control statements. The generated element, with the name and version specified by the user, is placed in the job library. The element may then be activated by the GO statement, or the element may be retained for subsequent use by means of the OUT statement.

8.3.1. The Utility (UTL) Control Statement

The Utility Processor is activated in response to the UTL control statement. Options on the UTL statement determine the disposition of illegal control statements or errors encountered during execution.

- Format:

  The UTL statement has the following general form:

  #UTLƀoptionsƀname/version

- Options:

  X Abort the entire job if errors are detected.

  Y Continue processing the task if nonfatal errors are detected.

  Z Terminate the task, but continue the job, if nonfatal errors are detected.

  Absence of X,Y, and Z options causes the task to terminate if errors are encountered during processing (same as Z option).

- Specifications:

  Name/version specifies the name, and version if any, which will be assigned to the element produced by the Utility Processor. The name may contain up to ten alphanumeric characters; the version may contain up to five alphanumeric characters. If the version is used, the version field must be separated from the name field by a slash (/). The entry in the name/version fields must be used in all future references to the element.

8.3.2. Secondary Control Statements

The secondary control statements of the Utility Processor are of two types, action and test. The action statements provide for the transfer of data, for looping, and for unconditional transfer of control. The test statements check for a specified condition, and, if the condition is true, transfer control. The test operators can be used to call user own-code subroutines which may return a value of true or false. The secondary control statements may also contain OMEGA control cards if a blank is present in card column 1. Such statements are submitted internally during the execution phase.

If any subfields in the secondary control statements are to be left empty, and if other non-empty subfields occur to the right of the empty field, the comma indicating the end of the missing field must be included.

Example:

ƀBLKRDƀƀA, 20,,0,,10

In the example, subfields 3 and 5 are not used and the double commas indicate the missing subfields in the presence of the non-blank subfields 4 and 6. (The BLKRD statement is discussed in Section 8.3.2.2.)

8.3.2.1. GENERAL SPECIFICATIONS AND OPTIONS

The general specifications and options of secondary control statements are described in this subsection. The formats of the statements, which are given in the following subsections, show the specifications and options applying to each statement.

- Specifications:

  – File code

  File code is a one or two-character alphabetic code which identifies the file being referenced. All file codes, other than ZA, ZB and ZC, must be assigned by either the ASG or MFD statement before the file codes are used. File code ZA specifies primary input, file ZB specifies primary output, and file ZC specifies secondary output.

  – Length

  Length specifies the maximum length of one record which is to be processed. The actual length of the record can be less than the length specified in the length field of the statement.

  – Logical increment

  Logical increment is the address in a direct access storage file where the operation will begin. The address is relative to the beginning of the file. Upon completion of the operation, the logical increment is updated, unless the R option is specified, by the length of the record that is read or written. The subfield is optional. If the logical increment is not specified, the operation is performed at the logical increment generated by the last operation on the specified file code. When writing to primary output (file code ZB), a logical increment of 1 must be specified to prevent overprinting. When data file conventions are followed, (see option D below) the logical increment is supplied to the packet and is updated in accordance with the R option. Only when the data buffers have been filled or emptied, and it is necessary to issue an I/O request, is the logical increment to be used. This generated increment may be meaningless, and it is not recommended that the logical increment be used. When data file conventions are followed, both READ and WRITE operations on the same file code are not allowed.

  – Relative address

  Relative address is an address which is added to the beginning of the buffer and at which the operation is initiated. The subfield is optional and, if not specified, or specified as zero, the operation is initiated at or from the normal buffer base. If the subfield has been specified previously as nonzero, and the subfield is not filled on the present statement, the value which has been specified previously is used.

- Options:

  D Follow data file conventions established by the Basic File Handler (see *UNIVAC 490/491/492/494 Real-Time Systems Basic File Handler Programmer Reference Manual, UP–7573* (current version).

  R Do not update the logical increment after this operation.

  F Transfer control (jump) if the condition being checked for is not present (not satisifed or false). Used with test operators only.

  A Translate element as action operator. Used for special user own code.

The absence of an option implies that the opposite course of action will be taken; that is, absence of the R option implies that the logical increment will be updated.

8.3.2.2. ACTION STATEMENTS (READ, BLKRD, SEARCH, WRITE, REWIND, WRTEOF, MOVE, GO TO, DO) ←

Action statements provide for the transfer of data, for looping, and for transfer of control.

- READ Statement

  The READ statement causes data to be moved from a specified file into primary storage.

  – Format:

  ƀREADƀoptionsƀfile code, length, logical increment, relative address.

  – Options:

  D Follow data file conventions.

  R Do not update logical increment for this file.

  – Examples:

  Read 100 words from file A.

  ƀREADƀƀA,100

  Read 100 words from file A beginning at 10000.

  ƀREADƀƀA,100,10000

  The first example reads from file A, with an unspecified logical increment, the second example reads from the file starting at the increment 10000.

- Block Read (BLKRD) Statement

  The BLKRD statement causes several blocks of data to be read into a contiguous storage area, and is equivalent to a series of READ statements. The area which is to be read into is filled with Fieldata spaces before the read requests are initiated.

  – Format:

  ƀBLKRDƀoptionsƀfile code, length, logical increment, relative address, number of records.

  – Options:

  D Follow data file conventions.

  R Do not update logical increment for this file.

  – Specifications:

  Number of records represents the number of read operations of the length specified.

  – Read 10 cards from file ZA:

  ƀBLKRDƀƀZA,16D,0,0,,10

In the example, 10 records are read into one large data area. To read 10D card images of 20_8 words each from file A, and write the images out as a single block on file B:

ƀBLKRDƀƀA,20,,0,,10D

ƀWRITEƀƀB,160D

- SEARCH Statement

The SEARCH statement causes successive words from the specified file to be read, with the first record being read from the specified logical increment. When a record is read, the first word is compared to the sentinel, and if a match is made, the operation is terminated. If the R option is specified, the logical increment at which the match is made is saved; otherwise, the logical increment is updated to search the next record. If no match is found before reaching an end-of-file condition, the operation terminates with end-of-file status.

– Format:

ƀSEARCHƀoptionsƀfile code, length, logical increment, relative address, sentinel

– Options:

D Follow data file conventions.

R Do not update logical increment for this file.

– Specifications:

Length is the maximum length of one record which is to be searched. The logical increment operates as follows: If a match is made at the n^{th} record, and the R option is present, the new logical increment equals the old logical increment plus (n – 1) x length. If the R option is not present, the new logical increment equals the old logical increment plus n x length.

Sentinel is the word compared to the first word of each block read operation. If a match is found, the operation is terminated. A sentinel must be followed by a D to denote a decimal number or by an A to denote Fieldata characters; octal numbers need no trailing character.

– Example:

Search file C in blocks of 100 records for a block beginning with 2345678D:

ƀSEARCHƀƀC,100,0,,2345678D

- WRITE Statement

The WRITE statement causes a record of information to be written from primary storage to the specified file. (When writing on file ZB, the number of lines spaced before printing is given in the logical increment field.)

– Format:

ƀWRITEƀoptionsƀfile code, length, logical increment, relative address

– Options:

D Follow data file conventions.

R Do not update logical increment for this file.

– Specifications:

The length subfield is optional. If the length is not specified, the length is the same as the record that is read.

– Examples:

Write out 300D words to file CD:

```
ƀWRITEƀƀCD,300D
```

Read and write a variable length record of less than 512D words:

```
ƀREADƀƀA,512D

ƀWRITEƀƀCD
```

- Rewind (RWND) Statement

The RWIND statement causes the resetting of the accumulated logical increment to zero. If the device is a tape unit, physical rewinding of the tape is initiated. File codes ZA, ZB, and ZC cannot be rewound.

– Format:

```
ƀRWNDƀoptionsƀfile code
```

– Options:

D Follow data file conventions.

- Write End of File (WRTEOF) Statement

The WRTEOF statement requests the write-end-of-file operation as specified for the device type and data conventions. If the D option is present, a standard end-of-file sentinel block is written on the file. On direct access files, no hardware operation is performed.

– Format:

```
ƀWRTEOFƀoptionsƀfile code
```

– Options:

D Follow data file conventions.

- MOVE Statement

The MOVE statement causes transfer of data between files. Blocks of data are read from one file, designated as file code 1, and written to another file, designated as file code 2, until an end-of-file or an error condition occurs on file code 1. An end-of-file condition is then recorded on file code 2.

– Format:

```
ƀMOVEƀoptionsƀfile code 1, file code 2, length
```

– Options:

D Follow data file conventions.

- GO TO Statement

The GO TO statement causes an unconditional transfer of program control to another statement in the program.

– Format:

```
ƀGOTOƀƀn
```

– Specifications:

n represents a signed or unsigned decimal or octal integer indicating the number of statements forward or backward to which control will be transferred.

– Examples:

To transfer control to the third statement past the GO TO statement, either one of the following statements may be used:

```
ƀGOTOƀƀ3
```

```
ƀGOTOƀƀ+3
```

The following statement transfers control to the second statement prior to the GO TO statement:

```
ƀGOTOƀƀ–2
```

- DO Statement

The DO statement causes the iteration of a fixed number of source statements.

– Format:

```
ƀDOƀƀn,m
```

– Specifications:

n represents the number of statements (not counting the DO operation) to repeat.

m represents the number of repetitions desired.

Nested DO statements are allowed if the inner DO loop is completely contained within the outer DO loop.

– Examples:

Perform 10_{10} read/write cycles from file code A to file code B:

```
ƀDOƀƀ2,10D
```

```
ƀREADƀƀA,100
```

ƀWRITEƀƀB,100

Set up a nested loop to read 10_{10} data cards, and copy each card 10_{10} times to file code T:

ƀDOƀƀ4,10D

ƀREADƀƀZA,16D

ƀDOƀƀ1,9D

ƀWRITEƀƀT,16D

ƀWRITEƀƀT,16D

In the above statements, the inner loop is completely contained in the outer loop (the terminal statement of the inner loop comes before the terminal statement of the outer loop).

8.3.2.3. TEST STATEMENTS

Test statements, the mnemonic operators for which are listed below, are used to check for various abnormal conditions, such as end-of-tape, end-of-file, and other contingencies.

- Format:

  The test statements have the following general form:

  ƀnameƀoptionsƀfile code, n

  Name represents the mnemonic operator for the test condition specified.

  – Options:

    F Transfer control if the condition being checked for is false.

  – Specifications:

    n represents a signed or unsigned decimal integer (with D appended) or octal integer which represents the number of statements forward or backward to which control is transferred if the test condition has occurred.

The test or error conditions and their associated mnemonics and codes are as follows:

MNEMONIC OPERATOR	CONDITION
FCN	Illegal function (41)
PRM	Parameter error (42)
SYS	Subsystem error (43)
EOF	End of file (44)
EOT	End of tape or end of transmission (45)

SCH	Unsuccessful search (46)
ICR	Illegal character (47)
NAS	Device not assigned (50)
ILK	Hardware interlock (51)
HFS	Hidden file segment (52)
ERR	Any error or abnormal condition (any negative number)

- Examples:

The following statement causes control to be transferred to the third statement preceding the EOF statement if the end-of-file condition is not present on file A:

ƀEOFƀFƀA,–3

The following statement causes a skip of the next five statements if any error condition (including end-of-file condition) has occurred on any previous read or write operation:

ƀERRƀƀA,6

The entire MOVE operation can now be expressed by the following statements:

ƀREADƀoptionsƀfile 1, length

ƀWRITEƀoptionsƀfile code 2

ƀERRƀFƀfile code 1,–2

ƀWRITEOFƀƀfile code 2

8.3.2.4. USER OWN CODE ROUTINES

User own code routines fall into the class of test operations. The user subroutine is entered by a return jump instruction. The subroutine must be acted upon previously by the #LOAD control statement on which the M option must be specified. The user program must not be segmented if the operation is to be correct.

- Format:

The test statement for the user own code subroutine has the following general form:

ƀnameƀoptionsƀfile code,n

name represents an operand which is the name of an absolute element in the job library.

- Options:

F Transfer control if the condition being checked for is false.

A Translate the element as an action operator.

- Specifications:

File code represents the specified file code of the packet to which index register B7 is set.

n represents the number of statements forward or backward to which control is transferred if the own code routine returns a test value of true.

- Examples:

The following statement activates the user routine OWNCODE.

ƀOWNCODEƀƀB,8D

This statement sets the contents of index register B7 to the address of the packet of file code B. If a result of true is returned, seven statements are skipped. The decks for assembly are illustrated below:

```
#SPURT  OWNCODE/RB                 #ASM    OWNCODE/RB
        START*BEGIN
        ___________                        ___________
        ___________                        ___________
        ___________                        ___________
BEGIN   ___________                BEGIN   ___________
        ___________                        ___________
        ___________                        ___________
        ___________                        ___________
        ___________                        ___________
        ___________                        END BEGIN

                                   #END

#END                               #LOAD   M OWNCODE/RB,OWNCODE

#LOAD   M OWNCODE/RB,OWNCODE       #END

#END                               #UTL    name/version
                                   _______
#UTL    name/version               _______
_______                            _______
_______                            _______
_______                            _______
_______                            _______
_______                            #END
_______
#END                               #END

#GO                                #GO
_______                            _______
_______optional data cards         _______optional data cards
_______                            _______
_______                            _______
_______                            _______
_______                            _______
_______                            #END
_______
#END
```

To translate the user OWNCODE statement as an action operator, the A option is specified. The statement is then translated to place the values in the parameter list into the appropriate place in the packet. See the example below:

ƀOWNCODEƀAƀB,100,10000,,ABCDEA

The above statement first causes the following operations: Register B7 is set to the address of the packet;

100 is placed in the length field;

10000 is placed in the logical increment field;

the last relative address of the buffer remains the same. (This is an actual address in the packet).

0607101112 is placed in the sentinel subfield.

After the packet is set up and executed, a return jump instruction is issued to activate the user own code. When the OWNCODE statement is translated with the A option, register B1 holds the address of the word containing the exit point. If the contents of relative location 0 are positive, the exit point is in the upper halfword of the location; if the contents of relative location 0 are negative, the exit point is in the lower halfword. If location 0 is positive, the value must be set negative; if location 0 is negative, the value must be set positive; that is, the setting must alternate, and register B1 must be incremented by one for correct operation upon resumption of the Utility Processor.

For correct operation of the Utility Processor, the user own code should restore any index registers used. User routines are activated with 15-bit index registers and with PLR=RIR.

For user own code routines, the I/O packet addressed by index register B7 is relative to the lower lock, and the RIR value of the user code is in the following format:

B7	FILE CODE	NO. OF WORDS
+1	BUFFER BASE	
+2	LOGICAL INCREMENT	
+3	SENTINEL	
+4	STATUS	
+5	FILE TYPE	NO. OF BUFFERS

File Types

0 — not yet defined for run

1 — unit record

2 — direct access files

3 — data file conventions

4 — remote devices

Upon return of control to the main program, a result of true is specified by setting (A) = 0, and a false result if specified by returning a value of 0.

8.3.3. Applicable OMEGA Control Statements

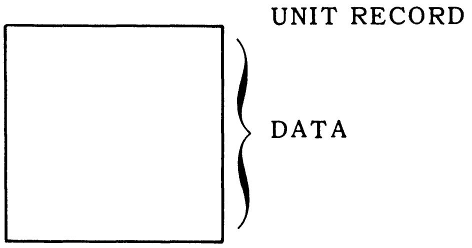
The following control statements are recognized by the Utility Processor:

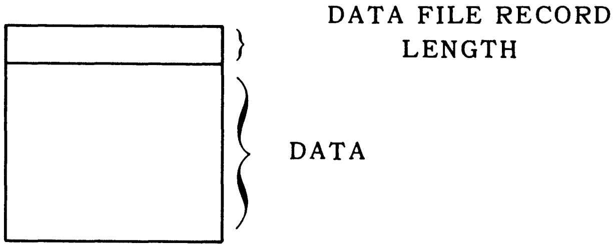
ASG, MFD, FREE, LASG, SWITCH, LFREE, CTMFREE, and LOG.

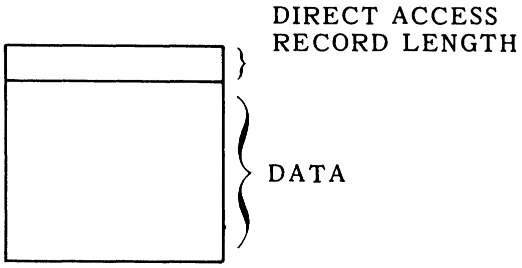
A negative status returned on the execution of any of the statements causes run termination. No card may extend past column 60, and continuation cards are not allowed.

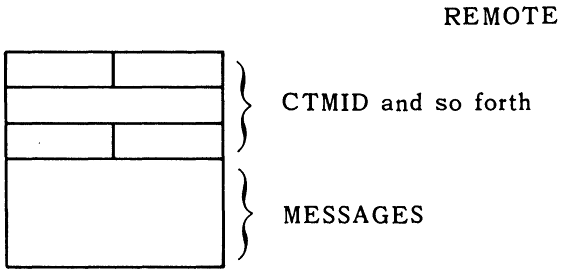
8.3.4. Data Conventions

Data conventions followed by the Utility Processor are:



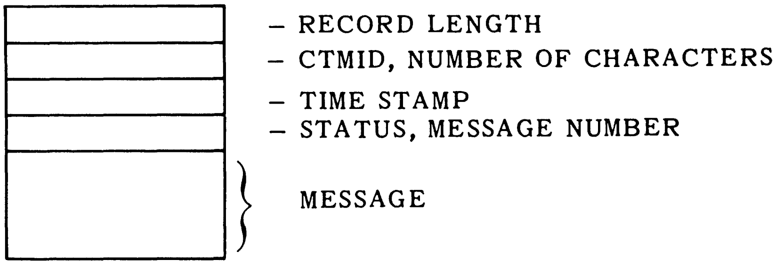






When remote messages or unit record data are transmitted to a data file or to direct access files, a one word header is added giving the length of the data. When access files or data file records are transmitted to unit record or remote devices, the one word header is stripped off. All transmissions to a remote device should contain in the first three words (after possible deletion of one word length header) the CTMID, number of characters, time stamp, status, and message number.

A message which is to be sent to a remote device from direct access files or data files appears in the following format:



The record length is stripped off before transmission to the remote device. A record of this format is generated for transfer of records from remote to direct access on data files.

8.3.5. Utility Processor Examples

The following examples illustrate the use of the UTL Processor:

- Example 1:

  Perform coding to a card to drum transfer operation of card images from the primary input. File code ZA designates primary input.

```
#JOB

#ASGƀJƀRAN,D,5000ƀ20000

#UTLƀƀname/version

ƀMOVEƀƀZA,D,16D

#END

#GOƀƀname/version

______________
______________  } Data Cards
______________
______________

#END
```

- Example 2:

  Perform coding to a drum to remote device transfer operation.

```
#JOB__________________

#LASGƀƀCTMID,C

#MFDƀJƀB, __________________

#UTLƀƀname/version

ƀMOVEƀDƀB,C,16D

#END

#GOƀƀname/version

#END
```

- Example 3:

Perform coding to a print operation from drum with a user own code editing routine.

```
#JOB ________________

____ Job steps necessary to create the data file.

════

════

#UTLƀƀname/version

ƀDOƀƀ3,10000

ƀREADƀƀD,10

ƀEDITƀƀD,+2            User routine checks for end of file.

ƀWRITEƀƀZB,26D

#END

#GOƀƀname/version

#END
```

- Example 4:

Coding to perform a card to tape operation with blocking by a factor of five.

```
#JOB _______________

#ASGƀJƀTAPE,T

#UTLƀƀname/version

ƀDOƀƀ3,10000D          Less than 50000D cards to be transferred.

ƀBKLRDƀƀZA,16D,,0,,5

ƀWRITEƀƀT,80D

ƀEOFƀƀT,1              Check for end of input.

#END

#GOƀƀname/version
_____________
_____________
_____________
_____________
_____________
#END
```

8.3.6. Error Messages

The following messages describe the error conditions which may be encountered during the running of the Utility Processor. If a statement contains an error, and execution is attempted, the program exits on an error condition. If an error is fatal, the error is identified as such in the following descriptions. Each error is denoted by the following lead sentence on the high speed printer:

```
***** ERROR CONDITION *****
```

The lead sentence is followed by a description of the error. The description refers to the card which immediately precedes the line giving the error condition. The total number of errors detected during execution of the task is also specified.

- Example of Error Message

```
#JOB   UTL/DEW,477920,C,C,C
#UTL   ABC
 FILE  ZH OPENED
  LINE    LOC.        STATEMENT
   100  002146          RWND A
  ***** ERROR CONDITION  *****  ILLEGAL FILE CODE
 FILE ZH CLOSED
  ***** ERROR CONDITION  *****  00001 ERRORS DETECTED.
 ERROR CONDITION, TASK TERMINATED
 RIR       343500     REL P    015006      LLOCK    343500    ULOCK      366377
 B REG        1 000000  2 002206  3 001144   4 000000  5 000000 6 000000 7012563
 A REG     0000000000     Q REG     0000000000       P-1     7754020503 LMR000000
 # END
```

- Description of Error Diagnostic Numbers

  Errors of type 1, 3 and 5 are printed following the card in error. All other error types appear after the secondary control statements.

Number	Type of Error
1	ABOVE CARD IN ERROR
2	NOT FOUND
3	DO LOOP, NESTING BAD
4	DO LOOP, GOTO, OR TEST JUMP EXTENDS PAST END OF SOURCE STATEMENTS (fatal error)
5	SYMBOL TABLE OVERFLOW (fatal error)
6	DRUM ERROR (fatal error)
7	PROGRAM EXCEEDS 77700_8 LOCATION LIMIT (fatal error)
8	xxxxx ERRORS DETECTED (xxxxx represents the octal number of errors)

Error Number	Examples	Description of Error
1	READ 1, 100	Illegal file code
1	WRITE @A, 1	Illegal option
1	DO –3, 10	Backwards DO loop
1	DO 2, –3	Negative repeat count
1	EOF FCA	File code too long
1	NAME1234567 B,2	Name too long
1	PRM 12	Missing file code
1	READ A,+,5	Illegal subfield
2	TESTxxx A	Undefined subroutines
3,1	DO 3,3 DO 4,2	Out of range of outer loop
	DO 3,3 DO 2,8	Same, terminal statement
4	GOTO 1000	Source statement exceeded on DO' GO TO, or TEST JUMP (fatal error)
5	Maximum number of file codes and user own code subroutines may not exceed 200_{10}	Symbol table overflow (fatal error)
6		Drum error
7		Program exceeds 77700_8 location limit (fatal error)
8	xxxxx	Number of errors detected in octal (represented by xxxxx)

9. REMOTE DEVICE CONTROL

9.1. GENERAL

OMEGA Provides a flexible environment for activating and controlling the flow of remote communications data to and/or from the UNIVAC 494. The environment is structured such that the following tasks may utilize remote communications facilities concurrently in a multiprogram system.

- OMEGA Scheduling Elements

  Job control streams originating from a remote site will be accepted and processed by OMEGA in the normal manner. Subsequent primary and secondary output streams from the scheduled job are automatically transmitted to the remote user, providing the remote-site user with the same scheduling abilities as those provided to an on-site user.

- Batch Programs

  Batch programs are allowed to utilize one or more remote devices for input/output. This is provided at a level which allows the program use of simple acquire, read and write service requests, with OMEGA performing the required buffering, conversion and transmission of data.

- Control Program

  OMEGA provides an interface whereby remote data based control programs can operate without impact on other remote application programs or other users of remote-site data. The type of application tasks assumed here are message switching, transaction control, airline reservations, and such operations which require a control program for the loading, activiation and termination of small data dependent worker routines. (The called worker program is dependent upon the content of the message.)

- Real Time Program

  A real time program with abnormally high turnaround constraints is allowed to co-exist with other users of communications data.

9.1.1. Device Control Elements

To provide a system which satisfies all of the needs of remote devices, OMEGA has a remote communications interface which functions in much the same manner as that provided for on-site devices. Essentially this interface is composed of Externally Specified Index (ESI) channel control, remote line handlers, the communications director and remote facility assignment. (See Figure 9–1 which illustrates the logical flow of messages to and from the system through the various elements.)

- ESI Channel Control

The ESI channel control is composed of the basic OMEGA elements which control the physical hardware channel. The elements are responsible for sending functions to the communications subsystem as directed by a remote line handler, allocating and switching buffers upon interrupts, and activating remote line handlers on predetermined interrupts.

- Remote Line Handlers

The remote line handlers operate as the interface between the ESI channel control and the user task or communications director. Each handler is responsible for directing hardware control of communications lines, and for accepting and transmitting resultant data. Remote line handlers are normally written to control a particular type of remote unit, and, in general, to perform the following functions in one form or another:

– Initiate input transmission through polling techniques or establishing input data buffers, and acknowledge receipt of input data to the remote device.

– Perform, or request the performance of, required process functions to convert, pack into message staging buffers, and detect end-of-message (EOM) status on all incoming communications data.

– Submit messages or message segments to the communications director or user task.

– Accept messages for output transmission to a remote site from the communications director or user. The handler unpacks and converts data contained in staging buffers to communications buffers, and submits communications buffers contents to ESI channel control for transmission.

Univac will supply remote UNIVAC 1004, UNIVAC 9200/9300, DCT 2000, UNISCOPE 100/DCT 1000, DCT 500, and UNISCOPE 300 handlers. User handlers will be written for special purpose devices or to control remote devices in a manner prescribed by a particular installation. Remote handlers can be written as reentrant or nonreentrant, dependent upon specific requirements as dictated by the type of remote unit and communication control procedure defined by the installation.

- Communications Director

The communications director is responsible for providing a high level interface between multiple user tasks and their assigned remote devices. The functions of the communications director are as follows:

– Accept messages from remote line handlers, stage the messages to direct access storage; and, on READM service request, transfer messages to the user task, from a communications line assigned to the task. Own code options are available as data is transferred from handler to direct access storage to a user task.

– Accept messages from the user task on WRITEM service request; stage the messages to direct access storage and activate and/or transfer to the remote line handler messages which will be transmitted across communications lines assigned to a task. Own code options are available as messages flow from the user to direct access storage and from direct access storage to the remote line handler.

- Remote Facility Assignment

Remote facility assignment elements of OMEGA direct the assignment and status of remote devices devoted to a user task. The direction is exercised through control of communications terminals. Assignment of communications terminals is initiated when the user task submits the line assign (LASG) statement for output operation or the line acquire (LACQ) service request for input operations.

Submission of the LASG statement causes the following actions:

- dedication of a communications terminal;
- formation of tables used in the interface;
- establishment of the link between the line handler and the communications terminal; and
- establishment and linkage of output message queues and output own-code options.

Submission by the user task of the LACQ service request causes establishment of input own-code options and input message queues. When the task wishes to relinquish control of a particular communications terminal, the task must submit a line free (LFREE) service request which delinks and releases all tables and linkages established for the communications terminal. Status changes for the communications terminal are effected in response to console operator type-ins or to internal packets from the remote line handler.

9.1.2. Levels of Interface

As can be seen in Figure 9–1, OMEGA provides the user task with two interface points in the control of his remote environment. For ease of description in subsequent paragraphs of this section, the two interface points will be referred to as Level 1 (interface at ESI control) and Level 2 (interface at communications director).

Independent tasks operating under control of OMEGA may interface at either of these points concurrently when OMEGA remote facility assignment routines and table structures are utilized.

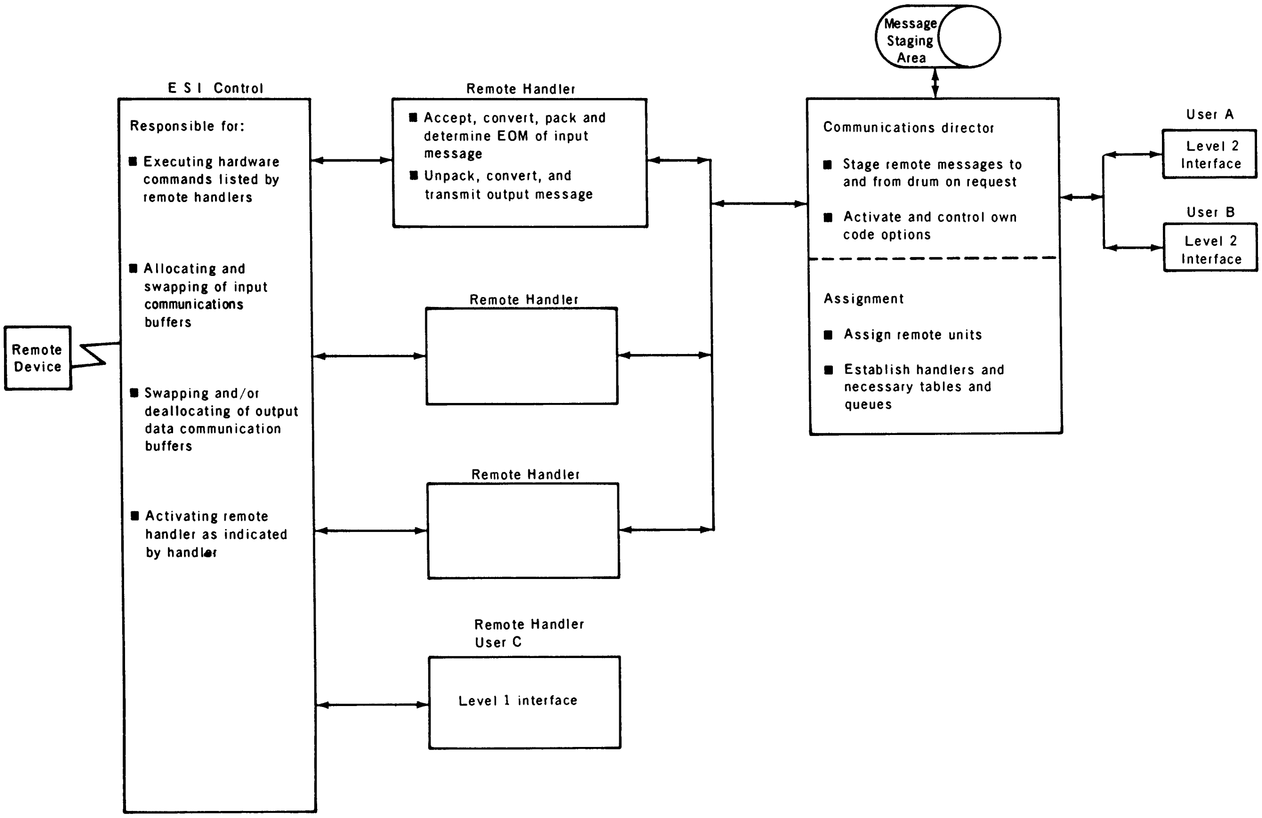

Message Staging Area
E S I Control
Responsible for:
■ Executing hardware commands listed by remote handlers
■ Allocating and swapping of input communications buffers
■ Swapping and/or deallocating of output data communication buffers
■ Activating remote handler as indicated by handler
Remote Device
Remote Handler
■ Accept, convert, pack and determine EOM of input message
■ Unpack, convert, and transmit output message
Remote Handler
Remote Handler
Remote Handler User C
Level 1 interface
Communications director
■ Stage remote messages to and from drum on request
■ Activate and control own code options
Assignment
■ Assign remote units
■ Establish handlers and necessary tables and queues
User A
Level 2 Interface
User B
Level 2 Interface

Figure 9–1. OMEGA Remote Device Control Elements

9.1.2.1. LEVEL 1 INTERFACE

The OMEGA Level 1 interface allows the task to interface directly with the remote line handler. The interface between the user task and the remote line handler is designed and maintained by the installation, permitting the installation to develop its communications environment based upon the specific requirements prescribed by the user's applications. This ability assures the user task of a predictable elapsed period between the time when data enters the machine from communications lines and when the task has control of the data. The remote line handler has complete control of the hardware by commands which are given to ESI channel control. As directed by the handler, ESI channel control performs the actual input/output hardware instructions.

The Level 1 user task or the remote line handler performs all editing, translating, packing, unpacking and staging of message. The remote line handler is responsible for forming output poll messages, monitoring input replies, transmitting output messages, effecting dial connections, and maintaining control of general communications procedures. In installations where Level 1 is used, the user is responsible for developing a system which is compatible with the various OMEGA elements of the Level 1 and Level 2 environments.

The Level 1 interface effectively fulfills the needs of specialised communications tasks such as:

- Real time program with time critical response constraints which cannot be guaranteed through normal interface.
- Applications programs which neither require nor need the generality provided by the communications director and associated routines. This is especially true of installations with one application program using remote data which do not plan to utilize remote scheduling or batch program use of remote devices.

9.1.2.2. LEVEL 2 INTERFACE

The OMEGA Level 2 interface allows the task program to utilize simple READ/WRITE macros to handle communications traffic. The operating system handles the details of message buffering, translating, packing, unpacking, polling and establishing remote connections as needed.

The user may include his own remote line handler for special units or networks and may still use the structure of the Level 2 interface. He may also include optional own code routines which are activated by the system as messages are written or read from direct access storage.

To familiarize the Level 2 user with the interface structure and control of messages by OMEGA, the steps and functions performed in the transfer of a message from a remote unit to user task (INPUT) and from user task to remote device (OUTPUT) are described:

- Step 1

  Task Program (Assign)

  The task program is responsible for requesting the assignment of the line by the LASG statement and the LACQ service requests. Upon submission of these request, OMEGA performs the following functions:

  – Dedicates the Communications Terminal Module (CTM) to the user task.

  – Loads and limits the CTM control block for the specified communications terminal.

  – Loads and activates the remote line handler.

  – Established direct access message, staging queue, and own code routines.

- Step 2

  Remote Line Handler (Activate)

  Upon initial activation, the remote line handler performs the following:

  – Establishes connection with remote units as dictated by line type (dial connection, direct), or enters remote into established polling sequence.

  – Readies the CTM block and submits hardware requests to ESI channel control to activate input transmission, indicating external functions to perform, input message buffers, conventions to follow and when to reactivate the handler.

- Step 3

  ESI Channel Control (In)

  ESI channel control executes channel functions dictated by the remote line handler. Upon input monitor interrupt, ESI channel control performs the following:

  – Allocates and/or switches the input message buffer, dependent upon which option is utilized by the remote line handler.

  – Optionally reactivates the remote line handler.

- Step 4

  Remote Line Handler (In)

  The remote line handler, upon occurrence of one or more input monitor interrupts on communications buffers, is responsible for performing the following:

  – Translates and packs into the staging buffer the message received from the remote device.

  – Determines the status of the communications director and submits messages or message segments through the PUTM service request.

- Step 5

  Communications Director (In)

  The communications director, upon submission of a complete input message from the remote line handler, will:

  – Activate an optional own code routine specified by the user to edit or divert input message.

  – Record messages on the direct access input staging area until requested by the task program.

- Step 6

  Task Program (In)

  The task program requests the transfer of messages from the staging area to the program with the READM service requests using the appropriate file code. The communications director, upon receiving the READM service requests, will transfer a message from the input staging area on a FIFO order and will activate the optional own code routine specified by the user. Upon completion of own code processing, program control is returned to the task program.

  The task program, in turn, processes messages and may compose an output message, in which case the message and header are formed by the user task and are submitted to the communications director for staging with CTM identification. The message is submitted to the communications director by the WRITEM service request.

- Step 7

  Communications Director (Out)

  The communications director, upon submission of an output message from the task through the WRITEM service request, performs the following:

  – Activates an optional own code routine specified by the user to edit or divert output messages.

  – Records messages in the direct access output staging area until requested for transmission by the remote device handler.

  – An own code option is also supplied as the output message is transferred from direct access storage to the remote line handler.

- Step 8

  Remote Line Handler (Out)

  The remote line handler will, on output, request messages from the communications director, translate and unpack them into message buffers, and request ESI channel control to begin transmission. Output messages are supplied to the remote line handler by the communications terminal from a FIFO queue.

- Step 9

  ESI Channel Control (Out)

  On output, ESI chnnel control executes hardware commands as indicated by the remote line handler, swaps and optionally deallocates output communications buffers on monitor interrupt, and reactivates the remote line handler when output transmission is complete.

9.1.3. Summary of Service Requests and Control Statements

The following list contains all service requests and control statements available to the user of remote device control elements in OMEGA. The summary is composed of the mnemonics used to recognize the statement or service request. Brief descriptions of the requests are given, with references to the sections containing detailed explanations and indications of whether the functions are for Level 1 or Level 2 users.

Mnemonic	Description	Level 1 or Level 2	Section
LASG	Assign communications terminal to user task; load remote line handler; load own code output routines and form output direct access staging queue.	1 and 2	9.2.1
LACQ	Load and/or establish input own code routines. Form input direct access staging queue.	2	9.2.2
LFREE	Release file code and associated input queue assigned to user task. Close output queues associated with file code.	2	9.2.3
CTMFREE	Release communications terminal and associated output queue. Release handler if inactive.	1 and 2	9.2.4.1
Console Type-in	Update remote facility status.	1 and 2	9.2.4.2
LUP	Update remote facility status to reflect that facility is in an 'up' state.	1 and 2	9.2.4.2
LDOWN	Update remote facility status to reflect that facility is in a 'down' state.	1 and 2	9.2.4.2
READM	Requests the next complete message contained in users input queue.	2	9.3.2.1
READMW	Same as READM except that control will not be returned until a message is available.	2	9.3.2.1
LOOKM	Same as READM except that the message will not be removed from the input queue until requested by a READM or READMW request.	2	9.3.2.1
WRITEM	Submit message to output queue for transmission.	2	9.3.2.2
PUTM	Used by remote line handler to submit message to input queue.	2	9.3.3.2
PUTMP	Used by the remote line handler to resubmit an input message to the input queue upon recognition of a transmission error. Used only when partial message queuing is being used and the beginning of the complete message has been queued previously.	2	9.3.3.3
GETM	Used by remote line handler to request message from output queue for transmission.	2	9.3.3.1
GETMW	Same as GETM except that control will not be returned until a message is available.	2	9.3.3.1

9.2. REMOTE FACILITY ASSIGNMENT

The remote facility assignment elements of OMEGA perform the necessary functions for assigning remote units to a task and for forming the tables and linkages required by the task to access its assigned units. Remote facility assignment's explicit functions are:

- Maintain status and avaiability of remote units and/or lines. Remote facility assignment maintains a map of all units which are eligible for direct access to the UNIVAC 494, together with all dial type lines. As units or lines are assigned to tasks or marked as inoperable, the map is changed to reflect their current status. This provides the user with the ability to load and activate multiple tasks which utilize remote devices, with OMEGA applying the same task selection rules in regard to remote units as those provided for on-site peripheral units.

- Assign remote units or lines to a task by communications terminal. Remote facility assignment dedicates CTMs and establishes necessary linkage to the assigned line prior to activation or during initialization of the requesting task.

- Load into primary storage and register the remote line handler required for controlling the assigned communications terminal.

- For users of Level 2 interface, remote facility assignment establishes direct access storage staging queues to contain messages prior to processing (input) or prior to transmission (output) and loads and establishes linkages for user own code routines to edit or divert messages prior to processing or transmission.

In general, the user is responsible for performing the following functions required to ready a unit or line for access. Further details are in subsequent sections of this manual.

- Systems Generation Time

  The user participates in forming the remote facility map used to list all units eligible for access by the system (see 7.2). Developed and/or included in the systems library are all remote line handlers and own code routines required by the installation.

- Task Selection Time

  The user submits LASG statements required to assign the communications terminal, establish the remote line handler and output own code routines.

- Task Initialization

  Users of Level 2 interface, during the initialization phase, submit the LACQ service request to activate the handler, establish the input queue and specify optional input own code routines.

- Task Termination

  When the user task has no further need for the communications terminal, the user releases the line through the LFREE statement for use by the system or other tasks.

- Console Operator

  The operator participates by marking down, for OMEGA, the units or lines which are inoperable. For lines without automatic answering or dial connection, the console operator will perform necessary program calls and dial functions.

9.2.1. The LASG Statement

The line assign (LASG) statement is used to connect a communications terminal to a user task and to specify the remote line handler to be used in controlling the communication line associated with the communications terminal. The LASG statement is required for each device assignment.

Before the assigned unit can be activated to receive messages in a Level 2 environment, the acquire (LACQ) service request must be performed by the task to establish input staging queues and input own code options.

- Format:

The LASG statement has the following form:

#LASGƀoptionsƀperipheral code, file code, name/version of handler, priority, name/version of output own code (user task to staging), name/version of output own code (staging to handler)

- Options:

A Register an additional queue process activity addendum for the handler. This will establish multiple control paths within the ESI channel control, remote line handler and communications director structure by communications terminal.

This provides the user with an ability to vary processing priority by communications terminal line and allows the remote line handler to control two or more (dependent upon the number of activity addendum) lines concurrently or in parallel. However, the line handler must be reenetrant, or must utilize test and set procedures if it is registered as two or more activities.

The priority field specifies the operating priority in which the handler should be registered when the A option is used.

H Load and link a copy of the named remote line handler although a copy may currently exist in primary storage. The H option is used in situations where the handler is nonreentrant and controls only one communications terminal at a time.

O The described communications terminal is optional and is not required for the selection or operation of the task. If the unit defined by the peripheral code is not available, the current LASG statement will be ignored.

J Load the remote line handler from the job library.

D If the requested communications terminal is not available, delay the request until the communications terminal is released by the current user, at which time the request will be processed. (For internally submitted LASG statements only.)

M Assign a communications terminal only if the terminal has been updated for maintenance testing. (Maintenance testing.)

X The assignment is for Level 1. Presence of file code and/or own code options are ignored.

- Specifications:

Peripheral Code is a one- to five-character mnemonic used to name the CTM block which is to be assigned. The permissible mnemonics for this field are determined through the names applied by the installation at systems generation time to represent a particular communications terminal line configuration. (See 3.2.2 for more information on definition of peripheral code.)

File code is a one- or two-alphabetic character code used to link the communications terminal assignment to the task. (See 3.2.3 for detailed explanation of file code.) Any number of terminals can be linked to a file code. However, for users of the communications director, normal input messages from all terminals assigned to a particular file code are staged to a common direct access queue unless own code redirects the terminal to another file code. In the case of output, the unit is identified by the CTM/unit identification and the direct access queue is associated with the CTM block rather than the file code.

Name/Version of remote line handler specifies the alphanumeric name and version of the remote line handler responsible for controlling the assigned communications terminal. If the handler is currently not resident in primary storage, or the H option is given, the handler will be located in the system library, loaded into primary storage, registered as a queue process activity and the CTM block will be linked to the handler.

If the specified handler is currently in primary storage and the H option is not given, the CTM block will be linked to the current copy of the handler. In this case, an additional queue process activity addendum will be registered to the handler only when the A option is specified.

Priority is a one- or two-digit field in the range 0 to 17_8, specifying the response priority which should be associated with the handler when registered as an activity. For information on the relationship between response priority and operating priority, see 2.7.1.

NOTE: This does not mean that the handler will not be interrupted for a higher priority. The implication is that the handler will not be rotated for an equal priority.

Because the ESI channel control performs interrupt answering, buffer allocation and swapping, handlers should be based on speed of communications line, i.e. a high speed line uses high priority. This will help to achieve good turnaround of communications and staging buffers, as well as keep lines active.

The priority at which a remote line handler may operate will range from priority 1 through 7. The priority is calculated by subtracting from the service priority of Executive task (normally priority 7) the priority specification on the LASG statement. If the priority specification on the LASG statement is greater than six, the remote line handler will be assigned a priority of 1.

Priority on LASG Statement	Operating Priority
0	7-0 = 7
1	7-1 = 6
2	7-2 = 5
3	7-3 = 4
4	7-4 = 3
5	7-5 = 2
6	7-6 = 1
7	= 1
8	1
9	1

The user must consult the *Uniscope 100 Display Terminal – Communication Control Procedures, UP-7779.1* for additional information.

Name/Version of Output Own Code (user task to staging - Optional) specifies the own code routine activated by the communications director, as messages flow from user task to direct access storage staging. Name/Version specifies the alphanumeric name/version of the own code routine contained in the system library. The communications director controls the requests to the own code routines in a first in first out order.

The two own code routines specified on the assignment statement are for output. The same routine may be specified for both phases of output and input, and on subsequent assign or acquire requests. OMEGA maintains one entry and one routine for each distinctively named own code. All requests for the routine are directed to the single copy.

- Method of Submission

The line assign statements may be submitted externally through the control stream and/or collected with the task element, or they may be submitted dynamically during execution of the task. All LASG statements contained in the control stream which are to be taken into consideration during the task selection phase should precede the task activation card to which they pertain.

Any number of LASG statements may be submitted prior to and during execution of the task. Each may request the same or a different peripheral code and handler from that previously assigned to the task or file code. A file code has only one FIFO input direct access queue, but may have any number of communications terminals feeding messages to it.

LASG statements submitted internally during execution of the task are normally used to allow shared use of communications lines between independent tasks or in situations where select units are used only during given time periods of the day. In cases where the requested communications terminal is not available, the user may use the D option, or the task must be prepared for return of nonavailability status.

Upon return of program control from an internal LASG service request, the following status information is conveyed to the user task through the A register:

Normal completion: The register is set to 00000CTMID, where CTMID is the identifier of the communications terminal line assigned.

Abnormal completion: No assignment is made and the A register contains one of the following status codes:

4100000000	Inappropriate function: the peripheral code for the required unit is not contained in the configuration; or the requested handler or the own code routine was not contained in the system library.
4200000000	Incorrect parameter: the LASG statement could not be interpreted by the system.
4300000000	Systems error caused by inability to read from system library or remote facility map; or a central processor contingency error occurred during processing of statement.
5000000000	Requested unit or type of unit is not concurrently available. All such units are assigned and/or are currently down. When the D option is given, this status code indicates that all such units are currently down.

■ Method of Operation

Remote facility assignment performs the following functions upon submission of the LASG statement either externally or internally during task execution.

– Locates the available associated CTM control block to satisfy the peripheral code named on the LASG statement. If the units are available and in an up status, the CTM control block is loaded with all associated unit control blocks (UCB) into primary storage, the file code is assigned, and the communications terminal is linked to the ESI control table.

  If all such units are in a down state, or are currently assigned and the D option is not specified, the status code 5000000000 is returned to the requester. If all such units are currently assigned and the D option is specified, the request is held in abeyance until the required terminal is released by the current user task or the requesting task is terminated.

– Loads and/or registers the remote line handler. The specified remote line handler is located and loaded into primary storage if not currently in primary storage or the H option is specified. Handlers loaded into primary storage are automatically registered in OMEGA's Remote Line Handler Register, which specifies the name/version of the handler and links the handler to the chain of the CTM control blocks currently controlled by the handler.

– Registers the handler as an OMEGA queue process activity at the specified priority if copy is loaded or the A option is specified. Links the assigned CTM block to the handler through the Remote Line Handler Register. Submits a QREF request to the handler if the X option is specified on the statement.

– Loads and/or registers output own code routine with OMEGA's own code register for users of Level 2 interface.

Remote facility assignment returns program control to the requester when the LASG statement is submitted externally in the control stream or when the statement is submitted internally to the task activity during task execution.

■ Error Messages

The following error messages may appear in the primary output stream as a result of processing LASG statements. In all cases, an image of the LASG statement precedes the diagnostic message.

STATEMENT NOT INTERPRETABLE

The format or specifications cannot be interpreted by the system. 4200000000 status code is returned to requester.

ILLEGAL PERIPHERAL CODE

The specified peripheral code is not contained in the remote facility assignments map. An error was made on submission of the peripheral code, or systems generation did not include the entry. 4200000000 status code is returned to requester.

INAPPROPRIATE FILE CODE

The specified file code is not included in the eligible set or was previously assigned to an onsite device. 4200000000 status code is returned to the requester.

NAMED REMOTE LINE HANDLER CANNOT BE LOCATED

The remote line handler specified on the LASG statement cannot be located in the system library. An error was made in the name/version specification, or the remote line handler was not included in the system library at library generation time. 4100000000 status code is returned to the requester.

INCORRECT PRIORITY SPECIFICATION

The specified priority is not within the accepted range of 0 to 17_8. 4200000000 status code is returned to the requester.

NAMED OWN CODE ROUTINE CANNOT BE LOCATED

The own code routine specified on the LASG statement cannot be located in the system library. An error was made in the name/version specification, or the own code routine was not included in the system library at library generation time. 4100000000 status code is returned to the requester.

REQUESTED CTM IS NOT AVAILABLE

All communications terminal which are eligible for the specified peripheral code are currently down or assigned to other tasks. This message is submitted only when the D option is not given. 5000000000 status code is returned to the requester.

REQUESTED CTM IS IN A DOWN STATE

All communications terminals which are eligible for the specified peripheral code are currently down and inoperable. 5000000000 status code is returned to the requester.

REMOTE FACILITY ASSIGNMENT MAP ERROR

The remote facility assignment map was not generated properly (regenerate map). 4300000000 status code is returned to the requester.

SYSTEM ERROR

The facility map or routine cannot be retrieved from the system library, or a central processor contingency has occurred during facility assignment processing. 4300000000 status code is returned to the requester.

9.2.2. The LACQ Service Request

The line acquire (LACQ) service request is used by the Level 2 interface task to establish the input direct access storage staging queue and input own code routine. The service request is normally submitted during the initialization phase of the user task and prior to the input message access by the operating task.

The line acquire operator is:

LACQ$ƀfc,name/version of input own code (handler to staging), name/version of input own code (staging to user)

fc is the file code used to link the input side of the assigned communications terminal to the task addendum. File code is the one- or two-alphabetic character code specified on all LASG statements submitted which are to use established input queues.

Name/Version of Input Own Code (handler to staging – Optional) specifies the own code routine activated by the communications director as messages flow from the handler to direct access storage. Name/Version specifies the alphanumeric name and version of the own code routine contained in the system library. The communications director controls requests to the own code routines in a first in, first out order.

Name/Version of Input Own Code (staging to user task – Optional) specifies the own code routine activated by the communications director as messages flow from direct access storage to the user task. Name/Version specifies the alphanumeric name and version of the own code routine contained in the systems library. The communications director controls requests to the own code routines in a first in, first out order.

The two own code routines specified in the acquire service request are for input. The same routine may be specified for both phases of input and output, and on subsequent assign or acquire requests. OMEGA maintains one entry and one routine for each distinctively named own code. All requests for the routine are directed to the single copy.

The generated packet is:

	EBJP*B7	N
		FC
	NAME-HANDLER TO STAGING VERSION	
	NAME-STAGING TO USER TASK VERSION	
N→	EXRN	21202

Left justified
in Fieldata form

NOTE: The address contained in B7 upon submission of the request is assumed relative to the lower lock of the requesting activity

- Method of Operation

Remote facility assignment performs the following functions upon submission of the LACQ service request:

– Forms the input direct access storage queue descriptor and links the descriptor to the indicated file code.

– Locates and loads and/or registers the input own code routine with OMEGA's own code register.

– Submits a QREF service request to the specified remote line handler indicating that the communications terminal is assigned.

– Returns program control to the requesting activity.

Upon return of program control to the requester, registers B1 through B6 contain the original values held prior to request. Register B7 contains the address of the LACQ packet. The A register contains one of the following status indication:

Normal Completion: The register is set to 0000000XXX, where XXX is a binary number indicating the number of the CTM blocks interfaced to the established queue at acquisition time (the number of LASG statements successfully processed).

Abnormal Completion: No processing is performed, and the A register contains one of the following:

4100000000	Inappropriate function: the file code cannot be used because of prior assignment of an onsite device, or the own code routine is not contained in the systems library.
4200000000	Incorrect parameter: the LACQ service request could not be interpreted by the system.
4300000000	Systems error: caused by inability to read from the systems library or by a central processor contingency error.

- Error Messages

The following error messages may appear in the primary output stream as a result of processing the LACQ service request. In all error cases, an image of the LACQ service request will precede the diagnostic message:

SERVICE REQUEST NOT INTERPRETABLE

The format or specifications cannot be interpreted by the system. 4200000000 status code is returned to the requester.

INAPPROPRIATE FILE CODE

The specified file code is not included in the eligible set or was previously assigned to an onsite device. 4200000000 status code is returned to the requester.

NAMED OWN CODE ROUTINE CANNOT BE LOCATED

The own code routine specified on the LACQ request cannot be found in the system library. An error has been made in the name/version specification, or the own code routine was not included in the system library at library generation time. 4100000000 status code is returned to the requester.

SYSTEMS ERROR

The system library cannot be accessed due to subsystem errors, or a central processor contingency error has occurred during processing. 4300000000 status code is returned to the requester.

9.2.3. The LFREE Statement

The line free (LFREE) statement is used to release input staging queues and submit output stream closures to all CTM output queues associated with the file code specified.

- Format:

The LFREE statement has the following form:

#LFREEb̸b̸file code

- Specifications:

File Code is the one- or two-alphabetic character code used to release input queues established by the #LACQ statement.

- Method of Submission

The LFREE statement may be submitted externally through the control stream to be processed upon termination of the task or may be submitted dynamically during task execution.

LFREE statements submitted internally during task execution are normally used to allow shared use or communications lines between independent tasks or in situations where select units are used only during given time periods of the day.

- Method of Operation

Remote facility assignment performs the following upon submission of the LFREE statement either externally or internally:

- Releases the file code link and associated input queue descriptor.
- Submits a call to the communications director, closing the output queues for all communications terminals linked to the specified file code.
- Returns program control to the requester.

Upon return of program control to requester, registers B1 through B6 contain the original values held prior to the request. Register B7 contains the address of the LFREE packet. The A register contains one of the following status indications:

Normal Completion: The register is set to 0000000000.

Abnormal Completion: No release of function was performed, and the register contains one of the following:

4100000000	Inappropriate function: no communications device assignment exists for this file code.
4200000000	Incorrect parameter: the LFREE statement could not be unstrung or interpreted properly.
4300000000	Systems error.

- Error Messages

The following error messages may appear in the primary output stream as a result of processing LFREE statements. In all cases, an image of the LFREE statement precedes the diagnostic message:

FILE CODE NOT ASSIGNED

The specified file code was not assigned to a communications device. 4100000000 status code is returned to the requester.

STATEMENT NOT INTERPRETABLE

The format or specifications cannot be interpreted by the system. 4200000000 status code is returned to the requester.

SYSTEM ERROR

A system error occurred during processing of the LFREE statement. 4300000000 status code is returned to the requester.

9.2.4. Miscellaneous Remote Facility Assignment Requests

The following is a description of miscellaneous service requests provided by remote facility assignment.

9.2.4.1. THE CTMFREE SERVICE REQUEST

The CTMFREE service request is used by the remote line handler to deallocate the CTM control block and the associated output queues.

- Format:

The CTMFREE request has the following form:

CTMFREE$ƀCTMID

The internal packet generated is:

ENT*B7	CTMID
EXRN	21204

- Specifications:

CTMID is the unique numeric line identification associated with each CTM control block.

- Method of Operation

Remote facility assignment performs the following upon submission of the CTMFREE Service request:

– Locates the CTM control block having the specified CTMID, delinks the associated output queue from the output queue chain, releases the CTM block primary storage and updates the CTM usage list.

– Returns control to the requester.

Upon return of program control to the requester, the following status information is returned through the A register.

Normal Completion: the register is set to 0000000000.

Abnormal Completion: no processing is performed and the register contains one of the following:

4100000000	Inappropriate function: the CTM with the specified CTMID was not currently in use.
4300000000	Systems error, caused by inability to read from the systems library or by a central processor contingency error.

9.2.4.2. REMOTE FACILITY UPDATE

Remote facility update allows the console operator and/or remote line handler to mark remote lines and/or units up or down and allows the console operator to reserve lines for maintanance testing.

- Format:

The format of the operator type-in for remote facility update is as follows:

RFƀCTMID/UNITƀFunction ⓢ

The internal packet generated is:

Mark a line and/or units up

ENT*B7	CTMID/UNIT
EXRN	21206

Mark a line and/or units down

ENT*B7	CTMID/UNIT
EXRN	21207

- Specifications:

CTMID is the unique nine bit logical numeric line identification associated with each communications line.

UNIT is a number $\leq 64_{10}$, specifying the unit within the CTM that will be updated. If the unit specification is 0 or is not given, the entire CTM will be updated.

Function specifies the update operation to be performed. Valid functions are:

D Mark as inoperable, not available for assignment.

U Mark as operable.

M Mark as assignable only for maintenance. The LASG statement must have the M option in order to have this CTM assigned. See 9.2.1 for further information on maintenance.

Remote facility update may also be accomplished through the control stream, using the mnemonics LUP and LDOWN. The formats for the requests are:

LUP$ƀCTMID

for the up condition; and

LDOWN$ƀCTMID

for the down condition.

Parameters and specifications for the service requests are the same as for the console type-ins, with the exception that the functions, being themselves the mnemonics, need not be specified after the CTMID.

- Method of Operation

Remote facility assignment performs the following functions upon submission of a remote facility update request:

– Locates and updates the CTM descriptor as determined by the function on the direct access device and in core if present.

– Returns control to the requester for internal packets and to the Content Supervisor for console type-in.

Upon return of program control from an internal remote facility update request, the following status information will be returned to the requester through the A register:

Normal Completion: The register is set to 0000000000.

Abnormal Completion: No processing is performed and the register contains one of the following:

4200000000	Incorrect parameter: the remote facility update request could not be interpreted by the system.
4300000000	System error: caused by inability to read from the system library or by a central processor contingency error.

9.3. REMOTE DATA ACCESS SERVICE REQUESTS

These paragraphs describe in detail the OMEGA services provided to the users of Level 2 interface for transferring messages to and from the UNIVAC 494. Service requests included are those used by the task program, remote line handler and own code routines. See Figure 9–3, which illustrates the direction of message flow upon submission of the various service requests executed by the task program and remote line handler.

9.3.1. Message Queue Control

As can be seen in Figure 9–3, normally all messages accepted by or directed to the remote line handler or task elements are staged on direct access storage until the messages are eligible for processing by the task or handler. The read/write service requests used by the task and the get/put service requests used by the remote line handler are, in effect, requesting the transfer of messages to or from direct access storage, as opposed to direct transfer of a message from the task to remote equipment or from remote equipment to the task.

An exception to this rule arises when the task or handler is waiting on an empty direct access queue (READMW or GETMW). When this occurs, an incoming message will be transferred directly to the task or the handler and will bypass the storage process.

The use of direct access storage for message staging allows the system to balance the intermittent task utilization of messages having the normally slow transfer rate of the communications line. The remote line handler can operate communications lines to their full capacity without encountering intermittent delays due to the task program waits. The task program can process messages at the rate dictated by the central processor and its online subsystems, allowing message data to collect on direct access storage until the remote line can accept it.

A second feature provided by staging is the ability to establish multiple input/output message queues. Each queue is established for a particular line on output or for a task file code on input. The messages contained in each queue are threaded together by chaining techniques within a common pool of direct access storage. Each message queue or chain is described by a table referred to as the "direct access queue descriptor" (see Figure 9–2). The descriptor points to the head and tail link of the message queue, and contains values pertinent to the control of message flow. Messages are linked or unlinked from the chain upon execution of service requests in a first in, first out order.

The input message queue (remote device to task) is established by the LACQ service request and linked to one of the normal task addendum file codes. Upon service requests being directed to the input queue, the file code is specified as one of the parameters; e.g., when the task program is requesting a message from the head of the input chain (READM), as when the remote line handler is submitting a message to the tail of the input chain (PUTM), the file code is located in the CTM block.

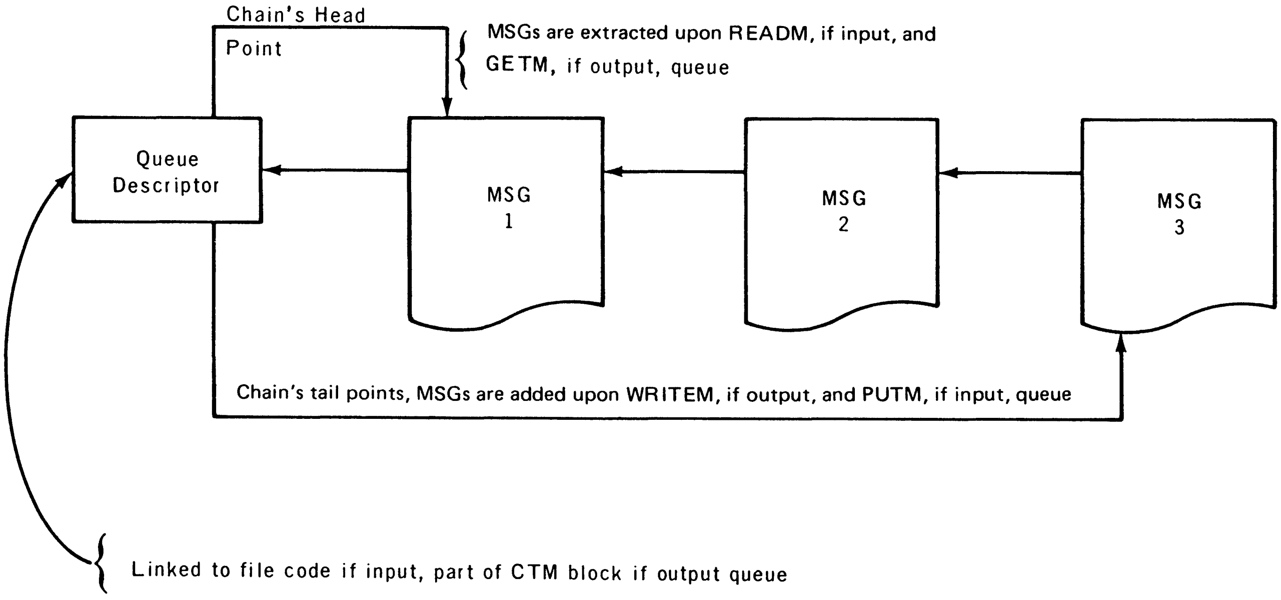

Chain's Head
Point
MSGs are extracted upon READM, if input, and
GETM, if output, queue
Queue
Descriptor
MSG
1
MSG
2
MSG
3
Chain's tail points, MSGs are added upon WRITEM, if output, and PUTM, if input, queue
Linked to file code if input, part of CTM block if output queue

Figure 9–2. Direct Access Queue

The output message queue (task to remote device) is established by the LASG statement and is part of the CTM control block used to monitor a line pair. Upon service requests being directed to the output queue, the CTMID is specified as one of the parameters; e.g., when the remote line handler requests a message from the head of the output chain (GETM), or when the task program submits a message to the tail of the output chain (WRITEM), the CTMID is located in the message header.

Figure 9–3 illustrates the direction of message flow between elements of the system upon submission of READM, WRITEM, PUTM and GETM service requests.

9.3.2. Read/Write Service Requests

The following read/write service requests are provided to the task utilizing the Level 2 interface for the transfer of messages to or from the input/output message queues. The requesting task prior to submission of read/write requests, is required to have previously assigned the line pair and message queue through the LASG statements and LACQ requests.

9.3.2.1. READ MESSAGE (READM$, READMW$ AND LOOKM$)

The read message macros request that the next message contained in the specified input queue be transferred from direct access storage to the task program. The own code routine specified by the LACQ request is automatically activated upon completion of message transfer from direct access storage to primary storage, but prior to return of program control to the requesting task activity.

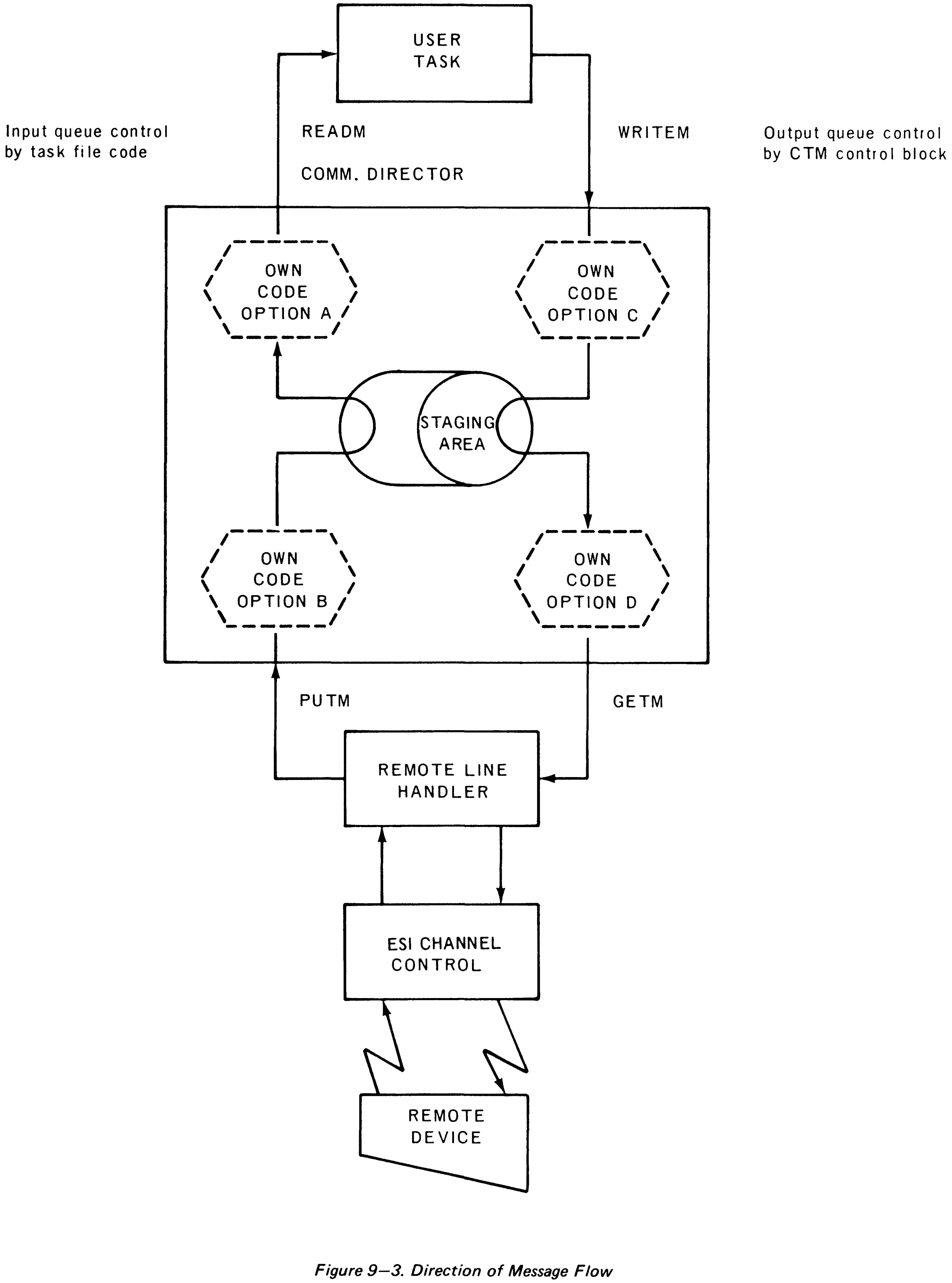


Figure 9–3. Direction of Message Flow

Three read service requests are provided by OMEGA: READM, read; READMW, read and wait; and LOOKM, look ahead. Differences between these requests are: the READM request causes the return of an end-of-file status if there are no messages currently on the queue, while READMW delays the requesting activity until a message comes in. LOOKM acts in the same manner as READM in transferring the next message contained in the specific input queue from Direct Access storage to the task program, with the exception that the message is not removed from the input queue; i.e., the subsequent READM request will transfer the same message.

■ READM$

– Format:

The read service request macro is as follows:

READM$ƀfc, number of words, buffer base

– Specifications:

fc is a one- or two-alphabetic character file code established by the LACQ service request and used to link the read request to the input queue.

Number of words specifies the length of the task primary storage area reserved as a buffer to contain the complete or pattially complete message being transferred. The value may be specified by either an octal or decimal constant, or by a tag which is equated to an octal or decimal constant through an EQUALS statement.

Buffer base specifies the base address of the buffer, relative to the requesting activity's lower lock, used to contain the complete or partially complete message. This address may be specified either as a label, plus or minus an increment, or as an EDEF to be satisifed at collection time.

The generated service request is:

EBJP*B7	N
FC	NUMBER OF WORDS
	BUFFER BASE
EXRN	3 0 0 2 1

■ READMW$

– Format:

The read and wait service request macro is:

READMW$ƀfc, number of words, buffer base

Specifications and generated service request for the READMW macro and the READM macro are identical, with the exception of the Y operand of the EXRN instruction. In the case of READMW, the Y operand is 30022.

- LOOKM$

The look ahead service request macro is:

LOOKM$b̸fc, number of words, buffer base

Specifications and generated service request for the LOOKM macro and the READM macro are identical with the exception of the Y operand of the EXRN instruction. In the case of LOOKM, the Y operand is 30027.

- Method of Operation

Upon submission of the read message service request, OMEGA performs the following functions:

– Interprets the packet addressed by register B7 and locates the input queue. If the input queue is currently empty, and the READM or LOOKM request was executed, an end-of-file status code is returned. If the input queue is currently empty, and the READMW request was executed, the activity will be PUSHed on a unit queue until the remote line handler submits a message for the requested input queue; at this time, the requesting activity will be POPed and the request will be completed.

– Transfers a message to the task primary storage area specified in the packet by the buffer base and number of words. If the complete message cannot be contained in the specified buffer because of insufficient length, a partial message (message segment) will be transferred. The length of the message segment will be the number of words specified as the buffer length. Subsequent read operations by the task will retrieve residue segments of the incomplete message. Appropriate indicators and status codes will be submitted on each request to indicate incomplete message transfers. Each message segment will specify the number of characters in the complete message.

– Activates the user-supplied own code routine, if specified, to reformat or to perform further editing.

– Returns program control to the instruction following EXRN under the requesting activity's addendum.

Upon return of program control to the requesting activity, registers B1 through B6 contain the original values held prior to the request. Register B7 contains the address of the read packet. If the message has been successfully transferred, the A register is set to 0; and the requester's buffer is filled as described in 9.3.4. This does not preclude the possibility of an error within the message. The status bits of the message itself must be checked (see 9.3.4).

If the transfer of the message from direct access storage to the task was abnormally completed, one of the following status codes is returned in the A register.

4200000000 Incorrect parameter: an error has been detected in the specification of the operation (e.g. buffer is outside program lock register limits; or there is an undefined function code). The operation has not been performed.

4400000000 Returned on a READM or LOOKM request, indicating that the input message queue is currently empty, but is potentially eligible to receive messages; i.e. CTM blocks are currently assigned to the queue.

4500000000 End-of-input signal returned on READM, READMW and LOOKM requests, indicating that the input message queue is empty and all CTMs associated with the queue have been released or transferred.

5000000000 No assignment made: the file code referenced has not been assigned through the LACQ service request, or has been assigned to an on-site device.

9.3.2.2. WRITE MESSAGE (WRITEM$)

The write message macro requests the transfer of a message from the specified task's primary storage buffer to the indicated direct access storage queue. The direct access message queue is associated with the CTM control block. The own code option specified on the LASG statement is activiated by the operating system prior to the transfer of the message to the queue. Once the message transfer to the output queue is complete, the message is eligible for retrieval and transmission by the remote line handler.

- Format:

  The write service request macro is as follows:

  WRITEM$b̸buffer base, number of words

- Specifications:

  Buffer base specifies the base address of the buffer, relative to the requesting activity's lower lock, containing the message to be submitted and header information as described in 9.3.4, with the following exceptions: the time (word b) and the message number (word c – lower) may be left blank or filled by the user – information for both will be inserted by the communications director. The buffer base may be specified either as a label, plus or minus an increment, or as an EDEF which will be satisfied at collection time.

  Number of words specifies the length of the message and the header information, and may be specified as an octal or decimal constant, or may be specified by a tag which is equated to an octal or decimal constant by an EQUALS statement.

  The generated service request is:

ENT*B7	BUFFER BASE
ENT * Q	NUMBER OF WORDS
EXRN	30023

- Method of Operation

  Upon submission of the write message service request, the operating system performs the following functions:

  – Interprets the header addressed by B7 and locates the CTM control block.

  – Activates the own code routine if specified at assignment time.

  – Enters the message together with the header in the direct access storage queue associated with the CTM control block.

  – Returns program control to the instruction following EXRN under the requesting activity's addendum.

  Upon return of program control to the requesting activity, registers B1 through B6 contain the original values held prior to the request. Register B7 contains the address of the buffer. If the message was successfully transferred, the A register is set to binary 0. Unrecoverable direct access storage errors are reflected in the status bits of the message for use by the remote line handler.

If the requested transfer was not successfully completed, one of the following status codes is returned in the A register:

4200000000 Incorrect parameter: an error has been detected in the specification of the operation (e.g., the buffer is outside the program lock register limits; or there is an undefined function code). The requested operation has not been performed.

5000000000 No assignment made: the specified CTM is not assigned to the requesting task, or there has been a program error.

9.3.3. Get/Put Service Requests

The following get/put service requests are provided to the handler as part of the Level 2 interface for the transfer of messages from the output queues and to the input queues.

9.3.3.1. GET MESSAGE (GETM$ AND GETMW$)

The get message macros request the next message contained in the specified output queue to be transferred from direct access storage to the remote line handler. The own code routine specified by the LASG statement is automatically activated upon completion of the message transfer from direct access storage to primary storage, but prior to return of control to the handler.

Two get service requests are provided by OMEGA: GETM, get; and GETMW, get and wait. The GETM request causes the return of an end-of-file status if there are no messages currently on the queue, while GETMW delays the requesting handler until a message comes in.

- Format:

  The get service request macro is as follows:

  GETM$ƀCTMID, number of words, buffer base

- Specifications:

  CTMID is a 15-bit identifier established on the LASG statement to link a handler to a CTM block.

  Number of words specifies the length of task primary storage area reserved as a buffer to contain the complete or partially complete message being transferred. The value may be specified by either an octal or decimal constant or by a tag which is equated to an octal or decimal constant through an EQUALS statement.

  Buffer base specifies the base address of the buffer, relative to the lower lock of the requesting activity, used to contain the complete or partially complete message. This address may be specified either as a label, plus or minus an increment, or as an EDEF which will be satisfied at collection time.

  The generated service request is:

	EBJP*B7	N
	CTMID	NUMBER OF WORDS
	BUFFER BASE	
N→	EXRN	30025

The get and wait service request macro is:

GETMW$ƀCTMID, number of words, buffer base

The specifications and generated service request for the GETMW and GETM macros are identical with the exception of the Y operand of the EXRN instruction. In the case of GETMW, the Y operand is 30026.

■ Method of Operation

Upon submission of the get message service request, OMEGA performs the following functions:

– Interprets the packet address by B7 and locates the output queue. If the output queue is currently empty and the GETM request was executed, an end-of-file status code is returned. If the output queue is currently empty and the GETMW request was executed, the activity will be PUSHed on a unit queue until the worker task submits a message for the requested output queue; at this time, the requesting activity will be POPed and the request will be completed.

– Transfers a message to the handler primary storage area specified in the packet by the buffer base and number of words. If the complete message cannot be contained in the specified buffer, because of insufficient buffer length, a partial message (message segment) will be transferred. The length of the message segment will be the number of words specified as the buffer length. Subsequent read instructions by the handler will retrieve residue segments of the incomplete message. Appropriate indicators and status codes will be submitted on each request to indicate incomplete message transfers. The number of characters in the header will reflect the total number of characters in the message although the complete message may not be transferred.

– Activates the user supplied own code routine, if specified, to reformat or to perform further editing.

– Returns program control to the instruction following EXRN under the requesting activity's addendum.

Upon return of program control to the requesting activity, registers B1 through B6 contain the original values held prior to the request. Register B7 contains the address of the get packet. If the message was successfully transferred, the A register is set to 0, and the handler's buffer is filled as described in 9.3.4. Direct access storage errors are indicated in the message status word.

If the transfer of message from direct access storage to the handler was abnormally completed, one of the following status codes is returned in the A register.

4200000000 Incorrect parameter: an error has been detected in the specification of the operation; e.g., the buffer is outside the program lock register limits; there is an undefined function code. The operation has not been performed.

4400000000 Returned on a GETM request only: indicates that the input message queue is currently empty, but is potentially eligible to receive a message; i.e., CTM blocks are currently assigned to the queue.

4500000000 End-of-input signal: returned on both GETM and GETMW requests, indicating that the input message queue is empty and has been closed out.

9.3.3.2. PUT MESSAGE (PUTM$)

The put message macro requests the transfer of a message or message segment from the specified handler's primary storage buffer to the indicated direct access storage queue. The direct access message queue is associated with the CTM control block. The own code option specified on the LACQ request is activated by OMEGA prior to the transfer of the message to the queue. Once a complete message has been transferred to the queue, the message is eligible for retrieval and transfer to the worker task.

- Format:

  The put service request macro is as follows:

  PUTM$ƀbuffer base, number of words

- Specification:

  Buffer base specifies the base address of the buffer, relative to the lower lock of the requesting activity, used to contain the message to be submitted and the header information as described in 9.3.4, with the following exceptions: the time (word b) and the message number (word c – lower) may be left blank or be hash filled by the user - information for both will be inserted by the communications director.

  The buffer base may be specified either as a label, plus or minus an increment, or as an EDEF to be satisfied at collection time.

  Number of words specifies the length of the message and the header information, and may be specified as an octal or decimal constant, or may be specified by a tag which is equated to an octal or decimal constant by an EQUALS statement.

  The generated service request is:

ENT*B7	BUFFER BASE
ENT*Q	NUMBER OF WORDS
EXRN	3 0 0 2 4

- Method of Operation

  Upon submission of the put message service request, OMEGA performs the following functions:

  – Interprets the header addressed by B7 and locates the CTM control block.

  – If the message is not complete, deposits the message segment on a direct access storage hold queue and returns to the handler, continuing to do so until the complete message is in the system.

  – Activates the own code routine specified at acquisition time.

  – Enters the message together with the header into the direct access storage input queue associated with the file code contained in the CTM control block, unless directed to use a different input queue by the own code routine.

  – Returns program control to the handler at the instruction following EXRN.

Upon return of program control to the handler, registers B1 through B6 contain the original values held prior to the request. Register B7 contains the address of the buffer. Register A is set to 0, indicating that the message has been received.

9.3.3.3. PUT MESSAGE AFTER PURGE (PUTMP$)

The PUTMP$ variation of the put message macro requests the purging of all segments of the message previously put, and then proceeds as a put message request with the start-of-message segment (see 9.3.3.2). Normally used by remote line handler upon recognition of transmission error in the previous segment.

- Format:

  The put message after purge service request macro is as follows:

  PUTMP$b̸buffer base, number of words

- Specifications:

  Buffer base specifies the base address of the buffer, relative to the lower lock of the requesting activity, used to contain the message to be submitted and the header information as described in 9.3.4, with the following exceptions: the time (word b) and the message number (word c - lower) may be left blank or be hash filled by the user - information for both will be inserted by the communications director. Buffer base may be specified either as a label, plus or minus an increment, or as an EDEF to be satisfied at collection time.

  Number of words specifies the length of the message and the header information, and may be specified as an octal or decimal constant, or may be specified by a tag which is equated to an octal or decimal constant by an EQUALS statement.

  The generated service request is:

ENT*B7	BUFFER BASE
ENT*Q	NO. OF WORDS
EXRN	30033

- Method of Operation

  The function operates in the following manner:

  – time stamp incoming message segment;

  – interpret the header addressed by register B7 and locate the CTM control block and index table;

  – purge all segments already in the index table, deallocate direct access storage modules emptied by purge, and deallocate the index table;

  – continue as a normal PUTM request on a start-of-message segment.

At exit time, this function changes to function code 24 and proceeds through its operations to an exit.

9.3.4. Message Buffer Format

The following buffer format will be used, with the noted exceptions, for messages submitted to the communications director through PUTM and WRITEM requests, and for messages returned from the director on READM and GETM requests.

Word a	CTM (AND UNIT) ID	NUMBER OF CHARACTERS
b	TIME STAMP	
c	STATUS	MSG NUMBER
d	MESSAGE	
⋮		

Word a

The CTM and unit identity (if there are multiple remote units connected to a communications terminal) specifies the remote device which has submitted or which will receive the message or message segment. The identity may be used to direct the message to the proper queue, handler, unit, own code routine, etc.

Number of characters specifies the length of the actual message or message segment contained in the buffer. The value must be a multiple of five unless the message is complete or is the end segment of a message.

Word b

As each message or message segment enters the communications director, the director inserts the time stamp, which is the number of 100-millisecond units elapsed since the first of the year. The time will then be passed on to the user or handler on a READ or GETM request. On a WRITEM or PUTM request, the user or handler must make word b available for this stamp.

Word c

Status information is broken down as follows:

	22	21	20	19	18	17	16	15
UNASSIGNED	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0

15 – Incomplete message status indicator. The bit is set upon input by the handler to indicate that the complete message was not received at the central site. A remote device or line problem is indicated.

16 – Parity error indicator. The bit is set upon input by the handler to indicate one or more parity errors in the message which could not be corrected by retransmission, etc.

17 – Translation error indicator. The bit is set upon input by the handler or own code, or upon output by own code to indicate that an error occurred in code conversion.

18 – Internal error indicator. The bit is set upon input or output by any one of the data handling elements, indicating data loss during transfer of message between elements due to power loss, systems parity error, etc.

19 – Data loss indicator. The bit is set upon input or output by the communications director or handler to indicate that the complete message cannot be retrieved from direct access storage or that data was lost from communications lines.

20 } – Message segment indicator. These bits are set by the user upon output or by the handler upon input
21 } describe the message or message segment, with the codes following:

11 The complete message is contained in the buffer.

10 The leading segment of the message is contained in the buffer.

00 The middle segment of the message is contained in the buffer.

01 The ending segment of the message is contained in the buffer.

22 – Output queue indicator. This bit is set by the user upon output to indicate that a separate queue has been set up for each output unit associated with the specified CTM block, and that the unit ID specifies which output queue will be used for this message.

Message number is a 15-bit binary value assigned by the communications director to each message entering the director. The number resets to 0 when it reaches 77777_8. When a message is submitted to the director in segments, the director will assign the same number to all segments for the message. The message number will be passed on to the user or handler on a READM or GETM request; on a WRITEM or PUTM request, the user or handler must make the lower portion of word c available for this number.

Word d . . .

The message or message segment itself may be on any format depending upon the own code routines, the handler and the worker program, but is normally packed five characters per word in Fieldata.

9.3.5. Own Code Routines

Own code routines are optional elements normally designed and implemented by the installation to provide various functions which are desirable in a sophisticated communications network.

- Own code routines provide an interface point by which the user may edit messages prior to and subsequent to processing by the responsible task. Editing functions may include: reformatting of a message to satisfy the conventions of a particular file; logging the message on an on-site tape unit; or diversion of the message to an alternate queue as determined by the content of the message. The concept of message diversion within the own code routines enhances the flexibility of the system as input messages can be diverted to alternate input queues or to an output queue (message switching). Output to on-site devices may also be accomplished at this point.

- Own code routines may be used for exception processing; e.g., recording output messages on an on-site device due to an inoperable communications line; or aborting or requesting retransmission of incomplete or erroneous messages.

- Own code routines will be specified when an LASG$ and/or LACQ$ service request is submitted to communications facility assignment. Own code routines will be programmed as common subroutines and will be included in the system at system generation. When called by the communications director, an own code routine will operate on the message whose base address is contained in register B7.

Exit from the own code will be made to the communications director, with register B7 containing the address of the message. The A register may contain an indication of the subsequent operation of the communications director with respect to the message. If the A register value is equal to zero, the message is staged or passed to a requester. If the A register value is two, the message is deleted and control is returned to the requester.

9.4. REMOTE SYSTEMS

OMEGA includes remote line handlers for the standard Univac remote devices to provide complete remote systems through which data data and job stream information may enter the system.

When used in conjunction with the OMEGA Level 2 communications environment, these handlers permit the use of simple read/write macros in a user's batch type program for transferring data to and from a remote unit.

The remote systems available with OMEGA, and the means for using them, are discussed in the following sections.

9.4.1. The Remote UNIVAC 1004 and UNIVAC 9200/9300 Systems

OMEGA provides the user with complete remote UNIVAC 1004 and UNIVAC 9200/9300 systems which will enable a user task to transmit and receive messages from one or more remote sites. The system uses those elements which form a Level 2 environment (see 9.1.2.2), combined with the remote line handler and the remote UNIVAC 1004 plugboard program (RMS1) or the UNIVAC 9200/9300 systems Remote Communications program (UP9607). OMEGA scheduling elements will enable a remote user to execute job streams originating at the remote site, thereby providing the remote user with the same capabilities as the central site user.

As the user task will interface with the OMEGA Level 2 communications environment, the task must submit various service requests as defined for the system. Although the service requests are explained in the previous sections, there are instances when the parameters submitted, or results obtained, will be characteristic of the remote system, specifically.

- Remote Facility Assignment

  The user task must submit LASG, LACQ and LFREE requests as defined in section 9.2. When using the remote systems, the name/version of the handler is RMS1LH.

  LASG

  The LASG statement without own code options has the following form:

  #LASGƀoptionsƀperipheral code, file code, RMS1LH, priority

  LACQ

  The line acquire operator without own code options is:

  LACQ$ƀfile code

  The user may request activation of own code routines on the LASG and LACQ statements. An own code routine is not required by the remote system. The remote line handler will convert each input image to Fieldata and will expect each output message to be in Fieldata format.

LFREE

The LFREE statement has the following form:

#LFREEƀƀfile code

- Remote Data Access Service Requests

The service requests READM, READMW and WRITEM used by the task for the transfer of messages to and from the OMEGA system are described in section 9.3.

When submitting the READM or READMW service request, the user task must specify a deposit buffer length of 19 words for receiving a complete image. If the area is less than 19 words, multiple requests will be required for obtaining a complete image.

Number of words for the WRITEM service request is defined as the total length of the buffer including the three words of message header. The number of words in the message text may vary. The remote line handler will insert trailing spaces to expand each message to a complete print or punch image.

If the user task receives a 'no assignment' status (5000000000) in response to a WRITEM service request, the indication is that the remote line handler has encountered an unrecoverable error and has submitted a CTMFREE service request.

- Message Buffer Format

Message buffer format will be as specified in section 9.3.4, with the following exceptions:

– When a WRITEM service request is submitted, the unit ID in the upper portion of word a will define the remote output media. A unit ID of 01 indicates that the image is to be printed. A unit ID of 02 indicates that the image is to be punched.

– The number of characters in the lower portion of word a will be set to 80 when a complete message is requested by the user task through a read request.

When the user task submits a message through the WRITEM request, the message length may be specified as less than or equal to 132 characters for each print request, and less than or equal to 80 characters for each punch request. The 1004 remote line handler will expand the message to a complete image if the user specifies a number less than a full image.

– If the image is to be printed, the upper seven bits of word c will contain page control information, as follows:

Bits	
29	Page eject
28 through 23	Number of lines to space before print

– The following status indicators in the upper portion of word c of the message header will be set by the remote line handler if applicable:

Bits

16 is set when an image has been salvaged from a transmitted buffer containing a parity error; i.e. the handler was unable to receive the message without a parity error after three successive attempts.

18 is set when an unrecoverable error has been encountered by the remote line handler. The message area will not contain an image.

20–21 are set accordingly as the remote line handler treats each image as a complete message. The handler expects a complete image when requesting a message by submission of the GETM service request. These bits may vary at the task level dependent upon the user's parameterization of the buffer area on the read macros.

9.4.2. The Remote DCT 2000 System

OMEGA provides the user with a DCT 2000 remote line handler which, combined with the OMEGA Level 2 communications environment, enables a user program to transmit and receive messages from remote DCT 2000 sites. OMEGA scheduling will be provided, allowing a remote user to execute job streams originating at a remote site.

The DCT 2000 remote line handler will provide the interface between the OMEGA ESI control routines and the OMEGA communications director. The remote line handler will accommodate one or more remote DCT 2000 terminals, and is reentrant with respect to the different DCT 2000 sites.

The OMEGA Level 2 communications environment requires the user program to assign the remote DCT 2000 terminal, submit service requests which enable direct access storage queueing of communication messages, generate a recognizable header for messages submitted for transmission and release the terminal when further use is not required. The various service requests and message buffer format required for remote device control are described in this section.

- Remote Facility Assignment

  The user program must submit LASG and LACQ requests for each remote DCT 2000 terminal needed by his task. The statements are defined in 9.2. The name/version of the DCT 2000 remote line handler is DCTASC for the ASCII version of the DCT 2000 and DCTXS3 for the XS–3 version.

  LASG

  The LASG statement for the ASCII version without own code options has the following format:

  #LASGƀoptionsƀperipheral code, file code, DCTASC, priority

  LACQ

  The line acquire operator without own code options has the following form:

  LACQ$ƀfile code

  The user may request activation of own code routines on the LASG and/or LACQ requests. An own code routine is not required by the remote DCT 2000 system. The remote line handler will convert each input image to Fieldata and will expect each output image to be in the same format.

  LFREE

The LFREE statement has the following form:

#LFREEƀ file code

The LFREE statement is submitted when the DCT 2000 terminal is no longer required by the user task.

- Remote Data Access Service Requests

The service requests READM, READMW and WRITEM used by the task for the transfer of messages to and from the OMEGA system are described in 9.3.

– When submitting the READM or READMW service request, the user must specify a deposit buffer length of 19 words for receiving a complete image.

– Number of words for the WRITEM service request is the total length of the buffer including three words of message header.

If the user task receives a NO ASSIGNMENT status (5000000000) in response to a WRITEM service request, the DCT 2000 remote line handler has encountered an unrecoverable error and has submitted a CTMFREE service request.

- Message Buffer Format

Message buffer format is as specified in Section 9.3.4.

– When submitting a WRITEM service request the unit ID in the upper portion of word a defines the remote output media. A unit ID of 01 indicates that the image is to be printed. A unit ID of 02 indicates that the image is to be punched.

– The number of characters, lower portion of word a, is set to the number of characters received from the DCT 2000 when a complete message is requested by the user task via a read request.

– When the user task submits a message via the WRITEM service request, message length may be specified as less than or equal to 128 for each print request and less than or equal to 80 for each punch request. Only the number of characters specified is transmitted, as systems software requires that the DCT 2000 contain the short block feature.

– If the image is to be printed, the upper seven bits of word c contain page control information as follows:

Bit Position	
29	form feed (page eject)
28 through 23	number of lines to space before print (Zero specification for the number of lines to space will cause the image to be discarded as there is no overprint on the DCT 2000).

The following status indicators in the upper portion of word c of the message header are set by the DCT 2000 remote line handler if applicable.

Bit Position

16 set when a message has been received which contained a parity error. The handler has requested retransmission of the message three times previous to submitting the message for queuing.

18 set when an unrecoverable error has been encountered. The remote line handler has submitted a CTMFREE service request.

20 }
21 } The remote line handler treats each image as a complete message and sets the bits accordingly. The handler expects a complete message when requesting a message via the GETM service request. These bits may vary at the task level depending on the parameters set by the user in the buffer area on the READMW statement.

9.4.3. The Remote UNISCOPE 300 System

OMEGA furnishes a complete UNISCOPE 300 system which permits a user task to transmit and receive message to and from one or more remote UNISCOPE 300 sites. The system uses those elements which form a Level 2 interface (see 9.1.2.2). When combined with the UNISCOPE 300 system, the Level 2 interface permits the user task to execute basic read/write macros to handle communications traffic between the UNIVAC 494 CPU and a remote UNISCOPE 300 site.

To operate in a Level 2 environment, the user task must submit various service requests as defined for the system. These service requests and the specific parameters required when interfacing with a UNISCOPE 300 are presented in this subsection.

(a) Remote Facility Assignment

The user task must submit LASG, LACQ and LFREE requests as defined in Section 9.2. When using the remote UNISCOPE 300 system, the name/version of the remote line handler is U300.

LASG

The line assign statement without own code options has the following form:

#LASGƀoptionsƀperipheral code, file code, U300, priority

The A option (establish multiple activities) and the H option (load and link an additional copy of the remote line handler) should not be specified on the #LASG statement, as the handler is re-entrant and is so structured that it automatically links and services additional lines.

LACQ

The line acquire operator without own code options has the form:

LACQ$ƀfile code

The user may request activation of own code routines on the LASG and LACQ requests. The remote UNISCOPE 300 system does not require an own code routine.

LFREE

The line free statement has the following form:

#LFREEƀƀfile code

- Remote Data Access Service Requests

The user program requests the transfer of messages from the staging area to the program with the READM$ and READMW$ service requests. The communications director, upon receiving the service request, transfers a message from the input staging area on a first in, first out basis. The user program may specify a deposit area of any length. If the value specified is less than a complete message, only the portion of the message which fills the area is received by the task. The user program, in the latter case, must submit several requests to receive a complete message.

– READM

The format of the READM service request is as follows:

READM$ƀfile code, number of words, buffer base

If there are no messages currently in the queue, the READM$ request causes a 4400000000 status indicator to be returned to the user, indicating that there are no messages currently on queue, but that the file is still eligible to receive messages.

– READMW

A READMW$ service request delays the requesting activity until a message comes in.

The format of the READMW service request is as follows:

READMW$ƀfile code, number of words, buffer base

– LOOKM

Also available to the user is the LOOKM$ service request, which is similar to the READM$ request with the exception that the message is not removed from the input queue.

The format of a LOOKM$ request is as follows:

LOOKM$ƀfile code, number of words, buffer base

– WRITEM

The user program submits output messages with communications terminal identification to the communications director for staging. The message is submitted by the WRITEM$ service request. Number of words is the total length of the message including three words for the message header. The number of words in the message text may vary, but must not exceed 280 words.

The format of a WRITEM$ service request is as follows:

WRITEM$ƀ, buffer base, number of words

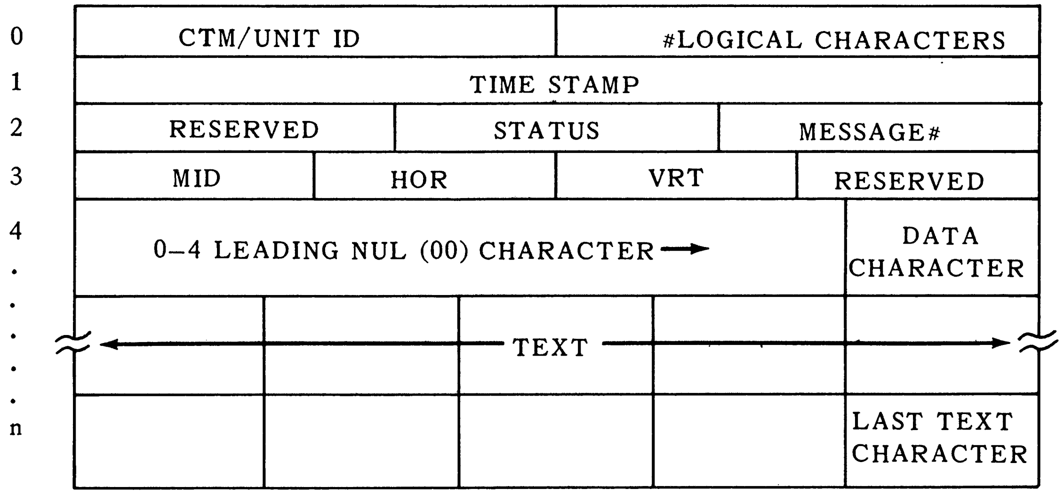
More information on communications read/write service requests is found in Section 9.3.2.

- Message Buffer Format

The UNISCOPE 300 remote Line Handler converts each input message to Fieldata and each output message is expected to be in the same format. An expanded code has been provided to describe editing functions, special characters and other symbols.

– Input Message Buffer

The message format shown below is used for input message received by the user from the communications director.



The input message header and message text is represented as follows:

Word 0 Bits 29–21 Communications terminal logical identifier set by the handler to indicate the terminal and device from which the message is received.

Bits 20–15 Unit identifier specifying a particular display unit.

Bits 14–0 Number of characters in this message, as set by the handler. The count begins with the first character in word 4 (leading NUL characters, if present, are counted) and continues to the last character in word n.

Hence, the number of characters on an input message is always a multiple of 5.

Word 1 A time stamp supplied by the communications director upon processing the PUTM request, representing the number of 100 millisecond units elapsed since the first of the year.

Word 2 Bits 29–22 Not currently used.

Bits 21–20 Set to 11 to indicate that complete message is contained in buffer.

Bits 19–15 Set to denote various error conditions. See Section 9.3.4 for explanation.

Bits 14–0 Message number assigned to each message that is input to the communication director by way of PUTM service request.

Word 3 Bits 29–22 Mode identifier, an 8 bit character identifying the overlay plate being used at the remote device.

Bits 21–15 Seven bits used to denote the horizontal screen position of the beginning of the input message.

Bits 14–10 Five bits used to denote the vertical screen position of the beginning of the input message.

Bits 9–0 Not currently used.

Word 4 This word and each succeeding word in the message contains 5 six-bit Fieldata-coded logical characters. An actual character may require one, two or four logical characters, as in the case of the 12-bit function key characters or the 24-bit cursor positioning sequence. The character count in the lower half of word 0 refers to the logical character count, not to the actual number of characters that appear on the screen before transmission. A 12 or 24 bit sequence may be divided between two words within the pack buffer; for example, a function key character may occupy bits 0–5 of word n-1 and bits 24–29 of word n. Since data is received by the handler in reverse order, characters are packed into the buffer beginning with the last character of word n. This may result in one to four logical character positions being left unfilled at the beginning of the data in word 4. These positions are packed with NUL characters (Fieldata 00). These NUL characters are included in the character count in word 0.

Output Message Buffer

The message header and message text of the output message have the following format in the UNISCOPE software system:

<table>
<tr><td>0</td><td colspan="5">CTM/UNIT ID</td><td colspan="5">#LOGICAL CHARACTERS</td></tr>
<tr><td>1</td><td colspan="10">TIME STAMP</td></tr>
<tr><td>2</td><td colspan="3">MESSAGE TYPE</td><td colspan="1">O
Q
D</td><td colspan="2">STATUS</td><td colspan="4">MESSAGE#</td></tr>
<tr><td>3</td><td colspan="2">FIRST DATA CHARACTER</td><td colspan="2"></td><td colspan="2"></td><td colspan="2"></td><td colspan="2"></td></tr>
<tr><td>n</td><td colspan="10">← DATA →</td></tr>
</table>

Word 0 Bits 29–21 CTM Logical identifier set by the user to indicate the terminal and device for which message is intended.

Bits 20–15 Unit identifier set by the user specifying a particular display unit.

Bits 14–0 Number of characters in this message, set by the user. The first logical character occupies bits 24–29 of word 3, and data continues to word n.

Word 1 A time of storage indicator supplied by the communication director upon processing the communication director upon processing the WRITEM request.

Word 2 Bits 29–27 Message type, set by the user as follows:

000 – normal output

001 – unconditional message

010 – conditional message

In addition, the following message types are sent by the handler only:

011 – poll

100 – request for conditional message

101 – request for retransmission

110 – request to unlock keyboard

Bits 26–23 Reserved.

Bit 22 Output queue indicator. Must be set to 1 by the user to indicate that a separate queue has been established for each output unit associated with the specified CTM control block.

Bits 21–20 Set to 11 to indicate that the complete message is contained in the buffer.

Bits 19–15 Set to indicate various error conditions. See Section 9.3.4 for details.

Bits 14–0 Message number assigned to each message entering the communication director by way of the WRITEM service request.

Word 3 This word contains the beginning of data to the remote device; normally, an output message begins with the cursor positioning sequence. The first logical character must occupy bits 24–29, although this character position and any succeeding position may be occupied by the NUL characters may be used to facilitate message construction, and the characters DEL, ERL, INL and ERD must be followed by three NUL characters to allow time for the function to complete. All NUL characters are included in the character count. Function key characters and certain editing functions (INL, ERL, EOF and ERD) occupy 12 bits and count as two logical characters in word 0; the cursor positioning sequence occupies 24 bits and counts as four logical characters.

- Output Editing functions

Various functions are made available to the user that put cursor positioning and screen editing functions, which are summarized in the table below, under control of user software. These functions are as follows:

FUNCTION	ASCII	FIELDATA	FUNCTION	ASCII	FIELDATA
NUL	000	00	INL	030	7724
SOM	004	7734	SBF	031	43
ERD	012	7727	ERL	032	7725
CRF	013	04	TAB	033	02
KBU	024	57	DEL	034	76
CUR	027	7723	EBF	035	45

- Cursor positioning sequence (CUR). The cursor positioning sequence moves the cursor to the screen position specified by the user. The special character 7723 is followed by a seven-bit horizontal coordinate and a five bit vertical coordinate.

- Nul character (NUL). The nul character (Fieldata 00) may be used freely to facilitate the composition of output messages. The nul character is disregarded by the handler and produces no action at the UNISCOPE. As a timing consideration, the output editing functions ERL, DEL, INL and ERD (see below) should each be followed by three nul characters to allow time for the functions to complete.

- Cursor Return (CRF). The cursor return (Fieldata 04) positions the cursor at the beginning of the next line.

- Blink Field delimiters (SBF and EBF). Bracketing data with the Start Blink Field character (Fieldata 43) and the End Blink Field character (Fieldata 45) causes the so enclosed characters to blink. This is accomplished by decreasing the restoration rate of the bracketed character. The SBF and EBF characters cannot be displayed.

- Keyboard Unlock (KBU). The keyboard unlock character (Fieldata 57) is required at the end of every output message to return console control to the operator.

- Line Delete (DEL). The line delete function (Fieldata 76) causes the line where the cursor is located to be erased and all lines below it to move up one line, leaving the bottom line blank.

- Line Insert (INL). The line insert function (Fieldata 7724) causes all lines below and including the line at which the cursor is placed to move down one line; the bottom line disappears and is lost from memory.

- Erase to End of Line (ERL). The erase to end of line function (Fieldata 7725) causes all characters from and including the cursor to the end of the line to be erased. The locations are filled with space codes. All other lines are unaffected.

- Erase to End of Display (ERD). The erase to end of display function (Fieldata 7727) causes all characters beginning at the cursor position and ending at the end of the display to be erased. The locations are filled with space codes.

- Tab Stop Set (TAB). The tab stop function (Fieldata 02) can be set only from the central processor; the function is similar to the tab operation on the standard electric typewriter. When received in an output message, the code provides a stop position for the cursor when the TAB key on the UNISCOPE keyboard is depressed.

- Start of Message (SOM). The start of message character (Fieldata 7734) is not transmitted from the UNISCOPE unit to the CPU. The character may be included in an output message to allow the operator to reply to the message with data typed into a selected field. The data between the cursor and the first preceding start of message character are transmitted.

Whenever used, the output editing functions must be counted in the character count of the message header, according to the length of the character. That is, the functions count either as one, two or (in the case of the cursor positioning sequence) four characters.

9.4.4. Remote UNISCOPE 100/DCT 1000 System

OMEGA furnishes a complete UNISCOPE 100/DCT 1000 system which permits a user task to transmit and receive messages from one or more UNISCOPE 100/DCT 1000 Sites. The level 2 interface permits the user software to utilize the basic read/write macros to handle communication traffic.

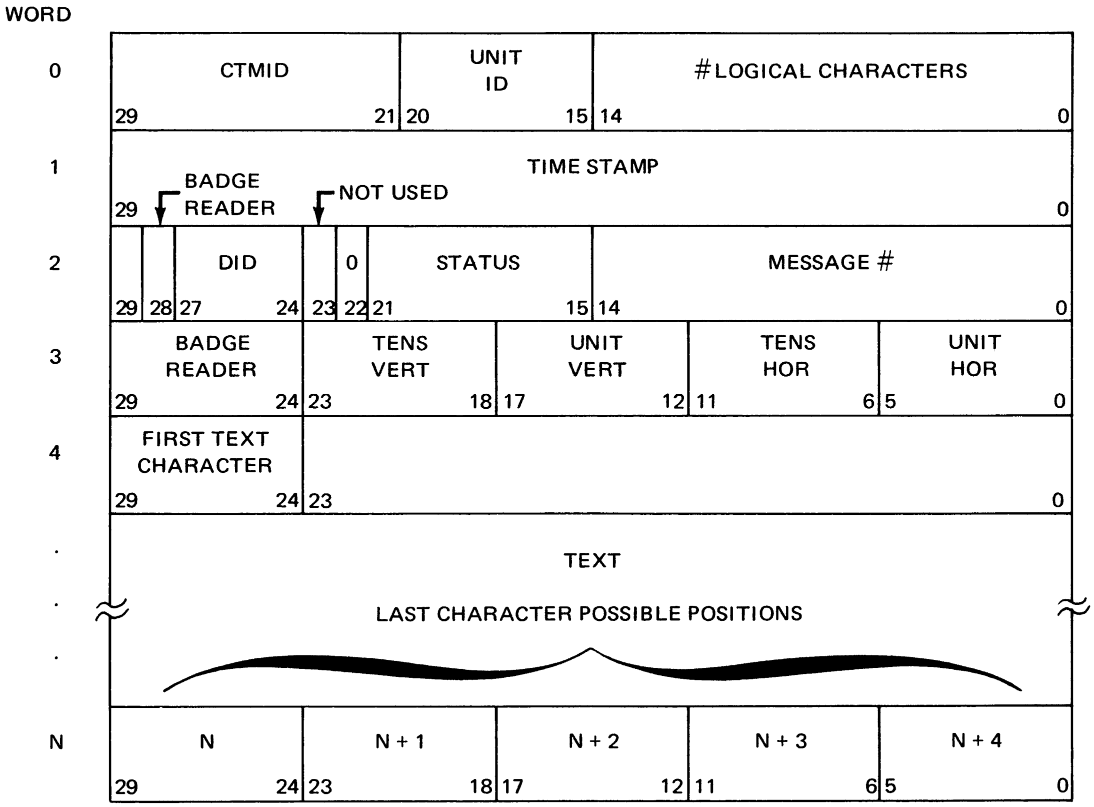
To operate in a Level 2 environment, the user task must submit various service requests as defined for the subsystem. These service requests are similar to those described for UNISCOPE 300 except the name/version of handler in LASG statement is U100.

The handler is reentrant and is so structured that it automatically links and services additional lines and as such A and H options should not be specified in LASG statement.

The features specific to the UNSICOPE 100/DCT 1000 such as buffer formats and output editing are presented in the following subsection.

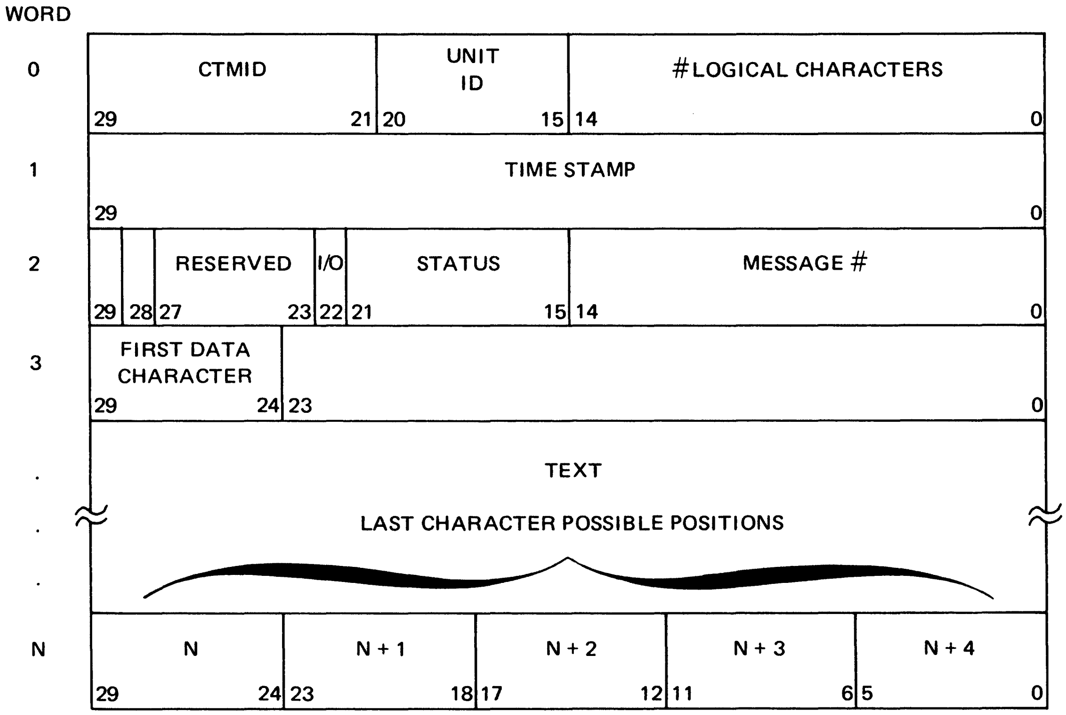
— Input Message Buffer Formats

(a) UNISCOPE 100



INPUT MESSAGE TO THE USER FROM UNISCOPE 100

(b) DCT 1000



TEXT FROM DCT 1000 PASSED TO USER OR CONTINUATION OF SAME TEXT FROM TERMINAL

9.4.4.1. INPUT MESSAGE BUFFER

The input message header and text will have the following format:

Word 0 Bits 29–21 CTM logical identifier set by the handler to indicate the terminal and device from which message received.

Bits 20–15 Unit identifier specifying a particular display unit.

Bits 14–0 Number of characters in this message, set by the handler. The count begins with the first character in Word 3 (NUL characters, if present, are counted) and continue to the last character in Word n. The last character can be in any one of the five positions in Word n. If an error condition exists, this will be the logical character count of the last valid character in the user's output message buffer.

Word 1 A time stamp supplied by the Communications Director.

Word 2

Bit 29 Set to a 1 indicates an error condition as defined in Word 3 of the input message packed buffer.

Bit 28 Set to a 1 indicates that the badge reader character is a character in ASCII range 0–40 and 140–177. This applies to a UNISCOPE 100 terminal only. For a DCT 1000 this bit set to a 1 indicates text is in binary.

Bits 27–24 DID of input relative to 160_8.

Bit 23 Not currently used.

Bit 22 If set to a 1, this packed buffer is not the last buffer of the text received from a terminal.

Bits 21–20 Set 11 to indicate complete message is contained in buffer.

Bits 19–15 Set to denote various error conditions. The handler will not set any of these error conditions.

Bits 14–0 Message number assigned to each message input to the Communication Director via the PUTM$ Service Request.

Word 3

The beginning of input data to the user. If UNISCOPE 100, this will contain the badge reader and cursor coordinates or status conditions. If DCT 1000, this will be the first text character. If error condition, this will be the error flags. Refer to the input message buffer figures for each condition. This word will contain text if it is a continuation of a user's packed buffer regardless of the terminal.

This word and each succeeding word in the message contains five six-bit Fieldata-coded logical characters. The character count in the lower of Word 0 refers to the logical character count, not the actual number of characters that appear on the screen before transmission. NUL characters, badge reader, and cursor characters are included in the character count in Word 0. All function codes and lower case letter sequences will have a leading and ending 77 $_8$ (hat = 77HAT77).

Bits 29–24 Badge reader character (character from columns 2, 3, 4, or 5 if bit 28 of Word 2 is zero or columns 0, 1, 6, or 7 if bit 28 of Word 2 is one.) Figure 2–2.

Bits 23–18 Tens digit of cursor column (y) coordinate. Fieldata zero if there is not a tens digit.

Bits 17–12 Units digit of cursor column (y) coordinate.

Bits 11–6 Tens digit of cursor row (x) coordinate. Fieldata zero if there is not a tens digit.

Bits 5–0 Units digit of cursor row (x) coordinate.

If word 3 is used to pass status from either the print request or special function keys, word 3 will be zero and the status will be returned in word 4.

If word 3 is used as an error indicator to the user program, the following format is used to inform the user of his bad output message buffer. The output message buffer is lost and the handler will process the next output message buffer in the queue after informing the user.

Bit 29 Set to a 1 indicates an invalid character was in the user's text of the output message buffer.

Bit 28 Set to a 1 indicates an invalid DLE sequence (DLE[or DLE]) for UNISCOPE 100.

Bit 27 Set to a 1 indicates a zero character count in word 0. If the user requests a keyboard unlock function for UNISCOPE 100 or master clear for DCT 1000 then a zero character count is valid.

Bit 26 Set to a 1 indicates an invalid y cursor coordinate for UNISCOPE 100.

Bit 25 Set to a 1 indicates an invalid x cursor coordinate for UNISCOPE 100.

Bit 24 Set to a 1 indicates too many characters (maximum 1400_{10} characters allowed) in count of word 0.

Bit 23 Set to a 1 indicates five input retransmissions in a row were issued without a positive response. The terminal was declared inactive.

Bit 22 Set to a 1 indicates five output retransmissions in a row were issued without a positive response. The terminal was declared inactive.

Bit 21 Not used.

Bit 20 Not used.

Bit 19 Not used.

Bit 18 Not used.

Bits 17 thru 15 Refer to the DCT 1000 only.

Bit 17 Set to a 1 indicates over 160 characters for DCT 1000, but received a conversational DID. Only 160 characters are transmitted to terminal. The remaining characters are lost.

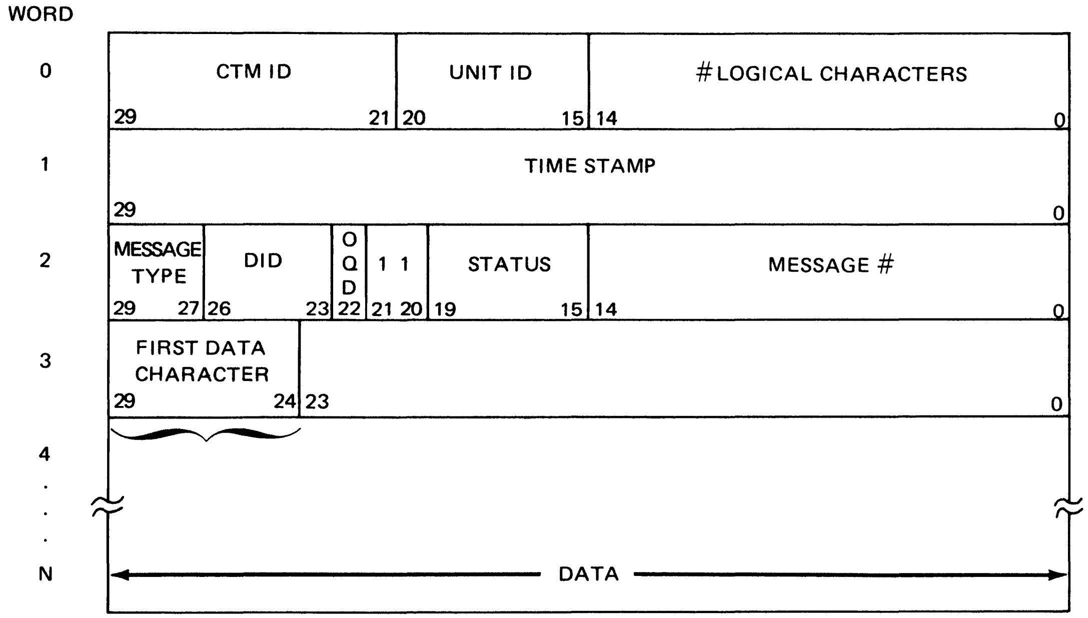
Bit 16 Received an SO character in text (shift into binary) and DID was conversational. This only applies if the SO character is detected after the first text character. If user sends only 160 binary characters (SO and SI not included in character count) with conversational DID, this flag will not be set.

Bit 15 User passed an input DID to the handler or message mode other than normal master clear or special function. The handler will change DID to output for same device and will output the text as a normal output message.

Bits 0–14 This will have the users output packed buffer message number that is currently being processed.

9.4.4.2. OUTPUT MESSAGE BUFFER

The message header and text of the output message submitted for transmission by the user through the WRITEM$ service request will have the following format:



Output Packed Buffer

Word 0 — Bits 29–21 — CTM logical identifier set by the user to indicate the terminal and device for which message is intended.

Bits 20–15 — Unit identifier set by the user specifying a particular display unit (UCB).

Bits 14–0 — Number of logical octal characters in this message, set by the user. The character count begins with the first character (bits 29–24) of word 3 and continues to word n (bits 5–0).

Word 1 — A time stamp supplied by the Communication Director upon processing the WRITEM$ request.

Word 2 — Bits 29–27 — Message type, set by the user as follows:

000 – normal output message (sent after poll)
001 – fast select message (unsolicited) – U100 only
010 – conditional message (message waiting) – U100 only
101 – UP UCB and/or modify RID, SID, DID in UCB
110 – request to unlock the keyboard for U100 or programmable master clear fc
DCT 1000 (master clear feature will be available when hardware supportec

All other message types will be interpreted as a normal output message (000).

Bits 26–23 DID relative to 160_8 (160_8=0000,177_8=1111, etc.)

Bit 22 Output queue indicator. Must be set to 1 by the user to indicate that a separate queue has been established for each output unit associated with the specified CTMCB.

Bits 21–20 Set 11 to indicate complete message is contained in buffer.

Bits 19–15 Set to indicate various error conditions.

Bits 14–0 Message number assigned to each message entering the Communication Director by a WRITEM$ service request (set by the Communications Director).

Word 3 This word contains the beginning of data to the remote device; normally, an output message will begin with the cursor positioning sequence if a UNISCOPE 100. This may be a cursor coordinate sequence, cursor to home, clear to end of display or combination of various sequences. The first logical character must occupy bits 24–29. A NUL character will be interpreted as a space.

9.4.4.3. UNISCOPE 100 OUTPUT EDITING FUNCTIONS

There are various functions available which will allow the user task to control cursor positioning, screen editing functions and output to all terminals editing functions. NUL characters are added by the handler to allow time for certain functions to be completed by the UNISCOPE 100 terminal depending on the line speed.

A Fieldata 77 (≠) is used to start and end all escape sequences received by and sent to the user by the handler.

The handler will position the escape sequence characters on all inputs to the user, but it is the user's responsibility to position all escape sequence characters on output to the handler.

An escape sequence is defined as any character in the ASCII range 0_8 to 40_8 and 140_8 to 177_8.

Editing Function	Description
Cursor Address	The format for the cursor address in a text message from the user to the handler is as follows: ESC,VT,Y,Y,X,X,SI Fieldata equivalent is: 77,73,42,60,61,70,60,74,77 (positions the cursor at row 1 and column 80) The ESC (Escape) character determines the meaning of the following character to permit extended control sequences. The VT character following ESC denotes that the next four characters are the cursor address: the first two identifies the row, and the last two the column. The first digit of both the row and column coordinate is the tens digit and the second is the unit digit. If only the unit digit is required, the tens digit must be a zero (Fieldata 60).

Table 9–1. UNISCOPE 100 Output Editing Functions (Part 1 of 5)

<table>
<tr><th>Editing Function</th><th>Description</th></tr>
<tr><td>Print Transparent (output)</td><td>The Print Transparent function (Fieldata 776277) causes output from the display memory to the auxiliary channel in the same manner as the print function. The area transmitted is the same as in the print function, but the carriage return characters normally inserted by the logic of the UNISCOPE 100 terminal are not transmitted to the auxiliary channel. This is only functional when defined in the auxiliary interface.</td></tr>
<tr><td>Transmit</td><td>The Transmit code (Fieldata 776177) may be transmitted from the computer to the UNISCOPE 100. It will not be acted upon until the receipt of a satisfactory BCC character in the associated message. The result of action is the same as for the depression of the transmit key (FR 6). (DC 1)</td></tr>
<tr><td>Start of Entry (SOE) ▷</td><td>The Start of Entry (Fieldata 774577) character defines the beginning of the area to be transmitted to the computer or to the auxiliary interface. More than one SOE ▷ character may be on the screen (in memory) at one time. The SOE ▷ closest to the cursor defines the beginning of the area that will actually be transmitted. (RS)</td></tr>
<tr><td>SI – SO</td><td>The Shift In (SI-Fieldata 777477) and Shift Out (SO-Fieldata 77577) characters are used to switch to an alternate graphic set and to return to the standard. By their use, transmission is not limited to the 96 characters defined in the ASCII character set. These characters are reserved for this function.</td></tr>
<tr><td>LF, FF, EM</td><td>The Line Feed (LF-Fieldata 775077), Form Feed (FF-Fieldata 775677), and the End of Medium (EM-Fieldata 771777) are intended for use by the UNISCOPE 100 terminal for the auxiliary interface. The UNISCOPE 100 terminal itself is transparent to these characters; they may be placed in memory by either the communications interface or the auxiliary interface, and do not affect the display on the UNISCOPE 100 terminal.

The following control characters are NOT valid in the text and if found, the handler will give an error indicator to the user (Input Message Packed Buffer Word 3):
<table>
<tr><th>Character</th><th>Fieldata</th><th>ASCII</th></tr>
<tr><td>SOH</td><td>775577</td><td>01</td></tr>
<tr><td>STX</td><td>777677</td><td>02</td></tr>
<tr><td>ETX*</td><td>770377</td><td>03</td></tr>
<tr><td>EOT</td><td>774777</td><td>04</td></tr>
<tr><td>DLE</td><td>7706 followed by any sequence.</td><td>20 followed by any sequence.</td></tr>
</table>
</td></tr>
</table>

*The ETX will not be transmitted to the UNISCOPE but will terminate the user's message at the point where the ETX is encountered and will transmit all characters before the ETX. Therefore, it is suggested to use the ETX with extreme caution.

Table 9–1. UNISCOPE 100 Output Editing Functions (Part 4 of 5)

Editing Function	Description
Delete Line	The Delete Line (Fieldata 77732077) function causes all of the lines below the line at the cursor position to move up one line. (ESC k)
Message Waiting	The Message Waiting function is sent from the computer to the UNISCOPE 100 display to indicate that the computer has an unsolicited message available for the operator. BEL has this meaning only when it is included in the message before STX. The handler will position the BEL in the correct position before the STX when the user program indicates that the text is a conditional message in the header of the output message packed buffer. The only sequence in the users text which will cause the Message Waiting function is a DLE { ,DLE } or an invalid DLE sequence. As a consequence, the handler checks all DLE sequences and informs the user by passing an error status 20000 back to the user (word 3 of the Input Message Packed Buffer).
Request Computer Message	When the operator presses the Message Waiting key, a message is generated for transmission to the computer; it contains the BEL character, indicating to the computer that the Message Waiting key was pressed. BEL has this meaning only if it comes before the STX Message. (BEL)
Tab Stop Set	The Tab Stop Set (Fieldata 77734077) function from the computer causes the display to place the HT character in memory to act as a tab stop (see Tab). (ESC HT)
Start Blink Field	Start Blink Field, (Fieldata 774377) FS, is placed in memory; it causes the Start Blink Field character to appear on the screen and blink. (FS)
End Blink Field	End Blink Field (Fieldata 774477) GS, is placed in memory; it causes the End Blink character to appear on the screen and blink. (GS)
Lock Keyboard	When either the Transmit key or the Message Waiting is pressed, the keyboard becomes locked (disabled). The keyboard is also locked when the Start of Text character (STX) is received from the computer. If the text message does not include the locked keyboard (DC 4) characters at the end of the Text (ETX followed by a satisfactory BCC) the keyboard is automatically unlocked. This gives the programmer the ability to lock the keyboard and makes it unnecessary to use an unlocking character in each text message. The Lock Keyboard Fieldata equivalent is 776477. Either a text message (STX ETX combination) or an unlock keyboard must be sent by the user to the UNISCOPE 100 to unlock the keyboard. (DC 4)
Print (output)'	The Print code (Fieldata 776277) causes the terminal to output data from the display memory to the auxiliary channel.

Table 9–1. UNISCOPE 100 Output Editing Functions (Part 3 of 5)

Editing Function	Description
SI	The SI character is the character specified to end the sequence started by the ESC VT codes.
Carriage Return	The Carriage Return character (Fieldata 774177) causes the cursor on the UNISCOPE 100 terminal display to move to the beginning of the next line. (CR)
Erase to End of Line	The Erase to End of Line function (Fieldata 77730777) will erase from the current cursor position to the end of the line. (ESC b)
Erase to End of Display	Erase to End of Display (Fieldata 77730677) will erase all unprotected data from the current cursor position to the end of the display. (ESC a)
Delete in Line	The Delete in Line function (fieldata 77731077) operates the same as the Delete in Line key on the UNISCOPE 100 terminal keyboard. (ESC c)
Delete in Display	The Delete in Display function (Fieldata 77737710) operates the same as the Delete in Display key (upper case) on the UNISCOPE 100 terminal keyboard. (ESC C)
Insert in Display	The Insert in Display function (Fieldata 77737711) operates the same as the Insert in Display (lower case) key on the UNISCOPE 100 terminal keyboard. (ESC D)
Insert in Line	The Insert in Line function (Fieldata 77731177) operates the same as the Insert in Line (lower case) key on the UNISCOPE 100 terminal keyboard. (ESC d)
Scan Left	This function (Fieldata 77731477) causes the cursor to move one position to the left. (ESC g)
Scan Right	This function (Fieldata 77731577) causes the cursor to move one space to the right. (ESC h)
Scan Down	This function (Fieldata 77731677) causes the cursor to move down one line. (ESC i)
Scan Up	This function (Fieldata 77731377) causes the cursor to move up one line. (ESC f)
Space	The Space character (Fieldata 05) is a nondisplay character used as a space.
Tab	The function (Fieldata 774077) of Tab is similar to that of the Tab key on the UNISCOPE 100 terminal keyboard. (HT)
Cursor to Home	The Cursor to Home (Fieldata 77731277) function moves the cursor to the upper left hand corner of the screen (row 1, column 1). (ESC e)
Insert Line	The Insert Line (Fieldata 77731777) function causes the line at the cursor position, and all following lines, to move down one line, resulting in a blank line at the cursor position and the loss of the bottom line. (ESC j)
Clear Form	The Clear Form (Fieldata 77737722) or Erase Unprotected to End of Display will erase all data (protected or unprotected) between the cursor position and End of Display. (ESC m)

Table 9–1. UNISCOPE 100 Output Editing Functions (Part 2 of 5)

<table>
<tr><th>Editing Function</th><th>Description</th></tr>
<tr><td>LF, FF, EM</td><td>The following control characters do not affect the text and at present are not defined.
<table>
<tr><th>Character</th><th>Fieldata</th><th>ASCII</th></tr>
<tr><td>ENQ</td><td>775277</td><td>05</td></tr>
<tr><td>ACK</td><td>774677</td><td>06</td></tr>
<tr><td>BS</td><td>775177</td><td>10</td></tr>
<tr><td>DC3</td><td>776377</td><td>23</td></tr>
<tr><td>NAK</td><td>776577</td><td>25</td></tr>
<tr><td>ETB</td><td>776777</td><td>27</td></tr>
<tr><td>CAN</td><td>777077</td><td>30</td></tr>
<tr><td>SUB</td><td>775377</td><td>32</td></tr>
<tr><td>US</td><td>775477</td><td>37</td></tr>
</table>
Whenever used, the output editing functions and escape sequence character (≠) must be counted in the character count of the message header.</td></tr>
<tr><td>Tab Stop Set</td><td>The computer can insert tab stops (analogous to a typewriter) in text transmitted to the UNISCOPE 100 terminal. The sequence ESC HT is inserted at each point in the text where a tab stop is desired.</td></tr>
<tr><td>Roll and Scroll</td><td>The effect of the data rolling downward or upward across the face of the screen can be achieved by the use of the line insert (ESC j) or line delete sequences (ESC k), respectively.</td></tr>
</table>

Table 9–1. UNISCOPE 100 Output Editing Functions (Part 5 of 5)

9.4.4.4. DCT 1000 OUTPUT EDITING FUNCTIONS

The following control codes are the responsibility of the user for including in the text message sent to the terminal or computer.

Editing Function	Description
CR – Carriage Return	The carriage return character (Fieldata 774177) from the computer to the DCT 1000 causes the printer to move the print wheel to the beginning of the same line. The CR function will be performed at the point it appears in the text.
SP – Space	The space character (Fieldata 05) is a non-printable character (punchable) used as a space.
LF – Line Feed	The line feed character (Fieldata 775077) from the computer to the DCT 1000 causes the printer to perform one of two operations depending on an internal switch setting in the DCT 1000. (1) line feed one line. (2) line feed one line and return the print wheel to the beginning of this new line. The LF character will be punched in the card or tape.
FF – Form Feed	The form feed character (Fieldata 775677) from the computer to the DCT 1000 causes the printer to advance paper to the first line of the next 11 inch form and returns the print wheel to the beginning of line. The FF character will be punched in either card or tape.
EM – End of Medium	The end of medium character (Fieldata 777177) from the computer to the DCT 1000 indicates that the block length is less than 160 characters. The EM will appear as a space on the printer but will be punched on cards or paper tape. If EM is not used on short blocks the ETX or ETB will be punched.
	The DCT 1000 will transmit the EM character under the following conditions: (1) When entering less than 160 characters from the keyboard, the EM character is automatically entered when the release key is depressed. (2) When read from paper tape or cards. The EM character will terminate all further processing of the users output message buffer when encountered in the packed output text.
SO – Shift Out	The shift out character (Fieldata 777577) indicates that all data characters following SO are binary characters. The SO character must be followed by 160 binary characters.
SI – Shift In	The shift in character (Fieldata 777477) delimits the binary data characters.

Table 9–2. DCT 1000 Output Editing Functions

9.4.5. Remote DCT 500 and Teletypewriter System

UNIVAC DCT 500 is a remote unbuffered input/output device which provides low speed data communication.

There are two versions of the UNIVAC DCT 500: a standard version, and an automatic version.

The automatic operation version is available in three configurations:

- the basic configuration of the DCT 500 consisting of an incremental printer and a control unit;
- the expanded configuration, consisting of the printer, control unit, and keyboard;
- the maximum configuration, consisting of the printer, control unit, keyboard, and paper tape unit.

The standard version is available in two configurations:

- the basic configuration, consisting of an incremental printer and a control unit;
- the expanded configuration, consisting of the printer, control unit, and keyboard.

FUNCTIONAL OPERATIONS

The DCT 500 provides three types of functional operations depending on the equipment configuration. In the basic configuration, consisting of a printer and control unit, the DCT provides receive only (RO) operation. In the expanded configuration that includes a keyboard the DCT provides keyboard send receive (KSR) operation. In the maximum configuration that includes a paper tape unit, the DCT provides automatic send receive (ASR) operation.

The OMEGA Level 2 interface permits the user task to utilize the Read/Write macros to transmit and receive to and from DCT 500 sites.

The DCT 500 communication handler is reentrant and to operate in Level 2, the user task must submit various service requests similar to those described for UNISCOPE 300 except the name/version of handler in the LASG statement is DCT 500.

The features specific to DCT 500 are presented in this subsection.

9.4.5.1. INPUT/OUTPUT MESSAGE BUFFER

The user task message buffer is a packed buffer, five Fieldata characters per word, similar to UNISCOPE 100. Included in the buffer is the 3 word header where:

Word 0	Same as for UNISCOPE 100
Word 1	Same as for UNISCOPE 100

Word 2 Bit 29–26 = MSC status

To the user

00 = Message complete and normal
01 = Input from paper tape
02 = Input time out

From user

00 = Normal Message
01 = Output to paper tape
03 = User program terminating

Bit 21–20 = Message Segment

03 = Complete Message
02 = The leading segment
00 = The middle segment
01 = The end segment of the message.

Escape Sequence character Fieldata 77 (≠) performs the same function in bracketing the variable length escape sequences to and from user task as UNISCOPE 100.

9.4.5.2. CONSTRAINTS

1. Asynchronous transmission of 110, 150 and 300 bauds lines only supported.

2. The user will be responsible for data formatting including paper, form control.

3. TTY mode of operation only one terminal per line is permissible and no operational status shall be passed to the user.

4. Message buffer should be no longer than 16_{10} words including the control words as one of the main functions of handler is to allocate time between polled multidrops DCT 500.

5. The user program shall be started from central site.

9.5. REMOTE SCHEDULER

The Remote Scheduler (RS) is a worker program of the OMEGA System Library. The purpose of the program is to act as an interface with remote sites and the OMEGA Level 2 communications environment in controlling the input/output of the remote job streams.

The Remote Scheduler (RS) controls the time period, as determined by a remote time schedule, when a remote site is in contact with OMEGA. During these periods, remote job streams may be sent, processed and received by a remote site. In the event that there is still data to be returned to a remote site when a time period elapses, the data is transmitted, further input is ignored and the line is released after completion of the last job in the system for that site.

The Remote Scheduler handles as many devices as may be connected to the system. In normal operation, the RS is loaded at the start of the work day and remains in primary storage, making contacts and idling until the end of the work day, when RS is terminated by an operator type-in.

9.5.1. Operation of Remote Scheduler

Before the Remote Scheduler can run, it must have a time schedule to be constructed as follows:

1. The user submits a SPURT assembly of an element (or elements) to OMEGA, the elements contain a time schedule and constant block for each remote site on the system.

2. The element(s) is then loaded to form an absolute element. This element is the remote site time schedule, and may be saved (by way of #OUT) or given directly to the Scheduler.

3. When the Scheduler is called for the first time, the remote site time schedule must be in either the job, group or systems library.

4. The Scheduler is notified of this by a schedule parameter card.

5. The Scheduler then brings the schedule into primary storage, acquires a mass storage file, writes the schedule on the file and registers the file with the Master File Directory for future reference.

6. The Schedule is then updated according to incoming parameter cards, and scheduling is begun.

A remote time schedule, one of which is required for each remote site, has the following format:

REMOTE TIME SCHEDULE

Word 0	SITE ID	
1	PHONE	
2	NUMBER	
3	OF SITE	
4	START TIME FOR PERIOD 1	
5	STOP TIME FOR PERIOD 1	
	: : :	
	/LASG STATEMENT	
	⋮ ⋮	
32		FILE CODE
33	OWN CODE	
34	NAME AND	
35	VERSION	
36	/LFREE	
37		
40	OPEN	

A further description of each word in the format is furnished below:

Word 0

SITE ID

Five character remote site identification

Word 1
2
3

PHONE NUMBER OF SITE

Telephone number of remote site (if any)

Words 4, 6, 10, 12, 14

START TIME FOR PERIOD n

Field data starting time for a given period (HH:MM)

Words 5, 7, 11, 13, 15

STOP TIME FOR PERIOD n

Field data stop time for a given period (HH:MM)

Words 16–31

/LASG STATEMENT

Control image used to assign a remote device when contact is to be made. Maximum of 60_{10} fieldata characters and must be space-filled unless a termination character is used (04). Own code routines are not used.

Word 32

OPEN	FILE CODE

File code specified on the LASG image.

Words 33–35	OWN CODE NAME AND VERSION

Contains the name and version of the own code routine which examines each message coming from the remote site before the message is placed on the input queue. If not specified, the standard OMEGA own code routine will be inserted. Binary zero will be inserted if the field is not used.

Words 36, 37	/LFRE
	EƀƀFC

Contains the LFREE control image to be submitted when the RS wishes to close out the input and output queues.

Word 40 OPEN

9.5.2. SCHEDULE Entry

The SCHEDULE entry is used to notify the Scheduler that a complete schedule is contained in the job or systems library. This Schedule is used until replaced by the element named on a new SCHEDULE entry.

- Format:

  The format of the SCHEDULE entry is as follows:

  SCHEDULEƀƀname/version, library

- Specifications:

  Name/version specifies the name and version of the remote site time schedule.

  Library indicates the library where the schedule may be found. The library may be JOB, GROUP or SYS. If the library field is blank, JOB is assumed.

- Method of Submission

  The SCHEDULE entry may be made only as the first parameter in the on-site initializing of the JOB stream.

9.5.3. TIME Entry

The TIME entry is used to notify the Scheduler of a deletion, correction, or addition in the contact period specified for a remote site time schedule.

- Format:

  The format of the TIME entry is as follows:

  TIMEƀoptionƀidentify, hh:mm$_1$/hh:mm$_2$,hh:mm$_3$/hh:mm$_4$

- Options:

  T – indicates a temporary change; the change is in effect for one work period only.

  P – indicates a permant change to the schedule.

- Specifications:

  Identity is the five character alphanumeric site identity which was given to the remote site whose schedule is to be changed.

  hh:mm$_1$/hh:mm$_2$ is the time on/time off for this site as it currently exists in the time schedule. If hh:mm$_3$/hh:mm$_4$ is given, hh:mm$_1$/hh:mm$_2$ is replaced by hh:mm$_3$/hh:mm$_4$. If not, hh:mm$_1$/hh:mm$_2$ is deleted. hh:mm$_3$/hh:mm$_4$ is the new time on/time off which is to be applied to the time schedule for this site. If hh:mm$_1$/hh:mm$_2$ is not given, hh:mm$_3$/hh:mm$_4$ is added to the schedule.

- Method of Submission:

  The TIME entry may be made as part of the on-site initializing job stream, as part of an incoming transmission from a remote site, or as an unsolicited on-site operator type-in. If the entry is to be made as an on-site type-in, the entry must be prefixed by the job number under which the scheduler is running, such as:

  Jx¤ TIMEƀoptionƀidentity, hh:mm$_1$/hh:mm$_2$,hh:mm$_3$/hh:mm$_4$

  where x is the scheduler's job number.

9.5.4. PHONE Entry

The PHONE entry is used to notify the scheduler of a change in the telephone number to be used in contacting the specified site.

- Format:

  The format of the PHONE entry is as follows:

  PHONEƀoptionsƀidentity, number

- Options:

  T – indicates a temporary change. The change is in effect for one work period only.

  P – indicates a permanent phone number change.

- Specifications:

  Identity is the five character alphanumeric site identity which was given to the remote site whose number is to be changed.

  Number is the new phone number for the specified site and may be a maximum of 15_{10} characters.

- Method of Submission:

  The PHONE entry may be made as part of the on-site initializing job stream, as part of an incoming transmission form a remote site (for future contacts) or as an unsolicited on-site operator type-in. If the entry is made as an on-site type-in, the entry must be prefixed by the job number under which the scheduler is running; for example:

  Jx¤ PHONEƀoptionsƀidentity, number Ⓢ

  where x is the scheduler's job number.

9.5.5. STOP Entry

The STOP entry is used to terminate a schedule period before the preset time. If this entry is transmitted by the remote site, it causes the termination of th.: scheduling for that particular time period for that site only. If the on-site operator makes the entry as an unsolicited type-in, he has the option to terminate a specific site, group of sites, or all sites. This entry must not be given as part of an initializing job stream.

Upon receipt of this entry, the scheduler indicates to the input own code routine that further jobs received from the specified site (or sites) are to be ignored. After all jobs which are currently active for the site are completed, the line is freed until the next time period.

- Format:

  The format of the STOP entry made from a remote site is as follows:

  STOPƀƀidentity

- Specifications:

  Identity is the five character alphanumeric identity which was given to the remote site which is to be terminated.

- Method of Submission:

  The STOP entry may be made as part of an incoming transmission from a remote site and as an unsolicited on-site operator type-in. In this case, the absence of the identity specification causes all sites to terminate. Otherwise, only the site or group of sites identified is terminated. The on-site type-in must be prefixed by the job number under which the scheduler is running; for example:

  Jx¤ STOPƀƀidentity, identity, etc Ⓢ

  where x is the scheduler's job number.

9.5.6. HOLD Entry

The HOLD entry causes the scheduler to hold all data after or before current jobs until the next scheduled time period; however, input from the site (or sites) specified can continue for the duration of the current time period.

- Format:

  The format of the HOLD entry made from a remote site is as follows:

  HOLDƀoptionsƀidentity

- Options:

  J – indicates that all output is to be held after the current job.

  Lack of options implies that output will be held upon occurrence of the HOLD entry. In order to abort output for the job currently being transmitted to the remote site, the ABORT1/ABORT2 control cards or key settings must be used.

- Specifications:

  Identity is the five character alphanumeric name which was given to the remote site which is to have its output held.

- Method of Submission:

  The HOLD entry may be made as part of an incoming transmission from a remote site and as an unsolicited on-site operator type-in. In the latter case, the absence of the identity specification causes output for all sites to be held. Otherwise, only the site, or group of sites, specified will have their output held. The on-site type-in must be prefixed by the job number under which the scheduler is running, as:

  Jx¤ HOLDƀƀidentity, identity, etc.

  where x is the scheduler's job number.

9.5.7. TERM Entry

The TERM entry is used to terminate a site, group of sites, or all sites. At any time that all sites have been terminated by this entry, the scheduler exits. Once a TERM type-in has been made for a site, that site cannot be reactivated until termination and restart of the scheduler have occurred.

- Format:

  The TERM entry may be made only as an unsolicited console type-in consisting of the following:

  Jx¤ TERMƀƀidentity, identity, etc. Ⓢ

  where x is the remote scheduler's job number.

- Specifications:

  Identity is the five character alphanumeric identity which was given to the remote site which is to be terminated. Lack of specification indicates that all sites and the remote scheduler are to be terminated.

9.5.8. Error Conditions

Error conditions which can arise during operation of the remote scheduler are given in this section.

9.5.8.1. ERROR CODES

Error codes and their resultant diagnostic messages are given in the following list:

01 *FATAL* SCHEDULE CARD IS NOT FIRST CARD IN DECK

02 *FATAL* SCHEDULE CARD FORMAT ERROR

03 *FATAL* N/V FROM SCHEDULE CARD NOT IN LIBRARY

04 *FATAL* ILLEGAL DIRECTOR

06 CARD IGNORED ILLEGAL DIRECTOR

07 CARD IGNORED ILLEGAL IDENTITY FIELD

10 CARD IGNORED TIME TABLE CURRENTLY FULL

11 CARD IGNORED ILLEGAL FIELD SEPARATER

12 CARD IGNORED SITE ID NOT FOUND

13 CARD IGNORED ILLEGAL FIELD ENTRY

14 CARD IGNORED START/STOP ERROR

15 LASG ERROR

16 LACQ ERROR

17 LFREE ERROR

20 PHONE ERROR

21 BAD STOP OR HOLD ENTRY

22 SITE NOT PRESENTLY ACTIVE

23 CARD IGNORED CAN'T BE UNSTRUNG

24 ILLEGAL OPTION

25 RECORDED OUTPUT UNREADABLE FOR RESTARTED SITE

77 *FATAL* SUB SYSTEM ERROR

9.5.8.2. CONNECTION MESSAGE

The following message informs the operator that a connection is to be made. A positive response indicates that the connection was made by the operator.

CTM XXXXX PH# YYYYYYYYYY

If a connection cannot be made, the response A should be made on the console. The response causes the remote scheduler to abort this attempt. If the time period is not deleted from the time schedule, RS attempts to start the site up in two minutes with a repeat of the above message.

9.5.9. Examples

The following is an example of a job stream that is used to produce and save two remote time schedules, and to start the remote scheduler.

```
#JOB...
#SPURT   REMOTESCHR
000100   SITE 1                  FD*1*SITE1                         SITE ID
000200                           FDS*3*2126331122                   PHONE NUMBER
  .                              FD*12*08:0009:3012:3025:00         CST
  .                              FDS*14*/LASGƀƀCTM34,A,DCTASC,0
  .                              06                                 FILE CODE A
                                 FDS*3*REMSCHOC                     SYSTEM OWN CODE
                                 FSD*3*/LFREEƀƀA

         SITE 2                  FD*1*00002                         SITE ID
                                 FDS*3*6336220                      PHONE NUMBER
                                 FD*1*09:45                         TIME ON CST
                                 FD*1*11:30                         TIME OFF
                                 FD*1*14:30                         TIME ON
                                 FD*1*16:00                         TIME OFF
                                 FD*1*20:00                         TIME ON
                                 FD*1*23:30                         TIME OFF
                                 FD*4*                              OPEN
                                 FDS*14*/LASGƀƀCTM34,B,RMS1LH,0     LASG
                                 07                                 FILE CODE B
                                 FDS*3*                             OWN CODE INSERTED BY RS
                                 FDS*3*/LFREEƀƀ
                                 FDS*3*                             OPEN

#END
#LOADƀƀREMOTESCHR,REMOTESCH/68128
#END
#ASGƀMURƀTAPE,C, BLANK
#OUTƀFƀC,JOB,REMOTESCH
#END
#GOƀƀREMSCH,SYS
SCHEDULEƀƀREMOTESCH/68128,JOB
#END
#FIN
```

An example using the tape output from the above run follows:

```
#JOB...
#ASGƀMURƀTAPE,C,,REMOTESCH
#INƀƀC,REMOTESCH
#END
#GOƀƀRESMCH,SYS
SCHEDULEƀƀREMOTESCH
TIMEƀTƀSITE1,,17:30/19:30
#END
#FIN
```

10. BASIC OPERATIONAL DESCRIPTION

10.1. JOB SUBMISSION

Submission of a JOB deck to the system for processing may be made from a card reader, magnetic tape or paper tape unit local to the UNIVAC 494, or from a remote terminal.

Assuming that the job will be submitted from an on-site card reader, the operator requests from OMEGA the activation of a card reader unit record routine. This is accomplished by the following type-in:

```
URƀName/Version Ⓢ
```

where name/version is the name, and version, if any, of the desired input unit record routine. Absence of name/version implies that the standard primary input unit record routine as determined at systems generation time will be used.

OMEGA, upon seeing the call for unit record (UR) routine, locates the requested UR routine in the system library, assigns the input device, loads and activates the UR routine. The primary input stream from the card reader is entered into the system, and may be composed of any number of job decks, one behind the other. Individual jobs are separated into separate chains by co-operative control as the jobs are entered. For each #JOB statement, an entry is made in the job stack of work to be performed and each entry is eligible for selection. At this time, an entry in the systems log is also made, the task addendum is formed and the #JOB statement is unstrung.

10.2. JOB SCHEDULING

The scheduling elements of OMEGA control the sequencing, setup, and execution of all jobs entering the UNIVAC 494 system. OMEGA is designed to control the execution of an unlimited number of programs in a multiprogram environment while allowing each program to be unaffected by the co-existence of other programs.

The scheduling technique used by OMEGA provides a liberal foundation for the advancement of an installation into the use of multiprogramming techniques. The scheduling technique is easily understood, and any installation can readily modify or extend its capabilities as necessary to meet the particular needs of the installation.

The scheduling routines effect the retrieval of task description statements from the co-operative library, and perform functions necessary for selection and activation of each task. For descriptive purposes, scheduling can be thought of as preselection, selection, and termination. All of the tasks within a job deck are processed in the order in which the tasks enter the system. However, the processing of a job may be interrupted between tasks to allow a task from another job to be processed. Such a situation generally occurs whenever consecutive tasks of a job have sufficiently different primary storage or facility requirements that the succeeding task in the stream cannot be activated. In this case, the scheduling routines may select a task from another job whose requirements permit it to be activated at this time. For this reason, the procedure for initiating each succeeding task of a job is essentially the same as that for the first task of the job.

The tasks involving SPURT, library maintenance and the Loader call are normally processed consecutively; however, the processing of the task involving the execution of the allocated program may or may not immediately follow collection depending upon availability of required facilities or primary storage requirements.

10.2.1. Preselection

The function of preselection is to obtain explicit and implicit scheduling parameters for the next task within a job deck and to summarize facilities required to activate the task. Preselection is activated upon entrance of a job deck into the system or upon termination of the previous task. All executive control statements down to and including the task activation statement are summarized to provide selection parameters. The control statements, normally #ASG, #LOG, or #PRAM cards, can be presented to selection by one of two ways, described as follows, or as a combination of both.

- Primary Input Stream: The normal method for submitting schedule cards is through the primary input stream, immediately preceding the task activation card to which the schedule cards pertain.

- Collected Program: A second method for submission of schedule cards is at the time that RB elements are collected into an object program. The Loader recognizes an option to collect control statements as part of the preamble of the collected program. All systems processors, utility routines, assemblers and compilers use this method of facility assignment.

  Preselection, upon locating the task program, summarizes all control statements contained in the preamble of the program with statements in the control stream. #ASG control statements contained in the primary input stream override those statements in the preamble at any time when duplication exists in assignment of a device to a particular file code. This feature allows the user to attach all facility assignments to the object program for simplicity of submission at execution time, while providing him with an override mechanism at execution time for abnormal assignments.

10.2.2. Control

Control statements which cause the activation of a task are as follows:

- #GO, used to activate a user program.

- Calls for system assemblers and compilers, used to translate source code to relative binary:

  (a) #SPURT (used to activate the UNIVAC 494 SPURT assembler)

  (b) #FOR (used to activate the FORTRAN compiler)

  (c) #COB (used to activate the COBOL compiler)

  (d) #ASM (used to activate the UNIVAC 494 ASM assembler)

- Calls for system processors such as:

  (a) #LOAD (used to activate the UNIVAC 494 Loader)

- Calls for system utility routines such as:

  (a) #IN, #OUT, #PRT, and #LINK (used to activate the Program Library Editor)

  (b) #TEST (used to activate the Test System package)

  (c) #ELM (used to form and write a bootstrap block)

(d) #REX (used to activate the UNIVAC 490 REXecutor)

(e) #UTL (used to activate utility routines)

In summary, preselection performs the following steps as part of the scheduling algorithm for the operating system:

(1) Collects control statements from the primary input stream down to the task activation statement.

(2) Locates the called-for task program in the systems or job library. (Group libraries are considered extensions to the job library).

(3) Collates control statements from the preamble with those contained in the control stream, and resolves any ambiguities which may exist.

(4) Extracts all scheduling parameters for the task program such as primary storage, direct access storage and peripheral device requirements.

(5) Enters the task, and its summary control statements and requirements, onto the selection queue.

10.2.3. Selection

The selection function consists of the selection, setup, and execution of tasks from the backlog of tasks provided by preselection. Selection ascertains which tasks may be activated by criteria relative to the available facilities. The prime criterion is the priority specified on the #JOB control statement. Within any one priority class, a finer breakdown is performed by selection to determine the eligibility of a task for execution. This breakdown is performed by application of the following tests to all tasks within a primary group, given in their order of importance:

(1) Next task of a partially completed job deck,

(2) Task requiring the greatest number of facilities, peripheral units, direct access storage or primary storage, and

(3) Time of arrival in the system on a first in, first out basis.

Once a task is selected, the task is eligible for processing of its scheduled control statements as collected by preselection. This normally involves the assignment of facilities to the selected task; storage of parameters as submitted by the #PRAM control statements; and special operator instructions conveyed by the #MSG control statements.

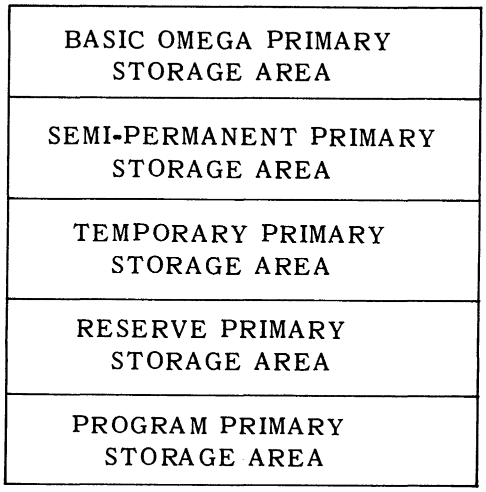
Peripheral units are assigned to the task prior to its execution allowing the computer operator to: mount specific tape reels on specific tape units; supply input card decks or paper tape rolls; and supply specific card punch or line printer forms prior to allocation of primary storage and execution of the task. Upon completion of all manual intervention, core storage is allocated, and the task program is entered from the direct access storage library and is activated.

10.3. ALLOCATION AND MAINTENANCE OF PRIMARY STORAGE

The following paragraphs apply to the allocation and maintenance of primary storage during execution of tasks under executive control of OMEGA.

OMEGA allocates and maintains five areas of primary storage. These five areas are as follows:

address 0



BASIC OMEGA PRIMARY STORAGE AREA

SEMI-PERMANENT PRIMARY STORAGE AREA

TEMPORARY PRIMARY STORAGE AREA

RESERVE PRIMARY STORAGE AREA

PROGRAM PRIMARY STORAGE AREA

- Basic OMEGA Primary Storage Area

  This is the area of primary storage required to hold the resident Operating System, and includes basic tables, interrupt processors, and executive routines. The length of this area is established at systems generation time. The area is located in lower primary storage, starting at address zero. The size of this area usually depends upon the number of ESI buffer control registers in the configuration, and upon whether the communications director is included in the operating system.

- Semi-Permanent Primary Storage Area

  The area allocated to semi-permanent primary storage is small in size and semi-permanent in nature. The size of the area is determined at systems generation time. If the area allocated is insufficient during peak periods of operation, OMEGA may expand the area into reserve primary storage or program primary storage. The semi-permanent primary storage area is used for remote line handlers, unit record routines, on-site handlers, ESI primary storage buffers, and all addendums and storage modules.

- Temporary Primary Storage Area

  The size of the temporary primary storage area is determined at systems generation time. Usage of the area is restricted to routines and buffers that, if necessary, can be purged to ensure sufficient primary storage area for loading the compaction routine during processing of a temporary primary storage extension. The temporary primary storage area can be expanded into reserve primary storage or program primary storage, but this expansion occurs only under unusual conditions; that is, when everything possible has been purged from the area, and a request for primary storage still cannot be satisfied. The temporary primary storage area is used for secondary OMEGA elements as called for, and for I/O co-operatives and communications director buffers.

- Reserve Primary Storage Area

  The reserve primary storage area is used for extensions from other areas and worker programs. The size of this primary storage area is a systems generation parameter.

- Program Primary Storage Area

  The program primary storage area is used for worker programs and to provide extension area for other primary storage areas when required.

Batch programs are loaded starting at the high address in the area, and real time programs are loaded at the beginning of the area. This loading arrangement allows OMEGA practically to assure the success of real time jobs when dynamically adding to the primary storage limits by using the MADD$ service request. This loading arrangement also makes primary storage compaction more feasible, since real time programs are not amenable to compaction. The compaction process involves moving of programs to locations as high as possible in primary storage or rollout. The #CORE control card with R option may be used to declare the need for memory expansion by a real time job. If this is done, anything loaded in the expansion area while it is not in use will be rolled out to satisfy MADD$ request.

APPENDIX A. ELEMENT FORMAT AND DESCRIPTION

The element is the basic logical unit of information which is processed through the UNIVAC 494 Real Time System. Individual and collected elements may serve as input to the system components or may be output on various media. The elements may also be stored internally and manipulated through libraries and associated routines of the system.

- Element Types

  An element may be one of three types:

  (1) Relative Binary (RB) Element

  Relative binary (RB) elements are intermediate output codes generated by Language Processors, such as UNIVAC 494 SPURT, UNIVAC 494 ASM, COBOL and FORTRAN, from source language input. RB elements serve as input to the Loader which transforms them into Load elements.

  (2) Load Element

  Load elements are object or executable program components produced by the Loader through the collection and loading process which combines RB elements.

  (3) Source Element

  Source elements are input program or subprogram components, comprising a series of card images, the contents of which are dictated by the use for which they are intended; e.g., as input to language processors for generating RB elements, as secondary control statements for processors, or as complete or partial job streams.

- General Format

  All elements have a similar general format which comprises descriptive information for identifying and locating the element in a file, declarative and/or control information for defining the element, and textual information carried by the element. Differences in individual formats are occasioned by differences in element functions and content, type of media involved in storage and handling, appendage of modifications and segmentation information, and other variables. A description of the various formats, their included fields and the information contained, is given in the following sections, together with an explanation of their purposes.

A.1. RELATIVE BINARY (RB) ELEMENT (Element Type 1)

All UNIVAC 494 language processors produce relative or relocatable binary (RB) elements for assimilation into absolute program components. The RB elements represent processed source language programs or subprograms, which may be complete, or which may be dependent upon collection with other elements to form the executable programs. RB output may be stored on mass storage devices or cards.

RB elements are input to the Loader which, through collection and allocation of elements, and resolving of references and cross references, completes synthesis of the executable program. The Loader output may be base relative (to zero) and directly executable on computers operating in the UNIVAC 494 mode under control of OMEGA. The output may also be base absolute and indirectly executable; that is, subject to base modification, for use on computers operating in the UNIVAC 490 mode (without the use of the Relative Index Register).

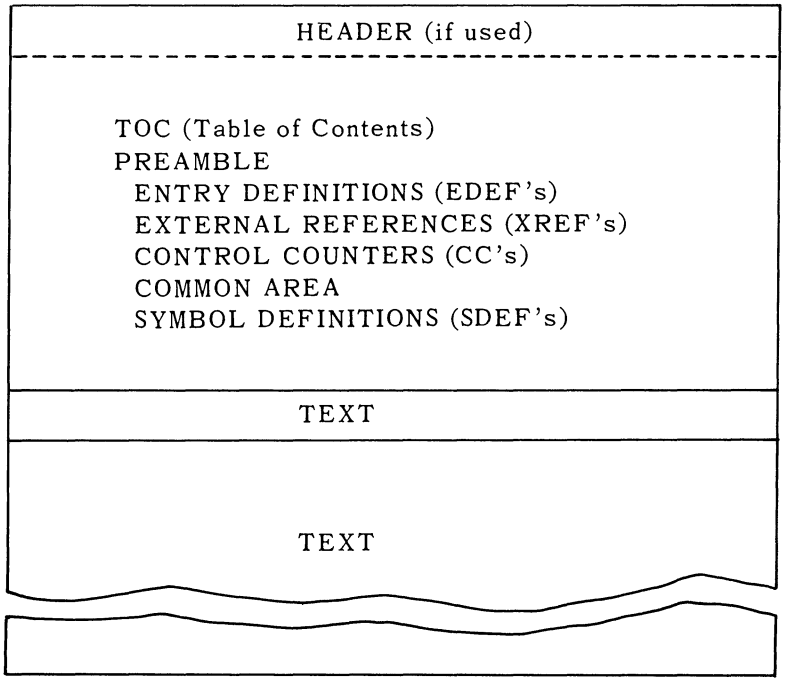
A.1.1. Element Organization

The RB code for an element consists of a table of contents (TOC), preamble and text. The basic order and structure is retained in the libraries and on external media. However, when the RB element is output on tape and cards, initial header or identification information is added (see tape and card format).

The TOC contains the element name and version, assigned by the language processor, together with information describing the element. The preamble supplies declarative information, such as references, definitions, common information areas, and starting points, and is necessary to the collection process in forming the absolute program from a number of elements. The text contains the machine instructions generated by the processor, together with codes necessary for directing the instruction modification at collection time. All parts of the RB element are internally stored in Fieldata (FD) code; however, some parts of the element may be interpreted as series of bits rather than FD characters.

- General Format

  The relative positions of the various portions of the RB element may be represented as follows:



HEADER (if used)

TOC (Table of Contents)
PREAMBLE
ENTRY DEFINITIONS (EDEF's)
EXTERNAL REFERENCES (XREF's)
CONTROL COUNTERS (CC's)
COMMON AREA
SYMBOL DEFINITIONS (SDEF's)

TEXT

TEXT

A description of the components of the RB element, and the fields within them, is given in the following subsections.

A.1.1.1. TOC (Table of Contents)

The TOC serves as the element descriptor, and is generated for use by OMEGA in storage and retrieval of the element. The TOC may be internal, stored within the system, or external, as part of the card or tape output. The TOC contains the following information:

0	0 0 0 0 1	ERROR INDICATOR
1	N N N N N	
2	N N N N N	
3	V V V V V	
4	INCREMENT TO ELEMENT BASE	
5	INDEX TO XREF	INDEX TO CC
6	INDEX TO COMMON	INDEX TO SDEF
7		INDEX TO TEXT
10	LENGTH OF TEXT	

Word 0 Element type, in the upper portion, indicating whether the element is RB, load or source. Type 1 indicates an RB element.

Error indicator, in the lower portion, which indicates the type of errors, if any, which may be present in the element.

1–2 Name of the element, comprising 1–10 alphanumeric characters, left justified, as assigned by the language processor during translation. This name is used in referring to the element on control statements, in libraries, etc. All unused portions of the field are space filled.

3 Version of the named element, comprising 1–5 alphanumeric characters, left justified, as assigned for distinguishing between adaptions of elements bearing the same name. Unused portions of the word are space filled. If no version is given, the entire word is space filled. When an element is referenced by control statements, the version field will be checked only if a version is specified on the statements.

4 Increment to element base, assigned relative to the base of the file in which the element is contained.

5 Index to external references (XREF's), in the upper portion, relative to the incremented element base (Word 4), specifying the start of the XREF's. The value reflects the length of the entry definitions (EDEF's).

Index to control counters (CC's), in the lower portion, relative to the incremented element base, specifying the start of the CC's. The value reflects the length of the XREF's and EDEF's.

6 Index to common, in the upper portion, relative to the incremented element base, specifying the start of the common area of information shared between elements. The value reflects the length of the CC's, XREF's and EDEF's.

Index to symbol definitions (SDEF's), in the lower portion, relative to the incremented element base, specifying the start of the SDEF's. The value reflects the length of the common, CC's, XREF's and EDEF's.

7 The upper portion of this word is reserved. Index to text, in the lower portion relative to the incremented element base, specifying the beginning of the text. The value reflects the total length of the preamble (see A.1.1.2).

10 Length of text, including data, instructions and modifications, for the named element.

NOTE: The TOC for the output media is identical to the above; however, because the header information occupies the first 10_8 words of the header block, the TOC begins in word 11 and continues through word 21.

- Example:

A sample TOC module may be represented as follows (see the sample source code for RB text, Figure A–1, for reference to the entries).

Word 0	0 0 0 0 1	0 0 0 0 0	0	(Upper) Type Code (Lower) Error Indicator
1	S A M P L		1,2	(Whole) Name of the element – 1 to 10 characters (space filled)
2	E ƀ ƀ ƀ ƀ			
3	ƀ ƀ ƀ ƀ ƀ		3	(Whole) Version – 1 to 5 characters (space filled)
4	0 0 0 0 0 0 0 2 0 4		4	(Whole) Increment to element base
5	0 0 0 0 6	0 0 0 1 4	5	(Upper) Index to external references (XREF's) (Lower) Index to control counters (CC's)
6	0 0 0 2 3	0 0 0 3 1	6	(Upper) Index to common information area (Lower) Index to symbol definitions (SDEF's)
7	0 0 0 0 0	0 0 0 3 7	7	(Upper) Reserved (Lower) Index to text
10	0 0 0 0 0 0 0 0 3 4			(Whole) Length of text

A.1.1.2. PREAMBLE

The preamble is a table of indefinite length comprising lists of element references and reference points necessary in the collection process, and contains the following information:

ENTRY DEFINITIONS (EDEF's)
EXTERNAL REFERENCES (XREF's)
CONTROL COUNTERS (CC's)
COMMON AREA
SYMBOL DEFINITIONS (SDEF's)

A.1.1.2.1. Entry Definition (EDEF)

The entry or external definition (EDEF) defines an entry point or constant for an element at which the element may be referenced by other elements, and is used to satisfy external references (XREF's) from these elements.

An element may contain no, one or many EDEF's.

The EDEF entry is as follows:

0	N	N	N	N	N
1	N	N	N	N	N
2	i	CONTROL COUNTER NUMBER			VALUE

Word 0–1 Name of the EDEF, comprising 1–10 alphanumeric characters, left justified, defining the entry point. The name is compared to that of an XREF in satisfying the reference.

2 An indicator, in bit 29, showing whether the associated value is a constant or is relative, with modification being required by the base of the element containing the EDEF. If the bit is positive (0), modification will be made relative to a control counter (CC).

Control counter (CC) number, in bits 28–15, indicating the CC under which the EDEF is defined. If bit 29 is positive, the base assigned to the CC will be used in modification of the value of the EDEF.

Value, in bits 14–0, as assigned to the EDEF, representing a constant value or a position relative to the base of the element in which the EDEF is contained.

A.1.1.2.2. External Reference (XREF)

The external reference (XREF) invokes a reference point, external to the element being translated, but necessary for its execution, which is not defined at assembly or compilation time. The name of the reference, label or tag, is specified in the RB output so that the element may be associated with the reference at collection or load time. The reference must always be externally defined (by the EDEF), and each XREF must be matched with its EDEF before the element can be executed. An element may have no, one or many XREF's.

The XREF entry is as follows:

0	N	N	N	N	N
1	N	N	N	N	N

Word 0–1 Name of the XREF, comprising 1–10 alphanumeric characters, left justified. The name will be compared against that of an EDEF in satisfying a reference. In text modification, the XREF's are referenced by a number which relates the order in which they are listed in the RB output.

A.1.1.2.3. Control Counter (CC)

The control counter (CC) entry specifies the total storage in consecutive words of core that is required by a set of code, instructions, data, etc., under control of this counter, and provides the Loader with information for grouping lines of coding. During the loading process, CC's are fixed to the starting address of the assigned area. The CC number is implicit by order within the list, starting with zero.

The CC entry is as follows:

0		VALUE

Word 0 The value in the lower portion, specifying the number of computer words required for a set of code.

A.1.1.2.4. Common Area

A common area specification is used to define an area of information in storage which may be shared by more than one element in a collection. During a collection, a common area assumes the size of the largest definition. An element may have no, one or many common areas. A named or labeled common area may be present at collection time by a block data element.

The common area entry is as follows:

0	N	N	N	N	N
1	N	N	N	N	N
2	GROUP NUMBER		CONTROL COUNTER		

Word 0–1 Name of the common area, comprising 1–10 alphanumeric characters, left justified.

2 Group number, in the upper portion, specifying the type of common area. Normal or labeled common area is indicated by 0; blank common is indicated by 1.

CC number, in the lower portion, specifying the counter which defines the size of the common area.

A.1.1.2.5. Symbol Definition (SDEF)

The symbol definition (SDEF) references a location, tag or label, within the element and is included as part of the RB output under the S option. SDEF's are used by system routines, such as the Test package, in debugging procedures for locating test points and data areas. An element may have no, one or many SDEF's.

The SDEF entry is as follows:

0	N	N	N	N	N
1	N	N	N	N	N
2	CONTROL COUNTER				VALUE

Word 0–1 Name of the SDEF, comprising 1–10 alphanumeric characters, left justified, defining a relative position within the element.

2 CC number, in the upper portion, indicating the counter which defines the relative position of the element containing the SDEF. Value, in the lower portion, specifying the position of the SDEF relative to the base of the element.

A.1.1.2.6. Preamble Length

The preamble precedes the text information in the RB output. The length of the preamble is dependent upon the number of XREF's, EDEF's and other entries which it declares, and is determined through the indices of the TOC, particularly the value in the lower portion of word 6 which reflects the total length of all entries. For direct access storage and tape output, the preamble is distinguished from the text by initial positioning in the data block, which is separated into 16_8 word items. Being of indefinite length, the preamble may occupy more than one item. When the preamble partially fills an item, as when the last part of the preamble overflows from one item to another, the unused portion of the item is space filled. For card output, the preamble is distinguished by the type code A in card column 10.

(a) Example:

A sample preamble may be represented as follows (see the sample source code for RB text, Figure A–1, for reference to these entries).

Word 0	F D B C ƀ		0,1	(Whole) Name of an EDEF-1 to 10 characters (space filled)
1	ƀ ƀ ƀ ƀ ƀ			
2	0 0 0 0 0	0 0 0 0 0	2	(Upper) CC indicator and number (Lower) EDEF value
3	E X I T ƀ		3,4	(Whole) Name of second EDEF
4	ƀ ƀ ƀ ƀ ƀ			
5	0 0 0 0 6	0 0 0 0 6	5	(Upper) CC indicator and number (Lower) EDEF value
6	S U M ƀ ƀ		6,7	(Whole) Name of an XREF–1 to 10 characters (space filled)
7	ƀ ƀ ƀ ƀ ƀ			
10	P R I N T		10,11	(Whole) Name of a second XREF
11	ƀ ƀ ƀ ƀ ƀ			
12	F D B ƀ ƀ		12,13	(Whole) Name of a second XREF
13	ƀ ƀ ƀ ƀ ƀ			
14	0 0 0 0 0	0 0 0 0 7	14	(Upper) Unassigned (Lower) CC 0 (primary storage requirement)
15	0 0 0 0 0	0 0 0 3 0	15	(Lower) CC 1 (primary storage requirement)
16	0 0 0 0 0	0 0 0 2 0	16	(Lower) CC 2 (primary storage requirement)
17	0 0 0 0 0	0 0 0 0 0	17	(Lower) CC 3 (primary storage requirement)
20	0 0 0 0 0	0 0 0 0 0	20	(Lower) CC 4 (primary storage requirement)
21	0 0 0 0 0	0 0 0 0 0	21	(Lower) CC 5 (primary storage requirement)
22	0 0 0 0 0	0 0 0 1 1	22	(Lower) CC 6 (primary storage requirement)
23	B U F F ƀ		23,24	(Whole) Name of a common area of information 1 to 10 characters (space filled)
24	ƀ ƀ ƀ ƀ ƀ			
25	0 0 0 0 0	0 0 0 0 1	25	(Upper) Group number defining type of common (Lower) CC number

Word	Contents	
26	S B U F ƀ	26,27 (Whole) Name of a second common area
27	ƀ ƀ ƀ ƀ ƀ	
30	0 0 0 0 0 │ 0 0 0 0 2	30 (Upper) Group number (Lower) CC number

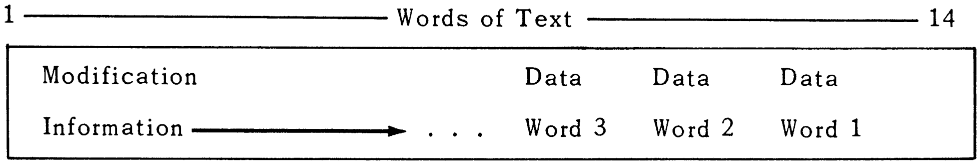
Word	Contents	
31	F D B C ƀ	31,32 (Whole) Name of a SDEF – 1 to 10 characters (space filled)
32	ƀ ƀ ƀ ƀ ƀ	
33	0 0 0 0 0 │ 0 0 0 0 0	33 (Upper) CC number 34,35 (Lower) CC value

Word	Contents	
34	E X I T ƀ	34,35 (Whole) Name of a second SDEF
35	ƀ ƀ ƀ ƀ ƀ	

A.1.1.3. TEXT

The text is composed of machine codes generated from the source language and supplied by the Language Processor. The text is divided into 14_{10}-word groups or items, each containing a variable number of modification bits and associated binary instruction or data words (see A.1.4, Instruction Modification). This procedure enables moving of the text and accomodation of variable length appended information. Modification information extends from the leftmost bit in an uninterrupted stream which specifies the requirements for loading and associated instructions. The instructions being modified start at the right of the image and progress to the left. The amount of modification information determines the number of instruction words which will be contained in the item. A new item is started when insufficient space is available in the middle of the item for another instruction word and its modification bits. The number of items which may be included in the text is dependent upon the length of the element.

The item has the following general form:



A sample RB text, including instructions and modifications, and the source code from which the text is generated, is given at the end of Section 1.

A.1.2. Tape/Drum Output Format

The RB element may be output on Tape/Drum through the OUT statement under various options (see 4.7, Program Library Editor). A header block containing the TOC and other descriptive information is written at the beginning of the element. The preamble and text of the element are written into data blocks. An end sentinel block is written at the end of the element.

The following subsections will discuss header, data and end sentinel blocks, and the information contained.

A.1.2.1. HEADER BLOCK

The header block serves as the element descriptor and is generated for use in storage and retrieval of the element. The size of the block is 30_8 words on tape or 41_8 words on drum. The identity of the element, including the TOC, is given with information for uniquely characterizing the element on the tape file. Reference to the information in the header block will bring the element into the active file.

The header block contains the following information:

0	7 3 7 3 7 3 7 3 7 3	
1	Y Y D D D	
2	N N N N N	
3	N N N N N	
4	V V V V V	
5	REEL NUMBER	
6	H H : M M	
7	MAXIMUM DATA BLOCK SIZE	
10	NUMBER OF ITEMS PER BLOCK	
11	0 0 0 0 1	ERROR INDICATOR
12	N N N N N	
13	N N N N N	
14	V V V V V	
15	INCREMENT TO ELEMENT BASE	
16	INDEX TO XREF	INDEX TO CC
17	INDEX TO COMMON	INDEX TO SDEF
20		INDEX TO TEXT
21	LENGTH OF TEXT	
22	(UNASSIGNED)	
23		
24		
25		
26		
27	7 3 7 3 7 3 7 3 7 3	

Word 0 Beginning sentinel.

1 Date (from console type-in) of the output, comprising the year, YY (two digits 00–99) and the day of the year, DDD (three digits, 000–366).

2–3 Name of the element, comprising 1–10 alphanumeric characters, left justified, as assigned by the language processor for use in referencing the element. Unused portions of the words are space filled.

4 Version of the named element, comprising 1–5 alphanumeric characters, left justified, as assigned for distinguishing between adaptions of elements bearing the same name. Unused portions of the word are space filled.

5 Reel number, comprising five digits beginning at 00001, identifying the tape reel upon which the element will be written.

6 Time of day at which the output is written, comprising the hour, HH (two digits, 00–24), and the minute, MM (two digits, 00–60), with the two values separated by a colon (:), as 05:59.

7 Maximum data block size, comprising 1–10 digits, with leading zeros, specifying the maximum number of words which will be written in a data block and may not exceed 306_8.

10 Number of items per block, comprising 1–10 digits, with leading zeros, specifying the number of 16_8-word items which will be written per data block. (The maximum number of items per data block may not exceed 16_8).

11–21 TOC (see A.1.1.1), carrying the same information in the same order as the internal TOC, but with the number sequence starting at 11 rather than 0.

22–26 Unassigned

27 Sentinel

A sample output header block may be represented as follows:

Word				
0	7 3 7 3 7 3 7 3 7 3	0	(Whole)	Beginning sentinel
1	6 7 2 6 2	1	(Whole)	Year and day (YYDDD)
2	S A M P L	2, 3	(Whole)	Name of the element –1 to 10 characters (space filled)
3	E ƀ ƀ ƀ ƀ			
4	ƀ ƀ ƀ ƀ ƀ	4	(Whole)	Version –1 to 5 characters (space filled)
5	2 1 0 0 8	5	(Whole)	Reel number
6	1 1 : 5 4	6	(Whole)	Time of day (HH:MM)
7	0 0 0 0 0 0 0 3 0 6	7	(Whole)	Maximum data block size
10	0 0 0 0 0 0 0 0 1 6	10	(Whole)	Number of items per block
11	0 0 0 0 1 | 0 0 0 0 0	11,21	Table of contents (TOC) of the element (Upper) (Lower)	 Type Code Error Indicator
12	S A M P L	12,13	(Whole)	Name of the element
13	ƀ ƀ ƀ ƀ ƀ			
14	ƀ ƀ ƀ ƀ ƀ	14	(Whole)	Version
15	0 0 0 0 0 0 0 2 0 4	15	(Whole)	File increment to element base
16	0 0 0 0 6 | 0 0 0 1 4	16	(Upper) (Lower)	Index to XREF's Index to CC's
17	0 0 0 2 3 | 0 0 0 3 1	17	(Upper) (Lower)	Index to common Index to SDEF's
20	0 0 0 0 0 | 0 0 0 3 7	20	(Upper) (Lower)	Reserved Index to text
21	0 0 0 0 0 0 0 0 3 4	21	(Whole)	Length of text
22		22,23	Unassigned	
24		24	(Whole)	Logical increment to be used for # IN function (Drum File)
25		25	(Whole)	Logical increment is to be used for # OUT function (Drum File)
26		26	Unassigned	
27	7 3 7 3 7 3 7 3 7 3	27	(Whole)	Sentinel

A.1.2.2. DATA BLOCK

The preamble and the text are written into data blocks having a maximum size of 306_8 words. The number of blocks used and the block size are dependent upon the element length. The different parts of the element are not separated into different types of data blocks. However, the element is separated into 16_8-word items within the data blocks. This procedure enables moving of the text and accommodation of appended information.

The data block is filled on the basis of the number of complete 16_8-word items which the block may receive, the maximum being 16_8. An item is not split between blocks. When an item contains less than 16_8 words, as may be the case when the preamble overflows from one item to another, the remainder of the item is space filled. The preamble is distinguished from the text by initial positioning in the first data block.

The data block contains the following information:

0	ITEM SIZE
1	PREAMBLE (if carried in the block)
. .	
n	TEXT
. .	
305 (or last)	CHECK SUM

Word 0 Item (upper): the number of 16_8-word items within the data block.

Size (lower): the total length of the data block.

1 Preamble, including EDEF's, XREF's, CC's, commons, and SDEF's, if carried in the element, in the form of 16_8-word items and comprising as many items as necessary.

n Text, including modification and instruction, beginning after the last item of the preamble or at the start of the block if the preamble is not present, in the form of 16_8-word items and continued to the last word of the block.

305 Check sum: the complement of the sum of all words in the data block with the exception of the check sum entry. When the block is less than 306_8 words, the check sum is entered into the last word of the block (the first word after the end of the last text item).

A sample data block containing a preamble and text may be presented as follows:

<table>
<tr><td>Word 0</td><td>0 0 0 0 4</td><td>0 0 0 7 2</td><td>0</td><td>(Upper) Number of 16_8 in block
(Lower) Block size</td></tr>
<tr><td>1</td><td colspan="2">F D B C ƀ</td><td>1,37
1,2</td><td>Preamble (two 16_8-word items)
(Whole) Name of an EDEF</td></tr>
<tr><td>2</td><td colspan="2">ƀ ƀ ƀ ƀ ƀ</td><td></td><td></td></tr>
<tr><td>3</td><td>0 0 0 0 0</td><td>0 0 0 0 0</td><td>3</td><td>(Upper) CC indicator and number
(Lower) EDEF value</td></tr>
<tr><td>4</td><td colspan="2">E X I T ƀ</td><td>4,5</td><td>Name of second EDEF</td></tr>
<tr><td>5</td><td colspan="2">ƀ ƀ ƀ ƀ ƀ</td><td></td><td></td></tr>
<tr><td>6</td><td>0 0 0 0 6</td><td>0 0 0 0 6</td><td>6</td><td>(Upper) CC indicator and number
(Lower) CC value</td></tr>
<tr><td>7</td><td colspan="2">S U M ƀ ƀ</td><td>7,10</td><td>(Whole) Name of an XREF</td></tr>
<tr><td>10</td><td colspan="2">ƀ ƀ ƀ ƀ ƀ</td><td></td><td></td></tr>
<tr><td>11</td><td colspan="2">P R I N T</td><td>11,12</td><td>(Whole) Name of a second XREF</td></tr>
<tr><td>12</td><td colspan="2">ƀ ƀ ƀ ƀ ƀ</td><td></td><td></td></tr>
<tr><td>13</td><td colspan="2">F D B ƀ ƀ</td><td>13,14</td><td>(Whole) Name of a third XREF</td></tr>
<tr><td>14</td><td colspan="2">ƀ ƀ ƀ ƀ ƀ</td><td></td><td></td></tr>
<tr><td>15</td><td>0 0 0 0 0</td><td>0 0 0 0 7</td><td>15</td><td>(Upper) Unassigned
(Lower) CC value for</td></tr>
<tr><td>16</td><td>0 0 0 0 0</td><td>0 0 0 3 0</td><td>16</td><td>(Lower) CC value for</td></tr>
<tr><td>17</td><td>0 0 0 0 0</td><td>0 0 0 2 0</td><td>17</td><td>(Lower) CC value for</td></tr>
<tr><td>20</td><td>0 0 0 0 0</td><td>0 0 0 0 0</td><td>20</td><td>(Lower) CC value for</td></tr>
<tr><td>21</td><td>0 0 0 0 0</td><td>0 0 0 0 0</td><td>21</td><td>(Lower) CC value for</td></tr>
<tr><td>22</td><td>0 0 0 0 0</td><td>0 0 0 0 0</td><td>22</td><td>(Lower) CC value for</td></tr>
<tr><td>23</td><td>0 0 0 0 0</td><td>0 0 0 1 1</td><td>23</td><td>(Lower) CC value for</td></tr>
<tr><td>24</td><td colspan="2">B U F F ƀ</td><td>24,25</td><td>(Whole) Name of a common area of information</td></tr>
<tr><td>25</td><td colspan="2">ƀ ƀ ƀ ƀ ƀ</td><td></td><td></td></tr>
<tr><td>26</td><td>0 0 0 0 0</td><td>0 0 0 0 0</td><td>26</td><td>(Upper) Group number defining type of common
(Lower) CC number</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>27</td><td colspan="2">S B U F ƀ</td><td>27,30</td><td>(Whole) Name of a common area</td></tr>
<tr><td></td><td colspan="2"></td><td></td><td></td></tr>
<tr><td>30</td><td colspan="2">ƀ ƀ ƀ ƀ ƀ</td><td></td><td></td></tr>
<tr><td>31</td><td colspan="2">0 0 0 0 0 0 0 0 0 2</td><td>31</td><td>(Upper) Group number
(Lower) CC number</td></tr>
<tr><td>32</td><td colspan="2">F D B C ƀ</td><td>32,33</td><td>(Whole) Name of an SDEF</td></tr>
<tr><td>33</td><td colspan="2">ƀ ƀ ƀ ƀ ƀ</td><td></td><td></td></tr>
<tr><td>34</td><td colspan="2">0 0 0 0 0 0 0 0 0 0</td><td>34</td><td>(Upper) CC number
(Lower) CC value</td></tr>
<tr><td>35</td><td colspan="2">E X I T ƀ</td><td>35,36</td><td>(Whole) Name of a second SDEF</td></tr>
<tr><td>36</td><td colspan="2">ƀ ƀ ƀ ƀ ƀ</td><td></td><td></td></tr>
<tr><td>37</td><td colspan="2">0 0 0 0 6 0 0 0 0 6</td><td>37</td><td>(Upper) CC number
(Lower) CC value</td></tr>
<tr><td>40–52</td><td colspan="2">ƀ ƀ ƀ ƀ ƀ</td><td>40-52</td><td>(Space fill to complete item)</td></tr>
</table>

53	2 0 0 0 0 0 3 0 2 5	53 53	Text (16$_8$-words per item) Modification Code
54	4 3 0 0 3 1 2 5 4 1		
55	1 0 0 0 0 0 6 0 0 6		
56	6 3 1 4 4 3 4 2 0 0		
57	0 0 0 0 0 0 0 0 0 0	57	First data word; last instruction word of item
60	7 7 4 5 0 0 0 0 0 0		
61	7 7 4 5 0 0 0 0 0 0		
62	1 2 7 0 0 0 0 0 0 0		
63	1 2 4 0 0 0 0 0 0 0		
64	0 2 0 0 0 0 0 0 0 1		
65	2 1 0 0 0 0 0 0 0 2		
66	6 0 7 0 0 0 0 0 0 6		
67	7 7 5 4 0 3 0 0 0 1		
70	1 2 7 0 0 0 0 0 0 0	70	First instruction word; last word of item
71	5 4 3 0 0 3 1 4 0 3	71	Modification code beginning second 16$_8$-word item of text
72			
73			
74			
75			
76			
77			
100			
101	7 7 5 4 0 0 0 0 0 5	101	Last instruction word of second text item; first data word of item
102	7 7 4 5 0 0 0 0 0 0		
103	1 1 0 3 0 0 0 0 0 2		
104	6 1 0 0 0 0 0 0 0 0		
105	7 2 4 0 0 0 0 0 0 0		
106	1 2 7 0 7 0 0 0 0 2	106	First instruction word of second item of text; last word of second item
107	1 3 2 1 4 5 6 7 2 1	107	Check sum

A.1.2.3. END SENTINEL BLOCKS

An end sentinel block is written at the end of each output element. The end sentinel block comprises 30_8 words, the first and last of which contain end sentinels (7's and 6's), the remaining words being unassigned.

The end sentinel block may be presented as follows:

Word 0	7 6 7 6 7 6 7 6 7 6	0, Beginning end sentinel
1		1, 26 Unassigned
.		
.		
.		
27	7 6 7 6 7 6 7 6 7 6	27 Last end sentinel

A.1.3. Card Output Format

The RB element may be output on cards by the OUT function under a C option (see 4.7, Program Library Editor). An identification field, sequence field and card type field are appended to each image punched. The output is punched on the basis of 80 columns of information.

The RB element output card has the following form:

1 5	6 9	10	11	80
IDENTIFICATION	SEQUENCE NO.	TYPE	INFORMATION	

Column 1–5 Identity, comprising the first five characters of the name of the element. The contents of this field may be used for deck identification.

Column 6–9 Sequence number, comprising four decimal characters, which will be checked if the cards are used as input (by the IN function, under the C option).

Column 10 Type code which is alphabetic. A preamble card is indicated by A; a text card is indicated by B.

Column 11–80 Information carried in the element, comprising the preamble or the text.

The terminal card of the punched output will be a slash (/) card containing slashes (/////) in columns 1–5.

A.1.4. Instruction Modification

Modification information is included in the RB element by the language processor for directing the collection process. The information determines modification of data words, the loading address at which one or more data words will be stored, and the starting address of the element.

There are four cases of major control in the bit stream from the initial state:

00 End-of-stream: start a new image

01ar Address: a is a 15-bit unadjusted address, and r is a reference number of 2–12 bits defining the control counter (CC) or XREF used in the modification. If r>511, then a + CC (r – 512) is the starting address of the element.

10 No modification: load data word.

11m Modification: modify according to the string m.

The modification sequence for a specific word or group of words is of the form:

i1i1...i0

where i is the signal for a single modification. For each modification, the signal is followed by 1, with 01 being a new address and 11 being a new instruction. The modification sequence or string is ended by 0. Thus, '00,10' combinations end the modification string immediately.

The modification string, m, for each application is of the form:

f s t r

f indicates the field that is to be modified, and is of variable length. When f is 0, modification is applied to bits 14–0 of the instruction; when f is 10, modification is applied to bits 29–15; and when f is 11, the next bits define the portion of the word which will be modified.

s is the sign of the modification. Plus is indicated by 0; minus is indicated by 1.

t is the modifier type. A CC is indicated by 0; and an XREF is indicated by 1.

r is a reference number of 2–12 bits specifying the number of the CC or XREF which will be used. If the first two digits (left) are 00, CC or XREF 0 is indicated; if the digits are 01, CC or XREF 1 is indicated; if the digits are 10, CC or XREF 2 is specified; if the digits are 11, the CC or XREF specified in the following 10 bits is used.

Figure A–1 is a sample source code for RB text.

RB element modification is illustrated in the sample RB text of Figure A–2.

```
#SPURT  RB SAMPLE
ERROR   CC LOC    FFJKB   YYYYY  MU ML   CARD    LABEL  STATEMENT                                           A1  6

                                         000100         'CONVERT,SUM,PRINT FD DECIMAL-SAMPLE
                                         000200         'INST                       NOTES                   MOD NO
                                         000300         EDEF*FDBC*EXIT              'ENTRY DEFINITIONS
                                         000400         START*FDBC                  'START ADDRESS
                                         000500         XREF*SUM*PRINT*FDB          'EXTERNAL REFERENCES
        00 00000  12700   00000      01  000600  FDBC   CARD$*BUFF                  'READ DATA              M1,M2
           00001  77540   30001
           00002  60700   00006      06  000700         JP*EXIT*ANEG                'EXIT ON NEGATIVE       M3
           00003  21000   00002          000800         SUB*A*2                     'SET LOOP COUNT         M4
           00004  02000   00001          000900         RSH*A*1                     'SUB 2 and DIV BY 2     M5
           00005  12470   00000          001000         ENT*B4*A                    'COUNT TAKING 2 WORDS   M6
           00006  12700   00000      01  001100         ENT*B7*BUFF                 'ADDRESS DATA           M7
                                         001150         CC*6
                                         001160         'SWITCH CC SO DATA CHECK MAY BE ADDED
        06 00000  77450   00000      X   001200         EBJP*B5*FDB                 'CONVERT FD DEC TO B    M8
           00001  77450   00000      X   001300         EBJP*B5*SUM                 'SUBMIT TO SUM ROUT     M9
           00002  00000   00000  02      001350         U-TAG*SBUF*Q                'SUM BUFF AND COUNT     M10
           00003  12707   00002          001400         ENT*B7*B7+2                 'ADVANCE PICK UP ADDR   M11
           00004  72400   00000      06  001500         BJP*B4*$-4                  'PROCESS NEXT 2         M12
           00005  61000   00000      00  001600         JP*FDBC                     'GET ANOTHER CARD       M13
           00006  11030   00002      06  001700  EXIT   ENT*A*W($-4)                'SUM DATA DESC
           00007  74450   00000      X   001750         EBJP*B5*PRINT               'PRINT DATA
           00010  77540   00005          001800         RETURN$                     'GIVE UP CONTROL        M14
                                         003600  BUFF   COMMON*1                    'CARD BUFFER
                                         003700         CC*1
        01 00000                         003800  BUFF   RESERVE*30
                                         003900  SBUF   COMMON*2
                                         004000         CC*2
        02 00000                         004100  SBUF   RESERVE*20                  'SUM BUFFER
```

LEGEND

ERROR	Error code
CC	Control counter
LOC	Relative address under current control counter
FFKKB YYYYY	Assembler instructions (composed of f, j, k and b designators)
MU	Modification upper half-word (number refers to control counter; X refers to XREF (external reference))
ML	Modification lower half-word (number refers to control counter; X refers to XREF (external reference))
CARD	Card number
LABEL	Source statement label
MODIFICATION NUMBER	Modification

Figure A–1. Sample Source Code for Relative Binary Text

```
OCTAL FORMAT:

1              2              3              4              5              6              7              10
20000  03025   43003  12541   10000  06006   63144  34200   00000  00000   77450  00000   77450  00000   12
Modification Information                                                                        Instructions

                                             11             12             13             14             15
                                             12470  00000   02000  00001   21000  00002   60700  00006   77
                                                                            Instructions          ...     Words

Modification information extends from left to right in an unbroken bit stream; instructions,
words fill in from right to left.  When space is insufficient for a new word and its associate
item is started. Unused portions of an item are zero filled.

BINARY FORMAT:

Modification Bit Stream

1

                    m1        m2m3 2                m4m5m6m7
010000000000000000011000010101100011000000011001010101100001
A a                r F fstr DC F fstr B             DC C C   fstr

3                               4
                                m8      m9      m10
001000000000000000110000000110110011001100100011100010000000
DA a                r B         F fstr DF fstr DF f str D E

LEGEND

A    New address : take next 15 bits
a    15 bit new address
r    Reference number defining control counter or XREF: r=00, use 0; r=01, use 1;
     r=10, use 2; r=11, take next ten bits (B)
B    Number of control counter or XREF (for r=11)
C    No modification:  load data word
D    End of modification string
E    End of group modification
F    Modification string follows:  f, s, t, r
f    Field definition: f=0, modify bits 14–10; f=10, modify bits 29–15; f=11, next
     10 bits define portion of word to be modified
s    Sign of modification:  0=plus; 1=minus
t    Modifier type:  0=control counter;  1=XREF
```

Figure A–2. Sample Relative Binary Text – 16_8-word Item

A.2. LOAD ELEMENT (Element Type 2)

The load element is a collection of RB elements which have been modified and interconnected to form the executable program. The collection is performed by the Loader under control of OMEGA (see 4.6).

The load element may take one of two forms dependent upon the presence or absence of the M option on the LOAD statement at the time of collection. When the M option is not used, an absolute element is produced, with base relative to 0, which is directly executable under OMEGA on the UNIVAC 494 computer. Upon execution, the Relative Index Register (RIR) of the computer will be set to the address at which the element is read into primary storage, eliminating the need for modification. If the M option is used, a relative element is produced, with base relative to an absolute address, which is designed for execution on computers that do not use an RIR and which operate in the UNIVAC 490 mode. The relative element is not directly executable until modification of the instructions is made to an operating base.

A.2.1. Element Organization

The absolute or relative load element consists of a table of contents (TOC), control statements which will be processed at execution time, instructions for collected elements, and symbol definitions (SDEF's) which may be used in a debugging environment (see Section 6). The basic order and structure is retained in the libraries and on external media. However, when the load element is output on tape and cards, initial header or identification information is appended. For both internal and external storage, segment descriptor information is appended in the case of segmentation of the element.

The TOC contains the element name and version assigned on the LOAD control statement, together with information describing the element. The control statements, which are not necessarily present in all load elements, consists of secondary control statements, such as imbedded ASG or MSG statements, which were included with the secondary control statements for the Loader but were not involved in the collection process. These statements are appended to, or imbedded in, the element so that they may be processed at the time that the element is read into core for execution. The element instructions comprise the coding for the RB elements collected by the Loader. If the M option were present on the LOAD statement, the instructions would be accompanied by the associated modification information included in the RB elements. The SDEF's are labels from the original source code bearing the relative addresses of locations within the collection of elements. The SDEF's may be used to pinpoint test points and test areas in conjunction with the TEST package. The S option must be present on the LOAD statement if SDEF's are to be produced as part of the load element.

- General Format

  The relative positions of the various portions of the load element may be represented as follows:

HEADER (if used)
TOC (Table of Contents)
CONTROL STATEMENTS
RESIDENT SEGMENT INSTRUCTIONS
SEGMENT 1 INSTRUCTIONS
SEGMENT 2 INSTRUCTIONS
SDEF's (Symbol Definitions) FOR ALL ELEMENTS

  A description of the components of the load element, and the fields within them, is given in the following subsections.

A.2.1.1. TOC (Table of Contents)

The TOC serves as the element descriptor, and is generated for use by OMEGA in storage and retrieval of the element. The TOC contains the following information:

<table>
<tr><td>0</td><td colspan="5">ELEMENT TYPE</td><td colspan="5">ERROR INDICATOR</td></tr>
<tr><td>1</td><td colspan="2">N</td><td colspan="2">N</td><td colspan="2">N</td><td colspan="2">N</td><td colspan="2">N</td></tr>
<tr><td>2</td><td colspan="2">N</td><td colspan="2">N</td><td colspan="2">N</td><td colspan="2">N</td><td colspan="2">N</td></tr>
<tr><td>3</td><td colspan="2">V</td><td colspan="2">V</td><td colspan="2">V</td><td colspan="2">V</td><td colspan="2">V</td></tr>
<tr><td>4</td><td colspan="10">INCREMENT TO ELEMENT BASE</td></tr>
<tr><td>5</td><td colspan="5">CORE REQUIREMENTS</td><td colspan="5">LENGTH CONTROL</td></tr>
<tr><td>6</td><td colspan="5">NUMBER OF SEGMENTS</td><td colspan="5">NUMBER OF CONTROL STATEMENTS</td></tr>
<tr><td>7</td><td colspan="5">START ADDRESS</td><td colspan="5">NUMBER OF SDEF'S</td></tr>
<tr><td>10</td><td colspan="10">INDEX TO SDEF'S</td></tr>
</table>

Word 0 Element type, in the upper portion, indicating whether a load element is absolute (494 mode) or relative (490 mode). An absolute element is indicated by 00002; a relative element is indicated by 20002.

Error indicator, in the lower portion, which accounts the type of errors, if any, which may be present in the element.

1–2 Name of the element, comprising 1–10 alphanumeric characters, left justified, as assigned by the Loader during collection. This name should be used in all references to the element. Unused portions of the words are space filled.

3 Version of the named element, comprising 1–5 alphanumeric characters, left justified, assigned to distinguish between elements bearing the same name. In referring to the element, the version is checked only if the version is specified. Unused portions of the word are space filled. If no version is specified on the LOAD statement, the word will be space filled.

4 Increment to the element base, assigned relative to the base of the file in which the element is contained.

5 Primary storage requirements, in the upper portion, specifying the maximum amount of primary storage needed for containing the coding for the element upon execution.

Length control, in the lower portion, specifying the length of the control or resident portion of the element.

6 Number of segments, in the upper portion, enumerating the segments exclusive of the control or resident segment. Number of control statements, in the lower portion, accounting the control statements appended to or imbedded in the element. Each statement is 20_8 words in length.

7 Start address, in the upper portion, indicating the relative address at which control will be given at execution time. In the case of a segmented routine, the start address will be relative 0. An instruction is appended to the collection at relative 0 which will cause a transfer of control to the appropriate location.

Number of SDEF's, in the lower portion, accounting the SDEF's associated with the element.

10 Index to SDEF's, relative to the incremented base of the element (word 4), indicating where the SDEF's may be found.

NOTE: The TOC for the output file is identical to the above; however, because the header information occupies the first 10_8 words of the header block, the TOC begins in word 11 and continues through word 21.

A.2.1.2. CONTROL STATEMENTS

The control statement images, appended to and imbedded in the element, are each 20_8 words in length, and may consists of any OMEGA control statements which may be processed as pretask initialization statements, e.g., ASG, MSG, etc. The statements are identical to the actual control statements used for producing the RB element, with the exception that a space code ƀ is substituted for the control symbol # in the initial position of the statement. Such statements are output in the order in which they are encountered with the secondary control statements for the Loader.

As the actual control statement may be less than 20_8 words, the last word of the control statement image specifies the number of pertinent words contained within the image. The area between the last pertinent word of the image and the last word of the image (pertinent word count) is hash filled. Because the control statement image length transcends the item length (16_8 words), the last image in a series of images may partially fill an item after completely filling a previous item. The remainder of the partially filled item is space filled. The total length of the control statement images may be determined by multiplying 20_8 by the number of images.

A sample data block (tape output) containing imbedded ASG images may be represented as follows:

(a) Control Statements

ƀASGƀƀRAN,ZH,200000ƀƀ △

ƀASGƀƀCORE,10000/20000ƀƀ △

(b) Data Block

Word						
0	0 0 0 0 6 0 0 3 0 6					0 (Upper) Number of 16_8-word items in block (Lower) Block size 1,52 Length of three 16_8-word items
1	ƀ	A	S	G	ƀ	1,20 First ASG statement image
2	ƀ	R	A	N	,	
3	Z	H	,	2	0	
4	0	0	0	0	0	
5 . . .	ƀ	ƀ (Hash fill)	Δ	ƀ	ƀ	
20	0 0 0 0 0 0 0 0 0 5					20 Pertinent word count; end of image
21	ƀ	A	S	G	ƀ	21,40 Second ASG statement image

22	ƀ	C	O	R	E	
23	,	1	0	0	0	
24	0	/	2	0	0	
25	0	0	ƀ	ƀ	Δ	
. . .	(Hash fill)					
40	0 0	0 0	0 0	0 0	0 5	40 Word count; end of image
. . .	(Space fill)					
52	ƀ	ƀ	ƀ	ƀ	ƀ	52 End of last item of control statements
. . .	(Text)					
305	CHECK SUM					

A.2.1.3. INSTRUCTIONS

The coding for the collected elements will be relative to a base of zero. When the collected routine is not segmented, the load element will contain only the coding which was contained in the RB elements before collection. If the load element is segmented, a resident segment is created, with all other segments being designated as secondary. For segmented load elements, the Loader appends segment descriptor information at the beginning of the element. This information defines the location of each segment on mass storage and specifies the relative location within the collected routine at which the segment will be loaded. Access information is also included which enables the segment loader to control the segment load process.

The appended segment descriptor information has the following general form:

A	ENTRY INSTRUCTION	
B	FILE CODE	SEGMENT LENGTH
		RELATIVE SEGMENT BASE
B	LOGICAL INCREMENT TO SEGMENT	
C	FILE CODE	SEGMENT LENGTH
		RELATIVE SEGMENT BASE
	LOGICAL INCREMENT TO SEGMENT	
D	7 7 7 7 7	7 7 7 7 7
E	7 7 7 7 7	SEGMENT DESCRIPTOR ADDRESS
	ENTRY POINT ADDRESS	7 7 7 7 7
F	7 7 7 7 7	SEGMENT DESCRIPTOR ADDRESS
	ENTRY POINT ADDRESS	7 7 7 7 7
G	7 7 7 7 7	7 7 7 7 7

Location A — Entry instruction. When a routine is segmented, control will be given to the routine at the base address upon initiation. The instruction at the base address will be a jump to the user defined starting address or will be an indirect jump through the entry point table (E,F) to the segment containing the starting address.

B — A three word segment descriptor is created for each segment defined. The order of the descriptors in the segmentation information is dependent upon the order in which the segments are defined. For example, B would be the descriptor for the first defined segment, and C would be the descriptor for the second segment.

B+0 — File code, alphabetic, in the upper portion, indicating the file in which the segment is stored. The code is set when the load element is read into memory for execution.

Segment length, in the lower portion, specifying the length of the segment, this being equal to the sum of the lengths of all elements included in the segment.

B+1 — Relative segment base, assigned relative to the base of the load element, indicating the location at which the segment will be loaded. The relative segment base is modified to the absolute address when the element is read into memory for execution.

B+2 — Logical increment to segment, relative to the base of the load element, for locating the defined segment on the drum file. This increment is added to the logical increment of the instruction base, relative to the base of the file containing the element, when the load element is read into memory for execution. This produces the actual increment to the segment on the drum file, which will be used to load the segment.

C — A three word descriptor defining the second segment specified, with the segment undergoing modification at execution time similar to the first segment.

D — Terminal sentinel, comprising a string of binary ones (octal sevens), placed at the end of the segment descriptors.

E — Beginning of a series of entry points within a segment. Each segment within the collection will cause a list of entry points to be created within the segment information vector table.

E+0 — Beginning sentinel, in the upper portion, comprising a string of binary ones, indicating the beginning of a series of entry points within the segment.

Segment descriptor address, in the lower portion, which is a relative address specifying the location of the descriptor for the segment containing the entry, such as B or C.

E+1 — Entry point address within the segment, defined in the whole word. The relative address of the entry point within the element is specified in the upper portion of the word.

When the segment is not in memory, the lower portion of the word will contain binary ones. When the segment is in memory, the relative address of the entry point is contained in the lower portion (as in the upper portion).

F — Beginning of a series of entry points for a second segment.

G — Terminal sentinel, comprising a string of binary ones, placed at the end of the entry point definitions to signal the end of the load element segmentation information.

The instructions for the collected elements follow the segmentation information. Allocation is made relative to the base of the collection, 0, such that the code may be read into memory, the Relative Index Register (RIR) set to the load base, and the element executed directly.

A.2.1.4. MODIFICATION – M OPTION

When the M option is present on the LOAD statement, modification bits are appended, indicating the adjustment necessary for loading and executing the element on a processor operating without the Relative Index Register (RIR); that is, a computer operating in the UNIVAC 490 mode.

Coding for the load element, under the M option, is grouped in 80_{10}-word modules. Each module contains 75 instructions and five words of modification bits pertaining to the 75 instructions. If the element is segmented, descriptor information is generated which will describe the segments as they will appear after modification.

The following diagram illustrates the appearance of a load element produced under the M option:

MC1	MC2	MC3	MC4	MC5	MC6	MC7	MC8	MC9	MC10	MC11	MC12	MC13	MC14	MC15
MC16	MC17													
MC31														
MC46														
MC61														

```
INSTRUCTION ASSOCIATED WITH MC1
    ‘‘           ‘‘        ’’  MC2
    ‘‘           ‘‘        ’’  MC3
    ‘‘           ‘‘        ’’  MC4
    ‘‘           ‘‘        ’’  MC5
    ‘‘           ‘‘        ’’  MC73
    ‘‘           ‘‘        ’’  MC74
    ‘‘           ‘‘‘       ’’  MC75
```

The first five words of each 80-word module are reserved for the modification bits for the 75 instructions which follow. Each modification code is two bits long. Each word of modification contains the bits for modifying 15 instructions.

The modification bit meanings are as follows:

00	No modification necessary.
01	Modify the lower 15 bits (14–0) of the instruction word by the load base.
10	Modify the upper 15 bits (29–15) of the instruction word by the load base.
11	Modify the upper (29–15) and lower (14–0) of the instruction word by the load base.

A.2.1.5. SYMBOL DEFINITIONS (SDEF's)

Symbol definitions (SDEF's) are labels for locations within the element which may be used at execution time to define test points and test areas when operating in a debugging environment. The SDEF's are originally formed by the language processor which produces the RB elements used to form the load element. An S option must be present on the LOAD statement if the SDEF's are desired to be included in the load element. Under the S option, the Loader will assign the address of the SDEF relative to the base of the collection. The number of the segment in which the SDEF appears is also included. A load element may contain no, one or many SDEF's.

The form of the SDEF is as follows:

<table>
<tr><td>0</td><td>N</td><td>N</td><td>N</td><td>N</td><td>N</td></tr>
<tr><td>1</td><td>N</td><td>N</td><td>N</td><td>N</td><td>N</td></tr>
<tr><td>2</td><td colspan="3">SEGMENT NUMBER</td><td colspan="2">RELATIVE ADDRESS</td></tr>
</table>

Word 0–1 Name of the SDEF, comprising 1–10 alphanumeric characters, left justified, as assigned by the language processor, defining a relative location in the element.

2 Segment number, in the upper portion, specifying the segment in which the SDEF is contained.

Relative address, in the lower portion, indicating the address of the SDEF as assigned relative to the base of the collection.

A.2.2. Tape Output Format

The load element may be output on tape through the OUT control statement under various options (see 4.7, Program Library Editor). A header block containing the TOC and other descriptive information is written at the beginning of the element. Vector tables, for describing segmentation, and instructions and SDEF's are written into data blocks. An end sentinel block is written at the end of the element.

The header and end sentinel blocks have the same general form as the corresponding blocks for the RB element, and carry much the same information, with the main difference being the TOC information of the header block.

A.2.2.1. DATA BLOCK

The data block has the same format, and carries information in the same manner, as the RB data block; that is, in 16_8-word items to a maximum block length of 306_8 words. The information carried in the data block is not separated according to type; however, the vector information, and the control statements will occupy the initial blocks; and the SDEF's will follow the instructions. Because each control statement is 20_8 words in length, a control statement may fill an item and partially fill a second item. In this case, the remainder of the second item is space filled.

NOTE: Except for elements produced under the M option, the load element does not carry modification bits as does the RB element. In either case, the modification bit stream of the RB element does not correspond to the modification bit word pattern of the load element.

A.2.3. Card Output Format

When the load element is punched on cards (using the OUT control statement), ten characters of descriptive information are appended to each card image. The information includes identification, sequence and card codes.

The load element output card has the following form:

Columns

1 5	6 9	10	11	80
IDENTIFICATION	SEQUENCE NO.	TYPE	INFORMATION	

Columns 1–5 Identification, comprising the first five characters of the name of the element. This field may be used for deck information.

6–9 Sequence number, comprising four decimal characters which will be checked if the cards are used as input (using the IN control statement with the C option).

10 Card type code, which is alphabetic, and will indicate the kind of information which will be carried by the card, as follows:

A TOC and control images

B Instructions

C SDEF's

11–80 Information, comprising the coding for the element.

A.2.4. Sample Listings of Load Element Output

The following pages show listings of load elements produced under the L and N options (see Section 4), and elements produced without options.

A.3. SOURCE ELEMENT (Element Type 3)

A source element is the input to a language processor as a base for forming an RB element and, finally, a load element (Figure A–3). The source element is made up from source language statements, in any of the system's assembler and compiler languages, and from data sets and control statements.

The source element input into the system is in the form of variable length images in Fieldata code, each image having been edited to delete full words of trailing blank columns in order to conserve drum storage space. The input may be from mass storage media or cards. Normally the SOURCE statement (see Section 4) is used to input elements which originate within the primary input stream; and the IN control statement is used for elements which originate outside the primary input stream. The source element may be entered into the program library complex and also may be output on various media.

```
#LOAD YL CP,  LOADER
SEGMENT A
INCLUDE CP1
SEGMENT B, A
INCLUDE CP2
SEGMENT C, B
INCLUDE CP3
              SEGMENT   NAME     NUMBER     BASE     LENGTH
                   CONTROL        000      00000     1160
                 ELEMENT   NAME        VERSION    LENGTH      BASE     CC#   COMMON NAME    VALUE
                      CP                          01077      00061     000
                      CP                                     00027     001    PB             00032
                                              SATISFIED EXTERNAL REFERENCE
              SXREF         VALUE       ELEMENT   N/V    SEG        SXREF      VALUE     ELEMENT   N/V SEG
             CP1            01160       CP1               01
               SEGMENT  NAME     NUMBER      BASE       LENGTH
                  A                001      01160       2150
                 ELEMENT   NAME        VERSION    LENGTH      BASE     CC#   COMMON NAME    VALUE
                      CP1                         02150      01160     000
                                              SATISFIED EXTERNAL REFERENCE
              SXREF         VALUE       ELEMENT   N/V    SEG     SXREF         VALUE     ELEMENT   N/V SEG
              SH            01143      CP                00     SHV            00173     CP            0
              ABSTOC        00402      CP                00     SP             01142     CP            0
              CCT           00063      CP                00     ELMN2          00055     CP            0
              ENT           00060      CP                00     LMCB           00173     CP            0
              MBASE         01017      CP                00     EP             00725     CP            0
              CP1S11        77777      UNDEFINED                SMAP           00415     CP            0
              LL            01022      CP                00     ED             00071     CP            0
              JDB           00216      CP                00     CPJN           00066     CP            0
              TBS           00373      CP                00     LH             00320     CP            0
              EDW           01144      CP                00     EBS            00374     CP            0
              END           12146      CP                00     CMAP           00216     CP            0
              EWR           00376      CP                00     TBLMT          00173     CP            0
              ATOC2         00177      CP                00     TED            00067     CP            0
              PB            00053      CP                00     CP2            01160     CP2           0

            SEGMENT     NAME       NUMBER       BASE       LENGTH
              B           NAME  000       01160         2100
            ELEMENT         NAME        VERSION    LENGTH        BASE        CC#       COMMON    NAME      VALUE
                CP2                                 02100      01160       000
                                              SATISFIED EXTERNAL  REFERENCE
            REF             VALUE    ELEMENT    N/V      SEG       SXREF      VALUE        ELEMENT   N/V    SEG
            0               00053    CP                  00        BINDA      00067        CP               00
            B               00206    CP                  00        EQUT2      00176        CP               00
            C2              00177    CP                  00        EP         00725        CP               00
                            00204    CP                  00        EREF       00202        CP               00
                            00200    CP                  00        INFO       00201        CP               00
            DT              00200    CP                  00        XREF       00205        CP               00
                            00401    CP                  00        SNT2       00174        CP               00
            P               00415    CP                  00        MBASE      01017        CP               00
            TOC             00402    CP                  PP        E6         00773        CP               00
            F               00203    CP                  00        EXCT2      00175        CP               00
                            00071    CP                  00        LL         01022        CP               00
                            01160    CP3                 03
            SEGMENT     NAME       NUMBER       BASE       LENGTH
              C                     003       01160        1140
            ELEMENT         NAME         VERSION   LENGTH        BASE        CC #      COMMON    NAME      VALUE
                CP3                                 01140      01160       000
                                              SATISFIED EXTERNAL  REFERENCE
            REF             VALUE    ELEMENT    N/V      SEG       SXREF      VALUE        ELEMENT   N/V    SEG
            TOC             00402    CP                  00        REF1   CCT 00063        CP               00
            C2              00177     CP                 00        INFOT      00200        CP               00
            D               77777     UNDEFINED                    VTC        00401        CP               00
                            00053     CP                 00        EP         00725        CP               00
            OC2             00173     CP                 00        SNT2       00174        CP               00
```

Figure A–3. Sample Listing – Load Element (Part 1 of 2)

```
Sample listing (with the exception of error messages) produced under N option

# LOAD YN CP,   LOADER
  SEGMENT A
  INCLUDE DP1
  SEGMENT B, A
  INCLUDE CP2
  SEGMENT C, B
  INCLUDE CP3


             SEGMENT      NAME      NUMBER      BASE       LENGTH
                CONTROL              000       00000        1160
                ELEMENT      NAME       VERSION     LENGTH       BASE       CC #     COMMON     NAME     VALUE
                     CP                              01077      00061       000
                                                                00027       001        PB                0032

             SEGMENT      NAME      NUMBER      BASE       LENGTH
                A                    001       01160        2150
                ELEMENT      NAME       VERSICN     LENGTH       BASE       CC#      COMMON     NAME     VALUE
                     CP1                             02150      01160       000

             SEGMENT      NAME      NUMBER      BASE       LENGTH
                B                    002       01160        2100
                ELEMENT      NAME       VERSION     LENGTH       BASE       CC#      COMMON     NAME     VALUE
                     CP2                             02100      01160       000

             SEGMENT      NAME      NUMBER      BASE       LENGTH
                C                    003       01160        1140
                ELEMENT      NAME       VERSION     LENGTH       BASE       CC#      COMMON     NAME     VALUE
                     CP3                             01140      01160       000
```

Figure A–3. Sample Listing – Load Element (Part 2 of 2)

A.3.1. Element Organization

Each source element is composed of a table of contents (TOC) and text. The TOC contains the element name and version, together with information describing the element. The text comprises the source and control language statements making up the body of the element. The basic order and structure is retained in the libraries and on external media. However, when the source element is output on tape and cards, initial header or identification information is appended.

(a) General Format

The relative positions of the various portions of the source element may be represented as follows:

HEADER (if used)
TABLE OF CONTENTS (TOC)
DATA
DATA

A.3.1.1. TABLE OF CONTENTS (TOC)

The TOC serves as the element descriptor and contains the following information.

0	0 0 0 0 3			ERROR INDICATOR	
1	N	N	N	N	N
2	N	N	N	N	N
3	V	V	V	V	V
4	INCREMENT TO ELEMENT BASE				
5	(UNASSIGNED)				
6	(UNASSIGNED)				
7	TYPE	ƀ̿	ƀ	NUMBER OF IMAGES	
10	ELEMENT LENGTH				

Word 0 Element type, in the upper portion, indicating whether the element is RB, load or source. Type 3 indicates a source element.

Error indicator, in the lower portion, which accounts the number of errors, if any, which may be present in the element.

1–2 Name of the element, comprising 1–10 alphanumeric characters, left justified, as assigned on the source input statement.

3 Version of the named element, comprising 1–5 alphanumeric characters, left justified, assigned for distinguishing between adaptations of elements bearing the same name.

4 Increment to the element base, assigned relative to the base of the file in which the element is contained.

5–6 Unassigned.

7 Source type, in the upper portion, bits 29–24, indicating the origin of the element:

B indicates an unmodified source element previously output from the system (using the OUT statement).

Number of images, in the lower portion, contained in the source input.

10 Element length, specifying the storage requirements for a type B element.

A.3.1.2. TEXT

Any set of statements which may appear in the primary control stream can be included in the source element text. This embraces primary and secondary control language, data sets and programming language statements. The source input statements are compressed by deleting words of trailing space, with codes being appended to each image for relating the length of the compressed data within the image. The length specified includes the data and the appended code. Any number of statements may be contained in the element, and may comprise original source information, update information, and corrections.

A.3.2. Tape Output Format

The source element may be output on tape through the OUT control statement under various options (see 4.7, Program Library Editor). A header block containing the TOC and other descriptive information is written at the beginning of the element. The text is written into data blocks. An end sentinel block is written at the end of the element.

The header and end sentinel blocks have the same general form as the corresponding blocks for the RB element, and carry much of the same information; the difference being the TOC information of the header block. The data block carries information in a slightly different manner, as described in the following subsection.

A.3.2.1. DATA BLOCK

The source element is written on tape carrying the maximum number of whole images that can be contained in the available 306_8 word block size. Each source image may be 30_8 or less words in length, with whole words of trailing space deleted to conserve library space. A source image on tape comprises a descriptor and the image proper. The descriptor specifies the length of the image proper. A source data block may be represented as follows:

0	NUMBER OF ITEMS	SIZE
1	IMAGE LENGTH DESCRIPTOR	
2	(IMAGE)	
• • •		
n	IMAGE LENGTH DESCRIPTOR	
•	(IMAGE)	
• •		
305	CHECK SUM	

Word 0 Number of items, in the upper portion, indicating the number of images which will be contained in the block.

Size, in the lower portion, indicating the length of the block in words, the maximum being 306_8 words.

1 Image length descriptor, indicating the length of the image to follow, the maximum being 30_8 words.

2 The image proper, continuing to the length specified by the preceding descriptor.

n Image length descriptor for a second image to follow.

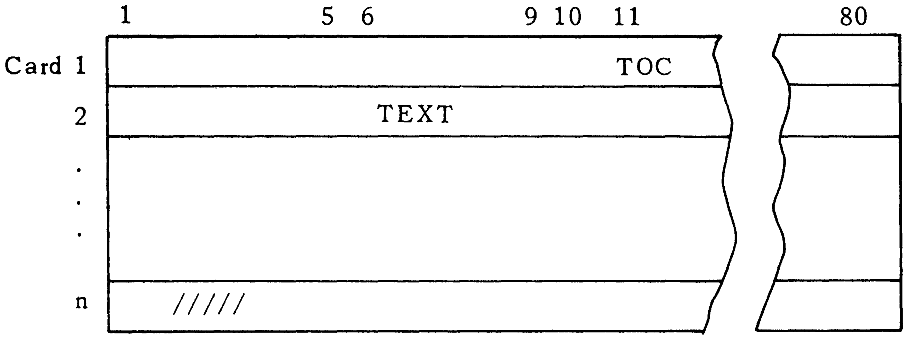
n+1 A second image, continuing to the length specified by the second descriptor.

305 Check sum: the complement of the sum of all words in the data block, with the exception of the check sum entry. When the block is less than 306_8 words, the check sum is entered into the last word of the block after the last text entry.

A.3.3. Card Output Format

The source element may be output in card image format through the #SOURCE statement or through the OUT control statement under the C option. The TOC information is carried on the first image in columns 11–56. No information is contained in the first 10 columns. All subsequent data is output without appended information. The final image of text is followed by an image containing slashes (/////) in the first five columns, signifying the end of the deck. An example of a source element is shown in Figure A1–1, the source element being used in the language processor for creating the RB element.

A source element submitted to the secondary input stream in card image format may be represented as follows:



Card 1 Identification, in columns 1–5, comprising the name of the element. Sequence number, in columns 6–9, comprising four decimal characters. Card type code, in column 10, which is A. TOC, in columns 11–80, as described previously.

Card 2 Text, in columns 1–80, comprising both control statements and information, and continuing to the end of the element.

Card n Terminal signal, in columns 1–5, comprising five slashes (/////), and signifying the end of the element.

APPENDIX B. TAPE LABELLING

It has been found necessary to develop a system of software tape control which will increase the integrity of the file handling. In order to accomplish this the system will automatically check the first block of any tape used, compare it with the file name specified on the job control cards and give a suitable diagnostic to the operator if there is any disparity. By this means, the risk of a tape being written to or read from by accident is greatly reduced.

NOTE: Tape labelling may not be available on your system. If tape labelling is available but not compulsory then the labelling parameters may be used without detriment to other processes. If tape labelling is compulsory then all relative parameters associated with this facility must be used on all tasks, otherwise the tasks will be terminated.

Your systems programmers should be consulted to establish the set-up of your system as tape labelling is an optional feature.

In order to provide the tape labelling system a new common subroutine (D$LABEL) has been written to supplement FACILITY ASSIGNMENT and the IN, OUT and LINK processors. The new subroutine is referenced via a normal EXRN call from the processors which controls all tape label checking and all operator interfaces. By these means the system is virtually transparent to the user except for the fact that another field has been added to the processor control cards. This field will be explained in the following paragraphs.

B.1. GENERAL

The important concepts to remember regarding tape labelling are the R (Rewind) option and the FILE NAME which apply to all the processors. They act as control parameters for the tape labelling routine, specifying the functions that D$LABEL is to perform.

Ideally, the R option and FILE NAME should be used on all assignments for TAPE or its derivatives, as this immediately initiates a call to D$LABEL for tape verification. This is the most efficient method of usage.

Alternatively, the R option and the FILE NAME can be specified on the first IN or OUT concerned with TAPE.

This will also initiate a call to the D$LABEL, but by a less efficient method.

It is essential that either one or the other of these conditions are adhered to, otherwise, if tape labelling is compulsory, all subsequent tasks concerning the tape will be terminated.

NOTE: The FILE NAME field consists of up to 15D characters, but only the first 10D are used for label validation.

B.1.1. Tape Labelling and Facility Assignment

The following example illustrates the format of the ASG statement and the use of the fields therein when using tape labelling.

```
#ASGƀRƀTAPE,FILE CODE,,FILE NAME
```

It can be seen from this that the general format of the ASG statement is unchanged as a field for the FILE NAME already exists. This is the name which will correspond to the name on the header block of the tape. For the most efficient use of the tape labelling, this format should always be used. Other options are permitted in addition to the R option.

■ Example:

```
#ASGƀJUMRWIƀTAPE,A,,TESTTAPE
#ASGƀRWIƀUN8C,BJ,,1234
```

In both cases, because the R option and the file name are present, the tape requested will be located and the first block will be read. If the header on the tape and the specified file name match, assignment will be completed in the normal manner. If they do not match, the message:

```
FILE XX TAPE LABELS DO NOT MATCH.
```

will be printed on primary output, where XX represent the file code. This is a diagnostic and the assignment will still be completed as normal. If either the FILE NAME or R option was not stated, no label checking would have occurred at all.

B.1.2. Tape Labelling and the IN Processor

The following example illustrates the format of the IN statement and the fields therein when using tape labelling.

■ Example:

```
#INƀRƀFILE CODE/FILE NAME,NAME1/VER1,NAME2/VER2,.....
```

It can be seen that the only change to this card is the addition of the sub-field FILE NAME. This field should match the FILE NAME field on the ASG card and the FILE NAME on the header block of the tape, if known. (This field is not required for IN's from peripherals other than TAPE or its derivatives.)

It is not imperative that the R option be used on the first IN from a tape, however, without it, the operator will not be given any opportunity of over-riding a mistake or mis-match in facility assignment. (See B.2.)

```
#ASGƀRWƀTAPE,A,,INPUT
#INƀRƀA/INPUT,NAME1/VER1,NAME2,NAME3
#END

#ASGƀJUWRIƀUN8C,BC,,TAPEIN
#INƀRPƀBC/TAPEIN
#END
```

B.1.3. Tape Labelling and the OUT Processor

The following example illustrates the format of the OUT statement and the fields therein, when using tape labelling.

- Example:

```
#OUTƀRƀFILE CODE/FILE NAME,LIBRARY/IDENTITY,NAME/VERSION,ETC.
```

It can be seen that the only change to the card is the addition of the sub-field FILE NAME. This field should correspond exactly to the FILE NAME field on the ASG card, and the FILE NAME in the header block of the tape, if known. (This field is not required for OUT's to peripherals other than TAPE or its derivatives.)

It is recommended that the R option always be included on the first OUT card relating to a specific assignment. If this is not done, and errors have occurred during assignment, the operator will not be given the opportunity of intervening and allowing the job to continue. (See B.2.)

```
#ASGƀJUMRWƀTAPE,D,,OUTPUT
#OUTƀRFLƀD/OUTPUT,JOB,NAME1,NAME2,NAME3/VER3
#END
```

```
#ASGƀRƀTAPE,AA,,GRPLIB2
#OUTƀRSFƀAA/GRPLIB2,JOB/14
#END
```

B.1.4. Tape Labelling and the LINK Processor

The following example illustrates the format of the LINK statement and the fields therein when using tape labelling.

- Example:

```
#LINKƀRƀIDENTITY,LIBRARY FILE CODE,P-NAME,INPUT FILE CODE,INPUT FILE NAME
```

It can be seen that the format of the LINK statement has not been changed as the field for the FILE NAME (INPUT FILE NAME) already exists.

In fact, the LINK processor utilizes much of the code for FACILITY ASSIGNMENT when requesting an input tape, so the same rules apply for both.

```
#LINKƀUMRWƀ14,Q,UN8C,A,GRPLIB2
#END
```

```
#LINKƀRƀ193D,AB,TAPE,F,GROUP INPUT
#END
```

B.2. OPERATOR COMMUNICATION

In certain circumstances during the tape label verification process, the operator will be informed that a discrepancy has occurred in the labelling and will have the option to intervene to provide instructions. The only circumstances where this will occur are:

(1) If an error has occurred during FACILITY ASSIGNMENT; i.e., there was a discrepancy between the FILE NAME stated on the card and that in the tape label; or tape labelling was not called by FACILITY ASSIGNMENT because of the omission of either the R option or the FILE NAME from the card.

(2) If both the R option and the FILE NAME are present on the IN or OUT card and the FILE NAME on the card does not correspond to the name on the tape header block. If the R option has been omitted from the IN or OUT statements and a fault has occurred in FACILITY ASSIGNMENT, the task or job will be terminated, depending on normal X option conventions, with no chance for operator intervention. However, the diagnostic:

ERRORS IN TAPE LABELLING

will be printed on primary output.

Assuming that these conditions have been satisfied and there has been a disparity in tape label verification, the following messages will appear on the operator console:

```
JN: FILE XX WRONG LABEL
JN: WANTED REQD. NAME
JN: FOUND ACTL. NAME
*DN:JN: TYPE F FOR FORCE/R-REJECT/A-ABORT
```

Where JN represents the job number, XX represents the file code of the tape, and DN is the delay number for the operator response. REQD. NAME represents the FILE NAME specified on the IN or OUT card and ACTL. NAME the name held in the tape header block. If the words NOT KNOWN appear in this position the tape label could not be read by the system.

When this message appears, the operator has three options which he can reply; F, R or A, as shown. If he types anything but these three options, the last line of the message will be repeated on the console until a meaningful reply is provided.

If he replies F (Force), the discrepancy in the labels will be ignored and the task will continue as normal. If the task was an OUT, the new label (i.e., the label stated on the OUT card) will be written to the tape. If the task was an IN, the label specified on the IN card will be 'remembered' by the tape labelling system so that any subsequent IN's on that assignment will not result in a label discrepancy (unless the FILE NAME on the IN card is changed).

If the operator replies R (Reject), the tape in question will be re-wound with interlock and the following message output to the console:

```
*DN:JN: REMOUNT FILE XX
```

A different tape can now be mounted in place of the other one before the message is answered. If the label on the new tape matches the FILE NAME on the card, the task will continue as normal. If they do not match, the whole sequence will be repeated.

If the operator replies A (Abort) to the console request, the task or job will be terminated, depending on whether or not there is an X option on the IN or OUT card. Also the diagnostic:

ERRORS IN TAPE LABELLING

will be submitted to primary output.

All console messages are reproduced on primary output, with the addition of the following:

```
JN: CONSOLE REPLY:    X
```

to indicate which of the three options the operator chose.

B.3. EXAMPLE JOB STREAM

```
001 #JOBƀƀTESTONE/JIM, 1234,B, C,C/C
002 #ASGƀUMRWIƀTAPE,A,,SOURCE
003 #INƀRPƀA/SOURCE
004 #END
005 #LINKƀUMRWIƀ15,B,TAPE,C,JIMSGRPLIB
006 #END
007 #ASGƀJUMRWIƀTAPE,D,,NEWSOURCE
008 #OUTƀXƀD/NEWSOURCE,JOB,ELMTONE,ELMNTTWO
009 #END
010 #OUTƀFƀD/NEWSOURCE,15
011 #END
012 #DEL
013 #INƀRPƀD/NEWSOURCE
014 #END
015 #FIN
```

CARD 001

Standard job card.

CARD 002

Assign a tape with the file name SOURCE for the length of the next task only. Read the first block of the tape and execute a comparison check on the tape label and the file name on the card. If the first ten characters match, continue as normal: if not print a diagnostic and continue as normal.

CARD 003

Check that the FILE NAME on the card matches the FILE NAME on the tape. It it does not, display a diagnostic on the console, asking the operator to intervene. If it does, continue and transfer all the elements on the first file of the tape to the job library, printing a list of all elements transferred (P option).

CARD 004

Standard END card.

CARD 005

Assign tape C – JIMSGRPLIB for the duration of that task only, execute all those checks that were done on CARD 002, read the first file of the tape and create a group library with the I.D. of 15.

CARD 0006

Standard END card.

CARD 007

Assign a tape, beyond the duration of the next task, with the file name NEWSOURCE. Execute label checking and verification in the same manner as for CARD 002.

CARD 008

Execute a validity check on the tape label and the FILE NAME stated on the card. If they do not match, print a diagnostic and terminate the remainder of the job. (No request is made to the operator due to the lack of an R option. The JOB is terminated, as opposed to just the TASK, due to the X option.)

If they do match, write the elements ELMNTONE and ELMNTTWO to the tape from the JOB library.

CARD 009

Standard END card.

CARD 010

Execute a validity check on the tape label. (As long as CARD 008 was correct and the FILE NAMES on the two cards are identical, the task will continue normally.) If there is any discrepancy, output a diagnostic to the console requesting intervention. Otherwise, write all of the elements in Group Library 15 to the tape followed by an end-of-file sentinel.

CARD 011

Standard END card.

CARD 012

Delete all elements from the job library to conserve random access storage.

CARD 013

Execute a validity check on the tape label. (Again, if CARD 008 was valid, and the FILE NAME has not been changed, the check will be successful.) If the labels do not match, output a console diagnostic asking the operator to intervene. If they do match, transfer all the elements from the tape to the job library, supplying a list of all elements on the tape (P option).

CARD 014

Standard END card.

CARD 015

Standard FIN card.

B.4. ERROR MESSAGES

- FILE XX TAPE LABELS DO NOT MATCH

  Produced after a comparison check between the FILE NAME on the IN/OUT/ASG card and the tape header block has shown a discrepancy. It is purely a diagnostic and will not result in an error condition unless no further label checks are executed by IN/OUT.

- ERRORS IN TAPE LABELLING

Displayed by the IN/OUT processors after an error condition in tape labelling, (e.g., labels do not match) has resulted in a negative status being passed to the processors causing the task or job to be terminated.

- JN: FILE XX WRONG LABEL
  JN: WANTED REQD. NAME
  JN: FOUND ACTL. NAME
  *DN: JN: TYPE F FOR FORCE/R-REJECT/A-ABORT

This message is displayed on the console when there has been a disparity in label checking and the R option and FILE NAME are both present on the IN/OUT card. It allows the operator to intervene and supply further instructions to the job. (See B.3.)

- *DN: JN: REMOUNT FILE XX

This message is displayed on the console when the operator has replied R to this message. It gives him the opportunity of mounting a new tape before the job continues. (See B.3.)

CUT

Comments concerning this manual may be made in the space provided below. Please fill in the requested information.

System: ____________________

Manual Title: ____________________

UP No: __________ Revision No: __________ Update: __________

Name of User: ____________________

Address of User: ____________________

Comments:

FOLD

FIRST CLASS
PERMIT NO. 21
BLUE BELL, PA.

BUSINESS REPLY MAIL NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY

UNIVAC
P.O. BOX 500
BLUE BELL, PA. 19422

ATTN: SYSTEMS PUBLICATIONS DEPT.

CUT

FOLD

UNIVAC
SYSTEMS
PUBLICATIONS

UNIVAC 494

UPDATE

Updating Package A
UP-7504 Rev. 2

Operating System
Programmer Reference

This UNIVAC 494 Real-Time System Library Memo announces the release and availability of Updating Package A to " UNIVAC 494 Real-Time System Operating System Programmer Reference," UP 7504 Rev. 2.

Updating Package A should be utilized as specified on the Updating Summary Sheet. Upon completion of manual updating, file the Updating Summary Sheet after the Contents Section so that a record of updating is maintained.

Changes made by this update include information added to the control statements, data management system, processor control, utility service routines, and remote device control. Also, Appendix B has been added to provide information for software tape control.

Copies of Updating Package A are now available for requisitioning. Either the updating package alone, or the complete manual with the updating package already incorporated may be requisitioned by your Univac Manager via the Technical and Advertising Materials Requisition, UDI–578. To receive the updating package alone, order UP-7504 Rev. 2-A. To receive the complete manual, order UP-7504 Rev. 2.

Manual and Updating Package are available from:

Customer Information Distribution Center (CIDC)
Univac Division Sperry Rand Corporation
P.O. Box 448
King of Prussia, Pa, 19406

H. MASTERS
Group Manager
Systems Publications

217, 630 and
692 Library
Memo only.

Package A to UP-7504 Rev. 2 (cover, 83
pages, Memo and a 2-page Updating
Summary Sheet) to Univac Library
Mailing Lists 43 and 44

Library Memo
to UP-7504
Rev. 2-A

May 1973