



TSX-Plus Programmer's Reference Manual

s&h computer systems, inc.

TSX-Plus
Programmer's Reference Manual

Eighth Edition—First Printing—December 1990

Copyright © 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1988, 1989, 1990.
S&H Computer Systems, Inc.
1027 Seventeenth Avenue South
Nashville, Tennessee 37212-2299 USA
(615) 327-3670

The information in this document is subject to change without notice and should not be construed as a commitment by S&H Computer Systems, Inc. S&H assumes no responsibility for any errors that may appear in this document.

NOTE: TSX, TSX-Plus, PRO/TSX-Plus, COBOL-Plus, PRO/COBOL-Plus, RTSORT, PRO/RTSORT and CLASS are proprietary products owned and developed by S&H Computer Systems, Inc., Nashville, Tennessee, USA. The use of these products is governed by a licensing agreement that prohibits the licensing or distribution of these products except by authorized dealers. Unless otherwise noted in the licensing agreement, each copy of these products may be used only with a single computer at a single site. S&H will seek legal redress for any unauthorized use of these products.

A license for RT-11 is required to use this product. S&H assumes no responsibility for the use or reliability of this product on equipment which is not fully compatible with that of Digital Equipment Corporation.

Use, duplication, or disclosure by the Government, is subject to restrictions as set forth in subdivision (b)(3)(ii) of the Rights in Technical Data and Computer Software clause at 52.227-7013.

Questions regarding the licensing arrangements for these products should be addressed to S&H Computer Systems, Inc., 1027 Seventeenth Avenue South, Nashville, Tennessee 37212, (615) 327-3670, Telex 786577 S AND H UD.

Contents

1	Introduction	1
1.1	Management of System Resources	2
1.1.1	Memory Management	2
1.1.2	Execution Scheduling	2
1.1.3	Subprocesses and Process Windowing	2
1.1.4	Directory and Data Caching	2
1.1.5	System Administrative Control	3
1.1.6	Time-sharing lines and CL units	3
1.2	Summary of Chapter Contents	3
1.2.1	TSX-Plus Job Environment	3
1.2.2	Program Controlled Terminal Options	3
1.2.3	TSX-Plus EMTs	3
1.2.4	Shared Files, Record Locking and Data Caching	4
1.2.5	Inter-program Message Communication	4
1.2.6	Programming for CL and Special Device Handlers	4
1.2.7	Real-time Support	4
1.2.8	Extended Memory Features	4
1.2.9	Program Debugging Facility	4
1.2.10	Program Performance Monitor Facility	5
1.2.11	Differences from RT-11	5
1.2.12	Appendices	5
2	TSX-Plus Job Environment	7
2.1	Simulated RMON	7
2.2	Virtual and physical memory	7
2.3	User virtual address mapping	8
2.4	Normal programs and virtual programs	9
2.5	Access to system I/O page	9
2.6	Extended memory (PLAS) regions	10
2.7	Shared run-time systems	12
2.8	Rules for Mapping to extended memory regions	12

2.9	Support for Separate I- & D-Space	14
2.9.1	How to break the 64 Kb barrier	14
2.9.2	User memory organization and mapping	15
2.9.3	I/O to separate I- and D-space regions	16
2.10	VM pseudo-device handler	16
2.11	Job priorities	17
2.12	User command interface	18
3	Program Controlled Terminal Options	21
3.1	Terminal input/output handling	21
3.1.1	Activation characters	21
3.1.2	Single character activation	22
3.1.3	Non-blocking .TTINR	22
3.1.4	Non-blocking .TTOUR	23
3.2	Program controlled terminal options	23
3.2.1	"A" function—Set rubout filler character	25
3.2.2	"B" & "C" functions—Set VT52, VT100 and VT200 escape-letter activation	26
3.2.3	"D" function—Define new activation character	26
3.2.4	"E" and "F" functions—Control character echoing	26
3.2.5	"H" function—Disable subprocess use	26
3.2.6	"I" and "J" functions—Control lower case input	26
3.2.7	"K" and "L" functions—Control character echoing	26
3.2.8	"M" function—Set transparency mode of output	27
3.2.9	"N" and "O" Functions—Control command file input	27
3.2.10	"P" function—Reset activation character	27
3.2.11	"Q" function—Set activation on field width	27
3.2.12	"R" function—Turn on high-efficiency terminal mode	27
3.2.13	"S" function—Turn on single-character activation mode	28
3.2.14	"T" function—Turn off single-character activation mode	28
3.2.15	"U" function—Enable non-wait TT I/O testing	28
3.2.16	"V" function—Set field width limit	28
3.2.17	"W" and "X" functions—Control tape mode	28
3.2.18	"Y" and "Z" functions—Control line-feed echo	29
4	TSX-Plus EMTs	31
4.1	Obtaining TSX-Plus system values (.GVAL)	31
4.2	Determining if a job is running under TSX-Plus	32
4.3	Determining number of free blocks in spool file (0,107)	33
4.4	Determining number of blocks in use in spool file (1,107)	33
4.5	Determining the TSX-Plus line number (0,110)	34
4.6	Determining subprocess job number (1,110)	35

4.7 Set/Reset ODT activation mode (111)	35
4.8 Sending a block of characters to the terminal (114)	36
4.9 Accepting a block of characters from the terminal (115)	37
4.10 Checking for terminal input errors (0,116)	37
4.11 Determining input characters pending for a line (1,116)	38
4.12 Set terminal read time-out value (117)	39
4.13 Turning high-efficiency terminal mode on and off (0/1,120)	39
4.14 Suspending terminal output (2,120)	40
4.15 Checking for activation characters (123)	41
4.16 Obtaining site information (124)	41
4.17 Sending a message to another line (127)	42
4.18 Starting A Detached Job (0,132)	43
4.19 Checking the status of a detached job (1,132)	44
4.20 Killing a job (2,132)	45
4.21 Establishing break sentinel control (0,133)	45
4.22 Terminal input completion routine (1,133)	46
4.23 Mount a file structure (134)	47
4.24 Dismount a file structure (0,135)	48
4.25 Dismounting all file structures and logical disks (2,135)	49
4.26 Dismounting Logical Disks (3,135)	49
4.27 Determining Status of Logical Disks (4,135)	50
4.28 Dismounting all logical disks (5,135)	52
4.29 Performance monitoring (136)	53
4.30 Determining the terminal type (137)	53
4.31 Real-time requests (140)	54
4.32 Controlling the size of a job (141)	54
4.33 Determining project and programmer number (142)	54
4.34 Shared run-time requests (143)	55
4.35 Enabling separate I- and D-space (4,143)	55
4.36 User PAR control (10,143)	56
4.37 Determining job status information (144)	57
4.38 Determining file directory information (145)	60
4.39 Setting file creation time (146)	61
4.40 Determining or changing the user name (0/1,147)	63
4.41 Determining or changing the program name (2/3,147)	64
4.42 Determining or changing the terminal name (4/5,147)	65
4.43 Setting job priority (0,150)	66
4.44 Determining or changing job privileges (1,150)	67
4.45 Specifying a file for HOLD or NOHOLD mode (151(0))	69
4.46 Turning flag pages on or off for a device (151(1))	70

4.47	Setting width of spooler flag pages (151(2))	71
4.48	Program controlled terminal options (152)	71
4.49	Forcing [non]interactive job characteristics (153)	71
4.50	Setting terminal baud rates (0,154)	72
4.51	Raising and lowering the DTR signal on a line (1/2/3,154)	74
4.52	CL device control (0,155)	75
4.53	Clearing and Resetting a CL Unit (1/2,155)	77
4.54	Allocating a device for exclusive use (156)	78
4.55	Job monitoring (157)	79
4.55.1	Establishing a monitoring connection	80
4.55.2	Cancel a monitoring connection	81
4.55.3	Broadcast status report to monitoring jobs	81
4.56	Acquiring another job's file context (160)	82
4.57	Manipulating process windows (161)	84
4.57.1	Creating a window	85
4.57.2	Selecting a current window	87
4.57.3	Deleting windows	87
4.57.4	Suspending window processing	88
4.57.5	Resuming window processing	88
4.57.6	Printing a window	88
4.58	Switching Between Subprocesses (162)	89
4.59	Mounting Logical Disks (163)	90
5	Shared File Record Locking	93
5.1	Opening a shared file	94
5.2	Saving the status of a shared file channel	97
5.3	Waiting for a locked block	98
5.4	Trying to lock a block	100
5.5	Unlocking a specific block	102
5.6	Unlocking all locked blocks in a file	102
5.7	Checking for writes to a shared file	103
5.8	Data caching	103
6	Message Communications Facilities	105
6.1	Message channels	105
6.2	Sending a message	105
6.3	Checking for pending messages	107
6.4	Waiting for a message	108
6.5	Scheduling a message completion routine	109

7	Programming for CL and Special Device Handlers	113
7.1	Communication line handler (CL)	113
7.1.1	VTCOM/TRANSF support and CL handler	114
7.1.2	Terminal/Communication line cross connection	115
7.1.3	Redirecting CL and time-sharing lines	115
7.1.4	CL I/O operations	116
7.1.5	CL control character processing	116
7.1.6	CL .SPFUN operations	117
7.2	RK06/RK07 handler (DM)	125
7.3	DU (MSCP) handler	125
7.4	IEEE GPIB handler (IB)	126
7.5	Logical subest disk handler (LD)	126
7.6	MU (TMSCP) handler	129
7.7	Address Translation (AT) and Unibus (UB) handlers	129
7.8	Virtual memory handler (VM)	129
7.9	Communications handler (XL)	131
7.10	Spooled devices	131
8	Real-Time Program Support	133
8.1	Accessing the I/O page	134
8.1.1	EMT to map the I/O page into the program space	134
8.1.2	EMT to remap the program region to the simulated RMON	135
8.1.3	EMT to peek at the I/O page	136
8.1.4	EMT to poke into the I/O page	137
8.1.5	EMT to bit-set a value into the I/O page	137
8.1.6	EMT to do a bit-clear into the I/O page	139
8.2	Mapping to a physical memory region	139
8.3	Requesting exclusive system control	141
8.4	Locking a job in memory	142
8.5	Unlocking a job from memory	143
8.6	Suspending/Resuming program execution	143
8.7	Converting a virtual address to a physical address	143
8.8	Specifying a program-abort device reset list	145
8.9	Reading the Processor Status Word (PSW)	145
8.10	Setting processor priority level	146
8.11	Setting Job Execution Priority	146
8.12	Connecting interrupts to real-time jobs	147
8.12.1	Interrupt service routines	150
8.12.2	Interrupt completion routines	153
8.13	Releasing an interrupt connection	156
8.14	Scheduling a completion routine	156
8.15	Adapting real-time programs to TSX-Plus	157

9 Extended Memory Regions	159
9.1 Shared run-times	159
9.1.1 Associating a run-time system with a job	160
9.1.2 Mapping a run-time system into a job's region	161
9.1.3 Using I- and D-space with Shared Run-Time Regions	162
9.1.4 Fast mapping to shared run-time regions	163
9.2 PLAS regions	164
9.2.1 Fast mapping to PLAS regions — 20 times faster	164
9.2.2 Fast mapping extended to multiple PARs	165
10 Program Debugger	167
10.1 Debugger requirements	167
10.2 Invoking the debugger	168
10.2.1 RUN/DEBUG switch	168
10.2.2 CTRL-D Break	168
10.3 BPT Instruction	168
10.4 Commands	169
10.5 CTRL-D breakpoints	170
10.6 Address relocation	171
10.7 Internal registers	171
10.8 Data watchpoints	171
10.9 Symbolic instruction decoding	172
10.10 Special notes	172
11 TSX-Plus Performance Monitor Feature	175
11.1 Starting a performance analysis	175
11.2 Displaying the results of the analysis	176
11.3 Performance monitor control EMTs	177
11.3.1 Initializing a performance analysis	177
11.3.2 Starting a performance analysis	179
11.3.3 Stopping a performance analysis	180
11.3.4 Terminating a performance analysis	180
12 TSX-Plus Restrictions	183
12.1 System service call (EMT) differences between RT-11 and TSX-Plus	183
12.2 Special program suggestions	186
12.2.1 DIBOL	186
12.2.2 FILEX utility	186
12.2.3 FORTRAN virtual arrays	187
12.2.4 IND .ASKx timeouts	187
12.2.5 MicroPower/Pascal	187
12.2.6 Overlaid programs	187

A SETSIZ Program	189
A.1 Running the SETSIZ program	190
A.2 Setting total allocation for a SAV file	190
A.3 Setting amount of dynamic memory space	191
A.4 Setting virtual-image flag in SAV file	191
B RT-11 & TSX-Plus EMT Codes	193
B.1 TSX-Plus RT-11 Compatible EMTs	193
B.2 TSX-Plus Specific EMTs	196
C Subroutines Used in Example Programs	199
C.1 PRTOCT—Print an octal value	199
C.2 PRTDEC—Print a decimal value	199
C.3 PRTDE2—Print a two-digit decimal value	200
C.4 PRTR50—Print a RAD50 word at the terminal	200
C.5 R50ASC—Convert a RAD50 string into an ASCIZ string	201
C.6 DSPDAT—Print a date value at the terminal	202
C.7 DSPTI3—Display a 3-second format time value	202
C.8 ACRTI3—Convert a time value to special 3-second format	203
C.9 ACRDEC—Convert an ASCII decimal value to a numeric value	204
C.10 RADASC—Convert a RADIX-50 string to ASCII	205
D DIBOL TSX-Plus Support Subroutines	207
D.1 Record locking subroutines	207
D.1.1 Opening the file	207
D.1.2 Locking and reading a record	208
D.1.3 Writing a record	209
D.1.4 Unlocking records	209
D.1.5 Closing a shared file	209
D.1.6 Record Locking Example	210
D.1.7 Modifying programs for TSX-Plus	210
D.2 Message communication subroutines	210
D.2.1 Message Channels	210
D.2.2 Sending a Message	211
D.2.3 Checking for Pending Messages	211
D.2.4 Waiting for a Message	211
D.2.5 Message Examples	212
D.3 Using the subroutines	212
D.4 Miscellaneous functions	212
D.4.1 Determining the TSX-Plus line number	212

Chapter 1

Introduction

TSX-Plus is a high performance operating system for Digital Equipment Corporation PDP-11 and LSI-11 computers, supporting as many as forty concurrent time-sharing jobs. TSX-Plus provides a multi-user programming environment that is similar to extended memory (XM) RT-11.

- TSX-Plus keyboard commands are compatible with those of RT-11.
- TSX-Plus supports most RT-11 system service calls (EMTs).
- Most programs that run under RT-11 will run without modification under TSX-Plus. This includes RT-11 utility programs such as PIP, DUP, DIR, LINK, KED, BUP and MACRO.
- TSX-Plus uses RT-11 XM version device handlers.
- TSX-Plus provides PLAS extended memory services such as virtual overlays and virtual arrays, as well as support for extended memory regions.

TSX-Plus can simultaneously support a wide variety of jobs and programming languages including COBOL-Plus, FORTRAN, BASIC, DIBOL, DBL, Pascal, C, MACRO, IND, TECO and KED. TSX-Plus is used in educational, business, scientific and industrial environments. It can concurrently support commercial users doing transaction processing, engineering users performing scientific processing, system programmers doing program development, and real-time process control. Numerous application software packages compatible with TSX-Plus are available from other vendors.

TSX-Plus supports RT-11 system service calls (EMTs) as its basic mode of operation. The result is low system overhead and substantially improved performance over systems that emulate RT-11 services. TSX-Plus overlaps terminal interaction time, I/O wait time, and CPU execution time for all jobs on the system. The result is a tremendous increase in the productivity of the computer system.

In addition to the basic RT-11 functionality, TSX-Plus provides extended features such as: process windows; shared file record locking; inter-job message communication; program performance monitoring; command file parameters; logon and usage accounting; directory and data caching; multitasking; and system I/O buffering.

This manual describes all the features unique to TSX-Plus as well as any differences from RT-11. Many of the special features of TSX-Plus are available as EMTs available to the MACRO programmer. Access to these features from other languages requires the appropriate subroutine interface.

TSX-Plus will run on any PDP-11 or LSI-11 computer with memory management hardware and at least 256 Kb of memory, although additional memory may be needed for satisfactory multi-user performance. The system must also have a disk suitable for program swapping (the swapping disk can be used for regular file storage as well). Time-sharing lines and serial printers may be connected to the system through either single-line or multiplexer type interfaces. See the *TSX-Plus Installation Guide* for a list of interfaces supported for use with terminals and communications lines. Both hardwired and dial-up time-sharing lines are supported by TSX-Plus.

1.1 Management of System Resources

1.1.1 Memory Management

TSX-Plus uses the memory management facilities of PDP-11 computers to keep several user jobs in memory simultaneously and switch rapidly among them. TSX-Plus protects the system by preventing user jobs from halting the machine or storing outside their program regions. TSX-Plus provides several ways to control the amount of memory used by individual jobs. Programs may be allowed to use up to 64 Kb of memory and, if additional space is needed, may also use extended memory regions, virtual overlays and virtual arrays. The system manager may enable job swapping to accommodate more user jobs than can fit in existing memory.

1.1.2 Execution Scheduling

TSX-Plus provides fast response to interactive jobs but minimizes job swapping by use of the patented Adaptive Scheduling Algorithm. TSX-Plus permits job scheduling on both an absolute priority basis and by a method based on job states. For most applications, the method based on job states is preferred. The *state driven* method provides the most transparent time-sharing scheduling, suitable for interactive environments. The absolute priority method always runs the highest priority executable job, when not servicing interrupts, regardless of that job's state. This *state free* method is most suitable for an environment in which several real-time jobs must be assigned absolute priorities. TSX-Plus permits both kinds of jobs to co-exist in the same system, with interactive jobs being scheduled whenever higher priority *state free* jobs are not executing.

Job priorities may be assigned over a range of 0 to 127. The lowest priority jobs, typically 0 to 19, are reserved for fixed priority jobs which can soak up system idle time without disturbing interactive or real-time jobs. The medium priority range, typically 20 to 79, is assigned to interactive jobs which are scheduled according to the Adaptive Scheduling Algorithm which makes time-sharing nearly transparent to several users. The highest priority range, typically 80 to 127, is reserved for jobs which must execute according to a rigid priority scheme such as might be found in a real-time environment. In addition, real-time jobs may execute interrupt service routines at fork level processing or schedule interrupt completion routines to run as *fixed high priority* jobs.

Job scheduling is controlled by several system parameters relating job priorities, system timing and other events. The *TSX-Plus System Manager's Guide* includes a more complete description of job priority and scheduling.

1.1.3 Subprocesses and Process Windowing

TSX-Plus allows the time-sharing user to interact with and control multiple executing processes. The primary process for a user is invoked when the user logs on; subprocesses can easily be initiated by typing a *control character digit* sequence. The Process Windowing (tm) facility causes the system to monitor all characters sent to the terminal and maintain an in-memory image of what is currently presented on the terminal screen. This allows the system to restore the terminal display when switching between processes. Process Windowing also provides a *print window* function which, when invoked by typing a control character, causes the current window contents to be printed on a specified hardcopy printer or saved to a disk file.

1.1.4 Directory and Data Caching

TSX-Plus provides a mechanism to speed up directory operations by caching device directories. This reduces disk I/O necessary to open existing files. Two methods of caching file data are also possible to further improve system throughput. When directory and generalized data caches are full, new data replaces *least recently used* existing data. When the shared file data cache is full, new data replaces *least frequently used* existing data. Directory and data caching are discussed in the *TSX-Plus System Manager's Guide*.

1.1.5 System Administrative Control

TSX-Plus allows the system manager to limit access to the system through a logon facility and to restrict user access to peripheral devices. These features are described in the *TSX-Plus System Manager's Guide*.

1.1.6 Time-sharing lines and CL units

TSX-Plus supports time-sharing terminal lines interfaced through several types of interface cards (see the *TSX-Plus Installation Guide* for a list of supported interface protocols). Using the CL feature provided with TSX-Plus, other serial devices such as printers, plotters and communications devices can also be attached using the same types of interfaces. Up to 16 CL units may be attached to TSX-Plus. (Units 0 through 7 are referenced as CL0 through CL7 respectively; units 8 through 15 are referenced as C10 through C17 respectively.) These devices may either be permanently assigned as CL units or may be used at some times as time-sharing lines and other times as I/O devices by having a CL unit *take over* an inactive time-sharing line. See the *TSX-Plus User's Reference Manual* description of the SET CL and SHOW CL commands for more information on using CL units. See the *TSX-Plus Installation Guide* for more information on including dedicated and extra CL units during system generation.

1.2 Summary of Chapter Contents

1.2.1 TSX-Plus Job Environment

TSX-Plus supports the use of extended memory regions through EMT calls compatible with those provided by the RT-11 XM monitor. This allows the use of virtual overlays and FORTRAN virtual arrays. Users can also expand programs to use the full 16-bit virtual address space. That is, by giving up direct access to the I/O page and direct access to fixed offsets in RMON, a program may directly address a full 64 Kb. On hardware which supports separate Instruction and Data space, programs may also use separate I- and D-space in conjunction with shared run-time regions and PLAS regions for a direct addressing capability of 128 Kb. A special "fast mapping" facility may be used to increase performance when it is necessary to re-map within shared run-time or PLAS regions.

1.2.2 Program Controlled Terminal Options

TSX-Plus allows the programmer to modify terminal handling characteristics during program execution. For example, a program may disable character echoing, use single character activation, use high efficiency output mode, enable lower case input, activate on field width, or disable automatic echoing of line-feed after carriage-return. These terminal options may be selected either by issuing the appropriate EMT request or by writing a special sequence of two or three characters to the terminal.

1.2.3 TSX-Plus EMTs

TSX-Plus supports most of the system service calls (EMTs) provided by RT-11 and, in addition, provides many more to utilize the special features of TSX-Plus. For example, EMTs are provided to determine the TSX-Plus line number and the user name, to send messages to another time-sharing line, to check for terminal input errors, and to check for activation characters. EMTs related to specific features, such as detached jobs, inter-program messages or real-time programming, are described in the relevant chapters. The TSX-Plus EMTs which are not closely related to features described elsewhere are discussed in Chapter 4.

1.2.4 Shared Files, Record Locking and Data Caching

TSX-Plus provides a file sharing mechanism whereby several cooperating programs may coordinate their access to common data files. Programs may request different levels of shared file access, and control shared access on a *block by block* basis. Two methods of data caching are also provided: 1) generalized data caching which is enabled when devices are MOUNTed; and 2) record caching which is only available to shared files. Directory caching is also enabled by the MOUNT request. This accelerates directory searching for file LOOKUPS.

1.2.5 Inter-program Message Communication

TSX-Plus offers a message communication facility that allows running programs to exchange messages. Messages are transmitted through named *message channels*. A program can queue messages on one or more message channels. Receiving programs can test for the presence of messages on a named channel and can suspend their execution until a message arrives. Receiving programs may also schedule a completion routine to be entered when a message arrives and continue other processing in the meanwhile. A message can be queued for a program that will run at a later time.

1.2.6 Programming for CL and Special Device Handlers

TSX-Plus provides a special serial communications line facility implemented as a device handler (CL), which may be used as a general purpose replacement for LS and XL (XC on the PRO). An interface port may be either defined as a time-sharing line or a communications port. Time-sharing lines may be later "taken over" as communications lines if the CL facility is included during system generation, either by a keyboard command or from within a program. Normal read/write functions may be used to CL units, but many special control features are available as special functions (.SPFUN requests). Certain other devices have special requirements or characteristics such as: DM, IB and VM.

1.2.7 Real-time Support

TSX-Plus provides real-time program support services that allow multiple real-time programs to run concurrently with normal time-sharing operations. Real-time programs may optionally lock themselves in memory, directly access the I/O page, redefine their memory mapping, and connect device interrupts to subroutines within the program.

1.2.8 Extended Memory Features

TSX-Plus allows one or more shared run-time systems to be mapped into the address space of multiple TSX-Plus time-sharing jobs. This saves memory space when multiple users are running the same types of programs (e.g., COBOL-Plus or DBL) and can also be used in situations where programs wish to communicate through a shared data region. Fast mapping may be used to improve performance when re-mapping within a shared run-time region. Program Logical Address Space (PLAS) facilities corresponding to those in the RT-11 XM monitor are available. Job memory images may be split into separate I- and D-space on those processors which support it. Separate I- and D-space may be used with shared run-time regions, PLAS regions, and fast-mapping.

1.2.9 Program Debugging Facility

TSX-Plus includes a symbolic (MACRO instructions) debugger with ODT styled commands. Debugger support is optionally included depending on a system generation parameter. The debugger need not be linked with the program being debugged and does not decrease the virtual memory space available to

programs. The debugger may be invoked as part of the RUN command, with a BPT instruction from within the program or with a special keyboard control character.

1.2.10 Program Performance Monitor Facility

TSX-Plus includes a performance analysis facility that can be used to monitor the execution of a program and determine what percentage of the run time is spent within certain program regions. When the performance monitor is being used, TSX-Plus examines the program being monitored at each clock tick (50 or 60 times per second) and notes the value of the program counter. On completion of the analysis, the TSX-Plus performance reporting program can produce a histogram of the time spent in various parts of the monitored program.

1.2.11 Differences from RT-11

Some inevitable differences exist between RT-11 and TSX-Plus. The *TSX-Plus User's Reference Manual* describes the additional keyboard commands provided by TSX-Plus, the minor differences in some commands, and the RT-11 keyboard commands not supported by TSX-Plus. Some other differences between RT-11 and TSX-Plus may not be obvious. The FORMAT utility is not supported. A few system service calls (EMTs) behave slightly differently in the two systems and some RT-11 EMTs are not supported by TSX-Plus (notably those supporting multi-terminal operations). A section is also included on special use and programming characteristics of some utility programs which may cause confusion because of some non-obvious interaction of their features with TSX-Plus.

1.2.12 Appendices

Appendix A describes the SETSIZ utility program which may be used to control the amount of memory available to programs.

Appendix B provides a table of EMT function and subfunction codes, and brief descriptions of both RT-11 and TSX-Plus EMTs; these are useful in conjunction with the SET EMT TRACE command.

Appendix C contains listings of common subroutines called by the example programs throughout this manual.

Appendix D describes a library of subroutines which are available to the DIBOL user to take advantage of some of the special features of TSX-Plus.

Chapter 2

TSX-Plus Job Environment

2.1 Simulated RMON

While TSX-Plus implements the system monitor differently than RT-11, it does simulate relevant RMON fixed-offset locations. (See the section below on user virtual address mapping.) TSX-Plus defines certain special negative offset values for use with .GVAL (they cannot be obtained by reading relative to the simulated base of RMON). These special negative offsets are described in Chapter 4. Some notable features of the simulation of offsets defined by RT-11 are described in the following table:

Offset	Interpretation
276	This word is normally copied from RT-11 during TSX-Plus initialization and indicates the version and release level of RT-11.
366	If the program is being run from a command file, then bits 8, 12, and 15 will be set in this word (mask 110400). If the program is not being run from a command file, then this word will be clear (0). The value of this offset is only reliable when accessed through the .GVAL request.
372	Bit 15 (mask 100000) is always set in this word by TSX-Plus; it is always clear under RT-11

2.2 Virtual and physical memory

The memory space that is accessible by a job is known as the *virtual address space* for the job. Because of the architectural design of the PDP-11 computer which uses 16 bits to represent a virtual memory address, the maximum amount of virtual address space that can be accessed at one time by a job is limited to 65,536 (64 Kb) bytes. Thus, the virtual addresses for a job range from 000000 to 177777 (octal). This may be doubled when separate I- and D-space addressing is enabled (see section 2.9).

The actual amount of virtual address space available to a job may be as large as 64 Kb but it may be restricted to less than this amount. The following factors control the size of the virtual address space available to a job:

- The maximum amount of memory allowed for each job as determined by the HIMEM system generation parameter.
- The amount of memory specified with the MEMORY keyboard command (initialized by the DFLMEM system generation parameter).
- The memory limit reserved in the disk file image by the SETSIZ program (see Appendix A).

- The amount of memory acquired by use of the TSX-Plus EMT that expands or contracts the job space.

The *physical address space* for a PDP-11 computer is not limited to 64 Kb. The maximum physical address space depends on the model of PDP-11 and the amount of memory installed on the computer. LSI-11/23 and 11/34 computers can access up to 256 Kb of physical memory. The 11/23-Plus, 11/73, 11/83, 11/24, 11/44, and 11/84 computers can access up to 4 Mb of memory.

The process by which an address in the job's virtual address space is transformed into an address in the physical address space is known as *mapping*. The mapping of the virtual address space for a job into the physical memory space assigned to the job is performed by the memory management hardware facility of the PDP-11 computer. This facility divides the virtual address space into eight sections, called *pages*, each of which can address up to 8 Kb of memory. The mapping of a page of virtual address space to a page of physical address space is accomplished by setting up information in a *page address register* (PAR). There is one page address register for each of the eight virtual address pages. These registers are not directly accessible by a user job but are loaded by the TSX-Plus system when it starts a program, changes the size of a program, or switches execution between different jobs. The relationship between the eight pages of memory and the corresponding sections of virtual address is shown in the following table:

Page	Virtual address range
0	000000-017777
1	020000-037777
2	040000-057777
3	060000-077777
4	100000-117777
5	120000-137777
6	140000-157777
7	160000-177777

Because of the design of the memory management system in the PDP-11, it is not possible to divide the virtual address space more finely than eight pages of 8 Kb each. However, it is possible to map each page of virtual address space into any section of physical memory. (This facility allows TSX-Plus to keep multiple user jobs in physical memory and to switch rapidly among them by reloading the page address registers.)

On newer PDP-11 models, a second set of user mapping registers is also available. In this case, and only when specifically enabled, the user virtual address space can be separated into 64 Kb of Instruction space and another 64 Kb of Data space. If the hardware does not support separate I- and D-space, or if it has not been explicitly enabled, then all references, both instruction and data, are made through I-space. See section 2.9 for more information on using separate I- and D-space.

2.3 User virtual address mapping

The virtual address space accessed by a job can be divided into five categories:

- Normal program space—which is used by instructions and data for programs.
- Simulated RMON—This is the virtual address region from 160000 to about 161300 which is mapped to a simulation of the fixed offset portion of RMON (RT-11 resident monitor).
- Extended memory regions—Programs can create regions in physical memory and then cause one or more pages of virtual address space to be mapped to the regions.
- Shared run-time systems—Several TSX-Plus jobs can cause a portion of their virtual address space to be mapped to the same area of physical memory. This allows several users to execute the same program or share common data without having to allocate a separate area of physical memory for each user.

- System I/O page—TSX-Plus real-time programs may map the system I/O page into their virtual address space.

2.4 Normal programs and virtual programs

Programs run under TSX-Plus may be divided into two categories: *normal programs* and *virtual programs*. The only difference between the two types of programs is the manner in which TSX-Plus handles page 7 (addresses 160000–177777) of the virtual address space. In the case of normal programs, page 7 is mapped to a simulated RMON. RMON is the name of the resident RT-11 monitor. When running under RT-11 this is the actual system control program. When running under TSX-Plus, the simulated RMON does not contain any of the instructions that are part of RT-11 but contains only a table that provides information about the system and the job. This information includes such items as the system version number, and information about the hardware configuration. The cells in this table are known as *RMON fixed offsets*. Their position within the table and their contents are documented in the *RT-11 Software Support Manual*, although not all cells are relevant to or maintained by TSX-Plus.

The address of the base of the simulated RMON table is stored in location 54 of the job's virtual address space. Modern RT-11 and TSX-Plus programs should not directly access the RMON table but rather should use the .GVAL EMT to obtain values from the table. However, since some older programs and some RT-11 utility programs directly access the RMON tables, it is mapped through page 7 for normal programs. As a result, normal programs are restricted to using pages 0 to 6 (56 Kb) for their own instructions and data.

Note that when simulated RMON is mapped into the job's virtual address space, it is mapped with read/write access. This makes it possible to corrupt data in the simulated RMON which should only be managed by the system. If a job does corrupt data in the simulated RMON cells, then it is possible for the job to receive erroneous error messages or to hang until the system is restarted.

Virtual programs are programs that do not require *direct* access to the simulated RMON table. These programs may still access the RMON values with the .GVAL and .PVAL EMTs. Since direct access to the simulated RMON is not needed, page 7 is available for the program to use for its own instructions and data, thus providing a total of 64 Kb of virtual address space. A program may indicate that it is a virtual program by any of the following techniques:

- Set bit 10 (VIRT\$—mask 2000) in the Job Status Word (location 44) of the SAV file. See Appendix A for information about how this bit can be set by use of the SETSIZ program.
- Use the /V LINK switch (/XM switch for the LINK keyboard command) which stores the RAD50 value for VIR in location 0 of the SAV file.
- Use the TSX-Plus SETSIZ program (see Appendix A) and indicate that more than 56 Kb of memory is to be used for the program.

2.5 Access to system I/O page

The *system I/O page* is an 8 Kb section of addresses which is not connected with ordinary memory but rather is used to control peripheral devices and hardware operation. Access to the I/O page is risky in that a program can interfere with peripheral devices and cause system crashes. For this reason, programs do not ordinarily have access to the I/O page. However, a program that is running with MEMMAP privilege may issue a system service call to cause a job's virtual address page 7 (addresses 160000–177777) to be mapped to the system I/O page. See Chapter 8 for more information on real-time programs.

2.6 Extended memory (PLAS) regions

Programs running under TSX-Plus have available the *Program's Logical Address Space* (PLAS) facility that is compatible with the RT-11XM monitor. This facility allows a program to allocate *regions* of physical memory and then create *virtual windows* that can be used to access the regions. There are 7 system service calls (EMTs) provided for PLAS support:

EMT	Meaning
.CRRG	Create a region
.ELRG	Eliminate a region
.CRAW	Create a virtual address window
.ELAW	Eliminate a virtual address window
.MAP	Map a virtual window to a region
.UNMAP	Unmap a virtual window
.GMCX	Get information about the status of a window

In addition, PLAS regions may be mapped into a program's virtual address space by using "fast mapping". See Chapter 9 for use of "fast mapping" with shared run-time and PLAS regions.

PLAS regions may be mapped (or fast mapped) through either I- or D-space. See section 2.9 for more information on using I- and D-space with PLAS regions.

A region is an area of physical memory set aside for use by a job in addition to its normal job space. The .CRRG EMT is used by a program to request that a region be created. The size of a region is not restricted to 64 Kb and may be as large as the physical memory installed on the system (less the space used by the TSX-Plus system, device handlers, tables, and the remainder of the program). Up to eight unnamed regions may be created by each job.

PLAS memory regions can be grouped into two main categories: named regions and unnamed regions. Unnamed PLAS regions can only be accessed by the job that created them and only remain in existence as long as the job that created them is running. They are always deleted when the job exits or chains. Named PLAS regions may be either private or sharable between multiple jobs. Use of named PLAS regions requires SYSGBL privilege. Unlike unnamed PLAS regions, named regions are not necessarily deallocated when the program which created the region terminates execution. Named regions may be used to communicate between programs, to hold common code executed by multiple users (e.g., shared run-times), and to pass information from one program to another program—possibly run at a later time.

In addition to supporting named global PLAS regions in a fashion compatible with RT-11, TSX-Plus also provides an additional facility known as *local named regions*. Local named regions are regions which can only be accessed by the job that created them. Their names are private to the creating job and more than one job may create (different) local regions with the same name. Local named regions are deallocated when the creating job specifies that they are to be eliminated or when the creating job logs off.

Local named regions are allocated memory space associated with the creating job and are swapped in and out of memory with the job (like unnamed regions). Unnamed regions and local named regions may not be "fast mapped". Global named regions are allocated memory space at the top of the area of memory used for jobs. They are never swapped out of memory and are only deallocated when a job eliminates the region. Global named regions may be "fast mapped". Thus global named regions may continue to occupy memory after the job that created the region logs off. Caution should be exercised when creating global regions since it is possible to lock jobs out of memory by creating large global regions.

Local named regions are distinguished from global named regions at creation time by setting bit 0 (mask 000001) in the status word (the third word—R.GSTS) of the region definition block used with the .CRRG EMT.

Summary of Characteristics for Extended Memory (PLAS) Region Types

Region Type	Max lifetime of region	Accessible by other jobs?	Swapped out of memory with job?	Fast mappable?
Unnamed	Until program exits	No	Yes	No
Named local	Until job logs off	No	Yes	No
Named global	Indefinite	Selectable	No	Yes

A SHOW REGIONS keyboard command may be used to display information about local and global named regions accessible by the job.

The REMOVE keyboard command may be used to eliminate a local or global named region. The form of this command is:

REMOVE region

In order to access a region, a program must use the .CRAW and .MAP EMTs to create a *virtual window* and map the virtual window to a selected portion of the region. A virtual window is a section of virtual address space mapped to a region rather than to the normal job physical address space. Up to eight virtual address windows can be created by each job. The same virtual window (i.e., the same range of virtual addresses) may be mapped to different regions or different sections of the same region at different times by use of the .MAP EMT. This allows a program to selectively access different sections of code or data in extended memory regions during the course of its execution.

Note that RT-11 allows a .CRAW and .MAP with a region ID equal to zero. TSX-Plus requires that the window definition block specified by the .MAP EMT contain a non-zero region ID returned from the successful creation of a region by use of the .CRRG EMT. TSX-Plus will return an error code 2 on the .MAP EMT when a zero region ID is specified.

When an unnamed or named local PLAS memory region is created by use of the .CRRG EMT, space is allocated in physical memory and in a TSX-Plus region swap file. Whenever a job is swapped out of memory, its extended memory regions are swapped to the region swap file. Space in the region swap file is allocated and deallocated dynamically as regions are created and eliminated. In order to create a region, space must be available in physical memory and in the region swap file.

The PLAS facility is most often used implicitly through the virtual overlay and virtual array features. Using the PLAS facilities, it is possible for a single job to use all of the physical memory space available on a system (exclusive of the space used by the TSX-Plus system, handlers, tables, etc.). Proper use of the PLAS facilities such as with reasonable size virtual overlays or arrays can lead to substantial performance improvements for programs. Excessive use of memory space with the PLAS facility can lead to excessive job swapping and degraded system performance.

Using I- and D-space with PLAS Regions

PLAS is an acronym for Program's Logical Address Space. This is a feature of the RT-11 XM (eXtended Memory) monitor which allows programs to address extended memory regions. TSX-Plus implements EMT's which are compatible with the PLAS requests of RT-11 XM so that programs which use PLAS features can be run without modification. The PLAS EMT's are used to implement FORTRAN virtual arrays (the FORTRAN IV OTSGEN VIRP option) and to support the virtual overlay handler from LINK. They may also be manipulated directly from MACRO programs.

When a PLAS region is created, an appropriate size region of memory is reserved from the pool of physical memory available to user jobs. If the region is "global", then it remains reserved until the region is eliminated and is never swapped out of memory. If the region is "local", then it may be swapped out to the PLAS region swap file if it becomes necessary to swap the job's static region. Creation of a PLAS region does not, by itself, affect the job's memory mapping. This is accomplished either explicitly by the .MAP request or

implicitly as an option to the .CRAW request. PLAS regions may also be mapped using TSX-Plus “fast mapping” as described in section 9.2.1. PLAS regions are mapped by default through I-space, analogous to shared run-time mapping. However, TSX-Plus defines a bit in the Region Status Word which specifies that when separate I- and D-space has been enabled then all windows into the region should be mapped instead through D-space. Using separate I- and D-space mapping allows PLAS regions to be mapped either through I-space or through D-space, leaving the static regions (which otherwise would have been “mapped away”) accessible through the other mapping space (D or I). In fact, it is possible to map some PLAS regions through I-space and simultaneously map other PLAS regions through D-space. TSX-Plus also defines a bit in the window status word which allows PLAS windows to have overlapping virtual address ranges, thus allowing complete remapping of a job’s virtual address space — even to “map away” the static region when separate I- and D-space is enabled.

Note that when separate I- and D-space is enabled, the space through which windows may be mapped into PLAS regions is determined by region characteristics, not the window characteristics. This means that it is not allowed to map one window into a PLAS region through I-space and another window into the same region through D-space. The space through which all windows into a single region are mapped is determined by the state of the “use D-space” bit in the region status word at the time of region creation.

The following non-standard PLAS status bits are defined:

Word	New bit	Mask	Meaning when set
R.GSTS	RS.PVT	000001	This is a private region (defined since v4.0)
R.GSTS	RS.DSP	000002	This region is to be mapped through D-space (new)
W.NSTS	WS.OVR	002000	This window may overlap other windows (new)

The following new error codes are defined for the .CRRG request:

Error Code	Meaning
20	Attempt to create D-space region and hardware does not support separate I- and D-space.
21	Attempt to create D-space region and job has not already enabled separate I- and D-space mapping.

2.7 Shared run-time systems

A *shared run-time system* is a program or data area in physical memory that can be accessed by multiple TSX-Plus jobs. Shared run-time systems are somewhat similar to extended memory regions in that they are both allocated in extended memory areas and must be accessed by mapping a portion of the job’s virtual address space to the physical memory area. The difference is that extended memory regions are private to the job that creates them and may not be accessed by any other job. Shared run-time systems can be simultaneously accessed (hence “shared”) by any number of TSX-Plus jobs. Another difference between regions and shared run-time systems is that regions can be created dynamically and can be swapped out of memory; shared run-time systems are specified when the system is generated and reside in memory as long as the system is running. See Chapter 9

2.8 Rules for Mapping to extended memory regions

The static region of a job is always mapped by a call to SETMAP. Mapping to extended memory regions may be done by a variety of requests: real-time EMT 375, 17,140 to map to a physical address (I-space only); shared run-time mapping (EMT 375, 1,143; or fast mapping with the TRAP instruction) to a shared

run-time region (I-space only); and PLAS requests (.MAP, WS.MAP set during .CRAW or fast mapping with the TRAP instruction) to extended memory regions (either I- or D-space). The following paragraphs discuss special mapping notes (idiosyncrasies if you will) for these methods.

Note that combinations of shared run-time, real-time, and PLAS mapping are possible but are strongly discouraged.

The static region

The SETSIZ program can be used to set the size of the static region to be allocated when a program is started. (See the appendix on SETSIZ in the *TSX-Plus Programmer's Reference Manual* for more information on job sizing.) This may also be done with the LINK /K:n switch or internally to the program as follows:

```
.ASECT
. = 000056          ;Address modified by SETSIZ
.WORD 28.          ;# of K-words for program == 56 Kb
```

If a program is restricted to a reduced memory allocation with SETSIZ or with the MEMORY command or from within the program by the TSX-Plus EMT to change a job's size (EMT 375 0,141), then the highest valid static region address will be correspondingly restricted. This is done by establishing the program top address and then calling SETMAP. This does not prevent a job from mapping virtual addresses above this limit to physical memory, to shared run-times or to PLAS regions.

Real-time mapping

When PAR 7 is mapped to the IO page or to simulated RMON (with real-time EMTs 5,140 and 6,140 respectively or as a consequence of the RUN/IOPAGE command or the program being installed with the IOPAGE attribute), addresses 160000-177777 are not treated as extended memory. Thus, the value returned by the PHYADD EMT (0,140) will be incorrect and will instead return the physical address that would correspond to the specified virtual address offset from the physical base of the job.

The MAPPHY EMT to map to a physical address (17,140) is treated as extended memory and will be correctly handled by PHYADD (I-space only). MAPPHY simply sets the appropriate I-space PAR and PDR values; it does not invoke SETMAP. It does not support D-space. When MAPPHY is executed and the specified size is zero (0), then the specified extended PAR is released and SETMAP is called.

Shared run-times

The EMT to associate with a shared run-time does not immediately affect job mapping. However, when it is called with zero (0) as the pointer to the region name, then all shared run-times are released. In this second form all extended PAR and PDR values are cleared, all PLAS regions are remapped and SETMAP is called. Note that a side effect of this behavior is that any regions mapped via MAPPHY will be unmapped.

The EMT to map to a shared run-time region (1,143) simply sets the appropriate I-space PAR and PDR values; it does not invoke SETMAP. These are treated as extended memory - PHYADD (the I-space form) will return correct values. Shared run-times may not be mapped through D-space.

The EMT to establish fast-mapping to shared run-time regions does not immediately affect job mapping but rather creates entries in the job's fast-map tables. See the section on fast-mapping below.

PLAS regions

PLAS regions may be mapped either through I-space or D-space and are always treated as extended memory regions. Thus, PHYADD will return correct results, but the correct form must be used (I-space or D-space).

When a PLAS region is mapped with the .MAP request or because the WS.MAP bit was set during a .CRAW request, the appropriate I-space or D-space PARs and PDRs are set immediately; SETMAP is not called.

When a window is unmapped with the .UNMAP request or as a consequence of the .ELAW or .ELRG requests, then the PAR and PDR registers affected by the window are cleared, as well as any affected fast-map entries. Then SETMAP is called.

The EMT to establish fast-mapping to PLAS windows does not immediately affect job mapping but rather creates entries in the job's fast-map tables. See the section on fast-mapping below.

Fast mapping

Fast-mapping can be used to map shared run-time regions into the job's virtual I-space addresses, or to map PLAS regions into the job's virtual I- or D-space addresses. Fast-map regions are treated as extended memory and PHYADD will return valid results. When the TRAP instruction is used to fast-map an extended memory region (either shared run-time or PLAS), then the appropriate PAR and PDR values are set immediately based on information in the job's fast-map tables. SETMAP is not called. The .UNMAP request does result in a call to SETMAP and will clear any current fast-mapping.

2.9 Support for Separate I- & D-Space

2.9.1 How to break the 64 Kb barrier

The PDP-11 architecture has a 16-bit address range, which implies that programs may only directly access at any single instant up to 64 Kb of memory. Several schemes have been developed to circumvent this program size limitation, such as disk and virtual overlays, virtual arrays, shared run-time regions, automatic program segmentation and program chaining. However, the architectural restriction of a 16-bit address space means that use of any of these techniques imposes some extra computational or I/O overhead. The PDP-11/23 has eight mapping registers for user programs which TSX-Plus uses to position jobs throughout physical memory. This means that each mapping register controls up to 8 Kb of user memory ($64 \text{ Kb address space} / 8 \text{ mapping registers} = 8 \text{ Kb per register}$). The memory management hardware of more powerful PDP-11 implementations supports two separate sets of eight user mapping registers, one for instructions and one for data. This allows simultaneous mapping to 64 Kb of instructions (I-space) and separate mapping to an additional 64 Kb of data (D-space). The processors which have memory management units that support separate I- and D-space are: 11/73, 11/83, 11/44, 11/84, and the Pro-380. This feature is not available on the 11/23, 11/24, 11/34 or Pro-350.

Starting with TSX-Plus V6.40, it is now possible for programs which use certain special coding techniques to take advantage of the separate I- and D-space memory management hardware on processors which support it. This allows programs to have up to 64 Kb of program space and 64 Kb of data space, for a total instantaneous addressing capability of 128 Kb. Separate I- and D-space may be used in conjunction either with shared run-time regions or with PLAS regions. This makes the additional addressing capacity of separate I- and D-space most conveniently available when using language processors and run-time systems that transparently manage I- and D-space mapping. Currently, DISC plans to use this feature to improve performance in future releases of DBL for TSX-Plus.

Separate I- and D-space is expected to be used in combination with shared run-time code regions mapped through I-space or PLAS regions mapped through either I- or D-space. Special shared run-time fast mapping works as before. If separate I- and D-space is not enabled, then shared run-time mapping and fast mapping are unchanged from their behavior in previous versions.

For the adventuresome, the next sections present a brief description of program virtual memory organization and an overview of the memory management services available to the TSX-Plus programmer as background for the subsequent description of the system services which have been implemented to support separate I- and D-space addressing.

It is not within the scope of this discussion to present a tutorial on separate I- and D-space programming techniques. Briefly, instruction fetches are made from I-space and data references are to and from D-space. See the *PDP-11 Architecture Handbook* from *Digital Equipment Corporation* for more information on the technical aspects of separate I- and D-space.

2.9.2 User memory organization and mapping

When a program is started, TSX-Plus obtains memory mapping information for the job from block 0 of the .SAV image. Among other things, it checks the program high limit, whether it is overlaid, the initial stack pointer, the job status word, location 0, and location 56 (used by SETSIZ). Based on the program size and characteristics and how much memory is allowable for the job, the memory mapping is set up, the program is loaded and its execution begun. This contiguous section of physical memory into which the program is loaded is called the **static region**. In general, the static region is set up so that the program is loaded with virtual addresses from 000000 up to 157777 (PAR 0 through PAR 6). PAR 7 is normally used to map to a simulated copy of RMON fixed offsets, and is mapped from 160000 through the top of RMON (typically about 700. bytes). Virtual programs are those which do not require direct access to simulated RMON and so can have a static region which extends all the way through PAR 7. (In this case, the contents of fixed RMON offsets may still be obtained by the .GVAL request.) Another common alternative is to map PAR 7 to the hardware I/O page. All these initial mapping permutations occur solely and exclusively through user mode I-space.

Once started, programs can further influence their memory mapping through PLAS requests, or by mapping to shared run-time regions, or through real-time requests. Using the new split I- and D-space requests, programs may simultaneously access more than 64 Kb of memory when using shared run-time or PLAS regions. When a program issues a request to map to an extended region, the appropriate memory management register settings are made (and copied into the job's context region for use in the event of later remapping).

A job's initial memory image and mapping is determined when loading a program (.SAV) image. It may be affected during execution by the job size or mapping EMTs described in the previous paragraph. The system routine used to establish a job's memory mapping when first loaded, after a context switch, after the job sizing EMT (0,141), or after certain extended memory operations is called SETMAP. The SETMAP routine uses the following algorithm:

- Set up mapping registers for I-space through the program top (which may extend into PAR 7).
- If separate I- and D-space is enabled, set identical values in the D-space mapping registers.
- Disallow access to I-space (and D-space if enabled) virtual addresses above the program top. (Attempting to access disallowed virtual addresses results in a trap to 4.)
- Set PAR 7 mapping to the I/O page (if requested) through I-space (and identically through D-space if enabled). This is *not* treated as extended memory.
- If the program is not a virtual job and has not mapped PAR 7 to the I/O page, map part of PAR 7 to a simulated copy of RMON through I-space (and D-space if enabled). Note that the simulated RMON does not extend through the full PAR and references above its top are disallowed. This is *not* treated as extended memory.
- Map extended memory regions which should be directed to PLAS, shared run-times or regions mapped to a physical address with the real-time EMT (17,140). Shared run-times and real-time mapped regions may only be mapped through I-space. PLAS regions may be mapped through either I- or D-space.

2.9.3 I/O to separate I- and D-space regions

When I/O buffers specified in read or write requests reside in virtual addresses which are covered by separately mapped I- and D-space regions, then the data transfer is always directed to the D-space. (When separate I- and D-space is **not** enabled, then data transfers are directed to I-space.) The real-time EMT to convert a virtual address to a physical address (0,140) applies only to I-space. A new EMT (22,140) has been created to convert D-space virtual addresses to a physical address. When it is necessary to convert a virtual address to a physical address, be sure to use the appropriate request. If it is necessary to perform I/O to a buffer in an I-space region, then D-space must be first disabled.

The entire I/O buffers specified by read or write requests must be contiguously mapped. It is not allowed to specify an I/O transfer buffer which spans mapping boundaries. For example, assume a program has enabled separate I- and D-space and used PLAS mapping requests so that its memory organization looks something like:

	Virtual addresses	I-space	D-space
PAR 0	000000 - 017777	static	static
PAR 1	020000 - 037777	static	PLAS-1
PAR 2	040000 - 057777	static	PLAS-3
PAR 3	060000 - 077777	SRT-1	static
PAR 4	100000 - 117777	SRT-1	static
PAR 5	120000 - 137777	PLAS-2	static
PAR 6	140000 - 157777	PLAS-2	PLAS-3
PAR 7	160000 - 177777	RMON	RMON

In this example, portions of the job are mapped to six different contiguous segments of memory. Under TSX-Plus, the static region is always contiguous in physical memory, as are PLAS regions and shared run-time regions, although mapping windows into them need not be contiguous. Because I/O is never allowed to I-space regions, it would not be possible to read or write to either the SRT-1 shared run-time region or to the window into the PLAS-2 region. Although it would be allowed to write the simulated RMON region to disk, that would not be particularly useful, and it would be a *terrible* idea to read into it. So, I/O to the simulated RMON addresses is irrelevant. Transfers could be made to or from either of the PLAS-1 or PLAS-3 regions. However, no single transfer should be specified which would span the boundary between them, or between either of them and the static region.

TSX-Plus does not monitor or enforce restrictions on I/O requests which attempt to cross mapping boundaries — that responsibility remains with the programmer. Attempts to perform I/O across mapping boundaries will either result in unwanted results or will cause an I/O error for attempting to access unauthorized memory. For example, it might be possible to have mapped the PAR 2 and PAR 6 D-space virtual address windows to physically adjacent segments of the PLAS-3 region. Then, specifying a 12 Kb I/O request which started at 040000 (base of PAR 2) should go midway through PAR 3. However, because of the way this job is mapped and the invalid crossing of a mapping boundary, the actual transfer would probably actually cover virtual addresses 040000 to 057777 and 140000 to 147777. Besides being stylistically offensive and a nightmare to debug, I/O requests which attempt to cross mapping boundaries are not supported by TSX-Plus and any unpredictable results which may result from them are explicitly disclaimed.

2.10 VM pseudo-device handler

While the VM handler is not actually mapped into a job's memory space, its use can dramatically increase job performance. The VM handler enables the use of a portion of physical memory as a pseudo-disk device. This permits very rapid access to programs and data which are placed on the VM unit. For programs such as compilers which heavily utilize overlay segments, a considerable speedup can be achieved by loading them

onto the VM device. A similar improvement for overlaid programs can be obtained with the general data cache facility. However, when the data cache is full the least recently used blocks are lost. The presence of particular programs and their overlay segments in memory can be guaranteed by copying them to the VM pseudo-device. Another example of the usefulness of the VM device is with compilers which heavily use temporary work files. Depending on the number of write operations, which are not helped by general data caching, significant improvements in speed can be obtained by directing the work files to the VM pseudo-device.

In order to use the VM device, it must be included in the device definitions during TSX-Plus system generation. An upper limit must also be placed on the amount of memory available to TSX-Plus. The physical memory above that available to TSX-Plus can then be used as a memory based pseudo-disk. If for some reason, it is desirable to use less than all of the memory above the top of TSX-Plus, the SET VM BASE command can be used to restrict the memory available to VM. Each time TSX-Plus is restarted, VM must be initialized just as you would for a new physical disk or a fresh logical subset disk. For example:

INITIALIZE VM:

Only one unit (VM0:) is available. However, logical subset disks may be created within the VM pseudo-device to partition it if necessary. On initialization, the VM handler automatically determines the amount of memory available to it.

See the *TSX-Plus System Manager's Guide* for more information on the use of data caching (general and shared files) and the VM pseudo-disk. See Chapter 7 for descriptions of the special function requests supported by VM.

2.11 Job priorities

TSX-Plus jobs may be assigned execution priorities to control their scheduling relative to other jobs. The priority values range from 0 to 127. The priority values are arranged in three groups:

- The fixed-low-priority group consists of priority values from 0 up to the value specified by the PRILOW sysgen parameter.
- The fixed-high-priority group ranges from the value specified for the PRIHI sysgen parameter up to 127.
- The middle priority group ranges from (PRILOW+1) to (PRIHI-1).

Job scheduling is performed differently for jobs in each category. Jobs in the fixed-high-priority range are scheduled strictly according to their priority and execute before any other job with lower priorities, including all interactive and fixed-low-priority jobs. Jobs in the interactive range are scheduled according to a patented algorithm which gives precedence to terminal operations. Jobs in the fixed-low-priority range are also scheduled strictly according to priority, but execute only when no other jobs of higher priority are executable, including all fixed-high-priority and interactive jobs.

Job priorities may be influenced by the SET PRIORITY and SET PROCESS/PRIORITY commands (see the *TSX-Plus User's Reference Manual*). Limits may be set on job priorities by the TSAUTH program and the SET MAXPRIORITY command (see the *TSX-Plus System Manager's Guide*). An executing program may also influence its own priority (see EMT 375, function 0,150 in Chapter 4). The current priority for a job and the maximum authorized priority can be displayed by use of the SHOW PRIORITY keyboard command, and may be obtained from within programs with the .GVAL request.

See the description of EMT 375 function 0,150 in Chapter 4 and the *TSX-Plus System Manager's Guide* for more detailed discussions of job priorities and their effect on job execution.

2.12 User command interface

TSX-Plus provides a method of intercepting and preprocessing user typed commands. This is called the "User Command Interface (UCI)". This may be implemented by writing a program to handle user input and enabling the facility with the SET KMON UCI command. After UCI is enabled, the TSX-Plus keyboard monitor will run the user-written UCI program each time it needs a new command. The program must prompt the user for a new command, accept the command, process it as desired and may optionally pass the command to the TSX-Plus keyboard monitor by doing a *special chain exit*. A special chain exit is performed by issuing the .EXIT request with bit 5 (mask 40) set in the job status word and with R0 cleared. Any commands to be executed by the keyboard monitor are passed through the chain data area. See the *RT-11 Programmer's Reference Manual* for more information on *special chain exits*. Commands passed to the TSX-Plus keyboard monitor in this fashion behave as though the keyboard monitor obtained them from a command file. A command file name may also be passed to the keyboard monitor by passing a command of the form "@name". If a command file name is passed to the keyboard monitor, then it must be the last or only command passed in the chain data area. When a command file name is passed in this manner, then all of the commands included in the command file are executed by the keyboard monitor before returning to the user-written UCI program for another command.

The following program provides a simple example of the techniques for writing a User Command Interface program. This program accepts a command from the keyboard and passes it through to the TSX-Plus keyboard monitor if it is a legal command.

Example

```
.TITLE MYKMON
.ENABL LC
; Simple example of User Command Interface
; Refuses to pass SET KMON SYSTEM, but otherwise does nothing but
; pass commands thru to KMON.
.MCALL .PRINT,.EXIT,.GTLIN,.SCCA
JSW      = 44          ;Job status word address
SPXIT$   = 40          ;Special exit flag to pass command to KMON
MONPTR   = 54          ;Pointer to base of RMON
SYSGEN   = 372         ;Offset into RMON of SYSGEN options word
BEL      = 7           ;ASCII bell
BS       = 10          ;ASCII backspace
LF       = 12          ;ASCII line feed
FF       = 14          ;ASCII form feed
CR       = 15          ;ASCII carriage return
ESC      = 33          ;ASCII escape
.DSABL   GBL           ;Disable undefined globals
START:   .SCCA #AREA,*TTSTAT ;Inhibit control-C abort
MOV      @#MONPTR,R0    ;Get pointer to base of RMON
TST      SYSGEN(R0)     ;Are we running under TSX?
BPL      QUIT           ;Normal exit if not
MOV      #TTYTYPE,R0    ;Point to EMT arg block to
EMT      375            ;Get TSX-Plus terminal type
ASL      R0             ;Convert to word offset
CMP      R0,#4          ;Legal types are unknown, VT52 and VT100
BLOS     1$
CLR      R0             ;If not VT52 or VT100, make unknown
1$:      MOV      R0,R1    ;Save terminal type
.PRINT   CLRSCR(R1)     ;Clear the screen
.PRINT   #MENU          ;Display simple menu
MOV      #SETRUB,R0     ;Point to EMT arg block to
EMT      375            ;Set rubout filler character
2$:      .PRINT   CENTER(R1) ;Move to screen center and clear the line
.GTLIN   #BUFFER,#PROMPT ;Accept input line
CALL     MATCH          ;See if it is legal
BCS      2$             ;Repeat if illegal command
MOV      #1000,SP       ;Ensure stack pointer safe
CALL     MOVCMO         ;Move command from buffer to chain data area
BIS      #SPXIT$,@JSW   ;Set special chain exit bit in JSW
```

```

CLR      R0                ;Required for special chain exit
QUIT:    .EXIT              ;And pass command to KMON
                ;Simple matching. Easy to defeat by
                ; inserting extra spaces!!!
MATCH:    MOV      #BUFFER,R2    ;Point to beginning of input buffer
          MOV      #ILLCMD,R3    ;Point to beginning of illegal command
1$:      TSTB      (R2)          ;At end of input string?
          BEQ      2$            ;Yes, matched so far, probably illegal
          CMPB     (R2)+,(R3)+    ;No, test through end of illegal string
          BNE      9$            ;No match, not illegal command
          CMP      R3,#ILLEND     ;Past end of illegal command?
          BLO      1$            ;No, keep checking
2$:      SEC                      ;Strings match, signal illegal command
          BR       10$
9$:      CLC                      ;Strings do not match, signal legal command
10$:     RETURN
;
;Move command from input buffer to chain data area.
;
MOVCMD:   MOV      #BUFFER,R2    ;Point to beginning of input string
          MOV      #512,R3        ;Point to chain data area
1$:      MOVB     (R2)+,R0        ;Get next char
          BNE      2$            ;Continue if not nul
          CLRB     (R3)+          ;If end of input command
          BR       9$            ; then done
2$:      CMPB     R0,#'\          ;Command separator?
          BNE      3$            ;No, move it
          CLRB     R0            ;Yes, replace with nul
3$:      MOVB     R0,(R3)+        ;Move command into chain data area
          CMP      R2,#BUFEND     ;Do not want to overflow
          BLO      1$            ;Keep moving if characters left
          CLRB     -1(R3)         ;Mark end of command (ensure it is ASCII)
9$:      SUB      #512,R3         ;How many bytes did we move?
          MOV      R3,#510        ;Mark the number for .CHAIN
          RETURN
AREA:     .BLKW     10            ;GP EMT argument area
TTSTAT:   .WORD     0            ;Terminal status word for .SCCA
TTYTYPE:  .BYTE     0,137        ;EMT arg block to get terminal type
SETRUB:   .BYTE     0,152        ;EMT arg block to control terminal funtions
          .WORD     'A            ;Function code - set rubout filler
          .WORD     '_'          ;Rubout filler = underline
CLRSCR:   .WORD     CLRUNK,CLR52,CLR100 ;Terminal specific screen clears
CENTER:   .WORD     CNTUNK,CNT52,CNT100 ;Terminal specific move and clear
          .NLIST    BEX
CLRUNK:   .BYTE     FF,FF,FF,CR,200 ;Emulate clear screen with 3*(8LFs)
CLR52:    .BYTE     ESC,'H,ESC,'J,200 ;VT52 clear screen sequence
CLR100:   .ASCII    <ESC>/[H/<ESC>/[J/<200> ;VT100 clear screen sequence
CNTUNK:   .ASCII    <CR><LF><LF><LF>/ /<200>
CNT52:    .ASCII    <ESC>/Y% / ;Line 6, column 1
          .ASCII    <ESC>/K/ ;Erase to end of line
          .ASCII    / /<200> ;Move to column 6
CNT100:   .ASCII    <ESC>/[6;6f/ ;Line 6, column 6
          .ASCII    <ESC>/[2K/<200> ;Erase entire line
MENU:     .ASCII    <LF><LF><LF>/ ***** Simple Menu *****/<200>
PROMPT:   .ASCII    <BEL>/Command: -----/
          .NLIST
          .REPT     32.
          .BYTE     BS ;Backspace to beginning of field
          .ENDR
          .LIST
          .BYTE     <200> ;End of string
ILLCMD:   .ASCII    /SET KMON SYSTEM/ ;Do not permit UCI disable
ILLEND:
BUFFER:   .BLKB     81. ;Command line input buffer
BUFEND:
          .END      START

```


Chapter 3

Program Controlled Terminal Options

3.1 Terminal input/output handling

The terminal keyboard and screen provide the principal interface between a time-sharing user and the TSX-Plus operating system. TSX-Plus accepts characters from the keyboard, echoes them to the screen, and stores them in a separate buffer for each time-sharing user. When a program (either a user written program, a utility, or the operating system keyboard monitor) requests input from the terminal, characters are removed from the internal buffer and passed to the program.

3.1.1 Activation characters

The low-level requests for input from a program can call for a single character (.TTYIN), for an entire line (.GTLIN, .CSIGEN, .CSISPC), or for a whole block of characters (.READ). Since the requests for a whole line of input are most common, TSX-Plus improves overall efficiency for many users by retaining characters typed at the keyboard in an internal buffer until a special character is typed which indicates that the line of input is complete. This special character, which indicates that keyboard input is ready, is called an *activation character*. The standard activation characters are carriage return and line feed. Several control keys will also cause immediate system response. For example, CTRL-C is used to abort the execution of a running program. If the program is waiting for input, one CTRL-C will cause an immediate abort. If the program is not waiting for input, it is necessary to type two CTRL-Cs to get the system's attention to abort a program.

When a program requests terminal input, TSX-Plus puts the program in a suspended state until an activation character is typed. This state, in which a program is waiting for input but no activation character has been typed, is identified as the TI state by the SYSTAT command. When characters are typed at the terminal, TSX-Plus responds quickly and stores them in the terminal input buffer for that line, then returns to process other jobs which need its attention. Thus, the amount of time the CPU spends processing input characters is kept to a minimum, and the amount of CPU time used by a program in the TI state is also very small. Some programs request single characters with the .TTYIN request. Normally, these programs are treated by TSX-Plus in the same way as those requesting lines of input (e.g., .GTLIN requests). That is, the job is suspended, input characters are stored in the terminal input buffer, and characters are only passed to the program after an activation character is typed. If a program requests a character with a single .TTYIN, the user can type as many characters as the terminal input buffer will hold (allocated during system generation), but the program will remain suspended and no characters are passed to the program. Then, when an activation is typed, the program is restored to an active state, the first character in the input buffer is passed to the program and processing continues. If the program requested no more characters, then on program exit the remainder of the input buffer, including the activation character, would be passed to the next program (usually the keyboard monitor) which would try to interpret them. This may result in an invalid command error message.

3.1.2 Single character activation

TSX-Plus gives the programmer a wide variety of ways to influence the normal input scheme outlined above. One of the most common methods is the use of *single character activation*. With this technique, all characters are regarded as activation characters. If a program requests a single character with a .TTYIN, then as soon as a character is typed and becomes available in the input buffer, it is passed to the program and the program resumes execution.

The standard way to request single character activation under RT-11 is by setting bit 12 in the user's Job Status Word (JSW). Under TSX-Plus, this is not by itself sufficient to cause single character activation. The reason is that quite a few programs designed for a single user environment use this method in a way that causes constant looping back and consequently *burns up* a large amount of processor time. In a single user environment this is of minor importance since no other jobs are trying to use the processor at the same time. In a multi-user system, this is wasteful and should be avoided. Therefore, under TSX-Plus, setting bit 12 in the JSW is not by itself sufficient to initiate single character activation. It is necessary *BOTH* to set bit 12 and to issue a special command to TSX-Plus indicating that single character activation is actually desired. This may be done in any of the following ways:

- Specify the /SINGLECHAR switch with the RUN or R command that starts the program.
- Issue a SET TERMINAL SINGLE command.
- Use the INSTALL command to install the program with the SINGLECHAR attribute (the description of the INSTALL command is in the *TSX-Plus System Manager's Guide* .
- Use the "S" program controlled terminal option described in this chapter.

Note that when a program is in *single character activation* mode, the system does not echo terminal input, it is the program's responsibility to do so.

3.1.3 Non-blocking .TTINR

The situation in which a program requests single characters but none are available in the input buffer also receives special treatment. The single character input request is eventually coded as EMT 340. The .TTYIN request repeats this request until a character is finally obtained, whereas the .TTINR request supposedly permits processing to continue if no character is available. In fact, the EMT 340 call will suspend the job until a character is available from the input buffer. This is referred to as *stalling* on a .TTYIN. The purpose is to avoid the unnecessary looping back to get a character. Under RT-11, if the programmer decides not to wait for a character to become available, but rather proceed with execution, it is only necessary to set bit 6 (100 octal) in the Job Status Word. Some programs abuse this technique and would waste the system resources in a time-sharing environment. TSX-Plus requires confirmation that the user is aware of the extra system load that could be caused by the constant looping back to check for a character. If you wish to have TSX-Plus return from the EMT 340 with the carry flag set if a character is not available, you must set bit 6 in the Job Status Word and also do one of the following things:

- Specify the /SINGLECHAR switch with the RUN or R command that starts the program.
- Issue a SET TERMINAL NOWAIT command.
- Use the INSTALL command to install the program with the NOWAIT attribute (the description of the INSTALL command is in the *TSX-Plus System Manager's Guide* .
- Use the "U" program controlled terminal option described in this chapter.

3.1.4 Non-blocking .TTOUTR

Normally when a program sends output to the terminal using a .WRITE, .PRINT, .TTYOUT, or .TTOUTR EMT, if the terminal output buffer is full, TSX-Plus suspends the program until space becomes available in the output buffer. If you wish to use the .TTOUTR EMT to send output to the terminal and have TSX-Plus return from the EMT with the carry flag set if the output buffer is full, you must do the following things:

- Use the .TTOUTR EMT rather than .TTYOUT.
- Set bit 6 in the Job Status Word.
- The instruction following the .TTOUTR EMT must not be [BCS .-2].
- The program must be run with no-wait mode by performing one of the following actions:
 - Specify the /SINGLECHAR qualifier with the RUN or R command that starts the program.
 - Issue a SET TERMINAL NOWAIT command.
 - Use the INSTALL command to install the program with the NOWAIT attribute.
 - Use the “U” program controlled terminal option described in this chapter.

TSX-Plus allows many other ways of modifying terminal input and output for special circumstances. These are provided to allow maximum versatility in the system while still maintaining the high efficiency needed in a multi-user environment. The programmer communicates the need for special terminal handling to the system through the use of special *program controlled terminal options*. These are described individually in the next section.

3.2 Program controlled terminal options

The following table lists the functions which may be used during program execution.

Function Character	Meaning
A	Set rubout filler character
B	Enable VT52 & VT100 escape-letter activation
C	Disable VT52 & VT100 escape-letter activation
D	Define new activation character
E	Turn on character echoing
F	Turn off character echoing
H	Disable subprocesses
I	Enable lower case input
J	Disable lower case input
K	Enable deferred character echo mode
L	Disable deferred character echo mode
M	Set transparency mode for output
N	Suspend command file input
O	Restart command file input
P	Reset activation character
Q	Set activation on field width
R	Turn on high-efficiency TTY mode
S	Turn on single-character activation mode
T	Turn off single-character activation mode
U	Enable no-wait TT input test
V	Set field width limit
W	Turn tape mode on
X	Turn tape mode off
Y	Disable echo of line-feed after carriage-return
Z	Enable echo of line-feed after carriage-return

These functions have a temporary effect and are automatically reset to their normal values when a program exits to the keyboard monitor. They are not reset if the program chains to another program until control is finally returned to the monitor. Some terminal options (notably high-efficiency and single-character modes) are incompatible with, and override, some other terminal options.

TSX-Plus provides two methods for a running program to dynamically alter some of the parameter settings relating to the user's timesharing line. The preferred method of selecting these functions is to use the TSX-Plus EMT for that purpose. This is readily available from MACRO programs and an appropriate MACRO subroutine should be linked into jobs written in other languages. The form of the EMT to select program controlled terminal options is:

```
EMT      375
```

with R0 pointing to the following argument block:

```
.BYTE    0,152
.WORD    function-code
.WORD    argument-value
```

where *function-code* is the character from the table above which selects the terminal option, and *argument-value* may be a third value used only with some of the functions. An advantage of the EMT method of selecting program controlled terminal options is that they may be used even when the terminal is in high-efficiency mode.

Example

```
.TITLE EMTMTH
.ENABL LC
; Demonstrate TSX-Plus program controlled terminal options using
; the EMT method.
.MCALL .GVAL,.PRINT,.EXIT,.TTYIN
.GLOBL PRTDEC
; Determine and display current lead-in char
; Default = 35, but don't count on it
START: .GVAL #AREA,#-4.      ;Determine current leadin character
      MOV    R0,R1          ;Save it
      .PRINT #LEADIS        ;"Current lead-in is"
      MOV    R1,R0          ;Retrieve lead-in char value
      CALL   PRTDEC         ;Display it
      .PRINT #LEADND
; Set rubout filler character
      MOV    #SETRUB,R0     ;Point to EMT arg block to
      EMT    375            ;Set rubout filler character
; Now demonstrate the current rubout filler character
; Back space is the default, which we changed to underline
      .PRINT #TRYIT        ;"Enter some and delete them"
      MOV    #BUFFER,R1     ;Point to input buffer
1$:    .TTYIN               ;Get next char into input buffer
      CMPB   R0,#12         ;End of input (CR/LF pair)?
      BEQ    2$             ;Yes, terminate input
      CMP    R1,#BUFEND     ;Buffer overflow?
      BLO    1$             ;Get more if not
2$:    .PRINT #THANKS
      .EXIT
AREA:   .BLKW 10            ;General EMT arg block
SETRUB: .BYTE 0,152         ;EMT arg block to
      .WORD  'A             ;Set rubout filler character
      .WORD  '_'            ; to underline
TRYIT:  .ASCII /Enter some characters at the prompt and then /
      .ASCII /erase them with/<15><12>/DELETE or CTRL-U./
      .ASCII / They should be replaced with underlines./<15><12>*/<200>
LEADIS: .ASCII /We don't care that the current lead-in char is /<200>
LEADND: .ASCIZ ./
THANKS: .ASCIZ <15><12>/STOP -- Thank you./
BUFFER: .BLKB 81.
BUFEND:
      .END    START
```


When it is not practical to incorporate the EMT method of selecting program controlled terminal options into a program, an alternate method using a *lead-in* character may be used. This is conveniently done by sending a sequence of characters to the terminal using the normal terminal output operations of the language. Examples are the FORTRAN TYPE, COBOL-Plus DISPLAY, BASIC PRINT, and Pascal WRITE statements. Program controlled terminal options are selected by having the running program send the lead-in character immediately followed by the function character and, for some functions, a third character defining the argument value for the function. TSX-Plus intercepts the lead-in character and the one or two following characters and sets the appropriate terminal option. It does not pass these intercepted characters through to the terminal. The default value for the lead-in character is the ASCII GS character (octal value 35; decimal value 29). However, the lead-in character may be redefined during system generation when the value conflicts with other uses of the system. For example, some graphics terminals use the GS character as either a command or parameter value. The lead-in character function may be disabled during program execution with the M and R functions described below. Programmers should not rely on the default value of the lead-in character, but may obtain the current value of the lead-in character from the .GVAL request with an offset of -4.

Note that when in high-efficiency mode (set with either the RUN/HIGH switch or the "R" program controlled terminal option) output character checking is disabled and the lead-in character method of selecting program controlled terminal options is disabled; in this case, the lead-in character, function-code character, and argument value character are passed through to the terminal. When in high-efficiency mode, the EMT method of selecting program controlled terminal options is still functional. See Chapter 4 for information on turning high-efficiency terminal mode off by means of an EMT.

Example

```

      PROGRAM LEADIN
C
C Demonstrate TSX-Plus program controlled terminal options using
C the "lead-in" character method.
C
      BYTE LEADIN(2)
      INTEGER ILEAD
      EQUIVALENCE (ILEAD,LEADIN(1))
C Determine and display current lead-in char
C Default = 29, but don't count on it
      ILEAD = ISPY(-4)      !.GVAL with offset = -4.
      TYPE 820,LEADIN(1)    !Display the current lead-in char value
C Set rubout filler character
      TYPE 800,LEADIN(1),'A','_'
C Now demonstrate the current rubout filler character
C Back space is the default, which we changed to underline
      TYPE 810              !Ask for something to be erased
      ACCEPT 830            !Wait for input before exiting
      STOP 'Thank you.'
      FORMAT(1H+,A1,$)
800  FORMAT(1H0,'Enter some characters at the prompt and then ',
1    'erase them with'/' DELETE or CTRL-U.',
1    ' They should be replaced with underlines.'/' *,$)
820  FORMAT(' The current value of the lead-in character is ',I3,'.')
830  FORMAT(40H
      )
      END

```

The following paragraphs explain the uses of each of the program controlled terminal option function-codes. Any of these options may be selected by either the EMT method or by the lead-in character method.

3.2.1 "A" function—Set rubout filler character

When a scope type terminal is being used, the normal response of TSX-Plus to a DELETE character is to echo *backspace space backspace* which replaces the last character typed with a space. TSX-Plus responds to a CTRL-U character in a similar fashion, echoing a series of backspaces and spaces. Some programs that display forms use underscores or periods to indicate the fields where the user may enter values. In this case it is desirable for TSX-Plus to echo *backspace character backspace* for DELETE and CTRL-U where *character* may be period or underscore as used in the form. The character to use as a rubout filler is specified by the argument-value with the EMT method or by the third character with the lead-in character method.

3.2.2 “B” & “C” functions—Set VT52, VT100 and VT200 escape-letter activation

VT52, VT100 and VT200 terminals are equipped with a set of special function keys marked with arrows and other symbols. When pressed, they transmit two or three character escape sequences. The “B” function tells TSX-Plus to consider these as activation sequences. The escape character and the letter are not echoed to the terminal, but are passed to the user program. The “C” function disables this processing and causes escape to be treated as a normal character (initial setting).

3.2.3 “D” function—Define new activation character

Under normal circumstances TSX-Plus only schedules a job for execution and passes it a line of input when an *activation* character such as carriage return is received. The “D” function provides the user with the ability to define a set of activation characters in addition to carriage return.

The new activation character is specified by the argument-value with the EMT method or by the third character with the lead-in character method. The maximum number of activation characters that a program may define is specified when the TSX-Plus system is generated.

Using this technique, *any* character may be defined as an activation character, including such characters as letters, DELETE, CTRL-U, and CTRL-C. When a user-defined activation character is received, it is not echoed but is placed in the user’s input buffer which is then passed to the running program.

By specifying CTRL-C as an activation character, a program may lock itself to a terminal in such a fashion that the user may not break out of the program in an uncontrolled manner.

If carriage return is specified as a user activation character, neither it nor a following line feed will be echoed to the terminal. TSX-Plus will also not add a line feed to the input passed to the program.

3.2.4 “E” and “F” functions—Control character echoing

The “E” and “F” functions are used to turn on and off character echoing. The “E” function turns it on, and the “F” function turns it off. An example of a possible use is to turn off echoing while a password is being entered.

3.2.5 “H” function—Disable subprocess use

The “H” function disables the subprocess window facility for the time-sharing line.

3.2.6 “I” and “J” functions—Control lower case input

The “I” function allows lower case characters to be passed to the running program. The “J” function causes TSX-Plus to translate lower case letters to upper case letters. The SET TT [NO]LC keyboard command also performs these functions.

3.2.7 “K” and “L” functions—Control character echoing

The “K” function causes TSX-Plus to enter *deferred* character echo mode. The “L” function causes TSX-Plus to enter *immediate* character echo mode. Any characters in the input buffer which have not been echoed when the “L” function is selected will be immediately echoed. See the description of the SET TT [NO]DEFER command for an explanation of deferred echo mode.

3.2.8 “M” function—Set transparency mode of output

If transparency mode is set, TSX-Plus will pass each transmitted character through to the program without performing any special checking or processing. Transparency mode allows the user's program to send any character to the terminal. Note that once transparency mode is set on, TSX-Plus will no longer recognize the lead-in character (octal 35, which means a program control function follows). The only way to turn off transparency mode is to exit to KMON.

3.2.9 “N” and “O” Functions—Control command file input

When a command file is being used to run programs (see the *TSX-Plus User's Reference Manual*), input which would normally come from the user's terminal is instead drawn from the command file. Occasionally, it is desirable to allow a program running from a command file to accept input from the user's terminal rather than the command file. The “N” function suspends input from the command file so that subsequent input operations will be diverted to the terminal. The “O” function redirects input to the command file. These functions are ignored by TSX-Plus if the program is not being run from a command file.

If the “N” option is used and a .GTLIN is issued, then command file input cannot be restored. This is normally done to require operator response — in this case, use the optional *type* argument to the .GTLIN request to force terminal input rather than disabling command file input.

3.2.10 “P” function—Reset activation character

The “P” function performs the complement operation to the “D” function. The “P” function is used to remove an activation character that was previously defined by the “D” function. The character to be removed from the activation character list is specified by the argument-value with the EMT method or by the third character with the lead-in character method.

Only activation characters that were previously defined by the “D” function may be removed by the “P” function.

3.2.11 “Q” function—Set activation on field width

The “Q” function allows the user to define the width of an input field so that activation will occur if the user types in as many characters as the field width, even if no activation character is entered. The field width is specified by the ASCII code value of the argument-value with the EMT method or of the third character with the lead-in character method. If an activation character is entered before the field is filled, the program will be activated as usual. Each time activation occurs the field width is reset and must be set again for the next field by reissuing the “Q” function. For example, the following sequence of characters could be sent to TSX-Plus to establish a field width of 43 characters: “<lead-in>Q+”. Note that the character “+” has the ASCII code of 053 (octal) which is 43 decimal.

3.2.12 “R” function—Turn on high-efficiency terminal mode

The “R” function causes TSX-Plus to place the line in *high efficiency* terminal mode. The effect is to disable most of the character testing overhead that is done by TSX-Plus as characters are transmitted and received by the line. Before entering high-efficiency mode the program must declare a user-defined activation character that will signal the end of an input record. Once a program has entered high-efficiency mode, characters sent to the terminal are processed with minimum system overhead. For example, tab characters are not expanded to spaces. Also, TSX-Plus does not check to see if the character being sent is the TSX-Plus terminal control *leadin* character. This means that the lead-in character method may not be used to control terminal options until the program exits or the EMT to turn off high efficiency mode is used (see Chapter 4). Characters received from the terminal are passed to the program with minimum processing: they are not echoed; and control characters such as DELETE, CTRL-U, CTRL-C, CTRL-W and carriage-return are all treated as ordinary characters and passed directly to the program. High-efficiency mode terminal I/O is designed to facilitate machine-to-machine communication; it is also useful for dealing with buffered terminals that transmit a page of information at a time.

3.2.13 “S” function—Turn on single-character activation mode

The “S” function causes TSX-Plus to allow a program to do single-character activation by setting bit 12 in the Job Status Word. Normally TSX-Plus stores characters received from the terminal and only activates the program and passes the characters to it when an activation character, such as carriage-return, is received. It does this even if bit 12 is set in the Job Status Word, which under RT-11 causes the program to be passed characters one-by-one as they are received from the terminal. The “S” function can be used to cause TSX-Plus to honor bit 12 in the Job Status Word. If JSW bit 12 is set and the program is in single-character activation mode, TSX-Plus passes characters one-by-one to the program as they are received and does not echo the characters to the terminal. The /SINGLECHAR switch for the R[UN] command and the SET TT SINGLE command can also be used to cause TSX-Plus to honor JSW bit 12. Since the high-efficiency mode implies certain terminal characteristics (such as buffered input and no echo), it is not possible to override these inherent modes by using other function codes.

3.2.14 “T” function—Turn off single-character activation mode

The “T” function is the complement of the “S” function. It turns off single-character activation mode.

3.2.15 “U” function—Enable non-wait TT I/O testing

The “U” function causes TSX-Plus to allow a program to do a .TTINR EMT that will return with the carry bit set if no terminal input is pending or a .TTOUTR EMT that will return with the carry bit set if the terminal output buffer is full. Normally TSX-Plus suspends the execution of a program if it attempts to obtain a terminal character by doing a .TTINR EMT and no input characters are available. Or if it does a .TTYOUT or .TTOUTR EMT and there is no free space in the terminal output buffer. It does this even if bit 6 of the Job Status Word is set, which under RT-11 would enable non-blocking .TTINRs. This is done to prevent programs from burning up CPU time by constantly looping back to see if terminal input is available. The “U” function causes TSX-Plus to honor bit 6 in the Job Status Word and allows a program to do a .TTINR to check for pending TT input or a .TTOUTR EMT to check for space in the output buffer without blocking if none is available. The SET TT NOWAIT command and the /SINGLECHAR switch for the R[UN] command also perform this function. Because the single character terminal option determines several terminal operating modes (such as no echo and transparent input), it is incompatible with other terminal functions which would conflict with the implied single-character operation. See the description of special terminal mode in the *RT-11 Programmer's Reference Manual*.

3.2.16 “V” function—Set field width limit

The “V” function is used to set a limit on the number of characters that can be entered in the next terminal input field. Once the “V” function is used to set a field limit, if the user types in more characters to the field than the specified limit, the excess characters are discarded and the bell is rung rather than echoing the characters. An activation character still must be entered to complete the input. The field width is specified by the ASCII code value of the argument-value with the EMT method or of the third character with the lead-in character method. The field size limit is automatically reset after each field is accepted and must be re-specified for each field to which a limit is to be applied. Note the difference between the “Q” and “V” functions. The “Q” function sets a field size which causes automatic activation when the field is filled; the “V” function sets a field size which causes characters to be discarded if they exceed the field size. Note that any field width limit is ignored for command file input.

3.2.17 “W” and “X” functions—Control tape mode

The “W” function turns on *tape* mode and the “X” function turns it off. Turning on *tape* mode causes the system to ignore line-feed characters received from the terminal or external device. The SET TT [NO]TAPE keyboard command may also be used to control tape mode.

3.2.18 “Y” and “Z” functions—Control line-feed echo

The “Y” function is used to disable the echoing of a line-feed character when a carriage-return is received. Normally, when TSX-Plus receives a carriage-return character, it echoes carriage-return and line-feed characters to the terminal and passes carriage-return and line-feed characters to the program. The “Y” function alters this behavior so that it only echoes carriage-return but still passes both carriage-return and line-feed to the program. This function can be used to advantage with programs that do cursor positioning and which do not want line-feed echoed because it might cause the screen display to scroll up a line. The “Z” function restores the line-feed echoing to its normal mode.

Chapter 4

TSX-Plus EMTs

TSX-Plus provides several system service calls (EMTs) in addition to those compatible with RT-11. In order to take advantage of the special features of TSX-Plus, programs written to run under both TSX-Plus and RT-11 should check to see if they are under TSX-Plus. This chapter describes the preferred method of checking and goes on to describe several of the special EMTs provided by TSX-Plus. EMTs which relate specifically to features described elsewhere in this manual are included in the appropriate chapters.

4.1 Obtaining TSX-Plus system values (.GVAL)

The .GVAL EMT that is normally used to obtain RT-11 system values can also be used to obtain TSX-Plus system values. Although a simulated RMON is normally mapped into each job so that it may directly access fixed offsets into RMON, the .GVAL function is the preferred method for obtaining system values. Under TSX-Plus, the simulated RMON need not be mapped into a job's virtual address space (see Chapter 2). The .GVAL EMT will still function correctly even if RMON is not mapped into the job. In addition to the positive offset values which are documented for use with RT-11, the following negative offset values may be used to obtain TSX-Plus system values:

Offset	Value
-2.	Job number
-4.	<i>Lead-in</i> character used for terminal control options
-6.	1 if job has SYSPRV privilege; 0 if job does not have SYSPRV
-8.	1 if PAR 7 mapped to I/O page; 0 otherwise
-10.	Project number job is logged on under
-12.	Programmer number job is logged on under
-14.	TSX-Plus incremental license number
-16.	Current job priority
-18.	Maximum allowed job priority
-20.	Number of blocks per job in SY:TSXUCL.TSX
-22.	Job number of primary process (0 if primary process, always 0 for detached jobs)
-24.	Name of system device (RAD50) (device on which RT-11 was booted when TSX-Plus was started; may not correspond to current SY assignment)
-26.	Maximum fixed-low-priority value
-28.	Minimum fixed-high-priority value
-30.	Job number of parent process (0 if primary process, same as offset -22. if subprocess)
-32.	System version number—Decimal value times 100.
-34.	Relative subprocess number (0 if primary process or detached job)

As with the standard .GVAL function, the system values are returned in R0.

Example

```
.TITLE TSGVAL
.ENABL LC
;Demonstrate usage of .GVAL with both positive (RT-11)
; and negative (TSX-Plus) offsets
.MCALL .GVAL,.PRINT,.EXIT
.GLOBL PRTDEC ;Subroutine to print a word in decimal
.GLOBL PRTR50 ;Subroutine to print a RAD50 word
SYSGEN = 372 ;RMON offset to sysgen options word
START: .GVAL #AREA,#SYSGEN ;Examine system options word
TST R0 ;See if we are running TSX-Plus
BPL 9$ ;Exit if not
.PRINT #LICENSES ;"License # is"
.GVAL #AREA,#-14. ;Obtain last 4 digits of license #
CALL PRTDEC ;And display it
.PRINT #SYSTEM ;"Started from"
.GVAL #AREA,#-24. ;Get system device
CALL PRTR50 ;And display it
.PRINT #JOBNUM ;"Job # is"
.GVAL #AREA,#-2 ;Get TSX-Plus job number
CALL PRTDEC ;Display the job number
.GVAL #AREA,#-22. ;See if this is the primary line
TST R0 ;0 if primary
BEQ 9$ ;Done if so
.PRINT #VIRT ;Else say subprocess
9$: .EXIT
AREA: .BLKW 2 ;2 word EMT arg area
.NLIST BEX
LICENSES: .ASCII /TSX-Plus license number /<200>
SYSTEM: .ASCII <15><12>/TSX-Plus started from /<200>
JOBNUM: .ASCII /:/<15><12>/TSX-Plus line number /<200>
VIRT: .ASCII / (This is a subprocess)/<200>
.END START
```

4.2 Determining if a job is running under TSX-Plus

In cooperation with Digital Equipment Corporation, a bit has been allocated in the RT-11 sysgen options word at fixed offset 372 into the RMON. The high order bit (bit 15; mask 100000) of this word will be set (1) if the current monitor is TSX-Plus version 5.0 or later. This bit will be clear if the monitor is any version of RT-11. Testing this bit is the preferred method of determining if a job is running under TSX-Plus. However, if a program is expected to also be used under older versions of TSX-Plus, then an alternative method is necessary. For older versions of TSX-Plus, first issue the .SERR request to trap invalid EMT requests and then issue the TSX-Plus EMT to determine the time-sharing line number. If the job is running under RT-11, this EMT will be invalid and the carry bit (indicating an error) will be set on return. If the job is running under TSX-Plus, then the EMT will return without error and the line number will be in R0.

Example

```
.TITLE TSXENV
.ENABL LC
; Demonstrate preferred method of determining whether job is
; running under TSX-Plus or RT-11
.MCALL .PRINT,.EXIT,.SERR,.HERR
JSW = 44 ;Job Status Word address
RMON = 54 ;Pointer to base of RMON
SYSGEN = 372 ;Index into RMON for sysgen features
START: .PRINT #UNDER ;"Running under"
; This is the preferred method, but will not work prior to
```



```

; TSX-Plus version 5.0
      MOV     RM0N,R1      ;Point to base of RM0N
      TST     SYSGEN(R1)   ;See if running under TSX-Plus
      BPL     RT11         ;Branch if running under RT-11
; This is the old method, but will work correctly with
; all versions of TSX-Plus
;      .SERR      ;Trap invalid EMT error
;      MOV     #TSXLN,R0   ;Point to EMT arg block to
;      EMT     375         ;Determine TSX-Plus line number
;      BCS     RT11         ;Branch if running under RT-11
;      .PRINT   #TSXPLS    ;"TSX-Plus"
;      .EXIT
RT11:  .HERR      ;Reset SERR trap
      .PRINT   #NOTPLS     ;"RT-11"
      .EXIT
TSXLN: .BYTE     0,110     ;EMT arg block to get line number
      .NLIST   BEX
UNDER: .ASCII    /Monitor is /<200>
TSXPLS: .ASCIZ   /TSX-Plus./
NOTPLS: .ASCIZ   /RT-11./
      .END      START

```

4.3 Determining number of free blocks in spool file (0,107)

The following EMT will return in R0 the number of free blocks in the spool file. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument area:

```
.BYTE    0,107
```

Example

```

      .TITLE    SPLFRE
      .ENABL    LC
;Demonstrate EMT to determine number of free spool blocks
      .MCALL    .PRINT,.EXIT
      .GLOBL    PRTDEC
START: .PRINT   #NUMFRE      ;Preface number message
      MOV     #SPLFRE,R0    ;Point to EMT arg block to
      EMT     375          ;Determine number of free spool blocks
                                ;Number is returned in R0
      CALL    PRTDEC        ;Display the number
      .PRINT   #BLOKS       ;End of message
      .EXIT
      .NLIST   BEX
SPLFRE: .BYTE    0,107      ;EMT arg to get # free spool blocks
NUMFRE: .ASCII    /The spool file has /<200>
BLOKS:  .ASCIZ   / free blocks./
      .EVEN
      .END      START

```

4.4 Determining number of blocks in use in spool file (1,107)

The following EMT will return in R0 the number of spool blocks currently in use. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument area:

```
.BYTE    1,107
```

4.5 Determining the TSX-Plus line number (0,110)

The following EMT will return in R0 the number of the line to which the job is attached. Physical lines are numbered consecutively starting at 1 in the same order as specified when TSX-Plus is generated. Detached job lines occur next and subprocesses are numbered last.

The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument area:

```
.BYTE    0,110
```

Example

```
.TITLE LMTT
.ENABL LC
; What TSX line number is this terminal attached to?
; And what type terminal does TSX-Plus think it is?
.MCALL .PRINT,.EXIT,.TTYOUT,.SERR,.HERR
.GLOBL PRTDEC          ;Subroutine to print a word in decimal
                        ;Are we under TSX-Plus?
START: .SERR           ;Stop error aborts
      MOV  #TSXLN,R0    ;Set up EMT request to
      EMT  375          ;Get TSX-Plus line number
      BCS  NOTTSX       ;If error, not under TSX-Plus
      MOV  R0,LINE      ;Save it
      .HERR            ;Enable error aborts
      .PRINT #LINMSG    ;Display line number message
      MOV  LINE,R0      ;Recall line number
      CALL PRTDEC       ;Display line number
      .PRINT #TRMSG     ;Display term type message
      MOV  #TTYTYPE,R0  ;Set up EMT request to
      EMT  375          ;Get terminal type from TSX-Plus
                        ;Returns into R0
      ASL  R0           ;Convert to word offset
      .PRINT TYPE(R0)   ;Print type from index into table
      .EXIT            ;All done
NOTTSX: .PRINT #TSXERR   ;Say we are not under TSX-Plus
      .EXIT
LINE:   .WORD  0        ;Storage for TSX line number
TERM:   .WORD  0        ;Storage for TSX term type code
TSXLN:  .BYTE  0,110    ;TSX line number EMT parameters
TTYTYPE: .BYTE  0,137   ;TSX term type EMT parameters
; Table of pointers to TSX term type names
      .EVEN
TYPE:   .WORD  UNK,VT52,VT100,HAZEL,ADM3A,LA36,LA120
      .WORD  DIABLO,QUME,VT200
      .NLIST BEX
LINMSG: .ASCII  /TSX-Plus line number: /<200>
TRMSG:  .ASCII  <15><12>/Terminal type: /<200>
TSXERR: .ASCIZ  /?LMTT-F-Not running under TSX-Plus/
UNK:    .ASCIZ  /Unknown/
VT52:   .ASCIZ  /VT52/
VT100:  .ASCIZ  /VT100/
HAZEL:  .ASCIZ  /Hazeltime/
ADM3A:  .ASCIZ  /ADM3A/
LA36:   .ASCIZ  /LA36/
LA120:  .ASCIZ  /LA120/
DIABLO: .ASCIZ  /Diablo/          ;Diablo and Qume are equivalent
QUME:   .ASCIZ  /Qume/            ;Diablo and Qume are equivalent
VT200:  .ASCIZ  /VT200/
      .EVEN
      .END  START
```

4.6 Determining subprocess job number (1,110)

The following EMT may be used to determine the line number on which subprocesses are executing. This is useful in conjunction with the EMT to initiate a subprocess.

The form of the EMT used to determine a subprocess job number is:

EMT 375

with R0 pointing to an argument block of the following form:

```
.BYTE        1,110
.WORD        relative-subprocess-number
```

The relative subprocess number is in the range 1 – MAXSEC, and is the same as the number used when switching to a subprocess from the keyboard with the <^W><n> sequence. The line number on which the relative subprocess is running is returned in R0. This EMT can return the following error codes:

Error Code	Meaning
0	Specified subprocess is not active.
1	Invalid relative subprocess number (> MAXSEC).
2	Attempt to issue this EMT fro a detached job. (Detached jobs are not allowed to create subprocesses.)

4.7 Set/Reset ODT activation mode (111)

The following EMT can be used to set TSX-Plus to activate on characters that are appropriate to ODT. In this mode TSX-Plus considers all characters to be activation characters except digits, “,”, “\$”, and “.”. The form of the EMT is:

EMT 375

with R0 pointing to the following argument area:

```
.BYTE        code,111
```

where *code*=1 to turn on ODT activation mode, and *code*=0 to reset to normal mode.

Example

```
.TITLE ACTODT
.ENABL LC
;Demonstrate EMT which sets ODT activation mode
.MCALL .PRINT,.EXIT
START: .PRINT #ODTTYP            ;Say we are entering ODT activation mode
MOV    #ACTODT,R0               ;Point to EMT arg block to
EMT    375                       ;Set ODT activation mode
1$:    CALL GETLIN               ;Get some terminal input
CMPB   BUFFER,#'Q               ;Back to regular mode?
BNE    1$                       ;No, get more lines
.PRINT #REGTYP                 ;Say we are going back to regular activation
CLRB   ACTODT                  ;Make arg block into RESET mode request
MOV    #ACTODT,R0               ;Point to EMT arg block to
```

```

2$:      EMT      375          ;Reset ODT activation mode
        CALL     GETLIN       ;Get more input
        CMPB     BUFFER,#'Q   ;Want to quit?
        BNE      2$          ;No, repeat
        .EXIT

GETLIN:  .PRINT   #PROMPT      ;Request some input
        MOV      #TTIBLK,R0    ;Point to EMT arg block to
        EMT      375          ;Accept a block of characters
        CLRB     BUFFER(R0)    ;Make input string ASCIZ
        .PRINT   #BUFFER      ;And echo same string back
        RETURN

ACTODT:  .BYTE    1,111        ;EMT arg block to SET/RESET ODT act'n mode
TTIBLK:  .BYTE    0,115        ;EMT arg block to get block input from term
        .WORD    BUFFER        ;Pointer to input buffer
        .WORD    79.           ;Number of input chars requested
        .MLIST   BEX

ODTTYP:  .ASCIZ   /Starting ODT activation mode./
REGTYP:  .ASCIZ   /Restoring regular activation mode./
PROMPT:  .ASCIZ   /?/<200>
        .EVEN

BUFFER:  .BLKB    79.          ;TTIBLK input buffer
        .BYTE    0            ;CLRB could go here on full buffer
        .END      START

```

4.8 Sending a block of characters to the terminal (114)

The following EMT can be used to efficiently send a block of characters to the terminal. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument block:

```

.BYTE    0,114
.WORD    buffer
.WORD    count

```

where *buffer* is the address of the buffer containing the characters to be sent and *count* is a count of the number of characters to be sent. This EMT is much more efficient to use than a series of .TTYOUT EMTs—it has the same efficiency as a .PRINT EMT but it uses a count of the number of characters to send rather than having the character string in ASCIZ form.

Example

```

        .TITLE   TTOBLK
        .ENABL   LC
;Demonstration of the use of the TSX-Plus EMT to send a block of
;characters to the terminal.
        .MCALL   .EXIT
START:  MOV      #TTOBLK,R0    ;Point to EMT arg block to
        EMT      375          ;Send a block of chars to the terminal
        .EXIT
TTOBLK: .BYTE    0,114        ;EMT arg block to send a block of chars
        .WORD    BUFFER      ;Pointer to character buffer
        .WORD    <BUFEND-BUFFER> ;Count of characters to be output
        .MLIST   BEX
BUFFER: .ASCIZ   /This EMT is used to send a block of characters /
        .ASCIZ   /to the terminal./<15><12>
        .ASCIZ   /It is similar to .PRINT, except that it uses /
        .ASCIZ   /a count of characters/<15><12>
        .ASCIZ   /rather than a special terminating character /
        .ASCIZ   /(<0> or <200>)./<15><12>
BUFEND:
        .END      START

```

4.9 Accepting a block of characters from the terminal (115)

The following EMT can be used to accept all characters from the terminal input buffer up to and including the last activation character entered. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument block:

```
.BYTE    0,115
.WORD    buffer
.WORD    size
```

where *buffer* is the address of the buffer where the characters are to be stored and *size* is the size of the buffer (number of bytes). This EMT causes a program to wait until an activation character is entered and then returns all characters received up to and including the last activation character. On return R0 contains a count of the number of characters received. If the specified buffer overflows, the carry flag is set on return. This EMT is substantially more efficient than doing a series of .TTYIN EMTs; it is particularly well suited for accepting input from page buffered terminals.

Example

```
.TITLE TTIBLK
.ENABL LC
;Demonstrates the use of TSI-Plus EMT to accept a block of characters
;from a terminal.
.MCALL .EXIT,.PRINT,.TTYIN
START: .PRINT #PROMPT      ;Request input
      MOV #TTIBLK,R0      ;Point to EMT arg block to
      EMT 375             ;Accept a block of chars from the terminal
      MOV R0,R1           ;Save input character count
;Char count includes activation char (and LF after CR)
      BCC 1$             ;Buffer overflow on input?
      .PRINT #OVFLOW      ;Yes, warn user
1$:    ADD #BUFFER,R1      ;Point past last char in buffer
      CLRB (R1)           ;Make the input ASCIZ
      .PRINT #BUFFER      ;Reproduce the input
      .EXIT
TTIBLK: .BYTE 0,115        ;EMT arg block to accept block from terminal
      .WORD BUFFER        ;Start of input buffer
      .WORD <BUFEND-BUFFER> ;Length of buffer in chars (May not exceed
                          ;input buffer size declared in TSGEN.)

.NLIST BEX
PROMPT: .ASCIZ /70 character input buffer ready./
OVFLOW: .ASCIZ /?TTIBLK-F-Buffer overflow/
BUFFER: .ASCII /XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/ ;35 chars
      .ASCII /XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/ ;35 chars
BUFEND: .ASCII /??/       ;TTIBLK will never write over these
      .END START
```

4.10 Checking for terminal input errors (0,116)

The following EMT can be used to determine if any terminal input errors have occurred. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument block:

```
.BYTE 0,116
```

On return from the EMT, the carry flag is set if an input error has occurred since the line logged on or since the last time a check was made for input errors. The two types of errors that are monitored by this EMT are hardware reported errors (parity, silo overflow, etc.) and characters lost due to TSX-Plus input buffer overflow.

Example

```
.TITLE CKTTIE
.ENABLE LC
;Check for terminal input errors
.MCALL .PRINT,.TTYIN,.EXIT
START: .PRINT #PROMPT      ;Ask to overflow buffer
      MOV #100.,R1        ;Set up counter for input loop
      MOV #SETTTO,R0       ;Point to EMT arg block to
      EMT 375              ;Set terminal time out for 0.5 secs
;Note that this is reset after every activation character!!!
;Start requesting characters. Input characters are stacked in the user
;input buffer until an activation character is seen (\eg carriage return).
;So, all we have to do to overflow is enter more than the input buffer
;size (defined in TSGEN either by DINSPC or with the BUFSIZ macro)
;and type in too many before activating.
;Use a time out so we do not have to hit return.
1$: .TTYIN                ;Get a character from the terminal
    CHPB RO,#37           ;Was it time-out activation char?
    BEQ TIMOUT            ;Yes, exit loop
    SOB R1,1$            ;Repeat for 100. characters
;For a system with input buffer size=100. in TSGEN, we should be
;able to overflow the buffer before we see an activation char
TIMOUT: MOV #CKTTIE,R0    ;Point to EMT arg block to
      EMT 375             ;Check for terminal input errors
      BCS HADERR          ;Say we had errors
      .PRINT #NOERR       ;Say we had no errors
      .EXIT
HADERR: .PRINT #YESERR     ;Error message
      CMP R1,#3           ;Did we fill the buffer?
;Note that last two chars of input buffer are reserved
;for activation chars. Any excess input is discarded.
      BLE TOOMNY          ;Yes, buffer overflow
      .PRINT #HDWERR      ;No, hardware error message
      .EXIT
TOOMNY: .PRINT #OVFERR     ;Buffer overflow message
      .EXIT
SETTTO: .BYTE 0,117        ;EMT arg block to set terminal time out
      .WORD 20.           ;to 10 seconds (20 half sec units)
      .WORD 37            ;Passed as activation char on time-out
CKTTIE: .BYTE 0,116        ;EMT arg block to check for input errors
      .WLIST BEX
YESERR: .ASCIZ <15><12>/There were errors during terminal input./<7>
OVFERR: .ASCIZ /(Probably input buffer overflow.)/
HDWERR: .ASCIZ /(Probably hardware error ... parity, stop bits, data bits)/
NOERR: .ASCIZ <15><12>/There were no terminal input errors./
PROMPT: .ASCIZ /Please enter more than 100 input characters and wait. . ./
      .END START
```

4.11 Determining input characters pending for a line (1,116)

This EMT can be used to determine the number of input characters pending for the current line. The number of input characters pending is returned in R0.

The form of the EMT is:

```
EMT 375
```

with R0 pointing to the following EMT argument block:

```
.BYTE 1,116
```

4.12 Set terminal read time-out value (117)

This EMT can be used to specify a time-out value that is to be applied to the next terminal input operation. This EMT allows you to specify the maximum time that will be allowed to pass between the time that you issue a command to get input from the terminal and the time that an activation character is received to terminate the input field. You also specify with this EMT a special activation character that is returned as the terminating character for the field if the input operation times out without receiving an activation character from the terminal. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument block:

```
.BYTE    0,117
.WORD    time-value
.WORD    activation-character
```

where *time-value* is the time-out value specified in 0.5 second units and *activation-character* is a single character value that is to be returned as the last character of the field if a time-out occurs. The time value specified with this EMT only applies to the next terminal input field. The time value is reset when the next field is received from the terminal or the time-out occurs. A new time-out value must be specified for each input field that is to be time controlled.

Example

```
.TITLE  DUNJUN
; Demonstrate use of terminal input time-out testing
.MCALL  .TTYOUT,.TTYIN,.EXIT,.PRINT
START:  .TTYOUT #'?'
1$:     MOV    #SETTO,R0      ;Point to EMT arg block to
      EMT     375            ;Set terminal input time-out
      .TTYIN                      ;Get a character from the terminal
      CMP     RO,#<15>        ;Skip over carriage returns
      BEQ     1$
      CMP     RO,#<12>        ;and line feeds
      BEQ     START          ;prompt for next char
      CMP     RO,#'Q'         ;Should we quit?
      BNE     1$             ;No, get next char
      .PRINT  #DONE          ;Quit or time-out
      .EXIT                                ;Bye
SETTO:  .BYTE   0,117         ;EMT arg block
      .WORD   6*60.*2        ;6.min * 60.sec/min * 2.half-sec-units/sec
      .WORD   'Q'           ;Activation character on time-out
      .NLIST  BEX
DONE:   .ASCIZ  /STOP - /
      .END    START
```

See also the example program CKTTIE in the section on checking for terminal input errors.

4.13 Turning high-efficiency terminal mode on and off (0/1,120)

TSX-Plus offers a *high-efficiency* mode of terminal operation that eliminates a substantial amount of system overhead for terminal character processing by reducing the amount of processing that is done on each character. When in high-efficiency mode, characters are sent directly to the terminal with minimum handling by TSX-Plus; operations such as expanding tabs to spaces and form-feeds to line-feeds are omitted as well as input processing such as echoing characters and recognizing control characters such as DELETE, CTRL-U and CTRL-C. The only characters treated specially on input are user-defined activation characters and the user-specified break character. At least one user specified activation character must be declared if high-efficiency mode is to be used. This form of terminal I/O is designed to facilitate high-speed machine-to-machine communication. It can also be used effectively to communicate with buffered mode terminals. The form of the EMT used to control high-efficiency mode is:

EMT 375

with R0 pointing to the following argument block:

```
.BYTE code,120
```

where *code* is 1 to turn high-efficiency mode on and 0 to turn it off.

Example

```
.TITLE HIEFF
.ENABL LC
;Demonstrate the use of TSX-Plus Hi-efficiency terminal mode
.MCALL .EXIT,.PRINT
START: .PRINT #DCLCC ;Make ^C an activation char
.PRINT #PROMPT ;Ask for input
MOV #HIEFF,R0 ;Point to EMT arg block to
EMT 375 ;Turn on hi-efficiency mode
MOV #TTIBLK,R0 ;Point to EMT arg block to
EMT 375 ;Accept a block of characters
;Actual character count returned in R0
;Do something useful with the input?
MOV #15,<BUFFER-1>(R0) ;Replace the activation char with
MOV #12,BUFFER(R0) ;Carriage return, line feed
INC R0 ;Count LF for output
MOV R0,<TTIBLK+4> ;Set up count for output
MOV #TTIBLK,R0 ;Point to EMT arg block to
EMT 375 ;Display a block of characters
CLRB HIEFF ;Get ready to turn hi-eff off
MOV #HIEFF,R0 ;Point to EMT arg block to
EMT 375 ;Turn off hi-efficiency mode
.EXIT
HIEFF: .BYTE 1,120 ;EMT arg block to turn hi-eff mode on (off)
TTIBLK: .BYTE 0,115 ;EMT arg block to accept a block of chars
.WORD BUFFER ;Pointer to input buffer
.WORD BUFSIZ ;Number of chars to input
TTIBLK: .BYTE 0,114 ;EMT arg block to display a block of chars
.WORD BUFFER ;Pointer to buffer for output
.WORD BUFSIZ ;Size of buffer to output
BUFFER: .BLKB 82. ;I/O buffer---Cannot exceed line's I/O
BUFSIZ = . - BUFFER ; buffer sizes declared in TSGEN
.WORD 0 ;Spacer in case of buffer overflow
.NLIST BEX
DCLCC: .ASCII <35><'D><3><200> ;Declare ^C as special activation char
PROMPT: .ASCII /Please enter 1 line of characters (^C ends)./<15><12>
.ASCIIZ /No special processing or echoing will be done./
.END START
```

4.14 Suspending terminal output (2,120)

The following EMT can be used to emulate the behavior of pressing a Control-O (^O). This EMT flushes the terminal output buffer and suppresses any further terminal output until the .RCTRL0 (reset CTRL-O) EMT is issued. The form of the EMT is:

EMT 375

with R0 pointing to an argument block of the form:

```
.BYTE 2,120
```

No errors are returned by this EMT. Terminal output suppression remains in effect for the job until the .RCTRL0 EMT is issued or until CTRL-O is received from the terminal.

4.15 Checking for activation characters (123)

The following EMT can be used to determine if any activation characters have been received by the line but not yet accepted by the program. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument block:

```
.BYTE    0,123
```

If there are pending activation characters, the carry flag is cleared on return from the EMT; if there are no pending activation characters, the carry flag is set on return from the EMT.

Example

```
.TITLE  CKACT
.ENABL  LC
;Demonstrate use of check for activation characters
LEADIN = 35                      ;TSX-Plus program controlled terminal
                                ;option lead-in character
.MCALL .PRINT,.EXIT,.GTLIN,.TWAIT,.TTYOUT
START: .PRINT #PROMPT            ;Request some characters
                                ;And disallow deferred echoing
;Do some processing. Simulated here by .TWAIT
      MOV    #80,R1              ;Line length counter
1$:   .TTYOUT #'                  ;Tick, tock
      DEC    R1                  ;End of line?
      BNE    2$                  ;No, go on
      .TTYOUT #<15>              ;New line
      .TTYOUT #<12>
      MOV    #80,R1              ;Reset line length counter
2$:   .TWAIT #AREA,#TIME         ;Wait 1 second here
; . . .
      MOV    #CKACT,R0           ;Point to EMT arg block to
      EMT    375                 ;Check for pending activation characters
      BCS    1$                  ;Continue if input not complete
      .GTLIN #BUFFER             ;Collect the pending input
; . . .
                                ;Do something with it
      CMP    BUFFER,EX           ;Exit command?
      BNE    1$                  ;No, continue processing
      .PRINT #BYE
      .EXIT
AREA:  .BLKW  10.                 ;EMT arg block
TIME:  .WORD  0,1.*60.            ;1.sec * 60.tics/sec
CKACT:  .BYTE  0,123              ;EMT arg block for activation char check
BUFFER: .BLKB  81.                ;Local input buffer
      .MLIST BEX
      .EVEN
EX:     .ASCII  /EX/
PROMPT: .ASCII  <LEADIN>/L/      ;Disallow deferred echoing
      .ASCIZ  /Please enter up to 80 characters, then RETURN:/
BYE:    .ASCIZ  /Thank you./
      .END    START
```

4.16 Obtaining site information (124)

The following EMT can be used to obtain the TSX-Plus site incremental license number, full license string, and site name. Because of the implementation, the sub-functions to get the incremental license number and site name are only available when the spooler is included during system generation. If the spooler has not been included, the EMT will immediately return with the carry flag set. However, the sub-function to obtain the full license string does **not** require inclusion of the spooler.

The form of the EMT to obtain the site incremental license number (last 5 digits of the full TSX-Plus license number) is:

EMT 375

with R0 pointing to an argument block of the form:

```
.BYTE 0,124
```

On success, the incremental license number is returned in R0.

In order to obtain the site name string, the argument block should be of the form:

```
.BYTE 1,124
.WORD buff-ptr
```

where *buff-ptr* is the address of a **word-aligned** buffer to hold the returned site name. The buffer should be at least 46 bytes long. The actual site name length is returned in R0.

In order to return the site license number as a string (even if spooling has not been generated into the system), the argument block should be of the form:

```
.BYTE 2,124
.WORD buff-ptr
```

where *buff-ptr* is the address of a **word-aligned** buffer to hold the returned license number string. This buffer should be at least 26 bytes long. The actual license number string length is returned in R0.

If the carry bit is set on return from any of these EMTs the system was generated without spooling or the buffer address was invalid.

4.17 Sending a message to another line (127)

The following EMT can be used to cause a message to display on another line's terminal. (This feature is not related to message communication channels, but is the same as the keyboard SEND command.) The form of the EMT is:

EMT 375

with R0 pointing to the following argument block:

```
.BYTE sub-function,127
.WORD line-number
.WORD message-address
```

where *line-number* is the number of the line to which the message is to be sent and *message-address* is the address of the start of the message text that must be in ASCIZ form. The message length must be less than 88 bytes. Note that if the target screen is only 80 characters wide, characters past column 80 will be overwritten. If *sub-function* bit 0 is set and the job issuing this EMT has OPER privilege, then the GAG setting of the destination terminal can be overridden. If *sub-function* bit 1 is clear, this EMT waits for a free message buffer if none is available. If bit 1 is set and no free message buffer is available, then the EMT returns immediately with error code 2. Use of this EMT requires SEND privilege. Note that information sent to a line with this EMT is not processed by the window manager for the target line. Thus, if the window is refreshed the message sent will disappear.

Error Code	Meaning
0	Job does not have SEND privilege.
1	Line has SET TT GAG and is executing a program.
2	No free message buffers are available.

Example

See the example program CKSTAT in the section on determining job status information.

4.18 Starting A Detached Job (0,132)

This EMT can be used to start the execution of a detached job. Use of this EMT requires DETACH privilege. The process which issues this EMT is known as the *parent* of the detached job it starts. The form of the EMT is:

EMT 375

with R0 pointing to the following argument block:

```
.BYTE    0,132
.WORD    name-address
```

where *name-address* is the address of an area containing the name of the command file to be started as a detached job. The command file name must be stored in ASCIZ form and may contain an extension. If a free detached job line is available, the specified command file is initiated as a detached job and the number of the detached job line is returned in R0. A detached job started by use of this EMT inherits the characteristics of the (sub)process that is executing the EMT. If there are no free detached job lines, the carry bit will be set on return.

Example

```
.TITLE  STRTDJ
.EMABL  LC
;Start a job on a TSX--Plus detached line
CR = 15
LF = 12
ERRBYT = 52
.MCALL  .ENTER,.WRITW,.CLOSE,.PRINT,.EXIT
START:  CLR    R1            ;Channel number
;No .FETCH is necessary under TSX--Plus, handlers are always resident.
1$: .ENTER  *AREA,R1,*FILR50,*1 ;Open a one block file on first free channel.
     BCC    2$            ;Branch on successful .ENTER
     TSTB   @*ERRBYT       ;Why didn't .ENTER work?
     BNE    NOROOM        ;Error = 1 :not enough room for file
     INC    R1            ;Try next higher channel
     CMP    R1,*17        ;Last channel? .CDFW not supported by
                         ;TSX--Plus, so legal channels are 0-15.
     BLE    1$            ;OK, retry on next channel
     .PRINT #NCA          ;Ran out of channels
     BR     DONE
2$: .WRITW  *AREA,R1,*COMNDS,*<<CMDEND-COMNDS+1>/2>,*0
     BCS    WRERR          ;Bad write?
     .CLOSE R1            ;Close the file
     MOV    *STRTDJ,R0     ;Point to EMT arguments to
     EMT    375            ;Start the detached job
     BCS    DTCHER        ;Bad start of detached job?
     BR     DONE
NOROOM: .PRINT #NER        ;Not enough room error
     BR     DONE
WRERR:  .PRINT #BADWRT     ;.WRITW error
     .CLOSE R1
     BR     DONE
DTCHER: .PRINT #BADDET     ;STRTDJ error
DONE:   .EXIT
AREA:   .BLKW   5           ;EMT Argument area
```

```

STRTDJ: .BYTE 0,132           ;EMT arguments to start a detached job
        .WORD FILNAM         ;Pointer to name of command file
        .MLIST BEX
FILR50: .RAD50 /SY CKSTATCOM/  ;RAD50 name of command file to be detached
FILNAM: .ASCIZ /SY:CKSTAT.COM/ ;ASCII name of command file to be detached
COMNDS: .ASCIZ /R CKSTAT/<15><12> ;Start a monitoring program
CMDEND:
NER:    .ASCIZ /?STRTDJ-F-Not enough room for command file./<7>
NCA:    .ASCIZ /?STRTDJ-F-No channels available for command file./<7>
BADWRT: .ASCIZ /?STRTDJ-F-Error writing command file./<7>
BADDET: .ASCIZ /?STRTDJ-F-Error starting detached job./<7>
        .END START

```

4.19 Checking the status of a detached job (1,132)

This EMT may be used to check the status of a detached job. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument block:

```

.BYTE 1,132
.WORD job-number

```

where *job-number* is the number of the detached job to be checked. If the detached job is still active the EMT returns with the carry-flag cleared. If the detached job has terminated and the detached job line is free, the EMT returns with the carry-flag set.

Example

```

        .TITLE CKABDJ
        .ENABL LC
; Check status of a detached job and abort it if running
        .MCALL .EXIT,.PRINT
START:  MOV  #STATDJ,R0      ;Point to EMT arg block to
        EMT  375            ;Check status of a detached job
        BCC  1$             ;If still on, kill it
        .PRINT #NOTON      ;Else, say it isn't active
        .EXIT
1$:     MOV  #ABRTDJ,R0      ;Point to EMT arg block to
        EMT  375            ;Abort detached job
        BCS  ABERR          ;Since we checked, should never err
        .PRINT #KILLED      ;Say we killed it
        .EXIT
ABERR:  .PRINT #ABERMS
        .EXIT
STATDJ: .BYTE 1,132         ;EMT arg value to check detached job status
        .WORD 9             ;Line number of detached job to be checked
ABRTDJ: .BYTE 2,132         ;EMT arg value to abort a detached job
        .WORD 9             ;Line number of detached job to be killed
        .MLIST BEX
ABERMS: .ASCIZ /?CKABDJ-F-Invalid detached job number/<7>
NOTON:  .ASCIZ /?CKABDJ-I-No detached job on line #9/<7>
KILLED: .ASCIZ /Detached job on line #9 killed./
        .END START

```

4.20 Killing a job (2,132)

This EMT may be used to kill any job if the appropriate privileges are present. If it is used to kill a detached job, then DETACH privilege is required; to kill primary and secondary lines, DETACH privilege is not required. The expected checking is always done for SAME, GROUP, and WORLD privileges. That is: a process may always kill its parent or child, may only kill a different line with the same PPN if it has SAME privilege, may only kill a process with the same project but different programmer number if it has GROUP privilege, and may only kill another job with a different project number if it has WORLD privilege.

This EMT is equivalent to the keyboard KILL command and checks for DETACH privilege when appropriate.

The form of this EMT is:

EMT 375

with R0 pointing to an argument block like:

```
.BYTE 2,132
.WORD job_number
```

This EMT may return the following error codes:

Error Code	Meaning
0	Invalid subfunction code
1	Invalid job number
2	You do not have privilege to kill that job

4.21 Establishing break sentinel control (0,133)

The following EMT can be used to declare a completion routine that will be triggered when the BREAK key is pressed. (Note that receipt of the break sentinel character while the terminal is already in terminal input state does not activate entry to the completion routine.) The form of the EMT is:

EMT 375

with R0 pointing to the following argument block:

```
.BYTE 0,133
.WORD brkchr
.WORD cplrtn
```

where *brkchr* is a user defined character that is to be declared the BREAK character and *cplrtn* is the address of the completion routine that is to be called when the break character is received from the terminal.

The specified completion routine will be called if the user presses either the key labeled "BREAK" (which transmits a long space) or types the character that is declared as the user-specified break character (*brkchr*). If no user-specified break character is wanted, specify the value zero (0) for *brkchr* in the argument block and only the real BREAK key will be activated. Note that on some systems the console terminal BREAK key causes entry to the hardware ODT module and for this reason cannot be used with this TSX-Plus function. Only one break routine may be specified at a time for each user. If a break routine was previously specified, it is cancelled when a new routine is declared. If an address of zero (0) is specified as the address of the completion routine (*cplrtn*), any previously specified break routine is cancelled and the break key

connection is cancelled. On return from the EMT, R0 contains the address of any completion routine previously connected to the break sentinel character. If none was previously connected, then R0 will contain zero.

The specified break completion routine request only remains in effect for one break character. If you wish to be notified about additional break characters, you must reestablish the break connection each time a break character is received. This can be done from within the break completion routine.

A break routine can be used to signal an asynchronous request for service to a running program. A good example of this use would be to trigger entry to an interactive debugging program.

The break sentinel character does not activate entry to the completion routine if the job is already interterminal input state (TI).

Example

```
.TITLE BRKSNT
;Demo use of break sentinel control
.MCALL .PRINT,.WAIT,.EXIT
.ENABL LC
START: MOV    #BRKSNT,R0      ;Point to argument area to
      EMT     375            ;Establish break sentinel control
      .PRINT  #MESSAG        ;Prompt for key
      .WAIT   #AREA,#TIME     ;Give the user 2 seconds to hit the break key
      TST     YES             ;Ever see a break?
      BNE     DONE            ;Yes, all done
      .PRINT  #NOBRK          ;No, never saw it
DONE:  .EXIT
CMPRTN: .PRINT #GOTBRK        ;Say we caught the break
      MOV     #1,YES          ;Remember it
      RETURN                ;And continue
      ;Completion routines are ALWAYS exited with
      ;RTS PC under TSX-Plus, NEVER via RTI
BRKSNT: .BYTE  0,133          ;EMT arg value block to break sentinel control
      .WORD   0               ;Declare only 'BREAK' key as break char
      .WORD   CMPRTN          ;Address of completion routine to be called
      ;when system notices break
YES:    .WORD  0               ;Flag for break seen
AREA:   .BLKW  2               ;2 word arg area for .WAIT
TIME:   .WORD  0               ;high word of time
      .WORD  2.*60.            ;2 sec * 60.tics/sec
      .MLIST  BEX
MESSAG: .ASCIZ  /You have 2 seconds to hit the break key./
GOTBRK: .ASCIZ  <15><12>/Break key pressed./
NOBRK:  .ASCIZ  /Never saw the break key./
      .END    START
```

4.22 Terminal input completion routine (1,133)

The following EMT can be used to specify a completion routine which will be triggered when the next character is entered at the terminal. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument block:

```
.BYTE  1,133
.WORD  completion-routine
```

where *completion-routine* is the address of the completion routine to be entered when the next character is typed.

On entry to the completion routine, the received character is in R0. The EMT to connect the completion routine must be rescheduled before returning if it is desired to continue accepting characters via the completion routine. The program should be running with single character activation and bit 12 set in the job status word.

Example

```
.TITLE TTICPL
; Demonstrate terminal character input completion routine.
.ENABL LC
.DSABL GBL
.GLOBL PRTDEC
.MCALL .TTYOUT,.SPND,.PRINT
JSW      = 44                ;Job Status Word address
TTSPC$   = 10000             ;TT special mode (single char input)
TT$LC    = 40000             ;Enable lower case bit
SPACE    = 40                ;ASCII Space
START:   BIS      #TT$LC!TTSPC$,@#JSW ;Set single character mode
         MOV      #TTICPL,R0        ;Point to EMT arg block to
         EMT      375               ;Schedule TT input char compl routine
         .SPND                    ;Suspend main-line (forever in example)
CPLRTN:  MOV      R0,-(SP)          ;Save input char carried in R0
         .TTYOUT                    ;Echo the char
         .TTYOUT #SPACE
         MOV      (SP)+,R0          ;Retrieve char
         CALL     PRTDEC            ;Display its ASCII value
         .PRINT  #CRLF
         MOV      #TTICPL,R0        ;Point to EMT arg block to
         EMT      375               ;Schedule TT input char compl routine
         RETURN
TTICPL:  .BYTE    1,133             ;EMT arg block for TT input compl routine
         .WORD    CPLRTN           ;Completion routine to run
CRLF:    .ASCIZ   //
         .END     START
```

4.23 Mount a file structure (134)

This EMT is used to tell TSX-Plus that a file structure is being mounted and that TSX-Plus should begin caching the file directory for the device. The effect of this EMT is the same as doing a system MOUNT keyboard command to enable caching. It cannot be used to mount a logical subset disk. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument block:

```
.BYTE    0,134
.WORD    device-spec-address
.WORD    0
```

where *device-spec-address* is the address of a word containing the RAD50 form of the name of the device on which the file structure is being mounted. If there is no room left in the table of mounted devices, the carry bit is set on return and the error code returned is 1.

Example

```

        .TITLE MOUNT
        .ENABL LC
; Demonstrate TSX--Plus EMT to MOUNT (do directory caching on) a device
        .GLOBL IRAD50          ;SYSLIB RAD50 conversion subroutine
BS      = 10                   ;ASCII Backspace
        .MCALL .PRINT,.GTLIN,.EXIT
START:  .GTLIN #BUFFER,#PROMPT ;Ask for name of device
        MOV     #R5OBLK,R5     ;Point to arg block for next call
        CALL    IRAD50         ;Convert ASCII device name to RAD50
        MOV     #MOUNT,R0      ;Point to EMT arg block to
        EMT     375            ;Mount a file structure (directory caching)
        BCC     START          ;Ask for more if OK
        .PRINT  #NOGOOD        ;Say it was not good
        .EXIT
        .MLIST BEX
MOUNT:  .BYTE   0,134           ;EMT arg block to mount a file structure
        .WORD   DEVNAM          ;Pointer to RAD50 name of device
        .WORD   0               ;Required 0 argument
R5OBLK: .WORD   3               ;Number of args for IRAD50 call
        .WORD   THREE           ;Pointer to number of chars to convert
        .WORD   BUFFER          ;Pointer to chars to convert
        .WORD   DEVNAM          ;Pointer to RAD50 name of device
THREE:  .WORD   3               ;Number of chars to convert
DEVNAM: .WORD   0               ;RAD50 representation of device name
BUFFER: .BLKB   80.            ;GTLIN input buffer
PROMPT: .ASCII  /Name of device to be mounted:  :/<BS><BS><BS><BS><200>
NOGOOD: .ASCIZ  /Attempt to MOUNT too many devices./<7>
        .END     START

```

4.24 Dismount a file structure (0,135)

This EMT can be used to tell TSX-Plus to stop doing directory caching on a particular drive. The effect of this EMT is the same as a DISMOUNT keyboard command to disable caching. The form of the EMT is:

```
EMT      375
```

with R0 pointing to an argument block of the following form:

```

        .BYTE   0,135
        .WORD   device-spec-address
        .WORD   0

```

where *device-spec-address* is the address of a word containing the RAD50 name of the device to be dismounted.

Example

```

        .TITLE DISMNT
        .ENABL LC
; Demonstrate TSX--Plus EMT to DISMOUNT (stop caching on) a device
        .GLOBL IRAD50          ;SYSLIB RAD50 conversion subroutine
BS      = 10                   ;ASCII Backspace
        .MCALL .GTLIN
START:  .GTLIN #BUFFER,#PROMPT ;Ask for name of device
        MOV     #R5OBLK,R5     ;Point to arg block for next call
        CALL    IRAD50         ;Convert ASCII device name to RAD50
        MOV     #DISMNT,R0     ;Point to EMT arg block to
        EMT     375            ;dismount a file structure (stop caching)
        BR      START          ;Repeat (no errors returned)
        .MLIST BEX
DISMNT: .BYTE   0,135           ;EMT arg block to dismount a file structure

```



```

        .WORD  DEVNAM      ;Pointer to RAD50 name of device
        .WORD  0           ;Required 0 argument
R5OBLK: .WORD  3           ;Number of args for IRAD50 call
        .WORD  THREE      ;Pointer to number of chars to convert
        .WORD  BUFFER     ;Pointer to chars to convert
        .WORD  DEVNAM     ;Pointer to RAD50 name of device
THREE:  .WORD  3           ;Number of chars to convert
DEVNAM: .WORD  0           ;RAD50 representation of device name
BUFFER: .BLKB  80.        ;GTLIN input buffer
PROMPT: .ASCII /Name of device to be dismounted: :/<BS><BS><BS><BS><200>
        .END    START

```

4.25 Dismounting all file structures and logical disks (2,135)

This EMT can be used to dismount all file structures for the job from directory and generalized data cache. This dismounts all logical disks from cache as well as cached physical disks. Note that this simply removes all logical disks from the cache tables; the logical disks are still accessible. Also note that issuing the SHOW SUBSETS (also SHOW LD) command will cause the logical disks to be cached again. The form of the EMT is:

```
EMT      375
```

with R0 pointing to an argument block of the following form:

```

        .BYTE  2,135
        .WORD  0

```

Example

```

        .TITLE  DMTFLS
        .ENABL  LC
;
; Demonstrate EMT to dismount (stop caching on) all files structures.
;
        .MCALL  .PRINT,.EXIT
        .DSABL  GBL

ERRBYT = 52                ;EMT error byte address

START:  MOV     #DMTFLS,R0   ;Point to EMT arg block to
        EMT     375         ;Dismount all file structures
        BCC     9$          ;Branch if OK
        .PRINT  #EMTERR     ;"Unable to dismount all files structures"
9$:     .EXIT

DMTFLS: .BYTE  2,135        ;EMT arg block to dismount file structures
        .WORD  0

        .NLIST  BEX
EMTERR: .ASCIZ  /?DMTFLS-F-Unable to dismount all files structures/
        .EVEN
        .END    START

```

4.26 Dismounting Logical Disks (3,135)

The following EMT can be used to dismount a logical disk. The form of the EMT is:

EMT 375

with R0 pointing to the following argument block:

```
.BYTE 3,135
.BYTE ld-unit,0
```

where *ld-unit* is the logical disk unit number which must be in the range 0 to 7.

The following error codes can be returned by this EMT:

Error Code	Meaning
0	Specified LD unit is not associated with a file
1	Invalid LD unit number (must be in the range 0 to 7)
3	Some channel is opened to a file on the logical disk

Example

```
.TITLE DMTILD
;
;Demonstration of the use of the TSX-Plus EMT to dismount a logical disk.
;
.MCALL .PRINT,.EXIT,.CLOSE
.GLOBAL PRTDEC
.DSABL GBL

ERRBYT = 52 ;EMT error byte address

START: MOV #DMTLD5,R0 ;Point to EMT arg. block to
      EMT 375 ;Dismount LD5:
      BCS 1$ ;Branch if not OK
      MOV #DMTLD6,R0 ;Point to EMT arg. block to
      EMT 375 ;Dismount LD6:
      BCC 2$ ;Branch if OK
1$: MOVB @#ERRBYT,-(SP) ;Fetch EMT error code
   .PRINT #ERRIS ;"EMT error is:"
   MOV (SP)+,R0 ;Retrieve error code
   CALL PRTDEC ;Display it
2$: .EXIT
   .NLIST BEI
DMTLD5: .BYTE 3,135 ;EMT arg. block to dismount
        .BYTE 5,0 ;LD5:
DMTLD6: .BYTE 3,135 ;EMT arg. block to dismount
        .BYTE 6,0 ;LD6:
ERRIS: .ASCII /?DMTILD-F-EMT error is: /<200>
        .EVEN
        .END START
```

4.27 Determining Status of Logical Disks (4,135)

The following EMT can be used to provide the status of a LD unit. The form of this EMT is:

EMT 375

with R0 pointing to the following argument block:

```
.BYTE 4,135
.BYTE ld-unit,0
.WORD buffer-address
```

where *ld-unit* is the LD unit number in the range 0 to 7, and *buffer-address* is the address of a five-word buffer which will receive information about the logical disk.

If the logical disk is not associated with a file then all zeros will be stored into the buffer. If the LD is associated with a file, the first four words of the buffer will receive the RAD50 device name, file name (2 words), and extension. The fifth word is a flag word. If the LD is mounted for read-only access, bit 0 of the flag word is set. If the LD is not currently accessible, bit 1 is set.

The following error code can be returned by this EMT:

Error Code	Meaning
1	Invalid LD unit number (must be in the range 0 to 7)

Example

```
.TITLE LDSTAT

.MCALL .PRINT,.EXIT,.LOOKUP,.SPFUN,.CLOSE,.SERR,.HERR
.GLOBAL PRTOCT,PRTR50,PRTDEC
.DSABL GBL

START: CLR LDUNIT ;Start with LDO
1$: CMP LDUNIT,#7 ;Have we print info on all LD's
    BGT 9$ ;Branch if not
    .PRINT #LDPRT ;Specify LD number
    MOV LDUNIT,R0
    CALL PRTDEC
    .PRINT #COLEQU
    MOV #LDSTAT,R0 ;Set up to...
    EMT 375 ;Get status for this LD
    BCC 2$ ;Branch if O.K.
    .PRINT #OOPS ;Else "LD Status EMT Error"
    BR 9$ ;And give up
2$: CLR SIZE ;Assume not mounted
    MOV #BUFF,R1 ;Get buffer address
    MOV (R1)+,R0 ;Get Device name
    BEQ 91$ ;BR if not mounted
    INC SIZE ;Remember to get size
21$: CALL PRTR50 ;Print device name
    .PRINT #COLON ;":"
    MOV (R1)+,R0 ;Get first half of file name
    CALL PRTR50 ;Print it
    MOV (R1)+,R0 ;Get last half of file name
    CALL PRTR50 ;Print it
    .PRINT #DOT ;"."
    MOV (R1)+,R0 ;Get extension
    CALL PRTR50 ;Print it

.SERR ;Trap Errors
MOVB LDUNIT,R0 ;Get Device and unit number
ADD #~RLDO,R0
MOV R0,DEV
.LOOKUP #AREA,#0,#DEVSPC ;Open a channel to it
BCS 3$ ;Branch on error
.SPFUN #AREA,#0,#373,#SIZE,#1,#1 ;Get the size of LD
BCS 3$ ;Branch on error
.PRINT #LSQBR ;"["
MOV SIZE,R0 ;Get the size
CALL PRTDEC ;and print it
```

```

        .PRINT #RSQBR      ;".]"
3$:      .HERR              ;Report errors
        .CLOSE #0          ;Close the channel
4$:      .PRINT #STAT      ;" STATUS= "
        MOV (R1)+,R0       ;Get the status
        CALL PRTOCT        ;and print it
        .PRINT #CRLF       ;Finish the line
        BR 92$
91$:     .PRINT #NOTMNT     ;"Not mounted"
92$:     INCB LDUNIT        ;Increment LD unit number
        BR 1$              ;Print info. on next LD
9$:      .EXIT

LDSTAT: .BYTE 4,135        ;Argument block for EMT get LD status
LDUNIT: .BYTE 0,0          ;LD unit number
        .WORD BUFF

BUFF:    .BLKW 5            ;Area where status info. is put
AREA:    .BLKW 10           ;Area .LOOKUP EMT block
DEV:
DEVSPC:  .RAD50 /LD0/
        .WORD 0,0,0
SIZE:    .WORD 0
COLON:   .ASCII /:/<200>
DOT:     .ASCII /./<200>
STAT:    .ASCII / STATUS= /<200>
LDPRT:   .ASCII /LD/<200>
COLEQU:  .ASCII /: = /<200>
CRLF:    .ASCIZ //
LSQBR:   .ASCII /[ /<200>
RSQBR:   .ASCII /./<200>
OOPS:    .ASCIZ /LD Status EMT Error/
NOTMNT:  .ASCIZ /Not mounted/
        .EVEN
        .END START

```

4.28 Dismounting all logical disks (5,135)

The following EMT allows dismounting of all logical disks with one EMT. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument block:

```

        .BYTE 5,135
        .WORD 0

```

The following error code can be returned by this EMT:

Error Code	Meaning
3	Some channel is opened to a file on the logical disk.

This EMT is similar to the SET LD EMPTY command. All LD's are completely dismounted, and become inaccessible until remounted. This is unlike the EMT to dismount all files structures which simply removes file structures from the caching tables.

Example

```

        .TITLE  DMTALD
        .EWABL  LC
;
; Demonstrate EMT to dismount all logical disks.
;
        .MCALL  .PRINT, .EXIT
        .GLOBL  PRTDEC
        .DSABL  GBL

ERRBYT = 52                ;EMT error byte address

START:  MOV     #DMTALD,R0    ;Point to EMT arg block to
        EMT     375          ;Dismount all logical disks
        BCC     9$           ;Branch if OK
        MOV     @#ERRBYT,-(SP) ;Fetch EMT error code
        .PRINT  #EMTERR      ;"Unable to dismount all logical disks. Error"
        MOV     (SP)+,R0     ;Retrieve error code
        CALL    PRTDEC       ;And print it
9$:     .EXIT

DMTALD: .BYTE    5,135        ;EMT arg block to dismount all logical disks
        .WORD    0

        .MLIST  BEX
EMTERR: .ASCII   /?DMTALD-F-Unable to dismount all logical disks. Error: /<200>
        .EVEN
        .END     START

```

4.29 Performance monitoring (136)

Performance monitoring EMTs are described in Chapter 11.

4.30 Determining the terminal type (137)

The following EMT will return in R0 a value that indicates what type of time-sharing terminal is being used with the line. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument block:

```
.BYTE    0,137
```

The terminal type is specified either when the TSX-Plus system is generated or by use of the SET TT command (e.g., SET TT VT100). The terminal type codes which are currently defined are listed below. The types Diablo and Qume are functionally equivalent.

Terminal-type	Code
<i>Unknown</i>	0
VT52	1
VT100	2
Hazeltine	3
ADM3A	4
LA36	5
LA120	6
Diablo & Qume	7
VT200	9

A type code of zero (0) is returned if the terminal type is unknown.

Example

See the example program LNTT in section 4.5 on determining the TSX-Plus line number.

4.31 Real-time requests (140)

Real-time EMTs are described in Chapter 8.

4.32 Controlling the size of a job (141)

Under RT-11, the .SETTOP EMT is used to set the top address of a job. The TSX-Plus .SETTOP EMT does not actually alter the memory space allocated to a job but simply checks to see if the requested top of memory is within the region actually allocated to the job and if not returns the address of the top of the allocated job region. The TSX-Plus .SETTOP EMT was implemented this way because many programs written for RT-11 routinely request all of memory when they start regardless of how much space they actually need.

The memory space actually allocated for a job can be controlled by use of the MEMORY keyboard command or by use of the EMT described below. The memory size specified by the most recently executed MEMORY keyboard command is considered to be the *normal* size of the job. The EMT described here can be used to alter the memory space allocated to a job but the job size reverts to the normal size when the job exits or chains to another program.

The form of the EMT used to change a job's size is:

```
EMT      375
```

with R0 pointing to the following argument area:

```
.BYTE    0,141
.WORD    top-address
```

where *top-address* is the requested top address for the job. If this address is larger than the allowed size of a job, the job will be expanded to the largest possible size. On return from the EMT, R0 contains the address of the highest available word in the program space.

A program is not allowed to change its size if it was started by use of the RUN/DEBUG command or the system was generated without allowing program swapping. In either of these cases the EMT operates exactly like a .SETTOP request (i.e., the requested program top address will not be allowed to exceed the normal program size).

See also the description of the SETSIZ program in Appendix A for information about how the default memory allocation for a program can be built into the SAV file for the program.

Example

See the example program CKSTAT in section 4.37 on determining job status information.

4.33 Determining project and programmer number (142)

This EMT can be used to obtain the project and programmer number under which a program is currently running. It is redundant with the information provided by .GVAL offsets -10. (project number) and -12. (programmer number). It is equivalent to the job status EMT (see section 4.37) sub-function 6 when the specified job number is the same as the one which issues the request. The form of the EMT is:

EMT 375

with R0 pointing to the following argument block:

```
.BYTE    0,142
.WORD    buf-address
```

where *buf-address* is the address of the first word of a 2-word buffer into which the current project and programmer number are returned. The project number is returned in the first word and the programmer number is stored in the second word.

4.34 Shared run-time requests (143)

Most EMTs related to shared run-time regions are described in Chapter 9.

4.35 Enabling separate I- and D-space (4,143)

The following EMTs to enable and disable separate I- and D-space do not by themselves increase a job's addressing capability, but may be used in conjunction with remapping a portion of the job's virtual address space to shared run-time regions or to PLAS regions. They are only supported on processors whose memory management hardware supports separate I- and D-space mapping and will return an error if not supported by the hardware. For example, the 11/23, 11/24, 11/34 and Pro-350 processors do not support separate I- and D-space, whereas the 11/73, 11/83, 11/93, 11/44, 11/84 and Pro-380 processors do support separate I- and D-space.

When a program is loaded, user memory mapping for the job is initialized from virtual address 000000 through the program's memory allocation limit. This is the static region, mapped solely through I-space (by default), and both instruction fetches and data references are made to the static region through I-space. After execution of the EMT to enable separate I- and D-space, then the same static region is doubly mapped both through I-space and through D-space. Thus, although mapped separately, both address spaces refer to the same physical memory locations and the program behaves as though no change had been made to its mapping. However, at this point, portions of the I-space or the D-space may be separately remapped to different physical memory locations, thus doubling the total amount of physical memory which may be directly addressed by the program.

The form of the EMT to enable separate I- and D-space is:

EMT 375

with R0 pointing to an argument block of the form:

```
.BYTE    4,143
```

After successful execution of this request, the job's instruction and data space are mapped separately and may be manipulated (more or less) independently of each other. This separate mapping remains in effect until the program exits (or aborts) or until the following EMT is issued to disable separate I- and D-space mapping. After execution of this EMT, all memory is once again mapped through I-space. The form of the EMT to disable separate I- and D-space is:

EMT 375

with R0 pointing to an argument block of the form:

.BYTE 5,143

A special case occurs when an overlayed program also attempts to enable separate I- and D-space. Since all I/O is normally directed to D-space, when a program overlay is read from disk, it could be read into the wrong memory, depending on the current job mapping. This would ordinarily preclude the use of overlayed programs with separate I- and D-space since program overlays need to be mapped into I-space. For overlayed programs, use function 7,143 rather than function 4,143 to enable separate I- and D-space. This service sets a flag that allows overlays to be read into I-space. To be specific, when separate I- and D-space is enabled in this mode, then all .READ operations on channel 17 (octal) are directed to I-space. I/O operations on any other channel, writes to channel 17, or .SPFUN operations on channel 17 are not affected and are still directed to D-space.

The form of the EMT which enables separate I- and D-space, except that reads on channel 17 are directed to I-space, is:

EMT 375

with R0 pointing to an argument block of the form:

.BYTE 7,143

These EMTs report the following error code:

<i>Error Code</i>	<i>Meaning</i>
6	The memory management hardware does not support separate I- and D-space.

4.36 User PAR control (10,143)

This is an unsupported feature!

In cases where a user program directly manipulates memory management registers, a system service is provided which allows the last memory mapping set by the user to be re-established after a context switch operation. User manipulation of memory management registers is highly discouraged and we explicitly disclaim responsibility for any and all damage to the system which may result from such actions.

One possible rationale for using such techniques is the need to update a large bank of shared or dual-ported memory (such as a graphics board) in the fastest possible way. Jobs mapped to the I/O page have always been able to do this, but previously job context switching had to be disabled so that the system did not re-establish its own idea of the appropriate contents of the PAR and PDR registers on returning to execution of the job which managed its own PAR registers. Now, if this request has been executed, then on switching context away from the critical job, the current contents of the hardware user memory management registers is saved in the job context block and restored from it on switching back to it.

The form of the EMT to allow user memory management control is:

EMT 375

with R0 pointing to an argument block of the form:

.BYTE 10,143

The pre-requisites to issue this request are that the job be locked in memory (either in place or in low memory) and that the job be mapped to the I/O page. After this request has been issued, subsequently unlocking the job from memory or using a system service to unmap the I/O page will disable user PAR control. Any request which causes the system to re-calculate memory mapping for the job will invalidate any current user changes to the mapping registers. Examples of requests which will cause such side effects are: re-sizing the job (EMT 375, function 0,141), and unmapping a PLAS window.

The errors returned by this EMT are:

Error Code	Meaning
0	Job is not locked in memory
1	Job is not mapped to the I/O page

4.37 Determining job status information (144)

The information about various jobs on the system which is displayed by the SYSTAT command may also be obtained by application programs. An EMT is provided with several subfunctions to obtain the desired job status information. This EMT may obtain information about any job on the system, not only itself. The form of the EMT is:

EMT 375

with R0 pointing to the following argument block:

```
.BYTE 0,144
.BYTE line-#,sub-function
.WORD buf-address
```

where *line-#* is the number of the time-sharing line about which information is to be returned. Line numbers are in the range 1 up to the highest valid line number for the system. *Sub-function* is a function code which indicates the type of information to be returned by the EMT (see below). *Buf-address* is the address of the first word of a 2-word buffer area into which the returned value is stored. Note: some of the functions only return a single word value in which case the value is returned into the first word of the buffer area.

If an error occurs during the execution of the EMT, the carry-flag is set on return and the following error codes indicate the type of error:

Error Code	Meaning
0	Indicated line number is not currently logged on.
1	Invalid sub-function code.
2	Invalid line number (0, or higher than largest valid line number).

Each of the sub-functions is described below:

Subfunction # 0 Check status of line. The value returned contains bit flags that indicate the status of the job. The following bit flags are defined:

Bit Flags	Meaning
000001	This is a subprocess.
000002	This is a detached job line.
000100	Job has locked itself in memory.
000200	Job has SYSPRV privilege.

Subfunction # 1 Get job's execution state. This subfunction returns a code that indicates a job's current execution state. The following code values are defined:

Code	Meaning
1	Non-interactive high priority run state.
2	Normal priority run state.
3	Fixed-low-priority run state.
4	Waiting on input from the terminal.
5	Waiting for output to be written to terminal.
6	Doing a timed wait.
7	Suspended because .SPND EMT done.
8	Waiting for access to a shared file.
9	Waiting for a inter-job message.
10	Waiting for access to USR (file management) module.
11	Waiting for non-terminal I/O to finish.
12	Waiting for access to spool file.
13	Interactive high priority run state.
14	Fixed-high-priority run state.
15	Waiting for memory expansion.

Subfunction # 2 Determine amount of memory used by job. This function returns the number of 256-word blocks of memory that are currently being used by the job, including PLAS regions.

Subfunction # 3 Determine connect time for job. This function returns the number of minutes that a job has been logged onto the system.

Subfunction # 4 Determine position of job in memory. This function returns the 256-word block number of the start of the memory area allocated to the job.

Subfunction # 5 Get name of program being run by job. This function returns a 2-word value. The two words contain the RAD50 value for the name of the program currently being run by the job.

Subfunction # 6 Get project and programmer number for job. This function returns a two-word value. The first word contains the project number that the job is logged on under; the second word contains the programmer number.

Subfunction # 7 Get CPU time used by job. This function returns a two-word value that contains the number of clock ticks of CPU time used by the job. The first word contains the high-order 16-bits of the value, the second word contains the low-order 16-bits.

Subfunction # 8 Get current job execution priority This function returns one word that contains the current job execution priority level (0-127).

Subfunction # 9 Get job name for the specified line number. The return buffer must be at least 12 bytes long.

Subfunction # 10 Get job numbers of primary and parent processes This function returns the job number of the primary process in the first word of the two word buffer, and the job number of the parent process in the second word. The results of the EMT are explained in the following table.

Type of job specified by <i>line-num</i>	Primary (word 1)	Parent (word 2)
Primary	self	0
Virtual	primary	primary
Detached (command)	self	command issuer
Detached (start-up)	self	0

Example

```

.TITLE CKSTAT
.ENABL LC
; Demonstration of MEMTOP, JSTAT and SNDMSG EMTs of TSX--Plus
ERRBYT = 52 ;EMT error code location
PRGNAM = 5 ;JSTAT subfunction code to get prog. name
.MCALL .TWAIT,.EXIT
START: MOV #MEMTOP,RO ;Point to EMT arg block to
      EMT 375 ;Set job size
;Only works in swapping environment, otherwise behaves like .SETTOP
      CMP RO,#HILIM ;See if we got what we wanted
      BHS AGAIN ;Go on if so
      .EXIT ;else quit (cannot disp err msg from det line)
AGAIN: MOV #1,LINE ;Check all lines starting with #1
CHECK: MOV #JSTAT,RO ;Point to EMT arg block
      EMT 375 ;Get name of job being run
      BCS ERRRTYP ;Go find out what kind of error
      CMP BUFADD,DUNJUN ;Is this line goofing off?
      BNE NEXT ;No, proceed
      CMP BUFADD+2,DUNJUN+2 ;May be, check for sure
      BNE NEXT ;No, proceed
;Send a message to the offending line
;(Each message must be < 88. bytes)
      MOV #LINE,YOOHOO ;Who is the guilty party?
      MOV #MESAG1,MSGADD ;Prepare part one of message
      MOV #SEND,RO ;Point to EMT arg block to
      EMT 375 ;Send a message to that line
      MOV #MESAG2,MSGADD ;Prepare for part two of the message
      MOV #SEND,RO ;Point to EMT arg block to
      EMT 375 ;Send part 2 of message
      MOV #MESAG3,MSGADD ;Prepare for part three of the message
      MOV #SEND,RO ;Point to EMT arg block to
      EMT 375 ;Send part 3 of message
      MOV #MESAG4,MSGADD ;Prepare for part four of the message
      MOV #SEND,RO ;Point to EMT arg block to
      EMT 375 ;Send part 4 of message
NEXT: INCB LINE ;Try next line
      CMPB LINE,MAXLIN ;Have we checked them all?
      BGT SLEEP ;Yes, wait awhile
      BR CHECK ;Go check the rest of the lines
SLEEP: .TWAIT #AREA,#TIME ;Come back in 5 minutes
      BR AGAIN ;And try again
ERRRTYP: CMPB @ERRBYT,#1 ;Which error is it
      BLT NEXT ;0 --> line not logged on, try next line
      BEQ 2$ ;1 --> invalid sub-function code, give up
      MOV #LINE,MAXLIN ;2 --> line > last valid line
      DECB MAXLIN ;Largest valid line number
      BR SLEEP ;should only happen first time
2$: .EXIT ;Invalid code should never happen
      ;Might as well kill job
MEMTOP: .BYTE 0,141 ;Argument block for MEMTOP EMT
       .WORD HILIM ;Upper address limit
JSTAT: .BYTE 0,144 ;Argument for JSTAT EMT
LINE: .BYTE 0 ;TSX-Plus line number to be checked
SUBFUN: .BYTE PRGNAM ;EMT subfunction
       .WORD BUFADD ;Address of 2-word buffer for returned value
BUFADD: .BLKW 2 ;2 word buffer to hold stat result
MAXLIN: .BYTE 30. ;Maximum number of lines under TSX--Plus
       .EVEN ;Will be altered to max valid line #
SEND: .BYTE 0,127 ;EMT arg block to send a message
YOOHOO: .WORD 0 ;Destination line number
MSGADD: .WORD MESAG1 ;Message to be sent
AREA: .BLKW 2 ;.TWAIT arg area
TIME: .WORD 0 ;time high word
       .WORD 5*60.*60. ;5min * 60.sec/min * 60.ticks/sec
.NLIST BEX

```

```

DUNJUN: .RAD50 /DUNJUN/          ;Name of illicit program
MESAG1: .ASCII <7><15><12>
        .ASCII /*****<15><12>
        .ASCIZ /*                */<15><12>
MESAG2: .ASCII /* Continued use of this system */<15><12>
        .ASCIZ /* for game playing will result */<15><12>
MESAG3: .ASCII /* in loss of user privileges!! */<15><12>
        .ASCIZ /*                */<15><12>
MESAG4: .ASCIZ /*****<15><12><7>
HILIM: .END    START

```

4.38 Determining file directory information (145)

This EMT returns directory information about a file. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument block:

```

.BYTE    chan,145
.WORD    dblk
.WORD    rblk

```

where *chan* is a channel number in the range 0–16 (octal) that is currently not in use, *dblk* is the address of a 4-word block containing the RAD50 file specification (device, file name, extension), and *rblk* is the address of a 7-word block that will receive the information about the file. The information returned in *rblk* is:

Word Number	Meaning
1	Size of the file (number of blocks).
2	0=File not protected; 1=File is protected.
3	File creation date (standard RT-11 date format).
4	File creation time (number of 3-second units).
5	Starting block number of file.
6	Unused (reserved)
7	Unused (reserved)

Error Code	Meaning
0	Channel is currently in use.
1	Unable to locate specified file.
2	Specified device is not file structured.

Example

```

.TITLE  FILINF
.ENABL  LC
; Demonstrate TSX--Plus EMT to return information about a file
.MCALL  .PRINT,.EXIT,.CSISPC,.TTYOUT
.GLOBL  DSPDAT,DSPTI3,PRTEC
ERRBYT = 52                ;EMT error code location
START:  .CSISPC #OUTSPC,#DEFLT,#0,#BUFFER ;Get file name

```

```

        MOV     #FILINF,RO      ;Point to EMT arg block to
        EMT     375             ;Get information about a file
        BCC     5$              ;No error?
        MOVB    @#ERRBYT,R1     ;What error
        ASL     R1              ;Convert to word index
        .PRINT  FIERR(R1)       ;Explain it
        .EXIT
5$:      MOV     #BUFFER,RO      ;Find the end of the file spec.
10$:     TSTB    (RO)+           ;End?
        BNE     10$             ;No, keep looking
        MOVB    #200,-(RO)      ;No CR,LF at end
        .PRINT  #BUFFER        ;File name
        .TTYOUT #'[
        MOV     FILSIZ,RO       ;File size
        CALL    PRTDEC
        TST     PRTCTD         ;Was file protected?
        BEQ     15$
        .TTYOUT #'P
15$:     .TTYOUT #']
        .PRINT  #SPACE2
        MOV     FILDAT,RO       ;File creation date
        CALL    DSPDAT         ;Display date
        .PRINT  #SPACE2
        MOV     FILTIM,RO       ;File creation time (3 sec resolution)
        CALL    DSPTI3         ;Display special 3-sec time
        .PRINT  #SPACE2
        MOV     FILLOC,RO       ;File starting block #
        CALL    PRTDEC
        .EXIT
        .NLIST  BEX
FILINF:  .BYTE   0,145          ;EMT arg block to get file info.
        .WORD   INSPC           ;Pointer to RAD50 file name
        .WORD   FILSIZ          ;Pointer to 7 word result buffer
FILSIZ:  .WORD   0              ;File size
PRTCTD:  .WORD   0              ;Protected=1, unprotected=0
FILDAT:  .WORD   0              ;File date (standard format)
FILTIM:  .WORD   0              ;File time (special 3-sec format)
FILLOC:  .WORD   0              ;File starting block number
        .WORD   0,0            ;Pad for 2 reserved words
OUTSPC:  .BLKW   15.            ;Output file specifications
INSPC:   .BLKW   24.            ;Input file specifications
DEFLT:   .WORD   0,0,0,0        ;No default extensions
FIERR:   .WORD   CINUSE         ;EMT error message table
        .WORD   NOFILE
        .WORD   BADDEV
BUFFER:  .BLKB   81.            ;Input string buffer
SPACE2:  .ASCII  / /<200>
CINUSE:  .ASCIZ  /?FILINF-F-Channel in use./
NOFILE:  .ASCIZ  /?FILINF-F-Can't find file./
BADDEV:  .ASCIZ  /?FILINF-F-Non-directory device./
        .END    START

```

4.39 Setting file creation time (146)

The time that a file is created is stored along with other directory information under TSX-Plus. In order to pack the time into a single word, TSX-Plus represents the file creation time in three second units. For example, if a file was created at 11:13:22, then the special time representation would be 13467 (decimal).

$$\begin{array}{rclcl}
 11 \text{ hr} & \times & 60 \text{ min/hr} & \times & 60 \text{ sec/min} & = & 39600 \text{ sec} \\
 13 \text{ min} & & & \times & 60 \text{ sec/min} & = & 780 \text{ sec} \\
 22 \text{ sec} & & & & & = & 22 \text{ sec} \\
 \hline
 & & & & & 40402 \text{ sec} & \div 3 = 13467 \text{ 3-sec units}
 \end{array}$$

A utility program is provided with TSX-Plus to display the creation time and other directory information about a file. See the *TSX-Plus User's Reference Manual* for more information on the *FILTIM* utility.

The creation date and time for a file are automatically stored by TSX-Plus in the directory entry for the file at the time that the file is closed after being created. An EMT is provided for those unusual situations where a different creation time is to be specified for a file after the file is created. The form of this EMT is:

EMT 375

with R0 pointing to the following argument block:

```
.BYTE  chan,146
.WORD  dblk
.WORD  time
```

where *chan* is the number of an unused channel, *dblk* is the address of a 4-word block containing the RAD50 file specification, and *time* is the time value (in 3-second units since midnight) that is to be set as the creation time for the file.

Error Code	Meaning
0	Channel is currently in use.
1	Unable to locate specified file.
2	Specified device is not file structured.

Example

```
.TITLE  SFTIM
.ENABL  LC
; Demonstrate TSX-Plus EMT to set file creation time
.MCALL  .PRINT,.CSISPC,.EXIT,.GTLLIN
.GLOBL  ACRTI3          ;Subroutine to convert hh:mm:ss to special
                        ;3-sec internal time format in R0
ERRBYT = 52            ;EMT error code location
START:  .PRINT  #GETNAM  ;Prompt for file name
        .CSISPC #OUTSPC,#DEFLT ;Get file name in RAD50
        .GTLLIN #BUFFER,#GETTIM ;Prompt for and get a time
        MOV     #BUFFER,R0    ;Point to time input buffer
        CALL    ACRTI3        ;Get special time in R0
        BCC     1$            ;Time error?
        .PRINT  #BADTIM       ;Yes, incorrect format
        .EXIT
1$:      MOV     RO,NEWTIM      ;Save special time
        MOV     #SFTIM,R0      ;Point to EMT arg block to
        EMT     375            ;Set creation time in file
        BCC     2$            ;Error?
        MOVB    @ERRBYT,R0     ;Yes, get error code
        ASL     RO             ;Convert to word offset
        .PRINT  SFTERR(RO)     ;Explain
2$:      .EXIT
        .NLIST  BEX
SFTIM:  .BYTE   0,146          ;EMT arg block to set file creation time
        .WORD   INSPC          ;Pointer to RAD50 file name
NEWTIM:  .WORD   0             ;Will contain new creation time
OUTSPC:  .BLKW   15.           ;.CSISPC output files
INSPC:   .BLKW   24.           ;.CSISPC input files (first is the one)
DEFLT:   .WORD   0,0,0,0       ;Default file extensions
SFTERR:  .WORD   INUSE         ;SFTIM error message table
        .WORD   NOFILE
        .WORD   BADDEV
```

```

BUFFER: .BLKB 81. ;.GTLIN input buffer - holds time hh:mm:ss
GETNAM: .ASCII /Set creation time in file: /<200>
GETTIM: .ASCII /New creation time: /<200>
BADTIM: .ASCIZ /?SFTIM-F-Invalid time./
INUSE: .ASCIZ /?SFTIM-F-Channel in use./
NOFILE: .ASCIZ /?SFTIM-F-Can't find file./
BADDEV: .ASCIZ /?SFTIM-F-Non-directory device./
.END START

```

4.40 Determining or changing the user name (0/1,147)

When using the LOGON system access program, each user is assigned both a user name and a project-programmer number. TSX-Plus provides an EMT which allows an application program to obtain the user name or (with SETNAME privilege) to change it. User names may be up to twelve characters in length. If the LOGON program is not used, the user name will initially be blank, although it may be changed to a non-blank name. The form of the EMT is:

```
EMT 375
```

with R0 pointing to the following argument block to determine the user name:

```
.BYTE 0,147
WORD buff-addr
```

where *buff-addr* is a pointer to a 12-byte area to contain the user name which is returned.

To change the current user name, R0 should instead point to the following argument block:

```
.BYTE 1,147
WORD buff-addr
```

where *buff-addr* is a pointer to a 12-byte area containing the new user name. SETNAME privilege is required to change the user name. If changing the user name is attempted without SETNAME privilege, the name will not be changed and the carry bit will be set on return.

Example

```

.TITLE GSUNAM
.ENABL LC
; Demonstrate TSX-Plus EMT to get/set user name
ERRBYT = 52 ;EMT error code location
.MCALL .PRINT,.EXIT
START: .PRINT #NAMEIS ;Preface user name
MOV #GSUNAM,R0 ;Point to EMT arg block to
EMT 375 ;Get user name
.PRINT #NAMBUF ;And display it
MOV #NEUNAM,NAMADD ;Point to new user name
INCB GSUNAM ;Set low bit to set name
MOV #GSUNAM,R0 ;Point to EMT arg block to
EMT 375 ;Set new user name
BCC 1$ ;Error?
.PRINT #NOPRIV ;Must have SETNAME privilege
1$: .EXIT
.NLIST BEX
GSUNAM: .BYTE 0,147 ;EMT arg block to get user name
NAMADD: .WORD NAMBUF ;Pointer to receive area
NAMBUF: .BLKW 6 ;Six word name area (12 bytes)
.WORD 0 ;Make it ASCIZ
NEUNAM: .ASCII /CHAUNCY / ;The new name (12 bytes)
NAMEIS: .ASCII /Your current user name is: /<200>
NOPRIV: .ASCIZ /SETNAME privilege necessary to set user name./
.END START

```

4.41 Determining or changing the program name (2/3,147)

The following EMTs allow you to determine or change the name of the current program. This changes the name in the SYSTAT (WHO, SHOW JOBS) display, not the directory name of the executable program image. The form of the EMT is:

EMT 375

with R0 pointing to the following argument block:

```
.BYTE    sub-function,147
.WORD    buff-addr
```

If *sub-function* is 2 the EMT returns the current program name in RADIX-50 format in the two word buffer pointed to by *buff-addr*. If *sub-function* is 3 the EMT changes the program name to the RADIX-50 format name pointed to by *buff-addr*.

The following error code can be returned by this EMT:

Error Code	Meaning
2	The name was not specified in RADIX-50 format.

Example

```
.TITLE    SETNAM
.MCALL    .PRINT,.EXIT
.GLOBL    PRTDEC,RADASC
.DSABL    GBL

ERRBYT    = 52                    ;EMT error byte address

START:    MOV        #GSPNAM,R0        ;Point to EMT arg. to
EMT        375                    ;Get program name
BCS        1$                    ;Branch if not OK
MOV        R3,-(SP)                ;Save start addr. of name
MOV        #NAMBUF,R5              ;Get addr of string to convert
MOV        #2,R4                  ;Say convert 2 words
CALL        RADASC
MOV        (SP)+,R3                ;Get start addr. of name
.PRINT     R3                    ;Display program name
MOV        RADNAM,NAMBUF          ;Point to new program name
MOV        RADNAM+2,NAMBUF+2
INCB        GSPNAM                ;Set low byte to set name
MOV        #GSPNAM,R0              ;Point to EMT arg. block to
EMT        375                    ;Set program name
BCC        2$                    ;Branch if OK
1$:        MOVB       @ERRBYT,-(SP)   ;Fetch EMT error code
.PRINT     #ERRIS                ;"EMT error is:"
MOV        (SP)+,R0                ;Retrieve error code
CALL        PRTDEC                ;Display it
2$:        DECB       GSPNAM                ;Set low byte to get name
MOV        #GSPNAM,R0              ;Point to EMT arg. to
EMT        375                    ;Get program name
BCS        1$                    ;Branch if not OK
MOV        R3,-(SP)                ;Push R3
MOV        #NAMBUF,R5              ;Get addr of string to convert
MOV        #2,R4                  ;Say convert 2 words
CALL        RADASC
MOV        (SP)+,R3                ;Pop R3
```



```

.PRINT R3           ;Display program name
.EXIT
.WLIST BEX
GSPNAM: .BYTE 2,147 ;EMT arg. block to get or
.WORD NAMBUF        ;Set program name
NAMBUF: .BLKW 6      ;Two word name area
RADNAM: .RAD50 /NEWNAM/
ERRIS:  .ASCII  /?SETNAM-F-EMT error is: /<200>
.EVEN
.END START

```

4.42 Determining or changing the terminal name (4/5,147)

The following EMTs allow you to determine or change the name of a terminal line. This changes the name in the SHOW TERMINAL display, and the SYSMON terminal status display. This change remains in effect until this EMT is reissued, the SET TT NAME command is issued, or the system is rebooted. This request requires terminal privilege. The form of the EMT is:

EMT 375

with R0 pointing to the following argument block:

```

.BYTE sub-function,147
.WORD buff-addr
.WORD line-num

```

If *sub-function* is 4 the EMT returns the current terminal name in ASCII format in the buffer pointed to by *buff-addr*. If *sub-function* is 5 the EMT changes the program name to the ASCII format name pointed to by *buff-addr*. The longest possible name length for the specified terminal is always returned in R0.

The following error code can be returned by this EMT:

Error Code	Meaning
3	The string was too long.

Example

```

.TITLE GSTNAM
.ENABL LC

; Demonstrate TSX-Plus EMT to get/set TERMINAL name

ERRBYT = 52 ;EMT error code location

.MCALL .PRINT,.EXIT

START: .PRINT #NAMEIS ;Preface console name
MOV #GSTNAM,R0 ;Point to EMT arg block to
EMT 375 ;Get console terminal name
.PRINT #NAMBUF ;And display it

MOV #NEWNAM,NAMADD ;Point to new terminal name
INCB GSTNAM ;Set low bit to set name
MOV #GSTNAM,R0 ;Point to EMT arg block to
EMT 375 ;Set new terminal name

```

```

        BCC      1$          ;Error?

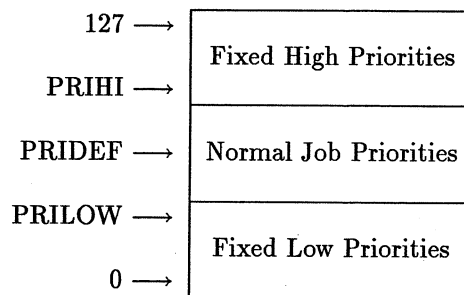
1$:      .PRINT  #NOPRIV      ;Must have Terminal privilege
        .EXIT

        .LIST   BEX
GSTNAM:  .BYTE   4,147        ;EMT arg block to get terminal name
NAMADD:  .WORD   NAMBUF       ;Pointer to receive area
        .WORD   1.           ;Say get/set console terminal name
NAMBUF:  .BLKW   8.           ;Old name buffer
        .WORD   0            ;Make it ASCIZ
NEWNAM:  .ASCIZ  /My Term/    ;The new name buffer
NAMEIS:  .ASCII  /The current terminal name for the console is: /<200>
NOPRIV:  .ASCIZ  /Terminal privilege necessary to set terminal name./
        .END    START

```

4.43 Setting job priority (0,150)

Jobs may be assigned priority values in the range 0 to 127 to control their execution scheduling relative to other jobs. The priority values are arranged in three groups: the fixed-low-priority group consists of priority values from 0 up to the value specified by the PRILOW sysgen parameter; the fixed-high-priority group ranges from the value specified for the PRIHI sysgen parameter up to 127; the middle priority group ranges from (PRILOW+1) to (PRIHI-1). The following diagram illustrates the priority groups:



Job scheduling is performed differently for jobs in the fixed-high-priority and fixed-low-priority groups than for jobs with normal interactive priorities. Jobs with priorities in the fixed-low-priority group (0 to PRILOW) and the fixed-high-priority group (PRIHI to 127) execute at fixed priority values. That is, the priority absolutely controls the scheduling of the job for execution relative to other jobs. A job with a fixed priority is allowed to execute as long as it wishes until a higher priority job becomes active.

The fixed-high-priority group is intended for use by real-time programs. The fixed-low-priority group is intended for use by very low priority background tasks. Normal time-sharing jobs should not be assigned priorities in either of the fixed priority groups.

The middle group of priorities from (PRILOW+1) to (PRIHI-1) are intended to be used by normal, interactive, time-sharing jobs. Jobs with these assigned priorities are scheduled in a more sophisticated manner than the fixed-priority jobs. In addition to the assigned priority, external events such as terminal input completion, I/O completion, and timer quantum expiration play a role in determining the effective scheduling priority.

When a job with a normal priority switches to a subprocess, the priority of the disconnected job is reduced by the amount specified by the PRIVIR sysgen parameter. This causes jobs that are not connected to terminals to execute at a lower priority than jobs that are. This priority reduction does not apply to jobs with priorities in the fixed-high-priority group or the fixed-low-priority group. The priority reduction is also constrained so that the priority of jobs in the normal job priority range will never be reduced below the value of (PRILOW+1).

The following EMT can be used to set the job priority from within a program. The job priority can also be set from the keyboard with the SET PRIORITY command. The current job priority, maximum allowed priority, and fixed-high-priority and fixed-low-priority boundaries may be determined with the .GVAL request. See the *TSX-Plus System Manager's Guide* for more information on the significance of priority in job scheduling. The form of this EMT is:

EMT 375

with R0 pointing to the following EMT argument block:

```
.BYTE 0,150
.WORD value
```

where *value* is the priority value for the job. The valid range of priorities is 0 to 127 (decimal). The maximum job priority may be restricted by the system manager. If a job attempts to set its priority to zero or less, its priority will be set to the default value. If a job attempts to set its priority above its maximum allowed priority, its priority will be set to the maximum allowed. This EMT does not return any errors.

Example

```
.TITLE GSPRI
.ENABL LC
; Demonstrate EMT to set job priority
.MCALL .GVAL,.GTIN,.PRINT,.EXIT
.GLOBL PRTDEC
CURPRI = -16. ;GVAL offset to get current priority
MAXPRI = -18. ;GVAL offset to get maximum priority
START: .PRINT #CURIS ;"current priority is"
       .GVAL #AREA,#CURPRI ;Obtain current job priority in R0
       MOV R0,R1 ;Save it
       CALL PRTDEC ; and display it
       .PRINT #MAXIS ;"maximum priority is"
       .GVAL #AREA,#MAXPRI ;Obtain maximum allowable job priority
       MOV R0,R2 ;Save it
       CALL PRTDEC ; and display it
       ADD #10.,R1 ;Try to boost priority by 10
       CMP R1,R2 ;Unless exceeds maximum
       BLE 1$ ;Use 10 larger if <= maxpri
       MOV R2,R1 ;Else use maxpri
1$: .MOV R1,NEWPRI ;Set new priority in EMT arg block
   .MOV #SETPRI,R0 ;Point to EMT arg block to
   .EMT 375 ;Sset new job priority
   .PRINT #NEWIS ;"new priority is"
   .GVAL #AREA,#CURPRI ;Obtain new priority
   CALL PRTDEC ; and display it
   .EXIT
AREA: .BLKW 10 ;General EMT arg block
SETPRI: .BYTE 0,150 ;EMT arg block to set job priority
NEWPRI: .WORD 50. ;New job priority goes here
       .NLIST BEX
CURIS: .ASCII /Current job priority = /<200>
MAXIS: .ASCII <15><12>/Maximum job priority = /<200>
NEWIS: .ASCII <15><12>/- New - job priority = /<200>
       .END START
```

4.44 Determining or changing job privileges (1,150)

A TSX-Plus EMT is available to allow running programs to determine the privileges for the job and to change privileges. This is particularly useful to check user-defined privileges. See the *TSX-Plus System Manager's Guide* for complete information on job privileges. The form of the EMT argument block is:

EMT 375

with R0 pointing to an argument block of the following form:

```
.BYTE 1,150
.BYTE function,privtype
.WORD buffer
.WORD 0
```

where *buffer* is the address of a four word buffer which contains the privilege flags to be set or cleared or which will receive the privilege flags. Note, not all of the privilege words may be in current use, but four words should be reserved to allow for future expansion. *Function* indicates the type of operation being done and must be 0, 1, or 2 according to the following table:

Function	Meaning
0	Read job's privilege flags into <i>buffer</i>
1	Clear bits set in <i>buffer</i> in job's privilege flag bits
2	Set bits set in <i>buffer</i> in job's privilege flag bits

Functions 1 and 2 are bit-clear and bit-set operations so that individual privileges may be selectively changed without affecting other privileges.

Privtype indicates which of the three sets of privilege flags are to be accessed, and must be 0, 1, or 2 according to the following table:

<i>Privtype</i>	Privilege table
0	Current privileges
1	Set privileges
2	Authorized privileges

The current privileges for the job are reset to the set privileges when the currently executing program exits. If set privileges are changed, then current privileges are changed as well. If authorized privileges are changed, then set and current privileges are changed as well. SETPRV privilege is required to set any new privileges in the authorized privilege set.

Error Code	Meaning
1	Attempt to enable privileges for which the job is not authorized. Only those privileges for which the job is authorized are set.

Example

```

        .TITLE  SETPRV
        .ENABL  LC
        .DSABL  GBL
; Demonstrate EMT to determine or set privileges.
        .MCALL  .PRINT, .EXIT, .TTYOUT
        .GLOBL  PRTOCT
SPACE   = 40                ;ASCII SPACE
ERRBYT  = 52                ;EMT ERROR BYTE
START:  CALL    SHOPRV       ;DISPLAY PRIVILEGES BEFORE
        .PRINT  #CRLF
        MOVB    #2,RCS       ;WANT TO SET PRIV FLAGS
        MOVB    #1,CSA       ;SELECT "SET" PRIVILEGES
        MOV      #700,PRVS    ;SET 3 PRIV FLAGS IN PRIV WORD 1
        CLR     PRVS+2        ;DO NOT CHANGE ANY IN PRIV WORD 2
        MOV     #GSPRV,RO     ;POINT TO EMT ARG BLOCK TO
        EMT     375           ;SET SOME PRIVILEGE FLAGS
        BCC     1$           ;BRANCH ON ERROR
        MOVB    @#ERRBYT, -(SP) ;ELSE GET ERROR BYTE
        ADD     #'0, (SP)     ;CONVERT TO DIGIT
        .PRINT  #ERR
        .TTYOUT (SP)+
        .PRINT  #CRLF
1$:     CALL    SHOPRV       ;DISPLAY PRIVILEGES AFTER
        .EXIT

```

```

SHOPRV: .PRINT  #AUTPRV      ;"AUTHORIZED"
        MOV     #2,CSA      ;LOOK AT AUTHORIZED PRIVILEGES
        CALL    DSPPRV      ;READ AND DISPLAY FIRST 2 PRIV WORDS
        .PRINT  #SETPRV      ;"SET"
        MOV     #1,CSA      ;LOOK AT SET PRIVILEGES
        CALL    DSPPRV
        .PRINT  #CURPRV      ;"CURRENT"
        CLRB    CSA         ;LOOK AT CURRENT PRIVILEGES
        CALL    DSPPRV      ;DISPLAY PRIVILEGE BITS
        RETURN
DSPPRV: CLRB     RCS         ;READ PRIVILEGES
        MOV     #GSPRV,R0    ;POINT TO EMT ARG BLOCK TO
        EMT     375         ;READ SELECTED PRIVILEGES
        MOV     PRVS,R0      ;GET PRIV WORD 1
        CALL    PRTCT       ;DISPLAY ITS BITS
        .TTYOUT  #SPACE
        MOV     PRVS+2,R0    ;GET PRIV WORD 2
        CALL    PRTCT
        .PRINT  #CRLF
        RETURN
GSPRV:  .BYTE    1,150      ;GET/SET PRIVILEGE EMT
RCS:    .BYTE    0          ;0=READ,1=CLEAR,2=SET
CSA:    .BYTE    0          ;0=CURRENT,1=SET,2=AUTHORIZED
        .WORD    PRVS       ;PUT PRIV FLAGS HERE
        .WORD    0          ;REQUIRED 0
PRVS:   .WORD    0,0,0,0    ;PRIVILEGE FLAGS 2 OF 4 USED
        .WLST     BEX
AUTPRV: .ASCII   /Authorized privileges: /<200>
SETPRV: .ASCII   /Set privileges: /<200>
CURPRV: .ASCII   /Current privileges: /<200>
CRLF:   .ASCIZ   //
ERR:    .ASCII   /Set privilege error # /<200>
        .LIST     BEX
        .END      START

```

4.45 Specifying a file for HOLD or NOHOLD mode (151(0))

The following EMT is used to dynamically request that a file being printed through the spooler be either held until the file is closed or begin printing as data is made available from the program. This could be used in a situation where NOHOLD is the normal condition, but a program which uses the printer generates data slowly. If data were passed to the printer as soon as available, then printer output from all other jobs would be delayed until the slow job closes the output. This can be avoided by the having the slow program select hold mode for its output. Then, other jobs can proceed to use the printer without being delayed by the slow job. The form of the EMT to select HOLD or NOHOLD mode on an individual file basis is:

```
EMT      375
```

with R0 pointing to the following argument block:

```

.BYTE    chan,151
.WORD    0
.WORD    flag

```

where *chan* is the channel number which has been used to open the print file and *flag* indicates whether the file is to be printed as it is generated or held until the file is closed. If *flag*=0, the output is printed as generated (equivalent to NOHOLD); if *flag*=1, then the output is not printed until the file is closed. This EMT must be issued *after* a channel has been opened to the printer (through the spooler), but before any data has been written to it. If the channel is open to any non-spoiled device, then the EMT is ignored.

Example

```

        .TITLE  SPHOLD
        .ENABL  LC
; Demonstrate the EMT to hold spooler output until the file is closed
        .MCALL  .CSISPC, .LOOKUP, .READW, .WRITW, .CLOSE, .EXIT
ERRBYT = 52 ;EMT error code byte location
START:  .CSISPC #OUTSPC, #DEFEXT, #0 ;Get name of file to copy
        BCS     START ;Proceed unless error
        MOV     #INSPC, R1 ;Point to first input filspc
OPWFIL: .LOOKUP #AREA, #0, R1 ;Try to open input file
        BCS     START ;Get a new command on error
        MOV     #OUTSPC, R2 ;Point to output filspc
        MOV     #~RLP, (R2)+ ;Put LP: in output filspc
        ADD     #2, R1 ;Point to input filspc filename
        MOV     (R1)+, (R2)+ ;Move file name into LP filspc
        MOV     (R1)+, (R2)+ ; (not necessary, but convenient)
        TST     (R1)+ ;Skip over file extension
        .LOOKUP #AREA, #1, #OUTSPC ;Open channel to printer (spooled)
        BCC     NOHOLD ;Proceed unless error
GIVEUP: .CLOSE #0 ;Close input file
        .EXIT ;And give up
; Tell spooler to hold file until it is closed
; (must be issued before any writes to file)
NOHOLD: MOV     #SPHOLD, R0 ;Point to EMT arg block to
        EMT     375 ;Hold output until close
        CLR     R2 ;Initialize block pointer
6$:     .READW #AREA, #0, #BUFFER, #256, R2 ;Copy a block from the file
        BCS     NXTFIL ;Try next file on error
        .WRITW #AREA, #1, #BUFFER, #256, R2 ;Copy the file block to LP
        BCC     8$ ;Error?
        .CLOSE #1 ;Close print file
        BR      GIVEUP ;Forget it
8$:     INC     R2 ;Point to next block
        BR      6$ ;And get next block
NXTFIL: .CLOSE #0 ;Close input file
        .CLOSE #1 ;and print file
        TST     2(R1) ;Any input file?
        BNE     OPWFIL ;Repeat if so
        BR      START ;Else ask for more files
AREA:   .BLKW   10 ;General EMT arg block area
SPHOLD: .BYTE   1, 151 ;EMT arg block to hold spool output
        .WORD   0 ;on channel 1 until file is closed
HNH:    .WORD   1 ;HNH=0 immed; HNH=1 hold til close
OUTSPC: .BLKW   15. ;Output file specs
INSPC:  .BLKW   24. ;Input file specs
DEFEXT: .WORD   0, 0, 0, 0 ;No default file types
BUFFER: .BLKW   256. ;I/O buffer area
        .END    START

```

4.46 Turning flag pages on or off for a device (151(1))

The following EMT is used to dynamically turn flag pages on or off for a spooled device. The form of this EMT is:

```
EMT      375
```

with R0 pointing to the following argument block:

```

        .BYTE   chan, 151
        .WORD   1
        .WORD   flag

```

where *chan* is the channel number that has been opened to the spooled device, and *flag* indicates whether flag pages are to be turned on or off. If *flag*=0, flag pages are turned off; if *flag*=1, flag pages are turned on. This EMT must be issued *after* a channel has been opened to the spooled device, but *before* anything is written to it. If the channel is opened to a non-spooled device, the EMT is ignored. If the specified device is not found, the carry bit will be set on return from the EMT.

4.47 Setting width of spooler flag pages (151(2))

The following EMT is used to dynamically change the width used to center flag pages for a spooled device. The form of the EMT is:

EMT 375

with R0 pointing to the following argument block:

```
.BYTE     chan,151
.WORD     2
.WORD     flag
```

where *chan* is the channel number of the channel that has been opened to the spooled device, and *flag* indicates whether flag pages are to be centered on 80 or 132 columns. If *flag*=0, flag pages are set to 80 columns in width; if *flag*=1, flag pages are set to 132 columns wide. This EMT must be issued *after* a channel is opened to the spooled device, but *before* anything is written to it. If the channel is opened to a non-spooled device, the EMT is ignored. If the specified device is not found, the carry bit will be set on return from the EMT.

4.48 Program controlled terminal options (152)

Programs may dynamically change various parameters related to terminal control. The following EMT may be used to set various program controlled terminal options:

EMT 375

with R0 pointing to the following argument block:

```
.BYTE     0,152
.WORD     function-code
.WORD     argument-value
```

where *function-code* is a character which specifies which option is to be set or changed, and *argument-value* specifies a value used only by some options. See Chapter 3 on program controlled terminal options earlier in this manual for more information on the specific options which may be selected and details on their effects.

Example

See the example program EMTMTII in Chapter 3.

4.49 Forcing [non]interactive job characteristics (153)

The following EMT can be used to cause a job to be scheduled either as an interactive job or as a non-interactive job. Programs which do a large amount of terminal input, but which are not truly interactive jobs in the usual sense, such as file transfer programs, should use this EMT to avoid excessive interference with normal interactive time-sharing jobs. This feature may also be selected with the R[UN]/NONINTERACTIVE command. See the *TSX-Plus System Manager's Guide* for more information on job scheduling and the significance of interactive vs. non-interactive jobs. The form of this EMT is:

EMT 375

with R0 pointing to the following argument block:

```
.BYTE 0,153
.WORD mode
.WORD 0
```

If the value of *mode* is 0, then the job will never be scheduled as an interactive job. If *mode* is 1, then the job will be scheduled as other interactive jobs are, dependent on terminal input.

Example

```
.TITLE NONINT
.ENABL LC
; Demonstrate EMT to schedule job as interactive or non-interactive
.MCALL .TTYIN,.TTYOUT,.PRINT,.EXIT
JSW    = 44          ;Job Status Word address
TTSPC  = 10000       ;TT special mode bit (single-char)
CTRLZ  = 32          ;ASCII CTRL-Z (move on command)
START: MOV    #SINGLE,R0 ;Point to EMT arg block to
      EMT     375       ;Turn on single character activation
      BIS     #TTSPC,@#JSW ;Finish turning on single char mode
      MOV     #NONINT,R0 ;Point to EMT arg block to
      EMT     375       ;Schedule this as non-interactive job
      .PRINT  #SLOW      ;"May be slow now if system busy"
1$:    .TTYIN          ;Get a char
      CMPB   RO,#CTRLZ   ;If CTRL-Z
      BEQ    2$         ;Then move on
      .TTYOUT          ;Else echo it back (we have to echo
                        ; when in single char mode)
      BR     1$         ;And repeat
2$:    MOV     #1,SELECT ;Want to be interactive now
      MOV     #NONINT,R0 ;Point to EMT arg block to
      EMT     375       ;Schedule this as an interactive job
      .PRINT  #FAST      ;"See how much faster now"
3$:    .TTYIN          ;Get a char
      CMPB   RO,#CTRLZ   ;If CTRL-Z
      BEQ    4$         ;Then move on
      .TTYOUT          ;Else echo it back
      BR     3$         ;And repeat
4$:    .EXIT
SINGLE: .BYTE 0,152       ;EMT arg block to set term option
      .WORD  'S         ;Single char activation
      .WORD  0
NONINT: .BYTE 0,153       ;EMT arg block to sched as [non]interactive
SELECT: .WORD 0          ;Initially make non-interactive
      .WORD  0
      .MLIST BEX
SLOW:  .ASCII /Type some characters in now. If the system has several /
      .ASCII /interactive jobs/<15><12>
      .ASCII /response will be slow. (Control-Z to get out /
      .ASCIZ /of this mode.)/
FAST:  .ASCIZ <15><12>/Try again. Response should be much better./
      .END    START
```

4.50 Setting terminal baud rates (0,154)

The transmit/receive speed for time-sharing lines or CL lines may be set either with the keyboard command SET TT SPEED or from within a program. Use of this EMT requires TERMINAL privilege. Line speeds may only be set for terminal interfaces which support programmable baud rates, such as: DLV11-E, DZ(V)11, DH(V)11 type interfaces and the PRO-350 printer, communication and QSL ports. The EMT to set line speeds from within a program is:

EMT 375

with R0 pointing to an argument block of the form:

```
.BYTE 0,154
.WORD line-number
.WORD line-parameters (OPLxSSSS)
```

where *line-number* indicates the TSX-Plus line number for which the speed is to be set. If *line-number* is 0, then the speed is set on the line from which the EMT is issued. *Line-parameters* is a combined word which is used to specify the speed, number of data bits, and parity control. Bits 0-3 "SSSS" select the baud rate according to the following table:

Speed	Octal Code
75	1
110	2
134.5	3
150	4
300	5
600	6
1200	7
1800	10
2000	11
2400	12
3600	13
4800	14
7200	15
9600	16
19200	17

Split speeds (different transmit and receive baud rates) are not supported. Bit 4 (*x*) is not used. Bit 5 (*L*) specifies the character length. If this bit is 0, the character length is 8 bits; if this bit is 1, the character length is 7 bits. Bit 6 (*P*) specifies if parity control is wanted. If this bit is 0, no parity is selected and bit 7 is ignored; if this bit is 1, parity generation and checking is enabled. Bit 7 (*O*) selects even or odd parity and is only meaningful if bit 6 is 1. If bit 7 is 0, even parity is selected; if bit 7 is 1, odd parity is selected. Note that if only the speed value is specified, with all other bits zero, 8 bit characters with no parity are selected.

A baud rate of 19200 is not supported by DEC DZ(V)11 controllers. DHV11 interfaces do not support 3600 or 7200 baud. DH11 interfaces do not support 2000, 3600 or 7200 baud. The PRO QSL interface does not support 3600 or 7200 baud.

Error Code	Meaning
1	Job does not have TERMINAL privilege
2	Invalid line number specified

Example

```
.TITLE SETSPD
.ENABLE LC
; Demonstrate EMT to set a line's transmit/receive baud rate
```

```

        .MCALL .PRINT,.EXIT
        .GLOBL PRTDEC
        .DSABL GBL
ERRBYT = 52                ;EMT error byte address
START:  MOV    #SETSPD,R0   ;Point to EMT arg block to
        EMT    375         ;Set line speed
        BCC    9$          ;Branch if OK
        MOVB   @#ERRBYT,-(SP) ;Fetch EMT error code
        .PRINT #ERRIS      ;"EMT error is:"
        MOV    (SP)+,R0     ;Retrieve error code
        CALL   PRTDEC       ;Display it
9$:     .EXIT
SETSPD: .BYTE   0,154       ;EMT arg block to set line speed
        .WORD   6          ;Line number
        .WORD   14.        ;Speed code for 9600 baud
        .MLIST BEX
ERRIS:  .ASCII  /?SETSPD-F-EMT error code = /<200>
        .EVEN
        .END    START

```

4.51 Raising and lowering the DTR signal on a line (1/2/3,154)

The following EMT can be used to raise or lower the DTR (data terminal ready — pin 20 on and EIA/RS232-C connector) signal on a line. This is only effective if the interface supports modem control signals (e.g., this has no effect on a DLV11J port). This can be useful in situations where a modem or data PBX requires the DTR signal.

The DTR signal is also manipulated by the system when connecting and disconnecting “phone” lines.

The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following EMT argument block:

```

        .BYTE   sub-function,154
        .BYTE   line-number,0
        .WORD   speed

```

where *line-number* indicates which physical line is to be affected, *speed* is only used with subfunction 0 (zero), and *sub-function* controls the subfunction to be performed, as follows:

Error Code	Meaning
0	Set line speed (see previous description for 0,154)
1	Reset line XOFF status
2	Raise line DTR
3	Lower line DTR

TERMINAL privilege is required to issue this EMT unless the line to be affected is the job issuing the EMT.

Example

```

        .TITLE  SETDTR
        .MCALL  .PRINT,.EXIT,.GTLIN

```

```

.DSABL GBL
.GLOBL ACRDEC,PRTDEC

START: MOV    #BUFFER,R1
.PRINT #INSTRC      ;Print instructions
.GTLLIN R1,#WHICH   ;Ask for line number
CALL   ACRDEC       ;Convert it from ascii number
TST    RO           ;Set or clear?
BGT    1$           ;Branch if set
NEG    RO           ;Positive
MOV    RO,CLRLIN    ;Save line number
MOV    #CLRDTR,RO   ;Point to EMT arg block to clear DTR
BR     2$           ;Branch to EMT
1$:    MOV    RO,SETLIN ;Save line number
MOV    #SETDTR,RO   ;Point to EMT arg block to raise DTR
2$:    EMT    375     ;Do EMT
BCC    9$           ;Branch if no error
MOVB   @#52,-(SP)   ;Else get error code
.PRINT #EMTERR      ;"SET DTR error number:"
MOV    (SP)+,RO     ;Retrieve error code
CALL   PRTDEC       ;And display it
9$:    .EXIT
.NLIST BEX
CLRDTR: .BYTE 3,154 ;EMT arg. block to lower DTR
CLRLIN: .WORD 0,0
SETDTR: .BYTE 2,154 ;EMT arg. block to raise DTR
SETLIN: .WORD 0,0
BUFFER: .BLKB 81.
INSTRC: .ASCIZ /To raise DTR enter the line number;/
WHICH:  .ASCII /to lower enter the negative line number: /<200>
EMTERR: .ASCII /SET DTR error # /<200>
.EVEN
.END    START

```

4.52 CL device control (0,155)

The CL handler allows Input/Output operations to be performed to serial communication lines connected to any type of interface controller which may be used for terminal lines. The following EMT's can be used to control the function of individual CL units from within a program. For further explanation of the CL handler see Chapter 7 section 7.1.

Time-sharing terminal lines which are not in use may be reassigned as general purpose serial I/O lines by directing the TSX-Plus CL facility to assign a CL unit to the line. This can be done either with a keyboard command (see the SET CL LINE=*n* command in the *TSX-Plus User's Reference Manual*) or from within a program. A special system service call (EMT) is available to assign a CL line from within a program. Use of this EMT requires TERMINAL privilege. The form of the EMT is:

```
EMT    375
```

with R0 pointing to an argument block of the form:

```

.BYTE 0,155
.WORD cl-unit
.WORD line-number

```

where *cl-unit* specifies the CL unit number to be assigned to the line, and *line-number* identifies the TSX-Plus line number to be disabled as a time-sharing line and reassigned as a CL unit. The valid range of CL unit numbers is determined by the number of CL units defined during TSX-Plus system generation. For example, if 3 CL units are defined, then the valid CL units are CL0, CL1 and CL2. If *line-number* is zero,

then the CL unit is disassociated from the line and the line is restored to its previous function. The line number may also refer to lines generated as dedicated CL lines. In this case, when a CL unit is disassociated from the line, it is simply returned to a pool of lines available for CL use. It does not become redefined as a time-sharing line.

See Chapter 7 and the *TSX-Plus System Manager's Guide* for more information on CL units.

Error Code	Meaning
1	Job issuing request does not have TERMINAL privilege
2	Invalid CL unit number specified
3	Invalid line number specified
4	Specified line number already assigned to a CL unit
5	Specified line number in use for time-sharing
6	Specified CL unit is currently busy

Example

```
.TITLE GETCL
.ENABLE LC
; Demonstrate EMT to switch time-sharing line to CL line
; Demonstrate EMT to allocate a device
; Demonstrate .SPFUN request to CL
; This example attempts to attach a CL unit to
; a line which is linked to another machine,
; allocate it for exclusive use,
; modify the default CL settings,
; and then start up the RT-11 VTCOM utility.
; Since it is difficult to make logical assignments from within
; a program, use a special .EXIT to pass the ASSIGN command to
; KMON and then run VTCOM.
.MCALL .PRINT,.EXIT,.LOOKUP,.SPFUN,.PURGE
.DSABL GBL
ERRBYT = 52 ;EMT error byte address
JSW = 44 ;Job Status Word address
CHNIF$ = 4000 ;Chain information bit in JSW
START:
; Issue EMT to attach the line
MOV #ATTCL,RO ;Point to EMT arg block to
EMT 375 ;Attach CL unit to T/S line
BCC 1$ ;Branch if OK, else
; EMT error, explain it before exiting
MOVB @ERRBYT,R1 ;Get EMT error code
ASL R1 ;Convert error byte to word index
.PRINT ATTERR(R1) ;Print appropriate error message
BR 10$ ;And force simple exit
; Allocate device so nobody else infringes it
1$: MOV #ALOCAT,RO ;Point to EMT arg block to
EMT 375 ;Allocate CL for exclusive use
BCC 2$ ;Branch if OK, else
; EMT error, explain it before exiting
MOVB @ERRBYT,R1 ;Get EMT error code
ASL R1 ;Convert to word index
.PRINT ALLERR(R1) ;Print appropriate error message
BR 10$ ;And exit
; Issue SET CL command (not the easiest way to do this).
; If a channel were already open to CL, then this would
; make more sense. But, as an example, why not?
2$: .LOOKUP #AREA,#0,#CLUNAM ;Open a channel to CL
.SPFUN #AREA,#0,#251,#CLFLAG,#0,#0 ;Turn off flagged options
```

```

        .PURGE #0                ;Done with channel
; Set up for special exit passing commands to KMON
        MOV #COMAND,R1          ;Point to command line text
        MOV #510,R2            ;Point to chain info area
        MOV #COMLEN,(R2)+       ;# of bytes in chain data area
3$:      MOV (R1)+,(R2)+         ;Move command lines into chain area
        CMP R1,#COMEND          ;Reached end yet?
        BLO 3$                  ;Repeat if not
        BIS #CHNIF$,@#JSW       ;Set pass command bit in JSW
                                   ;(aborts any pending command file)
        CLR RO                  ;Required for special exit
        MOV #1000,SP            ;Reset stack pointer
10$:     .EXIT                   ;Done (Note: unit remains allocated
                                   ; after program exit!)
; EMT arg blocks and word buffers
AREA:    .BLKW 10               ;General purpose EMT arg block
CLFLAG:  .WORD 10               ;WOLFOUT flag for CL .SPFUN
ATTCL:   .BYTE 0,155            ;EMT arg block to take over TS line by CL
        .WORD 0                 ;Selected CL unit number
        .WORD 6                 ;Selected TS line number
ALOCAT:  .BYTE 0,156            ;EMT arg block to allocate a device
        .WORD CLUNAM            ;Address of 4-word device specification
CLUNAM:  .RAD50 /CLO/           ;Start with first CL unit
        .WORD 0,0,0            ;Dummy file specification
; General messages and byte buffers
        .MLIST BEX
COMAND:  .ASCIZ /ASSIGN CLO XL/ ;Make logical assignment of XL for VTCOM
        .ASCIZ /R VTCOM/       ;Run VTCOM (part of RT11 V5.01 kit)
COMEND:
        .LIST BEX
        .EVEN
COMLEN = COMEND-COMAND          ;# bytes in commands passed to KMON
        .IF GT <COMLEN-<1000-512>>
        .ERROR 1               ; Chain data area overflow
        .ENDC
ATTERR:  .WORD 0                ;Attach CL EMT error message table
        .WORD NOPRIV            ; 1
        .WORD BADCLU            ; 2
        .WORD BADTSL            ; 3
        .WORD ALRDCL            ; 4
        .WORD INUSE             ; 5
        .WORD CLBUSY            ; 6
        .MLIST BEX
NOPRIV:  .ASCIZ /Not privileged to use this program./
BADCLU:  .ASCIZ /Attempt to use invalid CL unit./
BADTSL:  .ASCIZ /Attempt to use invalid time-sharing line./
ALRDCL:  .ASCIZ /Time-sharing line already in use as CL unit./
INUSE:   .ASCIZ /Somebody is already using time-sharing line./
CLBUSY:  .ASCIZ /CL unit already active./
        .LIST BEX
        .EVEN
ALLERR:  .WORD 0                ;Allocate EMT error table
        .WORD ALRDAL            ; 1
        .WORD BADDEV            ; 2
        .WORD ALLFUL            ; 3
        .WORD DEVUSE            ; 4
        .MLIST BEX
ALRDAL:  .ASCIZ /CL unit already allocated by someone else./
BADDEV:  .ASCIZ /Cannot allocate that device./
ALLFUL:  .ASCIZ /Too many devices already allocated./
DEVUSE:  .ASCIZ /Device in use by someone else./
        .LIST BEX
        .EVEN
        .END START

```

4.53 Clearing and Resetting a CL Unit (1/2,155)

The following system services perform the functions of clearing and resetting a CL unit. They are equivalent to the special functions to clear and reset a CL unit (.SPFUNs 201 and 265 respectively). The advantage of

performing these functions with EMTs is that, unlike .SPFUNs, no channel has to be opened to the specified CL unit.

Using the EMT forms rather than the SPFUN forms can be particularly useful when dealing with busy spooled CL units connected to a printer or other device which needs to be cleared. If the device is busy enough that all system spool file control blocks are in use, then the SPFUN method cannot be used. This is because the job will hang when the attempt is made to open a channel to the spooled device, so that the special function may be issued. This becomes a deadlock condition when the device cannot proceed because of an XOFF condition.

Another application of these EMTs is allow another job to clear a terminal/CL cross connection (SET HOST) which has received an XOFF and the terminal output buffer is full. When this happens, no more characters can be accepted from the keyboard, including the keystrokes necessary to either break the cross connection (^\) or to clear the XOFF condition (^AR). As with jobs using VTCOM, cross connected lines should allocate the CL unit to prevent communication conflicts. This allocation ordinarily prevents any other jobs from resetting the CL unit, although a subprocess of the cross connected job would not suffer from allocation conflict. However, because of the very low processing level at which the cross connection is serviced (for speed), the job cannot switch to a sub-process. Thus, the job cannot proceed or clear itself. Since these EMTs do not open a channel to the device, normal device allocation checking is bypassed. However, as a measure of protection to prevent hostile use of these functions, they have been implemented to require both TERMINAL and BYPASS privileges.

The form of both EMTs is:

EMT 375

with R0 pointing to an argument block of the form:

```
.BYTE    n,155
.WORD    cl-unit-number
```

where *n* is 1 to clear the CL unit (clear XOFF received flag and transmit an XON) or 2 to reset the CL unit (empty the input silo and output ring buffer, stop sending any break, clear XOFF received flag, send an XON, clear end-of-file status, and reset line and column numbers); *cl-unit-number* specifies the CL or C1 unit number – CL0 is 0, CL3 is 3, C10 is 8, C17 is 15, etc.

These EMTs return the following error codes:

Error Code	Meaning
0	Invalid EMT subfunction code
1	Job issuing request does not have TERMINAL privilege
2	Invalid CL unit number specified
7	Specified CL unit is not assigned to a line
8	Job issuing request does not have BYPASS privilege

4.54 Allocating a device for exclusive use (156)

Devices may be allocated for exclusive use by a single user. This prevents mixing input and output on common, but non-spooled, devices like a communications line (XL or CL devices) or a magnetic tape. Access restriction by device allocation remains in effect until deallocated by the job or until the job logs off. See the description of the ALLOCATE command in the *TSX-Plus User's Reference Manual* for more information on device allocation.

There are three system service calls relating to device allocation: allocate a device; deallocate a device; check to see if a device is allocated by another user. Use of these EMTs requires ALLOCATE privilege. The form of the EMT is the same for all three:

EMT 375

with R0 pointing to an argument block of the form:

```
.BYTE    n,156
.WORD    device-pointer
```

where *device-pointer* is the address of a four-word block in which the first word contains the RAD50 name of the device to be allocated and the next three words contain zeros. The specific function is defined by the first byte of the argument block, as follows:

<i>n</i>	Function
0	Allocate a device
1	Deallocate a device
2	Check to see if a device is allocated by another user

You can only allocate a device if no other user has already allocated the device or has a channel open to it. If the device is allocated to another job or another job has a channel open to the device, then the number of the job which is accessing the device is returned in R0. If a job which has already allocated the device or has a channel open to the device is not associated with the same primary line as the job attempting to allocate the device, then the carry bit will be set on return from the EMT and the number of the job which is accessing the device will be returned in R0. If the device is currently allocated by a job which was started from the same primary line as the job which is now attempting to allocate it, then the carry flag will be clear on return, but the other job number will be returned in R0. If both the job with the current allocation and the job attempting to allocate the device are subprocesses, the first one to allocate the device gets the allocation. If either is the primary line, then the primary line will get the allocation.

<i>Error Code</i>	<i>Meaning</i>
1	Device is already allocated by another job
2	Invalid device specified
3	Device allocation table is full (TSGEN parameter MAXALC)
4	Device is currently in use by another job
5	Job does not have ALLOCATE privilege (subfunction to check device allocation does not require privilege)

Example

See the example program GETCL in section 4.52 on assigning a CL unit to a time-sharing line.

4.55 Job monitoring (157)

A monitoring watch may be established to allow a job to monitor the status of other time-sharing jobs. For example, if a detached job were being used as a common file server for several other jobs, it would be useful to know if a served program with a pending request aborts. The outstanding request could then be purged. The monitoring facility may also be used in lieu of message channels (see Chapter 6) when only a small amount of information needs to be communicated (e.g., one word).

When a job is being monitored, the operating system will report certain job status changes, such as logging on, starting or exiting a program, or logging off. In addition, the monitored job itself may issue status reports to any job which may be monitoring it. After a job has established a monitor watch for a given line, a completion routine is entered in the monitoring job whenever a monitor status report is issued for the

monitored line, whether the report was issued by the monitored job itself or by the system because of a status change. Job monitoring may be used effectively in conjunction with inter-job message communications and with detached jobs.

There are three system service calls related to job monitoring. All have the form:

EMT 375

with R0 pointing to an argument block of the form:

```
.BYTE    n,157
.WORD    job-number
.WORD    completion-routine
```

where *n* designates which job monitoring function is to be performed, *job-number* designates the line number which is to be monitored, and *completion-routine* is the address of a completion routine in the monitoring job which is to be entered whenever a monitor status report is generated for the line being monitored (*job-number*). The *job-number* may designate any primary time-sharing line, detached job or subprocess job number. It may not refer to a dedicated CL line. Job-numbers (lines) are assigned in the following order: time-sharing lines in the order they were declared during system generation, starting with number 1; detached job lines; subprocess lines. Detached jobs and subprocesses are assigned to line numbers in the order they were activated. Unused detached lines are reserved and are never assigned to subprocesses.

4.55.1 Establishing a monitoring connection

The form of the argument block to establish a monitoring connection with a line is:

```
.BYTE    0,157
.WORD    job-number
.WORD    completion-routine
```

If *job-number* is zero, then all jobs are monitored, including those not currently logged on.

Error Code	Meaning
1	Invalid job number specified
2	No free job monitoring control blocks (increase TSGEN parameter MAXMON)

The specified completion routine will be entered whenever a monitor status report is issued for the specified job-number. On entry to the completion routine, the low order byte of R0 contains the line number of the job originating the monitor status report. Note that the same completion routine may be specified when monitoring more than one job. The high order bit (mask 100000) of R0 will be clear (0) if the monitor status was originated by the system and will be set if the monitored job itself issued the status. R1 will contain a 16-bit status value. Status values generated by the system for monitored lines are:

Status Code	Meaning
1	Job has been initialized
2	Job has logged on using the LOGON program
3	Job has started running a program
4	Job has returned control to the keyboard monitor
5	Job has logged off

Status code 1 is generated when a logged off line is started, usually by typing a carriage-return at that terminal. Status code 3 is generated whenever a program is either run or chained to. Status code 4 is generated whenever a program exits or is aborted, but not when it chains to another program.

4.55.2 Cancel a monitoring connection

The form of the argument block to cancel a monitoring connection with a line is:

```
.BYTE 1,157
.WORD job-number
```

If *job-number* is zero (0), then all job monitoring connections established by the monitoring job (the job issuing this EMT) are cancelled. All job monitoring requests are also cancelled if the job exits, aborts, chains or issues a .SRESET or .HRESET.

Error Code	Meaning
1	Invalid job number specified

4.55.3 Broadcast status report to monitoring jobs

The form of the argument block to broadcast a monitor status report is:

```
.BYTE 2,157
.WORD status-value
```

where *status-value* is a 16-bit value to be broadcast to all jobs which are monitoring the job which broadcasts the status-value (the job issuing this EMT). On entry to the completion routine in the monitoring job(s), the status-value is in R1, the low byte of R0 contains the job-number of the job broadcasting the status-value, and the high-order bit of R0 (mask 100000) is set (1).

Error Code	Meaning
0	No jobs are monitoring this line

Example

```
.TITLE MONCPL
.ENABL LC
; Demonstrate job monitoring completion routines.
; Watch a dial-in line and announce when it is activated.
.MCALL .PRINT,.EXIT,.SPND,.RSUM
.DSABL GBL
.GLOBL R50ASC
ERRBYT = 52          ;EMT error byte address
MONLIN = 8.          ;Line number to be monitored
NFYLIN = 1.          ;Line number to be notified
START: MOV #MONJOB,R0 ;Point to EMT arg block to
      EMT 375         ;Schedule job monitor compl rtn
      BCC 1$         ;Branch if OK
; Job monitoring scheduling error
      MOVB @ERRBYT,R1 ;Get EMT error code
      ADD #'0,R1      ;Convert to ASCII
      MOVB R1,ERRCOD  ;Stuff into error message
4$: .PRINT #CPLERR    ;Report error
      .EXIT          ;And abort
; Wait for event on monitored line
1$: .SPND             ;Wait for event
      BR 4$          ;Only get here on error
```

```

; Completion routine which is entered when event occurs on monitored line
MONCPL: MOV    RO,SENDER      ;Remember whose monitor report
        MOV    #MONJOB,RO    ;Point to EMT arg block to
        EMT    375           ;Reschedule myself
        BCC    1$           ;Branch if OK
        MOVB   @#ERRBYT,RO    ;Get error code
        ADD    #'0,RO        ;Convert to ASCII
        MOVB   RO,ERRCOD     ;Save error code
        .RSUM              ;Restart mainline and report error
        BR     3$           ;Abort completion routine
1$:     TST     SENDER        ;Was this a system generated message?
        BMI    3$           ;Ignore if not
        CMP    R1,#3         ;Running program?
        BNE    2$           ;Branch if not
        MOV    #GPRGMM,RO    ;Point to EMT arg block to
        EMT    375           ;Get program name
; Note that a short program may already have returned to KNOWN by now!
        MOV    #R5OBLK,RO    ;Point to program name
        CALL   R5OASC        ;Convert into ASCII in message
2$:     DEC     R1            ;0 index status code
        MUL    #CODLEN,R1    ;Convert status code to message offset
        ADD    #CODBEG,R1    ;Convert to message address
        MOV    R1,MSADR      ;Point to correct status message
        MOV    #NOTIFY,RO    ;Point to EMT arg block to
        EMT    375           ;Send notification to operator
3$:     RETURN
; EMT argument blocks and word buffers
MONJOB: .BYTE  0,157        ;EMT arg block to monitor a line
        .WORD  MONLIN       ;Line (job) number to be monitored
        .WORD  MONCPL       ;Address of completion routine
NOTIFY: .BYTE  0,127        ;EMT arg block to send text
        .WORD  NFYLIN       ;Line number to be notified
MSADR:  .WORD  0            ;Address of message to be sent
GPRGMM: .BYTE  0,144        ;EMT arg block to get program name
        .BYTE  MONLIN,5     ;Line number, subfunction
        .WORD  R5ONAM       ;Address to put RAD50 name
R5OBLK: .WORD  R5ONAM       ;Address of input buffer (RAD50)
        .WORD  PRGNAM       ;Address of output buffer (ASCII)
        .WORD  6            ;Number of chars to convert
R5ONAM: .WORD  0,0          ;RAD50 value of program name
SENDER: .WORD  0            ;Copy of sending infor
; Text and byte buffers
        .MLIST BEX
AC      = .
CODBEG: .ASCIZ  <15><12><7>/Job has been initialized      /
CODLEN  = . - AC
        .ASCIZ  <15><12><7>/Job has completed LOGON      /
        .ASCII  <15><12><7>/Job executing program /
PRGNAM: .ASCIZ  /PRGNAM /
        .ASCIZ  <15><12><7>/Job returned to KNOWN        /
        .ASCIZ  <15><12><7>/Job logged off                /
CPLERR: .ASCII  /Job monitoring completion routine error /
ERRCOD: .ASCIZ  /0/
        .END    START

```

4.56 Acquiring another job's file context (160)

One job may acquire the file context of another job. This is principally intended for detached jobs (such as RTSORT) which operate as file servers, so that they may access files for the job they are servicing even if those files are on logical subset disks or are access restricted.

The following EMT may be used to acquire another job's file context:

```
EMT    375
```

with R0 pointing to an argument block of the following form:

```
.BYTE    0,160
.WORD    job-number
.WORD    0
```

where *job-number* is the number of the job whose file context is to be acquired.

The job which issues this EMT must have GETCXT privilege except in the case where *job-number* specifies the parent of the job issuing the EMT.

On successful return from the EMT, the following actions have been taken:

1. All channels for the issuing job which are opened to files on logical disks are purged.
2. All devices mounted by the issuing job are dismounted.
3. The following items are copied from the target job, replacing the previous information for the issuing job:
 - ASSIGN commands
 - ACCESS command restrictions
 - Logical disk information
 - Mounted device information

See the *TSX-Plus User's Reference Manual* for more information on detached jobs. See the *TSX-Plus System Manager's Guide* for more information on GETCXT privilege.

Error Code	Meaning
0	Issuing job is not privileged to use this EMT
1	Target job number is invalid or is not logged on

Example

```
.TITLE  GETCXT
.ENABLE LC
; Demonstrate EMT to get context of another job.
.MCALL  .GTILN,.EXIT,.PRINT,.ENTER,.CSTAT,.PURGE,.TTYOUT
.DSABL  GBL
.GLOBL  ACRDEC,PRTR50
ERRBYT  = 52 ;EMT error byte address
BS      = 10 ;ASCII backspace BS
START:  .GTILN #BUFFER,#PRMPT1 ;Prompt for line # to check
        MOV   #BUFFER,R1      ;Get pointer to line # string
        CALL  ACRDEC          ;Convert to number
        MOV   R0,JOBNUM       ;Put line number in EMT arg block
        MOV   #GETCXT,R0      ;Point to EMT arg block to
        EMT   375             ;Get context of another job
        BCC   1$              ;Branch if no error
        TSTB  @ERRBYT         ;Privilege or no job?
        BGT   2$              ;Branch if requested line not logged on
        .PRINT #NOPRIV        ;No privilege for this EMT
        BR    9$              ;Quit
2$:      .PRINT #NOTON         ;That job not logged on
        BR    9$              ;Quit
1$:      MOV   #~RSY,DEVNAM    ;Locate SY first
```

```

        CALL    SHODEV          ;Determine and display its device
        MOV     #~RDK,DEVNAM    ;Same for DK
        CALL    SHODEV
9$:      .EXIT
AREA:    .BLKW    10            ;General purpose EMT arg block
GETCXT:  .BYTE    0,160        ;EMT arg block to get job context
JOBNUM:  .WORD    0            ;Line number of job in question
        .WORD    0            ;Required
STATUS:  .WORD    0,0,0,0,0,0  ;.CSTAT channel status result
DEVNAM:  .RAD50    /DK TEMP FIL/ ;Name of temporary file
        .NLIST    BEX
BUFFER:  .BLKB    81.
PRMPT1:  .ASCII    /Get context of line number - /<200>
NOPRIV:  .ASCIZ    /Sorry, you are not privileged to run this program./
NOTON:   .ASCIZ    /Sorry, that job is not currently logged on./
ARROW:   .ASCII    / --/<76><40><200>
COLRTN:  .ASCIZ    /:/
CRLF:    .ASCIZ    //
        .LIST    BEX
        .EVEN
; Let system translate device assignments by opening
; a temporary file and use the .CSTAT EMT.
SHODEV:  .ENTER    #AREA,#0,#DEVNAM,#1 ;Try to open temporary file
        BCS     9$            ;Give up on error
        .CSTAT   #AREA,#0,#STATUS ;Try to get channel info on file
        BCS     9$            ;Give up on error
        MOV      DEVNAM,R0      ;Recover logical device name
        CALL     PRTR50        ;Display it
        .PRINT   #ARROW        ;" --> "
        MOV      STATUS+12,R0   ;Recover physical device name
        CALL     PRTR50        ;Display it
        .TTYOUT  #BBS          ;Back up over empty unit number
        MOV      STATUS+10,R0   ;Recover physical unit number
        ADD      #'0,R0        ;Convert to ASCII
        .TTYOUT  ;Display it
        .PRINT   #COLRTN       ;Format
9$:      .PURGE   #0            ;Throw away temp file
        RETURN
        .END     START

```

4.57 Manipulating process windows (161)

The TSX-Plus process windowing facility allows the system to remember the contents and status of the terminal screen display and to redisplay windows as you switch between processes or on demand by programs.

The process windowing facility also provides a *print window* function which allows you to print the contents of a window on a printer by typing a control character or by use of an EMT. Windows are only allowed on VT200, VT100, and VT52 series terminals. The contents and status of each window is stored in a named global region and you must have SYSGBL privilege in order to use process windows. See the *TSX-Plus User's Reference Manual* for more information on the use of windows and window related commands (SET WINDOW, SET PRINTWINDOW).

Each job may have up to 26 windows active at one time. A window is identified by an ID number in the range 1 to 26. Two jobs may have windows with the same ID number without conflict.

When windowing is turned on, the system monitors all characters sent to the terminal and maintains an updated screen image in memory. Terminal attributes such as line width, reverse/normal video, application keypad mode, etc. are saved along with line attributes (double wide, double high), and character attributes. The attributes retained for each character consist of blinking, bold, underlined, reverse video, and character set information (ascii, U.K. national, DEC supplemental, or graphics—line drawing).

The most common use of windows is to completely refresh the display when switching among subprocesses to avoid the confusion of mixed displays. In addition, a keyboard command can be used to send a copy of the current window to a printer.

For some special program applications, it may be useful to utilize multiple windows from the same job. System service calls (EMTs) are provided to create, select the current, delete, and print process windows.

4.57.1 Creating a window

The system service call (EMT) used to create a window has the form:

EMT 375

with R0 pointing to an argument block of the following form:

```
.BYTE 0,161
.BYTE window-id,perm-flag
.BYTE window-width,max-scroll
.BYTE copy-id,copy-job
.WORD 0
```

where *window-id* is a window identification number in the range 1 to 26 which identifies the particular job window. The global memory regions created for windows have names of the form *WINjji* where *jj* is the job number and *i* is a letter which corresponds to the window ID (*A*=1,...,*Z*=26). Different jobs may have windows with the same ID without conflict.

Perm-flag should be either 0 or 1. If it is zero (0), the window is temporary and will be automatically deleted when the program exits to the keyboard monitor (other than doing a chain). If *perm-flag* is 1, the window is permanent and is only deleted when explicitly requested (by the delete-window EMT, or by SET WINDOW OFF) or when the job logs off.

Window-width is the width of the window in columns and should not exceed 132. (VT52 may use only 80.)

Max-scroll is the maximum number of lines which are allowed to scroll off the window during a time when the window is disconnected from the terminal because a different subprocess has been selected. A value of zero (0) may be specified to disable any scrolling. If a value of 255 is specified, an unlimited amount of scrolling is allowed.

The *copy-id* and *copy-job* parameters can be used to cause the system to copy certain window information from another window as it is initializing the new window. If *copy-id* and *copy-job* are both zero, no information is copied from another window and the supplied values (or default values) are used for the new window. If *copy-id* is non-zero, it is used as the id of the window from which the information is to be copied. If *copy-job* is non-zero it is the number of the job that owns the window from which the information is copied. If *copy-job* is zero, the current job is assumed. The following information is (or is not) copied:

1. The number of columns per line.
2. The maximum allowed number of lines to be scrolled while switched to a different process.
3. 80 or 132 column mode.
4. Light or Dark (normal or reverse video) mode.
5. Character set mapping is NOT copied.

Error Code	Meaning
0	Window management not included in system generation or invalid EMT argument block.
1	Maximum allowed number of windows are already in use. Increase value of MAXWIN sysgen parameter.
2	Unable to create global memory region for window. Job may not have SYSGBL privilege or you may need to increase the value of the NGR sysgen parameter, or there may be insufficient memory space available.

Creating a window allocates a window control block and a global memory region for the window. The window contents are set to all blank. Default language specification is *not* an option. If it is necessary to select the National Replacement Character set, the appropriate terminal control sequence must be sent to the terminal after window creation. See the SET WINDOW/LANGUAGE command in the *TSX-Plus User's Reference Manual*.

Example

```
.TITLE WINDOW
; Demonstrate use of multiple process windows.
.DSABL GBL
.ENABL LC
.MCALL .PRINT,.EXIT,.TWAIT
ERRBYT = 52 ;EMT error code address
; Create and initialize contents of 10 windows
START: MOV #10.,R1 ;Init window number
1$: MOV R1,WINID1 ;Point to next window
MOV #MAKWIN,RO ;Point to EMT arg block to
EMT 375 ;Create a window
BCS MAKERR ;Branch on error
MOVB R1,WINID2 ;Point to next window
MOV #SELWIN,RO ;Point to EMT arg block to
EMT 375 ;Select a window
BCS SELERR ;Branch on error
MOV R1,R2 ;Copy window number
DEC R2 ;0 index
ASL R2 ;Convert to word index
.PRINT WINTXT(R2) ;Write something to each screen
SOB R1,1$ ;Repeat through ten windows
; Cyle through the windows for pseudo-animation.
MOV #5,R2 ;Loop 5 times with all windows
2$: MOV #10.,R1 ;Init window number
22$: MOVB R1,WINID2 ;Point to next window
MOV #SELWIN,RO ;Point to EMT arg block to
EMT 375 ;Select next window
BCS SELERR ;Branch on error
.TWAIT #AREA,#TIME ;Delay a little bit
SUB NWINS,R1 ;Step to next window
BGT 22$ ;And repeat
CMP NWINS,#2 ;Already doing half?
BEQ 2$ ;Branch if so
SOB R2,2$ ;Repeat all windows 5 times
; Now delete half the windows
MOV #2,NWINS ;Only do half of them
MOV #9.,R1 ;Delete odd numbered windows
3$: MOVB R1,WINID3 ;Point to next window
MOV #DELWIN,RO ;Point to EMT arg block to
EMT 375 ;Delete a window
BCS DELERR ;Branch on error
SUB NWINS,R1 ;Point to next window
BGT 3$ ;Repeat through 5 windows
BR 2$ ;Repeat rest forever
MAKERR: MOV #METXT,RO ;Point to error message
BR DSPERR ;Go display error
SELERR: MOV #SETXT,RO ;Point to error message
BR DSPERR ;Go display error
DELERR: MOV #DETX,RO ;Point to error message
DSPERR: MOVB @ERRBYT,R1 ;Get error code
ADD #0,R1 ;Convert to digit
MOVB R1,EC1 ;Store in all
MOVB R1,EC2 ; output
MOVB R1,EC3 ; strings
.PRINT ;Display error message
.EXIT
MAKWIN: .BYTE 0,161 ;EMT arg block to make window
```

```

WINID1: .BYTE 0           ;Window id number
        .BYTE 0           ;Temporary window
        .BYTE 80.         ;Narrow window
        .BYTE 0           ;No scrolling restrictions
        .WORD 0
        .WORD 0
SELWIN: .BYTE 1,161       ;EMT arg block to select window
WINID2: .BYTE 0,0         ;Window id number,0
        .WORD 0
DELWIN: .BYTE 2,161       ;EMT arg block to delete window
WINID3: .BYTE 0,0         ;Window id number,0
TIME:   .WORD 0,18.       ;Time delay in ticks (arbitrary)
NWINS:  .WORD 1           ;Window id increment
AREA:   .BLKW 10          ;General EMT arg block
WINTXT: .WORD L1L,L2L,L3L,L4L,L5L,L5R,L4R,L3R,L2R,L1R
        .NLIST BEX
L1L:    .ASCII <12>        /      */<200>
L2L:    .ASCII <12><12>     /      */<200>
L3L:    .ASCII <12><12><12><12> /    */<200>
L4L:    .ASCII <12><12><12><12><12><12> /  */<200>
L5L:    .ASCII <12><12><12><12><12><12><12>/ */<200>
L5R:    .ASCII <12><12><12><12><12><12><12>/ */<200>
L4R:    .ASCII <12><12><12><12><12><12> /    */<200>
L3R:    .ASCII <12><12><12><12> /      */<200>
L2R:    .ASCII <12><12> /        */<200>
L1R:    .ASCII <12> /          */<200>
NETXT:  .ASCII /Window creation error #/
EC1:    .ASCIZ /0/
SETXT:  .ASCII /Window selection error #/
EC2:    .ASCIZ /0/
DETXT:  .ASCII /Window deletion error #/
EC3:    .ASCIZ /0/
        .LIST BEX
        .END START

```

4.57.2 Selecting a current window

When this EMT is executed, the terminal screen is cleared and the selected window is drawn on the screen. The EMT argument block used to select the current window has the form:

```

.BYTE 1,161
.BYTE window-id,0
.WORD 0

```

where *window-id* is the window identification number as specified when the window was created. A window-id value of zero (0) has a special meaning: it causes windowing to be disabled for the job but the contents of all existing windows are retained and may be reselected later.

Error Code	Meaning
0	Window management not included in system generation or invalid EMT argument block.
3	Unable to locate window with specified <i>window-id</i> .

4.57.3 Deleting windows

The EMT argument block used to delete a window has the following form:

```
.BYTE 2,161
.BYTE window-id,0
```

where *window-id* is the identification number of the window to be deleted. There are two special values for *window-id* which may be used with this EMT: a value of zero (0) causes all temporary windows for the job to be deleted; a value of 255 (decimal) causes all windows for the job, both temporary and permanent, to be deleted.

The global memory region used by a window is freed when the window is deleted.

<i>Error Code</i>	<i>Meaning</i>
0	Window management not included in system generation or invalid EMT argument block.
3	Unable to locate window with specified <i>window-id</i> .

4.57.4 Suspending window processing

Window processing may be suspended to allow characters to be sent to the terminal which are not processed by the window manager. If the job does not have a currently active window, the suspend and resume functions have no effect. The EMT 375 argument block to suspend window processing is:

```
.BYTE 3,161
```

4.57.5 Resuming window processing

Window processing which has been suspended may be resumed by EMT 375 with the following argument block:

```
.BYTE 4,161
```

4.57.6 Printing a window

The contents of a window may be printed by issuing EMT 375 with the following argument block:

```
.BYTE 5,161
.BYTE window-id,0
```

where *window-id* is the id number used when the window was created. The WINPRT program must be executing in order for the window to be printed.

<i>Error Code</i>	<i>Meaning</i>
0	Window management not included in system generation
3	Unable to locate window with specified <i>window-id</i>
4	WINPRT program not running

4.58 Switching Between Subprocesses (162)

The following EMT allows a running program to switch terminal control between subprocesses as if a CTRL-W-digit sequence had been typed. The EMT has two functions: it initiates new subprocesses, and switches terminal control between processes.

The form of the EMT is:

EMT 375

with R0 pointing to the following argument block:

```
.BYTE  sub-function,162
.BYTE  subprocess-number,0
.BYTE  return-process,initiate-only
.WORD  command-file-pointer
.WORD  0
```

If *sub-function* is 1 the EMT initiates a new subprocess and optionally switches terminal control to the subprocesses. Error code 5 is returned if the specified subprocess is already active. If *sub-function* is 2 the EMT switches terminal control between already-active subprocesses. Error code 6 is returned if the specified subprocess is not already active. If *sub-function* is 0 (zero), the EMT initiates a new subprocess if the specified subprocess is not already active (and switches to it if the *initiate-only* flag is 0 (zero)), or just switches control to the subprocess if it is already active.

Subprocess-number indicates the subprocess number and is analogous to the digit typed after CTRL-W when manually switching between subprocesses. The value must be in the range 0 to MAXSEC (a TSGEN parameter). If the number is too large, then the EMT returns with error code 2 and MAXSEC in R0. Specifying 0 (zero) for *subprocess-number* switches back to the primary process (just as typing CTRL-W-0). If a -1 (minus one) is specified for *subprocess-number*, the system locates the first unused subprocess number and uses it. The number of the selected subprocess is returned in R0 on completion of the EMT.

Return-process indicates which process is to be returned to when the process initiated by the EMT logs off. Specify 0 (zero) to cause return to the primary process; specify 1 to return control to the process which is executing the EMT to initiate the subprocess.

Initiate-only controls whether the terminal connection is switched to the subprocess. If *initiate-only* is 0 (zero), the subprocess is initiated (if it has not already been initiated) and terminal control is switched to the subprocess. If *initiate-only* is 1 the subprocess is initiated but terminal control remains with the current process.

Command-file-pointer is the address of an ASCIZ string containing the file specification of a command file to be executed when the subprocess is initiated. This command file executes without start-up privilege following any other start-up command file specified with the SET SUBPROCESS/FILE command. If *command-file-pointer* is 0 (zero) no command file is executed.

The *return-process*, *initiate-only*, and *command-file-pointer* parameters are only used when initiating a new subprocess. They are ignored if this EMT is used to switch between subprocesses that have already been initiated.

The following error codes can be returned by this EMT:

Error Code	Meaning
1	You are not authorized to use subprocesses
2	Specified subprocess number is too large
3	All of your subprocesses are already active
4	Insufficient memory space available for process
5	Specified subprocess has already been initiated
6	Specified subprocess has not been initiated

Example

```

        .TITLE  SUBPRC
;
;Demonstration of the use of the TSX-Plus EMT to initiate subprocesses
;
        .MCALL  .PRINT,.EXIT,.TWAIT
        .GLOBL  PRTDEC
        .DSABL  GBL

ERRBYT  = 52                ;EMT error byte address

START:  MOV     #SUBPRC,R0    ;Point to EMT arg. block to
2$:     EMT     375           ;Intiate subprocess
        BCC     9$           ;Branch if OK
        MOVB    @#ERRBYT,-(SP) ;Fetch EMT error code
        .PRINT  #ERRIS       ;"EMT error is:"
        MOV     (SP)+,R0      ;Retrieve error code
        CALL    PRTDEC        ;Display it
9$:     .EXIT                ;And exit
        .MLIST  BEX

SUBPRC: .BYTE    1,162        ;EMT arg. block to initiate subprocess
        .BYTE    1,0         ;number 1
        .BYTE    1,1         ;Return to starting process and init. only
        .WORD    STRTFL       ;Startup command file
        .WORD    0

ERRIS:  .ASCII   /?SUBPRC-F-EMT error is: /<200>
STRTFL: .ASCIZ   /SY:SUBPRC.COM/ ;File spec. for startup com file for subproc.
        .EVEN
        .END    START

```

4.59 Mounting Logical Disks (163)

The following EMT allows logical disks (LD's) to be mounted from within running programs just as if MOUNT command was used. The form of the EMT used to mount a logical disk is:

```
EMT      375
```

with R0 pointing to the following argument block:

```

        .BYTE    chan,163
        .BYTE    ld-unit,read-only-flag
        .WORD     file-name-pointer

```

where *chan* is the number of an I/O channel which must be closed at the time that the EMT is executed. This channel is used during EMT processing to access the file to which the LD unit is being associated. The channel will be closed on return from the EMT. *Ld-unit* is the binary value of the LD unit number that is being mounted; it must be in the range 0 (zero) to 7. *Read-only-flag* is 0 (zero) to allow both reads and writes to take place to the LD unit. Set *read-only-flag* to 1 to disallow writes to the LD unit. *File-name-pointer* is the address of a four-word block of data containing the RAD50 device name, file name, and extension, of the file with which the LD unit is to be associated. The specified LD unit must not be in use (associated with a file) at the time that this EMT is executed.

The following error codes can be returned by the EMT:

Error Code	Meaning
0	Channel provided is already open
1	Invalid logical disk unit number (must be 0-7)
2	Logical disk support not generated into system
3	Logical disk unit is already associated with a file
4	Invalid file specification (null device or file)
5	Invalid logical unit number nesting
6	Unable to open specified file

Example

```

        .TITLE  MNTLD
;
;Demonstration of the use of the TSX-Plus EMT to mount a logical disk.
;
        .MCALL  .PRINT,.EXIT,.TWAIT,.CLOSE
        .GLOBL  PRTDEC
        .DSABL  GBL

ERRBYT  = 52                ;EMT error byte address

START:  .CLOSE  #0           ;Insure channel is closed
        .CLOSE  #1           ;Insure channel is closed
        MOV     #MNTLD5,R0    ;Point to EMT arg. block to
        EMT     375           ;Mount LD5:
        BCS     1$            ;Branch if not OK
        MOV     #MNTLD6,R0    ;Point to EMT arg. block to
        EMT     375           ;Mount LD6:
        BCC     2$            ;Branch if OK
1$:     MOVB     @#ERRBYT,-(SP) ;Fetch EMT error code
        .PRINT  #ERRIS        ;"EMT error is:"
        MOV     (SP)+,R0       ;Retrieve error code
        CALL    PRTDEC         ;Display it
2$:     .CLOSE  #0           ;Close channel
        .CLOSE  #1           ;Close channel
        .EXIT

        .WLST   BEX

MNTLD5: .BYTE   0,163         ;EMT arg. block to
        .BYTE   5,0          ;Mount LD5:
        .WORD   LD5NAM

MNTLD6: .BYTE   1,163         ;EMT arg. block to
        .BYTE   6,0          ;Mount LD6:
        .WORD   LD6NAM

LD5NAM: .RAD50   /DU5JNOWRKDSK/ ;.RAD50 of file to mount as LD5:
LD6NAM: .RAD50   /DU5SYSMONDSK/ ;.RAD50 of file to mount as LD6:
ERRIS:  .ASCII  /?MNTLD-F-EMT error is: /<200>
        .EVEN
        .END    START

```


Chapter 5

Shared File Record Locking

TSX-Plus allows several programs to have the same file open simultaneously. In order to control access to such files, TSX-Plus provides system calls to *lock* shared files and records within shared files. Through the record locking facility a program may gain exclusive access to one or more blocks in a file by locking those blocks. Other users attempting to lock the same blocks will be denied access until the first user releases the locked blocks. The TSX-Plus shared file facility also provides data caching on blocks being read from shared files.

Note that shared file access protection is only meaningful for *cooperating* jobs requesting shared access. This scheme does not prevent other jobs from opening or writing to files if those jobs do not adhere to the file sharing protocol.

The usual protocol for updating a shared file being accessed by several users is as follows.

1. Open file.
2. Tell TSX-Plus that file is *shared*.
3. Lock all blocks in file which contain desired record.
4. Read locked blocks into memory.
5. Make update to record.
6. Write updated blocks to file.
7. Unlock blocks.
8. Repeat steps 3-7 as needed.
9. Close file.

Use of shared files and record locking requires RLOCK privilege.

Shared file and record locking information is not carried across .CHAIN requests, although the channels do remain open.

DIBOL record locking procedures

Subroutines to control record locking from within DIBOL programs are provided with TSX-Plus. These are discussed in Appendix D.

Record locking from other languages

Record locking may be interfaced to other languages with appropriate subroutine calls. Record locking under COBOL-Plus is built into the run-time library provided with COBOL-Plus. The remainder of this chapter describes the techniques used to control shared file access and record locking.

5.1 Opening a shared file

Before a file can be used with shared access it must be opened by using a standard .LOOKUP EMT. After the file has been successfully opened, the following EMT may be used to declare the file to be opened for shared access. The form of this EMT is:

```
EMT      375
```

with R0 pointing to the following argument area:

```
.BYTE    chan,125
.WORD    access-code
```

where *chan* is the number of the I/O channel open to the desired file and *access-code* is a value indicating the type of access protection desired for the file. The following access codes are recognized:

Code	Protection	Access
0	Exclusive	Input
1	Exclusive	Update
2	Protected	Input
3	Protected	Update
4	Shared	Input
5	Shared	Update

The access code specifies two things:

- the type of access that you intend to make to the file (input only or update) and
- the type of access that you are willing to grant to other users of the file.

There are three protection classes:

EXCLUSIVE access means that you demand exclusive access to the file and will allow no other users to access the file in any fashion (input or update).

PROTECTED access means that you will allow other users to open the file for input but wish to prohibit any other users from opening the file for update.

SHARED access means that you are willing to allow other users to open the file for both input and update access.

When this EMT is executed, TSX-Plus checks your specified protection mode and access type with that previously declared for the file by other users. If an access conflict arises because of your specified access characteristics an error code of 4 is returned for the EMT. If no access conflict is detected, your specified access code is saved with the file and will be used to check for conflicts with future shared access requests issued by other users.

Normally all files that are declared to TSX-Plus using this EMT are enabled for use of the data caching facility (see description below). However, in some cases it may be desirable to suppress data caching for

certain files. For example, sequential access files usually benefit little from data caching and enabling data caching for these files causes the data cache buffers to be used non-productively when they could be providing a better service for other types of files. To disable data caching for a file set bit 8 (octal 400) in the access-code word. When shared access is declared with bit 8 set, new data is not brought into the data cache when the file is read. However, if the data being read is already stored in the cache because of a read by another user, it is used. When data being written to a file is currently stored in the cache, the data in the cache is updated even if the file is declared to be non-cached.

It is possible to have several channels simultaneously open to different shared files. The exact number of channels that can be open to shared files and the total number of shared files that may be opened are specified when the TSX-Plus system is generated.

Once all access to a shared file is completed, the I/O channel should be closed using the standard .CLOSE or .PURGE EMTs. See the next section for information about saving the status of a channel that has been opened to a shared file.

The error codes that can be returned by this EMT are listed below:

Error Code	Meaning
0	Job does not have RLOCK privilege or system does not include shared-file support
1	Channel has not been opened to a file.
2	Too many channels opened to shared files.
3	Too many shared files open.
4	File protection-access conflict

Example

```

.TITLE SHARED
.ENABL LC
; This program cooperates with the example program (SHARE2) in the
; following section to demonstrate shared file access protection.
.MCALL .PRINT, .GTLLW, .TWAIT, .EXIT, .READW
.MCALL .LOOKUP, .CLOSE, .SAVESTATUS, .REOPEN, .PURGE
ERRBYT = 52          ;EMT error byte
EXUP = 1.            ;Shared file access code: Exclusive, Update
PRIN = 2.            ;Shared file access code: Protected, Input
BUFSIZ = 256.        ;Number of words in a disk block
START: .LOOKUP #AREA, #0, #SHR1 ;Open SHR1.DAT
      BCC 1$          ;Branch if OK
      .PRINT #LKPERR   ;Lookup error message
      .EXIT
1$:   MOV #EXUP, <SHRFIL+2> ;Set Exclusive, Update access
      MOV #SHRFIL, R0      ;Point to EMT arg block to
      EMT 375              ;Declare SHR1.DAT as a shared file
                          ;with Exclusive and Update access
      BCC 2$              ;Branch if sharing OK
      JMP EMTERR           ;Explain the error and quit
2$:   .READW #AREA, #0, #BUFFER, #BUFSIZ, #0 ;Read block 0 of SHR1.DAT
      BCC 3$              ;Branch if read OK
      .PRINT #RDWERR      ;Say there was a read error
      .EXIT
3$:   .PRINT #BUFFER       ;Print out the file (must have 0 or 200 byte)
      .SAVESTATUS #AREA, #0, #BLOK1 ;Save channel 0 status for reuse
      BCC 4$              ;Branch if savestatus OK
      .PRINT #SVSERR      ;Savestatus error message
      .EXIT
4$:   MOV #SAVSHR, R0      ;Point to EMT arg block to

```

```

EMT      375          ;Save shared file status
.PURGE   #0           ;Purge the channel for reuse
.LOOKUP  #AREA,#0,#SHR2 ;Open SHR2.DAT
BCC      5$           ;Branch if OK
.PRINT   #LKPER2      ;Say bad lookup on SHR2
.EXIT

5$:      MOV          #PRIN,<SHRFIL+2> ;Set Shared, Input access
        MOV          #SHRFIL,R0       ;Point to EMT arg block to
EMT      375          ;Declare SHR2.DAT as a shared file
BCC      6$           ;Branch on no error
JMP      EMTER2        ;Say error on SHR2 sharing

6$:      .READW       #AREA,#0,#BUFFER,#BUFSIZ,#0 ;Read block 0 of SHR2.DAT
BCC      7$           ;Branch if read OK
.PRINT   #RDWER2      ;Say read error on SHR2
.EXIT

7$:      .PRINT       #BUFFER          ;Print out the contents (1 line, null filled)
        .PRINT       #PROMPT          ;Say it's time to try companion program
        .WAIT        #AREA,#TIME       ;Wait 30 seconds to run other program
        .PURGE       #0               ;Now, release SHR2
        .GTLIN       #BUFFER,#PRMPT2 ;Wait for return from subprocess
;This job will be suspended for output while gone to subprocess
        .REOPEN      #AREA,#0,#BLOK1 ;And get SHR1 back
        .READW       #AREA,#0,#BUFFER,#BUFSIZ,#1 ;Read in second block of SHR1
        .PRINT       #BUFFER          ;And print it to prove status was saved
        .CLOSE       #0               ;Release SHR1
        .EXIT

EMTER2:  MOV          #SHR2NM,FILNUM ;Point to alternate file error
EMTERR:  MOV          @#ERRBYT,R0     ;Get the error type
        DEC          R0               ;Zero offset
        ASL          R0               ;Convert to word offset
        .PRINT       SHRERR(R0)      ;Print the appropriate error message
        .PRINT       FILNUM          ;And the file name
        .EXIT

AREA:    .BLKW        10              ;EMT arg block area
BLOK1:   .BLKW        5               ;Save status area for SHR1.DAT
FILNUM:  .WORD         SHR1NM         ;File name for error message
TIME:    .WORD         0,30.*60.      ;30.sec * 60.tics/sec
SHRERR:  .WORD         NOTOPN         ;Pointer to EMT error messages
        .WORD        XSSCHN
        .WORD        XSSFIL
        .WORD        AISCON
        .NLIST       BEX

SHR1:    .RAD50       /DK SHR1 DAT/   ;File descriptor for SHR1.DAT
SHR2:    .RAD50       /DK SHR2 DAT/   ;File descriptor for SHR2.DAT
SHRFIL:  .BYTE        0,125          ;EMT arg block to declare shared file
        .WORD        1               ;Exclusive Update access (GETS CHANGED)
SAVSHR:  .BYTE        0,122          ;EMT arg block to save shared file status
NOTOPN:  .ASCIZ       /Attempt to share unopened channel/<7><200>
XSSCHN:  .ASCIZ       /Too many channels opened to shared files/<7><200>
XSSFIL:  .ASCIZ       /Too many shared files open/<7><200>
AISCON:  .ASCIZ       /Attempt to protect already protected shared file/<7><200>
SHR1NM:  .ASCIZ       /: SHR1.DAT/
SHR2NM:  .ASCIZ       /: SHR2.DAT/
LKPERR:  .ASCIZ       /Lookup error for SHR1.DAT/<7>
LKPER2:  .ASCIZ       /Lookup error for SHR2.DAT/<7>
SVSERR:  .ASCIZ       /Error occurred attempting to save SHR1.DAT file status/<7>
RDWERR:  .ASCIZ       /Error occurred while reading SHR1.DAT/<7>
RDWER2:  .ASCIZ       /Error occurred while reading SHR2.DAT/<7>
PROMPT:  .ASCIZ       /Go to a subprocess and RUN SHARE2 which attempts /<15><12>
        .ASCIZ       /to share the same files (SHR1.DAT, SHR2.DAT)./<15><12>
        .ASCIZ       /Waiting 30 seconds . . . ./<7>
PRMPT2:  .ASCIZ       /When you have returned, hit RETURN to continue/<200>
BUFFER:  .BLKW        BUFSIZ
        .END         START

```

See also the example program SHARE2 in the section on saving the status of a shared file channel.

5.2 Saving the status of a shared file channel

A standard .SAVESTATUS EMT may be used to save the status of a shared file channel. If this is done, all blocks that are being held locked in the file remain locked until the channel is reopened and the blocks are unlocked (see below).

When using a single channel number to access several shared files it is convenient to initially do a .LOOKUP on each file, then declare the file to be shared (EMT above), and then do a .SAVESTATUS. The channel being used to access the set of files can then be switched from one file to another by doing a .PURGE followed by a .REOPEN. However, before doing the .PURGE, TSX-Plus must be told that you wish to save the shared-file status of the file, otherwise all locked blocks will be unlocked and the file will be removed from the shared-file list. The form of the EMT used to perform this function is:

```
EMT      375
```

with R0 pointing to the following argument block:

```
.BYTE    chan,122
```

where *chan* is the I/O channel number. The effect of this EMT is to suspend the connection between the shared file information table and the I/O channel. Any blocks that are currently locked in the file remain locked until the channel is reopened to the file (by using a standard .REOPEN EMT). After saving a shared file status, the channel may be freed by using a .PURGE EMT.

If the same file is opened more than once concurrently using different channels in the same program, and the channels are suspended by using the preceding EMT, the system reassociates the correct shared-file context with the channel when it is reopened by matching the channel number used with the .REOPEN EMT with that used with the suspend EMT. Thus if the same file is opened more than once using different channels in the same program and is used with record locking, it is important to use the same channel number when suspending the file and reopening it. It is all right to switch the channel number between suspending file access and reopening it if the file is only open on one channel at a time within the program.

Example

```
.TITLE  SHARE2
.ENABLE LC
; This program cooperates with the example program (SHARED) in the
; previous section to demonstrate saving of shared file status with
; the .SAVESTATUS EMT.
.MCALL .LOOKUP,.PRINT,.EXIT,.READW
.MCALL .CLOSE,.TWAIT

ERRBYT = 52      ;EMT error byte
BUFSIZ = 256.    ;Size of disk file block
START: .LOOKUP #AREA,#0,#SHR1 ;Try to open a file which is access locked
      BCC 1$      ;Branch if OK
      .PRINT #LKPERR ;Say couldn't get the file
      BR 3$      ;Go on to try second file
1$: MOV #SHRFIL,R0 ;Point to EMT arg block to
      EMT 375      ;Declare file for shared access
      BCC 2$      ;If got the file, branch to read it
      CALL EXPLER ;Else explain why
      .PRINT #INSHR1 ;Say we can't share SHR1
      .CLOSE #0    ;Release channel 0
      BR 3$      ;Go on to next file
2$: .READW #AREA,#0,#BUFFER,#BUFSIZ,#0 ;Try to read block 0 of SHR1
      .PRINT #BUFFER ;Display it for kicks
      .CLOSE #0    ;Done with SHR1 for the moment
3$: .LOOKUP #AREA,#0,#SHR2 ;Try to open SHR2
      BCC 4$      ;Branch if OK
      .PRINT #LKPERR2 ;Say we couldn't even open it
```

```

4$:   BR      6$           ;Go on to try SHR1 again
      MOV     $SHRFIL,RO   ;Point to EMT arg block to
      EMT     375          ;Declare file for shared access
                                ;Access Input, Update to show lockout
                                ; though we don't write in this example
      BCC     5$           ;Branch if we can share it
      CALL    EXPLER       ;Explain why not
      .PRINT  $AGAIN       ;Say we will try again later
      .TWAIT  $AREA,$TIME  ;Wait 5 seconds and
      BR      4$           ;Try again
5$:   .READW  $AREA,$0,$BUFFER,$BUFSIZ,$0 ;Read block 0 of SHR2
      .PRINT  $BUFFER       ;Prove that we got it
      .CLOSE  $0            ;Done with SHR2
6$:   .LOOKUP $AREA,$0,$SHR1 ;Try SHR1 again
      BCC     7$           ;Branch if it worked
      .PRINT  $LKPERR       ;Say we couldn't do it
      .EXIT
7$:   MOV     $SHRFIL,RO   ;Point to EMT arg block to
      EMT     375          ;Declare shared file
      BCC     10$          ;Branch if OK
      CALL    EXPLER       ;And explain the error
      .PRINT  $STLLOK       ;Say it was still locked
      BR      11$          ;And quit
10$:  .READW  $AREA,$0,$BUFFER,$BUFSIZ,$1 ;Read in block 1
      .PRINT  $BUFFER       ;And display it
11$:  .CLOSE  $0            ;Done with SHR1
      .EXIT
EXPLER: MOVB  @ERRBYT,RO   ;Find out why can't share it
      DEC     RO           ;Convert to zero index
      ASL     RO           ;Make into word offset
      .PRINT  SHRERR(RO)    ;Say why we couldn't get it
      RETURN
      .LIST  BEX
AREA:  .BLKW  10           ;EMT arg block
SHRFIL: .BYTE  0,125       ;EMT arg block to declare file shared
      .WORD  5             ;Access Shared, Update
SHRERR: .WORD  NOTOPW      ;Shared file error message pointers
      .WORD  XSSFCH
      .WORD  XSSFOP
      .WORD  AXSCON
TIME:  .WORD  0,5.*60.     ;5.sec * 60.tics/sec
SHR1:  .RAD50 /DK SHR1 DAT/ ;Input file #1 name
SHR2:  .RAD50 /DK SHR2 DAT/ ;Input file #2 name
NOTOPW: .ASCIZ /Cannot share unopened file/
XSSFCH: .ASCIZ /Too many channels opened to shared files/
XSSFOP: .ASCIZ /Too many shared files open/
AXSCON: .ASCIZ /Shared file protected by another job/
INSHR1: .ASCIZ /On first try at SHR1.DAT/
LKPERR: .ASCIZ /Unable to lookup SHR1.DAT/
LKPER2: .ASCIZ /Unable to lookup SHR2.DAT/
AGAIN:  .ASCIZ /Can't access SHR2.DAT, will try again in 5 seconds/
STLLOK: .ASCIZ /On second try at SHR1.DAT/
      .EVEN
BUFFER: .BLKW  BUFSIZ      ;Input read buffer
      .END  START

```

See also the example program SHARED in the section on opening a shared file.

5.3 Waiting for a locked block

The following EMT can be used to lock a specific block in a file. If the requested block is locked by another job, the requesting job will be suspended until the desired block becomes available. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following area:

```
.BYTE   chan,102
.WORD   block
```

where *chan* is the number of an I/O channel that has previously been declared to be open to a shared file and *block* is the number of the block in the file to be locked. Other blocks in the file which were previously locked remain locked. The maximum number of blocks which may be simultaneously held locked is specified when TSX-Plus is generated. A block number of -1 (octal 177777) can be used to request that all blocks in the file be locked. If several users request the same block, access will be granted sequentially in the order that the requests are received.

Error Code	Meaning
0	Job does not have RLOCK privilege or system does not include shared-file support
1	Channel is not open to a shared file
2	Request to lock too many blocks in file

Example

```
.TITLE LOCKW
.ENABL LC
; This program cooperates with the example program (LOCK) in the next
; section to demonstrate shared file record locking.
.MCALL .PRINT,.EXIT,.TWAIT,.LOOKUP,.READW,.CLOSE
ERRBYT = 52 ;EMT error code byte
BUFSIZ = 256 ;Words per disk block
START: .LOOKUP #AREA,#0,#SHR1 ;Open SHR1.DAT
      BCC 1$ ;Branch if OK
      .PRINT #LKPERR ;Say bad lookup
      .EXIT
1$: MOV #SHRFIL,R0 ;Point to EMT arg block to
   EMT 375 ;Declare shared file
   BCC 3$ ;Branch if OK
   MOVB @ERRBYT,R0 ;Get the error code
   DEC R0 ;Make zero index
   ASL R0 ;Convert to word offset
   .PRINT SHRERR(R0) ;Print the error message
2$: .CLOSE #0 ;Give back the channel
   .EXIT
3$: MOV #LOCKW,R0 ;Point to EMT arg block to
   EMT 375 ;Lock block 0 of SHR1.DAT
      ;(Job is suspended until block available
      ;to be locked)
   BCC 5$ ;Branch when block is ready
   CMPB @ERRBYT,#1 ;Which error?
   BHI 4$ ;Too many blocks locked?
   .PRINT #SFCNOP ;Wasn't open to shared file!
   BR 2$ ;Give up
4$: .PRINT #ISLKBL ;Too many locked blocks in file
      ;(Defined by MXLBLK parameter in TSGEN)
   BR 2$ ;Give up
5$: .PRINT #PROMPT ;Switch lines to attempt access
   .TWAIT #AREA,#SEC20 ;Wait 20 seconds before unlocking
   MOV #UNLOCK,R0 ;Point to EMT arg block to
   EMT 375 ;Unlock a single block
   BCC 6$ ;Branch if OK
   .PRINT #SFCNOP ;Wasn't shared file!
   BR 2$ ;Give up
```

```

G$:      .TWAIT  #AREA,#SEC10      ;Wait 10 seconds for companion
        MOV     #CKWSHR,R0        ;Point to EMT arg block to
        EMT     375                ;Check for writes to shared file
        BCC     8$                ;If none, wrap up
        .PRINT  #CHANGED          ;Say we have new data
        .READW  #AREA,#0,#BUFFER,#BUFSIZ,#0 ;Get block 0
        .PRINT  #BUFFER          ;Show current contents block 0
        .READW  #AREA,#0,#BUFFER,#BUFSIZ,#1 ;Get block 1
        .PRINT  #BUFFER          ;Show contents block 1
        BR      DONE
8$:      .PRINT  #NOCHNG          ;Say nothing has changed
DONE:    .CLOSE  #0                ;Free up channel
        .EXIT
        .NLIST  BEX
AREA:    .BLKW   10                ;EMT arg block area
SHRFIL:  .BYTE   0,125            ;EMT arg block to declare shared file
        .WORD   4                ;Access Shared, Input
LOCKW:   .BYTE   0,102           ;EMT arg block to lock shared file block
        .WORD   0                ;Block number to be locked
UNLOCK:  .BYTE   0,113           ;EMT arg block to unlock shared file block
        .WORD   0                ;Block number to be unlocked
CKWSHR:  .BYTE   0,121           ;EMT arg block to check writes to shared file
SHRERR:  .WORD   NOTOPM          ;Shared file error message table
        .WORD   XSSFCH
        .WORD   XSSFOP
        .WORD   AXSCON
SHR1:    .RAD50  /DK SHR1 DAT/    ;File name to be shared
SEC20:   .WORD   0,20.*60.        ;20.sec * 60.tics/sec
SEC10:   .WORD   0,10.*60.        ;10.sec * 60.tics/sec
NOTOPM:  .ASCIZ  /Channel not opened to shared file/<7>
XSSFCH:  .ASCIZ  /Too many channels opened to shared files/<7>
XSSFOP:  .ASCIZ  /Too many shared files open/<7>
AXSCON:  .ASCIZ  /File protection access conflict/<7>
LKPERR:  .ASCIZ  /Unable to open SHR1.DAT/
SFCNOP:  .ASCIZ  /Can't lock or unlock block not open to shared file/
XSLKBL:  .ASCIZ  /Can't lock so many blocks in one file/
CHANGED: .ASCIZ  /Data has been written to file. Contents follow:/
NOCHNG:  .ASCIZ  /Data in file is unchanged/
PROMPT:  .ASCIZ  /Block 0 in SHR1.DAT will remain locked for 20 sec/<15><12>
        .ASCIZ  /Go to another line and RUN TLOCK to test it/<7>
        .EVEN
BUFFER:  .BLKW   BUFSIZ
        .END    START

```

5.4 Trying to lock a block

This EMT is similar in operation to the previous EMT—it is also used to request that file blocks be locked. The difference is that if the requested block is already locked by another user the previous EMT suspends the requesting program whereas this EMT does not suspend the program but rather returns an error code. As above, a request to lock block #-1 is treated as a request to lock the entire file. If the block is available it is locked for the requesting user and no error is reported. The form of this EMT is:

```
EMT      375
```

with R0 pointing to the following argument area:

```

.BYTE   chan,103
.WORD   block

```

where *chan* is the number of the I/O channel associated with the file and *block* is the number of the block which is to be locked.

Error Code	Meaning
0	Job does not have RLOCK privilege or system does not include shared-file support
1	Channel is not open to a shared file.
2	Request to lock too many blocks in file.
3	Requested block is locked by another user.
4	Entire file is locked by another user.

Example

```

        .TITLE LOCK
        .ENABL LC
; This program cooperates with the example program (LOCKW) in the
; previous section to demonstrate shared file record locking.
        .MCALL .PRINT,.EXIT,.WRITW,.TWAIT,.LOOKUP,.CLOSE
ERRBYT = 52                ;Error byte address
START:  .LOOKUP #AREA,#0,#SHR1 ;Try to open SHR1.DAT
        BCC 1$              ;Branch if OK
        .PRINT #LKPERR      ;Say we couldn't open
        .EXIT
1$:     MOV #SHRFIL,RO        ;Point to EMT arg block to
        EMT 375              ;Declare shared file
        BCC 3$              ;Branch if OK
        MOVB @#ERRBYT,RO     ;Get error type
        DEC RO               ;Convert to zero index
        ASL RO               ;Make into word offset
        .PRINT SHRERR(RO)    ;Display the error type
2$:     .CLOSE #0            ;Release the channel
        .EXIT               ;And give up
3$:     MOV #LOCK,RO         ;Point to EMT arg block to
        EMT 375              ;Try to unlock block 0
        BCC 6$              ;Branch if OK
        CMPB @#ERRBYT,#2     ;Which error was it
        BLO 4$              ;Wasn't open to share file?
        BEQ 5$              ;Request to open too many blocks in file?
        .PRINT #WAITNG      ;Block locked by another user
        .TWAIT #AREA,#TIME   ;Wait 3 seconds
        BR 3$               ;And try again
4$:     .PRINT #NOPNSF       ;Not open to shared file
        BR 2$               ;Give up
5$:     .PRINT #XSLKBL       ;Too many blocks locked in file
        BR 2$               ;Give up
6$:     .WRITW #AREA,#0,#BUFFER,#BUFSIZ,#0 ;Rewrite block 0
        MOV #UNLALL,RO       ;Point to EMT arg block to
        EMT 375              ;Release all blocks locked by this program
        .PRINT #GOBACK       ;Message: done, go back to original line
        BR 2$               ;Done
        .NLIST BEX
AREA:   .BLKW 10              ;EMT arg block
SHR1:   .RAD50 /DK SHR1 DAT/ ;Name of shared file
SHRFIL: .BYTE 0,125          ;EMT arg block to share file on chan 0
        .WORD 3              ;Access Protected, Update
LOCK:   .BYTE 0,103          ;EMT arg block to lock block on chan 0
        .WORD 0              ;number of block to be locked
UNLALL: .BYTE 0,101          ;EMT arg block to unlock all blocks on chan 0
        ;(Only applies to blocks locked by this job)
TIME:   .WORD 0,3.*60.       ;3.sec * 60.tics/sec
SHRERR: .WORD SFCNOP          ;File sharing EMT error table
        .WORD XSSFCW
        .WORD XSSFOP
        .WORD AXSCON

```

```

SFCNOP: .ASCIZ  /Channel not open to file/<7>
XSSFCM: .ASCIZ  /Too many channels open to shared files/<7>
XSSFOP: .ASCIZ  /Too many shared files open/<7>
AXSCON: .ASCIZ  /Shared file access conflict/<7>
LKPERR: .ASCIZ  /Couldn't open SHR1.DAT/<7>
WAITNG: .ASCII  /Requested block not available for locking/<15><12>
        .ASCIZ  /Will try again in 3 seconds/
GOBACK: .ASCIZ  /Done, log off and go back to original line/
NOPNSF: .ASCIZ  /Channel not open to shared file/<7>
XSLKBL: .ASCIZ  /Attempt to lock too many blocks in file/<7>
        .EVEN
BUFFER: .ASCII  /(SHR1)This line was written by the program LOCK./
        .ASCIZ  /                                /<15><12>
BUFSIZ = <.-BUFFER+1>/2      ;Number of words to write
        .BYTE   0,0          ;Safety bumper
        .END     START

```

5.5 Unlocking a specific block

The following EMT is used to unlock a specific block in a file. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument block:

```

.BYTE   chan,113
.WORD   block-number

```

where *chan* is the number of the I/O channel opened to the shared file and *block-number* is the number of the block to be unlocked.

Error Code	Meaning
0	Job does not have RLOCK privilege or system does not include shared-file support
1	Specified channel not opened to a shared file

Example

See the example program LOCKW in the section on waiting for a locked block.

5.6 Unlocking all locked blocks in a file

The following EMT is used to unlock all blocks held locked in a file. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument area:

```
.BYTE   chan,101
```

where *chan* is the I/O channel number open to the shared file. When this EMT is executed all blocks previously locked by the user on the shared file are unlocked. Blocks locked by the user on other files are not released nor are blocks of the same file that are locked by other users.

<i>Error Code</i>	<i>Meaning</i>
0	Job does not have RLOCK privilege or system does not include shared-file support
1	Channel is not open to a shared file.

Example

See the example program LOCK in the section on trying to lock a block.

5.7 Checking for writes to a shared file

The following EMT can be used to determine if any other user has written to a shared file. The form of the EMT is:

EMT 375

with R0 pointing to the following argument block:

.BYTE chan,121

where *chan* is the I/O channel number opened to the shared file. If no other user has written to the file since the file was opened by the user issuing this EMT or since that last time this EMT was issued for the file, the carry-flag is clear on return from the EMT.

<i>Error Code</i>	<i>Meaning</i>
0	Job does not have RLOCK privilege or system does not include shared-file support or channel not opened to shared file
2	Some other job has written to file since last check

This EMT is useful when data from a shared file is being held in a program buffer. If no other user has written to the file, then the data is still valid. However, if the data in the file has been rewritten then it must be reread. The usual sequence of operations in this situation is to first lock the block whose data is in the program's buffer, then do the EMT to see if the file has been written to. If the file has not been modified the data in memory is valid and can be used, otherwise the block must be reread from the file.

Example

See the example program LOCKW in section 5.3 on waiting for a locked block.

5.8 Data caching

Data caching is a technique provided by TSX-Plus to speed access to files. When TSX-Plus is generated a certain number of 512-byte buffer areas may be set aside for data caching. These buffer areas are part of the resident system data area and are not associated with any particular job. There are two kinds of data caching: generalized data caching; and shared-file data caching. Both kinds may be used automatically with minimal intervention on the part of the programmer or operator. Generalized data caching applies to all files on MOUNTed devices, while shared-file data caching applies only to files which have been declared as shared files. Generally, only one of these types is selected during generation of a TSX-Plus system. The

following discussion applies to shared-file data caching. See the *TSX-Plus System Manager's Guide* for more information on data caching.

Each time a request is issued to read a shared file, a check is made to see if the blocks being read are currently stored in the data cache. If so, the data is moved from the cache buffer to the program buffer and no disk I/O operations are performed. When data in the cache buffers is accessed, a use count is incremented. Periodically, the use counts for all buffers are divided by two. If the data blocks being read are not currently in the cache, the data is read from the disk into the program buffer and then it is moved into the cache buffers with the lowest use count.

When a write operation is done to a file that is being cached, a check is made to see if the data being written is currently stored in the cache. If so, the cache buffers are updated. In any case the data is written to the disk. In other words, this is a *write-through* cache; the disk file is always updated and caching does not improve the performance of *writes*.

All data files that are declared to TSX-Plus for shared access (using EMT 375 with function code 125) are eligible for data block caching regardless of their access protection type. Data caching on a shared file may be disabled by setting bit 8 (octal 400) in the access-code word of the EMT argument block when the file is declared for shared access. Data caching is particularly effective for COBOL-Plus ISAM files.

Chapter 6

Message Communications Facilities

TSX-Plus provides an optional facility that allows running programs to send messages to each other. This message communication facility allows programs to send messages through named channels, check to see if messages are pending, and suspend execution until a message is received. TSX-Plus provides EMTs for each of these operations which are described below. Use of named message channels requires MESSAGE privilege.

6.1 Message channels

Messages are transferred to and from programs by using TSX-Plus *Message Channels*. A message channel accepts a message from a sending program, stores the message in a queue associated with the channel and delivers the message to a receiving program when requested. Message channels are totally separate from I/O channels.

Each message channel is identified to the sending and receiving programs by a one to six character name. The total number of message channels is defined when TSX-Plus is generated. The names associated with the channels are defined dynamically by the running programs. A message channel is said to be *active* if any messages are being held in the queue associated with the channel or if any program is waiting for a message from the channel. When message channels become inactive they are released and may be reused.

Once a message is queued on a channel, that message will remain in the queue until some program receives it or the TSX-Plus system is halted. A program may exit after queuing a message without affecting the queued message. This allows one program to leave a message for another program that will run later.

6.2 Sending a message

The following EMT is used to queue a message on a named channel. If other messages are already pending on the channel, the new message is added to the end of the list of waiting messages. The sending program continues execution after the EMT and does not wait for the message to be accepted by a receiving program. During processing of the EMT the message is copied to an internal buffer, and the sending program is free to destroy its message on return from the EMT. The form of the EMT is:

EMT 375

with R0 pointing to the following argument area:

```
.BYTE 0,104
.WORD chnadr
.WORD msgadr
.WORD msgsiz
```

where *chnadr* is the address of a six byte field containing the name of the message channel (ASCII with trailing blanks if the name is less than six characters), *msgadr* is the address of the beginning of the message text, and *msgsiz* is the message length in bytes.

Error Code	Meaning
0	Job does not have MESSAGE privilege or system does not include message channel support.
1	All message channels are busy.
2	Maximum allowed number of messages already in message queues.
4	The transmitted message is too long. The message is truncated to maximum allowed length.
5	Maximum number of message requests pending.

The system manager may alter parameters during TSX-Plus generation to alleviate these error conditions.

Example

```

        .TITLE  SNDMSG
        .ENABL  LC
; Demonstrates use of the TSX-Plus EMT to queue a message to the interprocess
; message communication facility.
        .MCALL  .EXIT,.PRINT,.GTLIN
ERRBYT = 52                ;EMT error byte
START:  .GTLIN  #MSGBUF,#MSGPRT ;Get the message to be queued
        MOV    #MSGBUF,R1      ;Point to beginning of buffer
1$:     TSTB   (R1)+            ;Find end of message
        BNE    1$              ;
        SUB    #<MSGBUF+1>,R1  ;Determine message length
                                ;accounting for post-increment
        MOV    R1,MSGLEN       ;Set message length in EMT arg block
        .GTLIN  #CNLBUF,#CNLPRT ;Get the six character channel name
        MOV    #MSGBLK,RO      ;Point to EMT arg block to
        EMT     375            ;Send message on named channel
        BCC     9$             ;Branch if no error
        MOVB   @#ERRBYT,RO     ;Which error?
        ASL     RO             ;Convert to word index
        .PRINT  ERRTEL(RO)     ;Display the appropriate message
        .EXIT
9$:     CLRB    CNLBUF+6        ;Make channel name ASCII
        .PRINT  #DONEOK        ;Inform user message queued
        .PRINT  #CNLBUF        ;on channel CNLBUF
        .EXIT
        .MLIST  BEX
MSGBLK: .BYTE   0,104           ;EMT block: send message on named channel
        .WORD   CNLBUF          ;Address of channel name
        .WORD   MSGBUF          ;Address of message
MSGLEN: .WORD   0               ;Char length of message
ERRTEL: .WORD   NOPRIV          ;Table of send error messages
        .WORD   BSYERR
        .WORD   FULERR
        .WORD   OHOH           ;Error code 3 not used
        .WORD   TRNERR
NOPRIV: .ASCIZ  /?SNDMSG-F-No privilege or no message support./
BSYERR: .ASCIZ  /?SNDMSG-F-All message channels are busy./
FULERR: .ASCIZ  /?SNDMSG-F-Maximum number of messages have been queued./
OHOH:   .ASCIZ  /?SNDMSG-F-This is a non-existent error./
TRNERR: .ASCIZ  /?SNDMSG-W-Message was too long, truncated./
MSGPRT: .ASCIZ  /Message to be queued: /
CNLPRT: .ASCII  <15><12>/Channel Name (six characters max): /<200>
DONEOK: .ASCII  /Message queued on channel /<200>
        .EVEN
CNLBUF: .BLKB   80.             ;First 6 chars to contain file name
MSGBUF: .BLKB   80.             ;Message buffer.
        .END    START

```

6.3 Checking for pending messages

The following EMT is used to receive a message from a named channel if a message is pending on the channel. If no message is pending, an error code (3) is returned, and the program is allowed to continue execution. The form of the EMT is:

EMT 375

with R0 pointing to the following argument area:

```
.BYTE 0,105
.WORD chnadr
.WORD msgadr
.WORD msgsiz
```

where *chnadr* points to a field with a six character channel name, *msgadr* points to the buffer in which the message is to be placed, and *msgsiz* is the size of the message buffer (bytes).

If a message is received, its length (bytes) is placed in R0 on return from the EMT. If the message is longer than the message buffer (*msgsiz*), only the first part of the message will be received.

Error Code	Meaning
0	Job does not have MESSAGE privilege or system does not include message channel support.
1	All message channels are busy.
3	No message was queued on the named channel.
4	Message was longer than the receiving buffer.
5	Maximum number of message requests pending.

Example

```
.TITLE GETMSG
; Demonstrates use of the TSX-Plus EMT to check for pending messages in the
; interprocess message communication facility.
.MCALL .EXIT,.PRINT,.GTLM
ERRBYT = 52
START: .GTLM #CNLBUF,#CHLPRT ;Get the channel name
      MOV #MSGBLK,R0 ;Put EMT argument block address in R0
      EMT 375 ;EMT to check channel for message
      BCC 5$ ;Error?
      TSTB @#ERRBYT ;No privilege?
      BNE 1$ ;Branch if different error
      .PRINT #NOPRIV
      BR 9$
1$: CMPB @#ERRBYT,#4 ;Only two errors possible
      BEQ 2$ ;Overflow message buffer?
      .PRINT #NOMERR ;No message
      BR 9$
2$: .PRINT #TRNERR ;Print truncation warning
5$: .PRINT #PNDMSG ;Print message preamble
      .PRINT #MSGBUF ;Print actual message
9$: .EXIT
MSGBLK: .BYTE 0,105 ;GETMSG EMT block
        .WORD CNLBUF ;Channel name buffer address
        .WORD MSGBUF ;Buffer address to receive message
        .WORD 81. ;Buffer length
        .NLIST BEX
```

```

CNLBUF: .BLKB 80.           ;First 6 chars are channel name
MSGBUF: .BLKB 80.           ;Message buffer
        .WORD 0             ;Insure ASCIZ
        .WLIST BEX
CNLPRT: .ASCII /Channel Name (6 chars): /<200>
NOMERR: .ASCIZ /?GETMSG-F-No messages pending in named channel./
TRNERR: .ASCIZ /?GETMSG-W-Message truncated/<7>
PNMSG: .ASCIZ /Message pending in named queue is:/
NOPRIV: .ASCIZ /?GETMSG-F-No privilege or no message support./
        .LIST BEX
        .END START

```

6.4 Waiting for a message

The following EMT is used to suspend execution of a program until a message becomes available on a named channel. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument area:

```

.BYTE 0,106
.WORD chnadr
.WORD msgadr
.WORD msgsiz

```

where *chnadr* points to a six byte field containing the channel name, *msgadr* points to the buffer where the message is to be placed, and *msgsiz* is the size of the message buffer (bytes).

The length of the received message (bytes) is placed in R0 on return from the EMT.

Error Code	Meaning
0	Job does not have MESSAGE privilege or system does not include message channel support.
1	All message channels are busy.
4	Message was longer than the receiving buffer.
5	Maximum number of message requests pending.

Example

```

        .TITLE WATMSG
        .ENABL LC
; Demonstrate TSX-Plus EMT to wait for a queued message from the
; interprocess message communication facility.
        .MCALL .EXIT,.PRINT,.GTLIN
ERRBYT = 52
START:  .GTLIN #CNLBUF,#CNLPRT      ;Get the channel name
        .PRINT #WAITNG              ;Explain waiting
        MOV    #MSGBLK,R0           ;Put EMT argument block address in R0
        EMT    375                  ;EMT to check channel for message
        BCC    5$                   ;Check for error
        TSTB   @#ERRBYT             ;No privilege?
        BNE    1$                   ;Branch if different error
        .PRINT #NOPRIV
        BR     9$

```

```

1$:  CMPB    @#ERRBYT,#1          ;Error?
      BHI     2$                  ;Message truncated
      .PRINT  #NOMERR             ;All channels busy
      BR      9$
2$:  .PRINT  #TRNERR              ;Print truncation warning
5$:  .PRINT  #RCVMSG              ;Print message preamble
      .PRINT  #MSGBUF             ;Print actual message
9$:  .EXIT
MSGBLK: .BYTE 0,106              ;WATMSG EMT block
      .WORD  CNLBUF               ;Channel name buffer address
      .WORD  MSGBUF              ;Buffer address to receive message
      .WORD  81.                 ;Buffer length
CNLBUF: .BLKB 80.                 ;Channel name first 6 chars
MSGBUF: .BLKB 80.                 ;Message buffer
      .WORD  0                   ;Insure ASCIZ
      .LIST  BEX
CNLPRT: .ASCII /Channel Name (6 chars): /<200>
WAITNG: .ASCII /Waiting for a message . . ./<15><12>
      .ASCIIZ /Go to another line and send me something./
NOMERR: .ASCIIZ /?WATMSG-F-All message channels are busy./
TRNERR: .ASCIIZ /?WATMSG-W-Message truncated./<7>
RCVMSG: .ASCIIZ /Message received in named queue is:/
NOPRIV: .ASCIIZ /?WATMSG-F-No privilege or no message support./
      .LIST  BEX
      .END    START

```

6.5 Scheduling a message completion routine

The following EMT is used to schedule a completion routine to be entered when a message is received on a named channel. The form of the EMT is:

```
EMT      375
```

with R0 pointing to an argument block of the form:

```

.BYTE  1,106
.WORD  chnadr
.WORD  msgadr
.WORD  msgsiz
.WORD  cplrtn

```

where *chnadr* points to a six byte field containing the message channel name, *msgadr* is the address of the buffer where the message is to be placed, *msgsiz* is the size of the message buffer (bytes), and *cplrtn* is the address of the completion routine to be entered when a message is received.

On entry to the completion routine, R0 contains the number of bytes in the message that was received and R1 contains the number of the job that sent the message. Another message completion routine may be scheduled from within the completion routine if desired.

The .SPND and .RSUM system service calls may be used with this facility if the program reaches a point where it must suspend its execution until a message arrives.

Error Code	Meaning
0	Job does not have MESSAGE privilege or system does not include message channel support.
1	All message channels are busy
4	Message was longer than the receiving buffer.
5	Maximum number of message requests pending.

Example

```

        .TITLE MSGCPL
        .ENABL LC
; Demonstrate message completion routine
; Do processing while waiting for a message,
; then display the message and repeat.
        .MCALL .PRINT,.EXIT,.TTYOUT
        .GLOBL PRTDEC
        .DSABL GBL

ERRBYT = 52          ;EMT error byte address
SPACE  = 40          ;ASCII space char
BELL   = 7           ;ASCII bell char
REPTS  = 50          ;Outer idle loop repeat counter
WIDTH  = 32.         ;Screen formatting width
; Schedule message completion routine
START:  MOV    #MSGCPL,R0      ;Point to EMT arg block to
        EMT    375            ;Schedule completion routine
        BCC    2$             ;Branch if OK
        MOV    @ERRBYT,R1     ;Get error code
1$:     ASL     R1              ;Convert to word index
        .PRINT SCHERR(R1)     ;Print appropriate error message
        .EXIT                  ;Quit
; Idle processing loop while waiting for completion routine
2$:     MOV    #REPTS,R2      ;Outer idle loop counter
3$:     MOV    #-1,R3         ;Inner idle loop counter
        TST    MESSAG        ;Any message pending?
        BEQ    5$             ;Do other processing if not
        BMI    4$             ;Msg compl rtn error?
        CALL   DSPMSG         ;No error, display message
        BR     2$             ;Continue waiting for next
4$:     MOV    MESSAG,R1      ;Recover error code
        BR     1$             ;Go print error and die
5$:     SOB    R3,5$          ;Inner idle loop
        SOB    R2,3$          ;Outer idle loop
        .TTYOUT #BELL         ;Indicate activity
        BR     2$             ;And repeat forever
; Display the received message
DSPMSG: MOV    R3,-(SP)
        .PRINT #CRLF          ;Format display
        MOV    MSGLEN,R2      ;Get length count
        MOV    R2,R0          ;Again for display
        CALL   PRTDEC         ;Show message size
        .PRINT #MSGFRM        ;"byte message rec'd from job #"
        MOV    JOBNUM,R0      ;Get sending job number
        CALL   PRTDEC         ;Display job number
        MOV    #MSGADR,R1     ;Get pointer to message
        ADD    R1,R2          ;Save pointer to message end
1$:     MOV    #WIDTH,R4      ;Display-width counter
        .PRINT #LEFT          ;Format display
2$:     MOVB   (R1)+,R0        ;Get next character
        CMPB   R0,#SPACE      ;Printable?
        BGE    3$             ;Branch if so
        .TTYOUT #'~          ;Mark control character
        DEC    R4              ;Sub 1 from width
        MOVB   -1(R1),R0      ;Recover character
        ADD    #'@,R0         ;Convert to uppercase char
3$:     .TTYOUT                ;Display char
        CMP    R1,R2          ;End of display?
        BHIS   4$             ;Branch if so
        SOB    R4,2$          ;Continue unless need to wrap
        .PRINT #RIGHT         ;Format display
        BR     1$             ;Next line
4$:     .PRINT #RIGHT          ;Format display
        CLR    MESSAG         ;Say we are done with this one
        MOV    (SP)+,R3
        RETURN

```

```

; Completion routine to be entered when message received
CPLRTN: TST     MESSAG      ;Done with last message?
        BNE     1$         ;Branch (and lose this one) if not
        MOV     R0,MSGLEN   ;Save message length
        MOV     R1,JOBNUM   ;Save sending job number
        MOV     #1,MESSAG   ;Flag message received
1$:      MOV     #MSGCPL,R0  ;Point to EMT arg block to
        EMT     375         ;Reschedule myself
        BCC     2$         ;Branch if OK
        MOVB    @#ERRBYT,MESSAG ;Else save error code
        BIS     #100000,MESSAG ;And flag it as error
2$:      RETURN

; EMT arg blocks and word buffers
MSGCPL: .BYTE   1,106      ;EMT arg block - msg compl rtn
        .WORD   CHANAM     ;Pointer to channel name
        .WORD   MSGADR     ;Address of message buffer
        .WORD   MSGEND-MSGADR ;Size of message buffer
        .WORD   CPLRTN     ;Address of completion routine
JOBNUM: .WORD   0          ;Cell for # of message sender
MSGLEN: .WORD   0          ;Cell for received message length
MESSAG: .WORD   0          ;Cell for msg rcvd signal
SCHERR: .WORD   NOPRIV     ;Msg compl rtn sched error table
        .WORD   NOFREE     ; 1 No free message channels
        .WORD   UNUSED     ; 2 Max messages already queued
        .WORD   UNUSED     ; 3 No message queued on channel
        .WORD   TOOLNG     ; 4 Message overflowed buffer
        .WORD   NOMRB      ; 5 Max messages requests pending

; Messages and byte buffers
        .MLIST   BEX
CHANAM: .ASCII  /CHANEL/   ;Name of watched message channel
NOPRIV: .ASCIZ  /No privilege or no message support./
NOFREE: .ASCIZ  /All message channels are busy./
UNUSED: .ASCIZ  /MSGCPL should never generate this error./
TOOLNG: .ASCIZ  /Message was longer than message buffer./
NOMRB:  .ASCIZ  /Maximum number of message requests pending./
MSGFRM: .ASCII  / byte message received from job number /<200>
CRLF:   .BYTE   15,12,200
LEFT:   .ASCII  <15><12>/--/<76><200>
RIGHT:  .ASCII  <74>/--/<200>
MSGADR: .BLKB   200.       ;Message buffer
MSGEND:

        .LIST   BEX
        .EVEN
        .END    START

```


Chapter 7

Programming for CL and Special Device Handlers

Several device handlers are uniquely integrated into the TSX-Plus environment. The communication line handler (CL), the logical subset disk handler (LD), the terminal line handler (TT), and the single line editor (SL) are provided as integrated system features and do not require device declarations (DEVDEF). The professional interface handler (PI) is provided with PRO/TSX-Plus and when used is defined as a shared run-time system and not as a device handler. The TSX-Plus virtual memory handler (VM) utilizes knowledge of the TSX-Plus environment to determine the usable memory space available. VM is the only special TSX-Plus device handler which requires a device declaration (DEVDEF) in order to be used.

7.1 Communication line handler (CL)

The CL handler allows Input/Output operations to be performed to serial communication lines connected to any type of interface controller which may be used for terminal lines. With the CL handler it is possible to have some lines on a multiplexer used as TSX-Plus time-sharing lines, and other lines on the same multiplexer used to drive I/O devices such as printers, plotters, and modems. It is also possible to use a line as a time-sharing line some of the time and as a communications line at other times. Some of the important features of the CL handler are summarized below:

- Up to 16 communication lines may be controlled through the CL handler. The first 8 CL units are named CL0 through CL7, the second set of 8 units are named C10 through C17. The lines may be connected to any type of communication controller that is supported by TSX-Plus and may share the same multiplexer controllers as TSX-Plus time-sharing lines.
- Lines may be dedicated as communication lines or may be switched between time-sharing lines and communication lines.
- Internal queueing is used within the handler to allow concurrent input/output operations to be performed on all of the lines.
- The CL handler allows both input (read) and output (write) operations. Full duplex (simultaneous) read and write operations may take place on each line.
- The communication lines may be used with the TSX-Plus spooling system to allow spooled output to devices on communication lines.
- The CL handler responds to XON/XOFF (CTRL-Q/CTRL-S) control characters to stop and start its transmission and will generate XON/XOFF characters to control the speed of a device transmitting to a CL line.

- A *binary mode* is available for CL lines to allow full 8-bit, transparent I/O to devices.
- Modem control is supported. Ring and carrier detect signals may be monitored and data terminal ready (DTR) can be controlled by a program or SET command.
- The CL handler is implemented as a system virtual overlay, minimizing the amount of code and data that is required in the unmapped portion of the system.
- The CL handler can be used as a replacement for the LS, XL, and XC handlers (the XL and XC handlers are used with the RT-11 VTCOM program).
- A terminal can be *cross connected* to a CL line by use of the SET HOST command so that characters typed at the terminal are sent directly out the CL line and characters received on the CL line are displayed at the terminal.

Once a system has been generated with communication lines, the lines may be accessed as normal devices using the names CL0, CL1, C10, C17, etc. If the CL handler is used to drive the system printer, it is convenient to use an assign command to assign the logical name LP to the corresponding CL device.

The device name CL is functionally equivalent to CL0. C1 is equivalent to C10. Attempts to use a CL unit which is not currently associated with a line will return an error status just as if the CL device was not recognized by the system.

CL units specified using the CLDEF macro in TSGEN are initially connected to dedicated CL lines. Note that although these lines are dedicated for use by CL, the CL units which are initially assigned to these lines may be reassigned to other lines. The unallocated CL units declared by use of the CLXTRA parameter in TSGEN are initially not associated with any line. The SET CL n LINE= n and SET HOST/PORT=CL n keyboard command or the system service call (EMT) can be used to assign any CL unit to any free time-sharing line or free dedicated CL line. Thus it is possible to use a line as a TSX-Plus time-sharing line during certain portions of the day and then assign a CL unit to the line and use it to drive a modem or other device at other times. Dedicated CL lines use less memory space than time-sharing lines but may only be accessed as CL units. See the *TSX-Plus User's Reference Manual* for a full description of the SET CL and SET HOST command. TERMINAL privilege is required to use these SET commands.

Subsequent sections in this manual discuss special programming techniques for CL units.

7.1.1 VTCOM/TRANSF support and CL handler

The RT-11 VTCOM/TRANSF file transfer programs may be used to communicate and transfer files between RT-11 and/or TSX-Plus systems.

When VTCOM is used to communicate with another system, the system where the user is located and running VTCOM is known as the *local* system whereas the remote system to which communication is taking place is known as the *host* system. TSX-Plus may be used either as the local system, the host system, or both.

The user at the local system runs the VTCOM program to initiate communication with the host system. The VTCOM program uses the CL handler to connect to a communications line. The CL handler must be associated with a serial port of any type which is valid for terminal lines that is connected either directly or through a modem to the host system.

When TSX-Plus is used as the local system, the IOABT sysgen parameter must be set to 1 to enable handler abort entry code.

If the CL handler is used with the VTCOM program, it is necessary to assign the logical name XL (or XC if on the Professional) to the CL (or C1) unit controlling the communications line. It is also a good idea to allocate the device so that conflicts with other users will not occur. The NOLFOUT option should be specified for a CL line used with VTCOM. For example, the following commands would be appropriate to direct VTCOM to use line CL0:

```
SET CLO NOLFOUT
ASSIGN CLO XL (or XC)
ALLOCATE XL (or XC)
```

When TSX-Plus is used as the host system, the connection from the local system may be made through any TSX-Plus time-sharing line on the host system. When used for file transfers between a TSX-Plus system and a VMS system, VMS must be the host system (there is currently only a TRANSF program for VMS, but no VTCOM).

7.1.2 Terminal/Communication line cross connection

It is possible to cross connect a time-sharing line with a CL (communication line) unit in such a fashion that all characters received from the time-sharing line are transmitted directly to the CL unit and all characters received from the CL unit are transmitted directly to the time-sharing line. This is useful to allow a time-sharing line on one TSX-Plus system to be used as a terminal on another system connected through a CL unit.

This function is similar to using VTCOM to communicate through a CL line but has the advantage that there is much less overhead. The cross connection is made at a low level within TSX-Plus and characters do not have to be passed to a running application program. Of course the internal cross-connection feature does not provide the file transfer capabilities of VTCOM. Other disadvantages are that a cross-connected line cannot be switched to a sub-process and Process Windowing is temporarily disabled.

The keyboard command used to establish a cross connection has the form:

```
SET HOST/PORT=ddn
```

where *ddn* is the name of a CL or C1 device to which your terminal is to be cross connected. For example, the following commands would connect CL unit 1 with terminal line 4 at 9600 baud and then cross connect the current terminal with the CL unit:

```
SET CL1 LINE=4,SPEED=9600
SET HOST/PORT=CL1
```

TERMINAL privilege is required to use the SET HOST command. Once the cross connection is established, characters typed at your terminal are transmitted to the CL line.

See the *TSX-Plus User's Reference Manual* for more information on the SET HOST command and cross-connections.

7.1.3 Redirecting CL and time-sharing lines

A system service call (EMT) is available to allow a program to assign a CL unit to a particular line. This EMT is equivalent to the SET CL*n* LINE=*n* command. The form of the EMT is:

```
EMT      375
```

with R0 pointing to an argument block of the following form:

```
.BYTE    0,155
.WORD    cl-unit
.WORD    line-number
```

where *cl-unit* is the CL unit number, and *line-number* is the number of a TSX-Plus time-sharing line or dedicated CL line. If the specified line number is 0 (zero), the CL unit is disassociated from any line. TERMINAL privilege is required to use this EMT.

If an error is detected, the C-flag is set on return and the following error codes are returned:

Code	Meaning
1	User issuing the EMT does not have TERMINAL privilege
2	An invalid CL unit number was specified
3	An invalid line number was specified
4	The specified line is already assigned to a CL unit
5	A time-sharing user is logged onto the specified line
6	The specified CL unit is currently busy

The following example commands illustrate how CL unit 1 can be assigned to time-sharing line 2. The logical name LP is then assigned to CL1 so that the **PRINT** command will direct output through CL1. CL1 can be declared to be a spooled device in **TSGEN**:

```
SET CL1 LINE=2
ASSIGN CL1 LP
```

The **SHOW CL** and **SHOW TERMINALS** keyboard commands can be used to display information about which CL units are associated with which lines. The **SHOW CL** command also indicates if a CL unit is spooled and lists the options which are set for the unit. See the *TSX-Plus User's Reference Manual* for information concerning the **SHOW** command.

7.1.4 CL I/O operations

The **.READ/.READC/.READW** and **.WRITE/.WRITC/.WRITW** EMTs may be used to perform standard read/write operations to the CL lines. The CL handler allows full duplex input/output operation, which means that read and write operations may be simultaneously active on a CL line.

When a **.READ[C/W]** EMT is used to read from a CL line, the operation is complete when the requested number of words have been accepted or a **CTRL-Z** character is received.

The input character silo is used to store characters received from the line. This buffer prevents characters from being lost during the interval when one read EMT is completed and another is issued to the CL handler.

The CL handler responds to received **XON (CTRL-Q)** and **XOFF (CTRL-S)** characters, starting and suspending transmission to synchronize its character flow with the device connected to the line.

7.1.5 CL control character processing

Processing of certain control characters through CL units depends on the individual character, the settings of the CL unit, and the particular operation in progress.

The following table illustrates the handling of special input characters to a CL unit. Control characters not listed are treated as normal characters on input.

Char	Octal Value	Input Handling
NUL	0	Discarded unless in binary input mode (BININ).
LF	12	Discarded unless in LFIN mode (always input by .SPFUN 203).
CR	15	Always input. Also terminates read for .SPFUN 260 .
XON	21	Re-enables transmission to CL unit, except in BININ mode when it is input as a normal character.
XOFF	23	Halts transmission to CL unit, except in BININ mode when it is input as a normal character.
^Z	32	Sets <i>end of file</i> flag and terminates read (except for .SPFUN 203 by which ^Z is treated as a normal character).

The following table illustrates the handling of special output characters to a CL unit. In NOCTRL mode control characters not listed are not sent. In BINOUT mode, all special control character output processing is bypassed. In addition, in BINOUT mode automatic XOFF transmission when the input silo becomes full is disabled.

Char	Octal Value	Output Handling
NUL	0	Never sent except in BINOUT mode.
TAB	11	Expanded to spaces in NOTAB mode. In TAB mode and width not set to 0, TAB is discarded if it would exceed the set width, just as normal characters are.
LF	12	Discarded in NOLFOUR mode if preceding character was a carriage return.
FF	14	In NOFORM mode, FF is expanded to enough line feeds to advance to the top of the next page.
CR	15	Discarded in NOCR mode.

7.1.6 CL .SPFUN operations

The following special function codes (.SPFUN EMTs) are recognized by the CL handler. The special functions apply to the specific CL unit to which the channel was opened.

Code	Function
201	Clear XOFF status
202	Control break transmission
203	Read with byte count
204	Get handler status
205	Terminate I/O
206	Raise or drop DTR signal
250	Set option flags
251	Clear option flags
252	Set page length
253	Set skip lines
254	Set page width
255	Get modem status
256	Set line speed
257	Abort pending I/O
260	Read a line of input
261	Get number of input characters pending
262	Get number of output characters pending
263	Write with byte count
264	Set ENDPAGE and ENDSTRING parameters
265	Reset CL unit
266	Get current characteristics of a CL unit

Function 201 Clear handler

This function clears the internal handler flag that says an XOFF (CTRL-S) character has been received and transmits an XON (CTRL-Q) to the CL device.

Function 202 Control break transmission

This function starts or stops the transmission of a break signal. The word count specified with the .SPFUN controls whether transmission of a break signal is started or stopped. If the word count is non-zero, break transmission is started; break transmission continues until another .SPFUN is done with function code 202 and a word count of zero.

Function 203 Read with byte count

This function performs a read operation but the *word count* value specifies a byte count instead. This function does not complete until at least one byte is read. However, if a byte count greater than one is specified, bytes are moved from the input silo buffer until either the specified byte count is satisfied or the input silo buffer is emptied. If fewer than the requested number of bytes are available, the remainder of the buffer is filled with nulls. The CTRL-Z character does not signal end of file for this type of read—CTRL-Z is read as an ordinary character.

Function 204 Get handler status

A status code is stored into the first word of the buffer specified with this function. The meaning of the flag bits is as specified below:

If	Then
Bit 0=1	XOFF has been sent to stop transmission
Bit 1=1	XOFF has been received from the remote device
Bit 2=1	Carrier has been detected (same as bit 3) (This bit does <i>not</i> indicate CTS status)
Bit 3=1	Carrier has been detected (same as bit 2)
Bit 4=1	Ring signal is present

Function 205 Terminate I/O to the line

This function *turns off* a communication line. The input silo buffer is emptied (its contents are discarded) and a flag is set causing any other characters received from the line to be discarded. Data Terminal Ready (DTR) status is dropped. The line will be turned on again whenever another I/O operation is performed to it.

Function 206 Raise or drop DTR signal

This function can be used to raise or drop the Data Terminal Ready (DTR) signal for a line connected to a CL unit. If the word count value specified with the .SPFUN EMT is non-zero, the DTR signal is raised; if the word count value is 0 (zero) the DTR signal is dropped.

Functions 250 and 251 Control option flags

These special functions are used to set and clear handler option flags. Function 250 sets the specified flag bits (via a BIS instruction), function 251 clears the specified flag bits (via a BIC instruction). The flag bits which are set (1) correspond to handler SET options. If the option flag is cleared (0), this corresponds to the NOOption setting. The bit positions of the options are shown in the following table. The option flags are contained in a one word buffer for the .SPFUN. For a detailed description of the SET commands, see the *TSX-Plus User's Reference Manual*.

Bit	Mask	Option	Function summary
0	000001	FORM	Send form feed characters
1	000002	TAB	Send tab characters
2	000004	LC	Send lower case characters
3	000010	LFOUT	Send line feed characters
4	000020	LFIN	Accept line feed characters
5	000040	FORM0	Send form feed on block 0 write
6	000100	BINOUT	Send binary output characters
7	000200	BININ	Accept binary input characters
8	000400	CR	Send carriage return characters
9	001000	CTRL	Send control characters
10	002000	DTR	Raise Data Terminal Ready (DTR)
11	004000	EIGHTBIT	Accept and send 8 bit characters

Function 252 Set page length

This function performs the operation of the SET CL LENGTH=n command. The .SPFUN must have a one word buffer containing the number of lines per page.

Function 253 Set skip lines

This function performs the operation of the SET CL SKIP=*n* command. The .SPFUN must have a one word buffer containing the number of lines to skip at the bottom of the page.

Function 254 Set page width

This function performs the operation of the SET CL WIDTH=*n* command. The .SPFUN must have a one word buffer containing the line width.

Function 255 Get modem status

This function is used to check on the status of a modem connected to a CL line. The modem status is returned into the first word of the buffer specified with the .SPFUN. The flag bits returned are described below:

Bit	Meaning when set
0	Ring indication
1	Carrier is detected
2	Data Terminal Ready is asserted

Function 256 Set line speed, character length and parity control

This function is used to set the transmit/receive speed for a CL line. This .SPFUN requires a one word buffer containing a value which has the following form (in binary):

OPLxSSSS

The low-order 4 bits (*SSSS*) specify the speed. The following baud rates are represented by the indicated speed codes (speed code values are shown in decimal): 50=0, 75=1, 110=2, 134.5=3, 150=4, 300=5, 600=6, 1200=7, 1800=8, 2000=9, 2400=10, 3600=11, 4800=12, 7200=13, 9600=14, 19200=15. Bit 5 ("L") specifies the character length. If this bit is 0, the character length is 8 bits; if this bit is 1, the character length is 7 bits. Bit 6 ("P") specifies parity control selection. If this bit is 0, parity is disabled and bit 7 is ignored; if this bit is 1, parity generation and checking is enabled. Bit 7 ("O") selects even or odd parity and is only meaningful if bit 6 is 1 (enable parity). If bit 7 is 0, even parity is selected; if bit 7 is 1, odd parity is selected. Note that if only the speed value is specified, with all other bits zero, 8 bit characters with no parity are selected.

This .SPFUN can only be used for lines connected to hardware controllers that support programmable baud rates (such as DZ11 ports and the Professional communication port). A baud rate of 19200 is not supported by some hardware controllers including the DEC DZ11 controller (although it actually works with most DEC DZ11 controllers). The DH11 does not support baud rates of 2000, 3600, or 7200; and the DHV11 does not support baud rates of 3600 or 7200.

Function 257 Abort pending I/O

Abort all pending read and write operations issued by the job executing the .SPFUN on the CL unit.

Function 260 Read a line of input

This special function reads a line of input terminated by a carriage return character. The *word count* value specified with this special function is interpreted as a byte count. The read terminates when any of the following conditions is met:

- A carriage return character is received. The carriage return is stored in the buffer and the remainder of the buffer is null filled.
- The buffer is filled before a carriage return is received.
- A CTRL-Z is received.

Function 261 Determine number of input characters pending

One word is returned into the user buffer, containing the number of characters available to be read from the CL unit associated with the specified channel. This function can be used to test for pending input prior to issuing a read (.READx or .SPFUN) on the channel. If no characters are pending, attempts to read from the channel will not complete until the word (or byte) count is fulfilled. If you do not wish to have a read request pending until the word (or byte) count is fulfilled, first determine the number of characters pending in the input buffer. If there are none, do not issue the read.

Function 262 Determine the number of output characters pending

One word is returned into the user buffer, containing the number of characters in the output buffer which have not yet been transmitted.

Function 263 Write with byte count

This special function is used to write a block of characters to a CL line with the number of characters specified by a byte count rather than the word count used with the .WRITE EMT. This is useful in situations where an odd number of bytes must be written and null characters cannot be used to pad out the last word. The .SPFUN follows the standard form except a byte count is specified for the fifth parameter (which is normally used to specify a word count).

Function 264 Set ENDPAGE and ENDSTRING parameters

This special function is used to specify the number of form-feed characters (ENDPAGE) and a seven character string (ENDSTRING) which will be appended to the end of each output file. The buffer address must point to a word aligned storage area of which the first word contains the number of form-feed characters. The second and subsequent words contain the string in ASCIZ form to append to the end; any characters beyond the first seven are ignored.

Function 265 Reset CL unit status

This special function resets the status of a CL unit by performing the following operations:

1. Empty the input silo of received characters.
2. Empty the output ring buffer of transmitted characters.
3. Stop sending a break if one is currently being sent.
4. Clear XOFF (CTRL-S) received flag.
5. Send an XON if we have previously sent an XOFF.
6. Clear end-of-file status and reset line and column numbers.

Function 266 Get CL characteristics

This special function returns current information about the CL unit. The buffer address must start on an even boundary and must be at least 12 words long. The values returned are:

Offset	Contents
0	Handler status word (same as for .SPFUN code 204)
2	CL options flags (same as for .SPFUN codes 250 and 251)
4	Internal status word (see bit description below)
6	Page length
10	Number of skip lines
12	Page width
14	CL unit number (CL units are 10-17 octal) (low byte).
15	Line number in use as CL unit (high byte)
16	Number of end of file form feeds
20-27	ASCIZ string to send at end of file (can be up to 7 bytes plus one null)

The following bits are defined in the internal status word:

Mask	Meaning
10	Carriage return was the last character transmitted
40	Next read will receive end of file
200	Break being transmitted
4000	DTR has been asserted (explicitly by SET CL n DTR or setting the option flag, or implicitly by a READ or WRITE request)

Other bits are undefined and may vary.

Example

The following example program demonstrates use of some CL programming techniques which might be used to simulate a virtual terminal through a CL line.

```
.TITLE CLCOMM
;
; "Simple" CL communications program
;
        .DSABL GBL
        .GLOBL PRTR50,PRIDEC
        .MCALL .LOOKUP,.SPFUN,.CLOSE
        .MCALL .TTYOUT,.PRINT
        .MCALL .HERR,.SERR,.EXIT
;
TTUSE   = 1           ;Line number to attach CL unit to
CMDCHR  = 20 ;ctrl-P  ;Exit command character
;
JSW     = 44           ;Job Status Word address
TTSPC$  = 10000        ;Terminal special mode bit (SINGLE)
TTLCS$  = 40000        ;Enable lower case input bit
ERRBYT  = 52           ;EMT error byte address
;
CLRBYT  = 203          ;SPFUN code to read CL with byte count
CLOFF   = 206          ;SPFUN code to turn CL unit off
CLCOP   = 251          ;SPFUN code to clear CL flags
CLIPND  = 261          ;SPFUN code to get # chars pending CL input
CLWBYT  = 263          ;SPFUN code to write CL with byte count
CLRST   = 265          ;SPFUN code to reset CL unit
;
CLCHAN  = 0            ;IO channel to use to access CLn
;
; First two routines handle their own fatal errors
START:  CALL  TTSET    ;Set up terminal characteristics
        CALL  CLGET    ;Get and initialize CL line
; From here on, errors are handled by main line code
        CALL  TTINPC   ;Queue terminal input completion routine
        BCC   1$       ;Continue if OK
        .PRINT #BADQUE ;Else say what's wrong
        BR    9$       ;And abort
;
; Start main loop -----
1$:     TST    DONE     ;Did terminal give exit command char?
        BNE   9$       ;Exit on command
;
        CALL  CLIO     ;Service any pending CL input or output
        BCC   1$       ;Continue if no error
; End main loop -----
;
9$:     CALL  CLFREE    ;Clear out CL line and close unit
        .PRINT #OFF
        .EXIT
;
        .MLIST BEX
OFF:    .ASCIZ /<Exit>/
BADQUE: .ASCIZ /Error queueing terminal input routine/<?>
        .EVEN
        .LIST  BEX
;
        .SBTTL TTSET -- Set terminal single, lc
;-----
;
TTSET:  .SERR          ;Inhibit monitor EMT error handling
        MOV    #SINGLE,R0 ;Point to EMT arg block to
        EMT    375      ;Set single character activation
        BCS    7$       ;Abort on EMT error
```

```

        .HERR          ;Return error control to monitor
;
        BIS            *<TTLCS!TTSPC$>,@#JSW ;Enable single/lc
        RETURN
;
7$:      .HERR          ;Return error control to monitor
        .PRINT        #NOTTSX ;EMT error, may be RT-11
        .EXIT
;
        .NLIST        BEX
SINGLE:   .BYTE         0,152 ;EMT arg block to enable
        .WORD         'S    ;Single character input
        .WORD         0     ;Unused
NOTTSX:  .ASCIZ        /Invalid EMT error - cannot continue/<7>
        .EVEN
        .LIST         BEX
;
        .SBTTL        CLGET -- find and initialize CL line
;-----
;
CLGET:   CALL          ALCCL ;Try to allocate a CL unit
        BCC           2$    ;Continue if we own a CL unit
        .PRINT        #CNTALC ;Say we cannot get a CL unit
        .EXIT          ;And abort
;
2$:      CALL          ATTCL ;Try to attach CL unit to line
        BCC           3$    ;Continue if OK
        .PRINT        #NOCLS ;Say can't connect
        CALL          ABORT1 ;Release allocation
        .EXIT          ;And abort
;
3$:      CALL          CLINI ;Try to initialize the CL line
        ;Handles its own errors
;
        .PRINT        #CONNECT ;Connecting
        MOV           CLDEV,RO ;Get CL unit
        CALL          PRTR50 ;and display it
        .PRINT        #TOLINE ;to line number
        MOV           LINNUM,RO ;Get line number
        CALL          PRTDEC ;and display it
        .PRINT        #CRLF
        RETURN
;
        .NLIST        BEX
CNTALC:  .ASCIZ        /Unable to allocate any CL unit/<7>
CONNECT: .ASCII        /Connecting /<200>
TOLINE:  .ASCII        / to line number /<200>
CRLF:    .ASCIZ        //
NOCLS:   .ASCIZ        /Unable to attach CL unit to a line/<7>
        .EVEN
        .LIST         BEX
;
        .SBTTL        ALCCL -- allocate a CL unit
;-----
;
ALCCL:   MOV           #ALCENT,RO ;Point to EMT arg block to
        EMT           375 ;Allocate a CL unit
        MOV           RO,CLOWN ;Remember who owns the CL unit
        BCC           9$    ;Normal return if allocation OK
;
        MOVB         @#ERRBYT,RO ;See which error occurred
        CMPB         RO,#1 ;Already allocated to another job?
        BEQ           1$    ;Try next if so
        CMPB         RO,#4 ;In use by another job?
        BNE           8$    ;If any other error, return with error
;
1$:      MOV           CLDEV,RO ;Get current CL unit name
        CALL          PRTR50 ;And display it
        .PRINT        #XTLDBY

```

```

        MOV     CLOWN,RO ;Get job number of line using CL
        CALL    PRTEDEC ;And display it
        .PRINT  #CRLF
        INC     CLDEV ;Try next unit number
        INC     CLUNIT ;and remember which one
        BR      ALCCL ;And try to allocate it
;
8$:     SEC ;Set return error flag
9$:     RETURN
;
        .MLIST  BEX
ALCEMT: .BYTE 0,156 ;EMT arg block to
        .WORD  CLDEV ;Allocate a CL unit
CLOWN:  .WORD 0 ;Owner of the CL unit
XTLDBY: .ASCII / in use by job /<200>
        .EVEN
        .LIST  BEX
;
        .SBTTL  ATTCL -- attach CL unit to a TT line
;-----
;
ATTCL:  MOV     #GETCL,RO ;Point to EMT arg block to
        EMT     375 ;Attach a CL unit to a TS line
        RETURN
;
        .MLIST  BEX
GETCL:  .BYTE 0,155 ;EMT arg block to connect CL to a line
CLUNIT: .WORD 0 ;CL unit number
LINNUM: .WORD TTUSE ;TT line number
        .LIST  BEX
;
        .SBTTL  CLINI -- open CL unit and init it
;-----
;
CLINI:  .LOOKUP #AREA,#CLCHAN,#CLDEV ;Try to open CL channel
        BCC     1$ ;Continue if OK
        .PRINT  #BADLKP
        CALL    ABORT2 ;Dissociate and deallocate
        .EXIT ;And abort
;
1$:     .SPFUN #AREA,#CLCHAN,#CLRST,#CLOBUF,#0,#0 ;Flush CL
        BCS     2$ ;Abort on error
        .SPFUN #AREA,#CLCHAN,#CLCOP,#CLNFLG,#0,#0 ;Clear flags
        BCC     9$ ;Continue if OK
2$:     .PRINT  #CLIOER
        CALL    CLFREE ;Purge channel, dissociate and deallocate
        .EXIT
;
9$:     RETURN
;
        .MLIST  BEX
CLDEV:  .RAD50 /CLOFILNAMDAT/ ;CL device and dummy file spec
CLNFLG: .WORD 10 ;no(LFOUT)
BADLKP: .ASCIZ /Error opening CL unit/<7>
CLIOER: .ASCII "CL I/O error"<7>
        .EVEN
        .LIST  BEX
;
        .SBTTL  CLFREE -- close and deallocate CL unit
;-----
;
; Flush the channel and ignore further input from it
CLFREE: .SPFUN #AREA,#CLCHAN,#CLOFF,#CLOBUF,#1,#1 ;Turn off CL
        .CLOSE  #CLCHAN ;Clear channel
;
; Release the terminal from the CL line
ABORT2: CLR     LINNUM ;Clear terminal assignment
        MOV     #GETCL,RO ;Point to EMT arg block to
        EMT     375 ;Dissociate CL unit from line

```

```

;
; Deallocate the CL unit
ABORT1: MOV    #1,ALCEMT ;Change allocate to DEallocate
        MOV    #ALCEMT,RO ;Point to EMT arg block to
        EMT    375      ;Deallocate the device
        RETURN

;
        .SBTTL  TTINP -- accept and handle terminal input
;-----
;
TTINP:  CMP    RO,#CMDCHR ;Is it the exit command char?
        BNE    1$        ;Continue if not
        INC    DONE      ;Say we should quit
        RETURN          ;And exit without requeusing

;
1$:     MOV    RO,@TTIPTR ;Save char to send to CL
        INC    TTIPTR    ;And point to next position
        CMP    TTIPTR,#CLOBND ;Time to wrap?
        BLO    2$        ;Br if not
        MOV    #CLOBUF,TTIPTR ;Wrap to beginning of buffer
2$:     .TTYOUT          ;Display on TT (we must echo)

;
TTINPC: MOV    #QTTINC,RO ;Point to EMT arg block to
        EMT    375      ;Requeue terminal inp comp routine
        RETURN

;
        .MLIST  BEX
DONE:   .WORD  0        ;Exit flag (not 0 --> exit)
QTTINC: .BYTE  1,133    ;EMT arg block to queue TT
        .WORD  TTINP    ; input completion routine
        .LIST  BEX

;
        .SBTTL  CLIO -- read and write pending CL I/O
;-----
;
; Transmit terminal input out through CL line
CLIO:   CMP    TTOPTR,TTIPTR ;Is there any new char to send?
        BEQ    1$        ;Br if not
        .SPFUN #AREA,#CLCHAN,#CLWBYT,TTOPTR,#1,#1 ;Write 1 byte
        BCS    9$        ;Return with error?
        INC    TTOPTR    ;Point to next output char position
        CMP    TTOPTR,#CLOBND ;Time to wrap to beginning?
        BLO    1$        ;Br if not
        MOV    #CLOBUF,TTOPTR ;Wrap to beginning of buffer

;
; See if any characters are coming in from the CL line
1$:     .SPFUN #AREA,#CLCHAN,#CLIPND,#CLSBUF,#1,#1 ;Check for input
        BCS    9$        ;Error return?
        MOV    CLSBUF,R3 ;Get number of characters to read
        CLC          ;Make sure error flag is off
        BEQ    9$        ;Return if no CL input pending (CLSBUF=0)
        .SPFUN #AREA,#CLCHAN,#CLRBYT,#CLIBUF,R3,#1 ;Read what is pending
        BCS    9$        ;Error return?
        MOV    R3,TTOCNT ;Set number of characters to display
        MOV    #TTOBLK,RO ;Point to EMT arg block to
        EMT    375      ;Display R3 chars on the terminal

;
9$:     RETURN

;
        .MLIST  BEX
AREA:   .BLKW  10        ;General purpose EMT arg block
TTOBLK: .BYTE  0,114    ;EMT arg block to send # of chars to TT
        .WORD  CLIBUF    ;Point to chars received from CL
TTOCNT: .WORD  0        ;Count of chars to be displayed
CLSBUF: .WORD  0        ;1 word CL status buffer
CLIBUF: .BLKW  128      ;Can hold up to 255 characters
CLOBUF: .BLKW  40       ;CL output ring buffer
CLOBND = .             ;First location past CLOBUF
TTIPTR: .WORD  CLOBUF    ;Index into buffer for next input char
TTOPTR: .WORD  CLOBUF    ;Index into buffer for next output char
        .LIST  BEX

;
        .END  START

```

7.2 RK06/RK07 handler (DM)

The DM device handler .SPFUN function codes 376 and 377 attempt to return a status code into the first word of a user buffer which is one word longer than the actual transfer size. This is incompatible with system I/O mapping. It appears that the only system utility program which issues these functions is DUP (SQUEEZE and INITIALIZE commands). If it is necessary to use the MAPIO option with the DM device handler, it is recommended that both INITIALIZE and SQUEEZE commands for DM units be issued only under RT-11.

The DM handler supports 22-bit (as well as 18-bit) Q-Bus I/O with the DILOG DQ215 and the Emulex SC02C controllers.

The DM handler has been modified to allow the Emulex SC02/C controller to work on faster processors (such as the 11/93). This controller, which is popular on Q-Bus systems because it supports 22-bit addressing, appears to have a latency in its registers after receiving certain controller commands. This controller register latency results in a "Fatal System Error, KTP-Kernel mode trap" with the "Arg. value" pointing to an address within the DM handler. This problem first appeared when 11/23 processors were replaced with (faster) 11/73's. At that time, a patch was developed for the DM handler which inserted some "No Operation" (NOP) instructions after loading certain commands into the controller registers. With the advent of even faster processors, such as the 11/83 and 11/93, the delay after loading these controller registers needs to be lengthened even more.

To compensate for the variety of processor speeds, the delaying sequence of NOP's has been replaced by a call to a delay loop. The number of iterations of this loop can be controlled with a handler SET command. The handler option is named DELAY and accepts values from 0 to 127. For 11/23 processors, a value of 0 is appropriate. For 11/73 processors, a value of 1 is sufficient. The appropriate values for 11/83 and 11/93 processors have not yet been determined, but a value of 4 should be adequate. The default value is 1. The following example would set the number of iterations of the delay loop to 4 for use on a 11/93 processor:

```
SET DM DELAY=4
$STOP
```

Note that the SET DM DELAY=n command must be issued *while running TSX* and then the system should be rebooted.

7.3 DU (MSCP) handler

The DU handler does not support extended unit addressing. That is, the handler only supports a maximum of eight units (DU0: — DU7:) for a total of about 256Mb through a single controller. To support multiple drives with up to 256Mb on each, they must each be connected through a separate (or logically separate) controller. Use a separate copy of the DU.TSX handler for each such device. Be sure that subsequent DU handler copies are incorporated with the correct TSGEN options. For example, to include a second DU handler which has been copied to DA.TSX, the appropriate DEVDEF options would be:

```
DEVDEF <DA>,DMA,NOMAPH,NOSET
```

For greater flexibility in supporting devices larger than 256Mb, third-party *front-end* or *pass-off* handlers are available.

The BYPASS special function (371) is not supported with the DU (MSCP) device handler on extended Unibus hardware.

7.4 IEEE GPIB handler (IB)

The normal IB supplied subroutines attempt to open the IB device on decimal channel numbers 16, 17, 18, and 19. TSX-Plus normally allocates 20 (decimal) channels and allows these IB subroutines to execute without changes. Non-standard configurations where multiple devices may be used and more than 20 (decimal) channels are required are not supported.

A change is also necessary to the IB device handler to alter the mapping register used from PAR1 to PAR6. See the section concerning device handlers use of PARs discussed in the chapter on Device Handlers in the *TSX-Plus System Manager's Guide*. See the Patching and Building TSX-Plus Device Handlers chapter in the *TSX-Plus Installation Guide* for information on how to build an IB handler which will function with TSX-Plus.

The IB subroutine IBSRQ is implemented in the DEC IB handler as a subroutine call from the handler directly to the user code region. Since TSX-Plus does not load any user job in the same map region as the operating system, the call will execute part of the operating system, usually resulting in a fatal system error or halt. Therefore, the IBSRQ call is unsupported in TSX-Plus.

7.5 Logical subest disk handler (LD)

The LD device handler is implemented as an overlay in TSX-Plus and does not use the RT-11 LD handler. However, its implementation is compatible with logical subset disks from RT-11 so that LDs created, mounted and used under either operating system may also be accessed under the other, using the same commands.

The TSX-Plus LD pseudo-handler accepts two special function codes: 372 to read the LD translation tables, and 373 to return the device size. A request to return the device size returns the size of the file which is mounted as the LD unit to which the channel is open on which the .SPFUN was issued. Unlike RT-11, a channel must be open to a valid LD unit in order to read the LD translation tables. That is, some unit must be mounted and accessible and a channel opened to such an LD unit by a successful .LOOKUP (usually non-file structured). The translation table for all current LD units is returned, regardless of the unit to which the .SPFUN is issued. If the unit is not currently mounted, but was previously, some of the information may be retained from the previous mount. The information is only valid if bit 15 is set in the LD unit's status word. The LD translation tables are read-only, regardless of the value of the special function word-count parameter; both reads and writes to the LD translation tables are treated as reads and will return the translation tables into the user buffer.

The contents of the LD translation tables returned by .SPFUN code 373 differs according to the version of RT-11 from which TSX-Plus was started. If the RT-11 version is 5.4 or later, then the table is prefaced by two additional words and one of the status bits is redefined. If the RT-11 version is 5.3 or earlier, then the table begins immediately with the LD unit status words and one of the status bits is redefined, otherwise the translation tables are the same. The format of the table is as follows:

Size	Use
2 bytes	RAD50 value of "LD " if RT-11 V5.4 or later, absent if not
1 byte	Number of LD units available (8) V5.4 or later, absent if not
1 byte	Unused V5.4 or later, absent if not
8*2 bytes	Status word for each LD unit
8*2 bytes	Starting block number of each LD
8*2 bytes	Number of blocks in each LD
8*8 bytes	Device:Filename.Extension in RAD50 of each LD (device name includes unit number)

The status word for each LD unit is defined as follows:

Bits	Mask	Meaning
15	100000	Flag bit if LD unit is currently mounted
14	40000	Flag bit if LD unit is read-only (RT-11 V5.4 and later)
13	20000	Flag bit if LD unit is read-only (RT-11 V5.3 and earlier)
10-8	3400	Unit number of device holding LD file
5-1	76	Index of device holding LD file

Example

```

        .TITLE  LDTBL
;
; Example program to display LD translation tables
;
        .MCALL  .PRINT,.EXIT,.LOOKUP,.CLOSE,.SPFUN,.SERR,.HERR
        .DSABL  GBL
        .GLOBL  PRTR50,PRTOCT
;
BEL      = 7                ;ASCII BELL character
NLDS     = 8.               ;Default number of LD units
;
; Start by trying to find a channel to any LD unit.
; Can only read translation tables after getting
; a channel (any channel) to the handler.
;
START:   .SERR              ;Trap LOOKUP errors
        .LOOKUP #AREA,#0,#DBLK ;Is this LD unit mounted?
        BCC     1$          ;Branch if so
        INC     DBLK         ;If not, try the next one
        CMP     DBLK,#~RLD7  ;Past last possible LD yet?
        BLOS    START        ;Keep looking if not
        .PRINT  #NLDS        ;Oops, no LDs on line
        BR      9$           ;Go exit
;
; Now try to read and display the LD translation tables
;
1$:      .HERR              ;Reenable system error trapping
        .SPFUN #AREA,#0,#372,#BUFF,#1,#0 ;Get translation tables
        BCC     2$          ;Br if SPFUN OK
        .PRINT  #BADSPF      ;Else report error
        BR      8$           ;And quit
2$:      CALL    SHOLDS       ;Display LD translation tables
;
8$:      .CLOSE  #0           ;Close channel to LD unit
9$:      .EXIT
;
; LD translation table format is:
;
; IF      RT-VERSION >= 5.4      ;These 2 words are absent if <RT V5.4
;      .RAD50 /LD /              ;1 word of device name
;      .BYTE  NLDS,0             ;1 byte = # LD units, 1 byte 0
; END IF
; STATUS: .BLKW NLDS             ;1 word/unit LD status
;      INUSE = 100000 ;Unit is allocated mask in STAT word
;      RONLY = 040000 ;Unit is read-only RT V5.4 and later
;      RONLY = 020000 ;Unit is read-only RT V5.3 and earlier
;      DVUNIT = 003400 ;Mask for real device unit #
;      DVINDX = 000076 ;Mask for real device index #
; Note: The following words may contain residue from previous mounts.
;      They are only meaningful if the INUSE flag is set in STATUS.
; START:  .BLKW NLDS             ;1 word/unit real disk start block #
; SIZE:   .BLKW NLDS             ;1 word/unit device size (blocks)
; NAME:   .BLKW 4*NLDS           ;4 words/unit name of file mounted
;
SHOLDS:  MOV      #NLDS,R3       ;Assume 8 LD units

```

```

MOV     #BUFF,R1           ;Point to returned LD table
CMP     (R1),#~RLD        ;New table format?
BNE     1$                 ;Br if not
TST     (R1)+              ;Step over RAD50 LD
MOV     (R1)+,R3           ;Get count of LD units (low byte)
BIC     #~C377,R3         ;Will be 8. under TSI-Plus
;
1$:      .PRINT #TBLHDR      ;Explain display
MOV     #2,R5              ;# bytes per unit for STATUS,START,SIZE
MUL     R3,R5              ;# bytes per table for STATUS,START,SIZE
MOV     R1,STATS           ;Set ptr to status word vector
ADD     R5,R1              ;Step ptr over status words
MOV     R1,STRTS           ;Set ptr to starting block vector
ADD     R5,R1              ;Step ptr over starts
MOV     R1,SIZES           ;Set ptr to size vector
ADD     R1,R5              ;Use R5 as ptr to names
;
MOV     #~RLD0,R1          ;Current unit name
2$:      CALL    SH01LD      ;Display each unit
INC     R1                 ;Step each pointer to next unit
ADD     #2,STATS
ADD     #2,STRTS
ADD     #2,SIZES
SOB     R3,2$              ;Through the entire list
RETURN
;
SH01LD: MOV     R1,R0        ;Display unit name
CALL    PRTR50
.PRINT  #SP2
MOV     @STATS,R0           ;Display unit status
CALL    PRTOCT
.PRINT  #SP2
MOV     @STRTS,R0           ;Display unit starting block #s
CALL    PRTOCT
.PRINT  #SP2
MOV     @SIZES,R0           ;Display .DSK file sizes
CALL    PRTOCT
.PRINT  #SP2
MOV     (R5)+,R0            ;Display DEV:FILNAM.EXT
CALL    PRTR50
.PRINT  #COLON
MOV     (R5)+,R0
CALL    PRTR50
MOV     (R5)+,R0
CALL    PRTR50
.PRINT  #DOT
MOV     (R5)+,R0
CALL    PRTR50
.PRINT  #CRLF
RETURN
;
.NLIST  BEX
AREA:   .BLKW  10           ;General EMT arg block
DBLK:   .RAD50  /LD0/       ;LD device name
        .WORD  0,0,0
BUFF:   .BLKW  256.         ;At least 2*(2+(8*3)+(8*4))
STATS:  .WORD  0            ;Pointer to Status words
STRTS:  .WORD  0            ;Pointer to Starting blocks
SIZES:  .WORD  0            ;Pointer to Sizes
NOLDS:  .ASCIZ  /Sorry, no LD's currently mounted./<7>
BADSPF: .ASCIZ  /Oops, .SPFUM error!/<BEL>
CRLF:   .ASCIZ  //
SP2:    .ASCII  /  /<200>
COLON:  .ASCII  /:/<200>
DOT:    .ASCII  /./<200>
TBLHDR: .ASCII  /Unit Status  Offset  Size  DEV:FILNAM.EXT/<15><12>
        .ASCIZ  /---  -----  -----  -----/
        .LIST  BEX
        .EVEN
        .END  START

```


Some of the information available in the LD translation tables may be obtained without opening a channel to an LD unit by using EMT 375, function code 4,135. This EMT returns five words of information for each LD unit: four words of RAD50 Device:Filename.Extension and one word of status. The status word has two bits defined: bit 0 (mask 1) is set if the unit is read-only; bit 1 (mask 2) is set if the unit is not currently accessible (for example if it is mounted within an outer LD and the outer LD has been dismounted).

Other EMT's are also available to mount and dismount LD units from within programs. See the descriptions of EMT 375 functions 3,135 4,135 5,135 and 163 in Chapter 4.

7.6 MU (TMSCP) handler

The MU handler is not supported on extended Unibus hardware. The BYPASS special function (371) is not supported for use with the MU handler.

7.7 Address Translation (AT) and Unibus (UB) handlers

The AT handler is not supported. See the chapter on "Real-Time Program Support" for a description of the system service to convert a virtual address to a physical address.

The UB handler is not supported. Support for Unibus mapping is provided by the kernel.

7.8 Virtual memory handler (VM)

The virtual memory handler (VM) allows memory which is not allocated for use by the operating system to be used as a RAM based pseudo-disk device. VM may not be used to contain either the swap or spool system files due to the nature of system completion routine nesting. When VM is used as a spool or swap device, unpredictable operation may occur resulting in fatal system errors and system halts. VM is implemented as a device handler (VM.TSX) and not as a system overlay (like LD and CL). VM.TSX is not derived from the RT-11 VM handler (there is no VM.SLP). As a device handler, it requires an entry in the device tables and must be included with a DEVDEF macro during system generation.

The VM handler uses the memory space above the top of memory used by TSX-Plus. TSX-Plus can be limited to using less than all installed memory by specifying the TSGEN MEMSIZ parameter. See the *TSX-Plus Installation Guide* for details on the MEMSIZ setting. Since a memory access is quite a bit faster than a disk access, VM can be used for greater speed in locating and reading files which are frequently accessed.

Since most machines will lose the contents of memory during a power outage, VM should be restricted to read-only, scratch, or executable files. It may be used to speed the execution of heavily overlaid programs or store temporary intermediate sort or work files.

After TSX-Plus is started, VM must be initialized before it can be used (except see using with RT-11 VM below). Since VM is implemented as a block structured device, and each block contains 512 bytes, the number of blocks available to VM will be two times the number of kilobytes allocated. The directory does require some storage and therefore the number of blocks reported after initialization will be slightly smaller than this total. For instance, in a system which contains 512 Kb total physical memory and with MEMSIZ=256., VM will have 256 Kb available. After initialization, a directory of VM will then show slightly less than 512 blocks.

VM will normally calculate the correct base address to use to be just above the last address used by TSX-Plus. You may increase this base address. The format of the SET command used to adjust the base address used by VM is:

```
SET VM BASE=nnnnnn
```

where *nnnnnn* represents bits 6 through 22 of the base memory address (in octal) which VM is allowed to use. However, if you specify a base address below the top address of TSX-Plus, VM will dynamically adjust this base address back above the top of TSX-Plus. For example, if you wish to set the base address of VM to start after the first 512 Kb, then *nnnnnn* should be 20000 since the memory address is 2000000 (octal). Any time a new base address is defined, VM should be initialized.

VM will normally calculate the correct top address to use to be at the absolute top of physical memory. You may decrease this top address. The format of the SET command used to adjust the top address used by VM is:

```
SET VM TOP=nnnnnn
```

where *nnnnnn* represents bits 6 through 22 of the top memory address (in octal) which VM is allowed to use. For example, if you wish to set the top address of VM to end at 1280 Kb, then *nnnnnn* should be 50000 since the memory address is 5000000 (octal). Any time a new top address is defined, VM should be initialized.

You may also use the same memory region which was defined as VM under RT-11. In order to do so, you must make sure that the MEMSIZ parameter restricts TSX-Plus to use of memory below the existing RT-11 VM base or else it will be corrupted during the TSX-Plus initialization phase. You must also be careful to SET VM BASE and TOP to match the settings used under RT-11. If you match the TSX-Plus VM to overlay an existing RT-11 VM, then the contents will be intact and data stored there while running RT-11 will be available after starting TSX-Plus.

VM accepts two special functions: 372 to return two words of VM status information; and 373 to return the device size in blocks. For function 372, the first word returned in the user buffer is the current base of VM (the actual 64 byte block number, not the value set in the handler) and the second word is the current VM top.

Example

```
.TITLE  VMSHOW
;
; Demonstrate .SPFUN to return VM information
;
.MCALL  .SPFUN,.PRINT,.EXIT,.LOOKUP,.CLOSE
.DSABL  GBL
.GLOBL  PRTOCT,PRTDEC
;
VMSHOW = 372                ;SPFUN code to return VM stats
;
START:  .LOOKUP #AREA,#0,#DEVNAM ;Open channel to VM
        BCC     1$           ;Br if OK
        .PRINT  #LKPERR      ;Else complain
        BR      9$           ;And abort
1$:     .SPFUN  #AREA,#0,#372,#BUFFER,#0,#0 ;Try to get VM stats
        BCC     2$           ;Br if OK
11$:    .PRINT  #SPFERR       ;Else complain
        BR      9$           ;And abort
2$:     .PRINT  #BASE         ;Display VM info
        MOV     BUFFER,R0     ;Get base value (in 64 byte blocks)
        CALL    PRTOCT
        .PRINT  #TOP
        MOV     BUFFER+2,R0   ;Get top value (in 64 byte blocks)
        CALL    PRTOCT
        .SPFUN  #AREA,#0,#373,#BUFFER,#0,#0 ;Get VM size
        BCS     11$          ;Complain on error
        .PRINT  #SIZE
        MOV     BUFFER,R0     ;Get size in blocks
        CALL    PRTDEC
9$:     .CLOSE  #0            ;Close channel to VM
        .EXIT
;
```

```

        .MLIST  BEX
AREA:  .BLKW  10           ;General EMT arg block
DEVNAM: .RAD50  /VMO/      ;Non-file structured lookup
BUFFER: .WORD  0,0,0       ;Return SPFUN data here
;
LKPERR: .ASCIZ  /.LOOKUP error/
SPFERR: .ASCIZ  /.SPFUN error/
BASE:   .ASCII  /VM Base=/<200>
TOP:    .ASCII  / Top=/<200>
SIZE:   .ASCII  / Size=/<200>
        .LIST  BEX
        .END   START

```

7.9 Communications handler (XL)

The RT-11 VTCOM utility communicates with host systems through a device named XL. You may either install and use the XL device handler or you may make a logical assignment of XL to a current CL unit. If you use XL.TSX, then the interface is dedicated solely to use with the XL device, whereas by defining the line as a time-sharing line and then “taking it over” as a CL unit, you gain increased flexibility in utilization of the port. In addition, XL is restricted to DL(V)11 type interfaces, while CL may be used on any type of serial interface supported for terminals. This allows you to alter the port speed with a keyboard command (on programmable rate interfaces) and monitor the line’s state with SYSMON. Because of this increased flexibility, we strongly recommend the use of CL units rather than the XL device handler. For more information on SET CLn commands see the *TSX-Plus User’s Reference Manual*. Also see the other sections in this chapter for use of CL with VTCOM and for information on programming with CL.

7.10 Spooled devices

Output sent to spooled devices with .WRITx EMTs is intercepted by the TSX-Plus spooling system and transferred to a system managed disk file. Data in this file is then supplied to the handler for the spooled device by the system as needed. For a more general discussion of device spooling, see the *TSX-Plus User’s Reference Manual*.

Data blocks in the spool file begin with two words of header information, followed by up to 508 bytes of data. The first header word is a pointer to the next block in the spool file for this data stream. The second word is the number of data *words* in the current block. Information such as spool file starting blocks, file names and output device is stored and managed elsewhere by the system.

Requests to read input from spooled devices are not intercepted or otherwise affected by the spooling system. Special function (.SPFUN) requests of all kinds to spooled devices are similarly ignored and unaffected by the spooler. Thus, in order to have output to a spooled device actually spooled, the I/O request should ordinarily be done via the .WRITW EMT. The fact that .READx and .SPFUN requests bypass the spooler is not ordinarily important to print devices such as LS and LP. However, when programming I/O to special device handlers for devices like plotters and modems, it is important to remember that only .WRITx requests utilize spooling.

An example of a potentially undesirable effect of spooling can be seen by running the VTCOM program with XL assigned to a spooled CL unit. Since VTCOM uses a .WRITC to transmit data, it is intercepted by the spooler and buffered through the spool file. No information will actually be sent to the CL unit until the block is full (508 bytes) or the channel is closed. Because of this unusual behavior, use of VTCOM through a spooled CL unit is not recommended.

In some cases, it may be desirable to bypass the spooling system. For special device handlers and for CL units which support special function write requests, .SPFUN requests can accomplish this. Since LS and LP do not (currently) support special function requests, another approach must be used. The spooling system identifies spooled devices both by device name and unit number. Thus, if LP (equivalent to LP0) is spooled,

then only writes to LP0 will be spooled. Since the LP device handler ignores unit numbers, writes to LP1 will bypass the spooler, but will be handled normally by the LP handler. The same holds true for LS - writing to LS1 bypasses spooled LS. Note that CL does support device units and thus you cannot bypass the spooler on spooled CL units by selecting a different unit number.

Since mixed output will result if two jobs write to a non-file-structured device like LS, LP or CL unless managed by the spooling system, care must be taken to prevent this when bypassing the spooler. We recommend that non-spoiled serial devices or spooled devices which are being bypassed be allocated by the job before writing to them.

Chapter 8

Real-Time Program Support

TSX-Plus provides a real-time program support facility that allows multiple real-time programs to run concurrently with normal time-sharing operations. The basic functions provided by this facility are summarized below:

- The ability to map the I/O page into the user's virtual memory region so that device status and control registers may be directly accessed by the program.
- The ability to connect device interrupt vectors to program interrupt service routines running at fork level or to program completion routines running at user-selectable priority levels with full job context.
- The ability for a program to lock itself in memory so that rapid interrupt response can be assured.
- The ability for a program to suspend its execution until an interrupt occurs.
- The ability to set execution priorities for tasks.
- The ability to convert a virtual address within the job's region to a physical address for DMA I/O control.
- The ability to map a virtual address region to a physical address region.
- The ability for a program to declare a list of addresses of device control registers to be reset when the program exits or aborts (.DEVICE EMT).

Most of the features associated with the real-time facility are a basic part of TSX-Plus, but the ability to attach interrupt service routines or interrupt completion routines to interrupt vectors is optional during system generation.

Special privilege is required to perform some of the requests described in this chapter. For example:

REALTIME privilege is required to use the .DEVICE request or to get exclusive system control;

MEMMAP privilege is required to map to the I/O page or other specific locations in physical memory;

PSWAPM privilege is required to lock a program in memory.

The real-time facilities are available to both normal jobs controlled by time-sharing lines and to detached jobs. Note that detached jobs that are specified during system generation for automatic startup run with all privileges, unless specifically changed; detached jobs started by time-sharing users have the same privilege as the job starting them.

8.1 Accessing the I/O page

A basic facility required by most real-time programs is the ability to access the PDP-11 I/O page (160000–177777) which contains the device control and status registers. Under TSX-Plus, addresses in this range are normally mapped to a simulated RMON or may be used as normal program space. This is done since many old programs require direct access to certain system values at fixed offsets into RMON, although recent programs should access these values with the .GVAL and .PVAL EMTs. TSX-Plus provides several EMTs to deal with mapping of the I/O page and accessing locations within it. These are discussed below. See Chapter 2 for a discussion of various mapping techniques which may be used under TSX-Plus. The /IOPAGE switch to the R[UN] command may also be used to map a program's PAR 7 to the I/O page.

A TSX-Plus real-time program can access the I/O page in one of two ways: It can cause the program's virtual address region in the range 160000 to 177777 to be mapped directly to the I/O page so that it can directly access device registers; or it can leave the virtual address range mapped to the simulated RMON and use a set of EMTs to peek, poke, bit-set and bit-clear registers in the I/O page. It is much more efficient to directly access the device control registers by mapping the I/O page into the program's virtual address region than to use EMTs to perform each access. However, this technique will not work if the program must also directly access offsets inside RMON. The correct way for a program to access RMON offsets is to use the .GVAL EMT which will work even if the I/O page is mapped into the program region.

8.1.1 EMT to map the I/O page into the program space

The following EMT can be used to cause the program's virtual address region in the range 160000 to 177777 to be mapped to the I/O page. Use of this EMT requires MEMMAP privilege. The /IOPAGE switch to the R[UN] command may also be used to map a program's PAR 7 to the I/O page. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument area:

```
.BYTE    5,140
```

The I/O page mapping set up by this EMT remains in effect until the program exits, chains, or the EMT described in the next section is used to remap to RMON. Note that completion routines and interrupt service routines run with the same memory mapping as the mainline code of the job.

The .GVAL EMT with offset value -8. may be used to determine if PAR 7 is currently mapped to the I/O page or to the simulated RMON. See the description in Chapter 4 of the special use of .GVAL for further information.

Example

```
.TITLE  MAPIOP
.ENABLE LC
;Demonstrate TSX-Plus EMTs to map to the I/O page and back to RMON
RMONST = 54                ;SYSCOM location holding base of RMON
CONFIG = 300               ;Fixed offset into RMON of CONFIG word
RCSR    = 176540           ;Serial line RCSR address
RBUF    = RCSR+2           ;Line input buffer address
CTRLC   = 3               ;^C
.MCALL  .PRINT,.EXIT,.TTYOUT,.GVAL
.GLOBAL PRTOUT
START:  CALL  SHOMAP        ;Display current PAR 7 mapping
        CALL  SHODAT        ;Demonstrate RMON mapping
        .PRINT #IOPMSG      ;Say we are switching to I/O page mapping
        MOV   #MAPIOP,R0    ;Point to EMT arg block to
        EMT   375           ;Map to I/O page
        BCC   1$            ;Branch if OK to map
        .PRINT #NOPRIV      ;You are not allowed to do that
```

```

.EXIT
1$: CALL SHOMAP ;Display current PAR 7 mapping
.PRINT #TYPE ;Prompt for input from I/O page
MOV @#RCSR,CSRSAB ;Save a copy of current CSR
CLR @#RCSR ;Disable interrupts on serial line
2$: TSTB @#RCSR ;Is anything available from the line
BPL 2$ ;No, keep checking
MOVB @#RBUF,R0 ;Get the new character
CMPB R0,#CTRLC ;Should we quit?
BEQ 3$ ;Quit on ^C
.TTYOUT ;Display the character
BR 2$ ;And repeat
3$: MOV CSRSAB,@#RCSR ;Restore original CSR
.PRINT #GOBACK ;Say we are returning to original mapping
MOV #MAPMON,R0 ;Point to EMT arg block to
EMT 375 ;Map back to simulated RMON
CALL SHOMAP ;Display current PAR 7 mapping
CALL SHODAT ;And prove we are back
.EXIT
SHOMAP: .PRINT #MAPMSG ;Current mapping preface
.GVAL #AREA,*-8. ;What is our current PAR 7 mapping?
ASL R0 ;Convert to word offset
.PRINT CURMAP(R0) ;Show which one it is
RETURN
SHODAT: .PRINT #CNFGIS ;Preface config value
MOV @#RMONST,R0 ;Pick up pointer to RMON base
MOV CONFIG(R0),R0 ;Get the current CONFIG value
CALL PRTOCT ;Display it
.PRINT #CRLF
RETURN
MAPIOP: .BYTE 5,140 ;EMT arg block to map to I/O page
MAPMON: .BYTE 6,140 ;EMT arg block to map to RMON
AREA: .WORD 10 ;EMT arg block
CSRSAB: .WORD 0 ;Save CSR for restoration on exit
CURMAP: .WORD TORMON ;Current mapping message table
.WORD TOIOPG
.WLIST BEX
CRLF: .ASCIZ //
TORMON: .ASCIZ /simulated RMON./
TOIOPG: .ASCIZ "I/O page."
MAPMSG: .ASCII /PAR 7 is currently mapped to /<200>
CNFGIS: .ASCII /Current value of CONFIG word in simulated RMON is /<200>
TYPE: .ASCIZ /Characters entered on serial line will be displayed here:/
IOPMSG: .ASCIZ <12>"Now switching PAR 7 mapping to the I/O page."
GOBACK: .ASCIZ <15><12>/Returning PAR 7 mapping to simulated RMON./
NOPRIV: .ASCII /Real-time support not specified during TSGEN or /
.ASCIZ /user not privileged./
.END START

```

8.1.2 EMT to remap the program region to the simulated RMON

The following EMT can be used to cause the virtual address mapping region of the job in the range 160000 to 177777 to be returned to normal mapping if it had previously been mapped to the I/O page. Use of this EMT requires MEMMAP privilege. The form of the EMT is:

```
EMT 375
```

with R0 pointing to the following argument area:

```
.BYTE 6,140
```

Example

See the example program MAPIOP in section 8.1.1 on mapping the I/O page into the program space.

8.1.3 EMT to peek at the I/O page

The following EMT can be used to access a word in the I/O page without requiring the job's virtual address region to be mapped to the I/O page. (Note that the .PEEK and .POKE EMTs can also be used to access parts of the I/O page. See Chapter 12 for more information on the effects of the .PEEK and .POKE EMTs with TSX-Plus.) Use of this EMT requires MEMMAP privilege. The form of this EMT is:

```
EMT      375
```

with R0 pointing to the following argument area:

```
.BYTE    1,140
.WORD    address
```

where *address* is the address of the word in the I/O page to be accessed. The contents of the specified word in the I/O page are returned in R0. Note that with this and other EMTs that access the I/O page, if an invalid address is specified, an error will result with the message:

```
?MON-F-Kernel mode trap within TSX-Plus
```

Example

```
.TITLE PEEKIO
.ENABL LC
;Demonstrate TSX--Plus EMTs to peek and poke into the I/O page
CTRLC = 3          ;^C
RMONST = 54         ;Pointer in SYSCOM area to start of RMON
CONFIG = 300        ;Fixed offset into RMON of config word
RCSR = 176540       ;Serial line RCSR address
RBUF = RCSR+2       ;Line input buffer address
.MCALL .PRINT,.EXIT,.TTYOUT,.GVAL
.GLOBL PRTOCT
START: CALL SHOMAP      ;Display current PAR 7 mapping
      CALL SHOCOM      ;Demonstrate RMON mapping
      MOV  #PEEKIO,R0   ;Point to EMT arg block to
      EMT  375          ;Peek into the I/O page
      BCC  1$          ;Branch if OK
      .PRINT #NOPRIV    ;You are not allowed to do that
      .EXIT
1$:   MOV  R0,CSRSAB     ;Save a copy of current CSR
      CLR  POKVAL        ;Want to disable interrupts
      MOV  #POKEIO,R0   ;Point to EMT arg block to
      EMT  375          ;Poke a value into the I/O page
                        ;Should not be error if peek worked
      .PRINT #TYPE      ;Prompt for input from I/O page
2$:   MOV  #PEEKIO,R0   ;Point to EMT arg block to
      EMT  375          ;Check the RCSR
      TSTB R0           ;Is anything available from the line
      BPL  2$          ;No, keep checking
      ADD  #2,PEKADD     ;Point to the receiver buffer
      MOV  #PEEKIO,R0   ;Point to EMT arg block to
      EMT  375          ;Get the input character
      CMPB R0,#CTRLC    ;Should we quit?
      BEQ  3$          ;Quit on ^C
      .TTYOUT           ;Display the character
      SUB  #2,PEKADD     ;Point back to the RCSR
      BR   2$          ;And repeat
3$:   MOV  CSRSAB,POKVAL ;Want to restore the original CSR status
      MOV  #POKEIO,R0   ;Point to EMT arg block to
      EMT  375          ;Restore the CSR
      .EXIT
SHOMAP: .PRINT #MAPMSG   ;Current mapping preface
```



```

        .GVAL  #AREA,#-8.      ;What is our current PAR 7 mapping?
        ASL    RO              ;Convert to word offset
        .PRINT CURMAP(RO)      ;Show which one it is
        RETURN
SHOCOM: .PRINT #CHFGIS        ;Preface config value
        MOV    @#RMONST,RO     ;Pick up pointer to RMON base
        MOV    CONFIG(RO),RO   ;Get the current CONFIG value
        CALL   PRTOCT          ;Display it
        .PRINT #CRLF
        RETURN
PEEKIO: .BYTE  1,140          ;EMT arg block to peek into the I/O page
PEKADD: .WORD  RCSR           ;Address to be read
POKEIO: .BYTE  2,140          ;EMT arg block to poke into the I/O page
POKADD: .WORD  RCSR           ;Address to be modified
POKVAL: .WORD  0              ;Word to be moved to POKADD
AREA:   .WORD  10             ;EMT arg block
CSRSAB: .WORD  0              ;Save CSR for restoration on exit
CURMAP: .WORD  TORMON         ;Current mapping message table
        .WORD  TOIOPG
        .NLIST BEX
CRLF:   .ASCIZ //
TORMON: .ASCIZ /simulated RMON./
TOIOPG: .ASCIZ "I/O page."
MAPMSG: .ASCII /PAR 7 is currently mapped to /<200>
CHFGIS: .ASCII /Current value of CONFIG word in simulated RMON is /<200>
TYPE:   .ASCIZ /Characters entered on serial line will be displayed here:/
NOPRIV: .ASCII /Real-time support not specified during TSGEN or /
        .ASCIZ /user not privileged./
        .END  START

```

8.1.4 EMT to poke into the I/O page

The following EMT can be used to store a value into a cell in the I/O page without requiring the job's virtual address region to be mapped to the I/O page. Use of this EMT requires MEMMAP privilege. The form of the EMT is:

EMT 375

with R0 pointing to the following argument area:

```

        .BYTE  2,140
        .WORD  address
        .WORD  value

```

where *address* is the address of the cell in the I/O page and *value* is the value to be stored.

Example

See the example program PEEKIO in section 8.1.3 on peeking into the I/O page.

8.1.5 EMT to bit-set a value into the I/O page

The following EMT can be used to perform a bit-set (BIS) operation into a cell in the I/O page without requiring the job's virtual address region to be mapped to the I/O page. Use of this EMT requires MEMMAP privilege. The form of the EMT is:

EMT 375

with R0 pointing to the following argument area:

```
.BYTE 3,140
.WORD address
.WORD value
```

where *address* is the address of the cell in the I/O page and *value* is the value that will be bit-set into the cell.

Example

```
.TITLE BISIO
.ENABL LC
;Demonstrate TSX--Plus EMTs to bit-set and bit-clear into the I/O page
CTRLC = 3 ;~C
RMONST = 54 ;Pointer in SYSCOM area to start of RMOW
INTNBL = 100 ;RCSR interrupt enable bit
CONFIG = 300 ;Fixed offset into RMOW of config word
RCSR = 176540 ;Serial line RCSR address
RBUF = RCSR+2 ;Line input buffer address
.MCALL .PRINT,.EXIT,.TTYOUT,.GVAL
.GLOBL PRTOCT
START: CALL SHOMAP ;Display current PAR 7 mapping
CALL SHOCON ;Demonstrate RMOW mapping
;Want to clear input interrupts
MOV #BICIO,RO ;Point to EMT arg block to
EMT 375 ;Clear a bit in the I/O page
BCC 1$ ;Branch if OK
.PRINT #NOPRIV ;You are not allowed to do that
.EXIT
1$: ;OK, interrupts should be disabled
.PRINT #TYPE ;Prompt for input from I/O page
2$: MOV #PEEKIO,RO ;Point to EMT arg block to
EMT 375 ;Check the RCSR
TSTB RO ;Is anything available from the line
BPL 2$ ;No, keep checking
ADD #2,PEKADD ;Point to the receiver buffer
MOV #PEEKIO,RO ;Point to EMT arg block to
EMT 375 ;Get the input character
;Value from peek is returned in RO
CMPB RO,CTRLC ;Should we quit?
BEQ 3$ ;Quit on ~C
.TTYOUT ;Display the character
SUB #2,PEKADD ;Point back to the RCSR
BR 2$ ;And repeat
3$: ;Want to set input interrupts
MOV #BISIO,RO ;Point to EMT arg block to
EMT 375 ;Set a bit in the I/O page
.EXIT
SHOMAP: .PRINT #MAPMSG ;Current mapping preface
.GVAL #AREA,*-8. ;What is our current PAR 7 mapping?
ASL RO ;Convert to word offset
.PRINT CURMAP(RO) ;Show which one it is
RETURN
SHOCON: .PRINT #CNFGIS ;Preface config value
MOV @RMONST,RO ;Pick up pointer to RMOW base
MOV CONFIG(RO),RO ;Get the current CONFIG value
CALL PRTOCT ;Display it
.PRINT #CRLF
RETURN
PEEKIO: .BYTE 1,140 ;EMT arg block to peek into the I/O page
PEKADD: .WORD RCSR ;Address to be read
BICIO: .BYTE 4,140 ;EMT arg block to clear a bit in the I/O page
.WORD RCSR ;Input status register
.WORD INTNBL ;Interrupt enable mask
BISIO: .BYTE 3,140 ;EMT arg block to set a bit in the I/O page
.WORD RCSR ;Input status register
.WORD INTNBL ;Interrupt enable mask
AREA: .BLKW 10 ;EMT arg block
```

```

CSRSV: .WORD 0 ;Save CSR for restoration on exit
CHARS: .BLKB 6 ;6 char output buffer
EOW: .WORD 0 ;Terminator for .PRINT
CURMAP: .WORD TORMON ;Current mapping message table
        .WORD TOIOPG
        .WLIST BEX
CRLF: .ASCIZ //
TORMON: .ASCIZ /simulated RMON./
TOIOPG: .ASCIZ "I/O page."
MAPMSG: .ASCII /PAR 7 is currently mapped to /<200>
CNFGIS: .ASCII /Current value of CONFIG word in simulated RMON is /<200>
TYPE: .ASCIZ /Characters entered on serial line will be displayed here:/
NOPRIV: .ASCII /Real-time support not specified during TSGEN or /
        .ASCIZ /user not privileged./
        .END START

```

8.1.6 EMT to do a bit-clear into the I/O page

The following EMT can be used to perform a bit-clear (BIC) operation into a cell in the I/O page without requiring the job's virtual address region to be mapped to the I/O page. Use of this EMT requires MEMMAP privilege. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument area:

```

.BYTE  4,140
.WORD  address
.WORD  value

```

where *address* is the address of the cell in the I/O page and *value* is the value to be bit-cleared into the specified cell.

Example

See the example program BISIO in section 8.1.5 on doing a bit-set into the I/O page.

8.2 Mapping to a physical memory region

In certain circumstances, it is desirable to map a portion of virtual memory to a *specific* area in physical memory which is not in the I/O page. Possible examples would be ROM memory or an array processor. This mapping is done by altering one or more of the Page Address Registers (PARs) for the job. Each PAR maps 8192 bytes of memory from the virtual job space into physical memory. There are 8 PARs ($8 \times 8192 = 64$ Kb). The region of memory mapped through each PAR is shown by the table below:

PAR	Virtual Region
0	000000-017777
1	020000-037777
2	040000-057777
3	060000-077777
4	100000-117777
5	120000-137777
6	140000-157777
7	160000-177777

Use of this EMT requires MEMMAP privilege. The form of this EMT is:

EMT 375

with R0 pointing to the following argument area:

```
.BYTE 17,140
.WORD par-number
.WORD phys-address
.WORD size
.BYTE access-type,cache-bypass
```

where *par-number* is the number of the Page Address Register (PAR) that corresponds to the beginning of the program virtual address region that is to be mapped; *phys-address* is the physical address to which the virtual address is to be mapped. The physical address is specified as an address divided by 64 (decimal). That is, the physical address represents the 64-byte block number of the start of the physical region. Note that the physical address of any 64-byte block within a 22-bit physical address space can be represented in 16 bits.

Size is the number of 64-byte blocks of memory to be mapped through the virtual region. Each PAR can map up to 128 64-byte blocks of memory. If more than 128 blocks are mapped, successively higher PARs are set up to map the remainder of the region.

Access-type indicates if the mapped region is to be allowed read-only access or both read and write access: 0=read-only; 1=read/write.

Cache-bypass causes hardware cache to be bypassed if this byte is 1. If this byte is 0, the normal case, then hardware caching is enabled. When *cache-bypass* is set to 1, the actual effect is to set bit 15 in the Page Description Register (PDR) for each physical page mapped by this EMT. This can be important when mapping to devices such as bit-mapped graphic displays and array processors that interface to the Q-bus by dual-port memory.

This EMT may be used to map any of the PARs to any physical address regions desired; even to map PAR 7 to the I/O page. The use of this EMT does not affect mapping set up previously for other PARs. If *size* is zero (0), all PAR mapping is reset and the normal virtual address mapping for the job is restored.

Note that this EMT is not equivalent to the extended memory EMTs used with PLAS (Program's Logical Address Space) requests and virtual overlays and virtual arrays. See Chapter 2 for a discussion of job environment and the appropriate RT-11 manuals for a more complete discussion of PLAS features.

Note that disassociation of shared run-time regions (EMT 375, function 0,143 with a null name-pointer) has the side effect that regions mapped with this EMT will revert to the normal job mapping in effect prior to mapping the physical region.

Example

```
.TITLE MAPPHY
.ENABL LC
; Demonstrate TSX--Plus EMT to map to any physical address
CTRLC = 3 ; ^C
INTNBL = 100 ; Interrupt enable bit
RCSR = 176540 ; Serial line RCSR address
RBUF = RCSR+2 ; Line input buffer address
.MCALL .PRINT,.EXIT,.TTYOUT,.GVAL
START: .PRINT #MAPMSG ; Say we are switching to physical mapping
MOV #MAPPHY,R0 ; Point to EMT arg block to
EMT 375 ; Map to physical memory
BCC 1$ ; Branch if OK to map
.PRINT #NOPRIV ; You are not allowed to do that
.EXIT
1$: .PRINT #TYPE ; Prompt for input from I/O page
BIC #INTNBL,#RCSR ; Disable interrupts on serial line
```

```

2$:  TSTB    @#RCSR      ;Is anything available from the line
      BPL     2$         ;No, keep checking
      MOVB    @#RBUF,R0  ;Get the new character
      CNPB    R0,*CTRLC  ;Should we quit?
      BEQ     3$         ;Quit on ^C
      .TTYOUT      ;Display the character
      BR      2$         ;And repeat
3$:  CLR     SIZE        ;If .size is 0, original mapping is restored
      MOV     #MAPPHY,R0 ;Point to EMT arg block to
      EMT     375        ;Revoke mapping to physical address
      .EXIT
MAPPHY: .BYTE 17,140     ;EMT arg block to map to physical memory
        .WORD 7         ;remap PAR 7
        .WORD 177600    ;17760000/64. address to map into PAR 7
SIZE:   .WORD 200       ;size of region to be mapped - full 8 Kb
        .WORD 1         ;access 0=read-only, 1=read/write
        .NLIST BEX
MAPMSG: .ASCIZ /PAR 7 mapping is now directed to top of physical memory./
TYPE:   .ASCIZ /Characters entered on serial line will be displayed here:/
NOPRIV: .ASCIZ /You are not privileged to run this program./
        .END    START

```

8.3 Requesting exclusive system control

The following EMT allows real-time jobs to gain exclusive access to the system while they perform time critical tasks. Use of this EMT requires REALTIME privilege. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument block:

```
.BYTE    14,140
```

The effect of this EMT is to cause the TSX-Plus job scheduler to ignore all other jobs, even higher priority jobs—including fixed-high-priority jobs until the real-time job relinquishes exclusive access control. Note that this is different (and more powerful) than giving the real-time job a higher execution priority because all other jobs are completely prevented from executing even if the real-time job goes into a wait state causing the CPU to become idle.

The form of the EMT used to relinquish exclusive system access is:

```
EMT      375
```

with R0 pointing to the following argument block:

```
.BYTE    15,140
```

The following restrictions apply to a job that has issued an exclusive access EMT:

1. The job is automatically locked in memory during the time it has exclusive access to the system. If you wish to use the TSX-Plus EMT that locks a job in the lowest portion of memory, that EMT must be executed before calling this EMT to gain exclusive access to the system.
2. The size of the job may not be changed while it has exclusive access to the system.

Exclusive access is automatically relinquished if the job exits or traps but is retained if the job chains to another job.

Example

```

        .TITLE  STEALS
        .ENABL  LC
; Demonstrate TSX--Plus EMT to obtain exclusive system control
LEADIN  =      35          ;TSX-Plus program controlled terminal
                        ;option lead-in character.
JSW     =      44          ;Job status word address
TTSPC   =      10000       ;TT special mode bit in JSW
NOWAIT  =      100        ;TT nowait bit in JSW
        .MCALL  .PRINT,.TTYOUT,.TTYIN,.EXIT
START:  MOV     #TTSPC!NOWAIT,@#JSW ;Set TT special mode and nowait
                        ;bits in the JSW
        .TTYOUT #LEADIN      ;And make TSX--Plus understand both
        .TTYOUT #'S
        .TTYOUT #LEADIN
        .TTYOUT #'U
        .PRINT  #MSG         ;Prompt for input
        MOV     #STEALS,R0    ;Point to EMT arg block to
        EMT     375           ;Steal exclusive system control
; . . .                      ;Time critical processing goes here
        .TTYIN                ;Simulated here with terminal input
        MOV     #LETGO,R0     ;Point to EMT arg block to
        EMT     375           ;Relinquish exclusive system control
        .EXIT
        .NLIST  BEX
STEALS: .BYTE   14,140        ;EMT arg block for exclusive system control
LETGO:  .BYTE   15,140        ;EMT arg block to relinquish exclusive control
MSG:    .ASCII  /Other users are locked out until you press a key:/<200>
        .END    START

```

8.4 Locking a job in memory

In time-critical real-time applications where a program must respond to an interrupt with minimum delay, it may be necessary for the job to lock itself in memory to avoid program swapping. Use of the EMTs to lock a program in memory or to lock into low memory (below 256 Kb) or to unlock a program from memory (re-enable swapping) requires PSWAPM privilege. This facility should be used with caution since if a number of large programs are locked in memory there may not be enough space left to run other programs.

TSX-Plus provides two program locking facilities. The first moves the program to the low end of memory before locking it; this is done to avoid fragmenting available free memory. This type of lock should be done if the program is going to remain locked in memory for a long period of time. However, this form of locking is relatively slow since it may involve program swapping. The second locking facility simply locks the program into the memory space it is occupying when the EMT is executed without doing any repositioning. This EMT has the advantage that it is extremely fast but free memory space may be non-contiguous.

The form of the EMT used to lock a program in low memory (re-positioning if necessary) is:

```
EMT      375
```

with R0 pointing to the following argument area:

```
.BYTE    7,140
```

The form of the EMT used to lock a job in memory without repositioning it is:

```
EMT      375
```

with R0 pointing to the following argument area:

```
.BYTE    13,140
```

<i>Error Code</i>	<i>Meaning</i>
0	Job does not have PSWAPM privilege

Example

See the example program ATTVEC in section 8.12.2 on connecting a real-time interrupt to a completion routine.

8.5 Unlocking a job from memory

When a job locks itself in memory, it remains locked until the job exits or the following EMT is executed. The form of the EMT used to unlock a job from memory is:

```
EMT      375
```

with R0 pointing to the following argument area:

```
.BYTE    10,140
```

Example

See the example program ATTVEC in section 8.12.2 on connecting a real-time interrupt to a completion routine.

8.6 Suspending/Resuming program execution

The RT-11 standard .SPND and .RSUM EMTs are used by TSX-Plus real-time jobs to suspend and resume their execution. Frequently, a real-time job will begin its execution by connecting interrupts to completion routines and doing other initialization and then will suspend its execution while waiting for a device interrupt to occur.

The .SPND EMT causes the mainline code in the job to be suspended. Completion routines are not affected and execute when interrupts occur. If a completion routine executes a .RSUM EMT, the mainline code will continue execution at the point following the .SPND when the completion routine exits. Refer to the *RT-11 Programmer's Reference Manual* for further information about the use of .SPND and .RSUM.

Example

See the example program ATTVEC in section 8.12.2 on connecting a real-time interrupt to a completion routine.

8.7 Converting a virtual address to a physical address

When controlling devices that do direct memory access (DMA), it is necessary to be able to obtain the physical memory address (22-bit) that corresponds to a virtual address in the job. Note that a job should lock itself in memory before performing these EMTs. These virtual to physical address translation EMTs do not require any special privilege.

There are two forms for this EMT, depending on whether or not the job has enabled separate I- and D-space. If the job has not enabled separate I- and D-space, then the two forms are equivalent. If the job has mapped to an extended memory region, that will be accounted for by these EMTs. If the specified address

is not mapped to an extended region, then the virtual address is treated as a simple offset from the physical base of the job. When PAR 7 is mapped to simulated RMON or to the I/O page, then PAR 7 is not treated as an extended memory region. This means that, for either form of the EMT, when the specified address is above the valid top of the job or is a PAR7 address mapped either to simulated RMON or to the I/O page, then the EMT will return without error but the physical address returned will probably be erroneous. See Chapter 2 for more information on a job's memory organization and Chapter 9 for more information on mapping to extended memory regions.

If the job has not enabled separate I- and D-space or if the address of interest is mapped to an extended I-space region, the form of the EMT to use is:

```
EMT      375
```

with R0 pointing to the following argument area:

```
.BYTE    0,140
.WORD    virtual-address
.WORD    result-buffer
```

If the job has enabled separate I- and D-space and the address of interest is mapped to an extended D-space region or coincides with an I-space address mapped to an extended I-space region, then the form of the EMT to use is:

```
EMT      375
```

with R0 pointing to an argument block of the form:

```
.WORD    22,140
.WORD    virtual-address
.WORD    result-buffer
```

where *virtual-address* is the virtual address that is to be converted to a physical address and *result-buffer* is the address of a two word area that is to receive the physical address. The low order 16 bits of the physical address are stored in the first word of the result buffer and the high order 6 bits of the physical address are stored in bit positions 4-9 of the second word of the result buffer.

Example

```
.TITLE  PHYADD
.ENABL  LC
; Demonstrate TSX--Plus EMT to convert a virtual to a physical address
.MCALL  .PRINT,.EXIT
START:  MOV    #LOKJOB,R0      ;Point to EMT arg block to
      EMT     375             ;Lock job in memory
      BCC     1$              ;Continue if OK
      .PRINT  #NOPRIV         ;Explain the problem
      BR      2$              ;and quit
1$:     .PRINT  #STRADD        ;Say where this program was loaded
      MOV     #START,R0       ;Get the virtual address
      CALL    PRTADD          ;and display it to the terminal
2$:     .EXIT
PRTADD: MOV     R0,VIRADD       ;Put the virtual address into the arg block
      MOV     #PHYADD,R0      ;Point to EMT arg block to
      EMT     375             ;Convert virtual to physical address
      MOV     BUFFER,R1       ;Retrieve the low address
      MOV     <BUFFER+2>,R2    ;And the high address
      BIC     #~C1760,R2      ;Use only bits 4-9
      MOV     #4,R3           ;Throw away bits 0-3
1$:     ASR     R2
      SOB     R3,1$
```



```

        MOV    #CHREND,R4      ;Point to end of character output buffer
        MOV    #8,R3          ;Need to convert 8 digits
2$:     MOV    R1,R0           ;Get a copy of low bits into R0
        BIC    #~C7,R0        ;Mask out all but low digit
        ADD    #'0,R0         ;Convert to ASCII
        MOVB   R0,-(R4)        ;Put low digit in output buffer
        MOV    #3,R5          ;Shift over 3 bits to next digit
3$:     ASR    R2              ;Low bit of high address
        ROR    R1              ;into high bit of low address
        SOB    R5,3$          ;Do all 8 digits
        SOB    R3,2$          ;Print out the result
        .PRINT R4
        RETURN
LOKJOB: .BYTE   13,140         ;EMT arg block to lock job in memory
PHYADD: .BYTE   0,140         ;EMT arg block to determine physical address
VIRADD: .WORD    0            ;Virtual address to be located
        .WORD   BUFFER        ;Location of 2 word result buffer
BUFFER: .WORD    0,0          ;Will hold low and high physical address
        .BLKB   10            ;Buffer to hold ASCII value of address
CHREND: .WORD    0            ;End of ASCII buffer
        .NLIST BEX
NOPRIV: .ASCIZ   /User not privileged./
STRADD: .ASCII   /This program is loaded at START = /<200>
        .END    START

```

8.8 Specifying a program-abort device reset list

The standard RT-11 .DEVICE EMT is used by TSX-Plus real-time jobs to specify a list of device control registers to be loaded with specified values when the job terminates. This feature is useful in allowing real-time devices to be turned off, if the real-time control program aborts. The TSX-Plus .DEVICE EMT has the same form and options as the standard RT-11 .DEVICE EMT. Use of this EMT requires REALTIME privilege. See the *RT-11 Programmer's Reference Manual* for further information.

The connection between interrupt vectors and interrupt service routines or interrupt completion routines is automatically dropped when a job terminates. The interrupt vector is then connected to the system unexpected interrupt routine. If an address specified in a .DEVICE list corresponds to an interrupt vector connected to the job during job cleanup, then the vector is not connected to the unexpected interrupt routine.

8.9 Reading the Processor Status Word (PSW)

Because of the variety of ways access to the PSW has been implemented in different PDP-11 processors, the current value is normally obtained with the .MFPS macro, which calls a subroutine in the resident monitor to obtain the value. Under TSX-Plus this monitor routine always returns the value 0 (current mode *user*, previous mode *user*, general register set 0, priority 0, and all condition code and trace bits clear), which would be the normal case for user jobs. Note that .MFPS will fail with a trap to 4 when the job is not currently mapped to simulated RMON (e.g. virtual jobs and jobs which have mapped PAR 7 to the I/O page or to a shared run-time). Sometimes, it is desirable to obtain the actual PSW value (for example, when experimenting with supervisor mode). The following request may be used to obtain the current PSW value:

```
EMT      375
```

On entry, R0 should point to an argument block of the form:

```
.BYTE    23,140
```

The current value of the PSW, as pushed onto the stack by the EMT instruction, is returned in R0. This request returns no errors and will work even if the job is not mapped to RMON.

8.10 Setting processor priority level

The following EMT allows a program to set the processor priority level. Use of this EMT requires REALTIME privilege. This EMT can be useful in a situation where it is necessary for a real-time job to block interrupts for a short period of time while it is performing some critical operation. On return from this EMT, the job is executing in user mode with the specified processor priority level. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following EMT argument block:

```
.BYTE    16,140
.WORD    prio-level
```

where *prio-level* should be 7 to cause interrupts to be disabled or 0 to re-enable interrupts. Interrupts should not be disabled for a long period of time or clock interrupts and terminal I/O interrupts will be lost.

Note that it is not possible to raise the processor priority level by storing directly into the processor status word from a real-time program because the PDP-11 hardware disallows modification of the PSW by a user mode program even if the program has access to the I/O page.

Example

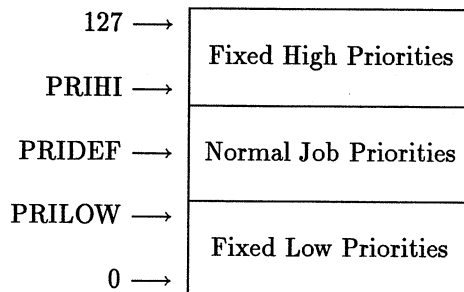
See the example program ATTVEC in section 8.12.2 on connecting a real-time interrupt to a completion routine.

8.11 Setting Job Execution Priority

Jobs may be assigned priority values in the range 0 to 127 to control their execution scheduling relative to other jobs. The priority values are arranged in three groups:

- the fixed-low-priority group consists of priority values from 0 up to the value specified by the PRILOW sysgen parameter
- the fixed-high-priority group ranges from the value specified for the PRIHI sysgen parameter up to 127
- the middle priority group ranges from (PRILOW+1) to (PRIHI-1)

The current values for PRIHI and PRILOW may be determined with the SHOW PRIORITY command, or from within a program with a .GVAL request. The following diagram illustrates the priority groups:



Job scheduling is performed differently for jobs in the fixed-high-priority and fixed-low-priority groups than for jobs with normal interactive priorities. Jobs with priorities in the fixed-low-priority group (0 to PRILOW) and the fixed-high-priority group (PRIHI to 127) execute at fixed priority values. That is, the priority

absolutely controls the scheduling of the job for execution relative to other jobs. A job with a fixed priority is allowed to execute as long as it wishes until a higher priority job becomes active.

The fixed-high-priority group is intended for use by real-time programs. The fixed-low-priority group is intended for use by very low priority background tasks. Normal time-sharing jobs should not be assigned priorities in either of the fixed priority groups.

The middle group of priorities from (PRILOW+1) to (PRIHI-1) are intended to be used by normal, interactive, time-sharing jobs. Jobs with these assigned priorities are scheduled in a more sophisticated manner than the fixed-priority jobs. In addition to the assigned priority, external events such as terminal input completion, I/O completion, and timer quantum expiration play a role in determining the effective scheduling priority.

When a job with a normal priority switches to a subprocess, the priority of the disconnected job is reduced by the amount specified by the PRIVIR sysgen parameter. This causes jobs that are not connected to terminals to execute at a lower priority than jobs that are. This priority reduction does not apply to jobs with priorities in the fixed-high-priority group or the fixed-low-priority group. The priority reduction is also constrained so that the priority of jobs in the normal job priority range will never be reduced below the value of (PRILOW+1).

The following EMT can be used to set the job priority from within a program. The maximum priority which may be used by a job is set by the system manager. The job priority can also be set from the keyboard with the SET PRIORITY command. The current job priority, maximum allowed priority, and fixed-high-priority and fixed-low-priority boundaries may be determined with the .GVAL request. See the *TSX-Plus System Manager's Guide* for more information on the significance of priority in job scheduling. The form of this EMT is:

```
EMT      375
```

with R0 pointing to the following EMT argument block:

```
.BYTE    0,150
.WORD    value
```

where *value* is the priority value for the job. The valid range of priorities is 0 to 127 (decimal). The maximum job priority may be restricted through the logon mechanism. If a job attempts to set its priority above its maximum allowed priority, its priority will be set to the maximum allowed. This EMT does not return any errors.

Example

See Chapter 4 for an example of the use of this EMT.

8.12 Connecting interrupts to real-time jobs

One of the most important uses for real-time jobs is to service interrupts for non-standard devices. TSX-Plus provides three mechanisms for connecting user-written real-time programs to interrupts: device handlers, interrupt service routines, and interrupt completion routines. Interrupt service routines and interrupt completion routines permit jobs to process interrupts without the necessity of writing a special device handler. However, there are restrictions on the use of these two methods which must be considered when deciding the appropriate method for handling special device interrupts.

REALTIME privilege is required to connect either interrupt service routines or interrupt completion routines to an interrupt vector or to release an interrupt connection.

The fastest method of handling interrupts is to write a device handler (driver). Device handlers execute in kernel mode and provide the fastest possible response to interrupts. A device handler is the best choice

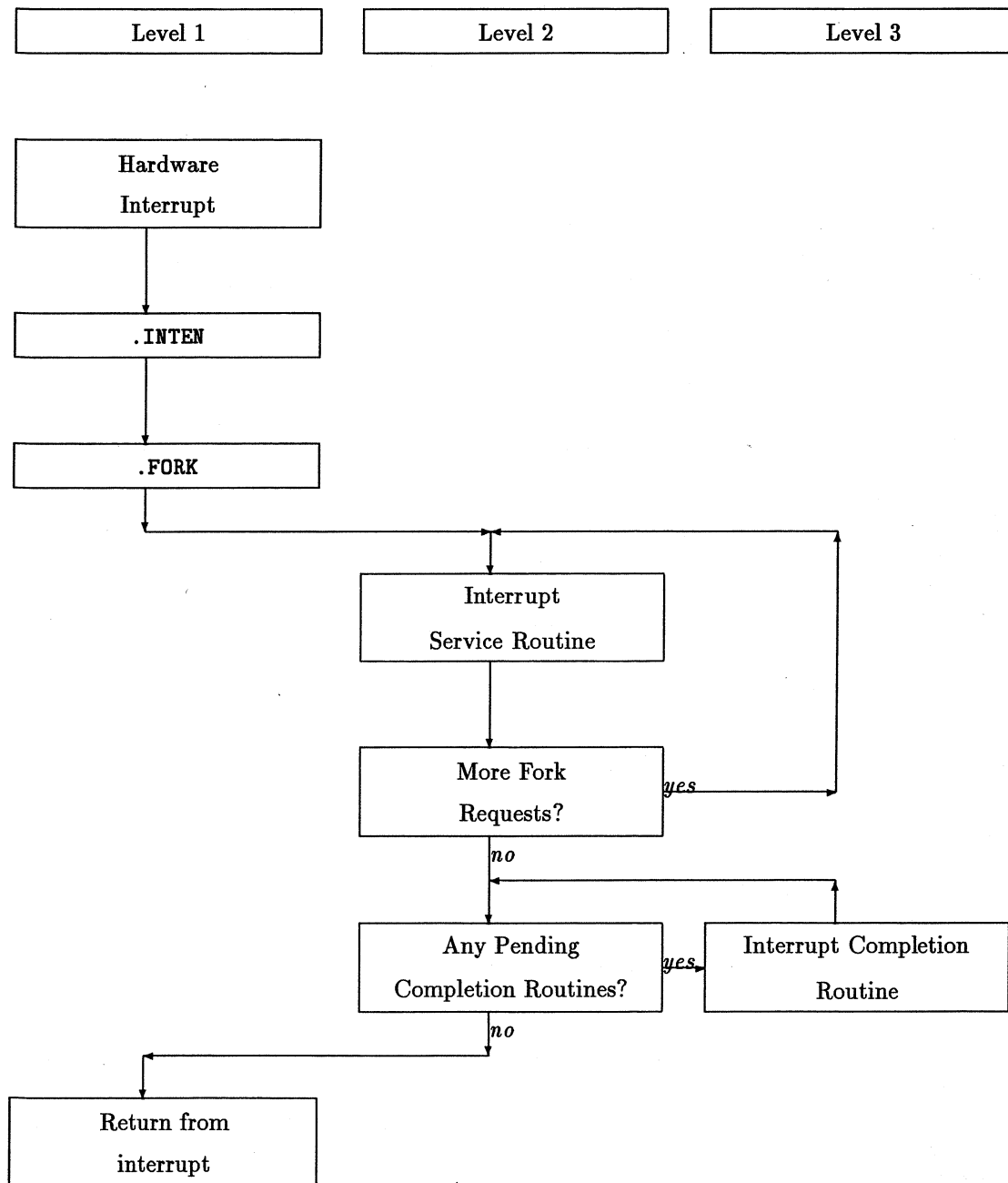
when interrupts occur at a rate which significantly loads the system. Device handlers written for use with TSX-Plus should conform to the rules for writing device handlers for RT-11XM.

The second method for processing real-time interrupts is to connect the interrupt to a real-time interrupt service routine. The interrupt service routine is written as a subroutine in a real-time job; it is not necessary to write a separate device handler. Interrupt service routines are connected with minimal system overhead and can service interrupts quite rapidly; approximately 2000 per second on an 11/44. Interrupt service routines are called in user mode but can only execute a limited set of system service calls (EMTs). Interrupt service routines can trigger the execution of interrupt completion routines to perform additional processing.

The third method for processing real-time interrupts is to connect the interrupt to an interrupt completion routine. Real-time interrupt completion routines have access to the full job context and can use most system service calls (EMTs), but have more overhead and should not be used for interrupts that occur more frequently than 200 times per second.

The following diagram illustrates the different levels of interrupt processing:

Interrupt Processing



This diagram shows that there are three *levels* of interrupt processing. Level 1 is entered when a hardware interrupt occurs. In this level the processor (hardware) priority is set to 7 which causes other interrupt requests to be temporarily blocked. After some brief interrupt entry processing, the system performs a **.FORK** operation which queues a request for processing at fork level and then drops the processor priority to 0. At this time another hardware interrupt can occur, in which case the cycle will be repeated and another request for fork level processing will be placed on the queue. Note that if **.FORK** requests are issued at a sustained high rate, such that numerous prior requests cannot be serviced, a system error may eventually occur.

Level 2 processing is also known as *fork level* processing. This level of interrupt processing services requests that were placed on a queue by the .FORK operation. Hardware interrupts are enabled during this processing and if any other interrupts occur their .FORK requests are placed at the end of the queue. Interrupt service requests are processed serially in the order that the interrupts occurred. Only two system service calls may be used from service routines running at fork level: a request to queue a user completion routine for subsequent processing; and the .RSUM EMT. Fork level processing also has associated priority levels. Real-time processing has the highest priority. For more information on prioritized .FORK requests, see the *TSX-Plus System Manager's Guide*.

Level 3 processing occurs in *job state*. That is, the TSX-Plus job execution scheduler selects the highest priority job or completion routine and passes execution to it. Completion routines run with full job context and may issue system service calls (except USR calls) as needed. Completion routines are serialized for each job. That is, all other completion routines (including higher priority interrupt completion routines) which are scheduled for the same job are queued for execution and will not be entered until the current completion routine exits. During level 3 processing, interrupts are enabled and job execution may be interrupted to process fork level interrupt service routines or by higher priority completion routines for other jobs.

8.12.1 Interrupt service routines

Interrupt service routines execute in user mode but require a minimal amount of system overhead. When an interrupt is received through a vector which has been connected to an interrupt service routine, TSX-Plus executes a .INTEN and a .FORK, sets up memory management for the appropriate job, saves the status of the floating point unit (FPU) if it is in use, and passes control to the interrupt service routine. Using this approach, interrupts may be serviced at about 2000 per second on a PDP-11/44. Lower rates should be expected on slower processors.

Several restrictions apply to this method of interrupt processing:

1. The job must be locked in memory before connecting to the interrupt vector and must remain locked in memory as long as the interrupt connection is in effect.
2. The processing done by the interrupt service routine should be brief since other interrupts that do .FORKs will be queued until the interrupt service routine exits.
3. Only two system service calls (EMTs) are valid within a interrupt service routine:
 - A .RSUM EMT may be issued to cause the job's mainline code to continue processing if it has done a .SPND.
 - The TSX-Plus EMT which schedules execution of a completion routine.

Access to the I/O page is possible if the mainline code sets up such mapping prior to the interrupt. Registers are undefined on entry to an interrupt service routine, and do not have to be preserved by the interrupt service routine.

An interrupt service routine must exit with a RETURN instruction (RTS PC), *not an RTI*. Since interrupt service routines execute at fork level, job scheduling is not relevant for them. All fork level processing, whether queued for system processing or for an interrupt service routine, is executed in the order in which the interrupts were received and execute before any completion routines, fixed-priority jobs or normal interactive time-sharing jobs.

Use of this EMT requires REALTIME privilege. The form of the request to connect an interrupt service routine to a vector is:

EMT 375

with R0 pointing to the following argument block:

```

.BYTE 20,140
.WORD vector-address
.WORD service-routine
.WORD 0

```

where *vector-address* is the address of the interrupt vector to which the service routine is to be connected, and *service-routine* is the address of the entry point of the interrupt service routine.

The total number of vectors that can be connected either to interrupt service routines or interrupt completion routines is determined by the RTVECT parameter during TSX-Plus system generation.

Error Code	Meaning
0	Real time not included or job not privileged
1	Maximum number of vectors already in use
2	Some other job is already connected to that vector
3	Job is not locked in memory

The association between an interrupt service routine and an interrupt vector may be released by the same EMT as used to disconnect an interrupt completion routine. See the section on releasing an interrupt connection later in this chapter.

Example

```

.TITLE INTSVC
.ENABL LC
; Demonstrate EMT to implement an interrupt service routine
; Simple file capture program. Steal a time-sharing line
; and put everything that comes in on it into a file.
.MCALL .TTYIN,.TTYOUT,.EXIT,.PEEK,.POKE,.GVAL,.SCCA
.MCALL .ENTER,.LOOKUP,.CSISPC,.WRITE,.PURGE,.CLOSE,.DEVICE
VECTOR = 330 ;Serial line vector
RCSR = 176530 ;Serial line RCSR address
RBUF = RCSR+2 ;Serial input buffer address
BUFSIZ = 256. ;Size of buffer in words
INTNBL = 100 ;Interrupt enable bit in RCSR
CTRLC = 3 ;ASCII ^C
CTRLD = 4 ;ASCII ^D
START: .GVAL #AREA,*-6. ;See if job has SYSPRV
TST R0 ;1 if priv, 0 if not
BEQ QUIT ;Immediate exit if not priv
MOV #LOKJOB,R0 ;Point to EMT arg block to
EMT 375 ;Lock job in memory
MOV #MAPIO,R0 ;Point to EMT arg block to
EMT 375 ;Map PAR7 into the I/O page
GETFIL: MOV SP,SPSAVE ;Save SP prior to CSI call
1$: .CSISPC #OUTSPC,#DEFEXT,#0 ;Get file specification
BCS 1$ ;Repeat until valid command line
MOV SPSAVE,SP ;Restore SP, no valid switches
.ENTER #AREA,#0,#INSPC,*-1 ;Open largest possible output file
BCS 1$ ;On error, ask for new file
MOV #BUFF1,BUFPTR ;Initialize buffer pointer
CLR BLOCK ;Initialize output file block count
; Set up to accept terminal commands
; ^C, ^C to abort; ^D to end transmission
.SCCA #AREA,*TSTAT ;Disable ^C abort
MOV #ACTCTD,R0 ;Point to EMT arg block to
EMT 375 ;Activate input on ^D
; Since we are going to borrow a current time-sharing line, we need
; to save its vector pointers for later restoration
MOV @#RCSR,CSRSAV ;Save old status bits, esp. int. enable
.DEVICE #AREA,#DEVLST ;Force restoration of RCSR on exit

```

```

        BIC      #INTNBL,@#RCSR ;Disable interrupts until ready
        .PEEK    #AREA,#VECTOR ;Get normal vector pointer
        MOV      RO,VECSAV      ;And save it for later restoration
        .PEEK    #AREA,#VECTOR+2 ;Get normal priority
        MOV      RO,VECSV2      ;And save it for later restoration
; Now attach interrupt service routine to input vector
        MOV      #INTSVC,RO     ;Point to EMT arg block to
        EMT      375            ;Schedule interrupt service routine
GO:      BIS      #INTNBL,@#RCSR ;Enable interrupts and wait for input
        .TTYIN   INBUF          ;Wait for terminal command during transfer
; Resume here when transfer is complete, or want to abort
WHOA:    BIC      #INTNBL,@#RCSR ;Disable interrupts (data lost if mistake)
        CMPB     INBUF,#CTRLC   ;Was it a ^C?
        BNE      5$            ;Branch if not
        .PURGE    #0            ;If ^C, throw away input
        TST      TTSTAT         ;Did we get double ^C's?
        BEQ      GETFIL        ;If not, get another file name
        BR       QUIT          ;If double ^C, then abort
5$:      CMPB     INBUF,#CTRLD   ;Is the transmission done?
        BNE      GO            ;Branch if not
        CALL     FINISH         ;Write out remainder of current buffer
        .CLOSE    #0            ;If done, close out file
; Done or abort, restore time-sharing line conditions
QUIT:    MOV      #RELVEC,RO     ;Point to EMT arg block to
        EMT      375            ;Release interrupt connection
        .POKE    #AREA,#VECTOR+2,VECSV2 ;Restore old priority
        .POKE    #AREA,#VECTOR,VECSAV ;And old vector pointer
        .EXIT      ;And done!
; Interrupt service routine, executes at .FORK level
ISR:     MOVB     @#RBUF,@BUFPTR ;Put char in buffer
        INC      BUFPTR        ;And point to next location
        CMP      BUFPTR,#BUFF2 ;Which buffer in use?
        BHI      1$            ;Branch if already in second
        BLO      9$            ;Return if first not full
        MOV      #BUFF1,WRTPTR ;Exact end of buffer 1, point to it
        BR       2$            ;Schedule write
1$:      CMP      BUFPTR,#BUFEND ;Second buffer full yet?
        BLO      9$            ;Not yet, return
        MOV      #BUFF1,BUFPTR ;Second buffer full, reset pointer
        MOV      #BUFF2,WRTPTR ;Point to second buffer for write
2$:      MOV      #SCHWRT,RO     ;Point to EMT arg block to
        EMT      375            ;Schedule completion routine
        ;to write buffer to file
9$:      TST      TTSTAT         ;Does mainline want to quit? (^C^C)
        BEQ      10$           ;Branch if not
        BIC      #INTNBL,@#RCSR ;If so, disable interrupts
10$:     RETURN                ;Wait for another char
        ;Note RTS PC, not RTI !
; Completion routine to save buffer to file
CMPRTN:  .WRITE   #AREA,#0,R1,#256,BLOCK ;Write buffer to file
        BCC      1$            ;Br if no error
        MOVB     CTRLC,INBUF    ;On error, set abort flags
        MOV      #-1,TTSTAT     ;Skip .TTYIN, and abort
1$:      INC      BLOCK          ;Point to next output block
        RETURN
; Routine to complete write of last block
FINISH:  MOV      BUFPTR,RO     ;Get current buffer pointer
        MOV      #BUFF2,R1      ;Point to end of first buffer
        CMP      RO,R1          ;See which buffer in use
        BLO      1$            ;Skip if in first
        ADD      #2*BUFSIZ,R1   ;If in second, point to end
1$:      CLRB     (RO)+          ;Zero out buffer
        CMP      RO,R1          ;All the way to the end
        BLO      1$            ;
        SUB      #2*BUFSIZ,R1   ;Now point back to buffer beginning
        CALL     CMPRTN         ;Call write routine (NOT AS COMPLETION)
        RETURN
; EMT arg block areas

```



```

AREA: .BLKW 10 ;General EMT arg block
LOKJOB: .BYTE 13,140 ;EMT arg block to lock job in mem
MAPI0: .BYTE 5,140 ;EMT arg block to map I/O page to PAR 7
ACTCTD: .BYTE 0,152 ;EMT arg block to
        .WORD 'D ;Set activation character
        .WORD CTRLD ;~D
INTSVC: .BYTE 20,140 ;EMT arg block to set up interrupt svc routine
        .WORD VECTOR ;Vector to attach
        .WORD ISR ;Address of Interrupt Service Routine
        .WORD 0 ;Required
SCHWRT: .BYTE 21,140 ;EMT arg block to sched compl routine
        .WORD CMPRTN ;Write buffer contents to file
        .WORD 7 ;Real-time priority (adds to PRIHI)
WRTPTR: .WORD 0 ;Passed in R1 to compl routine (buffer addr)
        .WORD 0 ;Required
RELVEC: .BYTE 12,140 ;EMT arg block to release interrupt connection
        .WORD VECTOR ;Vector to be released
; General storage
OUTSPC: .BLKW 15. ;CSI output file specs
INSPC: .BLKW 24. ;CSI input file specs
DEFEKT: .WORD 0,0,0,0 ;No default file specs
SPSAVE: .WORD 0 ;Save stack pointer during CSI call
BLOCK: .WORD 0 ;Output file block counter
TTSTAT: .WORD 0 ;SCCA terminal status word
VECSAV: .WORD 0 ;Save original vector contents
VECSV2: .WORD 0 ; and priority
DEVLST: .WORD RCSR ;.DEVICE restoration list
CSRSV2: .WORD 0 ;To hold original value of RCSR
        .WORD 0 ;End of list
BUFPTR: .WORD 0 ;Current input char pointer
INBUF: .BYTE 0,0,0,0 ;Terminal command buffer
BUFF1: .BLKW BUFSIZ ;1 block input buffer
BUFF2: .BLKW BUFSIZ ;Second buffer
BUFEND:
        .END START

```

8.12.2 Interrupt completion routines

The TSX-Plus real-time support facility allows a program to connect a real-time interrupt to a completion routine. If this is done, TSX-Plus schedules the completion routine to be executed each time the specified interrupt occurs.

Interrupt completion routines have much greater flexibility than interrupt service routines, but require more overhead and are only capable of servicing interrupts at a lower rate. Interrupt completion routines run with full job context. This allows them to use all system service calls (except to the USR). Registers will be preserved between calls to the completion routine.

Real-time completion routines can service interrupts at rates up to about 200 interrupts per second. Devices which interrupt at a faster rate should be connected through an interrupt service routine or through a special device handler.

The total number of interrupt vectors that can be connected either to interrupt completion routines or interrupt service routines is determined by the RTVECT parameter during TSX-Plus system generation. It is possible for several interrupts to be connected to the same completion routine in a job but it is illegal for more than one job to try to connect to the same interrupt. When an interrupt completion routine is entered, R0 contains the address of the interrupt vector that caused the completion routine to be executed.

Real-time completion routines, whether directly connected to an interrupt or scheduled with the EMT for that purpose, have an associated job priority. These are software priorities, not hardware priorities; all completion routines are synchronized with the job and execute at hardware priority level 0. Completion routine priorities 1 and larger are classified as *real-time* priorities and are added to the system parameter PRIHI to yield the job execution priority, unless the resultant priority would exceed 127 in which case the priority will be 127. These completion routines will then be scheduled for execution whenever there are no executable jobs with a higher priority.

A real-time completion routine for one job will be suspended if an interrupt occurs which causes a higher priority completion routine to be queued for another job. However, a real-time completion routine for one job will never be interrupted by a completion routine for that same job regardless of the subsequent completion routine's priority. If additional requests are made to trigger the same or different completion routines while a completion routine is executing, the requests are queued for the job and are serviced in order based on their priority and the order in which they were queued.

A real-time completion routine is allowed to run continuously until one of the following events occurs:

1. the completion routine finishes execution and returns
2. a higher priority completion routine from another job interrupts its execution—it is re-entered when the higher priority routine exits
3. the completion routine enters a system wait state such as waiting for an I/O operation to complete or waiting for a timed interval. If a real-time completion routine enters a wait state, it returns to the same real-time priority when the wait condition completes.

Real-time completion routines with a priority of 0 are treated slightly differently than those with priorities greater than zero. The action taken depends on the priority of the mainline job when the completion routine is scheduled. If the base priority is equal to or greater than PRIHI, then the completion routine is treated just as those with real-time priorities greater than zero. That is, the completion routine is assigned a priority of PRIHI (PRIHI+the real-time priority of 0). On the other hand, if the base job priority is less than PRIHI, then the job is scheduled as a very high priority normal (non-interactive) job. The effect of this is that completion routines with a real-time priority of 0 and a base job priority less than PRIHI will interrupt normal time-sharing jobs, but are time-sliced in the normal fashion and lose their high priority if they execute longer than the system parameter QUAN1A.

Jobs that have real-time, interrupt completion routines need not necessarily be locked in memory. If an interrupt occurs while the job is swapped out of memory, it is scheduled for execution like any other job and swapped in before the completion routine is executed. Note, however, that this condition is not optimal for timely servicing of interrupts.

When a real-time interrupt occurs, a request is placed in a queue to execute the appropriate completion routine. If the interrupt occurs again before the completion routine is entered, another request is placed in the queue so the completion routine will be invoked twice. A TSX-Plus fatal system error occurs if an interrupt occurs and there are no free completion routine request queue entries.

When a real-time completion routine completes its execution, it must exit by use of a RETURN instruction (RTS PC), *not an RTI*.

Use of this EMT requires REALTIME privilege. The form of the EMT used to connect an interrupt to a real-time completion routine is:

EMT 375

with R0 pointing to the following argument area:

```
.BYTE    11,140
.WORD    interrupt-vector
.WORD    completion-routine
.WORD    priority
```

where *interrupt-vector* is the address of the interrupt vector, *completion-routine* is the address of the completion routine, and *priority* is the execution priority for the completion routine.

Error Code	Meaning
0	Real-time support not included or job not privileged
1	Maximum number of vectors already in use
2	Some other job is already connected to that vector
3	Job is not locked in memory

Example

```

        .TITLE  ATTVEC
        .ENABL  LC
; Demonstrate TSX--Plus EMTs to attach a completion routine to an interrupt
CTRLC = 3          ;~C
ERRBYT = 52
RCSR = 176540      ;Serial line RCSR address
RBUF = RCSR+2      ;Line input buffer address
INTNBL = 100       ;Interrupt enable bit in RCSR
        .MCALL  .PRINT,.EXIT,.TTYOUT,.SPND,.RSUM
START:  MOV     #MAPIOP,RO      ;Point to EMT arg block to
        EMT     375            ;Map to I/O page
        BCC     1$            ;Branch if OK to map
        .PRINT  #NOPRIV        ;You are not allowed to do that
        .EXIT

;+
;If this job were to be resident for a long time, it would be better
;to lock into low memory to avoid memory fragmentation by job swapping.
;However, since this job may have to be swapped now to get into low
;memory, locking into low memory takes longer to execute.
;
1$:      MOV     #LOKLOW,RO      ;Point to EMT arg block to lock into low mem.
;-
1$:      MOV     #LOKJOB,RO      ;Point to EMT arg block to
        EMT     375            ;Lock the job in memory
        MOV     #ATTVEC,RO      ;Point to EMT arg block to
        EMT     375            ;Attach to an interrupt vector
        BCC     2$            ;Continue if OK
        .PRINT  #BADATT        ;Notify cannot attach to interrupt
        MOVB    @ERRBYT,RO      ;Find out which error
        ASL     RO             ;Convert to word offset
        .PRINT  ATTERR(RO)      ;And explain reason for error
        BR      3$            ;Go on to unlock and exit
2$:      .PRINT  #TYPE          ;Prompt for input from interrupting device
        BIS     #INTNBL,@RCSR   ;Enable input interrupts
        .SPND                    ;Now wait for an interrupt
; . . .

        MOV     #RELVEC,RO      ;Resume here on exit from completion code
        EMT     375            ;Release an interrupt vector
;+
; NOTE: Any interrupts through this vector after it has been released
; will cause a TSX--Plus fatal system error - Unexpected Interrupt.
;-
3$:      MOV     #UNLOKJ,RO      ;Point to EMT arg block to
        EMT     375            ;Unlock this job from memory
        .EXIT

;+
; The following code will be executed as a completion routine
; when the device attached to the vector interrupts.
;-
CMPRTN: MOVB    @RBUF,RO        ;Enter here when new character is available
        CMPB    RO,#CTRLC      ;Get the new character
        BEQ     1$            ;Should we quit?
        .TTYOUT                    ;Quit on ~C
        MOV     #SETPRI,RO      ;Display the character
        EMT     375            ;Point to EMT arg block to
; . . .
        CLR     PRILEV          ;Set processor priority (block interrupts)
        MOV     #SETPRI,RO      ;Time critical processing goes here
        EMT     375            ;Set priority back down
        BR      2$            ;Point to EMT arg block to
1$:      .RSUM                    ;Set processor priority (re-enable interrupts)
2$:      RETURN                    ;Return to mainline code
MAPIOP: .BYTE    5,140          ;Wait for next interrupt (NOTE: Not RTI)
        ;EMT arg block to map to I/O page

```

```

LOKLOW: .BYTE 7,140      ;EMT arg block to lock job into low mem.
LOKJOB: .BYTE 13,140     ;EMT arg block to lock job in place
UNLOKJ: .BYTE 10,140     ;EMT arg block to unlock job from memory
ATTVEC: .BYTE 11,140     ;EMT arg block to attach to interrupt
        .WORD 340        ;Interrupt vector
        .WORD CMPRTH     ;Address of completion routine
        .WORD 7          ;Real-time priority (NOT processor priority!)
RELVEC: .BYTE 12,140     ;EMT arg block to release interrupt vector
        .WORD 340        ;Interrupt vector
SETPRI: .BYTE 16,140     ;EMT arg block to set processor priority
PRILEV: .WORD 7          ;Desired priority level (modifiable)
ATTERR: .WORD NOPRIV     ;Attach to interrupt error table
        .WORD MAXINT
        .WORD INUSE
        .MLIST BEX
TYPE:   .ASCIZ /Characters entered on serial line will be displayed here:/
BADATT: .ASCIZ /?ATTVEC-F-Cannot attach to interrupt./<7>
NOPRIV: .ASCII /Real-time support not specified during TSGEN or /
        .ASCIZ /user not privileged./
MAXINT: .ASCIZ /Maximum number of interrupts already in use./
INUSE:  .ASCIZ /Another job already connected to interrupt./
        .END START

```

8.13 Releasing an interrupt connection

A connection between an interrupt vector and an interrupt service routine or an interrupt completion routine remains in effect until the job exits or the following EMT is executed to release the connection. Use of this EMT requires REALTIME privilege.

Warning: An interrupt through a vector which has been released with this EMT or which was connected to a job that has terminated will cause a fatal system error: "UEI-Interrupt occurred at unexpected location". This can be circumvented by including the vector in a .DEVICE list.

The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument area:

```

.BYTE 12,140
.WORD interrupt-vector

```

where *interrupt-vector* is the address of the interrupt vector whose connection is to be released.

Example

See the example program ATTVEC in section 8.12.2 on connecting a real-time interrupt to a completion routine.

8.14 Scheduling a completion routine

Real-time programs may schedule a completion routine for execution with a special EMT provided for that purpose. This is particularly valuable from within interrupt service routines which do not have access to system service calls other than this EMT and the .RSUM request. The primary intent of this request is to provide a mechanism for access to system service calls from interrupt service routines. The form of this EMT is:

```
EMT      375
```

with R0 pointing to the following argument block:

```
.BYTE    21,140
.WORD    completion-routine
.WORD    priority
.WORD    R1-value
.WORD    0
```

where *completion-routine* is the address of the entry point of the completion routine that is being started, *priority* is the real-time priority level for the completion routine being started, and *R1-value* is a value to be placed in R1 on entry to the completion routine.

Completion routines scheduled in this manner follow the same rules as for interrupt completion routines described above, except that they are scheduled as a result of the EMT call rather than in response to an interrupt.

Example

See the example program INTSVC in section 8.12.1 on connecting interrupts to real-time jobs with interrupt service routines.

8.15 Adapting real-time programs to TSX-Plus

The following points should be kept in mind when converting an RT-11 real-time program for use under TSX-Plus.

- The I/O page (160000–177777) is not directly accessible to the program unless the program executes the TSX-Plus real-time EMT that maps the job's virtual region to the I/O page or is started with the /IOPAGE switch to the R[UN] command.
- If the program's virtual region 160000 to 177777 is mapped to the I/O page, the program must use .GVAL to access offsets in the simulated RMON.
- Since FORTRAN uses PAR 7 to map to virtual arrays, if both direct I/O page access and FORTRAN virtual arrays are required in the same program, then some PAR other than 7 must be used to map to the I/O page.
- Real-time interrupts are connected to interrupt service routines and interrupt completion routines by use of the TSX-Plus real-time EMT for that purpose. The program should not try to connect interrupts by storing into the interrupt vector cells.
- The .PROTECT EMT is ignored under TSX-Plus and always returns with the carry flag cleared.
- Interrupt service routines and completion routines connected to real-time interrupts should exit by use of a RTS PC instruction rather than RTI.
- Programs that require very rapid response to interrupts should use the interrupt service routine method and must lock themselves in memory.
- Real-time interrupts must not occur unless an interrupt service routine or completion routine is connected to the interrupt. If a real-time interrupt occurs with no associated interrupt service or completion routine, a fatal TSX-Plus system error (UEI) occurs and the *argument value* displays the address of the vector of the interrupt.
- A higher priority real-time completion routine for one job will interrupt a lower priority completion routine being executed by another job but will not interrupt a lower priority completion routine being executed by the same job.
- A real-time completion routine running at priority 1 or above is not time-sliced and will lock out all lower priority jobs until it completes its processing or enters a wait state.

Chapter 9

Extended Memory Regions

This chapter describes facilities that allow jobs to extend their memory addressing capacity.

Shared run-time regions are regions of extended memory that are reserved and loaded when TSX-Plus is started. They are commonly used to contain shareable, re-entrant code regions for languages such as DBL and COBOL-Plus. They may also be used as shareable data regions. System service requests to support shared run-times are discussed in section 9.1.

PLAS facilities are compatible with the extended memory requests provided by the RT-11XM monitor. They may also be used to contain shareable code or data regions, but are most commonly used to support virtual arrays and virtual program overlays. PLAS regions are not automatically reserved or initialized when the system is started, but are loaded and unloaded under program control. General information on PLAS requests may be found in the *RT-11 Programmer's Reference Manual* and the *RT-11 Software Support Manual*. PLAS features which are unique to TSX-Plus are described in section 9.2.

Both shared run-times and PLAS regions can be quickly mapped or re-mapped using "fast mapping" with the TRAP 1 instruction. Fast mapping facilities are discussed in sections 9.1.4 and 9.2. Both shared run-times and PLAS regions may also be used in conjunction with separate I- and D-space for additional virtual memory addressing capability. Use of separate I- and D-space under TSX-Plus is discussed in section 2.9. See Chapter 2 for more information on a job's virtual memory and addressing environment under TSX-Plus.

9.1 Shared run-times

TSX-Plus provides a facility that allows one or more shared run-time systems or data areas to be mapped into the address space of multiple TSX-Plus time-sharing jobs. There are two primary uses of this facility:

- Memory space can be saved by having multiple jobs that use the same run-time system access a common copy rather than having to allocate space within each job for a copy.
- Programs can communicate with each other through the use of a common shared memory region to which all of the communicating jobs have direct access.

To use this facility, information about all of the shared run-time systems must be declared when the TSX-Plus system is generated. During system initialization the shared run-time system files are opened and read into memory. These shared run-time systems remain in memory as long as the system is running and are never swapped out of memory even if there are no jobs actively using them.

The EMTs described below can be used to associate one or more shared run-time systems with a job. When such an association is made, a portion of the job's virtual memory space is mapped to allow access to part or all of one or more run-time systems.

9.1.1 Associating a run-time system with a job

The following EMT is used to associate a shared run-time system with a job. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument area:

```
.BYTE    0,143
.WORD    name-pointer
```

where *name-pointer* is the address of a two word cell containing the six character name of the shared run-time system in RAD50 form. This name corresponds to the file name which was specified for the run-time system when TSX-Plus was generated. If the name pointer value is zero, the effect of the EMT is to disassociate all shared run-time systems from the job and to re-establish normal memory mapping for the job. Note that disassociating shared run-time systems in this manner also disassociates any existing real-time mapping.

If the run-time system whose name is specified with this EMT is not recognized, the carry flag is set on return with an error code of 1.

The effect of this EMT is to associate a particular shared run-time system with the job. However, this EMT does not affect the memory mapping for the job or make the run-time system visible to the job; that is done by the EMT described below. If some other run-time system has been previously mapped into the job's region, that mapping is unaffected by this EMT. Thus it is possible to have multiple run-time systems mapped into the job's region by associating and mapping them one at a time into different regions of the job's virtual memory space.

Example

```
.TITLE  USERTS
.ENABL  LC
; Demonstration of TSX-Plus EMT to map to a shared run-time region
.MCALL  .PRINT,.EXIT
PAR1    = 20000                ;Base address of PAR 1 window
START:  MOV    #USERTS,R0      ;Point to EMT arg block to
      EMT      375             ;Associate a shared run-time region
      BCC      1$             ;Error?
      .PRINT   #NOSHRT        ;Can't find named shared run-time
      .EXIT
1$:     MOV    #MAPRTS,R0      ;Point to EMT arg block to
      EMT      375             ;Map to shared run-time system
2$:     CALL   @*<PAR1+200>    ;JSR to entry point in shared region
      ;which prints some data from there.
      CLR     <USERTS+2>      ;Undeclare any shared run-times
      MOV     #USERTS,R0      ;Point to EMT arg block to
      EMT      375             ;Dissociate all shared regions
      ;Go on with other processing
      .EXIT
USERTS: .BYTE   0,143          ;EMT arg block to associate
      .WORD   SHRNAM           ;Pointer to file name
      SHRNAM: .RAD50 /RTCOM /  ;Shared region file name
MAPRTS: .BYTE   1,143          ;EMT arg block to map shared region
      .WORD   1                ;Map PAR 1
      .WORD   0                ;Offset into region
      .WORD   1000/64.         ;Size of region (64. byte blocks)
      .NLIST  BEX
NOSHRT: .ASCIZ  /Can't find the shared run-time system./<?>
      .END    START
```

The example program RTCOM declared in the above program was defined during TSX-Plus system generation with the RTDEF macro as follows:


```
RTDEF <SY RTCOM SAV>,RW,1
```

and the shared program itself was:

```
.TITLE RTCOM
.ENABL LC
; Demonstration shared run-time file
; (Position Independent Code.)
.MCALL .PRINT
.MLIST BEX
.PSECT RTCOM,I ;RTDEF in TSGEN specifies
;skip 1 block so default start address of 1000 is OK
START: .ASCII /This data was produced by the /
       .ASCIIZ /RTCOM shared run-time region./
       . = START + 200 ;Entry point for shared code
ENTRY:: MOV PC,RO ;Find out where we are
       SUB #<.-START>,RO ;Point back to data area
       .PRINT RO ;Display what is there
       RETURN ;And go back to MAIN
       .END ENTRY
```

9.1.2 Mapping a run-time system into a job's region

Once a shared run-time system has been associated with a job by use of the previous EMT, the run-time system (or a portion thereof) can be made visible to the job by mapping it into the job's virtual address region. The form of the EMT to do this is:

```
EMT 375
```

with R0 pointing to the following argument area:

```
.BYTE 1,143
.WORD par-region
.WORD run-time-offset
.WORD mapped-size
```

The *par-region* parameter is a number in the range 0 to 7 that indicates the Page Address Register (PAR) that is to be used to access the run-time system. The PAR number selects the region of virtual memory in the job that will be mapped to the run-time system. The following correspondence exists between PAR numbers and virtual address regions within the job:

PAR	Address Range
0	000000-017777
1	020000-037777
2	040000-057777
3	060000-077777
4	100000-117777
5	120000-137777
6	140000-157777
7	160000-177777

The *run-time-offset* parameter specifies which portion of the run-time system is to be mapped into the PAR region. The *run-time-offset* specifies the number of the 64-byte block within the run-time system where the mapping is to begin. This makes it possible to access different sections of a run-time system at different times or through different regions.

The *mapped-size* parameter specifies the number of 64-byte blocks to be mapped. If this value is larger than can be contained within a single PAR region, multiple PAR regions are automatically mapped as necessary to contain the entire specified section of the run-time.

If an error is detected during execution of the EMT, the carry-flag is set on return with an error code of 1 to indicate that there is no run-time system associated with the job.

This EMT only affects the mapping of the PAR region specified in the argument block (and following PARs if the size so requires). It does not affect the mapping of any other PAR regions that may previously have been mapped to a run-time system. Thus a job may have different PAR regions mapped to different sections of the same run-time system or to different run-times. Real-time programs may map PAR 7 to the I/O page and also map other PAR regions to shared run-time systems.

The memory size of a job is not affected by the use of the shared run-time control EMTs. That is, mapping a portion of a job's virtual address space to a shared run-time system neither increases nor decreases the size of memory occupied by the job. If a job's size is such that a portion of its normal memory is under a PAR region that is mapped to a run-time system, that section of its normal memory becomes inaccessible to the job as long as the run-time system mapping is in effect, but the memory contents are not lost and may be re-accessed by disassociating all run-time systems from the job.

Note that any of the PAR regions may be mapped to a run-time system including PAR 7 (160000–177777).

Example

See the example program USERTS in section 9.1.1 on associating a run-time system with a job.

9.1.3 Using I- and D-space with Shared Run-Time Regions

When a program is loaded into memory its mapping registers are set up so that all memory references (both instruction and data) are mapped through I-space. Memory is allocated and mapped from virtual address 000000 up to the high limit of the program. This is called the *static region*. Then, when a program subsequently maps to a shared run-time region, portions of the the I-space mapping are altered to point into the shared run-time as specified. However, although the entire static region is still allocated to the job, portions of it are no longer accessible because they have been “mapped away”. By using separate I- and D-space mapping, these unmapped portions of the static region can be used for data, mapped through D-space, while the same virtual address range is mapped through I-space to the shared run-time. (Note that mapping the shared run-time through D-space, leaving the static region mapped through I-space, is not allowed; use global PLAS regions for shared data regions.)

The normal sequence of events for a program to use the unmapped portions of the static region as data space in conjunction with I-space mapping to a shared run-time region is as follows:

1. When a program is first loaded, only I-space is enabled, and all instruction fetches and all data references are made through I-space.
2. Enable separate I- and D-space (using EMT 375, 4,143). (See Chapter 4.) Both I- and D-space are now mapped identically to the same static region.
3. Associate with a shared run-time. This does not change the mapping.
4. Map a portion of the shared run-time into the program space. At this point, the virtual address range mapped to the shared run-time is split so that instructions will be fetched from the shared run-time and data references and I/O are directed to the remaining static region.
5. (... Do the real work ...)
6. Disassociate from the shared run-time. This restores mapping so that both I- and D-space are again identically mapped to the static region.
7. Disable separate I- and D-space (using EMT 375, 5,143 – see Chapter 4). Now, the static region is again mapped only through I-space and all instruction fetches and data references occur through I-space.

9.1.4 Fast mapping to shared run-time regions

The following EMT allows a program to define a set of "fast map regions" in the current shared run-time system. Up to 40 fast map regions can be defined by each job. The TRAP instruction (rather than EMT) is then used to cause a selected fast map region to be mapped into the virtual address range of the job. The TRAP instruction is used to avoid the overhead involved in general EMT processing. This method of mapping shared run-times is approximately 10 times as fast as the method using EMT 375 function 143, subfunction 1.

The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument area:

```
.BYTE    2,143
.WORD    par-number
.WORD    offset
.WORD    size
.WORD    region-number
```

where *par-number* is the number of the Page Address Register (PAR) through which the fast map region is to be mapped. PAR numbers are in the range 0 to 7 and each PAR can map up to 8 Kb. If the size specified for a fast map region exceeds 8 Kb, then subsequent fast map regions are assigned to cover the entire specified size which will be mapped through succeeding PARs. Different regions can be mapped through different PAR's.

The *offset* specifies the number of 64-byte blocks from the beginning of the shared run-time system where the mapping is to begin.

The *size* specifies the length of the fast map region in 64-byte units. If the size exceeds 8 Kb, then subsequent fast map region numbers and PAR numbers are assigned to cover the entire specified region. If the specified size and offset would cause the fast map region to extend beyond the end of the shared run-time, then the size is truncated to the top of the shared run-time. On return from this EMT, R0 contains the actual size allocated for the region (in 64-byte blocks). If the region size is truncated in this fashion because it would extend beyond the end of the shared run-time, then the EMT may return without error (carry bit clear). Because of this, the actual size (in R0) should be checked against the size requested.

The *region-number* specifies the base index number of the fast map region. This is a number in the range 0 to 39 that identifies the fast map region. The base index number is later used with the TRAP instruction to select the fast map region being mapped. If the region size exceeds 8 Kb, subsequent index numbers are assigned to cover the entire fast map region. When the fast map region size exceeds 8 Kb, several fast map regions will be used to cover the specified size. In this case of multiple PARs, when the TRAP instruction is used to actually perform the mapping, specifying the base fast map region number will cause that region to be mapped as well as all successive fast map regions needed to cover the entire specified fast map region size. The system does not check that previously defined fast map regions are re-used. It is the programmer's responsibility to manage fast map region index allocation.

A region can be redefined to perform different mappings at different times, but most commonly a set of regions is defined during program initialization and then the TRAP instruction is used to select the region that is to be mapped as the program executes.

It is possible to define a set of fast map regions in the current shared run-time, then associate with a different shared run-time and also define a different set of fast map regions within it.

This EMT can return the following error codes:

Code	Meaning
0	Attempt to map past PAR 7
1	No shared run-time has been associated with job
2	Fast map region number is invalid
5	Specified offset is beyond the end of the run-time

The following sequence of operations must be performed to define and map fast map regions within a shared run-time:

1. Associate a shared run-time system with the job by use of the EMT with function code 143, subfunction 0 (see section 9.1.1. This does not cause any portion of the run-time to be mapped into the job's address space but simply specifies which run-time is being referenced by subsequent EMT's.
2. Use EMT 375 with function code 143, subfunction 2, to define the regions of the run-time (see above). This simply defines the characteristics of the regions, it does not cause them to be mapped into the job's address space. You may define regions in different shared run-times by associating a different run-time before you use the define-region EMT.
3. When you are ready to map a region into the job's address space, load R0 with the region number you specified when you defined the region, and execute a "TRAP 1" instruction. The carry flag will be cleared following TRAP if the mapping was successful. Carry flag being set indicates that an invalid region number was specified in R0.

If any fast map regions have been defined by the job, the TRAP instruction is dedicated for use in region mapping. If there are no defined fast map regions, the TRAP instruction can be used for other purposes as before. All run-time and fast map region definitions are cleared when a program exits or chains. In addition, the following EMT can be used to clear all run-time regions and return the TRAP instruction to its normal function:

EMT 375

with R0 pointing to the following argument block:

.BYTE 3,143

This EMT does not return any errors.

9.2 PLAS regions

9.2.1 Fast mapping to PLAS regions — 20 times faster

Fast mapping using the TRAP instruction which was previously restricted to use with shared run-time regions now also supports global PLAS regions. (It may not be used with swappable PLAS regions — either unnamed or private named regions.) When compared with the old method used to map around within a PLAS region (.CRAW with WS.MAP), fast mapping within the PLAS region is about 20 times faster. Fast mapping also tracks whether the region is to be mapped through I-space (the normal case) or through D-space if the region was set up for D-space mapping. Fast mapping may be used in place of the .MAP request. Like the .MAP request, it uses information from the region and window control blocks to define the region to be mapped. This request does not actually perform the mapping, but instead defines the fast mapping values to be used when mapping is requested by the TRAP 1 instruction. The form of the request is:

EMT 375

with R0 pointing to an argument block of the following form:

```
.BYTE    6,143
.BYTE    window-id, fast-map-number
.WORD    region-id
```

The *window-id* is the value returned by the .CRAW request, and is in the range 1–8. The *fast-map-number* is the value to be placed in R0 when the TRAP 1 instruction is executed to identify the fast-map region number, and is in the range 0–39. The *region-id* is the value returned by the .CRRG request. The PLAS region need not be currently mapped in order to associate a fast-map region with it. On successful return, R0 contains the actual length (number of 64-byte blocks) that will be mapped through the window by the TRAP 1 instruction — analogous to WS.LEN which is not maintained when using this mapping method.

See the descriptions of fast mapping to shared run-time regions in section 9.1.4 for more details on the use of fast mapping.

Errors returned by this request are:

<i>Error Code</i>	<i>Meaning</i>
1	Specified window or region is not defined
2	Invalid fast-map-number (must be 0-39.)
3	Invalid window-id (must be 1-8.)
4	Attempt to fast map swappable region

9.2.2 Fast mapping extended to multiple PARs

In the original implementation of fast mapping regions, it was implicitly assumed that the region was 8 Kb or less. Now, it is possible to automatically define several fast-mapping regions and map to them in single operations. When the fast-map region size is defined to exceed 8 Kb, subsequent fast-map regions are also automatically assigned. If the combination of fast-map region number and size of the region would cause subsequent fast-map numbers to overflow the allowed limit (40 regions, numbered 0–39.), then an invalid fast-map region number error code will be returned. In addition, no checking is performed to prevent or warn about overwriting of previously defined fast-map regions. It is up to the user to manage fast-map region number assignments.

When a TRAP instruction is issued to cause fast-mapping to occur, any subsequent fast-map regions which are required to fully map the region are also automatically mapped.

Chapter 10

Program Debugger

Programming errors can sometimes be very difficult to identify and correct without a detailed stepwise examination of a program's progress. Some higher level languages, like COBOL-Plus, include a debugging tool that allows such examination. For programs written in MACRO assembly language or when it is desirable to examine a program at the level of individual instructions, TSX-Plus provides a debugging facility with the following features:

- The debugger is included as an optional system overlay and does not share memory space with the program being debugged. This allows programs up to the full 64 Kb virtual address space to be run with the debugger. It can also be used with programs which are mapped to the I/O page or which use overlays, shared run-time systems, or extended memory regions (PLAS regions; virtual arrays or overlays).
- The debugger does not have to be linked with the program being debugged. It can be invoked when the program is executed by the RUN/DEBUG command.
- A keyboard command can be issued to permit debugging of any subsequent user program or utility even if the program is not started with the RUN/DEBUG command.
- Execution of a program can be interrupted at any time by typing CTRL-D. This causes entry to the debugger similar to hitting a breakpoint in the program.
- The debugger can decode instructions into symbolic assembly language.
- Program traps to 4 and 10 are caught by the debugger rather than causing a program abort with return to the keyboard monitor, unless the program has issued a .TRPSET EMT.
- A data *watchpoint* facility is included which can cause an execution break to occur when the value of a monitored data item is changed.

10.1 Debugger requirements

In order to make effective use of the debugger facility, it is necessary to have a link map of the program being debugged. This minimizes the usefulness of the debugger for interpreted languages (e.g., BASIC) and languages with a run-time system (e.g., DIBOL) unless one is interested in debugging the run-time interpreter. For other high-level languages (e.g., FORTRAN and Pascal) the debugger is most useful for small program sections for which the intermediate assembly code is available.

The debugger facility is an optional feature of TSX-Plus, included during system generation. Debugger support must have been selected during system generation (DBGFLG=1) in order to use this feature.

10.2 Invoking the debugger

The debugger need not be linked with the program being debugged. Programs may either be started under debugger control with the RUN/DEBUG command, or can be interrupted while running by typing CTRL-D if the SET CTRLD DEBUG command has been previously issued.

10.2.1 RUN/DEBUG switch

A program may be started under control of the debugger with the /DEBUG switch to the RUN command. For example:

```
RUN/DEBUG PRGNAM
```

When a program is started under debugger control, the program is loaded and the debugger prompt appears. For example:

```
RUN/DEBUG PRGNAM .  
TSX-Plus debugger  
DBG:
```

Whenever the debugger is in control, the DBG: prompt will appear and the debugger will be ready to accept a command.

On entry to the debugger, the program start address is loaded into R0 and set so that program execution may be initiated with the “;G” command without specifying a starting address. It is usually desirable to set initial values for relocation registers and breakpoints before starting program execution.

10.2.2 CTRL-D Break

CTRL-D normally only interrupts program execution if the program has been started by the RUN/DEBUG command. However, a keyboard command is available to enable CTRL-D to interrupt a program and pass control to the debugger even if the program was not started under debugger control. See the description of the SET CTRLD DEBUG command in the *TSX-Plus User's Reference Manual*. If CTRL-D interruption has been enabled, then any program may be interrupted and control transferred to the debugger regardless of whether the program was started under debugger control. The effect is similar to hitting a breakpoint at the current location. The same debugger commands and facilities are available when the debugger is entered this way as when the program is started under debugger control. Note however, that if the SET CTRLD DEBUG command has not been issued prior to starting a program's execution and that program is not started with the RUN/DEBUG command, then CTRL-D will have no special effect on the program and debugger control cannot be obtained without restarting the program. The SET CTRLD DEBUG command is local to each time-sharing line and must be invoked from that line to enable CTRL-D breakpoints in programs which are not started with the RUN/DEBUG command. If a program is started with the RUN/DEBUG command, then CTRL-D may be used to interrupt its execution regardless of whether the SET CTRLD DEBUG command has been issued.

See the Special Notes section at the end of this appendix for information about interactions of CTRL-D with special terminal mode and special activation characters.

10.3 BPT Instruction

A program which includes a BPT instruction can trigger execution of the system debugger. If a BPT instruction is executed by a program which is not currently associated with the system debugger and location 14 (trap vector for BPT) in the program is zero (0), then the system debugger is entered with the instruction that follows the BPT as the current instruction. If a program is already associated with the system debugger (RUN/DEBUG or by a CTRL-D) when the BPT is executed, then the BPT is ignored and does not cause entry to the debugger.

10.4 Commands

The commands and syntax of the debugging facility are similar to those of RT-11's ODT. However, not all ODT commands are implemented and some additional features are provided. In the following list of debugger commands, angle brackets are used to indicate special terminal keys such as <RETURN>, <TAB> and <LINE FEED>. The angle brackets are not part of debugger command syntax. Many commands accept a numerical value or address, indicated here as *value*, *address*, *repeat* or *n*. The italicized portion should be substituted with the appropriate number when issuing the command. The value is optional in some cases. All numerical values are assumed to be octal unless terminated by a decimal point. The period character (".") may be used to indicate the current value of the program counter; for example, see the **address;nB** command to set an instruction breakpoint.

Debugger Commands

Command	Meaning
<i>address</i> /	Display the contents of the addressed word. Default to D-space if separate I- and D-space are enabled. See the ;nI command below.
<i>address</i> \	Display the contents of the addressed byte. Default to D-space if separate I- and D-space are enabled. See the ;nI command below.
<i>value</i> <RETURN>	Store <i>value</i> into the currently open cell. If <i>value</i> is omitted, close the current location without changing its contents.
<i>value</i> <LINE FEED>	Store <i>value</i> into the currently open cell and then advance to the next word or byte. If <i>value</i> is omitted, advance to the next location without changing the value of the current location.
<i>value</i> ~	Store <i>value</i> into the currently open cell and then advance to the previous word or byte. If <i>value</i> is omitted, back up to the previous location without changing the contents of the current location.
<BACKSPACE>	Equivalent to up-arrow ("~").
@	Open the cell whose address is specified by the contents of the currently open cell.
- (underscore)	Open the cell whose address is specified by the contents of the currently open cell plus the current location plus 2.
<i>address</i> [Open the cell whose address is specified and display its contents as a symbolic instruction. If <i>address</i> is omitted, decode the contents of the currently open cell into symbolic instruction form. Always refers to I-space if separate I- and D-space are enabled.
<i>address</i>]	Open the cell whose address is specified and display its contents as a symbolic instruction. If <i>address</i> is omitted and a cell is currently open as an instruction, close the current cell and open the next cell and display its contents in symbolic instruction format. The location of the next cell opened is determined by the number of words used by the instruction in the currently open cell. Always refers to I-space if separate I- and D-space are enabled.
<i>value</i> <TAB>	Execute the next instruction in single step mode. It is not necessary to issue the ";1S" command to single step using the TAB key. If <i>value</i> is non-blank, the value is used to calculate the address of the next instruction to be single stepped (similar to ";1S", then <i>value</i> ;P). If zero (0) is substituted for <i>value</i> before <TAB> is typed and the next instruction is a JSR (CALL), then a temporary breakpoint is set beyond the JSR to catch the return from the called subroutine and the subroutine is executed without single stepping. This can be used to avoid single stepping through subroutines called from code that is being single stepped.

Command	Meaning
<i>address;nR</i>	Set relocation base register <i>n</i> to <i>address</i> . There are 8 relocation registers, numbered 0 through 7.
<i>nR</i>	Convert the contents of the currently open cell into an offset relative to relocation register <i>n</i> .
<i>n!</i>	Convert the current address to an offset relative to relocation register <i>n</i> .
<i>address;nB</i>	Set an instruction breakpoint at <i>address</i> . There are 8 instruction breakpoints, numbered 0 through 7. To clear a breakpoint, substitute 0 for <i>address</i> . The period character may be used to indicate the current value of the program counter. For example, to set breakpoint 1 at the current address: .;1B
<i>address;G</i>	Go to location <i>address</i> and start program execution there.
<i>repeat;P</i>	Proceed from a breakpoint. If <i>repeat</i> is non-blank, do not break again until <i>repeat</i> breakpoints have been hit. If <i>repeat</i> is omitted, proceed until the next breakpoint is hit.
<i>;nS</i>	Set or reset single stepping mode. If <i>n</i> is 1 (or any other non-zero value), begin single step mode. When single stepping, execution will proceed one instruction for each execution of the “;P” command. If <i>n</i> is 0 or is omitted, single stepping mode is cancelled and the program will resume normal execution when the “;P” command is issued. (See also the <TAB> command.)
<i>;nI</i>	Set or reset I-space data mode. This command is only relevant when separate I- and D-space has been enabled. If <i>n</i> is 1 (or any other non-zero value), then subsequent <i>address/</i> and <i>address\</i> commands refer to I-space. These commands normally refer to D-space when separate I- and D-space has been enabled. If <i>n</i> is 0 or omitted, then return to D-space data mode.
X	Interpret the contents of the currently open word as a RAD50 value and display it as a three character ASCII string. Up to three characters may be entered after the current value is displayed. The new characters will be converted to a RAD50 value and stored in the current location. Any characters beyond the first three are ignored.
<i>address;valueM</i>	Set a data watchpoint. The word specified by <i>address</i> is monitored. If <i>value</i> is specified, a data watchpoint occurs when the value of the word matches <i>value</i> . It is possible to <i>watch</i> only selected bits of the monitored address by setting the mask register (see \$M and data watchpoint description below). If <i>value</i> is omitted, a data watchpoint occurs any time the value of the word changes. If separate I- and D-space is enabled, then watchpoints always refer to D-space.

10.5 CTRL-D breakpoints

When a program has been started by the RUN/DEBUG command, its execution can be interrupted at any time by typing CTRL-D. (If the SET CTRLD DEBUG command has been issued, then the program need not have been started under debugger control.) Typing CTRL-D causes the program to stop as though a breakpoint had been encountered at the current location and passes control to the debugger. Any valid debugger command may be issued at this point, and program execution may be resumed with the “;P” command. If a system service call (EMT) is being executed when CTRL-D is typed, the service call will run to completion and control will be passed to the debugger at the point of return to the program from the EMT.

See the Special Notes section at the end of this appendix for more information on interactions of CTRL-D with special terminal mode and special activation characters.

10.6 Address relocation

Address values in commands may be specified absolutely or in the form *n, offset*, where *n* is a relocation register number and *offset* is the offset relative to that relocation register.

When an instruction is decoded into symbolic form, relative addresses are displayed in the form [*n,offset*] as an offset from the nearest relocation register whose value is lower than the address. The format register (\$F) can be used to control the display format of addresses.

Only two bits are significant in the \$F register, bits 0 and 1. Bit 0 controls the display of instruction address operands, and bit 1 controls the display of location values. Setting either bit causes the corresponding addresses to be displayed in absolute format. Clearing either bit causes the corresponding addresses to be displayed in relative format. Both default to relative format (\$F=0).

Address Display Format

\$F	Locations	Operands
0	relative	relative
1	relative	absolute
2	absolute	relative
3	absolute	absolute

10.7 Internal registers

The following special address symbols are used to specify machine registers:

\$0-\$5	Registers R0 through R5
\$6	Register 6, the stack pointer (SP)
\$7	Register 7, the program counter (PC)
\$S	Processor Status Word

The following special address symbols are used to specify internal debugger registers:

\$M	Mask register, used with data watchpoints (see below)
\$F	Print format control register (see above)
\$B	Start of instruction breakpoint address registers
\$R	Start of relocation registers

10.8 Data watchpoints

The data watchpoint facility is a powerful tool for determining where a particular data cell is being modified. The form of the command used to initiate a watchpoint is:

address;valueM

where *address* is the address of the word that is to be monitored, and *value* is an optional parameter that specifies a value to be watched for. If *value* is omitted, a data watchpoint occurs any time the value of the word changes. If *value* is specified, a watchpoint only occurs when the value of the word matches the specified

value. When a watchpoint occurs, the program counter is pointing past the instruction which modified the word.

A data mask may be specified to select a portion of the word being monitored with the watchpoint. The mask value may be referenced using the register name "\$M". Initially the value of the mask is 177777 which causes the entire word contents to be monitored. If some other value is stored in \$M, only the bits selected by the mask are monitored.

Use of the data watchpoint facility causes the program to execute very slowly. When a watchpoint is in effect the debugger causes a break to occur on each instruction executed and then checks to see if the monitored data word has changed value. If a watchpoint is in effect but no instruction breakpoints are set, the execution of the program is slowed down by a factor of about 55 (i.e., the program takes 55 times as long to execute). If a watchpoint is in effect and instruction breakpoints are also set, the speed reduction factor is about 80. There is no significant speed reduction when instruction breakpoints are set but no watchpoint is in effect. The best strategy is to set an instruction breakpoint as close as possible to the code which is modifying the data word and then, when that breakpoint is reached, remove the instruction breakpoint, set the data watchpoint, and proceed with execution.

10.9 Symbolic instruction decoding

Symbolic instruction decoding is used to interpret values according to their numerical op codes and convert them into symbolic assembly language format. The display format of address operands is controlled by the \$F register. When a breakpoint is reached, the instruction on which the breakpoint was set is displayed in its symbolic form. Specific locations can also be symbolically decoded with the "[" and "]" operators.

The "[" command can be issued to symbolically decode a value which has just been displayed using the "/" command or to symbolically decode the contents of a specific location. The following examples illustrate how this might be done (the commands typed by the user are italicized):

```
DBG:2030/ 010246 [ MOV R2,-(SP) DBG:3000[ BIT #2000,[3,100]
```

The "]" operator closes the currently open cell, opens the next cell and displays its contents in symbolic instruction form. The number of words to skip from the currently open cell to the next cell is determined by the number of words used by the instruction in the currently open cell. Thus, the "]" operator is convenient for examining a consecutive set of instructions.

10.10 Special notes

Address location 0 is special to the debugger. For example, symbolic instruction decoding is not done for address 0, and line-feed does not step to the next location after examining location 0.

Since CTRL-D is used by the debugger to dynamically interrupt a program, CTRL-D will not be passed through to a running program even if the program is in special terminal input mode (TTSPC\$ bit set in the JSW). If the program is waiting for terminal input when CTRL-D is typed, then it is necessary to type an activation character (usually <RETURN>) before causing a break to occur and passing control to the debugger.

If a program declares CTRL-D as a special activation character, then CTRL-D is always passed to the program and cannot be used to cause a break in that program and pass control to the debugger.

When the debugger is operating in single step mode, normal breakpoints are temporarily removed and are restored when leaving single step mode. A side effect of this is that if the program executes an instruction (such as RTT) which clears the T (trace) bit in the processor status word, then single stepping is lost, breakpoints are not restored and the program continues execution.

Breakpoints set in overlay regions are only valid while that overlay is resident. If another overlay becomes resident, any breakpoints set in the previous overlay are lost and must be reset.

An attempt to set a break-point in a shared run-time region which is mapped as read-only will result in the error:

?MON-F-Kernel mode trap within TSX-Plus
Abort location = xxxxxx

When examining the processor status word (\$S register), keep in mind that only the low 4 bits (condition codes) are significant to the program. The trace bit is used by the debugger. The priority bits are not significant in user mode under TSX-Plus. And, the high 4 bits will always be set, indicating user as both previous and current mode.

If the user program has issued a .TRPSET call, then traps to 4 and 10 are not intercepted by the debugger, but are passed to the user program's trap handling routine.

During development phase of a program, it can be helpful to always automatically start the program in debug mode. This can be implemented by installing the program with the /DEBUG attribute.

Debugger commands may be included in command files. This allows a convenient method of starting a program under debugger control and setting initial breakpoints. Upon encountering the end of the command file, debugger input reverts to the terminal.

Handling of the BPT instruction depends on whether the program is already running under debugger control. If it is not, (and if the system debugger has been included and if the current privileges allow use of the debugger,) then a BPT instruction embedded in a program causes entry to the debugger and the current location will be the instruction immediately following the BPT instruction. However, if the program is already under debugger control (because it was started with the RUN/DEBUG command or installed with the /DEBUG attribute or previously encountered a BPT instruction or was entered with CTRL-D), then BPT instructions are subsequently ignored.

When using the debugger with programs which enable separate I- and D-space, data references ("/" and "\" commands) refer to I-space (the only space) until separate I- and D-space has been enabled, then they refer to D-space. However, the ";1I" command can be used to force data references to I-space after separate I- and D-space has been enabled. Subsequent data references are then directed to I-space until this mode is disabled with the ";I" (or ";0I") command. If separate I- and D-space is enabled, then data watchpoints always refer to D-space. Symbolic instruction decoding commands ("[" and "]") *always* refer to I-space. (Remember that on processors which do not support separate I- and D-space or if separate I- and D-space is not currently enabled, then I-space and D-space are equivalent.)

Chapter 11

TSX-Plus Performance Monitor Feature

TSX-Plus includes a performance analysis facility that can be used to monitor the execution of a program and determine what percentage of the run time is spent at various locations within the program. During performance analysis, TSX-Plus examines the program being monitored when each clock tick occurs (50 or 60 times per second) and notes at what location in the program execution is taking place. Once the analysis is completed the TSX-Plus performance reporting program (TSXPM) can be used to produce a histogram showing the percentage of time spent at various locations during the monitored run.

There are three steps involved in performing a performance analysis on a program:

1. Use the MONITOR command to begin the analysis.
2. Run the program to be monitored.
3. Run the TSXPM program to print a histogram of the result.

11.1 Starting a performance analysis

The first step in doing a performance analysis is to use the MONITOR keyboard command to tell TSX-Plus that a performance analysis is to be done on the program that will be run next. The form of the MONITOR command is:

MONITOR *base-address*,*top-address*[,*cell-size*]/switches

where *base-address* is the lowest address in the region to be monitored, *top-address* is the highest address in the region to be monitored, and *cell-size* is an optional parameter that specifies the number of bytes of address in the region being monitored to be grouped together into each histogram cell. If *cell-size* is not specified, TSX-Plus calculates the cell size by dividing the number of bytes in the region being monitored (*base-address* to *top-address*) by the total number of histogram cells available (specified when TSX-Plus is generated). This gives the finest resolution possible. The only available switch is /IO which causes I/O wait time to be included in the analysis. If this switch is not specified, only CPU execution time is included in the analysis. A link map of the program should be available to determine the addresses in the program that are appropriate to monitor.

Examples

```

MONITOR 1000,13000/IO
MONITOR 20000,40000,10
MONITOR 2000,6000

```

The effect of the MONITOR command is to set up parameters within TSX-Plus which will be used to monitor the next program run. It does not actually begin the analysis, so there is no rush in running the program to be monitored. Once the MONITOR command has been issued, the program to be monitored is run by using the standard RUN or R commands. If a program being monitored does a .CHAIN to another program, the analysis continues and the times reported will be the composite of the programs run.

Only one user may be doing a performance analysis at a time. This is because the performance analysis histogram buffer is a common memory area that may not be in use by more than one user at a time. An analysis is in effect for a user between the time the MONITOR command is issued and the TSXPM program is run to display the results of the analysis. Running the TSXPM program terminates the performance analysis and allows other users to perform analyses. Note that space for the performance analysis data buffer must be reserved when TSX-Plus is generated.

If the program to be monitored is overlayed and the region to be monitored is in the overlay area, the analysis technique is more complex. In this situation, the EMTs described below must be used to control the performance analysis as overlay segments are run in the segment being monitored.

11.2 Displaying the results of the analysis

After the program being monitored has exited and returned control to the keyboard monitor, the TSXPM performance reporting program is used to generate a histogram of the time spent in the region being monitored. The TSXPM program is started by typing R TSXPM; it responds by printing an asterisk (*). In response to the asterisk, enter the file specification for the device/file where the histogram is to be written. An optional switch of the form /M:nnn may be specified following the file specification. This switch is used to specify the minimum percentage of the total run-time that a histogram cell must contain in order to be included in the display. If this switch is not specified, the default cut-off percentage is 1%.

After receiving the file specification, TSXPM prompts for a title line. Enter a line of text which will be printed as a page title in the histogram file. Press RETURN if you wish no title.

The next item of information requested by TSXPM is a set of base offset values. The base offset values are optional. Base offsets are useful in the situation where you have several modules making up a program being monitored and you want the addresses displayed on the performance analysis histogram to be relative to the base of each module. You may specify up to 10 offset values. Each offset value is specified as an offset module number (in the range 0 to 9) followed by a comma and the base address of the module (see example below). If offset values are specified, TSXPM determines in which module each cell of the histogram falls and displays the address as a module number and offset within the module. After you enter all desired module offsets, enter RETURN without a value.

After the base offset values are entered, the histogram will be produced and written to the specified device and file. After the histogram is generated, TSXPM prints the asterisk prompt again, at which point you may enter the name of another device/file and produce the histogram again or you may type CTRL-C to return to the keyboard monitor.

Example use of TSXPM

```

.R TSXPM
*LP:/M:5
Title:PERFORMANCE ANALYSIS OF EIGENVALUE CALCULATION
Base offsets:
>1,1000
>2,2134
>3,5212
>
(Histogram is produced at this point)
*<CTRL-C>

```


The histogram produced by TSXPM consists of one line per histogram cell. Each line contains the following information:

- the base module offset number (if offsets were specified)
- the address range covered by the histogram cell (relative to the module base if base offsets were used)
- the percentage of the total execution time spent at the address range covered by the histogram cell
- a line of stars presenting a graphic representation of the histogram

11.3 Performance monitor control EMTs

For most applications the method described above can be used to do a performance analysis. However, in special cases (such as analyzing the performance of an overlayed program) it is necessary to have more explicit control over the performance analysis feature as a program is running. The following set of EMTs may be used to control a performance analysis.

11.3.1 Initializing a performance analysis

This EMT is used to set up parameters that will control a performance analysis. It does not actually begin the analysis. The form of the EMT is:

EMT 375

with R0 pointing to the following argument block:

```
.BYTE  0,136
.WORD  base-address
.WORD  top-address
.WORD  cell-size
.WORD  flags
```

where *base-address* is the address of the base of the region to be monitored, *top-address* is the address of the top of the region to be monitored, and *cell-size* is the number of bytes to group in each histogram cell. If zero (0) is specified as the cell size, TSX-Plus calculates the cell size to use by dividing the number of bytes in the region being monitored (top-address minus base-address) by the number of cells available in the histogram data area (specified when TSX-Plus is generated). The *flags* parameter is used to control whether or not I/O wait time is to be included in the analysis. If a value of 1 is specified as the *flags* parameter, I/O wait time is included in the analysis; if a value of zero (0) is specified, I/O wait time is not included in the analysis.

Error Code	Meaning
0	Performance analysis being done by some other user.
1	Performance analysis feature not included in TSX-Plus generation.

Example

```

        .TITLE DEMOPA
        .ENABL LC
; Demonstrate use of TSX--Plus EMTs to initialize, start, stop, and
; terminate a performance analysis
        .MCALL .EXIT,.PRINT,.LOOKUP,.READW,.CLOSE,.TTYOUT
        .GLOBL PRTOCT                ;Display an octal word in RO
ERRBYT = 52                        ;EMT error byte
SPACE  = 40                        ;ASCII 'space'
STAR   = 52                        ;ASCII 'asterisk'
START: MOV    #INITPA,RO            ;Point to EMT arg block to
        EMT    375                  ;Initialize the performance analysis
        BCC    5$                  ;No error
        MOVB   @#ERRBYT,RO          ;Which error?
        ASL    RO                   ;Convert to word offset
        .PRINT INIERR(RO)           ;Print the error message
        .EXIT                      ;And depart this world of woe.
5$:     MOV    #STRIPA,RO            ;Point to EMT arg block to
        EMT    375                  ;Start the performance analysis
        BCC    10$                 ;No error
        .PRINT #STRERR              ;Start error
        .EXIT                      ;and depart.

10$:
; Dummy section of code to do some I/O and computation for analysis
BEGIN: .LOOKUP #AREA,#0,#FILNAM
        MOV    #10,R1
; Disk I/O
1$:     .READW #AREA,#0,#BUFFER,#256.,#0
        SOB    R1,1$
        .CLOSE #0
; Terminal I/O
        .PRINT #BUFFER
; Compute bound
        MOV    #123456,R3
2$:     MOV    #12345,R0
        CLR    R1
        DIV    #345,R0
        SOB    R3,2$
ENDB:
NCELLS = <<ENDB-BEGIN>/2>          ;Number of cells in histogram
        MOV    #STOBLK,R0           ;Point to EMT arg block to
        EMT    375                  ;Stop the performance analysis
        BCC    15$                 ;Check for error return
        .PRINT #STOERR              ;Only one error - PA not initialized
        .EXIT                      ;and leave.
15$:    MOV    #HALBLK,RO            ;Put HALTPA block address in RO
        EMT    375                  ;Terminate the performance analysis
        BCC    20$                 ;Check for errors
        MOVB   @#ERRBYT,RO          ;Get error type
        ASL    RO                   ;Convert to word offset
        .PRINT HLTERR(RO)           ;Print out proper error message
        .EXIT                      ;And depart.
20$:    TST    HALFLG               ;Check HALTPA return flag
        BPL    25$                 ;No cell overflow?
        .PRINT #OVRWRN              ;Issue overflow warning
25$:    BIT    #1,HALFLG             ;I/O time included?
        BEQ    30$                 ;Skip message if not
        .PRINT #IOWNOT              ;Print I/O wait time included msg
; Print a histogram of the performance analysis
30$:    CLR    R3                   ;Set histogram cell counter
35$:    CMP    HSTTBL(R3),#64.       ;Normalize the table for 64. longest
        BHI    40$                 ;Too many counts?
        ADD    #2,R3                ;Point to next cell
        CMP    R3,#<2*NCELLS>       ;End of table?
        BLE    35$                 ;No, repeat
        BR     50$                 ;All cells less than 64. counts now
40$:    CLR    R3                   ;Re-init. histogram cell counter
45$:    CLC                     ;Divide each cell count by 2
        ROR    HSTTBL(R3)

```

```

      ADD      #2,R3                ;Point to next cell
      CMP      R3,#<2*NCELLS>      ;End of table?
      BLE      45$                 ;No, repeat
      BR       30$                 ;Go check all cells again
50$:  .PRINT   #HSTMSG              ;Caption histogram
      MOV      #BEGIN,R2           ;Set first analyzed address
      CLR      R3                  ;Init. cell counter
55$:  .MOV      R2,R0               ;Get ready to print it out
      CALL     PRTOUT              ;Display analyzed address
      .TTYOUT  #SPACE              ;For formatting
      MOV      HSTTBL(R3),R1       ;Get address use counter
      BEQ      65$                 ;No stars on 0 count
60$:  .TTYOUT  #STAR                ;Print a '*' for each count
      SOB      R1,60$
65$:  .PRINT   #CRLF                ;Go to next line
      CMP      (R2)+,(R3)+         ;Point to next address and count
      CMP      R3,#<2*NCELLS>      ;End of histogram?
      BLE      55$                 ;No, display next cell
      .EXIT                          ;Else done.
INITPA: .BYTE   0,136              ;EMT arg block for perform. analysis
      .WORD    BEGIN               ;Start analysis address
      .WORD    ENDB-2              ;End analysis address
      .WORD    2                   ;Count 1 address in each cell
      .WORD    1                   ;Include I/O wait time.
STRTPA: .BYTE   1,136              ;Start Performance Analysis EMT block
STOBLK: .BYTE   2,136              ;EMT arg block to stop perf. analysis
HALBLK: .BYTE   3,136              ;EMT arg block to release analysis
      .WORD    PRMBUF              ;PA parameter buffer address
      .WORD    HSTTBL              ;Histogram Table buffer address
      .WORD    <HSTEND-HSTTBL>     ;Histogram Table buffer length
PRMBUF: .BLKW   3                   ;PA four word parameter buffer
HALFLG: .WORD   0                   ;PA return flags
HSTTBL: .BLKW   <2*NCELLS>+2       ;Histogram table
HSTEND:
AREA:   .BLKW   10                  ;EMT arg block area
BUFFER: .BLKW   256.                ;Data input buffer
      .WORD    0                   ;Make sure buffer is ASCIZ
FILNAM: .RAD50  /DK DEMOPAMAC/      ;Read this file
INIERR: .WORD    INPRG              ;Initialize PA error table
      .WORD    NOGEN
HLTERR: .WORD    NOPA                ;HALTPA Error message table
      .WORD    TOSMAL
      .MLIST   BEX
INPRG:  .ASCIZ  /?INITPA-F-Performance analysis being done by another user./
NOGEN:  .ASCIZ  /?INITPA-F-Performance analysis feature not genmed./
NOPA:   .ASCIZ  /?HALTPA-F-This job is not doing a performance analysis./
TOSMAL: .ASCIZ  /?HALTPA-F-Area provided for histogram table too small./
STRERR: .ASCIZ  /?INITPA-F-Performance analysis not initialized yet./
STOERR: .ASCIZ  /?STOPPA-F-Performance Analysis has not been initialized./
OVRWRN: .ASCIZ  /?HALTPA-W-Some histogram cell(s) overflowed during analysis./
IOWNOT: .ASCIZ  | HALTPA-I-I/O wait time included in performance analysis.|
OKDONE: .ASCIZ  / STOPPA-S-Performance Analysis stopped./
HSTMSG: .ASCIZ  /Address      Frequency/
CRLF:   .ASCIZ  <15><12><200>
      .END      START

```

11.3.2 Starting a performance analysis

This EMT is used to begin the actual collection of performance analysis data. The previous EMT must have been executed to set up parameters about the performance analysis before this EMT is called. The form of the EMT is:

EMT 375

with R0 pointing to the following argument block:

.BYTE 1,136

The carry-flag will be set on return from this EMT if performance analysis has not been initialized yet.

Example

See the example program DEMOPA in section 11.3.1 on initializing a performance analysis.

11.3.3 Stopping a performance analysis

The following EMT can be used to suspend the data collection for a performance analysis. The data collection can be restarted by using the start-analysis EMT described above. This EMT could, for example, be used to suspend the analysis when an overlay module is loaded that is not to be monitored. The start-analysis EMT would then be used to re-enable the data collection when the overlay of interest is re-loaded. The form of this EMT is:

EMT 375

with R0 pointing to the following argument block:

.BYTE 2,136

The carry flag will be set on return from this EMT if performance analysis has not been initialized yet.

Example

See the example program DEMOPA in section 11.3.1 on initializing a performance analysis.

11.3.4 Terminating a performance analysis

This EMT is used to conclude a performance analysis. It has the effect of returning into a user supplied buffer the results of the analysis and releasing the performance analysis feature for other users. The form of this EMT is:

EMT 375

with R0 pointing to the following argument block:

```
.BYTE 3,136
.WORD parameter-buffer
.WORD histogram-buffer
.WORD buffer-size
```

where *parameter-buffer* is the address of a four-word buffer into which will be stored some parameter values describing the analysis that was being performed, *histogram-buffer* is the address of the buffer that will receive the histogram count values, and *buffer-size* is the size (in bytes) of the histogram buffer area.

The values returned in the parameter buffer consist of the following four words:

1. base address of the monitored region
2. top address of the monitored region
3. number of bytes per histogram cell
4. control and status flags

The control and status flags are a set of bits that provide the following information:

Flag	Meaning
1	I/O wait time was included in the analysis.
100000	Some histogram cell overflowed during the analysis.

The data returned in the histogram buffer consists of a vector of 16-bit binary values, one value for each cell in the histogram. The first value corresponds to the histogram cell that starts with the base address of the region that was being monitored.

Error Code	Meaning
0	This job is not doing a performance analysis.
1	Area provided for histogram count vector is too small.

Example

See the example program DEMOPA in section 11.3.1 on initializing a performance analysis.

Chapter 12

TSX-Plus Restrictions

12.1 System service call (EMT) differences between RT-11 and TSX-Plus

The following list describes the differences in the way TSX-Plus implements some RT-11 system service calls (EMTs). If an EMT is not listed, it provides the same functions as described in the *RT-11 Programmer's Reference Manual*.

- .ABTIO** Action depends on setting of IOABT system parameter. If IOABT is set to 1 then .ABTIO operates the same as RT-11 and calls handler abort entry points. If IOABT is set to 0 then .ABTIO does not call handler entry points, but instead does a .WAIT on all channels. The default method of handling I/O abort requests is chosen during TSX-Plus system generation with the IOABT parameter. The I/O abort handling method may also be changed while TSX-Plus is running with the SET IO [NO]ABORT keyboard command. See the *TSX-Plus System Manager's Guide* for more details on this request.
- .CALLK** This system service is not supported.
- .CHAIN** Shared files and locked records are not preserved across a chain, although the channels do remain open.
- .CDFN** User jobs may not define more than 20 channels. If a .CDFN EMT is done, all channels are purged if a .CHAIN is done. If .CDFN is not done, channels are not purged across a chain.
- .CHCOPY** Not implemented.
- .CNTXSW** Not implemented.
- .CRAW** RT-11 allows window creation with W.NRID = 0, to be filled in later. TSX-Plus requires a valid region ID at the time of window creation as a consequence of support for separate I- and D-space.
- .CSIGEN/.CSISPC** Differ from RT-11 with regard to activation by CTRL-C and CTRL-Z.
- .DATE** Extended dates (through 2099) are supported as in RT-11. The high two bits of the date word specify the number of 32-year offsets to be added to the 1972 base year.
- .DEVICE** REALTIME privilege is required to use this EMT.
- .DRxxx** See the *TSX-Plus System Manager's Guide* for information on programming device handlers for TSX-Plus.
- .EXIT** Error codes passed back from an exiting program (in byte 53) are interpreted by the keyboard monitor according to the following values:

<i>Code</i>	<i>Meaning</i>
1	Success
2	Warning
4	Error
10	Severe
	Fatal
20	Unconditional

- .FETCH** Returns in R0 the address specified for the handler load area. All TSX-Plus device handlers are resident, hence the .FETCH EMT performs no operation. If the device handler specified was not loaded during TSX-Plus initialization, then an error code of 0 is returned.
- .FORK** May be used in device handlers but not in user jobs. See the *TSX-Plus System Manager's Guide* for information on programming device handlers for TSX-Plus.
- .GTJB** The job number returned in word 1 of the result area is two times the TSX-Plus line number. That is, the first line specified in the system generation will be number 2, the second 4, etc. Words 8 through 12 of the result area are not altered by this EMT.
- .GTLIN** Differs from RT-11 with regard to activation by CTRL-C and CTRL-Z. If the program controlled terminal option N is used to suspend command file input and a .GTLIN request is issued, then command file input cannot be restored. Do not use this method to force keyboard input from within command procedures. Use the optional *type* argument to the .GTLIN request to force terminal input rather than disabling command file input.
If a .GTLIN request is issued after the program controlled terminal option "N" is used to suspend command file input, then command file input cannot be restored. Normally this is done to force operator response — in this case, use the optional *type* argument to the .GTLIN request to force terminal input rather than disabling command file input.
- .GVAL** Not all of the fixed offsets documented by RT-11 are supported by TSX-Plus. Additional negative offsets are available to obtain TSX-Plus system values. See Chapter 4 for more information on .GVAL.
- .HERR** The previous state of this request is returned in R0. If user error handling was in effect (.SERR), then R0 will contain a one. If user error handling was not in effect (.HERR), then R0 will contain a zero.
- .HRESET** Action depends on setting of IOABT system generation parameter. If IOABT is set to 1 then .HRESET calls handler abort entry points. If IOABT is set to 0 then .HRESET does not call handler entry points but instead does a .WAIT on all channels. Messages queued on named message channels are not canceled. Otherwise, this EMT operates the same as under RT-11.
- .INTEN** May be used in device handlers but not in user jobs. See the *TSX-Plus System Manager's Guide* for information on programming device handlers for TSX-Plus.
- .LOCK** The TSX-Plus file management module (USR) is always *resident* and the .LOCK EMT is ignored.
- .MAP** When issuing a .MAP EMT, the window definition block must specify a non-zero region ID returned from the successful creation of a region by use of the .CRRG EMT. A zero region ID will return the error code 2.
- .MFPS** This request calls a monitor subroutine and will fail with a "Trap to 4" if the job is not mapped to simulated RMON. The Processor Status Word (PSW) value returned is always 000000 (current mode *user*, previous mode *user*, general register set 0, priority 0, and all condition code and trace bits clear), which would be the normal case for user jobs. See the chapter on Real-Time Program Support for a system service to obtain the actual PSW value.
- .MRKT** Expiration of repeated mark-time requests can cause delay of an outstanding .TWAIT request.

- .MTxxx** Multi-terminal control EMTs (.MTIN, .MTOU, .MTPRNT, etc.) are not supported.
- .MTPS** The processor priority level may not be changed from user mode, hence this macro performs no operation. TSX-Plus provides a special real-time EMT to set the processor priority.
- .MWAIT** Not supported. See Chapter 6 on inter-job message communication.
- .PEEK** Non-privileged jobs may use .PEEK to access cells within the simulated RMON (addresses 160000 to 160626) although the .GVAL EMT is a recommended alternative. Jobs with MEMMAP privilege may use .PEEK to access the I/O page or low memory cells in kernel space. References to addresses in the virtual address range of the simulated RMON (160000 to 160626) are always directed to RMON rather than the I/O page.
- .POKE** Non-privileged jobs may use .POKE to access cells within the simulated RMON (addresses 160000 to 160626) although the .PVAL EMT is a recommended alternative. Jobs with MEMMAP privilege may use .POKE to access the I/O page or low memory cells in kernel space. References to addresses in the virtual address range of the simulated RMON (160000 to 160626) are always directed to RMON rather than the I/O page.
- .PROTECT** Not supported. See Chapter 8 on real-time programming for information about how to connect an interrupt to a TSX-Plus job.
- .QSET** TSX-Plus uses an internal pool of I/O queue elements for all jobs hence it is not necessary to define additional I/O queue elements in order to perform overlapped I/O. The .QSET EMT is ignored. On return, R0 contains the address specified for the start of the new elements.
- .RCVD** Not supported. See Chapter 6 which describes inter-job message communication.
- .RELEASES** This EMT is ignored. Refer to .FETCH.
- .SCCA** All .SCCA requests are treated as local to the job which issues the request. Global SCCA is ignored.
- .SDTTM** Extended dates (through 2099) are supported as in RT-11. The high two bits of the date word specify the number of 32-year offsets to be added to the 1972 base year.
- .SERR** The previous state of this request is returned in R0. If user error handling was in effect (.SERR), then R0 will contain a one. If user error handling was not in effect (.HERR), then R0 will contain a zero.
- .SETTOP** Returns the job limit in R0 but does not actually expand or contract the allocated job region. The job region allocation can be changed by use of the MEMORY keyboard command or the TSX-Plus specific EMT for this purpose.
- .SDAT** Not supported. See Chapter 6 which describes inter-job message communication.
- .SFPA** This EMT functions the same as RT-11. It *must* be used if a job is going to use the floating-point unit.
- .SDTTM** OPER privilege is required to use this EMT.
- .SPFUN** SPFUN privilege is required to use this EMT. SPFUN function code 371 (BYPASS) is unsupported with the DU handler on extended Unibus hardware and with the MU handler in all configurations.
- .SRESET** Messages queued on named message channels are not canceled. Otherwise, this EMT operates the same as under RT-11.
- .SYNCH** May be used in device handlers but not in user jobs. When used in a handler, the number of the TSX-Plus job that is being synchronized with must be stored in word 2 of the SYNCH block. See the *TSX-Plus System Manager's Guide* for information on programming device handlers for TSX-Plus.

.TLOCK This EMT is ignored.

.TTINR Only honors bit 6 in the Job Status Word (TCBIT\$) if a SET TT NOWAIT command has been issued, or the running program has send the "U" program controlled terminal option to the system, or the program was R[UN] with the /SINGLECHAR switch.

.TTOUTR Returns with the carry bit set if the terminal output buffer is full. The following conditions must be met:

1. The .TTOUTR EMT must be used rather than .TTYOUT.
2. The instruction following the .TTOUTR must not be [BCS .-2].
3. Bit 6 must be set in the job status word.
4. The program must be run in single character activation mode or with the NOWAIT attribute set.

.TTYOUT If many characters are displayed by .TTYOUT requests from within a completion routine while a log file is open, then the job may hang.

.TWAIT Expiration of repeated mark-time requests can cause delay of an outstanding .TWAIT request.

.UNLOCK This EMT is ignored.

.UNPROT Not supported. See Chapter 8 on real-time programming for information about how to connect an interrupt to a TSX-Plus job.

A full list of RT-11 compatible and TSX-Plus specific EMTs is included in Appendix B. The list indicates the level of support TSX-Plus provides for each RT-11 compatible EMT.

12.2 Special program suggestions

Certain programs use system resources in ways which may not seem obvious or cause behavior which is not expected. Some of those features are described in this section to facilitate their use with TSX-Plus.

12.2.1 DIBOL

When DIBOL programs are run under TSX-Plus they must use SUD rather than TSD. Several components of the DIBOL system also use I/O channel numbers above 15 (decimal). Reducing the TSGEN parameter NUCHN from the default of 20. can result in the error ?INI-E3 or ?INI-E4 when using DIBOL or its utilities. This can be resolved either by patching SUD or the utility to use lower channel numbers or by regenerating TSX-Plus to permit use of higher channel numbers. Set the TSGEN parameter NUCHN to 20., reassemble TSGEN, and relink TSX-Plus.

Keep in mind that record locking does not automatically occur when DIBOL is used with TSX-Plus and unless the appropriate TSX-Plus record locking calls are made there is a danger of file corruption. Appendix D describes subroutines which may be linked into DIBOL programs to provide access to TSX-Plus record locking and inter-program messages. Do not use the DIBOL SEND statement, use an equivalent XCALL MSEND instead.

12.2.2 FILEX utility

The FILEX utility is used to read and write "Interchange" format disks. Because of the way FILEX does I/O to single density disks in RX02 drives, FILEX must either be run with the MEMLOCK switch or installed with the MEMLOCK attribute.

12.2.3 FORTRAN virtual arrays

When the object-time system for FORTRAN-IV is generated, three options are available relating to use of virtual arrays: NOVIR, VIRP, or VIRNP. If virtual arrays are to be used at all, either VIRP or VIRNP must be selected. Under RT11SJ, the VIRNP option is commonly selected. This causes FORTRAN virtual array handling to directly manipulate the memory management registers in the I/O page. Clearly, in a multi-user environment, like TSX-Plus, this is not advisable. The VIRP option is used with RT11XM and must also be selected for use with TSX-Plus. TSX-Plus must also be generated with PLAS support; see your system manager. If the wrong type of virtual array support is selected, various types of FORTRAN errors may occur, including virtual array initialization failures.

F77 and FORTRAN-IV use PAR 7 to map to virtual arrays. (Page Address Register 7 maps virtual addresses 160000 to 177777.) TSX-Plus normally maps a job's PAR 7 to a simulated RMON so that older programs which require direct access to fixed offsets in RMON may operate without modification. When a program requires direct access to the I/O page to manipulate device registers, that is commonly done by mapping PAR 7 to the I/O page. This can either be done with the /IOPAGE switch to the R[UN] command or with a TSX-Plus EMT. Mapping PAR 7 to the I/O page only conflicts slightly with direct RMON access since the parameters available there may also (and should be) obtained with the .GVAL request (SYSLIB ISPY function).

Because use of virtual arrays also affects PAR 7 mapping, mapping PAR 7 to the I/O page causes more severe problems when programs must have both virtual arrays and direct I/O page access. The solution to this is to map some other part of the job's virtual address space to the I/O page. TSX-Plus provides an EMT to map any part of a job's address space to any physical region, including the I/O page. See Chapter 8 for information on mapping to a physical region to resolve the virtual array vs. I/O page conflict. The simplest way to do this is to allow PAR 7 to be used for virtual arrays and map PAR 6 to the I/O page. To avoid conflict with other parts of the program image, the program must be able to run in less than 48 Kb. Use the SETSIZ facility to restrict the program to a total memory space of 48 Kb. Call a MACRO interface routine from the FORTRAN program to map PAR 6 to the I/O page. Be sure to normalize I/O page addresses to virtual PAR 6 addresses. Note that mapping to the I/O page or to physical memory requires MEMMAP privilege.

12.2.4 IND .ASKx timeouts

When using the IND control file processor, the .ASKx directives provide an optional timeout parameter. This parameter causes IND to proceed when no terminal response is obtained before the designated timeout period. When using .ASKx timeouts under TSX-Plus, it is necessary to SET TT NOWAIT to permit IND to proceed when the timeout period expires.

12.2.5 MicroPower/Pascal

Various program components of MicroPower/Pascal (TM of Digital Equipment Corporation) should be run as virtual programs. That is, they should run without direct RMON access. Under TSX-Plus, this permits them to obtain a virtual job address space of a full 64 Kb. TSX-Plus must also be generated to allow 64 Kb jobs. To define the Pascal compiler and the PASDBG and MIB programs as virtual jobs, use the SETSIZ program (see Appendix A) to allow a 64 Kb memory partition for them. For example:

```
.R SETSIZ
*SY: PASDBG/T:64.
*~C
```

12.2.6 Overlaid programs

TSX-Plus permits programs to open I/O channels 0 through 23 (octal). However, the overlay handler uses channel 17 (octal). Therefore, overlaid programs should not use I/O channel 17 (octal).

Appendix A

SETSIZ Program

The SETSIZ program can be used to store into a SAV file information about how much memory space should be allocated for the program when it is executed. SETSIZ can also be used to set the *virtual job* flag in the job status word (JSW) for a program.

There are three ways that the amount of memory allocated to a job can be controlled:

- The TSX-Plus EMT with function code 141 (described in Chapter 4) may be used by a running program to dynamically set the job's size.
- If a size is specified in a SAV file (by use of the SETSIZ program) the specified amount of memory is allocated for the program when it is started.
- If no size is specified in the SAV file, the size specified by the last MEMORY keyboard command is used.

Note that the .SETTOP EMT does not alter the amount of memory space allocated to a job but can be used by a running program to determine the amount of memory allocated.

The effect of the SETSIZ program is to store into location 56 of block 0 of the SAV file the number of kilowords of memory to allocate for the program when it is run (the LINK /K:n switch can also be used to do this). This value has no effect when the SAV file is run under RT-11 but causes TSX-Plus to allocate the specified amount of memory when starting the program.

If a size value is specified in a SAV file, it takes precedence over the size specified by the last MEMORY keyboard command. The TSX-Plus EMT with function code 141 may still be used to dynamically alter the memory allocation while the program is running.

Most programs allocate memory in two ways:

- a static region that includes the program code and data areas of fixed size
- a dynamic region that is allocated above the static region—usually the .SETTOP EMT is used to determine how much dynamic space is available to the program.

The size of the static region for a program is fixed at link time. If the program is overlaid the static region includes space for the largest overlay segment as well as the program root. Location 50 in block 0 of the SAV file is set by the linker to contain the address of the highest word in the static region of the program.

The amount of memory space to be allocated for a SAV file can be specified to the SETSIZ program in either of two ways:

- as the total amount of memory for the program which includes space for the static plus dynamic regions
- as the amount of memory for the dynamic region only, in which case SETSIZ automatically adds the size of the static region.

A.1 Running the SETSIZ program

The SETSIZ program is started by use of the command:

```
R SETSIZ
```

it responds by displaying an asterisk to prompt for a command line. The form of the command line is:

```
*filespec/switch:value
```

where *filespec* is a file specification of the standard form dev:name.ext with the default device being DK and the default extension being .SAV.

If a file specification is entered without a switch, SETSIZ displays information about the size of the SAV file and the SAV file is not altered.

SETSIZ also displays the following status message if the SAV file is flagged as being a virtual image.

```
Virtual-image flag is set
```

The virtual message is displayed if either of the following two conditions exist for the SAV file:

- Bit 10 (mask 2000) is set in the job status word (location 44) of the SAV file.
- Location 0 of the SAV file contains the RAD50 value for VIR.

These are the same two conditions that cause TSX-Plus to recognize the SAV file as being a virtual image when it is started.

Examples

```
.R SETSIZ
*TSTPRG
Base size of program is 22 Kb
Size of allocation space is 28 Kb
*SY:PIP
Base size of program is 10 Kb
Size of allocation space is 22 Kb
*PROG1
Base size of program is 31 Kb
No allocation size specified in SAV file
Virtual-image flag is set
```

A.2 Setting total allocation for a SAV file

The /T switch is used to specify the total amount of memory space to be allocated for a program when it is run. The form of the /T switch is /T:*value*. where *value* is the number of kilobytes of memory to allocate. Note that a decimal point must be specified with the value if it is entered as a decimal value.

If the /T switch is used without a value, the effect is to clear the TSX-Plus size allocation information in the SAV file.

Examples

```
.R SETSIZ
*SY:PIP/T:18.
*TSTPRG/T:32.
*PROG1/T
```

A.3 Setting amount of dynamic memory space

The /D switch is used to specify the amount of dynamic memory space to be reserved for a program. The SETSIZ program calculates the total amount of space to allocate for the program by adding the static size (stored in location 50 of the SAV file by LINK) to the specified dynamic size. The form of the /D switch is /D:*value*. where *value* is the number of kilobytes of dynamic memory space to reserve. If a program does not use any dynamic memory space, the /D switch may be used without an argument value to cause the total memory space allocation to be set equal to the static size of the program. FORTRAN programs use dynamic space for I/O buffers and the exact amount required depends on the number of I/O channels used. However, 4 Kb of dynamic memory space seems to be adequate for most FORTRAN programs.

Examples

```
.R SETSIZ
*SY:PIP/D:11.
*TSTPRG/D:4.
*PROG1/D
```

A.4 Setting virtual-image flag in SAV file

The /V switch is used to cause SETSIZ to set the virtual-image flag in the SAV file. This flag is bit 10 (mask 2000) in the job status word (location 44) of the SAV file.

Setting this flag indicates that the program will not directly access RMON, although it may do so indirectly by use of the .GVAL and .PVAL EMTs. The significance with regard to TSX-Plus is that it allows the program to use more than 56 Kb (if that much memory is also allowed by use of a MEMORY command). The virtual-image flag should not be set unless you are sure the job does not need direct access to the RMON area.

Appendix B

RT-11 & TSX-Plus EMT Codes

B.1 TSX-Plus RT-11 Compatible EMTs

In the following table, the level of compatibility with the RT-11 functionality of these EMTs is indicated by a special character in the *Code* column. The codes are:

- * Not supported, will cause error
- 0 Treated as a nul operation (NOP)
- Minor differences, see Chapter 12

EMT	Code	Chan	Name	Description
340	-		.TTINR	Get character from terminal
341			.TTYOUT	Send character to terminal
342			.DSTATUS	Get device information
343	-		.FETCH/.RELEASES	Load/Unload device handlers
344			.CSIGEN	Call command string interpreter
345			.CSISPC/.GT LIN	Get command line
346	0		.LOCK	Lock USR in memory
347	0		.UNLOCK	Allow USR to swap
350			.EXIT	Return to monitor
351			.PRINT	Display string at terminal
352	-		.SRESET	Software reset
353	0		.QSET	Increase I/O queue size
354	-		.SETTOP	Set program upper limit
355			.RCTRL-O	Reset CTRL-O
357	-		.HRESET	Stop I/O then .SRESET

EMT	Code	Chan	Name	Description
374	0		.WAIT	Wait for I/O completion
374	1		.SPND	Suspend mainline program
374	2		.RSUM	Resume mainline program
374	3		.PURGE	Free a channel
374	4		.SERR	Inhibit abort on error
374	5		.HERR	Enable abort on error
374	6		.CLOSE	Close channel
374	0 7		.TLOCK	Try to lock USR
374	10		.CHAIN	Pass control to another program
374	* 11		.MWAIT	Wait for message
374	12		.DATE	Get current date
374	- 13		.ABTIO	Abort I/O in progress
375	0		.DELETE	Delete a file
375	1		.LOOKUP	Open existing file
375	2		.ENTER	Create file
375	3		.TRPSET	Intercept traps to 4 and 10
375	4		.RENAME	Rename a file
375	5		.SAVESTATUS	Save channel information
375	6		.REOPEN	Restore channel information
375	7		.CLOSE	Close channel
375	10		.READ[C][W]	Read from channel to memory
375	11		.WRIT[C][E][W]	Write from memory to channel
375	12		.WAIT	Wait for I/O completion
375	* 13		.CHCOPY	Open channel to file in use
375	14		.DEVICE	Load device registers on exit
375	- 15		.CDFN	Define extra I/O channels
375	- 20		.GTJB	Get job information
375	21		.GTIM	Get time of day
375	22		.MRKT	Schedule completion routine
375	23		.CMKT	Cancel mark time
375	24		.TWAIT	Timed wait
375	* 25		.SDAT[C][W]	Send data to another job
375	* 26		.RCVD[C][W]	Receive data from another job
375	27		.CSTAT	Return channel information
375	30		.SFPA	Trap floating point errors
375	0 31	0	.PROTECT	Control interrupt vector
375	0 31	1	.UNPROTECT	Release interrupt vector
375	32		.SPFUN	Special device functions
375	* 33		.CNTXSW	Context switch

EMT	Code	Chan	Name	Description
375	34	0	.GVAL	Get monitor offset value
375	- 34	1	.PEEK	Get low memory value
375	34	2	.PVAL	Change monitor offset value
375	- 34	3	.POKE	Change low memory value
375	35		.SCCA	Inhibit CTRL-C abort
375	36	0	.CRRG	Create an extended memory region
375	36	1	.ELRG	Eliminate an extended memory region
375	36	2	.CRAW	Create a virtual address window
375	36	3	.ELAW	Eliminate a virtual address window
375	36	4	.MAP	Map virtual window to XM region
375	36	5	.UNMAP	Get window mapping status
375	36	6	.GMCX	Obtain window status
375	* 37	0	.MTSET	Set terminal status
375	* 37	1	.MTGET	Get terminal status
375	* 37	2	.MTIN	Get character from terminal
375	* 37	3	.MTOUT	Send character to terminal
375	* 37	4	.MTRCTO	Reset CTRL-O
375	* 37	5	.MTATCH	Lock terminal to job
375	* 37	6	.MTDTCH	Release terminal from job
375	* 37	7	.MTPRNT	Display string at terminal
375	* 37	10	.MTSTAT	Get system status
375	40		.SDTTM	Set date and time
375	41		.SPCPS	Change mainline control flow
375	42		.SFDAT	Change file date
375	43		.FPROT	Change file protection

B.2 TSX-Plus Specific EMTs

EMT	Code	Chan	Name	Description
375	101		UNLALL	Unlock all blocks
375	102		LOCKW	Wait for locked block
375	103		TLOCK	Try to lock a block
375	104	0	SENDMSG	Send message on named channel
375	105	0	GETMSG	Get message from named channel
375	106	0	WATMSG	Wait for message on named channel
375	106	1	MSGCPL	Queue message receipt completion routine
375	107	0	SPLFRE	Get number of free spool blocks
375	107	1	SPLUSE	Get number of spool blocks in use
375	110	0	TSXLN	Get line number
375	110	1	SUBPLN	Get subprocess line number
375	111	0/1	ACTODT	Reset/set ODT activation mode
375	113		UNLOCK	Unlock a block
375	114	0	TTOBLK	Send block of text to terminal
375	115	0	TTIBLK	Get block of text from terminal
375	116	0	CKTTIE	Check for terminal input errors
375	116	1	GTINCH	Get number of pending input characters
375	117	0	SETTTO	Set terminal read time-out
375	120	0/1	HIEFF	Reset/set high-efficiency mode
375	121		CKWSHR	Check for writes to shared file
375	122		SAVSHR	Save shared file information
375	123	0	CKACT	Check for activation characters
375	124	0	SITNUM	Get the site incremental license number
375	124	1	SITNAM	Get the site name
375	125		SHRFIL	Declare file for shared access
375	127	0-3	SEND	Send message to another line
375	132	0	STRTDJ	Start a detached job
375	132	1	STATDJ	Check detached job status
375	132	2	KILL	Abort a job
375	133	0	DCLBRK	Establish break sentinel control
375	133	1	TTICPL	Set terminal input completion routine
375	134	0	MOUNT	Mount a directory structure
375	135	0	DISMNT	Dismount a directory structure
375	135	2	DMTFLS	Dismount all mounted file structures
375	135	3	DMTLD	Dismount a logical disk
375	135	4	LDSTAT	Get the status of a logical disk
375	135	5	DMTALD	Dismount all logical disks
375	136	0	INITPA	Initiate performance analysis
375	136	1	STRTPA	Start monitoring performance
375	136	2	STOPPA	Stop monitoring performance
375	136	3	HALTPA	Terminate performance analysis

EMT	Code	Chan	Name	Description
375	137	0	TTYTYPE	Get terminal type
375	140	0	PHYADD	Convert virtual I-space to physical address
375	140	1	PEEKIO	Peek into the I/O page
375	140	2	POKEIO	Poke into the I/O page
375	140	3	BISIO	Bit-set into the I/O page
375	140	4	BICIO	Bit-clear into the I/O page
375	140	5	MAPIOP	Map PAR7 to the I/O page
375	140	6	MAPMON	Map PAR7 to simulated RMON
375	140	7	LOKLOW	Lock job into lowest memory
375	140	10	UNLOKJ	Unlock job from memory
375	140	11	ATTVEC	Attach to interrupt vector
375	140	12	RELVEC	Release interrupt vector
375	140	13	LOKJOB	Lock job without re-positioning
375	140	14	STEALS	Get exclusive system access
375	140	15	RTURNS	Relinquish exclusive system access
375	140	16	SETPRI	Set user mode priority level
375	140	17	MAPPHY	Map to physical memory
375	140	20	ATTSVC	Attach interrupt service routine
375	140	21	SCHCMP	Schedule completion routine
375	140	22	PHYDAD	Convert virtual D-space to physical address
375	141	0	MEMTOP	Control size of job
375	143	0	USERTS	Associate with run-time system
375	143	1	MAPRTS	Map run-time system into job
375	143	2	REGRTS	Define a region for fast map
375	143	3	CLRRTS	Clear region definitions for fast map
375	143	4	SIDON	Enable separate I- and D-space mapping
375	143	5	SINOFF	Disable separate I- and D-space mapping
375	143	6	REGGBL	Define a PLAS region for fast map
375	144	0	JSTAT	Get job status information
375	145		FILINF	Get file directory information
375	146		SFTIM	Set file creation time
375	147	0	GTUNAM	Get user name
375	147	1	STUNAM	Set user name
375	147	2	GTPNAM	Get program name
375	147	3	STPNAM	Set program name
375	150	0	SJBPRI	Set job execution priority
375	150	1	GSJPRV	Get or set job privileges
375	151		SPFLAG	Enable/disable spooler flag pages
375	151		SPHOLD	Set spooler HOLD/NOHOLD
375	151		SPWIDE	Set spooler flag page width
375	152	0	SELOPT	Select terminal option
375	153	0	NONINT	Set [non]interactive job status
375	154	0	STTSPD	Set line baud rate
375	154	1	STTXON	Reset line XOFF status
375	154	2	RTTDTR	Raise line DTR
375	154	3	LTDDTR	Lower line DTR
375	155	1	SCLXON	Clear a CL units XOFF status
375	155	2	SCLRES	Reset a CL unit

EMT	Code	Chan	Name	Description
375	155	0	GETCL	Assign CL unit to a line
375	156	0	ALCDEV	Allocate a device for exclusive use
375	156	1	DEALOC	Deallocate a device from exclusive use
375	156	2	TSTALC	Test device exclusive use allocation
375	157	0	MONJOB	Start monitoring job status
375	157	1	NOMONJ	Stop monitoring job status
375	157	2	STAMON	Broadcast status to monitoring jobs
375	160	0	GETCXT	Acquire another job's file context
375	161	0	MAKWIN	Create a refreshable process window
375	161	1	SELWIN	Select current process window
375	161	2	DELWIN	Delete a process window
375	161	3	SPNWIN	Suspend window processing
375	161	4	RSMWIN	Resume window processing
375	161	5	PRTWIN	Print window contents (via WINPRT)
375	162	0	GISUBP	Initiate or switch to a subprocess
375	162	1	INISBP	Initiate a subprocess
375	162	2	GOSUBP	Switch to a subprocess
375	163		MNTLD	Mount a logical disk

Appendix C

Subroutines Used in Example Programs

C.1 PRTOCT—Print an octal value

The following subroutine accepts a value in R0 and prints the six-digit octal representation of that value at the terminal.

```
.TITLE PRTOCT
.ENABLE LC
; Print octal value of the word in R0
.MCALL .PRINT
.GLOBAL PRTOCT
PRTOCT: MOV    R1,-(SP)          ;Save R1-R3 on the stack
        MOV    R2,-(SP)
        MOV    R3,-(SP)
        MOV    #EOW,R2         ;Point to end of 6 char output buffer
        MOV    #6,R3           ;Set up counter for 6 chars
1$:     MOV    R0,R1            ;Set up mask for low 3 bits (1s digit)
        BIC    #177770,R1      ;Get low 3 bits
        ADD    #'0,R1          ;Convert to ASCII
        MOVB   R1,-(R2)        ;Fill octal digits in from end
        CLC
        ROR    R0
        ASH    #-2,R0          ;Shift out bits just converted
        SOB    R3,1$          ;Repeat for 6 chars
        .PRINT #CHARS         ;Display result at console
        MOV    (SP)+,R3        ;Restore registers R1-R3
        MOV    (SP)+,R2
        MOV    (SP)+,R1
        RETURN
CHARS:   .BLKB  6              ;6 char output buffer
EOW:     .ASCII <200>         ;No CR terminator for .PRINT
        .EVEN
        .END
```

C.2 PRTDEC—Print a decimal value

The following subroutine accepts a value in R0 and prints the decimal representation of that value at the terminal with no leading zeroes.

```

        .TITLE  PRTDEC
        .ENABLE LC
; Print the decimal value of the word in R0
        .MCALL  .PRINT
        .GLOBL  PRTDEC
PRTDEC: MOV     R1,-(SP)      ;Save R1 and R2
        MOV     R2,-(SP)
        MOV     #BUFEND,R2   ;Point to end of output buffer
        MOV     R0,R1        ;Set up for DIV
1$:     CLR     R0            ;Clear high word for DIV
        DIV     #10.,R0      ;Get least significant digit
        ADD     #'0,R1       ;Make remainder into ASCII
        MOVB    R1,-(R2)     ;Save digit in output buffer
        MOV     R0,R1        ;Set up for next DIV
        BNE     1$           ;Until nothing left
        .PRINT  R2           ;Display number at the terminal
        MOV     (SP)+,R2     ;Restore R1 and R2
        MOV     (SP)+,R1
        RETURN
        .BLKB   5            ;5 char output buffer
BUFEND: .BYTE   200          ;No CR terminator for .PRINT
        .EVEN
        .END

```

C.3 PRTDE2—Print a two-digit decimal value

The following subroutine accepts a value in R0 and prints the decimal representation of it at the terminal. The value must be in the range of 0 to 99.

```

        .TITLE  PRTDE2
        .ENABLE LC
; Print a 2-digit decimal value from R0
        .MCALL  .PRINT
        .GLOBL  PRTDE2
PRTDE2: MOV     R1,-(SP)
        MOV     R2,-(SP)
        MOV     R3,-(SP)
        MOV     R0,R1        ;Get copy of char in R1
        MOV     #<BUFFER+2>,R2 ;Point to buffer
        MOVB    #200,(R2)    ;Set end for .PRINT
        MOV     #2,R3
2$:     CLR     R0            ;Clear high word for DIV
        DIV     #10.,R0      ;Get low digit
        ADD     #'0,R1       ;Convert low digit to ASCII
        MOVB    R1,-(R2)     ;Put digit in char buffer
        MOV     R0,R1        ;Roll quotient for next DIV
        SOB     R3,2$        ;Two digits only
        .PRINT  #BUFFER      ;Display the result
        MOV     (SP)+,R3
        MOV     (SP)+,R2
        MOV     (SP)+,R1
        RETURN
BUFFER: .BLKB   6
        .EVEN
        .END

```

C.4 PRTR50—Print a RAD50 word at the terminal

The following subroutine accepts a one word RAD50 value in R0 and prints its ASCII representation at the terminal.


```

        .TITLE  PRTR50
        .ENABL  LC
; Display the RAD50 value of a word passed in R0
        .MCALL  .TTYOUT
        .GLOBL  PRTR50
PRTR50: MOV     R1,-(SP)      ;Need place to store last char
        MOV     R4,-(SP)      ;Need R4 and R5 for DIV
        MOV     R5,-(SP)
        MOV     R0,R5        ;Get copy of r0 into dividend
        CLR     R4           ;Zero high word for DIV
        DIV     #50,R4       ;Extract last char
        MOV     R5,R1        ;Save last char (remainder)
        MOV     R4,R5        ;Move quotient for next DIV
        CLR     R4           ;Zero high word for second DIV
        DIV     #50,R4       ;Get first and second chars
        .TTYOUT R5OTBL(R4)    ;Display first char (quotient)
        .TTYOUT R5OTBL(R5)    ; and second char (remainder)
        .TTYOUT R5OTBL(R1)    ; and last char
        MOV     (SP)+,R5      ;Restore registers
        MOV     (SP)+,R4
        MOV     (SP)+,R1
        RETURN
        .NLIST  BEX
R5OTBL: .ASCII  / ABCDEFGHIJKLMNOPQRSTUVWXYZ$. 0123456789/
        .EVEN
        .END

```

C.5 R50ASC—Convert a RAD50 string into an ASCIZ string

The following subroutine accepts a pointer in R0 to an argument block containing pointers to a RAD50 input string and an output buffer to hold the ASCII equivalent of the RAD50 string, and a word containing the number of characters to be converted.

```

        .TITLE  R50ASC
; Convert RAD50 value into ASCII
; Output buffer must be at least 3 characters extra long
        .GLOBL  R50ASC
        .DSABL  GBL
R50ASC: MOV     R1,-(SP)      ;Pointer to input buffer
        MOV     R2,-(SP)      ;Pointer to output buffer
        MOV     R3,-(SP)      ;Number of chars to convert
        MOV     R4,-(SP)      ;Dividend low word and quotient
        MOV     R5,-(SP)      ;Dividend hi word and remainder
        MOV     (R0)+,R1      ;Fetch input buffer address
        MOV     (R0)+,R2      ;Output buffer address
        MOV     (R0),R3       ;Number of chars to convert
        TST     R3           ;Any chars to convert?
        BLE     2$           ;Branch if not
1$: ADD     #3,R2             ;Point to end of 3 char set
        MOV     (R1)+,R5      ;Fetch RAD50 value
        CLR     R4           ;Set up for divide
        DIV     #50,R4       ;Get first char
        MOVB    R5OTBL(R5),-1(R2) ;Put 3/3 into output buffer
        MOV     R4,R5        ;Set up for divide
        CLR     R4
        DIV     #50,R4       ;Convert next char
        MOVB    R5OTBL(R5),-2(R2) ;Put 2/3 into output buffer
        MOVB    R5OTBL(R4),-3(R2) ;Put 1/3 into output buffer
        SUB     #3,R3        ;Converted 3 chars this round
        BGT     1$          ;Branch if more chars left to convert
2$: ADD     R3,R2            ;Locate last real char
        CLRB    (R2)         ;Make output string ASCIZ

```

```

MOV    (SP)+,R5
MOV    (SP)+,R4
MOV    (SP)+,R3
MOV    (SP)+,R2
MOV    (SP)+,R1
RETURN
.NLIST BEX
R50TBL: .ASCII / ABCDEFGHIJKLMNOPQRSTUVWXYZ$. 0123456789/
.EVEN
.END

```

C.6 DSPDAT—Print a date value at the terminal

The following subroutine accepts a date value in R0 and prints the date representation at the terminal.

```

.TITLE DSPDAT
.ENABL LC
; Print a date value from R0
.MCALL .PRINT
.GLOBAL DSPDAT,PRTDEC
MONMSK = 036000
DAYMSK = 001740
YRMSK = 000037
DSPDAT: MOV    R1,-(SP)
MOV    R0,R1          ;Get copy of date in R1
BIC    #~CDAYMSK,R0    ;Mask in only day bits (5-9)
ASH    #-5,R0          ;Shift down
CALL   PRTDEC          ;Print it out
MOV    R1,R0          ;Get fresh copy of date
BIC    #~CMONMSK,R0    ;Use only month bits (10-13)
ASH    #-7,R0          ;Shift down to index into month table
ADD    #MONTBL,R0      ;Point into table
.PRINT R0              ;Display the month
MOV    R1,R0          ;Fresh copy again
BIC    #~CYRMSK,R0     ;Use only year bits
ADD    #72.,R0         ;Year since 1972
CALL   PRTDEC          ;Display the year
MOV    (SP)+,R1
RETURN
.NLIST BEX
MONTBL: .ASCII /-MON-/<200><0><0>
.ASCII /-Jan-/<200><0><0>
.ASCII /-Feb-/<200><0><0>
.ASCII /-Mar-/<200><0><0>
.ASCII /-Apr-/<200><0><0>
.ASCII /-May-/<200><0><0>
.ASCII /-Jun-/<200><0><0>
.ASCII /-Jul-/<200><0><0>
.ASCII /-Aug-/<200><0><0>
.ASCII /-Sep-/<200><0><0>
.ASCII /-Oct-/<200><0><0>
.ASCII /-Nov-/<200><0><0>
.ASCII /-Dec-/<200><0><0>
.EVEN
.END

```

C.7 DSPTI3—Display a 3-second format time value

The following subroutine accepts a special three-second time value in R0 and prints the time value at the terminal.

```

        .TITLE DSPTI3
        .ENABL LC
; Display special 3-sec format time value from R0
        .MCALL .TTYOUT
        .GLOBL DSPTI3,PRTDE2
DSPTI3: MOV     R1,-(SP)
        MOV     R0,R1          ;Set up for divide
        CLR     R0
        DIV     #20.,R0        ;Get # of 3-SEC'S since midnight
        MOV     R1,-(SP)      ;Put on stack
        ASL     R1             ;2X 3-SEC'S
        ADD     R1,(SP)        ;Plus 1X gives 3X = seconds
        MOV     R0,R1          ;Get rest of time
        CLR     R0             ;Set up for next divide
        DIV     #60.,R0        ;Get number of minutes
        MOV     R1,-(SP)      ;And save on stack
        CALL    PRTDE2         ;What's left is hours, display
        .TTYOUT #' :
        MOV     (SP)+,R0       ;Recover minutes
        CALL    PRTDE2         ;Display minutes
        .TTYOUT #' :
        MOV     (SP)+,R0       ;Recover seconds
        CALL    PRTDE2
        MOV     (SP)+,R1
        RETURN
        .END

```

C.8 ACRTI3—Convert a time value to special 3-second format

The following subroutine accepts a time value from the terminal and converts it to a special three-second internal format. The value is returned in R0.

```

        .TITLE ACRTI3
        .ENABL LC
; Accept a time from the keyboard and return it in
; a special 3-second format in R0
        .GLOBL ACRTI3
ACRTI3: MOV     R1,-(SP)
        MOV     R2,-(SP)
        MOV     R3,-(SP)
        CLR     HOURS         ;Make sure it's reentrant
        CLR     MINITS
        CLR     R2
        CALL    GETNUM         ;Accrue decimal hours
        BCS     2$             ;Return with error
        MUL     #<60.*20.>,R3   ;Convert hours to 3-sec periods
        MOV     R3,HOURS       ;Save hours in 3-sec units
        TST     NUMERR         ;Did we hit end of input?
        BNE     1$             ;Yes, return value
        CALL    GETNUM         ;Accrue decimal minutes
        BCS     2$             ;Return with error
        MUL     #20.,R3        ;Convert minutes to 3-sec periods
        MOV     R3,MINITS      ;Save minutes in 3-sec units
        TST     NUMERR         ;End of input?
        BNE     1$             ;Yes, return value
        CALL    GETNUM         ;Accrue decimal seconds
        BCS     2$             ;Return with error
        CLR     R2             ;Convert seconds into 3-sec periods
        DIV     #3,R2          ;Quotient stays in R2
1$:     ADD     MINITS,R2       ;Add in 3-secs from minutes
        ADD     HOURS,R2       ;Add in 3-secs from hours
        MOV     R2,R0          ;Return it in R0

```

```

        CLC                ;Say no error
        BR      3$         ;Return
2$:     SEC                ;Say there was an error
3$:     MOV      (SP)+,R3
        MOV      (SP)+,R2
        MOV      (SP)+,R1
        RETURN

GETNUM: CLR      NUMERR    ;Say no error yet
        CLR      R3       ;Initialize number
1$:     MOVB     (R0)+,R1   ;Get next digit into R1
        CMPB     R1,#'0    ;Less than '0?
        BLT      2$        ;Not a digit
        CMPB     R1,#'9    ;Greater than '9?
        BGT      2$        ;Not a digit
        MUL      #10.,R3   ;Shift previous digits
        SUB      #'0,R1    ;Convert current digit to binary
        ADD      R1,R3     ;And include in number
        BR      1$        ;Get digits til next separator
2$:     CMPB     R1,#':'    ;Is it a legal separator?
        BEQ      3$        ;Yes, return
        INC      NUMERR    ;Say it may be end
        TSTB     R1        ;Was it a nul (end of input string)?
        BEQ      3$        ;Yes, return
        SEC                ;No, say invalid input
        RETURN          ;Error return
3$:     CLC                ;No error
        RETURN

HOURS:  .WORD    0
MINITS:  .WORD    0
NUMERR:  .WORD    0
        .END

```

C.9 ACRDEC—Convert an ASCII decimal value to a numeric value

The following subroutine converts the ascii decimal value in R1 to a numeric value and places it in R0.

```

        .TITLE  ACRDEC
;
;  Accrue an ASCII decimal number from an input string into a word in R0
;  Accepts values over range 0 to 32767.
;
;  Inputs:
;      R1      Points to buffer containing ASCII string containing number
;
;  Outputs:
;      R0      Contains internal representation of number
;      R1      Points past end of input string
;
        .MCALL  .PRINT
        .GLOBL  ACRDEC
;
ACRDEC: MOV      R2,-(SP)    ;Save Registers
        MOV      R3,-(SP)
        CLR      R3        ;Clear number
        CLR      NEGFLG    ;Init. NEGative FLag
1$:     MOVB     (R1)+,R0   ;Fetch next digit
        BEQ      DONE      ;Branch if no more to convert
        BIC      #'C177,R0 ;Mask out anything non-ascii
        CMPB     R0,#'-    ;Negative number?
        BNE      2$        ;BR if not
        INC      NEGFLG    ;Remember negative

```

```

2$:   BR      1$           ;Get next char
      CMPB   RO,*'.       ;Decimal point?
      BEQ    DONE         ;Branch if so, integers only
      CMPB   RO,*40       ;Space?
      BEQ    DONE         ;Branch if separator
      CMPB   RO,*54       ;Comma?
      BEQ    DONE         ;Branch if separator
      SUB    #'0,RO       ;Convert ascii to number
      BMI    INVNUM       ;Branch if invalid number
      CMP    RO,*9.       ;Greater than 0?
      BGT    INVNUM       ;BR if invalid number
      MUL    #10.,R3      ;Cycle previous digits
      BCS    INVNUM       ;Branch on overflow
      ADD    RO,R3        ;Add in new digit
      BCS    INVNUM       ;Branch on overflow
      BR     1$           ;Repeat until all digits included
DONE:  MOV    R3,RO       ;Return result in RO
      TST    NEGFLG       ;Negate?
      BEQ    1$           ;BR if not
      NEG    RO           ;Negate!
1$:   CLC              ;Signal OK return
      BR     ALLDON       ;Skip error stuff
INVNUM: .PRINT #BADNUM    ;Bad number message
      SEC
ALLDON: MOV    (SP)+,R3    ;Restore registers
      MOV    (SP)+,R2
      RETURN
      .MLIST BEX
NEGFLG: .WORD  0
BADNUM: .ASCIZ  /Illegal number./<7>
      .LIST  BEX
      .EVEN
      .END

```

C.10 RADASC—Convert a RADIX-50 string to ASCII

The following subroutine converts a RADIX-50 string whose address is in R5 to ASCII. R4 contains the number of words to convert and R3 points to the end of the converted string.

```

      .TITLE RADASC
      .ENABL LC

      .GLOBL RADASC
;
; Convert a RAD50 string whose address is in r5 to ascii. R4 contains
; the number of words to convert and r3 contains the storage result
;
RADASC:
      MOV    R1,-(SP)      ; Save r1
      MOV    R2,-(SP)      ; Save r2
1$:   MOV    (R5)+,R1      ; Obtain a word to convert
      CLR    RO           ; Clear high order
      DIV    #50*50,RO     ; Isolate the high letter
      MOVB   RADTAB(RO),(R3)+ ; Store converted character
      CLR    RO           ; Clear high order
      DIV    #50,RO        ; Isolate the middle letter
      MOVB   RADTAB(RO),(R3)+ ; Store converted character
      MOVB   RADTAB(R1),(R3)+ ; Store last converted character
      SOB    R4,1$        ; Loop until all words converted
      MOV    (SP)+,R2      ; Restore r2
      MOV    (SP)+,R1      ; Restore r1

```

RETURN

.NLIST BEX

RADTAB: .ASCII / ABCDEFGHIJKLMNOPQRSTUVWXYZ\$. 0123456789/

.LIST BEX

.EVEN

.END

Appendix D

DIBOL TSX-Plus Support Subroutines

A set of subroutines is provided with TSX-Plus to perform DIBOL record locking and message transmission functions. DIBOL ISAM files are not supported by TSX-Plus. These subroutines may not be used with DBL, which provides most of their functionality separately. Note that if these TSX-Plus features are to be used, they must be enabled when the TSX-Plus system is generated.

D.1 Record locking subroutines

The record locking subroutines coordinate access to a common file being shared and updated by several TSX-Plus users. The five subroutines parallel the operation of the DIBOL statements: OPEN, CLOSE, READ, WRITE and UNLOCK. The normal DIBOL I/O statements cannot be used to perform record locking under TSX-Plus.

D.1.1 Opening the file

The first subroutine is used to open a shared file in update mode. The form of the call is:

```
XCALL FOPEN(chan,devlbl,errflg)
```

where *chan* is a decimal expression that evaluates to a number in the range 1–15. This is the channel number used in associated calls to FREAD, FWRIT, FUNLK and FCLOS subroutines. *Devlbl* is the name of an alphanumeric literal, field or record that contains the file specification in the general form *dev:filnam.ext*. The file size must not be specified with the file name. An optional /W switch may be appended to the file name to cause the “WAITING FOR *dev:file*” message to be printed. *Errflg* is a numeric variable capable of holding at least two digits, into which is stored an indication of the result of the FOPEN call. The following values are returned:

Error Code	Meaning
0	No error. File is open and ready for access.
17	File name specification is invalid.
18	File does not exist or channel is already open.
72	Too many channels are open to shared files. Re-gen TSX-Plus and increase the value of MAXSFC parameter.
73	Too many shared files are open. Re-gen TSX-Plus and increase the value of MAXSF parameter.

The FOPEN subroutine should only be used to open files that will be updated by several users. The normal DIBOL OPEN READ/WRITE sequence should be used for other files. Several files may be opened for update by calling FOPEN with different channel numbers. The ONERROR DIBOL statement does not apply to these record locking subroutines. Instead the *errflg* argument is used to indicate the outcome of the operation.

D.1.2 Locking and reading a record

The FREAD subroutine is used to lock and read a record. The form of the call is:

```
XCALL FREAD(chan,record,rec-#, 'T' or 'W',errflg)
```

where *chan* is a decimal expression in the range 1-15 that identifies a channel previously opened by FOPEN. *Record* is the name of the record or alphanumeric field in which the record read is to be placed. *Rec-#* is a decimal expression that specifies the sequence number of the record to be read. This value must be between 1 and the total number of records in the file. It may be larger than 65535.

'T' If 'T' is specified as the fourth parameter, FREAD will return a value of 40 in *errflg* if the requested record is locked by some other user.

'W' If 'W' is specified, FREAD will wait until the record is unlocked by all other users and will never return the record-locked error code.

Errflg is a decimal variable into which is stored one of the following values:

Error Code	Meaning
0	No error. Record has been locked and read.
1	End-of-file record has been read.
22	I/O error occurred on read.
28	Invalid record number (possibly beyond end of file).
40	Record locked by another user. (Only returned if 'T' is specified as fourth argument.)
71	Channel was not opened by calling FOPEN, or channel not open.
72	Request to lock too many blocks in file. Re-gen TSX-Plus and increase value of MXLBLK parameter.

The FREAD subroutine functions like the DIBOL READ statement. However, whereas the DIBOL READ statement always returns an error code (40) if the requested record is locked, FREAD offers the user a choice: If 'T' is specified as the fourth argument to FREAD, a code of 40 will be returned in *errflg* if the record is already locked. If 'W' is specified as the fourth argument and the record is locked, FREAD does not return an error code, but rather waits until the requested record is unlocked. It is *much* more efficient to wait for a locked record by using the 'W' option rather than re-executing the FREAD with the 'T' option. It may be desirable to perform the first FREAD using the 'T' option. If the record is locked a "WAITING FOR RECORD ..." message can be displayed on the user's console and another FREAD can be issued with the 'W' option to wait for the record. On return from this FREAD the WAITING message can be erased.

Note that although record locking is requested on a *record by record* basis, the actual locking is done on a *block within file* basis. A block contains 512 characters. The result of this is that a record is locked if any record contained in the same block(s) as the desired record is locked.

Once a record is locked and read using FREAD, the record remains locked until the program performs one of the following operations:

- Issues an FWRIT to the channel from which the record was read.
- Issues another FREAD to the channel.
- Issues an FUNLK to the channel.
- Issues an FCLOS to the channel.
- Terminates execution by use of the STOP statement or because of an error.

The same set of rules that apply to the DIBOL READ statement apply to FREAD.

D.1.3 Writing a record

The FWRIT subroutine is called to write a record to a shared file. The form of the call is:

```
XCALL FWRIT(chan,record,rec-#,errflg)
```

where *chan* is the channel number associated with the file, *record* is the name of the record or alphanumeric field that contains the record to be written, *rec-#* is a decimal expression that specifies the sequence number of the record to be written (it may be larger than 65535), and *errflg* is a decimal variable into which is stored one of the following values:

<i>Error Code</i>	<i>Meaning</i>
0	No error.
22	I/O error occurred during write or channel is not open.
28	Bad record number specified.

The FWRIT subroutine writes the indicated record to the file then unlocks any blocks that were locked by the program. FWRIT appends a <CR><LF> to the end of the written record as does the DIBOL WRITE statement. The rules for the DIBOL WRITE statement also apply to FWRIT.

D.1.4 Unlocking records

The FUNLK subroutine is used to unlock records that were locked by calling FREAD. The form of the call is:

```
XCALL FUNLK(chan)
```

where *chan* is the channel number.

D.1.5 Closing a shared file

The FCLOS subroutine is called to close a channel that was previously opened to a shared file by calling FOPEN. The form of the call is:

```
XCALL FCLOS(chan)
```

where *chan* is the channel number.

FCLOS unlocks any locked records and closes the file. Other users accessing the file are unaffected. After calling FCLOS, the channel may be reopened to some other file.

D.1.6 Record Locking Example

In the following example a program performs the following functions:

1. Opens a shared file named INV.DAT on channel 2.
2. Reads a record whose record number is stored in RECN into the field named ITEM and waits if the record is locked by another user.
3. Updates the information in the record.
4. Rewrites the record to the same position in the file.
5. Closes the shared file.

```
XCALL FOPEN(2, 'INV.DAT', ERRFL)
XCALL FREAD(2, ITEM, RECN, 'W', ERRFL)
; <update record>
XCALL FWRT(2, ITEM, RECN, ERRFL)
XCALL FCLOS(2)
```

D.1.7 Modifying programs for TSX-Plus

It is a straightforward process to modify DIBOL programs to use the TSX-Plus record locking subroutines. OPEN, CLOSE, READ, WRITE, and UNLOCK statements that apply to shared files must be replaced by the appropriate subroutine calls. Error conditions must be tested by IF statements following the subroutine calls rather than by using the ONERROR statement.

D.2 Message communication subroutines

Three subroutines are included in the DIBOL support package to allow programs to transfer messages to each other. When running under TSX-Plus these subroutines must be used instead of the DIBOL SEND and RECV statements.

D.2.1 Message Channels

Messages are transferred to and from programs by using TSX-Plus Message Channels. A message channel accepts a message from a sending program, stores the message in a queue associated with the channel and delivers the message to a receiving program that requests a message from the channel. Message channels are totally separate from I/O channels. The total number of message channels is defined when TSX-Plus is generated.

Each active message channel has associated with it a one to six character name that is used by the sending and receiving programs to identify the channel. The names associated with the channels are defined dynamically by the running programs. A message channel is said to be *active* if any messages are being held in the queue associated with the channel or if any program is waiting for a message from the channel. When message channels become inactive they are returned to a free pool and may be reused by another program.

The DIBOL SEND command directs a message to a program by using the name of the receiving program. Under TSX-Plus, a sending program transmits a message using an arbitrary channel name. *Any* program may receive the message by using the same channel name when it requests a message.

D.2.2 Sending a Message

The MSEND subroutine is called to queue a message on a named channel. If other messages are already pending on the channel the new message is added to the end of the list of waiting messages. The form of the call is:

XCALL MSEND(chan,message,errflg)

where *chan* is an alphanumeric literal or variable that contains the channel name (1 to 6 characters), *message* is an alphanumeric or decimal literal, field or record that contains the message to be sent, *errflg* is a decimal variable into which will be stored one of the following values:

Error Code	Meaning
0	No error. Message has been sent.
1	All message channels are busy. (Re-gen TSX-Plus and increase the value of MAXMC parameter).
2	Maximum allowed number of messages are being held in message queues. (Re-gen TSX-Plus and increase the value of MAXMSG parameter).

Note that the maximum message length that may be transferred is defined during system generation by the MSCHRS parameter. If a message longer than this is sent, only the first part of the message will be delivered.

D.2.3 Checking for Pending Messages

The MSGCK subroutine may be called to determine if any messages are pending on a named channel. The form of the call is:

XCALL MSGCK(chan,message,errflg)

where *chan* is an alphanumeric literal or variable that contains the name of the channel (1 to 6 characters), *message* is an alphanumeric or decimal field or record where the received message is to be placed, *errflg* is a decimal variable into which will be stored one of the following values:

Error Code	Meaning
0	No error. A message has been received.
3	No message was queued on the named channel.

If a received message is shorter than the receiving message field the remainder of the field is filled with blanks. If the message is longer than the field, only the first part of the message is received.

D.2.4 Waiting for a Message

The MSGWT subroutine is used by a receiving program to suspend its execution until a message is available on a named channel. It is *much* more efficient for a program to wait for a message by calling MSGWT rather than repeatedly calling MSGCK. The form of the call is:

XCALL MSGWT(chan,message,errflg)

where the arguments have the same meaning as for MSGCK, and the following values may be returned in *errflg*:

<i>Error Code</i>	<i>Meaning</i>
0	No error. A message has been received.
1	All message channels are busy.

D.2.5 Message Examples

In the following example a program sends a message to another program by using a message channel named SORT and then waits for a reply to come back through a message channel named REPLY.

```

XCALL MSEND('SORT','DK:PAYROL.DAT',ERRFL)
IF(ERRFL.NE.0)GO TO ERROR
XCALL MSGWT('REPLY',MSGBF,ERRFL)
IF(ERRFL.NE.0)GO TO ERROR

```

D.3 Using the subroutines

The subroutines described above are part of the MACRO program called DTSUB.MAC. Once assembled, the object file for DTSUB (DTSUB.OBJ) may be linked with DIBOL programs that use the record locking or message facilities. An example of a LINK command is shown below.

```

R LINK
*PROG=PROG,DTSUB,DIBOL

```

D.4 Miscellaneous functions

D.4.1 Determining the TSX-Plus line number

The TSLIN subroutine can be called to determine the number of the TSX-Plus time-sharing line from which the program is being run. Real lines are numbered consecutively starting at 1 in the same order they are specified when TSX-Plus is generated. Detached job lines occur next and subprocesses are numbered last.

The form of the call of TSLIN is:

```
XCALL TSLIN(lnum)
```

where *lnum* is a numeric variable capable of holding at least two digits into which is stored the TSX-Plus line number value.

Index

- .ABTIO, 183
- ACRDEC, example program, 204
- ACRTI3, example program, 203
- Activation characters, 21
 - Checking for, 40
 - CTRL-D, 172
 - Defining, 26
 - Field width, 27
 - ODT activation mode, 35
 - Resetting, 27
 - Time-out activation, 38
- Adapting RT-11 Real-time programs, 157
- Address 0
 - Debugger, 172
- Administrative control, 2
- Allocating devices, 78
- ATTVEC, example program, 155
- Baud rate
 - Setting, 72
- Block locking
 - See Shared files.
- BPT Instruction
 - Debugger, 168
- Break sentinel control, 45
- BYPASS, 125, 129
- C1 handler
 - see CL handler.
- Caching
 - Data, 103
 - See *TSX-Plus System Manager's Guide*.
 - Shared file, 94
- .CALLK, 183
- Carriage-return
 - Automatic line-feed, 28
- .CDFN, 183
- .CHAIN
 - Record locking, 183
- Channel numbers, 186, 187
- Chapter summaries, 3
- Character echoing, 26
- .CHCOPY, 183
- CL device control, 75
- CL handler, 113
 - As a spooled device, 116
 - control character processing, 116
 - Cross connection, 115
 - Flow control, 116
 - I/O operations, 116
 - Installing, 113
 - Modem control, 113, 119
 - Special functions, 117
 - VTCOM/TRANSF support, 114
- CL units, 3
 - Assigning, 75
 - reset, 77
 - XON, 77
- .CNTXSW, 183
- Command files
 - Controlling input, 27
- Common data areas, 159
- Communication
 - TSX-Plus to other machines, 115
- Communication line handler
 - See CL handler
- Completion routine
 - Connecting to an interrupt, 153
 - Scheduling, 156
 - Scheduling message, 109
 - Terminal input, 46
- Configuration requirements, 1
- Control characters
 - CTRL-], 25
 - CTRL-C, 26
 - CTRL-D, 167, 168, 172
- Control-O, 40
- Cooperative file access
 - See Shared files.
- .CRAW, 183
- Creating a window, 84
- Cross connection
 - clearing, 78
- .CSIGEN, 183
- .CSISPC, 183
- CTRL-D breakpoints, 170
- D-space, 14
 - Physical address, 144
- Data caching, 103
 - See *TSX-Plus System Manager's Guide*.
 - Shared file, 94

- Suppression of, 104
- Data watchpoint
 - Debugger, 167
- .DATE, 183
- Debugger, 167
 - Address 0, 172
 - Address relocation, 171
 - Breakpoints, 170
 - CTRL-D, 172
 - Data watchpoints, 170, 171
 - Features, 167
 - Format register, 171
 - I-space contents, 170
 - Internal registers, 171
 - Mask register, 172
 - Overlays, 172
 - RAD50 values, 170
 - Relocation base, 170
 - Shared run-time, 172
 - Single stepping, 169, 170, 172
 - Special activation characters, 172
 - Special terminal mode, 172
 - Speed, 172
 - Symbolic decoding, 169, 172
 - System generation, 167
 - Traps to 4 & 10, 173
- DELETE
 - Rubout filler character, 25
- Deleting windows, 87
- DEMOPA, example program, 177
- Detached jobs, 82
 - Checking status of, 44
 - Starting, 43
- .DEVICE, 145, 183
- Device allocation, 78
- Device handlers
 - AT, 129
 - CI, 113
 - CL, 113
 - DM, 124
 - DU, 125
 - IB, 125
 - LD, 126
 - MU, 129
 - UB, 129
 - VM, 129
 - XL, 131
- DIBOL, 186
 - Record locking, 186
 - Record locking procedures, 93
 - SEND statement, 186
 - Support subroutines, 207
- DILOG DQ215, 125
- Directory caching
 - Dismounting a file structure, 48
 - Dismounting all file structures, 49
 - Mounting a file structure, 47
- Directory information
 - EMT to obtain, 60
- Dismounting a file structure
 - EMT to dismount all, 49
 - EMT to dismount one, 48
- DL-11, 1
- DM handler, 124
- .DRxxx, 183
- DSPDAT, example program, 202
- DSPTI3, example program, 202
- DTR
 - EMT to raise or lower, 74
- DU handler, 125
- DZ-11, 1
- Echo control, 26
- EMT
 - Differences, 183
 - Table, 193
 - TSX-Plus specific, 31
- Emulex SC02C, 125
- Error codes, 183
- Escape sequence
 - Processing, 25
- Exclusive access to a file, 94
- Exclusive system control, 141
 - Releasing, 141
- Execution priority, 17
- .EXIT, 183
- Extended memory, 159
- Extended memory regions, 9
 - and I/O, 16
- Fast mapping multiple PARs, 165
- Fast mapping to PLAS regions, 164
- .FETCH, 184
- Field width activation, 27
- Field width limit for TT input, 28
- File
 - Block locking, 94
 - Context, 82
 - Creation time, 61
 - Data caching, 103
 - Directory information, 60
 - Exclusive access, 94
 - Opening for shared access, 94
 - Protection modes, 94
 - Shared access, 94
- FILEX, 186
- Fixed offsets, 7
- Fixed-high-priority
 - Determining, 31
- Fixed-low-priority

- Determining, 31
- Flag pages
 - EMT to set width, 71
 - EMT to turn off for a device, 70
 - EMT to turn on for a device, 70
- Floating point, 185
- Foreground programs
 - See Real-time support.
- .FORK, 184
- FORTRAN
 - I/O page mapping, 187
 - Virtual arrays, 157, 186
- Global data areas, 159
- Global PLAS regions, 10
- Graphics
 - CTRL-], 25
- .GTJB, 184
- .GTLIN, 184
- .GVAL, 184
 - Checking I/O page mapping, 134
 - Special TSX-Plus use, 31
- Hardware requirements, 1
- .HERR, 184
- High-efficiency terminal mode, 24, 25, 27, 39
- HOLD mode
 - EMT to specify for a file, 69
- .HRESET, 184
- I- and D-space
 - enabling and disabling, 55
- I- and D-space I/O, 15
- I-space, 14
 - Physical address, 144
- I/O channels, 186, 187
- I/O page
 - Accessing, 9, 133
 - FORTRAN, 187
- I/O to extended memory, 16
- I/O to I- and D-space, 15
- IB handler, 125
 - IBSRQ unsupported, 126
- IEEE handler
 - see IB handler.
- IND
 - .ASK directive, 187
- INI-E3, 186
- .INTEN, 184
- Interactive jobs
 - Selecting dynamically, 71
- Interprogram communication, 105
 - Checking for messages, 106
 - Common memory areas, 159
 - Completion routine, 109
 - Message channels, 105
 - Sending a message, 105
 - Waiting for a message, 108
- Interrupt completion routine, 153
- Interrupt processing
 - (Diagram), 149
- Interrupts
 - Connecting to real-time jobs, 147
- Introduction, 1
- INTSVC, example program, 151
- IOABT parameter
 - Needed for VTCOM, 114
- Job context
 - Acquiring, 82
- Job monitoring, 79
- Job number
 - Determining, 31
- Job priority
 - Determining, 31
 - Setting, 66, 146
- Job scheduling, 2
- Job status information, 57
- Job status word
 - Non-wait TT I/O, 28
 - Setting virtual flag with SETSIZ, 191
 - Virtual-image flag, 9
- JSW
 - See Job status word.
- KILL command, 45
- KILL EMT, 44
- LD
 - Special functions, 126
- LD handler, 126
- Lead-in character, 23, 25
 - Determining, 31
- License number
 - Determining TSX-Plus, 31, 41
- License string
 - Determining TSX-Plus, 41
- Line number
 - Determining, 33
 - Determining primary, 31
- Line-feed
 - Echoing of, 28
 - Ignored with tape mode, 28
- Lines
 - Assigning to CL, 75
- LINK command
 - /XM switch, 9
- Link map, 167
- Local named regions, 10
- .LOCK, 184

- Locking a job in memory, 142
- Logical Disks
 - determining status, 50
 - Dismounting, 49
 - EMT to dismount all LD's, 52
 - Mounting, 90
- Logical subset disks
 - EMT to dismount all, 49
- Lower-case character input, 26
- .MAP, 184
- Mapping boundaries, 16
- Mapping interactions, 12
- Mapping to physical memory
 - EMT for, 139
- Maximum priority
 - Determining, 31
- Memory
 - Using as pseudo-disk, 16
- Memory allocation
 - EMT to control, 54
 - Setting size in SAV file, 189
- MEMORY command, 54, 189
- Memory mapping, 7
- Memory mapping conflicts, 12
- MEMSIZ parameter
 - and VM handler, 129
- Message channels
 - See Interprogram communication.
- Message communication
 - See Interprogram communication.
- Messages
 - Sending to another line, 42
- .MFPS, 184
- MicroPower/Pascal, 187
- Modification of shared files
 - Checking for, 103
- MONITOR command, 175
- Monitoring another job, 79
- MOUNT command, 47, 90
- Mounting a file structure, 47
- .MRKT, 184
- .MTPS, 185
- MU handler, 129
- Multi-terminal EMTs, 184
- .MWAIT, 185
- Named PLAS regions, 10
- NOHOLD mode
 - EMT to specify for a file, 69
- Non-blocking .TTOUTR, 22
- Non-blocking .TTYIN, 22
- Non-interactive jobs
 - Selecting dynamically, 71
- Non-wait TT I/O, 28
- Normal programs, 9
- Obtaining TSX-Plus system values, 31
- ODT
 - See Debugger.
- ODT activation mode, 35
- Opening shared files, 94
- Overlaid programs, 187
- Paint character
 - See Rubout filler character.
- PAR 7 mapping
 - Determining, 31
 - FORTTRAN, 187
- .PEEK, 136, 185
- Performance monitor, 175
 - Control EMTs, 177
 - Displaying results, 176
 - MONITOR command, 175
 - Starting, 175, 177
- PHYADD, example program, 144
- Physical address
 - D-space, 144
 - I-space, 144
- Physical address calculation, 143
- Physical address space, 8
- Physical memory
 - EMT to map to, 139
- PLAS, 164
- PLAS regions
 - and I/O, 16
 - Fast mapping to, 164
- PLAS support, 9
- .POKE, 136, 185
- Poke EMT, 137
- Primary line
 - Determining, 31
- Printing windows
 - EMT, 88
- Priorities, 17
- Priority level
 - Setting, 66, 145
- Privilege
 - EMT to determine or change, 67
 - Real-time, 133
- PRIVIR parameter, 66, 147
- Process windows
 - EMTs, 84
- Processor priority level
 - Setting, 145
- Processor status word, 145
 - Debugger, 173
- Program controlled terminal options
 - See Terminal control.
- Program debugger, 167

- Program name
 - EMT to get or change, 63
- Program's Logical Address Space
 - See PLAS support.
- Programmer number
 - Determining, 31, 54
- Project number
 - Determining, 31, 54
- .PROTECT, 157, 185
- Protected access to a file, 94
- PRTDE2, example program, 200
- PRTDEC, example program, 199
- PRTOCT, example program, 199
- PRTR50, example program, 200
- Pseudo-disk in memory, 16
- PSW, 145
- .PURGE
 - Shared files and, 97
- .QSET, 185
- R command
 - /DEBUG switch, 54
 - /NONINTERACTIVE switch, 71
 - /SINGLECHAR switch, 27
- R50ASC, example program, 201
- RADASC, example program, 205
- .RCVD, 185
- Read time-out value for TT inputs, 38
- Real-time, 133
 - Accessing the I/O page, 133, 135, 139
 - Adapting RT-11 programs, 157
 - Completion routine, 153
 - Device reset on exit, 145
 - Interrupt connections, 147
 - Locking a job in memory, 142
 - Mapping to physical addresses, 139
 - Physical address calculation, 143
 - Poke EMT, 137
 - Priority, 154
 - Privilege, 133
 - Suspending/resuming execution, 143
- Record locking, 93
 - .CHAIN, 93
 - DIBOL, 186
 - See Shared files.
- Redefining time-sharing lines, 3
- Reentrant run-times
 - See Shared run-time systems.
- Regions in extended memory, 9
- .RELEAS, 185
- Releasing exclusive system control, 141
- Requesting exclusive system control, 141
- Resident run-times
 - See Shared run-time systems.
- Restrictions
 - DM handler, 124
 - IB handler, 125
 - VM handler, 129
- Restrictions on .MAP EMT, 11
- Resuming windows, 88
- RK06/RK07 handler, 124
- RMON
 - Real-time support consideration, 134
 - Simulated, 7, 9
 - SYSGEN options word, 32
- .RSUM, 143
- RTSORT, 82
- Rubout filler character, 25
- Run-time systems
 - See Shared run-time systems.
- .SAVESTATUS
 - Shared files and, 96
- .SCCA, 185
- Scheduling a completion routine, 156
- Scheduling of jobs, 2
- .SDAT, 185
- .SDTTM, 185
- Selecting window, 87
- SEND
 - DIBOL statement, 186
- Sending messages
 - EMT for, 42
- Separate I- and D-space, 14
- Serial devices
 - CL units, 3
- .SERR, 185
- SET command
 - TT TAPE, 28
 - TT terminal-type, 53
- SETMAP algorithm, 15
- SETSIZ program, 54, 189
- Setting processor priority level, 145
- .SETTOP, 54, 185, 189
- .SFPA, 185
- Shared access to a file, 94
- Shared files, 93
 - .CHAIN, 93
 - Checking for modification of, 103
 - DIBOL, 186
 - Opening, 94
 - Protection modes, 94
 - Saving channel status, 96
 - Testing for locked blocks, 100
 - Unlocking a block, 102
 - Unlocking all locked blocks, 102
 - Waiting for locked block, 98
- Shared run-time systems, 159
 - Associating with job, 159

- Debugger, 172
- Mapping into job region, 12, 161
- With I- and D-space, 162
- Shared run-times
 - and I/O, 16
 - Fast mapping, 162
- Simulated RMON, 7
 - Access through page 7, 9
 - Real-time support consideration, 134
- Single character activation, 21, 27, 172
- Site information EMT
 - license number, 41
 - license string, 41
 - site name, 41
- Site name
 - Determining, 41
- Special Chain Exit, 18
- Speed
 - See terminal speed.
- SPFUN, 117, 185
 - BYPASS, 125, 129
- .SPND, 143
- Spooled CL units, 78
- Spooled devices, 131
- Spooler
 - Bypassing, 131
- Spooling
 - CL device, 116
 - Number of blocks in use, 33
 - Number of free spool blocks, 33
- .SRESET, 185
- Static region, 162
- Subprocess, 89
 - determining job number, 34
 - Disabling use of, 26
- Summaries of chapters, 3
- Suspending windows, 88
- Symbolic instruction decoding, 167
- .SYNCH, 185
- SYSGEN options word, 32
- SYSPRV privilege
 - Determining, 31
- System device
 - Determining, 31
- System generation parameter
 - CLDEF, 114
 - CLXTRA, 114
 - MEMSIZ, 129
- System resource management, 2
- System values
 - Obtaining, 31
- Tape mode, 28
- Terminal
 - Determining type, 53
 - Handler, 21
 - Input completion routine, 46
 - Setting options, 71
 - Setting speed, 72
- Terminal control, 21
 - Break sentinel, 45
 - Character echoing, 26
 - Checking for activation, 40
 - Checking for input errors, 37
 - Command file input, 27
 - Defining activation characters, 26
 - Determining input chars. pending, 38
 - Disabling subprocess use, 26
 - Echo control, 26
 - Field width activation, 27
 - Field width limit, 28
 - High-efficiency mode, 27, 39
 - Line-feed echoing, 28
 - Lower-case character input, 26
 - Non-wait TT I/O, 28
 - ODT activation mode, 35
 - Read time-out value, 38
 - Resetting activation characters, 27
 - Rubout filler character, 25
 - Single character activation, 27
 - Tape mode, 28
 - Transparency mode output, 26
 - VT52 & VT100 escape sequences, 25
- Terminal name
 - EMT to get or change, 65
- Time-out value for TT reads, 38
- Time-sharing lines
 - Assigning to CL, 3, 75
- .TLOCK, 185
- TRANSF program, 114
- Transparency mode output, 26
- Trap to 10
 - Debugger, 173
- Trap to 4
 - Debugger, 173
- TSX-Plus
 - Determining if under, 32
- TSX-Plus license number
 - Determining, 31
- TSX-Plus version number
 - Determining, 31
- TSXPM program, 175, 176
- TSXUCL
 - File Size, 31
- TT/CL cross connection, 115
- .TTINR, 22, 186
- .TTOUTR, 22, 186
 - Non-wait output, 28
- .TTYIN
 - Non-wait input, 28

- Time-out value, 38
- .TTYOUT, 186
- .TWAIT, 186
- UCI, 17
- .UNLOCK, 186
- Unlocking a job from memory, 143
- Unnamed PLAS regions, 10
- .UNPROTECT, 186
- User command interface, 17
- User name
 - Changing, 63
 - Determining, 63
- User PAR control, 56
- USERTS, example program, 160
- Version number
 - Determining, 31
- Virtual arrays
 - FORTRAN, 186
 - I/O page, 187
 - PAR 7 mapping, 157
- Virtual memory, 7
- Virtual programs, 9
 - Setting flag with SETSIZ, 191
- Virtual region mapping, 139
- Virtual to physical address, 8, 143
- Virtual windows, 11
- VM
 - Handler, 16
 - Initializing, 17
 - SET BASE command, 17
- VM handler, 129
- VMS file transfer, 115
- VT100 support, 25
- VT200 support, 25
- VT52 support, 25
- VTCOM
 - Spooled CL unit, 131
- VTCOM program, 114
- Watchpoint
 - Debugger, 167
- Window
 - Creating, 84
 - Deleting, 87
 - EMTs, 84
 - Printing, 88
 - Resuming, 88
 - Selecting, 87
 - Suspending, 88
- XL handler, 131