

7-	1	* * * TSX Initialization * * *
7-	2	* * * Initialization taking over control from RT-11 * * *
8-	1	LODINI -- Load a segment over TSINIT
9-	1	INIOVL -- Load system overlays over TSINIT
10-	1	ENTVEC -- Set up entry point vector for overlay
11-	1	KEYSEG -- Remember memory position of system overlays
12-	2	SETUMP -- Set up Unibus mapping if needed
13-	1	DEVVEC -- Set up device vectors
14-	1	SETVEC -- Set up an interrupt vector for a device
16-	2	LININI -- Initialize time-sharing lines
17-	1	DHLPRM -- Set line parameters for a DH11 line
18-	1	VHLPRM -- Set line parameter values for DHV11 line
19-	1	DZINIT -- Initialize a DZ11 multiplexer
20-	1	MUXVEC -- Set up interrupt vectors for a multiplexer
21-	1	DHINIT -- Initialize a DH11 multiplexer
22-	1	VHINIT -- Initialize a DHV11 multiplexer
24-	1	LINTYP -- Determine the type of a line
25-	1	* * * Initialization done with RT-11 running * * *
27-	1	* * * Subroutines * * *
27-	2	ALCWRK -- Allocate a work buffer
28-	1	ALCHRB -- Allocate Region Control Blocks for handlers
29-	2	LINCHK -- Check validity of T/S line
30-	1	OPNSWP -- Open system swap file
31-	1	OPNRSF -- Open PLAS region swap file
32-	1	SPLINI -- Initialize spooling system
33-	1	SPLCLD -- Set up spooling to a CL device
34-	1	CHKCLD -- See if a device name is a CL or C1 unit
35-	1	CVTDVU -- Convert device name to dev index and unit #
36-	1	FORCEO -- Force a 2-char dev name to unit 0
37-	1	ALOCBF -- Allocate buffer space
38-	1	ALCSLO -- Allocate silo buffers for lines
39-	1	ALBFX -- Allocate buffers in extended memory region
40-	1	OPNKMN -- Open channel to TSKMON
41-	1	CLINIT -- Initialize CL handler
42-	1	INDINI -- Initialize IND program
43-	1	UCLINI -- Initialize TSXUCL data file
44-	1	MEMINI -- Initialize memory management
45-	1	MEMTST -- Set up information about available memory space
46-	1	CXTALC -- Set up info about job context area
47-	1	MAPALC -- Allocate memory usage table
48-	1	SETJSZ -- Set up information about maximum job sizes
49-	2	PARSET -- Setup memory parity control
50-	1	GETHNL -- Load device handlers into memory
51-	1	LDHAND -- Load a device handler
52-	1	INSCK1 -- Determine if a handler should be installed
53-	1	INSCK2 -- Additional checking for handler installation
54-	1	STDVTB -- Set up device table entries for a device
55-	1	LDHNLQ -- Load device handler into low memory
56-	1	GETHNH -- Load handlers into extended memory
57-	1	LDHNHI -- Load device handler into extended memory
58-	1	STHNPV -- Initialize pointer vector in a handler
60-	1	DOHNLC -- Execute and handler load/fetch code
61-	1	LDREAD -- Perform I/O for handler load code
62-	1	HANMAP -- Set up KPAR5 to access a mapped handler
62-	42	HANUMP -- Turn off memory mapping to a handler
63-	18	FNDHRB -- Try to find a handler global region
64-	1	HANXMR -- Allocate XM region during handler load

Table of contents

65-	2	SETMID	-- Set up information about mapped devices
66-	1	OVLPOS	-- Determine which overlays go over TSINIT
67-	1	OVLBLD	-- Build overlay information table
68-	1	GETMAP	-- Load any mapped system code regions
69-	1	ALCOVL	-- Allocate space for a system overlay region
70-	1	OPTOVL	-- Check for optional system overlay regions
71-	1	OVLTRY	-- Find an overlay to place over TSINIT
72-	1	GETOVL	-- Load system overlay into high memory
73-	1	LODOVL	-- Read and relocate system overlay
74-	1	GETSRT	-- Load any shared run-time systems
75-	1	CSHBUF	-- Allocate space for data cache tables
76-	2	GETODT	-- Load ODT
77-	1	OPNCHN	-- Open a TSX-Plus channel
78-	1	SETCHN	-- Copy RT-11 channel information into TSX system chan
79-	1	SETSY	-- Set up information about SY device
80-	1	RTFTCH	-- Fetch a RT-11 device handler
81-	1	CHKMEM	-- Check for memory space overflow
82-	1	PRTUCT	-- Print octal value
83-	1	PRTDEC	-- Print decimal value
84-	1	PRTR50	-- Print Rad-50 value

```

1          .TITLE  TSINIT -- TSX startup initialization
2          .ENABL  LC
3          .ENABL  AMA
4          .DSABL  GBL
5          .CSECT  TSINIT
6          TSINIT:
7          ;
8          ; There are two external assembly-time switches related to assembling
9          ; TSINIT for execution on a PRO or a PDP-11.
10         ;
11         ; The following values for the PROASM flag are defined:
12         ; 0 ==> Assemble for PDP-11 (not Pro) only.
13         ; 1 ==> Assemble for Pro only.
14         ; 2 ==> Assemble for either PDP-11 or Pro execution.
15         ;
16         ; The following values for the PROCID flag are defined:
17         ; 0 ==> Do not lock system to ID number.
18         ; 1 ==> Lock system to ID number.
19         ;
20         .IF      NDF,PROASM      ; If PROASM not defined
21         PROASM  =      0          ; Default value for PROASM if not defined
22         .ENDC    ; NDF,PROASM
23         ;
24         .IF      NDF,PROCID      ; If PROCID not defined
25         .IF      EQ,<PROASM-1>    ; If assembling for PRO only
26         PROCID  =      1          ; Then check ID by default
27         .IFF      ; If not assembling for PRO only
28         PROCID  =      0          ; Then don't check ID number
29         .ENDC    ; EQ,<PROASM-1>
30         .ENDC    ; NDF,PROCID
31         ;
32         .IF      EQ,PROASM
33         .GLOBL  TSXPRO
34         TSXPRO  =      0          ; Define dummy base for TSXPRO if not PRO
35         .ENDC
36         ;
37         -----
38         ; TSINIT is the initialization module of TSX that is executed once
39         ; during system startup. Time-sharing character buffers and other
40         ; run-time data areas are allocated over TSINIT.
41         ;
42         ; Copyright 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1988, 1989.
43         ; S&H Computer Systems, Inc.
44         ; Nashville, TN USA
45         ;
46         ;
47         ; Macro calls
48         ;
49         .MCALL  .LOOKUP, .ENTER, .READW, .SAVESTATUS, .GVAL
50         .MCALL  .TRPSET, .SETTOP, .CLOSE, .TTYOUT, .PRINT, .PURGE
51         .MCALL  .DELETE, .WRITW, .SERR, .HERR, .EXIT, .UNLOCK
52         .MCALL  .FETCH, .RELEASE, .LOCK, .GTIM, .DATE, .DSTATUS
53         .MCALL  .SCCA, .CSTAT
54         ;
55         ; Global definitions
56         ;
57         .GLOBL  TSINIT, INITGO, INITOP, PPTERM, PROITP, PROASM, PISRT

```

```

58          .GLOBL  DSKBUF, PROBUF, FNDHRB, HANXMR
59          ;
60          ;   Following global only needed for the Pro distribution
61          ;   creation program - MAKPRO and installation program - INSTSX
62          ;
63          .IF      NE, PROASM
64          ;
65          ;** Assemble this code if we are generating for a Pro
66          ;
67          .GLOBL  PROSIZ, PROINI, PROLIN, PROHAN, PRONOP
68          .GLOBL  PIHAN, PIDPTR, PIDRIV
69          .IFF      ;NE, PROASM
70          ;
71          ;** Assemble this code if we are not generating for a Pro
72          ;
73          .GLOBL  TSXPRO
74          TSXPRO =      0
75          ;
76          ;** End of conditional Pro code
77          ;
78          .ENDC    ;NE, PROASM
79          ;
80          ;
81          ;   Global references
82          ;
83          .GLOBL  HANDSK, MAXDEV, NDVRCB, HANRCB, HANRCO
84          .GLOBL  LXCL, VSYDMP, STKLVL, INTSSZ, INDFIL, NXIVMH, EXCBUF
85          .GLOBL  VNUIP, NSIP, INSTBL, INSTBN, II$$SZ, DCCSIZ, VNUMDC, NUMCDB
86          .GLOBL  NSCP, SCPFHD, SP$$SZ, CSHDEV, CSHDVN, VMXCSH, CD$$SZ
87          .GLOBL  LSTPL, LMXNUM, MXCSR, MXVEC, RSR, INVEC, VHIMEM, CXTPAG
88          .GLOBL  LSWPBK, LSTSL, SWDBLK, SWPCHN, NUMDEV, CS$NMX, SCHED
89          .GLOBL  H. DSTS, DVSTAT, HANENT, H. GEN, FORK, INTEN, PNAME
90          .GLOBL  $SXON, LSW10, LHIRBB, LHIRBE, LHIRBP, LHIRBG, LHIRBA
91          .GLOBL  LHIRBS, LHIRBC, VNC$LO, VNCXOF, VNCXON, SDDVU, VMSCHR, MAXSLO
92          .GLOBL  MPAR0, MPAR16, PAREN1, MPARFL, TSEMT, VDBFLG, DX$EBA
93          .GLOBL  H. SIZ, HANSIZ, H. DVSZ, DEVSIZ, LOMAP, MMENBL, UPAR7
94          .GLOBL  PSW, HIMAP, FSTD1, LSTD1, LINBUF, LINSIZ, NUMCCB, TK1SEC
95          .GLOBL  FRKINI, FRKGEN, NUMFRK, FQ$$SZ, H. CSR, H. INS, VSWPSL, DMYDEV
96          .GLOBL  LINEND, LOTBUF, LOTSIZ, LOTEND, KMNTOP, KMNHI, NSL, NDL
97          .GLOBL  DX$MPH, DX$NHM, DX$IBH, HANPAR, HANXIT, MAPPAR, LINSPC
98          .GLOBL  KMNPGS, KMNSTK, KMNSTR, KMNCHN, SROMMR, KPARO, PROFLG
99          .GLOBL  EMMAP, IOMAP, SR3MMR, IOPAGE, MAPSIZ, SR3FLG, NSPLDV
100         .GLOBL  UDDRO, IDSFLG
101         .GLOBL  UPARO, KPDRO, UPDRO, KPAR7, BASMAP, PTWRD, PTBYT, LOKMEM
102         .GLOBL  GTBYT, MPPHY, RELOC, BRKPT, TSGEN, TSEEXEC, VSWPFL
103         .GLOBL  CW$GDH, CW$BTH, CW$LGS, CW$FB, CW$FGJ, MSGBAS, RPRVEC
104         .GLOBL  CW$USR, CW$XM, CONFIG, CW$50H, JMPO, DTLX, USRBAS, WINBAS
105         .GLOBL  DATIML, DATIMH, RMON, CONFG2, SG$ELG, SG$IOT, CSHBAS
106         .GLOBL  SG$PAR, SG$MTS, SG$MMU, SG$MTM, LTTPAR, LOKBAS, CSHVEC, LOKVEC
107         .GLOBL  SYSGEN, AUTHAN, AHEND, CLKRTI, TRP4, CW$PRO, TIOVEC
108         .GLOBL  TRP10, TRP20, TRP24, EMTEMT, TRP34, INIIMP, MHNSIZ
109         .GLOBL  TK1VAL, INRECV, OTRECV, INMXV, OTMXV, DHBF$SZ, MXTYPE
110         .GLOBL  ZCLR, MXRBUF, MXDTR, INTMX1, $PHONE, LCDTYP, TIOBAS
111         .GLOBL  LDHB1B, LDHB1P, LDHB2B, CLVERS, CXTSIZ, CXTWDS, CXTPDR
112         .GLOBL  CLORSZ, TSXSIT, JM$$SZ, VMXMON, MONFQH, CXTRMN, CXTBAS
113         .GLOBL  ILSW2, $NOIN, LSW3, MXLPR, CW$ESP, CLTOTL, RMNPDR, MA$SYS
114         .GLOBL  SFCB, SFCBND, SFCBFH, SFCBSZ, NSPLFL, NSPLBL, INTSTK, INTSND

```

```

115 . GLOBL NFRESB, PVSPBL, VMXWIN, DW$$SZ, LDVERS, CW$QBS
116 . GLOBL FC$LBN, VMLBLK, VMXSF, VMXSFC, FF$$SZ, FW$$SZ, SWPJOB, SWPPOS
117 . GLOBL TSR, RBR, RDINT, LSTMX, SS, CHAIN, JSWLOC, MU$TXT, SLTSIZ
118 . GLOBL NUMIOQ, FREIOQ, UMODE, FPTRAP, MXLNT, DI$LD, DI$CL, CLSTS
119 . GLOBL FREPGS, IOGSIZ, SYUNIT, UMSYTP, DI$TT, CXTBUF, SSEND
120 . GLOBL SYINDX, MONVEC, KMNBAS, SDANAM, VBUSTP, MINCTR
121 . GLOBL NUMRDB, RDB, RDBEND, RT$SKP, RT$TOP, NLINES, SHRRCB, SHRRCN
122 . GLOBL RT$BAS, UPMODE, SPLNB, CSHALC, NIOL, CHNSIZ, RC$$SZ, VNCR
123 . GLOBL UPAR6, UPDR6, RT$$SZ, VINABT, $DEAD, LSW6
124 . GLOBL SYTIMH, SYSDAT, TRP250, ODTTRP, TRP14, SYTIML
125 . GLOBL DS$ABT, CL$ORB, CL$ORE, CL$ORG, CL$ORP, CL$ORA
126 . GLOBL $TAB, $FORM, CO$TAB, CO$FF, CO$DEF, CL$EPS, CLEOFS
127 . GLOBL CL$OPT, CL$STA, CL$ORS, LSTLIN, VCSHNB, CL$EPP, CL$EPN
128 . GLOBL CCLSAV, SPLND, SDCB, SPLDEV, SPLANM, MIODBG
129 . GLOBL SDNAME, SDCHAN, SDCBSZ, SPLDVN, DTYPE
130 . GLOBL DS$NRD, DX$NMT, $BBIT, CO$BBT, UEXRTN, VUXIFL
131 . GLOBL SPLBLK, SPLCHN, MVSIZ, MEMPAR, UEXINT, DX$NRD
132 . GLOBL NMSNMB, SNMSHD, SB$$SZ, VPMSIZ, PMPAR, PMCELS
133 . GLOBL NUMDCD, MEM256, LOKCSH, DC$$SZ
134 . GLOBL JCXPGS, MXJMEM, VDFMEM, DFJMEM, TK5VAL, TK3SVL
135 . GLOBL VPAR6, IOTIMR, ERRLOG, VNFCSH, FC$$SZ
136 . GLOBL O. ADR, O. BLK, O. PAR, O. SIZ, VPAR5, KPAR5, DZOINT, DHOINT
137 . GLOBL OVRADD, $OVRH, SYSMAP, MAPSYS, VSLEDT, LCLUNT
138 . GLOBL UBUSMP, UMRADR, IOMAP, QBUS, UNIBUS, DX$NST
139 . GLOBL DVFLAG, DX$DMA, RT$NAM, DS$DIR, LDDEVX, DS$VSZ
140 . GLOBL INDSAV, INDDBL, INDTSV, INDDBS, DS$SFN
141 . GLOBL SYNAME, UCLNAM, RSFBLK, VPLAS, SEGCHN
142 . GLOBL MXJADR, $MEMSZ, PHMEM, SG$TSX, CDX$DH
143 . GLOBL CLHEAD, CLSIZE, CLDEVX, C. CSW, C. SBLK, C. DEVQ, C1DEVX
144 . GLOBL VU$CL, VUCLMC, UK$$SZ, US$$SZ, UC$$SZ, UCLBLK, UCLDAT
145 . GLOBL VLDSYS, VMXMSG, VMAXMC, MB$$SZ, MR$$SZ, CS$OPN, CS$ENT
146 . GLOBL DX$MAP, MIOFLG, MI$SBP, MI$LNK, MIOBHD, VMIOSZ
147 . GLOBL VMIOBF, MI$$SZ, MW$$SZ, MIONWB, MIOWHD, MW$LNK
148 . GLOBL CSHSIZ, CSHBFP, CA$BLK, CA$DVU, CA$WCT, VMXMRB
149 . GLOBL CA$UFL, CA$UBL, CA$HFL, CA$HBL, CA$HSH, NUMRDB
150 . GLOBL SRTSIZ, SMRSIZ, CCBHD, CC$$SZ, CDX$DZ, MF$LIN
151 . GLOBL CDX$DL, HF$TSB, MH$SCR, LMXLN, HF$LIN, HF$RIE, HF$TIE
152 . GLOBL MH$LPR, DM$CSR, MF$LE, DM$LSR, HF$MC, MF$CS, MF$CM
153 . GLOBL CDX$VH, VH$CSR, VH$LPR, MH$PBR, VF$TIE, VF$RIE, VF$MR
154 . GLOBL VF$LIN, VF$SC, VF$RE, VH$LCR, VHOINT, TTINCP
155 . GLOBL $HARD, LOUTIR, LINIR, NEDCHR, CLOTIR, CLINCP, FSTIOL, LSTHL
156 . GLOBL SYSVER, SYSUPD, DI$DU, DI$XL, DI$MU, CL$LIX
157 . GLOBL CL$LEN, DI$PI, GENTOP
158 . GLOBL LSW5, DX$NCA, KPAR6, CLKVEC

```

```

; Macros to enable and disable interrupts
;

```

```

. MACRO DISABL ; Disable interrupts
BIS #340, @#PSW
. ENDM DISABL

. MACRO ENABL
BIC #340, @#PSW
. ENDM ENABL

```

```

; Offsets in block 0 of ODT REL file.

```

159
160
161
162
163
164
165
166
167
168
169
170
171

```

172          ;
173          000040 STA      =      40          ;PROGRAM START ADDRESS
174          000042 STK      =      42          ;INITIAL STACK POINTER
175          000052 RSZ      =      52          ;ROOT SIZE
176          000056 OSZ      =      56          ;OVERLAY SIZE
177          000060 RID      =      60          ;REL FILE ID
178          000062 RBD      =      62          ;DISPLACEMENT TO 1ST REL BLOCK
179          001000 ODTBAS  =      1000         ;BASE ADDRESS ODT WAS LINKED FOR
180          ;
181          ; Data areas
182          ;
183          000000 AREA:    .BLKW  8.
184          000020 000000 000000 000000 NFSBLK: .WORD  0,0,0,0,0,0 ;EXTENDED TO 6 WORDS FOR .CSTAT
185          000026 000000 000000 000000
186          000034 000000 ODTFLG: .WORD  0
187          000036 000000 ODTTOP: .WORD  0
188          000040 000000 CCAFLG: .WORD  0
189          000042 000000 CLK100: .WORD  0
190          000044 000000 RTTRP4: .WORD  0
191          000046 000000 RTMNVC: .WORD  0
192          000062 075250 100020 000000 SAVBLK: .BLKW  5
193          000070 073376 TSXSAV: .RAD50 /SY TSX SAV/
194          000072 073376 100003 051646 KMNNAM: .RAD50 /SY TSKMONSAV/
195          000100 073376
196          000102 075250 011504 000000 CCLNAM: .RAD50 /SY CCL SAV/
197          000110 073376
198          000112 000000 000000 000000 DSTBLK: .WORD  0,0,0,0
199          000120 000000
200          000122 000000 XMVBAS: .WORD  0
201          000124 000000 NMXHAN: .WORD  0
202          000126 000000 HMAP:   .WORD  0
203          000130 000000 FETDEV: .WORD  0
204          000132 000000 TOPMEM: .WORD  0
205          000134 000000 FMEMHI: .WORD  0 ;64-byte block # below high alloc memory
206          000136 000000 FMEMLO: .WORD  0 ;64-byte block # above top of low alloc memory
207          000140 000000 OVLBAS: .WORD  0 ;Start loading overlays over TSINIT from here
208          000142 000000 FILBLK: .WORD  0
209          000144 000000 CURDEV: .WORD  0
210          000146 000000 CURNAM: .WORD  0
211          000150 000000 PROBUF: .WORD  0
212          000152 030074' WRKBUF: .WORD  INITOP
213          000154 004000' WRKSIZ: .WORD  2048.
214          000156 000322' RTVPTR: .WORD  RTVEND ;Initially no limit on system version number
215          ; The following RAD50 definitions were converted from word cells to
216          ; definitions to conserve space at V6.40. All subsequent references to
217          ; one of these cells are converted from:
218          ; relative (cmp R50xxx,r1) to immediate (cmp #R50xxx,r1).
219          052077 R50MSG  = ^RMSG
220          110466 R50WIN  = ^RWIN
221          046543 R50LOK  = ^RLOK
222          103112 R50USR  = ^RUSR
223          012700 R50CSH  = ^RCSH
224          077167 R50TID  = ^RTID
225          100040 R50TT   = ^RTT ;"TT "
226          075250 R50SY   = ^RSY ;"SY "
227          045640 R50LD   = ^RLD ;"LD "

```

```

224          062550          R50PI   = ^RPI           ;"PI "
225          012240          R50CL   = ^RCL           ;"CL "
226          012276          R50CLO  = ^RCL0          ;
227          012305          R50CL7  = ^RCL7          ;
228          013630          R50C1   = ^RC1           ;"C1 "
229          013666          R50C10  = ^RC10          ;
230          013675          R50C17  = ^RC17          ;
231          105610          R50VM   = ^RVM           ;"VM "
232          046770          R50LS   = ^RLS           ;"LS "
233          057164          R50ODT  = ^RODT          ;
234 000160  100040  015270  075250  SKPDEV: .RAD50 /TT DK SY CL C1 PI /
          000166  012240  013630  062550
          000174  000000
235 000176  000          110          GTLIN: .BYTE 0,110
236 000200  075250  114730  000000  HANNAM: .RAD50 /SY XXX TSX/
          000206  100020
237 000210  075250  075273  057164  ODTBLK: .RAD50 /SY SYSODTREL/
          000216  070524
238 000220  075250  035164  000000  INDNAM: .RAD50 /SY IND SAV/
          000226  073376
239 000230  000000          RLBF: .WORD 0
240 000232  000000          RLBFND: .WORD 0
241 000234  000000          ODTSTA: .WORD 0
242 000236  000000          MEMLIM: .WORD 0
243 000240  000000          HGENFL: .WORD 0
244          ;
245          ; Initialization configuration word
246          ;
247 000242  000000          ICONFG: .WORD 0 ;Initialization configuration word
248          ;
249          ; Flag bits in ICONFIG
250          ;
251          000001          EXTLSI = 1 ;Q-bus system with more than 256Kb
252          ;
253          ; Simulated shared run-time control block for PI handler
254          ;
255 000244  075250  062550  000000  PISRT: .RAD50 /SY PI TSX/
          000252  100020
256 000254  000000  000000  000000          .WORD 0,0,0
257          ;
258          ; Byte data cells
259          ;
260 000262  000          PPTERM: .BYTE 0 ;1 if printer port is T/S terminal
261          .EVEN

```

```

1
2 ; -----
3 ; The following table is used to identify the supporting RT-11 monitor
4 ; version number for those features which depend on it.
5 ;
6 ; The DL, XL and MU handlers require a minimum supporting RT-11 version.
7 ; The CL version number emulates the supporting RT-11 XL's $$$VER.
8 ; The LD translation table format changed at RT-11 V5.4.
9 ;
10 ; The format of the table is:
11 ;
12 ; RT$VER = 0 ; Major system version number
13 ; RT$UPD = 1 ; Minor system version number (update number)
14 ; CL$VER = 2 ; Emulated XL version number
15 ; RTV$SZ = 3 ; Size of a version table entry
16
17 ; RTVER: ; V4.0 is the earliest supported version of RT-11
18 ; RT40: .BYTE 4,0,-1 ; 4.0 did not support XL
19 ; RT50: .BYTE 5,0,-1 ; 5.0 did not support XL
20 ; RT51: .BYTE 5,1,16.
21 ; RT51X: .BYTE 5,35,16. ; Another flavor of 5.1
22 ; RT51B: .BYTE 5,6,16.
23 ; RT51C: .BYTE 5,44,16.
24 ; RT52: .BYTE 5,2,17.
25 ; RT53: .BYTE 5,3,17.
26 ; RTVDEF: ; Default emulation version
27 ; RT54: .BYTE 5,4,18.
28 ; RT55: .BYTE 5,5,18.
29 ; RTVEND:
30 ; .EVEN

```

```

1      ; -----
2      ; The following tables are used to determine the minimum RT-11 monitor
3      ; and update versions required for particular devices.
4      ; There are three arguments for each handler definition:
5      ;   Arg 1 = Handler id code.
6      ;   Arg 2 = Ptr to minimum acceptable RT-11 version and update entry.
7      ;
8      ;       .MACRO  HANVER  DEVID, MNVPTR
9      ;       .BYTE  DEVID          ;ID code for device type
10     ;       .BYTE  0              ;Unused filler entry
11     ;       .WORD  MNVPTR         ;Minimum RT-11 version and update label
12     ;       .ENDM  HANVER
13     ;
14     ; Define offsets into handler version table
15     ;
16     000000 HV$ID = 0              ;Handler identification code
17     000001 HV$DMY = 1              ;Unused entry
18     000002 HV$VER = 2              ;Minimum RT-11 version
19     000004 HV$$SZ = 4              ;Size of handler version table entry
20     ;
21     ; Define minimum versions for various handlers
22     ;
23     000322 HVTBL:
24     ;       HANVER  DI$DU, RT50    ;DU - (5.0)
25     ;       HANVER  DI$XL, RT51B   ;XL - (5.1B)
26     ;       HANVER  DI$MU, RT54    ;MU - (5.4)
27     HVEND:

```

```

1          ; -----
2          ; The following table defines default control flags for certain devices.
3          ;
4          000000 DV$NAM = 0 ;Rad50 name of device
5          000002 DV$FLG = 2 ;Flags for device
6          000004 DV$$SZ = 4 ;Size of a table entry
7          ;
8          . MACRO DEFFLG DEV, FLAGS
9          . RAD50 /'DEV/' ; DV$NAM
10         . WORD FLAGS ; DV$FLG
11         . ENDM DEFFLG
12         ;
13         DVFLBS:
14         DEFFLG <CR>, <DX$MPH>
15         DEFFLG <CT>, <DX$MPH>
16         DEFFLG <DB>, <DX$DMA!DX$MPH>
17         DEFFLG <DD>, <DX$NHM>
18         DEFFLG <DL>, <DX$DMA!DX$MPH!DX$IBH>
19         DEFFLG <DM>, <DX$DMA!DX$NHM>
20         DEFFLG <DP>, <DX$DMA>
21         DEFFLG <DS>, <DX$DMA>
22         DEFFLG <DT>, <DX$DMA>
23         DEFFLG <DU>, <DX$DMA!DX$NHM!DX$NST>
24         DEFFLG <DW>, <DX$MPH>
25         DEFFLG <DX>, <DX$MPH>
26         DEFFLG <DY>, <DX$DMA!DX$NHM>
27         DEFFLG <DZ>, <DX$MPH>
28         DEFFLG <FW>, <DX$DMA>
29         DEFFLG <LP>, <DX$MPH>
30         DEFFLG <LS>, <DX$MPH>
31         DEFFLG <MM>, <DX$DMA!DX$MPH!DX$IBH>
32         DEFFLG <MS>, <DX$DMA!DX$MPH!DX$IBH>
33         DEFFLG <MT>, <DX$DMA!DX$MPH!DX$IBH>
34         DEFFLG <MU>, <DX$DMA!DX$NHM!DX$IBH!DX$NST>
35         DEFFLG <NL>, <DX$MPH>
36         DEFFLG <PC>, <DX$MPH>
37         DEFFLG <RF>, <DX$DMA>
38         DEFFLG <RK>, <DX$DMA!DX$MPH>
39         DEFFLG <VM>, <DX$EBA!DX$NCA!DX$NHM>
40         DEFFLG <XC>, <DX$MPH>
41         DEFFLG <XL>, <DX$MPH>
42         DVFLND:

```

```

1          ; -----
2          ; The following data structures are used to hold information about
3          ; TSX-Plus overlays as they are being initialized.
4          ;
5          ; Offsets in structure for each overlay
6          ;
7          000000 OS$SIZ =      0          ; Total space needed for overlay
8          000002 OS$FLG =      2          ; 0==>Load into XM space, 1==>over TSINIT
9          000004 OS$OVL =      4          ; Pointer to overlay table entry
10         000006 OS$$SZ =      6          ; Size of each overlay entry
11         ;
12         000031 MAXOVL =     25.         ; Maximum number of system overlays
13         ;
14 000516 OSTABL: .BLKB OS$$SZ*MAXOVL ; Reserve room for table
15 000744 OSEND: ; -Define end of table
16 000744 000516' OSLAST: .WORD OSTABL ; Pointer past last used entry in table
17         ;
18         ; Table of system overlays that must be loaded over TSINIT
19         ;
20 000746 012700 LOWOVL: .RAD50 /CSH/ ; TSCASH
21 000750 077167 .RAD50 /TIO/ ; TSTIO
22 000752 046543 .RAD50 /LOK/ ; TSLOCK
23 000754 LOWEND: ; End of table

```

```

1          ; -----
2          ;   Text messages
3          ;
4          ; .NLIST BEX
5 000754   077   124   123 TSXHD:  .ASCII  /?TSX-F-/<200>
6 000764   111   156   166 BADLIN: .ASCII  'Invalid CSR for T/S line: '<200>
7 001017   111   156   166 BDVMSG: .ASCII  'Invalid vector for T/S line: '<200>
8 001055   114   151   156 BDLMSG: .ASCII  /Line # = /<200>
9 001067   000
10 001070  124   123   130 REQMIS: .ASCII  /TSX generation did not include device /<200>
11 001137  103   141   156 BADOPN: .ASCIZ  /Cannot open program swap file/
12 001175  103   141   156 RSFERR: .ASCIZ  /Cannot open PLAS region swap file/
13 001237  116   165   155 CONSPC: .ASCII  /Number of contiguous blocks needed = /<200>
14 001305  103   141   156 BDSPOP: .ASCII  /Cannot open spooled device: /<200>
15 001342  111   156   163 BQSF:  .ASCIZ  /Insufficient disk space for spool file/
16 001411  103   141   156 NOKMON: .ASCIZ  /Cannot find "SY:TSKMON.SAV" file/
17 001452  103   141   156 NOCCL: .ASCIZ  /Cannot find "SY:CCL.SAV" file/
18 001510  103   141   156 CFHMSG: .ASCII  /Cannot find device handler file: /<200>
19 001552  105   162   162 ERHMSG: .ASCII  /Error reading device handler file: /<200>
20 001616  111   156   166 ERHNDV: .ASCII  /Invalid RT-11 version for device handler: /<200>
21 001670  111   156   166 NOCSRR: .ASCII  /Invalid CSR for device: /<200>
22 001721  105   162   162 ERHINS: .ASCII  /Error executing installation code for device: /<200>
23 002000  124   123   130 TSXRUN: .ASCIZ  /TSX is already running/
24 002027  110   141   156 HSGER: .ASCII  /Handler not generated with XM support: /<200>
25 002077  103   157   155 NOCLOK: .ASCIZ  /Computer line time clock (50 or 60 Hz) is not working/
26 002165  116   157   040 NXMMSG: .ASCIZ  /No memory management hardware/
27 002223  116   157   040 NEXMSG: .ASCIZ  /No extended memory management hardware/
28 002272  103   141   156 NOODT: .ASCIZ  /Cannot locate "SY:SYSODT.REL" file/
29 002335  115   141   160 HN2BIG: .ASCII  /Mapped handler is larger than BKB: /<200>
30 002401  105   162   162 ODTRDM: .ASCIZ  /Error on read of SYSODT rel file/
31 002442  110   141   156 NOSYDV: .ASCIZ  /Handler for SY device was not loaded/
32 002507  107   145   156 TOOBIG: .ASCIZ  /Generated TSX system is too large/
33 002551  122   145   144 REDUCE: .ASCII  /Reduce size of TSGEN by /<200>
34 002602  056   040   142 BYTES:  .ASCIZ  /. bytes/
35 002612  111   156   163 PHSQVF: .ASCIZ  /Insufficient total physical memory for generated system/
36 002702  103   141   156 COSRT: .ASCII  /Cannot open shared run-time file: /<200>
37 002745  103   141   156 SVERR:  .ASCIZ  /Cannot locate "SY:TSX.SAV"/
38 003000  111   156   163 TSXSIZ: .ASCIZ  /Insufficient memory to load all mapped system regions/
39 003066  105   162   162 RDERR:  .ASCIZ  /Error reading "SY:TSX.SAV"/
40 003121  111   156   163 SRTOVF: .ASCIZ  /Insufficient memory to load all shared run-time systems/
41 003211  111   156   163 CSHOVF: .ASCIZ  /Insufficient memory space for data cache/
42 003262  103   141   156 INDOPN: .ASCIZ  /Cannot open TSXIND file/
43 003312  103   141   156 UCLOPN: .ASCIZ  /Cannot open TSXUCL data file/
44 003347  040   101   102 R50CHR: .ASCII  / ABCDEFGHIJKLMNOPQRSTUVWXYZ$. *0123456789/
45          ; .EVEN
46          ; .LIST BEX

```

* * * TSX Initialization * * *

```

1          .SBTTL * * * TSX Initialization * * *
2          .SBTTL * * * Initialization taking over control from RT-11 * * *
3          ;-----
4          ; The initialization code from this point onward takes over
5          ; control from RT-11.
6          ; This code is placed at the front of TSINIT so that non-initialized
7          ; data structures can be allocated over it.
8          ;
9 003420    TAKOVR:
10         ;
11         ; Read in system overlays that go over TSINIT
12         ;
13 003420    004737    004260'    CALL    INIOVL        ;Read overlays over TSINIT
14         ;
15         ; Set pointer to monitor offset vector
16         ;
17 003424    012737    000000G 000000G    MOV     #MONVEC,@#RMON ;SET POINTER TO MONVEC TABLE
18         ;
19         ; Initialize last word in interrupt stack area so we won't report a
20         ; stack overflow if an interrupt occurs.
21         ; Set STKLVL to 0 to cause INTEN not to switch to interrupt
22         ; stack during initialization.
23         ;
24 003432    012777    123456 000000G    MOV     #123456,@INTSND ;Say stack has not overflowed
25 003440    105037    000000G    CLR    STKLVL        ;Say we are already on interrupt stack
26         ;
27         ; If we are running on a Professional, disable its interrupts
28         ;
29         .IF     NE,PROASM
30         TSTB   PROFLG        ;Are we running on a PRO?
31         BEQ    7$            ;Br if not on a PRO
32         CALL   PRONOP        ;Disable its interrupts
33         BR     5$            ;Ignore unexpected interrupts on PRO
34         .ENDC   ;NE,PROASM
35         ;
36         ; Set up vectors to catch unexpected interrupts
37         ; Note: We encode the interrupt vector address in the PS --
38         ; the low-order two bits of the address are dropped (they are
39         ; always zero) and the remainder of the address is encoded in the
40         ; PS fields priority (high-order 3 bits) and n-z-v-c (low-order 4 bits).
41         ;
42 003444    012702    000000G    7$:    MOV     #UEXINT,R2        ;SEND UNEXPECTED INTERRUPTS TO THIS ROUTINE
43 003450    012700    000044        MOV     #44,R0            ;120 ENCODED IN PS FIELDS
44 003454    105737    000000G    TSTB   VUXIFL          ;ARE WE TO IGNORE UNEXPECTED INTERRUPTS?
45 003460    001004        BNE     10$            ;BR IF NOT
46 003462    012702    000000G    5$:    MOV     #UEXRTN,R2       ;ROUTINE TO GO TO TO IGNORE INTERRUPT
47 003466    012700    000340        MOV     #340,R0          ;SET PRIO=7 IN PS
48 003472    012701    000120        10$:   MOV     #120,R1          ;INIT ALL VECTORS STARTING AT 120
49 003476    010221        1$:    MOV     R2,(R1)+        ;SET PC FOR INTERRUPT
50 003500    010021        MOV     R0,(R1)+        ;SET PS FOR INTERRUPT (ENCODED ADDRESS VALUE)
51 003502    105737    000000G    6$:    TSTB   VUXIFL          ;ARE WE TO IGNORE UNEXPECTED INTS?
52 003506    001411        BEQ     2$            ;BR IF YES
53 003510    105737    000000G    TSTB   PROFLG          ;IS THIS A PRO?
54 003514    001006        BNE     2$            ;BR IF YES
55 003516    005200        INC     R0            ;ADVANCE ENCODED ADDRESS
56 003520    032700    000020        BIT     #20,R0          ;DID WE CARRY INTO "T"-FIELD?
57 003524    001402        BEQ     2$            ;BR IF NOT

```

* * * Initialization taking over control from RT-11 * * *

```

58 003526 062700 000020          ADD    #20,R0          ;FORCE CARRY OUT OF T-FIELD AND INTO PRID FIELD
59 003532 020127 000420          2$:   CMP    R1,#420      ;DONE ALL INTERRUPT VECTORS OF INTEREST?
60 003536 103757                   BLD    1$             ;BR IF NOT
61 003540 010237 000060          MOV    R2,@#60       ;CATCH CONSOLE TERMINAL VECTOR TOO
62 003544 012737 000014 000062   MOV    #14,@#62      ;ENCODED 60
63 003552 010237 000064          MOV    R2,@#64
64 003556 012737 000015 000066   MOV    #15,@#66      ;ENCODED 64
65                                     ;
66                                     ; Direct clock interrupt to a dummy RTI instruction to avoid it causing
67                                     ; trouble during the initialization process when things aren't set up
68                                     ; and ready to go.
69                                     ;
70 003564 012700 000340          11$:  MOV    #340,R0      ;PRIORITY 7 PS
71 003570 012737 000000G 000000G  MOV    #CLKRTI,@#CLKVEC;Send clock interrupt to RTI instruct for now
72 003576 010037 000002G          MOV    R0,@#CLKVEC+2
73                                     ;
74                                     ; Take over traps, EMT, BPT, etc.
75                                     ;
76 003602 005001                   CLR    R1             ;Start at location 0
77 003604 012721 000137          MOV    #137,(R1)+    ;[JMP @#JMPO] ==> 0
78 003610 012721 000000G          MOV    #JMPO,(R1)+   ;CATCH JUMPS TO LOCATION 0
79 003614 012721 000000G          MOV    #TRP4,(R1)+   ;TRAP 4
80 003620 005021                   CLR    (R1)+
81 003622 012721 000000G          MOV    #TRP10,(R1)+  ;TRAP 10
82 003626 005021                   CLR    (R1)+
83 003630 012721 000000G          MOV    #TRP14,(R1)+  ;TRAP 14 (BREAKPOINTS)
84 003634 010021                   MOV    R0,(R1)+
85 003636 012721 000000G          MOV    #TRP20,(R1)+  ;IOT TRAP
86 003642 005021                   CLR    (R1)+
87 003644 012721 000000G          MOV    #TRP24,(R1)+  ;POWER FAIL
88 003650 010021                   MOV    R0,(R1)+
89 003652 012721 000000G          MOV    #EMTENT,(R1)+ ;EMT
90 003656 005021                   CLR    (R1)+
91 003660 012721 000000G          MOV    #TRP34,(R1)+  ;TRAP
92 003664 005021                   CLR    (R1)+
93 003666 012737 000000G 000114   MOV    #MEMPAR,@#114 ;MEMORY PARITY TRAP
94 003674 010037 000116          MOV    R0,@#116
95 003700 012737 000000G 000244   MOV    #FPTRAP,@#244 ;TRAP 244 -- FLOATING POINT TRAP
96 003706 010037 000246          MOV    R0,@#246      ;Enter FPU trap at priority 7
97 003712 012737 000000G 000250   MOV    #TRP250,@#250 ;TRAP 250 -- MEMORY MANAGEMENT TRAP
98 003720 005037 000252          CLR    @#252
99                                     ;
100                                     ; Initialize the system mapped region.
101                                     ;
102 003724 010546                   MOV    R5,-(SP)      ;SAVE THE CURRENT CONTENTS OF R5
103 003726 012705 000006          MOV    #6,R5         ;INITIALIZE TO THE FIRST REGION
104 003732 004737 000000G          CALL   MAPSYS        ;CALL THE SYSTEM MAPPING ROUTINE
105 003736 012605                   MOV    (SP)+,R5      ;RESTORE THE PREVIOUS CONTENTS OF R5
106                                     ;
107                                     ; Set up Unibus mapping if needed
108                                     ;
109 003740 004737 004616'          CALL   SETUMP        ;SET UP UNIBUS MAPPING
110                                     ;
111                                     ; Initialize time-sharing lines.
112                                     ;
113 003744 004737 005234'          CALL   LININI        ;INIT LINES & SET UP VECTORS
114                                     ;

```

* * * Initialization taking over control from RT-11 * * *

```

115          ; Enable memory management
116          ; (The kernel-mode mapping registers are already set up)
117          ;
118 003750 052737 000000G 000000G      BIS      #MMENBL, @SR0MMR ; Turn on memory management
119 003756 105737 000000G              TSTB     SR3FLG          ; Does machine have memory management reg 3?
120 003762 001415                      BEQ      4$              ; Br if register does not exist (no ext. mem.)
121 003764 023727 000000G 010000      CMP      PHYMEM, #4096. ; Does machine have at least 256Kb phys memory?
122 003772 103411                      BLD      4$              ; Br if not
123 003774 052737 000000G 000000G      BIS      #EMMAP, @SR3MMR ; Set extended memory on
124 004002 105737 000000G              TSTB     MEM256          ; Will TSX-Plus use at least 256Kb?
125 004006 001403                      BEQ      4$              ; Br if not
126 004010 052737 000000G 000000G      BIS      #IOMAP, @SR3MMR ; Turn on 22-bit memory management for I/O
127          ;
128          ; Initialize the memory allocation table
129          ;
130 004016 013737 000000G 000000G 4$:  MOV      MAPPAR, @KPAR5 ; Map to memory allocation table
131 004024 013702 000000G              MOV      LOMAP, R2      ; Point to 1st user-page entry
132 004030 105022 8$:  CLR      (R2)+          ; Say page is free
133 004032 020237 000000G              CMP      R2, HIMAP      ; Done all user pages?
134 004036 103774                      BLD      8$              ; Loop if not
135 004040 112712 000000G              MOV      #MA$SYS, (R2) ; Set flag marking start of system pages
136          ;
137          ; Set up I/O device interrupt vectors.
138          ;
139 004044 004737 004706'              CALL     DEVVEC          ; SET UP DEVICE INTERRUPT VECTORS
140          ;
141          ; If we are running on a Professional, initialize the PI handler
142          ;
143          .IF      NE, PROASM
144          TSTB     PROFLG          ; Are we running on a Professional?
145          BEQ      3$              ; Br if not
146          CALL     PROHAN          ; Initialize the PI handler
147          CALL     PIDVEN          ; Make device table entry for PI
148          .ENDC      ; NE, PROASM
149          ;
150          ; Initialize interrupt stack area
151          ;
152 004050 013702 000000G 3$:  MOV      INTSND, R2      ; Point to base of stack area
153 004054 012700 123456              MOV      #123456, R0    ; Get initialization value
154 004060 010022 12$:  MOV      R0, (R2)+          ; Initialize the interrupt stack area
155 004062 020237 000000G              CMP      R2, INTSTK     ; Finished?
156 004066 103774                      BLD      12$            ; Loop if not
157 004070 112737 177777 000000G      MOV      #-1, STKLVL    ; Say we are not running on interrupt stack
158          ;
159          ; Enter TSEXEC to complete initialization
160          ;
161 004076 000137 000000G              JMP      INIJMP          ; ENTER INITIALIZATION ROUTINE IN TSEXEC
162          ;
163          ; Abort the initialization
164          ;
165 004102 013737 000042' 000000G INISTP: MOV      CLK100, @CLKVEC ; Restore RT-11 clock vector
166 004110 013737 000044' 000000G      MOV      RTTRP4, @#4   ; Restore trap 4 vector
167 004116 013737 000046' 000000G      MOV      RTMNV, @#RMON ; Restore RT-11 monitor pointer
168 004124 9$:  .EXIT              ; RETURN TO RT-11

```

LODINI -- Load a segment over TSINIT

```

1          .SBTTL  LODINI -- Load a segment over TSINIT
2          ;-----
3          ; LODINI is called to read an overlay segment over TSINIT.
4          ;
5          ; Inputs:
6          ; R2 = Pointer to OSTABL entry for segment to be loaded.
7          ; R5 = 64-byte block # where segment is to be loaded.
8          ;
9 004126 010146 LODINI: MOV      R1, -(SP)
10 004130 010346      MOV      R3, -(SP)
11 004132 010446      MOV      R4, -(SP)
12          ;
13          ; Get pointer to linker-built overlay entry
14          ;
15 004134 016201 000004      MOV      OS$OVL(R2), R1 ;Get pointer to linker-built table
16          ;
17          ; Determine how much code to read from the segment
18          ;
19 004140 016204 000000      MOV      OS$SIZ(R2), R4 ;Get # 64-byte blks allocated for segment
20 004144 072427 000005      ASH      #5, R4 ;Convert to # words
21 004150 020461 000000G    CMP      R4, 0, SIZ(R1) ;Compare with original segment code size
22 004154 101402            BLOS    1$ ;Br if segment was truncated by init
23 004156 016104 000000G    MOV      0, SIZ(R1), R4 ;Get code size
24          ;
25          ; Read the segment into memory
26          ;
27 004162 010503 1$:      MOV      R5, R3 ;Get 64-byte block #
28 004164 072327 000006      ASH      #6, R3 ;Convert to byte address
29 004170            .READW  #AREA, #17, R3, R4, 0, BLK(R1)
30 004224 103406            BCS      10$ ;Br if error on read
31          ;
32          ; Store the physical address of the segment into the overlay descriptor
33          ;
34 004226 010561 000000G    MOV      R5, 0, PAR(R1) ;Remember physical address of segment
35          ;
36          ; Finished
37          ;
38 004232 012604            MOV      (SP)+, R4
39 004234 012603            MOV      (SP)+, R3
40 004236 012601            MOV      (SP)+, R1
41 004240 000207            RETURN
42          ;
43          ; Error on read
44          ;
45 004242 10$:      .PRINT  #TSXHD
46 004250            .PRINT  #RDERR
47 004256            .EXIT

```

INIOVL -- Load system overlays over TSINIT

```

1          .SBTTL  INIOVL -- Load system overlays over TSINIT
2          ;-----
3          ; INIOVL is called to load into memory those system overlays that
4          ; are to be placed over the TSINIT code.
5          ;
6          ; Inputs:
7          ; Overlay segment information is in OSTABL.
8          ;
9 004260 010246 INIOVL: MOV      R2,-(SP)
10 004262 010546      MOV      R5,-(SP)
11          ;
12          ; Initialize pointer to start of memory area for overlays
13          ;
14 004264 013705 000140'      MOV      OVLBAS,R5      ;Start of area for overlays
15 004270 072527 177772      ASH      #-6,R5      ;Convert to 64-byte #
16 004274 042705 176000      BIC      #176000,R5    ;Clear possible propagated sign bits
17          ;
18          ; Begin loop to load each overlay that goes over TSINIT
19          ;
20 004300 012702 000516'      MOV      #OSTABL,R2    ;Point to 1st overlay segment entry
21 004304 005762 000002      1$:  TST      OS#FLG(R2)  ;Does this segment go over TSINIT?
22 004310 001406              BEQ      2$              ;Br if not
23 004312 004737 004470'      CALL     KEYSEG        ;Remember base of some segments
24 004316 004737 004126'      CALL     LODINI        ;Load the segment
25 004322 066205 000000      ADD      OS#SIZ(R2),R5 ;Advance memory pointer
26 004326 062702 000006      2$:  ADD      #OS##SZ,R2   ;Point to entry for next segment
27 004332 020237 000744'      CMP      R2,OSLAST    ;Finished all segments?
28 004336 103762              BLO      1$              ;Loop if not
29          ;
30          ; Initialize entry point vector for TSTIOX segment
31          ;
32 004340 013702 000000G      MOV      TIOBAS,R2    ;Get addr of base of TSTIOX
33 004344 072227 000006      ASH      #6,R2        ;Convert to byte address
34 004350 012705 000000G      MOV      #TIOVEC,R5   ;Point to entry point vector
35 004354 004737 004432'      CALL     ENTVEC        ;Set up entry point vector
36          ;
37          ; Initialize entry point vector for TSCASH segment
38          ;
39 004360 013702 000000G      MOV      CSHBAS,R2    ;Get addr of base of TSCASH
40 004364 001406              BEQ      3$              ;Br if TSCASH not loaded
41 004366 072227 000006      ASH      #6,R2        ;Convert to byte address
42 004372 012705 000000G      MOV      #CSHVEC,R5   ;Point to entry point vector
43 004376 004737 004432'      CALL     ENTVEC        ;Set up entry point vector
44          ;
45          ; Initialize entry point vector for TSLOCK segment
46          ;
47 004402 013702 000000G      3$:  MOV      LOKBAS,R2    ;Get addr of base of TSLOCK
48 004406 001406              BEQ      9$              ;Br if TSLOCK not loaded
49 004410 072227 000006      ASH      #6,R2        ;Convert to byte address
50 004414 012705 000000G      MOV      #LOKVEC,R5   ;Point to entry point vector
51 004420 004737 004432'      CALL     ENTVEC        ;Set up entry point vector
52          ;
53          ; Finished
54          ;
55 004424 012605      9$:  MOV      (SP)+,R5
56 004426 012602      MOV      (SP)+,R2
57 004430 000207      RETURN

```

ENTVEC -- Set up entry point vector for overlay

```

1                                     .SBTTL  ENTVEC -- Set up entry point vector for overlay
2                                     ;-----
3                                     ; ENTVEC is called to set up addresses in an entry point vector for
4                                     ; overlay segments such as TSCASH that are loaded at addresses different
5                                     ; from where they are linked.
6                                     ;
7                                     ; Inputs:
8                                     ; R2 = Address of base of segment.
9                                     ; R5 = Pointer to vector that is to be initialized (word with -1 terminates)
10                                    ;
11 004432 010246  ENTVEC: MOV      R2, -(SP)
12 004434 010446      MOV      R4, -(SP)
13 004436 010546      MOV      R5, -(SP)
14 004440 010204      MOV      R2, R4          ;Get addr of base of segment
15 004442 062704 000004  ADD      #4, R4          ;Point to start of vector in segment
16 004446 005722      TST      (R2)+          ;Get value to use to relocate offsets
17 004450 012415 1$:  MOV      (R4)+, (R5)      ;Get offset to entry point within segment
18 004452 060225      ADD      R2, (R5)+      ;Convert to absolute address
19 004454 005715      TST      (R5)          ;Any more words to initialize?
20 004456 001774      BEQ      1$          ;Br if yes
21                                    ;
22                                    ; Finished
23                                    ;
24 004460 012605      MOV      (SP)+, R5
25 004462 012604      MOV      (SP)+, R4
26 004464 012602      MOV      (SP)+, R2
27 004466 000207      RETURN

```

```

1                                     .SBTTL  KEYSEG -- Remember memory position of system overlays
2                                     ;-----
3                                     ; KEYSEG is called to remember the physical memory position of some
4                                     ; key system overlay segments.
5                                     ;
6                                     ; Inputs:
7                                     ; R2 = Pointer to segment entry in OSTABL overlay table.
8                                     ; R5 = Base 64-byte block physical memory for segment.
9                                     ;
10 004470 010446 KEYSEG: MOV      R4, -(SP)
11                                     ;
12                                     ; Get the name of the segment out of the linker-built segment block
13                                     ;
14 004472 016200 000004          MOV      OS$OVL(R2),R0  ;Point to linker-built entry
15 004476 016004 000000G        MOV      0.ADR(R0),R4  ;Get the name of the segment
16                                     ;
17                                     ; See if this is a segment whose address we want to remember
18                                     ;
19 004502 020427 052077          CMP      R4,#R5MSG      ;Is this the TSMSG segment?
20 004506 001003                BNE     1$            ;Br if not
21 004510 010537 000000G        MOV      R5,MSGBAS    ;Remember base of TSMSG segment
22 004514 000436                BR      8$
23 004516 020427 110466          1$:  CMP      R4,#R5WIN    ;Is this the TSWIN segment?
24 004522 001003                BNE     3$            ;Br if not
25 004524 010537 000000G        MOV      R5,WINBAS   ;Remember base of TSWIN segment
26 004530 000430                BR      8$
27 004532 020427 103112          3$:  CMP      R4,#R5USR    ;Is this the TSUSR segment?
28 004536 001003                BNE     4$            ;Br if not
29 004540 010537 000000G        MOV      R5,USBAS    ;Remember base of TSUSR segment
30 004544 000422                BR      8$
31 004546 020427 046543          4$:  CMP      R4,#R5LOK    ;Is this the TSLOCK segment?
32 004552 001003                BNE     5$            ;Br if not
33 004554 010537 000000G        MOV      R5,LOKBAS   ;Remember base of TSLOCK segment
34 004560 000414                BR      8$
35 004562 020427 012700          5$:  CMP      R4,#R5CSH    ;Is this the TSCASH segment?
36 004566 001003                BNE     6$            ;Br if not
37 004570 010537 000000G        MOV      R5,CSHBAS   ;Remember base of TSCASH segment
38 004574 000406                BR      8$
39 004576 020427 077167          6$:  CMP      R4,#R5OTIO   ;Is this the TSTIOX segment?
40 004602 001003                BNE     8$            ;Br if not
41 004604 010537 000000G        MOV      R5,TIOBAS   ;Remember base of module
42 004610 000400                BR      8$
43                                     ;
44                                     ; Finished
45                                     ;
46 004612 012604          8$:  MOV      (SP)+, R4
47 004614 000207          RETURN

```

KEYSEG -- Remember memory position of system overlays

```

1          .IF      NE,<PROASM-1>    ; Assemble for PDP-11
2          .SBTTL   SETUMP -- Set up Unibus mapping if needed
3          ;-----
4          ; SETUMP is called to set up the Unibus map registers for 11/44 and
5          ; 11/70 systems which more than 256Kb of memory.
6          ; If Unibus mapping is needed, the Unibus map registers # 0-4 are
7          ; initialized for a 1-to-1 mapping with the low 40Kb of memory
8          ; so that I/O to system buffers in the low memory area can be done without
9          ; having to do Unibus mapping.
10         ;
11         ; Outputs:
12         ;   UBUSMP:  1==>Do Unibus mapping;  0==>Don't do Unibus mapping.
13         ;
14 004616 010246 SETUMP: MOV      R2,-(SP)
15 004620 010346      MOV      R3,-(SP)
16 004622 013746 000004      MOV      @#4,-(SP)      ;SAVE TRAP VECTOR
17 004626 012737 004674' 000004      MOV      #9#,@#4      ;CATCH TRAPS
18         ;
19         ; See if this is a type of maching that needs unibus mapping
20         ;
21 004634 105737 000000G      TSTB     UBUSMP      ;Is UNIBUS mapping needed?
22 004640 001415      BEQ      9#      ;Br if not
23         ;
24         ; Unibus mapping is needed
25         ; Load unibus map registers # 0-4 to point to low 48Kb of memory.
26         ;
27 004642 012705 000000G 2#:      MOV      #UMRADR,R5      ;POINT TO CONTROL REGISTER FOR UNIBUS MAP 0
28 004646 005004      CLR      R4      ;START MAPPING TO BOTTOM OF MEMORY
29 004650 012700 000005      MOV      #5,R0      ;LOAD 5 MAP REGISTERS
30 004654 010425 1#:      MOV      R4,(R5)+      ;SET LOW-ORDER VALUE IN MAP REGISTER
31 004656 005025      CLR      (R5)+      ;CLEAR HIGH-ORDER VALUE IN MAP REGISTER
32 004660 062704 020000      ADD      #8192.,R4      ;ADVANCE MEMORY ADDRESS
33 004664 077005      SOB      R0,1#      ;LOOP IF MORE MAP REGISTERS TO LOAD
34         ;
35         ; Turn on Unibus mapping
36         ;
37 004666 052737 000000G 000000G      BIS      #IOMAP,@#SR3MMR ;ENABLE UNIBUS MAPPING
38         ;
39         ; Finished
40         ;
41 004674 012637 000004 9#:      MOV      (SP)+,@#4      ;RESTORE TRAP VECTOR
42 004700 012605      MOV      (SP)+,R5
43 004702 012604      MOV      (SP)+,R4
44 004704 000207      RETURN
45         .IFF      ;NE,<PROASM-1>    ;Following code for Pro-only assembly
46         ;
47         ; Define dummy SETUMP routine for Pro
48         ;
49 SETUMP: RETURN
50         .ENDC    ;NE,<PROASM-1>

```

```

1          .SBTTL  DEVVEC -- Set up device vectors
2          ;-----
3          ; DEVVEC is called to set up device interrupt vectors for handlers
4          ; that have been loaded.
5          ;
6 004706 010146 DEVVEC: MOV     R1,-(SP)
7 004710 010346          MOV     R3,-(SP)
8 004712 010546          MOV     R5,-(SP)
9 004714 013746 000000G  MOV     @#KPAR5,-(SP) ; Save PAR 5 mapping
10         ;
11         ; Begin loop to set up vectors for each device
12         ;
13 004720 012701 000002          MOV     #2,R1          ; Get index # of 1st device after TT
14 004724 016103 000000G 1$:  MOV     HANENT(R1),R3 ; Get handler entry point address
15 004730 020327 000006          CMP     R3,#6          ; Is this a real device?
16 004734 101436          BLOS   6$          ; Br if not
17         ;
18         ; See if we need to map PAR 5 to this handler
19         ;
20 004736 016100 000000G          MOV     HANPAR(R1),R0 ; Get PAR 5 base for this handler
21 004742 001402          BEQ    2$          ; Br if this is not a mapped handler
22 004744 010037 000000G          MOV     R0,@#KPAR5    ; Map PAR 5 to the handler
23         ;
24         ; Clear CQE and LQE in handler header
25         ;
26 004750 005023 2$:  CLR     (R3)+          ; Clear LQE (4th word in handler)
27 004752 005013          CLR     (R3)          ; Clear CQE (5th word in handler)
28 004754 162703 000010          SUB     #10,R3        ; Point to 1st word of handler
29         ;
30         ; Set up interrupt vectors for this handler
31         ;
32 004760 005005          CLR     R5          ; Assume vector base address is 0
33 004762 005713          TST    (R3)          ; Any vectors to set up?
34 004764 001422          BEQ    6$          ; Br if no vectors to set up
35 004766 002403          BLT    5$          ; Br if multiple-vector
36 004770 004737 005060'          CALL   SETVEC        ; Set up the vector
37 004774 000416          BR     6$          ; Finished
38         ;
39         ; Multiple vectors.
40         ;
41 004776 012300 5$:  MOV     (R3)+,R0        ; Get offset to list of vector info
42 005000 006300          ASL    R0            ; Get byte offset to vector list
43 005002 060003          ADD    R0,R3        ; Get absolute address of vector table
44 005004 005713          TST    (R3)          ; Is this a PRO device with floating vectors?
45 005006 002005          BGE    7$          ; Br if not
46 005010 005723          TST    (R3)+        ; Point to word with device ID
47 005012 012346          MOV    (R3)+,-(SP)   ; Get device ID
48 005014 004777 000000G          CALL   @RPRVEC       ; Get base vector location for device
49 005020 012605          MOV    (SP)+,R5      ; This is base of vector locations
50 005022 004737 005060' 7$:  CALL   SETVEC        ; Set up the vector
51 005026 005713          TST    (R3)          ; Any more vectors to set up?
52 005030 003374          BGT    7$          ; Br if yes
53         ;
54         ; See if there are more devices to set up.
55         ;
56 005032 062701 000002 6$:  ADD    #2,R1          ; Advance device table index
57 005036 020137 000000G          CMP    R1,NUMDEV     ; More to do?

```

```
58 005042 101730          BLOS      1$          ;Br if yes
59                      ;
60                      ; Finished
61                      ;
62 005044 012637 000000G  MOV      (SP)+, @#KPAR5
63 005050 012605          MOV      (SP)+, R5
64 005052 012603          MOV      (SP)+, R3
65 005054 012601          MOV      (SP)+, R1
66 005056 000207          RETURN
```

```

1          .SBTTL  SETVEC -- Set up an interrupt vector for a device
2          ;-----
3          ; SETVEC is called to set up one interrupt vector for a device.
4          ;
5          ; Inputs:
6          ; R1 = Device index number.
7          ; R3 = Pointer into device handler to 3 word cells:
8          ;     1. Address of interrupt vector.
9          ;     2. Offset to interrupt entry point in handler.
10         ;     3. PS for interrupt.
11         ; R5 = Base address to add to vector locations.
12         ;
13         ; Outputs:
14         ; R3 = Points beyond 3 word info block in handler.
15         ;
16         ; Size of interrupt catching routine compiled for interrupts to
17         ; mapped handlers:
18         ;
19         ; MPIVSZ =      26.          ; Amt of code compiled for mapped ints
20         ;
21 005060 010446 SETVEC: MOV      R4, -(SP)
22         ;
23         ; See if this is a mapped handler
24         ;
25 005062 005761 0000000 TST      HANPAR(R1)      ; Is this a mapped handler
26 005066 001010 BNE      1$            ; Br if yes
27         ;
28         ; This is an unmapped handler.
29         ; Vector interrupts directly to the handler.
30         ;
31 005070 012300 MOV      (R3)+, R0      ; Get address of interrupt vector
32 005072 060500 ADD      R5, R0        ; Add base address to vector location
33 005074 010310 MOV      R3, (R0)      ; Store address of cell in handler
34 005076 062320 ADD      (R3)+, (R0)+  ; Add offset to interrupt entry point
35 005100 012310 MOV      (R3)+, (R0)   ; Set PS for interrupt
36 005102 052710 000340 BIS      #340, (R0)    ; Make sure priority = 7
37 005106 000450 BR       9$
38         ;
39         ; This is a mapped handler.
40         ; Vector the interrupt to a routine that performs the following functions:
41         ; 1. Save the current PAR 5 mapping.
42         ; 2. Map PAR 5 to the handler.
43         ; 3. Push a dummy PC and PS on stack that will send return from handler
44         ;    to a routine that will restore the PAR 5 mapping.
45         ; 4. Jump into the handler interrupt entry point.
46         ;
47 005110 013704 000122' 1$: MOV      XMVBAS, R4    ; Point to area where we store interrupt rtn
48 005114 012300 MOV      (R3)+, R0      ; Get address of interrupt vector
49 005116 060500 ADD      R5, R0        ; Add base address to interrupt location
50 005120 010420 MOV      R4, (R0)+     ; Direct interrupt to our routine
51 005122 012724 013746 MOV      #013746, (R4)+ ; [ MOV @#KPAR5, -(SP) ]
52 005126 012724 0000000 MOV      #KPAR5, (R4)+
53 005132 012724 012737 MOV      #012737, (R4)+ ; [ MOV #par5val, @#KPAR5 ]
54 005136 016124 0000000 MOV      HANPAR(R1), (R4)+
55 005142 012724 0000000 MOV      #KPAR5, (R4)+
56 005146 012724 012746 MOV      #012746, (R4)+ ; [ MOV #340, -(SP) ]
57 005152 012724 000340 MOV      #340, (R4)+

```

```
58 005156 012724 012746      MOV      #012746,(R4)+    ; [ MOV #HANXIT,-(SP) ]
59 005162 012724 000000G     MOV      #HANXIT,(R4)+
60 005166 012724 000257      MOV      #000257,(R4)+  ; [ CCC - Clear all condition codes ]
61 005172 016314 000002      MOV      2(R3),(R4)     ; [ SEx - Set condition codes specified in PS]
62 005176 042714 177760      BIC      #^C17,(R4)
63 005202 052724 000260      BIS      #260,(R4)+
64 005206 012724 000137      MOV      #000137,(R4)+  ; [ JMP @#handler_entry ]
65 005212 010314              MOV      R3,(R4)        ; Store address of int entry point
66 005214 062324              ADD      (R3)+,(R4)+
67 005216 012310              MOV      (R3)+,(R0)     ;Set PS for interrupt entry
68 005220 052710 000340      BIS      #340,(R0)     ;Make sure priority = 7
69                               ;
70                               ; Save address beyond end of compiled interrupt catcher routine
71                               ;
72 005224 010437 000122'     MOV      R4,XMVBAS      ;Save address beyond end of routine
73                               ;
74                               ; Finished
75                               ;
76 005230 012604              9$: MOV      (SP)+,R4
77 005232 000207              RETURN
```

```
1      .IF      NE,PROASM
2      .SBTTL  PIDVEN -- Make device table entry for PI device
3      ;-----
4      ; If we are running on a Professional computer, PIDVEN is called to
5      ; make an entry in the device tables for the PI device.
6      ;
7      PIDVEN: MOV      R1,-(SP)
8      ;
9      ; Increase number of defined devices and get device table entry index
10     ; to use for the PI device.
11     ;
12     ADD      #2,NUMDEV      ;One more device
13     MOV      NUMDEV,R1      ;Get device table index
14     ;
15     ; Set up information about the PI device
16     ;
17     MOV      #R5OPI,PNAME(R1) ;Set device name
18     MOV      #<DS$SFN!DI$PI>,DVSTAT(R1) ;Set device status flags
19     CLR      DVFLAG(R1)      ;Clear other flags
20     CLR      DEVSIZ(R1)      ;Clear device size
21     MOV      #PIHAN+6,HANENT(R1);Set handler entry point (4th word)
22     MOV      #PROSIZ,HANSIZ(R1) ;Set handler size
23     ;
24     ; Finished
25     ;
26     MOV      (SP)+,R1
27     RETURN
28     .ENDC      ;NE,PROASM
```

SETVEC -- Set up an interrupt vector for a device

```

1          .IF      NE,<PROASM-1> ;If assembling for PDP-11
2          .SBTTL  LININI -- Initialize time-sharing lines
3          ;-----
4          ; LININI is called to initialize the time-sharing lines.
5          ; This consists of setting up interrupt vectors and setting control
6          ; flags in the status registers.
7          ;
8 005234 010146 LININI: MOV      R1,-(SP)
9 005236 010246      MOV      R2,-(SP)
10 005240 010346      MOV      R3,-(SP)
11 005242 010446      MOV      R4,-(SP)
12 005244 010546      MOV      R5,-(SP)
13          ;
14          ; Set up interrupt vectors for DL11 lines
15          ;
16 005246 012701 000002      MOV      #2,R1          ;Index for 1st line
17 005252 012704 000340      MOV      #340,R4        ;Priority 7 PS
18 005256 032761 000000G 000000G 1$: BIT      #$DEAD,LSW3(R1) ;Is this line uninstalled?
19 005264 001027      BNE      B$                ;Br if yes
20 005266 032761 000000G 000000G      BIT      #$HARD,LSW3(R1) ;Is this line connected to hardware?
21 005274 001423      BEQ      B$                ;Br if not
22 005276 026127 000000G 000000G      CMP      LCDTYP(R1),#CDX$DL ;Is this a DL11 line?
23 005304 001017      BNE      B$                ;Br if not
24          ;
25          ; DL-11 line
26          ;
27 005306 016105 000000G      MOV      INVEC(R1),R5      ;GET ADDRESS OF INPUT VECTOR
28 005312 012702 000000G      MOV      #INRECV,R2      ;END OF RECEIVING VECTOR
29 005316 012703 177772G      MOV      #<OTRECV-6>,R3 ;START OF INPUT INTERRUPT ENTRY POINTS
30 005322 010100      MOV      R1,R0          ;GET LINE NUMBER
31 005324 006300      ASL      R0                ;4 BYTES PER INPUT INTERRUPT ENTRY POINT
32 005326 160002      SUB      R0,R2          ;GET ADDRESS OF INPUT INTERRUPT ENTRY POINT
33 005330 060100      ADD      R1,R0          ;6 BYTES PER OUTPUT INTERRUPT ENTRY POINT
34 005332 060003      ADD      R0,R3          ;GET ADDRESS OF OUTPUT INTERRUPT ENTRY POINT
35 005334 010225      MOV      R2,(R5)+        ;SET PC FOR INPUT INTERRUPT ENTRY POINT
36 005336 010425      MOV      R4,(R5)+        ;SET PS FOR INPUT INTERRUPT
37 005340 010325      MOV      R3,(R5)+        ;SET PC FOR OUTPUT INTERRUPT
38 005342 010425      MOV      R4,(R5)+        ;SET PS FOR OUTPUT INTERRUPT
39          ;
40          ; Try next line
41          ;
42 005344 062701 000002      B$: ADD      #2,R1          ;ADVANCE LINE INDEX NUMBER
43 005350 020127 000000G      CMP      R1,#LSTHL      ;MORE TO DO?
44 005354 101740      BLOS     1$                ;BR IF YES
45          ;
46          ; Initialize multiplexers.
47          ;
48 005356 012701 000000G      SETMUX: MOV      #LSTMX,R1      ;Get last mux index #
49 005362 001423      BEQ      SETLIN      ;Br if there are no mux lines
50 005364 026127 000000G 000000G 3$: CMP      MXTYPE(R1),#CDX$DZ ;Is this a DZ11, DH11, or DHV11?
51 005372 001412      BEQ      1$                ;Br if DZ11
52 005374 026127 000000G 000000G      CMP      MXTYPE(R1),#CDX$VH ;Is this a DHV11?
53 005402 001003      BNE      2$                ;Br if not
54 005404 004737 006154'      CALL     VHINIT      ;Initialize a DHV11
55 005410 000405      BR      4$                ;
56 005412 004737 006104'      2$: CALL     DHINIT      ;Initialize a DH11
57 005416 000402      BR      4$                ;

```

LININI -- Initialize time-sharing lines

```

58 005420 004737 005714'      1$:      CALL      DZINIT      ;Initialize a DZ11
59 005424 162701 000002      4$:      SUB        #2,R1      ;More to enable?
60 005430 001355              BNE        3$          ;Br if yes
61                               ;
62                               ; Enable all lines
63                               ;
64 005432 012701 000000G      SETLIN:  MOV      #LSTHL,R1      ;INDEX # OF LAST REAL LINE
65 005436 032761 000000G 000000G 4$:      BIT        $$DEAD,LSW3(R1) ;IS THIS LINE INSTALLED?
66 005444 001057              BNE        2$          ;BR IF NOT
67 005446 032761 000000G 000000G      BIT        $$HARD,LSW3(R1) ;Is this line connected to hardware?
68 005454 001453              BEQ        2$          ;Br if not
69 005456 032761 000000G 000000G      BIT        $$PHONE,ILSW2(R1); IS THIS A DIAL-UP LINE?
70 005464 001403              BEQ        3$          ;BR IF NOT
71 005466 052761 000000G 000000G      BIS        $$NOIN,LSW3(R1) ;IGNORE INPUT TILL DIAL UP OCCURS
72 005474 016105 000000G      3$:      MOV      LCDTYP(R1),R5      ;Get comm device type code
73 005500 016100 000000G      MOV      LMXNUM(R1),R0      ;IS THIS A DL-11 OR MUX LINE?
74 005504 001423              BEQ        1$          ;BR IF DL-11
75 005506 020527 000000G      CMP      R5,#CDX$DZ        ;Is this a DZ11 or DH11?
76 005512 001411              BEQ        6$          ;Br if DZ11
77 005514 020527 000000G      CMP      R5,#CDX$VH        ;Is this a DH11 or DHV11?
78 005520 001403              BEQ        7$          ;Br if DHV11
79                               ;
80                               ; DH11 line
81                               ;
82 005522 004737 005626'      CALL      DHLPRM          ;Set line parameters for DH11 line
83 005526 000426              BR        2$
84                               ;
85                               ; DHV11 line
86                               ;
87 005530 004737 005664'      7$:      CALL      VHLPRM          ;Set line parameters for DHV11 line
88 005534 000423              BR        2$
89                               ;
90                               ; DZ-11 line
91                               ;
92 005536 016102 000000G      6$:      MOV      LMXLN(R1),R2      ;Get line # within mux group
93 005542 052702 017030      BIS        #017030,R2      ;Set line enable flags
94 005546 010270 000000G      MOV      R2,@MXLPR(R0)      ;Enable the line
95 005552 000414              BR        2$
96                               ;
97                               ; DL-11 line
98                               ;
99 005554 016102 000000G      1$:      MOV      TSR(R1),R2      ;ADDRESS OF TRANSMITTER STATUS REGISTER
100 005560 011203              MOV      (R2),R3          ;CLEAR TRANSMITTER STATUS REGISTER
101 005562 005012              CLR      (R2)
102 005564 016102 000000G      MOV      RBR(R1),R2      ;ADDRESS OF RECEIVER BUFFER REGISTER
103 005570 011203              MOV      (R2),R3          ;CLEAR RECEIVER BUFFER REGISTER
104 005572 016102 000000G      MOV      RSR(R1),R2      ;ADDRESS OF RECEIVER STATUS REGISTER
105 005576 005012              CLR      (R2)
106 005600 012712 000000G      MOV      #RDINT,(R2)      ;ENABLE RECEIVER INTERRUPTS
107                               ;
108                               ; Do next line
109                               ;
110 005604 162701 000002      2$:      SUB        #2,R1
111 005610 003312              BGT      4$
112                               ;
113                               ; Finished
114                               ;

```

LININI -- Initialize time-sharing lines

115	005612	012605	MOV	(SP)+, R5
116	005614	012604	MOV	(SP)+, R4
117	005616	012603	MOV	(SP)+, R3
118	005620	012602	MOV	(SP)+, R2
119	005622	012601	MOV	(SP)+, R1
120	005624	000207	RETURN	

DHLPRM -- Set line parameters for a DH11 line

```

1          .SBTTL  DHLPRM -- Set line parameters for a DH11 line
2          ;-----
3          ; DHLPRM is called to set up the line parameters for a DH11 line.
4          ;
5          ; Inputs:
6          ; R1 = Physical line index number.
7          ;
8 005626   DHLPRM:
9          ;
10         ; Enable DM11 for this line
11         ;
12 005626  016100  000000G      MOV     LMXNUM(R1),R0    ;Get mux index number
13 005632  005760  000000G      TST     DM$CSR(R0)      ;Does this DH11 have DM11 modem control?
14 005636  001411                BEQ     2$              ;Br if not
15 005640  142770  000000G 000000G  BICB   #MF$LIN,@DM$CSR(R0) ;Clear line # field in DM11 CSR
16 005646  156170  000000G 000000G  BISB   LMXLN(R1),@DM$CSR(R0);Select line of interest
17 005654  012770  000000G 000000G  MOV     #MF$LE,@DM$LSR(R0);Enable the line
18         ;
19         ; Finished
20         ;
21 005662  000207                2$:   RETURN

```

VHLPRM -- Set line parameter values for DHV11 line

```

1          .SBTTL  VHLPRM -- Set line parameter values for DHV11 line
2          ;-----
3          ; Set the line parameter values for a DHV11 line.
4          ;
5          ; Inputs:
6          ;   R1 = Physical line index number.
7          ;
8 005664   VHLPRM:
9          ;
10         ; Enable the line
11         ;
12 005664  016100  000000G          MOV     LMXNUM(R1),R0 ;Get mux index number
13 005670  042770  000000G 000000G  BIC     #VF$LIN,@VH$CSR(R0) ;Clear line # field in mux CSR
14 005676  156170  000000G 000000G  BISB   LMXLN(R1),@VH$CSR(R0) ;Set our line #
15 005704  012770  000000G 000000G  MOV     #<VF$RE>,@VH$LOR(R0) ;Enable the line
16         ;
17         ; Finished
18         ;
19 005712  000207          RETURN

```

DZINIT -- Initialize a DZ11 multiplexer

```

1                                     .SBTTL  DZINIT -- Initialize a DZ11 multiplexer
2                                     ;-----
3                                     ; DZINIT is called to initialize a DZ11 multiplexer.
4                                     ;
5                                     ; Inputs:
6                                     ; R1 = Mux index number.
7                                     ;
8 005714 DZINIT:
9                                     ;
10                                    ; See if this DZ11 is installed
11                                    ;
12 005714 005761 000000G          TST      MXCSR(R1)      ;Is this DZ-11 installed?
13 005720 001416                  BEQ      4$             ;Br if not
14                                    ;
15                                    ; Set up interrupt vector connections for this MUX
16                                    ;
17 005722 004737 005760'          CALL     MUXVEC          ;Set up interrupt vectors for this DZ11
18                                    ;
19                                    ; Start up the mux operation
20                                    ;
21 005726 052771 000000G 000000G      BIS      #ZCLR,@MXCSR(R1);Do master clear on DZ-11
22 005734 032771 000000G 000000G 1$:  BIT      #ZCLR,@MXCSR(R1);Wait for clear to finish
23 005742 001374                  BNE      1$
24 005744 017100 000000G          2$:  MOV      @MXRBUF(R1),R0 ;Clear silo
25 005750 100775                  BMI      2$
26 005752 105071 000000G          CLRB     @MXDTR(R1)      ;Disable all data sets
27                                    ;
28                                    ; Finished
29                                    ;
30 005756 000207          4$:  RETURN

```

MUXVEC -- Set up interrupt vectors for a multiplexer

```

1                                     .SBTTL  MUXVEC -- Set up interrupt vectors for a multiplexer
2                                     ;-----
3                                     ; MUXVEC is called to set up the interrupt vector connections for
4                                     ; a DZ11, DH11, or DHV11 multiplexer.
5                                     ;
6                                     ; Inputs:
7                                     ; R1 = Mux index number.
8                                     ;
9 005760 010246 MUXVEC: MOV      R2, -(SP)
10 005762 010346    MOV      R3, -(SP)
11 005764 010546    MOV      R5, -(SP)
12                                     ;
13                                     ; Set interrupt vector for mux
14                                     ;
15 005766 016105 000000G    MOV      MUXVEC(R1), R5    ;Get address of input interrupt vector
16 005772 012702 000000G    MOV      #INMXV, R2    ;End of receiving vector
17 005776 012703 177772G    MOV      #<OTMXV-6>, R3 ;Output interrupt table
18 006002 010100    MOV      R1, R0    ;Get mux index number
19 006004 006300    ASL      R0    ;4 bytes per line in input int table
20 006006 160002    SUB      R0, R2    ;Get address of input entry point
21 006010 060100    ADD      R1, R0    ;6 bytes per mux in output entry point table
22 006012 060003    ADD      R0, R3    ;Get address of output int entry point
23 006014 010225    MOV      R2, (R5)+    ;Set PC for input interrupt
24 006016 012725 000340    MOV      #340, (R5)+ ;Set PS for output interrupt
25 006022 010325    MOV      R3, (R5)+    ;Set PC for output interrupt
26 006024 012715 000340    MOV      #340, (R5)   ;Set PS for output interrupt
27                                     ;
28                                     ; Now store an instruction sequence of the form:
29                                     ;
30                                     ; JSR      R5, @#interrupt_routine
31                                     ; .WORD   mux_index
32                                     ;
33                                     ; to catch mux output interrupts and vector them to the interrupt routine.
34                                     ;
35 006030 012723 004537    MOV      #004537, (R3)+ ;JSR R5, @#x
36 006034 012700 000000G    MOV      #DZOINT, R0   ;Assume this is a DZ11
37 006040 026127 000000G 000000G    CMP      MXTYPE(R1), #CDX#DZ ;Is this a DZ11?
38 006046 001410    BEQ     1$            ;Br if yes
39 006050 012700 000000G    MOV      #VHOINT, R0   ;Assume this is a DHV11
40 006054 026127 000000G 000000G    CMP      MXTYPE(R1), #CDX#VH ;Is this a DHV11?
41 006062 001402    BEQ     1$            ;Br if yes
42 006064 012700 000000G    MOV      #DHOINT, R0   ;Get interrupt routine for DH11's
43 006070 010023 1$:    MOV      R0, (R3)+    ;Store address of interrupt routine
44 006072 010113    MOV      R1, (R3)     ;Store mux index number
45                                     ;
46                                     ; Finished
47                                     ;
48 006074 012605 9$:    MOV      (SP)+, R5
49 006076 012603    MOV      (SP)+, R3
50 006100 012602    MOV      (SP)+, R2
51 006102 000207    RETURN

```

DHINIT -- Initialize a DH11 multiplexer

```

1          .SBTTL  DHINIT -- Initialize a DH11 multiplexer
2          ;-----
3          ; DHINIT is called to initialize a DH11 multiplexer
4          ;
5          ; Inputs:
6          ;   R1 = Mux index number
7          ;
8 006104   DHINIT:
9          ;
10         ; See if this DH11 is installed
11         ;
12 006104   005761   000000G   TST      MH$SCR(R1)   ;Is this DH11 installed?
13 006110   001420   BEQ      9$      ;Br if not
14         ;
15         ; Connect interrupt vector to DH11
16         ;
17 006112   004737   005760'   CALL     MUXVEC      ;Set up interrupt vectors for DH11
18         ;
19         ; Clear the multiplexer
20         ;
21 006116   012771   000000G 000000G   MOV     #HF$MC,@MH$SCR(R1) ;Set the master-clear flag
22 006124   032771   000000G 000000G 1$:  BIT     #HF$MC,@MH$SCR(R1) ;Wait for the master clear to be completed
23 006132   001374   BNE     1$
24         ;
25         ; Clear the DM11 scanner
26         ;
27 006134   016100   000000G   MOV     DM$CSR(R1),R0   ;Is there an associated DM11?
28 006140   001404   BEQ     3$      ;Br if not
29 006142   012710   000000G   MOV     #MF$CS,(R0)   ;Clear the scanner
30 006146   052710   000000G   BIS     #MF$CM,(R0)   ;Clear the multiplexer
31 006152   3$:
32         ;
33         ; Finished
34         ;
35 006152   9$:
36 006152   000207   RETURN

```

VHINIT -- Initialize a DHV11 multiplexer

```

1          .SBTTL  VHINIT -- Initialize a DHV11 multiplexer
2          ;-----
3          ; Perform initialization for a DHV11 mux.
4          ;
5          ; Inputs:
6          ; R1 = Mux index number.
7          ;
8 006154   VHINIT:
9          ;
10         ; See if this DHV11 is installed
11         ;
12 006154   005761   000000G          TST      VH$CSR(R1)      ;Is this DHV11 installed?
13 006160   001414          BEQ      9$              ;Br if not
14         ;
15         ; Connect interrupt vector to DHV11
16         ;
17 006162   004737   005760'          CALL    MUXVEC          ;Set up interrupt vectors
18         ;
19         ; Clear the multiplexer
20         ;
21 006166   012771   000000G 000000G          MOV     #VF$MR,@VH$CSR(R1) ;Reset the multiplexer
22 006174   032771   000000G 000000G 1$:    BIT     #VF$MR,@VH$CSR(R1) ;Wait for reset to finish
23 006202   001374          BNE     1$
24         ;
25         ; Clean out the FIFO buffer in the mux
26         ;
27 006204   017100   000000G          2$:    MOV     @MXRBUF(R1),R0 ;Get contents of receiver buffer register
28 006210   002775          BLT     2$              ;Loop until RBUF empty
29         ;
30         ; Finished
31         ;
32 006212   000207          9$:    RETURN

```

VHINIT -- Initialize a DHV11 multiplexer

```

1          ;-----
2          ; End of code that can be omitted for Pro-only systems
3          ;
4          .IFF      ;NE,<PROASM-1> ;Begin code for Pro only
5          ;
6          ; This code is assembled only for Pro systems.
7          ; T/S line init routines for Pro only.
8          ;
9          LINCHK:
10         DHLPRM:
11         VHLPRM:
12         DZINIT:
13         MUXVEC:
14         DHINIT:
15         VHINIT:
16         RETURN
17         ;
18         ; LININI routine for Pro systems
19         ;
20         LININI: MOV     R1,-(SP)
21                MOV     #LSTLIN,R1      ;Get # of last line
22         ;
23         ; Determine if this line is connected to hardware
24         ;
25         1$: CALL     LINTYP      ;Determine the type of this line
26                BIT     #$HARD,LSW3(R1) ;Is this line connected to hardware?
27                BEQ     2$        ;Br if not
28         ;
29         ; Call Pro line initialization routine
30         ;
31                CALL     PROLIN      ;Initialize Pro line
32         ;
33         ; Do some special init for phone lines
34         ;
35                BIT     #$PHONE,ILSW2(R1);Is this a dialup line?
36                BEQ     2$        ;Br if not
37                BIS     #$NOIN,LSW3(R1);Ignore input till dial up occurs
38         ;
39         ; Check next line
40         ;
41         2$: SUB     #2,R1      ;Get index # of next line
42                BGT     1$        ;Loop if more lines to do
43         ;
44         ; Finished
45         ;
46                MOV     (SP)+,R1
47                RETURN
48         ;
49         ; End of Pro-only code
50         ;
51         .ENDC      ;NE,<PROASM-1>

```


* * * Initialization done with RT-11 running * * *

```

1          .SBTTL * * * Initialization done with RT-11 running * * *
2          ;-----
3          ; Initialization at start of execution of TSX.
4          ;
5          ; The initialization done in this section uses the running RT-11 system
6          ; to perform functions for it.
7          ;
8 006326   INITGD:
9          ;
10         ; Save some RT-11 pointers in case we abort the initialization
11         ;
12 006326   013737   000004   000044'   MOV     @#4,RTTRP4       ;Save trap 4 vector
13 006334   013737   000000G 000046'   MOV     @#RMON,RTMNVG   ;RT-11 monitor pointer
14 006342   013737   000000G 000042'   MOV     @#CLKVEC,CLK100 ;Clock vector (defined in TSGEN at 100)
15         ;
16         ; Get the current time of day which we will use later to make sure
17         ; the line time clock is working.
18         ;
19 006350   .GTIM    #AREA,#SYTIMH     ;Get the current time of day
20         ;
21         ; Trap ^C for later test so we can restore clock vector
22         ;
23 006370   .SCCA    #AREA,#CCAFLE     ;Catch control-C
24         ;
25         ; Check for TSGEN size overflow
26         ;
27 006410   012700   000000G   MOV     #GENTOP,R0      ;Get top of TSGEN
28 006414   162700   037776     SUB     #<40000-2>,R0   ;Will TSKMON have problems?
29 006420   003422     BLE     15#            ;Continue if not
30 006422   010046     MOV     R0,-(SP)       ;Save overflow size
31 006424     .PRINT  #TSXHD     ;Print error message
32 006432     .PRINT  #TOOBIG
33 006440     .PRINT  #REDUCE
34 006446   012600     MOV     (SP)+,R0      ;Recover amount of overflow
35 006450   004737   027754'   CALL   PRTDEC
36 006454     .PRINT  #BYTES
37 006462   000137   004102'   JMP    INISTP
38         ;
39         ; Initialize the system stack (below 1000)
40         ;
41 006466   012701   000000G 15#:   MOV     #SSEND,R1     ;Point to bottom of stack
42 006472   012700   123456     MOV     #123456,R0    ;Get initialization value
43 006476   010021 13#:   MOV     R0,(R1)+     ;Initialize the stack
44 006500   020127   000000G   CMP     R1,#SS       ;Reached top of the stack area?
45 006504   103774     BLO    13#          ;Loop if not
46 006506   010106     MOV     R1,SP        ;Run on system stack
47         ;
48         ; Make sure we are not already running under TSX.
49         ;
50 006510     .SERR                    ;DON'T DIE ON ERRORS
51 006516   012700   000176'   MOV     #GTLIN,R0    ;TSX EMT TO GET LINE NUMBER
52 006522   104375     EMT    375          ;TRY A TSX EMT
53 006524   103410     BCS    1#           ;BR IF NOT UNDER TSX
54 006526     .PRINT  #TSXHD     ;ALREADY UNDER TSX
55 006534     .PRINT  #TSXRUN
56 006542   000137   004102'   JMP    INISTP
57 006546 1#:   .HERR                    ;RENABLE FATAL ERRORS

```

* * * Initialization done with RT-11 running * * *

```

58      ;
59      ;   Make sure this machine has memory management facilities.
60      ;
61 006554      .TRPSET #AREA,#NOXM      ; CATCH TRAPS
62 006574 005737 000000G      TST      @#SROMMR      ; TRY TO ACCESS MEMORY MANAGEMENT REGISTER
63 006600      .TRPSET #AREA,#0      ; Release trap control
64      ;
65      ;   Request all available memory from RT-11.
66      ;
67 006616      .SETTOP #-2      ; REQUEST ALL AVAILABLE MEMORY
68 006624 010037 000132'      MOV      R0, TOPMEM      ; REMEMBER WHERE TOP OF MEMORY IS
69 006630 020027 000000G      CMP      R0, #VPAR5      ; TSX CANNOT EXTEND ABOVE PAR5 BASE ADDRESS
70 006634 101402      BLOS      3$      ; BR IF RT-11 IS BELOW THAT
71 006636 012700 000000G      MOV      #VPAR5, R0      ; SET PAR5 BASE AS UPPER LIMIT ON TSX SIZE
72 006642 010037 000236'      3$:     MOV      R0, MEMLIM      ; TSX MAY NOT EXCEED THIS UPPER LIMIT
73      ;
74      ;   Lock USR in memory for speed
75      ;   (Set USR to swap over TSEMT to get out of the way)
76      ;
77 006646 012705 177776'      MOV      #TSINIT-2, R5      ; GET THE BASE OF TSINIT
78 006652      .GVAL #AREA, #374      ; GET SIZE OF RT-11 USR MODULE
79 006672 160005      SUB      R0, R5      ; ALLOCATE SPACE BELOW TSINIT FOR USR
80 006674 010537 000046      MOV      R5, @#46      ; SET USR TO SWAP OVER TSEMT
81 006700      5$:     .LOCK      ; LOCK USR IN MEMORY
82      ;
83      ;   Determine if we are to run system with the system debugger
84      ;
85 006702 032737 000000G 000000G      BIT      #CHAIN, @#JSWLOC ; WERE WE CHAINED TO?
86 006710 001406      BEQ      10$      ; BR IF NOT
87 006712 023727 000510 057164      CMP      @#510, #R500DT ; SHOULD WE RUN UNDER ODT?
88 006720 001002      BNE      10$      ; BR IF NOT
89 006722 005237 000034'      INC      ODTFLG      ; SET FLAG SAYING DEBUGGER WANTED
90      ;
91      ;   Call Pro TSX initialization only if assembling for the Pro
92      ;   Jump to INISTP if checking fails.
93      ;
94 006726      10$:
95      .IF      NE, PROCID      ; ** Do if assembling for pro only **
96      CALL      INSCHK      ; PERFORM VERIFICATION AND DECRYPTION FOR PRO
97      .ENDC      ; NE, PROCID
98      ;
99      ;   Allocate non-initialized buffer space over TSINIT.
100     ;
101 006726 012705 000000'      MOV      #TSINIT, R5      ; Allocate buffer space over TSINIT
102 006732 004737 013060'      CALL      ALOCBF      ; Do allocation
103 006736 004737 013524'      CALL      ALCSLO      ; Allocate silo buffers for lines
104 006742 020527 006326'      CMP      R5, #INITGD      ; Are we beyond code that takes over control?
105 006746 103002      BHIS      12$      ; Br if yes
106 006750 012705 006326'      MOV      #INITGD, R5      ; Advance up to initial code
107     ;
108     ;   Allocate the interrupt stack over TSINIT
109     ;   If we are running on a Pro, allocate buffer for the PI handler
110     ;   initialization code over the interrupt stack area.
111     ;
112     PIINSZ = 700.      ; Space needed for PI init code
113 006754 010537 000000G      12$:   MOV      R5, INTSND      ; Ptr to base of interrupt stack
114 006760 062705 000002      ADD      #2, R5      ; Always leave last word of stack for flag val

```

* * * Initialization done with RT-11 running * * *

```

115 006764 013701 000000G      MOV      @#RMON,R1      ;Get pointer to RT-11 RMON base
116 006770 032761 000000G 000370  BIT      #CW$PRO,370(R1) ;Are we running on a PRO?
117 006776 001407          BEQ      11$           ;Br if not
118 007000 105237 000000G      INCB     PROFLG        ;Set flag saying this is a PRO-350
119 007004 010537 000150'      MOV      R5,PROBUF     ;Save pointer to buffer area
120 007010 062705 001274      ADD      #PIINSZ,R5    ;Allocate space for buffer
121 007014 000402          BR       14$           ;
122 007016 062705 000000G      11$:    ADD      #INTSSZ,R5 ;Allocate space for interrupt stack
123 007022 010537 000000G      14$:    MOV      R5,INTSTK ;Address of top of interrupt stack
124          ;
125          ; Allocate space for those overlays that go over TSINIT
126          ;
127 007026 004737 023372'      CALL     OVLPOS        ;Determine how much space to alloc for overlay
128          ;
129          ; Note: from this point onward we are carrying the address of the
130          ; base of the free memory area in R5.
131          ;
132 007032 020527 030074'      CMP      R5,#INITOP    ;Have we allocated up to top of TSINIT?
133 007036 103002          BHS      4$           ;Br if yes
134 007040 012705 030074'      MOV      #INITOP,R5    ;Advance to top of TSINIT
135          ;
136          ; Allocate a 2048. byte work buffer
137          ;
138 007044 004737 010340'      4$:    CALL     ALCWRK        ;Allocate work buffer
139          ;
140          ; Allocate empty Region Control Blocks for use by handlers
141          ;
142 007050 004737 010374'      CALL     ALCHRB        ;
143          ;
144          ; If we were started in debug mode, load ODT.
145          ;
146          ; .IF      EQ,PROCID      ;Don't allow ODT for production PRO version
147 007054 005737 000034'      TST     ODTFLG        ;Are we to load system debugger?
148 007060 001402          BEQ      2$           ;Br if not
149 007062 004737 026314'      CALL     GETODT        ;Load ODT and start it
150          ; .ENDC      ;EQ,PROCID
151          ;
152          ; Initialize memory management registers for 1-to-1 mapping but
153          ; leave memory management turned off
154          ;
155 007066 004737 016436'      2$:    CALL     MEMINI        ;Initialize memory management
156          ;
157          ; Extract information from RT-11 configuration and sysgen words.
158          ;
159 007072 013701 000000G      MOV      @#RMON,R1      ;GET POINTER TO BASE OF RMON
160 007076 016102 000300      MOV      300(R1),R2     ;GET RT-11 CONFIGURATION WORD
161 007102 042702 000000C      BIC     #CW$GDH+CW$BTH+CW$LGS,R2 ;RESET A FEW FLAGS
162 007106 052702 000000C      BIS     #CW$FB+CW$FQJ+CW$USR+CW$XM,R2 ;SET A FEW FLAGS
163 007112 010237 000000G      MOV      R2,CONFIG      ;INITIALIZE OUR CONFIGURATION WORD
164          ; Now get extended configuration word.
165 007116 016137 000370 000000G  MOV      370(R1),CONF2  ;EXTENDED CONFIGURATION WORD
166 007124 052737 000000G 000000G  BIS     #CW$ESP,CONF2   ;SET EXIT NO SWAP FLAG
167 007132 123727 000000G 000000G  CMPB    VBUSTP,#QBUS    ;Is this a Q-bus machine?
168 007140 001003          BNE     25$           ;Br if not
169 007142 052737 000000G 000000G  BIS     #CW$QBS,CONF2   ;Set QBUS flag
170          ; And sysgen option word.
171 007150 016102 000372      25$:    MOV      372(R1),R2

```

*** Initialization done with RT-11 running ***

```

172 007154 042702 0000000      BIC      #SG$ELG+SG$PAR+SG$MTS,R2
173 007160 052702 0000000      BIS      #SG$MMU+SG$MTM+SG$IOT+SG$TSX,R2
174 007164 010237 000000G      MOV      R2,SYSGEN      ;INITIALIZE OUR SYSGEN WORD
175
176      ; If a system version number was specified, use it.
177      ; Else, get version number from RT-11, but limit to default.
178
179 007170 013700 000000G      MOV      SYSVER,R0      ;Has user specified version to emulate?
180 007174 001015      BNE      30$      ;If so, keep SYSVER
181 007176 012737 000314' 000156'      MOV      #RTVDEF,RTVPTR ;If using RT version, set cutoff
182 007204      .GVAL    #AREA,#276      ;GET RT-11 SYSTEM VERSION NUMBER
183 007224 010037 000000G      MOV      R0,SYSVER      ;SET AS TSX-PLUS VERSION NUMBER
184
185      ; Now scan the known version number table and try to locate a match.
186      ; RO contains version # in low byte, update # in high byte
187
188 007230 012702 000264'      30$:    MOV      #RTVER,R2      ;Get ptr to first entry in table
189 007234 120062 000000      31$:    CMPB    RO,RT$VER(R2) ;Does main version match?
190 007240 001005      BNE      32$      ;Br if not
191 007242 000300      SWAB    R0      ;Main version matches, get update to low byte
192 007244 120062 000001      CMPB    RO,RT$UPD(R2) ;Does the update match also?
193 007250 001425      BEQ     34$      ;Its a match! Use this entry
194 007252 000300      SWAB    R0      ;Get SYSVER back to low byte
195 007254 062702 000003      32$:    ADD     #RTV$SZ,R2      ;If not, step up to the next entry
196 007260 020227 000322'      CMP     R2,#RTVEND      ;Past end of table?
197 007264 103763      BLO     31$      ;Keep checking if not
198
199      ; Couldn't find version in tables. If we picked it up from RT, reset
200      ; everything to the default. If it was user-specified, then keep
201      ; SYSVER, but use last entry ptr.
202
203 007266 023727 000156' 000322'      CMP     RTVPTR,#RTVEND ;Was a limit specified? (Got ver. from RT?)
204 007274 103404      BLO     33$      ;Br if so
205      ; Unknown user-specified version, keep user-specified SYSVER,
206      ; but use ptr to last known version
207 007276 012737 000317' 000156'      MOV     #<RTVEND-RTV$SZ>,RTVPTR ;Use latest known defaults
208 007304 000414      BR     36$
209      ; Got version from RT, but don't recognize it, reset SYSVER and use defaults
210      ; RTVPTR was already set to default RTVDEF when we got RT version
211 007306 113737 000314' 000000G      33$:    MOVB   RTVDEF+RT$VER,SYSVER ;Set SYSVER to default
212 007314 113737 000315' 000000G      MOVB   RTVDEF+RT$UPD,SYSUPD ;and update
213 007322 000405      BR     36$
214
215      ; Version was identified in tables. RTVPTR contains limiting version ptr
216      ; (RTVDEF if got vers from RT, RTVEND if user specified version)
217
218 007324 020237 000156'      34$:    CMP     R2,RTVPTR      ;Is it past limit?
219 007330 103366      BHIS   33$      ;If so, keep limit, and go limit SYSVER
220 007332 010237 000156'      MOV     R2,RTVPTR      ;If not, use what we found
221
222      ; Now set some information that depends on the emulated version number
223
224 007336 105737 000000G      35$:    TSTB   CLVERS      ;Use table value for CL version number?
225 007342 001005      BNE     38$      ;Br if not, use value supplied in TSGEN
226 007344 013702 000156'      MOV     RTVPTR,R2      ;Get ptr to version table
227 007350 116237 000002 000000G      MOVB   CL$VER(R2),CLVERS ;Auto set CLVERS from version table
228 007356 105737 000000G      38$:    TSTB   LDVERS      ;Auto-select LD translation table type?

```

*** Initialization done with RT-11 running ***

229	007362	001011		BNE	39#	;Br if not, use value supplied in TSGEN
230	007364	112737	000001 000000G	MOVB	#1,LDVERS	;Assume old translation table format
231	007372	023727	000156' 000314'	CMP	RTVPTR,#RT54	;At or beyond 5.4?
232	007400	103402		BLO	39#	;Br if not, retain old format
233	007402	105237	000000G	INCB	LDVERS	;LD translation table format changed at 5.4
234	007406		39#:			

*** Initialization done with RT-11 running ***

```

1
2 ; Set up a few clock constants based on clock frequency.
3 ; See if we have a 50 or 60 Hz clock
4 ;
5 007406 032737 000000G 000000G INICLK: BIT #CW$50H, CONFIG ; 50 or 60 Hz clock?
6 007414 001017 ; BNE 2$ ; Br if 50 Hz
7 ;
8 ; 60 Hz clock
9 ;
10 007416 012737 000074 000000G MOV #60., TK1SEC ; Clock ticks per 1 second
11 007424 012737 000036 000000G MOV #30., TK5VAL ; Clock ticks per 0.5 seconds
12 007432 012737 000264 000000G MOV #180., TK3SVL ; Clock ticks per 3 seconds
13 007440 012737 000006 000000G MOV #6., TK1VAL ; Clock ticks per 0.1 seconds
14 007446 012700 001130 MOV #600., R0 ; Get # clock ticks per 10 seconds
15 007452 000416 ; BR 8$
16 ;
17 ; 50 Hz clock
18 ;
19 007454 012737 000062 000000G 2$: MOV #50., TK1SEC ; Clock ticks per 1 second
20 007462 012737 000031 000000G MOV #25., TK5VAL ; Clock ticks per 0.5 seconds
21 007470 012737 000226 000000G MOV #150., TK3SVL ; Clock ticks per 3 seconds
22 007476 012737 000005 000000G MOV #5., TK1VAL ; Clock ticks per 0.1 seconds
23 007504 012700 000764 MOV #500., R0 ; Get # clock ticks per 10 seconds
24 ;
25 ; Set number of clock ticks per day
26 ;
27 007510 012702 020700 8$: MOV #8640., R2 ; (# seconds per day) / 10.
28 007514 070200 MUL R0, R2 ; Get # clock ticks per day
29 007516 010237 000000G MOV R2, DATIMH ; High-order value
30 007522 010337 000000G MOV R3, DATIML ; Low-order value
31 ;
32 ; Do a fast check to make sure specified T/S line addresses are ok.
33 ;
34 007526 004737 010450' CKLIN: CALL LINCHK ; CHECK T/S LINE ADDRESSES
35 ;
36 ; Do PRO-350 system initialization
37 ;
38 .IF NE, PROASM
39 BIT #CW$PRO, CONFIG2 ; Are we running on a PRO-350?
40 BEQ INIDEV ; Br if not
41 CALL PROINI ; Do PRO-350 initialization
42 MOV #PIDRIV, PIDPTR ; Set up pointer to clock driven PI routine
43 .ENDC ; NE, PROASM
44 ;
45 ; Make entry in device handler table for TT device.
46 ;
47 007532 012737 100040 000000G INIDEV: MOV #R50TT, PNAME ; PERMANENT NAME "TT"
48 007540 012737 000000G 000000G MOV #DI$TT, DVSTAT ; SET DEVICE STATUS FLAGS FOR TT
49 007546 005037 000000G CLR DVFLAG
50 007552 005037 000000G CLR DEVSIZ
51 007556 012737 000002 000000G MOV #2, HANENT ; SET UP HANENT SO HANDLER LOOKS RESIDENT
52 007564 005037 000000G CLR NUMDEV ; IT IS DEVICE # 0
53 ;
54 ; Make device table entry for LD (logical disk) device
55 ;
56 007570 012737 177777 000000G MOV #-1, LDDEVX ; ASSUME LD SUPPORT NOT WANTED
57 007576 105737 000000G TSTB VLDSYS ; IS LD SUPPORT GENNED IN?

```

* * * Initialization done with RT-11 running * * *

```

58 007602 001425          BEQ      6$          ;BR IF NOT
59 007604 062737 000002 000000G  ADD     #2,NUMDEV    ;ONE MORE DEVICE
60 007612 013701 000000G  MOV     NUMDEV,R1    ;GET DEVICE TABLE INDEX
61 007616 010137 000000G  MOV     R1,LDDEVX    ;REMEMBER INDEX NUMBER FOR LD DEVICE
62 007622 012761 045640 000000G  MOV     #R5OLD,PNAME(R1) ;SET DEVICE NAME ("LD")
63 007630 012761 000000C 000000G  MOV     #<DS$DIR+DS$SFN+DS$VSZ+DI$LD>,DVSTAT(R1);SET DEV STATUS FLAGS
64 007636 012761 000000G 000000G  MOV     #DX$EBA,DVFLAG(R1);Say buffers must be on even byte boundaries
65 007644 005061 000000G  CLR     DEVSIZ(R1)
66 007650 012761 000002 000000G  MOV     #2,HANENT(R1) ;SAY HANDLER IS RESIDENT
67
68 ; Make device table entry for CL (communications line) device
69 ;
70 007656 005727 000000G  6$:    TST     #CLTOTL    ;Are there any communications lines?
71 007662 001402          BEQ     8$          ;Br if not
72 007664 004737 014652'  CALL    CLINIT      ;Initialize CL handler
73
74 ; Disable clock interrupts.
75 ;
76 007670 012737 000002 000000 8$:    MOV     #2,@#0      ;LOAD RTI IN LOCATION 0
77 007676 005037 000000G  CLR     @#CLKVEC    ;ATTACH CLOCK INTERRUPT TO 0
78 007702 032737 000000G 000000G  BIT     #CW$PRO,CONFQ2 ;ARE WE RUNNING ON A PRO?
79 007710 001402          BEQ     1$          ;BR IF NOT
80 007712 005037 000230          CLR     @#230       ;380 CLOCK INTERRUPT VECTOR
81
82 ; Set up memory parity control
83 ;
84 007716 004737 017642'  1$:    CALL    PARSET     ;SET UP MEMORY PARITY CONTROL
85
86 ; Determine how much memory is installed on machine
87 ;
88 007722 004737 016542'  CALL    MEMTST      ;FIND OUT HOW MUCH PHYSICAL MEMORY THERE IS
89
90 ; Set up information about the size of the job context area
91 ;
92 007726 004737 017172'  CALL    CXTALC      ;Determine size of job context area
93
94 ; Load TSX-Plus device handlers that go in low memory
95 ;
96 007732 004737 017730'  CALL    GETHNL      ;Load low memory handlers
97
98 ; Reserve space for interrupt vector intercept routines for mapped handlers
99 ;
100 007736 010537 000122'  MOV     R5,XMVBAS   ;Save address of base of area for XM vectors
101 007742 013701 000124'  MOV     NMXHAN,R1   ;Get # mapped handlers
102 007746 006301          ASL     R1          ;Reserve room for 2 interrupts per handler
103 007750 062701 000000G  ADD     #NXIVMH,R1  ;Add # requested extra interrupt vectors
104 007754 070127 000032          MUL    #MPIVSZ,R1  ;Calc space needed for interrupt entry code
105 007760 060105          ADD     R1,R5      ;Advance the address of free memory
106
107 ; Set up device index number and unit number for "SY:" device.
108 ;
109 007762 004737 027324'  CALL    SETSY       ;SET UP INFO ABOUT SY DEVICE
110
111 ; Open channel to TSKMON and set up information about it.
112 ;
113 007766 004737 014350'  CALL    OPNKMN      ;OPEN CHANNEL TO TSKMON
114

```

* * * Initialization done with RT-11 running * * *

```

115          ; Set up information about the IND program
116          ;
117 007772 004737 015316'          CALL    INDINI          ; INITIALIZE FOR IND PROGRAM
118          ;
119          ; Initialize the TSXUCL data file
120          ;
121 007776 004737 016060'          CALL    UCLINI
122          ;
123          ; Set name of device that UCL program is to be run from
124          ;
125 010002 013737 000000G 000000G      MOV     SYNAME,UCLNAM      ; SET DEVICE NAME FOR UCL PROGRAM
126          ;
127          ; Initialize spooling system
128          ;
129 010010 004737 011730'          CALL    SPLINI          ; INITIALIZE SPOOLING SYSTEM
130          ;
131          ; Open system swap file
132          ;
133 010014 105737 000000G          TSTB   VSWPFL            ; IS JOB SWAPPING ALLOWED?
134 010020 001402          BEQ     3$                ; BR IF NOT
135 010022 004737 010772'          CALL    OPNSWP            ; OPEN THE SYSTEM SWAP FILE
136          ;
137          ; Open swap file used for PLAS regions
138          ;
139 010026 004737 011424' 3$:    CALL    OPNRSF            ; Open PLAS region swap file
140          ;
141          ; Set up information about which devices need to have their I/O mapped
142          ;
143 010032 004737 023150'          CALL    SETMIO            ; Set up information about mapped I/O
144          ;
145          ; We are finished allocating low-memory buffer space.
146          ;
147 010036 010500          MOV     R5,R0            ; ENSURE WE DON'T OVERFLOW 40KB
148 010040 004737 027630'          CALL    CHKMEM            ; ABORT IF > 40KB OR INTO RT-11
149          ;
150          ; From this point on carry the free memory address in R5
151          ; as a 64-byte block # in physical memory.
152          ;
153 010044 010537 000000G          MOV     R5,UMSYTP        ; SAVE ADDRESS OF NON-EXTENDED SYSTEM TOP
154 010050 062705 000077          ADD     #77,R5          ; BOUND UP TO 64-BYTE BOUNDARY
155 010054 072527 177772          ASH    #-6,R5           ; CONVERT TO 64-BYTE BLOCK #
156 010060 042705 176000          BIC    #176000,R5       ; KILL SIGN EXTENSION
157          ;
158          ; Allocate buffer space that is not constrained to 40Kb TSX-Plus region.
159          ;
160 010064 004737 013744'          CALL    ALBFX            ; ALLOCATE EXTENDED BUFFERS
161          ;
162          ; We will now do some allocation from the top of physical memory downward.
163          ; Save the base of free memory in R4 and get the top of free memory to R5.
164          ;
165 010070 010504          MOV     R5,R4            ; Save the base of free memory in R4
166 010072 010437 000136'          MOV     R4,FMEMLO        ; Save pointer above top of alloc low memory
167 010076 013705 000134'          MOV     FMEMHI,R5        ; Get 64-byte block # of free high memory
168          ;
169          ; Load any mapped system code
170          ;
171 010102 004737 024002'          CALL    GETMAP            ; LOAD USR, EMT, MSG, LOCK, SPOOL, etc.

```

* * * Initialization done with RT-11 running * * *

```

172 ;
173 ; Load any shared run-time systems
174 ;
175 010106 005727 000000G TST #NUMRDB ;Do we need to load any shared run-times?
176 010112 001415 BEQ 4$ ;Br if not
177 010114 012701 000000G MOV #RDB,R1 ;Point to 1st run-time descriptor block
178 010120 010502 MOV R5,R2 ;Save initial memory pointer
179 010122 004737 025464' 5$: CALL GETSRT ;Load a shared run-time system
180 010126 062701 000000G ADD #RT#$SZ,R1 ;Point to next shared run-time descriptor
181 010132 020127 000000G CMP R1,#RDBEND ;Are there more to load?
182 010136 103771 BLO 5$ ;Br if yes
183 010140 160502 SUB R5,R2 ;Compute amt of space used by run-times
184 010142 010237 000000G MOV R2,SRTSIZ ;Save total run-time size
185 010146 4$:
186 .IF NE,PROASM
187 ;
188 ; If we are running on a Pro, load the PI handler like a shared run-time
189 ;
190 TSTB PROFLG ;Are we running on a Pro?
191 BEQ 10$ ;Br if not
192 MOV #PISRT,R1 ;Point to dummy shared run-time block for PI
193 MOV R5,R2 ;Save current memory pointer
194 CALL GETSRT ;Load PI handler like a shared run-time
195 SUB R5,R2 ;Calculate amt of space used by PI handler
196 ADD R2,MHNSIZ ;Count in mapped-handler size
197 .ENDC ;NE,PROASM
198 ;
199 ; Load any mapped handlers
200 ;
201 010146 004737 021244' 10$: CALL GETHND ;Load mapped handlers
202 ;
203 ; Allocate space for data cache buffers and control tables
204 ;
205 010152 004737 026076' CALL CSHBUF ;Allocate space for data cache
206 ;
207 ; We have finished allocating all of the memory used by the system.
208 ; Allocate and initialize a memory map table that will be used to
209 ; show which pages are available for user jobs.
210 ;
211 010156 004737 017332' CALL MAPALC ;Allocate memory map table
212 ;
213 ; Set up info about maximum memory space available to jobs
214 ;
215 010162 004737 017516' CALL SETJSZ ;SET JOB SIZE INFO
216 ;
217 ; Set up date and time
218 ;
219 010166 013702 000000G MOV SYTIML,R2 ;Save time we got at start of init
220 010172 .GTIM #AREA,#SYTIMH ;SET TIME OF DAY
221 010212 .DATE ;GET DATE
222 010220 010037 000000G MOV R0,SYSDAT ;SET SYSTEM DATE
223 010224 020237 000000G CMP R2,SYTIML ;Make sure some time has elapsed
224 010230 001010 BNE 11$ ;Br if clock is running
225 010232 .PRINT #TSXHD ;Print error message heading
226 010240 .PRINT #NOCLOK ;Print clock-not-working message
227 010246 000137 004102' JMP INISTP ;Abort initialization
228 ;

```

*** Initialization done with RT-11 running ***

```

229      ; Unlock the USR so that TSEMT will be swapped back in.
230      ;
231 010252 11#      .UNLOCK          ;RELEASE USR
232      ;
233      ; Read back into memory that part of the resident portion of TSX
234      ; that we overlaid with our work buffer.
235      ;
236 010254 013702 000154'      MOV      WRKSIZ,R2      ;Get size of work buffer
237 010260 006202          ASR      R2          ;Convert to # words
238 010262 013703 000152'      MOV      WRKBUF,R3      ;Get address of work buffer area
239 010266 000241          CLC          ;Convert to block # in TSX.SAV file
240 010270 006003          ROR      R3
241 010272 000303          SWAB     R3
242 010274          .READW #AREA,#17,WRKBUF,R2,R3 ;Read back TSX over work buffer
243      ;
244      ; See if user requested control-C abort
245      ;
246 010330 004737 027670'      CALL     CCATST          ;JUMP TO INISTP IF ^C^C BEFORE THIS POINT
247      ;
248      ; Jump to code at end of TSINIT which takes over control from RT-11
249      ;
250 010334 000137 003420'      JMP      TAKOVR

```

*** Subroutines ***

```

1           .SBTTL *** Subroutines ***
2           .SBTTL ALCWRK -- Allocate a work buffer
3           ;-----
4           ; Allocate a 2048. byte work buffer over a resident portion of TSX.
5           ; This area will be restored from the TSX.SAV disk file after we
6           ; are finished using the work area.
7           ;
8           ; Outputs:
9           ;   WRKBUF = Address of base of work buffer.
10          ;   WRKSIZ = Size of work buffer (2048).
11          ;
12 010340 010246 ALCWRK: MOV      R2,-(SP)
13          ;
14          ; Get address of start of area where buffer can go and then bound
15          ; up to a block boundary.
16          ;
17 010342 012702 000000G      MOV      #EXCBUF,R2      ;Get address of base of buffer area
18 010346 062702 000777      ADD      #777,R2      ;Bound up to block boundary
19 010352 042702 000777      BIC      #777,R2      ;Set to block boundary
20 010356 010237 000152'      MOV      R2,WRKBUF
21 010362 012737 004000 000154'  MOV      #2048.,WRKSIZ
22          ;
23          ; Finished
24          ;
25 010370 012602      MOV      (SP)+,R2
26 010372 000207      RETURN

```

ALCHRB -- Allocate Region Control Blocks for handlers

```

1          .SBTTL  ALCHRB -- Allocate Region Control Blocks for handlers
2          ;-----
3          ; This routine allocates and initializes empty Region Control Blocks for
4          ; use by device handlers.  Handler XM regions not supported on Pro/TSX-Plus.
5          ;
6          .IF      NE,<PROASM-1>      ;Only for 11's
7          ;
8          ; Inputs:
9          ; R5 = Pointer to start of memory area where RCB's are to be built.
10         ;
11         ; Outputs:
12         ; R5 = Pointer past end of RCB area.
13         ;
14 010374  010246  ALCHRB: MOV      R2,-(SP)
15         ;
16         ; Get count of # RCB's to build
17         ;
18 010376  013700  000000G      MOV      NDVRCB,R0      ;Get # RCB's to build for handlers
19         ;
20         ; Store pointer to start of RCB area and store -1 at beginning of area
21         ;
22 010402  010537  000000G      MOV      R5,HANRCB      ;Start of RCB area
23 010406  010537  000000G      MOV      R5,HANRC0      ;Store offset relative to MONVEC
24 010412  162737  000000G 000000G  SUB      #MONVEC,HANRC0      ;Convert address to offset
25 010420  012725  177777      MOV      #-1,(R5)+      ;Store -1 at start of area
26         ;
27         ; Allocate and initialize to zero the RCB's
28         ;
29 010424  012702  000005      1$:      MOV      #5,R2      ;Each RCB has 5 words
30 010430  005025      2$:      CLR      (R5)+      ;Zero the RCB
31 010432  077202      SOB      R2,2$
32         ;
33         ; See if there are more RCB's to build
34         ;
35 010434  005300      DEC      R0      ;More RCB's to initialize?
36 010436  003372      BGT      1$      ;Loop if yes
37         ;
38         ; Store -1 at end of RCB area
39         ;
40 010440  012725  177777      MOV      #-1,(R5)+      ;Mark end of RCB list
41         ;
42         ; Finished
43         ;
44 010444  012602      MOV      (SP)+,R2
45 010446  000207      RETURN
46         ;
47         .IFF      NE,<PROASM-1>      ;Dummy for Pro-only
48 ALCHRB: RETURN
49         .ENDC      NE,<PROASM-1>

```

ALCHRB -- Allocate Region Control Blocks for handlers

```

1          .IF      NE,<PROASM-1>    ;If not assembling for Pro only
2          .SBTTL  LINCHK -- Check validity of T/S line
3          ;-----
4          ; LINCHK is called to check the validity of specified T/S line
5          ; vector and status register addresses.
6          ; If an uninstalled line is detected this routine aborts if
7          ; INIABT=1 or sets the $DEAD flag for the line if INIABT=0.
8          ;
9 010450   010146   LINCHK: MOV     R1,-(SP)
10 010452  010246   MOV     R2,-(SP)
11 010454  010346   MOV     R3,-(SP)
12 010456  010446   MOV     R4,-(SP)
13 010460  013746  000004   MOV     @#4,-(SP)    ;SAVE ORIGINAL TRAP VECTOR
14          ;
15          ; Take over trap control
16          ;
17 010464  012737  010664' 000004   MOV     #6@,#4      ;CATCH TRAPS
18          ;
19          ; Loop through the test for each line.
20          ;
21 010472  012701  000000G   MOV     #LSTLIN,R1   ;NUMBER OF LAST LINE
22          ;
23          ; Determine if this is a primary line or an I/O line and set the
24          ; addresses of the interrupt service routines.
25          ;
26 010476  004737  006214'   1$:    CALL    LINTYP    ;Determine the type of this line
27 010502  032761  000000G 000000G   BIT     #$HARD,LSW3(R1) ;Is this line connected to hardware?
28 010510  001440   BEQ     31$          ;Br if not
29 010512  016103  000000G   MOV     LMXNUM(R1),R3 ;IS THIS A DL-11 OR MULTIPLEXER LINE?
30 010516  001411   BEQ     2$          ;BR IF DL-11
31 010520  016302  000000G   MOV     MXCSR(R3),R2  ;GET DZ11 OR DH11 STATUS REGISTER ADDRESS
32 010524  001403   BEQ     11$         ;BR IF ALREADY MARKED AS DEAD
33 010526  016304  000000G   MOV     MXVEC(R3),R4  ;GET MUX INTERRUPT VECTOR ADDRESS
34 010532  000407   BR      3$          ;
35 010534  004737  010636'   11$:   CALL    4$          ;MARK LINE AS DEAD
36 010540  000424   BR      31$         ;CONTINUE CHECKING TERMINALS
37 010542  016102  000000G   2$:   MOV     RSR(R1),R2  ;GET DL-11 STATUS REGISTER ADDRESS
38 010546  016104  000000G   MOV     INVEC(R1),R4  ;GET DL-11 INTERRUPT VECTOR ADDRESS
39          ; Check validity of status register address.
40 010552  020227  160000   3$:   CMP     R2,#160000   ;IS IT IN I/O PAGE?
41 010556  103445   BLO    LINTRP        ;ERROR IF NOT
42 010560  032702  000007   BIT     #7,R2        ;IS IT ON 8-BYTE BOUNDARY?
43 010564  001042   BNE    LINTRP        ;ERROR IF NOT
44 010566  005712   TST    @R2          ;TRY TO ACCESS IT AND SEE IF WE TRAP
45          ; Check validty of interrupt vector address.
46 010570  020427  000060   CMP     R4,#60       ;CAN'T BE BELOW 60
47 010574  103445   BLO    BADVEC        ;
48 010576  020427  000500   CMP     R4,#500      ;OR ABOVE 500
49 010602  103042   BHIS   BADVEC        ;
50 010604  032704  000007   BIT     #7,R4        ;MUST BE ON 8-BYTE BOUNDARY
51 010610  001037   BNE    BADVEC        ;
52          ; This line looks good. Check next.
53 010612  162701  000002   31$:  SUB     #2,R1        ;MORE TO CHECK?
54 010616  003327   BGT    1$           ;BR IF YES
55          ;
56          ; Finished -- all lines look ok.
57          ;

```

LINCHK -- Check validity of T/S line

```

58 010620 012637 000004      MOV      (SP)+,@#4      ;RESTORE TRAP VECTOR
59 010624 012604      MOV      (SP)+,R4
60 010626 012603      MOV      (SP)+,R3
61 010630 012602      MOV      (SP)+,R2
62 010632 012601      MOV      (SP)+,R1
63 010634 000207      RETURN
64
65      ; See if we should abort or just mark the line as dead.
66      ;
67 010636 105737 000000G    4$:      TSTB     VINABT      ;DOES HE WANT TO ABORT?
68 010642 001013      BNE     LINTRP      ;YES
69 010644 052761 000000G 000000G    BIS     #$DEAD,LSW3(R1) ;MARK LINE AS DEAD
70 010652 005703      TST     R3          ;IS THIS A DL11 OR A MUX LINE?
71 010654 001402      BEQ     5$          ;BR IF DL11
72 010656 005063 000000G    CLR     MXCSR(R3)   ;MARK DZ OR DH AS DEAD
73 010662 000207      5$:      RETURN
74
75      ;
76      ; Trap occured while trying to access status register.
77      ;
78 010664 004737 010636'    6$:      CALL     4$          ;REPORT ERROR OR MARK AS DEAD LINE
79 010670 000002      RTI              ;RETURN TO LINE CHECKING
80      ;
81      ; Error: Invalid status register address.
82      ; R1 = Line number, R2 = status register address
83      ;
84 010672      LINTRP: .PRINT  #TSXHD      ;PRINT ERROR MESSAGE
85 010700      .PRINT  #BADLIN
86 010706 000407      BR      ERP
87      ;
88      ; Error: Invalid interrupt vector address.
89      ; R1 = Line number, R4 = interrupt vector address
90      ;
91 010710      BADVEC: .PRINT  #TSXHD      ;PRINT ERROR MESSAGE
92 010716      .PRINT  #BDVMSG
93 010724 010402      MOV     R4,R2      ;GET VECTOR ADDRESS TO R2
94 010726 010200      ERP:    MOV     R2,R0 ;GET ADDRESS TO R0
95 010730 004737 027704'    CALL   PRTOCT      ;PRINT OCTAL VALUE
96 010734      .PRINT  #CRLF
97 010742      .PRINT  #BDLMSG      ;LINE # =
98 010750 010100      MOV     R1,R0      ;GET LINE NUMBER
99 010752 006200      ASR    R0          ;# 1
100 010754 004737 027754'    CALL   PRTDEC      ;PRINT LINE NUMBER
101 010760      .PRINT  #CRLF
102 010766 000137 004102'    JMP     INISTP      ;ABORT INITIALIZATION
103      .ENDC      ;NE, <PROASM-1>

```

```

1          .SBTTL  OPNSWP  -- Open system swap file
2          ;-----
3          ; OPNSWP is called to open the TSX job swap file.
4          ; It also assigns swap file slots for each line.
5          ;
6          ; Inputs:
7          ;   R5 = Address of base of free memory region
8          ;
9          ; Outputs:
10         ;   SWPCHN = Set up for access to swap file.
11         ;   LSWPBK(i) = Starting block number in swap file for swap area for line.
12         ;
13 010772 010146 OPNSWP: MOV      R1, -(SP)
14 010774 010246      MOV      R2, -(SP)
15 010776 010346      MOV      R3, -(SP)
16         ;
17         ; Load RT-11 handler for swap device.
18         ;
19 011000 013700 0000000      MOV      SWDBLK, R0      ;Get name of device
20 011004 004737 027514'     CALL     RTFTCH      ;Fetch the RT-11 handler
21 011010 103546             BCS     11$      ;Br if invalid device
22         ;
23         ; Compute the maximum number of slots in swap file that we could need
24         ;
25 011012 012703 0000000     MOV      #NSL+NDL, R3      ;Get # virtual lines and detached jobs
26 011016 012701 0000000     MOV      #LSTPL, R1      ;Get index to last primary line
27 011022 032761 0000000 0000000 1$: BIT     #$DEAD, LSW3(R1) ;Is this line installed?
28 011030 001001             BNE     5$      ;Br if not
29 011032 005203             INC     R3      ;Count another primary line
30 011034 162701 0000002     5$: SUB     #2, R1      ;Get next line index
31 011040 003370             BGT     1$      ;Loop if more lines to check
32 011042 020337 0000000     CMP     R3, VSWPSL      ;Compare with # slots specified
33 011046 002002             BGE     6$      ;Br if VSWPSL value is ok
34 011050 010337 0000000     MOV     R3, VSWPSL      ;Reduce number of slots in swap file
35         ;
36         ; Determine how many blocks are needed for each slot in swap file.
37         ;
38 011054 013703 0000000 6$: MOV     VHIMEM, R3      ;GET # BLOCKS NEEDED FOR LARGEST JOB SIZE
39 011060 006303             ASL     R3
40 011062 063703 0000000     ADD     CXPAG, R3      ;ADD # BLOCKS NEEDED FOR JOB CONTEXT AREA
41 011066 010337 0000000     MOV     R3, SLTSIZ      ;Save size of swap file slot
42         ;
43         ; Compute the total number of blocks needed for the swap file.
44         ;
45 011072 070337 0000000     MUL     VSWPSL, R3      ;Multiply by # slots in swap file
46         ;
47         ; R3 now contains total number of blocks needed in swap file.
48         ; See if swap file already exists on disk.
49         ;
50 011076 4$: .LOOKUP #AREA, #1, #SWDBLK; DOES SWAP FILE EXIST NOW?
51 011116 103415             BCS     2$      ;BR IF NOT
52         ; Swap file exists. See if it is the right size.
53 011120 020003             CMP     R0, R3      ;IS SWAP FILE THE RIGHT SIZE?
54 011122 001453             BEQ     3$      ;BR IF YES
55         ; Old swap file is not of correct size.
56         ; Delete it and open a new swap file.
57 011124             .CLOSE #1

```

OPNSWP -- Open system swap file

```

58 011132          .DELETE #AREA,#1,#SWDBLK;DELETE THE OLD SWAP FILE
59
60                ; Create a new swap file.
61
62 011152          2#: .ENTER #AREA,#1,#SWDBLK,R3 ;CREATE A NEW SWAP FILE
63 011176 103443   BCS      9#                ;BR IF SOME ERROR ON OPEN
64
65                ; Swap file has been created.
66                ; Write to last block to reserve full space in file then close
67                ; and reopen the channel using a .lookup.
68
69 011200 005303   DEC      R3                ;GET # OF LAST BLOCK IN FILE
70 011202          .WRITW #AREA,#1,#TSINIT,#256.,R3 ;WRITE TO LAST BLOCK IN FILE
71 011240 005203   INC      R3                ;GET BACK # BLOCKS IN FILE
72 011242          .CLOSE #1                ;CLOSE FILE WE CREATED
73 011250 000712   BR      4#                ;NOW GO REOPEN USING A .LOOKUP
74
75                ; Swap file has been successfully opened using a .lookup.
76                ; Now copy channel status to TSX swap channel.
77
78 011252 012700 000000G 3#:  MOV      #SWPCHN,R0          ;POINT TO SWAP CHANNEL BLOCK
79 011256 013702 000000G   MOV      SWDBLK,R2          ;GET DEVICE NAME
80 011262 004737 027134'   CALL     SETCHN           ;SET UP SWAP CHANNEL INFO
81
82                ; Release the RT-11 device handler
83
84 011266          .RELEAS #SWDBLK          ;Release RT-11 device handler
85
86                ; Finished
87
88 011276 012603   MOV      (SP)+,R3
89 011300 012602   MOV      (SP)+,R2
90 011302 012601   MOV      (SP)+,R1
91 011304 000207   RETURN
92
93                ; Error: Cannot open swap file
94
95 011306          9#: .PRINT #TSXHD          ;PRINT ERROR MESSAGE
96 011314          .PRINT #BADOPN
97 011322 004737 011350'   CALL     SPNEED          ;Print info about number of blocks needed
98
99                ; Error: Invalid device specification.
100
101 011326 010001   11#:  MOV      R0,R1          ;Save device name
102 011330          .PRINT #TSXHD          ;Print error message
103 011336          .PRINT #BADOPN
104 011344 004737 011376'   CALL     BADDEV          ;Print invalid device specification
105
106                ; Error: Number of contiguous blocks required.
107
108 011350          SPNEED: .PRINT #CONSPC   ;Print contiguous blocks needed
109 011356 010300   MOV      R3,R0          ;GET # BLOCKS NEEDED FOR FILE
110 011360 004737 027754'   CALL     PRTDEC          ;DISPLAY # BLOCKS NEEDED
111 011364          .PRINT #CRLF
112 011372 000137 004102'   JMP      INISTP          ;ABORT INITIALIZATION
113
114                ; Bad file specification.

```

OPNSWP -- Open system swap file

115

116 011376

117 011404 010100

118 011406 004737 030020'

119 011412

120 011420 000137 004102'

BADDEV: .PRINT #CFHMSG

MOV R1,R0

CALL PRTR50

.PRINT #CRLF

JMP INISTP

;Print invalid device specification

;Get the rad50 device name

;Print rad50 device name

;Print carriage return/line feed

;Abort initialization

OPNRSF -- Open PLAS region swap file

```

1                                     .SBTTL  OPNRSF -- Open PLAS region swap file
2                                     -----
3                                     ; OPNRSF is called to open the swap file used for PLAS regions.
4                                     ;
5                                     ; Inputs:
6                                     ;   R5 = Address of base of free memory area.
7                                     ;
8                                     ; Outputs:
9                                     ;   SEGCHN = Set up to access swap file.
10                                    ;
11 011424 010346 OPNRSF: MOV      R3, -(SP)
12                                    ;
13                                    ; Return if this is a non-swapping system or if region swap file is
14                                    ; not wanted.
15                                    ;
16 011426 105737 000000G          TSTB   VSWPFL          ; Is this a non-swapping system?
17 011432 001513                  BEQ    9$              ; Br if yes
18 011434 005737 000000G          TST   VPLAS           ; Is a PLAS swap file wanted?
19 011440 001510                  BEQ    9$              ; Br if not
20                                    ;
21                                    ; Load RT-11 device handler for swap device
22                                    ;
23 011442 013700 000000G          MOV    RSFBLK, R0        ; Get name of device
24 011446 004737 027514'          CALL  RTFTCH         ; Try to fetch the RT-11 device handler
25 011452 103515                  BCS   11$           ; Br if error on handler fetch
26 011454 013703 000000G          MOV    VPLAS, R3       ; Get # blocks in PLAS swap file
27                                    ;
28                                    ; See if PLAS swap file already exists on disk
29                                    ;
30 011460          4$: .LOOKUP #AREA, #1, #RSFBLK ; Try to find existing PLAS swap file
31 011500 103416          BCS   2$              ; Br if file does not now exist
32                                    ;
33                                    ; PLAS swap file exists.
34                                    ; See if it is the correct size.
35                                    ;
36 011502 020037 000000G          CMP    R0, VPLAS      ; Is swap file of the correct size?
37 011506 001453          BEQ    3$              ; Br if yes
38                                    ;
39                                    ; Old PLAS swap file is not of correct size.
40                                    ; Delete it and open a new swap file.
41                                    ;
42 011510          .CLOSE #1              ; Close and delete the old file
43 011516          .DELETE #AREA, #1, #RSFBLK ; Delete the old file
44                                    ;
45                                    ; Create new swap file
46                                    ;
47 011536          2$: .ENTER #AREA, #1, #RSFBLK, VPLAS ; Create a new PLAS swap file
48 011564 103440          BCS   10$           ; Br if cannot create new file
49                                    ;
50                                    ; New swap file has been created.
51                                    ; Write to last block to reserve full file size
52                                    ; and then close and reopen with a lookup.
53                                    ;
54 011566 005303          DEC    R3              ; Get # of last block in file
55 011570          .WRITW #AREA, #1, #TSINIT, #256, R3
56 011626          .CLOSE #1
57 011634 000711          BR    4$              ; Go back and lookup file

```

OPNRSF -- Open PLAS region swap file

```

58 ;
59 ; Swap file has been successfully opened using lookup.
60 ; Copy channel status to TSX channel block.
61 ;
62 011636 012700 0000009 3$: MOV #SEGCHN,R0 ;Point to TSX PLAS swap channel
63 011642 013703 0000009 MOV RSFBLK,R3 ;Get device name
64 011646 004737 027134' CALL SETCHN ;Set up TSX channel block
65 ;
66 ; Release the RT-11 device handler
67 ;
68 011652 .RELEAS #RSFBLK ;Release RT-11 device handler
69 ;
70 ; Finished
71 ;
72 011662 012603 9$: MOV (SP)+,R3
73 011664 000207 RETURN
74 ;
75 ; Error -- Cannot open PLAS swap file
76 ;
77 011666 10$: .PRINT #TSXHD ;Print error prefix
78 011674 .PRINT #RSFERR ;Print error message
79 011702 004737 011350' CALL SPNEED ;Print information about amt of space needed
80 ;
81 ; Error: Invalid device specification.
82 ;
83 011706 010001 11$: MOV R0,R1 ;Save device name
84 011710 .PRINT #TSXHD ;Print error message
85 011716 .PRINT #RSFERR
86 011724 004737 011376' CALL BADDEV ;Print invalid device specification

```

SPLINI -- Initialize spooling system

```

1          .SBTTL  SPLINI -- Initialize spooling system
2          ;-----
3          ; SPLINI performs the initialization of the spooling system.
4          ; Inputs:
5          ; R5 = Current base of free memory area.
6          ;
7 011730 005727 000000G SPLINI: TST      #SPLND      ;Are there any spooled devices?
8 011734 001401          BEQ      13$          ;Br if not
9 011736 000401          BR       10$          ;Initialize the spooled devices
10 011740 000207          13$:  RETURN
11          ;
12          ; There are some spooled devices
13          ;
14 011742 010146          10$:  MOV      R1,-(SP)
15 011744 010246          MOV      R2,-(SP)
16 011746 010346          MOV      R3,-(SP)
17 011750 010446          MOV      R4,-(SP)
18 011752 010546          MOV      R5,-(SP)
19          ;
20          ; Open each spooled device
21          ;
22 011754 105037 000000G          CLRB   NSPLDV      ;INIT COUNT OF # ACTUAL SPOOLED DEVICES
23 011760 012701 000000G          MOV    #SDCB,R1      ;POINT TO 1ST SDCB
24 011764 012703 000000G          MOV    #SPLDEV,R3     ;POINT TO TABLE OF RAD50 DEV NAMES
25 011770 012704 000000G          MOV    #SPLANM,R4     ;POINT TO TABLE OF ASCII DEV NAMES
26 011774 004737 013002'          2$:  CALL  FORCE0      ;FORCE UNIDENTIFIED UNIT #S TO 0
27 012000 011302          MOV    (R3),R2        ;GET RAD50 NAME OF SPOOLED DEVICE
28 012002 010261 000000G          MOV    R2,SDNAME(R1) ;SET NAME IN SDCB
29 012006 010100          MOV    R1,R0          ;GET ADDRESS OF SDCB
30 012010 062700 000000G          ADD   #SDANAM,R0     ;POINT TO CELL FOR ASCII NAME
31 012014 112420          MOVB  (R4)+,(R0)+    ;MOVE IN ASCII DEVICE NAME
32 012016 112420          MOVB  (R4)+,(R0)+
33 012020 112420          MOVB  (R4)+,(R0)+
34 012022 020227 000000G          CMP   R2,#DMYDEV     ;Is this a dummy entry for later patching?
35 012026 001451          BEQ   1$              ;Br if yes -- Ignore it
36 012030 010200          MOV   R2,R0          ;Get name to R0
37 012032 004737 012704'          CALL  CVTDVU         ;Convert name to device # and unit #
38 012036 010061 000000G          MOV   R0,SDDVU(R1)   ;Store device # and unit # in SDCB
39 012042 010200          MOV   R2,R0          ;Get device name
40 012044 004737 012600'          CALL  CHKCLD         ;See if this is a CL device?
41 012050 103406          BCS   14$            ;Br if not
42 012052 004737 012514'          CALL  SPLCLD         ;Set up for spooling to CL device
43 012056 103414          BCS   3$              ;Br if invalid unit
44 012060 105237 000000G          INCB  NSPLDV         ;Count # of actual spooled devices
45 012064 000432          BR    1$              ;Process next device
46 012066 010100          14$:  MOV   R1,R0      ;Get address of SDCB
47 012070 062700 000000G          ADD   #SDCHAN,R0     ;Point to channel block within SDCB
48 012074 004737 027046'          CALL  OPNCHN         ;Set TSX-Plus channel block open to device
49 012100 103403          BCS   3$              ;Br if did not recognize device
50 012102 105237 000000G          INCB  NSPLDV         ;Count # actual spooled devices
51 012106 000421          BR    1$              ;GO PROCESS NEXT DEVICE
52          ;
53          ; Error on opening spooled device
54          ; Determine if we should print an error message or simply
55          ; mark the spooled device as unavailable.
56          ;
57 012110 012761 000000G 000000G 3$:  MOV    #DMYDEV,SDNAME(R1); SAY THIS DEVICE IS NOT SPOOLED

```

SPLINI -- Initialize spooling system

```

58 012116 105737 000000G          TSTB    VINABT          ;ABORT OR CONTINUE ON ERRORS?
59 012122 001413                   BEQ     1$              ;BR TO IGNORE DEVICE AND CONTINUE INIT
60 012124                   .PRINT #TSXHD          ;PRINT ERROR MESSAGE
61 012132                   .PRINT #BDSPOP
62 012140 010200                   MOV     R2,R0          ;GET RAD50 DEVICE NAME
63 012142 004737 030020'          CALL    PRTR50         ;PRINT DEVICE NAME
64 012146 000137 004102'          JMP     INISTP         ;ABORT INITIALIZATION
65                               ;
66                               ;   Process next spooled device
67                               ;
68 012152 062701 000000G          1$:    ADD     #SDCBSZ,R1 ;POINT TO NEXT SDCB
69 012156 005723                   TST     (R3)+          ;POINT TO NEXT DEVICE NAME
70 012160 020327 000000G          CMP     R3,#SPLDVN    ;OPENED ALL SPOOLED DEVICES?
71 012164 103703                   BLD     2$              ;BR IF MORE TO DO
72                               ;
73                               ;   Open the spool file
74                               ;
75 012166 105737 000000G          TSTB    NSPLDV         ;ARE THERE ANY ACTUAL SPOOLED DEVICES?
76 012172 001521                   BEQ     12$            ;BR IF THERE ARE NO ACTUAL SPOOLED DEVICES
77 012174 013700 000000G          MOV     SPLBLK,R0     ;Get name of device for spool file
78 012200 004737 027514'          CALL    RTFTCH        ;Fetch the RT-11 device handler
79 012204 103532                   BCS     11$            ;Br if cannot fetch handler
80 012206 013702 000000G          MOV     NSPLBL,R2    ;GET # BLOCKS TO ALLOCATE FOR FILE
81 012212 005202                   INC     R2              ;Add 1 extra block
82                               ;
83                               ;   See if spool file already exists
84                               ;
85 012214                   5$:    .LOOKUP #AREA,#1,#SPLBLK;SEE IF SPOOL FILE ALREADY EXISTS
86 012234 103415                   BCS     6$              ;BR IF IT DOES NOT EXIST
87 012236 020002                   CMP     R0,R2          ;IS IT THE RIGHT SIZE?
88 012240 001453                   BEQ     7$              ;BR IF YES
89 012242                   .CLOSE #1              ;IT EXISTS BUT IS OF WRONG SIZE
90 012250                   .DELETE #AREA,#1,#SPLBLK;DELETE CURRENT FILE AND OPEN NEW ONE
91                               ;
92                               ;   Open new spool file
93                               ;
94 012270                   6$:    .ENTER #AREA,#1,#SPLBLK,R2;CREATE A NEW SPOOL FILE
95 012314 103456                   BCS     8$              ;BR IF ERROR ON ENTER
96                               ;   Write to last block in file to reserve full file space
97 012316 010203                   MOV     R2,R3          ;Get # of blocks in file
98 012320 005303                   DEC     R3              ;Get # of last block in file
99 012322                   .WRITW #AREA,#1,#TSINIT,#256.,R3
100                               ;   Now close and reopen using a lookup
101 012360                   .CLOSE #1              ;CLOSE SPOOL FILE
102 012366 000712                   BR     5$              ;GO BACK AND REOPEN USING LOOKUP
103                               ;
104                               ;   Spool file has been successfully opened with a lookup.
105                               ;   Save the channel status
106                               ;
107 012370 012700 000000G          7$:    MOV     #SPLCHN,R0 ;SAVE CHANNEL STATUS HERE
108 012374 013702 000000G          MOV     SPLBLK,R2     ;GET DEVICE NAME
109 012400 004737 027134'          CALL    SETCHN        ;SAVE CHANNEL STATUS
110 012404                   .RELEAS #SPLBLK       ;Release the RT-11 device handler
111                               ;
112                               ;   Set number of free public blocks in spool file
113                               ;
114 012414 113703 000000G          MOVVB  NSPLDV,R3      ;Get # spooled devices

```

SPLINI -- Initialize spooling system

```
115 012420 070327 0000000          MUL      #PVSPBL,R3      ;Times number of private blocks per dev
116 012424 005403                   NEG      R3
117 012426 063703 0000000          ADD      NSPLBL,R3      ;Get # public spool blocks
118 012432 010337 0000000          MOV      R3,NFRESB     ;This is number of public free spool blocks
119                                     ;
120                                     ; Finished
121                                     ;
122 012436 012605          12$:     MOV      (SP)+,R5
123 012440 012604          MOV      (SP)+,R4
124 012442 012603          MOV      (SP)+,R3
125 012444 012602          MOV      (SP)+,R2
126 012446 012601          MOV      (SP)+,R1
127 012450 000207          9$:     RETURN
128                                     ;
129                                     ; Error: Cannot open spool file.
130                                     ;
131 012452                   8$:     .PRINT  #TSXHD      ;PRINT ERROR MESSAGE
132 012460                   .PRINT  #BOSF
133 012466 000137 004102'        JMP      INISTP        ;ABORT INITIALIZATION
134                                     ;
135                                     ; Error: Invalid device specification.
136                                     ;
137 012472 010001          11$:     MOV      R0,R1      ;Save device name
138 012474                   .PRINT  #TSXHD      ;Print error message
139 012502                   .PRINT  #BOSF
140 012510 004737 011376'        CALL   BADDEV        ;Print invalid device specification
```

SPLCLD -- Set up spooling to a CL device

```

1          .SBTTL  SPLCLD -- Set up spooling to a CL device
2          ;-----
3          ; SPLCLD is called to set up a spool device control block when
4          ; spooling is being directed to a Communication Line (CL) device.
5          ;
6          ; Inputs:
7          ;   R0 = CL unit number
8          ;   R1 = Address of SDCB
9          ;
10         ; Outputs:
11         ;   C-flag set ==> Invalid CL unit
12         ;
13 012514 010546 SPLCLD: MOV      R5, -(SP)
14         ;
15         ; Make sure CL unit number is valie
16         ;
17 012516 010005         MOV      R0, R5          ;Get CL unit number
18 012520 020527 000000G CMP      R5, #CLTOTL      ;Is this a valid unit #
19 012524 103022         BHIS     8$             ;Br if invalid
20         ;
21         ; Set up channel control block in SDCB
22         ;
23 012526 005061 000000C CLR      SDCHAN+C.SBLK(R1); Starting block # = 0
24 012532 020527 000007  CMP      R5, #7          ;Is this a CL or C1 unit?
25 012536 101405         BLOS     1$             ;Br if CL unit
26 012540 162705 000010  SUB      #8, R5          ;Remove C1 unit # bias
27 012544 013700 000000G MOV      C1DEVX, R0      ;Get C1 device index number
28 012550 000402         BR       2$
29 012552 013700 000000G 1$: MOV      CLDEVX, R0      ;Get CL device index number
30 012556 010061 000000C 2$: MOV      R0, SDCHAN+C.CSW(R1); Set device index number
31 012562 110561 000000C MOV      R5, SDCHAN+C.DEVQ(R1); Set unit #
32         ;
33         ; We successfully set up a CL unit
34         ;
35 012566 000241         CLC              ;Signal success on error
36 012570 000401         BR       9$
37         ;
38         ; We cannot open this CL unit
39         ;
40 012572 000261 8$: SEC              ;Signal error
41         ;
42         ; Finished
43         ;
44 012574 012605 9$: MOV      (SP)+, R5
45 012576 000207         RETURN

```

CHKCLD -- See if a device name is a CL or C1 unit

```

1          .SBTTL  CHKCLD -- See if a device name is a CL or C1 unit
2          ;-----
3          ; Determine if a rad50 device and unit name is a CL or C1 device.
4          ;
5          ; Inputs:
6          ;   RO = Rad50 device spec
7          ;
8          ; Outputs:
9          ;   C-flag set      ==> Not a CL or C1 unit
10         ;   C-flag cleared ==> This is a CL or C1 unit
11         ;   RO = CL unit number (0-15)
12         ;
13 012600  CHKCLD:
14         ;
15         ; See if this is a CL unit
16         ;
17 012600 020027 012240      CMP      RO,#R50CL      ; Is name "CL"?
18 012604 001411           BEQ      1$              ; Br if yes
19 012606 020027 012276      CMP      RO,#R50CLO     ; Is name in the range CLO to CL7?
20 012612 103432           BLO      8$              ; Br if not
21 012614 020027 012305      CMP      RO,#R50CL7
22 012620 101005           BHI      2$
23 012622 162700 012276      SUB      #R50CLO,RO      ; Get CL unit number
24 012626 000422           BR       7$
25 012630 005000          1$: CLR      RO              ; CL = CLO
26 012632 000420           BR       7$
27         ;
28         ; See if this is a C1 unit
29         ;
30 012634 020027 013630      2$:  CMP      RO,#R50C1      ; Is name "C1"?
31 012640 001413           BEQ      3$              ; Br if yes
32 012642 020027 013666      CMP      RO,#R50C10     ; Is name in the range C10 to C17?
33 012646 103414           BLO      8$              ; Br if not
34 012650 020027 013675      CMP      RO,#R50C17
35 012654 101011           BHI      8$
36 012656 162700 013666      SUB      #R50C10,RO     ; Get unit number
37 012662 062700 000010      ADD      #8.,RO        ; Bias by 8 for C1 units
38 012666 000402           BR       7$
39 012670 012700 000010      3$:  MOV      #8.,RO        ; C1 = CLB
40         ;
41         ; This is a CL or C1 unit
42         ;
43 012674 000241          7$:  CLC
44 012676 000401           BR       9$              ; Signal success on return
45         ;
46         ; This is not a CL or C1 unit
47         ;
48 012700 000261          8$:  SEC
49         ;                               ; Signal failure on return
50         ;
51         ; Finished
52 012702 000207          9$:  RETURN

```

CVTDVU -- Convert device name to dev index and unit #

```

1          .SBTTL  CVTDVU -- Convert device name to dev index and unit #
2          ;-----
3          ; CVTDVU is called to convert a RAD50 device name into the corresponding
4          ; device index number and unit number.
5          ;
6          ; Inputs:
7          ;   RO = RAD50 device name.
8          ;
9          ; Outputs:
10         ;   C-flag cleared ==> Conversion successful.
11         ;   C-flag set    ==> Unable to find device name in tables.
12         ;   RO = Device index number (low byte), device unit number (high byte).
13         ;
14 012704 010246 CVTDVU: MOV      R2,-(SP)
15 012706 010346      MOV      R3,-(SP)
16         ;
17         ; Split the unit number off of the full device name
18         ;
19 012710 010003      MOV      RO,R3      ;Get full device name
20 012712 005002      CLR      R2        ;Set up for divide
21 012714 071227 000050 DIV      #50,R2      ;Split name and unit (RO=name, R1=unit)
22 012720 005703      TST      R3        ;Was a unit number specified?
23 012722 001402      BEQ      1$        ;Br if not
24 012724 162703 000036 SUB      #36,R3      ;Convert unit number to binary value
25 012730 010300 1$:  MOV      R3,RO      ;Get unit number
26 012732 000300      SWAB     RO        ;Position to high-order byte
27         ;
28         ; Look up the device name to get the device index
29         ;
30 012734 070227 000050 MUL      #50,R2      ;Now get the device name without unit number
31 012740 013702 000000G MOV      NUMDEV,R2   ;Get index number of last device
32 012744 020362 000000G 2$:  CMP      R3,PNAME(R2) ;Search for device in name table
33 012750 001407      BEQ      3$        ;Br if found it
34 012752 162702 000002 SUB      #2,R2      ;Try next device
35 012756 002372      BGE      2$        ;Loop if more to check
36         ;
37         ; Error, cannot find device name in tables
38         ;
39 012760 012700 177777 MOV      #-1,RO     ;Set device # = unit # = -1
40 012764 000261      SEC      R2        ;Signal error on return
41 012766 000402      BR      9$
42         ;
43         ; Found the device in the tables
44         ;
45 012770 050200 3$:  BIS      R2,RO      ;Combine device # and unit #
46 012772 000241      CLC      R2        ;Signal success on return
47         ;
48         ; Finished
49         ;
50 012774 012603 9$:  MOV      (SP)+,R3
51 012776 012602      MOV      (SP)+,R2
52 013000 000207      RETURN

```

FORCE0 -- Force a 2-char dev name to unit 0

```

1                                     .SBTTL  FORCE0 -- Force a 2-char dev name to unit 0
2                                     ;-----
3                                     ; Inputs: R3 points to a RAD50 device name
4                                     ;
5                                     ; Outputs: If the 3rd char of the device name pointed to by R3 is
6                                     ;         blank, then it is changed to 0
7                                     ;
8 013002 010346  FORCE0: MOV      R3, -(SP)
9 013004 010446      MOV      R4, -(SP)
10 013006 010546      MOV      R5, -(SP)
11 013010 011305      MOV      (R3), R5      ; MOVE CURRENT DEV NAME TO R5
12 013012 005004      CLR      R4          ; SET UP FOR DIVIDE
13 013014 071427 000050  DIV     #50, R4      ; SEPARATE INTO NAME AND UNIT
14 013020 005705      TST      R5          ; WAS 3RD CHAR BLANK?
15 013022 001012      BNE     9$          ; RETURN IF NOT
16 013024 010405      MOV      R4, R5      ; GET HIGH 2 CHARS
17 013026 005004      CLR      R4          ; SET UP FOR ANOTHER DIVIDE
18 013030 071427 000050  DIV     #50, R4      ; SEPARATE 1 & 2 CHARS
19 013034 005704      TST      R4          ; WAS CHAR 1 BLANK?
20 013036 001404      BEQ     9$          ; EMPTY OR INVALID DEV NAME!
21 013040 005705      TST      R5          ; WAS CHAR 2 BLANK?
22 013042 001402      BEQ     9$          ; 1-CHAR DEV NAME SHOULD BE INVALID???
23 013044 062713 000036  ADD     #^R 0, (R3)      ; FORCE TO UNIT 0
24 013050 012605 9$:  MOV      (SP)+, R5
25 013052 012604      MOV      (SP)+, R4
26 013054 012603      MOV      (SP)+, R3
27 013056 000207      RETURN

```

ALOCBF -- Allocate buffer space

```

1          .SBTTL  ALOCBF -- Allocate buffer space
2          ;-----
3          ; ALOCBF is called to allocate space for buffers.  The allocated space
4          ; is not initialized but simply reserved.
5          ;
6          ; Inputs:
7          ;   R5 = Start of area to allocate buffer space in.
8          ;
9          ; Outputs:
10         ;   R5 = Address beyond end of buffer area.
11         ;   CHNBAS = Address of base of I/O channel space.
12         ;   CHNEND = Address past end of I/O channel space.
13         ;
14 013060 010146 ALOCBF: MOV      R1,-(SP)
15         ;
16         ; Assign space for I/O queue elements.
17         ;
18 013062 010537 000000G      MOV      R5,FREIOQ      ;START OF I/O QUEUE SPACE
19 013066 062705 000000C      ADD      #IOQSIZ*NUMIOQ,R5;RESERVE SPACE FOR I/O QUEUE ELEMENTS
20         ;
21         ; Assign space for shared PLAS region control blocks
22         ;
23 013072 010537 000000G      MOV      R5,SHRRCB      ;Start of area for RCB's
24 013076 013701 000000G      MOV      VNGR,R1       ;Get number of RCB's wanted
25 013102 020137 000000G      CMP      R1,VMXWIN      ;Must have one for each display window
26 013106 103002              BHIS     13$              ;Br if ok
27 013110 013701 000000G      MOV      VMXWIN,R1     ;Force one for each window
28 013114 070127 000000G 13$:  MUL      #RC##SZ,R1    ;Multiply by size of each block
29 013120 060105              ADD      R1,R5         ;Allocate space for RCB's
30 013122 010537 000000G      MOV      R5,SHRRCN     ;Address of end of region
31         ;
32         ; Assign space for fork blocks
33         ;
34 013126 012700 000000C      MOV      #<NUMFRK-FRKGEN>,R0 ;Get # fork blocks to allocate
35 013132 003404              BLE     11$              ;Br if none to allocate
36 013134 010537 000000G      MOV      R5,FRKINI     ;Set pointer to start of area
37 013140 062705 000000C      ADD      #<<NUMFRK-FRKGEN>*FQ##SZ>,R5 ;Reserve space for fork blocks
38         ;
39         ; Assign space for job monitoring control blocks
40         ;
41 013144 013701 000000G 11$:  MOV      VMXMON,R1     ;Any job monitoring blocks wanted?
42 013150 001405              BEQ     10$              ;Br if not
43 013152 010537 000000G      MOV      R5,MONFQH     ;Start of job monitoring control blocks
44 013156 070127 000000G      MUL      #JM##SZ,R1    ;Compute space needed for control blocks
45 013162 060105              ADD      R1,R5         ;Allocate the space
46         ;
47         ; Allocate space for system message buffers.
48         ;
49 013164 010537 000000G 10$:  MOV      R5,SNMSHD     ;HEAD OF SYSTEM MESSAGE BUFFER AREA
50 013170 062705 000000C      ADD      #<NMSNMB*SB##SZ>,R5;RESERVE ROOM FOR MESSAGE BUFFERS
51         ;
52         ; Allocate space for INSTALLED program table
53         ;
54 013174 010537 000000G      MOV      R5,INSTBL     ;Base of table
55 013200 013701 000000G      MOV      VNUIP,R1     ;# slots for user installed programs
56 013204 062701 000000G      ADD      #NSIP,R1     ;Add # slots for system programs
57 013210 070127 000000G      MUL      #II##SZ,R1   ;Multiply by size of each slot

```

ALOCBF -- Allocate buffer space

```

58 013214 060105          ADD      R1,R5          ;Allocate space for table
59 013216 010537 000000G  MOV      R5,INSTBN      ;Pointer past end of INSTALL table
60
61                      ; Allocate space for device mount entries
62
63 013222 010537 000000G  MOV      R5,CSHDEV      ;Point to start of area
64 013226 012701 000000G  MOV      #CD##SZ,R1     ;Get size of each entry
65 013232 070137 000000G  MUL      VMXCSH,R1      ;Multiply by number of entries
66 013236 060105          ADD      R1,R5          ;Reserve space for table
67 013240 010537 000000G  MOV      R5,CSHDVN      ;Save pointer past end of table
68
69                      ; Allocate space for data cache control blocks
70
71 013244 005737 000000G  TST      CSHALC         ;Is data caching wanted?
72 013250 001404          BEQ      12$            ;Br if not
73 013252 010537 000000G  MOV      R5,CCBHD       ;Head of free list area
74 013256 062705 000000C  ADD      #NUMCCB*CC##SZ,R5 ;Allocate space for control blocks
75
76                      ; Allocate space for spool file control blocks
77
78 013262 013701 000000G  12$:    MOV      NSPLFL,R1   ;Get # spool file control blocks needed
79 013266 001407          BEQ      1$            ;Br if none needed
80 013270 070127 000000G  MUL      #SFCBSZ,R1     ;Compute space needed by control blocks
81 013274 010537 000000G  MOV      R5,SFCB        ;Base of control block area
82 013300 060105          ADD      R1,R5          ;Allocate space for control blocks
83 013302 010537 000000G  MOV      R5,SFCBND      ;End of control block area
84
85                      ; Allocate space for a 16 byte vector for each multiplexer.
86                      ; This vector is used to map from the mux line number to the
87                      ; TSX-Plus logical line number.
88
89 013306 012701 000002    1$:    MOV      #2,R1        ;START WITH FIRST MUX
90 013312 020127 000000G  2$:    CMP      R1,#LSTMX    ;HAVE WE DONE ALL MUX'S?
91 013316 101007          BHI      5$            ;BR IF YES
92 013320 010561 000000G  MOV      R5,MXLNT(R1)   ;SET ADDRESS OF START OF VECTOR
93 013324 062705 000020    ADD      #16.,R5        ;RESERVE SPACE FOR VECTOR
94 013330 062701 000002    ADD      #2,R1          ;ADVANCE TO NEXT MUX
95 013334 000766          BR       2$            ;
96
97                      ; Allocate buffers to hold characters for DMA transfers to DH11 multiplexers
98
99 013336 012701 000002    5$:    MOV      #2,R1        ;Start with first line
100 013342 020127 000000G  6$:    CMP      R1,#LSTPL   ;Is this a primary time-sharing line?
101 013346 101403          BLOS     3$            ;Br if yes
102 013350 020127 000000G  CMP      R1,#FSTIOL     ;Is this a CL line?
103 013354 103422          BLO      7$            ;Br if not
104 013356 026127 000000G 000000G  3$:    CMP      LCDTYP(R1),#CDX$DH ;Is this line connected to a DH11?
105 013364 001404          BEQ      8$            ;Br if yes
106 013366 026127 000000G 000000G  CMP      LCDTYP(R1),#CDX$VH ;Is this line connected to a DHV11?
107 013374 001012          BNE      7$            ;Br if not
108 013376 010561 000000G  8$:    MOV      R5,LDHB1B(R1) ;Set address of start of buffer 1
109 013402 010561 000000G  MOV      R5,LDHB1P(R1) ;Initialize pointer into buffer 1
110 013406 062705 000000G  ADD      #DHBFSZ,R5     ;Reserve space for buffer
111 013412 010561 000000G  MOV      R5,LDHB2B(R1) ;Set address of start of buffer 2
112 013416 062705 000000G  ADD      #DHBFSZ,R5     ;Reserve space for buffer
113 013422 062701 000002    7$:    ADD      #2,R1          ;Get # of next line
114 013426 020127 000000G  CMP      R1,#LSTHL     ;Have we checked all lines?

```

ALOCBF -- Allocate buffer space

```

115 013432 101743          BLOS  6$           ;Br if not
116 013434 005205          INC   R5           ;Bound up to next word
117 013436 042705 000001  BIC   #1,R5
118
119 ; Allocate space for tables that keep track of space in swap file
120 ;
121 013442 105737 000000G   TSTB  VSWPFL       ;Is this a swapping system?
122 013446 001415          BEQ   14$          ;Br if not
123 013450 013700 000000G   MOV   VSWPSL,R0   ;Get # slots in swap file
124 013454 006300          ASL   R0           ;Allocate 2 bytes per slot
125 013456 010537 000000G   MOV   R5,SWPPOS   ;Start of table with starting block #'s
126 013462 060005          ADD   R0,R5       ;Allocate space
127 013464 010537 000000G   MOV   R5,SWPJOB   ;Start of table with job #'s
128 013470 060005          ADD   R0,R5       ;Allocate space
129 013472 010537 000000G   MOV   R5,SCPFHD   ;Pointer to area with command packets
130 013476 062705 000000C   ADD   #NSCP*SP##SZ,R5 ;Allocate space for swap command packets
131 ;
132 ; Allocate a 512-byte buffer to use to access job context blocks
133 ;
134 013502 010537 000000G   14$: MOV   R5,CXTBUF ;Set address of buffer
135 013506 062705 001400   ADD   #1400,R5    ;Reserve space for the buffer
136 ;
137 ; Make sure TSX is not too big.
138 ;
139 013512 010500          MOV   R5,R0       ;GET CURRENT MEMORY ADDRESS
140 013514 004737 027630'   CALL  CHKMEM      ;CHECK FOR SPACE OVERFLOW
141 ;
142 ; Finished
143 ;
144 013520 012601          MOV   (SP)+,R1
145 013522 000207          RETURN

```

```

1                                     .SBTTL  ALCSLO -- Allocate silo buffers for lines
2                                     ;-----
3                                     ; Allocate the silo buffers that are used to hold characters as they
4                                     ; are received from serial lines.
5                                     ;
6                                     ; Inputs:
7                                     ;   R5 = Current pointer to start of free memory.
8                                     ;
9                                     ; Outputs:
10                                    ;   R5 = New pointer to start of free memory.
11                                    ;
12 013524 010146  ALCSLO: MOV      R1, -(SP)
13 013526 010246      MOV      R2, -(SP)
14                                    ;
15                                    ; Begin loop to check each line
16                                    ;
17 013530 012701 000000G      MOV      #LSTHL, R1      ;Get index to last hardware line
18                                    ;
19                                    ; Only allocate silo buffers for real lines
20                                    ;
21 013534 012702 000040 1$:      MOV      #32, R2      ;Set minimum size for time-sharing lines
22 013540 020127 000000G      CMP      R1, #LSTPL      ;Is this a primary line?
23 013544 101405      BLOS     2$      ;Br if yes
24 013546 020127 000000G      CMP      R1, #FSTIOL      ;Is this a CL line?
25 013552 103463      BLO     9$      ;Br if not
26 013554 012702 000020      MOV      #16, R2      ;Set minimum size for CL lines
27                                    ;
28                                    ; Determine how much space to allocate
29                                    ;
30 013560 016100 000000G 2$:      MOV      LHIRBA(R1), R0      ;Get requested size
31 013564 001002      BNE     8$      ;Br if a size was specified
32 013566 013700 000000G      MOV      VNCSLO, R0      ;Use default value
33 013572 020027 000000G 8$:      CMP      R0, #MAXSLO      ;Constrain size to 255 bytes
34 013576 101402      BLOS     3$      ;Br if ok
35 013600 012700 000000G      MOV      #MAXSLO, R0      ;Reduce size
36 013604 020002 3$:      CMP      R0, R2      ;Compare with acceptable minimum
37 013606 103001      BHIS     4$      ;Br if ok
38 013610 010200      MOV      R2, R0      ;Use minimum size allowed
39 013612 010061 000000G 4$:      MOV      R0, LHIRBA(R1)      ;Set # bytes to be allocated for silo
40                                    ;
41                                    ; Allocate the space
42                                    ;
43 013616 010561 000000G      MOV      R5, LHIRBB(R1)      ;Set base address of buffer
44 013622 010561 000000G      MOV      R5, LHIRBP(R1)      ;Init pointer where to store next char
45 013626 010561 000000G      MOV      R5, LHIRBG(R1)      ;Init pointer where to get next char
46 013632 010061 000000G      MOV      R0, LHIRBS(R1)      ;Set # free bytes in buffer
47 013636 060005      ADD      R0, R5      ;Allocate space for buffer
48 013640 010561 000000G      MOV      R5, LHIRBE(R1)      ;Set address beyond end of buffer
49                                    ;
50                                    ; Set up control information about when to send XON and XOFF
51                                    ;
52 013644 006200      ASR      R0      ;Get 1/2 of buffer size
53 013646 162700 000002      SUB      #2, R0      ;Minus two characters
54 013652 116102 000000G      MOVB   LHIRBC(R1), R2      ;Get specified size for XOFF point
55 013656 001002      BNE     5$      ;Br if value specified
56 013660 113702 000000G      MOVB   VNCXDF, R2      ;Try default
57 013664 020200 5$:      CMP      R2, R0      ;Is specified value ok?

```

ALCSLO -- Allocate silo buffers for lines

```

58 013666 101401          BLOS 6$           ;Br if yes
59 013670 010002          MOV  R0,R2         ;No, use size/2-2
60 013672 110261 000000G 6$:  MOVB R2,LHIRBC(R1) ;Set # of chars when XOFF sent
61 013676 116102 000001G  MOVB LHIRBC+1(R1),R2 ;Get specified size for XON point
62 013702 001002          BNE 7$           ;Br if a value was specified
63 013704 113702 000000G  MOVB VNCXON,R2     ;Try default
64 013710 020200          7$:  CMP  R2,R0         ;Is specified value ok?
65 013712 101401          BLOS 10$          ;Br if ok
66 013714 010002          MOV  R0,R2         ;No, use size/2-2
67 013716 110261 000001G 10$: MOVB R2,LHIRBC+1(R1) ;Set # of chars when XON sent
68 ;
69 ; Do the next line
70 ;
71 013722 162701 000002 9$:  SUB  #2,R1         ;Get next line index number
72 013726 003302          BGT 1$           ;Loop if more to do
73 ;
74 ; Finished
75 ;
76 013730 005205          INC  R5           ;Force R5 to be even
77 013732 042705 000001  BIC  #1,R5
78 013736 012602          MOV  (SP)+,R2
79 013740 012601          MOV  (SP)+,R1
80 013742 000207          RETURN

```

ALBFX -- Allocate buffers in extended memory region

```

1          .SBTTL  ALBFX  -- Allocate buffers in extended memory region
2          ;-----
3          ; ALBFX is called to allocate space for buffers that are not constrained
4          ; to fit in the 40Kb region that TSX-Plus occupies.
5          ;
6          ; Inputs:
7          ;   R5 = 64-Byte address of base of free memory region.
8          ;
9          ; Outputs:
10         ;   R5 = Address above top of buffers allocated.
11         ;
12 013744 010146 ALBFX:  MOV     R1, -(SP)
13 013746 010246      MOV     R2, -(SP)
14 013750 010346      MOV     R3, -(SP)
15         ;
16         ; Allocate character buffers for all lines
17         ; Note: Character buffer space will be accessed by mapping through PAR 6.
18         ;
19 013752 012701 000002      MOV     #2,R1          ; GET 1ST JOB INDEX NUMBER
20 013756 032761 000000G 000000G 3$:  BIT     ##DEAD,LSW3(R1) ; IS THIS LINE INSTALLED?
21 013764 001047      BNE     2$          ; BR IF NOT -- DON'T ALLOCATE ANY BUFFER SPACE
22 013766 020127 000000G      CMP     R1,#FSTD L      ; IS THIS A DETACHED JOB LINE?
23 013772 103403      BLO     1$          ; BR IF NOT
24 013774 020127 000000G      CMP     R1,#LSTD L      ; DETACHED JOB LINE?
25 014000 101441      BLOS    2$          ; BR IF DETACHED JOB -- DON'T ALLOCATE BUFFERS
26 014002 010561 000000G      1$:  MOV     R5,LTTPAR(R1) ; SET PHYSICAL MEMORY PAR OFFSET FOR BUFFER
27 014006 012702 000000G      MOV     #VPAR6,R2      ; GET VIRTUAL MEMORY ADDRESS FOR BASE OF PAR6
28 014012 010261 000000G      MOV     R2,LINBUF(R1) ; INPUT BUFFER STARTS AT BASE OF PAR6 REGION
29 014016 016100 000000G      MOV     LINSIZ(R1),R0 ; GET # BYTES FOR INPUT BUFFER
30 014022 010061 000000G      MOV     R0,LINSPC(R1) ; SET # FREE BYTES IN INPUT BUFFER
31 014026 060002      ADD     R0,R2          ; ADVANCE VIRTUAL ADDRESS
32 014030 010261 000000G      MOV     R2,LINEND(R1) ; POINTS PAST END OF INPUT BUFFER
33 014034 010003      MOV     R0,R3          ; Get # bytes in input buffer
34 014036 062703 000007      ADD     #7,R3          ; Bound up to multiple of 8
35 014042 072327 177775      ASH     #-3,R3         ; Get # bytes needed in activation-flag buffer
36 014046 060302      ADD     R3,R2          ; Reserve space for activation-flag buffer
37 014050 060300      ADD     R3,R0          ; Accumulate total buffer space
38 014052 010261 000000G      MOV     R2,LOTBUF(R1) ; POINTS TO START OF OUTPUT BUFFER AREA
39 014056 066100 000000G      ADD     LOTSIZ(R1),R0 ; ACCUMULATE # BYTES IN BOTH BUFFERS
40 014062 066102 000000G      ADD     LOTSIZ(R1),R2 ; ADVANCE VIRTUAL ADDRESS COUNTER
41 014066 010261 000000G      MOV     R2,LOTEND(R1) ; SAVE ADDRESS OF END OF OUTPUT BUFFER
42 014072 062700 000077      ADD     #77,R0         ; BOUND UP TO MULTIPLE OF 64 BYTES
43 014076 072027 177772      ASH     #-6,R0         ; CONVERT TO # 64-BYTE BLOCKS ALLOCATED
44 014102 060005      ADD     R0,R5          ; ADVANCE PHYSICAL MEMORY PAR ADDRESS
45 014104 062701 000002      2$:  ADD     #2,R1          ; ADVANCE LINE NUMBER
46 014110 020127 000000G      CMP     R1,#LSTSL     ; HAVE WE DONE ALL LINES YET?
47 014114 101720      BLOS    3$          ; BR IF MORE TO DO
48         ;
49         ; Allocate space for shared file record locking data structures
50         ;
51 014116 005737 000000G      TST     VMXSF          ; Shared file support wanted?
52 014122 001451      BEQ     5$          ; Br if not
53 014124 013737 000000G 000000G      MOV     VNUMDC,NUMDCD ; Set number of data cache blocks
54 014132 013737 000000G 000000G      MOV     VMXSFC,NUMCDB ; Set number of free CDB's
55 014140 010537 000000G      MOV     R5,LOKMEM     ; Set phys address of base of area
56 014144 012701 000000G      MOV     #FF##SZ,R1   ; Size of an FDB
57 014150 070137 000000G      MUL     VMXSF,R1     ; Times number of FDB's

```

ALBFX -- Allocate buffers in extended memory region

```

58 014154 062701 000000C      ADD      #<NLINES*FW$$SZ>,R1 ;Space needed for wait blocks
59 014160 013703 000000G      MOV      VMLBLK,R3          ;Max blocks a CDB may hold locked
60 014164 006303              ASL      R3                  ;Two bytes per entry
61 014166 062703 000000G      ADD      #FC$LBN,R3         ;Add base size of a CDB
62 014172 070337 000000G      MUL      VMXSFC,R3         ;Times number of shared file channels
63 014176 060301              ADD      R3,R1              ;Accumulate space needed
64 014200 012703 000000G      MOV      #DC$$SZ,R3        ;Size of a data cache descriptor
65 014204 070337 000000G      MUL      VNUMDC,R3         ;Times number of data cache entries
66 014210 060301              ADD      R3,R1              ;Reserve space for data cache descriptors
67 014212 062701 000100      ADD      #64.,R1           ;Bound up to 64 byte unit
68 014216 072127 177772      ASH      #-6,R1            ;Convert to # 64 byte units
69 014222 042701 176000      BIC      #176000,R1        ;Clear sign extension
70 014226 060105              ADD      R1,R5              ;Reserve space for data structures
71 014230 010537 000000G      MOV      R5,LOKCSH         ;Save pointer to start of cache buffer area
72 014234 013701 000000G      MOV      VNUMDC,R1         ;# shared-file data cache blocks wanted
73 014240 070127 000010      MUL      #8.,R1           ;8 64-byte blocks each (512 bytes each)
74 014244 060105              ADD      R1,R5              ;Reserve space for data cache buffers
75                               ;
76                               ; Allocate space for mapped I/O buffers
77                               ;
78 014246 105737 000000G      5$:     TSTB      MIOFLG      ;Are any mapped I/O buffers needed?
79 014252 001415              BEQ      7$                ;Br if not
80 014254 013701 000000G      MOV      MIOBHD,R1         ;Point to 1st mapped I/O control block
81 014260 001412              BEQ      7$                ;Br if no more buffers needed
82 014262 010561 000000G      6$:     MOV      R5,MI$SBP(R1) ;Set address of base of buffer
83 014266 113703 000000G      MOV      VMIOSZ,R3         ;Get # blocks for buffer
84 014272 072327 000003      ASH      #3,R3            ;Convert to # 64-byte pages
85 014276 060305              ADD      R3,R5            ;Allocate space for buffer
86 014300 016101 000000G      MOV      MI$LNK(R1),R1     ;Get address of next control block
87 014304 001366              BNE      6$                ;Loop if more to allocate
88                               ;
89                               ; Allocate space for performance monitor data buffer if it is wanted.
90                               ;
91 014306 013701 000000G      7$:     MOV      VPMSIZ,R1     ;DID USER GEN IN PERFORMANCE MONITOR FEATURE?
92 014312 001412              BEQ      9$                ;BR IF NOT
93 014314 006201              ASR      R1                ;CONVERT BYTES TO WORDS
94 014316 010137 000000G      MOV      R1,PMCELS         ;SET INTO CELL IN TSEXEC
95 014322 010537 000000G      MOV      R5,PMPAR         ;SET BASE ADDRESS OF PM BUFFER
96 014326 062701 000037      ADD      #37,R1           ;BOUND SIZE UP TO 64-BYTE MULTIPLE
97 014332 072127 177773      ASH      #-5,R1           ;CONVERT WORDS TO # 64-BYTE BLOCKS
98 014336 060105              ADD      R1,R5            ;ADVANCE FREE MEMORY POINTER
99                               ;
100                              ; Finished
101                              ;
102 014340 012603      9$:     MOV      (SP)+,R3
103 014342 012602      MOV      (SP)+,R2
104 014344 012601      MOV      (SP)+,R1
105 014346 000207      RETURN

```

```

1          .SBTTL  OPNKMN -- Open channel to TSKMON
2          ;-----
3          ; OPNKMN is called to open an I/O channel to TSKMON SAV file and to
4          ; set up information about TSKMON.
5          ;
6          ; Inputs:
7          ; R5 = Address of base of free memory area
8          ;
9          ; Outputs:
10         ; KMNCHN = Saved status of channel to use to access TSKMON SAV file.
11         ; KMNTOP = Top of memory address for TSKMON.
12         ; KMNHI  = Top address of TSKMON - KMNBAS.
13         ; KMNPGS = Number of 256-word memory pages needed for TSKMON & context area
14         ; KMNSTK = Address of stack to use while TSKMON running.
15         ; KMNSTR = Starting address of TSKMON.
16         ;
17 014350 010246 OPNKMN: MOV      R2,-(SP)
18         ;
19         ; Lookup TSKMON file.
20         ;
21 014352          .LOOKUP #AREA,#1,#KMNNAM ;TRY TO FILE KMON SAV FILE
22 014372 103517   BCS      9#          ;BR IF NOT THERE
23         ;
24         ; Read block 0 of save file and extract some information.
25         ;
26 014374 013702 000152'   MOV      WRKBUF,R2          ;Point to work buffer
27 014400          .READW #AREA,#1,R2,#256.,#0 ;READ BLOCK 0 OF SAV FILE
28         ;
29         ; Determine size of kmon
30 014434 016200 000050   MOV      50(R2),R0          ;GET TOP ADDRESS OF KMON
31 014440 062700 000003   ADD      #3,R0            ;BOUND UP TO NEXT WORD
32 014444 042700 000001   BIC      #1,R0            ;FORCE EVEN
33 014450 010037 000000G   MOV      R0,KMNTOP
34         ; Determine number of 256-word memory pages needed while kmon running.
35 014454 162700 000000G   SUB      #KMNBAS,R0       ;BASE ADDRESS OF KMON
36 014460 010037 000000G   MOV      R0,KMNHI        ;TOP OF TSKMON - KMNBAS
37 014464 062700 000777   ADD      #511.,R0        ;BOUND UP TO PAGE SIZE
38 014470 000241          CLC
39 014472 006000          ROR      R0                ;CVT TO # WORDS
40 014474 000300          SWAB     R0                ;CVT TO # PAGES
41 014476 042700 177400   BIC      #^C377,R0
42 014502 063700 000000G   ADD      CXTPA6,R0       ;# PAGES NEEDED FOR JOB CONTEXT AREA
43 014506 010037 000000G   MOV      R0,KMNPGS       ;# 256-wd pages needed for kmon + context area
44         ; Determine Kmon stack pointer.
45 014512 016237 000042 000000G   MOV      42(R2),KMNSTK   ;INITIAL STACK POINTER FOR KMON
46         ; Determine Kmon starting address.
47 014520 016237 000040 000000G   MOV      40(R2),KMNSTR   ;STARTING ADDRESS
48         ;
49         ; Set up demo-system time limit
50         ; (If this is a demo version of TSX-Plus, the number of minutes the system
51         ; is to run before it crashes is stored in location 300 of TSKMON)
52 014526 016237 000300 000000G   MOV      300(R2),DTLX    ;SET DEMO TIME-LIMIT
53         ;
54         ; Now save status of channel so we can do a reopen when we need kmon.
55         ;
56 014534 012700 000000G   MOV      #KMNCHN,R0      ;GET KMON CHANNEL SAVE AREA
57 014540 013702 000072'   MOV      KMNNAM,R2       ;GET KMON DEVICE NAME

```

```
58 014544 004737 027134'          CALL   SETCHN          ;SAVE CHANNEL STATUS
59                                     ;
60                                     ; Lookup CCL.SAV and save channel status for it.
61                                     ;
62 014550                                     . LOOKUP #AREA,#1,#CCLNAM;LOOKUP SY:CCL.SAV
63 014570 103410                          BCS     B#             ;BR IF CAN'T FIND CCL
64 014572 012700 000000G                   MOV     #CCLSAV,R0    ;CHANNEL SAVE AREA
65 014576 013702 000102'                   MOV     CCLNAM,R2    ;DEVICE NAME
66 014602 004737 027134'          CALL   SETCHN          ;SAVE CHANNEL STATUS
67                                     ;
68                                     ; Finished
69                                     ;
70 014606 012602          10#:   MOV     (SP)+,R2
71 014610 000207                   RETURN
72                                     ;
73                                     ; Error: We could not find SY:CCL.SAV
74                                     ;
75 014612                                     8#:   .PRINT #TSXHD
76 014620                                     .PRINT #NOCCL          ;PRINT ERROR MESSAGE
77 014626 000137 004102'          JMP     INISTP          ;ABORT INITIALIZATION
78                                     ;
79                                     ; Error: We could not locate TSKMON SAV file.
80                                     ;
81 014632                                     9#:   .PRINT #TSXHD          ;PRINT ERROR MESSAGE
82 014640                                     .PRINT #NOKMON
83 014646 000137 004102'          JMP     INISTP          ;ABORT INITIALIZATION
```

CLINIT -- Initialize CL handler

```

1          .SBTTL  CLINIT -- Initialize CL handler
2          ;-----
3          ; Perform initialization for CL (Communication Line) handler
4          ;
5          ; Inputs:
6          ;   R5 = Address of start of free memory area.
7          ;
8          ; Outputs:
9          ;   R5 = Address of new start of free memory area.
10         ;
11 014652 010146 CLINIT: MOV     R1,-(SP)
12 014654 010246      MOV     R2,-(SP)
13 014656 010346      MOV     R3,-(SP)
14         ;
15         ; Initialize tables for each CL unit
16         ;
17 014660 005003      CLR     R3          ;Accumulate ring buffer sizes in R3
18 014662 012701 000000C  MOV     #2*<CLTOTL-1>,R1;Get index # of last CL unit
19         ;
20         ; See if this CL unit is connected to hardware or is free to be
21         ; connected later to a time-sharing line.
22         ;
23 014666 016102 000000G 1$:  MOV     CL$LIX(R1),R2  ; Is this CL unit associated with a line?
24 014672 001416      BEQ     5$          ; Br if not
25 014674 012762 000000G 000000G  MOV     ##SXON,LSW10(R2); Send XON when we start the line
26 014702 010162 000000G  MOV     R1,LCLUNT(R2)  ; Associate the CL unit with this line
27 014706 005762 000000G  TST     RSR(R2)        ; Does this unit have a specified RSR addr?
28 014712 001006      BNE     5$          ; Br if yes
29 014714 005762 000000G  TST     LMXNUM(R2)    ; Is this a mux line?
30 014720 001003      BNE     5$          ; Br if yes
31 014722 005061 000000G  CLR     CL$EPS(R1)    ; Say no endstring buffer
32 014726 000472      BR      4$          ; Line is not genned in
33         ;
34         ; Allocate and set up pointers for the output ring buffers
35         ;
36 014730 010561 000000G 5$:  MOV     R5,CL$ORB(R1)  ; Start of output ring buffer
37 014734 010561 000000G  MOV     R5,CL$ORP(R1)  ; Input character pointer
38 014740 010561 000000G  MOV     R5,CL$ORQ(R1)  ; Next available character pointer
39 014744 012700 000000G  MOV     #CLORSZ,R0    ; Get default output ring buffer size
40 014750 005702      TST     R2          ; Is this CL unit connected to a line?
41 014752 001402      BEQ     6$          ; Br if not
42 014754 016200 000000G  MOV     LOTSIZ(R2),R0 ; Get size of output ring buffer
43 014760 010061 000000G 6$:  MOV     R0,CL$ORA(R1)  ; Set size of output ring buffer
44 014764 010061 000000G  MOV     R0,CL$ORS(R1)  ; Available space in ring buffer
45 014770 060005      ADD     R0,R5          ; Point beyond end of ring buffer
46 014772 010561 000000G  MOV     R5,CL$ORE(R1)  ; Address past end of ring buffer
47 014776 060003      ADD     R0,R3          ; Accumulate size of output ring buffers
48         ;
49         ; Allocate space for end-of-file string buffer
50         ;
51 015000 010561 000000G      MOV     R5,CL$EPS(R1)  ; Set pointer to end-of-file string buffer
52 015004 005061 000000G  CLR     CL$EPP(R1)    ; No string to print yet
53 015010 062705 000001G  ADD     #<CLEOFS+1>,R5 ; Reserve space for buffer
54 015014 062703 000001G  ADD     #<CLEOFS+1>,R3 ; Accumulate buffer space
55         ;
56         ; Initialize end-of-file form-feed count
57         ;

```

```

58 015020 005061 000000G          CLR      CL$EPN(R1)      ; Init ENDPAGE=0
59                                ;
60                                ; Initialize option word
61                                ;
62 015024 012700 000000G          MOV      #<CO$DEF>,R0      ; Get default option flags
63 015030 005702                    TST      R2                ; Is this CL unit connected with a line?
64 015032 001421                    BEQ      7$                ; Br if not
65 015034 016202 000000G          MOV      ILSW2(R2),R2     ; Get line options
66 015040 032702 000000G          BIT      #$TAB,R2        ; Does hardware support tabs?
67 015044 001402                    BEQ      2$                ; Br if not
68 015046 052700 000000G          BIS      #CO$TAB,R0      ; Set hardware-tab flag
69 015052 032702 000000G          2$: BIT      #$FORM,R2    ; Does hardware support form feeds?
70 015056 001402                    BEQ      3$                ; Br if not
71 015060 052700 000000G          BIS      #CO$FF,R0       ; Set hardware-form-feed flag
72 015064 032702 000000G          3$: BIT      #$8BIT,R2    ; Does hardware want 8 bit support?
73 015070 001402                    BEQ      7$                ; Br if not
74 015072 052700 000000G          BIS      #CO$8BT,R0     ; Enable 8 bit support for CL line
75 015076 010061 000000G          7$: MOV      R0,CL$OPT(R1) ; Set options for this CL line
76                                ;
77                                ; Initialize page length
78                                ;
79 015102 012761 000102 000000G    MOV      #66.,CL$LEN(R1) ; Say page length = 66 lines
80                                ;
81                                ; Initialize status flags
82                                ;
83 015110 005061 000000G          CLR      CL$STA(R1)      ; Initialize status flags
84                                ;
85                                ; Do next line
86                                ;
87 015114 162701 000002          4$: SUB      #2,R1         ; Get index # of next unit
88 015120 002262                    BGE      1$                ; Loop if more units to initialize
89                                ;
90                                ; Make a device table entry for "CL" device
91                                ;
92 015122 062737 000002 000000G    ADD      #2,NUMDEV        ; One more device
93 015130 013701 000000G          MOV      NUMDEV,R1       ; Get device table index
94 015134 010137 000000G          MOV      R1,CLDEVX       ; Remember index number of CL device
95 015140 012761 012240 000000G    MOV      #R5OCL,PNAME(R1) ; Set device name ("CL")
96 015146 012761 000000G 000000G  MOV      #CLSTS,DVSTAT(R1) ; Set dev status flags
97 015154 012761 000000C 000000G  MOV      #<DX$NCA!DX$NMT!DX$NRD>,DVFLAG(R1) ; Device info flags
98 015162 005061 000000G          CLR      DEVSIZ(R1)      ; Clear device size
99 015166 012761 000006G 000000G  MOV      #CLHEAD+6,HANENT(R1) ; Set handler entry point (4th word)
100 015174 062703 000000G          ADD      #CLSIZE,R3      ; Get size of handler
101 015200 062703 000000C          ADD      #<<CLTOTL*46.>+<NIOL*32.>,R3 ; Add size of tables in TSGEN
102 015204 010361 000000G          MOV      R3,HANSIZ(R1)   ; Set size of handler
103 015210 005205                    INC      R5                ; Make sure free-memory pointer is even
104 015212 042705 000001          BIC      #1,R5
105                                ;
106                                ; Make a device table entry for C1 if there are more than 8 CL units
107                                ;
108 015216 022727 000000G 000010  CMP      #CLTOTL,#8      ; Are there more than 8 CL units?
109 015224 101430                    BLOS     9$                ; Br if not -- Don't need C1
110 015226 062737 000002 000000G    ADD      #2,NUMDEV        ; One more device
111 015234 013701 000000G          MOV      NUMDEV,R1       ; Get device table index
112 015240 010137 000000G          MOV      R1,C1DEVX       ; Remember index number of CL device
113 015244 012761 013630 000000G    MOV      #R5OC1,PNAME(R1) ; Set device name ("C1")
114 015252 012761 000000G 000000G  MOV      #CLSTS,DVSTAT(R1) ; Set dev status flags

```

CLINIT -- Initialize CL handler

```
115 015260 012761 000000C 000000G      MOV      #<DX$NCA!DX$NMT!DX$NRD>, DVFLAG(R1) ;Device info flags
116 015266 005061 000000G      CLR      DEVSIZ(R1) ;Clear device size
117 015272 012761 000006G 000000G      MOV      #CLHEAD+6, HANENT(R1) ;Set handler entry point (4th word)
118 015300 012761 000004 000000G      MOV      #4., HANSIZ(R1) ;Set size of handler
119                                     ;
120                                     ; Finished
121                                     ;
122 015306 012603      9#:      MOV      (SP)+, R3
123 015310 012602      MOV      (SP)+, R2
124 015312 012601      MOV      (SP)+, R1
125 015314 000207      RETURN
```

INDINI -- Initialize IND program

```

1          .SBTTL  INDINI -- Initialize IND program
2          ;-----
3          ; Perform initialization for IND program.
4          ;
5          ; Outputs:
6          ; If IND is available, the following information is set up:
7          ;   INDSAV = 5 word .SAVESTATUS block for SY:IND.SAV file
8          ;   INDDBL = Lowest block # within IND.SAV file of data overlay segment.
9          ;   INDDBS = Number of blocks used for data overlay segment.
10         ;   INDTSV = 5 word .SAVESTATUS block for SY:TSXIND.TSX file
11         ;
12 015316 010246 INDINI: MOV     R2,-(SP)
13 015320 010346         MOV     R3,-(SP)
14         ;
15         ; Determine if IND support is wanted
16         ;
17 015322 005037 0000000 CLR     INDSAV      ;ASSUME IND SUPPORT NOT WANTED
18         ;
19         ; Lookup SY:IND.SAV file
20         ;
21 015326 013737 0000000 000220' MOV     SYNAME,INDNAM ;LOOK UP IND ON BOOT DEVICE
22 015334         .LOOKUP #AREA,#1,#INDNAM ;TRY TO FIND SY:IND.SAV
23 015354 103002         BCC     4$          ;BR IF FOUND IND
24 015356 000137 016010'         JMP     9$          ;IF CAN'T FIND IND, THEN NO IND SUPPORT
25         ;
26         ; Set up information about IND overlay data segment
27         ;
28 015362 013703 000152' 4$:   MOV     WRKBUF,R3      ;Get pointer to work buffer
29 015366         .READW #AREA,#1,R3,#256.,#0 ;READ IN BLOCK 0 OF SAV FILE
30 015422 016302 000064         MOV     64(R3),R2      ;GET POINTER TO OVERLAY TABLE
31 015426         .READW #AREA,#1,R3,#256.,#1 ;READ IN BLOCK 1 WITH OVERLAY TABLE
32 015464 162702 001000         SUB     #1000,R2      ;GET ADDRESS OF OVERLAY TABLE REL TO BLOCK 1
33 015470 060302         ADD     R3,R2          ;ADD BASE ADDRESS WHERE BLOCK 1 DATA IS
34 015472 011203         MOV     (R2),R3      ;Get virtual address of segment 0
35 015474 020312 5$:   CMP     R3,(R2)      ;Search for 1st segment with different addr
36 015476 001003         BNE     6$          ;Br if found it (this is the data segment)
37 015500 062702 000006         ADD     #6,R2          ;Point to overlay table entry for next seg
38 015504 000773         BR     5$
39 015506 005722 6$:   TST     (R2)+      ;Point to word with block # if SAV file
40 015510 012237 0000000         MOV     (R2)+,INDDBL ;GET BLOCK # IN SAV FILE OF DATA OVERLAY
41 015514 011202         MOV     (R2),R2      ;GET # OF WORDS IN OVERLAY SEGMENT
42 015516 062702 000377         ADD     #255.,R2     ;ROUND UP TO NEXT BLOCK
43 015522 000302         SWAB   R2          ;CONVERT # WORDS TO # BLOCKS
44 015524 042702 177400         BIC     #^C<377>,R2
45 015530 010237 0000000         MOV     R2,INDDBS    ;SAVE # BLOCKS USED FOR DATA OVERLAY
46         ;
47         ; Do .SAVESTATUS on channel opened to IND.SAV file so that we
48         ; can do a reopen to access it from KMON.
49         ;
50 015534 012700 0000000         MOV     #INDSAV,R0    ;GET ADDRESS OF SAVESTATUS BLOCK
51 015540 013702 000220'         MOV     INDNAM,R2    ;GET RAD50 DEVICE NAME
52 015544 004737 027134'         CALL   SETCHN      ;SAVE FILE STATUS
53         ;
54         ; Determine how much space is needed for SY:TSXIND.TSX swap file
55         ;
56 015550 013703 0000000         MOV     INDDBS,R3    ;GET # BLOCKS NEEDED PER JOB
57 015554 070327 0000000         MUL     #<LSTSL/2>,R3 ;TIMES TOTAL NUMBER OF JOBS

```

INDINI -- Initialize IND program

```

58
59
60 ; Load Rt-11 device handler for ind swap file.
61 ;
62 015560 013700 000000G      MOV      INDFIL,R0      ;Get name of the device
63 015564 004737 027514'     CALL     RTFTCH        ;Try to fetch the RT-11 device handler
64 015570 103522              BCS     11$           ;Br if error on handler fetch
65 ;
66 ; See if TSXIND file already exists
67 ;
68 015572                    .LOOKUP #AREA,#1,#INDFIL ; DOES SY:TSXIND.TSX FILE EXIST NOW?
69 015612 103415              BCS     1$           ;BR IF NOT
70 015614 020003              CMP     R0,R3         ;IS IT OF THE CORRECT SIZE?
71 015616 001462              BEQ     2$           ;BR IF YES
72 015620                    .PURGE #1           ;FILE IS OF WRONG SIZE
73 015626                    .DELETE #AREA,#1,#INDFIL;DELETE OLD FILE
74 ;
75 ; File does not now exist
76 ; Create new file
77 ;
78 015646                    1$: .ENTER #AREA,#1,#INDFIL,R3 ;CREATE NEW TSXIND FILE
79 015672 103451              BCS     10$          ;BR IF ERROR ON CREATE
80 015674 010302              MOV     R3,R2         ;# BLOCKS IN FILE
81 015676 005302              DEC     R2           ;GET # OF LAST BLOCK IN FILE
82 015700                    .WRITW #AREA,#1,WRKBUF,#256.,R2 ;WRITE TO LAST BLOCK OF FILE
83 015736                    .CLOSE #1          ;NOW CLOSE THE FILE
84 015744                    .LOOKUP #AREA,#1,#INDFIL ;REOPEN TSXIND FILE WITH LOOKUP
85 ;
86 ; Do .SAVESTATUS for SY:TSXIND.TSX file
87 ;
88 015764 012700 000000G     2$:  MOV     #INDTSV,R0      ;POINT TO SAVESTATUS BLOCK
89 015770 013702 000000G     MOV     INDFIL,R2      ;GET RAD50 DEVICE NAME
90 015774 004737 027134'     CALL     SETCHN       ;SAVE FILE INFO
91 016000                    .RELEAS #INDFIL      ;Release device handler
92 ;
93 ; Finished
94 ;
95 016010 012603             9$:  MOV     (SP)+,R3
96 016012 012602             MOV     (SP)+,R2
97 016014 000207             RETURN
98 ;
99 ; Error occurred while opening SY:TSXIND.TSX file
100 ;
101 016016                    10$: .PRINT #TSXHD        ;Print error message
102 016024                    .PRINT #INDOPN
103 016032 004737 011350'     CALL     SPNEED       ;Print info about number of blocks needed
104 ;
105 ; Error: Invalid device specification.
106 ;
107 016036 010001             11$: MOV     R0,R1         ;Save device name
108 016040                    .PRINT #TSXHD        ;Print error message
109 016046                    .PRINT #INDOPN
110 016054 004737 011376'     CALL     BADDEV       ;Print invalid device specification

```

UCLINI -- Initialize TSXUCL data file

```

1          .SBTTL  UCLINI -- Initialize TSXUCL data file
2          ;-----
3          ; UCLINI is called to initialize the TSXUCL data file which is used
4          ; to store user-defined commands.
5          ;
6          ; Outputs:
7          ; TSXUCL data file is initialized.
8          ; UCLBLK = Number of blocks in data file for each job.
9          ;
10         016060 010246 UCLINI: MOV      R2,-(SP)
11         016062 010346         MOV      R3,-(SP)
12         ;
13         ; Determine if TSXUCL data file is needed
14         ;
15         016064 105737 000000G      TSTB   VU$CL      ;Is TSXUCL being used at all?
16         016070 001536              BEQ    9$           ;Br if not
17         016072 013702 000000G      MOV    VUCLMC,R2   ;Get maximum number of commands
18         016076 001533              BEQ    9$           ;Br if none allowed
19         ;
20         ; Determine number of blocks needed in data file for each job
21         ;
22         016100 012700 000000G      MOV    #UK$$SZ,R0   ;Size of each keyword descriptor
23         016104 062700 000000G      ADD    #US$$SZ,R0   ;Size of each command string descriptor
24         016110 070200              MUL    R0,R2        ;Compute total # bytes for keywords+commands
25         016112 062703 000777G      ADD    #UC$$SZ+511.,R3 ;Add space for control information & round up
26         016116 005502              ADC    R2          ;Propogate carry
27         016120 071227 001000      DIV    #512.,R2    ;Convert to # of blocks needed
28         016124 010237 000000G      MOV    R2,UCLBLK  ;Save number of blocks needed per job
29         ;
30         ; Multiply by number of jobs to get total file size
31         ;
32         016130 070227 000000C      MUL    #<LSTSL/2>,R2 ;Times total number of jobs
33         ;
34         ; Load Rt-11 device handler for ind swap file.
35         ;
36         016134 013700 000000G      MOV    UCLDAT,R0   ;Get name of the device
37         016140 004737 027514'      CALL  RTFTCH       ;Try to fetch the RT-11 device handler
38         016144 103523              BCS   11$          ;Br if error on handler fetch
39         ;
40         ; The total required file size is now in R3.
41         ; See if the file already exists.
42         ;
43         016146              .LOOKUP #AREA,#1,#UCLDAT ;See if the file exists now
44         016166 103415              BCS   1$           ;Br if file does not exist
45         016170 020003              CMP   R0,R3        ;Is existing file of correct size?
46         016172 001446              BEQ   2$           ;Br if yes -- use the old file
47         016174              .PURGE #1             ;Purge the channel
48         016202              .DELETE #AREA,#1,#UCLDAT;Delete the old file
49         ;
50         ; Create a new data file
51         ;
52         016222 1$: .ENTER #AREA,#1,#UCLDAT,R3 ;Create new data file
53         016246 103452              BCS   10$          ;Br if error creating the file
54         016250 005303              DEC   R3           ;Get # of last block in the file
55         016252              .WRITW #AREA,#1,WRKBUF,#256.,R3 ;Write to last block of file
56         ;
57         ; Translate possible logical device name to physical name and close

```

UCLINI -- Initialize TSXUCL data file

```

58          ; (Physical name is needed for TSXUCL program.)
59          ;
60 016310   2$:      .CSTAT  #AREA,#1,#NFSBLK      ;GET CHANNEL STATUS INFORMATION
61 016330   013702 000032'  .MOV      <NFSBLK+12>,R2 ;FETCH DEVICE NAME IN RAD50
62 016334   063702 000030'  .ADD      <NFSBLK+10>,R2 ;ADD IN DEVICE UNIT NUMBER
63 016340   062702 000036   .ADD      #^R 0,R2      ;CONVERT UNIT NUMBER TO RAD50
64 016344   010237 000000G   .MOV      R2,UCLDAT     ;SET PHYSICAL NAME BACK INTO TSGEN CELL
65 016350   .CLOSE   #1      ;Close the file
66 016356   .RELEAS #UCLDAT ;Release device handler
67          ;
68          ; Finished
69          ;
70 016366   012603   9$:      .MOV      (SP)+,R3
71 016370   012602   .MOV      (SP)+,R2
72 016372   000207   .RETURN
73          ;
74          ; Error creating the data file
75          ;
76 016374   10$:     .PRINT  #TSXHD      ;Print error message
77 016402   .PRINT  #UCLOPN
78 016410   004737 011350' .CALL   SPNEED      ;Print info about number of blocks needed
79          ;
80          ; Error: Invalid device specification.
81          ;
82 016414   010001   11$:     .MOV      RO,R1      ;Save device name
83 016416   .PRINT  #TSXHD      ;Print error message
84 016424   .PRINT  #UCLOPN
85 016432   004737 011376' .CALL   BADDEV      ;Print invalid device specification

```

MEMINI -- Initialize memory management

```

1                                     .SBTTL MEMINI -- Initialize memory management
2                                     ;-----
3                                     ; Initialize memory management registers for a 1-to-1 mapping.
4                                     ; But leave memory management turned off.
5                                     ;
6 016436 010146 MEMINI: MOV      R1,-(SP)
7 016440 010246      MOV      R2,-(SP)
8 016442 010346      MOV      R3,-(SP)
9 016444 010446      MOV      R4,-(SP)
10 016446 010546      MOV      R5,-(SP)
11                                     ;
12                                     ; Initialize all pages for a 1-to-1 mapping.
13                                     ;
14 016450 012700 0000000 12$:  MOV      #KPAR0,R0      ;Kernel mode PAR 0
15 016454 012701 0000000      MOV      #UPAR0,R1      ;User mode PAR 0
16 016460 012702 0000000      MOV      #KPDR0,R2      ;Kernel mode PDR 0
17 016464 012703 0000000      MOV      #UPDR0,R3      ;User mode PDR 0
18 016470 012704 000010      MOV      #8,R4        ;Initialize 8 pages
19 016474 005005      CLR      R5          ;Set initial PAR value
20 016476 010520 2$:  MOV      R5,(R0)+      ;Set kernel PAR
21 016500 010521      MOV      R5,(R1)+      ;Set user PAR value
22 016502 012722 077406      MOV      #077406,(R2)+  ;Set kernel PDR
23 016506 012723 077406      MOV      #077406,(R3)+  ;Set user PDR value
24 016512 062705 000200      ADD      #200,R5       ;Advance block number
25 016516 077411      SOB      R4,2$       ;Init all pages
26                                     ;
27                                     ; Map kernel mode I/O page (160000) to 17760000.
28                                     ;
29 016520 012737 0000000 0000000      MOV      #IOPAGE,@#KPAR7 ;Map I/O page
30                                     ;
31                                     ; Finished
32                                     ;
33 016526 012605      MOV      (SP)+,R5
34 016530 012604      MOV      (SP)+,R4
35 016532 012603      MOV      (SP)+,R3
36 016534 012602      MOV      (SP)+,R2
37 016536 012601      MOV      (SP)+,R1
38 016540 000207      RETURN

```

MEMTST -- Set up information about available memory space

```

1          .SBTTL  MEMTST -- Set up information about available memory space
2          ;-----
3          ; MEMTST is called to set up information related to memory management.
4          ; MEMTST performs the following functions:
5          ;   1. Determine how much memory is installed on machine.
6          ;   2. Load Kernel mode mapping registers.
7          ;
8          ; Inputs:
9          ;   R5 = top of memory currently allocated for TSX and low memory buffers.
10         ;
11         ; Outputs:
12         ;   PHYMEM = 64-byte block # above top of physical memory.
13         ;   FMEMHI = 64-byte block # above top of memory available for system.
14         ;   Kernel mode mapping registers loaded.
15         ;   Memory management is left turned off.
16         ;
17         ;
18         ;
19         ; Offset word to test for memory wrap - choose a location which will not
20         ; effect RT-11 or TSX-Plus initialization.
21         ;
22         000110      TSTWRD = 110          ; Offset word to test for memory wrap
23         ;
24         016542      010146      MEMTST:  MOV     R1, -(SP)
25         016544      010246      MOV     R2, -(SP)
26         016546      010346      MOV     R3, -(SP)
27         016550      010446      MOV     R4, -(SP)
28         016552      010546      MOV     R5, -(SP)
29         016554      013746      000004      MOV     @#4, -(SP)          ; Save illegal mem. ref. trap vector
30         ;
31         ; Determine if this machine has a memory management register # 3.
32         ; If it does not, then machine cannot possibly have more than 256Kb.
33         ;
34         016560      012737      017132' 000004      MOV     #TRCSET, @#4          ; Catch trap
35         016566      000240      NOP                                ; Clean out 11/73 pipeline
36         016570      000240      NOP                                ; Before attempting trap
37         016572      005737      000000G      TST     @#SR3MMR          ; Try to access status register 3
38         016576      103411      BCS     22$          ; Br if MMU 3 status register is non-existent
39         016600      105237      000000G      INCB   SR3FLG          ; No trap. We must have SR3
40         ;
41         ; Now determine if it implements D-space (11/23 11/24 do not)
42         ; If there was no SR3MMR, then it certainly doesn't!
43         ;
44         016604      000240      NOP                                ; Clean out the pipeline again?
45         016606      000240      NOP                                ;
46         016610      005737      000000G      TST     @#UDDRO          ; It there a user D-space PDR0?
47         016614      103402      BCS     22$          ; Br if not
48         016616      105237      000000G      INCB   IDSFLG          ; No trap. Must implement D-space
49         ;
50         ; If we are running on a Professional, there is a register that tells
51         ; us how much memory is installed on the machine.
52         ;
53         016622      22$:
54         .IF     NE, PROASM
55         TSTB   PROFLG          ; Are we running on a Professional?
56         BEQ    26$            ; Br if not
57         CLR    R5              ; Load byte without sign extension

```

MEMTST -- Set up information about available memory space

```

58          BISB    @#173050,R5    ;Get 32Kb top of system RAM boundary
59          ASH     #9.,R5        ;Convert to # 64 byte blocks
60          SUB     #10.,R5       ;Don't use the last 512 bytes of memory
61          BR      7$
62          26$:
63          . ENDC   ; NE, PROASM
64          . IF    NE, <PROASM-1> ; Assemble if could be on a PDP-11
65          ;
66          ; We are not running on a Professional.
67          ; Test each page above TSX to see where the top of memory is.
68          ;
69 016622 012737 017142' 000020    MOV     #RTNKM,@#20    ;Use IOT instruction to get out of user mode
70 016630 005037 000022            CLR     @#22
71 016634 052737 000000G 000000G  BIS     #MMENBL,@#SR0MMR;Enable memory management
72 016642 105737 000000G          TSTB    SR3FLG        ;Does maching have mem management reg # 3?
73 016646 001403                    BEQ     4$            ;Br if non-existent
74 016650 052737 000000G 000000G  BIS     #EMMAP,@#SR3MMR ;Enable 22-bit extended memory
75          ;
76          ; Map user page 7 to each successive 256-word block and attempt to access.
77          ;
78 016656 012705 002000          4$:    MOV     #1024.,R5    ;Start checking at 64Kb
79 016662 010537 000000G        5$:    MOV     R5,@#UPAR7    ;Map user page 7 to page to be tested
80 016666 052737 000000G 000000G  BIS     #UMODE,@#PSW   ;Go into user mode
81 016674 005737 160000          TST     @#160000     ;Can we access the page?
82          ;
83          ; Use IOT to get back into kernel mode.
84          ;
85 016700 000004                    IOT                    ;Return to kernel mode
86 016702 103405                    BCS     6$            ;Br if memory is non-existent
87 016704 062705 000010          ADD     #10.,R5       ;Go try next page
88 016710 020527 177600          CMP     R5,#177600    ;Don't enter I/O page
89 016714 103762                    BLO     5$
90          ;
91          ; Check for potential memory wrap (on 18-bit 256K byte computers).
92          ;
93 016716 020527 010000        6$:    CMP     R5,#10000    ;Is physical memory above 256K bytes
94 016722 101421                    BLOS    7$            ;Br if below 256K bytes
95 016724 005037 000110          CLR     @#TSTWRD     ;Clear physical location
96 016730 012737 010000 000000G  MOV     #10000,@#UPAR7 ;Map to 256K byte boundary
97 016736 052737 000000G 000000G  BIS     #UMODE,@#PSW   ;Go into user mode
98 016744 012737 177777 160110    MOV     #-1,@#160000+TSTWRD ;Store -1 at 256K physical location
99 016752 000004                    IOT                    ;Return to kernel mode
100 016754 005737 000110          TST     @#TSTWRD     ;Test physical location
101 016760 001402                    BEQ     7$            ;Br if physical location is clear
102 016762 012705 010000          MOV     #10000,R5    ;Constrain memory to 256K byte total
103          . ENDC   ; NE, <PROASM-1>
104          ;
105          ; Reached end of available memory.
106          ;
107 016766 010537 000000G        7$:    MOV     R5,PHYMEM    ;set physical memory size
108 016772 020537 000000G          CMP     R5,MAPSIZ    ;Constrain kernel to user specified cutoff
109 016776 101402                    BLOS    8$            ;Br if below user specified
110 017000 013705 000000G          MOV     MAPSIZ,R5    ;Only use this much memory
111 017004 010537 000000G        8$:    MOV     R5,$MEMSZ    ;Set # 64-byte blocks of total memory
112 017010 010537 000134'          MOV     R5,FMEMHI    ;Save base 64-byte block # of top of free mem
113          ;
114          ; Turn off memory management

```

MEMTST -- Set up information about available memory space

```

115
116 017014 105737 000000G          TSTB   SR3FLG          ;Do we have memory management reg # 3?
117 017020 001403                BEQ    9$             ;Br if non-existent
118 017022 042737 000000G 000000G BIC    #EMMAP,@#SR3MMR ;Disable extended memory management
119 017030 042737 000000G 000000G 9$: BIC    #MMENBL,@#SR0MMR;Turn off memory management
120
121          ; If this is a Q-bus machine with >256Kb then set EXTLSI flag in ICONFG
122
123 017036 023727 000134' 010000    CMP    FMEMHI,#4096.  ;Does machine have at least 256Kb?
124 017044 103411                BLD    25$           ;Br if not
125 017046 105237 000000G          INCB   MEM256         ;Remember machine has at least 256kb
126 017052 123727 000000G 000000G CMPB   VBUSTP,#QBUS   ;Is this a Q-bus machine?
127 017060 001003                BNE    25$           ;Br if not
128 017062 052737 000001 000242'   BIS    #EXTLSI,ICONFG ;Set extended-LSI flag in ICONFG
129
130          ; See if this machine needs UNIBUS mapping
131
132 017070 123727 000000G 000000G 25$: CMPB   VBUSTP,#UNIBUS ;Is this a UNIBUS machine?
133 017076 001005                BNE    29$           ;Br if not
134 017100 105737 000000G          TSTB   MEM256         ;Does machine have at least 256kb of memory?
135 017104 001402                BEQ    29$           ;Br if not
136 017106 105237 000000G          INCB   UBUSMP         ;Say UNIBUS mapping is needed
137
138          ; Finished
139
140 017112 012637 000004          29$:  MOV    (SP)+,@#4     ;Reset trap vector
141 017116 012605                MOV    (SP)+,R5
142 017120 012604                MOV    (SP)+,R4
143 017122 012603                MOV    (SP)+,R3
144 017124 012602                MOV    (SP)+,R2
145 017126 012601                MOV    (SP)+,R1
146 017130 000207                RETURN
147
148          ; Trap - return with C-bit set.
149
150 017132 052766 000001 000002 TRCSET: BIS    #1,2(SP) ;Set c-bit for return
151 017140 000002                RTI                    ;Return from trap
152
153          ; IOI - return at kernel mode with c-bit preserved.
154
155 017142 042766 000000G 000002 RTNKM: BIC    #UMODE,2(SP) ;Clear user mode - return to kernel
156 017150 000002                RTI                    ;Return from trap
157
158          ; Error: System does not have memory management hardware.
159
160 017152                NOXM:  .PRINT  #TSXHDD ;PRINT ERROR MESSAGE
161 017160                .PRINT  #NXMMMSG
162 017166 000137 004102'        JMP    INISTP         ;ABORT INITIALIZATION

```

```

1          .SBTTL  CXTALC -- Set up info about job context area
2          ;-----
3          ; Set up information about the size of the job context area.
4          ;
5          ; Outputs:
6          ; CXTWDS = Number of words needed for job context area.
7          ; CXTPAG = Number of 512-byte pages needed for context area.
8          ; CXTPDR = Value to load into PDR when mapping job context area.
9          ; CXTRMN = Address of simulated RMON in context area.
10         ; RMNPDR = Value to load into PDR to map to simulated RMON.
11         ;
12 017172  CXTALC:
13         ;
14         ; Get size of base portion of job context area
15         ;
16 017172  012700  000000G      MOV      #CXTSIZ,RO      ;Get # bytes for base context area
17         ;
18         ; Bound up to 64-byte boundary and add size of simulated RMON
19         ; which is allocated above the base job context data.
20         ;
21 017176  062700  000077      ADD      #63.,RO      ;Bound up to 64 byte boundary
22 017202  042700  000077      BIC      #77,RO
23 017206  010037  000000G      MOV      RO,CXTRMN      ;Offset to start of simulated RMON
24 017212  062737  000000G 000000G  ADD      #CXTBAS,CXTRMN ;Add base virtual address of context area
25 017220  062700  000001G      ADD      #MVSIZ+1,RO    ;Add space for simulated RMON & channels
26         ;
27         ; Save number of words needed for context area
28         ;
29 017224  006200              ASR      RO            ;Convert to # words
30 017226  010037  000000G      MOV      RO,CXTWDS      ;This is # words for whole job context area
31         ;
32         ; Compute PDR value to use to map to job context area
33         ;
34 017232  062700  000037      ADD      #31.,RO      ;Bound up to # 32 word units
35 017236  072027  177773      ASH      #-5.,RO      ;Get # 32-word units for context area
36 017242  000300              SWAB     RO            ;Put # 32-word units in high-order byte
37 017244  052700  000006      BIS      #6,RO        ;Set PDR control flags
38 017250  010037  000000G      MOV      RO,CXTPDR     ;This is the PDR value
39         ;
40         ; Compute # 512-byte pages needed for job context block
41         ;
42 017254  013700  000000G      MOV      CXTWDS,RO     ;Get back # words for context area
43 017260  062700  000377      ADD      #255.,RO     ;Bound up to # 256-word blocks
44 017264  072027  177770      ASH      #-8.,RO      ;Get # 256-word pages for context area
45 017270  010037  000000G      MOV      RO,CXTPAG     ;# pages for job context area
46         ;
47         ; Set up PDR value used when mapping to simulated RMON
48         ;
49 017274  012700  000000G      MOV      #MVSIZ,RO     ;Get size of monitor vector table
50 017300  062700  000077      ADD      #63.,RO      ;Round up to # 32 word blocks
51 017304  072027  177772      ASH      #-6,RO       ;Cvt to # 32-word blocks
52 017310  005300              DEC      RO            ;Get # blocks - 1
53 017312  000300              SWAB     RO            ;Put # blocks in left byte
54 017314  052700  000006      BIS      #6,RO        ;Allow read and write access
55 017320  042700  100261      BIC      #100261,RO    ;Make sure unused PDR bits are zero
56 017324  010037  000000G      MOV      RO,RMNPDR     ;This is PDR value to map to sim. RMON
57         ;

```

```
58           ; Finished  
59           ;  
60 017330 000207           RETURN
```

MAPALC -- Allocate memory usage table

```

1          .SBTTL  MAPALC -- Allocate memory usage table
2          ;-----
3          ; MAPALC is called to allocate a table that keeps track of which pages
4          ; of memory are currently in use by user jobs and which are free.
5          ; Each byte in the table corresponds to a 512-byte block of physical memory.
6          ; The portion of physical memory used by the system is not represented
7          ; in the memory allocation table.
8          ;
9          ; Inputs:
10         ; R5 = 64-byte block number of top of free memory area.
11         ; FMEMLO = 64-byte block number of base of free memory area.
12         ;
13         ; Outputs:
14         ; FMEMHI = 64-byte block number of top of free memory area.
15         ; MAPPAR = 64-byte block number used to map to the memory alloc table.
16         ; BASMAP = Virtual address of memory allocation table that would
17         ; correspond to physical address 0. Note, the entries
18         ; in the allocation table between BASMAP and LOMAP are
19         ; actually not allocated.
20         ; LOMAP = Virtual address of memory allocation table that corresponds
21         ; to 1st physical 512-byte page that is available to user jobs.
22         ; Note, LOMAP always contains 120000 because we access the
23         ; allocation table by mapping it through PAR 5.
24         ; HIMAP = Virtual address of memory allocation table that corresponds
25         ; to 512-byte page above the top of the user area.
26         ;
27 017332 010246 MAPALC: MOV     R2,-(SP)
28 017334 010346         MOV     R3,-(SP)
29 017336 010446         MOV     R4,-(SP)
30         ;
31         ; Determine how many bytes will be required for the memory allocation table.
32         ; One byte in the table is required for each 512-byte physical page.
33         ;
34 017340 010503         MOV     R5,R3           ;Get 64-byte block # of top of free mem
35 017342 072327 177775  ASH     #-3,R3           ;Convert to 512-byte page #
36 017346 042703 160000  BIC     #160000,R3        ;Kill possible sign extension
37 017352 013702 000136  MOV     FMEMLO,R2        ;Get 64-byte block # of base of free memory
38 017356 062702 000007  ADD     #7,R2           ;Round up
39 017362 072227 177775  ASH     #-3,R2           ;Convert to 512-byte page #
40 017366 042702 160000  BIC     #160000,R2        ;Kill possible sign extension
41 017372 160203         SUB     R2,R3           ;Get # bytes needed for allocation table
42 017374 003440         BLE     10$           ;Br if memory overflow
43 017376 010304         MOV     R3,R4           ;Get # bytes for allocation table
44 017400 062704 001000  ADD     #512,R4         ;Add 1 extra byte and round up to 512-byte
45 017404 072427 177767  ASH     #-9,R4         ;Get # 512-byte units needed for alloc table
46         ;
47         ; Set up virtual address pointers for the allocation table
48         ;
49 017410 012700 000000G  MOV     #VPAR5,R0       ;We will map to alloc table through PAR 5
50 017414 010037 000000G  MOV     R0,LOMAP        ;Pointer to 1st entry in alloc table
51 017420 160200         SUB     R2,R0           ;Get pseudo virtual address for page # 0
52 017422 010037 000000G  MOV     R0,BASMAP       ;This would point to alloc entry for page 0
53 017426 012700 000000G  MOV     #VPAR5,R0       ;Get back base address of table
54 017432 060300         ADD     R3,R0           ;Add # bytes used by table
55 017434 160400         SUB     R4,R0           ;Subtract space used by table itself
56 017436 010037 000000G  MOV     R0,HIMAP        ;Virtual address of 1st entry for system page
57

```

```
58 ; Allocate space for the allocation table
59 ;
60 017442 072427 000003 ASH #3,R4 ;Get # 64-byte units for alloc table
61 017446 160405 SUB R4,R5 ;Compute physical 64-byte base for table
62 017450 020537 000136' CMP R5,FMEMLO ;Did we run out of memory space?
63 017454 101410 BLOS 10$ ;Br if memory overflow
64 017456 010537 000000G MOV R5,MAPPAR ;Use this value to map PAR 5 to alloc table
65 017462 010537 000134' MOV R5,FMEMHI ;Save new top of free memory area
66 ;
67 ; Finished
68 ;
69 017466 012604 MOV (SP)+,R4
70 017470 012603 MOV (SP)+,R3
71 017472 012602 MOV (SP)+,R2
72 017474 000207 RETURN
73 ;
74 ; Error: Generated system is too large
75 ;
76 017476 10$: .PRINT #TSXHD ;Print error message heading
77 017504 .PRINT #PHSOVF ;Physical memory overflow
78 017512 000137 004102' JMP INISTP ;Abort the initialization
```

SETJSZ -- Set up information about maximum job sizes

```

1          .SBTTL  SETJSZ -- Set up information about maximum job sizes
2          ;-----
3          ; SETJSZ is called to set up some information about the maximum
4          ; job sizes to be allowed. The maximum job size is chosen so that
5          ; we are guaranteed to be able to get at least one job logged on.
6          ;
7          ; Inputs:
8          ; LOMAP = Address of 1st MEMMAP entry available to user jobs.
9          ; HIMAP = Address of 1st MEMMAP entry above top of user job area.
10         ;
11         ; Outputs:
12         ; FREPGS = Total number of 512-byte pages available to user jobs.
13         ; MXJMEM = max # K bytes available to a job
14         ; DFJMEM = Default job memory size (kb)
15         ;
16 017516 010546 SETJSZ: MOV      R5,-(SP)
17         ;
18         ; Determine total number of pages of memory available to user jobs
19         ;
20 017520 013705 000000G      MOV      HIMAP,R5      ; POINTER ABOVE LAST FREE PAGE ENTRY
21 017524 163705 000000G      SUB      LOMAP,R5      ; GET TOTAL # OF FREE PAGES
22 017530 010537 000000G      MOV      R5,FREPGS      ; # FREE PAGES AVAILABLE TO USER JOBS
23         ;
24         ; Make sure there is enough free space to run TSKMON.
25         ;
26 017534 020537 000000G      CMP      R5,KMNPGS      ; COMPARE # FREE PAGES TO # PAGES FOR TSKMON
27 017540 103436              BLO      1$              ; BR IF INSUFFICIENT MEMORY TO RUN TSKMON
28         ;
29         ; Set up max memory limit for jobs
30         ;
31 017542 013700 000000G      MOV      CXTPAG,R0      ; # PAGES NEEDED FOR JOB CONTEXT AREA
32 017546 010037 000000G      MOV      R0,JCXPGS
33 017552 160005              SUB      R0,R5
34 017554 006205              ASR      R5              ; LEAVE ROOM FOR JOB CONTEXT AREA
35 017556 020537 000000G      ; Convert # pages to # KB
36 017562 101402              CMP      R5,VHIMEM      ; COMPARE WITH TSGEN SPECIFIED MAX SIZE
37 017564 013705 000000G      BLOS     10$          ; BR IF CONSTRAINED BY PHYSICAL SIZE
38 017570 010537 000000G      MOV      VHIMEM,R5      ; LIMIT BY VALUE SPECIFIED IN TSGEN
39 017574 010500              10$:  MOV      R5,MXJMEM      ; MAX # K BYTES AVAILABLE TO A JOB
40 017576 072027 000012      MOV      R5,R0
41 017602 001002              ASH     #10.,R0        ; CONVERT # KB TO BYTE ADDRESS
42 017604 012700 177774      BNE     2$              ; BR IF DIDN'T OVERFLOW 64KB
43 017610 010037 000000G      MOV      #177774,R0     ; GET 64KB TOP ADDRESS
44         ; 2$:  MOV      R0,MXJADR      ; ADDRESS ABOVE TOP OF JOB
45         ;
46         ; Set default memory size of jobs
47         ;
47 017614 020537 000000G      CMP      R5,VDFMEM      ; COMPARE TO DEFAULT SPECIFIED IN TSGEN
48 017620 101402              BLOS     11$          ; BR IF CONSTRAINED BY PHYSICAL LIMIT
49 017622 013705 000000G      MOV      VDFMEM,R5      ; CONSTRAIN BY TSGEN PARAMETER
50 017626 010537 000000G      11$:  MOV      R5,DFJMEM      ; SET DEFAULT JOB MEMORY SIZE
51         ;
52         ; Finished
53         ;
54 017632 012605              MOV      (SP)+,R5
55 017634 000207              RETURN
56         ;
57         ; Error -- Insufficient memory space available to run TSKMON.

```

TSINIT -- TSX startup initializ MACRO V05.05 Tuesday 17-Jan-89 13:55 Page 48-1
SETJSZ -- Set up information about maximum job sizes

58

59 017636 004737 027650'

;

1#:

CALL

SIZERR

;Generated system is too big -- abort

```

1          . IF      NE,<PROASM-1>    ;No parity control if PRO only
2          . SBTTL   PARSET -- Setup memory parity control
3          ;-----
4          ; PARSET is called to set up memory parity control.
5          ; Currently this consists of disabling memory parity.
6          ;
7 017642 005727 000000G PARSET: TST      #MPARFL      ;Does he want to disable memory parity?
8 017646 001027          BNE      20$         ;Br if not
9 017650 010246          MOV      R2,-(SP)
10 017652 013746 000004 MOV      @#4,-(SP)      ;Save contents of trap vector
11          ;
12          ; Catch traps that occur when we access unimplemented parity registers
13          ;
14 017656 012737 017702' 000004 MOV      #2@,@#4       ;Send traps to 2$
15 017664 000240          NOP                    ;Clean out instruction pipeline
16 017666 000240          NOP
17          ;
18          ; Disable parity for each block of memory
19          ;
20 017670 012702 000000G          MOV      #MPARO,R2      ;Point to 1st memory control register
21 017674 042712 000000G 1$:      BIC      #PARENL,(R2)    ;Disable memory parity
22 017700 000402          BR       3$          ;We did not trap
23 017702 062706 000004 2$:      ADD      #4,SP        ;Clean trap PS and PC off of stack
24 017706 062702 000002 3$:      ADD      #2,R2        ;Point to next parity control register
25 017712 020227 000000G          CMP      R2,#MPAR16    ;Have we cleared all registers?
26 017716 101766          BLOS     1$          ;Loop if not
27          ;
28          ; Finished
29          ;
30 017720 012637 000004          MOV      (SP)+,@#4      ;Restore trap vector
31 017724 012602          MOV      (SP)+,R2
32 017726 000207 20$:      RETURN
33          . IFF      ; NE,<PROASM-1>
34 PARSET: RETURN
35          . ENDC     ; NE,<PROASM-1>

```

GETHNL -- Load device handlers into memory

```

1          .SBTTL  GETHNL -- Load device handlers into memory
2          ;-----
3          ; GETHNL performs two functions:
4          ;   1. Set up information in the device tables about all devices.
5          ;   2. Load those handlers that reside in low memory.
6          ;
7          ; Inputs:
8          ;   R5 = Address of start of free memory.
9          ;
10         ; Outputs:
11         ;   R5 = Address of new start of free memory.
12         ;   NMXHAN = Number of handlers to load into extended memory.
13         ;
14 017730 010146 GETHNL: MOV     R1, -(SP)
15 017732 010246      MOV     R2, -(SP)
16 017734 010446      MOV     R4, -(SP)
17         ;
18         ; Begin loop to check all handlers specified in TSGEN with DEVDEF.
19         ;
20 017736 005001      CLR     R1          ; Init device table index
21 017740 020127 000000C 1$:    CMP     R1, #<AHEND-AUTHAN> ; Done all devices?
22 017744 103015      BHIS    2$          ; Br if yes
23 017746 016102 000000G      MOV     AUTHAN(R1), R2 ; Get the name of the device
24 017752 001407      BEQ     3$          ; Ignore null devices
25 017754 020227 000000G      CMP     R2, #DMYDEV      ; Is this a dummy device entry?
26 017760 001404      BEQ     3$          ; Skip it if yes
27 017762 016104 000000G      MOV     DTYPE(R1), R4 ; Get flags specified in TSGEN
28         ;
29         ; Load this handler
30         ;
31 017766 004737 020040'      CALL    LDHAND          ; Try to load handler into memory
32         ;
33         ; Check next device
34         ;
35 017772 062701 0000002 3$:    ADD     #2, R1          ; Advance device index
36 017776 000760      BR      1$          ; See if more devices to load
37         ;
38         ; Now see if there are spooled devices to contend with
39         ;
40 020000 012704 000000G 2$:    MOV     #SPLND, R4      ; Are there any spooled devices?
41 020004 001411      BEQ     9$          ; Br if not
42 020006 012701 000000G      MOV     #SPLDEV, R1     ; Point to spooled device name table
43 020012 012102 5$:    MOV     (R1)+, R2      ; Get the name of the next spooled device
44 020014 010446      MOV     R4, -(SP)      ; Save device count
45 020016 005004      CLR     R4          ; Say no TSGEN flags for device
46 020020 004737 020040'      CALL    LDHAND          ; Load the handler
47 020024 012604      MOV     (SP)+, R4      ; Recover the device count
48 020026 077407      SOB     R4, 5$          ; Loop if more handlers to load
49         ;
50         ; Finished
51         ;
52 020030 012604 9$:    MOV     (SP)+, R4
53 020032 012602      MOV     (SP)+, R2
54 020034 012601      MOV     (SP)+, R1
55 020036 000207      RETURN

```

```

1          .SBTTL  LDHAND -- Load a device handler
2          ;-----
3          ; LDHAND sets up the device tables for a handler and loads into memory
4          ; those handlers that reside in low memory.
5          ; The device interrupt vectors are NOT set up by LDHAND.
6          ;
7          ; Inputs:
8          ;   R2 = Rad-50 name of device.
9          ;   R4 = TSX-Plus DX$xxx status flags for device from TSGEN.
10         ;   R5 = Address where handler is to be loaded.
11         ;
12         ; Outputs:
13         ;   R5 = New free memory address.
14         ;   NUMDEV = Incremented by 2.
15         ;   PNAME(i) = Rad-50 name of device.
16         ;   ENTRY(i) = Handler entry point.
17         ;   DVSTAT(i) = Device status flags.
18         ;   DVFLAG(i) = TSX-Plus device status flags.
19         ;   HANPAR(i) = PAR offset if this is a mapped handler.
20         ;
21 020040 010446 LDHAND: MOV      R4, -(SP)
22         ;
23         ; Determine if we should ignore this device
24         ;
25 020042 004737 020212' CALL     INSCK1      ;Should we ignore this device?
26 020046 103457      BCS     9$          ;Br if yes
27         ;
28         ; The initial tests indicate that this handler should be loaded.
29         ; Now open the handler file and perform some additional checks.
30         ;
31 020050 004737 020370' CALL     INSCK2      ;Perform some additional checks on handler
32 020054 103451      BCS     8$          ;Br if we should not load this device
33         ;
34         ; At this point channel 1 is open to the handler file and block 0
35         ; of the handler is in WRKBUF.
36         ; Set up information tables for this device.
37         ;
38 020056 004737 021004' CALL     STDVTB      ;Set up info in tables for this device
39         ;
40         ; Determine if this handler is to be loaded into low memory or
41         ; extended memory.
42         ;
43 020062 016400 000000G MOV      DVFLAG(R4),R0 ;Get TSX-Plus status flags for device
44 020066 032700 000000G BIT      #DX$NHM,R0  ;Are we never to map this handler?
45 020072 001036      BNE     1$          ;Br if handler cannot be mapped
46 020074 032700 000000G BIT      #DX$MPH,R0  ;Is mapping wanted for this handler?
47 020100 001433      BEQ     1$          ;Br if not
48 020102 032700 000000G BIT      #DX$IBH,R0  ;Does this handler have an internal I/O buff?
49 020106 001412      BEQ     2$          ;Br if not
50 020110 105737 000000G TSTB    UBUSMP      ;Does this machine have a mapped UNIBUS?
51 020114 001025      BNE     1$          ;Br if yes -- Don't map this handler
52 020116 032700 000000G BIT      #DX$MAP,R0  ;Does this handler require I/O mapping?
53 020122 001404      BEQ     2$          ;Br if not
54 020124 032737 000001 000242' BIT      #EXTLSI,ICONFG ;Is this a Q-bus system with more than 256Kb?
55 020132 001016      BNE     1$          ;Br if yes -- Don't map this handler
56         ;
57         ; This handler can be mapped and will be loaded in extended memory

```

LDHAND -- Load a device handler

```

58 ;
59 020134 005237 000124' 2#: INC NMXHAN ;Count # of mapped handlers
60 020140 012764 000001 000000G MOV #1,HANPAR(R4) ;Set flag saying handler should be mapped
61 ;
62 ; Make sure size of mapped handler does not exceed 8KB
63 ;
64 020146 026427 000000G 020000 CMP HANSIZ(R4),#8192. ;Is mapped handler too big?
65 020154 101411 BLOS B# ;Br if not too big
66 020156 012700 002335' MOV #HN2BIG,R0 ;Get error message address
67 020162 004737 022102' CALL HLERR ;Abort initialization if iniabt
68 020166 000404 BR B#
69 ;
70 ; This handler must be loaded into low memory
71 ;
72 020170 005064 000000G 1#: CLR HANPAR(R4) ;Say this handler is not mapped
73 020174 004737 021114' CALL LDHNLO ;Load handler into low memory
74 ;
75 ; Close the handler file
76 ;
77 020200 B#: .CLOSE #1 ;Close the handler file
78 ;
79 ; Finished
80 ;
81 020206 012604 9#: MOV (SP)+,R4
82 020210 000207 RETURN

```

INSCK1 -- Determine if a handler should be installed

```

1          .SBTTL  INSCK1 -- Determine if a handler should be installed
2          ;-----
3          ;  INSCK1 is called to determe if a certain device handler should be
4          ;  loaded.
5          ;
6          ;  Inputs:
7          ;  R2 = Rad50 name of the device.
8          ;  R4 = Initial DX$xxx flags as specified in TSGEN.
9          ;
10         ;  Outputs:
11         ;  C-flag cleared ==> Load the handler.
12         ;  C-flag set    ==> Do not load the handler.
13         ;  R2 = Device name with unit number removed.
14         ;  R4 = DX$xxx combined with default flags for the device.
15         ;
16 020212 010146  INSCK1: MOV      R1, -(SP)
17         ;
18         ;  Strip off any specified unit number
19         ;
20 020214 010201          MOV      R2, R1          ;Get full device name
21 020216 005000          CLR      R0              ;Set for divide
22 020220 071027 000050  DIV      #50, R0          ;Split off last digit
23 020224 070027 000050  MUL      #50, R0          ;Now correct for divide
24 020230 010102          MOV      R1, R2          ;Get device name less 3rd digit
25 020232 010237 000146' MOV      R2, CURNAM        ;Set name of handler being loaded
26         ;
27         ;  See if this is a device such as DK, SY, or TT which we don't
28         ;  need to load as a device handler.
29         ;
30 020236 020227 045640  CMP      R2, #R50LD        ;Is device name LD?
31 020242 001004          BNE      1$              ;Br if not
32 020244 105737 000000G  TSTB   VLDSYS            ;Is standard system LD support included?
33 020250 001044          BNE      5$              ;Br if yes -- Don't load LD
34 020252 000417          BR       3$              ;Load LD
35 020254 012701 000160' 1$: MOV      #SKPDEV, R1      ;Point to table of devices to skip
36 020260 020221 2$: CMP      R2, (R1)+          ;Is this a device to be skipped?
37 020262 001437          BEQ      5$              ;Br if yes
38 020264 005711          TST     (R1)              ;Reached end of skip list?
39 020266 001374          BNE      2$              ;Loop if not
40         ;
41         ;  See if we have already loaded the handler for this device
42         ;
43 020270 013701 000000G  MOV      NUMDEV, R1        ;Get index for last device
44 020274 001406          BEQ      3$              ;Br if no devices installed yet
45 020276 020261 000000G 4$: CMP      R2, PNAME(R1)    ;See if this device is already installed
46 020302 001427          BEQ      5$              ;Br if already installed
47 020304 162701 000002  SUB      #2, R1            ;More installed devices to check?
48 020310 003372          BGT     4$              ;Loop if yes
49         ;
50         ;  This handler is to be loaded.
51         ;  Get default TSX-Plus control flags for this device.
52         ;
53 020312 012701 000336' 3$: MOV      #DVFLBS, R1      ;Point to start of table
54 020316 020261 000000 6$: CMP      R2, DV$NAM(R1)    ;Search for device in the table
55 020322 001003          BNE      7$              ;Br if this is not it
56 020324 056104 000002  BIS     DV$FLG(R1), R4     ;Combine default flags
57 020330 000405          BR       8$

```

INSCK1 -- Determine if a handler should be installed

```
58 020332 062701 000004      7$:      ADD      #DV##SZ,R1      ;Point to next entry
59 020336 020127 000516'      CMP      R1,#DVFLND      ;Checked all entries?
60 020342 103765              BLD      6$              ;Loop if not
61                          ;
62                          ; If this is a DMA device, set flag saying buffers must be on
63                          ; even byte boundaries.
64                          ;
65 020344 032704 000000G      8$:      BIT      #DX$DMA,R4      ;Is this a DMA device?
66 020350 001402              BEQ      10$              ;Br if not
67 020352 052704 000000G      BIS      #DX$EBA,R4      ;Set even-buffer-boundary flag
68                          ;
69                          ; Load this handler
70                          ;
71 020356 000241              10$:     CLC              ;Set flag saying to load the handler
72 020360 000401              BR      9$
73                          ;
74                          ; Do not load this handler
75                          ;
76 020362 000261              5$:      SEC              ;Set flag saying not to load the handler
77                          ;
78                          ; Finished
79                          ;
80 020364 012601              9$:      MOV      (SP)+,R1
81 020366 000207              RETURN
```

```

1                                     .SBTTL  INSCK2 -- Additional checking for handler installation
2                                     ;-----
3                                     ;  INSCK2 is called to determine if a device handler should be installed.
4                                     ;
5                                     ;  Inputs:
6                                     ;    R2 = Rad50 device name (without unit number).
7                                     ;
8                                     ;  Outputs:
9                                     ;    C-flag cleared ==> Load this handler.
10                                    ;    C-flag set    ==> Do not load this handler.
11                                    ;    If the handler is to be loaded, its block 0 is in WRKBUF and channel
12                                    ;    number 1 is opened to the handler file.
13                                    ;
14 020370 010146  INSCK2: MOV      R1,-(SP)
15 020372 010246      MOV      R2,-(SP)
16 020374 010346      MOV      R3,-(SP)
17 020376 010446      MOV      R4,-(SP)
18 020400 010546      MOV      R5,-(SP)
19 020402 013746 000004  MOV      @#4,-(SP)      ;Save the bus timeout vector
20 020406 013746 000010  MOV      @#10,-(SP)    ;Save illegal instruction vector
21                                    ;
22                                    ;  Try to lookup handler file on system disk
23                                    ;
24 020412 010237 000202'  MOV      R2,HANNAM+2    ;Set the device name for the lookup
25                                    ;;; Don't change channel # without changing STDVTB
26 020416          .LOOKUP #AREA,#1,#HANNAM;Try to open the handler file
27 020436 103006      BCC      1$          ;Br if we found the handler file
28                                    ;
29                                    ;  Error -- Cannot find handler file
30                                    ;
31 020440 012700 001510' 2$:  MOV      #CFHMSG,RO    ;Can't find handler
32 020444 004737 022102'  CALL     HLERR          ;See if should abort initialization
33 020450 000137 020760'  JMP      13$
34                                    ;
35                                    ;  We were able to open the handler file.
36                                    ;  Read in block 0 of handler.
37                                    ;
38 020454          1$:  .READW  #AREA,#1,WRKBUF,#256.,#0 ;Read block 0 into WRKBUF
39 020512 103006      BCC      3$          ;Br if read ok
40 020514 012700 001552'  MOV      #ERHMSG,RO    ;Error during read
41 020520 004737 022102'  CALL     HLERR
42 020524 000137 020760'  JMP      13$
43                                    ;
44                                    ;  Determine if the handler is supported under the current RT-11 version
45                                    ;
46 020530 012700 000322' 3$:  MOV      #HVTBL,RO      ;Point to table with handler version info
47 020534 013701 000152'  MOV      WRKBUF,R1      ;Point to buffer with block 0 of handler
48 020540 116101 000000G  MOVB    H.DSTS(R1),R1    ;Get device ID code from handler
49 020544 120160 000000    51$:  CMPB    R1,HV$ID(RO)    ;Compare handler ID code with table entry
50 020550 001011      BNE      53$          ;Br if this entry not for this handler
51 020552 026037 000002 000156'  CMP     HV$VER(RO),RTVPTR ;Is handler valid for this version?
52 020560 101412      BLOS    54$          ;Br if OK
53 020562 012700 001616'  MOV     #ERHNDV,RO      ;Wrong version of RT for handler
54 020566 004737 022102'  CALL    HLERR          ;See if should Report error and abort
55 020572 000472      BR      13$
56 020574 062700 000004 53$:  ADD     #HV$$SZ,RO      ;Point to next handler version table entry
57 020600 020027 000336'  CMP     RO,#HVEND      ;Are there more entries?

```

INSCK2 -- Additional checking for handler installation

```

58 020604 103757          BLD      51$          ;Loop if more to check
59
60          ; Check handler sysgen options
61
62 020606 013700 000152' 54$:  MOV      WRKBUF,RO      ;Point to buffer with handler block 0
63 020612 032760 000000G 000000G  BIT      #SG$MMU,H.GEN(RO);Was handler genned with XM support?
64 020620 001005          BNE      4$          ;Br if yes
65 020622 012700 002027'  MOV      #HSGER,RO      ;Error if not XM version of handler
66 020626 004737 022102'  CALL     HLERR
67 020632 000452          BR       13$
68 020634 016037 000000G 000240' 4$:  MOV      H.GEN(RO),HGENFL;Save handler sysgen flags for later
69
70          ; Check the CSR address specified in the handler to see if the
71          ; hardware device for this handler exists.
72
73 020642 012737 017132' 000004  MOV      #TRCSET,@#4      ;Catch bus timeout traps
74 020650 012737 017132' 000010  MOV      #TRCSET,@#10     ;Catch illegal instruction traps
75 020656 013700 000152'  MOV      WRKBUF,RO      ;Point to start of block 0 of handler
76 020662 016001 000000G  MOV      H.CSR(RO),R1    ;Get address of CSR for device
77 020666 001407          BEQ      5$          ;Br if no CSR specified
78 020670 005711          TST      (R1)          ;Is CSR accessible?
79 020672 103005          BCC      5$          ;Br if ok
80 020674 012700 001670'  MOV      #NDCSRR,RO      ;Trap occurred while accessing CSR
81 020700 004737 022102'  CALL     HLERR          ;See if should report error and abort
82 020704 000425          BR       13$
83
84          ; Execute the device installation code.
85          ; The installation code will set the C-flag if the handler should
86          ; not be loaded.
87
88 020706 062700 000000G 5$:  ADD      #H.INS,RO      ;Offset 200 in block 0
89 020712 005710          TST      @RO          ;Does any installation code exist?
90 020714 001420          BEQ      11$         ;Br if no driver installation code
91 020716 013746 000000G  MOV      @#RMON,-(SP)    ;Save RT-11 RMON pointer
92 020722 012737 000000G 000000G  MOV      #MONVEC,@#RMON ;Set TSX-Plus RMON pointer
93 020730 013703 000000G  MOV      RPRVEC,R3      ;Get pointer to Pro vec addr routine
94 020734 004710          CALL     @RO          ;Call the installation code
95 020736 012637 000000G  MOV      (SP)+,@#RMON    ;Restore RT-11 RMON pointer
96 020742 103006          BCC      13$         ;C-flag now indicates handler load status
97 020744 012700 001721'  MOV      #ERHINS,RO      ;Error occured in handler installation code
98 020750 004737 022102'  CALL     HLERR
99 020754 000401          BR       13$
100
101          ; Finished with installation verification.
102
103 020756 000241 11$:  CLC
104 020760 012637 000010 13$:  MOV      (SP)+,@#10     ;Restore illegal instruction vector
105 020764 012637 000004  MOV      (SP)+,@#4      ;Restore the bus timeout vector
106 020770 012605  MOV      (SP)+,R5
107 020772 012604  MOV      (SP)+,R4
108 020774 012603  MOV      (SP)+,R3
109 020776 012602  MOV      (SP)+,R2
110 021000 012601  MOV      (SP)+,R1
111 021002 000207  RETURN

```

STDVTB -- Set up device table entries for a device

```

1          .SBTTL  STDVTB -- Set up device table entries for a device
2          ;-----
3          ; STDVTB is called to set up device table entries for a device whose
4          ; handler is being loaded.
5          ;
6          ; Inputs:
7          ; R2 = Rad50 name of device (less unit number).
8          ; R4 = DX$xxx device flags for DVFLAG table.
9          ; Block 0 of the handler must be in WRKBUF.
10         ; Channel 1 must be open to the handler file.
11         ;
12         ; Outputs:
13         ; R4 = Device table index number for this device.
14         ; NUMDEV = Incremented by 2.
15         ; PNAME(i) = Rad50 name of the device.
16         ; DVSTAT(i) = Device status flags.
17         ; DVFLAG(i) = TSX-Plus control flags.
18         ; HANSIZ(i) = Size of handler (bytes).
19         ; DEVSIZ(i) = Size of device (blocks).
20         ;
21 021004  STDVTB:
22         ;
23         ; Increment device counter
24         ;
25 021004 062737 000002 000000G      ADD     #2,NUMDEV      ;Say another device added to tables
26 021012 013700 000000G      MOV     NUMDEV,R0      ;Get device index number
27         ;
28         ; Set up PNAME and DVFLAG.
29         ;
30 021016 010260 000000G      MOV     R2,PNAME(R0)   ;Set permanent device name
31 021022 010460 000000G      MOV     R4,DVFLAG(R0) ;Set up TSX-Plus control flags for the device
32         ;
33         ; Set HANDSK entry to 1.
34         ; This entry is supposed to hold the absolute block number on the disk where
35         ; block 1 of the handler is located. We set to 1 because all IO we do
36         ; on behalf of the handler is relative to the base of the handler rather than
37         ; relative to the start of the disk. This is critical during handler
38         ; load/fetch code.
39         ; NOTE: This table is replaced during KMINIT by the location of handler
40         ; block 1 relative to the start of the disk so that utilities can find
41         ; the handler files in the same way as under RT-11.
42         ;
43 021026 012760 000001 000000G      MOV     #1,HANDSK(R0) ;Set block 1 relative offset of handler file
44         ;
45         ; Extract parameters from handler block 0
46         ;
47 021034 010004              MOV     R0,R4          ;Carry device index in R4
48 021036 013700 000152'      MOV     WRKBUF,R0      ;Point to block 0 of handler
49 021042 016064 000000G 000000G  MOV     H.SIZ(R0),HANSIZ(R4) ;Set handler size
50 021050 016064 000000G 000000G  MOV     H.DVSZ(R0),DEVSIZ(R4) ;Number of blocks on device
51 021056 016064 000000G 000000G  MOV     H.DSTS(R0),DVSTAT(R4) ;Set device status flags
52         ;
53         ; Disable MOUNTs and data caching for certain devices
54         ;
55 021064 032764 000000G 000000G  BIT     #DS$DIR,DVSTAT(R4) ;Is this a directory structured device?
56 021072 001404              BEQ     1$              ;Br if not -- No mounts allowed
57 021074 032764 000000G 000000G  BIT     #DS$NRD,DVSTAT(R4) ;Non RT-11 directory structure (mag tape)?

```

STDVTB -- Set up device table entries for a device

```
58 021102 001403          BEQ      9#          ;Br if not
59 021104 052764 000000C 000000G 1#:    BIS      #<DX$NMT!DX$NCA>,DVFLAG(R4) ;Disable mounts and data caching
60                                     ;
61                                     ; Finished
62                                     ;
63 021112 000207          9#:      RETURN
```

```

1                                     .SBTTL LDHNL0 -- Load device handler into low memory
2                                     ;-----
3 LDHNL0 is called to load a device handler into low memory.
4
5 Inputs:
6 R4 = Device index number.
7 R5 = Address of start of free memory area.
8
9 Outputs:
10 R5 = Address of new start of free memory area.
11
12 021114 010346 LDHNL0: MOV R3, -(SP)
13
14 ; Determine if we have enough free memory space to read the handler
15 ;
16 021116 005064 000000G CLR HANPAR(R4) ; Say this handler is not mapped
17 021122 010500 MOV R5, R0 ; Get current top of memory address
18 021124 016403 000000G MOV HANSIZ(R4), R3 ; Get size of handler
19 021130 060300 ADD R3, R0 ; Get address above top of handler
20 021132 004737 027630' CALL CHKMEM ; See if handler will fit in memory
21
22 ; Handler will fit. Read it into memory.
23 ;
24 021136 006203 ASR R3 ; Get number of words to read
25 021140 . READW #AREA, #1, R5, R3, #1
26 021174 103005 BCC 1$ ; Br if read ok
27 021176 012700 001552' MOV #ERHMSG, R0 ; Error reading handler
28 021202 004737 022102' CALL HLERR ; See if should Abort initialization
29 021206 000414 BR 2$
30
31 ; Set address of handler entry point and compute address beyond
32 ; end of the handler.
33 ;
34 021210 010564 000000G 1$: MOV R5, HANENT(R4) ; Set address of handler entry point
35 021214 062764 000006 000000G ADD #6, HANENT(R4) ; (Point to fourth word of handler)
36 021222 006303 ASL R3 ; Convert handler size to bytes
37 021224 060305 ADD R3, R5 ; Point beyond end of handler
38
39 ; Set up table of addresses of support routines at end of handler.
40 ;
41 021226 010503 MOV R5, R3 ; Get address past end of handler
42 021230 004737 022010' CALL STHNPV ; Set up pointer vector in handler
43
44 ; If handler has any load-time execution code, run it now
45 ;
46 021234 004737 022152' CALL DOHNL0 ; Run any load-time code for handler
47
48 ; Finished
49 ;
50 021240 012603 2$: MOV (SP)+, R3
51 021242 000207 RETURN

```

```

1          .SBTTL  GETHNH -- Load handlers into extended memory
2          ;-----
3          ; GETHNH is called to load those handlers that can be placed in extended
4          ; memory. The status tables for these devices have already been set up
5          ; by GETHNL.
6          ;
7          ; Inputs:
8          ;   R5 = 64-byte block number of top of free memory area.
9          ;
10         ; Outputs:
11         ;   R5 = 64-byte block number of new top of free memory area.
12         ;
13 021244 010446  GETHNH: MOV      R4, -(SP)
14         ;
15         ; Begin looking for handlers that are to be loaded into extended memory.
16         ; GETHNL stored a non-zero (but meaningless) value in the HANPAR entry
17         ; for each handler that is to be mapped.
18         ;
19 021246 012704 000002      MOV      #2, R4          ;Get index for first device entry
20         ;
21         ; See if this device has a mapped handler
22         ;
23 021252 005764 000000G 1$:   TST      HANPAR(R4)      ;Is this handler mapped?
24 021256 001402          BEQ      2$              ;Br if not
25         ;
26         ; We found an entry for a device with a mapped handler.
27         ; Load the handler.
28         ;
29 021260 004737 021302'      CALL     LDHNHI          ;Load a mapped handler
30         ;
31         ; Look for more mapped handlers
32         ;
33 021264 062704 000002 2$:   ADD      #2, R4          ;Increment device index
34 021270 020437 000000G      CMP      R4, NUMDEV      ;Checked all devices?
35 021274 101766          BLOS    1$              ;Loop if not
36         ;
37         ; Finished
38         ;
39 021276 012604          MOV      (SP)+, R4
40 021300 000207          RETURN

```

LDHNHI -- Load device handler into extended memory

```

1          .SBTTL  LDHNHI -- Load device handler into extended memory
2          ;-----
3          ; LDHNHI is called to load a device handler into extended memory.
4          ;
5          ; Inputs:
6          ; R4 = Device index number.
7          ; R5 = 64-byte block number of top of free memory area.
8          ;
9          ; Outputs:
10         ; R5 = 64-byte block number of new top of free memory area.
11         ;
12 021302 010146 LDHNHI: MOV      R1,-(SP)
13 021304 010246      MOV      R2,-(SP)
14 021306 010346      MOV      R3,-(SP)
15 021310 010446      MOV      R4,-(SP)
16         ;
17         ; Open channel 1 to the handler file.
18         ;
19 021312 016437 000000G 000202'      MOV      PNAME(R4),HANNAM+2 ;Set the device name for the lookup
20 021320 016437 000000G 000146'      MOV      PNAME(R4),CURNAM;Set name in case we have an error
21 021326          .LOOKUP #AREA,#1,#HANNAM;Try to open the handler file
22 021346 103006      BCC      8$          ;Br if we found the handler file
23 021350 012700 001510'      MOV      #CFHMSG,R0          ;Can't find handler
24 021354 004737 022102'      CALL     HLERR          ;See if should Abort initialization
25 021360 000137 021770'      JMP      9$
26         ;
27         ; Read block 0 of the handler file and extract some information
28         ;
29 021364 013702 000152'      8$:    MOV      WRKBUF,R2          ;Get address of work buffer
30 021370          .READW #AREA,#1,R2,#256.,#0 ;Read block 0 of handler
31 021424 016237 000000G 000240'      MOV      H.GEN(R2),HGENFL;Save handler sysgen flags
32         ;
33         ; Set virtual address of handler entry point
34         ;
35 021432 012764 000006G 000000G      MOV      #VPAR5+6,HANENT(R4) ;Set virtual addr of handler entry point
36         ;
37         ; Get information about the size of the handler and determine the
38         ; address in extended memory where the handler is to be loaded.
39         ;
40 021440 016402 000000G      MOV      HANSIZ(R4),R2      ;Get size of handler (bytes)
41 021444 005202          INC      R2          ;Make sure handler size is even
42 021446 042702 000001      BIC      #1,R2
43 021452 010200          MOV      R2,R0
44 021454 062700 000077      ADD      #63.,R0          ;Round up to # 64-byte blocks
45 021460 072027 177772      ASH      #-6.,R0          ;Get # 64-byte blocks for handler
46 021464 060037 000000G      ADD      R0,MHNSIZ        ;Accumulate total space for mapped handlers
47 021470 160005          SUB      R0,R5          ;Reserve room for handler
48 021472 010564 000000G      MOV      R5,HANPAR(R4)    ;Set mapping value for handler
49 021476 010537 000126'      MOV      R5,HMAP          ;Set initial PAR base for handler
50 021502 012737 000001 000142'      MOV      #1,FILBLK        ;Set # of block to read from file
51         ;
52         ; Begin loop to read handler into memory
53         ;
54 021510 010203          1$:    MOV      R2,R3          ;Get remaining size of handler
55 021512 020327 001000      CMP      R3,#512.        ;Compare with max we can read at one time
56 021516 101402          BLOS     2$          ;Br if we can read remainder of handler
57 021520 012703 001000      MOV      #512.,R3        ;Read one block

```

LDHNHI -- Load device handler into extended memory

```

58 021524 160302          2$:      SUB      R3,R2          ;Reduce amt of handler left to read
59                      ;
60                      ; Read next block of handler
61                      ;
62 021526 006203          ASR      R3          ;Get # words to read
63 021530 013701 000152'  MOV      WRKBUF,R1    ;Get address of buffer for read
64 021534          . READW #AREA,#1,R1,R3,FILBLK ;Read a block
65 021570 103005          BCC     3$          ;Br if read ok
66 021572 012700 001552'  MOV      #ERHMSG,R0   ;Get error message
67 021576 004737 022102'  CALL    HLERR        ;See if should Abort initialization
68 021602 000472          BR      9$          ;
69                      ;
70                      ; Move the code we just read into the XM area for the handler
71                      ;
72 021604 012700 000000G   3$:      MOV      #VPAR5,R0    ;Get virtual address of mapped region
73 021610          DISABL          ;** Disable interrupts **
74 021616 013746 000000G   MOV      @#KPAR5,-(SP) ;;; Save current mapping of PAR 5
75 021622 013737 000126' 000000G   MOV      HMAP,@#KPAR5 ;;; Set up mapping to get to XM area
76 021630 052737 000000G 000000G   BIS      #MMENBL,@#SR0MMR ;;; Enable memory management
77 021636 105737 000000G   TSTB    MEM256       ;;; Does machine have > 256KB?
78 021642 001403          BEQ     4$          ;;; Br if not
79 021644 052737 000000G 000000G   BIS      #EMMAP,@#SR3MMR ;;; Enable extended memory addressing
80 021652 012120          4$:      MOV      (R1)+,(R0)+    ;;; Move from WRKBUF to XM region
81 021654 077302          SOB     R3,4$       ;;; Loop till all moved
82 021656 105737 000000G   TSTB    MEM256       ;;; Does machine have > 256KB?
83 021662 001403          BEQ     5$          ;;; Br if not
84 021664 042737 000000G 000000G   BIC      #EMMAP,@#SR3MMR ;;; Disable extended memory addressing
85 021672 042737 000000G 000000G   5$:      BIC      #MMENBL,@#SR0MMR ;;; Enable memory management
86 021700 012637 000000G   MOV      (SP)+,@#KPAR5 ;;; Replace PAR 5 mapping
87 021704          ENABL          ; ** Enable interrupts **
88                      ;
89                      ; See if there is more to read
90                      ;
91 021712 062737 000010 000126'  ADD     #8,HMAP       ; Increase XM region base
92 021720 005237 000142'  INC     FILBLK        ; Increment file block number
93 021724 005702          TST     R2           ; Is there more to read?
94 021726 001270          BNE     1$          ; Loop if more to read
95                      ;
96                      ; We have finished moving the handler into its XM region.
97                      ; Set up addresses of system routines in a vector at the end of the handler
98                      ;
99 021730 012703 000000G   MOV     #VPAR5,R3     ;Get virtual address of handler base
100 021734 066403 000000G   ADD     HANSIZ(R4),R3 ;Get virtual address beyond end of handler
101 021740 004737 022652'  CALL    HANMAP        ;;; Map KPAR5 to the handler
102 021744 004737 022010'  CALL    STHNPV        ;;; Set up handler pointer vector
103 021750 004737 022726'  CALL    HANUMP        ;Restore mapping
104                      ;
105                      ; If handler has any load-time code, run it now
106                      ;
107 021754 010537 000134'  MOV     R5,FMEMHI     ;Set addr of top of free memory area
108 021760 004737 022152'  CALL    DOHNLC        ;Run any load-time code for handler
109 021764 013705 000134'  MOV     FMEMHI,R5     ;Get new top of free memory address
110                      ;
111                      ; Close the handler file
112                      ;
113 021770          9$:      .CLOSE #1          ;Close the handler file
114                      ;

```

```
115          ; Finished  
116          ;  
117 021776 012604      MOV      (SP)+, R4  
118 022000 012603      MOV      (SP)+, R3  
119 022002 012602      MOV      (SP)+, R2  
120 022004 012601      MOV      (SP)+, R1  
121 022006 000207      RETURN
```

STHNPV -- Initialize pointer vector in a handler

```

1          .SBTTL  STHNPV -- Initialize pointer vector in a handler
2          ;-----
3          ; STHNPV is called to initialize the pointer vector at the end of a
4          ; handler which provides the addresses of various system routines to the
5          ; handler.
6          ;
7          ; Inputs:
8          ;   R3 = Address beyond the end of the handler.
9          ;   HGENFL = Sysgen option flags for the handler being loaded.
10         ;
11 022010   010346   STHNPV: MOV      R3, -(SP)
12         ;
13         ; Set up addresses in the pointer vector
14         ;
15 022012   012743   000000G      MOV      #FORK, -(R3)      ;Address of fork routine
16 022016   012743   000000G      MOV      #INTEN, -(R3)     ;Address of inten routine
17 022022   032737   000000G 000240' BIT      #SG$IOT, HGENFL ;Does handler want timeout support?
18 022030   001402           BEQ      2$              ;Br if not
19 022032   012743   000000G      MOV      #IOTIMR, -(R3)   ;Set address of timeout support routine
20 022036   032737   000000G 000240' 2$: BIT      #SG$ELG, HGENFL ;Does handler want error logging support?
21 022044   001402           BEQ      3$              ;Br if not
22 022046   012743   000000G      MOV      #ERRLOG, -(R3)   ;Set address of error logging routine
23 022052   012743   000000G      3$: MOV      #PTWRD, -(R3)
24 022056   012743   000000G      MOV      #PTBYT, -(R3)
25 022062   012743   000000G      MOV      #GTBYT, -(R3)
26 022066   012743   000000G      MOV      #MPPHY, -(R3)
27 022072   012743   000000G      MOV      #RELOC, -(R3)
28         ;
29         ; Finished
30         ;
31 022076   012603           MOV      (SP)+, R3
32 022100   000207           RETURN

```

STHNPV -- Initialize pointer vector in a handler

```

1          ;
2          ; Error occured while loading device handler.
3          ; RO = error message address; CURNAM = device name.
4          ;
5 022102 010001 HLERR: MOV RO,R1 ;SAVE ERROR MESSAGE ADDRESS
6 022104 105737 000000G TSTB VINABT ;ABORT OR JUST PRINT MESSAGE?
7 022110 001416 BEQ 9# ;BR IF NOT ABORT
8 022112 .PRINT #TSXHD ;PRINT ERROR MESSAGE HEADING
9 022120 .PRINT R1 ;PRINT ERROR MESSAGE
10 022124 013700 000146' MOV CURNAM,R0 ;GET RAD50 DEVICE NAME
11 022130 004737 030020' CALL PRTR50 ;PRINT DEVICE NAME
12 022134 .PRINT #CRLF
13 022142 000137 004102' JMP INISTP ;ABORT INITIALIZATION
14 022146 000261 9#: SEC ;MAKE SURE CARRY IS SET
15 022150 000207 RETURN

```

DOHNLC -- Execute and handler load/fetch code

```

1
2
3
4
5
6
7
8
9
10
11
12
13 022152 010146
14 022154 010246
15 022156 010346
16 022160 010446
17 022162 010546
18
19
20
21 022164 016405 0000009
22 022170 004737 022652'
23 022174 016500 000004
24 022200 004737 022726'
25 022204 020027 000240
26 022210 103516
27 022212 020027 000277
28 022216 101113
29 022220 132700 000004
30 022224 001510
31
32
33
34
35 022226 013702 000152'
36 022232
37 022266 022227 031066
38 022272 001065
39 022274 016203 000004
40 022300 001462
41
42
43
44
45 022302 020327 001000
46 022306 103424
47 022310 010302
48 022312 072227 177767
49 022316 042702 177400
50 022322
51
52
53
54 022360 010437 000144'
55 022364 042703 177000
56 022370 010300
57 022372 063700 000152'

```

```

.SBTTL DOHNLC -- Execute and handler load/fetch code
-----
; If the handler being loaded has any Load-time execution code, read it
; into our work buffer and execute it now.
;
; Inputs:
; R4 = Device index number of handler that is being loaded.
;
; Outputs:
; C-flag is set on return if load code signals an error during its
; execution.
;
DOHNLC: MOV R1, -(SP)
MOV R2, -(SP)
MOV R3, -(SP)
MOV R4, -(SP)
MOV R5, -(SP)
;
; Examine 1st word of handler to see if it could have any load-time code.
;
MOV HANENT(R4), R5 ; Get address of handler entry point
CALL HANMAP ; Map Kpar5 to handler if mapped handler
MOV 4(R5), R0 ; Get 1st instruction located at 4 in handler
CALL HANUMP ; Restore normal mapping
CMP R0, #240 ; Is it a NOP?
BLO 7$ ; Br if can't be any load code
CMP R0, #277 ;
BHI 7$ ; Br if can't be any load code
BITB #4, R0 ; Is there load code?
BEQ 7$ ; Br if not
;
; Handler may have load code.
; Read block 0 of handler and get offset to load code.
;
MOV WRKBUF, R2 ; Get addr of our work buffer
.READW #AREA, #1, R2, #256, #0 ; Read block 0 of handler
CMP (R2)+, #^RHAN ; Is this a new type handler?
BNE 7$ ; Br if not
MOV 4(R2), R3 ; Get offset to load code
BEQ 7$ ; Br if there is none
;
; There is load-time code.
; Read into WRKBUF the portion of the handler with the load code.
;
CMP R3, #1000 ; Is load code in block 0 of handler?
BLO 1$ ; Br if yes
MOV R3, R2 ; Get offset to start of load code
ASH #-9, R2 ; Convert to a block number
BIC #^C377, R2 ; Clear all but block number
.READW #AREA, #1, WRKBUF, #512, R2 ; Read 2 blocks from handler file
;
; The load code is now in WRKBUF. Set up and execute it.
;
1$: MOV R4, CURDEV ; Save current device index number
BIC #^C777, R3 ; Get offset within block of load code entry pt
MOV R3, R0 ; Get entry point offset
ADD WRKBUF, R0 ; Add base address

```

```

58 022376 013701 000000G      MOV      RPRVEC,R1      ;Get pointer to GETVEC routine for Pro
59 022402 012702 000000C      MOV      #MAXDEV*2,R2   ;Get 2## entries in device tables
60 022406 012703 0000004      MOV      #4,R3          ;Set code saying this is load code
61 022412 012705 000000G      MOV      #HANENT,R5     ;Point to handler entry address vector
62 022416 060405              ADD      R4,R5          ;Point to entry cell for this handler
63 022420 004737 022652'      CALL    HANMAP          ;;;Map Kpar5 to handler if it is a mapped
64 022424 012704 022502'      MOV      #LDREAD,R4    ;;;Get pointer to Read routine
65 022430 004710              CALL    (R0)            ;;;Execute the load code
66 022432 103407              BCS     2#              ;;;Br if handler load code signaled an error
67                               ;
68                               ; Fetch/load code ran ok.
69                               ; Turn off handler mapping.
70                               ;
71 022434 012737 001400 000000G  MOV      #1400,@#KPAR6   ;;;Restore original mapping for RT-11
72 022442 004737 022726'      CALL    HANUMP          ;Unmap the handler
73 022446 000241              7#:    CLC              ;Clear the carry flag for return
74 022450 000406              BR      9#
75                               ;
76                               ; Error occurred in fetch/load code
77                               ;
78 022452 012737 001400 000000G 2#:  MOV      #1400,@#KPAR6   ;;;Restore original mapping for RT-11
79 022460 004737 022726'      CALL    HANUMP          ;Unmap the handler
80 022464 000261              SEC              ;Set the carry flag for return
81                               ;
82                               ; Finished
83                               ;
84 022466 012605              9#:    MOV      (SP)+,R5
85 022470 012604              MOV      (SP)+,R4
86 022472 012603              MOV      (SP)+,R3
87 022474 012602              MOV      (SP)+,R2
88 022476 012601              MOV      (SP)+,R1
89 022500 000207              RETURN

```

LDREAD -- Perform I/O for handler load code

```

1          .SBTTL  LDREAD -- Perform I/O for handler load code
2          ;-----
3          ; This routine performs Read operations for handler load code.
4          ; It simulates the operation of the bootstrap read routine.
5          ; When called, Channel 1 must be open to the handler file.
6          ;
7          ; Inputs:
8          ;   R0 = Block number within handler file to be read.
9          ;   R1 = Number of words to read.
10         ;   R2 = Buffer address
11         ;
12         ; Outputs:
13         ;   C-flag is set if a read error occurs
14         ;
15 022502 010046 LDREAD: MOV     R0, -(SP)
16 022504 010446      MOV     R4, -(SP)      ;;
17 022506 010004      MOV     R0, R4      ;; Get starting block number
18         ;
19         ; Save current mapping information
20         ;
21 022510 105737 000000G      TSTB   MEM256      ;; Does machine have > 256?
22 022514 001402      BEQ     1$          ;; Br if not
23 022516 013746 000000G      MOV     @#SR3MMR, -(SP)  ;; Save extended memory address register
24 022522 013746 000000G      MOV     @#SR0MMR, -(SP)  ;; Save memory mapping
25 022526 013746 000000G      MOV     @#KPAR5, -(SP)  ;; Save current KPAR5 mapping
26 022532 013746 000000G      MOV     @#KPAR6, -(SP)  ;; Save current KPAR6 mapping
27 022536 012737 001400 000000G      MOV     #1400, @#KPAR6  ;; Restore original mapping for RT-11
28         ;
29         ; Turn off handler mapping
30         ;
31 022544 004737 022726'      CALL   HANUMP      ; Turn off handler mapping
32         ;
33         ; Read the requested data from the handler
34         ;
35 022550          .READW  #AREA, #1, R2, R1, R4 ; Read the blocks
36 022602 103420      BCS     9$          ; Br if read error
37         ;
38         ; Restore handler mapping
39         ;
40 022604 013704 000144'      MOV     CURDEV, R4      ; Get current device index
41 022610 004737 022652'      CALL   HANMAP      ;; Map Kpar5 if necessary
42         ;
43         ; Restore mapping information
44         ;
45 022614 012637 000000G      MOV     (SP)+, @#KPAR6  ;; Restore KPAR6 mapping
46 022620 012637 000000G      MOV     (SP)+, @#KPAR5  ;; Restore KPAR5 mapping
47 022624 012637 000000G      MOV     (SP)+, @#SR0MMR  ;; Restore memory mapping
48 022630 105737 000000G      TSTB   MEM256      ;; Does machine have > 256?
49 022634 001402      BEQ     2$          ;; Br if not
50 022636 012637 000000G      MOV     (SP)+, @#SR3MMR  ;; Restore extended memory address register
51         ;
52         ; Finished
53         ;
54 022642 000241      2$:   CLC          ;; Signal success on return
55 022644 012604      9$:   MOV     (SP)+, R4
56 022646 012600      MOV     (SP)+, R0
57 022650 000207      RETURN

```

HANMAP -- Set up KPAR5 to access a mapped handler

```

1          .SBTTL  HANMAP -- Set up KPAR5 to access a mapped handler
2          ;-----
3          ; This routine is called to determine if a handler is mapped and if so
4          ; to turn on mapping and set up KPAR5 to access the mapped handler.
5          ; If the handler is not mapped, mapping is not turned on and KPAR5 is
6          ; not altered.
7          ; Interrupts are left disabled by this routine.
8          ; In addition to setting up mapping, this routine also changes the RMON
9          ; pointer to point to the TSX-Plus simulated RMON vector.
10         ;
11         ; Inputs:
12         ;   R4 = Device index number
13         ;
14 022652  HANMAP:
15         ;
16         ;   Disable interrupts
17         ;
18 022652          DISABL          ;; Disable interrupts
19         ;
20         ;   Change RMON pointer to point to TSX-Plus vector
21         ;
22 022660 012737 000000G 000000G          MOV      #MONVEC,@#RMON ;; Say TSX-Plus is the monitor
23         ;
24         ;   See if this handler is mapped
25         ;
26 022666 005764 000000G          TST      HANPAR(R4) ;; Is this handler mapped?
27 022672 001403          BEQ      9$      ;; Br if not
28         ;
29         ;   This handler is mapped.
30         ;   Set up mapping to access it.
31         ;
32 022674 016437 000000G 000000G          MOV      HANPAR(R4),@#KPAR5;; Map KPAR5 to the handler code
33 022702 052737 000000G 000000G 9$:    BIS      #MMENBL,@#SR0MMR;; Enable memory mapping
34 022710 105737 000000G          TSTB   MEM256          ;; Does machine have > 256KB?
35 022714 001403          BEQ      10$     ;; Br if not
36 022716 052737 000000G 000000G          BIS      #EMMAP,@#SR3MMR ;; Enable extended memory addressing
37         ;
38         ;   Finished
39         ;
40 022724 000207          10$:    RETURN
41         ;
42          .SBTTL  HANUMP -- Turn off memory mapping to a handler
43          ;-----
44         ; This routine is the companion to HANMAP. It turns off memory mapping
45         ; and restores KPAR5 to its normal mapping value.
46         ; Enter with interrupts disabled. Interrupts are enabled on return.
47         ; This routine also changes the RMON pointer back to RT-11.
48         ;
49 022726  HANUMP:
50         ;
51         ;   Turn off memory management
52         ;
53 022726 105737 000000G          TSTB   MEM256          ;; Does machine have > 256KB?
54 022732 001403          BEQ      1$      ;; Br if not
55 022734 042737 000000G 000000G          BIC      #EMMAP,@#SR3MMR ;; Turn off extended memory addressing
56 022742 042737 000000G 000000G 1$:    BIC      #MMENBL,@#SR0MMR;; Turn off memory mapping
57 022750 012737 001200 000000G          MOV      #1200,@#KPAR5 ;; Reset KPAR5 to its normal mapping

```

```
58 ;  
59 ; Restore RMON pointer to RT11  
60 ;  
61 022756 013737 000046' 000000G      MOV      RTMNVC,@#RMON      ;;;Reset RMON pointer  
62 ;  
63 ; Enable interrupts  
64 ;  
65 022764      ENABL      ;Enable interrupts  
66 ;  
67 ; Finished  
68 ;  
69 022772 000207      RETURN
```

```

1          .IF      EQ,<PROASM-1> ;No handler XM support if Pro-only
2          .SBTTL   FNDHRB,HANXMR Inoperative Pro versions
3          ;-----
4          ; None of the current device handlers on the Pro require handler XM
5          ; region support. Make sure they return intelligent errors if
6          ; someone does try to use them.
7          ;
8          .ENABL   LSB
9          FNDHRB: CLR      R1          ;Point to next region control block (none)
10         BR       1$                ;Go signal error and return
11         ;
12         HANXMR: CLR      R2          ;Return largest possible region size (none)
13         1$:      SEC          ;Signal error on request
14         RETURN
15         .DSABL   LSB
16         ;
17         .IFF     ;EQ,<PROASM-1> ;Include handler XM support for 11 versions
18         .SBTTL   FNDHRB -- Try to find a handler global region
19         ;-----
20         ; This routine is called to try to locate an allocated XM region with
21         ; a specified name.
22         ; If a region control block with the specified name cannot be found,
23         ; the address of a free one is returned and the specified name is stored
24         ; into the free block.
25         ;
26         ; Inputs:
27         ; R5 = Pointer to 2-word cell containing Rad50 name of region to be found.
28         ;
29         ; Outputs:
30         ; C-flag cleared ==> Found the specified RCB.
31         ; R1 = Address of the RCB
32         ; C-flag set ==> Could not find the specified RCB.
33         ; R1 = Pointer to a free RCB or 0 if no available RCB's.
34         ;
35 022774 010246 FNDHRB: MOV      R2,-(SP)
36         ;
37         ; Search for specified RCB and also remember if we see a free RCB
38         ;
39 022776 005002          CLR      R2          ;Say no free RCB found
40 023000 013701 0000009  MOV      HANRCB,R1        ;Point to start of RCB area
41 023004 005721          TST      (R1)+        ;Skip over -1 word at front
42 023006 005711 1$:     TST      (R1)          ;What is the status of this RCB
43 023010 001002          BNE      2$          ;Br if this is not a free RCB
44 023012 010102          MOV      R1,R2          ;Remember address of a free RCB
45 023014 000412          BR       3$
46 023016 021127 177777 2$:     CMP      (R1), #-1        ;Are we at the end of the list?
47 023022 001412          BEQ      4$          ;Br if yes
48 023024 021561 000006  CMP      (R5),6(R1)        ;Compare the names
49 023030 001004          BNE      3$          ;Br if don't match
50 023032 026561 000002 000010 CMP      2(R5),10(R1)     ;Compare 2nd half of name
51 023040 001417          BEQ      6$          ;Br if found the RCB we were searching for
52 023042 062701 000012 3$:     ADD      #12,R1        ;Point to the next RCB
53 023046 000757          BR       1$          ;Continue searching
54         ;
55         ; We could not find the specified RCB.
56         ; If there was a free one, initialize the name.
57         ;

```

FNDHRB -- Try to find a handler global region

```
58 023050 010201          4$:   MOV     R2,R1      ;Was there a free RCB?
59 023052 001410          BEQ     5$           ;Br if not
60 023054 011561 000006    MOV     (R5),6(R1)   ;Set name in the RCB
61 023060 016561 000002 000010  MOV     2(R5),10(R1) ; (2nd word of name)
62 023066 063761 000144' 000010  ADD     CURDEV,10(R1) ;Make name unique to device
63 023074 000261          5$:   SEC             ;Signal that we did not find the RCB
64 023076 000401          BR      9$
65                          ;
66                          ; We found the RCB
67                          ;
68 023100 000241          6$:   CLC             ;Signal success on return
69                          ;
70                          ; Finished
71                          ;
72 023102 012602          9$:   MOV     (SP)+,R2
73 023104 000207          RETURN
```

22774
22152

622

HANXMR -- Allocate XM region during handler load

```

1          .SBTTL  HANXMR -- Allocate XM region during handler load
2          ;-----
3          ; This routine can be called by a handler as its is being loaded to
4          ; allocate an XM region for the handler.
5          ;
6          ; Inputs:
7          ;   R2 = Number of 64-byte units needed for XM region
8          ;
9          ; Outputs:
10         ;   C-flag cleared ==> Successfully allocated a region
11         ;   R1 = 64-byte address of base of allocated region
12         ;   R2 = Requested size
13         ;   C-flag set ==> Could not allocate the region
14         ;   R2 = Largest available region size
15         ;
16         ; Notes: FMEMLO and FMEMHI are used by this routine to indicate the
17         ; bottom and top of the free memory area that can be allocated.
18         ; The allocation is done from the top of free memory downward.
19         ; FMEMHI is updated to have the new top of free memory after the
20         ; allocation has been done.
21         ;
22 023106 010046 HANXMR: MOV      RO,-(SP)
23         ;
24         ; Get the total amount of free memory space available now
25         ; and see if the requested region can be allocated.
26         ;
27 023110 013700 000134'      MOV      FMEMHI,RO      ;Top of free memory
28 023114 163700 000136'      SUB      FMEMLO,RO      ;-Base of free memory
29 023120 020200              CMP      R2,RO      ;Do we have room for the requested region?
30 023122 101006              BHI      B$      ;Br if not
31         ;
32         ; There is room for the region so allocate it from the top of memory
33         ;
34 023124 160237 000134'      SUB      R2,FMEMHI      ;Allocate the region
35 023130 013701 000134'      MOV      FMEMHI,R1      ;Return the address of the base of the region
36 023134 000241              CLC      ;Signal success on return
37 023136 000402              BR      9$
38         ;
39         ; We are not able to allocate the region.
40         ; Return with C-flag set and the size of the largest possible region in R2.
41         ;
42 023140 010002 B$:      MOV      RO,R2      ;Get the size of the largest possible region
43 023142 000261              SEC      ;Signal failure on return
44         ;
45         ; Finished
46         ;
47 023144 012600 9$:      MOV      (SP)+,RO
48 023146 000207              RETURN
49         ;
50         . ENDC      ;EQ, <PRODASM-1>

```

```

1          .IF      NE,<PROASM-1> ;If assembling for PDP-11
2          .SBTTL  SETMIO -- Set up information about mapped devices
3          ;-----
4          ; SETMIO is called to set up information about which devices have to have
5          ; their I/O mapped through system buffers. I/O mapping is done for DMA
6          ; devices with 18-bit controllers being used on Q-bus systems with more
7          ; than 256Kb of memory.
8          ; The DX#MAP flag is set in the DVFLAG word for devices that require mapping.
9          ;
10         ; Inputs:
11         ; R5 = Pointer to low-memory free area
12         ;
13         ; Outputs:
14         ; MIOFLAG = 0==>I/O mapping not required for any device;
15         ;             1==>I/O mapping required for some device.
16         ; R5 = Pointer to new low-memory free area.
17         ;
18 023150 010146 SETMIO: MOV      R1,-(SP)
19         ;
20         ; Determine if this machine requires mapping at all
21         ;
22 023152 005727 000000G          TST      #MIODBG          ;Are we debugging mapped I/O system?
23 023156 001017                  BNE      2$              ;Br if yes
24 023160 032737 000001 000242'   BIT      #EXTLSI,ICONFG ;Is this a Q-bus machine with more than 256Kb?
25 023166 001013                  BNE      2$              ;Br if yes
26         ;
27         ; This is not a Q-bus system with more than 256Kb.
28         ; Mapping is not required at all.
29         ;
30 023170 012701 000002          MOV      #2,R1          ;Get initial device index number
31 023174 042761 000000G 000000G 1$: BIC      #DX#MAP,DVFLAG(R1) ;Clear mapped-I/O flag
32 023202 062701 000002          ADD      #2,R1          ;Get next device index
33 023206 020137 000000G          CMP      R1,NUMDEV      ;More to do?
34 023212 101770                  BLOS    1$              ;Br if yes
35 023214 000464                  BR       9$
36         ;
37         ; This is a Q-bus system with more than 256Kb.
38         ; See if any devices have requested mapped I/O.
39         ;
40 023216 005000 2$:          CLR      R0              ;Clear composite flag word
41 023220 012701 000002          MOV      #2,R1          ;Initialize device index number
42 023224 056100 000000G 3$:  BIS      DVFLAG(R1),R0 ;Combine flags from all devices
43 023230 062701 000002          ADD      #2,R1          ;Get next device index number
44 023234 020137 000000G          CMP      R1,NUMDEV      ;Checked all devices?
45 023240 101771                  BLOS    3$              ;Br if not
46 023242 032700 000000G          BIT      #DX#MAP,R0     ;Does any device require mapping?
47 023246 001447                  BEQ     9$              ;Br if not
48         ;
49         ; I/O mapping is required
50         ;
51 023250 105237 000000G          INCB    MIOFLG          ;Remember that mapping is required
52         ;
53         ; Zero the area where we will build the control structures
54         ;
55 023254 113701 000000G          MOVB   VMIOBF,R1        ;Get # buffers wanted
56 023260 070127 000000G          MUL    #MI##SZ,R1      ;Times size for each control block
57 023264 062701 000000C          ADD    #MIONWB*MW##SZ,R1 ;Add space for MIO wait blocks

```



```

1          .SBTTL  OVLPOS -- Determine which overlays go over TSINIT
2          ;-----
3          ; OVLPOS is called to determine which system overlays are to be placed
4          ; over the TSINIT code (specifically, between @OVLBAS and INITOP).
5          ;
6          ; Inputs:
7          ;   R5 = Base address in TSINIT where overlays may be loaded.
8          ;
9          ; Outputs:
10         ;   Overlay segment information is set up in OSTABL.
11         ;   OS$FLG(seg) = 0==>Load seg into high memory; 1==>Load over TSINIT.
12         ;   OVLBAS = Address of location within TSINIT where overlays start.
13         ;   R5 = Pointer past last overlay loaded over TSINIT.
14         ;
15 023372 010146 OVLPOS: MOV     R1, -(SP)
16 023374 010246      MOV     R2, -(SP)
17 023376 010346      MOV     R3, -(SP)
18 023400 010446      MOV     R4, -(SP)
19 023402 010504      MOV     R5, R4          ;Get address where we may load overlays
20         ;
21         ; Build the table that holds information about the overlays
22         ;
23 023404 004737 023602' CALL    OVLBLD          ;Build overlay information table
24         ;
25         ; First determine how much space will be used by those overlays that are
26         ; forced to be loaded over TSINIT.
27         ;
28 023410 062704 000077      ADD     #63, R4          ;Bound address to 64-byte boundary
29 023414 042704 000077      BIC     #77, R4
30 023420 010437 000140'      MOV     R4, OVLBAS      ;Remember address where we load overlays
31 023424 012702 000516'      MOV     #OSTABL, R2     ;Point to start of table
32 023430 005762 000000      1$:    TST     OS$SIZ(R2) ;Is this overlay to be loaded?
33 023434 001423              BEQ     2$              ;Br if not
34 023436 016200 000004      MOV     OS$OVL(R2), R0  ;Point to linker-built entry
35 023442 016000 000000G      MOV     0, ADR(R0), R0 ;Get Rad50 segment ID
36 023446 012701 000746'      MOV     #LOWOVL, R1    ;Point to table of overlays to go over TSINIT
37 023452 020021      6$:    CMP     R0, (R1)+    ;Must this overlay go over TSINIT?
38 023454 001404              BEQ     4$              ;Br if yes
39 023456 020127 000754'      CMP     R1, #LOWEND    ;End of low-overlay table?
40 023462 103773              BLO     6$              ;Br if not
41 023464 000407              BR      2$              ;This overlay is not forced over TSINIT
42 023466 005262 000002      4$:    INC     OS$FLG(R2) ;Set flag saying load over TSINIT
43 023472 016200 000000      MOV     OS$SIZ(R2), R0 ;Get # 64-byte blocks needed for overlay
44 023476 072027 000006      ASH     #6, R0          ;Get # bytes needed for overlay
45 023502 060004      ADD     R0, R4          ;Advance address within TSINIT
46 023504 062702 000006      2$:    ADD     #OS$$SZ, R2  ;Point to entry for next segment
47 023510 020237 000744'      CMP     R2, OSLAST     ;Have we finished?
48 023514 103745              BLO     1$              ;Loop if not
49         ;
50         ; Determine how much memory space is available in TSINIT for other overlays
51         ;
52 023516 020427 030012'      CMP     R4, #INITOP-50 ;Any space left for other overlays?
53 023522 103021              BHIS    9$              ;Br if not
54 023524 012705 030074'      MOV     #INITOP, R5    ;Point to top of overlay area
55 023530 160405              SUB     R4, R5          ;Total space available for overlays
56 023532 072527 177772      ASH     #-6, R5         ;Convert to # 64-byte blocks
57         ;

```

OVLPOS -- Determine which overlays go over TSINIT

```

58          ; Now begin loop which determines which other overlays go over TSINIT.
59          ; We do this in the order of largest to smallest to try to fill
60          ; the overlay area as completely as possible.
61          ;
62 023536 004737 025012' 3$: CALL OVLTRY          ; Try to find largest overlay that will fit
63 023542 103411          BCS 9$              ; Br if no more overlays will fit
64 023544 005262 000002  INC OS$FLG(R2)       ; Remember to load over TSINIT
65 023550 016200 000000  MOV OS$SIZ(R2),R0   ; Get # 64-byte blocks needed for overlay
66 023554 160005          SUB R0,R5          ; Reduce remaining free space in TSINIT
67 023556 072027 000006  ASH #6,R0          ; Get # bytes needed for overlay
68 023562 060004          ADD R0,R4          ; Advance overlay address in TSINIT
69 023564 000764          BR 3$              ; See if we can find more segments to load
70          ;
71          ; Finished
72          ;
73 023566 010405 9$: MOV R4,R5              ; Return top-of-overlay address in R5
74 023570 012604          MOV (SP)+,R4
75 023572 012603          MOV (SP)+,R3
76 023574 012602          MOV (SP)+,R2
77 023576 012601          MOV (SP)+,R1
78 023600 000207          RETURN

```

```

1          .SBTTL  OVLBLD -- Build overlay information table
2          ;-----
3          ; OVLBLD is called to build an overlay information table that is used
4          ; by TSINIT while loading TSX overlays into memory.
5          ;
6          ; Outputs:
7          ; Overlay segment information is set up in OSTABL.
8          ; OSLAST = Pointer past last entry in OSTABL.
9          ;
10         023602 010146 OVLBLD: MOV      R1,-(SP)
11         023604 010246          MOV      R2,-(SP)
12         023606 010346          MOV      R3,-(SP)
13         ;
14         ; Read 1st block of SAV file to get pointer to overlay table
15         ;
16         023610 013702 000152'      MOV      WRKBUF,R2          ;Point to work buffer
17         023614          .READW  #AREA,#17,R2,#256.,#0 ;read first block of the save file
18         023650 103444          BCS     22$              ;Br if error on read
19         023652 016201 000064      MOV      64(R2),R1         ;point to the overlay table
20         023656 001012          BNE     15$              ;br if overlays exist
21         ;
22         ; Must be verion 3B overlays structure at absolute location.
23         ;
24         023660 012737 000137 001000      MOV      #137,@#1000      ;position jump instruction over 3b ovly handler
25         023666 012737 000000G 001002      MOV      #OVRH,@#1002    ;position overlay intercept location
26         023674 012701 001104          MOV      #1104,R1        ;point to the overlay table
27         023700 010137 000000G          MOV      R1,OVRADD       ;save the address of the overlay table
28         ;
29         ; Initialize the table that holds information about the overlays
30         ;
31         023704 012703 000516'      15$:   MOV      #OSTABL,R3      ;Point to table for overlay info
32         023710 010163 000004          11$:   MOV      R1,OS$OVL(R3)    ;Save pointer to overlay control block
33         023714 005063 000002          CLR      OS$FLG(R3)      ;Assume seg will be loaded in high memory
34         023720 004737 024122'      CALL    ALCOVL          ;Determine if we should load this overlay
35         023724 010263 000000          MOV      R2,OS$SIZ(R3)   ;Remember total size of overlay+data
36         023730 062703 000006          ADD     #OS$SZ,R3       ;Point to next overlay table entry
37         023734 062701 000006          12$:   ADD     #6,R1        ;find the next region
38         023740 021127 004537          CMP     (R1),#4537      ;compare with a <JSR R5,$OVRH> instruction
39         023744 001361          BNE     11$            ;Br if not at end
40         023746 010337 000744'      MOV     R3,OSLAST      ;Save pointer past last overlay table entry
41         ;
42         ; Finished
43         ;
44         023752 012603          MOV     (SP)+,R3
45         023754 012602          MOV     (SP)+,R2
46         023756 012601          MOV     (SP)+,R1
47         023760 000207          RETURN
48         ;
49         ; Error -- Read error occured while reading overlay table
50         ;
51         023762          22$:   .PRINT  #TSXHD
52         023770          .PRINT  #RDERR
53         023776 000137 004102'      JMP     INISTP

```

```

1          .SBTTL  GETMAP -- Load any mapped system code regions
2          ;-----
3          ; GETMAP is called to load those system overlays that are placed
4          ; in high memory.
5          ;
6          ; Inputs:
7          ;   R5 = 64-byte block number of top of free memory.
8          ;
9          ; Outputs:
10         ;   R5 = New 64-byte block number of top of free memory.
11         ;
12 024002 010146 GETMAP: MOV     R1, -(SP)
13 024004 010246      MOV     R2, -(SP)
14 024006 010346      MOV     R3, -(SP)
15 024010 010537 000000G      MOV     R5, SMRSIZ      ; Save memory pointer at start of allocation
16         ;
17         ; Now that most of the system initialization is completed, we must check
18         ; again to see which overlays need to be loaded.
19         ;
20 024014 012703 000516'      MOV     #OSTABL, R3      ; Point to 1st overlay table entry
21 024020 004737 024266' 1#:  CALL    OPTOVL      ; See if this segment should be loaded
22 024024 010263 000000      MOV     R2, OS$SIZ(R3)  ; Save # 64-byte blocks needed for overlay
23 024030 062703 000006      ADD     #OS$$SZ, R3     ; Point to next overlay table entry
24 024034 020337 000744'      CMP     R3, OSLAST     ; Checked all entries in overlay table?
25 024040 103767      BLD     1#           ; Br if not
26         ;
27         ; Load those overlays that go into high memory
28         ;
29 024042 012702 000516'      MOV     #OSTABL, R2     ; Point to 1st overlay entry
30 024046 005762 000000 3#:  TST     OS$SIZ(R2)    ; Is this overlay segment wanted?
31 024052 001405      BEQ     4#           ; Br if not
32 024054 005762 000002      TST     OS$FLG(R2)    ; Load over TSINIT or into high memory?
33 024060 001002      BNE     4#           ; Br if load over TSINIT
34 024062 004737 025110'      CALL    GETOVL      ; Load overlay into high memory
35 024066 062702 000006 4#:  ADD     #OS$$SZ, R2     ; Point to next overlay table entry
36 024072 020237 000744'      CMP     R2, OSLAST     ; Have we done all overlays?
37 024076 103763      BLD     3#           ; Loop if not
38         ;
39         ; Finished
40         ;
41 024100 013700 000000G 19#:  MOV     SMRSIZ, R0     ; Get memory pointer at start of allocation
42 024104 160500      SUB     R5, R0         ; Calc amt of space allocated
43 024106 010037 000000G      MOV     R0, SMRSIZ     ; Save total space used for mapped regions
44 024112 012603      MOV     (SP)+, R3
45 024114 012602      MOV     (SP)+, R2
46 024116 012601      MOV     (SP)+, R1
47 024120 000207      RETURN

```

ALCOVL -- Allocate space for a system overlay region

```

1          .SBTTL  ALCOVL -- Allocate space for a system overlay region
2          ;-----
3          ; ALCOVL is called to determine if a system overlay region is wanted
4          ; (based on sysgen options), and if it is wanted to determine how
5          ; much space is needed for the code and data.
6          ;
7          ; Inputs:
8          ; R3 = Pointer to overlay table entry (OS$xxx)
9          ;
10         ; Outputs:
11         ; C-flag cleared ==> This segment is to be loaded.
12         ; C-flag set      ==> Do not load this overlay segment.
13         ; R2 = # 64-Byte blocks needed for segment including data areas within it.
14         ;
15 024122 010146 ALCOVL: MOV      R1, -(SP)
16         ;
17         ; Get pointer to linker-build overlay entry for segment
18         ;
19 024124 016301 000004      MOV      OS$OVL(R3), R1 ;Get pointer to linker-built entry for seg
20         ;
21         ; Read in the first block of the overlay segment
22         ;
23 024130 013702 000152'      MOV      WRKBUF, R2 ;Point to work buffer
24 024134          .READW  #AREA, #17, R2, #256., 0, BLK(R1) ;read the first block
25 024172 103415          BCS      3$ ;Br if read error
26         ;
27         ; Save the 3 character Rad50 segment ID in the 0.ADR cell of the
28         ; linker-built overlay table entry for this segment.
29         ;
30 024174 016261 000002 0000000      MOV      2(R2), 0.ADR(R1) ;save the rad50 overlay identifier
31         ;
32         ; Make sure the segment is not larger than 8Kb
33         ;
34 024202 016102 0000000      MOV      0.SIZ(R1), R2 ;get the word count of the code region
35 024206 006302          ASL      R2 ;convert to byte count
36 024210 020227 020000          CMP      R2, #20000 ;check for 8kb overflow
37 024214 101014          BHI      21$ ;Br if region is too big
38         ;
39         ; Don't load some optional segments if features were not selected
40         ; in TSGEN.
41         ;
42 024216 004737 024266'      CALL     OPTOVL ;See if we want to load this segment
43         ;
44         ; Finished
45         ; The C-flag is set or reset by OPTOVL.
46         ;
47 024222 012601          MOV      (SP)+, R1
48 024224 000207          RETURN
49         ;
50         ; Error -- Error on reading from SAV file
51         ;
52 024226          3$: .PRINT  #TSXHD ;Print heading
53 024234          .PRINT  #RDERR ;Read error
54 024242 000137 004102'      JMP      INISTP ;Abort initialization
55         ;
56         ; Error -- Insufficient memory space to load run-time systems
57         ;

```

ALCOVL -- Allocate space for a system overlay region

58 024246	21#:	.PRINT	#TSXHD	;PRINT HEADING
59 024254		.PRINT	#TSXSIZ	;PRINT ERROR MESSAGE
60 024262 000137 004102'		JMP	INISTP	;ABORT INITIALIZATION

```

1          .SBTTL  OPTOVL -- Check for optional system overlay regions
2          ;-----
3          ; OPTOVL is called to determine if a specific system overlay is or is
4          ; not to be loaded based on sysgen options.
5          ; This routine may also add space for buffers to the overlay regions size.
6          ;
7          ; Inputs:
8          ;   R3 = Pointer to overlay table entry for segment (OS#xxx)
9          ;
10         ; Outputs:
11         ;   C-flag cleared ==> Load this overlay.
12         ;   C-flag set      ==> Do not load this overlay.
13         ;   R2 = # 64-byte blocks needed for code + data for the segment.
14         ;
15 024266 010346 OPTOVL: MOV     R3, -(SP)
16 024270 010446      MOV     R4, -(SP)
17 024272 010546      MOV     R5, -(SP)
18         ;
19         ; Get the name of the overlay segment
20         ;
21 024274 016305 000004      MOV     OS#OVL(R3),R5 ;Get pointer to linker-built entry
22 024300 016504 000000G    MOV     D. ADR(R5),R4 ;Get name of the segment
23         ;
24         ; Get size of code portion of overlay segment
25         ;
26 024304 016502 000000G    MOV     D. SIZ(R5),R2 ;Get # words needed by code portion of seg
27 024310 006302          ASL     R2 ;Convert to # bytes
28         ;
29         ; See if this is an optional segment that we need to deal with specially
30         ;
31 024312 012700 024340'    MOV     #OVLST,R0 ;Point to overlay name list
32 024316 020420 1$:      CMP     R4,(R0)+ ;Found name of overlay?
33 024320 001406          BEQ     2$ ;Br if yes
34 024322 005720          TST     (R0)+ ;No -- Skip over address word
35 024324 020027 024430'    CMP     R0,#OVLEND ;Checked all names in the list?
36 024330 103772          BLO     1$ ;Loop if not
37 024332 000137 024750'    JMP     OXYYES ;Load this overlay
38         ;
39         ; Branch off to processing routine
40         ;
41 024336 000130 2$:      JMP     @(R0)+ ;Enter processing routine for the overlay
42         ;
43         ; Table of overlay names and processing routines
44         ;
45         .MACRO  OVLTBL  NAME
46         .RAD50  /'NAME'/
47         .WORD   OOR'NAME'
48         .ENDM   OVLTBL
49         ;
50 024340 OVLST:
51 024340      OVLTBL  USR ;TSUSR -- File management
52 024344      OVLTBL  SPL ;TSSPOL -- Spooling system
53 024350      OVLTBL  SP2 ;TSSPL2 -- Spooler flag pages
54 024354      OVLTBL  LOK ;TSLOCK -- Shared file record locking
55 024360      OVLTBL  MSG ;TSMMSG -- Inter-job message communication
56 024364      OVLTBL  SWP ;TSSWAP -- Job swapper
57 024370      OVLTBL  PLS ;TSPLAS -- PLAS support
    
```

OPTOVL -- Check for optional system overlay regions

58	024374			OVLTLBL	SLE		;TSSLE	-- Single line editor
59	024400			OVLTLBL	WIN		;TSWIN	-- Display window management
60	024404			OVLTLBL	MIO		;TSMIO	-- Mapped I/O
61	024410			OVLTLBL	CLO		;TSCLO	-- CL handler
62	024414			OVLTLBL	DBG		;TSDBG	-- Program debugger
63	024420			OVLTLBL	CSH		;TSCASH	-- Data caching
64	024424			OVLTLBL	DMP		;TSDUMP	-- Crash dump generator
65	024430					OVLEND:		
66								
67								
68								
69	024430	013703	000000G	ODRUSR:	MOV	VNFCSH,R3		;Get # file cache entries
70	024434	070327	000000G		MUL	#FC##SZ,R3		;Multiply by size of each entry
71	024440	060302			ADD	R3,R2		;Allocate space for directory cache
72	024442	000542			BR	OOXYES		;Load the segment
73								
74								
75								
76	024444	005727	000000G	ODRSPL:	TST	#SPLND		;Are there any spooled devices?
77	024450	001534			BEQ	OOXNO		;Br if not
78	024452	062702	000000C		ADD	#<SPLNB*512.>,R2		;Reserve room for spool buffers
79	024456	013703	000000G		MOV	NSPLBL,R3		;Get # blocks for spool file
80	024462	062703	0000007		ADD	#7,R3		;Bound up to byte boundary
81	024466	072327	177775		ASH	#-3,R3		;Divide by 8 to get # bytes for table
82	024472	005203			INC	R3		;Round up to word boundary
83	024474	042703	0000001		BIC	#1,R3		
84	024500	060302			ADD	R3,R2		;Add space for spool file allocation table
85	024502	000522			BR	OOXYES		;Load the segment
86								
87								
88								
89	024504	005727	000000G	ODRSP2:	TST	#SPLND		;Are there any spooled devices?
90	024510	001514			BEQ	OOXNO		;If not, don't load overlay
91	024512	000516			BR	OOXYES		;Load the segment
92								
93								
94								
95	024514	005737	000000G	ODRLOK:	TST	VMXSF		;Any shared files?
96	024520	001510			BEQ	OOXNO		;Br if not
97	024522	005737	000000G		TST	VNUMDC		;Shared file data caching wanted?
98	024526	001110			BNE	OOXYES		;Br if yes
99	024530	162702	000000G		SUB	#DCCSIZ,R2		;Reduce size of segment - Leave out cache code
100	024534	000505			BR	OOXYES		;Load the segment
101								
102								
103								
104	024536	013703	000000G	ODRMSG:	MOV	VMAXMC,R3		;Is message communication facility wanted?
105	024542	001477			BEQ	OOXNO		;Br if not
106	024544	070327	000000G		MUL	#MB##SZ,R3		;Space for message channel blocks
107	024550	060302			ADD	R3,R2		
108	024552	013703	000000G		MOV	VMXMRB,R3		;Number of message request blocks
109	024556	070327	000000G		MUL	#MR##SZ,R3		;Times size of request block
110	024562	060302			ADD	R3,R2		
111	024564	013703	000000G		MOV	VMSCHR,R3		;Max # chars in a message
112	024570	005203			INC	R3		;Bound up to word
113	024572	042703	0000001		BIC	#1,R3		;Reserve whole number of words
114	024576	062703	000000G		ADD	#MU\$TXT,R3		;Plus space for message header

```

115 024602 070337 0000000      MUL      VMXMSG,R3      ;Times maximum number of messages
116 024606 060302              ADD      R3,R2        ;Space for message buffers
117 024610 000457              BR       OOXYES
118
119      ; PLAS support
120
121 024612 013703 0000000      OORPLS: MOV      VPLAS,R3      ;PLAS support wanted?
122 024616 001451              BEQ      OOXNO        ;Br if not
123 024620 062703 0000021      ADD      #17.,R3      ;Bound up # blocks
124 024624 072327 177775      ASH      #-3,R3       ;Get # bytes needed for swap file bit map
125 024630 060302              ADD      R3,R2        ;Reserve room for swap file bit map
126 024632 000446              BR       OOXYES        ;Load the segment
127
128      ; Job swapper
129
130 024634 105737 0000000      OORSWP: TSTB     VSWPFL      ;Is this a swapping system?
131 024640 001440              BEQ      OOXNO        ;Br if not
132 024642 000442              BR       OOXYES        ;Br if yes -- Load the segment
133
134      ; Single line editor
135
136 024644 105737 0000000      OORSLE: TSTB     VSLEDT      ;Is SL editor wanted?
137 024650 001434              BEQ      OOXNO        ;Br if not
138 024652 000436              BR       OOXYES        ;Load the segment
139
140      ; Display windows
141
142 024654 013703 0000000      OORWIN: MOV      VMXWIN,R3     ;Are any display windows wanted?
143 024660 001430              BEQ      OOXNO        ;Br if not
144 024662 070327 0000000      MUL      #DW##SZ,R3      ;Amt of space needed for window control blks
145 024666 060302              ADD      R3,R2        ;Add to size of overlay
146 024670 000427              BR       OOXYES        ;Load the segment
147
148      ; Mapped I/O
149
150 024672 105737 0000000      OORMIO: TSTB     MIOFLG      ;Is I/O mapping needed?
151 024676 001421              BEQ      OOXNO        ;Br if not
152 024700 000423              BR       OOXYES        ;Load the segment
153
154      ; CL handler
155
156 024702 005727 0000000      OORCLO: TST      #CLTOTL     ;Any I/O lines?
157 024706 001415              BEQ      OOXNO        ;Br if not
158 024710 000417              BR       OOXYES        ;Yes, load the segment
159
160      ; Program debugger
161
162 024712 105737 0000000      OORDBG: TSTB     VDBFLG      ;Is the program debugger wanted?
163 024716 001411              BEQ      OOXNO        ;Br if not
164 024720 000413              BR       OOXYES        ;Load this segment
165
166      ; Data caching
167
168 024722 005737 0000000      OORCSH: TST      CSHALC      ;Is data caching wanted?
169 024726 001405              BEQ      OOXNO        ;Br if not
170 024730 000407              BR       OOXYES        ;Load this segment
171

```

```

172          ; Crash dump generator
173          ;
174 024732 105737 0000000  OORDMP: TSTB   VSYDMP           ; Is dump facility wanted?
175 024736 001401          BEQ     OOXNO           ; Br if not
176 024740 000403          BR      OOXYES          ; Br if yes
177          ;
178          ; Don't load this segment
179          ;
180 024742 005002  OOXNO:  CLR     R2           ; Say no space needed for overlay
181 024744 000261          SEC           ; Signal don't load the segment
182 024746 000415          BR      OOXFIN
183          ;
184          ; Load this segment
185          ;
186 024750 005202  OOXYES: INC     R2           ; Make sure size is even
187 024752 042702 000001          BIC     #1,R2
188 024756 020227 020000          CMP     R2,#8192.         ; Don't allow code + data to exceed 8Kb
189 024762 101402          BLOS   1#                ; Br if ok
190 024764 012702 020000          MOV     #8192.,R2        ; Note, init code in segment will truncate dat
191 024770 062702 000077 1#:   ADD     #63.,R2      ; Convert to # 64-byte blocks
192 024774 072227 177772          ASH    #-6,R2
193 025000 000241          CLC           ; Signal to load the segment
194          ;
195          ; Finished
196          ;
197 025002 012605  OOXFIN: MOV     (SP)+,R5
198 025004 012604          MOV     (SP)+,R4
199 025006 012603          MOV     (SP)+,R3
200 025010 000207          RETURN
    
```

OVLTRY -- Find an overlay to place over TSINIT

```

1          .SBTTL  OVLTRY -- Find an overlay to place over TSINIT
2          ;-----
3          ; OVLTRY is called to identify the largest overlay segment which
4          ; will fit in the TSINIT area and which is not already marked to go
5          ; over TSINIT.
6          ;
7          ; Inputs:
8          ;   R5 = # 64-byte blocks available for segment in TSINIT.
9          ;
10         ; Outputs:
11         ;   R2 = Pointer to OSTABL entry for segment
12         ;   C-flag set ==> No more segments will fit.
13         ;
14 025012  010346  OVLTRY: MOV      R3, -(SP)
15         ;
16         ;   Begin loop to examine all segments
17         ;
18 025014  005002          CLR      R2          ; Say we haven't found any segment yet
19 025016  012703  000516'  MOV      #OSTABL, R3      ; Point to entry for 1st segment
20 025022  005763  000000  1$:  TST      OS$SIZ(R3)      ; Is this segment to be loaded?
21 025026  001415          BEQ      2$          ; Br if not
22 025030  005763  000002  TST      OS$FLG(R3)      ; Is this segment already over TSINIT?
23 025034  001012          BNE      2$          ; Br if yes
24 025036  026305  000000  CMP      OS$SIZ(R3), R5   ; Will this segment fit?
25 025042  101007          BHI      2$          ; Br if not
26 025044  005702          TST      R2          ; Have we found any other seg yet?
27 025046  001404          BEQ      3$          ; Br if not
28 025050  026362  000000  000000  CMP      OS$SIZ(R3), OS$SIZ(R2) ; Is new seg larger than old?
29 025056  101401          BLOS   2$          ; Br if not
30 025060  010302  3$:  MOV      R3, R2          ; Remember largest segment
31 025062  062703  000006  2$:  ADD      #OS$$SZ, R3      ; Point to entry for next segment
32 025066  020337  000744'  CMP      R3, OSLAST     ; Have we checked all segments?
33 025072  103753          BLO      1$          ; Loop if not
34         ;
35         ;   Finished
36         ;
37 025074  000241          CLC          ; Assume we found a segment
38 025076  005702          TST      R2          ; Did we find a segment that will fit?
39 025100  001001          BNE      9$          ; Br if yes
40 025102  000261          SEC          ; Signal failure on return
41 025104  012603  9$:  MOV      (SP)+, R3
42 025106  000207          RETURN

```

GETOVL -- Load system overlay into high memory

```

1          .SBTTL  GETOVL -- Load system overlay into high memory
2          ;-----
3          ; GETOVL is called to load a system overlay into high memory.
4          ;
5          ; Inputs:
6          ;   R2 = Pointer to overlay table entry for segment in OSTABL.
7          ;   R5 = 64-byte physical memory block number where seg is to be loaded.
8          ;
9          ; Outputs:
10         ;   R5 = Update 64-byte physical memory block pointer for next segment.
11         ;
12 025110  GETOVL:
13         ;
14         ; Allocate space for the overlay segment
15         ;
16 025110 166205 000000      SUB    OS$SIZ(R2),R5    ;Allocate space for overlay
17 025114 020527 001600      CMP    R5,#1600        ;Are we about to run over RT-11?
18 025120 103405            BLO    10$                ;Br if yes -- Insufficient memory
19         ;
20         ; Remember the base address of some key segments
21         ;
22 025122 004737 004470'    CALL   KEYSEG          ;Remember address of some segments
23         ;
24         ; Load the segment
25         ;
26 025126 004737 025154'    CALL   LODOVL         ;Load the segment
27         ;
28         ; Finished
29         ;
30 025132 000207            RETURN
31         ;
32         ; Error: Memory overflow
33         ;
34 025134            .PRINT #TSXHD
35 025142            .PRINT #TSXSIZ
36 025150 000137 004102'    JMP    INISTP

```

LODOVL -- Read and relocate system overlay

```

1          .SBTTL  LODOVL -- Read and relocate system overlay
2          ;-----
3          ; LODOVL is called to load a system overlay region into memory.
4          ;
5          ; Inputs:
6          ; R2 = Pointer to OSTABL entry for segment being loaded.
7          ; R5 = 64-byte physical memory block number where segment is to be loaded.
8          ;
9 025154 010146 LODOVL: MOV      R1, -(SP)
10 025156 010246      MOV      R2, -(SP)
11 025160 010346      MOV      R3, -(SP)
12 025162 010446      MOV      R4, -(SP)
13 025164 010546      MOV      R5, -(SP)
14
15          ;
16          ; Get info about size of the overlay and position within SAV file
17          ;
18 025166 016201 000004      MOV      OS$OVL(R2), R1 ;Get pointer to linker-built segment entry
19 025172 016103 000000G      MOV      0, SIZ(R1), R3 ;Get size of overlay segment (# words)
20 025176 016137 000000G 000142'  MOV      0, BLK(R1), FILBLK;Get block in SAV file where segment starts
21 025204 010302      MOV      R3, R2 ;Get total number of words in segment
22 025206 062702 000377      ADD      #255, R2 ;round to the nearest number of blocks
23 025212 000302      SWAB     R2 ;Divide by 256. words per segment
24 025214 042702 177400      BIC     #177400, R2 ;kill sign extension bits
25 025220 010561 000000G      MOV      R5, 0, PAR(R1) ;Remember where segment is being loaded
26          ;
27          ; Read next block of overlay segment into low-memory buffer
28          ;
29 025224 013704 000152' 10$: MOV      WRKBUF, R4 ;Point to work buffer
30 025230      .READW  #AREA, #17, R4, #256, FILBLK ;read a block
31 025266 103466      BCS     22$ ;read error occurred
32          ;
33          ; Move from low buffer to high position in memory
34          ;
35 025270 012701 000000G      MOV      #VPAR5, R1 ;get the virtual address of the mapped region
36 025274 012700 000400      MOV      #256, R0 ;obtain the number of words to move
37 025300 020300      CMP     R3, R0 ;Do we need to move as many as 256 words?
38 025302 103001      BHIS   2$ ;Br if yes
39 025304 010300      MOV      R3, R0 ;Get number of words to move for last block
40 025306 160003 2$: SUB     R0, R3 ;Get number of words left after this move
41 025310      DISABL ;** Disable interrupts **
42 025316 013746 000000G      MOV      @#KPAR5, -(SP) ;save the contents of the mapping register
43 025322 010537 000000G      MOV      R5, @#KPAR5 ;change the mapping register
44 025326 052737 000000G 000000G  BIS     #MMENBL, @#SR0MMR;enable memory management
45 025334 105737 000000G      TSTB   MEM256 ;Does machine have at least 256Kb of memory?
46 025340 001403      BEQ     11$ ;Br if not
47 025342 052737 000000G 000000G  BIS     #EMMAP, @#SR3MMR ;enable extended memory addressing
48 025350 012421 11$: MOV     (R4)+, (R1)+ ;move into high memory
49 025352 077002      SOB     R0, 11$
50 025354 105737 000000G      TSTB   MEM256 ;Does this machine have at least 256Kb?
51 025360 001403      BEQ     12$ ;Br if not
52 025362 042737 000000G 000000G  BIC     #EMMAP, @#SR3MMR ;disable extended memory management
53 025370 042737 000000G 000000G 12$: BIC     #MMENBL, @#SR0MMR;disable memory management
54 025376 012637 000000G      MOV     (SP)+, @#KPAR5 ;restore the mapping register
55 025402      ENABL ;** Enable interrupts **
56 025410 062705 000010      ADD     #10, R5 ;advance 64-byte block # by 512-bytes
57 025414 005237 000142'      INC     FILBLK ;increment file block #

```

```
58 025420 005302          DEC      R2          ;More to be copied?
59 025422 001402          BEQ      5$          ;Br if not
60 025424 000137 025224'  JMP      10$          ;Read and copy rest of mapped segment
61                               ;
62                               ; Finished loading the segment
63                               ;
64 025430 012605 5$:      MOV      (SP)+,R5
65 025432 012604          MOV      (SP)+,R4
66 025434 012603          MOV      (SP)+,R3
67 025436 012602          MOV      (SP)+,R2
68 025440 012601          MOV      (SP)+,R1
69 025442 000207          RETURN
70                               ;
71                               ; Error occurred on read
72                               ;
73 025444 22$:      .PRINT #TSXHD      ;Print heading
74 025452          .PRINT #RDERR      ;Read error
75 025460 000137 004102'  JMP      INISTP      ;Abort initialization
```

GETSRT -- Load any shared run-time systems

```

1          .SBTTL  GETSRT -- Load any shared run-time systems
2          ;-----
3          ; GETSRT is called to load into memory a shared run-time system.
4          ; Shared run-time systems are loaded into the top of memory.
5          ;
6          ; Inputs:
7          ; R1 = Pointer to shared run-time descriptor block.
8          ; R5 = 64-byte block number of top of free memory.
9          ;
10         ; Outputs:
11         ; R5 = New top of memory block number
12         ;
13 025464 010146 GETSRT: MOV     R1,-(SP)
14 025466 010246     MOV     R2,-(SP)
15 025470 010346     MOV     R3,-(SP)
16 025472 010446     MOV     R4,-(SP)
17         ;
18         ; See if this is a dummy run-time entry to allow for patching
19         ;
20 025474 021127 000000G     CMP     (R1),#DMYDEV    ;Dummy run-time entry?
21 025500 001540     BEQ     7$          ;Br if yes
22         ;
23         ; Try to open a channel to run-time file
24         ;
25 025502     .LOOKUP #AREA,#1,R1    ;OPEN CHANNEL TO RUN-TIME FILE
26 025520 103010     BCC     8$          ;BR IF OPEN WAS SUCCESSFUL
27         ;
28         ; Cannot open shared run-time file.
29         ; See if he wants to abort or continue.
30         ;
31 025522 105737 000000G     TSTB   VINABT          ;ABORT OR CONTINUE
32 025526 001132     BNE     9$          ;BR IF ABORT WANTED
33 025530 005061 000000G     CLR     RT$NAM(R1)      ;Mark run-time as not-available
34 025534 005061 000002G     CLR     RT$NAM+2(R1)
35 025540 000520     BR      7$          ;GO LOAD NEXT RUN-TIME SYSTEM
36         ;
37         ; Set up information about position of run-time in physical memory
38         ;
39 025542 116102 000000G  8$:   MOVB   RT$SKP(R1),R2    ;GET # BLOCKS TO SKIP AT FRONT OF RUN-TIME
40 025546 042702 177400     BIC     #^C377,R2      ;CLEAR SIGN EXTENSION
41 025552 160200     SUB     R2,R0          ;GET # BLOCKS TO READ (LOOKUP SET ROW SIZE)
42 025554 010561 000000G     MOV     R5,RT$TOP(R1)  ;SET 64-BYTE BLOCK # ABOVE TOP OF RUN-TIME
43 025560 010003     MOV     R0,R3          ;GET # 512-BYTE BLOCKS IN RUN-TIME
44 025562 072027 000003     ASH    #3,R0          ;CONVERT TO # 64-BYTE BLOCKS
45 025566 160005     SUB     R0,R5          ;CALCULATE BASE 64-BYTE BLOCK # OF RUN-TIME
46 025570 020527 001600     CMP     R5,#1600      ;ARE WE ABOUT TO RUN OVER RT-11?
47 025574 103530     BLO    11$          ;BR IF YES
48 025576 010561 000000G     MOV     R5,RT$BAS(R1)  ;SET BASE 64-BYTE BLOCK # OF RUN-TIME
49         ;
50         ; Read run-time system into memory and position in high-memory
51         ;
52 025602 010546     MOV     R5,-(SP)        ;Save address of bottom of run-time
53 025604 013704 000152'  4$:   MOV     WRKBUF,R4      ;Point to work buffer
54 025610     .READW #AREA,#1,R4,#256.,R2 ;READ A BLOCK OF RUN-TIME FILE
55         ; Use memory management to access high-memory area.
56 025644 012701 000000G     MOV     #VPAR6,R1     ;GET VIRTUAL ADDRESS OF PAR6 ADDRESS REGION
57 025650 010537 000000G     MOV     R5,@#UPAR6    ;SET USER-MODE PAR6 MAP OFFSET VALUE

```

GETSRT -- Load any shared run-time systems

```

58 025654 012737 077406 000000G      MOV      #077406,@#UPDR6 ;SET PDR TO ALLOW FULL ACCESS TO PAGE
59 025662 052737 000000G 000000G      BIS      #UPMODE,@#PSW  ;SET PREVIOUS-MODE = USER FOR MTPD ACCESS
60 025670 012700 000400                MOV      #256.,R0      ;GET # WORDS TO MOVE
61 025674                DISABL                    ;** Disable interrupts **
62 025702 052737 000000G 000000G      BIS      #MMENBL,@#SR0MMR;enable memory management
63 025710 105737 000000G                TSTB    MEM256         ;DOES THIS MACHINE HAVE AT LEAST 256KB?
64 025714 001403                BEQ     3$             ;BR IF NOT
65 025716 052737 000000G 000000G      BIS      #EMMAP,@#SR3MMR;enable extended memory addressing
66 025724 012446                3$:    MOV      (R4)+,-(SP) ;TRANSFER DATA FROM BUFFER TO HIGH MEMORY
67 025726 106621                MTPD    (R1)+
68 025730 077003                SOB     R0,3$
69 025732 105737 000000G                TSTB    MEM256         ;DOES THIS MACHINE HAVE AT LEAST 256KB?
70 025736 001403                BEQ     31$           ;BR IF NOT
71 025740 042737 000000G 000000G      BIC      #EMMAP,@#SR3MMR;DISABLE EXTENDED MEMORY MANAGEMENT
72 025746 042737 000000G 000000G 31$:   BIC      #MMENBL,@#SR0MMR;DISABLE MEMORY MANAGEMENT
73 025754                ENABL                    ;** Enable interrupts **
74 025762 062705 000010                ADD     #10,R5         ;ADVANCE 64-BYTE BLOCK # BY 512-BYTES
75 025766 005202                INC     R2             ;INC FILE BLOCK #
76 025770 077373                SOB     R3,4$         ;READ AND COPY REST OF FILE
77                ;
78                ; Finished loading the run-time system.
79                ;
80 025772 012605                MOV      (SP)+,R5
81 025774                .CLOSE #1
82                ;
83                ; Finished
84                ;
85 026002 012604 7$:    MOV      (SP)+,R4
86 026004 012603                MOV      (SP)+,R3
87 026006 012602                MOV      (SP)+,R2
88 026010 012601                MOV      (SP)+,R1
89 026012 000207                RETURN
90                ;
91                ; Error -- Cannot find run-time system file
92                ;
93 026014 9$:    .PRINT #TSXHD ;PRINT MESSAGE HEADING
94 026022                .PRINT #COSRT ;PRINT ERROR MESSAGE
95 026030 012702 000004                MOV      #4,R2 ;PRINT 4 RAD50 VALUES
96 026034 012100 10$:   MOV      (R1)+,R0 ;GET PART OF NAME
97 026036 004737 030020'                CALL    PRTR50 ;PRINT RAD50 VALUE
98 026042 077204                SOB     R2,10$
99 026044                .PRINT #CRLF ;END LINE
100 026052 000137 004102'                JMP     INISTP ;ABORT INITIALIZATION
101                ;
102                ; Error -- Insufficient memory space to load run-time systems
103                ;
104 026056 11$:   .PRINT #TSXHD ;PRINT HEADING
105 026064                .PRINT #SRTOVF ;PRINT ERROR MESSAGE
106 026072 000137 004102'                JMP     INISTP ;ABORT INITIALIZATION

```

CSHBUF -- Allocate space for data cache tables

```

1          .SBTTL  CSHBUF -- Allocate space for data cache tables
2          ;-----
3          ; Allocate space for data cache blocks and control tables.
4          ;
5          ; Inputs:
6          ;   R4 = 64-byte block number of base of free memory.
7          ;   R5 = 64-byte block number of top of free memory.
8          ;
9          ; Outputs:
10         ;   R5 = Updated 64-byte block number of top of free memory.
11         ;
12 026076 010246 CSHBUF: MOV     R2, -(SP)
13 026100 010346      MOV     R3, -(SP)
14         ;
15         ; See if data caching is wanted
16         ;
17 026102 013737 0000000 0000000      MOV     CSHALC, VCSHNB ;Set # blocks in use = # blocks allocated
18 026110 013702 0000000      MOV     CSHALC, R2 ;Did used request data caching?
19 026114 001464      BEQ     9$ ;Br if not
20         ;
21         ; Calculate number of 64-byte blocks needed for each cache control table
22         ;
23 026116 062702 000037      ADD     #31, R2 ;Bound up to 32 word block
24 026122 072227 177773      ASH     #-5, R2 ;Convert to # 64-byte blocks
25         ;
26         ; Compute total space that will be used by all cache data
27         ;
28 026126 013703 0000000      MOV     CSHALC, R3 ;Get # blocks in cache
29 026132 072327 000003      ASH     #3, R3 ;Get # 64-byte blks used by cache data buffers
30 026136 012700 000010      MOV     #8, R0 ;Get number of cache control tables
31 026142 060203 1$: ADD     R2, R3 ;Accumulate total space needed
32 026144 077002      SOB     R0, 1$
33 026146 010337 0000000      MOV     R3, CSHSIZ ;Save total space used by cache data
34         ;
35         ; See if there is enough memory space available for the specified cache
36         ;
37 026152 010500      MOV     R5, R0 ;Get top of memory address
38 026154 160400      SUB     R4, R0 ;Compute # free 64-byte blocks
39 026156 020300      CMP     R3, R0 ;Is there enough total space?
40 026160 103045      BHIS   10$ ;Br if not
41         ;
42         ; Allocate space for cache data buffers
43         ;
44 026162 013700 0000000      MOV     CSHALC, R0 ;Get # blocks in data cache
45 026166 072027 000003      ASH     #3, R0 ;Get # 64-byte blocks needed for allocation
46 026172 160005      SUB     R0, R5 ;Allocate space for cache data buffers
47 026174 010537 0000000      MOV     R5, CSHBFP ;Save pointer to base of buffer area
48         ;
49         ; Allocate space for each control table
50         ;
51 026200 160205      SUB     R2, R5 ;Allocate space for table
52 026202 010537 0000000      MOV     R5, CA$BLK ;Block number associated with entry
53 026206 160205      SUB     R2, R5 ;Allocate space for table
54 026210 010537 0000000      MOV     R5, CA$DVU ;Device and unit number
55 026214 160205      SUB     R2, R5 ;Allocate space for table
56 026216 010537 0000000      MOV     R5, CA$WCT ;Number of words
57 026222 160205      SUB     R2, R5 ;Allocate space for table

```

CSHBUF -- Allocate space for data cache tables

```

58 026224 010537 000000G      MOV      R5,CA$UFL      ;LRU chain forward link
59 026230 160205              SUB      R2,R5          ;Allocate space for table
60 026232 010537 000000G      MOV      R5,CA$UBL      ;LRU chain backward link
61 026236 160205              SUB      R2,R5          ;Allocate space for table
62 026240 010537 000000G      MOV      R5,CA$HFL      ;Hash chain forward link
63 026244 160205              SUB      R2,R5          ;Allocate space for table
64 026246 010537 000000G      MOV      R5,CA$HBL      ;Hash chain backward link
65 026252 160205              SUB      R2,R5          ;Allocate space for table
66 026254 010537 000000G      MOV      R5,CA$HSH      ;Hash chain list heads
67 026260 020527 001600      CMP      R5,#1600      ;Did we run over RT-11?
68 026264 101403              BLOS    10$            ;Br if yes
69
70      ; Finished
71
72 026266 012603      9$:      MOV      (SP)+,R3
73 026270 012602      MOV      (SP)+,R2
74 026272 000207      RETURN
75
76      ; Insufficient memory space available for cache data
77
78 026274      10$:     .PRINT  #TSXHD      ;Print heading
79 026302      .PRINT  #CSHOVF     ;Overflow message
80 026310 000137 004102'     JMP      INISTP      ;Abort the initialization

```

CSHBUF -- Allocate space for data cache tables

```

1           .IF      EQ,PROCID      ;Don't allow ODT for production PRO version
2           .SBTTL  GETODT -- Load ODT
3           ;-----
4           ; GETODT is called to load ODT into memory above TSX and transfer control
5           ; to it. On return, ODT has been started.
6           ;
7           ; Inputs:
8           ; R5 = Address where ODT is to be loaded.
9           ;
10          ; Outputs:
11          ; R5 = Address above top of ODT.
12          ;
13 026314 010146 GETODT: MOV      R1,-(SP)
14 026316 010246      MOV      R2,-(SP)
15 026320 010346      MOV      R3,-(SP)
16 026322 010446      MOV      R4,-(SP)
17          ;
18          ; Try to lookup ODT rel file.
19          ;
20 026324          .LOOKUP #AREA,#1,#ODTBLK ;LOOKUP ODT REL FILE
21 026344 103010    BCC      1$          ;BR IF FOUND IT
22 026346          .PRINT #TSXHD          ;CAN'T FIND ODT
23 026354          .PRINT #NOODT
24 026362 000137 004102' JMP      INISTP          ;ABORT INITIALIZATION
25          ;
26          ; Read first block of ODT file and determine size of ODT.
27          ;
28 026366 062705 000310 1$:      ADD      #200.,R5          ;RESERVE SPACE FOR ODT STACK
29 026372 010500      MOV      R5,R0          ;CHECK MEMORY ADDRESS FOR OVERFLOW
30 026374 062700 001000      ADD      #512.,R0
31 026400 004737 027630'     CALL     CHKMEM
32 026404          .READW #AREA,#1,R5,#256.,#0
33 026440 103002      BCC      2$          ;Br if no read error
34 026442 000137 027024'     JMP      ODTRDX          ;Read error
35 026446 016502 000052     2$:      MOV      RSZ(R5),R2          ;GET SIZE OF ODT
36 026452 010203      MOV      R2,R3
37 026454 060503      ADD      R5,R3          ;GET ADDRESS ABOVE TOP OF ODT
38 026456 010300      MOV      R3,R0          ;CHECK MEMORY ADDRESS FOR OVERFLOW
39 026460 004737 027630'     CALL     CHKMEM
40 026464 000241      CLC
41 026466 006002      ROR      R2          ;GET # WORDS IN ODT
42          ; Get starting address of ODT
43 026470 016500 000040      MOV      STA(R5),R0          ;GET OFFSET TO START ADDRESS
44 026474 162700 001000      SUB      #ODTBAS,R0          ;CALCULATE ABSOLUTE STARTING ADDRESS
45 026500 060500      ADD      R5,R0
46 026502 010037 000234'     MOV      R0,ODTSTA          ;THIS IS REAL STARTING ADDRESS
47 026506 016501 000062     MOV      RBD(R5),R1          ;GET # OF BLOCK WITH RELOCATION INFO
48          ;
49          ; Read in ODT rel file image.
50          ;
51 026512          .READW #AREA,#1,R5,R2,#1
52 026546 103526      BCS      ODTRDX          ;BR IF READ ERROR
53          ;
54          ; Relocate addresses in ODT.
55          ; R5 = Address of base of ODT; R3 = Address above top of ODT.
56          ; R1 = Block number in rel file of start of relocation info.
57          ;

```

```

58 026550 010337 000036' RELFIL: MOV R3,ODTTOP ;SAVE ADDRESS ABOVE TOP OF ODT
59 026554 010337 000230' MOV R3,RLBF
60 .IF NE,PROCID ;Only if PRO protection code is included
61 TSTB PROFLG ;Are we running on a Pro?
62 BNE 1$ ;Br if yes
63 .ENDC ;NE,PROCID
64 026560 013737 000152' 000230' MOV WRKBUF,RLBF ;READ RELOCATION INFO HERE
65 026566 013737 000230' 000232' 1$: MOV RLBF,RLBFND
66 026574 062737 002000 000232' ADD #1024,RLBFND ;GET ADDRESS OF END OF BUFFER AREA
67 026602 010504 MOV R5,R4 ;GET BASE ADDRESS OF ODT
68 026604 162704 001000 SUB #ODTBAS,R4 ;SUBTRACT LINK BASE ADDRESS
69 ; Read in relocation address list.
70 026610 4$: .READW #AREA,#1,RLBF,#512,R1
71 026646 103003 BCC 7$ ;BR IF NO READ ERROR
72 026650 105737 000052 TSTB @#52 ;END OF FILE IS OK
73 026654 001063 BNE ODTRDX ;BR IF READ ERROR
74 ; Relocate some addresses in ODT.
75 026656 013702 000230' 7$: MOV RLBF,R2 ;POINT TO RELOCATION INFO
76 026662 012203 3$: MOV (R2)+,R3 ;GET ADDRESS OF LOCATION TO RELOCATE
77 026664 020327 177776 CMP R3,#-2 ;TIME TO STOP?
78 026670 001416 BEQ 9$ ;BR IF FINISHED
79 026672 012200 MOV (R2)+,R0 ;GET VALUE TO RELOCATE
80 026674 006303 ASL R3 ;CVT TO BYTE ADDRESS
81 026676 103002 BCC 5$ ;BR IF ADDITIVE RELOCATION
82 026700 160400 SUB R4,R0 ;RELOCATE THE ADDRESS
83 026702 000401 BR 6$
84 026704 060400 5$: ADD R4,R0 ;RELOCATE THE ADDRESS
85 026706 060503 6$: ADD R5,R3 ;GET LOCATION WHERE WORD GOES
86 026710 010013 MOV R0,@R3 ;STORE RELOCATED ADDRESS
87 026712 020237 000232' CMP R2,RLBFND ;TIME TO READ NEXT BUFFER FULL?
88 026716 103761 BLO 3$ ;BR IF NOT
89 026720 062701 000002 ADD #2,R1 ;ADVANCE BLOCK #
90 026724 000731 BR 4$ ;GO READ NEXT BUFFER FULL
91 ;
92 ; Finished relocation.
93 ; Close ODT rel file.
94 ;
95 026726 9$: .CLOSE #1
96 ;
97 ; Direct interrupts to 60 and 64 to an RTI instruction
98 ;
99 ; MOV #DORTI,@#60 ;Catch interrupt 60
100 ; MOV #DORTI,@#64 ;Catch interrupt 64
101 ;
102 ; Load registers with the following values for initial entry to ODT:
103 ; R0 = Base of TSINIT
104 ; R1 = Important breakpoint (^R) in TSX
105 ; R2 = Base of TSGEN
106 ; R3 = Base of TSEXC
107 ; R4 = Base of TSEMT
108 ; R5 = Return address to start execution
109 ; 0(SP) = Address of mapsys routine
110 ; 2(SP) = Address of sysmap cell
111 ;
112 026734 012700 000000' MOV #TSINIT,R0
113 026740 012701 000000G MOV #BRKPT,R1
114 026744 012702 000000G MOV #TSGEN,R2

```

GETODT -- Load ODT

```

115 026750 012703 0000000      MOV      #TSEXEC,R3
116 026754 012704 0000000      MOV      #TSEMT,R4
117 026760 012746 0000000      MOV      #SYSMAP,-(SP) ; PASS ADDRESS OF SYSMAP CELL TO ODT
118 026764 012746 0000000      MOV      #MAPSYS,-(SP) ; PASS ADDRESS OF MAPSYS ROUTINE
119 026770 012705 027000'      MOV      #10$,R5 ; ADDRESS FOR ODT TO RETURN TO
120
121 ; Enter ODT
122 ;
123 026774 000177 151234      JMP      @ODTSTA ; JUMP TO START OF ODT
124 ;
125 ; Return from ODT.
126 ; Continue initialization of TSX.
127 ;
128 027000 013737 000014 0000000 10$: MOV      @#14,ODTTRP ; SAVE ODT BREAKPOINT ENTRY ADDRESS
129 027006 013705 000036'      MOV      ODTTOP,R5 ; ADDRESS ABOVE TOP OF ODT
130 027012 012604      MOV      (SP)+,R4
131 027014 012603      MOV      (SP)+,R3
132 027016 012602      MOV      (SP)+,R2
133 027020 012601      MOV      (SP)+,R1
134 027022 000207      RETURN
135 ;
136 ; Error while reading ODI rel file.
137 ;
138 027024      ODTRDX: .PRINT #TSXHD ; PRINT ERROR MESSAGE
139 027032      .PRINT #ODTRDM
140 027040 000137 004102'      JMP      INISTP ; ABORT INITIALIZATION
141 ;
142 ; RTI instruction to disable interrupts
143 ;
144 027044 000002      DORTI: RTI
145 ; ENDC ; EQ,PROCID

```

OPNCHN -- Open a TSX-Plus channel

```

1          .SBTTL  OPNCHN -- Open a TSX-Plus channel
2          ;-----
3          ; OPNCHN is called to set up information in a TSX-Plus channel block
4          ; to make it look as if the channel has been opened to a specified
5          ; device with a .ENTER.
6          ;
7          ; Inputs:
8          ; R0 = Address of channel block to be opened.
9          ; R2 = Rad50 device name.
10         ;
11         ; Outputs:
12         ; C-flag set ==> Cannot open the device.
13         ;
14 027046 010146 OPNCHN: MOV     R1,-(SP)
15 027050 010346          MOV     R3,-(SP)
16 027052 010003          MOV     R0,R3          ;Carry channel block address in R3
17         ;
18         ; Initialize the channel block
19         ;
20 027054 010301          MOV     R3,R1          ;Point to the channel block
21 027056 012700 000000C  MOV     #<CHNSIZ/2>,R0 ;Get # words to zero
22 027062 005021 2$: CLR     (R1)+      ;Zero the channel block
23 027064 077002          SOB     R0,2$
24 027066 012763 000000C 000000G MOV     #<CS$OPN!CS$ENT>,C.CSW(R3) ;Initialize CSW to say chan open
25         ;
26         ; Convert the device name into device # and unit #
27         ;
28 027074 010200          MOV     R2,R0          ;Get the full device name
29 027076 004737 012704'  CALL    CVTDVU        ;Convert to dev # and unit #
30 027102 103411          BCS     9$           ;Br if we don't recognize the device name
31 027104 010001          MOV     R0,R1          ;Get index # and unit #
32 027106 000301          SWAB    R1           ;Get unit # to low byte
33 027110 110163 000000G  MOVB   R1,C.DEVQ(R3) ;Set unit # in channel block
34 027114 042700 000000C  BIC     #^C<CS$NMX>,R0 ;Clear all but device index number in R0
35 027120 050063 000000G  BIS     R0,C.CSW(R3) ;Store device index # into CSW
36         ;
37         ; Success
38         ;
39 027124 000241          CLC          ;Signal success on return
40         ;
41         ; Finished
42         ;
43 027126 012603 9$: MOV     (SP)+,R3
44 027130 012601          MOV     (SP)+,R1
45 027132 000207          RETURN

```

```

1          .SBTTL  SETCHN -- Copy RT-11 channel information into TSX system chan
2          ;-----
3          ; SETCHN is called to set up a TSX system channel block to access a file
4          ; that has been opened using RT-11.  The device index number is converted
5          ; from the RT-11 device number to the corresponding TSX device number.
6          ; Note: the channel must have been opened with a .lookup (not .enter)
7          ; to use this routine.
8          ;
9          ; Inputs:
10         ; Channel # 1 = Open to file of interest.
11         ; R0 = Address of TSX channel block which is to be set up.
12         ; R2 = Rad-50 device name.
13         ;
14         ; Outputs:
15         ; Channel block pointed to by R0 is set up for future TSX I/O.
16         ; Channel # 1 is closed.
17         ;
18 027134 010146 SETCHN: MOV      R1,-(SP)
19 027136 010246      MOV      R2,-(SP)
20 027140 010346      MOV      R3,-(SP)
21 027142 010001      MOV      R0,R1          ;GET ADDRESS OF TSX CHANNEL BLOCK
22         ;
23         ; Do .SAVESTATUS to store channel information into TSX channel block.
24         ;
25 027144         .SAVEST #AREA,#1,R1      ;STORE CHANNEL STATUS INTO TSX CHANNEL BLOCK
26         ;
27         ; Now convert RT-11 device table index number into corresponding TSX
28         ; device table index number.
29         ;
30 027162 011103      MOV      (R1),R3          ;GET CSW FOR CHANNEL
31 027164 042703 177701 BIC      #^C76,R3          ;GET RT-11 DEVICE INDEX NUMBER
32 027170         .GVAL   #AREA,#404      ;GET RT-11 OFFSET TO PNAME TABLE
33 027210 060003      ADD      R0,R3          ;GET ADDRESS OF NAME OF DEVICE IN PNAME TABLE
34 027212         .GVAL   #AREA,R3       ;GET NAME OF DEVICE FROM RT-11
35 027230 013703 000000G MOV      NUMDEV,R3        ;GET INDEX # FOR LAST TSX DEVICE
36 027234 020063 000000G 1$:  CMP      R0,PNAME(R3)      ;LOOK FOR DEVICE IN OUR TABLES
37 027240 001404      BEQ      2$          ;BR IF FOUND
38 027242 162703 000002 SUB      #2,R3          ;CHECK NEXT ENTRY
39 027246 002372      BGE      1$          ;BR IF MORE TO CHECK
40 027250 000407      BR       MTSXDV       ;VERY STRANGE THAT WE DIDN'T FIND IT
41 027252 042711 000076 2$:  BIC      #76,(R1)      ;CLEAR OUT RT-11 DEVICE #
42 027256 050311      BIS      R3,(R1)      ;STORE TSX DEVICE #
43         ;
44         ; Finished
45         ;
46 027260 012603      MOV      (SP)+,R3
47 027262 012602      MOV      (SP)+,R2
48 027264 012601      MOV      (SP)+,R1
49 027266 000207      RETURN
50         ;
51         ; Error: Could not locate Rt-11 device number in TSX device table.
52         ;
53 027270 MTSXDV: .PRINT #TSXHD          ;PRINT ERROR MESSAGE
54 027276         .PRINT #REQMIS        ;Missing a required device
55 027304 010200      MOV      R2,R0          ;GET RAD50 DEVICE NAME
56 027306 004737 030020' CALL     PRTR50          ;DISPLAY DEVICE NAME
57 027312         .PRINT #CRLF

```

TSINIT -- TSX startup initializ MACRO V05.05 Tuesday 17-Jan-89 13:55 Page 78-1
SETCHN -- Copy RT-11 channel information into TSX system chan

58 027320 000137 004102'

JMP INISTP

;ABORT INITIALIZATION

SETSY -- Set up information about SY device

```

1          .SBTTL  SETSY  -- Set up information about SY device
2          ;-----
3          ; SETSY is called to set up information about the SY device.
4          ; It does this by determining what device RT-11 recognizes as SY.
5          ;
6          ; Inputs:
7          ;   R5 = Address of base of free memory area
8          ;
9          ; Outputs:
10         ;   SYNAME = RAD50 spec for physical system disk
11         ;   SYINDX = TSX device table index for SY device
12         ;   SYUNIT = SY device unit number
13         ;
14 027324 010146 SETSY:  MOV     R1, -(SP)
15 027326 010246         MOV     R2, -(SP)
16         ;
17         ; Set up system device unit number
18         ;
19 027330         .GVAL   #AREA, #274      ; Get system unit # from RT-11 (high byte)
20 027350 010037 000000G MOV     R0, SYUNIT      ; Set system unit number
21         ;
22         ; Set up system device index number
23         ;
24 027354         .GVAL   #AREA, #364      ; Get RT-11 system device index number
25 027374 010002         MOV     R0, R2      ; Save device index number
26 027376         .GVAL   #AREA, #404      ; Get offset within RMON of PNAME table
27 027416 060002         ADD     R0, R2      ; Get offset to name of SY device
28 027420         .GVAL   #AREA, R2      ; Get name of RT-11 system device
29 027436 013701 000000G MOV     NUMDEV, R1      ; Get index to last TSX-Plus device entry
30 027442 020061 000000G 1$:  CMP     R0, PNAME(R1)    ; Search for device in TSX tables
31 027446 001405         BEQ     2$      ; Br if found it
32 027450 162701 000002         SUB     #2, R1      ; Keep looking if more
33 027454 002372         BGE     1$      ;
34 027456 010002         MOV     R0, R2      ; Save name of system device
35 027460 000703         BR      MTSXDV     ; Missing device error
36 027462 010137 000000G 2$:  MOV     R1, SYINDX    ; Store index # of TSX-Plus system device
37         ;
38         ; Set up RAD50 name of SY disk
39         ;
40 027466 113702 000001G         MOVB   SYUNIT+1, R2    ; GET SYSTEM UNIT NUMBER
41 027472 062702 000036         ADD     #36, R2      ; PUT IN "0" AS 3'RD CHARACTER OF NAME
42 027476 066102 000000G         ADD     PNAME(R1), R2    ; ADD DEVICE NAME
43 027502 010237 000000G         MOV     R2, SYNAME      ; THIS IS THE FULL SY DISK NAME
44         ;
45         ; Finished
46         ;
47 027506 012602         MOV     (SP)+, R2
48 027510 012601         MOV     (SP)+, R1
49 027512 000207         RETURN

```

RTFTCH -- Fetch a RT-11 device handler

```

1          .SBTTL  RTFTCH -- Fetch a RT-11 device handler
2          ;-----
3          ; RTFTCH is called to fetch an RT-11 device handler.
4          ; If the handler is already resident, nothing is done.
5          ; If the handler will fit in WRKBUF, it is fetched into there.
6          ; If the handler will not fit in WRKBUF, it is fetched into the top
7          ; of memory.
8          ;
9          ; Inputs:
10         ; RO = RAD50 device name.
11         ; R5 = Address of start of free memory.
12         ;
13         ; Outputs:
14         ; C-flag cleared ==> Fetch was successful.
15         ; C-flag set    ==> Error on fetch.
16         ;
17 027514 010046 RTFTCH: MOV     RO, -(SP)
18 027516 010246         MOV     R2, -(SP)
19 027520 010546         MOV     R5, -(SP)
20         ;
21         ; Set the name of the device being fetched
22         ;
23 027522 010037 000130' MOV     RO, FETDEV      ;Set name of device whose handler to fetch
24         ;
25         ; Do a .DSTAT to get information about the handler
26         ;
27 027526         .DSTAT  #DSTBLK, #FETDEV ;Get information about the device handler
28 027540 103425         BCS     9$          ;Br if device not recognized
29         ;
30         ; Determine if the handler is currently resident
31         ;
32 027542 005737 000116' TST     DSTBLK+4       ;Is the handler resident now?
33 027546 001021         BNE     8$          ;Br if yes
34         ;
35         ; The handler is not currently resident.
36         ; See if it will fit in WRKBUF.
37         ;
38 027550 013702 000152' MOV     WRKBUF, R2     ;Set address where handler will be loaded
39 027554 013700 000114' MOV     DSTBLK+2, RO   ;Get the size of the handler
40 027560 020037 000154' CMP     RO, WRKSIZ     ;Will handler fit in WRKBUF?
41 027564 101405         BLOS    1$          ;Br if handler will fit in WRKBUF
42         ;
43         ; Handler will not fit in WRKBUF.
44         ; See if there is room to load it into the top of memory.
45         ;
46 027566 060500         ADD     R5, RO          ;Get address above top of area needed
47 027570 020037 000132' CMP     RO, TOPMEM     ;Is there room for handler?
48 027574 101013         BHI     10$         ;Br if not
49 027576 010502         MOV     R5, R2          ;Set address where handler is to be loaded
50         ;
51         ; Fetch the handler
52         ;
53 027600 1$: .FETCH  R2, #FETDEV ;Try to fetch the handler
54 027610 103401         BCS     9$          ;Br if error on fetch
55         ;
56         ; We successfully fetched the handler
57         ;

```

```
58 027612 000241      8$:      CLC                ;Signal success on return
59                    ;
60                    ; Finished
61                    ;
62 027614 012605      9$:      MOV      (SP)+,R5
63 027616 012602      MOV      (SP)+,R2
64 027620 012600      MOV      (SP)+,R0
65 027622 000207      RETURN
66                    ;
67                    ; Insufficient memory available to load the handler
68                    ;
69 027624 004737 027650' 10$:     CALL     SIZERR          ;Generated system is too big -- abort
```

CHKMEM -- Check for memory space overflow

```

1          .SBTTL  CHKMEM -- Check for memory space overflow
2          ;-----
3          ;  CHKMEM is called to make sure we have not overflowed the available memory
4          ;  space while allocating space for TSX.
5          ;  If a memory overflow occurs, an error message is printed and
6          ;  the initialization is aborted.
7          ;
8          ;  Inputs:
9          ;  RO = Address to be tested for validity.
10         ;
11 027630 020037 000236'  CHKMEM: CMP      RO,MEMLIM      ; IS THE ADDRESS OK?
12 027634 103402          BLO      1$          ; BR IF OK
13 027636 004737 027650'          CALL    SIZERR      ; Generated system is too big -- abort
14 027642 004737 027670' 1$:      CALL    CCATST      ; CHECK FOR ^C ABORT REQUEST
15 027646 000207          RETURN
16
17         ;-----
18         ;  Generated system is too big.  Abort the initialization.
19         ;
20 027650          SIZERR: .PRINT  #TSXHD      ; PRINT MESSAGE HEADING
21 027656          .PRINT  #TOOBIG     ; PRINT ERROR MESSAGE
22 027664 000137 004102'          JMP     INISTP      ; ABORT INITIALIZATION
23
24         ;-----
25         ;  Check for control-C and abort initialization if requested.
26         ;
27 027670 005737 000040'  CCATST: TST     CCAFLG      ; DID USER REQUEST ^C ABORT?
28 027674 001402          BEQ     1$          ; BRANCH IF NOT
29 027676 000137 004102'          JMP     INISTP      ; ELSE ABORT INITIALIZATION
30 027702 000207          1$:      RETURN

```

PRTOCT -- Print octal value

```

1
2
3
4
5
6
7
8 027704 010146
9 027706 010246
10 027710 010001
11 027712 012702 000006
12 027716 005000
13 027720 073027 000001
14 027724 000403
15 027726 005000
16 027730 073027 000003
17 027734 062700 000060
18 027740
19 027744 077210
20 027746 012602
21 027750 012601
22 027752 000207

```

```

.SBTTL PRTOCT -- Print octal value
-----
; PRTOCT is called to print an octal value without trailing Cr-Lf.
;
; Inputs:
; RO = value to be printed.
;
PRTOCT: MOV R1, -(SP)
        MOV R2, -(SP)
        MOV RO, R1 ;GET VALUE TO PRINT
        MOV #6, R2 ;PRINT 6 DIGITS
        CLR RO
        ASHC #1, RO ;GET 1ST OCTAL DIGIT (1 BIT)
        BR 2$
1$: CLR RO ;INITIALIZE FOR SHIFT
   ASHC #3, RO ;SHIFT AN OCTAL DIGIT INTO RO
2$: ADD #'0, RO ;CONVERT TO ASCII CHARACTER
   .TTYOUT ;PRINT THE CHARACTER
   SOB R2, 1$ ;LOOP AND PRINT MORE DIGITS
   MOV (SP)+, R2
   MOV (SP)+, R1
   RETURN

```

```

1
2
3
4
5
6
7
8
9 027754 010146
10 027756 005046
11
12
13
14 027760 010001
15 027762 005000
16 027764 071027 000012
17 027770 062701 000060
18 027774 010146
19 027776 010001
20 030000 001370
21
22
23
24 030002 012600
25 030004 001403
26 030006
27 030012 000773
28
29
30
31 030014 012601
32 030016 000207

```

```

.SBTTL PRTDEC -- Print decimal value
-----
; PRTDEC is called to print a decimal value with leading zeroes suppressed
; and with no trailing Cr-Lf.
;
; Inputs:
; RO = Value to be printed
;
PRTDEC: MOV R1, -(SP)
        CLR -(SP) ; NULL ON STACK TO STOP US
;
; Convert value to ascii digit string and stack the digits.
;
        MOV RO, R1 ; GET VALUE TO BE CONVERTED
1$: CLR RO ; SET HIGH-ORDER PART OF VALUE TO 0
        DIV #10, RO ; DIVIDE RO-R1 BY 10.
        ADD #'0, R1 ; CONVERT REMAINDER TO ASCII DIGIT
        MOV R1, -(SP) ; AND STACK THE DIGIT
        MOV RO, R1 ; GET QUOTIENT
        BNE 1$ ; BR IF MORE DIGITS TO CONVERT
;
; Finished conversion. Print result.
;
2$: MOV (SP)+, RO ; GET A DIGIT FROM THE STACK
        BEQ 3$ ; BR IF REACHED END
        .TTYOUT ; PRINT THE DIGIT
        BR 2$ ; PRINT MORE
;
; Finished
;
3$: MOV (SP)+, R1
        RETURN

```

```

1                                     .SBTTL  PRTR50 -- Print Rad-50 value
2                                     ;-----
3                                     ; PRTR50 is called to print a Rad-50 value.
4                                     ;
5                                     ; Inputs:
6                                     ; R0 = value to be printed.
7                                     ;
8 030020 010146 PRTR50: MOV      R1,-(SP)
9 030022 010246      MOV      R2,-(SP)
10                                    ;
11                                    ; Convert value to ascii string and stack the characters.
12                                    ;
13 030024 012702 000003      MOV      #3,R2          ;GET # CHARS TO CVT
14 030030 010001      MOV      R0,R1          ;GET VALUE TO BE CONVERTED
15 030032 005000 1$:      CLR      R0          ;CLEAR HIGH-ORDER VALUE
16 030034 071027 000050      DIV      #50,R0          ;DIVIDE R0-R1 BY 50
17 030040 116101 003347'    MOVVB   R5OCHR(R1),R1      ;CONVERT REMAINDER TO ASCII CHARACTER
18 030044 010146      MOV      R1,-(SP)      ;STACK THE CHARACTER
19 030046 010001      MOV      R0,R1          ;GET QUOTIENT
20 030050 077210      SOB      R2,1$          ;BR IF MORE CHARS TO CONVERT
21                                    ;
22                                    ; Finished conversion. Print the result.
23                                    ;
24 030052 012702 000003      MOV      #3,R2          ;GET # CHARS TO PRINT
25 030056 012600 2$:      MOV      (SP)+,R0      ;GET NEXT CHARACTER
26 030060      .TTYOUT          ;PRINT THE CHARACTER
27 030064 077204      SOB      R2,2$          ;LOOP IF MORE CHARS TO PRINT
28                                    ;
29                                    ; Finished
30                                    ;
31 030066 012602      MOV      (SP)+,R2
32 030070 012601      MOV      (SP)+,R1
33 030072 000207      RETURN
34                                    ;-----
35                                    ; Define top of TSINIT
36                                    ;
37 030074 INITOP:
38                                    ;

```

```

1      .IF      NE,PROCID      ;Only assemble for protected Pro 350 version
2      ;
3      ; The following startup code is only included for the Pro version.
4      ; It is loaded here and executed very early during initialization
5      ; and subsequently overwritten by I/O buffers.
6      ;
7      .SBTTL  INSCHK -- Installation validation subroutines for Pro-350
8      .MCALL  .PRINT
9      ;
10     ; Reserve an arg block area for encryption calls
11     ;
12     EDARGB: .WORD  -32.      ;# OF BYTES TO BE DECRYPTED (EDMTH3)
13     EDADDR: .WORD  DSKBUF   ;POINTER TO BUFFER TO BE DECRYPTED
14     ;
15     ; Recover license number and decrypt disk image of Pro ID to intermed. state
16     ;
17     INSCHK: MOV      R1,-(SP)      ;SAVE REGISTERS
18             MOV      R2,-(SP)
19             MOV      R3,-(SP)
20             MOV      R4,-(SP)
21             MOV      R5,-(SP)
22             MOV      (PC)+,R0     ;DECRYPT LICENSE NUMBER
23             .RAD50  /SCB/        ;WITH THIS CODE
24             XOR      R0,LICNUM    ;BY XORING IT
25             MOV      LICNUM,TSXSIT ;MOVE LICENCE NUMBER TO TSGEN CELL
26             MOV      #EDARGB,R0   ;POINT TO ENC/DEC ARG BLOCK (PRESET)
27             CALL     EDMTH3       ;DECRYPT TO INTERMED STATE
28     ;
29     ; Copy Pro ID ROM low bytes into memory, and encrypt 1 step
30     ;
31     IDADDR  = 173600           ;ADDRESS OF START OF PRO 350 ID ROM
32             MOV      #IDADDR,R1   ;GET POINTER TO PRO ID ROM
33             MOV      #ROMBUF,R2   ;POINTER TO COPY OF HARDWARE ID
34             MOV      R2,EDADDR    ;SAVE ADDRESS FOR ENCRYPTION
35             NEG      EDARGB       ;MAKE +32. FOR ENCRYPTION
36             MOV      EDARGB,R0    ;ALSO USE AS LOOP COUNTER
37     3$:     MOVB     (R1)+,(R2)+   ;GET NEXT LOW BYTE
38             INC      R1           ;SKIP ID ROM HIGH BYTES
39             SOB      R0,3$        ;REPEAT THROUGH 32 BYTE ROM
40             MOV      #EDARGB,R0   ;POINT TO ENCRYPTION ARG BLOCK
41             CALL     EDMTH2       ;PERFORM METHOD 2 ENCRYPTION
42     ;
43     ; Have intermediate state of both hardware and disk copies of Pro ID
44     ; in memory. Verify them against each other and correct memory image
45     ; of SCHED at the same time.
46     ;
47             MOV      #ROMBUF,R1   ;POINT TO HARDWARE COPY OF ID
48             MOV      #DSKBUF,R2   ;POINT TO DISK COPY OF ID
49             MOV      #SCHED,R4    ;POINT TO CODE TO BE CORRECTED
50             MOV      EDARGB,R3    ;INIT LOOP COUNTER
51             CALL     GETLIC       ;USE LIC # AS SEED FOR EDPRNW, IN R0
52     4$:     CMPB     (R1)+,(R2)+   ;VERIFY ID'S ARE THE SAME
53             BNE      5$           ;ABORT IF NO MATCH ON ANY BYTE
54             CALL     EDPRNW       ;RANDOMIZE R0 FOR XOR (LIC# INIT SEED)
55             MOV      @R4,R5       ;GET ENCRYPTED CODE
56             XOR      R0,R5        ;RESTORE FUNCTIONAL CODE
57             MOV      R5,(R4)+     ;PUT DECRYPTED CODE BACK IN MEMORY

```

PRTR50 -- Print Rad-50 value

```
58          SOB      R3,4$          ; REPEAT THROUGH ID TESTS
59          MOV      (SP)+,R5       ; RESTORE REGISTERS
60          MOV      (SP)+,R4
61          MOV      (SP)+,R3
62          MOV      (SP)+,R2
63          MOV      (SP)+,R1
64          RETURN                    ; ID CHECKS AND CODE DECRYPTED
65          ;
66          5$:      .PRINT #TSXHD   ; ?TSX-F
67          .PRINT #NOTLIC         ; NOT LICENSED FOR THIS MACHINE
68          JMP      INISTP        ; ID'S DON'T MATCH, ABORT INIT.
69          ;
70          .NLIST BEX
71          NOTLIC: .ASCIZ /This copy of TSX-Plus not licensed for use on this machine./
72          .LIST BEX
73          .EVEN
74          ;
75          ; Subroutine to recover incremental license number. Assume it has been
76          ; decrypted already by XORing with .RAD50 /SCB/.
77          ;
78          GETLIC: MOV      LICNUM,R0 ; RETRIEVE DECRYPTED LIC # INTO R0
79          RETURN
80          ;
81          ; Reserve room for both disk and hardware copies of the Pro ID number
82          ; and for the incremental license number
83          ;
84          DSKBUF: .BLKB 32.         ; DISK IMAGE OF PRO ID
85          LICNUM: .WORD 0          ; INCREMENTAL LICENSE NUMBER
86          ROMBUF: .BLKB 32.         ; COPY OF ROM ID LOW BYTES
```

```

1          .SBTTL  EDEXPL -- Comments on encryption methods
2          ;
3          ; Encryption and decryption methods used here depend heavily on
4          ; pseudo-random numbers generated by the linear congrutential method.
5          ; See Hull and Dobell, SIAM Review, 4, 230, 1962.
6          ;
7          ; For the linear congruence relation:
8          ;
9          ;   X(I) == ( A * X(I-1) + C ) MOD M
10         ;
11         ;   X(I) is in the range 0 to M-1
12         ;
13         ; The sequence has full period M, provided that:
14         ;   1) C is relatively prime to M
15         ;   2) If p is a prime factor of M, A MOD p == 1
16         ;   3) If 4 is a factor of M, A MOD 4 == 1
17         ;
18         ; In the special case where M is a power of 2, these rules simplify to
19         ;   1) C must be odd
20         ;   2) A MOD 4 == 1
21         ;
22         .SBTTL  EDMTH2 -- Encryption method 2 (XOR with PRN high bytes)
23         ;
24         ; Using the license number as the initial seed, mask out the low 3 bits,
25         ; add 1 and call the PRN generator this many times to form the seed,
26         ; XOR each byte in the input buffer with the high byte of the next PRN
27         ; and replace the result in the input buffer. Decryption is accomplished
28         ; by a second application of the same process.
29         ;
30         ; Inputs:
31         ;   RO      Points to an arg block of the form:
32         ;           RO ---> buff_siz      ;word holding byte length of buffer
33         ;           buff_addr      ;address of buffer to be encrypted
34         ; Outputs:
35         ;   RO      Randomized
36         ;           input buffer encrypted
37         ;
38         EDMTH2:
39             MOV     R1,-(SP)      ;Save registers
40             MOV     R2,-(SP)
41             MOV     R3,-(SP)
42         ;
43         ; Fetch byte count, buffer pointer and initialize PRN seed
44         ;
45             MOV     (RO)+,R3      ;Fetch byte count of input buffer
46             MOV     (RO),R1      ;Fetch pointer to input buffer
47             CALL    GETLIC      ;Use license number as initial PRN seed
48             MOV     RO,R2      ;Copy license number to form repeat count
49             BIC     #^C7,R2     ;No more than 8 repeats
50             INC     R2          ;Make sure there is at least one
51         2$:     CALL    EDPRNW    ;Get a new PRN
52             SOB     R2,2$      ;Advance the seed between 1 and 8 times
53         ;
54         ; Now sweep the buffer, XORing each byte with the high PRN byte
55         ;
56         1$:     CALL    EDPRNW    ;With seed in RO, get next random number
57             MOVB   (R1),R2     ;Get next input byte

```

PRTR50 -- Print Rad-50 value

```

58          SWAB    R0          ;Reverse PRN high and low bytes
59          XOR     R0,R2       ;Encrypt the byte
60          SWAB    R0          ;Restore PRN high and low bytes for next seed
61          MOVB   R2,(R1)+     ;Save encrypted bytes back into input buffer
62          SOB    R3,1$        ;Repeat for entire input buffer
63          ;
64          MOV     (SP)+,R3     ;Restore registers
65          MOV     (SP)+,R2
66          MOV     (SP)+,R1
67          RETURN

```

```

1          .SBTTL  EDMTH3 -- Encryption/decryption meth 3 (swap bytes&shift bits)
2          ;
3          ; Using a prn of repeat length same as input string length, select prn
4          ; numbered bytes from the input string, combine them into a word,
5          ; shift the combined bytes a random number of bits, recombine the shifted
6          ; bits and set the confused bytes back into the prn selected string bytes.
7          ; Sign of the byte count indicates: + = encryption; - = decryption.
8          ; If the byte count is 0 or 1, no encryption occurs. If the byte count is
9          ; odd, then one random selected byte will not be encrypted.
10         ;
11         ; Inputs:
12         ;     RO      Points to an arg block of the form:
13         ;     RO ---> buff_siz      ; Word holding byte length of buffer.
14         ;                                     ; Note that buffer must be 512 or less
15         ;                                     ; in length. Flag encryption by using
16         ;                                     ; positive byte count ( 2 to 512. ).
17         ;                                     ; Flag decryption by using negative
18         ;                                     ; byte count (-2 to -512. ).
19         ;
20         ;     buff_addr      ; Address of buffer to be encrypted
21         ; Outputs:
22         ;     RO      Randomized
23         ;     Input buffer encrypted
24         ;
25         EDMTH3: MOV     R1, -(SP)      ; Save registers
26                 MOV     R2, -(SP)
27                 MOV     R3, -(SP)
28                 MOV     R4, -(SP)
29                 MOV     R5, -(SP)
30                 MOV     (RO)+, R3    ; Save the string length
31                 MOV     @RO, -(SP)  ; And save the input buffer pointer
32         ; Initialize prn generator of desired length
33                 MOV     R3, RO      ; Recover string length
34                 BGE     1$         ; Branch if encryption
35                 NEG     RO         ; If decryption, get real repeat
36                 INC     R3         ; If neg, correct for ASR round down
37         1$:      CALL    INPRNM     ; Set up for desired repeat length
38         ; Start encryption loop through string
39                 ASR     R3         ; Repeat for 1/2 the string length
40                 CALL    GETLIC     ; Get lic. num. for initial seed in RO
41                 CALL    EDPRNM     ; Seed PRN generator (-adjacent pairs)
42         2$:      TST     R3         ; Less than 2 bytes left?
43                 BEQ     9$         ; Quit if so (odd len -> 1 byte unch.)
44                 CLR     R4         ; Clean out shifting registers
45                 CLR     R5
46                 MOV     @SP, R1    ; Retrieve buffer pointer
47                 MOV     R1, R2     ; And second copy
48         ; Select first random byte
49                 CALL    EDPRNM     ; Randomize in range 0 - (strlen-1)
50                 ADD     RO, R1     ; Point to first random byte of pair
51         ; Select second random byte
52                 CALL    EDPRNM     ; Randomize again
53                 ADD     RO, R2     ; Point to next random byte
54         ; Use part of PRNM as semi-random shift amount
55                 MOV     RO, -(SP)  ; Save EDPRNM seed for later
56                 BIC     #^C6, RO   ; Get a semi-random shift amount
57         ; Select encryption or decryption

```

```

58          TST      R3          ;Positive for encryption
59          BMI      3$         ;Branch if decrypting
60          ; Do this part for encryption
61          BISB     @R1,R4      ;Get first byte without sign extend
62          SWAB     R4          ;And put it in the high byte
63          BISB     @R2,R4      ;Combine it with first byte
64          CLC          ;Always do at least one shift
65          ROR      R4          ;Shift once
66          ROR      R5          ;Get low bit into r5
67          NEG      R0          ;Right shifts for encryption
68          DEC      R3          ;Reduce count of pairs remaining
69          BR       4$         ;Skip decryption stuff
70          ; Do this part for decryption
71          3$: BISB     @R1,R5      ;Get first byte without sign extend
72          SWAB     R5          ;And put it in the high byte
73          BISB     @R2,R5      ;Combine it with the first byte
74          CLC          ;Always do at least one shift
75          ROL      R5          ;Shift once
76          ROL      R4          ;Get high bit into R4
77          INC      R3          ;Reduce count of pairs remaining
78          ; Shift and recombine the (enide)rypted bytes
79          4$: ASHC     R0,R4      ;Shift combined bytes 0,2,4 or 6 more
80          BIS      R5,R4        ;Recombine bytes
81          MOV      (SP)+,R0      ;Recover EDPRNM seed
82          ; Now put encrypted bytes back into input string
83          MOVB     R4,@R2        ;Store low byte at second byte place
84          SWAB     R4          ;Get high byte
85          MOVB     R4,@R1        ;Store high byte at first byte place
86          BR       2$         ;Repeat through string
87          ; Done, restore registers and return
88          9$: MOV      (SP)+,R0    ;Just pop saved buffer address
89          MOV      (SP)+,R5      ;Restore registers
90          MOV      (SP)+,R4
91          MOV      (SP)+,R3
92          MOV      (SP)+,R2
93          MOV      (SP)+,R1
94          RETURN

```

```
1          .SBTTL  EDPRNW -- Pseudo random number generator with MOD 2^16
2          ;
3          ; Linear congruential pseudo-random number generator with maximum repeat
4          ; length of 65536 (2^16). cf. Hull and Dobell and Knuth, vol 2.
5          ;
6          ; Inputs:
7          ;      RO      Seed value
8          ;
9          ; Outputs:
10         ;      RO      New PRN, should be used for next seed
11         ;
12         EDPRNW:
13             MOV     R4, -(SP)      ; Save registers
14             MOV     R5, -(SP)
15             MOV     RO, R4        ; Get seed to be multiplied
16             MOV     (PC)+, R0     ; Fetch multiplier
17         EDPRNA: .WORD 104375      ; Multiplier, can be replaced
18             MUL     R0, R4        ; Multiply by A
19             ADD     (PC)+, R5     ; Add C
20         EDPRNC: .WORD 012705     ; Additive, can be replaced
21             MOV     R5, R0        ; Return result mod 65536. as PRN
22             MOV     (SP)+, R5    ; Restore registers
23             MOV     (SP)+, R4
24             RETURN
```

PRTR50 -- Print Rad-50 value

```
1           .SBTTL  INPRNM -- Initialize PRN generator with repeat range M
2           ;
3           ; Using the Hull and Dobell rules, determine acceptable values for
4           ; A and C to get a repeat range of M.
5           ;
6           ; Outputs:
7           ;   EDMULA Set with first acceptable multiplier
8           ;   EDADDC Set with first acceptable additive factor
9           ;   EDMODM Set with desired repeat length
10          ;
11          INPRNM:
12             MOV     #32,EDMODM      ;Get repeat length to cover Pro ID
13             MOV     #5,EDMULA       ;Use first valid A
14             MOV     #3,EDADDC       ;And first valid C
15             RETURN
```

```

1          .SBTTL  EDPRNM -- Generate pseudo-random number in specified range M
2          ;
3          ; *****
4          ; * INPRNM MUST BE CALLED BEFORE FIRST SEED IS PASSED TO THIS ROUTINE!!!! *
5          ; *****
6          ;
7          ; Using linear congruential method (cf. Hull and Dobell), generate
8          ; pseudo-random number using seed passed in R0. Return new PRN in R0.
9          ;
10         ; Inputs:
11         ; R0      Seed value, must be in range 0 to M (EDMODM)
12         ;
13         ; Outputs:
14         ; R0      New pseudo-random number, should be used for next seed
15         ;
16         EDPRNM:
17         MOV     R4, -(SP)      ; Save R4 and R5
18         MOV     R5, -(SP)
19         CMP     R0, EDMODM    ; Is seed in range 0 to EDMODM?
20         BLO    1$            ; Branch and proceed if so
21         MOV     R0, R5        ; Set up to divide it by EDMODM
22         CLR     R4            ; Set up for divide
23         DIV     EDMODM, R4    ; Divide it
24         MOV     R5, R0        ; And use remainder as seed
25         1$: MOV     R0, R4     ; Get current seed ready to be multiplied
26         MUL     (PC)+, R4     ; Multiply by chosen A
27         EDMULA: .WORD 25173.  ; Replace at run-time with 5
28         ADD     (PC)+, R5     ; Add in C
29         EDADDC: .WORD 13849.  ; Replace at run-time with 3
30         CLR     R4            ; Clear high word for division
31         DIV     (PC)+, R4     ; Perform mod M
32         EDMODM: .WORD 256.    ; Replace at run-time with 32.
33         MOV     R5, R0        ; Return remainder
34         MOV     (SP)+, R5     ; Restore R4 and R5
35         MOV     (SP)+, R4
36         RETURN
37         ;
38         .IFF    ; NE, PROCID  ; Assemble if protection code not included
39 030074 DSKBUF:  ; Define dummy DSKBUF global symbol
40         .ENDC    ; NE, PROCID
41         ;
42         ; Address of real top of TSINIT, including PRO init code
43         ;
44 030074 PROITP:
45 000000 .CSECT  TSXEND
46         .END

```

Errors detected: 0

*** Assembler statistics

Work file reads: 0
 Work file writes: 0
 Size of work file: 11518 Words (45 Pages)
 Size of core pool: 17920 Words (70 Pages)
 Operating system: RT-11

Elapsed time: 00:02:16.32

TSINIT -- TSX startup initializ MACRO V05.05 Tuesday 17-Jan-89 13:55 Page 90-1
PRTR50 -- Print Rad-50 value

DK: TSINIT, LP: TSINIT=DK: TSINIT/C/N: SYM

CL\$ORE	1-125	41-46*						
CL\$ORG	1-125	41-38*						
CL\$ORP	1-125	41-37*						
CL\$ORS	1-127	41-44*						
CL\$STA	1-127	41-83*						
CL\$VER	2-13#	25-227						
CLDEVX	1-143	33-29	41-94*					
CLEDFS	1-126	41-53	41-54					
CLHEAD	1-143	41-99	41-117					
CLINCP	1-155	24-21						
CLINIT	26-72	41-11#						
CLK100	1-188#	7-165	25-14*					
CLKRTI	1-107	7-71						
CLKVEC	1-158	7-71*	7-72*	7-165*	25-14	26-77*		
CLORSZ	1-112	41-39						
CLOTIR	1-155	24-20						
CLSIZE	1-143	41-100						
CLSTS	1-118	41-96	41-114					
CLTOTL	1-113	26-70	33-18	41-18	41-101	41-108	70-156	
CLVERS	1-111	25-224	25-227*					
CO\$BBT	1-130	41-74						
CO\$DEF	1-126	41-62						
CO\$FF	1-126	41-71						
CO\$TAB	1-126	41-68						
CONFG2	1-105	25-165*	25-166*	25-169*	26-78			
CONFIG	1-104	25-163*	26-5					
CONSPC	6-13#	30-108						
COSRT	6-36#	74-94						
CRLF	6-9#	29-96	29-101	30-111	30-119	59-12	74-99	78-57
CS\$ENT	1-145	77-24						
CS\$NMX	1-88	77-34						
CS\$OPN	1-145	77-24						
CSHALC	1-122	37-71	70-168	75-17	75-18	75-28	75-44	
CSHBAS	1-105	9-39	11-37*					
CSHBFP	1-148	75-47*						
CSHBUF	26-205	75-12#						
CSHDEV	1-86	37-63*						
CSHDVN	1-86	37-67*						
CSHOVF	6-41#	75-79						
CSHSIZ	1-148	75-33*						
CSHVEC	1-106	9-42						
CURDEV	1-205#	60-54*	61-40	63-62				
CURNAM	1-206#	52-25*	57-20*	59-10				
CVTDVU	32-37	35-14#	77-29					
CW\$50H	1-104	26-5						
CW\$BTH	1-103	25-161						
CW\$ESP	1-113	25-166						
CW\$FB	1-103	25-162						
CW\$FGJ	1-103	25-162						
CW\$GDH	1-103	25-161						
CW\$LGS	1-103	25-161						
CW\$PRO	1-107	25-116	26-78					
CW\$QBS	1-115	25-169						
CW\$USR	1-104	25-162						
CW\$XM	1-104	25-162						
CXTALC	26-92	46-12#						

... CMO	30-84	31-68	32-110	42-91	43-66	80-27	80-53							
... CM1	8-29	26-242	30-50	30-62	30-70	31-30	31-47	31-55	32-85	32-94	32-99	40-21		
	40-27	40-62	42-22	42-29	42-31	42-68	42-78	42-82	42-84	43-43	43-52	43-55		
	43-60	53-26	53-38	55-25	57-21	57-30	57-64	60-36	60-50	61-35	67-17	69-24		
	73-30	74-25	74-54	76-20	76-32	76-51	76-70	78-25						
... CM2	8-29	8-29	8-29	8-29	25-19	25-23	25-61	25-63	25-78	25-182	26-220	26-242		
	26-242	26-242	26-242	30-50	30-50	30-58	30-58	30-62	30-62	30-62	30-70	30-70		
	30-70	30-70	31-30	31-30	31-43	31-43	31-47	31-47	31-47	31-55	31-55	31-55		
	31-55	32-85	32-85	32-90	32-90	32-94	32-94	32-94	32-94	32-99	32-99	32-99		
	40-21	40-21	40-27	40-27	40-27	40-27	40-27	40-62	40-62	42-22	42-22	42-29		
	42-29	42-29	42-31	42-31	42-31	42-31	42-31	42-68	42-68	42-73	42-73	42-78		
	42-78	42-82	42-82	42-82	42-82	42-84	42-84	43-43	43-43	43-48	43-48	43-52		
	43-52	43-52	43-55	43-55	43-55	43-55	43-60	53-26	53-26	53-38	53-38	53-38		
	53-38	55-25	55-25	55-25	55-25	57-21	57-21	57-30	57-30	57-30	57-30	57-30		
	57-64	57-64	57-64	60-36	60-36	60-36	60-36	60-50	60-50	60-50	60-50	60-50		
	61-35	61-35	61-35	67-17	67-17	67-17	67-17	69-24	69-24	69-24	69-24	69-24		
	73-30	73-30	73-30	74-25	74-25	74-54	74-54	74-54	74-54	74-54	76-20	76-32		
	76-32	76-32	76-32	76-51	76-51	76-51	76-51	76-70	76-70	76-70	76-70	76-70		
	78-32	78-34	79-19	79-24	79-26	79-28								
... CM3	30-57	30-72	31-42	31-56	32-89	32-101	42-72	42-83	43-47	43-65	51-77	57-113		
	74-81	76-95												
... CM5	8-29	8-45	8-46	25-19	25-23	25-31	25-32	25-33	25-36	25-54	25-55	25-61		
	25-63	25-67	25-78	25-182	26-220	26-225	26-226	26-242	29-84	29-85	29-91	29-92		
	29-96	29-97	29-101	30-50	30-58	30-62	30-70	30-84	30-95	30-96	30-102	30-103		
	30-108	30-111	30-116	30-119	31-30	31-43	31-47	31-55	31-68	31-77	31-78	31-84		
	31-85	32-60	32-61	32-85	32-90	32-94	32-99	32-110	32-131	32-132	32-138	32-139		
	40-21	40-27	40-62	40-75	40-76	40-81	40-82	42-22	42-29	42-31	42-68	42-73		
	42-78	42-82	42-84	42-91	42-101	42-102	42-108	42-109	43-43	43-48	43-52	43-55		
	43-60	43-66	43-76	43-77	43-83	43-84	45-160	45-161	47-76	47-77	53-26	53-38		
	55-25	57-21	57-30	57-64	59-8	59-9	59-12	60-36	60-50	61-35	67-17	67-51		
	67-52	69-24	69-52	69-53	69-58	69-59	72-34	72-35	73-30	73-73	73-74	74-25		
	74-54	74-93	74-94	74-99	74-104	74-105	75-78	75-79	76-20	76-22	76-23	76-32		
	76-51	76-70	76-138	76-139	78-25	78-32	78-34	78-53	78-54	78-57	79-19	79-24		
	79-26	79-28	80-27	80-53	81-20	81-21	82-18	83-26	84-26					
... CM6	25-19	25-23	25-61	25-63	25-78	25-182	26-220	78-32	78-34	79-19	79-24	79-26		
	79-28													
... CM7	8-29	26-242	30-70	31-55	32-99	40-27	42-29	42-31	42-82	43-55	53-38	55-25		
	57-30	57-64	60-36	60-50	61-35	67-17	69-24	73-30	74-54	76-32	76-51	76-70		
CLOSE	1-50#	30-57	30-72	31-42	31-56	32-89	32-101	42-83	43-65	51-77	57-113	74-81		
	76-95													
CSTAT	1-53#	43-60												
DATE	1-52#	26-221												
DELET	1-51#	30-58	31-43	32-90	42-73	43-48								
DSTAT	1-52#	80-27												
ENTER	1-49#	30-62	31-47	32-94	42-78	43-52								
EXIT	1-51#	7-168	8-47											
FETCH	1-52#	80-53												
GTIM	1-52#	25-19	26-220											
GVAL	1-49#	25-78	25-182	78-32	78-34	79-19	79-24	79-26	79-28					
HERR	1-51#	25-57												
LOCK	1-52#	25-81												
LOOKU	1-49#	30-50	31-30	32-85	40-21	40-62	42-22	42-68	42-84	43-43	53-26	57-21		
	74-25	76-20												
PRINT	1-50#	8-45	8-46	25-31	25-32	25-33	25-36	25-54	25-55	26-225	26-226	29-84		
	29-85	29-91	29-92	29-96	29-97	29-101	30-95	30-96	30-102	30-103	30-108	30-111		
	30-116	30-119	31-77	31-78	31-84	31-85	32-60	32-61	32-131	32-132	32-138	32-139		

