

Table of contents

8-	1	* * * TSX Initialization * * *
8-	2	* * * Initialization taking over control from RT-11 * * *
9-	1	LODINI -- Load a segment over TSINIT
10-	1	INIOVL -- Load system overlays over TSINIT
11-	1	ENTVEC -- Set up entry point vector for overlay
12-	1	KEYSEG -- Remember memory position of system overlays
14-	1	DEVVEC -- Set up device vectors
15-	1	SETVEC -- Set up an interrupt vector for a device
16-	2	PIDVEN -- Make device table entry for PI device
25-	1	LINTYP -- Determine the type of a line
26-	1	* * * Initialization done with RT-11 running * * *
28-	1	* * * Subroutines * * *
28-	2	ALCWRK -- Allocate a work buffer
29-	1	ALCHRB -- Allocate Region Control Blocks for handlers
31-	1	OPNSWP -- Open system swap file
32-	1	OPNRSF -- Open PLAS region swap file
33-	1	SPLINI -- Initialize spooling system
34-	1	SPLCLD -- Set up spooling to a CL device
35-	1	CHKCLD -- See if a device name is a CL or C1 unit
36-	1	CVTDVU -- Convert device name to dev index and unit #
37-	1	FORCEO -- Force a 2-char dev name to unit 0
38-	1	ALOCBF -- Allocate buffer space
39-	1	ALCSLO -- Allocate silo buffers for lines
40-	1	ALBFX -- Allocate buffers in extended memory region
41-	1	OPNKMN -- Open channel to TSKMON
42-	1	CLINIT -- Initialize CL handler
43-	1	INDINI -- Initialize IND program
44-	1	UCLINI -- Initialize TSXUCL data file
45-	1	MEMINI -- Initialize memory management
46-	1	MEMTST -- Set up information about available memory space
47-	1	CXTALC -- Set up info about job context area
48-	1	MAPALC -- Allocate memory usage table
49-	1	SETJSZ -- Set up information about maximum job sizes
51-	1	GETHNL -- Load device handlers into memory
52-	1	LDHAND -- Load a device handler
53-	1	INSCK1 -- Determine if a handler should be installed
54-	1	INSCK2 -- Additional checking for handler installation
55-	1	STDVTB -- Set up device table entries for a device
56-	1	LDHNLO -- Load device handler into low memory
57-	1	GETHNH -- Load handlers into extended memory
58-	1	LDHNHI -- Load device handler into extended memory
59-	1	STHNPV -- Initialize pointer vector in a handler
61-	1	DOHNLC -- Execute and handler load/fetch code
62-	1	LDREAD -- Perform I/O for handler load code
63-	1	HANMAP -- Set up KPAR5 to access a mapped handler
63-	42	HANUMP -- Turn off memory mapping to a handler
64-	2	FNDHRB, HANXMR Inoperative Pro versions
67-	1	OVLPOS -- Determine which overlays go over TSINIT
68-	1	OVLBLD -- Build overlay information table
69-	1	GETMAP -- Load any mapped system code regions
70-	1	ALCOVL -- Allocate space for a system overlay region
71-	1	OPTOVL -- Check for optional system overlay regions
72-	1	OVLTRY -- Find an overlay to place over TSINIT
73-	1	GETOVL -- Load system overlay into high memory
74-	1	LODOVL -- Read and relocate system overlay
75-	1	GETSRT -- Load any shared run-time systems
76-	1	CSHBUF -- Allocate space for data cache tables

TSINIT -- TSX startup initializ MACRO V05.05 Friday 20-Jan-89 11:30

Table of contents

78-	1	OPNCHN -- Open a TSX-Plus channel
79-	1	SETCHN -- Copy RT-11 channel information into TSX system chan
80-	1	SETSY -- Set up information about SY device
81-	1	RTFTCH -- Fetch a RT-11 device handler
82-	1	CHKMEM -- Check for memory space overflow
83-	1	PRTDCT -- Print octal value
84-	1	PRTDEC -- Print decimal value
85-	1	PRTR50 -- Print Rad-50 value
86-	7	INSCHK -- Installation validation subroutines for Pro-350
87-	1	EDEXPL -- Comments on encryption methods
87-	22	EDMTH2 -- Encryption method 2 (XOR with PRN high bytes)
88-	1	EDMTH3 -- Encryption/decryption meth 3 (swap bytes&shift bits)
89-	1	EDPRNW -- Pseudo random number generator with MOD 2 ¹⁶
90-	1	INPRNM -- Initialize PRN generator with repeat range M
91-	1	EDPRNM -- Generate pseudo-random number in specified range M

1 000001 PROASM = 1 ; Assemble for Professional

```

2          .TITLE  TSINIT -- TSX startup initialization
3          .ENABL  LC
4          .ENABL  AMA
5          .DSABL  GBL
6          .CSECT  TSINIT
7
8          TSINIT:
9          ;
10         ;   There are two external assembly-time switches related to assembling
11         ;   TSINIT for execution on a PRO or a PDP-11.
12         ;
13         ;   The following values for the PROASM flag are defined:
14         ;   0 ==> Assemble for PDP-11 (not Pro) only.
15         ;   1 ==> Assemble for Pro only.
16         ;   2 ==> Assemble for either PDP-11 or Pro execution.
17         ;
18         ;   The following values for the PROCID flag are defined:
19         ;   0 ==> Do not lock system to ID number.
20         ;   1 ==> Lock system to ID number.
21         ;
22         .IF      NDF,PROASM          ; If PROASM not defined
23         PROASM   =      0             ; Default value for PROASM if not defined
24         .ENDC      ; NDF,PROASM
25         ;
26         .IF      NDF,PROCID          ; If PROCID not defined
27         .IF      EQ,<PROASM-1>        ; If assembling for PRO only
28         PROCID   =      1             ; Then check ID by default
29         .IFF                          ; If not assembling for PRO only
30         PROCID   =      0             ; Then don't check ID number
31         .ENDC      ; EQ,<PROASM-1>
32         .ENDC      ; NDF,PROCID
33         ;
34         .IF      EQ,PROASM
35         .GLOBL   TSXPRO
36         TSXPRO   =      0             ; Define dummy base for TSXPRO if not PRO
37         .ENDC
38         ;
39         -----
40         ;   TSINIT is the initialization module of TSX that is executed once
41         ;   during system startup. Time-sharing character buffers and other
42         ;   run-time data areas are allocated over TSINIT.
43         ;
44         ;   Copyright 1980,1981,1982,1983,1984,1985,1986,1987,1988,1989.
45         ;   S&H Computer Systems, Inc.
46         ;   Nashville, TN  USA
47         ;
48         ;   Macro calls
49         ;
50         .MCALL   .LOOKUP,.ENTER,.READW,.SAVESTATUS,.GVAL
51         .MCALL   .TRPSET,.SETTOP,.CLOSE,.TTYOUT,.PRINT,.PURGE
52         .MCALL   .DELETE,.WRITW,.SERR,.HERR,.EXIT,.UNLOCK
53         .MCALL   .FETCH,.RELEASE,.LOCK,.GTIM,.DATE,.DSTATUS
54         .MCALL   .SCCA,.CSTAT
55         ;
56         ;   Global definitions
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          ;   Global references
81          ;
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$$SZ
87          .GLOBL  LSTPL, LMXNUM, MXCSR, MXVEC, RSR, INVEC, VHIMEM, CXTPAQ
88          .GLOBL  LSWPBK, LSTSL, SWDBLK, SWPCHN, NUMDEV, CS$NMX, SCHED
89          .GLOBL  H. DSTS, DVSTAT, HANENT, H. GEN, FORK, INTEN, PNAME
90          .GLOBL  $SXON, LSW10, LHIRBB, LHIRBE, LHIRBP, LHIRBG, LHIRBA
91          .GLOBL  LHIRBS, LHIRBC, VNC$LO, VNCXOF, VNCXON, SDDVU, VMSCHR, MAXSLO
92          .GLOBL  MPAR0, MPAR16, PAREN1, MPARFL, TSEMT, VDBFLG, DX$EBA
93          .GLOBL  H. SIZ, HANSIZ, H. DVSZ, DEVSIZ, LOMAP, MMENBL, UPAR7
94          .GLOBL  PSW, HIMAP, FSTD1, LSTD1, LINBUF, LINSIZ, NUMCCB, TK1SEC
95          .GLOBL  FRKINI, FRKGEN, NUMFRK, FQ$$SZ, H. CSR, H. INS, VSWPSL, DMYDEV
96          .GLOBL  LINEND, LOTBUF, LOTSIZ, LOTEND, KMNTOP, KMNHI, NSL, NDL
97          .GLOBL  DX$MPH, DX$NHM, DX$IBH, HANPAR, HANXIT, MAPPAR, LINSPL
98          .GLOBL  KMNPGS, KMNSTK, KMNSTR, KMNCHN, SROMMR, KPAR0, PROFLG
99          .GLOBL  EMMAP, IOMAP, SR3MMR, IOPAGE, MAPSIZ, SR3FLG, NSPLDV
100         .GLOBL  UDDRO, IDSFLG
101         .GLOBL  UPARO, KPDRO, UPDRO, KPAR7, BASMAP, PTWRD, PTBYT, LOKMEM
102         .GLOBL  GTBYT, MPPHY, RELOC, BRKPT, TSGEN, TSEEXEC, VSWPFL
103         .GLOBL  CW$GDH, CW$BTH, CW$LGS, CW$FB, CW$FGJ, MSGBAS, RPRVEC
104         .GLOBL  CW$USR, CW$XM, CONFIG, CW$5OH, JMPO, DTLX, USRBAS, WINBAS
105         .GLOBL  DATIML, DATIMH, RMON, CONFQ2, SG$ELG, SG$IOT, CSHBAS
106         .GLOBL  SG$PAR, SG$MTS, SG$MMU, SG$MTM, LTPAR, LOKBAS, CSHVEC, LOKVEC
107         .GLOBL  SYSGEN, AUTHAN, AHEND, CLKRTI, TRP4, CW$PRO, TIOVEC
108         .GLOBL  TRP10, TRP20, TRP24, EMTEMT, TRP34, INIIMP, MHNSIZ
109         .GLOBL  TK1VAL, INRECV, OTRECV, INMXV, OTMXV, DHBFSZ, 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$$SZ, SWPJOB, SWPPOS
117 . GLOBL TSR, RBR, RDINT, LSTMX, SS, CHAIN, JSWLOC, MU$TXT, SLTSIZ
118 . GLOBL NUMIOQ, FREIOQ, UMODE, FPTRAP, MXLNT, DI$LD, DI$CL, CLSTS
119 . GLOBL FREPGS, IOQSZ, SYUNIT, UMSYTP, DI$TT, CXTBUF, SSEND
120 . GLOBL SYINDX, MONVEC, KMNBAS, SDANAM, VBUSTP, MINCTR
121 . GLOBL NUMRDB, RDB, RDBEND, RT$SKP, RT$TOP, NLINES, SHRRCB, SHRRCN
122 . GLOBL RT$BAS, UPMODE, SPLNB, CSHALC, NIOL, CHNSIZ, RC$$SZ, VNGR
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, $OVRH, SYMAP, MAPSYS, VSLEDT, LCLUNT
138 . GLOBL UBUSMP, UMRADR, IOMAP, QBUS, UNIBUS, DX$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, $MEMSZ, PHYMEM, SG$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.

```

```

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      000050      000000      SAVBLK: .BLKW      5
193      000062      075250      100020      000000      TSXSAV: .RAD50    /SY TSX      SAV/
194      000070      073376
195      000072      075250      100003      051646      KMNAM: .RAD50    /SY TSKMONSAV/
196      000100      073376
197      000102      075250      011504      000000      CCLNAM: .RAD50    /SY CCL      SAV/
198      000110      073376
199      000112      000000      000000      000000      DSTBLK: .WORD      0,0,0,0
200      000120      000000
201      000122      000000      XMVBAS: .WORD      0
202      000124      000000      NMXHAN: .WORD      0
203      000126      000000      HMAP:   .WORD      0
204      000130      000000      FETDEV: .WORD      0
205      000132      000000      TOPMEM: .WORD      0
206      000134      000000      FMEMHI: .WORD      0 ; 64-byte block # below high alloc memory
207      000136      000000      FMEMLO: .WORD      0 ; 64-byte block # above top of low alloc memory
208      000140      000000      OVLBAS: .WORD      0 ; Start loading overlays over TSINIT from here
209      000142      000000      FILBLK: .WORD      0
210      000144      000000      CURDEV: .WORD      0
211      000146      000000      CURNAM: .WORD      0
212      000150      000000      PROBUF: .WORD      0
213      000152      025350      WRKBUF: .WORD      INITOP
214      000154      004000      WRKSIZ: .WORD      2048.
215      000156      000322      RTVPTR: .WORD      RTVEND ; Initially no limit on system version number
216      ; The following RAD50 definitions were converted from word cells to
217      ; definitions to conserve space at V6.40. All subsequent references to
218      ; one of these cells are converted from:
219      ; relative (cmp R50xxx,r1) to immediate (cmp #R50xxx,r1).
220      052077      R50MSG   = ^RMSG
221      110466      R50WIN   = ^RWIN
222      046543      R50LOK   = ^RLOK
223      103112      R50USR   = ^RUSR
224      012700      R50CSH   = ^RCSH
225      077167      R50TID   = ^RTID
226      100040      R50TT    = ^RTT ; "TT "
227      075250      R50SY    = ^RSY ; "SY "
228      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      MEM LIM: .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 000000 RTVER: ; V4.0 is the earliest supported version of RT-11
18 000001 RT40: .BYTE 4,0,-1 ; 4.0 did not support XL
19 000002 RT50: .BYTE 5,0,-1 ; 5.0 did not support XL
20 000003 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:
    . 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     000322         HANVER  DI$DU, RT50      ; DU - (5.0)
25     000326         HANVER  DI$XL, RT51B     ; XL - (5.1B)
26     000332         HANVER  DI$MU, RT54      ; MU - (5.4)
27     000336 HVEND:

```

```

1
2      ; -----
3      ; The following table defines default control flags for certain devices.
4      ;
5      ; DV$NAM = 0 ; Rad50 name of device
6      ; DV$FLG = 2 ; Flags for device
7      ; DV$$SZ = 4 ; Size of a table entry
8      ;
9      . MACRO DEFFLG DEV, FLAGS
10     . RAD50 /'DEV/ ; DV$NAM
11     . WORD FLAGS ; DV$FLG
12     . ENDM DEFFLG
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      ; -----
3      ; The following data structures are used to hold information about
4      ; TSX-Plus overlays as they are being initialized.
5      ;
6      ; Offsets in structure for each overlay
7      ;
8      000000      OS$SIZ =      0      ; Total space needed for overlay
9      000002      OS$FLG =      2      ; 0==>Load into XM space, 1==>over TSINIT
10     000004      OS$OVL =      4      ; Pointer to overlay table entry
11     000006      OS$$SZ =      6      ; Size of each overlay entry
12
13     000031      MAXOVL =     25.      ; Maximum number of system overlays
14
15     000516      OSTABL: .BLKB  OS$$SZ*MAXOVL  ; Reserve room for table
16     000744      OSEND:      ; -Define end of table
17     000744 000516'  OSLAST: .WORD  OSTABL      ; Pointer past last used entry in table
18
19     ; Table of system overlays that must be loaded over TSINIT
20
21     000746 012700  LOWOVL: .RAD50  /CSH/      ; TSCASH
22     000750 077167      .RAD50  /TIO/      ; TSTIO
23     000752 046543      .RAD50  /LOK/      ; TSLOCK
24     000754      LOWEND:      ; End of table

```

```

1
2      ; -----
3      ;   Text messages
4      ;
5      .NLIST  BEX
6 000754      077      124      123  TSXHD:  .ASCII  /?TSX-F-/<200>
7 000764      111      156      166  BADLIN: .ASCII  'Invalid CSR for T/S line: '<200>
8 001017      111      156      166  BDVMSG: .ASCII  'Invalid vector for T/S line: '<200>
9 001055      114      151      156  BDLMSG: .ASCII  /Line # = /<200>
10 001067      000
11 001070      124      123      130  REQDIS: .ASCII  /TSX generation did not include device /<200>
12 001137      103      141      156  BADOPN: .ASCIIZ  /Cannot open program swap file/
13 001175      103      141      156  RSFERR: .ASCIIZ  /Cannot open PLAS region swap file/
14 001237      116      165      155  CONSPC: .ASCII  /Number of contiguous blocks needed = /<200>
15 001305      103      141      156  BDSPOP: .ASCII  /Cannot open spooled device: /<200>
16 001342      111      156      163  BOSF:   .ASCIIZ  /Insufficient disk space for spool file/
17 001411      103      141      156  NOKMON: .ASCIIZ  /Cannot find "SY:TSKMON.SAV" file/
18 001452      103      141      156  NOCCL:  .ASCIIZ  /Cannot find "SY:CCL.SAV" file/
19 001510      103      141      156  CFHMSG: .ASCII  /Cannot find device handler file: /<200>
20 001552      105      162      162  ERHMSG: .ASCII  /Error reading device handler file: /<200>
21 001616      111      156      166  ERHNDV: .ASCII  /Invalid RT-11 version for device handler: /<200>
22 001670      111      156      166  NOCSRR: .ASCII  /Invalid CSR for device: /<200>
23 001721      105      162      162  ERHINS: .ASCII  /Error executing installation code for device: /<200>
24 002000      124      123      130  TSXRUN: .ASCIIZ  /TSX is already running/
25 002027      110      141      156  HSGER:  .ASCII  /Handler not generated with XM support: /<200>
26 002077      103      157      155  NOCLOK: .ASCIIZ  /Computer line time clock (50 or 60 Hz) is not working/
27 002165      116      157      040  NXMMMSG: .ASCIIZ  /No memory management hardware/
28 002223      116      157      040  NEXMSG:  .ASCIIZ  /No extended memory management hardware/
29 002272      103      141      156  NOODT:  .ASCIIZ  /Cannot locate "SY:SYSODT.REL" file/
30 002335      115      141      160  HN2BIG: .ASCII  /Mapped handler is larger than 8KB: /<200>
31 002401      105      162      162  ODTRDM: .ASCIIZ  /Error on read of SYSODT rel file/
32 002442      110      141      156  NOSYDV: .ASCIIZ  /Handler for SY device was not loaded/
33 002507      107      145      156  TOOBIG: .ASCIIZ  /Generated TSX system is too large/
34 002551      122      145      144  REDUCE: .ASCII  /Reduce size of TSGEN by /<200>
35 002602      056      040      142  BYTES:  .ASCIIZ  /. bytes/
36 002612      111      156      163  PHSOVF: .ASCIIZ  /Insufficient total physical memory for generated system/
37 002702      103      141      156  COSRT:  .ASCII  /Cannot open shared run-time file: /<200>
38 002745      103      141      156  SVERR:  .ASCIIZ  /Cannot locate "SY:TSX.SAV"/
39 003000      111      156      163  TSXSIZ: .ASCIIZ  /Insufficient memory to load all mapped system regions/
40 003066      105      162      162  RDERR:  .ASCIIZ  /Error reading "SY:TSX.SAV"/
41 003121      111      156      163  SRTOVF: .ASCIIZ  /Insufficient memory to load all shared run-time systems/
42 003211      111      156      163  CSHOVF: .ASCIIZ  /Insufficient memory space for data cache/
43 003262      103      141      156  INDOPN: .ASCIIZ  /Cannot open TSXIND file/
44 003312      103      141      156  UCLOPN: .ASCIIZ  /Cannot open TSXUCL data file/
45 003347      040      101      102  R50CHR: .ASCII  / ABCDEFGHIJKLMNOPQRSTUVWXYZ$. *0123456789/
46      .EVEN
      .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    004312'    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 003444    105737    000000G    TSTB     PROFLG    ; Are we running on a PRO?
31 003450    001403    BEQ         7$             ; Br if not on a PRO
32 003452    004737    000000G    CALL     PRNOP     ; Disable its interrupts
33 003456    000407    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 003460    012702    000000G    7$:    MOV     #UEXINT, R2    ; SEND UNEXPECTED INTERRUPTS TO THIS ROUTINE
43 003464    012700    000044    MOV     #44, R0    ; 120 ENCODED IN PS FIELDS
44 003470    105737    000000G    TSTB     VUXIFL    ; ARE WE TO IGNORE UNEXPECTED INTERRUPTS?
45 003474    001004    BNE         10$            ; BR IF NOT
46 003476    012702    000000G    5$:    MOV     #UEXRTN, R2    ; ROUTINE TO GO TO TO IGNORE INTERRUPT
47 003502    012700    000340    MOV     #340, R0    ; SET PRIO=7 IN PS
48 003506    012701    000120    10$:    MOV     #120, R1    ; INIT ALL VECTORS STARTING AT 120
49 003512    010221    1$:    MOV     R2, (R1)+    ; SET PC FOR INTERRUPT
50 003514    010021    MOV     R0, (R1)+    ; SET PS FOR INTERRUPT (ENCODED ADDRESS VALUE)
51 003516    105737    000000G    6$:    TSTB     VUXIFL    ; ARE WE TO IGNORE UNEXPECTED INTS?
52 003522    001411    BEQ         2$             ; BR IF YES
53 003524    105737    000000G    TSTB     PROFLG    ; IS THIS A PRO?
54 003530    001006    BNE         2$             ; BR IF YES
55 003532    005200    INC     R0    ; ADVANCE ENCODED ADDRESS
56 003534    032700    000020    BIT     #20, R0    ; DID WE CARRY INTO "T"-FIELD?
57 003540    001402    BEQ         2$             ; BR IF NOT

```

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

```

58 003542 062700 000020          ADD    #20,R0          ;FORCE CARRY OUT OF T-FIELD AND INTO PRIO FIELD
59 003546 020127 000420          2$:    CMP    R1,#420          ;DONE ALL INTERRUPT VECTORS OF INTEREST?
60 003552 103757                  BLO    1$              ;BR IF NOT
61 003554 010237 000060          MOV    R2,@#60          ;CATCH CONSOLE TERMINAL VECTOR TOO
62 003560 012737 000014 000062    MOV    #14,@#62          ;ENCODED 60
63 003566 010237 000064          MOV    R2,@#64
64 003572 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 003600 012700 000340          11$:    MOV    #340,R0          ;PRIORITY 7 PS
71 003604 012737 000000G 000000G    MOV    #CLKRTI,@#CLKVEC;Send clock interrupt to RTI instruct for now
72 003612 010037 000002G          MOV    R0,@#CLKVEC+2
73                                ;
74                                ; Take over traps, EMT, BPT, etc.
75                                ;
76 003616 005001                  CLR    R1              ;Start at location 0
77 003620 012721 000137          MOV    #137,(R1)+          ;[JMP @#JMP0] ==> 0
78 003624 012721 000000G          MOV    #JMP0,(R1)+          ;CATCH JUMPS TO LOCATION 0
79 003630 012721 000000G          MOV    #TRP4,(R1)+          ;TRAP 4
80 003634 005021                  CLR    (R1)+
81 003636 012721 000000G          MOV    #TRP10,(R1)+          ;TRAP 10
82 003642 005021                  CLR    (R1)+
83 003644 012721 000000G          MOV    #TRP14,(R1)+          ;TRAP 14 (BREAKPOINTS)
84 003650 010021                  MOV    R0,(R1)+
85 003652 012721 000000G          MOV    #TRP20,(R1)+          ;IOT TRAP
86 003656 005021                  CLR    (R1)+
87 003660 012721 000000G          MOV    #TRP24,(R1)+          ;POWER FAIL
88 003664 010021                  MOV    R0,(R1)+
89 003666 012721 000000G          MOV    #EMTENT,(R1)+          ;EMT
90 003672 005021                  CLR    (R1)+
91 003674 012721 000000G          MOV    #TRP34,(R1)+          ;TRAP
92 003700 005021                  CLR    (R1)+
93 003702 012737 000000G 000114    MOV    #MEMPAR,@#114          ;MEMORY PARITY TRAP
94 003710 010037 000116          MOV    R0,@#116
95 003714 012737 000000G 000244    MOV    #FPTRAP,@#244          ;TRAP 244 -- FLOATING POINT TRAP
96 003722 010037 000246          MOV    R0,@#246          ;Enter FPU trap at priority 7
97 003726 012737 000000G 000250    MOV    #TRP250,@#250          ;TRAP 250 -- MEMORY MANAGEMENT TRAP
98 003734 005037 000252          CLR    @#252
99                                ;
100                               ; Initialize the system mapped region.
101                               ;
102 003740 010546                  MOV    R5,-(SP)          ;SAVE THE CURRENT CONTENTS OF R5
103 003742 012705 000006          MOV    #6,R5              ;INITIALIZE TO THE FIRST REGION
104 003746 004737 000000G          CALL    MAPSYS          ;CALL THE SYSTEM MAPPING ROUTINE
105 003752 012605                  MOV    (SP)+,R5          ;RESTORE THE PREVIOUS CONTENTS OF R5
106                                ;
107                               ; Set up Unibus mapping if needed
108                               ;
109 003754 004737 004650'          CALL    SETUMP          ;SET UP UNIBUS MAPPING
110                                ;
111                               ; Initialize time-sharing lines.
112                               ;
113 003760 004737 005262'          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 003764 052737 000000G 000000G      BIS      #MMENBL,@#SROMMR; Turn on memory management
119 003772 105737 000000G              TSTB      SR3FLG      ; Does machine have memory management reg 3?
120 003776 001415                      BEQ       4$          ; Br if register does not exist (no ext. mem.)
121 004000 023727 000000G 010000      CMP       PHYMEM,#4096. ; Does machine have at least 256Kb phys memory?
122 004006 103411                      BLO       4$          ; Br if not
123 004010 052737 000000G 000000G      BIS      #EMMAP,@#SR3MMR ; Set extended memory on
124 004016 105737 000000G              TSTB      MEM256      ; Will TSX-Plus use at least 256Kb?
125 004022 001403                      BEQ       4$          ; Br if not
126 004024 052737 000000G 000000G      BIS      #IDMAP,@#SR3MMR ; Turn on 22-bit memory management for I/O
127          ;
128          ; Initialize the memory allocation table
129          ;
130 004032 013737 000000G 000000G 4$:    MOV      MAPPAR,@#KPAR5 ; Map to memory allocation table
131 004040 013702 000000G              MOV      LOMAP,R2      ; Point to 1st user-page entry
132 004044 105022                      8$:    CLRB      (R2)+    ; Say page is free
133 004046 020237 000000G              CMP      R2,HIMAP      ; Done all user pages?
134 004052 103774                      BLO      8$          ; Loop if not
135 004054 112712 000000G              MOVB     #MA$SYS,(R2) ; Set flag marking start of system pages
136          ;
137          ; Set up I/O device interrupt vectors.
138          ;
139 004060 004737 004652'              CALL     DEVVEC        ; SET UP DEVICE INTERRUPT VECTORS
140          ;
141          ; If we are running on a Professional, initialize the PI handler
142          ;
143          .IF      NE,PROASM
144 004064 105737 000000G              TSTB      PROFLG      ; Are we running on a Professional?
145 004070 001404                      BEQ      3$          ; Br if not
146 004072 004737 000000G              CALL     PROHAN        ; Initialize the PI handler
147 004076 004737 005200'              CALL     PIDVEN        ; Make device table entry for PI
148          .ENDC      ; NE,PROASM
149          ;
150          ; Initialize interrupt stack area
151          ;
152 004102 013702 000000G 3$:    MOV      INTSND,R2      ; Point to base of stack area
153 004106 012700 123456              MOV      #123456,R0      ; Get initialization value
154 004112 010022 12$:    MOV      R0,(R2)+    ; Initialize the interrupt stack area
155 004114 020237 000000G              CMP      R2,INTSTK      ; Finished?
156 004120 103774                      BLO      12$          ; Loop if not
157 004122 112737 177777 000000G      MOVB     #-1,STKLVL     ; Say we are not running on interrupt stack
158          ;
159          ; Enter TSEXEC to complete initialization
160          ;
161 004130 000137 000000G              JMP      INIJMP        ; ENTER INITIALIZATION ROUTINE IN TSEXEC
162          ;
163          ; Abort the initialization
164          ;
165 004134 013737 000042' 000000G INISTP: MOV      CLK100,@#CLKVEC ; Restore RT-11 clock vector
166 004142 013737 000044' 000000G      MOV      RTTRP4,@#4      ; Restore trap 4 vector
167 004150 013737 000046' 000000G      MOV      RTMNVC,@#RMON ; Restore RT-11 monitor pointer
168 004156          9$:    .EXIT                      ; RETURN TO RT-11

```

```

1
2
3
4
5
6
7
8
9 004160 010146
10 004162 010346
11 004164 010446
12
13
14
15 004166 016201 000004
16
17
18
19 004172 016204 000000
20 004176 072427 000005
21 004202 020461 000000G
22 004206 101402
23 004210 016104 000000G
24
25
26
27 004214 010503
28 004216 072327 000006
29 004222
30 004256 103406
31
32
33
34 004260 010561 000000G
35
36
37
38 004264 012604
39 004266 012603
40 004270 012601
41 004272 000207
42
43
44
45 004274
46 004302
47 004310
  
```

```

.SBTTL  LODINI -- Load a segment over TSINIT
-----
; LODINI is called to read an overlay segment over TSINIT.
;
; Inputs:
;   R2 = Pointer to OSTABL entry for segment to be loaded.
;   R5 = 64-byte block # where segment is to be loaded.
;
LODINI: MOV     R1, -(SP)
        MOV     R3, -(SP)
        MOV     R4, -(SP)
;
; Get pointer to linker-built overlay entry
;
        MOV     OS$OVL(R2), R1 ; Get pointer to linker-built table
;
; Determine how much code to read from the segment
;
        MOV     OS$SIZ(R2), R4 ; Get # 64-byte blks allocated for segment
        ASH     #5, R4          ; Convert to # words
        CMP     R4, 0, SIZ(R1) ; Compare with original segment code size
        BLOS    1$             ; Br if segment was truncated by init
        MOV     0, SIZ(R1), R4 ; Get code size
;
; Read the segment into memory
;
1$:      MOV     R5, R3          ; Get 64-byte block #
        ASH     #6, R3          ; Convert to byte address
        .READW  #AREA, #17, R3, R4, 0, BLK(R1)
        BCS     10$            ; Br if error on read
;
; Store the physical address of the segment into the overlay descriptor
;
        MOV     R5, 0, PAR(R1) ; Remember physical address of segment
;
; Finished
;
        MOV     (SP)+, R4
        MOV     (SP)+, R3
        MOV     (SP)+, R1
        RETURN
;
; Error on read
;
10$:     .PRINT  #TSXHD
        .PRINT  #RDERR
        .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 004312    010246
10 004314    010546
11          ;
12          ; Initialize pointer to start of memory area for overlays
13          ;
14 004316    013705    000140'      MOV     OVLBAS,R5      ;Start of area for overlays
15 004322    072527    177772      ASH     #-6,R5         ;Convert to 64-byte #
16 004326    042705    176000      BIC     #176000,R5      ;Clear possible propagated sign bits
17          ;
18          ; Begin loop to load each overlay that goes over TSINIT
19          ;
20 004332    012702    000516'      MOV     #OSTABL,R2      ;Point to 1st overlay segment entry
21 004336    005762    000002      1$:    TST     OS$FLG(R2)    ;Does this segment go over TSINIT?
22 004342    001406                        BEQ     2$         ;Br if not
23 004344    004737    004522'      CALL    KEYSEG          ;Remember base of some segments
24 004350    004737    004160'      CALL    LODINI          ;Load the segment
25 004354    066205    000000      ADD     OS$SIZ(R2),R5     ;Advance memory pointer
26 004360    062702    000006      2$:    ADD     #OS$SZ,R2    ;Point to entry for next segment
27 004364    020237    000744'      CMP     R2,OSLAST      ;Finished all segments?
28 004370    103762                        BLO     1$         ;Loop if not
29          ;
30          ; Initialize entry point vector for TSTIOX segment
31          ;
32 004372    013702    000000G      MOV     TIOBAS,R2      ;Get addr of base of TSTIOX
33 004376    072227    000006      ASH     #6,R2          ;Convert to byte address
34 004402    012705    000000G      MOV     #TIOVEC,R5     ;Point to entry point vector
35 004406    004737    004464'      CALL    ENTVEC        ;Set up entry point vector
36          ;
37          ; Initialize entry point vector for TSCASH segment
38          ;
39 004412    013702    000000G      MOV     CSHBAS,R2      ;Get addr of base of TSCASH
40 004416    001406                        BEQ     3$         ;Br if TSCASH not loaded
41 004420    072227    000006      ASH     #6,R2          ;Convert to byte address
42 004424    012705    000000G      MOV     #CSHVEC,R5     ;Point to entry point vector
43 004430    004737    004464'      CALL    ENTVEC        ;Set up entry point vector
44          ;
45          ; Initialize entry point vector for TSLOCK segment
46          ;
47 004434    013702    000000G      3$:    MOV     LOKBAS,R2    ;Get addr of base of TSLOCK
48 004440    001406                        BEQ     9$         ;Br if TSLOCK not loaded
49 004442    072227    000006      ASH     #6,R2          ;Convert to byte address
50 004446    012705    000000G      MOV     #LOKVEC,R5     ;Point to entry point vector
51 004452    004737    004464'      CALL    ENTVEC        ;Set up entry point vector
52          ;
53          ; Finished
54          ;
55 004456    012605      9$:    MOV     (SP)+,R5
56 004460    012602      MOV     (SP)+,R2
57 004462    000207      RETURN

```

```

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 004464 010246 ENTVEC: MOV     R2,-(SP)
12 004466 010446      MOV     R4,-(SP)
13 004470 010546      MOV     R5,-(SP)
14 004472 010204      MOV     R2,R4          ;Get addr of base of segment
15 004474 062704 000004 ADD     #4,R4        ;Point to start of vector in segment
16 004500 005722      TST     (R2)+         ;Get value to use to relocate offsets
17 004502 012415 1$: MOV     (R4)+,(R5)    ;Get offset to entry point within segment
18 004504 060225      ADD     R2,(R5)+     ;Convert to absolute address
19 004506 005715      TST     (R5)         ;Any more words to initialize?
20 004510 001774      BEQ     1$           ;Br if yes
21          ;
22          ; Finished
23          ;
24 004512 012605      MOV     (SP)+,R5
25 004514 012604      MOV     (SP)+,R4
26 004516 012602      MOV     (SP)+,R2
27 004520 000207      RETURN
  
```

KEYSEG -- Remember memory position of system overlays

```

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         004522  010446  KEYSEG: MOV      R4, -(SP)
11         ;
12         ;   Get the name of the segment out of the linker-built segment block
13         ;
14         004524  016200  000004      MOV      OS$OVL(R2), R0      ; Point to linker-built entry
15         004530  016004  000000G      MOV      0. ADR(R0), R4      ; Get the name of the segment
16         ;
17         ;   See if this is a segment whose address we want to remember
18         ;
19         004534  020427  052077      CMP      R4, #R5MSG          ; Is this the TSMSG segment?
20         004540  001003      BNE      1$                          ; Br if not
21         004542  010537  000000G      MOV      R5, MSGBAS          ; Remember base of TSMSG segment
22         004546  000436      BR      8$
23         004550  020427  110466      1$:  CMP      R4, #R5WIN          ; Is this the TSWIN segment?
24         004554  001003      BNE      3$                          ; Br if not
25         004556  010537  000000G      MOV      R5, WINBAS          ; Remember base of TSWIN segment
26         004562  000430      BR      8$
27         004564  020427  103112      3$:  CMP      R4, #R5USR          ; Is this the TSUSR segment?
28         004570  001003      BNE      4$                          ; Br if not
29         004572  010537  000000G      MOV      R5, USRBAS          ; Remember base of TSUSR segment
30         004576  000422      BR      8$
31         004600  020427  046543      4$:  CMP      R4, #R5LOK          ; Is this the TSLOCK segment?
32         004604  001003      BNE      5$                          ; Br if not
33         004606  010537  000000G      MOV      R5, LOKBAS          ; Remember base of TSLOCK segment
34         004612  000414      BR      8$
35         004614  020427  012700      5$:  CMP      R4, #R5CSH          ; Is this the TSCASH segment?
36         004620  001003      BNE      6$                          ; Br if not
37         004622  010537  000000G      MOV      R5, CSHBAS          ; Remember base of TSCASH segment
38         004626  000406      BR      8$
39         004630  020427  077167      6$:  CMP      R4, #R5OTIO          ; Is this the TSTIOX segment?
40         004634  001003      BNE      8$                          ; Br if not
41         004636  010537  000000G      MOV      R5, TIOBAS          ; Remember base of module
42         004642  000400      BR      8$
43         ;
44         ;   Finished
45         ;
46         004644  012604      8$:  MOV      (SP)+, R4
47         004646  000207      RETURN

```

```

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     SETUMP: MOV      R2,-(SP)
15             MOV      R3,-(SP)
16             MOV      @#4,-(SP)      ; SAVE TRAP VECTOR
17             MOV      #9$,@#4      ; CATCH TRAPS
18     ;
19     ; See if this is a type of maching that needs unibus mapping
20     ;
21             TSTB     UBUSMP      ; Is UNIBUS mapping needed?
22             BEQ      9$          ; 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     2$:      MOV      #UMRADR,R5      ; POINT TO CONTROL REGISTER FOR UNIBUS MAP 0
28             CLR      R4              ; START MAPPING TO BOTTOM OF MEMORY
29             MOV      #5,R0           ; LOAD 5 MAP REGISTERS
30     1$:      MOV      R4,(R5)+        ; SET LOW-ORDER VALUE IN MAP REGISTER
31             CLR      (R5)+          ; CLEAR HIGH-ORDER VALUE IN MAP REGISTER
32             ADD      #8192.,R4       ; ADVANCE MEMORY ADDRESS
33             SOB      R0,1$          ; LOOP IF MORE MAP REGISTERS TO LOAD
34     ;
35     ; Turn on Unibus mapping
36     ;
37             BIS      #IOMAP,@#SR3MMR ; ENABLE UNIBUS MAPPING
38     ;
39     ; Finished
40     ;
41     9$:      MOV      (SP)+,@#4      ; RESTORE TRAP VECTOR
42             MOV      (SP)+,R5
43             MOV      (SP)+,R4
44             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>
  
```

004650 000207

DEVVEC -- Set up device vectors

```

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 004652    010146
7 004654    010346
8 004656    010546
9 004660    013746    000000G
10
11          ;
12          ; Begin loop to set up vectors for each device
13          ;
14          ;      MOV      #2,R1          ;Get index # of 1st device after TT
15          1$:      MOV      HANENT(R1),R3 ;Get handler entry point address
16          ;      CMP      R3,#6          ;Is this a real device?
17          ;      BLOS     6$            ;Br if not
18          ;
19          ; See if we need to map PAR 5 to this handler
20          ;
21          ;      MOV      HANPAR(R1),R0 ;Get PAR 5 base for this handler
22          ;      BEQ      2$            ;Br if this is not a mapped handler
23          ;      MOV      R0,@#KPAR5    ;Map PAR 5 to the handler
24          ;
25          ; Clear CQE and LQE in handler header
26          ;
27          2$:      CLR      (R3)+        ;Clear LQE (4th word in handler)
28          ;      CLR      (R3)          ;Clear CQE (5th word in handler)
29          ;      SUB      #10,R3        ;Point to 1st word of handler
30          ;
31          ; Set up interrupt vectors for this handler
32          ;
33          ;      CLR      R5            ;Assume vector base address is 0
34          ;      TST      (R3)          ;Any vectors to set up?
35          ;      BEQ      6$            ;Br if no vectors to set up
36          ;      BLT      5$            ;Br if multiple-vector
37          ;      CALL     SETVEC        ;Set up the vector
38          ;      BR       6$            ;Finished
39          ;
40          ; Multiple vectors.
41          ;
42          5$:      MOV      (R3)+,R0      ;Get offset to list of vector info
43          ;      ASL      R0            ;Get byte offset to vector list
44          ;      ADD      R0,R3          ;Get absolute address of vector table
45          ;      TST      (R3)          ;Is this a PRO device with floating vectors?
46          ;      BGE      7$            ;Br if not
47          ;      TST      (R3)+        ;Point to word with device ID
48          ;      MOV      (R3)+,-(SP)    ;Get device ID
49          ;      CALL     @RPRVEC        ;Get base vector location for device
50          ;      MOV      (SP)+,R5      ;This is base of vector locations
51          ;      CALL     SETVEC        ;Set up the vector
52          ;      TST      (R3)          ;Any more vectors to set up?
53          ;      BGT      7$            ;Br if yes
54          ;
55          ; See if there are more devices to set up.
56          ;
57          6$:      ADD      #2,R1          ;Advance device table index
58          ;      CMP      R1,NUMDEV      ;More to do?

```

```
58 005006 101730          BLOS      1$          ; Br if yes
59
60          ; Finished
61          ;
62 005010 012637 000000G      MOV      (SP)+, @#KPAR5
63 005014 012605      MOV      (SP)+, R5
64 005016 012603      MOV      (SP)+, R3
65 005020 012601      MOV      (SP)+, R1
66 005022 000207      RETURN
```

SETVEC -- Set up an interrupt vector for a device

```

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         ; SETVEC: MOV R4, -(SP)
22         ;
23         ; See if this is a mapped handler
24         ;
25         ;   TST HANPAR(R1) ; Is this a mapped handler
26         ;   BNE 1$ ; Br if yes
27         ;
28         ; This is an unmapped handler.
29         ; Vector interrupts directly to the handler.
30         ;
31         ;   MOV (R3)+, R0 ; Get address of interrupt vector
32         ;   ADD R5, R0 ; Add base address to vector location
33         ;   MOV R3, (R0) ; Store address of cell in handler
34         ;   ADD (R3)+, (R0)+ ; Add offset to interrupt entry point
35         ;   MOV (R3)+, (R0) ; Set PS for interrupt
36         ;   BIS #340, (R0) ; Make sure priority = 7
37         ;   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         ; 1$: MOV XMVBAS, R4 ; Point to area where we store interrupt rtn
48         ;   MOV (R3)+, R0 ; Get address of interrupt vector
49         ;   ADD R5, R0 ; Add base address to interrupt location
50         ;   MOV R4, (R0)+ ; Direct interrupt to our routine
51         ;   MOV #013746, (R4)+ ; [ MOV @#KPAR5, -(SP) ]
52         ;   MOV #KPAR5, (R4)+
53         ;   MOV #012737, (R4)+ ; [ MOV #par5val, @#KPAR5 ]
54         ;   MOV HANPAR(R1), (R4)+
55         ;   MOV #KPAR5, (R4)+
56         ;   MOV #012746, (R4)+ ; [ MOV #340, -(SP) ]
57         ;   MOV #340, (R4)+

```

SETVEC -- Set up an interrupt vector for a device

```

58 005122 012724 012746          MOV    #012746,(R4)+    ; [ MOV #HANXIT,-(SP) ]
59 005126 012724 000000G        MOV    #HANXIT,(R4)+
60 005132 012724 000257          MOV    #000257,(R4)+    ; [ CCC - Clear all condition codes ]
61 005136 016314 000002          MOV    2(R3),(R4)       ; [ SEx - Set condition codes specified in PS]
62 005142 042714 177760          BIC    #^C17,(R4)
63 005146 052724 000260          BIS    #260,(R4)+
64 005152 012724 000137          MOV    #000137,(R4)+    ; [ JMP @#handler_entry ]
65 005156 010314                MOV    R3,(R4)          ; Store address of int entry point
66 005160 062324                ADD    (R3)+,(R4)+
67 005162 012310                MOV    (R3)+,(R0)       ; Set PS for interrupt entry
68 005164 052710 000340          BIS    #340,(R0)       ; Make sure priority = 7
69                               ;
70                               ; Save address beyond end of compiled interrupt catcher routine
71                               ;
72 005170 010437 000122'        MOV    R4,XMVBAS        ; Save address beyond end of routine
73                               ;
74                               ; Finished
75                               ;
76 005174 012604                9$:  MOV    (SP)+,R4
77 005176 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 005200 010146  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 005202 062737 000002 000000G      ADD      #2, NUMDEV      ;One more device
13 005210 013701 000000G      MOV      NUMDEV, R1      ;Get device table index
14     ;
15     ; Set up information about the PI device
16     ;
17 005214 012761 062550 000000G      MOV      #R50PI, PNAME(R1) ;Set device name
18 005222 012761 000000C 000000G      MOV      #<DS$SFN!DI$PI>, DVSTAT(R1) ;Set device status flags
19 005230 005061 000000G      CLR      DVFLAG(R1)      ;Clear other flags
20 005234 005061 000000G      CLR      DEVSIZ(R1)      ;Clear device size
21 005240 012761 000006G 000000G      MOV      #PIHAN+6, HANENT(R1);Set handler entry point (4th word)
22 005246 012761 000000G 000000G      MOV      #PROSIZ, HANSIZ(R1) ;Set handler size
23     ;
24     ; Finished
25     ;
26 005254 012601      MOV      (SP)+, R1
27 005256 000207      RETURN
28     .ENDC      ; NE, PROASM
  
```

```

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          LININI: MOV      R1,-(SP)
9                  MOV      R2,-(SP)
10                 MOV      R3,-(SP)
11                 MOV      R4,-(SP)
12                 MOV      R5,-(SP)
13          ;
14          ; Set up interrupt vectors for DL11 lines
15          ;
16                 MOV      #2,R1      ; Index for 1st line
17                 MOV      #340,R4    ; Priority 7 PS
18 1$:         BIT      #$DEAD,LSW3(R1) ; Is this line uninstalled?
19                 BNE      8$          ; Br if yes
20                 BIT      #$HARD,LSW3(R1) ; Is this line connected to hardware?
21                 BEQ      8$          ; Br if not
22                 CMP      LCDTYP(R1),#CDX$DL ; Is this a DL11 line?
23                 BNE      8$          ; Br if not
24          ;
25          ; DL-11 line
26          ;
27                 MOV      INVEC(R1),R5 ; GET ADDRESS OF INPUT VECTOR
28                 MOV      #INRECV,R2   ; END OF RECEIVING VECTOR
29                 MOV      #<OTRECV-6>,R3 ; START OF INPUT INTERRUPT ENTRY POINTS
30                 MOV      R1,R0        ; GET LINE NUMBER
31                 ASL      R0           ; 4 BYTES PER INPUT INTERRUPT ENTRY POINT
32                 SUB      R0,R2        ; GET ADDRESS OF INPUT INTERRUPT ENTRY POINT
33                 ADD      R1,R0        ; 6 BYTES PER OUTPUT INTERRUPT ENTRY POINT
34                 ADD      R0,R3        ; GET ADDRESS OF OUTPUT INTERRUPT ENTRY POINT
35                 MOV      R2,(R5)+     ; SET PC FOR INPUT INTERRUPT ENTRY POINT
36                 MOV      R4,(R5)+     ; SET PS FOR INPUT INTERRUPT
37                 MOV      R3,(R5)+     ; SET PC FOR OUTPUT INTERRUPT
38                 MOV      R4,(R5)+     ; SET PS FOR OUTPUT INTERRUPT
39          ;
40          ; Try next line
41          ;
42 8$:         ADD      #2,R1            ; ADVANCE LINE INDEX NUMBER
43                 CMP      R1,#LSTHL    ; MORE TO DO?
44                 BLOS     1$           ; BR IF YES
45          ;
46          ; Initialize multiplexers.
47          ;
48  SETMUX: MOV      #LSTMX,R1          ; Get last mux index #
49                 BEQ      SETLIN      ; Br if there are no mux lines
50 3$:         CMP      MXTYPE(R1),#CDX$DZ ; Is this a DZ11, DH11, or DHV11?
51                 BEQ      1$           ; Br if DZ11
52                 CMP      MXTYPE(R1),#CDX$VH ; Is this a DHV11?
53                 BNE      2$           ; Br if not
54                 CALL     VHINIT      ; Initialize a DHV11
55                 BR       4$
56 2$:         CALL     DHINIT          ; Initialize a DH11
57                 BR       4$

```

```

58          1$:      CALL    DZINIT          ; Initialize a DZ11
59          4$:      SUB     #2,R1          ; More to enable?
60                      BNE     3$           ; Br if yes
61          ;
62          ; Enable all lines
63          ;
64          SETLIN:  MOV     #LSTHL,R1      ; INDEX # OF LAST REAL LINE
65          4$:      BIT     #$DEAD,LSW3(R1) ; IS THIS LINE INSTALLED?
66                      BNE     2$           ; BR IF NOT
67                      BIT     #$HARD,LSW3(R1) ; Is this line connected to hardware?
68                      BEQ     2$           ; Br if not
69                      BIT     #$PHONE,ILSW2(R1) ; IS THIS A DIAL-UP LINE?
70                      BEQ     3$           ; BR IF NOT
71                      BIS     #$NOIN,LSW3(R1) ; IGNORE INPUT TILL DIAL UP OCCURS
72          3$:      MOV     LCDTYP(R1),R5   ; Get comm device type code
73                      MOV     LMXNUM(R1),R0 ; IS THIS A DL-11 OR MUX LINE?
74                      BEQ     1$           ; BR IF DL-11
75                      CMP     R5,#CDX$DZ   ; Is this a DZ11 or DH11?
76                      BEQ     6$           ; Br if DZ11
77                      CMP     R5,#CDX$VH   ; Is this a DH11 or DHV11?
78                      BEQ     7$           ; Br if DHV11
79          ;
80          ; DH11 line
81          ;
82                      CALL    DHLPRM      ; Set line parameters for DH11 line
83                      BR      2$
84          ;
85          ; DHV11 line
86          ;
87          7$:      CALL    VHLPRM      ; Set line parameters for DHV11 line
88                      BR      2$
89          ;
90          ; DZ-11 line
91          ;
92          6$:      MOV     LMXLN(R1),R2   ; Get line # within mux group
93                      BIS     #017030,R2 ; Set line enable flags
94                      MOV     R2,@MXLPR(R0) ; Enable the line
95                      BR      2$
96          ;
97          ; DL-11 line
98          ;
99          1$:      MOV     TSR(R1),R2     ; ADDRESS OF TRANSMITTER STATUS REGISTER
100                      MOV     (R2),R3     ; CLEAR TRANSMITTER STATUS REGISTER
101                      CLR     (R2)
102                      MOV     RBR(R1),R2   ; ADDRESS OF RECEIVER BUFFER REGISTER
103                      MOV     (R2),R3     ; CLEAR RECEIVER BUFFER REGISTER
104                      MOV     RSR(R1),R2   ; ADDRESS OF RECEIVER STATUS REGISTER
105                      CLR     (R2)
106                      MOV     #RDINT,(R2) ; ENABLE RECEIVER INTERRUPTS
107          ;
108          ; Do next line
109          ;
110          2$:      SUB     #2,R1
111                      BGT     4$
112          ;
113          ; Finished
114          ;

```

TSINIT -- TSX startup initializ MACRO V05.05 Friday 20-Jan-89 11:30 Page 17-2
PIDVEN -- Make device table entry for PI device

115	MOV	(SP)+, R5
116	MOV	(SP)+, R4
117	MOV	(SP)+, R3
118	MOV	(SP)+, R2
119	MOV	(SP)+, R1
120	RETURN	

```
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          DHLPRM:
9          ;
10         ; Enable DM11 for this line
11         ;
12         MOV     LMXNUM(R1),R0    ;Get mux index number
13         TST     DM$CSR(R0)       ;Does this DH11 have DM11 modem control?
14         BEQ     2$              ;Br if not
15         BICB    #MF$LIN,@DM$CSR(R0) ;Clear line # field in DM11 CSR
16         BISB    LMXLN(R1),@DM$CSR(R0);Select line of interest
17         MOV     #MF$LE,@DM$LSR(R0);Enable the line
18         ;
19         ; Finished
20         ;
21         2$:     RETURN
```

PIDVEN -- Make device table entry for PI device

```
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          VHLPRM:
9          ;
10         ; Enable the line
11         ;
12         MOV     LMXNUM(R1),R0    ;Get mux index number
13         BIC     #VF$LIN,@VH$CSR(R0) ;Clear line # field in mux CSR
14         BISB    LMXLN(R1),@VH$CSR(R0) ;Set our line #
15         MOV     #<VF$RE>,@VH$LCR(R0) ;Enable the line
16         ;
17         ; Finished
18         ;
19         RETURN
```

```

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          DZINIT:
9          ;
10         ; See if this DZ11 is installed
11         ;
12         TST      MXCSR(R1)      ; Is this DZ-11 installed?
13         BEQ      4$             ; Br if not
14         ;
15         ; Set up interrupt vector connections for this MUX
16         ;
17         CALL     MUXVEC         ; Set up interrupt vectors for this DZ11
18         ;
19         ; Start up the mux operation
20         ;
21         BIS      #ZCLR,@MXCSR(R1); Do master clear on DZ-11
22         1$:      BIT      #ZCLR,@MXCSR(R1); Wait for clear to finish
23         BNE      1$
24         2$:      MOV      @MXRBUF(R1),R0 ; Clear silo
25         BMI      2$
26         CLRB     @MXDTR(R1)     ; Disable all data sets
27         ;
28         ; Finished
29         ;
30         4$:      RETURN
  
```

PIDVEN -- Make device table entry for PI device

```

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          MUXVEC: MOV     R2, -(SP)
10             MOV     R3, -(SP)
11             MOV     R5, -(SP)
12          ;
13          ; Set interrupt vector for mux
14          ;
15             MOV     MXVEC(R1), R5      ; Get address of input interrupt vector
16             MOV     #INMXV, R2         ; End of receiving vector
17             MOV     #<OTMXV-6>, R3     ; Output interrupt table
18             MOV     R1, R0             ; Get mux index number
19             ASL     R0                  ; 4 bytes per line in input int table
20             SUB     R0, R2             ; Get address of input entry point
21             ADD     R1, R0             ; 6 bytes per mux in output entry point table
22             ADD     R0, R3             ; Get address of output int entry point
23             MOV     R2, (R5)+          ; Set PC for input interrupt
24             MOV     #340, (R5)+        ; Set PS for output interrupt
25             MOV     R3, (R5)+          ; Set PC for output interrupt
26             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             MOV     #004537, (R3)+      ; JSR R5, @#x
36             MOV     #DZOINT, R0         ; Assume this is a DZ11
37             CMP     MXTYPE(R1), #CDX$DZ ; Is this a DZ11?
38             BEQ     1$                  ; Br if yes
39             MOV     #VHOINT, R0         ; Assume this is a DHV11
40             CMP     MXTYPE(R1), #CDX$VH ; Is this a DHV11?
41             BEQ     1$                  ; Br if yes
42             MOV     #DHOINT, R0         ; Get interrupt routine for DH11's
43          1$: MOV     R0, (R3)+          ; Store address of interrupt routine
44             MOV     R1, (R3)           ; Store mux index number
45          ;
46          ; Finished
47          ;
48          9$: MOV     (SP)+, R5
49             MOV     (SP)+, R3
50             MOV     (SP)+, R2
51             RETURN

```

```

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          DHINIT:
9          ;
10         ; See if this DH11 is installed
11         ;
12         TST     MH$SCR(R1)      ; Is this DH11 installed?
13         BEQ     9$              ; Br if not
14         ;
15         ; Connect interrupt vector to DH11
16         ;
17         CALL    MUXVEC          ; Set up interrupt vectors for DH11
18         ;
19         ; Clear the multiplexer
20         ;
21         MOV     #HF$MC,@MH$SCR(R1) ; Set the master-clear flag
22         1$:     BIT     #HF$MC,@MH$SCR(R1) ; Wait for the master clear to be completed
23         BNE     1$
24         ;
25         ; Clear the DM11 scanner
26         ;
27         MOV     DM$CSR(R1),R0    ; Is there an associated DM11?
28         BEQ     3$              ; Br if not
29         MOV     #MF$CS,(R0)     ; Clear the scanner
30         BIS     #MF$CM,(R0)     ; Clear the multiplexer
31         3$:
32         ;
33         ; Finished
34         ;
35         9$:
36         RETURN

```

```

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          VHINIT:
9          ;
10         ; See if this DHV11 is installed
11         ;
12         TST     VH$CSR(R1)      ; Is this DHV11 installed?
13         BEQ     9$              ; Br if not
14         ;
15         ; Connect interrupt vector to DHV11
16         ;
17         CALL    MUXVEC          ; Set up interrupt vectors
18         ;
19         ; Clear the multiplexer
20         ;
21         MOV     #VF$MR,@VH$CSR(R1) ; Reset the multiplexer
22         1$:     BIT     #VF$MR,@VH$CSR(R1) ; Wait for reset to finish
23         BNE     1$
24         ;
25         ; Clean out the FIFO buffer in the mux
26         ;
27         2$:     MOV     @MXRBUF(R1),R0 ; Get contents of receiver buffer register
28         BLT     2$              ; Loop until RBUF empty
29         ;
30         ; Finished
31         ;
32         9$:     RETURN
  
```

```

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 005260    LINCHK:
10 005260    DHLPRM:
11 005260    VHLPRM:
12 005260    DZINIT:
13 005260    MUXVEC:
14 005260    DHINIT:
15 005260    VHINIT:
16 005260    000207    RETURN
17          ;
18          ; LININI routine for Pro systems
19          ;
20 005262    010146    LININI: MOV      R1,-(SP)
21 005264    012701    000000G    MOV      #LSTLIN,R1      ;Get # of last line
22          ;
23          ; Determine if this line is connected to hardware
24          ;
25 005270    004737    005340'    1$:      CALL      LINTYP      ;Determine the type of this line
26 005274    032761    000000G 000000G    BIT      #$HARD,LSW3(R1) ;Is this line connected to hardware?
27 005302    001411    BEQ      2$      ;Br if not
28          ;
29          ; Call Pro line initialization routine
30          ;
31 005304    004737    000000G    CALL      PROLIN      ;Initialize Pro line
32          ;
33          ; Do some special init for phone lines
34          ;
35 005310    032761    000000G 000000G    BIT      #$PHONE,ILSW2(R1);Is this a dialup line?
36 005316    001403    BEQ      2$      ;Br if not
37 005320    052761    000000G 000000G    BIS      #$NOIN,LSW3(R1) ;Ignore input till dial up occurs
38          ;
39          ; Check next line
40          ;
41 005326    162701    000002    2$:      SUB      #2,R1      ;Get index # of next line
42 005332    003356    BGT      1$      ;Loop if more lines to do
43          ;
44          ; Finished
45          ;
46 005334    012601    MOV      (SP)+,R1
47 005336    000207    RETURN
48          ;
49          ; End of Pro-only code
50          ;
51          .ENDC      ;NE,<PROASM-1>

```

LINTYP -- Determine the type of a line

```

1          .SBTTL  LINTYP -- Determine the type of a line
2          ;-----
3          ; LINTYP is called to determine if the current line is a time-sharing line
4          ; a CL line, or a non-hardware connected line.
5          ;
6          ; Inputs:
7          ; R1 = Line index number
8          ;
9 005340 020127 000000G LINTYP: CMP R1,#LSTPL ;Is this a time-sharing line?
10 005344 101422 BLOS 1$ ;Br if yes
11 005346 020127 000000G CMP R1,#FSTIOL ;Is this a CL line?
12 005352 103004 BHIS 2$ ;Br if yes
13 005354 012761 177777 000000G MOV #-1,LCLUNT(R1) ;Say line not in use as a CL line
14 005362 000432 BR 9$
15          ;
16          ; This is a CL line
17          ;
18 005364 016100 000000G 2$: MOV LCLUNT(R1),RO ;Get the CL unit index number
19 005370 010160 000000G MOV R1,CL$LIX(RO) ;Say which line is assoc with this CL unit
20 005374 012761 000000G 000000G MOV #CLOTIR,LOUTIR(R1) ;Set terminal output interrupt routine
21 005402 012761 000000G 000000G MOV #CLINCP,LINIR(R1) ;Set terminal input interrupt routine
22 005410 000414 BR 8$
23          ;
24          ; This is a time-sharing line
25          ;
26 005412 012761 177777 000000G 1$: MOV #-1,LCLUNT(R1) ;Say line is not in use as a CL unit
27 005420 012761 177777 000000G MOV #-1,LXCL(R1) ;Line is not cross-connected to CL unit
28 005426 012761 000000G 000000G MOV #NEDCHR,LOUTIR(R1) ;Set terminal output interrupt routine
29 005434 012761 000000G 000000G MOV #TTINCP,LINIR(R1) ;Set terminal input interrupt routine
30 005442 052761 000000G 000000G 8$: BIS #$HARD,LSW3(R1) ;This line is connected to hardware
31          ;
32          ; Finished
33          ;
34 005450 000207 9$: RETURN

```

* * * 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 005452    INITGD:
9          ;
10         ; Save some RT-11 pointers in case we abort the initialization
11         ;
12 005452    013737    000004    000044'    MOV     @#4,RTTRP4      ;Save trap 4 vector
13 005460    013737    000000G 000046'    MOV     @#RMON,RTMNVC    ;RT-11 monitor pointer
14 005466    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 005474    .GTIM     #AREA,#SYTIMH      ;Get the current time of day
20         ;
21         ; Trap ^C for later test so we can restore clock vector
22         ;
23 005514    .SCCA     #AREA,#CCAFLG     ;Catch control-C
24         ;
25         ; Check for TSGEN size overflow
26         ;
27 005534    012700    000000G    MOV     #GENTOP,R0      ;Get top of TSGEN
28 005540    162700    037776      SUB     #<40000-2>,R0    ;Will TSKMON have problems?
29 005544    003422      BLE     15$      ;Continue if not
30 005546    010046      MOV     R0,-(SP)    ;Save overflow size
31 005550      .PRINT    #TSXHD            ;Print error message
32 005556      .PRINT    #TOOBIG
33 005564      .PRINT    #REDUCE
34 005572    012600      MOV     (SP)+,R0    ;Recover amount of overflow
35 005574    004737    025230'    CALL    PRTDEC
36 005600      .PRINT    #BYTES
37 005606    000137    004134'    JMP     INISTP
38         ;
39         ; Initialize the system stack (below 1000)
40         ;
41 005612    012701    000000G    15$:    MOV     #SSEND,R1      ;Point to bottom of stack
42 005616    012700    123456      MOV     #123456,R0      ;Get initialization value
43 005622    010021      13$:    MOV     R0,(R1)+    ;Initialize the stack
44 005624    020127    000000G    CMP     R1,#SS      ;Reached top of the stack area?
45 005630    103774      BLO     13$      ;Loop if not
46 005632    010106      MOV     R1,SP      ;Run on system stack
47         ;
48         ; Make sure we are not already running under TSX.
49         ;
50 005634      .SERR      ;DON'T DIE ON ERRORS
51 005642    012700    000176'    MOV     #GTLIN,R0      ;TSX EMT TO GET LINE NUMBER
52 005646    104375      EMT     375      ;TRY A TSX EMT
53 005650    103410      BCS     1$      ;BR IF NOT UNDER TSX
54 005652      .PRINT    #TSXHD            ;ALREADY UNDER TSX
55 005660      .PRINT    #TSXRUN
56 005666    000137    004134'    JMP     INISTP
57 005672      1$:      .HERR      ;RENABLE FATAL ERRORS

```

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

```

58      ;
59      ;   Make sure this machine has memory management facilities.
60      ;
61      005700      .TRPSET #AREA,#NOXM      ;CATCH TRAPS
62      005720      005737      000000G      TST      @#SROMMR      ;TRY TO ACCESS MEMORY MANAGEMENT REGISTER
63      005724      .TRPSET #AREA,#0      ;Release trap control
64      ;
65      ;   Request all available memory from RT-11.
66      ;
67      005742      .SETTOP #-2      ;REQUEST ALL AVAILABLE MEMORY
68      005750      010037      000132'      MOV      R0,TOPMEM      ;REMEMBER WHERE TOP OF MEMORY IS
69      005754      020027      000000G      CMP      R0,#VPAR5      ;TSX CANNOT EXTEND ABOVE PAR5 BASE ADDRESS
70      005760      101402      BLOS      3$      ;BR IF RT-11 IS BELOW THAT
71      005762      012700      000000G      MOV      #VPAR5,R0      ;SET PAR5 BASE AS UPPER LIMIT ON TSX SIZE
72      005766      010037      000236'      3$:      MOV      R0,MEMLIM      ;TSX MAY NOT EXCEED THIS UPPER LIMIT
73      ;
74      ;   Lock USR in memory for speed
75      ;   (Set USR to swap over TSEMT to get out of the way)
76      ;
77      005772      012705      177776'      MOV      #TSINIT-2,R5      ;GET THE BASE OF TSINIT
78      005776      .GVAL      #AREA,#374      ;GET SIZE OF RT-11 USR MODULE
79      006016      160005      SUB      R0,R5      ;ALLOCATE SPACE BELOW TSINIT FOR USR
80      006020      010537      000046      MOV      R5,@#46      ;SET USR TO SWAP OVER TSEMT
81      006024      5$:      .LOCK      ;LOCK USR IN MEMORY
82      ;
83      ;   Determine if we are to run system with the system debugger
84      ;
85      006026      032737      000000G      000000G      BIT      #CHAIN,@#JSWLOC      ;WERE WE CHAINED TO?
86      006034      001406      BEQ      10$      ;BR IF NOT
87      006036      023727      000510      057164      CMP      @#510,#R500DT      ;SHOULD WE RUN UNDER ODT?
88      006044      001002      BNE      10$      ;BR IF NOT
89      006046      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      006052      10$:
95      006052      004737      025354'      .IF      NE,PROCID      ;** Do if assembling for pro only **
96      .CALL      INSCHK      ;PERFORM VERIFICATION AND DECRYPTION FOR PRO
97      .ENDC      ;NE,PROCID
98      ;
99      ;   Allocate non-initialized buffer space over TSINIT.
100     ;
101     006056      012705      000000'      MOV      #TSINIT,R5      ;Allocate buffer space over TSINIT
102     006062      004737      011652'      CALL      ALOCBF      ;Do allocation
103     006066      004737      012316'      CALL      ALCSLO      ;Allocate silo buffers for lines
104     006072      020527      005452'      CMP      R5,#INITG0      ;Are we beyond code that takes over control?
105     006076      103002      BHS      12$      ;Br if yes
106     006100      012705      005452'      MOV      #INITG0,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     006104      001274      PIINSZ      =      700.      ;Space needed for PI init code
113     006104      010537      000000G      12$:      MOV      R5,INTSND      ;Ptr to base of interrupt stack
114     006110      062705      000002      ADD      #2,R5      ;Always leave last word of stack for flag val

```

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

```

115 006114 013701 000000G      MOV      @#RMON,R1      ;Get pointer to RT-11 RMON base
116 006120 032761 000000G 000370  BIT      #CW$PRO,370(R1) ;Are we running on a PRO?
117 006126 001407          BEQ      11$              ;Br if not
118 006130 105237 000000G      INCB     PROFLG         ;Set flag saying this is a PRO-350
119 006134 010537 000150'      MOV      R5,PROBUF      ;Save pointer to buffer area
120 006140 062705 001274      ADD      #PIINSZ,R5      ;Allocate space for buffer
121 006144 000402          BR      14$
122 006146 062705 000000G      11$:  ADD      #INTSSZ,R5      ;Allocate space for interrupt stack
123 006152 010537 000000G      14$:  MOV      R5,INTSTK      ;Address of top of interrupt stack
124
125      ; Allocate space for those overlays that go over TSINIT
126
127 006156 004737 021400'      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 006162 020527 025350'      CMP      R5,#INITOP      ;Have we allocated up to top of TSINIT?
133 006166 103002          BHS      4$              ;Br if yes
134 006170 012705 025350'      MOV      #INITOP,R5      ;Advance to top of TSINIT
135
136      ; Allocate a 2048. byte work buffer
137
138 006174 004737 007526'      4$:  CALL     ALCWRK          ;Allocate work buffer
139
140      ; Allocate empty Region Control Blocks for use by handlers
141
142 006200 004737 007562'      CALL     ALCHRB
143
144      ; If we were started in debug mode, load ODT.
145
146      .IF      EQ,PROCID      ;Don't allow ODT for production PRO version
147      TST      ODTFLG         ;Are we to load system debugger?
148      BEQ      2$              ;Br if not
149      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 006204 004737 015230'      2$:  CALL     MEMINI          ;Initialize memory management
156
157      ; Extract information from RT-11 configuration and sysgen words.
158
159 006210 013701 000000G      MOV      @#RMON,R1      ;GET POINTER TO BASE OF RMON
160 006214 016102 000300      MOV      300(R1),R2      ;GET RT-11 CONFIGURATION WORD
161 006220 042702 000000C      BIC      #CW$GDH+CW$BTH+CW$LGS,R2 ;RESET A FEW FLAGS
162 006224 052702 000000C      BIS      #CW$FB+CW$FGJ+CW$USR+CW$XM,R2 ;SET A FEW FLAGS
163 006230 010237 000000G      MOV      R2,CONFIG      ;INITIALIZE OUR CONFIGURATION WORD
164
165 006234 016137 000370 000000G      ; Now get extended configuration word.
166 006242 052737 000000G 000000G      MOV      370(R1),CONFIG2 ;EXTENDED CONFIGURATION WORD
167 006250 123727 000000G 000000G      BIS      #CW$ESP,CONFIG2 ;SET EXIT NO SWAP FLAG
168 006256 001003          CMPB     VBUSTP,#QBUS      ;Is this a Q-bus machine?
169 006260 052737 000000G 000000G      BNE      25$              ;Br if not
170
171 006266 016102 000372      BIS      #CW$QBS,CONFIG2 ;Set QBUS flag
      ; And sysgen option word.
25$:  MOV      372(R1),R2

```

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

```

172 006272 042702 000000C      BIC      #SG$ELG+SG$PAR+SG$MTS,R2
173 006276 052702 000000C      BIS      #SG$MMU+SG$MTM+SG$IOT+SG$TSX,R2
174 006302 010237 000000G      MOV      R2,SYSGEN      ; INITIALIZE OUR SYSGEN WORD
175
176      ; If a system version number was specified, use it.
177      ; Else, get version number from RT-11, but limit to default.
178
179 006306 013700 000000G      MOV      SYSVER,R0      ; Has user specified version to emulate?
180 006312 001015      BNE      30$      ; If so, keep SYSVER
181 006314 012737 000314' 000156'  MOV      #RTVDEF,RTVPTR ; If using RT version, set cutoff
182 006322      .GVAL      #AREA,#276 ; GET RT-11 SYSTEM VERSION NUMBER
183 006342 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      ; R0 contains version # in low byte, update # in high byte
187
188 006346 012702 000264' 30$:      MOV      #RTVER,R2      ; Get ptr to first entry in table
189 006352 120062 000000      31$:      CMPB      R0,RT$VER(R2) ; Does main version match?
190 006356 001005      BNE      32$      ; Br if not
191 006360 000300      SWAB      R0      ; Main version matches, get update to low byte
192 006362 120062 000001      CMPB      R0,RT$UPD(R2) ; Does the update match also?
193 006366 001425      BEQ      34$      ; Its a match! Use this entry
194 006370 000300      SWAB      R0      ; Get SYSVER back to low byte
195 006372 062702 000003 32$:      ADD      #RTV$SZ,R2 ; If not, step up to the next entry
196 006376 020227 000322'      CMP      R2,#RTVEND ; Past end of table?
197 006402 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 006404 023727 000156' 000322'      CMP      RTVPTR,#RTVEND ; Was a limit specified? (Got ver. from RT?)
204 006412 103404      BLO      33$      ; Br if so
205      ; Unknown user-specified version, keep user-specified SYSVER,
206      ; but use ptr to last known version
207 006414 012737 000317' 000156'      MOV      #<RTVEND-RTV$SZ>,RTVPTR ; Use latest known defaults
208 006422 000414      BR      36$
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 006424 113737 000314' 000000G 33$:      MOVB      RTVDEF+RT$VER,SYSVER ; Set SYSVER to default
212 006432 113737 000315' 000000G      MOVB      RTVDEF+RT$UPD,SYSUPD ; and update
213 006440 000405      BR      36$
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 006442 020237 000156' 34$:      CMP      R2,RTVPTR      ; Is it past limit?
219 006446 103366      BHIS      33$      ; If so, keep limit, and go limit SYSVER
220 006450 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 006454 105737 000000G 36$:      TSTB      CLVERS      ; Use table value for CL version number?
225 006460 001005      BNE      38$      ; Br if not, use value supplied in TSGEN
226 006462 013702 000156'      MOV      RTVPTR,R2      ; Get ptr to version table
227 006466 116237 000002 000000G      MOVB      CL$VER(R2),CLVERS ; Auto set CLVERS from version table
228 006474 105737 000000G 38$:      TSTB      LDVERS      ; Auto-select LD translation table type?

```

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

229	006500	001011		BNE	39\$	;Br if not, use value supplied in TSGEN
230	006502	112737	000001 000000G	MOVB	#1,LDVERS	;Assume old translation table format
231	006510	023727	000156' 000314'	CMP	RTVPTR,#RT54	;At or beyond 5.4?
232	006516	103402		BLO	39\$	;Br if not, retain old format
233	006520	105237	000000G	INCB	LDVERS	;LD translation table format changed at 5.4
234	006524		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 006524 032737 000000G 000000G INICLK: BIT      #CW$50H, CONFIG ; 50 or 60 Hz clock?
6 006532 001017                BNE      2$      ; Br if 50 Hz
7
8      ; 60 Hz clock
9
10 006534 012737 000074 000000G      MOV      #60., TK1SEC      ; Clock ticks per 1 second
11 006542 012737 000036 000000G      MOV      #30., TK5VAL      ; Clock ticks per 0.5 seconds
12 006550 012737 000264 000000G      MOV      #180., TK3SVL      ; Clock ticks per 3 seconds
13 006556 012737 000006 000000G      MOV      #6., TK1VAL      ; Clock ticks per 0.1 seconds
14 006564 012700 001130                MOV      #600., R0      ; Get # clock ticks per 10 seconds
15 006570 000416                BR       8$
16
17      ; 50 Hz clock
18
19 006572 012737 000062 000000G 2$:      MOV      #50., TK1SEC      ; Clock ticks per 1 second
20 006600 012737 000031 000000G      MOV      #25., TK5VAL      ; Clock ticks per 0.5 seconds
21 006606 012737 000226 000000G      MOV      #150., TK3SVL      ; Clock ticks per 3 seconds
22 006614 012737 000005 000000G      MOV      #5., TK1VAL      ; Clock ticks per 0.1 seconds
23 006622 012700 000764                MOV      #500., R0      ; Get # clock ticks per 10 seconds
24
25      ; Set number of clock ticks per day
26
27 006626 012702 020700      8$:      MOV      #8640., R2      ; (# seconds per day) / 10.
28 006632 070200                MUL      R0, R2      ; Get # clock ticks per day
29 006634 010237 000000G      MOV      R2, DATIMH      ; High-order value
30 006640 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 006644 004737 005260'      CKLIN:  CALL      LINCHK      ; CHECK T/S LINE ADDRESSES
35
36      ; Do PRO-350 system initialization
37
38      . IF      NE, PROASM
39 006650 032737 000000G 000000G      BIT      #CW$PRO, CONFIG2 ; Are we running on a PRO-350?
40 006656 001405                BEQ      INIDEV      ; Br if not
41 006660 004737 000000G      CALL      PROINI      ; Do PRO-350 initialization
42 006664 012737 000000G 000000G      MOV      #PIDRIV, PIDPTR ; Set up pointer to clock driven PI routine
43      . ENDC      ; NE, PROASM
44
45      ; Make entry in device handler table for TT device.
46
47 006672 012737 100040 000000G INIDEV: MOV      #R50TT, PNAME      ; PERMANENT NAME "TT"
48 006700 012737 000000G 000000G      MOV      #DI$TT, DVSTAT      ; SET DEVICE STATUS FLAGS FOR TT
49 006706 005037 000000G      CLR      DVFLAG
50 006712 005037 000000G      CLR      DEVSIZ
51 006716 012737 000002 000000G      MOV      #2, HANENT      ; SET UP HANENT SO HANDLER LOOKS RESIDENT
52 006724 005037 000000G      CLR      NUMDEV      ; IT IS DEVICE # 0
53
54      ; Make device table entry for LD (logical disk) device
55
56 006730 012737 177777 000000G      MOV      #-1, LDDEVX      ; ASSUME LD SUPPORT NOT WANTED
57 006736 105737 000000G      TSTB      VLDSYS      ; IS LD SUPPORT GENNED IN?

```

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

```

58 006742 001425          BEQ      6$          ;BR IF NOT
59 006744 062737 000002 000000G    ADD      #2,NUMDEV      ;ONE MORE DEVICE
60 006752 013701 000000G    MOV      NUMDEV,R1      ;GET DEVICE TABLE INDEX
61 006756 010137 000000G    MOV      R1,LDDEVX      ;REMEMBER INDEX NUMBER FOR LD DEVICE
62 006762 012761 045640 000000G    MOV      #R5OLD,PNAME(R1) ;SET DEVICE NAME ("LD")
63 006770 012761 000000C 000000G    MOV      #<DS$DIR+DS$SFN+DS$VSZ+DI$LD>,DVSTAT(R1);SET DEV STATUS FLAGS
64 006776 012761 000000G 000000G    MOV      #DX$EBA,DVFLAG(R1);Say buffers must be on even byte boundaries
65 007004 005061 000000G    CLR      DEVSIZ(R1)
66 007010 012761 000002 000000G    MOV      #2,HANENT(R1) ;SAY HANDLER IS RESIDENT
67
68          ; Make device table entry for CL (communications line) device
69
70 007016 005727 000000G    6$:      TST      #CLTOTL      ;Are there any communications lines?
71 007022 001402          BEQ      8$          ;Br if not
72 007024 004737 013444'    CALL      CLINIT      ;Initialize CL handler
73
74          ; Disable clock interrupts.
75
76 007030 012737 000002 000000 8$:    MOV      #2,@#0      ;LOAD RTI IN LOCATION 0
77 007036 005037 000000G    CLR      @#CLKVEC      ;ATTACH CLOCK INTERRUPT TO 0
78 007042 032737 000000G 000000G    BIT      #CW$PRO,CONF02 ;ARE WE RUNNING ON A PRO?
79 007050 001402          BEQ      1$          ;BR IF NOT
80 007052 005037 000230          CLR      @#230      ;380 CLOCK INTERRUPT VECTOR
81
82          ; Set up memory parity control
83
84 007056 004737 016316'    1$:      CALL      PARSET      ;SET UP MEMORY PARITY CONTROL
85
86          ; Determine how much memory is installed on machine
87
88 007062 004737 015334'    CALL      MEMTST      ;FIND OUT HOW MUCH PHYSICAL MEMORY THERE IS
89
90          ; Set up information about the size of the job context area
91
92 007066 004737 015646'    CALL      CXTALC      ;Determine size of job context area
93
94          ; Load TSX-Plus device handlers that go in low memory
95
96 007072 004737 016320'    CALL      GETHNL      ;Load low memory handlers
97
98          ; Reserve space for interrupt vector intercept routines for mapped handlers
99
100 007076 010537 000122'    MOV      R5,XMVBAS      ;Save address of base of area for XM vectors
101 007102 013701 000124'    MOV      NM$HAN,R1      ;Get # mapped handlers
102 007106 006301          ASL      R1          ;Reserve room for 2 interrupts per handler
103 007110 062701 000000G    ADD      #NXIVMH,R1      ;Add # requested extra interrupt vectors
104 007114 070127 000032          MUL      #MPIVSZ,R1      ;Calc space needed for interrupt entry code
105 007120 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 007122 004737 024600'    CALL      SETSY      ;SET UP INFO ABOUT SY DEVICE
110
111          ; Open channel to TSKMON and set up information about it.
112
113 007126 004737 013142'    CALL      OPNKMN      ;OPEN CHANNEL TO TSKMON
114

```

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

```

115      ; Set up information about the IND program
116      ;
117 007132 004737 014110'      CALL      INDINI      ; INITIALIZE FOR IND PROGRAM
118      ;
119      ; Initialize the TSXUCL data file
120      ;
121 007136 004737 014652'      CALL      UCLINI
122      ;
123      ; Set name of device that UCL program is to be run from
124      ;
125 007142 013737 000000G 000000G      MOV      SYNAME,UCLNAM      ; SET DEVICE NAME FOR UCL PROGRAM
126      ;
127      ; Initialize spooling system
128      ;
129 007150 004737 010522'      CALL      SPLINI      ; INITIALIZE SPOOLING SYSTEM
130      ;
131      ; Open system swap file
132      ;
133 007154 105737 000000G      TSTB      VSWPFL      ; IS JOB SWAPPING ALLOWED?
134 007160 001402      BEQ      3$      ; BR IF NOT
135 007162 004737 007564'      CALL      OPNSWP      ; OPEN THE SYSTEM SWAP FILE
136      ;
137      ; Open swap file used for PLAS regions
138      ;
139 007166 004737 010216'      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 007172 004737 021376'      CALL      SETMIO      ; Set up information about mapped I/O
144      ;
145      ; We are finished allocating low-memory buffer space.
146      ;
147 007176 010500      MOV      R5,R0      ; ENSURE WE DON'T OVERFLOW 40KB
148 007200 004737 025104'      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 007204 010537 000000G      MOV      R5,UMSYTP      ; SAVE ADDRESS OF NON-EXTENDED SYSTEM TOP
154 007210 062705 000077      ADD      #77,R5      ; BOUND UP TO 64-BYTE BOUNDARY
155 007214 072527 177772      ASH      #-6,R5      ; CONVERT TO 64-BYTE BLOCK #
156 007220 042705 176000      BIC      #176000,R5      ; KILL SIGN EXTENSION
157      ;
158      ; Allocate buffer space that is not constrained to 40Kb TSX-Plus region.
159      ;
160 007224 004737 012536'      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 007230 010504      MOV      R5,R4      ; Save the base of free memory in R4
166 007232 010437 000136'      MOV      R4,FMEMLO      ; Save pointer above top of alloc low memory
167 007236 013705 000134'      MOV      FMEMHI,R5      ; Get 64-byte block # of free high memory
168      ;
169      ; Load any mapped system code
170      ;
171 007242 004737 022010'      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 007246 005727 000000G      TST      #NUMRDB      ;Do we need to load any shared run-times?
176 007252 001415      BEQ      4$      ;Br if not
177 007254 012701 000000G      MOV      #RDB,R1      ;Point to 1st run-time descriptor block
178 007260 010502      MOV      R5,R2      ;Save initial memory pointer
179 007262 004737 023472' 5$: CALL      GETSRT      ;Load a shared run-time system
180 007266 062701 000000G      ADD      #RT$$SZ,R1      ;Point to next shared run-time descriptor
181 007272 020127 000000G      CMP      R1,#RDBEND      ;Are there more to load?
182 007276 103771      BLO      5$      ;Br if yes
183 007300 160502      SUB      R5,R2      ;Compute amt of space used by run-times
184 007302 010237 000000G      MOV      R2,SRTSIZ      ;Save total run-time size
185 007306      4$:
186      .IF      NE,PROASM
187      ;
188      ; If we are running on a Pro, load the PI handler like a shared run-time
189      ;
190 007306 105737 000000G      TSTB     PROFLG      ;Are we running on a Pro?
191 007312 001410      BEQ      10$      ;Br if not
192 007314 012701 000244'      MOV      #PISRT,R1      ;Point to dummy shared run-time block for PI
193 007320 010502      MOV      R5,R2      ;Save current memory pointer
194 007322 004737 023472'      CALL      GETSRT      ;Load PI handler like a shared run-time
195 007326 160502      SUB      R5,R2      ;Calculate amt of space used by PI handler
196 007330 060237 000000G      ADD      R2,MHNSIZ      ;Count in mapped-handler size
197      .ENDC      ;NE,PROASM
198      ;
199      ; Load any mapped handlers
200      ;
201 007334 004737 017634' 10$: CALL      GETHNH      ;Load mapped handlers
202      ;
203      ; Allocate space for data cache buffers and control tables
204      ;
205 007340 004737 024104'      CALL      CSHBUF      ;Allocate space for data cache
206      ;
207      ; We have finished allocating all of the memory used by the system.
208      ; Allocate and initialize a memory map table that will be used to
209      ; show which pages are available for user jobs.
210      ;
211 007344 004737 016006'      CALL      MAPALC      ;Allocate memory map table
212      ;
213      ; Set up info about maximum memory space available to jobs
214      ;
215 007350 004737 016172'      CALL      SETJSZ      ;SET JOB SIZE INFO
216      ;
217      ; Set up date and time
218      ;
219 007354 013702 000000G      MOV      SYTIML,R2      ;Save time we got at start of init
220 007360      .GTIM     #AREA,#SYTIMH      ;SET TIME OF DAY
221 007400      .DATE      ;GET DATE
222 007406 010037 000000G      MOV      R0,SYSDAT      ;SET SYSTEM DATE
223 007412 020237 000000G      CMP      R2,SYTIML      ;Make sure some time has elapsed
224 007416 001010      BNE      11$      ;Br if clock is running
225 007420      .PRINT     #TSXHD      ;Print error message heading
226 007426      .PRINT     #NOCLOK      ;Print clock-not-working message
227 007434 000137 004134'      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 007440 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 007442 013702 000154'      MOV      WRKSIZ,R2      ; Get size of work buffer
237 007446 006202      ASR      R2      ; Convert to # words
238 007450 013703 000152'      MOV      WRKBUF,R3      ; Get address of work buffer area
239 007454 000241      CLC      ; Convert to block # in TSX.SAV file
240 007456 006003      ROR      R3
241 007460 000303      SWAB     R3
242 007462      .READW #AREA,#17,WRKBUF,R2,R3 ; Read back TSX over work buffer
243      ;
244      ; See if user requested control-C abort
245      ;
246 007516 004737 025144'      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 007522 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 007526 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 007530 012702 000000G      MOV      #EXCBUF,R2      ;Get address of base of buffer area
18 007534 062702 000777      ADD      #777,R2        ;Bound up to block boundary
19 007540 042702 000777      BIC      #777,R2        ;Set to block boundary
20 007544 010237 000152'      MOV      R2,WRKBUF
21 007550 012737 004000 000154'  MOV      #2048.,WRKSIZ
22         ;
23         ; Finished
24         ;
25 007556 012602      MOV      (SP)+,R2
26 007560 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         ALCHRB: MOV      R2,-(SP)
15         ;
16         ; Get count of # RCB's to build
17         ;
18         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         MOV      R5,HANRCB          ;Start of RCB area
23         MOV      R5,HANRCO          ;Store offset relative to MONVEC
24         SUB      #MONVEC,HANRCO      ;Convert address to offset
25         MOV      #-1,(R5)+          ;Store -1 at start of area
26         ;
27         ; Allocate and initialize to zero the RCB's
28         ;
29         1$: MOV      #5,R2            ;Each RCB has 5 words
30         2$: CLR      (R5)+           ;Zero the RCB
31         SOB      R2,2$
32         ;
33         ; See if there are more RCB's to build
34         ;
35         DEC      R0                  ;More RCB's to initialize?
36         BGT      1$                 ;Loop if yes
37         ;
38         ; Store -1 at end of RCB area
39         ;
40         MOV      #-1,(R5)+          ;Mark end of RCB list
41         ;
42         ; Finished
43         ;
44         MOV      (SP)+,R2
45         RETURN
46         ;
47         .IFF      ;NE,<PROASM-1>      ;Dummy for Pro-only
48         ALCHRB: RETURN
49         .ENDC      ;NE,<PROASM-1>

```

007562 000207

```

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 $DEAD flag for the line if INIABT=0.
8          ;
9          LINCHK: MOV     R1,-(SP)
10         MOV     R2,-(SP)
11         MOV     R3,-(SP)
12         MOV     R4,-(SP)
13         MOV     @#4,-(SP)        ; SAVE ORIGINAL TRAP VECTOR
14         ;
15         ; Take over trap control
16         ;
17         MOV     #6$,@#4          ; CATCH TRAPS
18         ;
19         ; Loop through the test for each line.
20         ;
21         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         1$: CALL     LINTYP        ; Determine the type of this line
27         BIT      #$HARD,LSW3(R1)  ; Is this line connected to hardware?
28         BEQ      31$              ; Br if not
29         MOV      LMXNUM(R1),R3     ; IS THIS A DL-11 OR MULTIPLEXER LINE?
30         BEQ      2$               ; BR IF DL-11
31         MOV      MXCSR(R3),R2      ; GET DZ11 OR DH11 STATUS REGISTER ADDRESS
32         BEQ      11$              ; BR IF ALREADY MARKED AS DEAD
33         MOV      MXVEC(R3),R4      ; GET MUX INTERRUPT VECTOR ADDRESS
34         BR       3$
35         11$: CALL    4$             ; MARK LINE AS DEAD
36         BR       31$              ; CONTINUE CHECKING TERMINALS
37         2$: MOV     RSR(R1),R2      ; GET DL-11 STATUS REGISTER ADDRESS
38         MOV      INVEC(R1),R4      ; GET DL-11 INTERRUPT VECTOR ADDRESS
39         ; Check validity of status register address.
40         3$: CMP     R2,#160000     ; IS IT IN I/O PAGE?
41         BLO      LINTRP            ; ERROR IF NOT
42         BIT      #7,R2            ; IS IT ON 8-BYTE BOUNDARY?
43         BNE      LINTRP            ; ERROR IF NOT
44         TST      @R2              ; TRY TO ACCESS IT AND SEE IF WE TRAP
45         ; Check validty of interrupt vector address.
46         CMP      R4,#60            ; CAN'T BE BELOW 60
47         BLO      BADVEC
48         CMP      R4,#500           ; OR ABOVE 500
49         BHI      BADVEC
50         BIT      #7,R4            ; MUST BE ON 8-BYTE BOUNDARY
51         BNE      BADVEC
52         ; This line looks good. Check next.
53         31$: SUB     #2,R1          ; MORE TO CHECK?
54         BGT      1$               ; BR IF YES
55         ;
56         ; Finished -- all lines look ok.
57         ;

```

```

58          MOV      (SP)+, @#4          ; RESTORE TRAP VECTOR
59          MOV      (SP)+, R4
60          MOV      (SP)+, R3
61          MOV      (SP)+, R2
62          MOV      (SP)+, R1
63          RETURN
64          ;
65          ; See if we should abort or just mark the line as dead.
66          ;
67          4$:      TSTB      VINABT          ; DOES HE WANT TO ABORT?
68                  BNE      LINTRP          ; YES
69                  BIS      #$DEAD, LSW3(R1) ; MARK LINE AS DEAD
70                  TST      R3              ; IS THIS A DL11 OR A MUX LINE?
71                  BEQ      5$              ; BR IF DL11
72                  CLR      MXCSR(R3)       ; MARK DZ OR DH AS DEAD
73          5$:      RETURN
74          ;
75          ; Trap occured while trying to access status register.
76          ;
77          6$:      CALL      4$              ; REPORT ERROR OR MARK AS DEAD LINE
78                  RTI                    ; RETURN TO LINE CHECKING
79          ;
80          ; Error: Invalid status register address.
81          ; R1 = Line number, R2 = status register address
82          ;
83          ;
84          LINTRP: .PRINT      #TSXHD          ; PRINT ERROR MESSAGE
85                  .PRINT      #BADLIN
86                  BR          ERP
87          ;
88          ; Error: Invalid interrupt vector address.
89          ; R1 = Line number, R4 = interrupt vector address
90          ;
91          BADVEC: .PRINT      #TSXHD          ; PRINT ERROR MESSAGE
92                  .PRINT      #BDVMSG
93                  MOV      R4, R2          ; GET VECTOR ADDRESS TO R2
94          ERP:    MOV      R2, R0          ; GET ADDRESS TO R0
95                  CALL      PRTOCT        ; PRINT OCTAL VALUE
96                  .PRINT      #CRLF
97                  .PRINT      #BDLMSG          ; LINE # =
98                  MOV      R1, R0          ; GET LINE NUMBER
99                  ASR      R0              ; # 1
100                 CALL      PRTDEC        ; PRINT LINE NUMBER
101                 .PRINT      #CRLF
102                 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 007564   010146
14 007566   010246
15 007570   010346
16
17         ; Load RT-11 handler for swap device.
18         ;
19 007572   013700   000000G          MOV     SWDBLK,R0          ;Get name of device
20 007576   004737   024770'          CALL    RTFTCH          ;Fetch the RT-11 handler
21 007602   103546          BCS      11$          ;Br if invalid device
22
23         ; Compute the maximum number of slots in swap file that we could need
24         ;
25 007604   012703   000000C          MOV     #NSL+NDL,R3          ;Get # virtual lines and detached jobs
26 007610   012701   000000G          MOV     #LSTPL,R1          ;Get index to last primary line
27 007614   032761   000000G 000000G 1$: BIT     #$DEAD,LSW3(R1) ;Is this line installed?
28 007622   001001          BNE      5$          ;Br if not
29 007624   005203          INC      R3          ;Count another primary line
30 007626   162701   0000002          SUB     #2,R1          ;Get next line index
31 007632   003370          BGT      1$          ;Loop if more lines to check
32 007634   020337   000000G          CMP     R3,VSWPSL          ;Compare with # slots specified
33 007640   002002          BGE      6$          ;Br if VSWPSL value is ok
34 007642   010337   000000G          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 007646   013703   000000G 6$: MOV     VHIMEM,R3          ;GET # BLOCKS NEEDED FOR LARGEST JOB SIZE
39 007652   006303          ASL      R3
40 007654   063703   000000G          ADD     CXTPAG,R3          ;ADD # BLOCKS NEEDED FOR JOB CONTEXT AREA
41 007660   010337   000000G          MOV     R3,SLTSIZ          ;Save size of swap file slot
42
43         ; Compute the total number of blocks needed for the swap file.
44         ;
45 007664   070337   000000G          MUL     VSWPSL,R3          ;Multiply by # slots in swap file
46
47         ; R3 now contains total number of blocks needed in swap file.
48         ; See if swap file already exists on disk.
49         ;
50 007670          4$: .LOOKUP #AREA,#1,#SWDBLK;DOES SWAP FILE EXIST NOW?
51 007710   103415          BCS      2$          ;BR IF NOT
52
53         ; Swap file exists. See if it is the right size.
54 007712   020003          CMP     R0,R3          ;IS SWAP FILE THE RIGHT SIZE?
55 007714   001453          BEQ      3$          ;BR IF YES
56
57         ; Old swap file is not of correct size.
58         ; Delete it and open a new swap file.
59         .CLOSE #1

```

```

58 007724          .DELETE #AREA,#1,#SWDBLK;DELETE THE OLD SWAP FILE
59
60          ; Create a new swap file.
61
62 007744          2$:      .ENTER  #AREA,#1,#SWDBLK,R3 ;CREATE A NEW SWAP FILE
63 007770 103443      BCS      9$          ;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 007772 005303      DEC      R3          ;GET # OF LAST BLOCK IN FILE
70 007774          .WRITW  #AREA,#1,#TSINIT,#256.,R3 ;WRITE TO LAST BLOCK IN FILE
71 010032 005203      INC      R3          ;GET BACK # BLOCKS IN FILE
72 010034          .CLOSE  #1          ;CLOSE FILE WE CREATED
73 010042 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 010044 012700 000000G 3$:      MOV      #SWPCHN,R0      ;POINT TO SWAP CHANNEL BLOCK
79 010050 013702 000000G      MOV      SWDBLK,R2      ;GET DEVICE NAME
80 010054 004737 024410'      CALL     SETCHN      ;SET UP SWAP CHANNEL INFO
81
82          ; Release the RT-11 device handler
83
84 010060          .RELEAS #SWDBLK      ;Release RT-11 device handler
85
86          ; Finished
87
88 010070 012603      MOV      (SP)+,R3
89 010072 012602      MOV      (SP)+,R2
90 010074 012601      MOV      (SP)+,R1
91 010076 000207      RETURN
92
93          ; Error: Cannot open swap file
94
95 010100          9$:      .PRINT  #TSXHD      ;PRINT ERROR MESSAGE
96 010106          .PRINT  #BADOPN
97 010114 004737 010142'      CALL     SPNEED      ;Print info about number of blocks needed
98
99          ; Error: Invalid device specification.
100
101 010120 010001      11$:     MOV      R0,R1      ;Save device name
102 010122          .PRINT  #TSXHD      ;Print error message
103 010130          .PRINT  #BADOPN
104 010136 004737 010170'      CALL     BADDEV      ;Print invalid device specification
105
106          ; Error: Number of contiguous blocks required.
107
108 010142          SPNEED: .PRINT  #CONSPC      ;Print contiguous blocks needed
109 010150 010300      MOV      R3,R0      ;GET # BLOCKS NEEDED FOR FILE
110 010152 004737 025230'      CALL     PRTDEC      ;DISPLAY # BLOCKS NEEDED
111 010156          .PRINT  #CRLF
112 010164 000137 004134'      JMP      INISTP      ;ABORT INITIALIZATION
113
114          ; Bad file specification.

```

115				
116	010170		BADDEV: .PRINT	#CFHMSG ;Print invalid device specification
117	010176	010100	MOV	R1,R0 ;Get the rad50 device name
118	010200	004737 025274'	CALL	PRTR50 ;Print rad50 device name
119	010204		.PRINT	#CRLF ;Print carriage return/line feed
120	010212	000137 004134'	JMP	INISTP ;Abort initialization

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	010216	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	010220	105737 000000G	TSTB VSWPFL ;Is this a non-swapping system?
17	010224	001513	BEQ 9\$;Br if yes
18	010226	005737 000000G	TST VPLAS ;Is a PLAS swap file wanted?
19	010232	001510	BEQ 9\$;Br if not
20			;
21			; Load RT-11 device handler for swap device
22			;
23	010234	013700 000000G	MOV RSFBLK, R0 ;Get name of device
24	010240	004737 024770'	CALL RTFTCH ;Try to fetch the RT-11 device handler
25	010244	103515	BCS 11\$;Br if error on handler fetch
26	010246	013703 000000G	MOV VPLAS, R3 ;Get # blocks in PLAS swap file
27			;
28			; See if PLAS swap file already exists on disk
29			;
30	010252		4\$: .LOOKUP #AREA, #1, #RSFBLK ;Try to find existing PLAS swap file
31	010272	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	010274	020037 000000G	CMP R0, VPLAS ;Is swap file of the correct size?
37	010300	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	010302		.CLOSE #1 ;Close and delete the old file
43	010310		.DELETE #AREA, #1, #RSFBLK; Delete the old file
44			;
45			; Create new swap file
46			;
47	010330		2\$: .ENTER #AREA, #1, #RSFBLK, VPLAS ;Create a new PLAS swap file
48	010356	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	010360	005303	DEC R3 ;Get # of last block in file
55	010362		.WRITW #AREA, #1, #TSINIT, #256., R3
56	010420		.CLOSE #1
57	010426	000711	BR 4\$;Go back and lookup file

```
58 ;
59 ; Swap file has been successfully opened using lookup.
60 ; Copy channel status to TSX channel block.
61 ;
62 010430 012700 0000000 3$:      MOV      #SEGCHN,R0      ;Point to TSX PLAS swap channel
63 010434 013703 0000000      MOV      RSFBLK,R3      ;Get device name
64 010440 004737 024410'      CALL     SETCHN      ;Set up TSX channel block
65 ;
66 ; Release the RT-11 device handler
67 ;
68 010444      .RELEAS #RSFBLK      ;Release RT-11 device handler
69 ;
70 ; Finished
71 ;
72 010454 012603 9$:      MOV      (SP)+,R3
73 010456 000207      RETURN
74 ;
75 ; Error -- Cannot open PLAS swap file
76 ;
77 010460 10$:      .PRINT  #TSXHD      ;Print error prefix
78 010466      .PRINT  #RSFERR      ;Print error message
79 010474 004737 010142'      CALL     SPNEED      ;Print information about amt of space needed
80 ;
81 ; Error: Invalid device specification.
82 ;
83 010500 010001 11$:      MOV      R0,R1      ;Save device name
84 010502      .PRINT  #TSXHD      ;Print error message
85 010510      .PRINT  #RSFERR      ;Print error message
86 010516 004737 010170'      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 010522 005727 000000G  SPLINI: TST      #SPLND      ;Are there any spooled devices?
8 010526 001401          BEQ      13$          ;Br if not
9 010530 000401          BR       10$          ;Initialize the spooled devices
10 010532 000207 13$:    RETURN
11          ;
12          ;   There are some spooled devices
13          ;
14 010534 010146 10$:    MOV      R1,-(SP)
15 010536 010246          MOV      R2,-(SP)
16 010540 010346          MOV      R3,-(SP)
17 010542 010446          MOV      R4,-(SP)
18 010544 010546          MOV      R5,-(SP)
19          ;
20          ;   Open each spooled device
21          ;
22 010546 105037 000000G          CLRB     NSPLDV      ;INIT COUNT OF # ACTUAL SPOOLED DEVICES
23 010552 012701 000000G          MOV      #SDCB,R1    ;POINT TO 1ST SDCB
24 010556 012703 000000G          MOV      #SPLDEV,R3   ;POINT TO TABLE OF RAD50 DEV NAMES
25 010562 012704 000000G          MOV      #SPLANM,R4   ;POINT TO TABLE OF ASCII DEV NAMES
26 010566 004737 011574' 2$:    CALL     FORCE0        ;FORCE UNIDENTIFIED UNIT #S TO 0
27 010572 011302          MOV      (R3),R2             ;GET RAD50 NAME OF SPOOLED DEVICE
28 010574 010261 000000G          MOV      R2,SDNAME(R1) ;SET NAME IN SDCB
29 010600 010100          MOV      R1,R0               ;GET ADDRESS OF SDCB
30 010602 062700 000000G          ADD      #SDANAM,R0    ;POINT TO CELL FOR ASCII NAME
31 010606 112420          MOVB     (R4)+,(R0)+          ;MOVE IN ASCII DEVICE NAME
32 010610 112420          MOVB     (R4)+,(R0)+
33 010612 112420          MOVB     (R4)+,(R0)+
34 010614 020227 000000G          CMP      R2,#DMYDEV    ;Is this a dummy entry for later patching?
35 010620 001451          BEQ      1$                  ;Br if yes -- Ignore it
36 010622 010200          MOV      R2,R0               ;Get name to R0
37 010624 004737 011476'          CALL     CVTDVU        ;Convert name to device # and unit #
38 010630 010061 000000G          MOV      R0,SDDVU(R1)  ;Store device # and unit # in SDCB
39 010634 010200          MOV      R2,R0               ;Get device name
40 010636 004737 011372'          CALL     CHKCLD        ;See if this is a CL device?
41 010642 103406          BCS      14$                 ;Br if not
42 010644 004737 011306'          CALL     SPLCLD        ;Set up for spooling to CL device
43 010650 103414          BCS      3$                  ;Br if invalid unit
44 010652 105237 000000G          INCB     NSPLDV        ;Count # of actual spooled devices
45 010656 000432          BR       1$                  ;Process next device
46 010660 010100 14$:    MOV      R1,R0               ;Get address of SDCB
47 010662 062700 000000G          ADD      #SDCHAN,R0    ;Point to channel block within SDCB
48 010666 004737 024322'          CALL     OPNCHN        ;Set TSX-Plus channel block open to device
49 010672 103403          BCS      3$                  ;Br if did not recognize device
50 010674 105237 000000G          INCB     NSPLDV        ;Count # actual spooled devices
51 010700 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 010702 012761 000000G 000000G 3$:    MOV      #DMYDEV,SDNAME(R1); SAY THIS DEVICE IS NOT SPOOLED

```

SPLINI -- Initialize spooling system

```

58 010710 105737 000000G          TSTB    VINABT          ;ABORT OR CONTINUE ON ERRORS?
59 010714 001413                   BEQ      1$              ;BR TO IGNORE DEVICE AND CONTINUE INIT
60 010716                   .PRINT  #TSXHD          ;PRINT ERROR MESSAGE
61 010724                   .PRINT  #BDSPOP
62 010732 010200                   MOV      R2,R0          ;GET RAD50 DEVICE NAME
63 010734 004737 025274'          CALL     PRTR50         ;PRINT DEVICE NAME
64 010740 000137 004134'          JMP      INISTP         ;ABORT INITIALIZATION
65                               ;
66                               ; Process next spooled device
67                               ;
68 010744 062701 000000G          1$:    ADD      #SDCBSZ,R1          ;POINT TO NEXT SDCB
69 010750 005723                   TST      (R3)+          ;POINT TO NEXT DEVICE NAME
70 010752 020327 000000G          CMP      R3,#SPLDVN        ;OPENED ALL SPOOLED DEVICES?
71 010756 103703                   BLO      2$              ;BR IF MORE TO DO
72                               ;
73                               ; Open the spool file
74                               ;
75 010760 105737 000000G          TSTB    NSPLDV          ;ARE THERE ANY ACTUAL SPOOLED DEVICES?
76 010764 001521                   BEQ      12$            ;BR IF THERE ARE NO ACTUAL SPOOLED DEVICES
77 010766 013700 000000G          MOV      SPLBLK,R0        ;Get name of device for spool file
78 010772 004737 024770'          CALL     RTFTCH          ;Fetch the RT-11 device handler
79 010776 103532                   BCS      11$            ;Br if cannot fetch handler
80 011000 013702 000000G          MOV      NSPLBL,R2         ;GET # BLOCKS TO ALLOCATE FOR FILE
81 011004 005202                   INC      R2              ;Add 1 extra block
82                               ;
83                               ; See if spool file already exists
84                               ;
85 011006                   5$:    .LOOKUP #AREA,#1,#SPLBLK;SEE IF SPOOL FILE ALREADY EXISTS
86 011026 103415                   BCS      6$              ;BR IF IT DOES NOT EXIST
87 011030 020002                   CMP      R0,R2           ;IS IT THE RIGHT SIZE?
88 011032 001453                   BEQ      7$              ;BR IF YES
89 011034                   .CLOSE  #1                    ;IT EXISTS BUT IS OF WRONG SIZE
90 011042                   .DELETE #AREA,#1,#SPLBLK;DELETE CURRENT FILE AND OPEN NEW ONE
91                               ;
92                               ; Open new spool file
93                               ;
94 011062                   6$:    .ENTER #AREA,#1,#SPLBLK,R2;CREATE A NEW SPOOL FILE
95 011106 103456                   BCS      8$              ;BR IF ERROR ON ENTER
96                               ; Write to last block in file to reserve full file space
97 011110 010203                   MOV      R2,R3           ;Get # of blocks in file
98 011112 005303                   DEC      R3              ;Get # of last block in file
99 011114                   .WRITW #AREA,#1,#TSINIT,#256.,R3
100                               ; Now close and reopen using a lookup
101 011152                   .CLOSE  #1                    ;CLOSE SPOOL FILE
102 011160 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 011162 012700 000000G          7$:    MOV      #SPLCHN,R0          ;SAVE CHANNEL STATUS HERE
108 011166 013702 000000G          MOV      SPLBLK,R2          ;GET DEVICE NAME
109 011172 004737 024410'          CALL     SETCHN          ;SAVE CHANNEL STATUS
110 011176                   .RELEAS #SPLBLK          ;Release the RT-11 device handler
111                               ;
112                               ; Set number of free public blocks in spool file
113                               ;
114 011206 113703 000000G          MOVVB   NSPLDV,R3          ;Get # spooled devices

```

```

115 011212 070327 000000G      MUL      #PVSPBL,R3      ;Times number of private blocks per dev
116 011216 005403              NEG      R3
117 011220 063703 000000G      ADD      NSPLBL,R3      ;Get # public spool blocks
118 011224 010337 000000G      MOV      R3,NFRESB      ;This is number of public free spool blocks
119
120      ; Finished
121
122 011230 012605      12$:      MOV      (SP)+,R5
123 011232 012604              MOV      (SP)+,R4
124 011234 012603              MOV      (SP)+,R3
125 011236 012602              MOV      (SP)+,R2
126 011240 012601              MOV      (SP)+,R1
127 011242 000207      9$:      RETURN
128
129      ; Error: Cannot open spool file.
130
131 011244              8$:      .PRINT   #TSXHD      ;PRINT ERROR MESSAGE
132 011252              .PRINT   #BOSF
133 011260 000137 004134'      JMP      INISTP      ;ABORT INITIALIZATION
134
135      ; Error: Invalid device specification.
136
137 011264 010001      11$:      MOV      R0,R1      ;Save device name
138 011266              .PRINT   #TSXHD      ;Print error message
139 011274              .PRINT   #BOSF
140 011302 004737 010170'      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 011306   010546  SPLCLD: MOV      R5, -(SP)
14         ;
15         ; Make sure CL unit number is valie
16         ;
17 011310   010005          MOV      R0, R5          ; Get CL unit number
18 011312   020527 000000G   CMP      R5, #CLTOTL    ; Is this a valid unit #
19 011316   103022          BHIS     8$             ; Br if invalid
20         ;
21         ; Set up channel control block in SDCB
22         ;
23 011320   005061 000000C   CLR      SDCHAN+C.SBLK(R1); Starting block # = 0
24 011324   020527 000007   CMP      R5, #7         ; Is this a CL or C1 unit?
25 011330   101405          BLOS     1$             ; Br if CL unit
26 011332   162705 000010   SUB      #8, R5         ; Remove C1 unit # bias
27 011336   013700 000000G   MOV      C1DEVX, R0      ; Get C1 device index number
28 011342   000402          BR       2$             ;
29 011344   013700 000000G  1$: MOV      CLDEVX, R0      ; Get CL device index number
30 011350   010061 000000C  2$: MOV      R0, SDCHAN+C.CSW(R1); Set device index number
31 011354   110561 000000C   MOVB    R5, SDCHAN+C.DEVQ(R1); Set unit #
32         ;
33         ; We successfully set up a CL unit
34         ;
35 011360   000241          CLC                      ; Signal success on error
36 011362   000401          BR       9$             ;
37         ;
38         ; We cannot open this CL unit
39         ;
40 011364   000261  8$: SEC                      ; Signal error
41         ;
42         ; Finished
43         ;
44 011366   012605  9$: MOV      (SP)+, R5
45 011370   000207          RETURN

```

```

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          ;   R0 = 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         ;   R0 = CL unit number (0-15)
12         ;
13 011372   CHKCLD:
14         ;
15         ; See if this is a CL unit
16         ;
17 011372   020027   012240          CMP     R0,#R50CL      ; Is name "CL"?
18 011376   001411          BEQ      1$                  ; Br if yes
19 011400   020027   012276          CMP     R0,#R50CLO     ; Is name in the range CLO to CL7?
20 011404   103432          BLO      8$                  ; Br if not
21 011406   020027   012305          CMP     R0,#R50CL7
22 011412   101005          BHI      2$
23 011414   162700   012276          SUB     #R50CLO,R0    ; Get CL unit number
24 011420   000422          BR       7$
25 011422   005000          1$:      CLR     R0            ; CL = CLO
26 011424   000420          BR       7$
27         ;
28         ; See if this is a C1 unit
29         ;
30 011426   020027   013630          2$:    CMP     R0,#R50C1      ; Is name "C1"?
31 011432   001413          BEQ      3$                  ; Br if yes
32 011434   020027   013666          CMP     R0,#R50C10     ; Is name in the range C10 to C17?
33 011440   103414          BLO      8$                  ; Br if not
34 011442   020027   013675          CMP     R0,#R50C17
35 011446   101011          BHI      8$
36 011450   162700   013666          SUB     #R50C10,R0    ; Get unit number
37 011454   062700   000010          ADD     #8.,R0        ; Bias by 8 for C1 units
38 011460   000402          BR       7$
39 011462   012700   000010          3$:    MOV     #8.,R0    ; C1 = CL8
40         ;
41         ; This is a CL or C1 unit
42         ;
43 011466   000241          7$:      CLC                      ; Signal success on return
44 011470   000401          BR       9$
45         ;
46         ; This is not a CL or C1 unit
47         ;
48 011472   000261          8$:      SEC                      ; Signal failure on return
49         ;
50         ; Finished
51         ;
52 011474   000207          9$:      RETURN

```

```

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          ;   R0 = RAD50 device name.
8          ;
9          ; Outputs:
10         ;   C-flag cleared ==> Conversion successful.
11         ;   C-flag set    ==> Unable to find device name in tables.
12         ;   R0 = Device index number (low byte), device unit number (high byte).
13         ;
14 011476 010246 CVTDVU: MOV     R2,-(SP)
15 011500 010346      MOV     R3,-(SP)
16
17         ; Split the unit number off of the full device name
18         ;
19         MOV     R0,R3          ;Get full device name
20         CLR     R2             ;Set up for divide
21         DIV     #50,R2        ;Split name and unit (R0=name, R1=unit)
22         TST     R3             ;Was a unit number specified?
23         BEQ     1$            ;Br if not
24         SUB     #36,R3        ;Convert unit number to binary value
25 011522 010300 1$:  MOV     R3,R0          ;Get unit number
26 011524 000300      SWAB     R0          ;Position to high-order byte
27
28         ; Look up the device name to get the device index
29         ;
30         MUL     #50,R2        ;Now get the device name without unit number
31         MOV     NUMDEV,R2     ;Get index number of last device
32         CMP     R3,PNAME(R2)  ;Search for device in name table
33         BEQ     3$            ;Br if found it
34         SUB     #2,R2         ;Try next device
35 011550 002372      BGE     2$            ;Loop if more to check
36
37         ; Error, cannot find device name in tables
38         ;
39         MOV     #-1,R0        ;Set device # = unit # = -1
40 011556 000261      SEC                     ;Signal error on return
41 011560 000402      BR      9$
42
43         ; Found the device in the tables
44         ;
45         BIS     R2,R0         ;Combine device # and unit #
46 011564 000241      CLC                     ;Signal success on return
47
48         ; Finished
49         ;
50 011566 012603 9$:  MOV     (SP)+,R3
51 011570 012602      MOV     (SP)+,R2
52 011572 000207      RETURN

```

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

```

1
2
3
4
5
6
7
8 011574 010346
9 011576 010446
10 011600 010546
11 011602 011305
12 011604 005004
13 011606 071427 000050
14 011612 005705
15 011614 001012
16 011616 010405
17 011620 005004
18 011622 071427 000050
19 011626 005704
20 011630 001404
21 011632 005705
22 011634 001402
23 011636 062713 000036
24 011642 012605
25 011644 012604
26 011646 012603
27 011650 000207

      .SBTTL  FORCE0 -- Force a 2-char dev name to unit 0
      -----
      ; Inputs: R3 points to a RAD50 device name
      ;
      ; Outputs: If the 3rd char of the device name pointed to by R3 is
      ;           blank, then it is changed to 0
      ;
FORCE0: MOV     R3, -(SP)
        MOV     R4, -(SP)
        MOV     R5, -(SP)
        MOV     (R3), R5      ; MOVE CURRENT DEV NAME TO R5
        CLR     R4            ; SET UP FOR DIVIDE
        DIV     #50, R4       ; SEPARATE INTO NAME AND UNIT
        TST     R5            ; WAS 3RD CHAR BLANK?
        BNE     9$            ; RETURN IF NOT
        MOV     R4, R5        ; GET HIGH 2 CHARS
        CLR     R4            ; SET UP FOR ANOTHER DIVIDE
        DIV     #50, R4       ; SEPARATE 1 & 2 CHARS
        TST     R4            ; WAS CHAR 1 BLANK?
        BEQ     9$            ; EMPTY OR INVALID DEV NAME!
        TST     R5            ; WAS CHAR 2 BLANK?
        BEQ     9$            ; 1-CHAR DEV NAME SHOULD BE INVALID???
        ADD     #^R 0, (R3)   ; FORCE TO UNIT 0
9$:     MOV     (SP)+, R5
        MOV     (SP)+, R4
        MOV     (SP)+, R3
        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 011652 010146 ALOCBF: MOV      R1, -(SP)
15         ;
16         ; Assign space for I/O queue elements.
17         ;
18 011654 010537 000000G      MOV      R5, FREIOQ      ; START OF I/O QUEUE SPACE
19 011660 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 011664 010537 000000G      MOV      R5, SHRRCB      ; Start of area for RCB's
24 011670 013701 000000G      MOV      VNGR, R1       ; Get number of RCB's wanted
25 011674 020137 000000G      CMP      R1, VMXWIN      ; Must have one for each display window
26 011700 103002          BHS      13$                ; Br if ok
27 011702 013701 000000G      MOV      VMXWIN, R1      ; Force one for each window
28 011706 070127 000000G 13$:  MUL      #RC$$SZ, R1      ; Multiply by size of each block
29 011712 060105          ADD      R1, R5              ; Allocate space for RCB's
30 011714 010537 000000G      MOV      R5, SHRRCN      ; Address of end of region
31         ;
32         ; Assign space for fork blocks
33         ;
34 011720 012700 000000C      MOV      #<NUMFRK-FRKGEN>, R0 ; Get # fork blocks to allocate
35 011724 003404          BLE      11$                ; Br if none to allocate
36 011726 010537 000000G      MOV      R5, FRKINI      ; Set pointer to start of area
37 011732 062705 000000C      ADD      #<NUMFRK-FRKGEN>*FQ$$SZ, R5 ; Reserve space for fork blocks
38         ;
39         ; Assign space for job monitoring control blocks
40         ;
41 011736 013701 000000G 11$:  MOV      VMXMON, R1      ; Any job monitoring blocks wanted?
42 011742 001405          BEQ      10$                ; Br if not
43 011744 010537 000000G      MOV      R5, MONFQH      ; Start of job monitoring control blocks
44 011750 070127 000000G      MUL      #JM$$SZ, R1      ; Compute space needed for control blocks
45 011754 060105          ADD      R1, R5              ; Allocate the space
46         ;
47         ; Allocate space for system message buffers.
48         ;
49 011756 010537 000000G 10$:  MOV      R5, SNMSHD      ; HEAD OF SYSTEM MESSAGE BUFFER AREA
50 011762 062705 000000C      ADD      #<NMSNMB*SB$$SZ>, R5; RESERVE ROOM FOR MESSAGE BUFFERS
51         ;
52         ; Allocate space for INSTALLED program table
53         ;
54 011766 010537 000000G      MOV      R5, INSTBL      ; Base of table
55 011772 013701 000000G      MOV      VNUIP, R1       ; # slots for user installed programs
56 011776 062701 000000G      ADD      #NSIP, R1        ; Add # slots for system programs
57 012002 070127 000000G      MUL      #II$$SZ, R1      ; Multiply by size of each slot

```

```

58 012006 060105          ADD      R1,R5          ; Allocate space for table
59 012010 010537 000000G   MOV      R5,INSTBN      ; Pointer past end of INSTALL table
60
61                          ; Allocate space for device mount entries
62
63 012014 010537 000000G   MOV      R5,CSHDEV      ; Point to start of area
64 012020 012701 000000G   MOV      #CD$$SZ,R1      ; Get size of each entry
65 012024 070137 000000G   MUL      VMXCSH,R1      ; Multiply by number of entries
66 012030 060105          ADD      R1,R5          ; Reserve space for table
67 012032 010537 000000G   MOV      R5,CSHDVN      ; Save pointer past end of table
68
69                          ; Allocate space for data cache control blocks
70
71 012036 005737 000000G   TST      CSHALC          ; Is data caching wanted?
72 012042 001404          BEQ      12$             ; Br if not
73 012044 010537 000000G   MOV      R5,CCBHD        ; Head of free list area
74 012050 062705 000000C   ADD      #NUMCCB*CC$$SZ,R5 ; Allocate space for control blocks
75
76                          ; Allocate space for spool file control blocks
77
78 012054 013701 000000G   12$:  MOV      NSPLFL,R1      ; Get # spool file control blocks needed
79 012060 001407          BEQ      1$              ; Br if none needed
80 012062 070127 000000G   MUL      #SFCBSZ,R1      ; Compute space needed by control blocks
81 012066 010537 000000G   MOV      R5,SFCB        ; Base of control block area
82 012072 060105          ADD      R1,R5          ; Allocate space for control blocks
83 012074 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 012100 012701 000002    1$:  MOV      #2,R1        ; START WITH FIRST MUX
90 012104 020127 000000G   2$:  CMP      R1,#LSTMX      ; HAVE WE DONE ALL MUX'S?
91 012110 101007          BHI      5$              ; BR IF YES
92 012112 010561 000000G   MOV      R5,MXLNT(R1)    ; SET ADDRESS OF START OF VECTOR
93 012116 062705 000020    ADD      #16.,R5        ; RESERVE SPACE FOR VECTOR
94 012122 062701 000002    ADD      #2,R1          ; ADVANCE TO NEXT MUX
95 012126 000766          BR       2$
96
97                          ; Allocate buffers to hold characters for DMA transfers to DH11 multiplexers
98
99 012130 012701 000002    5$:  MOV      #2,R1        ; Start with first line
100 012134 020127 000000G   6$:  CMP      R1,#LSTPL      ; Is this a primary time-sharing line?
101 012140 101403          BLOS     3$              ; Br if yes
102 012142 020127 000000G   CMP      R1,#FSTIOL      ; Is this a CL line?
103 012146 103422          BLO      7$              ; Br if not
104 012150 026127 000000G 000000G 3$:  CMP      LCDTYP(R1),#CDX$DH ; Is this line connected to a DH11?
105 012156 001404          BEQ      8$              ; Br if yes
106 012160 026127 000000G 000000G CMP      LCDTYP(R1),#CDX$VH ; Is this line connected to a DHV11?
107 012166 001012          BNE      7$              ; Br if not
108 012170 010561 000000G   8$:  MOV      R5,LDHB1B(R1) ; Set address of start of buffer 1
109 012174 010561 000000G   MOV      R5,LDHB1P(R1) ; Initialize pointer into buffer 1
110 012200 062705 000000G   ADD      #DHBFSZ,R5      ; Reserve space for buffer
111 012204 010561 000000G   MOV      R5,LDHB2B(R1) ; Set address of start of buffer 2
112 012210 062705 000000G   ADD      #DHBFSZ,R5      ; Reserve space for buffer
113 012214 062701 000002    7$:  ADD      #2,R1        ; Get # of next line
114 012220 020127 000000G   CMP      R1,#LSTHL      ; Have we checked all lines?

```

115	012224	101743		BLOS	6\$		; Br if not
116	012226	005205		INC	R5		; Bound up to next word
117	012230	042705	000001	BIC	#1,R5		
118							
119							
120							
							; Allocate space for tables that keep track of space in swap file
121	012234	105737	000000G	TSTB	VSWPFL		; Is this a swapping system?
122	012240	001415		BEQ	14\$		; Br if not
123	012242	013700	000000G	MOV	VSWPSL,R0		; Get # slots in swap file
124	012246	006300		ASL	R0		; Allocate 2 bytes per slot
125	012250	010537	000000G	MOV	R5,SWPPDS		; Start of table with starting block #'s
126	012254	060005		ADD	R0,R5		; Allocate space
127	012256	010537	000000G	MOV	R5,SWPJOB		; Start of table with job #'s
128	012262	060005		ADD	R0,R5		; Allocate space
129	012264	010537	000000G	MOV	R5,SCPFHD		; Pointer to area with command packets
130	012270	062705	000000C	ADD	#NSCP*SP\$\$SZ,R5		; Allocate space for swap command packets
131							
132							
133							; Allocate a 512-byte buffer to use to access job context blocks
134	012274	010537	000000G	14\$: MOV	R5,CXTBUF		; Set address of buffer
135	012300	062705	001400	ADD	#1400,R5		; Reserve space for the buffer
136							
137							
138							; Make sure TSX is not too big.
139	012304	010500		MOV	R5,R0		; GET CURRENT MEMORY ADDRESS
140	012306	004737	025104'	CALL	CHKMEM		; CHECK FOR SPACE OVERFLOW
141							
142							
143							; Finished
144	012312	012601		MOV	(SP)+,R1		
145	012314	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 012316  010146  ALCSLO: MOV      R1,-(SP)
13 012320  010246      MOV      R2,-(SP)
14         ;
15         ; Begin loop to check each line
16         ;
17 012322  012701  000000G      MOV      #LSTHL,R1      ;Get index to last hardware line
18         ;
19         ; Only allocate silo buffers for real lines
20         ;
21 012326  012702  000040  1$:    MOV      #32.,R2      ;Set minimum size for time-sharing lines
22 012332  020127  000000G      CMP      R1,#LSTPL      ;Is this a primary line?
23 012336  101405      BLOS      2$      ;Br if yes
24 012340  020127  000000G      CMP      R1,#FSTIOL      ;Is this a CL line?
25 012344  103463      BLO      9$      ;Br if not
26 012346  012702  000020      MOV      #16.,R2      ;Set minimum size for CL lines
27         ;
28         ; Determine how much space to allocate
29         ;
30 012352  016100  000000G  2$:    MOV      LHIRBA(R1),R0      ;Get requested size
31 012356  001002      BNE      8$      ;Br if a size was specified
32 012360  013700  000000G      MOV      VNCSLO,R0      ;Use default value
33 012364  020027  000000G  8$:    CMP      R0,#MAXSLO      ;Constrain size to 255 bytes
34 012370  101402      BLOS      3$      ;Br if ok
35 012372  012700  000000G      MOV      #MAXSLO,R0      ;Reduce size
36 012376  020002  3$:    CMP      R0,R2      ;Compare with acceptable minimum
37 012400  103001      BHIS      4$      ;Br if ok
38 012402  010200      MOV      R2,R0      ;Use minimum size allowed
39 012404  010061  000000G  4$:    MOV      R0,LHIRBA(R1)      ;Set # bytes to be allocated for silo
40         ;
41         ; Allocate the space
42         ;
43 012410  010561  000000G      MOV      R5,LHIRBB(R1)      ;Set base address of buffer
44 012414  010561  000000G      MOV      R5,LHIRBP(R1)      ;Init pointer where to store next char
45 012420  010561  000000G      MOV      R5,LHIRBG(R1)      ;Init pointer where to get next char
46 012424  010061  000000G      MOV      R0,LHIRBS(R1)      ;Set # free bytes in buffer
47 012430  060005      ADD      R0,R5      ;Allocate space for buffer
48 012432  010561  000000G      MOV      R5,LHIRBE(R1)      ;Set address beyond end of buffer
49         ;
50         ; Set up control information about when to send XON and XOFF
51         ;
52 012436  006200      ASR      R0      ;Get 1/2 of buffer size
53 012440  162700  000002      SUB      #2,R0      ;Minus two characters
54 012444  116102  000000G      MOVB     LHIRBC(R1),R2      ;Get specified size for XOFF point
55 012450  001002      BNE      5$      ;Br if value specified
56 012452  113702  000000G      MOVB     VNCXOF,R2      ;Try default
57 012456  020200  5$:    CMP      R2,R0      ;Is specified value ok?

```

```

58 012460 101401          BLOS 6$          ;Br if yes
59 012462 010002          MOV  R0,R2        ;No, use size/2-2
60 012464 110261 000000G  6$:  MOVB R2,LHIRBC(R1) ;Set # of chars when XOFF sent
61 012470 116102 000001G  MOVB LHIRBC+1(R1),R2 ;Get specified size for XON point
62 012474 001002          BNE 7$          ;Br if a value was specified
63 012476 113702 000000G  MOVB VNCXON,R2    ;Try default
64 012502 020200          7$:  CMP  R2,R0    ;Is specified value ok?
65 012504 101401          BLOS 10$         ;Br if ok
66 012506 010002          MOV  R0,R2        ;No, use size/2-2
67 012510 110261 000001G  10$: MOVB R2,LHIRBC+1(R1) ;Set # of chars when XON sent
68
69      ; Do the next line
70
71 012514 162701 000002  9$:  SUB  #2,R1    ;Get next line index number
72 012520 003302          BGT  1$          ;Loop if more to do
73
74      ; Finished
75
76 012522 005205          INC  R5           ;Force R5 to be even
77 012524 042705 000001  BIC  #1,R5
78 012530 012602          MOV  (SP)+,R2
79 012532 012601          MOV  (SP)+,R1
80 012534 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 012536 010146 ALBFX:  MOV     R1, -(SP)
13 012540 010246      MOV     R2, -(SP)
14 012542 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 012544 012701 000002      MOV     #2, R1          ; GET 1ST JOB INDEX NUMBER
20 012550 032761 000000G 000000G 3$:  BIT     #$DEAD, LSW3(R1) ; IS THIS LINE INSTALLED?
21 012556 001047          BNE     2$              ; BR IF NOT -- DON'T ALLOCATE ANY BUFFER SPACE
22 012560 020127 000000G      CMP     R1, #FSTDL      ; IS THIS A DETACHED JOB LINE?
23 012564 103403          BLO     1$              ; BR IF NOT
24 012566 020127 000000G      CMP     R1, #LSTDL      ; DETACHED JOB LINE?
25 012572 101441          BLOS    2$              ; BR IF DETACHED JOB -- DON'T ALLOCATE BUFFERS
26 012574 010561 000000G 1$:  MOV     R5, LTPAR(R1)   ; SET PHYSICAL MEMORY PAR OFFSET FOR BUFFER
27 012600 012702 000000G      MOV     #VPAR6, R2      ; GET VIRTUAL MEMORY ADDRESS FOR BASE OF PAR6
28 012604 010261 000000G      MOV     R2, LINBUF(R1)  ; INPUT BUFFER STARTS AT BASE OF PAR6 REGION
29 012610 016100 000000G      MOV     LINSIZ(R1), R0  ; GET # BYTES FOR INPUT BUFFER
30 012614 010061 000000G      MOV     R0, LINSPC(R1)  ; SET # FREE BYTES IN INPUT BUFFER
31 012620 060002          ADD     R0, R2              ; ADVANCE VIRTUAL ADDRESS
32 012622 010261 000000G      MOV     R2, LINEND(R1)  ; POINTS PAST END OF INPUT BUFFER
33 012626 010003          MOV     R0, R3              ; Get # bytes in input buffer
34 012630 062703 000007      ADD     #7, R3           ; Bound up to multiple of 8
35 012634 072327 177775      ASH     #-3, R3          ; Get # bytes needed in activation-flag buffer
36 012640 060302          ADD     R3, R2              ; Reserve space for activation-flag buffer
37 012642 060300          ADD     R3, R0              ; Accumulate total buffer space
38 012644 010261 000000G      MOV     R2, LOTBUF(R1)  ; POINTS TO START OF OUTPUT BUFFER AREA
39 012650 066100 000000G      ADD     LOTSIZ(R1), R0  ; ACCUMULATE # BYTES IN BOTH BUFFERS
40 012654 066102 000000G      ADD     LOTSIZ(R1), R2  ; ADVANCE VIRTUAL ADDRESS COUNTER
41 012660 010261 000000G      MOV     R2, LOTEND(R1)  ; SAVE ADDRESS OF END OF OUTPUT BUFFER
42 012664 062700 000077      ADD     #77, R0         ; BOUND UP TO MULTIPLE OF 64 BYTES
43 012670 072027 177772      ASH     #-6, R0         ; CONVERT TO # 64-BYTE BLOCKS ALLOCATED
44 012674 060005          ADD     R0, R5              ; ADVANCE PHYSICAL MEMORY PAR ADDRESS
45 012676 062701 000002 2$:  ADD     #2, R1          ; ADVANCE LINE NUMBER
46 012702 020127 000000G      CMP     R1, #LSTSL      ; HAVE WE DONE ALL LINES YET?
47 012706 101720          BLOS    3$              ; BR IF MORE TO DO
48         ;
49         ; Allocate space for shared file record locking data structures
50         ;
51 012710 005737 000000G      TST     VMXSF           ; Shared file support wanted?
52 012714 001451          BEQ     5$              ; Br if not
53 012716 013737 000000G 000000G      MOV     VNUMDC, NUMDCD ; Set number of data cache blocks
54 012724 013737 000000G 000000G      MOV     VMXSFC, NUMCDB ; Set number of free CDB's
55 012732 010537 000000G      MOV     R5, LOKMEM      ; Set phys address of base of area
56 012736 012701 000000G      MOV     #FF$SZ, R1     ; Size of an FDB
57 012742 070137 000000G      MUL     VMXSF, R1       ; Times number of FDB's

```

ALBFX -- Allocate buffers in extended memory region

```

58 012746 062701 000000C      ADD      #<NLINES*FW*$SZ>,R1 ;Space needed for wait blocks
59 012752 013703 000000G      MOV      VMLBLK,R3      ;Max blocks a CDB may hold locked
60 012756 006303              ASL      R3              ;Two bytes per entry
61 012760 062703 000000G      ADD      #FC$LBN,R3      ;Add base size of a CDB
62 012764 070337 000000G      MUL      VMXSFC,R3      ;Times number of shared file channels
63 012770 060301              ADD      R3,R1          ;Accumulate space needed
64 012772 012703 000000G      MOV      #DC*$SZ,R3      ;Size of a data cache descriptor
65 012776 070337 000000G      MUL      VNUMDC,R3      ;Times number of data cache entries
66 013002 060301              ADD      R3,R1          ;Reserve space for data cache descriptors
67 013004 062701 000100      ADD      #64.,R1          ;Bound up to 64 byte unit
68 013010 072127 177772      ASH      #-6,R1          ;Convert to # 64 byte units
69 013014 042701 176000      BIC      #176000,R1        ;Clear sign extension
70 013020 060105              ADD      R1,R5          ;Reserve space for data structures
71 013022 010537 000000G      MOV      R5,LOKCSH      ;Save pointer to start of cache buffer area
72 013026 013701 000000G      MOV      VNUMDC,R1      ;# shared-file data cache blocks wanted
73 013032 070127 000010      MUL      #8.,R1          ;8 64-byte blocks each (512 bytes each)
74 013036 060105              ADD      R1,R5          ;Reserve space for data cache buffers
75                               ;
76                               ;   Allocate space for mapped I/O buffers
77                               ;
78 013040 105737 000000G      5$:      TSTB      MIOFLG      ;Are any mapped I/O buffers needed?
79 013044 001415              BEQ      7$              ;Br if not
80 013046 013701 000000G      MOV      MIOBHD,R1        ;Point to 1st mapped I/O control block
81 013052 001412              BEQ      7$              ;Br if no more buffers needed
82 013054 010561 000000G      6$:      MOV      R5,MI$SBP(R1) ;Set address of base of buffer
83 013060 113703 000000G      MOV      VMIO$Z,R3      ;Get # blocks for buffer
84 013064 072327 000003      ASH      #3,R3          ;Convert to # 64-byte pages
85 013070 060305              ADD      R3,R5          ;Allocate space for buffer
86 013072 016101 000000G      MOV      MI$LNK(R1),R1    ;Get address of next control block
87 013076 001366              BNE      6$              ;Loop if more to allocate
88                               ;
89                               ;   Allocate space for performance monitor data buffer if it is wanted.
90                               ;
91 013100 013701 000000G      7$:      MOV      VPMSIZ,R1      ;DID USER GEN IN PERFORMANCE MONITOR FEATURE?
92 013104 001412              BEQ      9$              ;BR IF NOT
93 013106 006201              ASR      R1              ;CONVERT BYTES TO WORDS
94 013110 010137 000000G      MOV      R1,PMCELS      ;SET INTO CELL IN TSEXEC
95 013114 010537 000000G      MOV      R5,PMPAR      ;SET BASE ADDRESS OF PM BUFFER
96 013120 062701 000037      ADD      #37,R1          ;BOUND SIZE UP TO 64-BYTE MULTIPLE
97 013124 072127 177773      ASH      #-5,R1          ;CONVERT WORDS TO # 64-BYTE BLOCKS
98 013130 060105              ADD      R1,R5          ;ADVANCE FREE MEMORY POINTER
99                               ;
100                               ;   Finished
101                               ;
102 013132 012603      9$:      MOV      (SP)+,R3
103 013134 012602      MOV      (SP)+,R2
104 013136 012601      MOV      (SP)+,R1
105 013140 000207      RETURN

```

OPNKMN -- Open channel to TSKMON

```

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 013142 010246 OPNKMN: MOV      R2, -(SP)
18         ;
19         ; Lookup TSKMON file.
20         ;
21 013144         .LOOKUP #AREA, #1, #KMNNAM ; TRY TO FILE KMON SAV FILE
22 013164 103517     BCS      9$           ; BR IF NOT THERE
23         ;
24         ; Read block 0 of save file and extract some information.
25         ;
26 013166 013702 000152' MOV      WRKBUF, R2           ; Point to work buffer
27 013172         .READW #AREA, #1, R2, #256., #0 ; READ BLOCK 0 OF SAV FILE
28         ;
29         ; Determine size of kmon
30 013226 016200 000050 MOV      50(R2), R0           ; GET TOP ADDRESS OF KMON
31 013232 062700 000003 ADD      #3, R0           ; BOUND UP TO NEXT WORD
32 013236 042700 000001 BIC      #1, R0           ; FORCE EVEN
33 013242 010037 000000G MOV      R0, KMNTOP
34         ; Determine number of 256-word memory pages needed while kmon running.
35 013246 162700 000000G SUB      #KMNBAS, R0           ; BASE ADDRESS OF KMON
36 013252 010037 000000G MOV      R0, KMNHI           ; TOP OF TSKMON - KMNBAS
37 013256 062700 000777 ADD      #511., R0           ; BOUND UP TO PAGE SIZE
38 013262 000241 CLC
39 013264 006000 ROR      R0           ; CVT TO # WORDS
40 013266 000300 SWAB     R0           ; CVT TO # PAGES
41 013270 042700 177400 BIC      #^C377, R0
42 013274 063700 000000G ADD      CXTAG, R0           ; # PAGES NEEDED FOR JOB CONTEXT AREA
43 013300 010037 000000G MOV      R0, KMNPGS           ; # 256-wd pages needed for kmon + context area
44         ; Determine Kmon stack pointer.
45 013304 016237 000042 000000G MOV      42(R2), KMNSTK           ; INITIAL STACK POINTER FOR KMON
46         ; Determine Kmon starting address.
47 013312 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 013320 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 013326 012700 000000G MOV      #KMNCHN, R0           ; GET KMON CHANNEL SAVE AREA
57 013332 013702 000072' MOV      KMNNAM, R2           ; GET KMON DEVICE NAME

```

58 013336 004737 024410'	CALL	SETCHN	;SAVE CHANNEL STATUS
59			
60			; Lookup CCL.SAV and save channel status for it.
61			
62 013342		. LOOKUP	#AREA,#1,#CCLNAM;LOOKUP SY:CCL.SAV
63 013362 103410	BCS	B\$	;BR IF CAN'T FIND CCL
64 013364 012700 000000G	MOV	#CCLSAV,R0	;CHANNEL SAVE AREA
65 013370 013702 000102'	MOV	CCLNAM,R2	;DEVICE NAME
66 013374 004737 024410'	CALL	SETCHN	;SAVE CHANNEL STATUS
67			
68			; Finished
69			
70 013400 012602	10\$:	MOV	(SP)+,R2
71 013402 000207		RETURN	
72			
73			; Error: We could not find SY:CCL.SAV
74			
75 013404	B\$:	. PRINT	#TSXHD
76 013412		. PRINT	#NOCCL ;PRINT ERROR MESSAGE
77 013420 000137 004134'		JMP	INISTP ;ABORT INITIALIZATION
78			
79			; Error: We could not locate TSKMON SAV file.
80			
81 013424	9\$:	. PRINT	#TSXHD ;PRINT ERROR MESSAGE
82 013432		. PRINT	#NOKMON
83 013440 000137 004134'		JMP	INISTP ;ABORT INITIALIZATION

```

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 013444 010146 CLINIT: MOV     R1,-(SP)
12 013446 010246      MOV     R2,-(SP)
13 013450 010346      MOV     R3,-(SP)
14         ;
15         ; Initialize tables for each CL unit
16         ;
17 013452 005003      CLR      R3          ;Accumulate ring buffer sizes in R3
18 013454 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 013460 016102 000000G 1$:      MOV     CL$LIX(R1),R2  ;Is this CL unit associated with a line?
24 013464 001416          BEQ      5$          ;Br if not
25 013466 012762 000000G 000000G      MOV     #$SXON,LSW10(R2);Send XON when we start the line
26 013474 010162 000000G      MOV     R1,LCLUNT(R2)  ;Associate the CL unit with this line
27 013500 005762 000000G      TST      RSR(R2)       ;Does this unit have a specified RSR addr?
28 013504 001006          BNE      5$          ;Br if yes
29 013506 005762 000000G      TST      LMXNUM(R2)    ;Is this a mux line?
30 013512 001003          BNE      5$          ;Br if yes
31 013514 005061 000000G      CLR      CL$EPS(R1)    ;Say no endstring buffer
32 013520 000472          BR       4$          ;Line is not genned in
33         ;
34         ; Allocate and set up pointers for the output ring buffers
35         ;
36 013522 010561 000000G 5$:      MOV     R5,CL$ORB(R1)  ;Start of output ring buffer
37 013526 010561 000000G      MOV     R5,CL$ORP(R1)  ;Input character pointer
38 013532 010561 000000G      MOV     R5,CL$ORG(R1)  ;Next available character pointer
39 013536 012700 000000G      MOV     #CLORSZ,R0     ;Get default output ring buffer size
40 013542 005702          TST      R2              ;Is this CL unit connected to a line?
41 013544 001402          BEQ      6$          ;Br if not
42 013546 016200 000000G      MOV     LOTSIZ(R2),R0  ;Get size of output ring buffer
43 013552 010061 000000G 6$:      MOV     R0,CL$ORA(R1) ;Set size of output ring buffer
44 013556 010061 000000G      MOV     R0,CL$ORS(R1)  ;Available space in ring buffer
45 013562 060005          ADD      R0,R5            ;Point beyond end of ring buffer
46 013564 010561 000000G      MOV     R5,CL$ORE(R1)  ;Address past end of ring buffer
47 013570 060003          ADD      R0,R3            ;Accumulate size of output ring buffers
48         ;
49         ; Allocate space for end-of-file string buffer
50         ;
51 013572 010561 000000G      MOV     R5,CL$EPS(R1)  ;Set pointer to end-of-file string buffer
52 013576 005061 000000G      CLR      CL$EPP(R1)    ;No string to print yet
53 013602 062705 000001G      ADD      #<CLEDFS+1>,R5 ;Reserve space for buffer
54 013606 062703 000001G      ADD      #<CLEDFS+1>,R3 ;Accumulate buffer space
55         ;
56         ; Initialize end-of-file form-feed count
57         ;

```

CLINIT -- Initialize CL handler

```

58 013612 005061 000000G          CLR      CL$EPN(R1)      ; Init ENDPAGE=0
59                                     ;
60                                     ; Initialize option word
61                                     ;
62 013616 012700 000000G          MOV      #<CD$DEF>,R0      ; Get default option flags
63 013622 005702                   TST      R2              ; Is this CL unit connected with a line?
64 013624 001421                   BEQ      7$              ; Br if not
65 013626 016202 000000G          MOV      ILSW2(R2),R2      ; Get line options
66 013632 032702 000000G          BIT      #$TAB,R2         ; Does hardware support tabs?
67 013636 001402                   BEQ      2$              ; Br if not
68 013640 052700 000000G          BIS      #CD$TAB,R0        ; Set hardware-tab flag
69 013644 032702 000000G          2$: BIT      #$FORM,R2     ; Does hardware support form feeds?
70 013650 001402                   BEQ      3$              ; Br if not
71 013652 052700 000000G          BIS      #CD$FF,R0        ; Set hardware-form-feed flag
72 013656 032702 000000G          3$: BIT      #$BBIT,R2    ; Does hardware want 8 bit support?
73 013662 001402                   BEQ      7$              ; Br if not
74 013664 052700 000000G          BIS      #CD$BBT,R0       ; Enable 8 bit support for CL line
75 013670 010061 000000G          7$: MOV      R0,CL$OPT(R1) ; Set options for this CL line
76                                     ;
77                                     ; Initialize page length
78                                     ;
79 013674 012761 000102 000000G    MOV      #66.,CL$LEN(R1) ; Say page length = 66 lines
80                                     ;
81                                     ; Initialize status flags
82                                     ;
83 013702 005061 000000G          CLR      CL$STA(R1)       ; Initialize status flags
84                                     ;
85                                     ; Do next line
86                                     ;
87 013706 162701 000002           4$: SUB      #2,R1         ; Get index # of next unit
88 013712 002262                   BGE      1$              ; Loop if more units to initialize
89                                     ;
90                                     ; Make a device table entry for "CL" device
91                                     ;
92 013714 062737 000002 000000G    ADD      #2,NUMDEV       ; One more device
93 013722 013701 000000G          MOV      NUMDEV,R1        ; Get device table index
94 013726 010137 000000G          MOV      R1,CLDEVX        ; Remember index number of CL device
95 013732 012761 012240 000000G    MOV      #R5OCL,PNAME(R1) ; Set device name ("CL")
96 013740 012761 000000G 000000G    MOV      #CLSTS,DVSTAT(R1) ; Set dev status flags
97 013746 012761 000000C 000000G    MOV      #<DX$NCA!DX$NMT!DX$NRD>,DVFLAG(R1) ; Device info flags
98 013754 005061 000000G          CLR      DEVSIZ(R1)       ; Clear device size
99 013760 012761 000006G 000000G    MOV      #CLHEAD+6,HANENT(R1) ; Set handler entry point (4th word)
100 013766 062703 000000G          ADD      #CLSIZE,R3       ; Get size of handler
101 013772 062703 000000C          ADD      #<<CLTOTL*46.>+<NIOL*32.>,R3 ; Add size of tables in TSGEN
102 013776 010361 000000G          MOV      R3,HANSIZ(R1)    ; Set size of handler
103 014002 005205                   INC      R5              ; Make sure free-memory pointer is even
104 014004 042705 000001          BIC      #1,R5
105                                     ;
106                                     ; Make a device table entry for C1 if there are more than 8 CL units
107                                     ;
108 014010 022727 000000G 000010    CMP      #CLTOTL,#8.     ; Are there more than 8 CL units?
109 014016 101430                   BLOS     9$              ; Br if not -- Don't need C1
110 014020 062737 000002 000000G    ADD      #2,NUMDEV       ; One more device
111 014026 013701 000000G          MOV      NUMDEV,R1        ; Get device table index
112 014032 010137 000000G          MOV      R1,C1DEVX        ; Remember index number of CL device
113 014036 012761 013630 000000G    MOV      #R5OC1,PNAME(R1) ; Set device name ("C1")
114 014044 012761 000000G 000000G    MOV      #CLSTS,DVSTAT(R1) ; Set dev status flags

```

CLINIT -- Initialize CL handler

```
115 014052 012761 000000C 000000G      MOV    #<DX$NCA!DX$NMT!DX$NRD>,DVFLAG(R1) ;Device info flags
116 014060 005061 000000G      CLR     DEVSIZ(R1)      ;Clear device size
117 014064 012761 000006G 000000G      MOV    #CLHEAD+6,HANENT(R1) ;Set handler entry point (4th word)
118 014072 012761 000004 000000G      MOV    #4.,HANSIZ(R1) ;Set size of handler
119                                     ;
120                                     ; Finished
121                                     ;
122 014100 012603      9$:      MOV    (SP)+,R3
123 014102 012602      MOV    (SP)+,R2
124 014104 012601      MOV    (SP)+,R1
125 014106 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 014110   010246   INDINI: MOV      R2,-(SP)
13 014112   010346           MOV      R3,-(SP)
14         ;
15         ; Determine if IND support is wanted
16         ;
17 014114   005037   000000G      CLR      INDSAV      ; ASSUME IND SUPPORT NOT WANTED
18         ;
19         ; Lookup SY:IND.SAV file
20         ;
21 014120   013737   000000G 000220'  MOV      SYNAME,INDNAM ; LOOK UP IND ON BOOT DEVICE
22 014126           .LOOKUP #AREA,#1,#INDNAM ; TRY TO FIND SY:IND.SAV
23 014146   103002           BCC      4$      ; BR IF FOUND IND
24 014150   000137   014602'      JMP      9$      ; IF CAN'T FIND IND, THEN NO IND SUPPORT
25         ;
26         ; Set up information about IND overlay data segment
27         ;
28 014154   013703   000152'  4$:  MOV      WRKBUF,R3      ; Get pointer to work buffer
29 014160           .READW #AREA,#1,R3,#256.,#0 ; READ IN BLOCK 0 OF SAV FILE
30 014214   016302   000064      MOV      64(R3),R2      ; GET POINTER TO OVERLAY TABLE
31 014220           .READW #AREA,#1,R3,#256.,#1 ; READ IN BLOCK 1 WITH OVERLAY TABLE
32 014256   162702   001000      SUB      #1000,R2      ; GET ADDRESS OF OVERLAY TABLE REL TO BLOCK 1
33 014262   060302           ADD      R3,R2      ; ADD BASE ADDRESS WHERE BLOCK 1 DATA IS
34 014264   011203           MOV      (R2),R3      ; Get virtual address of segment 0
35 014266   020312           5$:  CMP      R3,(R2)      ; Search for 1st segment with different addr
36 014270   001003           BNE      6$      ; Br if found it (this is the data segment)
37 014272   062702   000006      ADD      #6,R2      ; Point to overlay table entry for next seg
38 014276   000773           BR      5$
39 014300   005722           6$:  TST      (R2)+      ; Point to word with block # if SAV file
40 014302   012237   000000G      MOV      (R2)+,INDDBL ; GET BLOCK # IN SAV FILE OF DATA OVERLAY
41 014306   011202           MOV      (R2),R2      ; GET # OF WORDS IN OVERLAY SEGMENT
42 014310   062702   000377      ADD      #255.,R2      ; ROUND UP TO NEXT BLOCK
43 014314   000302           SWAB     R2      ; CONVERT # WORDS TO # BLOCKS
44 014316   042702   177400      BIC      #^C<377>,R2
45 014322   010237   000000G      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 014326   012700   000000G      MOV      #INDSAV,R0      ; GET ADDRESS OF SAVESTATUS BLOCK
51 014332   013702   000220'      MOV      INDNAM,R2      ; GET RAD50 DEVICE NAME
52 014336   004737   024410'      CALL     SETCHN      ; SAVE FILE STATUS
53         ;
54         ; Determine how much space is needed for SY:TSXIND.TSX swap file
55         ;
56 014342   013703   000000G      MOV      INDDBS,R3      ; GET # BLOCKS NEEDED PER JOB
57 014346   070327   000000C      MUL      #<LSTSL/2>,R3 ; TIMES TOTAL NUMBER OF JOBS

```

```

58
59
60 ; Load Rt-11 device handler for ind swap file.
61 ;
62 014352 013700 000000G      MOV      INDFIL,R0      ;Get name of the device
63 014356 004737 024770'      CALL      RTFTCH      ;Try to fetch the RT-11 device handler
64 014362 103522              BCS      11$      ;Br if error on handler fetch
65
66 ; See if TSXIND file already exists
67 ;
68 014364              .LOOKUP #AREA,#1,#INDFIL ; DOES SY:TSXIND.TSX FILE EXIST NOW?
69 014404 103415              BCS      1$      ;BR IF NOT
70 014406 020003              CMP      R0,R3      ;IS IT OF THE CORRECT SIZE?
71 014410 001462              BEQ      2$      ;BR IF YES
72 014412              .PURGE #1      ;FILE IS OF WRONG SIZE
73 014420              .DELETE #AREA,#1,#INDFIL;DELETE OLD FILE
74
75 ; File does not now exist
76 ; Create new file
77 ;
78 014440      1$:      .ENTER #AREA,#1,#INDFIL,R3 ;CREATE NEW TSXIND FILE
79 014464 103451              BCS      10$      ;BR IF ERROR ON CREATE
80 014466 010302              MOV      R3,R2      ;# BLOCKS IN FILE
81 014470 005302              DEC      R2      ;GET # OF LAST BLOCK IN FILE
82 014472              .WRITW #AREA,#1,WRKBUF,#256.,R2 ;WRITE TO LAST BLOCK OF FILE
83 014530              .CLOSE #1      ;NOW CLOSE THE FILE
84 014536              .LOOKUP #AREA,#1,#INDFIL ;REOPEN TSXIND FILE WITH LOOKUP
85
86 ; Do .SAVESTATUS for SY:TSXIND.TSX file
87 ;
88 014556 012700 000000G      2$:      MOV      #INDTSV,R0      ;POINT TO SAVESTATUS BLOCK
89 014562 013702 000000G      MOV      INDFIL,R2      ;GET RAD50 DEVICE NAME
90 014566 004737 024410'      CALL      SETCHN      ;SAVE FILE INFO
91 014572              .RELEAS #INDFIL      ;Release device handler
92
93 ; Finished
94 ;
95 014602 012603      9$:      MOV      (SP)+,R3
96 014604 012602      MOV      (SP)+,R2
97 014606 000207      RETURN
98
99 ; Error occurred while opening SY:TSXIND.TSX file
100 ;
101 014610      10$:      .PRINT #TSXHD      ;Print error message
102 014616      .PRINT #INDOPN
103 014624 004737 010142'      CALL      SPNEED      ;Print info about number of blocks needed
104
105 ; Error: Invalid device specification.
106 ;
107 014630 010001      11$:      MOV      R0,R1      ;Save device name
108 014632      .PRINT #TSXHD      ;Print error message
109 014640      .PRINT #INDOPN
110 014646 004737 010170'      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         014652  010246
11         014654  010346
12
13         ; Determine if TSXUCL data file is needed
14         ;
15         014656  105737  000000G      TSTB    VU$CL          ; Is TSXUCL being used at all?
16         014662  001536              BEQ      9$             ; Br if not
17         014664  013702  000000G      MOV      VUCLMC,R2      ; Get maximum number of commands
18         014670  001533              BEQ      9$             ; Br if none allowed
19
20         ; Determine number of blocks needed in data file for each job
21         ;
22         014672  012700  000000G      MOV      #UK$$SZ,R0      ; Size of each keyword descriptor
23         014676  062700  000000G      ADD      #US$$SZ,R0      ; Size of each command string descriptor
24         014702  070200              MUL      R0,R2            ; Compute total # bytes for keywords+commands
25         014704  062703  000777G      ADD      #UC$$SZ+511.,R3 ; Add space for control information & round up
26         014710  005502              ADC      R2              ; Propagate carry
27         014712  071227  001000      DIV      #512.,R2        ; Convert to # of blocks needed
28         014716  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         014722  070227  000000C      MUL      #<LSTSL/2>,R2    ; Times total number of jobs
33
34         ; Load Rt-11 device handler for ind swap file.
35         ;
36         014726  013700  000000G      MOV      UCLDAT,R0      ; Get name of the device
37         014732  004737  024770'      CALL     RTFTCH        ; Try to fetch the RT-11 device handler
38         014736  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         014740          .LOOKUP #AREA,#1,#UCLDAT ; See if the file exists now
44         014760  103415          BCS      1$              ; Br if file does not exist
45         014762  020003          CMP      R0,R3           ; Is existing file of correct size?
46         014764  001446          BEQ      2$              ; Br if yes -- use the old file
47         014766          .PURGE #1                  ; Purge the channel
48         014774          .DELETE #AREA,#1,#UCLDAT ; Delete the old file
49
50         ; Create a new data file
51         ;
52         015014          1$: .ENTER #AREA,#1,#UCLDAT,R3 ; Create new data file
53         015040  103452          BCS      10$             ; Br if error creating the file
54         015042  005303          DEC      R3              ; Get # of last block in the file
55         015044          .WRITW #AREA,#1,WRKBUF,#256.,R3 ; Write to last block of file
56
57         ; Translate possible logical device name to physical name and close

```

```

58      ; (Physical name is needed for TSXUCL program.)
59      ;
60 015102      2$:      .CSTAT  #AREA,#1,#NFSBLK      ;GET CHANNEL STATUS INFORMATION
61 015122      013702 000032'      MOV      <NFSBLK+12>,R2 ;FETCH DEVICE NAME IN RAD50
62 015126      063702 000030'      ADD      <NFSBLK+10>,R2 ;ADD IN DEVICE UNIT NUMBER
63 015132      062702 000036      ADD      #^R 0,R2 ;CONVERT UNIT NUMBER TO RAD50
64 015136      010237 000000G      MOV      R2,UCLDAT ;SET PHYSICAL NAME BACK INTO TSGEN CELL
65 015142      .CLOSE #1 ;Close the file
66 015150      .RELEAS #UCLDAT ;Release device handler
67      ;
68      ; Finished
69      ;
70 015160      012603      9$:      MOV      (SP)+,R3
71 015162      012602      MOV      (SP)+,R2
72 015164      000207      RETURN
73      ;
74      ; Error creating the data file
75      ;
76 015166      10$:      .PRINT  #TSXHD ;Print error message
77 015174      .PRINT  #UCLOPN
78 015202      004737 010142'      CALL      SPNEED ;Print info about number of blocks needed
79      ;
80      ; Error: Invalid device specification.
81      ;
82 015206      010001      11$:      MOV      R0,R1 ;Save device name
83 015210      .PRINT  #TSXHD ;Print error message
84 015216      .PRINT  #UCLOPN
85 015224      004737 010170'      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 015230 010146 MEMINI: MOV      R1, -(SP)
7 015232 010246      MOV      R2, -(SP)
8 015234 010346      MOV      R3, -(SP)
9 015236 010446      MOV      R4, -(SP)
10 015240 010546      MOV      R5, -(SP)
11                                     ;
12                                     ; Initialize all pages for a 1-to-1 mapping.
13                                     ;
14 015242 012700 000000G 12$:  MOV      #KPAR0, R0      ; Kernel mode PAR 0
15 015246 012701 000000G      MOV      #UPAR0, R1      ; User mode PAR 0
16 015252 012702 000000G      MOV      #KPDR0, R2      ; Kernel mode PDR 0
17 015256 012703 000000G      MOV      #UPDR0, R3      ; User mode PDR 0
18 015262 012704 000010      MOV      #8, R4           ; Initialize 8 pages
19 015266 005005      CLR      R5                     ; Set initial PAR value
20 015270 010520 2$:  MOV      R5, (R0)+              ; Set kernel PAR
21 015272 010521      MOV      R5, (R1)+              ; Set user PAR value
22 015274 012722 077406      MOV      #077406, (R2)+  ; Set kernel PDR
23 015300 012723 077406      MOV      #077406, (R3)+  ; Set user PDR value
24 015304 062705 000200      ADD      #200, R5         ; Advance block number
25 015310 077411      SOB      R4, 2$                 ; Init all pages
26                                     ;
27                                     ; Map kernel mode I/O page (160000) to 17760000.
28                                     ;
29 015312 012737 000000G 000000G      MOV      #IOPAGE, @#KPAR7 ; Map I/O page
30                                     ;
31                                     ; Finished
32                                     ;
33 015320 012605      MOV      (SP)+, R5
34 015322 012604      MOV      (SP)+, R4
35 015324 012603      MOV      (SP)+, R3
36 015326 012602      MOV      (SP)+, R2
37 015330 012601      MOV      (SP)+, R1
38 015332 000207      RETURN

```

```

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 015334 010146  MEMTST: MOV      R1,-(SP)
25 015336 010246      MOV      R2,-(SP)
26 015340 010346      MOV      R3,-(SP)
27 015342 010446      MOV      R4,-(SP)
28 015344 010546      MOV      R5,-(SP)
29 015346 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 015352 012737 015606' 000004      MOV      #TRCSET,@#4      ; Catch trap
35 015360 000240      NOP                      ; Clean out 11/73 pipeline
36 015362 000240      NOP                      ; Before attempting trap
37 015364 005737 000000G      TST      @#SR3MMR      ; Try to access status register 3
38 015370 103411      BCS      22$      ; Br if MMU 3 status register is non-existent
39 015372 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 015376 000240      NOP                      ; Clean out the pipeline again?
45 015400 000240      NOP
46 015402 005737 000000G      TST      @#UDDRO      ; It there a user D-space PDRO?
47 015406 103402      BCS      22$      ; Br if not
48 015410 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 015414      22$:
54     .IF      NE,PROASM
55 015414 105737 000000G      TSTB     PROFLG      ; Are we running on a Professional?
56 015420 001410      BEQ      26$      ; Br if not
57 015422 005005      CLR      R5      ; Load byte without sign extension

```

MEMTST -- Set up information about available memory space

```

58 015424 153705 173050      BISB    @#173050,R5      ;Get 32Kb top of system RAM boundary
59 015430 072527 000011      ASH      #9.,R5        ;Convert to # 64 byte blocks
60 015434 162705 000010      SUB      #10,R5        ;Don't use the last 512 bytes of memory
61 015440 000400              BR        7$
62 015442      26$:
63      .ENDC      ;NE,PROASM
64      .IF        NE,<PROASM-1>      ; Assemble if could be on a PDP-11
65
66      ; We are not running on a Professional.
67      ; Test each page above TSX to see where the top of memory is.
68
69      MOV        #RTNKM,@#20      ;Use IOT instruction to get out of user mode
70      CLR        @#22
71      BIS        #MMENBL,@#SR0MMR;Enable memory management
72      TSTB       SR3FLG           ;Does maching have mem management reg # 3?
73      BEQ        4$              ;Br if non-existent
74      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 4$:      MOV        #1024.,R5      ;Start checking at 64Kb
79 5$:      MOV        R5,@#UPAR7     ;Map user page 7 to page to be tested
80      BIS        #UMODE,@#PSW      ;Go into user mode
81      TST        @#160000          ;Can we access the page?
82
83      ; Use IOT to get back into kernel mode.
84
85      IOT          ;Return to kernel mode
86      BCS        6$              ;Br if memory is non-existent
87      ADD        #10,R5           ;Go try next page
88      CMP        R5,#177600        ;Don't enter I/O page
89      BLO        5$
90
91      ; Check for potential memory wrap (on 18-bit 256K byte computers).
92
93 6$:      CMP        R5,#10000      ;Is physical memory above 256K bytes
94      BLOS       7$              ;Br if below 256K bytes
95      CLR        @#TSTWRD         ;Clear physical location
96      MOV        #10000,@#UPAR7    ;Map to 256K byte boundary
97      BIS        #UMODE,@#PSW      ;Go into user mode
98      MOV        #-1,@#160000+TSTWRD ;Store -1 at 256K physical location
99      IOT          ;Return to kernel mode
100      TST        @#TSTWRD         ;Test physical location
101      BEQ        7$              ;Br if physical location is clear
102      MOV        #10000,R5         ;Constrain memory to 256K byte total
103      .ENDC      ;NE,<PROASM-1>
104
105      ; Reached end of available memory.
106
107 7$:      MOV        R5,PHYMEM      ;set physical memory size
108      CMP        R5,MAPSIZ         ;Constrain kernel to user specified cutoff
109      BLOS       8$              ;Br if below user specified
110      MOV        MAPSIZ,R5         ;Only use this much memory
111 8$:      MOV        R5,$MEMSZ      ;Set # 64-byte blocks of total memory
112      MOV        R5,FMEMHI         ;Save base 64-byte block # of top of free mem
113
114      ; Turn off memory management

```

```

115
116 015470 105737 000000G          TSTB    SR3FLG          ; Do we have memory management req # 3?
117 015474 001403          BEQ      9$                ; Br if non-existent
118 015476 042737 000000G 000000G    BIC      #EMMAP, @#SR3MMR ; Disable extended memory management
119 015504 042737 000000G 000000G 9$: BIC      #MMENBL, @#SR0MMR ; Turn off memory management
120
121          ; If this is a Q-bus machine with >256Kb then set EXTLSI flag in ICONFG
122          ;
123 015512 023727 000134' 010000      CMP      FMEMHI, #4096.    ; Does machine have at least 256Kb?
124 015520 103411          BLO      25$                ; Br if not
125 015522 105237 000000G          INCB     MEM256          ; Remember machine has at least 256kb
126 015526 123727 000000G 000000G    CMPB     VBUSTP, #QBUS    ; Is this a Q-bus machine?
127 015534 001003          BNE      25$                ; Br if not
128 015536 052737 000001 000242'     BIS      #EXTLSI, ICONFG ; Set extended-LSI flag in ICONFG
129
130          ; See if this machine needs UNIBUS mapping
131          ;
132 015544 123727 000000G 000000G 25$: CMPB     VBUSTP, #UNIBUS ; Is this a UNIBUS machine?
133 015552 001005          BNE      29$                ; Br if not
134 015554 105737 000000G          TSTB     MEM256          ; Does machine have at least 256kb of memory?
135 015560 001402          BEQ      29$                ; Br if not
136 015562 105237 000000G          INCB     UBUSMP          ; Say UNIBUS mapping is needed
137
138          ; Finished
139          ;
140 015566 012637 000004          29$: MOV      (SP)+, @#4      ; Reset trap vector
141 015572 012605          MOV      (SP)+, R5
142 015574 012604          MOV      (SP)+, R4
143 015576 012603          MOV      (SP)+, R3
144 015600 012602          MOV      (SP)+, R2
145 015602 012601          MOV      (SP)+, R1
146 015604 000207          RETURN
147
148          ; Trap - return with C-bit set.
149          ;
150 015606 052766 000001 000002 TRCSET: BIS      #1, 2(SP)    ; Set c-bit for return
151 015614 000002          RTI                ; Return from trap
152
153          ; IOT - return at kernel mode with c-bit preserved.
154          ;
155 015616 042766 000000G 000002 RTNKM: BIC      #UMODE, 2(SP) ; Clear user mode - return to kernel
156 015624 000002          RTI                ; Return from trap
157
158          ; Error: System does not have memory management hardware.
159          ;
160 015626          NOXM:  .PRINT    #TSXHD          ; PRINT ERROR MESSAGE
161 015634          .PRINT    #NXMMMSG
162 015642 000137 004134'          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 015646   CXTALC:
13         ;
14         ; Get size of base portion of job context area
15         ;
16 015646   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 015652   062700   000077      ADD      #63.,RO      ;Bound up to 64 byte boundary
22 015656   042700   000077      BIC      #77,RO
23 015662   010037   000000G      MOV      RO,CXTRMN      ;Offset to start of simulated RMON
24 015666   062737   000000G 000000G      ADD      #CXTBAS,CXTRMN ;Add base virtual address of context area
25 015674   062700   000001G      ADD      #MVSIZ+1,RO    ;Add space for simulated RMON & channels
26         ;
27         ; Save number of words needed for context area
28         ;
29 015700   006200           ASR      RO      ;Convert to # words
30 015702   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 015706   062700   000037      ADD      #31.,RO      ;Bound up to # 32 word units
35 015712   072027   177773      ASH      #-5.,RO      ;Get # 32-word units for context area
36 015716   000300           SWAB      RO      ;Put # 32-word units in high-order byte
37 015720   052700   000006      BIS      #6,RO      ;Set PDR control flags
38 015724   010037   000000G      MOV      RO,CXTPDR      ;This is the PDR value
39         ;
40         ; Compute # 512-byte pages needed for job context block
41         ;
42 015730   013700   000000G      MOV      CXTWDS,RO      ;Get back # words for context area
43 015734   062700   000377      ADD      #255.,RO      ;Bound up to # 256-word blocks
44 015740   072027   177770      ASH      #-8.,RO      ;Get # 256-word pages for context area
45 015744   010037   000000G      MOV      RO,CXTPAG      ;# pages for job context area
46         ;
47         ; Set up PDR value used when mapping to simulated RMON
48         ;
49 015750   012700   000000G      MOV      #MVSIZ,RO      ;Get size of monitor vector table
50 015754   062700   000077      ADD      #63.,RO      ;Round up to # 32 word blocks
51 015760   072027   177772      ASH      #-6,RO      ;Cvt to # 32-word blocks
52 015764   005300           DEC      RO      ;Get # blocks - 1
53 015766   000300           SWAB      RO      ;Put # blocks in left byte
54 015770   052700   000006      BIS      #6,RO      ;Allow read and write access
55 015774   042700   100261      BIC      #100261,RO      ;Make sure unused PDR bits are zero
56 016000   010037   000000G      MOV      RO,RMNPDR      ;This is PDR value to map to sim. RMON
57

```

TSINIT -- TSX startup initializ MACRD V05.05 Friday 20-Jan-89 11:30 Page 47-1
CXTALC -- Set up info about job context area

58 ; Finished
59 ;
60 016004 000207 RETURN

```

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 016006 010246 MAPALC: MOV      R2,-(SP)
28 016010 010346      MOV      R3,-(SP)
29 016012 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 016014 010503      MOV      R5,R3          ;Get 64-byte block # of top of free mem
35 016016 072327 177775  ASH      #-3,R3      ;Convert to 512-byte page #
36 016022 042703 160000  BIC      #160000,R3   ;Kill possible sign extension
37 016026 013702 000136' MOV      FMEMLO,R2      ;Get 64-byte block # of base of free memory
38 016032 062702 000007  ADD      #7,R2       ;Round up
39 016036 072227 177775  ASH      #-3,R2      ;Convert to 512-byte page #
40 016042 042702 160000  BIC      #160000,R2   ;Kill possible sign extension
41 016046 160203      SUB      R2,R3          ;Get # bytes needed for allocation table
42 016050 003440      BLE      10$           ;Br if memory overflow
43 016052 010304      MOV      R3,R4          ;Get # bytes for allocation table
44 016054 062704 001000  ADD      #512.,R4     ;Add 1 extra byte and round up to 512-byte
45 016060 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 016064 012700 000000G  MOV      #VPAR5,R0      ;We will map to alloc table through PAR 5
50 016070 010037 000000G  MOV      R0,LOMAP    ;Pointer to 1st entry in alloc table
51 016074 160200      SUB      R2,R0          ;Get pseudo virtual address for page # 0
52 016076 010037 000000G  MOV      R0,BASMAP    ;This would point to alloc entry for page 0
53 016102 012700 000000G  MOV      #VPAR5,R0      ;Get back base address of table
54 016106 060300      ADD      R3,R0          ;Add # bytes used by table
55 016110 160400      SUB      R4,R0          ;Subtract space used by table itself
56 016112 010037 000000G  MOV      R0,HIMAP    ;Virtual address of 1st entry for system page
57

```

```

58          ; Allocate space for the allocation table
59          ;
60 016116 072427 000003          ASH      #3,R4          ;Get # 64-byte units for alloc table
61 016122 160405          SUB      R4,R5          ;Compute physical 64-byte base for table
62 016124 020537 000136'        CMP      R5,FMEMLO        ;Did we run out of memory space?
63 016130 101410          BLOS     10$          ;Br if memory overflow
64 016132 010537 000000G        MOV      R5,MAPPAR        ;Use this value to map PAR 5 to alloc table
65 016136 010537 000134'        MOV      R5,FMEMHI        ;Save new top of free memory area
66          ;
67          ; Finished
68          ;
69 016142 012604          MOV      (SP)+,R4
70 016144 012603          MOV      (SP)+,R3
71 016146 012602          MOV      (SP)+,R2
72 016150 000207          RETURN
73          ;
74          ; Error: Generated system is too large
75          ;
76 016152          10$: .PRINT  #TSXHD          ;Print error message heading
77 016160          .PRINT  #PHSQVF        ;Physical memory overflow
78 016166 000137 004134'        JMP      INISTP          ;Abort the initialization

```

```

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 016172 010546 SETJSZ: MOV      R5, -(SP)
17         ;
18         ; Determine total number of pages of memory available to user jobs
19         ;
20 016174 013705 000000G      MOV      HIMAP, R5      ; POINTER ABOVE LAST FREE PAGE ENTRY
21 016200 163705 000000G      SUB      LOMAP, R5      ; GET TOTAL # OF FREE PAGES
22 016204 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 016210 020537 000000G      CMP      R5, KMNPGS      ; COMPARE # FREE PAGES TO # PAGES FOR TSKMON
27 016214 103436              BLO      1$              ; BR IF INSUFFICIENT MEMORY TO RUN TSKMON
28         ;
29         ; Set up max memory limit for jobs
30         ;
31 016216 013700 000000G      MOV      CXTPAG, R0      ; # PAGES NEEDED FOR JOB CONTEXT AREA
32 016222 010037 000000G      MOV      R0, JCXPGS
33 016226 160005              SUB      R0, R5          ; LEAVE ROOM FOR JOB CONTEXT AREA
34 016230 006205              ASR      R5              ; Convert # pages to # KB
35 016232 020537 000000G      CMP      R5, VHIMEM      ; COMPARE WITH TSGEN SPECIFIED MAX SIZE
36 016236 101402              BLOS     10$            ; BR IF CONSTRAINED BY PHYSICAL SIZE
37 016240 013705 000000G      MOV      VHIMEM, R5      ; LIMIT BY VALUE SPECIFIED IN TSGEN
38 016244 010537 000000G 10$: MOV      R5, MXJMEM      ; MAX # K BYTES AVAILABLE TO A JOB
39 016250 010500              MOV      R5, R0
40 016252 072027 000012      ASH      #10, R0          ; CONVERT # KB TO BYTE ADDRESS
41 016256 001002              BNE      2$              ; BR IF DIDN'T OVERFLOW 64KB
42 016260 012700 177774      MOV      #177774, R0      ; GET 64KB TOP ADDRESS
43 016264 010037 000000G 2$:  MOV      R0, MXJADR      ; ADDRESS ABOVE TOP OF JOB
44         ;
45         ; Set default memory size of jobs
46         ;
47 016270 020537 000000G      CMP      R5, VDFMEM      ; COMPARE TO DEFAULT SPECIFIED IN TSGEN
48 016274 101402              BLOS     11$            ; BR IF CONSTRAINED BY PHYSICAL LIMIT
49 016276 013705 000000G      MOV      VDFMEM, R5      ; CONSTRAIN BY TSGEN PARAMETER
50 016302 010537 000000G 11$: MOV      R5, DFJMEM      ; SET DEFAULT JOB MEMORY SIZE
51         ;
52         ; Finished
53         ;
54 016306 012605              MOV      (SP)+, R5
55 016310 000207              RETURN
56         ;
57         ; Error -- Insufficient memory space available to run TSKMON.

```

TSINIT -- TSX startup initializ MACRO V05.05 Friday 20-Jan-89 11:30 Page 49-1
SETJSZ -- Set up information about maximum job sizes

58

59 016312 004737 025124'

1\$:

CALL

SIZERR

;Generated system is too big -- abort

SETJSZ -- Set up information about maximum job sizes

```

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          PARSET: TST      #MPARFL      ;Does he want to disable memory parity?
8                  BNE      20$          ;Br if not
9                  MOV      R2,-(SP)
10                 MOV      @#4,-(SP)      ;Save contents of trap vector
11          ;
12          ; Catch traps that occur when we access unimplemented parity registers
13          ;
14                 MOV      #2@, @#4      ;Send traps to 2$
15                 NOP                      ;Clean out instruction pipeline
16                 NOP
17          ;
18          ; Disable parity for each block of memory
19          ;
20                 MOV      #MPARO,R2      ;Point to 1st memory control register
21          1$:      BIC      #PARENL,(R2)  ;Disable memory parity
22                 BR        3$            ;We did not trap
23          2$:      ADD      #4,SP         ;Clean trap PS and PC off of stack
24          3$:      ADD      #2,R2         ;Point to next parity control register
25                 CMP      R2,#MPAR16     ;Have we cleared all registers?
26                 BLOS     1$             ;Loop if not
27          ;
28          ; Finished
29          ;
30                 MOV      (SP)+,@#4      ;Restore trap vector
31                 MOV      (SP)+,R2
32          20$:     RETURN
33          . IFF      ; NE,<PROASM-1>
34          PARSET:  RETURN
35          . ENDC     ; NE,<PROASM-1>

```

016316 000207

```

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 016320 010146
15 016322 010246
16 016324 010446
17
18         ; Begin loop to check all handlers specified in TSGEN with DEVDEF.
19         ;
20 016326 005001
21 016330 020127 000000C
22 016334 103015
23 016336 016102 000000G
24 016342 001407
25 016344 020227 000000G
26 016350 001404
27 016352 016104 000000G
28
29         ;
30         ; Load this handler
31 016356 004737 016430'
32         ; CALL LDHAND ; Try to load handler into memory
33         ;
34         ; Check next device
35 016362 062701 000002
36 016366 000760
37
38         ; Now see if there are spooled devices to contend with
39         ;
40 016370 012704 000000G
41 016374 001411
42 016376 012701 000000G
43 016402 012102
44 016404 010446
45 016406 005004
46 016410 004737 016430'
47 016414 012604
48 016416 077407
49
50         ; Finished
51         ;
52 016420 012604
53 016422 012602
54 016424 012601
55 016426 000207

```

```

          GETHNL: MOV     R1,-(SP)
                  MOV     R2,-(SP)
                  MOV     R4,-(SP)
          ;
          ; Begin loop to check all handlers specified in TSGEN with DEVDEF.
          ;
          CLR      R1          ; Init device table index
1$:          CMP      R1,#<AHEND-AUTHAN> ; Done all devices?
          BHS      2$          ; Br if yes
          MOV      AUTHAN(R1),R2 ; Get the name of the device
          BEQ      3$          ; Ignore null devices
          CMP      R2,#DMYDEV   ; Is this a dummy device entry?
          BEQ      3$          ; Skip it if yes
          MOV      DTYPE(R1),R4 ; Get flags specified in TSGEN
          ;
          ; Load this handler
          ;
          CALL     LDHAND      ; Try to load handler into memory
          ;
          ; Check next device
          ;
          3$:      ADD      #2,R1          ; Advance device index
                  BR       1$            ; See if more devices to load
          ;
          ; Now see if there are spooled devices to contend with
          ;
          2$:      MOV      #SPLND,R4      ; Are there any spooled devices?
                  BEQ      9$            ; Br if not
                  MOV      #SPLDEV,R1     ; Point to spooled device name table
          5$:      MOV      (R1)+,R2       ; Get the name of the next spooled device
                  MOV      R4,-(SP)       ; Save device count
                  CLR      R4            ; Say no TSGEN flags for device
                  CALL     LDHAND         ; Load the handler
                  MOV      (SP)+,R4       ; Recover the device count
                  SOB      R4,5$          ; Loop if more handlers to load
          ;
          ; Finished
          ;
          9$:      MOV      (SP)+,R4
                  MOV      (SP)+,R2
                  MOV      (SP)+,R1
                  RETURN

```

LDHAND -- Load a device handler

```

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$xxx status flags for device from TSGEN.
10         ;   R5 = Address where handler is to be loaded.
11         ;
12         ; Outputs:
13         ;   R5 = New free memory address.
14         ;   NUMDEV = Incremented by 2.
15         ;   PNAME(i) = Rad-50 name of device.
16         ;   ENTRY(i) = Handler entry point.
17         ;   DVSTAT(i) = Device status flags.
18         ;   DVFLAG(i) = TSX-Plus device status flags.
19         ;   HANPAR(i) = PAR offset if this is a mapped handler.
20         ;
21 016430 010446 LDHAND: MOV      R4, -(SP)
22         ;
23         ; Determine if we should ignore this device
24         ;
25 016432 004737 016602' CALL     INSCK1      ;Should we ignore this device?
26 016436 103457      BCS      9$          ;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 016440 004737 016760' CALL     INSCK2      ;Perform some additional checks on handler
32 016444 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 016446 004737 017374' 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 016452 016400 000000G MOV      DVFLAG(R4), R0 ;Get TSX-Plus status flags for device
44 016456 032700 000000G BIT      #DX$NHM, R0 ;Are we never to map this handler?
45 016462 001036      BNE      1$          ;Br if handler cannot be mapped
46 016464 032700 000000G BIT      #DX$MPH, R0 ;Is mapping wanted for this handler?
47 016470 001433      BEQ      1$          ;Br if not
48 016472 032700 000000G BIT      #DX$IBH, R0 ;Does this handler have an internal I/O buff?
49 016476 001412      BEQ      2$          ;Br if not
50 016500 105737 000000G TSTB     UBUSMP      ;Does this machine have a mapped UNIBUS?
51 016504 001025      BNE      1$          ;Br if yes -- Don't map this handler
52 016506 032700 000000G BIT      #DX$MAP, R0 ;Does this handler require I/O mapping?
53 016512 001404      BEQ      2$          ;Br if not
54 016514 032737 000001 000242' BIT      #EXTLSI, ICONFG ;Is this a Q-bus system with more than 256Kb?
55 016522 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 016524 005237 000124' 2$: INC NMXHAN ;Count # of mapped handlers
60 016530 012764 000001 000000G MOV #1,HANPAR(R4) ;Set flag saying handler should be mapped
61
62 ; Make sure size of mapped handler does not exceed 8KB
63
64 016536 026427 000000G 020000 CMP HANSIZ(R4),#8192 ;Is mapped handler too big?
65 016544 101411 BLOS B$ ;Br if not too big
66 016546 012700 002335' MOV #HN2BIG,R0 ;Get error message address
67 016552 004737 020472' CALL HLERR ;Abort initialization if iniabt
68 016556 000404 BR B$
69
70 ; This handler must be loaded into low memory
71
72 016560 005064 000000G 1$: CLR HANPAR(R4) ;Say this handler is not mapped
73 016564 004737 017504' CALL LDHNL0 ;Load handler into low memory
74
75 ; Close the handler file
76
77 016570 B$: .CLOSE #1 ;Close the handler file
78
79 ; Finished
80
81 016576 012604 9$: MOV (SP)+,R4
82 016600 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 016602 010146  INSCK1: MOV      R1, -(SP)
17
18         ;  Strip off any specified unit number
19         ;
20 016604 010201          MOV      R2, R1          ;Get full device name
21 016606 005000          CLR      R0              ;Set for divide
22 016610 071027 000050   DIV      #50, R0        ;Split off last digit
23 016614 070027 000050   MUL      #50, R0        ;Now correct for divide
24 016620 010102          MOV      R1, R2          ;Get device name less 3rd digit
25 016622 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 016626 020227 045640   CMP      R2, #R50LD     ;Is device name LD?
31 016632 001004          BNE      1$              ;Br if not
32 016634 105737 000000G  TSTB     VLDSYS        ;Is standard system LD support included?
33 016640 001044          BNE      5$              ;Br if yes -- Don't load LD
34 016642 000417          BR       3$              ;Load LD
35 016644 012701 000160'  1$: MOV      #SKPDEV, R1  ;Point to table of devices to skip
36 016650 020221 2$: CMP      R2, (R1)+          ;Is this a device to be skipped?
37 016652 001437          BEQ      5$              ;Br if yes
38 016654 005711          TST      (R1)           ;Reached end of skip list?
39 016656 001374          BNE      2$              ;Loop if not
40
41         ;  See if we have already loaded the handler for this device
42         ;
43 016660 013701 000000G  MOV      NUMDEV, R1      ;Get index for last device
44 016664 001406          BEQ      3$              ;Br if no devices installed yet
45 016666 020261 000000G  4$: CMP      R2, PNAME(R1) ;See if this device is already installed
46 016672 001427          BEQ      5$              ;Br if already installed
47 016674 162701 000002   SUB      #2, R1        ;More installed devices to check?
48 016700 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 016702 012701 000336'  3$: MOV      #DVFLBS, R1  ;Point to start of table
54 016706 020261 000000   6$: CMP      R2, DV$NAM(R1) ;Search for device in the table
55 016712 001003          BNE      7$              ;Br if this is not it
56 016714 056104 000002   BIS      DV$FLG(R1), R4  ;Combine default flags
57 016720 000405          BR       8$

```

INSCK1 -- Determine if a handler should be installed

```
58 016722 062701 000004      7$:      ADD      #DV$$SZ,R1      ;Point to next entry
59 016726 020127 000516'      CMP      R1,#DVFLND      ;Checked all entries?
60 016732 103765              BLO      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 016734 032704 000000G      8$:      BIT      #DX$DMA,R4      ;Is this a DMA device?
66 016740 001402              BEQ      10$      ;Br if not
67 016742 052704 000000G      BIS      #DX$EBA,R4      ;Set even-buffer-boundary flag
68                          ;
69                          ; Load this handler
70                          ;
71 016746 000241      10$:      CLC                      ;Set flag saying to load the handler
72 016750 000401              BR      9$
73                          ;
74                          ; Do not load this handler
75                          ;
76 016752 000261      5$:      SEC                      ;Set flag saying not to load the handler
77                          ;
78                          ; Finished
79                          ;
80 016754 012601      9$:      MOV      (SP)+,R1
81 016756 000207              RETURN
```

INSCK2 -- Additional checking for handler installation

```

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 016760 010146  INSCK2: MOV      R1,-(SP)
15 016762 010246      MOV      R2,-(SP)
16 016764 010346      MOV      R3,-(SP)
17 016766 010446      MOV      R4,-(SP)
18 016770 010546      MOV      R5,-(SP)
19 016772 013746 000004      MOV      @#4,-(SP)      ;Save the bus timeout vector
20 016776 013746 000010      MOV      @#10,-(SP)     ;Save illegal instruction vector
21         ;
22         ;  Try to lookup handler file on system disk
23         ;
24 017002 010237 000202'      MOV      R2,HANNAM+2      ;Set the device name for the lookup
25         ;;; Don't change channel # without changing STDVTB
26 017006          .LOOKUP #AREA,#1,#HANNAM;Try to open the handler file
27 017026 103006      BCC      1$      ;Br if we found the handler file
28         ;
29         ;  Error -- Cannot find handler file
30         ;
31 017030 012700 001510' 2$:      MOV      #CFHMSG,R0      ;Can't find handler
32 017034 004737 020472'      CALL     HLERR      ;See if should abort initialization
33 017040 000137 017350'      JMP      13$
34         ;
35         ;  We were able to open the handler file.
36         ;  Read in block 0 of handler.
37         ;
38 017044          1$:      .READW  #AREA,#1,WRKBUF,#256.,#0 ;Read block 0 into WRKBUF
39 017102 103006      BCC      3$      ;Br if read ok
40 017104 012700 001552'      MOV      #ERHMSG,R0      ;Error during read
41 017110 004737 020472'      CALL     HLERR
42 017114 000137 017350'      JMP      13$
43         ;
44         ;  Determine if the handler is supported under the current RT-11 version
45         ;
46 017120 012700 000322' 3$:      MOV      #HVTBL,R0      ;Point to table with handler version info
47 017124 013701 000152'      MOV      WRKBUF,R1      ;Point to buffer with block 0 of handler
48 017130 116101 000000G      MOVVB   H,DSTS(R1),R1    ;Get device ID code from handler
49 017134 120160 000000      51$:     CMPB     R1,HV$ID(R0) ;Compare handler ID code with table entry
50 017140 001011      BNE      53$      ;Br if this entry not for this handler
51 017142 026037 000002 000156'      CMP      HV$VER(R0),RTVPTR ;Is handler valid for this version?
52 017150 101412      BLOS     54$      ;Br if OK
53 017152 012700 001616'      MOV      #ERHNDV,R0      ;Wrong version of RT for handler
54 017156 004737 020472'      CALL     HLERR      ;See if should Report error and abort
55 017162 000472      BR       13$
56 017164 062700 000004      53$:     ADD      #HV$$SZ,R0 ;Point to next handler version table entry
57 017170 020027 000336'      CMP      R0,#HVEND      ;Are there more entries?

```

```

58 017174 103757          BLD      51$          ;Loop if more to check
59
60          ; Check handler sysgen options
61
62 017176 013700 000152' 54$:  MOV     WRKBUF,RO      ;Point to buffer with handler block 0
63 017202 032760 000000G 000000G  BIT     #SG$MMU,H.GEN(RO);Was handler genned with XM support?
64 017210 001005          BNE     4$              ;Br if yes
65 017212 012700 002027'  MOV     #HSGER,RO      ;Error if not XM version of handler
66 017216 004737 020472'  CALL    HLERR
67 017222 000452          BR      13$
68 017224 016037 000000G 000240' 4$:  MOV     H.GEN(RO),HGENFL;Save handler sysgen flags for later
69
70          ; Check the CSR address specified in the handler to see if the
71          ; hardware device for this handler exists.
72
73 017232 012737 015606' 000004  MOV     #TRCSET,@#4      ;Catch bus timeout traps
74 017240 012737 015606' 000010  MOV     #TRCSET,@#10     ;Catch illegal instruction traps
75 017246 013700 000152'  MOV     WRKBUF,RO      ;Point to start of block 0 of handler
76 017252 016001 000000G  MOV     H.CSR(RO),R1    ;Get address of CSR for device
77 017256 001407          BEQ     5$              ;Br if no CSR specified
78 017260 005711          TST     (R1)           ;Is CSR accessible?
79 017262 103005          BCC     5$              ;Br if ok
80 017264 012700 001670'  MOV     #NOCSSR,RO     ;Trap occurred while accessing CSR
81 017270 004737 020472'  CALL    HLERR
82 017274 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 017276 062700 000000G 5$:  ADD     #H.INS,RO      ;Offset 200 in block 0
89 017302 005710          TST     @RO            ;Does any installation code exist?
90 017304 001420          BEQ     11$            ;Br if no driver installation code
91 017306 013746 000000G  MOV     @#RMON,-(SP)    ;Save RT-11 RMON pointer
92 017312 012737 000000G 000000G  MOV     #MONVEC,@#RMON    ;Set TSX-Plus RMON pointer
93 017320 013703 000000G  MOV     RPRVEC,R3      ;Get pointer to Pro vec addr routine
94 017324 004710          CALL    @RO            ;Call the installation code
95 017326 012637 000000G  MOV     (SP)+,@#RMON    ;Restore RT-11 RMON pointer
96 017332 103006          BCC     13$            ;C-flag now indicates handler load status
97 017334 012700 001721'  MOV     #ERHINS,RO     ;Error occured in handler installation code
98 017340 004737 020472'  CALL    HLERR
99 017344 000401          BR      13$
100
101          ; Finished with installation verification.
102
103 017346 000241 11$:  CLC
104 017350 012637 000010 13$:  MOV     (SP)+,@#10     ;Restore illegal instruction vector
105 017354 012637 000004  MOV     (SP)+,@#4      ;Restore the bus timeout vector
106 017360 012605  MOV     (SP)+,R5
107 017362 012604  MOV     (SP)+,R4
108 017364 012603  MOV     (SP)+,R3
109 017366 012602  MOV     (SP)+,R2
110 017370 012601  MOV     (SP)+,R1
111 017372 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 017374   STDVTB:
22         ;
23         ; Increment device counter
24         ;
25 017374   062737   000002   000000G      ADD     #2,NUMDEV      ;Say another device added to tables
26 017402   013700   000000G      MOV     NUMDEV,R0      ;Get device index number
27         ;
28         ; Set up PNAME and DVFLAG.
29         ;
30 017406   010260   000000G      MOV     R2,PNAME(R0)    ;Set permanent device name
31 017412   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 017416   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 017424   010004      MOV     R0,R4      ;Carry device index in R4
48 017426   013700   000152'      MOV     WRKBUF,R0      ;Point to block 0 of handler
49 017432   016064   000000G 000000G      MOV     H.SIZ(R0),HANSIZ(R4) ;Set handler size
50 017440   016064   000000G 000000G      MOV     H.DVSZ(R0),DEVSIZ(R4) ;Number of blocks on device
51 017446   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 017454   032764   000000G 000000G      BIT     #DS$DIR,DVSTAT(R4) ;Is this a directory structured device?
56 017462   001404      BEQ     1$          ;Br if not -- No mounts allowed
57 017464   032764   000000G 000000G      BIT     #DS$NRD,DVSTAT(R4) ;Non RT-11 directory structure (mag tape)?

```

TSINIT -- TSX startup initializ MACRO V05.05 Friday 20-Jan-89 11:30 Page 55-1
STDVTB -- Set up device table entries for a device

```
58 017472 001403          BEQ      9$          ;Br if not
59 017474 052764 000000C 000000G 1$:    BIS      #<DX$NMT!DX$NCA>,DVFLAG(R4) ;Disable mounts and data caching
60                                     ;
61                                     ; Finished
62                                     ;
63 017502 000207          9$:    RETURN
```

```

1          .SBTTL  LDHNLO -- Load device handler into low memory
2          ;-----
3          ; LDHNLO 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 017504 010346 LDHNLO: MOV      R3, -(SP)
13         ;
14         ; Determine if we have enough free memory space to read the handler
15         ;
16 017506 005064 000000G      CLR      HANPAR(R4)      ; Say this handler is not mapped
17 017512 010500      MOV      R5, R0      ; Get current top of memory address
18 017514 016403 000000G      MOV      HANSIZ(R4), R3    ; Get size of handler
19 017520 060300      ADD      R3, R0      ; Get address above top of handler
20 017522 004737 025104'      CALL     CHKMEM      ; See if handler will fit in memory
21         ;
22         ; Handler will fit. Read it into memory.
23         ;
24 017526 006203      ASR      R3      ; Get number of words to read
25 017530      .READW  #AREA, #1, R5, R3, #1
26 017564 103005      BCC      1$      ; Br if read ok
27 017566 012700 001552'      MOV      #ERHMSG, R0      ; Error reading handler
28 017572 004737 020472'      CALL     HLERR      ; See if should Abort initialization
29 017576 000414      BR       2$
30         ;
31         ; Set address of handler entry point and compute address beyond
32         ; end of the handler.
33         ;
34 017600 010564 000000G 1$: MOV      R5, HANENT(R4)      ; Set address of handler entry point
35 017604 062764 000006 000000G      ADD      #6, HANENT(R4) ; (Point to fourth word of handler)
36 017612 006303      ASL      R3      ; Convert handler size to bytes
37 017614 060305      ADD      R3, R5      ; Point beyond end of handler
38         ;
39         ; Set up table of addresses of support routines at end of handler.
40         ;
41 017616 010503      MOV      R5, R3      ; Get address past end of handler
42 017620 004737 020400'      CALL     STHNPV      ; Set up pointer vector in handler
43         ;
44         ; If handler has any load-time execution code, run it now
45         ;
46 017624 004737 020542'      CALL     DOHNLC      ; Run any load-time code for handler
47         ;
48         ; Finished
49         ;
50 017630 012603 2$: MOV      (SP)+, R3
51 017632 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 017634 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 017636 012704 000002      MOV      #2, R4          ;Get index for first device entry
20         ;
21         ; See if this device has a mapped handler
22         ;
23 017642 005764 000000G 1$:      TST      HANPAR(R4)      ;Is this handler mapped?
24 017646 001402          BEQ      2$              ;Br if not
25         ;
26         ; We found an entry for a device with a mapped handler.
27         ; Load the handler.
28         ;
29 017650 004737 017672'      CALL      LDHNI          ;Load a mapped handler
30         ;
31         ; Look for more mapped handlers
32         ;
33 017654 062704 000002 2$:      ADD      #2, R4          ;Increment device index
34 017660 020437 000000G      CMP      R4, NUMDEV      ;Checked all devices?
35 017664 101766          BLOS     1$              ;Loop if not
36         ;
37         ; Finished
38         ;
39 017666 012604          MOV      (SP)+, R4
40 017670 000207          RETURN
  
```

```

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 017672 010146 LDHNHI: MOV     R1,-(SP)
13 017674 010246      MOV     R2,-(SP)
14 017676 010346      MOV     R3,-(SP)
15 017700 010446      MOV     R4,-(SP)
16         ;
17         ; Open channel 1 to the handler file.
18         ;
19 017702 016437 000000G 000202'      MOV     PNAME(R4),HANNAM+2 ;Set the device name for the lookup
20 017710 016437 000000G 000146'      MOV     PNAME(R4),CURNAM;Set name in case we have an error
21 017716          .LOOKUP #AREA,#1,#HANNAM;Try to open the handler file
22 017736 103006      BCC     B$          ;Br if we found the handler file
23 017740 012700 001510'      MOV     #CFHMSG,R0          ;Can't find handler
24 017744 004737 020472'      CALL    HLERR          ;See if should Abort initialization
25 017750 000137 020360'      JMP     9$
26         ;
27         ; Read block 0 of the handler file and extract some information
28         ;
29 017754 013702 000152'      B$:      MOV     WRKBUF,R2          ;Get address of work buffer
30 017760          .READW #AREA,#1,R2,#256.,#0 ;Read block 0 of handler
31 020014 016237 000000G 000240'      MOV     H.GEN(R2),HGENFL;Save handler sysgen flags
32         ;
33         ; Set virtual address of handler entry point
34         ;
35 020022 012764 000006G 000000G      MOV     #VPAR5+6,HANENT(R4) ;Set virtual addr of handler entry point
36         ;
37         ; Get information about the size of the handler and determine the
38         ; address in extended memory where the handler is to be loaded.
39         ;
40 020030 016402 000000G      MOV     HANSIZ(R4),R2          ;Get size of handler (bytes)
41 020034 005202          INC     R2          ;Make sure handler size is even
42 020036 042702 000001      BIC     #1,R2
43 020042 010200          MOV     R2,R0
44 020044 062700 000077      ADD     #63.,R0          ;Round up to # 64-byte blocks
45 020050 072027 177772      ASH     #-6,R0          ;Get # 64-byte blocks for handler
46 020054 060037 000000G      ADD     R0,MHNSIZ          ;Accumulate total space for mapped handlers
47 020060 160005          SUB     R0,R5          ;Reserve room for handler
48 020062 010564 000000G      MOV     R5,HANPAR(R4)          ;Set mapping value for handler
49 020066 010537 000126'      MOV     R5,HMAP          ;Set initial PAR base for handler
50 020072 012737 000001 000142'      MOV     #1,FILBLK          ;Set # of block to read from file
51         ;
52         ; Begin loop to read handler into memory
53         ;
54 020100 010203      1$:      MOV     R2,R3          ;Get remaining size of handler
55 020102 020327 001000      CMP     R3,#512.          ;Compare with max we can read at one time
56 020106 101402          BLOS    2$          ;Br if we can read remainder of handler
57 020110 012703 001000      MOV     #512.,R3          ;Read one block

```

LDHNHI -- Load device handler into extended memory

```

58 020114 160302      2$:      SUB      R3,R2      ;Reduce amt of handler left to read
59                      ;
60                      ; Read next block of handler
61                      ;
62 020116 006203      ASR      R3      ;Get # words to read
63 020120 013701 000152'  MOV      WRKBUF,R1      ;Get address of buffer for read
64 020124      .READW    #AREA,#1,R1,R3,FILBLK ;Read a block
65 020160 103005      BCC      3$      ;Br if read ok
66 020162 012700 001552'  MOV      #ERHMSG,R0      ;Get error message
67 020166 004737 020472'  CALL     HLERR      ;See if should Abort initialization
68 020172 000472      BR       9$
69                      ;
70                      ; Move the code we just read into the XM area for the handler
71                      ;
72 020174 012700 000000G  3$:      MOV      #VPAR5,R0      ;Get virtual address of mapped region
73 020200      DISABL      ;** Disable interrupts **
74 020206 013746 000000G  MOV      @#KPAR5,-(SP)      ;;; Save current mapping of PAR 5
75 020212 013737 000126' 000000G  MOV      HMAP,@#KPAR5      ;;; Set up mapping to get to XM area
76 020220 052737 000000G 000000G  BIS      #MMENBL,@#SR0MMR ;;; Enable memory management
77 020226 105737 000000G  TSTB     MEM256      ;;; Does machine have > 256KB?
78 020232 001403      BEQ      4$      ;;; Br if not
79 020234 052737 000000G 000000G  BIS      #EMMAP,@#SR3MMR ;;; Enable extended memory addressing
80 020242 012120      4$:      MOV      (R1)+,(R0)+      ;;; Move from WRKBUF to XM region
81 020244 077302      SOB      R3,4$      ;;; Loop till all moved
82 020246 105737 000000G  TSTB     MEM256      ;;; Does machine have > 256KB?
83 020252 001403      BEQ      5$      ;;; Br if not
84 020254 042737 000000G 000000G  BIC      #EMMAP,@#SR3MMR ;;; Disable extended memory addressing
85 020262 042737 000000G 000000G  5$:      BIC      #MMENBL,@#SR0MMR ;;; Enable memory management
86 020270 012637 000000G  MOV      (SP)+,@#KPAR5      ;;; Replace PAR 5 mapping
87 020274      ENABL      ; ** Enable interrupts **
88                      ;
89                      ; See if there is more to read
90                      ;
91 020302 062737 000010 000126'  ADD      #8,HMAP      ;Increase XM region base
92 020310 005237 000142'      INC      FILBLK      ;Increment file block number
93 020314 005702      TST      R2      ;Is there more to read?
94 020316 001270      BNE      1$      ;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 020320 012703 000000G  MOV      #VPAR5,R3      ;Get virtual address of handler base
100 020324 066403 000000G  ADD      HANSIZ(R4),R3      ;Get virtual address beyond end of handler
101 020330 004737 021242'      CALL     HANMAP      ;;; Map KPAR5 to the handler
102 020334 004737 020400'      CALL     STHNPV      ;;; Set up handler pointer vector
103 020340 004737 021316'      CALL     HANUMP      ;Restore mapping
104                      ;
105                      ; If handler has any load-time code, run it now
106                      ;
107 020344 010537 000134'      MOV      R5,FMEMHI      ;Set addr of top of free memory area
108 020350 004737 020542'      CALL     DOHNLC      ;Run any load-time code for handler
109 020354 013705 000134'      MOV      FMEMHI,R5      ;Get new top of free memory address
110                      ;
111                      ; Close the handler file
112                      ;
113 020360      9$:      .CLOSE    #1      ;Close the handler file
114                      ;

```

TSINIT -- TSX startup initializ MACRO V05.05 Friday 20-Jan-89 11:30 Page 58-2
LDHNHI -- Load device handler into extended memory

115			; Finished
116			;
117	020366	012604	MOV (SP)+, R4
118	020370	012603	MOV (SP)+, R3
119	020372	012602	MOV (SP)+, R2
120	020374	012601	MOV (SP)+, R1
121	020376	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 020400 010346  STHNPV: MOV      R3, -(SP)
12         ;
13         ; Set up addresses in the pointer vector
14         ;
15 020402 012743 000000G      MOV      #FORK, -(R3)      ; Address of fork routine
16 020406 012743 000000G      MOV      #INTEN, -(R3)     ; Address of inten routine
17 020412 032737 000000G 000240'  BIT      #SG$IOT, HGENFL ; Does handler want timeout support?
18 020420 001402          BEQ      2$                    ; Br if not
19 020422 012743 000000G      MOV      #IOTIMR, -(R3)    ; Set address of timeout support routine
20 020426 032737 000000G 000240' 2$:  BIT      #SG$ELG, HGENFL ; Does handler want error logging support?
21 020434 001402          BEQ      3$                    ; Br if not
22 020436 012743 000000G      MOV      #ERRLOG, -(R3)    ; Set address of error logging routine
23 020442 012743 000000G      MOV      #PTWRD, -(R3)
24 020446 012743 000000G      MOV      #PTBYT, -(R3)
25 020452 012743 000000G      MOV      #GTBYT, -(R3)
26 020456 012743 000000G      MOV      #MPPHY, -(R3)
27 020462 012743 000000G      MOV      #RELOC, -(R3)
28         ;
29         ; Finished
30         ;
31 020466 012603          MOV      (SP)+, R3
32 020470 000207          RETURN

```

1					
2					
3					
4					
5	020472	010001		HLERR:	MOV RO,R1 ;SAVE ERROR MESSAGE ADDRESS
6	020474	105737	0000000G		TSTB VINABT ;ABORT OR JUST PRINT MESSAGE?
7	020500	001416			BEQ 9\$;BR IF NOT ABORT
8	020502				.PRINT #TSXHD ;PRINT ERROR MESSAGE HEADING
9	020510				.PRINT R1 ;PRINT ERROR MESSAGE
10	020514	013700	000146'		MOV CURNAM,R0 ;GET RAD50 DEVICE NAME
11	020520	004737	025274'		CALL PRTR50 ;PRINT DEVICE NAME
12	020524				.PRINT #CRLF
13	020532	000137	004134'		JMP INISTP ;ABORT INITIALIZATION
14	020536	000261		9\$:	SEC ;MAKE SURE CARRY IS SET
15	020540	000207			RETURN

```

1          .SBTTL  DOHNLC -- Execute and handler load/fetch code
2          ;-----
3          ; If the handler being loaded has any Load-time execution code, read it
4          ; into our work buffer and execute it now.
5          ;
6          ; Inputs:
7          ;   R4 = Device index number of handler that is being loaded.
8          ;
9          ; Outputs:
10         ;   C-flag is set on return if load code signals an error during its
11         ;   execution.
12         ;
13 020542 010146 DOHNLC: MOV     R1,-(SP)
14 020544 010246      MOV     R2,-(SP)
15 020546 010346      MOV     R3,-(SP)
16 020550 010446      MOV     R4,-(SP)
17 020552 010546      MOV     R5,-(SP)
18         ;
19         ; Examine 1st word of handler to see if it could have any load-time code.
20         ;
21 020554 016405 0000000 MOV     HANENT(R4),R5 ;Get address of handler entry point
22 020560 004737 021242' CALL    HANMAP      ;;;Map Kpar5 to handler if mapped handler
23 020564 016500 0000004 MOV     4(R5),R0      ;;;Get 1st instruction located at 4 in handler
24 020570 004737 021316' CALL    HANUMP      ;Restore normal mapping
25 020574 020027 000240 CMP     R0,#240      ;Is it a NOP?
26 020600 103516      BLO     7$          ;Br if can't be any load code
27 020602 020027 000277 CMP     R0,#277      ;
28 020606 101113      BHI     7$          ;Br if can't be any load code
29 020610 132700 0000004 BITB    #4,R0      ;Is there load code?
30 020614 001510      BEQ     7$          ;Br if not
31         ;
32         ; Handler may have load code.
33         ; Read block 0 of handler and get offset to load code.
34         ;
35 020616 013702 000152' MOV     WRKBUF,R2      ;Get addr of our work buffer
36 020622      .READW    #AREA,#1,R2,#256.,#0 ;Read block 0 of handler
37 020656 022227 031066 CMP     (R2)+,#^RHAN      ;Is this a new type handler?
38 020662 001065      BNE     7$          ;Br if not
39 020664 016203 0000004 MOV     4(R2),R3      ;Get offset to load code
40 020670 001462      BEQ     7$          ;Br if there is none
41         ;
42         ; There is load-time code.
43         ; Read into WRKBUF the portion of the handler with the load code.
44         ;
45 020672 020327 001000 CMP     R3,#1000      ;Is load code in block 0 of handler?
46 020676 103424      BLO     1$          ;Br if yes
47 020700 010302      MOV     R3,R2      ;Get offset to start of load code
48 020702 072227 177767 ASH     #-9,R2      ;Convert to a block number
49 020706 042702 177400 BIC     #^C377,R2      ;Clear all but block number
50 020712      .READW    #AREA,#1,WRKBUF,#512.,R2 ;Read 2 blocks from handler file
51         ;
52         ; The load code is now in WRKBUF. Set up and execute it.
53         ;
54 020750 010437 000144' 1$: MOV     R4,CURDEV      ;Save current device index number
55 020754 042703 177000 BIC     #^C777,R3      ;Get offset within block of load code entry pt
56 020760 010300      MOV     R3,R0      ;Get entry point offset
57 020762 063700 000152' ADD     WRKBUF,R0      ;Add base address

```

DOHNLC -- Execute and handler load/fetch code

```

58 020766 013701 000000G      MOV      RPRVEC,R1      ;Get pointer to GETVEC routine for Pro
59 020772 012702 000000C      MOV      #MAXDEV*2,R2      ;Get 2**# entries in device tables
60 020776 012703 0000004      MOV      #4,R3      ;Set code saying this is load code
61 021002 012705 000000G      MOV      #HANENT,R5      ;Point to handler entry address vector
62 021006 060405      ADD      R4,R5      ;Point to entry cell for this handler
63 021010 004737 021242'      CALL     HANMAP      ;;;Map Kpar5 to handler if it is a mapped
64 021014 012704 021072'      MOV      #LDREAD,R4      ;;;Get pointer to Read routine
65 021020 004710      CALL     (R0)      ;;;Execute the load code
66 021022 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 021024 012737 001400 000000G      MOV      #1400,@#KPAR6      ;;;Restore original mapping for RT-11
72 021032 004737 021316'      CALL     HANUMP      ;Unmap the handler
73 021036 000241      7$:      CLC      ;Clear the carry flag for return
74 021040 000406      BR      9$
75      ;
76      ; Error occurred in fetch/load code
77      ;
78 021042 012737 001400 000000G 2$:      MOV      #1400,@#KPAR6      ;;;Restore original mapping for RT-11
79 021050 004737 021316'      CALL     HANUMP      ;Unmap the handler
80 021054 000261      SEC      ;Set the carry flag for return
81      ;
82      ; Finished
83      ;
84 021056 012605      9$:      MOV      (SP)+,R5
85 021060 012604      MOV      (SP)+,R4
86 021062 012603      MOV      (SP)+,R3
87 021064 012602      MOV      (SP)+,R2
88 021066 012601      MOV      (SP)+,R1
89 021070 000207      RETURN

```

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15 021072 010046
16 021074 010446
17 021076 010004
18
19
20
21 021100 105737 000000G
22 021104 001402
23 021106 013746 000000G
24 021112 013746 000000G
25 021116 013746 000000G
26 021122 013746 000000G
27 021126 012737 001400 000000G
28
29
30
31 021134 004737 021316'
32
33
34
35 021140
36 021172 103420
37
38
39
40 021174 013704 000144'
41 021200 004737 021242'
42
43
44
45 021204 012637 000000G
46 021210 012637 000000G
47 021214 012637 000000G
48 021220 105737 000000G
49 021224 001402
50 021226 012637 000000G
51
52
53
54 021232 000241
55 021234 012604
56 021236 012600
57 021240 000207

.SBTTL LDREAD -- Perform I/O for handler load code
-----
; This routine performs Read operations for handler load code.
; It simulates the operation of the bootstrap read routine.
; When called, Channel 1 must be open to the handler file.
;
; Inputs:
; R0 = Block number within handler file to be read.
; R1 = Number of words to read.
; R2 = Buffer address
;
; Outputs:
; C-flag is set if a read error occurs
;
LDREAD: MOV R0, -(SP)
        MOV R4, -(SP)
        MOV R0, R4 ;;; Get starting block number
;
; Save current mapping information
;
        TSTB MEM256 ;;; Does machine have > 256?
        BEQ 1$ ;;; Br if not
        MOV @SR3MMR, -(SP) ;;; Save extended memory address register
1$:      MOV @SR0MMR, -(SP) ;;; Save memory mapping
        MOV @KPAR5, -(SP) ;;; Save current KPAR5 mapping
        MOV @KPAR6, -(SP) ;;; Save current KPAR6 mapping
        MOV #1400, @KPAR6 ;;; Restore original mapping for RT-11
;
; Turn off handler mapping
;
        CALL HANUMP ; Turn off handler mapping
;
; Read the requested data from the handler
;
        READW #AREA, #1, R2, R1, R4 ; Read the blocks
        BCS 9$ ; Br if read error
;
; Restore handler mapping
;
        MOV CURDEV, R4 ; Get current device index
        CALL HANMAP ;;; Map Kpar5 if necessary
;
; Restore mapping information
;
        MOV (SP)+, @KPAR6 ;;; Restore KPAR6 mapping
        MOV (SP)+, @KPAR5 ;;; Restore KPAR5 mapping
        MOV (SP)+, @SR0MMR ;;; Restore memory mapping
        TSTB MEM256 ;;; Does machine have > 256?
        BEQ 2$ ;;; Br if not
        MOV (SP)+, @SR3MMR ;;; Restore extended memory address register
;
; Finished
;
2$:      CLC ;;; Signal success on return
9$:      MOV (SP)+, R4
        MOV (SP)+, R0
        RETURN

```

```

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 021242   HANMAP:
15         ;
16         ;   Disable interrupts
17         ;
18 021242   DISABL          ;;;Disable interrupts
19         ;
20         ;   Change RMON pointer to point to TSX-Plus vector
21         ;
22 021250   012737   000000G 000000G      MOV      #MONVEC,@#RMON ;;;Say TSX-Plus is the monitor
23         ;
24         ;   See if this handler is mapped
25         ;
26 021256   005764   000000G      TST      HANPAR(R4)          ;;;Is this handler mapped?
27 021262   001403      BEQ      9$          ;;;Br if not
28         ;
29         ;   This handler is mapped.
30         ;   Set up mapping to access it.
31         ;
32 021264   016437   000000G 000000G      MOV      HANPAR(R4),@#KPAR5;Map KPAR5 to the handler code
33 021272   052737   000000G 000000G 9$:   BIS      #MMENBL,@#SR0MMR;Enable memory mapping
34 021300   105737   000000G      TSTB     MEM256          ;;;Does machine have > 256KB?
35 021304   001403      BEQ      10$          ;;;Br if not
36 021306   052737   000000G 000000G      BIS      #EMMAP,@#SR3MMR ;;;Enable extended memory addressing
37         ;
38         ;   Finished
39         ;
40 021314   000207      10$:   RETURN
41
42          .SBTTL  HANUMP -- Turn off memory mapping to a handler
43          ;-----
44          ; This routine is the companion to HANMAP. It turns off memory mapping
45          ; and restores KPAR5 to its normal mapping value.
46          ; Enter with interrupts disabled. Interrupts are enabled on return.
47          ; This routine also changes the RMON pointer back to RT-11.
48          ;
49 021316   HANUMP:
50         ;
51         ;   Turn off memory management
52         ;
53 021316   105737   000000G      TSTB     MEM256          ;;;Does machine have > 256KB?
54 021322   001403      BEQ      1$          ;;;Br if not
55 021324   042737   000000G 000000G      BIC      #EMMAP,@#SR3MMR ;;;Turn off extended memory addressing
56 021332   042737   000000G 000000G 1$:   BIC      #MMENBL,@#SR0MMR;Turn off memory mapping
57 021340   012737   001200   000000G      MOV      #1200,@#KPAR5 ;;;Reset KPAR5 to its normal mapping

```

TSINIT -- TSX startup initializ MACRO V05.05 Friday 20-Jan-89 11:30 Page 63-1
HANUMP -- Turn off memory mapping to a handler

```
58      ;  
59      ; Restore RMON pointer to RT11  
60      ;  
61 021346 013737 000046' 000000G      MOV      RTMNVC,@#RMON      ;;;Reset RMON pointer  
62      ;  
63      ; Enable interrupts  
64      ;  
65 021354      ENABL      ;Enable interrupts  
66      ;  
67      ; Finished  
68      ;  
69 021362 000207      RETURN
```

HANUMP -- Turn off memory mapping to a handler

```

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 021364 005001    FNDHRB: CLR      R1          ;Point to next region control block (none)
10 021366 000401    BR        1$          ;Go signal error and return
11          ;
12 021370 005002    HANXMR: CLR      R2          ;Return largest possible region size (none)
13 021372 000261    1$:      SEC          ;Signal error on request
14 021374 000207    RETURN
15          .DSABL   LSB
16          ;
17          .IFF    ;EQ,<PROASM-1> ;Include handler XM support for 11 versions
18          .SBTTL   FNDHRB -- Try to find a handler global region
19          -----
20          ; This routine is called to try to locate an allocated XM region with
21          ; a specified name.
22          ; If a region control block with the specified name cannot be found,
23          ; the address of a free one is returned and the specified name is stored
24          ; into the free block.
25          ;
26          ; Inputs:
27          ; R5 = Pointer to 2-word cell containing Rad50 name of region to be found.
28          ;
29          ; Outputs:
30          ; C-flag cleared ==> Found the specified RCB.
31          ; R1 = Address of the RCB
32          ; C-flag set ==> Could not find the specified RCB.
33          ; R1 = Pointer to a free RCB or 0 if no available RCB's.
34          ;
35 FNDHRB: MOV      R2,-(SP)
36          ;
37          ; Search for specified RCB and also remember if we see a free RCB
38          ;
39          CLR      R2          ;Say no free RCB found
40          MOV      HANRCB,R1    ;Point to start of RCB area
41          TST      (R1)+        ;Skip over -1 word at front
42 1$:      TST      (R1)          ;What is the status of this RCB
43          BNE      2$          ;Br if this is not a free RCB
44          MOV      R1,R2        ;Remember address of a free RCB
45          BR       3$
46 2$:      CMP      (R1), #-1     ;Are we at the end of the list?
47          BEQ      4$          ;Br if yes
48          CMP      (R5), 6(R1)  ;Compare the names
49          BNE      3$          ;Br if don't match
50          CMP      2(R5), 10(R1) ;Compare 2nd half of name
51          BEQ      6$          ;Br if found the RCB we were searching for
52 3$:      ADD      #12,R1        ;Point to the next RCB
53          BR       1$          ;Continue searching
54          ;
55          ; We could not find the specified RCB.
56          ; If there was a free one, initialize the name.
57          ;

```

```
58          4$:      MOV      R2,R1          ;Was there a free RCB?
59                      BEQ      5$          ;Br if not
60                      MOV      (R5),6(R1)   ;Set name in the RCB
61                      MOV      2(R5),10(R1) ; (2nd word of name)
62                      ADD      CURDEV,10(R1) ; Make name unique to device
63          5$:      SEC                      ;Signal that we did not find the RCB
64                      BR        9$
65          ;
66          ; We found the RCB
67          ;
68          6$:      CLC                      ;Signal success on return
69          ;
70          ; Finished
71          ;
72          9$:      MOV      (SP)+,R2
73                      RETURN
```

```

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         HANXMR: MOV     R0, -(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         MOV     FMEMHI, R0      ; Top of free memory
28         SUB     FMEMLO, R0      ; -Base of free memory
29         CMP     R2, R0          ; Do we have room for the requested region?
30         BHI     B$             ; Br if not
31         ;
32         ; There is room for the region so allocate it from the top of memory
33         ;
34         SUB     R2, FMEMHI      ; Allocate the region
35         MOV     FMEMHI, R1      ; Return the address of the base of the region
36         CLC                      ; Signal success on return
37         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         B$: MOV     R0, R2      ; Get the size of the largest possible region
43         SEC                      ; Signal failure on return
44         ;
45         ; Finished
46         ;
47         9$: MOV     (SP)+, R0
48         RETURN
49         ;
50         .ENDC  ; EQ, <PROASM-1>

```

```

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

```

```

58          ASR      R1          ;Get # words to zero
59          MOV      R5,R0       ;Get pointer to start of area
60          4$:      CLR      (R0)+ ;Zero the entire area
61          SOB      R1,4$
62          ;
63          ; Allocate and initialize the control structures
64          ;
65          MOV      R5,MIOBHD     ;Start of area for I/O mapping control blks
66          MOV      VMIOBF,R0    ;Get # buffers wanted
67          5$:      MOV      R5,R1 ;Get pointer to current control block
68          ADD      #MI$$SZ,R1   ;Get pointer to next control block
69          DEC      R0           ;Need to link more together?
70          BLE      6$          ;Br if not
71          MOV      R1,MI$LNK(R5) ;Set pointer to next control block
72          MOV      R1,R5       ;Advance pointer to next block
73          BR       5$          ;See if more to do
74          6$:      MOV      R1,R5 ;Set pointer past last block
75          MOV      R5,MIOWHD     ;Start of wait blocks
76          MOV      #MIONWB-1,R0 ;Get # wait blocks wanted - 1
77          7$:      MOV      R5,R1 ;Get pointer to current block
78          ADD      #MW$$SZ,R1   ;Get pointer to next block
79          MOV      R1,MW$LNK(R5) ;Set pointer to next wait block
80          MOV      R1,R5       ;Advance pointer to next block
81          SOB      R0,7$        ;Loop if more to allocate
82          ADD      #MW$$SZ,R5   ;Allocate space for last block (with 0 link)
83          ;
84          ; Finished
85          ;
86          9$:      MOV      (SP)+,R1
87          RETURN
88          . IFF      ;NE,<PROASM-1>
89          ;
90          ; Define dummy routine for Pro
91          ;
92          SETMID: RETURN
93          . ENDC      ;NE,<PROASM-1>

```

92 021376 000207

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15 021400 010146
16 021402 010246
17 021404 010346
18 021406 010446
19 021410 010504
20
21
22
23 021412 004737 021610'
24
25
26
27
28 021416 062704 000077
29 021422 042704 000077
30 021426 010437 000140'
31 021432 012702 000516'
32 021436 005762 000000
33 021442 001423
34 021444 016200 000004
35 021450 016000 000000G
36 021454 012701 000746'
37 021460 020021
38 021462 001404
39 021464 020127 000754'
40 021470 103773
41 021472 000407
42 021474 005262 000002
43 021500 016200 000000
44 021504 072027 000006
45 021510 060004
46 021512 062702 000006
47 021516 020237 000744'
48 021522 103745
49
50
51
52 021524 020427 025266'
53 021530 103021
54 021532 012705 025350'
55 021536 160405
56 021540 072527 177772
57

```

```

      .SBTTL  OVLPOS -- Determine which overlays go over TSINIT
      -----
      ; OVLPOS is called to determine which system overlays are to be placed
      ; over the TSINIT code (specifically, between @OVLBAS and INITOP).
      ;
      ; Inputs:
      ;   R5 = Base address in TSINIT where overlays may be loaded.
      ;
      ; Outputs:
      ;   Overlay segment information is set up in OSTABL.
      ;   OS$FLG(seg) = 0==>Load seg into high memory; 1==>Load over TSINIT.
      ;   OVLBAS = Address of location within TSINIT where overlays start.
      ;   R5 = Pointer past last overlay loaded over TSINIT.
      ;
OVLPOS:  MOV     R1, -(SP)
        MOV     R2, -(SP)
        MOV     R3, -(SP)
        MOV     R4, -(SP)
        MOV     R5, R4          ; Get address where we may load overlays
      ;
      ; Build the table that holds information about the overlays
      ;
        CALL    OVLBLD          ; Build overlay information table
      ;
      ; First determine how much space will be used by those overlays that are
      ; forced to be loaded over TSINIT.
      ;
        ADD     #63, R4          ; Bound address to 64-byte boundary
        BIC     #77, R4
        MOV     R4, OVLBAS       ; Remember address where we load overlays
        MOV     #OSTABL, R2      ; Point to start of table
1$:      TST     OS$SIZ(R2)       ; Is this overlay to be loaded?
        BEQ     2$              ; Br if not
        MOV     OS$OVL(R2), R0   ; Point to linker-built entry
        MOV     O.ADR(R0), R0    ; Get Rad50 segment ID
        MOV     #LOWOVL, R1      ; Point to table of overlays to go over TSINIT
6$:      CMP     R0, (R1)+        ; Must this overlay go over TSINIT?
        BEQ     4$              ; Br if yes
        CMP     R1, #LOWEND      ; End of low-overlay table?
        BLO     6$              ; Br if not
        BR      2$              ; This overlay is not forced over TSINIT
4$:      INC     OS$FLG(R2)       ; Set flag saying load over TSINIT
        MOV     OS$SIZ(R2), R0   ; Get # 64-byte blocks needed for overlay
        ASH     #6, R0           ; Get # bytes needed for overlay
        ADD     R0, R4           ; Advance address within TSINIT
2$:      ADD     #OS$$SZ, R2      ; Point to entry for next segment
        CMP     R2, OSLAST       ; Have we finished?
        BLO     1$              ; Loop if not
      ;
      ; Determine how much memory space is available in TSINIT for other overlays
      ;
        CMP     R4, #INITOP-50.  ; Any space left for other overlays?
        BHIS    9$              ; Br if not
        MOV     #INITOP, R5      ; Point to top of overlay area
        SUB     R4, R5           ; Total space available for overlays
        ASH     #-6, R5          ; Convert to # 64-byte blocks

```

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 021544 004737 023020' 3$: CALL OVLTRY      ; Try to find largest overlay that will fit
63 021550 103411          BCS 9$             ; Br if no more overlays will fit
64 021552 005262 000002   INC OS$FLG(R2)    ; Remember to load over TSINIT
65 021556 016200 000000   MOV OS$SIZ(R2),R0 ; Get # 64-byte blocks needed for overlay
66 021562 160005          SUB R0,R5         ; Reduce remaining free space in TSINIT
67 021564 072027 000006   ASH #6,R0        ; Get # bytes needed for overlay
68 021570 060004          ADD R0,R4         ; Advance overlay address in TSINIT
69 021572 000764          BR 3$             ; See if we can find more segments to load
70      ;
71      ; Finished
72      ;
73 021574 010405 9$: MOV R4,R5             ; Return top-of-overlay address in R5
74 021576 012604          MOV (SP)+,R4
75 021600 012603          MOV (SP)+,R3
76 021602 012602          MOV (SP)+,R2
77 021604 012601          MOV (SP)+,R1
78 021606 000207          RETURN

```

OVLBLD -- Build overlay information table

```

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         OVLBLD: MOV      R1,-(SP)
11         MOV      R2,-(SP)
12         MOV      R3,-(SP)
13         ;
14         ; Read 1st block of SAV file to get pointer to overlay table
15         ;
16         MOV      WRKBUF,R2          ;Point to work buffer
17         .READW   #AREA,#17,R2,#256.,#0 ;read first block of the save file
18         BCS      22$                ;Br if error on read
19         MOV      64(R2),R1          ;point to the overlay table
20         BNE      15$                ;br if overlays exist
21         ;
22         ; Must be version 3B overlays structure at absolute location.
23         ;
24         MOV      #137,@#1000        ;position jump instruction over 3b ovly handler
25         MOV      #$OVRH,@#1002     ;position overlay intercept location
26         MOV      #1104,R1          ;point to the overlay table
27         MOV      R1,OVRADD          ;save the address of the overlay table
28         ;
29         ; Initialize the table that holds information about the overlays
30         ;
31         15$: MOV      #OSTABL,R3      ;Point to table for overlay info
32         11$: MOV      R1,OS$OVL(R3)  ;Save pointer to overlay control block
33         CLR      OS$FLG(R3)          ;Assume seg will be loaded in high memory
34         CALL     ALCOVL              ;Determine if we should load this overlay
35         MOV      R2,OS$SIZ(R3)       ;Remember total size of overlay+data
36         ADD      #OS$$SZ,R3          ;Point to next overlay table entry
37         12$: ADD      #6,R1          ;find the next region
38         CMP      (R1),#4537          ;compare with a <JSR R5,$OVRH> instruction
39         BNE      11$                ;Br if not at end
40         MOV      R3,OSLAST          ;Save pointer past last overlay table entry
41         ;
42         ; Finished
43         ;
44         MOV      (SP)+,R3
45         MOV      (SP)+,R2
46         MOV      (SP)+,R1
47         RETURN
48         ;
49         ; Error -- Read error occured while reading overlay table
50         ;
51         22$: .PRINT  #TSXHD
52         .PRINT  #RDERR
53         JMP     INISTP

```

GETMAP -- Load any mapped system code regions

```

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 022010   010146
13 022012   010246
14 022014   010346
15 022016   010537   000000G
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 022022   012703   000516'
21 022026   004737   022274'
22 022032   010263   000000
23 022036   062703   000006
24 022042   020337   000744'
25 022046   103767
26
27         ; Load those overlays that go into high memory
28         ;
29 022050   012702   000516'
30 022054   005762   000000
31 022060   001405
32 022062   005762   000002
33 022066   001002
34 022070   004737   023116'
35 022074   062702   000006
36 022100   020237   000744'
37 022104   103763
38
39         ; Finished
40         ;
41 022106   013700   000000G
42 022112   160500
43 022114   010037   000000G
44 022120   012603
45 022122   012602
46 022124   012601
47 022126   000207

          GETMAP:  MOV     R1, -(SP)
                   MOV     R2, -(SP)
                   MOV     R3, -(SP)
                   MOV     R5, SMRSIZ      ; Save memory pointer at start of allocation
                   ;
                   ; Now that most of the system initialization is completed, we must check
                   ; again to see which overlays need to be loaded.
                   ;
                   MOV     #OSTABL, R3      ; Point to 1st overlay table entry
1$:          CALL    OPTOVL      ; See if this segment should be loaded
                   MOV     R2, OS$SIZ(R3) ; Save # 64-byte blocks needed for overlay
                   ADD     #OS$$SZ, R3     ; Point to next overlay table entry
                   CMP     R3, OSLAST     ; Checked all entries in overlay table?
                   BLO     1$            ; Br if not
                   ;
                   ; Load those overlays that go into high memory
                   ;
                   MOV     #OSTABL, R2     ; Point to 1st overlay entry
3$:          TST     OS$SIZ(R2)          ; Is this overlay segment wanted?
                   BEQ     4$            ; Br if not
                   TST     OS$FLG(R2)     ; Load over TSINIT or into high memory?
                   BNE     4$            ; Br if load over TSINIT
                   CALL    GETOVL         ; Load overlay into high memory
4$:          ADD     #OS$$SZ, R2          ; Point to next overlay table entry
                   CMP     R2, OSLAST     ; Have we done all overlays?
                   BLO     3$            ; Loop if not
                   ;
                   ; Finished
                   ;
19$:         MOV     SMRSIZ, R0           ; Get memory pointer at start of allocation
                   SUB     R5, R0         ; Calc amt of space allocated
                   MOV     R0, SMRSIZ     ; Save total space used for mapped regions
                   MOV     (SP)+, R3
                   MOV     (SP)+, R2
                   MOV     (SP)+, R1
                   RETURN

```

```

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$xxx)
9          ;
10         ; Outputs:
11         ;   C-flag cleared ==> This segment is to be loaded.
12         ;   C-flag set      ==> Do not load this overlay segment.
13         ;   R2 = # 64-Byte blocks needed for segment including data areas within it.
14         ;
15 022130 010146 ALCOVL: MOV      R1, -(SP)
16         ;
17         ; Get pointer to linker-build overlay entry for segment
18         ;
19 022132 016301 000004      MOV      OS$OVL(R3), R1      ; Get pointer to linker-built entry for seg
20         ;
21         ; Read in the first block of the overlay segment
22         ;
23 022136 013702 000152'      MOV      WRKBUF, R2          ; Point to work buffer
24 022142      .READW  #AREA, #17, R2, #256., D.BLK(R1) ; read the first block
25 022200 103415      BCS      3$                          ; Br if read error
26         ;
27         ; Save the 3 character Rad50 segment ID in the D.ADR cell of the
28         ; linker-built overlay table entry for this segment.
29         ;
30 022202 016261 000002 000000G      MOV      2(R2), D.ADR(R1) ; save the rad50 overlay identifier
31         ;
32         ; Make sure the segment is not larger than 8Kb
33         ;
34 022210 016102 000000G      MOV      D.SIZ(R1), R2        ; get the word count of the code region
35 022214 006302      ASL      R2                          ; convert to byte count
36 022216 020227 020000      CMP      R2, #20000           ; check for 8kb overflow
37 022222 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 022224 004737 022274'      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 022230 012601      MOV      (SP)+, R1
48 022232 000207      RETURN
49         ;
50         ; Error -- Error on reading from SAV file
51         ;
52 022234      3$:      .PRINT  #TSXHD                      ; Print heading
53 022242      .PRINT  #RDERR                      ; Read error
54 022250 000137 004134'      JMP      INISTP              ; Abort initialization
55         ;
56         ; Error -- Insufficient memory space to load run-time systems
57         ;

```

TSINIT -- TSX startup initializ MACRO V05.05 Friday 20-Jan-89 11:30 Page 70-1

ALCOVL -- Allocate space for a system overlay region

58 022254	21\$:	.PRINT	#TSXHD	;PRINT HEADING
59 022262		.PRINT	#TSXSIZ	;PRINT ERROR MESSAGE
60 022270 000137 004134'		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$xxx)
9          ;
10         ; Outputs:
11         ;   C-flag cleared ==> Load this overlay.
12         ;   C-flag set    ==> Do not load this overlay.
13         ;   R2 = # 64-byte blocks needed for code + data for the segment.
14         ;
15 022274 010346 OPTOVL: MOV     R3, -(SP)
16 022276 010446      MOV     R4, -(SP)
17 022300 010546      MOV     R5, -(SP)
18
19         ; Get the name of the overlay segment
20
21 022302 016305 000004      MOV     OS$OVL(R3), R5 ;Get pointer to linker-built entry
22 022306 016504 000000G    MOV     O. ADR(R5), R4 ;Get name of the segment
23
24         ; Get size of code portion of overlay segment
25
26 022312 016502 000000G    MOV     O. SIZ(R5), R2 ;Get # words needed by code portion of seg
27 022316 006302           ASL     R2 ;Convert to # bytes
28
29         ; See if this is an optional segment that we need to deal with specially
30
31 022320 012700 022346'    MOV     #OVLST, R0 ;Point to overlay name list
32 022324 020420 1$:      CMP     R4, (R0)+ ;Found name of overlay?
33 022326 001406           BEQ     2$ ;Br if yes
34 022330 005720           TST     (R0)+ ;No -- Skip over address word
35 022332 020027 022436'    CMP     R0, #OVLEND ;Checked all names in the list?
36 022336 103772           BLO     1$ ;Loop if not
37 022340 000137 022756'    JMP     OXYYES ;Load this overlay
38
39         ; Branch off to processing routine
40
41 022344 000130 2$:      JMP     @(R0)+ ;Enter processing routine for the overlay
42
43         ; Table of overlay names and processing routines
44
45         .MACRO  OVLSTBL  NAME
46         .RAD50  /'NAME'/
47         .WORD   OOR'NAME'
48         .ENDM   OVLSTBL
49
50 022346 OVLST:
51 022346 OVLSTBL  USR ;TSUSR -- File management
52 022352 OVLSTBL  SPL ;TSSPOL -- Spooling system
53 022356 OVLSTBL  SP2 ;TSSPL2 -- Spooler flag pages
54 022362 OVLSTBL  LOK ;TSLOCK -- Shared file record locking
55 022366 OVLSTBL  MSG ;TSMMSG -- Inter-job message communication
56 022372 OVLSTBL  SWP ;TSSWAP -- Job swapper
57 022376 OVLSTBL  PLS ;TSPLAS -- PLAS support

```

OPTOVL -- Check for optional system overlay regions

58 022402		OVLTLBL	SLE	;TSSLE	-- Single line editor
59 022406		OVLTLBL	WIN	;TSWIN	-- Display window management
60 022412		OVLTLBL	MIO	;TSMIO	-- Mapped I/O
61 022416		OVLTLBL	CLO	;TSCLO	-- CL handler
62 022422		OVLTLBL	DBG	;TSDEBUG	-- Program debugger
63 022426		OVLTLBL	CSH	;TSCASH	-- Data caching
64 022432		OVLTLBL	DMP	;TSDUMP	-- Crash dump generator
65 022436		OVLEND:			
66					
67					
68					
69 022436	013703	000000G	OORUSR: MOV	VNFC	CSH, R3 ;Get # file cache entries
70 022442	070327	000000G	MUL	#FC##SZ, R3	;Multiply by size of each entry
71 022446	060302		ADD	R3, R2	;Allocate space for directory cache
72 022450	000542		BR	OOXYES	;Load the segment
73					
74					
75					
76 022452	005727	000000G	OORSPL: TST	#SPLND	;Are there any spooled devices?
77 022456	001534		BEQ	OOXNO	;Br if not
78 022460	062702	000000C	ADD	#<SPLNB*512.>, R2	;Reserve room for spool buffers
79 022464	013703	000000G	MOV	NSPLBL, R3	;Get # blocks for spool file
80 022470	062703	000007	ADD	#7, R3	;Bound up to byte boundary
81 022474	072327	177775	ASH	#-3, R3	;Divide by 8 to get # bytes for table
82 022500	005203		INC	R3	;Round up to word boundary
83 022502	042703	000001	BIC	#1, R3	
84 022506	060302		ADD	R3, R2	;Add space for spool file allocation table
85 022510	000522		BR	OOXYES	;Load the segment
86					
87					
88					
89 022512	005727	000000G	OORSP2: TST	#SPLND	;Are there any spooled devices?
90 022516	001514		BEQ	OOXNO	;If not, don't load overlay
91 022520	000516		BR	OOXYES	;Load the segment
92					
93					
94					
95 022522	005737	000000G	OORLOK: TST	VMXSF	;Any shared files?
96 022526	001510		BEQ	OOXNO	;Br if not
97 022530	005737	000000G	TST	VNUMDC	;Shared file data caching wanted?
98 022534	001110		BNE	OOXYES	;Br if yes
99 022536	162702	000000G	SUB	#DCCSIZ, R2	;Reduce size of segment - Leave out cache code
100 022542	000505		BR	OOXYES	;Load the segment
101					
102					
103					
104 022544	013703	000000G	OORMSG: MOV	VMAXMC, R3	;Is message communication facility wanted?
105 022550	001477		BEQ	OOXNO	;Br if not
106 022552	070327	000000G	MUL	#MB##SZ, R3	;Space for message channel blocks
107 022556	060302		ADD	R3, R2	
108 022560	013703	000000G	MOV	VMXMRB, R3	;Number of message request blocks
109 022564	070327	000000G	MUL	#MR##SZ, R3	;Times size of request block
110 022570	060302		ADD	R3, R2	
111 022572	013703	000000G	MOV	VMSCHR, R3	;Max # chars in a message
112 022576	005203		INC	R3	;Bound up to word
113 022600	042703	000001	BIC	#1, R3	;Reserve whole number of words
114 022604	062703	000000G	ADD	#MU#TXT, R3	;Plus space for message header

```

115 022610 070337 000000G          MUL      VMXMSG,R3      ;Times maximum number of messages
116 022614 060302          ADD      R3,R2                ;Space for message buffers
117 022616 000457          BR       OOXYES
118                                ;
119                                ; PLAS support
120                                ;
121 022620 013703 000000G  OORPLS: MOV      VPLAS,R3      ;PLAS support wanted?
122 022624 001451          BEQ      OOXNO                ;Br if not
123 022626 062703 000021          ADD      #17.,R3        ;Bound up # blocks
124 022632 072327 177775          ASH      #-3,R3        ;Get # bytes needed for swap file bit map
125 022636 060302          ADD      R3,R2                ;Reserve room for swap file bit map
126 022640 000446          BR       OOXYES                ;Load the segment
127                                ;
128                                ; Job swapper
129                                ;
130 022642 105737 000000G  OORSWP: TSTB     VSWPFL        ;Is this a swapping system?
131 022646 001440          BEQ      OOXNO                ;Br if not
132 022650 000442          BR       OOXYES                ;Br if yes -- Load the segment
133                                ;
134                                ; Single line editor
135                                ;
136 022652 105737 000000G  OORSLE: TSTB     VSLEDT        ;Is SL editor wanted?
137 022656 001434          BEQ      OOXNO                ;Br if not
138 022660 000436          BR       OOXYES                ;Load the segment
139                                ;
140                                ; Display windows
141                                ;
142 022662 013703 000000G  OORWIN: MOV      VMXWIN,R3      ;Are any display windows wanted?
143 022666 001430          BEQ      OOXNO                ;Br if not
144 022670 070327 000000G          MUL      #DW##SZ,R3    ;Amt of space needed for window control blks
145 022674 060302          ADD      R3,R2                ;Add to size of overlay
146 022676 000427          BR       OOXYES                ;Load the segment
147                                ;
148                                ; Mapped I/O
149                                ;
150 022700 105737 000000G  OORMIO: TSTB     MIOFLG        ;Is I/O mapping needed?
151 022704 001421          BEQ      OOXNO                ;Br if not
152 022706 000423          BR       OOXYES                ;Load the segment
153                                ;
154                                ; CL handler
155                                ;
156 022710 005727 000000G  OORCLO: TST      #CLTOTL       ;Any I/O lines?
157 022714 001415          BEQ      OOXNO                ;Br if not
158 022716 000417          BR       OOXYES                ;Yes, load the segment
159                                ;
160                                ; Program debugger
161                                ;
162 022720 105737 000000G  OORDBG: TSTB     VDBFLG        ;Is the program debugger wanted?
163 022724 001411          BEQ      OOXNO                ;Br if not
164 022726 000413          BR       OOXYES                ;Load this segment
165                                ;
166                                ; Data caching
167                                ;
168 022730 005737 000000G  OORCSH: TST      CSHALC        ;Is data caching wanted?
169 022734 001405          BEQ      OOXNO                ;Br if not
170 022736 000407          BR       OOXYES                ;Load this segment
171                                ;

```

```

172      ; Crash dump generator
173      ;
174 022740 10573/ 0000000  OORDMP: TSTB    VSYDMP    ; Is dump facility wanted?
175 022744 001401          BEQ      OOXNO      ; Br if not
176 022746 000403          BR       OOXYES     ; Br if yes
177      ;
178      ; Don't load this segment
179      ;
180 022750 005002  OOXNO:  CLR      R2          ; Say no space needed for overlay
181 022752 000261          SEC              ; Signal don't load the segment
182 022754 000415          BR       OOXFIN
183      ;
184      ; Load this segment
185      ;
186 022756 005202  OOXYES:  INC      R2          ; Make sure size is even
187 022760 042702          BIC      #1,R2
188 022764 020227 020000          CMP      R2,#8192. ; Don't allow code + data to exceed 8Kb
189 022770 101402          BLOS     1$         ; Br if ok
190 022772 012702 020000          MOV      #8192.,R2 ; Note, init code in segment will truncate dat
191 022776 062702 000077 1$:  ADD      #63.,R2    ; Convert to # 64-byte blocks
192 023002 072227 177772          ASH      #-6,R2
193 023006 000241          CLC              ; Signal to load the segment
194      ;
195      ; Finished
196      ;
197 023010 012605  OOXFIN:  MOV      (SP)+,R5
198 023012 012604          MOV      (SP)+,R4
199 023014 012603          MOV      (SP)+,R3
200 023016 000207          RETURN

```

```

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 023020 010346 OVLTRY: MOV     R3,-(SP)
15         ;
16         ; Begin loop to examine all segments
17         ;
18 023022 005002          CLR     R2          ; Say we haven't found any segment yet
19 023024 012703 000516'  MOV     #OSTABL,R3    ; Point to entry for 1st segment
20 023030 005763 000000  1$:  TST     OS$SIZ(R3) ; Is this segment to be loaded?
21 023034 001415          BEQ     2$          ; Br if not
22 023036 005763 000002  TST     OS$FLG(R3)    ; Is this segment already over TSINIT?
23 023042 001012          BNE     2$          ; Br if yes
24 023044 026305 000000  CMP     OS$SIZ(R3),R5 ; Will this segment fit?
25 023050 101007          BHI     2$          ; Br if not
26 023052 005702          TST     R2          ; Have we found any other seg yet?
27 023054 001404          BEQ     3$          ; Br if not
28 023056 026362 000000 000000  CMP     OS$SIZ(R3),OS$SIZ(R2) ; Is new seg larger than old?
29 023064 101401          BLOS   2$          ; Br if not
30 023066 010302 3$:  MOV     R3,R2          ; Remember largest segment
31 023070 062703 000006  2$:  ADD     #OS$$SZ,R3 ; Point to entry for next segment
32 023074 020337 000744'  CMP     R3,OSLAST ; Have we checked all segments?
33 023100 103753          BLO     1$          ; Loop if not
34         ;
35         ; Finished
36         ;
37 023102 000241          CLC          ; Assume we found a segment
38 023104 005702          TST     R2          ; Did we find a segment that will fit?
39 023106 001001          BNE     9$          ; Br if yes
40 023110 000261          SEC          ; Signal failure on return
41 023112 012603 9$:  MOV     (SP)+,R3
42 023114 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 023116   GETOVL:
13         ;
14         ;   Allocate space for the overlay segment
15         ;
16 023116   SUB     OS$SIZ(R2),R5    ;Allocate space for overlay
17 023122   CMP     R5,#1600         ;Are we about to run over RT-11?
18 023126   BLO     10$             ;Br if yes -- Insufficient memory
19         ;
20         ;   Remember the base address of some key segments
21         ;
22 023130   CALL    KEYSEG          ;Remember address of some segments
23         ;
24         ;   Load the segment
25         ;
26 023134   CALL    LODOVL         ;Load the segment
27         ;
28         ;   Finished
29         ;
30 023140   RETURN
31         ;
32         ;   Error: Memory overflow
33         ;
34 023142   10$:   .PRINT #TSXHD
35 023150       .PRINT #TSXSIZ
36 023156   JMP     INISTP

```

```

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 023162    010146    LODOVL: MOV      R1, -(SP)
10 023164    010246      MOV      R2, -(SP)
11 023166    010346      MOV      R3, -(SP)
12 023170    010446      MOV      R4, -(SP)
13 023172    010546      MOV      R5, -(SP)
14
15          ;
16          ; Get info about size of the overlay and position within SAV file
17          ;
18 023174    016201    000004      MOV      OS$OVL(R2), R1    ;Get pointer to linker-built segment entry
19 023200    016103    000000G      MOV      O.SIZ(R1), R3    ;Get size of overlay segment (# words)
20 023204    016137    000000G 000142'  MOV      O.BLK(R1), FILBLK ;Get block in SAV file where segment starts
21 023212    010302      MOV      R3, R2    ;Get total number of words in segment
22 023214    062702    000377      ADD      #255., R2        ;round to the nearest number of blocks
23 023220    000302      SWAB      R2        ;Divide by 256. words per segment
24 023222    042702    177400      BIC      #177400, R2      ;kill sign extension bits
25 023226    010561    000000G      MOV      R5, O.PAR(R1)    ;Remember where segment is being loaded
26
27          ; Read next block of overlay segment into low-memory buffer
28          ;
29 023232    013704    000152'  10$: MOV      WRKBUF, R4      ;Point to work buffer
30 023236      .READW  #AREA, #17, R4, #256., FILBLK    ;read a block
31 023274    103466      BCS      22$      ;read error occurred
32
33          ; Move from low buffer to high position in memory
34          ;
35 023276    012701    000000G      MOV      #VPAR5, R1      ;get the virtual address of the mapped region
36 023302    012700    000400      MOV      #256., R0        ;obtain the number of words to move
37 023306    020300      CMP      R3, R0        ;Do we need to move as many as 256 words?
38 023310    103001      BHIS     2$          ;Br if yes
39 023312    010300      MOV      R3, R0        ;Get number of words to move for last block
40 023314    160003      2$: SUB      R0, R3      ;Get number of words left after this move
41 023316      DISABL      ;** Disable interrupts **
42 023324    013746    000000G      MOV      @#KPAR5, -(SP)   ;save the contents of the mapping register
43 023330    010537    000000G      MOV      R5, @#KPAR5      ;change the mapping register
44 023334    052737    000000G 000000G  BIS      #MMENBL, @#SR0MMR ;enable memory management
45 023342    105737    000000G      TSTB     MEM256          ;Does machine have at least 256Kb of memory?
46 023346    001403      BEQ      11$          ;Br if not
47 023350    052737    000000G 000000G  BIS      #EMMAP, @#SR3MMR ;enable extended memory addressing
48 023356    012421      11$: MOV      (R4)+, (R1)+      ;move into high memory
49 023360    077002      SOB      R0, 11$
50 023362    105737    000000G      TSTB     MEM256          ;Does this machine have at least 256Kb?
51 023366    001403      BEQ      12$          ;Br if not
52 023370    042737    000000G 000000G  BIC      #EMMAP, @#SR3MMR ;disable extended memory management
53 023376    042737    000000G 000000G 12$: BIC      #MMENBL, @#SR0MMR ;disable memory management
54 023404    012637    000000G      MOV      (SP)+, @#KPAR5   ;restore the mapping register
55 023410      ENABL      ;** Enable interrupts **
56 023416    062705    000010      ADD      #10, R5          ;advance 64-byte block # by 512-bytes
57 023422    005237    000142'      INC      FILBLK          ;increment file block #

```

LODOVL -- Read and relocate system overlay

```
58 023426 005302          DEC    R2          ;More to be copied?
59 023430 001402          BEQ     5$          ;Br if not
60 023432 000137 023232'  JMP     10$          ;Read and copy rest of mapped segment
61                                     ;
62                                     ; Finished loading the segment
63                                     ;
64 023436 012605 5$:      MOV     (SP)+,R5
65 023440 012604          MOV     (SP)+,R4
66 023442 012603          MOV     (SP)+,R3
67 023444 012602          MOV     (SP)+,R2
68 023446 012601          MOV     (SP)+,R1
69 023450 000207          RETURN
70                                     ;
71                                     ; Error occurred on read
72                                     ;
73 023452          22$:    .PRINT  #TSXHD      ;Print heading
74 023460          .PRINT  #RDERR      ;Read error
75 023466 000137 004134'  JMP     INISTP     ;Abort initialization
```

```

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 023472 010146 GETSRT: MOV     R1, -(SP)
14 023474 010246     MOV     R2, -(SP)
15 023476 010346     MOV     R3, -(SP)
16 023500 010446     MOV     R4, -(SP)
17         ;
18         ; See if this is a dummy run-time entry to allow for patching
19         ;
20 023502 021127 000000G     CMP     (R1), #DMYDEV    ; Dummy run-time entry?
21 023506 001540     BEQ     7$                    ; Br if yes
22         ;
23         ; Try to open a channel to run-time file
24         ;
25 023510         .LOOKUP #AREA, #1, R1    ; OPEN CHANNEL TO RUN-TIME FILE
26 023526 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 023530 105737 000000G     TSTB    VINABT          ; ABORT OR CONTINUE
32 023534 001132     BNE     9$                    ; BR IF ABORT WANTED
33 023536 005061 000000G     CLR     RT$NAM(R1)      ; Mark run-time as not-available
34 023542 005061 000002G     CLR     RT$NAM+2(R1)
35 023546 000520     BR      7$                    ; GO LOAD NEXT RUN-TIME SYSTEM
36         ;
37         ; Set up information about position of run-time in physical memory
38         ;
39 023550 116102 000000G 8$:  MOVB    RT$SKP(R1), R2    ; GET # BLOCKS TO SKIP AT FRONT OF RUN-TIME
40 023554 042702 177400     BIC     #^C377, R2      ; CLEAR SIGN EXTENSION
41 023560 160200     SUB     R2, R0                ; GET # BLOCKS TO READ (LOOKUP SET ROW SIZE)
42 023562 010561 000000G     MOV     R5, RT$TOP(R1)  ; SET 64-BYTE BLOCK # ABOVE TOP OF RUN-TIME
43 023566 010003     MOV     R0, R3                ; GET # 512-BYTE BLOCKS IN RUN-TIME
44 023570 072027 000003     ASH     #3, R0          ; CONVERT TO # 64-BYTE BLOCKS
45 023574 160005     SUB     R0, R5                ; CALCULATE BASE 64-BYTE BLOCK # OF RUN-TIME
46 023576 020527 001600     CMP     R5, #1600      ; ARE WE ABOUT TO RUN OVER RT-11?
47 023602 103530     BLO     11$                    ; BR IF YES
48 023604 010561 000000G     MOV     R5, RT$BAS(R1)  ; SET BASE 64-BYTE BLOCK # OF RUN-TIME
49         ;
50         ; Read run-time system into memory and position in high-memory
51         ;
52 023610 010546     MOV     R5, -(SP)              ; Save address of bottom of run-time
53 023612 013704 000152' 4$:  MOV     WRKBUF, R4      ; Point to work buffer
54 023616         .READW #AREA, #1, R4, #256, R2    ; READ A BLOCK OF RUN-TIME FILE
55         ; Use memory management to access high-memory area.
56 023652 012701 000000G     MOV     #VPAR6, R1     ; GET VIRTUAL ADDRESS OF PAR6 ADDRESS REGION
57 023656 010537 000000G     MOV     R5, @#UPAR6    ; SET USER-MODE PAR6 MAP OFFSET VALUE

```

```

58 023662 012737 077406 000000G      MOV      #077406,@#UPDR6 ;SET PDR TO ALLOW FULL ACCESS TO PAGE
59 023670 052737 000000G 000000G      BIS      #UPMODE,@#PSW ;SET PREVIOUS-MODE = USER FOR MTPD ACCESS
60 023676 012700 000400                MOV      #256.,R0 ;GET # WORDS TO MOVE
61 023702                DISABL                ;** Disable interrupts **
62 023710 052737 000000G 000000G      BIS      #MMENBL,@#SROMMR;enable memory management
63 023716 105737 000000G                TSTB     MEM256 ;DOES THIS MACHINE HAVE AT LEAST 256KB?
64 023722 001403                BEQ      3$ ;BR IF NOT
65 023724 052737 000000G 000000G      BIS      #EMMAP,@#SR3MMR ;enable extended memory addressing
66 023732 012446                3$:      MOV      (R4)+,-(SP) ;TRANSFER DATA FROM BUFFER TO HIGH MEMORY
67 023734 106621                MTPD      (R1)+
68 023736 077003                SOB      R0,3$
69 023740 105737 000000G                TSTB     MEM256 ;DOES THIS MACHINE HAVE AT LEAST 256KB?
70 023744 001403                BEQ      31$ ;BR IF NOT
71 023746 042737 000000G 000000G      BIC      #EMMAP,@#SR3MMR ;DISABLE EXTENDED MEMORY MANAGEMENT
72 023754 042737 000000G 000000G 31$:    BIC      #MMENBL,@#SROMMR;DISABLE MEMORY MANAGEMENT
73 023762                ENABL                ;** Enable interrupts **
74 023770 062705 000010                ADD      #10,R5 ;ADVANCE 64-BYTE BLOCK # BY 512-BYTES
75 023774 005202                INC      R2 ;INC FILE BLOCK #
76 023776 077373                SOB      R3,4$ ;READ AND COPY REST OF FILE
77                                ;
78                                ; Finished loading the run-time system.
79                                ;
80 024000 012605                MOV      (SP)+,R5
81 024002                .CLOSE #1
82                                ;
83                                ; Finished
84                                ;
85 024010 012604                7$:      MOV      (SP)+,R4
86 024012 012603                MOV      (SP)+,R3
87 024014 012602                MOV      (SP)+,R2
88 024016 012601                MOV      (SP)+,R1
89 024020 000207                RETURN
90                                ;
91                                ; Error -- Cannot find run-time system file
92                                ;
93 024022                9$:      .PRINT #TSXHD ;PRINT MESSAGE HEADING
94 024030                .PRINT #COSRT ;PRINT ERROR MESSAGE
95 024036 012702 000004                MOV      #4,R2 ;PRINT 4 RAD50 VALUES
96 024042 012100                10$:     MOV      (R1)+,R0 ;GET PART OF NAME
97 024044 004737 025274'          CALL      PRTR50 ;PRINT RAD50 VALUE
98 024050 077204                SOB      R2,10$
99 024052                .PRINT #CRLF ;END LINE
100 024060 000137 004134'          JMP      INISTP ;ABORT INITIALIZATION
101                                ;
102                                ; Error -- Insufficient memory space to load run-time systems
103                                ;
104 024064                11$:      .PRINT #TSXHD ;PRINT HEADING
105 024072                .PRINT #SRTOVF ;PRINT ERROR MESSAGE
106 024100 000137 004134'          JMP      INISTP ;ABORT INITIALIZATION

```

```

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 024104   010246   CSHBUF: MOV      R2, -(SP)
13 024106   010346           MOV      R3, -(SP)
14         ;
15         ; See if data caching is wanted
16         ;
17 024110   013737   000000G 000000G      MOV      CSHALC, VCSHNB      ;Set # blocks in use = # blocks allocated
18 024116   013702   000000G      MOV      CSHALC, R2      ;Did used request data caching?
19 024122   001464           BEQ      9$      ;Br if not
20         ;
21         ; Calculate number of 64-byte blocks needed for each cache control table
22         ;
23 024124   062702   000037      ADD      #31., R2      ;Bound up to 32 word block
24 024130   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 024134   013703   000000G      MOV      CSHALC, R3      ;Get # blocks in cache
29 024140   072327   000003      ASH      #3, R3      ;Get # 64-byte blks used by cache data buffers
30 024144   012700   000010      MOV      #8., R0      ;Get number of cache control tables
31 024150   060203   1$:      ADD      R2, R3      ;Accumulate total space needed
32 024152   077002      SOB      R0, 1$
33 024154   010337   000000G      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 024160   010500      MOV      R5, R0      ;Get top of memory address
38 024162   160400      SUB      R4, R0      ;Compute # free 64-byte blocks
39 024164   020300      CMP      R3, R0      ;Is there enough total space?
40 024166   103045      BHIS     10$      ;Br if not
41         ;
42         ; Allocate space for cache data buffers
43         ;
44 024170   013700   000000G      MOV      CSHALC, R0      ;Get # blocks in data cache
45 024174   072027   000003      ASH      #3, R0      ;Get # 64-byte blocks needed for allocation
46 024200   160005      SUB      R0, R5      ;Allocate space for cache data buffers
47 024202   010537   000000G      MOV      R5, CSHBFP      ;Save pointer to base of buffer area
48         ;
49         ; Allocate space for each control table
50         ;
51 024206   160205      SUB      R2, R5      ;Allocate space for table
52 024210   010537   000000G      MOV      R5, CA$BLK      ;Block number associated with entry
53 024214   160205      SUB      R2, R5      ;Allocate space for table
54 024216   010537   000000G      MOV      R5, CA$DVU      ;Device and unit number
55 024222   160205      SUB      R2, R5      ;Allocate space for table
56 024224   010537   000000G      MOV      R5, CA$WCT      ;Number of words
57 024230   160205      SUB      R2, R5      ;Allocate space for table

```

CSHBUF -- Allocate space for data cache tables

```
58 024232 010537 000000G      MOV      R5,CA#UFL      ;LRU chain forward link
59 024236 160205              SUB      R2,R5      ;Allocate space for table
60 024240 010537 000000G      MOV      R5,CA#UBL      ;LRU chain backward link
61 024244 160205              SUB      R2,R5      ;Allocate space for table
62 024246 010537 000000G      MOV      R5,CA#HFL      ;Hash chain forward link
63 024252 160205              SUB      R2,R5      ;Allocate space for table
64 024254 010537 000000G      MOV      R5,CA#HBL      ;Hash chain backward link
65 024260 160205              SUB      R2,R5      ;Allocate space for table
66 024262 010537 000000G      MOV      R5,CA#HSH      ;Hash chain list heads
67 024266 020527 001600      CMP      R5,#1600      ;Did we run over RT-11?
68 024272 101403              BLOS     10$          ;Br if yes
69                               ;
70                               ;   Finished
71                               ;
72 024274 012603      9$:      MOV      (SP)+,R3
73 024276 012602              MOV      (SP)+,R2
74 024300 000207              RETURN
75                               ;
76                               ;   Insufficent memory space available for cache data
77                               ;
78 024302              10$:      .PRINT  #TSXHD      ;Print heading
79 024310              .PRINT  #CSHOFV      ;Overflow message
80 024316 000137 004134'      JMP      INISTP      ;Abort the initialization
```

```

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     GETODT: MOV     R1,-(SP)
14             MOV     R2,-(SP)
15             MOV     R3,-(SP)
16             MOV     R4,-(SP)
17     ;
18     ; Try to lookup ODT rel file.
19     ;
20     . LOOKUP  #AREA,#1,#ODTBLK ; LOOKUP ODT REL FILE
21     BCC      1$                ; BR IF FOUND IT
22     . PRINT  #TSXHD            ; CAN'T FIND ODT
23     . PRINT  #NOODT
24     JMP      INISTP            ; ABORT INITIALIZATION
25     ;
26     ; Read first block of ODT file and determine size of ODT.
27     ;
28     1$:      ADD     #200.,R5    ; RESERVE SPACE FOR ODT STACK
29             MOV     R5,R0        ; CHECK MEMORY ADDRESS FOR OVERFLOW
30             ADD     #512.,R0
31             CALL    CHKMEN
32             . READW  #AREA,#1,R5,#256.,#0
33             BCC     2$          ; Br if no read error
34             JMP     ODTRDX       ; Read error
35     2$:      MOV     RSZ(R5),R2   ; GET SIZE OF ODT
36             MOV     R2,R3
37             ADD     R5,R3        ; GET ADDRESS ABOVE TOP OF ODT
38             MOV     R3,R0        ; CHECK MEMORY ADDRESS FOR OVERFLOW
39             CALL    CHKMEN
40             CLC
41             ROR     R2           ; GET # WORDS IN ODT
42     ; Get starting address of ODT
43             MOV     STA(R5),R0   ; GET OFFSET TO START ADDRESS
44             SUB     #ODTBAS,R0   ; CALCULATE ABSOLUTE STARTING ADDRESS
45             ADD     R5,R0
46             MOV     R0,ODTSTA    ; THIS IS REAL STARTING ADDRESS
47             MOV     RBD(R5),R1   ; GET # OF BLOCK WITH RELOCATION INFO
48     ;
49     ; Read in ODT rel file image.
50     ;
51     . READW  #AREA,#1,R5,R2,#1
52     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      RELFIL: MOV      R3,ODTTOP      ;SAVE ADDRESS ABOVE TOP OF ODT
59      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      MOV      WRKBUF,RLBF            ;READ RELOCATION INFO HERE
65      1$:      MOV      RLBF,RLBFND
66      ADD      #1024.,RLBFND          ;GET ADDRESS OF END OF BUFFER AREA
67      MOV      R5,R4                  ;GET BASE ADDRESS OF ODT
68      SUB      #ODTBAS,R4             ;SUBTRACT LINK BASE ADDRESS
69      ; Read in relocation address list.
70      4$:      . READW  #AREA,#1,RLBF,#512.,R1
71      BCC      7$                     ;BR IF NO READ ERROR
72      TSTB     @#52                   ;END OF FILE IS OK
73      BNE      ODTRDX                 ;BR IF READ ERROR
74      ; Relocate some addresses in ODT.
75      7$:      MOV      RLBF,R2        ;POINT TO RELOCATION INFO
76      3$:      MOV      (R2)+,R3        ;GET ADDRESS OF LOCATION TO RELOCATE
77      CMP      R3,#-2                 ;TIME TO STOP?
78      BEQ      9$                     ;BR IF FINISHED
79      MOV      (R2)+,R0                ;GET VALUE TO RELOCATE
80      ASL      R3                      ;CVT TO BYTE ADDRESS
81      BCC      5$                     ;BR IF ADDITIVE RELOCATION
82      SUB      R4,R0                  ;RELOCATE THE ADDRESS
83      BR       6$
84      5$:      ADD      R4,R0            ;RELOCATE THE ADDRESS
85      6$:      ADD      R5,R3            ;GET LOCATION WHERE WORD GOES
86      MOV      R0,@R3                 ;STORE RELOCATED ADDRESS
87      CMP      R2,RLBFND              ;TIME TO READ NEXT BUFFER FULL?
88      BLO      3$                     ;BR IF NOT
89      ADD      #2,R1                  ;ADVANCE BLOCK #
90      BR       4$                     ;GO READ NEXT BUFFER FULL
91      ;
92      ; Finished relocation.
93      ; Close ODT rel file.
94      ;
95      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     MOV      #TSINIT,R0
113     MOV      #BRKPT,R1
114     MOV      #TSGEN,R2

```

CSHBUF -- Allocate space for data cache tables

```
115             MOV     #TSEEXEC,R3
116             MOV     #TSEMT,R4
117             MOV     #SYSMAP,-(SP) ;PASS ADDRESS OF SYSMAP CELL TO ODT
118             MOV     #MAPSYS,-(SP) ;PASS ADDRESS OF MAPSYS ROUTINE
119             MOV     #10$,R5      ;ADDRESS FOR ODT TO RETURN TO
120             ;
121             ; Enter ODT
122             ;
123             JMP      @ODTSTA      ;JUMP TO START OF ODT
124             ;
125             ; Return from ODT.
126             ; Continue initialization of TSX.
127             ;
128 10$:         MOV     @#14,ODTTRP  ;SAVE ODT BREAKPOINT ENTRY ADDRESS
129             MOV     ODTTOP,R5    ;ADDRESS ABOVE TOP OF ODT
130             MOV     (SP)+,R4
131             MOV     (SP)+,R3
132             MOV     (SP)+,R2
133             MOV     (SP)+,R1
134             RETURN
135             ;
136             ; Error while reading ODT rel file.
137             ;
138 ODTRDX:      .PRINT  #TSXHD      ;PRINT ERROR MESSAGE
139             .PRINT  #ODTRDM
140             JMP      INISTP      ;ABORT INITIALIZATION
141             ;
142             ; RTI instruction to disable interrupts
143             ;
144 DORTI:       RTI
145             .ENDC      ;EQ,PROCID
```

```

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 024322 010146 OPNCHN: MOV     R1,-(SP)
15 024324 010346      MOV     R3,-(SP)
16 024326 010003      MOV     R0,R3          ; Carry channel block address in R3
17         ;
18         ; Initialize the channel block
19         ;
20 024330 010301      MOV     R3,R1          ; Point to the channel block
21 024332 012700 000000C      MOV     #<CHNSIZ/2>,R0 ; Get # words to zero
22 024336 005021 2$:      CLR     (R1)+      ; Zero the channel block
23 024340 077002      SOB     R0,2$
24 024342 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 024350 010200      MOV     R2,R0          ; Get the full device name
29 024352 004737 011476'      CALL    CVTDVU      ; Convert to dev # and unit #
30 024356 103411      BCS     9$          ; Br if we don't recognize the device name
31 024360 010001      MOV     R0,R1          ; Get index # and unit #
32 024362 000301      SWAB    R1          ; Get unit # to low byte
33 024364 110163 000000G      MOVB   R1,C.DEVQ(R3) ; Set unit # in channel block
34 024370 042700 000000C      BIC     #^C<CS$NMX>,R0 ; Clear all but device index number in R0
35 024374 050063 000000G      BIS     R0,C.CSW(R3) ; Store device index # into CSW
36         ;
37         ; Success
38         ;
39 024400 000241      CLC          ; Signal success on return
40         ;
41         ; Finished
42         ;
43 024402 012603 9$:      MOV     (SP)+,R3
44 024404 012601      MOV     (SP)+,R1
45 024406 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 024410 010146 SETCHN: MOV      R1, -(SP)
19 024412 010246      MOV      R2, -(SP)
20 024414 010346      MOV      R3, -(SP)
21 024416 010001      MOV      R0, R1          ; GET ADDRESS OF TSX CHANNEL BLOCK
22
23         ; Do .SAVESTATUS to store channel information into TSX channel block.
24         ;
25 024420      .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 024436 011103      MOV      (R1), R3          ; GET CSW FOR CHANNEL
31 024440 042703 177701 BIC      #^C76, R3      ; GET RT-11 DEVICE INDEX NUMBER
32 024444      .GVAL    #AREA, #404          ; GET RT-11 OFFSET TO PNAME TABLE
33 024464 060003      ADD      R0, R3          ; GET ADDRESS OF NAME OF DEVICE IN PNAME TABLE
34 024466      .GVAL    #AREA, R3          ; GET NAME OF DEVICE FROM RT-11
35 024504 013703 0000000 MOV      NUMDEV, R3      ; GET INDEX # FOR LAST TSX DEVICE
36 024510 020063 0000000 1$: CMP      R0, PNAME(R3) ; LOOK FOR DEVICE IN OUR TABLES
37 024514 001404      BEQ      2$          ; BR IF FOUND
38 024516 162703 0000002 SUB      #2, R3          ; CHECK NEXT ENTRY
39 024522 002372      BGE      1$          ; BR IF MORE TO CHECK
40 024524 000407      BR       MTSXDV        ; VERY STRANGE THAT WE DIDN'T FIND IT
41 024526 042711 0000076 2$: BIC      #76, (R1)    ; CLEAR OUT RT-11 DEVICE #
42 024532 050311      BIS      R3, (R1)        ; STORE TSX DEVICE #
43
44         ; Finished
45         ;
46 024534 012603      MOV      (SP)+, R3
47 024536 012602      MOV      (SP)+, R2
48 024540 012601      MOV      (SP)+, R1
49 024542 000207      RETURN
50
51         ; Error: Could not locate Rt-11 device number in TSX device table.
52         ;
53 024544      MTSXDV: .PRINT  #TSXHD          ; PRINT ERROR MESSAGE
54 024552      .PRINT  #REQMIS          ; Missing a required device
55 024560 010200      MOV      R2, R0          ; GET RAD50 DEVICE NAME
56 024562 004737 025274' CALL      PRTR50        ; DISPLAY DEVICE NAME
57 024566      .PRINT  #CRLF

```

TSINIT -- TSX startup initializ MACRO V05.05 Friday 20-Jan-89 11:30 Page 79-1
SETCHN -- Copy RT-11 channel information into TSX system chan

58 024574 000137 004134'

JMP INISTP

;ABORT INITIALIZATION

```

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 024600   010146  SETSY:  MOV     R1,-(SP)
15 024602   010246      MOV     R2,-(SP)
16         ;
17         ; Set up system device unit number
18         ;
19 024604          .GVAL   #AREA,#274      ;Get system unit # from RT-11 (high byte)
20 024624   010037   000000G      MOV     R0,SYUNIT      ;Set system unit number
21         ;
22         ; Set up system device index number
23         ;
24 024630          .GVAL   #AREA,#364      ;Get RT-11 system device index number
25 024650   010002      MOV     R0,R2      ;Save device index number
26 024652          .GVAL   #AREA,#404      ;Get offset within RMON of PNAME table
27 024672   060002      ADD     R0,R2      ;Get offset to name of SY device
28 024674          .GVAL   #AREA,R2      ;Get name of RT-11 system device
29 024712   013701   000000G      MOV     NUMDEV,R1      ;Get index to last TSX-Plus device entry
30 024716   020061   000000G  1$:  CMP     R0,PNAME(R1)    ;Search for device in TSX tables
31 024722   001405      BEQ     2$        ;Br if found it
32 024724   162701   000002      SUB     #2,R1      ;Keep looking if more
33 024730   002372      BGE     1$
34 024732   010002      MOV     R0,R2      ;Save name of system device
35 024734   000703      BR      MTSXDV      ;Missing device error
36 024736   010137   000000G  2$:  MOV     R1,SYINDX      ;Store index # of TSX-Plus system device
37         ;
38         ; Set up RAD50 name of SY disk
39         ;
40 024742   113702   000001G      MOVB    SYUNIT+1,R2    ;GET SYSTEM UNIT NUMBER
41 024746   062702   000036      ADD     #36,R2      ;PUT IN "0" AS 3'RD CHARACTER OF NAME
42 024752   066102   000000G      ADD     PNAME(R1),R2    ;ADD DEVICE NAME
43 024756   010237   000000G      MOV     R2,SYNAME      ;THIS IS THE FULL SY DISK NAME
44         ;
45         ; Finished
46         ;
47 024762   012602      MOV     (SP)+,R2
48 024764   012601      MOV     (SP)+,R1
49 024766   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         ; R0 = 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 024770 010046 RTFTCH: MOV     R0,-(SP)
18 024772 010246      MOV     R2,-(SP)
19 024774 010546      MOV     R5,-(SP)
20         ;
21         ; Set the name of the device being fetched
22         ;
23 024776 010037 000130'      MOV     R0,FETDEV      ;Set name of device whose handler to fetch
24         ;
25         ; Do a .DSTAT to get information about the handler
26         ;
27 025002      .DSTAT  #DSTBLK,#FETDEV ;Get information about the device handler
28 025014 103425      BCS     9$          ;Br if device not recognized
29         ;
30         ; Determine if the handler is currently resident
31         ;
32 025016 005737 000116'      TST     DSTBLK+4      ;Is the handler resident now?
33 025022 001021      BNE     8$          ;Br if yes
34         ;
35         ; The handler is not currently resident.
36         ; See if it will fit in WRKBUF.
37         ;
38 025024 013702 000152'      MOV     WRKBUF,R2      ;Set address where handler will be loaded
39 025030 013700 000114'      MOV     DSTBLK+2,R0    ;Get the size of the handler
40 025034 020037 000154'      CMP     R0,WRKSIK      ;Will handler fit in WRKBUF?
41 025040 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 025042 060500      ADD     R5,R0          ;Get address above top of area needed
47 025044 020037 000132'      CMP     R0,TOPMEM      ;Is there room for handler?
48 025050 101013      BHI     10$         ;Br if not
49 025052 010502      MOV     R5,R2          ;Set address where handler is to be loaded
50         ;
51         ; Fetch the handler
52         ;
53 025054      1$: .FETCH  R2,#FETDEV      ;Try to fetch the handler
54 025064 103401      BCS     9$          ;Br if error on fetch
55         ;
56         ; We successfully fetched the handler
57         ;

```

TSINIT -- TSX startup initializ MACRO V05.05 Friday 20-Jan-89 11:30 Page 81-1
RTFTCH -- Fetch a RT-11 device handler

```
58 025066 000241      8$:      CLC                      ; Signal success on return
59                      ;
60                      ; Finished
61                      ;
62 025070 012605      9$:      MOV      (SP)+, R5
63 025072 012602              MOV      (SP)+, R2
64 025074 012600              MOV      (SP)+, R0
65 025076 000207              RETURN
66                      ;
67                      ; Insufficient memory available to load the handler
68                      ;
69 025100 004737 025124' 10$:    CALL    SIZERR          ; Generated system is too big -- abort
```

```

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 025104 020037 000236' CHKMEM: CMP      RO,MEMLIM      ; IS THE ADDRESS OK?
12 025110 103402          BLO      1$              ; BR IF OK
13 025112 004737 025124'          CALL    SIZERR        ; Generated system is too big -- abort
14 025116 004737 025144' 1$:      CALL    CCATST        ; CHECK FOR ^C ABORT REQUEST
15 025122 000207          RETURN
16
17         ;-----
18         ; Generated system is too big. Abort the initialization.
19         ;
20 025124          SIZERR: .PRINT  #TSXHD          ; PRINT MESSAGE HEADING
21 025132          .PRINT  #TOOBIG        ; PRINT ERROR MESSAGE
22 025140 000137 004134'          JMP      INISTP        ; ABORT INITIALIZATION
23
24         ;-----
25         ; Check for control-C and abort initialization if requested.
26         ;
27 025144 005737 000040' CCATST: TST      CCAFLG        ; DID USER REQUEST ^C ABORT?
28 025150 001402          BEQ      1$              ; BRANCH IF NOT
29 025152 000137 004134'          JMP      INISTP        ; ELSE ABORT INITIALIZATION
30 025156 000207          1$:      RETURN

```

```

1
2
3
4
5
6
7
8 025160 010146
9 025162 010246
10 025164 010001
11 025166 012702 000006
12 025172 005000
13 025174 073027 000001
14 025200 000403
15 025202 005000
16 025204 073027 000003
17 025210 062700 000060
18 025214
19 025220 077210
20 025222 012602
21 025224 012601
22 025226 000207

```

```

.SBTTL PRTOCT -- Print octal value
-----
; PRTOCT is called to print an octal value without trailing Cr-Lf.
;
; Inputs:
;   R0 = value to be printed.
;
PRTOCT: MOV     R1, -(SP)
        MOV     R2, -(SP)
        MOV     R0, R1          ; GET VALUE TO PRINT
        MOV     #6, R2          ; PRINT 6 DIGITS
        CLR     R0
        ASHC    #1, R0          ; GET 1ST OCTAL DIGIT (1 BIT)
        BR      2$
1$:     CLR     R0
        ASHC    #3, R0          ; INITIALIZE FOR SHIFT
        ADD     #'0, R0         ; SHIFT AN OCTAL DIGIT INTO R0
        .TTYOUT                     ; CONVERT TO ASCII CHARACTER
        SOB     R2, 1$          ; PRINT THE CHARACTER
        MOV     (SP)+, R2        ; LOOP AND PRINT MORE DIGITS
        MOV     (SP)+, R1
        RETURN

```

1				.SBTTL PRTDEC -- Print decimal value
2				-----
3				; PRTDEC is called to print a decimal value with leading zeroes suppressed
4				; and with no trailing Cr-Lf.
5				;
6				; Inputs:
7				; R0 = Value to be printed
8				;
9	025230	010146		PRTDEC: MOV R1, -(SP)
10	025232	005046		CLR -(SP) ; NULL ON STACK TO STOP US
11				;
12				; Convert value to ascii digit string and stack the digits.
13				;
14	025234	010001		MOV R0, R1 ; GET VALUE TO BE CONVERTED
15	025236	005000		1\$: CLR R0 ; SET HIGH-ORDER PART OF VALUE TO 0
16	025240	071027	000012	DIV #10, R0 ; DIVIDE R0-R1 BY 10.
17	025244	062701	000060	ADD #'0, R1 ; CONVERT REMAINDER TO ASCII DIGIT
18	025250	010146		MOV R1, -(SP) ; AND STACK THE DIGIT
19	025252	010001		MOV R0, R1 ; GET QUOTIENT
20	025254	001370		BNE 1\$; BR IF MORE DIGITS TO CONVERT
21				;
22				; Finished conversion. Print result.
23				;
24	025256	012600		2\$: MOV (SP)+, R0 ; GET A DIGIT FROM THE STACK
25	025260	001403		BEQ 3\$; BR IF REACHED END
26	025262			.TTYOUT ; PRINT THE DIGIT
27	025266	000773		BR 2\$; PRINT MORE
28				;
29				; Finished
30				;
31	025270	012601		3\$: MOV (SP)+, R1
32	025272	000207		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 025274    010146
9 025276    010246
10         ;
11         ; Convert value to ascii string and stack the characters.
12         ;
13 025300    012702    000003
14 025304    010001
15 025306    005000
16 025310    071027    000050
17 025314    116101    003347'
18 025320    010146
19 025322    010001
20 025324    077210
21         ;
22         ; Finished conversion. Print the result.
23         ;
24 025326    012702    000003
25 025332    012600
26 025334
27 025340    077204
28         ;
29         ; Finished
30         ;
31 025342    012602
32 025344    012601
33 025346    000207
34         ;-----
35         ; Define top of TSINIT
36         ;
37 025350
38         ;
          PRTR50: MOV     R1, -(SP)
                  MOV     R2, -(SP)
                  ;
                  ; Convert value to ascii string and stack the characters.
                  ;
                  MOV     #3, R2          ; GET # CHARS TO CVT
                  MOV     R0, R1          ; GET VALUE TO BE CONVERTED
1$:              CLR     R0              ; CLEAR HIGH-ORDER VALUE
                  DIV     #50, R0         ; DIVIDE R0-R1 BY 50
                  MOVB    R50CHR(R1), R1 ; CONVERT REMAINDER TO ASCII CHARACTER
                  MOV     R1, -(SP)       ; STACK THE CHARACTER
                  MOV     R0, R1          ; GET QUOTIENT
                  SOB     R2, 1$          ; BR IF MORE CHARS TO CONVERT
                  ;
                  ; Finished conversion. Print the result.
                  ;
                  MOV     #3, R2          ; GET # CHARS TO PRINT
2$:              MOV     (SP)+, R0        ; GET NEXT CHARACTER
                  .TTYOUT                  ; PRINT THE CHARACTER
                  SOB     R2, 2$          ; LOOP IF MORE CHARS TO PRINT
                  ;
                  ; Finished
                  ;
                  MOV     (SP)+, R2
                  MOV     (SP)+, R1
                  RETURN
          ;-----
          ; Define top of TSINIT
          ;
INITOP:
          ;

```

```

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     025350 177740 EDARGB: .WORD -32. ;# OF BYTES TO BE DECRYPTED (EDMTH3)
13     025352 025660' 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     025354 010146 INSCHK: MOV R1,-(SP) ;SAVE REGISTERS
18     025356 010246 MOV R2,-(SP)
19     025360 010346 MOV R3,-(SP)
20     025362 010446 MOV R4,-(SP)
21     025364 010546 MOV R5,-(SP)
22     025366 012700 MOV (PC)+,R0 ;DECRYPT LICENSE NUMBER
23     025370 073472 .RAD50 /SCB/ ;WITH THIS CODE
24     025372 074037 025720' XOR R0,LICNUM ;BY XORING IT
25     025376 013737 025720' 000000G MOV LICNUM,TSXSIT ;MOVE LICENCE NUMBER TO TSGEN CELL
26     025404 012700 025350' MOV #EDARGB,R0 ;POINT TO ENC/DEC ARG BLOCK (PRESET)
27     025410 004737 026046' CALL EDMTH3 ;DECRYPT TO INTERMED STATE
28     ;
29     ; Copy Pro ID ROM low bytes into memory, and encrypt 1 step
30     ;
31     173600 IDADDR = 173600 ;ADDRESS OF START OF PRO 350 ID ROM
32     025414 012701 173600 MOV #IDADDR,R1 ;GET POINTER TO PRO ID ROM
33     025420 012702 025722' MOV #ROMBUF,R2 ;POINTER TO COPY OF HARDWARE ID
34     025424 010237 025352' MOV R2,EDADDR ;SAVE ADDRESS FOR ENCRYPTION
35     025430 005437 025350' NEG EDARGB ;MAKE +32. FOR ENCRYPTION
36     025434 013700 025350' MOV EDARGB,R0 ;ALSO USE AS LOOP COUNTER
37     025440 112122 3$: MOVB (R1)+,(R2)+ ;GET NEXT LOW BYTE
38     025442 005201 INC R1 ;SKIP ID ROM HIGH BYTES
39     025444 077003 SOB R0,3$ ;REPEAT THROUGH 32 BYTE ROM
40     025446 012700 025350' MOV #EDARGB,R0 ;POINT TO ENCRYPTION ARG BLOCK
41     025452 004737 025762' 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     025456 012701 025722' MOV #ROMBUF,R1 ;POINT TO HARDWARE COPY OF ID
48     025462 012702 025660' MOV #DSKBUF,R2 ;POINT TO DISK COPY OF ID
49     025466 012704 000000G MOV #SCHED,R4 ;POINT TO CODE TO BE CORRECTED
50     025472 013703 025350' MOV EDARGB,R3 ;INIT LOOP COUNTER
51     025476 004737 025652' CALL GETLIC ;USE LIC # AS SEED FOR EDPRNW, IN R0
52     025502 122122 4$: CMPB (R1)+,(R2)+ ;VERIFY ID'S ARE THE SAME
53     025504 001014 BNE 5$ ;ABORT IF NO MATCH ON ANY BYTE
54     025506 004737 026250' CALL EDPRNW ;RANDOMIZE R0 FOR XOR (LIC# INIT SEED)
55     025512 011405 MOV @R4,R5 ;GET ENCRYPTED CODE
56     025514 074005 XOR R0,R5 ;RESTORE FUNCTIONAL CODE
57     025516 010524 MOV R5,(R4)+ ;PUT DECRYPTED CODE BACK IN MEMORY

```

INSCHK -- Installation validation subroutines for Pro-350

```

58 025520 077310          SOB      R3,4$              ; REPEAT THROUGH ID TESTS
59 025522 012605          MOV      (SP)+,R5           ; RESTORE REGISTERS
60 025524 012604          MOV      (SP)+,R4
61 025526 012603          MOV      (SP)+,R3
62 025530 012602          MOV      (SP)+,R2
63 025532 012601          MOV      (SP)+,R1
64 025534 000207          RETURN                      ; ID CHECKS AND CODE DECRYPTED
65
66 025536                5$:      .PRINT #TSXHD        ; ?TSX-F
67 025544                .PRINT #NOTLIC                ; NOT LICENSED FOR THIS MACHINE
68 025552 000137 004134'      JMP      INISTP          ; ID'S DON'T MATCH, ABORT INIT.
69
70                .NLIST BEX
71 025556      124      150      151 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 025652 013700 025720'      GETLIC: MOV      LICNUM,R0      ; RETRIEVE DECRYPTED LIC # INTO R0
79 025656 000207          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 025660                DSKBUF: .BLKB 32.                ; DISK IMAGE OF PRO ID
85 025720 000000          LICNUM: .WORD 0                ; INCREMENTAL LICENSE NUMBER
86 025722                ROMBUF: .BLKB 32.                ; COPY OF ROM ID LOW BYTES

```

EDEXPL -- Comments on encryption methods

```

1          .SBTTL  EDEXPL -- Comments on encryption methods
2
3          ;
4          ; Encryption and decryption methods used here depend heavily on
5          ; pseudo-random numbers generated by the linear congrutential method.
6          ; See Hull and Dobell, SIAM Review, 4, 230, 1962.
7          ;
8          ; For the linear congruence relation:
9          ;
10         ;   X(I) == ( A * X(I-1) + C ) MOD M
11         ;
12         ;   X(I) is in the range 0 to M-1
13         ;
14         ; The sequence has full period M, provided that:
15         ;   1) C is relatively prime to M
16         ;   2) If p is a prime factor of M, A MOD p == 1
17         ;   3) If 4 is a factor of M, A MOD 4 == 1
18         ;
19         ; In the special case where M is a power of 2, these rules simplify to
20         ;   1) C must be odd
21         ;   2) A MOD 4 == 1
22         ;
23         .SBTTL  EDMTH2 -- Encryption method 2 (XOR with PRN high bytes)
24         ;
25         ; Using the license number as the initial seed, mask out the low 3 bits,
26         ; add 1 and call the PRN generator this many times to form the seed,
27         ; XOR each byte in the input buffer with the high byte of the next PRN
28         ; and replace the result in the input buffer. Decryption is accomplished
29         ; by a second application of the same process.
30         ;
31         ; Inputs:
32         ;   RO      Points to an arg block of the form:
33         ;           RO ---> buff_siz      ; word holding byte length of buffer
34         ;           buff_addr      ; address of buffer to be encrypted
35         ;
36         ; Outputs:
37         ;   RO      Randomized
38         ;   input buffer encrypted
39
40 EDMTH2:
41         MOV     R1, -(SP)      ; Save registers
42         MOV     R2, -(SP)
43         MOV     R3, -(SP)
44         ;
45         ; Fetch byte count, buffer pointer and initialize PRN seed
46         ;
47         MOV     (RO)+, R3      ; Fetch byte count of input buffer
48         MOV     (RO), R1       ; Fetch pointer to input buffer
49         CALL    GETLIC         ; Use license number as initial PRN seed
50         MOV     RO, R2         ; Copy license number to form repeat count
51         BIC     #^C7, R2       ; No more than 8 repeats
52         INC     R2             ; Make sure there is at least one
53         2$:      CALL    EDPRNW   ; Get a new PRN
54         SOB     R2, 2$         ; Advance the seed between 1 and 8 times
55         ;
56         ; Now sweep the buffer, XORing each byte with the high PRN byte
57         ;
58         1$:      CALL    EDPRNW   ; With seed in RO, get next random number
59         MOV     (R1), R2       ; Get next input byte

```

TSINIT -- TSX startup initializ MACRO V05.05 Friday 20-Jan-89 11:30 Page 87-1
EDMTH2 -- Encryption method 2 (XOR with PRN high bytes)

58	026024	000300	SWAB	R0	;Reverse PRN high and low bytes
59	026026	074002	XOR	R0,R2	;Encrypt the byte
60	026030	000300	SWAB	R0	;Restore PRN high and low bytes for next seed
61	026032	110221	MOVB	R2,(R1)+	;Save encrypted bytes back into input buffer
62	026034	077310	SOB	R3,1\$	;Repeat for entire input buffer
63					
64	026036	012603	MOV	(SP)+,R3	;Restore registers
65	026040	012602	MOV	(SP)+,R2	
66	026042	012601	MOV	(SP)+,R1	
67	026044	000207	RETURN		

```

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

```

```

58 026150 005703          TST      R3          ;Positive for encryption
59 026152 100411          DMI      3$         ;Branch if decrypting
60                          ; Do this part for encryption
61 026154 151104          BISB     @R1,R4      ;Get first byte without sign extend
62 026156 000304          SWAB     R4          ;And put it in the high byte
63 026160 151204          BISB     @R2,R4      ;Combine it with first byte
64 026162 000241          CLC              ;Always do at least one shift
65 026164 006004          ROR      R4          ;Shift once
66 026166 006005          ROR      R5          ;Get low bit into r5
67 026170 005400          NEG      R0          ;Right shifts for encryption
68 026172 005303          DEC      R3          ;Reduce count of pairs remaining
69 026174 000407          BR       4$         ;Skip decryption stuff
70                          ; Do this part for decryption
71 026176 151105          3$: BISB     @R1,R5      ;Get first byte without sign extend
72 026200 000305          SWAB     R5          ;And put it in the high byte
73 026202 151205          BISB     @R2,R5      ;Combine it with the first byte
74 026204 000241          CLC              ;Always do at least one shift
75 026206 006105          ROL      R5          ;Shift once
76 026210 006104          ROL      R4          ;Get high bit into R4
77 026212 005203          INC      R3          ;Reduce count of pairs remaining
78                          ; Shift and recombine the {en|de}cryptd bytes
79 026214 073400          4$: ASHC     R0,R4      ;Shift combined bytes 0,2,4 or 6 more
80 026216 050504          BIS      R5,R4      ;Recombine bytes
81 026220 012600          MOV      (SP)+,R0     ;Recover EDPRNM seed
82                          ; Now put encrypted bytes back into input string
83 026222 110412          MOVB     R4,@R2      ;Store low byte at second byte place
84 026224 000304          SWAB     R4          ;Get high byte
85 026226 110411          MOVB     R4,@R1      ;Store high byte at first byte place
86 026230 000730          BR       2$         ;Repeat through string
87                          ; Done, restore registers and return
88 026232 012600          9$: MOV      (SP)+,R0     ;Just pop saved buffer address
89 026234 012605          MOV      (SP)+,R5     ;Restore registers
90 026236 012604          MOV      (SP)+,R4
91 026240 012603          MOV      (SP)+,R3
92 026242 012602          MOV      (SP)+,R2
93 026244 012601          MOV      (SP)+,R1
94 026246 000207          RETURN

```

```

1          .SBTTL  EDPRNW -- Pseudo random number generator with MOD 216
2          ;
3          ; Linear congruential pseudo-random number generator with maximum repeat
4          ; length of 65536 (216). cf. Hull and Dobell and Knuth, vol 2.
5          ;
6          ; Inputs:
7          ;      R0      Seed value
8          ;
9          ; Outputs:
10         ;      R0      New PRN, should be used for next seed
11         ;
12 026250   EDPRNW:
13 026250   MOV      R4, -(SP)      ; Save registers
14 026252   MOV      R5, -(SP)
15 026254   MOV      R0, R4        ; Get seed to be multiplied
16 026256   MOV      (PC)+, R0     ; Fetch multiplier
17 026260   EDPRNA: .WORD 104375   ; Multiplier, can be replaced
18 026262   MUL      R0, R4        ; Multiply by A
19 026264   ADD      (PC)+, R5     ; Add C
20 026266   EDPRNC: .WORD 012705   ; Additive, can be replaced
21 026270   MOV      R5, R0        ; Return result mod 65536. as PRN
22 026272   MOV      (SP)+, R5    ; Restore registers
23 026274   MOV      (SP)+, R4
24 026276   RETURN

```

TSINIT -- TSX startup initializ MACRO V05.05 Friday 20-Jan-89 11:30 Page 90
INPRNM -- Initialize PRN generator with repeat range M

```
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 026300      INPRNM:
12 026300 012737 000040 026366'  MOV    #32.,EDMODM    ;Get repeat length to cover Pro ID
13 026306 012737 000005 026354'  MOV    #5,EDMULA     ;Use first valid A
14 026314 012737 000003 026360'  MOV    #3,EDADDC     ;And first valid C
15 026322 000207                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 026324   EDPRNM:
17 026324   010446       MOV     R4, -(SP)          ; Save R4 and R5
18 026326   010546       MOV     R5, -(SP)
19 026330   020037 026366' CMP     R0, EDMODM        ; Is seed in range 0 to EDMODM?
20 026334   103405       BLO     1$                 ; Branch and proceed if so
21 026336   010005       MOV     R0, R5            ; Set up to divide it by EDMODM
22 026340   005004       CLR     R4                ; Set up for divide
23 026342   071437 026366' DIV     EDMODM, R4        ; Divide it
24 026346   010500       MOV     R5, R0            ; And use remainder as seed
25 026350   010004   1$: MOV     R0, R4            ; Get current seed ready to be multiplied
26 026352   070427       MUL     (PC)+, R4         ; Multiply by chosen A
27 026354   061125   EDMULA: .WORD 25173.          ; Replace at run-time with 5
28 026356   062705       ADD     (PC)+, R5         ; Add in C
29 026360   033031   EDADDC: .WORD 13849.          ; Replace at run-time with 3
30 026362   005004       CLR     R4                ; Clear high word for division
31 026364   071427       DIV     (PC)+, R4         ; Perform mod M
32 026366   000400   EDMODM: .WORD 256.            ; Replace at run-time with 32.
33 026370   010500       MOV     R5, R0            ; Return remainder
34 026372   012605       MOV     (SP)+, R5         ; Restore R4 and R5
35 026374   012604       MOV     (SP)+, R4
36 026376   000207       RETURN
37
38         ; IFF      ; NE, PROCID                ; Assemble if protection code not included
39 DSKBUF:   ;                                     ; Define dummy DSKBUF global symbol
40         . ENDC    ; NE, PROCID
41         ;
42         ; Address of real top of TSINIT, including PRO init code
43         ;
44 026400   PROITP:
45 000000   . CSECT  TSXEND
46         . END
000001

```

AHEND = ***** G	CLVERS= ***** G	CXTWDS= ***** G	EDADDR 025352R	002 HANXMR 021370R	002
ALBFX 012536R	002 CL\$EPN= ***** G	C.CSW = ***** G	EDARGB 025350R	002 HF\$LIN= ***** G	
ALCHRB 007562R	002 CL\$EPP= ***** G	C.DEVQ= ***** G	EDMODM 026366R	002 HF\$MC = ***** G	
ALCOVL 022130R	002 CL\$EPS= ***** G	C.SBLK= ***** G	EDMTH2 025762R	002 HF\$RIE= ***** G	
ALCSLO 012316R	002 CL\$LEN= ***** G	C1DEVX= ***** G	EDMTH3 026046R	002 HF\$TIE= ***** G	
ALCWRK 007526R	002 CL\$LIX= ***** G	DATIMH= ***** G	EDMULA 026354R	002 HF\$TSB= ***** G	
ALOCBF 011652R	002 CL\$OPT= ***** G	DATIML= ***** G	EDPRNA 026260R	002 HGENFL 000240R	002
AREA 000000R	002 CL\$ORA= ***** G	DCCSIZ= ***** G	EDPRNC 026266R	002 HIMAP = ***** G	
AUTHAN= ***** G	CL\$ORB= ***** G	DC\$\$SZ= ***** G	EDPRNM 026324R	002 HLERR 020472R	002
BADDEV 010170R	002 CL\$ORE= ***** G	DEVSIZ= ***** G	EDPRNW 026250R	002 HMAP 000126R	002
BADLIN 000764R	002 CL\$ORG= ***** G	DEVVEC 004652R	002 EMMAP = ***** G	HN2BIG 002335R	002
BADOPN 001137R	002 CL\$ORP= ***** G	DFJMEM= ***** G	EMTENT= ***** G	HSGER 002027R	002
BASMAP= ***** G	CL\$ORS= ***** G	DHBFSZ= ***** G	ENTVEC 004464R	002 HVEND 000336R	002
BDLMSG 001055R	002 CL\$STA= ***** G	DHINIT 005260R	002 ERHINS 001721R	002 HVTBL 000322R	002
BDSPOP 001305R	002 CL\$VER= 000002	DHLPRM 005260R	002 ERHMSG 001552R	002 HV\$DMY= 000001	
BDVMSG 001017R	002 CONFG2= ***** G	DHOINT= ***** G	ERHNDV 001616R	002 HV\$ID = 000000	
BOSF 001342R	002 CONFIG= ***** G	DI\$CL = ***** G	ERRLOG= ***** G	HV\$VER= 000002	
BRKPT = ***** G	CONSPC 001237R	002 DI\$DU = ***** G	EXCBUF= ***** G	HV\$\$SZ= 000004	
BYTES 002602R	002 COSRT 002702R	002 DI\$LD = ***** G	EXTLSI= 000001	H.CSR = ***** G	
CA\$BLK= ***** G	CO\$DEF= ***** G	DI\$MU = ***** G	FC\$LBN= ***** G	H.DSTS= ***** G	
CA\$DVU= ***** G	CO\$FF = ***** G	DI\$PI = ***** G	FC\$\$SZ= ***** G	H.DVSZ= ***** G	
CA\$HBL= ***** G	CO\$TAB= ***** G	DI\$TT = ***** G	FETDEV 000130R	002 H.GEN = ***** G	
CA\$HFL= ***** G	CO\$8BT= ***** G	DI\$XL = ***** G	FF\$\$SZ= ***** G	H.INS = ***** G	
CA\$HSH= ***** G	CRLF 001067R	002 DMYDEV= ***** G	FILBLK 000142R	002 H.SIZ = ***** G	
CA\$UBL= ***** G	CSHALC= ***** G	DM\$CSR= ***** G	FMEMHI 000134R	002 ICONFG 000242R	002
CA\$UFL= ***** G	CSHBAS= ***** G	DM\$LSR= ***** G	FMEMLO 000136R	002 IDADDR= 173600	
CA\$WCT= ***** G	CSHBFP= ***** G	DOHNLC 020542R	002 FNDHRB 021364R	002 IDSFLG= ***** G	
CCAFLG 000040R	002 CSHBUF 024104R	002 DSKBUF 025660R	002 FORCEO 011574R	002 II\$\$SZ= ***** G	
CCATST 025144R	002 CSHDEV= ***** G	DSTBLK 000112R	002 FORK = ***** G	ILSW2 = ***** G	
CCBHD = ***** G	CSHDVN= ***** G	DS\$ABT= ***** G	FPTRAP= ***** G	INDDBL= ***** G	
CCLNAM 000102R	002 CSHOVF 003211R	002 DS\$DIR= ***** G	FQ\$\$SZ= ***** G	INDDBS= ***** G	
CCLSAV= ***** G	CSHSIZ= ***** G	DS\$NRD= ***** G	FREIOQ= ***** G	INDFIL= ***** G	
CC\$\$SZ= ***** G	CSHVEC= ***** G	DS\$SFN= ***** G	FREPGS= ***** G	INDINI 014110R	002
CDX\$DH= ***** G	CS\$ENT= ***** G	DS\$VSZ= ***** G	FRKGEN= ***** G	INDNAM 000220R	002
CDX\$DL= ***** G	CS\$NMX= ***** G	DTLX = ***** G	FRKINI= ***** G	INDOPN 003262R	002
CDX\$DZ= ***** G	CS\$OPN= ***** G	DTYPE = ***** G	FSTDLD = ***** G	INDSAV= ***** G	
CDX\$VH= ***** G	CURDEV 000144R	002 DVFLAG= ***** G	FSTIOL= ***** G	INDTSV= ***** G	
CD\$\$SZ= ***** G	CURNAM 000146R	002 DVFLBS 000336R	002 FW\$\$SZ= ***** G	INICLK 006524R	002
CFHMSG 001510R	002 CVTDVU 011476R	002 DVFLND 000516R	002 GENTOP= ***** G	INIDEV 006672R	002
CHAIN = ***** G	CW\$BTH= ***** G	DVSTAT= ***** G	GETHNH 017634R	002 INIJMP= ***** G	
CHKCLD 011372R	002 CW\$ESP= ***** G	DV\$FLG= 000002	GETHNL 016320R	002 INIOVL 004312R	002
CHKMEM 025104R	002 CW\$FB = ***** G	DV\$NAM= 000000	GETLIC 025652R	002 INISTP 004134R	002
CHNSIZ= ***** G	CW\$FGJ= ***** G	DV\$\$SZ= 000004	GETMAP 022010R	002 INITGD 005452R	002
CKLIN 006644R	002 CW\$GDH= ***** G	DW\$\$SZ= ***** G	GETOVL 023116R	002 INITOP 025350R	002
CLDEVX= ***** G	CW\$LGS= ***** G	DX\$DMA= ***** G	GETSRT 023472R	002 INMXV = ***** G	
CLEOFS= ***** G	CW\$PRO= ***** G	DX\$EBA= ***** G	GTBYT = ***** G	INPRNM 026300R	002
CLHEAD= ***** G	CW\$QBS= ***** G	DX\$IBH= ***** G	GTLIN 000176R	002 INRECV= ***** G	
CLINCP= ***** G	CW\$USR= ***** G	DX\$MAP= ***** G	HANDSK= ***** G	INSCHK 025354R	002
CLINIT 013444R	002 CW\$XM = ***** G	DX\$MPH= ***** G	HANENT= ***** G	INSCK1 016602R	002
CLKRTI= ***** G	CW\$5OH= ***** G	DX\$NCA= ***** G	HANMAP 021242R	002 INSCK2 016760R	002
CLKVEC= ***** G	CXTALC 015646R	002 DX\$NHM= ***** G	HANNAM 000200R	002 INSTBL= ***** G	
CLK100 000042R	002 CXTBAS= ***** G	DX\$NMT= ***** G	HANPAR= ***** G	INSTBN= ***** G	
CLORSZ= ***** G	CXTBUF= ***** G	DX\$NRD= ***** G	HANRCB= ***** G	INTEN = ***** G	
CLOTIR= ***** G	CXTPAG= ***** G	DX\$NST= ***** G	HANRCO= ***** G	INTMX1= ***** G	
CLSIZE= ***** G	CXTPDR= ***** G	DZINIT 005260R	002 HANSIZ= ***** G	INTSND= ***** G	
CLSTS = ***** G	CXTRMN= ***** G	DZOINT= ***** G	HANUMP 021316R	002 INTSSZ= ***** G	
CLTOTL= ***** G	CXTSIZ= ***** G	EDADDC 026360R	002 HANXIT= ***** G	INTSTK= ***** G	

INVEC = ***** G	LOKVEC= ***** G	MPAR16= ***** G	ODTSTA 000234R	002 PMPAR = ***** G
IOMAP = ***** G	LOMAP = ***** G	MPIVSZ= 000032	ODTTOP 000036R	002 PNAME = ***** G
IOPAGE= ***** G	LOTBUF= ***** G	MPPHY = ***** G	ODTTRP= ***** G	002 PPTERM 000262RG 002
IOQSIZ= ***** G	LOTEND= ***** G	MR\$\$SZ= ***** G	OORCLO 022710R	002 PROASM= 000001 G
IOTIMR= ***** G	LOTSIZ= ***** G	MSGBAS= ***** G	OORCSH 022730R	002 PROBUF 000150RG 002
JCXPGS= ***** G	LOUTIR= ***** G	MTSXDV 024544R	002 OORDBG 022720R	002 PROCID= 000001
JMPO = ***** G	LOWEND 000754R	002 MUXVEC 005260R	002 OORDMP 022740R	002 PROFLG= ***** G
JM\$\$SZ= ***** G	LOWOVL 000746R	002 MU\$TXT= ***** G	OORLOK 022522R	002 PROHAN= ***** G
JSWLOC= ***** G	LSTDLD = ***** G	MVSIZ = ***** G	OORMIO 022700R	002 PROINI= ***** G
KEYSEG 004522R	002 LSTHL = ***** G	MW\$LNK= ***** G	OORMSG 022544R	002 PROITP 026400RG 002
KMNBAS= ***** G	LSTLIN= ***** G	MW\$\$SZ= ***** G	OORPLS 022620R	002 PROLIN= ***** G
KMNCHN= ***** G	LSTMX = ***** G	MXCSR = ***** G	OORSLE 022652R	002 PRONOP= ***** G
KMNHI = ***** G	LSTPL = ***** G	MXDTR = ***** G	OORSPL 022452R	002 PROSIZ= ***** G
KMNNAM 000072R	002 LSTSL = ***** G	MXJADR= ***** G	OORSP2 022512R	002 PRTDEC 025230R 002
KMNPGS= ***** G	LSWPBK= ***** G	MXJMEM= ***** G	OORSWP 022642R	002 PRTOCT 025160R 002
KMNSTK= ***** G	LSW10 = ***** G	MXLNT = ***** G	OORUSR 022436R	002 PRTR50 025274R 002
KMNSTR= ***** G	LSW3 = ***** G	MXLPR = ***** G	OORWIN 022662R	002 PSW = ***** G
KMNTOP= ***** G	LSW5 = ***** G	MXRBUF= ***** G	OOXFIN 023010R	002 PTBYT = ***** G
KPAR0 = ***** G	LSW6 = ***** G	MXTYPE= ***** G	OOXNO 022750R	002 PTWRD = ***** G
KPAR5 = ***** G	LTPAR= ***** G	MXVEC = ***** G	OOXYES 022756R	002 PVSPBL= ***** G
KPAR6 = ***** G	LXCL = ***** G	NDL = ***** G	OPNCHN 024322R	002 QBUS = ***** G
KPAR7 = ***** G	MAPALC 016006R	002 NDVRCB= ***** G	OPNKMN 013142R	002 RBD = 000062
KPDRO = ***** G	MAPPAR= ***** G	NEDCHR= ***** G	OPNRSF 010216R	002 RBR = ***** G
LCDTYP= ***** G	MAPSIZ= ***** G	NEXMSG 002223R	002 OPNSWP 007564R	002 RC\$\$SZ= ***** G
LCLUNT= ***** G	MAPSYS= ***** G	NFRESB= ***** G	OPTOVL 022274R	002 RDB = ***** G
LDDEVX= ***** G	MAXDEV= ***** G	NFSBLK 000020R	002 OSEND 000744R	002 RDBEND= ***** G
LDHAND 016430R	002 MAXOVL= 000031	NIOL = ***** G	OSLAST 000744R	002 RDERR 003066R 002
LDHB1B= ***** G	MAXSLO= ***** G	NLINES= ***** G	OSTABL 000516R	002 RDINT = ***** G
LDHB1P= ***** G	MA\$SYS= ***** G	NMSNMB= ***** G	OSZ = 000056	002 REDUCE 002551R 002
LDHB2B= ***** G	MB\$\$SZ= ***** G	NMXHAN 000124R	002 OS\$FLG= 000002	002 RELOC = ***** G
LDHNHI 017672R	002 MEMINI 015230R	002 NOCCL 001452R	002 OS\$OVL= 000004	002 REGMIS 001070R 002
LDHNLO 017504R	002 MEMLIM 000236R	002 NOCLOK 002077R	002 OS\$SIZ= 000000	002 RID = 000060
LDREAD 021072R	002 MEMPAR= ***** G	NOCSRR 001670R	002 OS\$\$SZ= 000006	002 RLBF 000230R 002
LDVERS= ***** G	MEMTST 015334R	002 NOKMON 001411R	002 OTMXV = ***** G	002 RLBFND 000232R 002
LHIRBA= ***** G	MEM256= ***** G	NOODT 002272R	002 OTRECV= ***** G	002 RMNPDR= ***** G
LHIRBB= ***** G	MF\$CM = ***** G	NOSYDV 002442R	002 OVLBAS 000140R	002 RMON = ***** G
LHIRBC= ***** G	MF\$CS = ***** G	NOTLIC 025556R	002 OVLBLD 021610R	002 ROMBUF 025722R 002
LHIRBE= ***** G	MF\$LE = ***** G	NOXM 015626R	002 OVLEND 022436R	002 RPRVEC= ***** G
LHIRBG= ***** G	MF\$LIN= ***** G	NSCP = ***** G	OVLLST 022346R	002 RSFBLK= ***** G
LHIRBP= ***** G	MHNSIZ= ***** G	NSIP = ***** G	OVLPOS 021400R	002 RSFERR 001175R 002
LHIRBS= ***** G	MH\$LPR= ***** G	NSL = ***** G	OVLTRY 023020R	002 RSR = ***** G
LICNUM 025720R	002 MH\$PBR= ***** G	NSPLBL= ***** G	OVRADD= ***** G	002 RSZ = 000052
LINBUF= ***** G	MH\$SCR= ***** G	NSPLDV= ***** G	O. ADR = ***** G	002 RTFTCH 024770R 002
LINCHK 005260R	002 MINCTR= ***** G	NSPLFL= ***** G	O. BLK = ***** G	002 RTMNVC 000046R 002
LINEND= ***** G	MIOBHD= ***** G	NUMCCB= ***** G	O. PAR = ***** G	002 RTNKM 015616R 002
LININI 005262R	002 MIODBG= ***** G	NUMCDB= ***** G	O. SIZ = ***** G	002 RTTRP4 000044R 002
LINIR = ***** G	MIOFLG= ***** G	NUMDCD= ***** G	PARENL= ***** G	002 RTVDEF 000314R 002
LINSIZ= ***** G	MIONWB= ***** G	NUMDEV= ***** G	PARSET 016316R	002 RTVEND 000322R 002
LINSPC= ***** G	MIOWHD= ***** G	NUMFRK= ***** G	PHSOVF 002612R	002 RTVER 000264R 002
LINTYP 005340R	002 MI\$LNK= ***** G	NUMIOQ= ***** G	PHYMEM= ***** G	002 RTVPTR 000156R 002
LMXLN = ***** G	MI\$SBP= ***** G	NUMRDB= ***** G	PIDPTR= ***** G	002 RTV\$SZ= 000003
LMXNUM= ***** G	MI\$\$SZ= ***** G	NXIVMH= ***** G	PIDRIV= ***** G	002 RT\$BAS= ***** G
LODINI 004160R	002 MMENBL= ***** G	NXMMSG 002165R	002 PIDVEN 005200R	002 RT\$NAM= ***** G
LODOVL 023162R	002 MONFGH= ***** G	ODTBAS= 001000	PIHAN = ***** G	002 RT\$SKP= ***** G
LOKBAS= ***** G	MONVEC= ***** G	ODTBLK 000210R	002 PIINSZ= 001274	002 RT\$TOP= ***** G
LOKCSH= ***** G	MPARFL= ***** G	ODTFLG 000034R	002 PISRT 000244RG	002 RT\$UPD= 000001
LOKMEM= ***** G	MPARO = ***** G	ODTRDM 002401R	002 PMCELS= ***** G	002 RT\$VER= 000000

RT\$\$SZ= ***** G	SETSY 024600R	002 SWDBLK= ***** G	UCLBLK= ***** G	VMLBLK= ***** G
RT40 000264R	002 SETUMP 004650R	002 SWPCHN= ***** G	UCLDAT= ***** G	VMSCHR= ***** G
RT50 000267R	002 SETVEC 005024R	002 SWPJOB= ***** G	UCLINI 014652R	002 VMXCSH= ***** G
RT51 000272R	002 SFCB = ***** G	SWPPOS= ***** G	UCLNAM= ***** G	VMXMON= ***** G
RT51B 000300R	002 SFCBFH= ***** G	SYINDX= ***** G	UCLOPN 003312R	002 VMXMRB= ***** G
RT51C 000303R	002 SFCBND= ***** G	SYNAME= ***** G	UC\$\$SZ= ***** G	VMXMSG= ***** G
RT51X 000275R	002 SFCBSZ= ***** G	SYSDAT= ***** G	UDDRO = ***** G	VMXSF = ***** G
RT52 000306R	002 SG\$ELG= ***** G	SYSGEN= ***** G	UEXINT= ***** G	VMXSFC= ***** G
RT53 000311R	002 SG\$IOT= ***** G	SYSMAP= ***** G	UK\$\$SZ= ***** G	VMXWIN= ***** G
RT54 000314R	002 SG\$MMU= ***** G	SYSUPD= ***** G	UMODE = ***** G	VNCSLO= ***** G
RT55 000317R	002 SG\$MTM= ***** G	SYsver= ***** G	UMRADR= ***** G	VNCXOF= ***** G
R50CHR 003347R	002 SG\$MTS= ***** G	SYTIMH= ***** G	UMSYTP= ***** G	VNCXON= ***** G
R50CL = 012240	SG\$PAR= ***** G	SYTIML= ***** G	UNIBUS= ***** G	VNFCSH= ***** G
R50CLO= 012276	SG\$TSX= ***** G	SYUNIT= ***** G	002 UPARO = ***** G	VNGR = ***** G
R50CL7= 012305	SHRRCB= ***** G	TAKQVR 003420R	UPAR6 = ***** G	VNUIP = ***** G
R50CSH= 012700	SHRRCN= ***** G	TIOBAS= ***** G	UPAR7 = ***** G	VNUMDC= ***** G
R50C1 = 013630	SIZERR 025124R	002 TIOVEC= ***** G	UPDR0 = ***** G	VPAR5 = ***** G
R50C10= 013666	SKPDEV 000160R	002 TK1SEC= ***** G	UPDR6 = ***** G	VPAR6 = ***** G
R50C17= 013675	SLTSIZ= ***** G	TK1VAL= ***** G	UPMODE= ***** G	VPLAS = ***** G
R50LD = 045640	SMRSIZ= ***** G	TK3SVL= ***** G	USRBAS= ***** G	VPMSIZ= ***** G
R50LOK= 046543	SNMSHD= ***** G	TK5VAL= ***** G	002 US\$\$SZ= ***** G	VSLEDT= ***** G
R50LS = 046770	SPLANM= ***** G	TOOBIG 002507R	002 VBUSTP= ***** G	VSWPFL= ***** G
R50MSG= 052077	SPLBLK= ***** G	TOPMEM 000132R	002 VCSHNB= ***** G	VSWPSL= ***** G
R50ODT= 057164	SPLCHN= ***** G	TRCSET 015606R	VDBFLG= ***** G	VSYDMP= ***** G
R50PI = 062550	SPLCLD 011306R	002 TRP10 = ***** G	VDFMEM= ***** G	VUCLMC= ***** G
R50SY = 075250	SPLDEV= ***** G	TRP14 = ***** G	VF\$LIN= ***** G	VUXIFL= ***** G
R50TIO= 077167	SPLDVN= ***** G	TRP20 = ***** G	VF\$MR = ***** G	VU\$CL = ***** G
R50TT = 100040	SPLINI 010522R	002 TRP24 = ***** G	VF\$RE = ***** G	WINBAS= ***** G
R50USR= 103112	SPLNB = ***** G	TRP250= ***** G	VF\$RIE= ***** G	WRKBUF 000152R
R50VM = 105610	SPLND = ***** G	TRP34 = ***** G	VF\$SC = ***** G	WRKSIZ 000154R
R50WIN= 110466	SPNEED 010142R	002 TRP4 = ***** G	VF\$TIE= ***** G	XMVBAS 000122R
SAVBLK 000050R	002 SP\$\$SZ= ***** G	TSEMT = ***** G	VHIMEM= ***** G	ZCLR = ***** G
SB\$\$SZ= ***** G	SRTOVF 003121R	002 TSEXEC= ***** G	VHINIT 005260R	\$DEAD = ***** G
SCHED = ***** G	SRTSIZ= ***** G	TSGEN = ***** G	002 VHLPRM 005260R	002 \$FORM = ***** G
SCPFHD= ***** G	SROMMR= ***** G	TSINIT 000000RG	VH\$INT= ***** G	002 \$HARD = ***** G
SDANAM= ***** G	SR3FLG= ***** G	TSR = ***** G	VH\$CSR= ***** G	\$MEMSZ= ***** G
SDCB = ***** G	SR3MMR= ***** G	TSTWRD= 000110	002 VH\$LCR= ***** G	\$NOIN = ***** G
SDCBSZ= ***** G	SS = ***** G	TSXHD 000754R	002 VH\$LPR= ***** G	\$OVRH = ***** G
SDCHAN= ***** G	SSEND = ***** G	TSXRUN 002000R	002 VINABT= ***** G	\$PHONE= ***** G
SDDVU = ***** G	STA = 000040	TSXSAV 000062R	VLDSYS= ***** G	\$SXON = ***** G
SDNAME= ***** G	STDVTB 017374R	002 TSXSIT= ***** G	002 VMAXMC= ***** G	\$TAB = ***** G
SEGCHN= ***** G	STHNPV 020400R	002 TSXSIZ 003000R	VMIOBF= ***** G	\$8BIT = ***** G
SETCHN 024410R	002 STK = 000042	TTINCP= ***** G	VMIOSZ= ***** G	... V1 = 000003
SETJSZ 016172R	002 STKLVL= ***** G	UBUSMP= ***** G		... V2 = 000027
SETMIO 021376R	002 SVERR 002745R	002		

ABS. 000000	000	(RW, I, GBL, ABS, OVR)
000000	001	(RW, I, LCL, REL, CON)
TSINIT 026400	002	(RW, I, GBL, REL, OVR)
TSXEND 000000	003	(RW, I, GBL, REL, OVR)

Errors detected: 0

*** Assembler statistics

Work file reads: 0
Work file writes: 0

TSINIT -- TSX startup initializ MACRO V05.05 Friday 20-Jan-89 11:30 Page 91-4
Symbol table

Size of work file: 12558 Words (50 Pages)
Size of core pool: 18432 Words (72 Pages)
Operating system: RT-11

Elapsed time: 00:02:30.52
,LP:PROINI=DK:PROASM,TSINIT