NORTEL NETWORKS

Meridian 1

Software

Debugging & Patching

Issue 2.2
Belleville Training Group
February 4, 2002

M1 Debugging & Patching

1.0 Objectives

- Basics of PDT
- Provide detailed information on how to debug the Meridian 1 Option 11 C software
- Learn to use the RAS Trace tool
- Learn to use LD 77 and 80
- Introduce information on patching and patching tools.

2.0 References

1.	Inside Option 11C	C. Cojeen	Issue: 1.0
2.	Option 11C Software Debuging	D. Priestley	Issue: 1.0

PDT

3.0 PDT

The PDT (Problem Determination Tool) is a collection of software tools which are intended to locate, examine and eliminate problems in the Option 11C system.

3.1 PDT Operation

3.1.1 Start -Up

<CNTL>+PDT invokes PDT and starts a **PDT shell**.

3.1.2 Security

Security is provided by the following:

- access restriction (passwords) and access levels (security levels)
- a password is required each time a new PDT shell is created
- an access level is established separately for each PDT shell
- a proper access level is selected depending on the password provided by the user when PDT shell is created.

There are two levels of access to PDT. Level 1 is intended for use by Distributors' High Level Technical Assistance Staff; Level 2 is used by Nortel CTS, Field Support, and Designers.

Level 1 allows use of a subset of pdt commands. Level 2 allows use of all pdt commands.

For Release 22, the PDT passwords are:

Level 1: **thorsgr8**

Level 2: **2tdp22ler** (rel22pdt2 spelt backwards)

Note: the PDT password IS CASE SPECIFIC. If your terminal or terminal emulator is set to caps lock, then you will be unable to get into pdt.

3.1.3 Getting out of PDT

To get out of PDT, the user must type: ***exit***. This command closes PDT shell and returns to SL1 I/O.

```
pdt> exit
<ret>
OVL000                    - out in the SL1 world
>
Ctrl-(pdt)
PDT:  login on /sio/0
Password:                 - password prompted again
```

3.1.4 Returning to the SL1 task from the PDT shell

From the PDT shell, the user can return to the SL1 task to perform I/O from that task by using ***sl1input*** command:

```
pdt> sl1input
<ret>
OVL000                    - out in the SL1 world
>
Ctrl-(pdt)
pdt>                      - back to pdt without entering the password
```

3.1.5 PDT Modes

There are three modes of pdt (not to be confused with the 2 levels of access)
- pdt
- debug
- superuser

3.1.5.1 pdt Mode

This mode is the default mode--accessed upon first entry of pdt. It is multi-user capable.

3.1.5.2 debug Mode

This mode can only be accessed if pdt was entered with the level 2 password. This allows you to work with breakpoints, as well as have all the other commands available in pdt mode. Because it is the breakpoint mode, it is one-user only.

```
pdt> debug              - enter debug mode
dbg> debug end          - exit debug mode
pdt>
```

3.1.5.3 superuser Mode

IMPORTANT: THIS MODE IS EXTREMELY DANGEROUS AND SHOULD NEVER BE USED ON A LIVE SITE, EXCEPT UNDER THE DIRECT SUPERVISION OF NORTEL FIELD SUPPORT PERSONNEL.

This mode is useful in the lab environment, but in most cases is not to be used on a live site, unless on the recommendation of a highly trained professional and then only in cases of severe need. It is actually the straight VxWorks shell.

One handy use for **su** is its on-line hex calculator. You can perform arithmetic operations and decimal/hex/ascii conversion.

To enter and exit superuser mode:

```
pdt> su
-> 0x13e6f + 0x22e9
value = 90456 = 0x16158
-> exit
pdt>
```

3.2 General PDT commands

debug	go into the debug mode
devs	list I/O devices
exit	exit from PDT
lkup	find symbol in table (prints its real address)
mRegs	modify registers
osVersion/sl1Version	os and sl1 software version
reboot/reboot -1	warm reboot/cold reboot
sl1input	return from the PDT shell to sl1input mode
sl1listen <on|off>	enables (on) or disables(off) SL1 broadcasting to the current PDT shell
su	go into superuser mode
symload	loads symbol table into memory
tyBackspaceSet 8	enables the use of the Backspace key

3.2.1 devs

This command lists all the devices active on an Option 11C. ioFdShow command in VxWorks displays similar information.

3.2.2 lkup

The string is case-specific, rather like UNIX's grep command.

```
pdt> lkup sho
_xdr_u_short              0x1013c426 text
_rdshow                   0x100f5e18 text
_xdr_short                0x1013c3d0 text
pdt>
```

3.2.3 mRegs

This is used to change contents of registers.
Example: change content register d0 to 1:

```
pdt> mRegs
d0    :         0 - 1 <ret>
d1    :      3004 - <ret>
d2    : 200463ea - . <ret>        '.' indicates end
value = 0 = 0x0
```

3.2.4 sl1Version and osVersion

sl1Version allows user to check the SL1 issue. **sl1Version** may require a **symload.**

osVersion shows the same information for the operating system.

Usage:

```
pdt> sl1Version
SL1: Date = Aug 28 1996, Time = 17:41:19, Base = x112208
value = 0 = 0x0
pdt>
```

3.2.5 symload

This command loads the symbol table (res.sym). symload if required if:

- calling a C function, an error message "cannot find symbol table entry for *<function>* is printed
- a task trace on SL1 does not print the globals and offsets.
- the Report Log's Return Address Stack does not show functions and offsets.

Usage:

```
pdt> symload
Loading symbols from "c:/p/sl1/res.sym"
pdt>
```

3.2.6 tyBackspaceSet

This command enables the use of the Backspace key

3.3 File System commands

cat / type	view a file
cd	change directory
chkdsk	check free disk space
cp / copy	copy file
ls / ll	list all files in current directoy
mkdir / mv	make a new directory / move file
pwd	print current working directory
rename	rename a file
rmdir	remove directory
rm / del	remove or delete a file

3.4 Memory print and modify commands

d <address> <n>	display the contents of unprotected memory starting at absolute <address> for <n> number of words
m <address>	modify the content of address
mem	shows the system memory partition blocks and statistics
p <address> <n>	display the contents of memory starting at relative <relative> address for <n> number of words
w <address>	this command is used to write to the protected memory location. This command calls a library function which performs the proper write to protected memory.

Note that only way to modify protected memory is to use the **w** command. Attempting to do so using any other command will result in a bus error. This is because the **w** command calls a library of routines which performs the write properly, including re-setting the checksums on the protected memory "pages".

3.4.1 d, m commands

We want to display the memory at the following address (taken from Xview):

movl **a5@(0x50d0)**,a0

The content of A5 is 2034000, the SL1 memory offset. Therefore the absolute memory address is 0x2034000 + 0x50d0 = 0x203450d0:

```
pdt> d 0x203450d0 4   - display 4 words
0x203450d0:  00000500 5f72656c  65617365 43726566  .releaseCref
pdt> m 0x20000000     - write to memory
20000000:  0000-5
20000002:  0018-46
20000004:  0000-.     -  "." end write

value = 1 = 0x1       - this just always prints out
```

3.4.2 p, w commands

```
pdt> tnt 0 3

EQPD SLOOP TN 0030C0
 GP 00048FC3 SLP 00049397 001D6731  CD 00048462 001D1963
                    LN 0004AC7E 001D1943
pdt> p 48462 10      - checks the unprotected card block for 10 hex
words
```

3.5 SL1 specific commands

crg < tn>	displays a snapshot of the call register for the specified <tn> at the time when the command is entered
dcp <customer no>	displays PCDATAPTR and UCDATAPTR for <customer no>
dnt <customer no> <dn>	perform DN translation and displays information
drp <customer no> <route no>	displays customer route pointers
tnt <tn>	perform TN translation and display information

3.5.1 crg - Call Register

pdt> **crg 2 4**

```
 SLOOP TN SLOOP   2  0  0   4 * 000048  EQPD
 CUST  0 PBLK 00050DEE UBLK 001B2FC8
BCS
A CR 001A8EA1
00000600 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000048 00000088 00000000
00000000 00000000 0000AAA7 00000000 00000084 0000AAA7 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000010 00000000 00000000 00000000 00000000
00000000 00004008 00008000 00000000 00000000 00000000 00000000 00000000
00000088 00000000 00000000 00000000 00000000 00000000 00000000 00007400
00000000 00000000 00000000 00008000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00008210 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000100 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 000000C5 00000000 00000000 00000000 00000000 00000000
00001FF8 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00001715 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00002000 00000000 00000000 00004000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000
KEY 00 001A8EA1
```

```
pdt>
```

3.5.2 dcp - display customer pointers

pdt> **dcp 0**

```
CUST 0 P 0507CB U 1B328C AUX 0508D1 ICI 000000 PREXL 000000 BGD 050A33
pdt>
```

3.5.3 dnt - DN Translate

pdt> **dnt 0 7000**

```
DIG 4 BCS
00053478 :   00008209 00000000 00000000 00000000 00000000 000003FF 00000000
00000003
00053480 :   00000088
pdt>
```

3.5.4 drp - display route pointers

pdt> **drp 0 65**

```
CUST 0 ROUT 65 P 0005415E U 001A53D0 T 000542BE R 00000000
pdt>
```

3.5.5 tnt - TN Translate

pdt> **tnt 2 4**

```
 EQPD SLOOP TN 000048
 GP 00050C5E SLP 00050C7E 001B30A1  CD 00050CD7 001B2FF8  LN 00050DEE 001B2FC8
```

pdt>

3.6 Task commands

i	list all the tasks on the system
ti	task information. By default this uses the task at a break point. If not at a break point type: ti tSL1
tt	task trace (return address stack)
ts/tr	task suspend/task resume

3.6.1 List all tasks (i)

pdt> **i**

```
  NAME        ENTRY       TID    PRI   STATUS      PC       SP     ERRNO  DELAY
---------- ------------ -------- --- ---------- -------- -------- ------- -----
tExcTask    _excTask     2033a8e8   0   PEND       1015b804 2033a850      0     0
tLogTask    _logTask     203317b8  0    PEND      1015b804 2033171c      0     0
tSwd        _swdTask      20330298  0   DELAY     10133410 20330258      0     6
tssDrv      _ssdrvDaemon 2031df48 10    PEND      1015b804 2031de98   30065    0
tRstTask   _rstMainTask 203355b8 11    PEND      1015b804 20335538      0     0
tRpt       100f09fe     202db0bc 11    PEND      10119594 202dadb8      0     0
tTimer      _timerTask   20337db8  11   DELAY     10133410 20337d78      0  2267
tEvtColl   _evtCollTask 202bae24 11    PEND      1015b804 202bad44      0     0
tEvt       _evtTask     2029abf0  11   PEND      1015b804 2029ab18      0     0
tRdbTask   _rdbTask     20300248   20  PEND      10119594 20300128   d0003    0
tNetTask   _netTask     20307a20  50   PEND      10119594 203079c8      0     0
tFtpdTask  _ftpdTask    203047e0  55   PEND      10119594 20304724      0     0
tTapeTask  _temuLoop    20399420  60   PEND     10119594 2039c574  380003    0
tAtaCsc0   1009b13e     20318da0  90   PEND      10119594 20318d50      0     0
tAtaCsc1   1009b13e     2030c860  90   PEND      10119594 2030c810      0     0
tRlogInetd 100c5264     20305178 100   PEND     10119594 203050a0  3d0002    0
tPortmapd  _portmapd    20301768 100   PEND      10119594 20301634     16     0
pdtLogin   100c4b8a     202d40f4 100   PEND     1015b804 202d4010   d0003    0
pdtBrkTask 100caa46     202d29f4 100   PEND      1015b804 202d27ec      0     0
tDTPreaper  _dtp_reaper  202ca3ec 100   DELAY     10133410 202ca370      0    10
tDTPlisten _dtp_listene 202c7b44  100   PEND      10119594 202c7a94      0     0
tod24      _tod24Main   2032d090 240    PEND      10119594 2032d040      0     0
tSNMP      _agentTask   202d005c 240   PEND      10119594 202cfed0      0     0
tScriptMgr _vScriptMgrT 20298258 240   PEND      1015b804 20298120      0     0
tSL1       _sl1Main     20398340 240    READY    10c775aa 2002bf0c  3d0002    0
thlpTask    _hlpTask     2028ad14 240   PEND     1015b804 2028ac78  380003    0
pdtShell01 100c065e     2027e760 240   READY     10133ba8 2027d68c  3d0002    0
```

3.6.2 Specific Task information (ti tSL1)

pdt> **ti tSL1**

```
  NAME       ENTRY       TID    PRI   STATUS      PC       SP     ERRNO  DELAY
---------- ------------ -------- --- ---------- -------- --------  ------- -----
tSL1       10c9aa5a     20397770 240 READY      10c78f0a 2002bf0c  3d0002     0

stack: base 0x2002c000  end 0x20020000  size 49144  high 4932   margin 44212

options: 0x10
VX_STDIO

d0   =      142    d1   =      1       d2   =      0       d3   =      0
d4   =      0      d5   =      0       d6   =      0       d7   =      0
a0   =      80     a1   = 2069fa30     a2   = 2002bf3c     a3   = 2002bf14
a4   =      0      a5   = 20340000     a6/fp  = 2002bf14    a7/sp  = 2002bf0c
sr   =    3010     pc   = 10c78f0a
```

3.6.3 Task Trace (tt)

```
Break at 0x106f0598 (bp# 2): __CARD_AUDIT   +0x902    Task: 0x203982e0 (tSL1)

dbg> tt
1012a8f0 _vxTaskEntry  +10 : _sl1Main (0, 0, 0, &_sl1Entry, 202c64f8, 10095314,
 202c6776, 202c6508, 10095430, 2)
10c92488 _sl1Main       +1ba: 10c704f0 ([0, 0, 1012a8f2, 0, 0])
10c70592 __WORKSHED     +1dc: 10c74218 ([0, 0, 2002bfd0, 10c9248e, 0])
10c74298 __WORKSHED     +3ee2: __SERVLOOP_MAINT (0)
10c80562 __SERVLOOP_MAINT+90 : 10c80ad4 ([0, 8fc0, ffff800c, 3f00, 2])
10c80afe __SERVLOOP_MAINT+62c: 10c80b70 ([2002bfb4, 2002bfa0, 10c8066e, 0,
8fc0]
)
10c80cf4 __SERVLOOP_MAINT+822: 10c82d86 (3f, 1)
10c82dac __SERVLOOP_MAINT+28da: 10c82ddc (0, 0, 1)
10c83040 __SERVLOOP_MAINT+2b6e: __CARD_AUDIT (31, 30)
106efe7c __CARD_AUDIT   +1e6: 106f0598 ([7, 7, 7, 0, 0])

dbg> c                      - to continue on after breakpoint
```

3.7 Code Disassembly and Look Up commands

fgn <address>	find global and offset of this absolute <address>
lkup <string>	looks up all functions including this string
adr <global> <offset>	find absolute address of this global and offset
l <0xaddress> <n>	lists assembler for n (decimal) lines

3.7.1 fgn

This command is used for SL1 code addresses only.

```
pdt> fgn 104b2cde          - NOTE: no “0x” in front of the address
GL= 031 (49) OF= 00000006
pdt>
```

3.7.2 adr

```
pdt> adr 31 6               - global and offset should be in hex
real addr:    0x00104b1fbe  - use this address for the l command  below
virtual addr: 00fbf5c7ef
op code:      4e56ffcc
pdt>
```

3.7.3 l command

```
pdt> l 0x00104b1fbe 10
                        __DIGPROC:
104b1fbe  4e56 ffcc                LINK  .W    A6,#0xffcc
104b1fc2  2d4a fffc                MOVE  .L    A2,(0xfffc,A6)
104b1fc6  244e                     MOVEA .L    A6,A2
104b1fc8  48e7 0c00                MOVEM .L    D4-D5,-(A7)
104b1fcc  4aad 01d8                TST   .L    (0x1d8,A5)
104b1fd0  6700 008c                BEQ         0x104b205e
104b1fd4  e9f5 0781 0170 0002 8188
                      BFEXTU  (0x28188,A5,D0.W*1){#30:#1},D0
104b1fde  4a80                     TST   .L    D0
104b1fe0  6744                     BEQ         0x104b2026
104b1fe2  2f2e 000c                MOVE  .L    (0xc,A6),-(A7)
pdt>
```

You can also disassemble code from a return address stack, using its symbol name and offset. You must do this in **su** mode, and the symbol table must be loaded:

```
pdt> symload
Loading symbols from "c:/p/sl1/res.sym"
pdt> su                        - must be in superuser mode
-> l DIGPROC+d452              - don't forget the "0x" for hex number
undefined symbol: d452
-> l DIGPROC+0xd452
104c012a  4e56 ff60                LINK  .W    A6,#0xff60
104c012e  2d4b fffc                MOVE  .L    A3,(0xfffc,A6)
104c0132  264e                     MOVEA .L    A6,A3
104c0134  48e7 1e00                MOVEM .L    D3-D6,-(A7)
104c0138  42ae ff90                CLR   .L    (0xff90,A6)
104c013c  4ab5 2c28               TST   .L    (0x28,A5,D2.L*4)
104c0140  670c                     BEQ         0x104c014e
104c0142  42a7                     CLR   .L    -(A7)
104c0144  4eb5 0171 0002 0d84    JSR       (0x20d84,A5,D0.W*1)
104c014c  588f                     ADDQ  .L    #0x4,A7
->
```

3.8 File Transfer

3.8.1 Xmodem

rx - pdt command to receive a file onto the switch
sx - pdt command to send a file to the PC/workstation

To use **rx**, PDT Level 1 or Level 2 password login is required.

To use **sx**, PDT Level 2 password login is required for security purposes.

IMPORTANT: When transferring data files (patch and database files), ensure that the files are transferred in BINARY mode.

3.8.2 Receive a file onto the switch

To transfer a file from a PC/workstation to the switch:

```
pdt> rx [path/]filename.ext
```

Then enter the appropriate commands to invoke xmodem file transfer on the PC or workstation. See the next sections for examples of how to do this on user's platform.

3.8.3 Send a file to PC/Workstation

To transfer a file from the switch to the PC/workstation:

```
pdt> sx [path/]filename.ext
```

Then enter the appropriate commands to invoke xmodem file transfer on the PC or workstation. See the next sections for examples of how to do this on user's platform.

3.8.4 UNIX Xmodem example

To transfer a patch to the switch:

```
pdt>  cd c:/u/patch
pdt>  rx newpatch.p
```

When the system prompts "`Ready to receive...`", invoke local command mode by typing **~C** (tilde capital-C) and issue the umodem (s)end (b)inary command.

```
~C          (tilde capital-C to enter local command)
umodem -sb ~mydir/patches/newpatch.p
```

When the transfer is completed, a transmission summary is displayed and the pdt prompt is shown.

```
total packets                       :  20
number of retries                   :   0
receive timeouts                    :   1
system errors                       :   0
unknown characters                  :   0
transfer cancelled                  :   0
packets received out of sequence    :   0
packets with corrupted sequence     :   0
packets failed checksum/crc check   :   0
incomplete packets                  :   0
duplicate packets                   :   0
pdt>
```

To transfer the direct.rec file to the workstation:

```
pdt>  cd c:/p/sl1
pdt>  sx direct.rec
```

When the system prompts "`Ready to send...`", invoke local command mode by typing **~C** (tilde capital-C) and issue the umodem (r)eceive (b)inary command.

```
~C          (tilde C to enter local command)
umodem -rb ~mydir/backup/direct.rec
```

A transmission summary will be printed (as above) and the pdt prompt will return.

3.8.5 Windows 95 Xmodem Example

Use the HyperTerminal application to dial up to the switch.

To transfer a patch to the switch:

```
pdt>  cd c:/u/patch
pdt>  rx newpatch.p
```

When the system prompts "`Ready to receive...`", invoke file transfer on the PC side using the (T)ransfer pull-down menu and selecting the (S)end File option.

Select the file to be sent and select XMODEM as the Protocol. Then start the transfer on the PC side.

To transfer the direct.rec file to the workstation:

```
pdt>  cd c:/p/sl1
pdt>  sx direct.rec
```

When the system prompts "Ready to send...", invoke file transfer on the PC side using the (T)ransfer pull-down menu and selecting the (R)eceive File option.

Select or create a file to be received as and select XMODEM as the Protocol. Then start the transfer on the PC side.

3.9 Report Log

This is a system history file. It is a circular file that wraps around at 500K bytes (739 report records). File name is **c:/u/rpt/rpt.log**.

It contains system messages, such as:

- bus errors (BERR)
- tape emulation errors (TEMU)
- INI notification
- SYSLOAD notification
- Midnight routines notification

Some common RPT categories are:

SRPT	- System Report messages
BERR	- Bus Error messages
DLO	- Disk LayOut messages
TEMU	- Tape EMUlation messages
PCH	- Patcher messages

The SL1 history file still exists and contains SL1 message information.

3.9.1 Report Log "Commands"

These are all actually "C" funtions, and therefore you must run **symload** from a level 2 pdt shell first.

- **rdhelp** - help on the report log commands
- **rdopen** - shows info on c:/u/rpt/rpt.log file
- **rdtail** - shows the most recent record headers
- **rd [n]** - prints the next n oldest record headers
- **rdgo nnn** - goes to record nnn
- **rds n** - prints n full records, starting with current and going more recent
- **rdshow** - shows current Report Log file information

See also pp.33-36 in "Inside Thor" documentation.

3.9.2 rdopen

pdt> **rdopen**

```
        Work file :       “c:/u/rpt/rpt.log”
        File status :     full(old reports are replaced by new ones)
        File capacity :   738
        oldest rec :      196 ( 3/ 9/96 21:26:48)
        current rec :     195 (11/ 9/96 01:00:45)
        newest rec :      195 (11/ 9/96 01:00:45)
        display size :    4 (11/ 9/96 16:23:17)

value = 0 = 0x0
pdt>
```

3.9.3 rdtail

pdt> **rdtail**

```
...rd : showing 16 records up to the newest record (rec 195)
  180 : (10/9/96 14:17:00.962) COM001 Ethernet driver: unit 0 is being restarted
  181 : (10/9/96 14:17:00.110) COM001 Ethernet driver: unit 0 is being restarted
  182 : (10/9/96 14:17:00.270) COM001 Ethernet driver: unit 0 is being restarted
  183 : (10/9/96 14:17:00.430) COM001 Ethernet driver: unit 0 is being restarted
  184 : (10/9/96 14:17:00.590) COM001 Ethernet driver: unit 0 is being restarted
  185 : (10/9/96 14:17:00.750) COM001 Ethernet driver: unit 0 is being restarted
  186 : (10/9/96 14:17:01.658) SRPT730 OS 0: Cold Start
        Release:  x112208b
        Created:  Thu Sep 5 22:09:24 PDT 1996
  187 : (10/9/96 14:17:01.705) COM001 Ethernet driver: unit 0 is being restarted
  188 : (10/9/96 14:17:16.077) SRPT750 INI 0: INI on side 0 due to System Cold Start
  189 : (10/9/96 14:17:25.340) SRPT752 INI 0: INI completed in 9 seconds
  190 : (11/9/96 1:00:00.150) SRPT770 TOD 0: Midnight job server starts on side 1
        Number of jobs to do: 0
  191 : (11/9/96 1:00:00.152) SRPT773 TOD 0: Starting midnight job ‘rstThr’
  192 : (11/9/96 1:00:00.153) SRPT774 TOD 0: Midnight jobs completed on side 1
  193 : (11/9/96 1:00:43.217) SRPT770 TOD 0: Midnight job server starts on side 2
        Number of jobs to do: 0
  194 : (11/9/96 1:00:43.219) SRPT773 TOD 0: Starting midnight job ‘pchMidNite’
  195 : (11/9/96 1:00:45.749) SRPT774 TOD 0: Midnight jobs completed on side 2
value = 0 = 0x0
pdt>
```

3.9.4 rd [n]

When this follows rdtail and a subsequent rd, it shows the next n oldest records.

```
pdt> rdtail
...rd : showing 16 records up to the newest record (rec 195)
  180 : (10/9/96 14:17:00.962) COM001 Ethernet driver: unit 0 is being restarted
  181 : (10/9/96 14:17:00.110) COM001 Ethernet driver: unit 0 is being restarted
                               etc....
  194 : (11/9/96 1:00:43.219) SRPT773 TOD 0: Starting midnight job 'pchMidNite'
   195 : (11/9/96 1:00:45.749) SRPT774 TOD 0: Midnight jobs completed on side 2
value = 0 = 0x0

pdt> rd
  164 : (10/9/96 13:42:50.306) COM008 Ethernet driver: unit 0 is being reset
  165 : (10/9/96 13:42:50.308) COM000 Ethernet driver: device unit 0 is
initialized OK.
                               etc....
  178 : (10/9/96 14:17:00.956) COM000 Ethernet driver: device unit 0 is
initialized OK.
  179 : (10/9/96 14:17:00.960) COM001 Ethernet driver: unit 0 is being restarted
value = 0 = 0x0

pdt> rd 4
  180 : (10/9/96 14:17:00.962) COM001 Ethernet driver: unit 0 is being restarted
  181 : (10/9/96 14:17:00.110) COM001 Ethernet driver: unit 0 is being restarted
  182 : (10/9/96 14:17:00.270) COM001 Ethernet driver: unit 0 is being restarted
  183 : (10/9/96 14:17:00.430) COM001 Ethernet driver: unit 0 is being restarted
value = 0 = 0x0
pdt>
```

3.9.5 rdgo nnn

This goes to record number nnn:

pdt> **rdgo 167**

```
  167 : (10/9/96 13:42:50.315) COM001 Ethernet driver: unit 0 is
being restarted
value = 0 = 0x0
pdt>
```

Then you can print the full record using **rds 1**...

3.9.6 rds n

NOTE!!! Always have a non-zero number for n! Otherwise, **ALL** the records will scroll down your screen and you can only stop them by typing Ctrl-(pdt) again!!!

pdt> **rds 1**

```
... rd : record 167

(10/9/96 13:42:50.315) COM001 Ethernet driver: unit 0 is being restarted
Registers (A0-A7, D0-D7):
 100b5964 202ea810 2005ab70 00000000 00000000 00000000 202ea794 202ea4e8
 00000000 00003000 00000000 00000000 00000000 00000000 00000000 00000000
Return Address Stack:
 100b5982 (_quTxRestart+1e)
 10122470 (_netTask+76)
Stack (base = 0x202ea79c):
 01600001 00000000 00000000 202ea7e0 10122470 2005ab70 00000000 00000000
 00000000 00000000 00000000 100b5964 2005ab70 00000000 00000000 00000000
 00000000 00000000 10119dca 00000000 00000000 00000000 00000000 00000000
 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000032
 00000000 00000000 00000000 00000000 00000000 202e7f88 20302e80 00000000
 00000000 200585a8 202e8100 00000007 00000000 00000032 00000032 00000001
 00000000 00000000 00000000 00000000 00000000 00000001 00000000 00000000
 00000000 20049fd8 101223fa 202ea810 202e810c 202e8100 00000000 00000000
 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

value = 0 = 0x0
pdt>
```

HINT: If you do not see the procedure names and offsets (bold above), then run the **symload** command in pdt.

3.9.7 Exercise: Using Report Log File

1. Open the report log file.
   Command used: _____

2. Use rdtail command and answer the following :
   - how many records are displayed? _____
   - what is the newest record number? _____

3. Using the rdgo command print the second last record in the report log file. What is the date/time stamp of this message?

   _____

4. Using rdgo, go to record number 20. Then type: rd 10. What is the result ?

   _____
   _____
   _____
   _____
   _____

5. Using rdshow command, determine the following:
   oldest record _____
   current record _____

4.0 Symbol Files

4.1 Finding symbol files

The symbol files are available from one of the following:

1. /auto/opt11c directory contains the .sym files for various releases of software. Check this directory out before using one of the other methods.

```
nbvws094{yoonhi}11: pwd
/auto/opt11c
nbvws094{yoonhi}12: ls
0100/            2353b/          2422d/
0101/            2354/           2422e/
0110/            2355/           2423/
0202/            2403e/          2424/
0202C/           2403g/          2425/
11C_DB/          2404a/          2501a/
2208d/           2404c/          2502a/
2216/            2404e/          2502e/
2216_ATV/        2404f/          CFI/
2245/            2404g/          DesignDocs/
2246/            2406/           DevPlan/
2262/            2407/           IPREMOTES/
2318/            2409/           Ideas/
2335/            2421a/          InsideOpt11C/
2335_ATV/        2421c/          Opt11CDebug/
2347/            2421d/          Opt11C_docs/
2351b/           2422a/
2353/            2422c/
nbvws094{yoonhi}13:
```

2. The patchres.sym file is found in the workfile directory. Use the prep tool to prep a required workfile. Then change directory to: /net/medpro1/data/products/SL1/mediapro/24.04E_2111_NATV_<mine>___.1 .

```
nbvws094{yoonhi}19: prep 23.35_2111_NATV
Prep Version 1.04
Copying: 23.35_2111_NATV
   From: workfile archive
    To: /net/medpro1/data/products/SL1/mediapro/23.35_2111_NATV_yoonhi__.1

Extracting files from tar (23.35_2111_NATV.Z)

nbvws094{yoonhi}20: cd /net/medpro1/data/products/SL1/mediapro/
23.35_2111_NATV_yoonhi__.1
nbvws094{yoonhi}21: ls
bootrom/          dramos.sym*       ovlres.txtsym*    patchos.sym*
dflt_db/          dramoscc.sym*     p/                patchres.sym*
dramos*           foxsys.audit*     patchdramos.sym*  u/
```

3. prep -s <workfile name> extracts all the .sym files (except ovlres.txtsym) from the specified workfile name and places the files in the current directory.

```
nbvws094{yoonhi}23: pwd
/auto/yoonhi
nbvws094{yoonhi}24: mkdir temp
nbvws094{yoonhi}24: chmod 777 temp
nbvws094{yoonhi}24: cd temp
nbvws094{yoonhi}25: ls
nbvws094{yoonhi}26: prep -s 23.35_2111_NATV
Prep Version 1.04
Copying: 23.35_2111_NATV
   From: workfile archive

All the symbol tables in the workfile will be copied to your current
directory.

nbvws094{yoonhi}27: ls
23.35_2111_NATV/
nbvws094{yoonhi}28: cd 23.35_2111_NATV
nbvws094{yoonhi}29: ls
dramos.sym*       p/                patchos.sym*
dramoscc.sym*     patchdramos.sym*  patchres.sym*
nbvws094{yoonhi}30:
```

4.2 Types of symbol files

1. patchres.sym
   - used when creating patches for flash system, diskos, sl1res, or ovlres.

2. patchdramos.sym
   - used when creating patches for dramos.

3. res.sym
   - this is a binary file of all symbols in the M1 system (c and SL1)
   - this file is used by diskos, sl1res, ovlres

4. dramos.sym
   - this file is used by dramos.

5. ovlres.txtsym
   - this is a text file readable by designers, and its contents is same is res.sym.

symload command loads the res.sym file into the memory.

Debugging

5.0 PDT

5.1 Breakpoint commands

b	show all breakpoints
b <0xaddress>	set a breakpoint at <address>
bd <0xaddress>	delete breakpoint
bdall	delete all the breakpoints
br <0xaddress>	breakpoint resume
bs <0xaddress>	breakpoint suspend
c	continue code execution after hitting the breakpoint
s	allows to single step a task that is stopped at a breakpoint
so	allows to single step a task that is stopped at a breakpoint. However, if the next instruction is a JSR or BSR, it breaks at the instruction following the subroutine call

5.2 Set breakpoint

To set any breakpoint it is necessary to find the address where the breakpoint is to be set. The easiest way to do this is to use Xview to find the Global and offset and then determine the actual address using the adr command in PDT.

Note: the following example is assuming that the **symbol table** has been loaded.

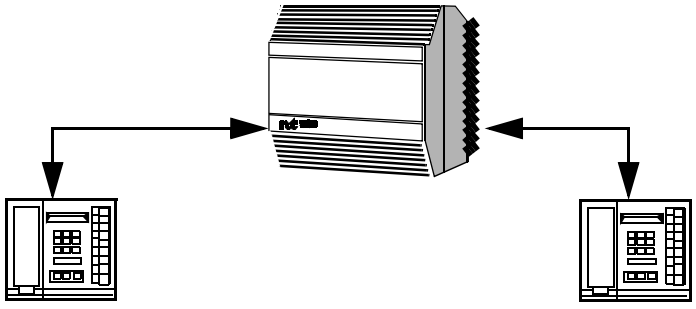
Example:

1. Find the real address of the SET_DIALLED_DN procedure in the DIGPROC module such that a breakpoint can be set at this address.
2. From the Xview Navigator type in SET_DIALLED_DN, then click on the phrase
3. Click on the NEXT icon in the Main Xview window.
4. This procedure is found in the DIGPROC Module which contains the global procedure DIGPROC (global number H.31). SET_DIALLED_DN is a local procedure.
5. Go into PDT
6. Enter debug mode
7. From Xview determine the offset into the global (in this case it's 1ce2)
8. Determine the real address from the global and offset:
   pdt> **adr 31 1ce2**
9. The output is as follows:
   real addr: 0x00104cc86a
   virtual addr: 00fc05f21a
   op code: 4e56ffe8
10. To set a breakpoint:
    pdt> **b 0x00104cc86a**
    the output is:
    0x104cc86a (bp# 1): __DIGPROC +0x1cdc Task: all Count: 0

The **BREAKPOINT** is now set.

5.3 Finding out information when a breakpoint is hit

Now that we can set breakpoints in the SL1 code, what information can we find out? Lets look at the above breakpoint and setup a simple call between 2 digital sets.



SET A
DN 4444
TN 4-0

SET B
DN 5432
TN 4-2

The originator of the call, SETA dials SET B

After the last digit is dialled from SETA (digit 2) the breakpoint is hit.

```
dbg>
Break at 0x104cc86a (bp# 1): __DIGPROC      +0x1cdc   Task: 0x20429da0 (tSL1)
```

5.3.1 Finding the tSL1 stack

The tSL1 stack can be found by using the command tt (task trace)

dbg> **tt**

10124418 _vxTaskEntry +10 : _sl1Main (0, 0, 0, &_sl1Entry, 20337318, 100976e8, &_foxLCDPrintf, 20337328, 10097804, 2)
10d45a42 _sl1Main +f8 : 1064f170 ([0, 0, 1012441a, 0, 0])
1064f280 __SYSLOAD1 +152: 10d18eec ([1064f200, 0, f, 1, 1])
10d18f9a __WORKSHED +1e8: 10d1cdaa ([f, 2002bfc4, 2002bfc4, 1064f288, 1064f200])
10d1cd84 __WORKSHED +3fd2: 10d1fd66 ([0, 2002bf30, 10d18fd4, f, 2002bfc4])
10d1ffd8 __WORKSHED +7226: 10d20008 ([0, f, 3, 2, 1])
10d204d6 __WORKSHED +7724: 10d204fc ([4, 0, 1, 2002bf30, 2002bf14])
10d204d6 __WORKSHED +7724: 10d20864 ([1, 2002bef4, 2002bef4, 10d204d8, 4])
10d20872 __WORKSHED +7ac0: __TCM_INPUT_MSG ([0, 2002bed0, 10d2058e, 1, 2002bef4])
10538778 __TCM_INPUT_MSG+1be: 10539214 ([4, 0, 1, 0, 4])
10539382 __TCM_INPUT_MSG+dc8: 1053c9de ([c5480, ce1ec, ce1ec, 2002beb4, 2002bed0])
1053cdb8 __TCM_INPUT_MSG+47fe: 102f9b44 (3)
102f9a28 __EES_CHECK +156: 1030266a ([2002beb4, 2002beb4, 2002be48, 1053cdc0, 3])
1030311a __SL1_FUNCTION +95e8: __INTERDIG_TIMING ([9ee8, c54db, 0, 10, 0])
1078a846 __INTERDIG_TIMING+1b0: __DIGPROC (0, 0, 0)
104cac5c __DIGPROC +ce : 104cace4 ([4, 0, 0, 2, 0])
104cadba __DIGPROC +22c: 104cbcc0 ([4, 0, 0, 2002bd84, 2002be18])
104cbe3e __DIGPROC +12b0: 104cbea8 ([4, 0, 4, 2002bd30, 104b6b9e])
104cc1fc __DIGPROC +166e: 104cc86a ([0, 0, 0, 2002bec0, 2002bd1c])

5.3.2 Finding the local variables

The Local variables that are stored on the stack can be found by entering the command loc.

```
dgb> loc
10124418 _vxTaskEntry   +10      : _sl1Main      +00000000 (0, 0, 0, 10200044, 20337318,
100976e8, 10103d76, 20337328, 10097804, 2)
10d45a42 _sl1Main       +f8      : __SYSLOAD1    +00000042 ()
                        local variables:
                        0000000f 00000001 00000001 00000000
                        00001806 00000000 00000000 00000000
                        00000000 00000000 00000000 00000000
                        00000000 00000000 00000000 00000000
                        00000000 00000000 00000000 00000000
                        00000000 00000000 00000000 00000000
                        00000000 00000000 00000000 00000000
                        00000000 00000000 100879d4 00000000
1064f280 __SYSLOAD1     +152     : __WORKSHED    +0000013a ()
10d18f9a __WORKSHED     +1e8     : __WORKSHED    +00003ff8 ()
10d1cd84 __WORKSHED     +3fd2    : __WORKSHED    +00006fb4 ()
                        local variables:
                        00000003 00000002 00000001
10d1ffd8 __WORKSHED     +7226    : __WORKSHED    +00007256 ()
                        local variables:
                        2002bf30 2002bf14 10d1fecc
10d204d6 __WORKSHED     +7724    : __WORKSHED    +0000774a ()
10d204d6 __WORKSHED     +7724    : __WORKSHED    +00007ab2 ()
10d20872 __WORKSHED     +7ac0    : __TCM_INPUT_MSG+00000000 ()
                        local variables:
                        00000004 2002be9c 105d415c 00000000
                        00000000 00000004 00048da8 2002becc
                        2002becc 105d40f0 00000000 00000000
                        00000002
10538778 __TCM_INPUT_MSG+1be     : __TCM_INPUT_MSG+00000c5a ()
                        local variables:
                        000ce1ec 2002beb4
10539382 __TCM_INPUT_MSG+dc8     : __TCM_INPUT_MSG+00004424 ()
                        local variables:
                        10611a12
1053cdb8 __TCM_INPUT_MSG+47fe    : __SL1_FUNCTION +00000012 (3)
                        local variables:
                        2002beb4
102f9a28 __EES_CHECK    +156     : __SL1_FUNCTION +00008b38 ()
                        local variables:
                        00000010 00000000 2002be38 103177ac
```

```
                00009ee8 1060c3b0 1060c25e 00000001
                2002be20 2002be20 2002be48 2002be20
                00000001 00000000 00000000 00000000
                00000000
1030311a __SL1_FUNCTION +95e8    : __INTERDIG_TIMING+00000000 ()
                local variables:
                00000000 2002be28 2002be18 00000001
                00000000
1078a846 __INTERDIG_TIMING+1b0     : __DIGPROC     +00000000 (0, 0, 0)
                local variables:
                00000000 00000002 00000000 00000000
                10953f64 10953d44 00000002 00000001
                00000000 00000000 00000000 00000000
                00000000 00000000
104cac5c __DIGPROC     +ce      : __DIGPROC     +00000156 ()
                local variables:
                00000000 2002bd84
104cadba __DIGPROC     +22c     : __DIGPROC     +00001132 ()
                local variables:
                00000004 2002bd30 104b6b9e 00000000
104cbe3e __DIGPROC     +12b0    : __DIGPROC     +0000131a ()
                local variables:
                00000000
104cc1fc __DIGPROC     +166e    : __DIGPROC     +00001cdc ()
                local variables:
                2002bd1c 2002bcf8 20350000 2002bcf8
                2002bcd8
```

5.3.3 Finding the tSL1 register information

To find the contents of the Registers:
>dbg **ti tSL1**

```
  NAME       ENTRY       TID    PRI   STATUS      PC       SP     ERRNO  DELAY
---------- ------------ -------- --- ---------- -------- -------- ------- -----
tSL1       _sl1Main     20429da0 240 SUSPEND    104cc86a 2002bce4  3d0002     0

stack: base 0x2002c000  end 0x20020000  size 49144  high 5188   margin 43956

options: 0x10
VX_STDIO

d0    =       0   d1    =    c57a4   d2    =    ce1ec   d3    =       2
d4    =       0   d5    =       0   d6    =       0   d7    =       0
a0    =    4d1ca   a1    =    ce1ec   a2    = 2002bd84   a3    = 2002bd1c
a4    = 2002bcf8   a5    = 20350000   a6/fp  = 2002bcf8   a7/sp  = 2002bce4
sr    =    3004   pc    = 104cc86a
dbg>
```

5.3.4 Finding the call register information

This can be done by using the crg command, i.e. in our case crg 4 0, this gives us the following:

dbg> **crg 4 0**

```
 SLOOP TN SLOOP   4  0  0   0 * 000010  EQPD
 CUST  0 PBLK 0004D19F UBLK 000C5480
BCS

A CR 000CE1EC
00000200 00000001 00000C00 00000000 00000000 00000001 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000010 00000000 00000000
00000000 00000000 00000000 00000000 00000084 00002345 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000001 00008000 00000000 00000000 00000000 00000000 00000002
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000010 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000200 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 000000B9 00000000 00000000 00000000 00000000 00000000
00001FF8 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
KEY 00 000CE1EC
```

An other way of finding the call register information can be done by looking at the Registers and determining which register holds the current call register pointer, CRPTR (Register D2 is used for the CRPTR).

To print the first H.30 words of call rgister:

dbg> **p ce1ec 30**

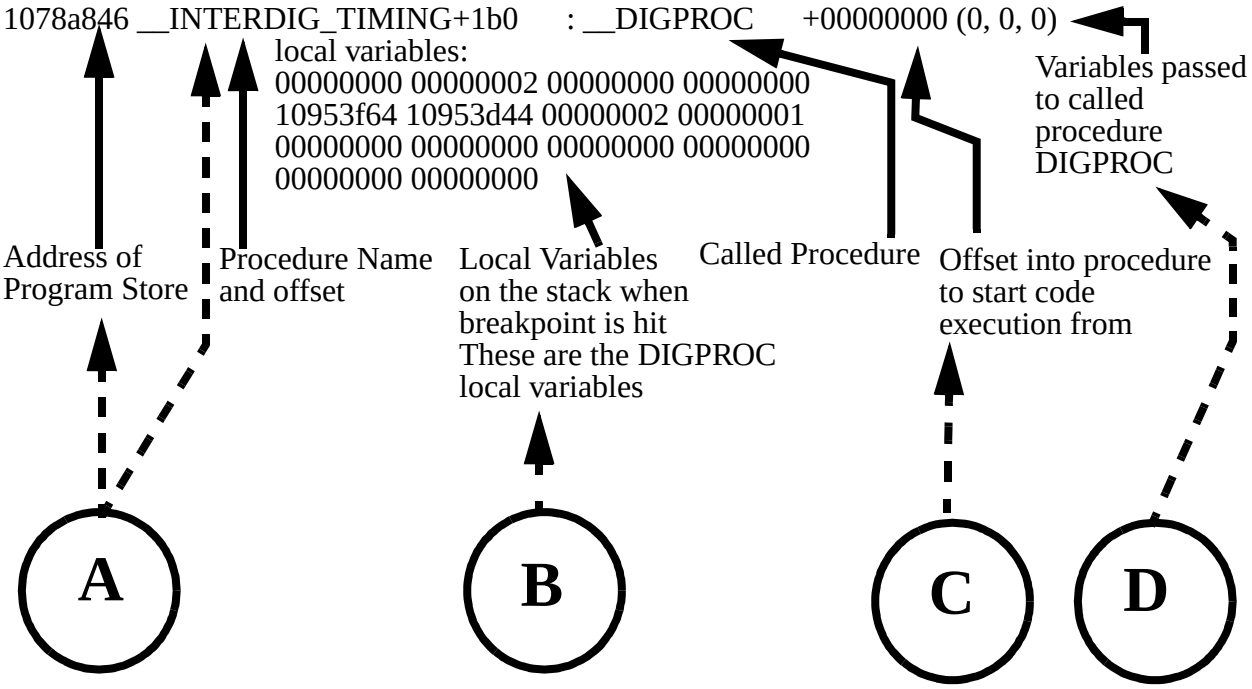
000CE1EC : 00000200 00000001 00000C00 00000000 00000000 00000001 00000000 00000000
000CE1F4 : 00000000 00000000 00000000 00000000 00000000 00000010 00000000 00000000

```
000CE1FC : 00000000 00000000 00000000 00000000 00000084 00002345 00000000 00000000
000CE204 : 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000CE20C : 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000CE214 : 00000000 00000001 00008000 00000000 00000000 00000000 00000000 00000002
```

5.4 Using the Information

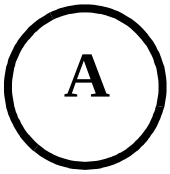
5.4.1 Local variables

Here is a section of local variable information from the loc command.



For correlation of A,B,C,and D with the Xview listing see below.

5.4.1.1 Examine A



This is the address of the program store.
:

In Xview, search for INTERDIG_TIMING, and offset 1b0.
INTERDIG_TIMING is Global H.10d and defined in the MIX311 module.

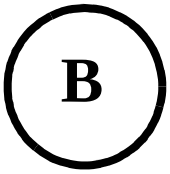
At offset 1b0, the following is seen:

```
                                    3375  1 3          DIGPROC(0,0,0);
0001b0 L269_6:   clrl    a7@-
                    clrl    a7@-
                    clrl    a7@-
                    jsr     a5@(0x20cc4:l)@(0)
                    addl    #0xc,a7
```

This is where DIGPROC was called from, if we do a list on the above address we see the following:

```
pdt> l 0x1078a846
1078a846  4eb5 0171 0002 0cc4    JSR          ([0x20cc4,A5],0)
1078a84e  dffc 0000 000c         ADDA   .L     #0000c,A7
1078a854  2f02                   MOVE   .L     D2,-(A7)
1078a856  4eb5 0171 0002 3660    JSR          ([0x23660,A5],0)
1078a85e  588f                   ADDQ   .L     #0x4,A7
1078a860  4a80                   TST    .L     D0
1078a862  677c                   BEQ           0x1078a8e0
1078a864  4aae fff8              TST    .L     (0xfff8,A6)
1078a868  671e                   BEQ           0x1078a888
1078a86a  200e                   MOVE   .L     A6,D0
```

5.4.1.2 Examine B



These are the values of the local variables declared in DIGPROC. Go to the declaration of DIGPROC in Xview, you see the following INTEGERS and POINTER declared.

```
INTEGER TRY_CFWNA
        DNTRANSVALUE,
        ESA_TRANSVALUE,
        TRY_NIGHT_SERV,ORIG_COSTGAR,TER_COSTGAR,
        OLDMAINPM, HUNT_INCR, ENDHUNT_CALLED,
        TRY_MW,
        INT_CHECK_DONE, INT_FLAG,
        TRY_SFNA;
PPOINTER [.P_CUST_DATA] ESA_BLK_PTR;
```

From the loc information we see the local variables values:

```
00000000 00000002 00000000 00000000
10953f64 10953d44 00000002 00000001
00000000 00000000 00000000 00000000
00000000 00000000
```

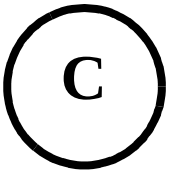
As you can see there are 14 local variables,ESA_BLK_PTR is the last and TRY_CFWNA is the first.

Knowing the variables on the stack can be very useful, but it is necessary to follow the s/w and see where the variable is set, for example consider the variable OLDMAINPM, this is the 7th variable on the stack, from the above it can be seen that its value = 2,
Go to Xview and click on the variable OLDMAINPM, we see that it occurs 3 times and is assigned once, (the 2nd instance). It is set equal to the MAINPM in the call register. From the other information that was obtained about the call register we can see that the MAINPM was set = 2, i.e.

```
dbg> p ce1ec 30

000CE1EC : 00000200 .............
```

5.4.1.3 Examine C



The calling procedure INTERDIG_TIMING calls DIGPROC with 3 variables. The address that the s/w starts to execute at is at offset 0 in DIGPROC, however, if you look at DIGPROC in Xview you will see that the first offset is 000006 with the first instruction being a link. To see what the s/w does at offset 0, type in from the pdt shell adr 31 0, the response should be:

pdt> **adr 31 0**

real addr: 0x00104cab88
virtual addr: 00fc05eae2
op code: 60044e75

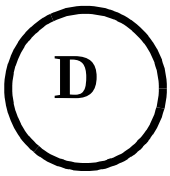
now unassemble this by entering: (note the command is l (list))

pdt> **l 0x00104cab88**

```
                _DIGPROC:
104cab88  6004              BRA       __DIGPROC
104cab8a  4e75              RTS
104cab8c  4e71              NOP
                __DIGPROC:
104cab8e  4e56 ffc4         LINK      .W    A6,#0xffc4
104cab92  2d4a fffc         MOVE      .L    A2,(0xfffc,A6)
104cab96  244e              MOVEA     .L    A6,A2
104cab98  48e7 0c00         MOVEM     .L    D4-D5,-(A7)
```

As you can see at offset 0 there is a BRA (Branch Always) instruction to offset 6.

5.4.1.4 Examine D



DIGPROC is called from INTERDIG_TIMING with three parameters, in our case they are 0,0,0. These values have to be stored on the stack in addition to the local variables.

It is outside the boundaries of this course to detail how the variable information is stored on the stack, an understanding of the MOTOROLA instruction set is needed in order to show how local variables are stored. HOWEVER, for debugging purpose we don’t need to know where they are stored, just what they are.

5.5 Exercise: using PDT and breakpoint commands

1. Set a breakpoint at top of global procedure SL1_FUNCTION.
   Command used:

   _____
   _____
   _____
   _____

2. From a digital set, press a DN key.
   - Did this action stop at the breakpoint set? Yes / No
   - If No, determine the reason why it did not break, correct and retry.
   - When the system stops at the required breakpoint,
     - Determine if there is a call register associated with the digital set: Yes / No
     - What command(s) did you use?

       _____
       _____
       _____
       _____
       _____
       _____

3. A parameter is passed to SL1_FUNCTION.
   - What is the value of the parameter passed? _____
   - How did you determine the answer?

     _____
     _____
     _____
     _____
     _____
     _____
     _____

4. Based on the parameter passed, another procedure is called.
   - What is the name of this procedure? _____
   - Set a breakpoint at the top of this procedure, and demonstrate that the procedure is called. What steps did you take?

     _____
     _____
     _____
     _____
     _____
     _____
     _____

5. Remove all breakpoints, and return to the non-breakpoint mode.

6. Set a breakpoint at top of global procedure RING500. From a digital set, call a 500 set. The call processing stops at the breakpoint before the 500 set rings.
   - Which procedure calls RING500?
     _____
   - Is this procedure local or global? _____
     - If it is a local procedure, in which global procedure is it residing?
       _____
   - What is the value of the parameter passed to RING500, and what does this value indicate?
     _____
     _____
     _____
   - What is the mainpm value and what does that indicate?
     _____
     _____
     _____

7. Using 2 digital sets (Set A dials Set B), determine the following:
   - In procedure RING_BCS_SET, variable SSDPTR is assigned a value. Determine its value.
     _____
     _____
   - Which datablock is accessed using this pointer?
     _____
     _____
   - After line 418: IF UNIT_SUBTYPE:PL_PTR:ITEMPTR ..... , determine which fields are assigned a value using the pointer variable SSDPTR.
     _____
     _____
     _____
     _____
     _____
     _____
     _____

8. What is the starting address of the global procedure WRITE_TSIC?
   _____
   _____

5.6 Using Single Stepping

Lets look at our previous breakpoint and single step through the code. We will tie this in with Xview and the dis-assembled listing on the switch. Remember the breakpoint is set at the start of procedure SET_DIALLED_DN, the address is 0x104cc86a.

First let look at the dis-assembled code starting at the breakpoint address.

dbg> **l 0x104cc86a**

```
104cc86a  4e56 ffe8          LINK     .W   A6,#0xffe8
104cc86e  2d6d 0010 fffc      MOVE     .L   (0x10,A5),(0xfffc,A6)
104cc874  200e               MOVE     .L   A6,D0
104cc876  908d               SUB      .L   A5,D0
104cc878  e488               LSR      .L   #0x2,D0
104cc87a  2b40 0010          MOVE     .L   D0,(0x10,A5)
104cc87e  48e7 0400          MOVEM    .L   D5,-(A7)
104cc882  2a35 2c48          MOVE     .L   (0x48,A5,D2.L*4),D5
104cc886  2035 2c4c           MOVE     .L   (0x4c,A5,D2.L*4),D0
104cc88a  8a80               OR       .L   D0,D5
```

5.7 Modifying the software flow

It is sometimes useful to change the software flow to see what happens if another path is taken. For example what would happen if a certain condition was met but under normal circumstances should never be met. This is a requirement when developing a new feature to unit test the exception and error conditions.

1. Search for **.BUG9005** in Xview.

2. There are two occurrances of the constant in DIGPROC.

3. Set a breakpoint with the intention of modifying the software to execute the BUG procedure and print the BUG9005.

4. The offset in DIGPROC where to set the breakpoint is c2.
   dbg> **adr 31 c2**

   real addr: 0x00104cac4a
   virtual addr: 00fc05eb12
   op code: e9f50702

5. Set breakpoint: dbg> **b 0x00104cac4a**

6. Dial any digit from a Digital Set.

7. The breakpoint is hit:
   Break at 0x104cac4a (bp# 6): __DIGPROC +0xbc Task: 0x20429da0 (tSL1)

8. Find the Register information
   dbg> **ti tSL1**

```
 NAME       ENTRY      TID    PRI   STATUS      PC       SP     ERRNO  DELAY
 ---------- ---------- -------- --- ---------- -------- -------- ------- -----
tSL1      _sl1Main     20429da0 240 SUSPEND    104cac4a 2002bd40  3d0002     0

stack: base 0x2002c000  end 0x20020000  size 49144  high 5004   margin 44140

options: 0x10
VX_STDIO

d0    =           1   d1    =       c7b57   d2    =       cb90f   d3      =           0
d4    =           1   d5    =           0   d6    =           0   d7      =           0
a0    =      484be    a1    =       cb90f   a2    = 2002bd84    a3      = 2002be18
a4    = 2002bec0    a5    = 20350000    a6/fp   = 2002bd84    a7/sp  = 2002bd40
sr    =      3010    pc    = 104cac4a
```

9. Single Step until the next instruction to be executed is the TSTL

```
dbg> s
d0    =          0   d1    =      c7b57   d2    =      cb90f   d3      =          0
d4    =          1   d5    =          0   d6    =          0   d7      =          0
a0    =      484be   a1    =      cb90f   a2    = 2002bd84   a3      = 2002be18
a4    = 2002bec0   a5    = 20350000   a6/fp  = 2002bd84   a7/sp  = 2002bd40
sr    =       3014   pc    = 104cac52
104cac52  4a80              TST   .L    D0
```

10. Looking through Xview, high level code is:

```
IF PTU_SRCH_NO_SEIZ:CRPTR ^= .ACTUAL_SEARCH THEN
BEGIN
    BUG(.BUG9005);
    RETURN;
END
```

11. The corresponding Assembly code for the IF ..... is

```
0000c2      bfextu   a5@(0x2cc,d2:l:4){#28:#2},d0
            tstl     d0
            beq      L49_4
```

12. The breakpoint is at where the next instruction to be executed is the TST D0 (which is checking if the Register D0 = 0).

13. D0 = 0 which means that the next instruction BEQ to L49_4 will occur.

14. Change the path of software by changing the value of D0.

15. Use the command mRegs

```
dbg> mRegs
d0    : 00000000 - 1 <cr>
d1    : 000c7b57 - . <cr>              % . to end the changes
value = 0 = 0x0
```

16. Let the software continue through the new path and execute the BUG:
    dbg> **c**

17. The bug message is printed:

```
BUG9005
BUG9005  +   1060A84C 104CAC64 1078A84E 10303122 102F9B94
BUG9005 + 1053CDC0 105393CC 105387E6 10D2087A 10D2058E
BUG9005  + 10D204D8 10D1FFDA 10D1CE1C
```

5.8 Exercise: dis-assembling code and single stepping

Modify the software flow to cause BUG165.

In procedure DNTRANS line 758 (offset 11c8), there is a check for the following:

```
IF (ATTNCUSTNO:CRPTR ^= CUSTNO:PLORIGPTR) THEN .....
```

1. Set a breakpoint at this point.
2. Make a call, and call processing stops at this breakpoint.
3. Disassemble the code.
4. Using single stepping and mRegs command, ensure that d0 and d4 comparison returns false, and cause BUG165 to be printed.

5.9 Breakpoints in non-SL1 code

It is often required to debug non SL1 code. Setting breakpoints is still the same. Below is an example of setting a breakpoint in a C code procedure.

1. Set a breakpoint in procedure BO3_WRITE.

2. From XView, BO3_WRITE is an external global procedure.

3. From a workstation terminal, **Establish to the 2335 context and go to the flatdir directory.**

   establish x112335 311

   flat

   cd nonsl1_dir

4. Find the code for BO3_WRITE. SL1 code for the Option 11C is written in CAPITALS, however the C code is written in lower case, so a good guess would be to look for bo3*.*. This will give all the bo3 related code including any header files.

5. ls bo3*.*

   bo3.h@ bo3Main.c@ bo3test.seg@

6. In this case look for the BO3_WRITE function in bo3Main.c

7. Finding the address of this procedure is the same as any other address, we can use the lkup command to give us the starting address of BO3_WRITE.

8. lkup BO3_WRITE gives us

   dbg> **lkup BO3_WRITE**
   _BO3_WRITE 0x10d4e03c text

9. Set a breakpoint and make a conference call.

5.10 Setting breakpoints in INITIALIZE

Breakpoints can be set in the procedure INITIALiZE, but the breakpoint will be cleared when the switch initializes. This can be overcome by first obtaining the required offset for the breakpoint, initialize the switch and go into PDT before the sl1 task starts to run.

1. In XView, lookup INITIALIZE. Look at files, select init311, scroll to offset e6.
2. Obtain the address at offset e6 in the global initialize (h.1)

   dbg> **adr 1 e6**
   real addr: 0x00106b4e76
   virtual addr: 00fc0d939d
   op code: 6100188c

3. The breakpoint address is 0x00106b4e76. Record this for reference.

4. Reboot (reboot) the switch and keep entering Cntl PDT until sl1run bypassed is printed.

5. Enter the pdt password. (Note: PDT not active ! message is received each time Cntrl PDT is entered until the password prompt.

6. Enter DEBUG.

7. The adr command cannot be used because the sl1 task is not running:

   dbg> adr 1 e6
   The SL-1 task, tSL1, is not defined.

8. Since the breakpoint address is known, set the breakpoint:

   dbg> **b 0x00106b4e76**
   0x106b4e76 (bp# 1): __INITIALIZE +0xe0 Task: all Count: 0

9. Resume the sl1 task:

   dbg> **sl1run**

   [Initialising Orbix Error handling]
   [Constructing a new Proxy Factory Table]
   Start SNMP agent
   Processed c:/p/data/view.cfg
   Loading MAT script files.....
   dbg>
   Break at 0x106b4e76 (bp# 1): __INITIALIZE +0xe0 Task: 0x20429da0 (tSL1)

10. Global Initialize is ready to be debugged.

11. To continue: bdall, c. Watch for INI messages.

5.11 setting breakpoints in overlays

Overlay debugging with breakpoints is not too different than other debugging.

The GLOBAL number for any overlay is 6. Load the overlay first before setting a breakpoint. It is possible in some of the overlays that the overlay calls other globlal procedures. In this case, the global number for the global procedure must be used to when finding the real address.

Set a breakpoint in OVL 10, at the point where the input for the Customer number is required.

1. In the Xview navigator window find the identifier INCUSTNO.

2. Then go to the SCPBX311 module.

3. At the second occurrence (PROCEDURE INCUSTNO), the offset is b52a.

4. Load overlay 10

5. Go into PDT
   dbg> **adr 6 b52a**
   real addr: 0x001166612a
   virtual addr: 00fc4c584a
   op code: 4e56fff8

6. Set a breakpoint at 0x001166612a.

7. Type: sl1input to return to overlay 10 and put in a new 500 type set.

8. Note: When the breakpoint is hit, enter PDT.

9. At this point the user cannot toggle back to the overlay while the breakpoint is set .

5.12 Exercise: setting breakpoint in overlay

Using XView, find the overlay program associated with overlay 11.

Determine the procedure that outputs the keyword KEY. Set a breakpoint before KEY is prompted, and demonstrate that this breakpoint is reached.

6.0 RAS Trace

When the call processing software detects information which is not in the correct format or when invalid information is detected, a BUG message is output.

Hardware problems are generally reported with ERR messages.

With X11 release 19 and later, BUG messages under 1000 will appear in 3 digit output. There is no preceding zero.

BUG messages generally appear in the following format:

BUGxxxx <return address stack>

Some messages contain a second line with additional data, as listed below.

crptr
CRWORD 0 :ptr
ORIGTN :ptr
TERTN:ptr
QUEUEIN:ptr

Example:

% BUG195 : 0050222E 00001001 00008008 0000395F 00000000 0 00000000
00000000 0000
00
% 00 0 84301 000000A5 0000A348 00000001 00000000 0000000A 00000000
00006000
% BUG195 + 046CA73C 046C12CE 046C0D80 046C2E90 048010CC
% BUG195 + 0480097C 047B83C2 047B7C74 047B4AAE 047AE03A
% BUG195 + 04A9C1B8 04A9C0F4 04A996EA

6.1 Starting RAS Trace

1. Look for the ovlres.txtsym file for the establish context. Many of the symbol files are found in /auto/opt11c directory. Look in the appropriate release number. If the release you are looking for is not found in this directory, *prep* (prep 23.35_2111_NATV) the workfile. The required ovlres.txtsym file is found in
   /net/medpro1/data/products/SL1/mediapro/23.35_2111_yoonhi__1 directory.

2. rasTrace -s /auto/opt11c/2335/ovlres.txtsym
   Address Procedure offset
   Next Address: 104b2cde
   104b2cde: DIGPROC 0x6
   Next Address: <cr> to exit

6.2 Exercise: using RAS Trace

Open /auto/yoonhi/M1Course_Files/bug6244.2335 file. Using RAS Trace tool determine where the BUG message was generated.

_____
_____
_____
_____

7.0 FBUG

FBUG is an Motorola 68040 low-level debugger. It is invoked by entering Ctrl-B during the boot-up sequence. Enter Ctrl-B when “Enter cntrl I to perform install” ... appears.

7.1 Accessing FBUG

FBUG is mostly used by hardware people or in the early stage of design. However, it can also be used in the lab to erase a corrupted c: drive.

`Ctrl-B` - when “Enter cntrl I to perform install” ... appears

This places the user at FBUG> prompt. At this point the user can enter any of the standard FBUG command.

7.2 FBUG commands

7.2.1 manufacturing test menu

FBUG> `ct` - for catalog

From here several choices are available.

Enter selection: `t` - for troubleshoot menu
Enter selection: 8 - flash test menu
Enter selection: 6 - erase c: drive on flash (8/16 Mb)
7 - erase program store on flash
99 - to reboot (equivalent to reboot -1)

7.2.2 help

FBUG> ?,help - view a list of allowable commands, and the syntax

Symbols used to describe the command usage can be looked up also.

7.2.3 Memory Display

Displaying addresses in FBUG (remember 16-bit access):

FBUG> **`md 80000182`** **-** note do not put the **`0x`** before the hex address

```
$80000182:  $FF01  $FFFF  $FF00  $FFFF  $FF00  $FFFF
$8000018E:  $FF00  $FFFF  $FF00  $FFFF  $FF00  $FFFF
```

```
$8000019A:  $FF00  $FFFF  $FF00  $FFFF  $FF00  $FFFF
$800001A6:  $FF00  $FFFF  $FF00  $FFFF  $FF00  $FFFF
$800001B2:  $FF00  $FFFF  $FF00  $FFFF  $FF00  $FFFF
$800001BE:  $FF00  $FFFF  $FF00  $FFFF  $FF00  $FFFF
$800001CA:  $FF00  $FFFF  $FF00  $FFFF  $FF00  $FFFF
$800001D6:  $FF00  $FFFF  $FF00  $FFFF  $FF00  $FFFF
```

7.2.4 Memory Modify

```
FBUG> mm 80000182

Directives.
Backup -[<number>], Advance +[<number>], Help ‘?’, Quit ‘q’ or
‘.’

$80000182 $FF01 ? 1
$80000184 $FFFF ? q
```

8.0 M1 Overlays for Debugging

8.1 Overlay 77

The Manual Print Program is a maintenance program which allows monitoring and simulation of input/output signals between a terminal CPU. Refer to LD 77 User's Guide for commands. Be advised that not all the commands are applicable for Option 11C.

- LD 77
- pswd 9950

Example 1: P 007 01 - prints incoming messages, for the duration of being in OVL 77.

- call from TN 7 1
- press DN key and hold
- release DN key
- <L S C U> <msg> <timestamp>

Example 2: DMTN - prints incoming/outgoing messages until stopped

- => TN 1 007 0 00 01 - first parameter is monitor number
- => ISSD <packed tn> <msg> <timestamp> - Incoming
- => OSSD <packed tn> <msg> <timestamp> - Outgoing
- kill 1 - monitor number

8.2 Overlay 80

LD 80 provides a means fo tracing a call by looking at a snap shot of the transient data (such as call register contents) associated with the call. The trace commands operate only when this overlay is active. If LD 80 is aborted (****), the trace function stops.

There are three basic commands in LD80:

- TRAT tracing attendants
- TRAC tracing sets and trunks
- TRAD tracing call through CPI, DTI loops

Example: LD 80

- TRAC <cust #> <DN>
- type TRAC command, as above, for each of the following steps:
  500 set offhook
  dial a digit (for a digital set DN)
  dial all digits
  answer digital set

Patching

9.0 Patching

What is a patch? It is code that replaces part of the X11 SW load on an M1 system.

9.1 Types of Patches:

There are three types of patches:

1. High level patches
   a) C patch (operating system, install program)
   b) SL1 patch global
   c) SL1 patch local
2. Memory Patch is used when patching only a few bytes in the memory.
3. Manufacturing Installed patch used for any one of the above. The patch(es) are programmed with the issue of software to be installed automatically with an upgrade.

WARNING: There is NO machine-type checking on the switch. Make sure that an Option 11C patch is loaded on an Option 11C system.

9.2 Patch Directories

1. DRAM OS patches are on the Software Delivery Card, in directory a:/u/patch.
2. Flash OS patches are run from c:/u/patch directory.
3. The patches which are to be installed on a system are stored on the Software Delivery Card in directory a:/dflt_db/patch.

In c:/u/patch directory, there is a reten directory with reten.pch file. This file contains information about all the patches that are loaded and retained in the system. The option to retained a patch appears during pload. All the retained patches survive a sysload.

To remove the patches available on the opt11c system, rm reten.pch.

Patches are backed up to the z: drive, using EDD in OVL 43.

9.3 Patching Commands

pins <n>	activates patch <n> and puts it in service
plis <n>	gives a detailed information about patch <n>
pload <file.p>	loads a patch file <file.p> from c:/u/patch directory into memory
pnew	creates a memory patch file
poos <n>	deactivated patch <n> and take it out of service
pout <n>	removes patch <n> from memory
pstat <n>	gives summary status of one or all loaded patches

9.4 Patch Creation tools

All the patch make tools are part of the Maestro tools set. Prior to using the patch making tools, the designer must setc <view context> (for Release 24.25 and later) and establish to the required context and machine type. Most commonly used patch make tools are:

- patchmklocal — Create local patch SL-1 local patch.
- patchmksl1 — Create global patch from SL-1 source code.
- patchmkc — Create patch from c source file.
- patchmkmem — Create memory patches
- patchview — Displays patch file information fields and code

Use man patchxxx for help.

9.5 Create & Install a Patch

For a high level C or SL1 patch: this process requires the patchres.sym file. Make sure that correct patchres.sym file is used to build the patch, otherwise the patch cannot be loaded in the system.

For memory patch, the patch can be created directly on the switch.

Step 1. create the patch on the workstation

- modify high level code
- compile (using patchmksl1 -- global SL1 patches only)
- make patch

Step 2. transfer patch file from workstation to a switch

Step 3. put patch into service

- load patch
- insert patch

9.5.1 Local Patches

Example 1: Simple local patch

Using the SET_DIALLED_DN information from the breakpoint section, make a local patch that will print out the message CRDN001 HELP when the last digit is completed dialling and if CR_DIALLED_DN = 0.

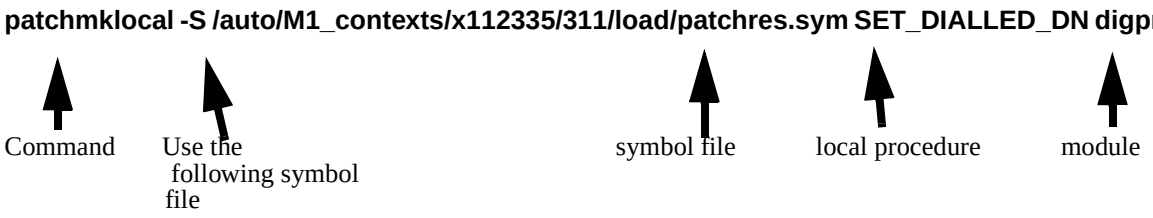
Step 1. create the patch on the workstation

1. Find the high level code:

i) First establish to the 2335 context
ii) Make a local directory for the modified segment (under sl1lib, mkdir 2335)
iii) Find the segment in Xview that has to be modified (in our case digprs20)
iv) Save the file as digprs20.seg into your 2335 directory
v) Include your 2335 directory so that the modified segment will be used
vi) Open the digprs20.seg file and make the changes:

```
AI04.0515 PROCEDURE SET_DIALLED_DN;
AI04.0516 BEGIN
AI04.0517     INTEGER SAVE_DN;
AI04.0518
AI04.0519     % do not overwrite cr_dialled_dn if it already
AI04.0520     % contains something
AI04.0521     IF (CR_DIALLED_DN:CRPTR ^= 0) THEN
AI04.0522     BEGIN
AI04.0523       RETURN;
AI04.0524     END
AI04.0525
                  IF CR_DIALLED_DN:CRPTR = 0 THEN
                  BEGIN
                      IF GET_PRINT_REG(MAINOP, 'CR','DN', 1) THEN
                      BEGIN
                          STORE_PARM(.PRT_CHAR, 'H');
                          STORE_PARM(.PRT_CHAR, 'E');
                          STORE_PARM(.PRT_CHAR, 'L');
                          STORE_PARM(.PRT_CHAR, 'P');
                      END
                  END
AI04.0526     % for RPA call, do not store the RPA trunk access
```

2. Make a patch



The patchmklocal tool compiles the necessary file and makes the patch file.

patchmklocal -S /auto/M1_contexts/x112335/311/load/patchres.sym SET_DIALLED_DN digpr
Patch base: /auto/M1_contexts/x112335/311/load/digpr311.o
**** Compiling the 311 machine
**** Compiling digprxxx
MSL-1 Compiler - Version 5.8
Obtain of digprxxx.msource for machine 311
Obtain complete.
Pass 1 complete.
Pass 2 complete.
Pass 3 complete.
Compilation complete.
**** Compilation of digprxxx complete
Using /auto/M1_contexts/x112335/311/load/patchres.sym
Global: DIGPROC at 0x104cab8e
Verifying SET_DIALLED_DN
Patch constituents:
SET_DIALLED_DN:REST_OF_DNTRANS:DNTRANS:DIGPROC: 133231 L49_215
Assembling patch
Enter patch name: **mypatch**
Enter patch reference number: **bv1**
Enter prs number: **bv0**
Enter engineer's name: **dap**
Created /auto/dpriest/sl1lib/sl1files/311/patch.p

Step 2. Transfer the patch file to Option 11C

Using the rx and umodem commands transfer the patch file from the workstation/PC to Option 11C.

1. On Option 11C, change to the patch directory: cd **c:/u/patch**

2. Assuming that the patch.p file is in your home directory,
dbg> **rx patch.p**

```
Ready to receive...
~C Local command? umodem -sb ~Kirk/sl1lib/sl1files/311/patch.p
      - use full pathname

UMODEM:  File Name: patch.p
Estimated File Size 12 Records, 1408 Bytes

UMODEM:  8-Bit Transmission Enabled
UMODEM:  Ready to SEND File
11

away for 8 seconds
!

total packets                  :  11
number of retries              :   0
receive timeouts               :   1
system errors                  :   0
unknown characters             :   0
transfer cancelled             :   0
packets received out of sequence    :   0
packets with corrupted sequence     :   0
packets failed checksum/crc check   :   0
incomplete packets             :   0
duplicate packets              :   0
```

Step 3. Load the patch

```
dbg> pload patch.p
OR
dbg> pload
Patch filename? patch.p
Retain patch (y/n)? [y]
Days patch vulnerable to sysload? [3]
In-service initialize threshold?  [5]
In-service days to monitor inits? [7]
Loading patch from “c:/u/patch/patch.p”
Patch handle is: 0
dbg> pstat
```

Step 4. Insert the patch

```
dbg> pins 0
Modifying a longword at 0x206b6178 from 0x0 to 0x104ef2d2
SET_DIALLED_DN at 0x104cc86a will be patched to jump to 0x206b6028
Proceed with patch activation (y/n)? [y]
Patch 0 has been activated successfully.
dbg> pstat
```

Step 5. Testing the patch

Now make a phone call and check that the message CRDN001 HELP is output.

Step 6. Remove Patch

```
dbg> poos 0
dbg> pstat
dbg> pout 0
dbg> pstat
```

9.5.2 Global Patches

Step 1. create the patch on the workstation

1. Find the high level code:

i) First establish to the 2335 context
ii) Make a local directory for the modified segment (under sl1lib, mkdir 2335)
iii) Find the segment in Xview that has to be modified (in our case digprs20)
iv) Save the file as digprs20.seg into your 2335 directory
v) Include your 2335 directory so that the modified segment will be used
vi) Open the digprs20.seg file and make the changes:

```
AI04.0515 PROCEDURE SET_DIALLED_DN;
AI04.0516 BEGIN
AI04.0517     INTEGER SAVE_DN;
AI04.0518
AI04.0519     % do not overwrite cr_dialled_dn if it already
AI04.0520     % contains something
AI04.0521     IF (CR_DIALLED_DN:CRPTR ^= 0) THEN
AI04.0522     BEGIN
AI04.0523       RETURN;
AI04.0524     END
AI04.0525
                  IF CR_DIALLED_DN:CRPTR = 0 THEN
                  BEGIN
                      IF GET_PRINT_REG(MAINOP, 'CR','DN', 1) THEN
                      BEGIN
                          STORE_PARM(.PRT_CHAR, 'H');
                          STORE_PARM(.PRT_CHAR, 'E');
                          STORE_PARM(.PRT_CHAR, 'L');
                          STORE_PARM(.PRT_CHAR, 'P');
                     END
                  END
AI04.0526     % for RPA call, do not store the RPA trunk access
```

2. Compile the module.

```
compsl1 digprxxx
**** Compiling the 311 machine
**** Compiling digprxxx
MSL-1 Compiler - Version 5.8
Obtain of digprxxx.msource for machine 311
Obtain complete.
Pass 1 complete.
Pass 2 complete.
Pass 3 complete.
```

```
Compilation complete.
-n Assembling digpr311.s...
complete.
**** Compilation of digprxxx complete
```

To check if your modified segment has been used just take a look at the digpr311.obaudit file:

```
cat digpr311.obaudit
/auto/M1_contexts/x112335/segments/digprxxx.msource AI03 Wed Nov  5 13:23:03 1997
/auto/M1_contexts/x112335/segments/digprpoo.seg     AW02 Fri May  8 15:49:13 1998
/auto/M1_contexts/x112335/segments/digprse1.seg     BH03 Fri May  8 15:49:23 1998
/auto/M1_contexts/x112335/segments/digprf2.seg      AD01 Thu May 23 06:10:16 1996
/auto/M1_contexts/x112335/segments/digprse2.seg     BB01 Fri May  8 15:49:33 1998
/auto/dpriest/sl1lib/2335/digprs20.seg              AI11 Sat Aug  1 11:55:05 1998
/auto/M1_contexts/x112335/segments/digprf0.seg      AD01 Thu May 23 06:09:53 1996
/auto/M1_contexts/x112335/segments/digprf1.seg      AD01 Thu May 23 06:10:16 1996
/auto/M1_contexts/x112335/segments/digprs25.seg     AG03 Fri May  8 20:33:14 1998
/auto/M1_contexts/x112335/segments/uipethf.seg      AO01 Fri May  8 22:10:48 1998
/auto/M1_contexts/x112335/segments/digprs27.seg     AP07 Fri May  8 20:35:18 1998
/auto/M1_contexts/x112335/segments/digprse3.seg     AE10 Wed Aug 27 06:11:14 1997
/auto/M1_contexts/x112335/segments/digprse4.seg     AT10 Fri May  8 15:50:07 1998
/auto/M1_contexts/x112335/segments/digprf3.seg      AG01 Wed Jan 22 06:10:31 1997
/auto/M1_contexts/x112335/segments/digprse5.seg     AW03 Fri May  8 15:50:21 1998
/auto/M1_contexts/x112335/segments/chekrcap.seg     AF01 Fri May  8 22:13:00 1998
/auto/M1_contexts/x112335/segments/digprse6.seg     AI01 Fri May  8 15:50:35 1998
/auto/M1_contexts/x112335/segments/digprs70.seg     AQ08 Fri May  8 20:52:17 1998
/auto/M1_contexts/x112335/segments/digprs75.seg     BI09 Fri May  8 20:52:29 1998
/auto/M1_contexts/x112335/segments/digprse8.seg     AQ10 Fri May  8 15:50:48 1998
/auto/M1_contexts/x112335/segments/digprsup.seg     AF01 Fri May  8 20:30:44 1998
```

To check where your output file is, digpr311.0:

```
qc
```

Step 2. Make a patch

The digpr311.o file is found in the sl1lib/sl1file/311 directory. This file is used to make the global patch.

```
patchmksl1 ~Kirk/sl1lib/sl1file/311/digpr311.o /auto/M1_contexts/x112335/311/
load/patchres.sym DIGPROC
Using symbol file:  /auto/M1_contexts/x112335/311/load/patchres.sym
Using code file:    /auto/dpriest/sl1lib/2335/311/digpr311.o
Output will go to:  /auto/dpriest/sl1lib/sl1files/311/digpr311.p
```

```
Making patch...
patchMkSL1Patch -p digpr311.o -s /auto/M1_contexts/x112335/311/load/
patchres.sym -o /auto/dpriest/sl1lib/sl1files/311/digpr311.p DIGPROC
patchmark /auto/dpriest/sl1lib/sl1files/311/digpr311.p
*****************************
Enter patch name: patch3
Enter patch reference number:
Enter prs number:
Enter engineer's name: dp
Done.
```

NOTE: MAKE SURE THAT THE CORRECT CODE FILE HAS BEEN USED

Look at the sizes of the patch file. The global patch file size is 149840 bytes, and the local patch file size is 1408 bytes.

Step 3. Load, Insert and test the patch

Same steps 3, 4, 5 for the local patch applies here.

9.5.3 Memory patch

A simple memory patch is the quickest way to debug a problem. This type of patch is implemented directly on the switch.

This method requires processor opcode knowledge.

On the switch, use the command **pnew**:

```
pdt> pnew
Enter new patch filename: /u/patch/mempatch.p
Enter patch name: mem pch example
Enter patch reference number: 12345
Enter prs number: BV54321
Enter engineer name: P. Thompson
Enter patch level (diskos | res): [res] diskos
PATCH ELEMENT #1
Enter start address: 52ab5a0
Enter byte count (1, 2, 4): [2]
ADDRESS  OLD    NEW
52ab5a0: 6004 - ffff
52ab5a2: 4e75 - ffff
52ab5a4: 4e71 - ffff
52ab5a6: 4e56 -
PATCH ELEMENT #2
Enter start address:
Creating patch /u/patch/mempatch.p...
Check sum = 0xa4e
Done.
pdt>
```

9.5.4 C Code Patches

Example:

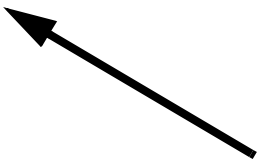
C code patch to output a message after a data dump has backed up its files

The data dump routine backs up certain files to the z: drive on the CPU. Write a patch to change the message that is output from “Internal backup complete” to “I love this course”

Step 1. First, find the function that is responsible for backing up the files to the z: drive.

The file is foxZDrv.c. Make a copy in local directory and change the following function foxMakeBackup.

```
pPart = zdrvBuildPart(ZDRV_CDB_PART, pPart, &custList, sl1Print);
      if (pPart != NULL) {
       sprintf(buffer, “Internal backup complete \n”);
       (*sl1Print)(buffer);
       zdrvListDispose(&sysList);
```



Just change this line
to “I love this course”

Make the patch.

patchmkc res foxZDrv.c /auto/M1_contexts/x112335/311/load/patchres.sym foxMakeBackup

```
Using symbol file:  /auto/M1_contexts/x112335/311/load/patchres.sym
Using C file:       /auto/dpriest/sl1lib/2335/foxZDrv.c
Output will go to:  /auto/dpriest/sl1lib/sl1files/311/foxZDrv.p
Making patch...
compsl1 foxZDrv.c -c -S
**** Compiling the 311 machine
**** Building Include path list
**** Compiling foxZDrv
**** Compilation of foxZDrv complete
/home/vw/gnu223/solaris.68k/bin/as68k-gnu137 foxZDrv.s -o tmp1foxZDrv.o
patchFixDotSFile -p tmp1foxZDrv.o -a foxZDrv.s -o newfoxZDrv.s  foxMakeBackup
*******************************************************

Enter function name or . to end .
/home/vw/gnu223/solaris.68k/bin/as68k-gnu137 newfoxZDrv.s -o tmp2foxZDrv.o
```

```
patchPruneSym -p tmp2foxZDrv.o -t tmp3foxZDrv.o -s zzztmp.sym -m foxZDrv.c -n
new.sym -i foxZDrv.pinfo
/home/vw/gnu223/solaris.68k/bin/ld68k -R new.sym -N -Ttext 0 -o tmp4foxZDrv.o
tmp3foxZDrv.o
patchMkCPatch -p tmp4foxZDrv.o -i foxZDrv.pinfo -l res -o /auto/dpriest/sl1lib/sl1files/311/
foxZDrv.p
patchmark /auto/dpriest/sl1lib/sl1files/311/foxZDrv.p
****************************
Enter patch name: patch 4
Enter patch reference number:
Enter prs number:
Enter engineer's name:
Done.
```

Step 2. Transfer the patch file to a switch.

Step 3. Load the patch.

Step 4. Insert the patch

Step 5. Test the patch

9.5.4.1 C code patching tricks

1. If a FOREVER loop exists anywhere within the module creating a patch is not allowed.

   A FOREVER loop is of the form:

   ```
   while ( TRUE )
   {
   }

   for(;;)
   {
   }

   while (1)
   {
   }
   ```

   To workaround the forever loop problem; do the following:

   ```
   int proc ()
   {

    ....
    ....
    int forever = 1;


    while ( forever )
    {
    }
   }
   ```

2. Embedded assembly code within a C file may cause problems when applying a patch to a procedure within the file. The best work around is to remove the offending assembly code by using the IFDEF statement. This only works if the assembly code is not in the function being patched.

3. When doing a C++ patch you must not specify the function name. This may cause a problem. When prompted to Enter the function name, enter a portion of the function name you wish to patch. e.g. processDuSignal for rxRingBuf::processDuSignal. The C++ filter routine will search the C++ file for the list of functions that have the same root name as the specified one. You will be prompted for y/n to each of the procedure with the similar name. Answer y to the one you want to patch and n for others.

patchmkc diskos mmihCard.cc patchres.sym

```
Trying the vxm1 patcher ...
Cannot find symbol file "patchres.sym"
```
patchmkc diskos mmihCard.cc /auto/opt11c/2422c/patchres.sym
```
Trying the vxm1 patcher ...
Using symbol file:  /auto/opt11c/2422c/patchres.sym
Using C++ file:       /auto/M1_contexts/x112422c/segments/mmihCard.cc
Output will go to:  /auto/amesser/sl1lib/sl1files/311/mmihCard.p
Making patch...
patchcompsl1 -ks mmihCard.cc -c -S
**** Compiling the 311 machine
C++ file mmihCard.cc not found in current directory searching for it !!
**** Building Include path list
**** Compiling mmihCard
/home/mtv3rd01/CenterLine/solaris/2.1.1/CenterLine/bin/../clc++/sparc-solaris2//
clpp  -C -lang-c++ -DCENTERLINE_CLPP=1 -Amachine(sparc)  -I- -Usun -I/auto/
M1_contexts/x112422c/pools -I/auto/M1_contexts/x112422c/segments -
D_NO_LONGLONG -D__STDC__=0 "$D" "$Dx" "$Dxx" "$Dxxx" "$Dxxxx"
"$Dxxxxx" "$Dxxxxxx" "$Dxxxxxxx" "$Dxxxxxxxx" "$Dxxxxxxxxx" "$Dxxxxxxxxxx"
"$Dxxxxxxxxxxx" "$Dxxxxxxxxxxxx" "$Dxxxxxxxxxxxxx" "$Dxxxxxxxxxxxxxx" -
DCENTERLINE_ANSI_CPP -Dc_plusplus=1 -D__cplusplus=1    -I/home/mtv3rd01/
CenterLine/solaris/2.1.1/CenterLine/bin/../clc++/sparc-solaris2/incl /auto/
M1_contexts/x112422c/segments/mmihCard.cc > /usr/tmp/CC.14953/cpptmp
/home/mtv3rd01/CenterLine/solaris/2.1.1/CenterLine/bin/../clc++/sparc-solaris2//
cfront.sl1  +f/auto/M1_contexts/x112422c/segments/mmihCard.cc +T/usr/tmp/
CC.14953/dirinst +s +t/usr/tmp/CC.14953/ptcf  +C +V +x/home/vw/rel511/bin/
solaris/szal.MC68040gnu +a1 +i/usr/tmp/CC.14953/cpptmp > /usr/tmp/CC.14953/
mmihCard.c
/home/vw/gnu223/solaris.68k/bin/cc68k    -S  -g -DVXWORKS -
D__PROTOTYPE_5_0 -D_MMU -DVXW -DCPU_FAMILY=MC680X0 -
DCPU=MC68040 -DHOST_SUN -DGAMMA_311 -DC_SSERIES -DRWDEBUG=1 -
DRW_MULTI_THREAD -D_REENTRANT -DWRITE_PROTECT_FLASH -Dsigned=
-Usun /auto/amesser/sl1lib/sl1files/311/mmihCard.o -m68040 -ffixed-d2 -ffixed-a5 -
nostdinc -fno-builtin -msoft-float -fvolatile -W -Wimplicit -Wreturn-type -Wswitch -
Wcomment -Wformat -Wchar-subscripts -Wuninitialized -Wparentheses -Dsigned= -
ansi -pedantic -O -v mmihCard.c
/home/vw/gnu223/solaris.68k/bin/cc68k    -S  -g -DVXWORKS -
D__PROTOTYPE_5_0 -D_MMU -DVXW -DCPU_FAMILY=MC680X0 -
DCPU=MC68040 -DHOST_SUN -DGAMMA_311 -DC_SSERIES -DRWDEBUG=1 -
DRW_MULTI_THREAD -D_REENTRANT -DWRITE_PROTECT_FLASH -Dsigned=
-Usun /auto/amesser/sl1lib/sl1files/311/mmihCard.o -m68040 -ffixed-d2 -ffixed-a5 -
nostdinc -fno-builtin -msoft-float -fvolatile -W -Wimplicit -Wreturn-type -Wswitch -
Wcomment -Wformat -Wchar-subscripts -Wuninitialized -Wparentheses -Dsigned= -
ansi -pedantic -O -v mmihCard.s
**** Compilation of mmihCard complete
```

```
/home/vw/gnu223/solaris.68k/bin/as68k-gnu137 mmihCard.s -o tmp1mmihCard.o
patchFixDotSFile -p tmp1mmihCard.o -a mmihCard.s -o newmmihCard.s -i
*********************************************************
processDuSignal
Enter function name or . to end .
Searching for matching names... This will take a few seconds
Patch 'rxRingBuf::processDuSignal(void)'? y
/home/vw/gnu223/solaris.68k/bin/as68k-gnu137 newmmihCard.s -o
tmp2mmihCard.o
patchPruneSym -p tmp2mmihCard.o -t tmp3mmihCard.o -s zzztmp.sym -m
mmihCard.cc -n new.sym -i mmihCard.pinfo
/home/vw/gnu223/solaris.68k/bin/ld68k -R new.sym -N -Ttext 0 -o tmp4mmihCard.o
tmp3mmihCard.o
patchMkCPatch -p tmp4mmihCard.o -i mmihCard.pinfo -l diskos -o /auto/amesser/
sl1lib/sl1files/311/mmihCard.p
patchmark /auto/amesser/sl1lib/sl1files/311/mmihCard.p
*****************************
Enter patch name:
Enter patch reference number:
Enter prs number:
Enter engineer's name:
```

Debug in patch:
For a local patch example:
lkup SET_DIALLED_DN
=> PATCH0_SET_DIALLED_DN
Likewise, can use other debug commands; l, b, ...

9.6 Exercises: Patching

1. In overlay 10, output “WHY” just after the input for the customer number is entered.

    - Using Xview, locate the code you want to modify. What is the name of the procedure?

      _____

    - Determine the segment name of the file which you want to modify. What is the name of the segment?

      _____

    - Copy the file to your home directory.

    - Make changes to the code.

    - Make a local patch, and test the patch. Which command did you use to make the patch? (may need to generate symbol file)

      _____

    - Make a global patch, and test the patch. Which command did you use to make the patch?

      _____

2. Implement the C code patch from section 9.5.4.

9.7 Product Enhancement Package (PEP)

PEP is used to track all the patches that are installed in the customer switches. When a designer develops a patch and the patch is installed and in service in a customer switch, all the information pertaining to the patch must be entered in PEP.

The PEP tool is maintained by the GCCS group.

To start the application:

- from the Root Menu, under Meridian-1 Tools, select Patch; or
- by typing Patch in a xterm window; or
- http://sulaco.europe.nortel.com:8080/mpl/index.cfm

10.0 Setting Package Restriction Bits

SERV_PACK_RESTRICT is an array of bits which is used to restrict/enable software packages. The designers can turn these bits on/off in the memory.

To turn a package off: eg. .UK_PACKAGE (pkg 190) on x112337 (511)

a) Check status of this package.
```
>ld 22
REQ  prt
TYPE pkg 190
UK     190
```

b) In Xview for this release and machine type, find some code that makes reference to the package.
```
IF ^SERV_PACK_REST[.UK_PACKAGE] THEN
```

c) Look at the corresponding assembler code.
```
bfextu a5@(0x24868){#17:#1},do
```

d) A5 contains the real address of the beginning of the SL1 address space, ie. we can access the offset using a virtual address. Divide offset (real address) by 4 to get virtual address.
```
0x24868 / 4 = 0x921A
```

e) In PDT, print the contents of this address.
```
pdt> p 921a
0000921A : 00003143
```

f) The Bit Field Extraction assembler instruction makes reference to bit 17 (for a length of 1 bit).
```
H.00003143 = B.0000 0000 0000 0000 0011 0001 0100 0011
```

g) Change this bit to a 1 to restrict the package.
```
B.0000 0000 0000 0000 0111 0001 0100 0011 = H.00007143
```

h) Write the new word to memory.
```
pdt> w 921a
0000921A : 00003143 /7143
```

i) Check status of this package again.
```
>ld 22
REQ prt
TYPE pkg 190
UK   PKG 190 IS RESTRICTED
```

This is the last page of this document.