Using the Debugger

What to do when you crash, hang, or Loop

When a program crashes in the office system, and the release has the debugger, you end up in the debugger. You can then poke around for a while, but eventually you will want to get on to other things. To get out of the debugger, you need to know a few things. The register display, on the right of the third line has a piece that says DO=0 (or 1,2,or 3). The DO stands for domain, and if the domain is nonzero and it does not say overridden to 0, then to resume you should type the debugger command G. This is the typical crash found in the office system, and using the G command forces the process into the terminate exception handler, and things can be put away neatly. If you are in domain 0, or overridden to zero, you should use the OSQUIT command.

If you are stuck and nothing is happening in response to power offs, key input or mouse clicks, you are either looping or are hung. In either case you want to hit NMI. If the display is not in domain 0, you are probably looping. To kill the process, you can type G 0, or PC 0 followed by G. This sets the program counter to 0 and tries to access location 0 which is illegal and causes a bus error. Typing G after this bus error will terminate the process neatly.

If you are in domain 0 and you are sitting on an RTS instruction, type id PC-4. If the result is a STOP instruction, then you may be hung. You should first make sure that you are not doing any i/o. Type G to continue and watch the profile lights and listen for diskette i/o. If i/o is in progress, you can wait for the i/o to complete, or you can follow instructions of looping which follows. If however, no i/o is in progress, and when you hit NMI youf are still on and RTS instruction and the STOP instruction preceeds the RTS, type OSQUIT to clean up the os and file structures.

If you are in domain 0 and are not on an RTS instruction, you should type G and then NMI again. Eventually you should get out of domain 0 or get to the STOP instruction. You can also use the USR command as described in the breakpoints section. If you cannot get out of domain 0, Type OSQUIT to clean up.

The ground rules are do everything you can to terminate processes normally. If you blow up in an application, type g to terminate cleanly. After looping, type pc 0 ; G to again terminate the process cleanly. Use OSQUIT as a last resort, and that means only in domain 0. You should never have to reset the machine using the reset button on the back of the machine.

The PU/PL dump

Frequently, a bug report will come with a three page printout that was made with the PU or PL debugger command. This command generates output similar to pages 1,2, and 3. The first page consists of a screen dump of the primary screen, the second page contains the screen dump of the alternate screen, and the third page has some additional stack crawl and memory locations displayed.

There is a wealth of information provided in these three pages. The first page gives us a big hint; some item from the arrangement menu was being executed. The second page gives us additional information. A bus error was detected and the access address is 0. This is a big clue because nil pointers are 0 and generate a bus error if you try to access location 0. Also included on page 2 is a register display, and the most interesting piece of information is that the program counter (PC) was at SUBFMCLS+94 at the time of the bus error. Note that the first line of the register display is Level 7 interrupt. This is basically a worthless piece of information, as far as applications are concerned. This is because NMI, address errors, and bus errors always show level 7 interrupt.

The third page of the dump gives us four distint groupings of information. The first is a register display, then a stack crawl, then a disassembly of the instructions surrounding the PC, and finally a portion of the stack is displayed. Using these pieces of information we can determine what went wrong.

To find out what the processor is objecting to, we start by looking at SUBFMCLS+94, the location which is at the top of the register display. Looking at the disassembly (marked 5 on page 3) we see instructions at SUBFMCLS+92 and at +96 but not at +94. Actually, it turns out that the PC leads (has already advanced past) the instruction being executed. This time the pc leads by 4, and the instruction being executed is at +90. There we see a MOVE.L (A0),(A1). This is marked 6 on page 3. Looking back to the register display, we can see that A0 looks Ok but that A1 is 0. It is the reference via A1 which caused the bus error.

There is also some other handy information on page 3. Register A6 points to the stack frame (marked 1 on page 3). Matching the address contained in A6 with the stack display, we can find the parameters to SUBFMCLS. The address is marked 2. The first 2 words at that address link to the calling stack frame and the return pc for the calling procedure. Following that are the parameters in REVERSE order (marked 3). See the section on Parameters for more details.

One final note on the using the PU and PL commands. These commands use a. Parallel printer connected to Slot2chan2 or Slot2Chan1 respectively. They do not work with serial printers. The commands should be used immediately following an occurrance of a bug so the error display is preserved. If you do a stack crawl and the call is pretty deep, the stack crawl can wipe out the error display, making the information on the alternate screen less valuable.

Finding out what parameters are passed and returned

Page 4 shows a more dynamic tracing of the same bus error. The first command used is the display memory command. Its arguments request using A6 indirectly to display 40 hex words. The next command TD gives the register display. The SC (stack crawl) command gives the trace back of who call whom.

So let's find out what parameters were passed to GEMenuCmd. This routine has the calling sequence written in to the right of the stack crawl command. To find the parameters we find GEMENUCM in the stack crawl display, and look down one line to find out the stack frame. The stack frame is at F7BE88. This part of memory was then displayed using the dm command. The first word contains F7C21E which is the stack frame pointer for GeMenuEvent. The next word is an address, and using the CV (convert) command shown at the bottom of the page, we see that this is the address of GeMenuEvent+492 which is the instruction in GeMenuEvent immediately following the call to GEMenuCmd.

Following the return PC, the stack has 0007 and 0006. The parameters are in reverse order so item is 7 and menu is 6.

This example shows a very simple case, one where two integers were passed by value. Now we'll do a more complicated example. Page 5 shows the calling sequence for the Select routine in the Field Editor. First a breakpoint was set at select+8 and then the register display is shown when the breakpoint was hit. (See the breakpoint section for more info on breakpoints). Once inside the Select routine (and past the Link instruction -- more on this in breakpoints), we proceed to display memory pointed to by A6. Remembering to skip the stack frame pointer and the return pc, the next word is the LAST parameter to Select, and it is F7F32E. This is an address because it is a var parameter -- so F7F32E is a pointer to t. Continuing, F7EE32 is a pointer to n; D60552 is a handle to a field state; D6054E is a handle to the field; The point consists of the next two integers 85 and 141.

Now let's assume we want to look at the field, and specifically, what the value of the field is currently. To do this, we have the handle to the field, and the record declaration of the field. We can use the dm command to look at D6054E and then access the first longint there D623d4 to get to the field, or we can use the shorthand dm (0D6054E) to get in one step to the field. The () means "indirect".

Examining the field, the coords rectangle is the first 4 words; maxlen is 8; growlen is 8; curlen is 1; align is 3, drawpad is 4 -- both packed into one integer; curvalue is E20802. Now we access curvalue to get the contents of the array. Looking at the display, the first byte is a lowercase g. We know that since curlen is 1 that is all the field contains.

There are a couple of other observations we can make. We can examine where these heaps map to data segments. Looking at the curvalue array we know that it is pointed to by the handle E20802. Knowing that the master pointer and the handle are in the same segment, using the first byte of the address, we can calculate the MMU number. E2/2 gives &113 which corresponds to LDSN 7. (LDSN 1 starts at 107, LDSN 2 is at 108 ...). Doing a Stop-Start calculation we see that the segment is 8K long and the handle, at 802 is at 2K into the segment, a valid address.

Note that 2 heaps (and segments) are being used here by the graphics editor. The field data structure is in one heap D6xxxx addresses and the other is used for the data components of the field and have addresses of E2xxxx. This is not the usual way of using fields and heaps, but see what you can figure out using the debugger!

Breakpoints

Breakpoints when debugging applications are useful when in the application's domain. This is noted on the register display. Note that domain 0 or another domain overridden to 0, are not application domains, and you cannot set breakpoints in the application there. There is one special case. When in the application process, but in domain 0 (the case indicated with the trace on page 6) you can use the ubr (user break) command. This sets a breakpoint at the first instruction in the user domain, and starts executing. In the case on page 6, the breakpoint is reached at LetOthersRun+34. From this location you are in a user domain (domain 3) and in your process (process id 6) and can set breakpoints. I did a stack crawl to show that the application symbols are available at this point. Next I did a cl pc to clear the breakpoint where I am currently stopped.

There are a few rules to remember to follow when setting breakpoints. First you should never set breakpoints on IUJSR or the future ILJSR instructions (or any other IUxxx or ILxxx instructions). However, you can trace through them if you don't mind seeing all the code for the trap handlers. They do not work, and will give unpredictable results. Many people have wasted hours of time because of this. The second rule is it is frequently desireable to set breakpoints after the LINK and befor the UNLK instructions. Page 7 shows why. After the first register display, two breakpoints were set, one at GEMenuCmd, and one at GEMenuCmd+8. I then ran until I reached the first breakpoint. Then I did a stack crawl and displayed the stack frame. Then I ran again, stopping after the Link instruction is executed. Then I did a stack crawl and a display of the stack frame again. Note that they are very different, and the one at +8 gives correct results. I generally set breakpoints at the Procedure+8. Note that this only works for code generated with a TST.W instruction before the UNLK (the usual case, but is not guaranteed).

To set breakpoints at the end of the procedure, you will have to use the IL command to find the end of the procedure. You can usually spot this because UNLK...RTS sequence followed by the procedure name dropped in the code. An example of the end of a procedure is shown on page 4. You can even see the procedure name, although the first character is not visable because the high bit is set to indicate to the debugger that this is a Pascal procedure. Setting a breakpoint just before executing the UNLK instruction will permit you to examine the var parameters that are being returned in exactly the way the input parameters were determined. However, to use this technique, beware of nested procedures and global gotos.

So far we have always used symbols for setting breakpoints. Sometimes it is not always possible. Sometimes the code is swapped out and the debugger cannot find the symbols, or the code was compiled without symbols. Then you will have to use a logical address to set the breakpoint. The usual way of finding out the address is to find the IUJSR call to the routine and break on the target address. Another technique suggested by Chris Moeller, is to first let the program fail, then do a cv on the symbolic name, and then rerun the program setting the breakpoint at the logical address.

To set breakpoints when the program is coming up, you have to use a few tricks. First, you'll run the office system under the OS shell (or workshop shell if you have compatible libraries). Then you use the Debug command, and respond shell.office sytem for the program, and yes for the question to debug all sons. Then each process launch will give you an opportunity to set breakpoints. These breakpoints may have to be logical addresses because the probability of the code being in memory is very low (unless the OS has left the program loaded). Note that this technique of remembering logical address across process executions only works for the exact same program. Relinking the program will invalidate the logical address assignments and you will have to let it break first, find out the logical address, and then rerun and set the breakpoints.

An alternative suggest by Rod Perkins is to bring up the filer, hit NMI opportunely in domain 0, then set a breakpoint on 0:Declare_Excep_Hdl. When it stops at the breakpoint (in domain 0), issue cl pc to clear the breakpoint, Then issue the UBR command. It will then break in your application.

Local and Global Variables

It is frequently useful to be able to trace through a routine and determine what the value of some variable is. To do this, you need to understand the layout of the stack. Page 8 shows a diagram of the stack. Note that in this diagram, the addresses go from low to high. Global variables are accessed by adding negative numbers to A5, and local variables are accessed by adding negative numbers to

A6. Intrinsic unit globals are accessed by first adding positive numbers to A5 to get to the data pointer table entry, and then taking the value found there and adding negative numbers to that.

To show how you can figure out values of local and global variables while stepping through a procedure, I picked out a very small procedure. Its source listing is on page 9. Page 10 contains the disassembly of the procedure. The process of determining where a variable is in memory requires some matching of the source with the code generated. What I usually do is use the IUJSR instructions to determine rough areas of code and then look in more detail at the generated code from there.

Page 10 also sets a breakpoint at copysel+8 and runs until the breakpoint was hit; then the stack frame was printed out. Note that the CutCopyField procedure is not in memory. We can tell that by the fact that at CopySel+5C there is an IUJSR to $8E000E instead of CutCopyS.

Looking on page 11, we see that an ID of $8E000E gives the invalid logical address message. To illustrate setting a breakpoint on a logical address, I set a breakpoint at $8E000E. Also a breakpoint was set at CopySel+14.

At the CopySel+14 breakpoint, we are about to compare TypeofSel with aCellTxTS1. TypeofSel is a Variable of an enumerated type, and aCellTxTS1 is one of the values. Its value is 1. So displaying ra4+$ffffffc9 displays the value of TypeofSel. Note that the instruction is CMPI.B #$0001,$ffc9(a4). The dm command uses $ffffffc9 because we want to maintain the fact that it is a negative quantity. RA4+$ffffffc9 yields 0f7cf49, an odd address. Note that the debugger, however starts the display at 0f7fcf48, so the byte we are testing is the rightmost byte of the first word. Note also that the access is relative to A4, but that A4 was loaded relative to A5. This is because this is a global variable in an intrinsic unit, and A4 contains the pointer to the base of the globals for this unit.

At CopySel+24 we access another intrinsic unit global, this time it is tblpars.editcoltitle. It is again at an odd address. The variable is a boolean, and hence its value is true.

At CopySel+3E we are pushing a parameter to SetPnlPort. It again is an intrinsic unit global. The parameter is an integer, and displaying the value shows it to be 3. Next, the trace command was used to step to the next instruction. Note that the value of A7 has changed, and that A7 points to the value just pushed on the stack.

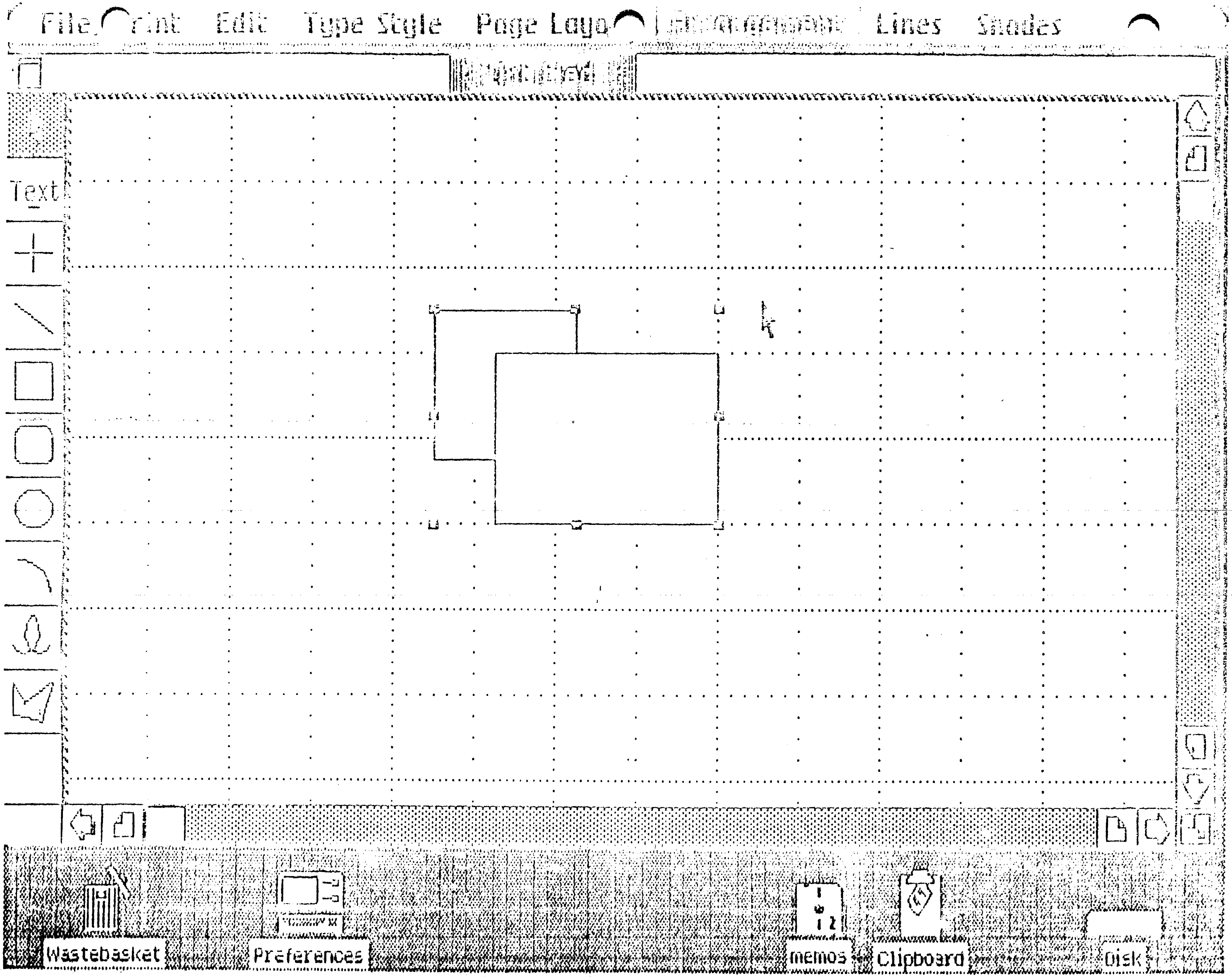
On page 12, we are pushing the effective address of a local variable, errnum. Note that the reference is relative to A6. When the value is displayed, its value is 3F52 (garbage since its value is set by the routine).

Continuing on, we hit the breakpoint at 8E000E. This is a digression from the flow of finding out the value returned from CutCopyField, so I'll just show how you can get into CutCopyField and get out. This address where we stopped is actually a jump table entry, so we trace through the instruction and get to CutCopyField. (A jumpt table is used when calling from one segment to another). After a few more traces to get past the LINK instruction, we check the address of the last parameter passed to CutCopyField, and it is indeed the address of errnum we found before. Next, a breakpoint was set to the return PC.

After continuing, we break in CopySel immediately after the return from CutCopyField. Displaying the location containing errnum, we see CutCopyField returned 0000.

Function returns

It is frequently useful to determine what a function returns. To do this break at the instruction immediately following the JSR or IUJSR to the function. Then the function return is on the top of the stack. DM ra7 will display the returned value.


File/Print
Edit
Type Style
Page Layo
Arrangement
Lines
Shades
Text
Wastebasket
Preferences
memos
Clipboard
Disk

```
Level 7 Interrupt
SUBFHOLS+0094 0000 5340                        ORI.B   #$5340,A0
PC=00281A32 SR=0000  0  US=00F7BDCC SS=00CBFED8 DO=1 P#=00007
D0=00000000 D1=00000100 D2=0000FFCE D3=00D007E4
D4=CC280805 D5=00145700 D6=0000000A D7=00DA0AB8
A0=00DA0AD8 A1=00000000 A2=00CE004C A3=00F7F466
A4=00F7F466 A5=00F7F4A6 A6=00F7BDD8 A7=00F7BDCC
>pu
```

```
BUS ERROR in process of gid       7
Process is about to be terminated.
access address =          0 = mmu#         0  ,   offset         0
 inst reg =     8840 sr =         0  pc = 2628146
 saved registers at 13369270
Going to Lisabug, type g to continue.
```

2

3

```
Level 7 Interrupt
SUBFMOLS+0094 0003 5340       PC        ORI.B   #$5340,A0
PC=00231A32 SR=0300  0  US=00F73DCC SS=00C3FED8 DO=! PM=00007
D0=00000000 D1=00000100 D2=0000FFDE D3=000007E4
D4=00C23006 D5=00145700 D6=0000000A D7=000A0A88
A0=000A0A88 A1=00000000 A2=00C3004C A3=00F7F466
A4=00F7F466 A5=00F7F4A6 A6=00F73DD8 A7=00F73DCC
                                  1
At SUBFMOLS+0094
Stack frame at 00F73DD8 called from COMMITLA+033A
Stack frame at 00F73E14 called from LOCKCMD+007A
Stack frame at 00F73E2E called from GEMENUCH+02A0
Stack frame at 00F73E38 called from GEMENUEV+048E
Stack frame at 00F7C21E called from PROCESST+011E
Stack frame at 00F7C258 called from MAINPROG+008A
Stack frame at 00F7C298 called from GRAPHICS+001E
Stack frame at 00F7F4A6
SUBFMOLS+0074 2968 0004 0004          MOVE.L  $0004(A0),$0004(A4)
SUBFMOLS+007A 6016                    BRA.S   *+$0018          ; 00231A90
SUBFMOLS+007C 2047                    MOVE.L  D7,A0
SUBFMOLS+007E 2247                    MOVE.L  D7,A1
SUBFMOLS+0080 2251                    MOVE.L  (A1),A1
SUBFMOLS+0082 2368 0004 0004          MOVE.L  $0004(A0),$0004(A1)
SUBFMOLS+0088 2047                    MOVE.L  D7,A0
SUBFMOLS+008A 2247                    MOVE.L  D7,A1
SUBFMOLS+008C 2269 0004               MOVE.L  $0004(A1),A1
SUBFMOLS+0090 2290                    MOVE.L  (A0),(A1) 6
SUBFMOLS+0092 302C 0008               MOVE.W  $0008(A4),D0
SUBFMOLS+0096 5340                    SUBQ.W  #$1,D0
SUBFMOLS+0098 3940 0008               MOVE.W  D0,$0008(A4)
SUBFMOLS+009C 4267                    CLR.W   -(A7)
SUBFMOLS+009E 2F07                    MOVE.L  D7,-(A7)
SUBFMOLS+00A0 4EBA F10A               JSR     CNTOFOBJ         ; 0023034A
SUBFMOLS+00A4 302C 000A               MOVE.W  $000A(A4),D0
SUBFMOLS+00A8 905F                    SUB.W   (A7)+,D0
SUBFMOLS+00AA 3940 000A               MOVE.W  D0,$000A(A4)
SUBFMOLS+00AE 42A7                    CLR.L   -(A7)
00F73DB8        00F7 3E02 002A 353C  00F7 3DEC 00F7 E338 .....*5<........
00F73DC8        000A 0A6C 0000 0001  000A 0A64 00F8 0436 ...l........d....
00F73DD8        00F7 3E14 002A 2764 3 000A 0A88 00F7 F466 .....*'d.......f
00F73DE8        0000 000A 0000 0197  000A 0A64 00F8 0496 ............d....
00F73DF8        0002 0038 5FC2 000A  0A88 00F7 3E12 002A ...._...........*
00F73E08        35A8 0000 0000 0000  0000 00F7 00F7 3E2E 5...............
00F73E18        0062 0038 0014 5700  0000 0001 0000 0007 .b....W.........
00F73E28        3E30 0036 3316 00F7  3E38 0064 1400 016E .0.63......d...n
```

PARAMETERS

4

```
>dm ra6 40
00F7BDD8        00F7 BE14 002A 2764  000A 0A88 00F7 F466 .....*'d.......f
00F7BDE8        0000 000A 0000 0197  000A 0A64 00F8 0486 ...........d....
00F7BDF8        0002 0038 5F02 000A  0A88 00F7 BE12 002A ...._..........*
00F7BE08        03A8 0000 0000 0000  0000 00F7 00F7 BE2E 5...............
>td
SUBFMOLS+0094 0008 5340       PC       ORI.B   #$5340,A0
PC=00281A32 SR=0000  0  US=00F7BDCC SS=00C3FED8 DO=1 PN=00007
D0=00000000 D1=00000100 D2=0000FFCE D3=000007E4
D4=0C280005 D5=00145700 D6=0000000A D7=000A0A38
A0=000A0A38 A1=00000000 A2=00CE004C A3=00F7F466
A4=00F7F466 A5=00F7F4A6 A6=00F7BDD8 A7=00F7BDCC
>sc
At SUBFMOLS+0094
Stack frame at 00F7BDD8 called from COMMITLA+033A
Stack frame at 00F7BE14 called from LOCKCMD+007A
Stack frame at 00F7BE2E called from GEMENUCM+02A0
Stack frame at 00F7BE88 called from GEMENUEV+048E
Stack frame at 00F7C21E called from PROCESST+011E
Stack frame at 00F7C258 called from MAINPROG+008A
Stack frame at 00F7C298 called from GRAPHICS+001E
Stack frame at 00F7F4A6
>dm 0f7be88 30
00F7BE88        00F7 C21E 0064 18DC  0007 0006 0C28 0002 .....d.......(..
00F7BE98        0014 5700 2F00 4267  2F2E FFD4 201F 0A01 ..W./.Bg/.......
00F7BEA8        00F8 0486 0001 1453  6574 2041 7369 6465 .......Set.Aside
>il 6418dc-20
GEMENUEV+0472 FFDC                         $$$$
GEMENUEV+0474 486E FFD2                    PEA      $FFD2(A6)
GEMENUEV+0478 486E FFD4                    PEA      $FFD4(A6)
GEMENUEV+047C A088 0234                    IUJSR    MENUSELE         ; 008861AC
GEMENUEV+0480 4A6E FFD4                    TST.W    $FFD4(A6)
GEMENUEV+0484 670C                         BEQ.S    *+$000E          ; 006418DC
GEMENUEV+0486 3F2E FFD2                    MOVE.W   $FFD2(A6),-(A7)
GEMENUEV+048A 3F2E FFD4                    MOVE.W   $FFD4(A6),-(A7)
GEMENUEV+048E 4EBA F332                    JSR      GEMENUCM         ; 0064115C
GEMENUEV+0492 4267                         CLR.W    -(A7)
GEMENUEV+0494 A088 022A                    IUJSR    HILITEME         ; 00885C18
GEMENUEV+0498 4CDF 18F0                    MOVEM.L (A7)+,D4-D7/A3/A4
GEMENUEV+049C 4E5E                         UNLK     A6
GEMENUEV+049E 2E9F                         MOVE.L   (A7)+,(A7)
GEMENUEV+04A0 4E75                         RTS
GEMENUEV+04A2 C745 4D45 4E55 4556  0060 2000 0000 0000 .EMENUEV.`......
GEMENUEV+04B2 0000 0000 0000 0000  0000 0000 0000 0000 ................
GEMENUEV+04C2 0000 0000 0000 0000  0000 0000 0000 0000 ................
GEMENUEV+04D2 0000 0000 0000 0000  0000 0000 0000 0000 ................
GEMENUEV+04E2 0000 0000 0000 0000  0000 0000 0000 0000 ................

>cv 6418dc
6418DC=6559964=GEMENUEV+0472
>pr 0
```

GemenuCmd (menu, item: Integer);

PARAMETERS

procedure Select (dxy:Point; hf:hndField; hfs:hndFState; var r:Rect;
var t:integer);

```
)br select+8
)g

Break Point
SELECT+0008  *48E7 0118                    MOVEM.L D7/A3/A4,-(A7)
PC=008C3FA8 SR=0000  0  US=00F7C010 SS=00C00000 DC=1 PM=00008
D0=00020001 D1=00E20000 D2=00000002 D3=001FFFFF
D4=0010FFFA D5=00000001 D6=FFFC3900 D7=0007FFFE
A0=00F30436 A1=00F7F32E A2=00CE004C A3=70061080
A4=00F30436 A5=00F7F4A6 A6=00F7C018 A7=00F7C010
)dm ra6 40
00F7C018      00F7 C034 002A 080E  00F7 F32E 00F7 EE32 ...4.*.........2
00F7C028      00D6 0552 00D6 054E  0085 0141 00F7 C07E ...R...N...A...~
00F7C038      0062 1884 0062 0085  0141 00DA 0896 397C .b...b...A....9|
00F7C048      0010 FFFC 397C 0007  FFFE 7006 1030 00F8 ....9|....p.....
)dm 0d6054e
00D6054E      00D6 23D4 00D6 23AE  00D6 2374 00D6 237A ..#...#...#...#z
)dm 0d623d4 40
00D623D4      007E 0133 008A 014D  0003 0003 0001 0304 .~.;...M........
00D623E4      00E2 0802 0001 0001  0001 00E2 0806 002E ................
00D623F4      4012 054A 001E FFFD  002A 0025 0008 0003 @..J.....*.%....
00D62404      0004 0304 00D6 053A  0001 0001 0001 0006 .......:........
)dm (0d6054e) 40
00D623D4      007E 0133 008A 014D  0003 0003 0001 0304 .~.;...M........
00D623E4      00E2 0802 0001 0001  0001 00E2 0806 002E ................
00D623F4      4012 054A 001E FFFD  002A 0025 0008 0003 @..J.....*.%....
00D62404      0004 0304 00D6 053A  0001 0001 0001 0006 .......:........
)  k (0e20802) 10
00E20832      67E2 1FF4 00E2 1FF4  0000 0306 00E2 0846 g..............F
)cv e2/2
$71=&113=00000071
)mm &113
D[1] Segment[71] Origin[650] Limit[F0] Control[7] Start[0C8000] Stop[0CD7FF]
)cv 0cd7ff-0cb800
$1FFF=&8191=00001FFF
)cv 802
$0802=&2050=00000802
```

```
field = record                              { static field characteristics  }
    coords:      Rect;                      { bounding rectangle            }
    maxLen:      integer; 492               { maximum number of chars       }
                                            {     (should equal size of     }
                                            {     curvalue array)           }
    growLen:     integer; 62                { size by which to grow value   }
                                            {     array - don't grow if 0   }
    curLen:      integer; 467               { current number of chars       }
    align:       byte;                      { alignment of chars when field }
                                            {     is displayed              }
    drawPad:     byte;                      { # of pixels to draw from left }
                                            {     or right (depending on    }
                                            {     alignment)                }
    curValue:    hndData;                   { handle of array of contents   }
    maxFmts:     integer;                   { maximum # of format records   }
    growFmts:    integer;                   { # of format records by which  }
                                            {     to grow - don't grow if 0 }
    curFmts:     integer;                   { current # of format records   }
    fmtInfo:     hndRuns;                   { handle to array of runs       }
    protect:     boolean;                   { true => changes not allowed   }
    end;
ptrField = ^field;
hndField = ^ptrField;
```

Setting Breakpoints / Domains

6

```
D0=00000013 D1=00000000 D2=00000002 D3=001F2714
D4=2D48F900 D5=0010A845 D6=2D48FE00 D7=00000000
A0=00004004 A1=00C3FF12 A2=00200004 A3=0020A022
A4=00CC310E A5=0000057A A6=00C3FF6A A7=00C3FF36
>g

Level 7 Interrupt
00220E62       4CDF 08E0                 MOVEM.L (A7)+,D5-D7/A3
PC=00220E62 SR=2004  0  US=00F7C05A SS=00C3FF38 DC=0
D0=00000000 D1=0000FFFF D2=000004A5 D3=00CE07F3
D4=0010FFFA D5=00020000 D6=00CC1F96 D7=00A80700
A0=0036024E A1=00A84270 A2=00D03000 A3=00000400
A4=00A8426C A5=00CC4038 A6=00C3FF34 A7=00C3FF38
>g

Level 7 Interrupt
QUEUE_PR+0066 D280                       ADD.L   D0,D1
PC=002608F4 SR=0700  0  US=00F7DC32 SS=00CC0000 DC=0 PM=00006
D0=FFFF3481 D1=00CCA083 D2=00000002 D3=000007E4
D4=2D48F900 D5=00103004 D6=2D480078 D7=00F7DC3E
A0=00CCA832 A1=00F7DC62 A2=00CE004C A3=0020A022
A4=00CC310E A5=00CC4088 A6=00F7DC40 A7=00F7DC32
>ubr

Break Point
LETOTHER+0034*4E5E                       UNLK    A6
PC=0088303C SR=0000  0  US=00F7DC72 SS=00CC0000 DC=3 PM=00006
D0=00002000 D1=00000002 D2=00000002 D3=001FFFFF
D4=2D48F900 D5=00103004 D6=2D48FE00 D7=4AAE0000
A0=00F7DC72 A1=00CCA083 A2=00CE004C A3=0020A022
A4=02E46010 A5=00F7F7A4 A6=00F7DC74 A7=00F7DC72
>sc
At LETOTHER+0034
Stack frame at 00F7DC74 called from MAINLOOP+0176
Stack frame at 00F7DCCA called from 00240030
Stack frame at 00F7F7A4
>cl pc
>g

Level 7 Interrupt
00208C68     4840                        SWAP    D0
PC=00208C68 SR=2700  0  US=00F7DC72 SS=00C3FF4A DC=3 overridden 0
D0=00FE00FE D1=40000000 D2=00000000 D3=00002704
D4=2D48F700 D5=0010C030 D6=2D48FE00 D7=4AAE0000
A0=0000402C A1=00004000 A2=00208C0C A3=0020A022
A4=02E46010 A5=0000057A A6=00C3FF7E A7=00C3FF4A
>sc
At 00208C68
Stack frame at 00C3FF7E called from 0020A9A4
Stack frame at 00C3FFA8 called from 0020A86
Stack frame at 00C3FF30 called from 0020C0A
Stack frame at 00C3FFDC called from 00208166
Stack frame at 00C3FFFC
>ubr

Level 7 Interrupt
00208474     4E75                        RTS
PC=00208474 SR=2000  0  US=00F7DC72 SS=00C3FFEC DC=3 overridden 0
D0=00000002 D1=00000002 D2=00000000 D3=000007E4
D4=2D48F900 D5=0010C639 D6=2D48FE00 D7=4AA30000
A0=0020C014 A1=00000414 A2=00CE004C A3=0020A022
A4=02E46010 A5=00CC4083 A6=00C3FF?C A7=00C3FFEC
>ubr

Level 7 Interrupt
```

BREAKPOINTS

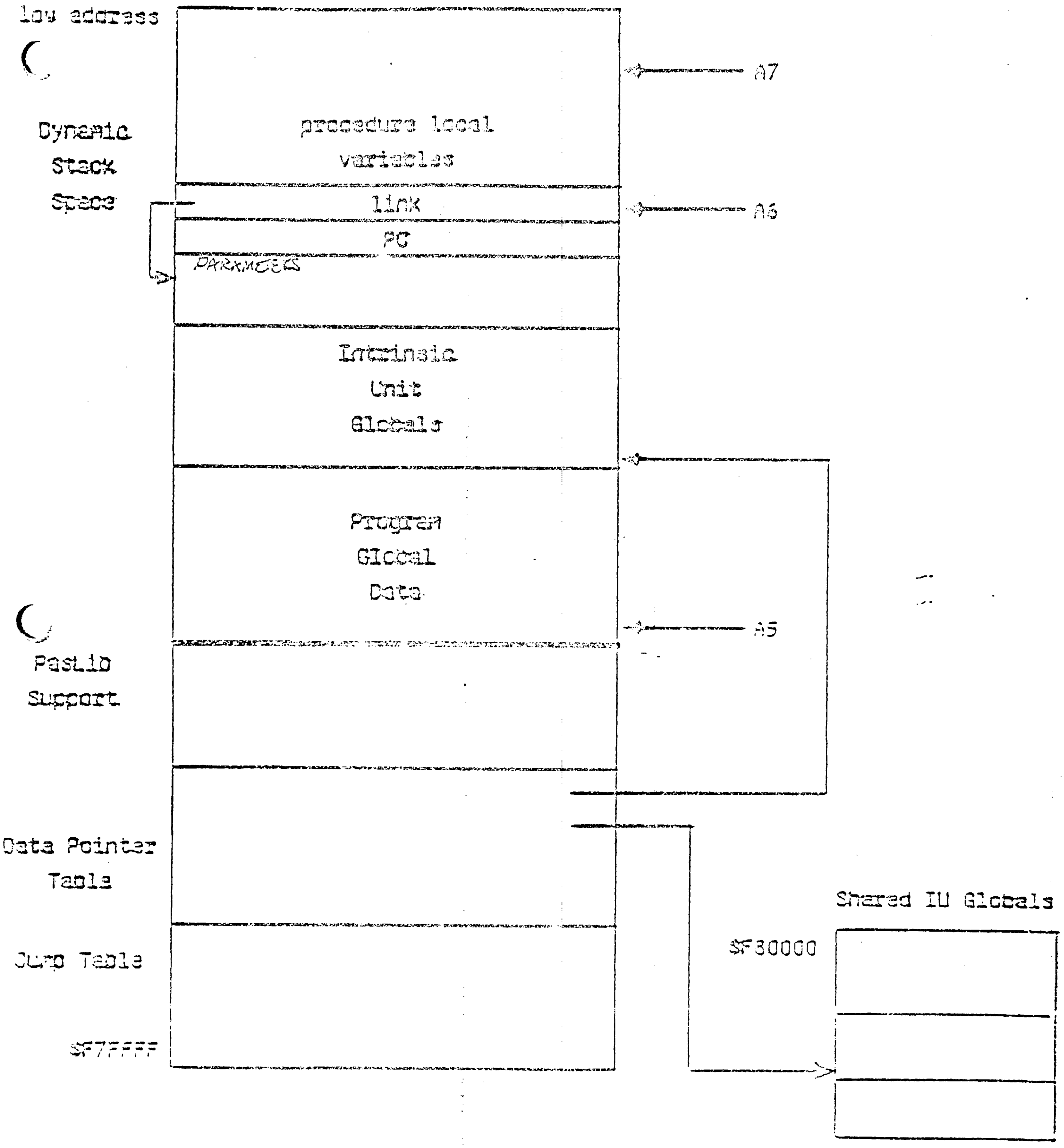
```
>g

Level 7 Interrupt
F0E%pack+0003 4A02                          TST.B   D2
PC=00AC0076 SR=0001  0  US=00F7C10C SS=00CC0000 DC=1 PR=00003
D0=FFFF0002 D1=0000000F D2=00000000 D3=000007E4
D4=0010FFFA D5=39700010 D6=FFFC397C D7=43180000
A0=00F7EBE3 A1=00F7C226 A2=00CE004C A3=00F7E054
A4=00F7EB64 A5=00F7F4A6 A6=00F7C20C A7=00F7C10C
>br gemenucm
>br gemenucm+8
>g

Break Point
GEMENUCM+0000*4A6F EFB4       GEMENUCM TST.W    $EFB4(A7)
PC=0064115C SR=0010  0  US=00F7BE3C SS=00CC0000 DC=1 PR=00003
D0=000000FF D1=000000FF D2=00000002 D3=001FFFFF
D4=00100003 D5=397C0000 D6=FFFC3900 D7=0000000E
A0=006413CA A1=00F3034A A2=00CE004C A3=70061080
A4=00F30436 A5=00F7F4A6 A6=00F7C21E A7=00F7BE3C
>sc
At GEMENUCM+0000
Stack frame at 00F7C21E called from PROCESST+011E
Stack frame at 00F7C258 called from MAINPROG+008A
Stack frame at 00F7C298 called from GRAPHICS+001E
Stack frame at 00F7F4A6
>dm ra6 30
00F7C21E      00F7 C258 0064 1F70  00F7 C22E 006E FFFC ...X.d.......n..
00F7C22E      00F3 0548 0001 0007  00C3 0010 706A 0000 ...H........pj..
00F7C23E      0000 0000 0100 06AC  0000 0000 0008 0000 ................

Break Point
GEMENUCM+0008*2F07                          MOVE.L  D7,-(A7)
PC=00641164 SR=0010  0  US=00F7BE3C SS=00CC0000 DC=1 PR=00003
D0=000000FF D1=000000FF D2=00000002 D3=001FFFFF
D4=00100003 D5=397C0000 D6=FFFC3900 D7=0000000E
A0=006413CA A1=00F3034A A2=00CE004C A3=70061080
A4=00F30436 A5=00F7F4A6 A6=00F7BE88 A7=00F7BE3C
>sc
At GEMENUCM+0008
Stack frame at 00F7BE88 called from GEMENUEV+048E
Stack frame at 00F7C21E called from PROCESST+011E
Stack frame at 00F7C258 called from MAINPROG+008A
Stack frame at 00F7C298 called from GRAPHICS+001E
Stack frame at 00F7F4A6
>dm ra6 30
00F7BE88      00F7 C21E 0064 180C  0008 0004 0010 FFFA .....d..........
00F7BE98      397C 0010 FFFC 397C  0007 FFFE 7006 1080 9|....9|....p...
00F7BEA8      00F3 0436 00AC 1453  6574 2041 7369 6465 ......Set.Aside
>il gemenucm
GEMENUCM+0000*4A6F EFB4       GEMENUCM TST.W    $EFB4(A7)
GEMENUCM+0004 4E56 FFB4                LINK     A6,#$FFB4
GEMENUCM+0008*2F07            PC       MOVE.L   D7,-(A7)
GEMENUCM+000A 3E2E 0008                MOVE.W   $0008(A6),D7
GEMENUCM+000E 4EAD 076E                JSR      SETWRKOR         ; 0036329E
GEMENUCM+0012 302E 000A                MOVE.W   $000A(A6),D0
GEMENUCM+0016 5540                     SUBQ.W   #2,D0
GEMENUCM+0018 6B00 02C2                BMI      *+$02C4          ; 00641438
GEMENUCM+001C 0C40 000A                CMPI.W   #$000A,D0
GEMENUCM+0020 6E00 02BA                BGT      *+$02BC          ; 00641438
GEMENUCM+0024 E348                     LSL.W    #1,D0
GEMENUCM+0026 303B 0006                MOVE.W   *+$0008(D0.W),D0 ; 0064118A
```

Stack Segment Layout



```
{$S smgrLoUse}    VAR
PROCEDURE CopySel(status : integer);
   VAR errnum : integer;
   BEGIN
   IF TraceSMGR then Writeln('Xtmsprocs CopySel' );
   if (typeofSel = aCellTxTSl) or
   (tblPars.EditColTitle and (typeOfSel = aColHedSl)) or
   (tblPars.EditRowTitle and (typeOfSel = aRowHedSl)) then
      Begin
      SetPnlPort(WidePnl);
      CutCopyField(wavFieldH,wavFstateH,false,true,errnum);
      Status := errnum;
      CutCopyField(selFieldH,selFstateH,false,false,errnum);
      END;
   END;
```

```
 il copysel
COPYSEL+0000  4A6F EFFE        COPYSEL  TST.W    $EFFE(A7)
COPYSEL+0004  4E56 FFFE                 LINK     A6,#$FFFE
COPYSEL+0008  48E7 0018                 MOVEM.L A3/A4,-(A7)
COPYSEL+000C  2860 02A0                 MOVE.L   $02A0(A5),A4
COPYSEL+0010  2660 029C                 MOVE.L   $029C(A5),A3
COPYSEL+0014  0C2C 0001 FFC9            CMPI.B   #$0001,$FFC9(A4)
COPYSEL+001A  57C0                      SEQ      D0
COPYSEL+001C  0C2C 0009 FFC9            CMPI.B   #$0009,$FFC9(A4)
COPYSEL+0022  57C1                      SEQ      D1
COPYSEL+0024  C22B FFD8                 AND.B    $FFD8(A3),D1
COPYSEL+0028  8001                      OR.B     D1,D0
COPYSEL+002A  0C2C 0008 FFC9            CMPI.B   #$0008,$FFC9(A4)
COPYSEL+0030  57C1                      SEQ      D1
COPYSEL+0032  C22B FFE2                 AND.B    $FFE2(A3),D1
COPYSEL+0036  8001                      OR.B     D1,D0
COPYSEL+0038  0240 0001                 ANDI.W   #$0001,D0
COPYSEL+003C  673A                      BEQ.S    *+$003C          ; 0056063C
COPYSEL+003E  3F2B FFCC                 MOVE.W   $FFCC(A3),-(A7)
COPYSEL+0042  A030 0170                 IUJSR    SETPNLPO         ; 0050089E
COPYSEL+0046  2F2C F442                 MOVE.L   $F442(A4),-(A7)
>il
COPYSEL+004A  2F2C F43C                 MOVE.L   $F43C(A4),-(A7)
COPYSEL+004E  4267                      CLR.W    -(A7)
COPYSEL+0050  1F3C 0001                 MOVE.B   #$0001,-(A7)
COPYSEL+0054  486E FFFE                 PEA      $FFFE(A6)
COPYSEL+0058  A08E 000E                 IUJSR    $008E000E
COPYSEL+005C  206E 0008                 MOVE.L   $0008(A6),A0
COPYSEL+0060  30AE FFFE                 MOVE.W   $FFFE(A6),(A0)
COPYSEL+0064  2F2C FFC4                 MOVE.L   $FFC4(A4),-(A7)
COPYSEL+0068  2F2C F44E                 MOVE.L   $F44E(A4),-(A7)
COPYSEL+006C  4267                      CLR.W    -(A7)
COPYSEL+006E  4267                      CLR.W    -(A7)
COPYSEL+0070  486E FFFE                 PEA      $FFFE(A6)
COPYSEL+0074  A08E 000E                 IUJSR    $008E000E
COPYSEL+0078  4CDF 1800                 MOVEM.L (A7)+,A3/A4
COPYSEL+007C  4E5E                      UNLK     A6
COPYSEL+007E  2E9F                      MOVE.L   (A7)+,(A7)
COPYSEL+0080  4E75                      RTS
COPYSEL+0082  C34F 5059 5345 4C20  0000 4A6F EFFE 4E56 .OPYSEL...Jo..NV
CUTSEL+0000   4A6F EFFE        CUTSEL   TST.W    $EFFE(A7)
CUTSEL+0004   4E56 FFFE                 LINK     A6,#$FFFE
>br copysel+8
>g

Break Point
COPYSEL+0008 >48E7 0018                 MOVEM.L A3/A4,-(A7)
PC=005605CC SR=0000  0  US=00F73EE8 SS=00000000 CC=1 P#=00005
D0=00000000 D1=00000000 D2=00000000 D3=001FFFFF
D4=000E2F20 D5=FAEA3F07 D6=A0300005 D7=4EA00005
A0=00521400 A1=00F73EE0 A2=00835000 A3=00F80436
A4=00F7D766 A5=00F7F73A A6=00F73EEA A7=00F73EE8
>dm ra6 40
00F73EEA      00F7 3F32 0052 150C  00F7 3F50 A03C 009E ...R.R.....P.<..
00F73EFA      4EA0 0005 00F7 D766  00F8 0436 00F7 BF00 N.......f.......
00F73F0A      000E 2F20 4E01 0002  00F8 0436 00F7 DF26 ../-N..........&
00F73F1A      00F7 3F38 0083 5D0E  00F7 0000 0002 001F ...8..].........
>pr 0
```

1

```
>  id 8e000e

Invalid log addr
>br 8e000e
>br copysel+14
>g

Break Point
COPYSEL+0014 *0C2C 0001 FFC9            CMPI.B  #$0001,$FFC9(A4)
PC=005605F8 SR=0000  0  US=00F78EE0 SS=00C00000 DC=1 P#=00005
D0=00000000 D1=00000000 D2=00000000 D3=001FFFFF
D4=000E2F2D D5=FAEA3F07 D6=A03C0005 D7=4EA00005
A0=005214C0 A1=00F78EE0 A2=00885C00 A3=00F7D766
A4=00F7CF80 A5=00F7F73A A6=00F78EEA A7=00F78EE0
>dm ra4+$ffffffc9
00F7CF48      0101 000C 0010 00D8  072E 0008 0746 00D8 ..............F..
>br copysel+24
>g

Break Point
COPYSEL+0024 *C22B FFDB                 AND.B   $FFDB(A3),D1
PC=00560608 SR=0009  0  US=00F78EE0 SS=00C00000 DC=1 P#=00005
D0=000000FF D1=00000000 D2=00000000 D3=001FFFFF
D4=000E2F2D D5=FAEA3F07 D6=A03C0005 D7=4EA00005
A0=005214C0 A1=00F78EE0 A2=00885C00 A3=00F7D766
A4=00F7CF80 A5=00F7F73A A6=00F78EEA A7=00F78EE0
>dm ra3+$ffffffdb
00F7D740      0101 0100 0001 0100  0000 0001 0000 0048 ...............H
>cv ra3+$ffffffdb
$F7D741=&16242497=00F7D741
>cv ra4+$ffffffc9
$F7CF49=&16240457=00F7CF49
>br copysel+3e
>g

Break Point
COPYSEL+003E *3F2B FFCC                 MOVE.W  $FFCC(A3),-(A7)
PC=00560622 SR=0000  0  US=00F78EE0 SS=00C00000 DC=1 P#=00005
D0=00000001 D1=00000000 D2=00000000 D3=001FFFFF
D4=000E2F2D D5=FAEA3F07 D6=A03C0005 D7=4EA00005
A0=005214C0 A1=00F78EE0 A2=00885C00 A3=00F7D766
A4=00F7CF80 A5=00F7F73A A6=00F78EEA A7=00F78EE0
>dm ra3+$ffffffcc
00F7D732      0003 0002 0001 0000  0001 0100 0101 0101 ................
>t

Trace Point
COPYSEL+0042  A053 0170                 IUJSR   SETPNLPO          ; 0050039E
PC=00560626 SR=0000  0  US=00F78EDE SS=00C00000 DC=1 P#=00005
D0=00000001 D1=00000000 D2=00000000 D3=001FFFFF
D4=000E2F2D D5=FAEA3F07 D6=A03C0005 D7=4EA00005
A0=005214C0 A1=00F78EE0 A2=00885C00 A3=00F7D766
A4=00F7CF80 A5=00F7F73A A6=00F78EEA A7=00F78EDE
:>dm ra7
00F78EDE      0003 00F8 0436 00F7  D766 BF32 00F7 BF32 .........f.R...R
>br copysel+54
>g
```

12

```
COPYSEL+0054 *486E FFFE                        PEA     $FFFE(A6)
PC=00560638 SR=0000  0  US=00F78ED4 SS=00CC0000 DO=1 P#=00005
D0=00000000 D1=00000000 D2=00000000 D3=001FFFFF
D4=000E2F2D D5=FAEA3F07 D6=A03C0005 D7=4EA00005
A0=0056062A A1=00F208EC A2=00885C00 A3=00F7D766
A4=00F7CF80 A5=00F7F73A A6=00F78EEA A7=00F78ED4
>dm ra6+$fffffffe
00F78EE8      3F52 00F7 8F52 0052  150C 00F7 8F50 A03C .R...R.R.....P.<
>dm ra6+$fffffffe
>g

Break Point
008E000E      *4EF9 008E 068E             JMP     $008E068E
PC=008E000E SR=0008  0  US=00F78ECC SS=00CC0000 DO=1 P#=00005
D0=00000000 D1=00000000 D2=00000000 D3=001FFFFF
D4=000E2F2D D5=FAEA3F07 D6=A03C0005 D7=4EA00005
A0=0056062A A1=00F208EC A2=00885C00 A3=00F7D766
A4=00F7CF80 A5=00F7F73A A6=00F78EEA A7=00F78ECC
>t

Trace Point
CUTCOPYF+0000 4A6F EFD0        CUTCOPYF TST.W   $EFD0(A7)
PC=008E068E SR=8008  0  US=00F78ECC SS=00CC0000 DO=1 P#=00005
D0=00000000 D1=00000000 D2=00000000 D3=001FFFFF
D4=000E2F2D D5=FAEA3F07 D6=A03C0005 D7=4EA00005
A0=0056062A A1=00F208EC A2=00885C00 A3=00F7D766
A4=00F7CF80 A5=00F7F73A A6=00F78EEA A7=00F78ECC
:>t

Trace Point
CUTCOPYF+0004 4E56 FFD0                   LINK    A6,#$FFD0
PC=008E0692 SR=8000  0  US=00F78ECC SS=00CC0000 DO=1 P#=00005
D0=00000000 D1=00000000 D2=00000000 D3=001FFFFF
D4=000E2F2D D5=FAEA3F07 D6=A03C0005 D7=4EA00005.
A0=0056062A A1=00F208EC A2=00885C00 A3=00F7D766
A4=00F7CF80 A5=00F7F73A A6=00F78EEA A7=00F78ECC
:>t

Trace Point
CUTCOPYF+0008 48E7 0318                   MOVEM.L D6/D7/A3/A4,-(A7)
PC=008E0696 SR=8000  0  US=00F78E98 SS=00CC0000 DO=1 P#=00005
D0=00000000 D1=00000000 D2=00000000 D3=001FFFFF
D4=000E2F2D D5=FAEA3F07 D6=A03C0005 D7=4EA00005
A0=0056062A A1=00F208EC A2=00835C00 A3=00F7D766
A4=00F7CF80 A5=00F7F73A A6=00F78EC8 A7=00F78E98
:>dm ra6
00F78EC8      00F7 8EEA 0056 0640  00F7 8EE8 01F8 0000 .....V.@........
>br 560640
>g

Break Point
COPYSEL+005C *206E 0008                   MOVE.L  $0008(A6),A0
PC=00560640 SR=0000  0  US=00F78EE0 SS=00CC0000 DO=1 P#=00005
D0=00002700 D1=00000000 D2=00000002 D3=001FFFFF
D4=000E2F2D D5=FAEA3F07 D6=A03C0005 D7=4EA00005
A0=00560640 A1=00009400 A2=00E0040 A3=00F7D766
A4=00F7CF80 A5=00F7F73A A6=00F78EEA A7=00F78EE0
>dm ra6+$fffffffe
00F78EE8      0000 00F7 8F52 0052  150C 00F7 8F50 A03C .....R.R.....P.<
>g
```