User Trap Handlers

Trap Handlers

The 68000 family of microprocessors has sixteen software trap exception vectors. The first (trap 0) is reserved for making OS-9 system calls. You may use the remaining fifteen as service requests to user-defined “user trap handlers.”

Microware provides standard trap handlers for I/O conversions in the C language, floating point math, and trigonometric functions. The following traps are reserved:

trap 13	CIO is automatically called for any C program.
trap 15	Math is called for floating point math, extended integer math and/or type conversion. It is also used for programs using transcendental and/or extended mathematical functions.

For further information about the math module, refer to Chapter 6.

A ***user trap handler*** is an OS-9 module that usually contains a set of related subroutines. Any user program may dynamically link to the user trap handler and call it at execution time. **NOTE:** While trap handlers reduce the size of the execution program, they do not do anything that could not be done by linking the program with appropriate library routines at compilation time. In fact, programs that call trap handlers execute slightly slower than linked programs that perform the same function.

Trap handlers must be written in a language that compiles to machine code (such as assembly language or C). They should be suitably generic for use by a number of programs.

Trap handlers are similar to normal OS-9 program modules, except that trap handlers have three execution entry points: a trap execution entry point, trap initialization entry point, and trap termination entry point.

Trap handler modules are of module type TrapLib and module language Objct.

The trap module routines usually execute as though they were called with a jsr instruction, except for minor stack differences. Any system calls or other operations that the calling module could perform are usable in the trap module.

It is possible to write a trap handler module that runs in system state. This is rarely advisable, but sometimes necessary. For a discussion of the uses of system state, refer to the System Call Overview in Chapter 2.

Installing and Executing Trap Handlers

A user program installs a trap handler by executing the F$TLink system request. When this is done, the OS-9 kernel links to the trap module, allocates and initializes its static storage (if any), and executes the trap module's initialization routine.

Typically, the initialization routine has very little to do. You could use it to open files, link to additional trap or data modules, or perform other startup activities. It is called only once per trap handler in any given program.

A trap module that is used by a program is usually installed as part of the program's initialization code. At initialization, a particular trap number (1-15) is specified that refers to the trap module. The program invokes functions in the trap module by using the 68000 trap instruction corresponding to the trap number specified. This is followed by a function word that is passed to the trap handler itself. The arrangement is very similar to making a normal OS-9 system call.

The OS-9 relocatable macro assembler has special mnemonics to make trap calls more apparent. These are OS9 for trap 0, and tcall for the other user traps. They work like built-in macros, generating code as illustrated in the following section.

OS9 and tcall: Equivalent Assembly Language Syntax

Mnemonic	*Code Generated*
OS9 F$TLink	**trap 0** **dc.w F$TLink**
tcall T$Math,T$DMul	**trap T$Math** **dc.w T$DMul**

From user programs, it is possible to delay installing a trap module until the first time it is actually needed. If a trap module has not been installed for a particular trap when the first tcall is made, OS-9 checks the program's exception entry offset (M$Excpt in the module header). The program aborts if this offset is zero. Otherwise, OS-9 passes control to the exception routine. At this point, the trap handler can be installed, and the first tcall reissued. The second example in this chapter shows how to do this.

Calling a Trap Handler

The actual details of building and using a trap handler are best explained by means of a simple complete example.

Example One: The following program (TrapTst) uses trap vector 5. It installs the trap handler and then calls it twice.

```
          nam     TrapTst1
          ttl     example one - link and call trap handler
          use     /dd/defs/oskdefs.d
Edition     equ     1
Typ_Lang    equu    (Prgrm<<8)+Objct
Attr_Rev    equ     (ReEnt<<8)+0
          psect   traptst,Typ_Lang,Attr_Rev,Edition,1024,Test

TrapNum     equ     5             trap number to use
TrapName    dc.b    "trap",0      name of trap handler

******************************
* Main program entry point

Test:     moveq   #TrapNum,d0     trap number to assign
          moveq   #0,d1           no optional memory override
          lea     TrapName(pc),a0 ptr to name of trap handler
          os9     F$TLink         install trap handler
          bcs.s   Test99          abort if error
          tcall   TrapNum,0       call trap function #0
          bcs.s   Test99          abort if error
          tcall   TrapNum,1       call trap function #1
          bcs.s   Test99          abort if error
          moveq   #0,d1           exit without error
Test99      os9     F$Exit          exit
          ends
```

Example Two: The following example shows how you could modify the preceding program to install the trap handler in an exception routine when the first tcall is executed. You might do this for a trap handler that may not be used at all by a program, depending on circumstances.

This example does not initialize the trap handler before using it, but is otherwise identical to Example One. It provides a LinkTrap subroutine to automatically install the trap handler when it is first used. Refer to the trace of Example Two later in this chapter for more information.

```
          nam     TrapTst2
          ttl     example two - call trap handler
          use     /dd/defs/oskdefs.d
Edition     equ     1
Typ_Lang    equ     (Prgrm<<8)+Objct
```

```
Attr_Rev    equ     (ReEnt<<8)+0
```

EXAMPLE TWO (continued):

```
        psect   traptst,Typ_Lang,Attr_Rev,Edition,1024,Test,LinkTrap

TrapNum     equ     5               trap number to use
TrapName    dc.b    "trap",0        name of trap handler

**********************************
* Main program entry point

Test:   tcall   TrapNum,0           call trap function #0
        bcs.s   Test99              abort if error
        tcall   TrapNum,1           call trap function #1
        bcs.s   Test99              abort if error
        moveq   #0,d1               exit without error
Test99  os9     F$Exit              exit

**********************************
* Subroutine LinkTrap
* Installs trap handler and then executes first trap call.
* Note:  Error checking is minimized to keep example simple.
*
* Passed:  d0-d7 = caller's registers
*       a0-a5 = caller's registers
*       (a6)  = trap handler static storage pointer
*       (a7)  = trap init/entry stack frame
*
* Returns: trap installed, backs up PC to execute "tcall" instruction
*
* The stack looks like this:
*       .-----------------------.
*     +8 |   caller's return PC   |
*       >-----------------------<
*     +6 |  vector #  |
*       >-----------<
*     +4 | func code  |
*       >-----------------------<
*       |  caller's a6 register  |
*   (a7)-> -----------------------

LinkTrap:   addq.l  #8,a7               discard excess stack info
        movem.l d0-d1/a0-a2,-(a7)   save registers
        moveq   #TrapNum,d0         trap number to assign
        moveq   #0,d1               no optional memory override
        lea     TrapName(pc),a0     ptr to name of trap handler
        os9     F$TLink             install trap handler
        bcs.s   Test99              abort if error
        movem.l (a7)+,d0-d1/a0-a2   retrieve registers
        subq.l  #4,(a7)             back up to tcall instruction
```

```
 rts   return          to tcall instruction
 ends
```

An Example Trap Handler

The following makefile makes the example trap handler and test programs:

```
# makefile - Used to make the example trap handler and test programs.

RDIR   = RELS
TRAP   = trap
TEST1  = traptst1
TEST2  = traptst2

# Dependencies for making the entire trap example.

trap.example: $(TRAP) $(TEST1) $(TEST2)
   touch trap.example

# Dependencies for making the trap handler.

$(TRAP): $(TRAP).r
   l68 -g $(RDIR)/$(TRAP).r -l=/dd/lib/sys.l -o=$(TRAP)

# Dependencies for making the traptst1 test program.

$(TEST1): $(TEST1).r
   l68 -g $(RDIR)/$(TEST1).r -l=/dd/lib/sys.l -o=$(TEST1)

# Dependencies for making the traptst2 test program.

$(TEST2): $(TEST2).r
   l68 -g $(RDIR)/$(TEST2).r -l=/dd/lib/sys.l -o=$(TEST2)
```

The trap handler itself is listed below. It is artificially simple to avoid confusion. Most trap handlers have several functions, and generally begin with a dispatch routine based on the function code.

```
        nam     Trap Handler
        ttl     Example trap handler module
        use     /dd/defs/oskdefs.d
Type    set     (TrapLib<<8)+Objct
Revs    set     ReEnt<<8
        psect   traphand,Type,Revs,0,0,TrapEnt
        dc.l    TrapInit        initialization entry point
        dc.l    TrapTerm        termination entry point

****************************************************
* TrapInit:  Trap handler initialization entry point.
*
* Passed:  d0.w  = User Trap number (1-15)
*          d1.l  = (optional) additional static storage
*          d2-d7 = caller's registers at the time of the trap
*       (a0)  = trap handler module name pointer
*       (a1)  = trap handler execution entry point
*       (a2)  = trap module pointer
```

EXAMPLE TRAP HANDLER (continued):

```
*       a3-a5 = caller's registers (parameters required by handler)
*       (a6)  = trap handler static storage pointer
*       (a7)  = trap init stack frame pointer
*
* Returns: (a0)  = updated trap handler name pointer
*       (a1)  = trap handler execution entry point
*       (a2)  = trap module pointer
*       cc    = carry set, d1.w=error code if error
*       Other values returned are dependent on the trap handler
*
* The stack looks like this:
*       .------------------------.
*     +8 |    caller's return PC   |
*       >------------------------<
*     +4 |    0000    |    0000    |
*       >------------|------------<
*       |   caller's a6 register  |
*   (a7)-> ------------------------

TrapInit  movem.l (a7),a6           restore user's a6 register
          addq.l  #8,a7             take other stuff off the stack
          rts                       return to caller

******************************************
* TrapEnt:  User trap handler entry point.
*
* Passed:  d0-d7 = caller's registers
*       a0-a5 = caller's registers
*       (a6)  = trap handler's static storage pointer
*       (a7)  = trap entry stack frame pointer
*
* Returns: cc    = carry set, d1.w=error code if error
*       Other values returned are dependent on the trap handler
*
* The stack looks like this:
*       .-----------------------.
*     +8 |   caller's return PC   |
*       >-----------------------<
*     +6 |  vector #  |
*       >------------<
*     +4 | func code  |
*       >-----------------------<
*       |  caller's a6 register  |
*   (a7)-> -----------------------
```

```
         org     0            stack offset definitions
S.d0     do.l    1            caller's d0 reg
S.d1     do.l    1            caller's d1 reg
S.a0     do.l    1            caller's a0 reg
S.a6     do.l    1            caller's a6 reg
S.func   do.w    1            trap function code
S.vect   do.w    1            vector number
```

EXAMPLE TRAP HANDLER (continued):

```
S.pc     do.l    1            return pc

TrapEnt: movem.l d0-d1/a0,-(a7)      save registers
         move.w  S.func(a7),d0       get function code
         cmp.w   #1,d0               is function in range?
         bhi.s   FuncErr             abort if not
         beq.s   Trap10              branch if function code #1
         lea     String1(pc),a0      get first string ptr
         bra.s   Trap20              continue
Trap10   lea     String2(pc),a0      get second string ptr
Trap20   moveq   #1,d0               standard output path
         moveq   #80,d1              maximum bytes to write
         os9     I$WritLn            output the string
         bcs.s   Abort               abort if error
Trap90   movem.l (a7)+,d0-d1/a0/a6-a7 restore regs
         rts                         return to user

FuncErr  move.w  #1<<8+99,d2         abort (return error 001:099)
Abort    move.w  d1,S.d1+2(a7)       put error code in d1.w
         ori     #Carry,ccr          set carry
         bra.s   Trap90              exit

String1  dc.b    "Microware Systems Corporation",C$CR,0
String2  dc.b    "    Quality keeps us #1",C$CR,0

************************************************
* TrapTerm:  Trap handler terminate entry point.
*
* As of this release (OS-9 V2.4) the trap termination entry
* point is never called by the OS-9 kernel.  Documentation
* details will be available when a working implementation
* exists.

TrapTerm move.w  #1<<8+199,d1        never called, if it gets here
         os9     F$Exit              crash program (Error 001:199)
         ends
```

Trace of Example Two using the Example Trap Handler

It is extremely educational to watch the OS-9 user debugger trace through the execution of Example Two (using the example trap handler). User trap handlers look like subroutines to the debugger, so it is possible to trace through them. The output should appear something like this:

```
(beginning of second example program)
Test            >4E450000     trap #5,0
```

NOTE: Because the trap handler has not been linked as in Example One, control jumps to the subroutine LinkTrap:

```
LinkTrap         >508F         addq.l #8,a7
LinkTrap+0x2     >48E7C0E0     movem.l d0-d1/a0-a2,-(a7)
LinkTrap+0x6     >7005         moveq.l #5,d0
LinkTrap+0x8     >7200         moveq.l #0,d1
LinkTrap+0xA     >41FAFFDC     lea.l bname+0xA(pc),a0
LinkTrap+0xE     >4E400021     os9 F$TLink
```

NOTE: Control switches to the subroutine TrapInit and then returns to LinkTrap:

```
trap:btext+0x50   >4CD74000     movem.l (a7),a6
trap:btext+0x54   >508F         addq.l #8,a7
trap:btext+0x56   >4E75         rts
LinkTrap+0x12     >65E8         bcs.b Test+0xE
LinkTrap+0x14     >4CDF0703     movem.l (a7)+,d0-d1/a0-a2
LinkTrap+0x18     >5997         subq.l #4,(a7)
LinkTrap+0x1A     >4E75         rts
```

NOTE: Control now returns to the main program to re-execute the tcall instruction.

```
Test               >4E450000       trap #5,0
trap:TrapEnt       >48E7C080       movem.l d0-d1/a0,-(a7)
trap:TrapEnt+0x4   >302F0010       move.w 16(a7),d0
trap:TrapEnt+0x8   >B07C0001       cmp.w #1,d0
trap:TrapEnt+0xC   >621C           bhi.b trap:TrapEnt+0x2A
trap:TrapEnt+0xE   >6706           beq.b trap:TrapEnt+0x16
trap:TrapEnt+0x10  >41FA0026       lea.l trap:TrapEnt+0x38(pc),a0
trap:TrapEnt+0x14  >6004           bra.b trap:TrapEnt+0x1A
trap:TrapEnt+0x1A  >7001           moveq.l #1,d0
trap:TrapEnt+0x1C  >7250           moveq.l #80,d1
trap:TrapEnt+0x1E  >4E40008C       os9 I$WritLn
Microware Systems Corporation
trap:TrapEnt+0x22  >650A           bcs.b trap:TrapEnt+0x2E
trap:TrapEnt+0x24  >4CDFC103       movem.l (a7)+,d0-d1/a0/a6-a7
trap:TrapEnt+0x28  >4E75           rts
Test+0x4           >6508           bcs.b Test+0xE
Test+0x6           >4E450001       trap #5,0x1

trap:TrapEnt       >48E7C080       movem.l d0-d1/a0,-(a7)
trap:TrapEnt+0x4   >302F0010       move.w 16(a7),d0
trap:TrapEnt+0x8   >B07C0001       cmp.w #1,d0
trap:TrapEnt+0xC   >621C           bhi.b trap:TrapEnt+0x2A
trap:TrapEnt+0xE   >6706           beq.b trap:TrapEnt+0x16->
trap:TrapEnt+0x16  >41FA003F       lea.l trap:TrapEnt+0x57(pc),a0
trap:TrapEnt+0x1A  >7001           moveq.l #1,d0
trap:TrapEnt+0x1C  >7250           moveq.l #80,d1
trap:TrapEnt+0x1E  >4E40008C       os9 I$WritLn
  Quality keeps us #1
trap:TrapEnt+0x22  >650A           bcs.b trap:TrapEnt+0x2E
trap:TrapEnt+0x24  >4CDFC103       movem.l (a7)+,d0-d1/a0/a6-a7
trap:TrapEnt+0x28  >4E75           rts
Test+0xA           >6502           bcs.b Test+0xE
Test+0xC           >7200           moveq.l #0,d1
Test+0xE           >4E400006       os9 F$Exit
```

End of Chapter 5