```
$EXEC (DoAsm,DoCompile,DoLink)
$clear screen
$writeln
$writeln "                        Generating new LAUNCH program"
a{ssemble}romdiag/ROMdiag.red
{no list file}
{use ROMdiag.red.obj as the output}
L{ink}?{select options}
+X{link for Mac}
{no more options}
romdiag/ALROMTALK
romdiag/ROMdiag.red
obj/QuickDraw
obj/OSTraps
obj/ToolTraps
obj/PackTraps
obj/macpaslib


romdiag/xROMTALK.red
Rmac/rmaker
romdiag/Launchres.red
$  { Copy the resource file to a Mac micro-floppy in the lower drive }
$
  r{un}mac/maccom
  F{inderInfo}Y{es}L{isa->Mac}romdiag/ROMTALK.Red.rsrc
  romdiag/ROMTALK.Red
  APPL

  N{o}
  E{ject}Q{uit}
$ENDEXEC
```

```
*
*     resource definition file source code
*     Launch
*
*     01/29/85 GRC, Original.
*

romdiag/ROMTALK.Red.Rsrc

Type CODE
  romdiag/xROMTALK.Red,0
```

```
PROGRAM LRomTalk;
{ Edit date: 01/29/85
          }

USES    {$U-}
        {$R-}
        {$M+}
        {$U -w-obj-QuickDraw }  QuickDraw,
        {$U -w-Obj-OsIntf    }  OsIntf,
        {$U -w-obj-toolIntf  }  ToolIntf,
        {$U -w-obj-PackIntf  }  PackIntf;

var
   Index,SubIndex,Dummy:integer;

   Procedure ROMTalk;EXTERNAL;


{*****************************************************************************}
{*****************************************************************************}
{*****************************************************************************}

begin

for Index:= 1 to 200 do
   for SubIndex:= 1 to 2000 do
      Dummy:= 1;

ROMTalk;


END.
```

```
;
; File: RomDiag.TEXT
; Function: This is a ROM based debugger, it uses only CPU registers and the
;           I/O space of the SCC.  No memory or stack is required.
;
; Edit date:    02/11/85 GRC,   Original.
;               02/19/85 twm,   Modified (RomTalk) for different hardware
;===============================================================================
;
;  Register usage:
;
;       a0 - SCC read address           d0 - Scratch
;       a1 - SCC write address          d1 - Scratch
;       a2 - Address looking at         d2 - Scratch
;       a3 - Scratch                    d3 - Scratch
;       a4 -                            d4 - Scratch
;       a5 - Subroutine calls           d5 -
;       a6 - Subroutine calls           d6 - Last byte read in
;       a7 - Subroutine calls           d7 - Command, building, lower word
;                                            Last command executed, upper word
;
;===============================================================================
;
        .PROC   ROMTALK

        .include  romdiag/HdweEqu.Red.Text              ; magic numbers to address hardware

Initialize
        move.w  #IntMask,sr                 ;Disable any interrupts
        .word   $A852                       ; trap to HideCursor
        clr.l   d0                          ;Clear all registers
        clr.l   d1
        clr.l   d2
        clr.l   d3
        clr.l   d4
        clr.l   d5
        clr.l   d6
        clr.l   d7
        move.l  d0,a0
        move.l  d0,a1
        move.l  d0,a2
        move.l  d0,a3
        move.l  d0,a4
        move.l  d0,a5
        move.l  d0,a6
        move.l  d0,a7
;
;
        bsr6    SetExceptions   ;Install exception vectors
        bsr7    Init7SCC        ;Initialize SCC port
        bsr7    Init7Msg        ;Put out init message
;
TopCommand
        bsr6    SetExceptions   ;Install exception vectors in case wiped out
;
        clr.w   d7
        bsr6    Get6Char        ;Get first character of next command
        move.b  d6,d7           ;...get command character
        LSL.W   #8,D7           ;...make room for next character
        cmp.b   #$0D,d6         ;Was it just a CR?
        bne     GetSecond       ;...no, so not a repeat
        swap    d7              ;Get to last command, a repeat
        move.w  d7,d0           ;...place last command in current command area
        swap    d7
        move.w  d0,d7           ;...yes, so repeat command
        bra     ProcessCommand
;
GetSecond
        bsr6    Get6Char        ;Get second character of next command
        move.b  d6,d7           ;...get second command character
        cmp.b   #$0D,d7         ;...see if it was a single char command
        bne     ProcessCommand
        move.b  #'*',d7         ;...it is single command so place * in second
;
;
ProcessCommand
        MOVE.w  D7,d0           ;save the command for later repeat
        swap    D7
        MOVE.w  D0,d7           ;save the command for later repeat
        lea     SYSCMDS,A6      ;Get start of command table
;
cmdLookup
        clr.l   d5
        MOVE.W  (A6)+,D4        ;Get possible command
        MOVE.W  (A6)+,D5        ;Get possible offset
        beq.s   What            ;...see if end of table
;
        CMPI.B  #'@',D4         ;see if wild card lookup (D@)
        BNE.S   @1              ;...no if not @
        CMPI.B  #'0',D7         ;...yes,see if 0 to 7
        BMI.S   cmdLookup       ;......fail if below range
        CMPI.B  #'8',D7
        BPL.S   cmdLookup       ;......fail if above range
;
@1      CMP.W   D4,D7           ;does this command match?
        BNE.S   cmdLookup       ;...no, keep looking
;
        move.l  #Stack,a7       ;Start of stack if non-memory test, eg VIA, SCC.
;
        LEA     SysCmds,A6      ;point to the command list
        JMP     0(A6,D5)        ;...dispatch to the command
;
What
```

```
        bsr6    Return6          ;Send CR
        lea     HuH,a3          ;Point the the 'Huh?' message
        bsr6    Write6Line      ;...write it out the SCC
        bra     TopCommand
;
;
;
FATAL   bra     Initialize      ;Initialize ROM, fatal error
;
;-------------------------------------------------------------------------
;
; Put out initialize message to tell user that we are here
;
Init7Msg
        bsr6    Return6         ;Send CR
        lea     MyInitMsg,a3    ;Point to message
        bsr6    Write6Line      ;...write it out the SCC
        rts7
;
;-------------------------------------------------------------------------

        .include  romdiag/CmdTable.Text         ; list and offset of all commands
        .include  romdiag/ROMSCC.Text           ; SCC control subroutines
        .include  romdiag/Exceptions.Text       ; exception handlers

;-------------------------------------------------------------------------
;
; Initialize debugger
;
IxCMD
        bra     Initialize
;
;-------------------------------------------------------------------------
;
; Run all diagnostics
RDCMD
        clr.l   d4
        bsr6    Return6                 ;Send CR
;
        move.l  #RamStart,a2            ;Address to test
        bsr7    BasicDataLine           ;Do Data line test
        bsr7    CellTest                ; and the rest of the memory tests
        bsr7    RAddrTest
        bsr7    RFixedTest
        bsr7    M1Test
        bsr7    M2Test
        bsr7    LWInvTest
        bsr7    WInvTest
        bsr7    BInvTest
;
        bsr6    SetExceptions       ;Install exception vectors in case wiped out
;
        move.l  #Stack,a7           ;Start of stack if non-memory test, eg VIA, SCC.
        bsr6    Via6Test            ; test all 3 VIA's
        bra     TopCommand          ; until SCC test is fixed

        bsr4    SCCREGISTERS
        bsr4    SCCCRESET
        bsr4    SCCINTERNAL
        bsr4    SCCGINTERRUPT
;
        bra     TopCommand
;
;
;-------------------------------------------------------------------------
;
; Run all memory tests
MTCMD
        clr.l   d4
;
        bsr6    Return6                 ;Send CR
;
        move.l  #RamStart,a2            ;Address to test
        bsr7    BasicDataLine           ;Do Data line test
        bsr7    CellTest
        bsr7    RAddrTest
        bsr7    RFixedTest
        bsr7    M1Test
        bsr7    M2Test
        bsr7    LWInvTest
        bsr7    WInvTest
        bsr7    BInvTest
;
        bsr6    SetExceptions       ;Install exception vectors in case wiped out
        move.l  #Stack,a7           ;Start of stack if non-memory test, eg VIA, SCC.
        bra     TopCommand
;
;-------------------------------------------------------------------------
;
; Diagnostics till a failure  DF
DFCMD
        clr.l   d4
        bsr6    Return6                 ;Send CR
        clr.l   d7                      ;Pass counter
;
DFLOOP
        bsr6    Return6                 ;Send CR
;
        add.l   #1,d7                   ;Increment pass counter
        lea     PHeadMsg,a3             ;Point to message
        bsr6    PWrite6Line             ;...write it out the SCC
        move.l  d7,d5                   ;Pass count
        bsr6    Dsp6LHex
```

```
        bsr6    Return6                 ;Send CR
;
        move.l  #RamStart,a2            ;Address to test
        bsr7    BasicDataLine           ;Do Data line test
        cmp.w   #0,d6                   ;...check the error flag
        bne     TopCommand              ;...exit on an error
;
        bsr7    CellTest
        cmp.w   #0,d6                   ;...check the error flag
        bne     TopCommand              ;...exit on an error
;
        bsr7    RAddrTest
        cmp.w   #0,d6                   ;...check the error flag
        bne     TopCommand              ;...exit on an error
;
        bsr7    RFixedTest
        cmp.w   #0,d6                   ;...check the error flag
        bne     TopCommand              ;...exit on an error
;
        bsr7    M1Test
        cmp.w   #0,d6                   ;...check the error flag
        bne     TopCommand              ;...exit on an error
;
        bsr7    M2Test
        cmp.w   #0,d6                   ;...check the error flag
        bne     TopCommand              ;...exit on an error
;
        bsr7    LWInvTest
        cmp.w   #0,d6                   ;...check the error flag
        bne     TopCommand              ;...exit on an error
;
        bsr7    WInvTest
        cmp.w   #0,d6                   ;...check the error flag
        bne     TopCommand              ;...exit on an error
;
        bsr7    BInvTest
        cmp.w   #0,d6                   ;...check the error flag
        bne     TopCommand              ;...exit on an error
;
        bsr6    SetExceptions           ;Install exception vectors in case wiped out
;
        move.l  #Stack,a7               ;Start of stack if non-memory test, eg VIA, SCC.
        bsr6    Via6Abort               ; will test 3 vias and abort on error
        bra     DFLOOP

        bsr4    SCCREGISTERS
        cmp.w   #0,d6                   ;...check the error flag
        bne     TopCommand              ;...exit on an error
;
        bsr4    SCCCRESET
        cmp.w   #0,d6                   ;...check the error flag
        bne     TopCommand              ;...exit on an error
;
        bsr4    SCCINTERNAL
        cmp.w   #0,d6                   ;...check the error flag
        bne     TopCommand              ;...exit on an error
;
        bsr4    SCCGINTERRUPT
        cmp.w   #0,d6                   ;...check the error flag
        bne     TopCommand              ;...exit on an error
;
        bra     DFLOOP

;-------------------------------------------------------------------------
;
; Memory Diagnostics till a failure  MF
MFCMD
        clr.l   d4
        bsr6    Return6                 ;Send CR
        clr.l   d7                      ;Pass counter
;
MFLOOP
        bsr6    Return6                 ;Send CR
;
        add.l   #1,d7                   ;Increment pass counter
        lea     PHeadMsg,a3             ;Point to message
        bsr6    PWrite6Line             ;...write it out the SCC
        move.l  d7,d5                   ;Pass count
        bsr6    Dsp6LHex
        bsr6    Return6                 ;Send CR
;
        move.l  #RamStart,a2            ;Address to test
        bsr7    BasicDataLine           ;Do Data line test
        cmp.w   #0,d6                   ;...check the error flag
        bne     TopCommand              ;...exit on an error
;
        bsr7    CellTest
        cmp.w   #0,d6                   ;...check the error flag
        bne     TopCommand              ;...exit on an error
;
        bsr7    RAddrTest
        cmp.w   #0,d6                   ;...check the error flag
        bne     TopCommand              ;...exit on an error
;
        bsr7    RFixedTest
        cmp.w   #0,d6                   ;...check the error flag
        bne     TopCommand              ;...exit on an error
;
        bsr7    M1Test
        cmp.w   #0,d6                   ;...check the error flag
        bne     TopCommand              ;...exit on an error
;
        bsr7    M2Test
```

```
        cmp.w   #0,d6                   ;...check the error flag
        bne     TopCommand              ;...exit on an error
;
        bsr7    LWInvTest
        cmp.w   #0,d6                   ;...check the error flag
        bne     TopCommand              ;...exit on an error
;
        bsr7    WInvTest
        cmp.w   #0,d6                   ;...check the error flag
        bne     TopCommand              ;...exit on an error
;
        bsr7    BInvTest
        cmp.w   #0,d6                   ;...check the error flag
        bne     TopCommand              ;...exit on an error
;
        bsr6    SetExceptions           ;Install exception vectors in case wiped out
;
        move.l  #Stack,a7               ;Start of stack if non-memory test, eg VIA, SCC.
        bra     MFLOOP

;-------------------------------------------------------------------------
;-------------------------------------------------------------------------
;
        .include  romdiag/Help.Text
        .include  romdiag/MEMCommands.Text
;
;-------------------------------------------------------------------------
;-------------------------------------------------------------------------
;
; Displays the word value currently in d5
;
DspWHex
        ror.w   #8,d5           ;Move to upper byte
        ror.w   #4,d5           ;Move to upper nibble of upper byte
        move.b  d5,d0           ;...display
        bsr6    Put6Hex
        rol.w   #4,d5           ;Move to lower nibble of upper byte
        move.b  d5,d0           ;...display
        bsr6    Put6Hex
        rol.w   #8,d5           ;Move to original position
;
; Displays the byte value currently in d5
;
DspBHex
        ror.w   #4,d5           ;Move to upper nibble of byte
        move.b  d5,d0           ;...display
        bsr6    Put6Hex
        rol.w   #4,d5           ;Move to lowest nibble
        move.b  d5,d0           ;Show lowest nibble
        bsr6    Put6Hex
        move.w  #' ',d0         ;Space between displays
        bsr5    Out5Char
        rts7
;
;-------------------------------------------------------------------------
;
; Displays the long word value currently in d5
;
DspLHex
        move.w  #7,d1           ;Counter for digits displayed
@1      rol.l   #4,d5
        move.b  d5,d0
        bsr6    Put6Hex
        dbra    d1,@1
        move.w  #' ',d0
        bsr5    Out5Char
        rts7
;
;-------------------------------------------------------------------------
;
; Displays the word value currently in d5
;
Dsp6WHex
        ror.w   #8,d5           ;Move to upper byte
        ror.w   #4,d5           ;Move to upper nibble of upper byte
        move.b  d5,d0           ;...display
        ANDI.B  #$0F,D0
        ORI.B   #$30,D0
        CMPI.B  #$39,D0
        BLE.S   @0
        ADDQ.W  #7,D0
@0      bsr5    Out5Char
        rol.w   #4,d5           ;Move to lower nibble of upper byte
        move.b  d5,d0           ;...display
        ANDI.B  #$0F,D0
        ORI.B   #$30,D0
        CMPI.B  #$39,D0
        BLE.S   @1
        ADDQ.W  #7,D0
@1      bsr5    Out5Char
        rol.w   #8,d5           ;Move to original position
;
; Displays the byte value currently in d5
;
Dsp6BHex
        ror.w   #4,d5           ;Move to upper nibble of byte
        move.b  d5,d0           ;...display
        ANDI.B  #$0F,D0
        ORI.B   #$30,D0
        CMPI.B  #$39,D0
        BLE.S   @0
        ADDQ.W  #7,D0
@0      bsr5    Out5Char
```

```
        rol.w   #4,d5           ;Move to lowest nibble
        move.b  d5,d0           ;Show lowest nibble
        ANDI.B  #$0F,D0
        ORI.B   #$30,D0
        CMPI.B  #$39,D0
        BLE.S   @1
        ADDQ.W  #7,D0
@1      bsr5    Out5Char
        move.w  #' ',d0         ;Space between displays
        bsr5    Out5Char
        rts6
;
;---------------------------------------------------------------------------
;
; Displays the long word value currently in d5
;
Dsp6LHex
        move.w  #7,d1           ;Counter for digits displayed
@1      rol.l   #4,d5
        move.b  d5,d0
        ANDI.B  #$0F,D0
        ORI.B   #$30,D0
        CMPI.B  #$39,D0
        BLE.S   @2
        ADDQ.W  #7,D0
@2      bsr5    Out5Char
        dbra    d1,@1
        move.w  #' ',d0
        bsr5    Out5Char
        rts6
;
;---------------------------------------------------------------------------
;
; Change from Hex to ASCII and pass out to SCC
; both PUT6HEX and Out5Char only use D0
;
PUT6HEX ANDI.B  #$0F,D0
        ORI.B   #$30,D0
        CMPI.B  #$39,D0
        BLE.S   @0
        ADDQ.W  #7,D0
@0      bsr5    Out5Char
        rts6
;
;---------------------------------------------------------------------------
;
; This routine looks for a value, for a count
; Inputs  -  d6 = last character read in
; Outputs -  d4.w = count
;
GetCount
        clr.w   d4
        cmp.b   #$0D,d6         ;Was last stroke a CR?
        beq     @2              ;...yes, use default count
;
; Get a count value, of bytes
@1      bsr6    Get6Char        ;Get next character of address
        cmp.b   #' ',d6         ;Was it a space?
        beq     @1              ;...yes, skip it
        cmp.b   #$0D,d6         ;Was it a CR?
        beq     @2              ;...yes, got count
        move.b  d6,d0           ;Position Ascii for conversion
        bsr6    GET6HEX         ;...convert Ascii to hex
        rol.l   #4,d4           ;...make room for it
        or.b    d0,d4           ;...place in D5 for use in A2
        bra     @1
;
@2      cmp.w   #0,d4
        bne     @12
        swap    d4              ;Get default in upper
        move.w  d4,d3
        beq     @11
        swap    d4
        move.w  d3,d4
@12     rts7
;
@11     move.w  #18,d4
        rts7
;
;---------------------------------------------------------------------------
;
; This routine looks for a value to write
; Inputs  -  d6 = last character read in
; Outputs -  d4.l = value
;
GetValue
        clr.l   d4
        cmp.b   #$0D,d6         ;Was last stroke a CR?
        beq     @12             ;...yes, use default value of zero
;
; Get a count value, of bytes
@1      bsr6    Get6Char        ;Get next character of address
        cmp.b   #' ',d6         ;Was it a space?
        beq     @1              ;...yes, skip it
        cmp.b   #$0D,d6         ;Was it a CR?
        beq     @12             ;...yes, got count
        move.b  d6,d0           ;Position Ascii for conversion
        bsr6    GET6HEX         ;...convert Ascii to hex
        rol.l   #4,d4           ;...make room for it
        or.b    d0,d4           ;...place in D5 for use in A2
        bra     @1
;
@12     rts7
```

```
;
;
;---------------------------------------------------------------------------
;
; This routine looks for a starting address
; Inputs  -  d6 = last character read in
; Outputs -  a2 = address to read from
;
GetAddress
        cmp.b   #$0D,d6         ;Was it just a CR?
        beq     @10             ;...yes, so skip check for address
@0      bsr6    Get6Char        ;Get first character of address
        cmp.b   #$0D,d6         ;Was it just a CR?
        beq     @10             ;...yes, take current a2 as default
        cmp.b   #' ',d6         ;Was it a space?
        beq     @0              ;...ignore spaces
;
; Get an address value
        clr.l   d5              ;...no, get A2 ready for a new address
        move.l  d5,a2           ;Initialize read address
@1      move.b  d6,d0           ;Position Ascii for conversion
        bsr6    GET6HEX         ;...convert Ascii to hex
        rol.l   #4,d5           ;...make room for it
        or.b    d0,d5           ;...place in D5 for use in A2
        move.l  d5,a2           ;...save as current address wanted
        bsr6    Get6Char        ;Get next character of address
        cmp.b   #' ',d6         ;Was it a space?
        beq     @10             ;...yes, end of address
        cmp.b   #$0D,d6         ;Was it a CR?
        bne     @1              ;...no, go after some more
;
@10     cmp.b   #'B',d7         ;See if a byte instruction, allow any address
        beq     @20
        cmp.b   #'W',d7         ;See if a word instruction, allow any word address
        bne     @15
        move.l  a2,d5
        and.l   #$FFFFFFFE,d5   ;...isolate on a word boundary
        move.l  d5,a2
        bra     @20
;
@15     move.l  a2,d5
        and.l   #$FFFFFFFC,d5   ;...isolate on a long word boundary
        move.l  d5,a2
@20     rts7
;
;
;---------------------------------------------------------------------------
;
; D0 contains ASCII digit -> D0 number
;
GET6HEX
        ANDI.L  #$FF,D0         ;Mask d0 into a byte value
        CMPI.B  #$30,D0         ;...see if it is a number or not
        BLT     SYNTAX          ;......syntax error.
        CMPI.B  #$39,D0         ;...see if a straignt number
        BGT.S   @1              ;......no, go handle A to F
@0      ANDI.w  #$0F,D0         ;......yes, mask for it
        rts6
;
@1      SUBQ.B  #7,D0           ;drop A to : position for easy masking
        CMPI.B  #$3F,D0         ;See if larger than F
        BLE.S   @0
;
SYNTAX  bra     What            ;Syntax error

        .include  romdiag/RAMD.Text
        .include  romdiag/StuckBit.Text
        .include  romdiag/RAMA.Text
        .include  romdiag/MEMFixed.Text
        .include  romdiag/March1.Text
        .include  romdiag/March2.Text
        .include  romdiag/ByteInv.Text
        .include  romdiag/WordInv.Text
        .include  romdiag/LWordInv.Text
        .include  romdiag/VIA.Text
        .include  romdiag/SCC.Text
        .include  romdiag/Messages.Text
;
        .END
```

```
;
; File: HdweEqu.TEXT
; Function: This file equates symbols to magic numbers in the actual hardware.
;           This version is specific for the Yacc hardware, designed by D. North
;
; Edit date:    02/11/85 GRC,   Original.
;               02/19/85 twm,   Modified for different hardware (Yacc)
;================================================================================
;

XtraVIA         .EQU    $EDFF80             ; base address of General Purpose VIA
MacVIA          .EQU    $EDFF40             ; base address of MAC compatible VIA
MiscVia         .EQU    $EDFF20             ; base address of miscellaneuos VIA

ORB       .EQU      2*0         ;and offsets for the various registers
ORA       .EQU      2*1
DDRB      .EQU      2*2
DDRA      .EQU      2*3
T1CL      .EQU      2*4
T1CH      .EQU      2*5
T1LL      .EQU      2*6
T1LH      .EQU      2*7
T2CL      .EQU      2*8
T2CH      .EQU      2*9
SHR       .EQU      2*10
ACR       .EQU      2*11
PCR       .EQU      2*12
IFR       .EQU      2*13
IER       .EQU      2*14
ORAP      .EQU      2*15
VRegIncr  .EQU      $0002       ;Increment between VIA registers
;
;
SCCRBase        .EQU    $DEFFF8         ; SCC base read address
SCCWBase        .EQU    $DEFFF9         ; SCC base write address
;
SCCREAD         .EQU    4               ;Offset to data read for SCCRBase
SCCWrite        .EQU    4               ;Offset to data write for SCCWBase
WCtrl           .EQU    0               ;Offset to write control register
RCtrl           .EQU    0               ;Offset to read control register
;
AllSent         .EQU    0               ;SCC transmit buffer all sent
RxBF            .EQU    0               ;SCC receive buffer full
TxBE            .EQU    2               ;SCC transmit buffer empty

SCCReg9K        .equ    $09             ; scc register #9 (K for constant)
SCCRstK         .equ    $C0             ; WR9 <== 11000000B

;
;SCCReset        .EQU    $09FFFFF        ;Reset the SCC
;

IWMBase   .EQU      $00DDFFE1
PH0L      .EQU      2*0
PH0H      .EQU      2*1
PH1L      .EQU      2*2
PH1H      .EQU      2*3
PH2L      .EQU      2*4
PH2H      .EQU      2*5
PH3L      .EQU      2*6
PH3H      .EQU      2*7
MotorOff  .EQU      2*8
MotorOn   .EQU      2*9
Drive1    .EQU      2*10
Drive2    .EQU      2*11
Q6L       .EQU      2*12
Q6H       .EQU      2*13
Q7L       .EQU      2*14
Q7H       .EQU      2*15
;
;
RamStart        .equ    $00C80000       ;RAM starting address
RamEnd          .equ    $00C80100       ;RAM ending address
Stack           .equ    $000B3FFE       ;Stack is in RAM
;
IntMask         .equ    $2500           ;Normally $2700, $2100 for debugging
;
;================================
; TRAP AND ERROR VECTOR LOCATIONS
;================================;
BUSVCT    .EQU      $0008           ; G0, bus error
ADRVCT    .EQU      $000C           ; G0, address error
ILLVCT    .EQU      $0010           ; G1, illegal instruction
DIVVCT    .EQU      $0014           ; G2, divide by zero
CHKVCT    .EQU      $0018           ; G2, CHK instruction
TPVVCT    .EQU      $001C           ; G2, TRAPV instruction
PRVVCT    .EQU      $0020           ; G1, Privilege violation
TRCVCT    .EQU      $0024           ; G1, Trace in progress
AXXVCT    .EQU      $0028           ; future traps (emulator), Line 1010
FXXVCT    .EQU      $002C           ; future traps (emulator), Line 1111
UNIVCT    .EQU      $003C           ; G1, Un initialized interrupt vector
SPRVCT    .EQU      $0060           ; G1, Spurious interrupt
LEV1VCT   .EQU      $0064           ; G1, General Via
LEV2VCT   .EQU      $0068           ; G1, Misc Via
LEV3VCT   .EQU      $006C           ; G1, Mac Via
LEV4VCT   .EQU      $0070           ; G1,
LEV5VCT   .EQU      $0074           ; G1, SCC
LEV6VCT   .EQU      $0078           ; G1,
NMIVCT    .EQU      $007C           ; G1, NMI
;
;
;================================================================================
;
```

```
        .MACRO  BSR7
        lea     @907,a7         ;Get address to return to
        bra     %1              ;Branch to subroutine
@907
        .ENDM
;
;
        .MACRO  RTS7
        jmp     (a7)            ;return from subroutine, addr in A7
        .ENDM
;
;
        .MACRO  BSR6
        lea     @906,a6         ;Get address to return to
        bra     %1              ;Branch to subroutine
@906
        .ENDM
;
;
        .MACRO  RTS6
        jmp     (a6)            ;return from subroutine, addr in A6
        .ENDM
;
;
        .MACRO  BSR5
        lea     @905,a5         ;Get address to return to
        bra     %1              ;Branch to subroutine
@905
        .ENDM
;
;
        .MACRO  RTS5
        jmp     (a5)            ;return from subroutine, addr in A5
        .ENDM
;
        .MACRO  BSR4
        lea     @904,a4         ;Get address to return to
        bra     %1              ;Branch to subroutine
@904
        .ENDM
;
;
        .MACRO  RTS4
        jmp     (a4)            ;return from subroutine, addr in A4
        .ENDM

        .MACRO  SCCDelay        ; wait for the scc to be ready - 16 cycles
        move    d0, d0          ;4 cycles
        move    d0, d0          ;4 cycles
        move    d0, d0          ;4 cycles
        move    d0, d0          ;4 cycles
        .ENDM
```

```
;-------------------------------------------------------------------------
;
;    File:     CmdTable.text
; 2/19/85 -twm
;    Separate this table of commands out from main program
;
SYSCMDS
; Memory commands
            .ASCII      'DB'            ;Display memory, bytes
            .WORD       DCMD-SYSCMDS
            .ASCII      'DL'            ;Display memory, long words
            .WORD       DCMD-SYSCMDS
            .ASCII      'DM'            ;Display memory, long words
            .WORD       DCMD-SYSCMDS
            .ASCII      'DW'            ;Display memory, words
            .WORD       DCMD-SYSCMDS
            .ASCII      'LB'            ;Loop memory, byte
            .WORD       LCMD-SYSCMDS
            .ASCII      'LL'            ;Loop memory, long word
            .WORD       LCMD-SYSCMDS
            .ASCII      'LM'            ;Loop memory, long word
            .WORD       LCMD-SYSCMDS
            .ASCII      'LW'            ;Loop memory, word
            .WORD       LCMD-SYSCMDS
            .ASCII      'RB'            ;Read memory, byte
            .WORD       RCMD-SYSCMDS
            .ASCII      'RL'            ;Read memory, long word
            .WORD       RCMD-SYSCMDS
            .ASCII      'RM'            ;Read memory, long word
            .WORD       RCMD-SYSCMDS
            .ASCII      'RW'            ;Read memory, word
            .WORD       RCMD-SYSCMDS
            .ASCII      'SB'            ;Set (write) memory, byte
            .WORD       SCMD-SYSCMDS
            .ASCII      'SL'            ;Set (write) memory, long word
            .WORD       SCMD-SYSCMDS
            .ASCII      'SM'            ;Set (write) memory, long word
            .WORD       SCMD-SYSCMDS
            .ASCII      'SW'            ;Set (write) memory, word
            .WORD       SCMD-SYSCMDS
;
; Initialize debugger command
            .ASCII      'I*'            ;Initialize debugger
            .WORD       IxCMD-SYSCMDS
;
; Help command
            .ASCII      '?*'            ;Help
            .WORD       HelpCMD-SYSCMDS
;
; Diagnostics
            .ASCII      'DF'            ;Run all diagnostics till failure
            .WORD       DFCMD-SYSCMDS
            .ASCII      'RD'            ;Run all diagnostics once
            .WORD       RDCMD-SYSCMDS
;
; Memory diagnostics
            .ASCII      'DD'            ;Data line test
            .WORD       DDCMD-SYSCMDS
            .ASCII      'SC'            ;Stuck cell test
            .WORD       SCCMD-SYSCMDS
            .ASCII      'RA'            ;RAM Address line test
            .WORD       RACMD-SYSCMDS
            .ASCII      'RF'            ;RAM Fixed pattern test
            .WORD       RFCMD-SYSCMDS
            .ASCII      'M1'            ;RAM March 1 test
            .WORD       M1CMD-SYSCMDS
            .ASCII      'M2'            ;RAM March 2 test
            .WORD       M2CMD-SYSCMDS
            .ASCII      'BI'            ;RAM Byte Moving Inversions test
            .WORD       BICMD-SYSCMDS
            .ASCII      'WI'            ;RAM Word Moving Inversions test
            .WORD       WICMD-SYSCMDS
            .ASCII      'LI'            ;RAM Long Word Moving Inversions test
            .WORD       LICMD-SYSCMDS
;
            .ASCII      'MT'            ;Memory tests
            .WORD       MTCMD-SYSCMDS
            .ASCII      'MF'            ;Memory tests till failure
            .WORD       MFCMD-SYSCMDS
;
; VIA diagnostics
            .ASCII      'VT'            ;VIA tests
            .WORD       VTCMD-SYSCMDS
;
; SCC diagnostics
            .ASCII      'ST'            ;SCC tests
            .WORD       STCMD-SYSCMDS
;
; ? diagnostics
;
; Last space holder, terminates this list
            .ASCII      '??'
            .WORD       0
```

```
;
; Edit date: 02/12/85 GRC, Original.
; File: ROMSCC.TEXT
; Function: These routines allow the SCC to be used in a debug type mode
;           from ROM based code, no memory is used, not even the stack.
;
;=============================================================================
;
; initialization data for SCC: 9600 baud RS-232 async communication
;   using the baud rate generator; interrupt conditions unchanged
;
SCCinitdata:
        .byte    4,$4C          ; x16 clk, 2 stop bits, no parity
        .byte   $B,$D0          ; baud rate gen clk to receiver, transmitter
        .byte   $F,$00          ; No interrupts
        .byte   $E,$00          ; disable baud rate generator
        .byte   $C,$0A          ; set baud rate to 9600x16 baud, 0A
        .byte   $D,$00          ; "
        .byte   $E,$01          ; enable baud rate generator
        .byte   $A,$00          ; NRZ mode
        .byte    3,$C1          ; 8 bits/char recv, enable receiver
        .byte    5,$6A          ; 8 bits/char xmit, enable xmitter
ESCCinitdata .equ    *-SCCinitdata
;
;-----------------------------------------------------------------------------
;
; Uses  a0,a1,a3
;       d1
Init7SCC
        MOVE.L  #<SCCRBASE+2>,A0    ;initialize scc (point to channel A) read
        Move.B  #SCCReg9K, (A0)     ; point to WR9
        SCCDelay                    ;Delay
        Move.B  #SCCRstK, (A0)      ; the reset constant
        SCCDelay
        SCCDelay
;        MOVE.L  #SCCRBase+2,A0          ;Pointer to SCC base read address A
        MOVE.L  #SCCWBase+2,A1          ;...generate SCC write address
        LEA     SCCinitdata,A3          ;Get pointer to init data
        MOVE.B  SCCREAD(A0),D1          ;Read to make sure SCC is sync'ed up
        MOVE.W  #ESCCinitdata,D1        ;...get size of init data table
        BRA.S   @2                      ; delay for timing, too
;
@1      MOVE.B  (A3)+,WCtrl(A1)         ;Write from table to the SCC
@2      SCCDelay
        DBRA    D1,@1                   ;...do all of data table
;
;        move.w  #$B000,d0
;@3      mulu    #1, d0
;        dbra    d0,@3
;
        TST.B   SCCREAD(A0)             ; CLEAR RX BUFFER
        rts7
;
;-----------------------------------------------------------------------------
;
; WriteLine
; Used to output a string, out the SCC port.
; Inputs:       a0 - SCC read address
;               a1 - SCC write address
;               a3 - Pointer to data string, string terminates with 0
;               a6 - this routine called with BSR6
; Outputs:
;               None
;
Write6Line
        move.b  (a3)+,d0        ;Get the next byte to send out
        beq     @1              ;...terminate when see a 0
        bsr5    Out5Char        ;...else, send out the character
        bra     Write6Line      ;...loop until get terminator
;
@1      move.b  #$0D,d0         ;Send CR
        bsr5    Out5Char
        move.b  #$0A,d0         ;Send LF
        bsr5    Out5Char        ;***
;
        move.w  #$1000,d0
@2      mulu    #1,d0           ;delay
        dbra    d0,@2
        rts6
;
;
PWrite6Line
        move.b  (a3)+,d0        ;Get the next byte to send out
        beq     @1              ;...terminate when see a 0
        bsr5    Out5Char        ;...else, send out the character
        bra     PWrite6Line      ;...loop until get terminator
;
@1      rts6
;
;-----------------------------------------------------------------------------
;
Fail6   move.b  #'F',d0         ;Send F
        bsr5    Out5Char
        move.b  #'A',d0         ;Send A
        bsr5    Out5Char
        move.b  #'I',d0         ;Send I
        bsr5    Out5Char
        move.b  #'L',d0         ;Send L
        bsr5    Out5Char
        move.b  #'E',d0         ;Send E
        bsr5    Out5Char
        move.b  #'D',d0         ;Send D
        bsr5    Out5Char
```

```
        move.b  #' ',d0         ;Send
        bsr5    Out5Char
        rts6
;
;-------------------------------------------------------------------------
;
Pass6   move.b  #'P',d0         ;Send P
        bsr5    Out5Char
        move.b  #'A',d0         ;Send A
        bsr5    Out5Char
        move.b  #'S',d0         ;Send S
        bsr5    Out5Char
        move.b  #'S',d0         ;Send S
        bsr5    Out5Char
        move.b  #' ',d0         ;Send
        bsr5    Out5Char
        rts6
;
;-------------------------------------------------------------------------
;
Return6 move.b  #$0D,d0         ;Send CR
        bsr5    Out5Char
        move.b  #$0A,d0         ;Send LF
        bsr5    Out5Char        ;***
;
        move.w  #$F000,d0
@2      mulu    #1,d0           ;delay
        dbra    d0,@2
        rts6
;
;-------------------------------------------------------------------------
;
; OutChar
; Used to output a character, out the SCC port.
; Inputs:       d0 - Character to send
;               a0 - SCC read address
;               a1 - SCC write address
;               a5 - this routine called with BSR5
; Outputs:
;               None
; Uses:
;               d0 upper and lower words
;
Out5Char
        move.b  #$30,WCtrl(a1)  ;Reset any errors
@1      SCCDelay
        BTST    #TxBE,RCtrl(A0) ;...wait until the transmit buffer is empty
        BEQ.S   @1
;
        MOVE.B  d0,SCCWrite(A1) ;put the byte in the transmit buffer
;@2      move.b  #1,WCtrl(a1)    ;Access the correct register
;       SCCDelay
;       MOVE.B  RCtrl(A0),D0    ;...Get SCC register value
;       SCCDelay
;       BTST    #AllSent,D0     ;...wait until the transmit buffer is empty
;       BEQ.S   @2
        rts5
;
Out4Char
        move.b  #$30,WCtrl(a1)  ;Reset any errors
@1      SCCDelay
        BTST    #TxBE,RCtrl(A0) ;...wait until the transmit buffer is empty
        BEQ.S   @1
;
        MOVE.B  d0,SCCWrite(A1) ;put the byte in the transmit buffer
;@2      move.b  #1,WCtrl(a1)    ;Access the correct register
;       SCCDelay
;       MOVE.B  RCtrl(A0),D0    ;...Get SCC register value
;       SCCDelay
;       BTST    #AllSent,D0     ;...wait until the transmit buffer is empty
;       BEQ.S   @2
        rts4
;
;
;-------------------------------------------------------------------------
;
;Get first character of next command, convert to upper case.
;Return that character to the terminal.
;
Get6Char
        bsr5    In5Char         ;Get the next character in d1
        and.w   #$7F,d6         ;...mask for ascii code
        cmp.b   #$61,d6         ;...see if lower case
        bmi     @1              ;.....no, so skip it
        sub.w   #$20,d6         ;.....yes, change lowercase to uppercase.
@1      move.b  d6,d0
        bsr5    Out5Char        ;Return the code to the caller
        rts6
;
;-------------------------------------------------------------------------
;
; InChar
; Used to input a character, from the SCC port.
; Inputs:       a0 - SCC read address
;               a1 - SCC write address
;               a5 - this routine called with BSR5
; Outputs:
;               d6.b - Character read.
; Uses:
;               d0 upper word,
;
In5Char
@1      MOVE.B  RCtrl(A0),D1    ;Get SCC register value for register 0
```

```
; Edit date: 02/11/85
;
; File: EXCEPTIONS.TEXT
; History: 02/11/85 GRC, Original.
;_____________________________________________________________________
;
; Exception handler setup.
;
; uses A4,A3,D1
;
SetExceptions
     lea       MISC,a4         ;initialize interrupt and trap vectors in case
     movea.l   #0,a3           ;...of exceptions
     move.w    #$FF,d1         ;...number of exceptions
@1   move.l    a4,(a3)+        ;First set all vectors to point to miscelaneous
;***     dbra      d1,@1
;
     lea       BUSERR,a4       ;Then set up special vectors for the more
     move.l    a4,BUSVCT       ;...common exceptions.
     lea       ADRERR,a4
     move.l    a4,ADRVCT
     lea       ILLERR,a4
     move.l    a4,ILLVCT
     lea       DIVERR,a4
     move.l    a4,DIVVCT
     lea       CHKERR,a4
     move.l    a4,CHKVCT
     lea       TRVERR,a4
     move.l    a4,TPVVCT
     lea       PRVERR,a4
     move.l    a4,PRVVCT
     lea       TRCERR,a4
;***     move.l    a4,TRCVCT
     lea       AXXERR,a4
     move.l    a4,AXXVCT
     lea       FXXERR,a4
     move.l    a4,FXXVCT
;
     lea       LEV1INT,a4
     move.l    a4,LEV1VCT
     lea       LEV2INT,a4
     move.l    a4,LEV2VCT
     lea       LEV3INT,a4
     move.l    a4,LEV3VCT
     lea       LEV4INT,a4
;****     move.l    a4,LEV4VCT
     lea       LEV5INT,a4
;***     move.l    a4,LEV5VCT
     lea       LEV6INT,a4
;***     move.l    a4,LEV6VCT
     lea       NMI,a4
;***     move.l    a4,NMIVCT
     lea       UNIERR,a4
     move.l    a4,UNIVCT
     lea       SPRERR,a4
     move.l    a4,SPRVCT
     rts6
;
;
;
;================================;
; DEFAULT EXCEPTION HANDLERS     ; these exception handlers just set an error
;================================;  code and exit . . .
MISC
        lea     EMISCMsg,a3     ;Miscellaneous exceptions (traps, divide by 0, ect).
        bra     Excp
BUSERR
        lea     EBUSSMsg,a3     ;Bus error exception
        bra     Excp0
ADRERR
        lea     EADRMsg,a3      ;Address error
        bra     Excp0
ILLERR
        lea     EILLMsg,a3      ;Illegal instruction error
        bra     Excp
DIVERR
        lea     EDIV0Msg,a3     ;Divide by 0.
        bra     Excp
CHKERR
        lea     ECHKMsg,a3      ;CHK instr
        bra     Excp
TRVERR
        lea     ETRAPVMsg,a3    ;TRAPV instr.
        bra     Excp
PRVERR
        lea     EPRIVMsg,a3     ;Privileged instr.
        bra     Excp
TRCERR
        lea     ETRACEMsg,a3    ;Trace instr.
        bra     Excp
AXXERR
        lea     E1010Msg,a3     ;Line 1010
        bra     Excp
FXXERR
        lea     E1111Msg,a3     ;Line 1111
        bra     Excp
UNIERR
        lea     EUIVMsg,a3      ;Uninitialized interrupt vector
        bra     Excp
SPRERR
        lea     ESpurMsg,a3     ;Spurious interrupt
        bra     Excp
;
```

```
LEV1INT
        lea     EL1Msg,a3       ;level 1 interrupt
        bra     Excp
LEV2INT
        lea     EL2Msg,a3       ;level 2 interrupt
        bra     Excp
LEV3INT
        lea     EL3Msg,a3       ;level 3 interrupt
        bra     Excp
LEV4INT
        lea     EL4Msg,a3       ;level 4 interrupt
        bra     Excp
LEV5INT
        lea     EL5Msg,a3       ;level 5 interrupt
        bra     Excp
LEV6INT
        lea     EL6Msg,a3       ;level 6 interrupt
        bra     Excp
NMI
        lea     EL7Msg,a3       ;level 7 (NMI) interrupt
        bra     Excp
;
;
EXCP0
     move.w     (sp)+,d0          ;throw away extra info for group 0
     move.l     (sp)+,d0
     move.w     (sp)+,d0
EXCP
     move.w     (sp)+,d0          ;throw away common exception info
     move.l     (sp)+,d0
     bra        ShowException     ;Go tell the Host about it
;
;
;_____________________________________________________________________
;
; Level 1, VIA (6522), interrupt handler.
;
ALEV1INT
     movem.l    d0-d1/a1-a2,-(sp)   ;save registers on interrupt entry
     move.l     #MacVIA,a2
     move.b     IFR(a2),d0      ;Get which interrupt flags are set.
     move.b     IER(a2),d1      ;Get which interrupts are enabled.
     and.b      d1,d0
;
@6   btst       #6,d0           ;See if Timer 1 flag
     beq        @0              ;...not timer if flag zero
     move.b     T1CL(a2),d0     ;Clear interrupt
;***      move.l     #T1CNT,a1
     add.w      #1,(a1)         ;...increment count
;
     move.b     #$50,T1CL(a2)   ;Load counter low   for 12ms interrupt
     move.b     #$34,T1CH(a2)   ;Load upper and start counter
;
;CA1 check
@0   btst       #1,d0           ;See if CA1, VSYNC.
     beq        @1              ;...not VSYNC if flag zero
     move.b     ORA(a2),d0      ;reset CA1 interrupt
;***      move.l     #CA1CNT,a1
     add.w      #1,(a1)         ;...increment count
     bra        @10
;
; CA2 check
@1   btst       #0,d0           ;See if CA2, 1 second clock.
     beq        @2              ;...not clock if flag zero
     move.b     ORA(a2),d0      ;reset CA2 interrupt
;***      move.l     #CA2CNT,a1
     add.w      #1,(a1)         ;...increment count
     bra        @10
;
; SR check
@2   btst       #2,d0           ;See if SR.
     beq        @3              ;...not shift register if flag zero
     move.b     SHR(a2),d0      ;reset sr interrupt
;***      move.l     #SRCNT,a1
     add.w      #1,(a1)         ;...increment count
     bra        @10
;
; CB2 check
@3   btst       #3,d0           ;See if CB2, keyboard data
     beq        @4              ;...not keyboard if flag zero
     move.b     ORB(a2),d0      ;reset CB2 interrupt
;***      move.l     #CB2CNT,a1
     add.w      #1,(a1)         ;...increment count
     bra        @10
;
@4   btst       #4,d0           ;See if CB1, keyboard clock
     beq        @5              ;...not clock if flag zero
     move.b     ORB(a2),d0      ;reset CB1 interrupt
;***      move.l     #CB1CNT,a1
     add.w      #1,(a1)         ;...increment count
     bra        @10
;
@5   btst       #5,d0           ;See if Timer 2 flag
     beq        @7              ;...not timer if flag zero
     move.b     T2CL(a2),d0     ;Clear interrupt
;***      move.l     #T2CNT,a1
     add.w      #1,(a1)         ;...increment count
     bra        @10
;
@7   move.b     #$7F,IER(a2)    ;turn off unknown 6522 interrupt
     move.b     #$7F,IFR(a2)    ;clear any pending interrupts

@10  movem.l    (sp)+,d0-d1/a1-a2   ;restore regs on interrupt exit
```

```
     rte
;
;_____________________________________________________________________________
;
; Level 2, SCC (8530), interrupt handler.
;
AALEV2INT
     movem.l   d0-d7/a0-a6,-(sp)  ;Save everyone
;***      move.l    #AWControl,a2  ;Get address of control register, port A write
;***      move.l    #ARControl,a3  ;Get address of control register, port A read
     move.w    #$03,d1         ;Read register #3
     move.b    d1,(a2)         ;Access correct register
     move.l    (sp),(sp)       ;Delay
     swap      d1
     move.b    (a3),d1         ;Read data from register
     move.l    (sp),(sp)       ;Delay
     and.w     #$00FF,d1       ;...mask read data so lower word is clean
     btst      #0,d1           ;Channel B?
     bne       @1
     btst      #3,d1           ;Channel A?
     bne       @2
     move      #IntMask,sr       ;disable all interrupts
     bra       @10
;
;
@1
;***   move.l    #BPORTEXT,a5
     add.w     #1,(a5)
;***      move.l    #BWControl,a2  ;Get address of control register, port B write
     bra       @10
;
@2
;***   move.l    #APORTEXT,a5
     add.w     #1,(a5)
     bra       @10
;
@10  move.b    #$10,(a2)       ; 0,Reset external status interrupts
     movem.l   (sp)+,d0-d7/a0-a6   ;Restore the world
     rte
;
;_____________________________________________________________________________
;
ShowException
        bsr6    Return6         ;Send CR
        bsr6    PWrite6Line     ;Write message out the SCC
        bsr6    Return6         ;Send CR
        bra     Initialize
```

```
;
; Edit date: 02/11/85 GRC, Original.
; File: Help.TEXT
; Function: This is by the ROM based debugger, it uses only CPU registers and
;           the I/O space of the SCC.  No memory or stack is required.
;           This command displays all the possible commands available.
;
;=============================================================================
;=============================================================================
;
;-----------------------------------------------------------------------------
;
; This command displays available commands
;
HelpCMD
        bsr6    Return6         ;Send CR
        lea     Help1Msg,a3     ;Point to message
        bsr6    Write6Line      ;...write it out the SCC
        lea     Help2Msg,a3     ;Point to message
        bsr6    Write6Line      ;...write it out the SCC
        lea     Help3Msg,a3     ;Point to message
        bsr6    Write6Line      ;...write it out the SCC
        lea     Help4Msg,a3     ;Point to message
        bsr6    Write6Line      ;...write it out the SCC
        lea     Help5Msg,a3     ;Point to message
        bsr6    Write6Line      ;...write it out the SCC
        lea     Help6Msg,a3     ;Point to message
        bsr6    Write6Line      ;...write it out the SCC
        lea     Help7Msg,a3     ;Point to message
        bsr6    Write6Line      ;...write it out the SCC
        lea     Help8Msg,a3     ;Point to message
        bsr6    Write6Line      ;...write it out the SCC
        lea     Help9Msg,a3     ;Point to message
        bsr6    Write6Line      ;...write it out the SCC
        lea     Help10Msg,a3    ;Point to message
        bsr6    Write6Line      ;...write it out the SCC
        lea     Help11Msg,a3    ;Point to message
        bsr6    Write6Line      ;...write it out the SCC
        lea     Help12Msg,a3    ;Point to message
        bsr6    Write6Line      ;...write it out the SCC
        lea     Help13Msg,a3    ;Point to message
        bsr6    Write6Line      ;...write it out the SCC
        lea     Help14Msg,a3    ;Point to message
        bsr6    Write6Line      ;...write it out the SCC
        lea     Help15Msg,a3    ;Point to message
        bsr6    Write6Line      ;...write it out the SCC
        lea     Help16Msg,a3    ;Point to message
        bsr6    Write6Line      ;...write it out the SCC
        lea     Help17Msg,a3    ;Point to message
        bsr6    Write6Line      ;...write it out the SCC
        lea     Help18Msg,a3    ;Point to message
        bsr6    Write6Line      ;...write it out the SCC
        lea     Help19Msg,a3    ;Point to message
        bsr6    Write6Line      ;...write it out the SCC
        lea     Help20Msg,a3    ;Point to message
        bsr6    Write6Line      ;...write it out the SCC
        lea     Help21Msg,a3    ;Point to message
        bsr6    Write6Line      ;...write it out the SCC
        lea     Help22Msg,a3    ;Point to message
        bsr6    Write6Line      ;...write it out the SCC
        bra     TopCommand
;
;-----------------------------------------------------------------------------
;
Help1Msg
;                1234567890123456789
        .ascii  'Initialize         I'
        .byte   0,0
Help2Msg
;                12345678901234567890123456789012345678901234567890123456789012345678901234
        .ascii  'Display memory     DB  DW  DL/DM   form: DB addr  cnt '
        .byte   0,0
Help3Msg
;                123456789012345678901234567890123456789012345678
        .ascii  'Read memory        RB  RW  RL/DM   form: RB addr'
        .byte   0,0
Help4Msg
;                12345678901234567890123456789012345678901234567890123456789012345678901234
        .ascii  'Set memory         SB  SW  SL/DM   form: SB addr value'
        .byte   0,0
Help5Msg
;                123456789012345678901234567890123456789012345678
        .ascii  'Loop forever       LB  LW  LL/LM   form: LB addr'
        .byte   0,0
Help6Msg
;                12345678901234567890123456789012345678901234567890123456789012345678901234567890
        .ascii  '             does  Write FF, Write 00, Read, Loop. '
        .byte   0,0
Help7Msg
;                123456789012
        .ascii  'DIAGNOSTICS '
        .byte   0,0
Help8Msg
;                12345678901234567890123456789012345678901234567890123456789012345678901234
        .ascii  'Diag till failure  DF              form: DF '
        .byte   0,0
Help9Msg
;                12345678901234567890123456789012345678901234567890123456789012345678901234
        .ascii  'Run Diagnostics(1) RD              form: RD '
        .byte   0,0
Help10Msg
;                123456789012345678901234567890123456789012345678901234567890123456789012345678
        .ascii  'Data lines         DD              form: DD addr'
```

```
        .byte   0,0
Help11Msg
;                1234567890123456789012345678901234567890123456789012345678901234
        .ascii  'Stuck cell          SC              form: SC '
        .byte   0,0
Help12Msg
;                1234567890123456789012345678901234567890123456789012345678901234
        .ascii  'RAM Address         RA              form: RA '
        .byte   0,0
Help13Msg
;                1234567890123456789012345678901234567890123456789012345678901234
        .ascii  'RAM Fixed pattern   RF              form: RF '
        .byte   0,0
Help14Msg
;                1234567890123456789012345678901234567890123456789012345678901234
        .ascii  'RAM March1 pattern  M1              form: M1 '
        .byte   0,0
Help15Msg
;                1234567890123456789012345678901234567890123456789012345678901234
        .ascii  'RAM March2 pattern  M2              form: M2 '
        .byte   0,0
Help16Msg
;                1234567890123456789012345678901234567890123456789012345678901234
        .ascii  'RAM Byte Mov Inv    BI              form: BI '
        .byte   0,0
Help17Msg
;                1234567890123456789012345678901234567890123456789012345678901234
        .ascii  'RAM Word Mov Inv    WI              form: WI '
        .byte   0,0
Help18Msg
;                1234567890123456789012345678901234567890123456789012345678901234
        .ascii  'RAM LWord Mov Inv   LI              form: LI '
        .byte   0,0
Help19Msg
;                1234567890123456789012345678901234567890123456789012345678901234
        .ascii  'Mem till failure    MF              form: MF '
        .byte   0,0
Help20Msg
;                1234567890123456789012345678901234567890123456789012345678901234
        .ascii  'Memory Tests        MT              form: MT '
        .byte   0,0
Help21Msg
;                1234567890123456789012345678901234567890123456789012345678901234
        .ascii  'Via Tests           VT              form: VT '
        .byte   0,0
Help22Msg
;                1234567890123456789012345678901234567890123456789012345678901234
        .ascii  'SCC Tests           ST              form: ST '
        .byte   0,0
;-------------------------------------------------------------------------------
```

```
;
; Edit date: 02/11/85 GRC, Original.
; File: MEMCommands.TEXT
; Function: This is by the ROM based debugger, it uses only CPU registers and
;           the I/O space of the SCC.  No memory or stack is required.
;           These commands do the memory direct commands, such as memory
;           display, read, and set.
;
;================================================================================
;================================================================================
;
;  Register usage:
;
;       a0 - SCC read address           d0 - Scratch
;       a1 - SCC write address          d1 - Scratch
;       a2 - Address looking at         d2 - Scratch
;       a3 - Scratch                    d3 - Scratch
;       a4 -                            d4 - Scratch
;       a5 - Subroutine calls           d5 -
;       a6 - Subroutine calls           d6 - Last byte read in
;       a7 - Subroutine calls           d7 - Command, building, lower word
;                                            Last command executed, upper word
;
;================================================================================
;--------------------------------------------------------------------------------
;
; Display memory, bytes, words, and long words.
;
DCMD    cmp.b   #$0D,d6         ;get here by a CR?
        bne     @0
        swap    d4              ;Get last count used
        move.w  d4,d3           ;...save it in d3
        swap    d4              ;...put last count back
        move.w  d3,d4           ;...place last count in current count
        bra     @1
;
@0      bsr7    GetAddress      ;Get address to read from
        bsr7    GetCount        ;...get a count value
        sub.w   #1,d4
        move.w  d4,d3
        swap    d4
        move.w  d3,d4
;
; a2 - address to read from.  d4 - byte count to display
;
@1      move.w  #17,d2          ;Count for number of bytes per line
        cmp.b   #'B',d7         ;..see if byte command
        beq     @2
        move.w  #9,d2           ;Count for number of words per line
        cmp.b   #'W',d7         ;..see if word command
        beq     @2
        move.w  #5,d2           ;Count for number of long words per line
@2      move.l  a2,d5           ;Display memory address
        bsr7    DspLHex
        move.w  #'>',d0
        bsr5    Out5Char
        move.w  #' ',d0
        bsr5    Out5Char
;
@3      cmp.b   #'B',d7         ;..see if byte command
        beq     @4
        cmp.b   #'W',d7         ;..see if word command
        beq     @5
        move.l  (a2)+,d5        ;Read memory
        bsr7    DspLHex
        sub.w   #4,d4           ;...Decrement byte displayed counter
        bra     @6
@4      move.b  (a2)+,d5        ;Read memory
        bsr7    DspBHex
        sub.w   #1,d4           ;...Decrement byte displayed counter
        bra     @6
@5      move.w  (a2)+,d5        ;Read memory
        bsr7    DspWHex
        sub.w   #2,d4           ;...Decrement byte displayed counter
@6      bmi     @10
        dbra    d2,@3           ;...loop until this line is complete
;
@10     bsr6    Return6         ;Send CR
        cmp.w   #0,d4
        beq     TopCommand
        bpl     @1
        bra     TopCommand
;
;--------------------------------------------------------------------------------
;
; Loop memory, byte, word, and long word.
;
LCMD
        cmp.b   #$0D,d6         ;get here by a CR?
        beq     @1
;
@0      bsr7    GetAddress      ;Get address to read from
        cmp.b   #$0D,d6         ;was last stroke a CR?
        beq     @1
        bsr6    Return6         ;Send CR
;
; a2 - address to read from.
;
@1      move.l  a2,d5           ;Read memory address
        bsr7    DspLHex
        move.w  #'>',d0
        bsr5    Out5Char
        move.w  #' ',d0
```

```
        bsr5    Out5Char
;
@3      cmp.b   #'B',d7         ;..see if byte command
        beq     @4
        cmp.b   #'W',d7         ;..see if word command
        beq     @5
@33     move.l  #$FFFFFFFF,(a2)
        clr.l   (a2)
        move.l  (a2),d5         ;Read memory
        bra     @33
;
@4      move.b  #$FF,(a2)
        clr.b   (a2)
        move.b  (a2),d5         ;Read memory
        bra     @4
;
@5      move.w  #$FFFF,(a2)
        clr.w   (a2)
        move.w  (a2),d5         ;Read memory
        bra     @5
;
;-------------------------------------------------------------------------
;
; Read memory, byte, word, and long word.
;
RCMD
        cmp.b   #$0D,d6         ;get here by a CR?
        beq     @1
;
@0      bsr7    GetAddress      ;Get address to read from
        cmp.b   #$0D,d6         ;was last stroke a CR?
        beq     @1
        bsr6    Return6         ;Send CR
;
; a2 - address to read from.
;
@1      move.l  a2,d5           ;Read memory address
        bsr7    DspLHex
        move.w  #'>',d0
        bsr5    Out5Char
        move.w  #' ',d0
        bsr5    Out5Char
;
@3      cmp.b   #'B',d7         ;..see if byte command
        beq     @4
        cmp.b   #'W',d7         ;..see if word command
        beq     @5
        move.l  (a2),d5         ;Read memory
        bsr7    DspLHex
        bra     @6
@4      move.b  (a2),d5         ;Read memory
        bsr7    DspBHex
        bra     @6
@5      move.w  (a2),d5         ;Read memory
        bsr7    DspWHex
@6      bsr6    Return6         ;Send CR
        bra     TopCommand
;
;-------------------------------------------------------------------------
;
; Set (write) memory, bytes, words, and long words.
;
SCMD    cmp.b   #$0D,d6         ;get here by a CR?
        bne     @0
        clr.l   d4              ;Make it set to zero
        bra     @1
;
@0      bsr7    GetAddress      ;Get address to write to
        clr.l   d4
        bsr7    GetValue        ;...get a count value
;
; a2 - address to read from.  d4 - value to write
;
@1      move.l  a2,d5           ;Display memory address
        bsr7    DspLHex
        move.w  #'<',d0
        bsr5    Out5Char
        move.w  #' ',d0
        bsr5    Out5Char
;
@3      cmp.b   #'B',d7         ;..see if byte command
        beq     @4
        cmp.b   #'W',d7         ;..see if word command
        beq     @5
        move.l  d4,(a2)         ;Write memory
        move.l  d4,d5
        bsr7    DspLHex
        bra     @6
@4      move.b  d4,(a2)         ;Write memory
        move.b  d4,d5
        bsr7    DspBHex
        bra     @6
@5      move.w  d4,(a2)         ;Write memory
        move.w  d4,d5
        bsr7    DspWHex
@6      bsr6    Return6         ;Send CR
        bra     TopCommand
;
```

```
;
; Edit date: 02/05/85 GRC, Original.
; File: RAMD.TEXT
; Function: This is by the ROM based debugger, it uses only CPU registers and
;           the I/O space of the SCC.  No memory or stack is required.
;           This command checks the data lines to the RAM, this test uses
;           only one location.
;
;   Using only one address, one word wide (all 16 data bits).  Write data and
;   read it back.  Do an EOR on written and read, OR it into error mask. An
;   Error points to either:
;       a) Bad data line.
;       b) Bad memory chip.
;   Numerous data patterns are used:
;       a) All ones.
;       b) All zeros.
;       c) $5555,$AAAA,$55AA,$AA55
;       d) Walking one.
;
;===============================================================================
;===============================================================================
;
;-------------------------------------------------------------------------------
;
; This command displays available commands
;
; Data line DIAGNOSTIC
;
DDCMD   cmp.b   #$0D,d6         ;get here by a CR?
        bne     @0
        clr.l   d4              ;Make it set to zero
        bra     @1
;
@0      bsr7    GetAddress      ;Get address to write to
;
; a2 - address to read from.
;
@1
        bsr7    BasicDataLine
        bra     TopCommand
;
BasicDataLine
        lea     DLHeadMsg,a3    ;Point to message
        bsr6    PWrite6Line     ;...write it out the SCC
;
        move.l  a2,d5           ;Display memory address
        bsr6    Dsp6LHex
        move.w  #'<',d0
        bsr5    Out5Char
        move.w  #' ',d0
        bsr5    Out5Char
;
; 1. Basic Data line test. A2 contains memory start location.
;

        clr.l   d5              ;ERROR mask
        move.l  #0,(a2)         ;Write test location to zero
;
        move.w  #5,d3
        lea     DataPattern,a3
@1      move.l  (a3)+,d1        ;Set pattern
        move.l  d1,(a2)         ;Write pattern
        move.l  (a2),d0         ;...read pattern
        eor.l   d0,d1           ;...get error bits
        or.l    d1,d5           ;...accumulate error bits
        dbra    d3,@1
;
        move.w  #31,d3
        move.l  #1,d4
@2      move.l  d4,d1           ;Walking pattern
        move.l  d1,(a2)         ;Write pattern
        move.l  (a2),d0         ;...read pattern
        eor.l   d0,d1           ;...get error bits
        or.l    d1,d5           ;...accumulate error bits
        lsl.l   #1,d4           ;Walk pattern left
        dbra    d3,@2
;
        cmp.l   #0,d5           ;Analysis of failures
        beq     @10
;
; Displays the long word value currently in d5
;
        bsr6    Dsp6LHex
;
        lea     ErrBitsMsg,a3    ;Point to message
        bsr6    Write6Line      ;...write it out the SCC
        move.w  #$FFFF,d6       ;Set the error flag
        bra     @20
;
@10     bsr6    Pass6           ;Send PASS message
        clr.w   d6              ;Clear the error flag
        bsr6    Return6         ;Send CR
@20     rts7
;
DataPattern
     .LONG      $FFFFFFFF,$00000000,$55555555,$AAAAAAAA,$55AA55AA,$A5A5A5A5
;
;-------------------------------------------------------------------------------
```

and.l #00FF00FF,d5

```
;
; Edit date: 02/05/85 GRC, Original.
; File: StuckBit.TEXT
; Function: This is by the ROM based debugger, it uses only CPU registers and
;           the I/O space of the SCC.  No memory or stack is required.
;           This command checks the RAMs for stuck bits, this test uses
;           all locations.
;
;
;===============================================================================
;===============================================================================
;
;-------------------------------------------------------------------------------
;
; This command displays available commands
;
; Stuck cell DIAGNOSTIC
;
SCCMD
        bsr6    Return6         ;Send CR
;
        bsr7    CellTest
;
        clr.l   d4
        bra     TopCommand
;
;
CellTest
        lea     SCHeadMsg,a3    ;Point to message
        bsr6    PWrite6Line      ;...write it out the SCC
        move.l  #RAMStart,d5    ;Display start memory address
        move.l  d5,a2           ;Start location
        bsr6    Dsp6LHex
        move.b  #'t',d0
        bsr5    Out5Char
        move.b  #'o',d0
        bsr5    Out5Char
        move.b  #' ',d0
        bsr5    Out5Char
        move.l  #RamEnd,d5      ;Display end memory address
        move.l  d5,a3           ;Ending location
        bsr6    Dsp6LHex
        move.b  #'<',d0
        bsr5    Out5Char
        move.b  #' ',d0
        bsr5    Out5Char
;
;  RAM cell test.
;
        move.l  a2,a4           ;Starting memory location
        clr.l   d0              ;Pattern of zeros
@1      move.l  d0,(a4)+        ;...write to memory
        cmp.l   a4,a3           ;...at end?
        bne     @1
;
        clr.l   d5              ;Error mask
        move.l  a2,a4           ;Starting memory location
@2      move.l  (a4)+,d1        ;...read pattern
        eor.l   d0,d1           ;...get error bits
        or.l    d1,d5           ;...accumulate error bits
        cmp.l   a4,a3           ;...at end?
        bne     @2
;
        move.l  #$FFFFFFFF,d0   ;Pattern of ones
        move.l  a2,a4           ;Starting memory location
@3      move.l  d0,(a4)+        ;...write to memory
        cmp.l   a4,a3           ;...at end?
        bne     @3
;
        move.l  a2,a4           ;Starting memory location
@4      move.l  (a4)+,d1        ;...read pattern
        eor.l   d0,d1           ;...get error bits
        or.l    d1,d5           ;...accumulate error bits
        cmp.l   a4,a3           ;...at end?
        bne     @4
;
        cmp.l   #0,d5           ;Analysis of failures
        beq     @10
;
; Displays the long word value currently in d5
;
        bsr6    Dsp6LHex        ;Display bad data found
        lea     ErrBitsMsg,a3   ;Point to message
        bsr6    Write6Line      ;...write it out the SCC
        move.w  #$FFFF,d6       ;Set the error flag
        bra     @20
;
@10     bsr6    Pass6           ;Send PASS message
        clr.w   d6              ;Clear the error flag
        bsr6    Return6         ;Send CR
@20     rts7
;
;
;-------------------------------------------------------------------------------
;===============================================================================
```

```
;
; Edit date: 02/05/85 GRC, Original.
; File: RAMA.TEXT
; Function: This is by the ROM based debugger, it uses only CPU registers and
;           the I/O space of the SCC.  No memory or stack is required.
;           This command checks the RAMs for addressing, this test tests
;           all locations.
;   This is an address uniqueness test which also checks the address lines to
;   each chip.  Errors are for address line faults.
;
;=============================================================================
;=============================================================================
;
;-----------------------------------------------------------------------------
;
; RAM Address line DIAGNOSTIC
;
RACMD
        bsr6    Return6         ;Send CR
;
        bsr7    RAddrTest
;
        clr.l   d4
        bra     TopCommand
;
;
;
RAddrTest
        lea     ALHeadMsg,a3    ;Point to message
        bsr6    PWrite6Line     ;...write it out the SCC
        move.l  #RAMStart,d5    ;Display start memory address
        move.l  d5,a2           ;Start location
        bsr6    Dsp6LHex
        move.b  #'t',d0
        bsr5    Out5Char
        move.b  #'o',d0
        bsr5    Out5Char
        move.b  #' ',d0
        bsr5    Out5Char
        move.l  #RamEnd,d5      ;Display end memory address
        move.l  d5,a3           ;Ending location
        bsr6    Dsp6LHex
        move.b  #'<',d0
        bsr5    Out5Char
        move.b  #' ',d0
        bsr5    Out5Char
;
;  RAM cell test.
;
        clr.l   d3              ;Start pattern at 0
        move.l  a2,a4           ;Starting memory location
@1      move.l  d3,(a4)         ;Write memory to its own address
        add.l   #1,d3           ;...increment pattern
        cmp.l   a3,a4           ;...see if did all of memory
        bmi     @1              ;...loop until all of memory is done
;
        clr.l   d3              ;Start pattern at 0
        move.l  a2,a4           ;Starting memory location
@2      move.l  (a4),d4         ;Read memory and verify its own address
        cmp.l   d3,d4           ;...See if expected and actual match
        bne     @98             ;...exit if they do not match
        add.l   #1,d3           ;...increment pattern
        cmp.l   a3,a4           ;...see if did all of memory
        bmi     @2              ;...loop until all of memory is done
;
        clr.l   d3              ;Start pattern at zero
        move.l  a3,a4           ;Get ending location
@3      suba.l  #4,a4           ;Next location down
        cmp.l   a4,a2           ;...if at bottom address then exit
        beq     @4              ;...must match exactly
        move.l  d3,(a4)         ;Write memory to an address pattern
        add.l   #1,d3           ;...increment pattern
        bra     @3              ;...loop until all of memory is done
;
@4      clr.l   d3              ;Start pattern at zero
        move.l  a3,a4           ;Get ending location
@5      suba.l  #4,a4           ;Next location down
        cmp.l   a4,a2           ;...if at bottom address then exit
        beq     @10             ;...must match exactly
        move.l  (a4),d4         ;Read memory
        cmp.l   d3,d4           ;...compare expected to actual
        bne     @99             ;...if not equal then exit with a fail
        add.l   #1,d3           ;...increment pattern
        bra     @5              ;...loop until all of memory is done
;
;
; Displays the long word value currently in d5
@98     sub.l   #4,a4           ;Failing memory location
        move.l  a4,d5
        bsr6    Dsp6LHex
;
        lea     FExpMsg,a3      ;Point to message ' failed, exp = '
        bsr6    PWrite6Line     ;...write it out the SCC
;
        move.l  d3,d5           ;Get expected data
        bsr6    Dsp6LHex
;
        lea     ActMsg,a3       ;Point to message ' act = '
        bsr6    PWrite6Line     ;...write it out the SCC
;
        move.l  d4,d5           ;Get actual data
        bsr6    Dsp6LHex
```

```
;
        bra     @11
;
; Displays the long word value currently in d5
@99     move.l  a4,d5           ;Failing memory location
        bsr6    Dsp6LHex
;
        lea     FExpMsg,a3      ;Point to message ' failed, exp = '
        bsr6    PWrite6Line     ;...write it out the SCC
;
        move.l  d3,d5           ;Get expected data
        bsr6    Dsp6LHex
;
        lea     ActMsg,a3       ;Point to message ' act = '
        bsr6    PWrite6Line     ;...write it out the SCC
;
        move.l  d4,d5           ;Get actual data
        bsr6    Dsp6LHex
        move.w  #$FFFF,d6       ;...Set the error flag
        bra     @11
;
@10     bsr6    Pass6           ;Send PASS message
        clr.w   d6              ;...clear the error flag
;
@11     bsr6    Return6         ;Send CR
@20     rts7
;
;
;-------------------------------------------------------------------------
```

```
;
; Edit date: 02/05/85 GRC, Original.
; File: MEMFixed.TEXT
; Function: This is by the ROM based debugger, it uses only CPU registers and
;           the I/O space of the SCC.  No memory or stack is required.
;           This command checks the RAMs with fixed memory patterns, this test
;           does all locations.
;
;
;================================================================================
;================================================================================
;
;--------------------------------------------------------------------------------
;
; RAM fixed pattern DIAGNOSTIC
;
RFCMD
        bsr6    Return6         ;Send CR
;
        bsr7    RFixedTest
;
        clr.l   d4
        bra     TopCommand
;
RFixedTest
        lea     FPHeadMsg,a3    ;Point to message
        bsr6    PWrite6Line     ;...write it out the SCC
;
        move.l  #RAMStart,d5    ;Display start memory address
        move.l  d5,a2           ;Start location
        bsr6    Dsp6LHex
        move.b  #'t',d0
        bsr5    Out5Char
        move.b  #'o',d0
        bsr5    Out5Char
        move.b  #' ',d0
        bsr5    Out5Char
        move.l  #RamEnd,d5      ;Display end memory address
        move.l  d5,a3           ;Ending location
        bsr6    Dsp6LHex
        move.b  #'<',d0
        bsr5    Out5Char
        move.b  #' ',d0
        bsr5    Out5Char
;
;  RAM fixed pattern test.
;
        lea     parlst1,a5      ;point to beginning of parameter list.
;
;initialize to write data.
t1loop
        move.l  (a5)+,d1        ;Get pattern, 4 long words wide
        move.l  (a5)+,d2
        move.l  (a5)+,d3
        move.l  (a5)+,d4
        movea.l a2,a4           ;Starting memory location
        move.l  a3,d0           ;loop count = (d0 count x 16 = Memory size)
        sub.l   a2,d0           ;Get memory area size
        asr.l   #4,d0           ;...Divide by 16
; write data loop.
@1      move.l  d1,(a4)+        ;write first 4 bytes.
        move.l  d2,(a4)+        ;...write second 4 bytes.
        move.l  d3,(a4)+
        move.l  d4,(a4)+        ;...write last 4 bytes.
        sub.l   #1,d0
        bne     @1              ;...loop until all bytes written (16 bytes at a time).
;
;initialize to read data.
        clr.l   d6              ;Error table
        movea.l a2,a4           ;Starting memory location
        move.l  a3,d0           ;loop count = (d0 count x 16 = Memory size)
        sub.l   a2,d0           ;Get memory area size
        asr.l   #4,d0           ;...Divide by 16
;read data and record errors by bit position.
@2      move.l  (a4)+,d5
        eor.l   d1,d5          ;read 16 bytes.
        or.l    d5,d6
        move.l  (a4)+,d5
        eor.l   d2,d5
        or.l    d5,d6
        move.l  (a4)+,d5
        eor.l   d3,d5
        or.l    d5,d6
        move.l  (a4)+,d5
        eor.l   d4,d5
        or.l    d5,d6
        sub.l   #1,d0
        bne     @2              ;loop until all bytes read (16 bytes at a time).
;
        move.l  (a5),d1         ;Get pattern, 4 long words wide
        cmp.l   #$12345678,d1   ;...is it the ending flag?
        bne     T1loop          ;branch to continue testing.
;
        move.l  d6,d5           ;Get failing bits
        beq     @10
;
;
; Displays the long word value currently in d5
;
        bsr6    Dsp6LHex        ;Display bad data found
        lea     ErrBitsMsg,a3   ;Point to message
        bsr6    Write6Line      ;...write it out the SCC
        move.w  #$FFFF,d6       ;...set the error flag
```

```
        bra     @20
;
@10     bsr6    Pass6           ;Send PASS message
        clr.w   d6              ;...clear the fail flag
        bsr6    Return6         ;Send CR
@20     rts7
;
;
;
;parameter list for memory test 1.
;
parlst1
     .long     $01010101,$80808080,$FFFFFFFF,$01010101  ;16 bytes of data.
     .long     $80808080,$FFFFFFFF,$01010101,$80808080  ;16 bytes of data.
     .long     $ffffffff,$01010101,$80808080,$ffffffff
     .long     $8080ffff,$01018080,$ffff0101,$8080ffff
     .long     $ffff0101,$8080ffff,$01018080,$ffff0101
     .long     $a5a5a5a5,$5a5a5a5a,$a5a5a5a5,$5a5a5a5a
     .long     $55555555,$aaaaaaaa,$55555555,$aaaaaaaa
     .long     $aaaaaaaa,$55555555,$aaaaaaaa,$55555555
     .long     $ffffffff,$ffffffff,$ffffffff,$ffffffff
     .long     $00000000,$00000000,$00000000,$00000000
     .long     $12345678           ;end of parameter table, flag.
;
;-------------------------------------------------------------------------
;-------------------------------------------------------------------------
```

```
; Edit date: 02/05/85
;
; File: March1.text
; Function: Marching pattern
; History: 02/05/85 GRC, Original.
;
; Enter with:
;   a2 = starting address
;   a3 = ending address
;
M1CMD
        bsr6    Return6         ;Send CR
;
        bsr7    M1Test
;
        clr.l   d4
        bra     TopCommand
;
;
;
M1Test
        lea     M1HeadMsg,a3    ;Point to message
        bsr6    PWrite6Line     ;...write it out the SCC
        move.l  #RAMStart,d5    ;Display start memory address
        move.l  d5,a2           ;Start location
        bsr6    Dsp6LHex
        move.b  #'t',d0
        bsr5    Out5Char
        move.b  #'o',d0
        bsr5    Out5Char
        move.b  #' ',d0
        bsr5    Out5Char
        move.l  #RamEnd,d5      ;Display end memory address
        move.l  d5,a3           ;Ending location
        bsr6    Dsp6LHex
        move.b  #'<',d0
        bsr5    Out5Char
        move.b  #' ',d0
        bsr5    Out5Char
;
        move.w  #$FFFF,d6       ;Set the error flag in case get out early
        clr.l   d5              ;Clear error counter
;
        clr.l   d1              ;Background pattern
        move.l  #$FFFFFFFF,d2   ;Marching pattern
        bsr5    Walk2it
;
        clr.l   d2              ;Marching pattern
        move.l  #$FFFFFFFF,d1   ;Background pattern
        bsr5    Walk2it
;
        bsr6    Pass6           ;Send PASS message
        bsr6    Return6         ;Send CR
        clr.w   d6              ;...clear error flag
        rts7
;
; Enter with:
;   d1 = background pattern
;   d2 = marching pattern
;   a2 = starting address, preserved
;   a3 = ending address, preserved
;
Walk2it
     movea.l   a2,a4
;
@1   move.l    d1,(a4)+        ;Write to all x
     cmpa.l    a4,a3
     bne       @1
;
     movea.l   a2,a4
@2   move.l    (a4)+,d0        ;Make sure at all x
     cmp.l     d0,d1           ;Make sure at all x
     bne       @100
     cmpa.l    a4,a3
     bne       @2
;
     movea.l   a2,a4
@3   move.l    (a4),d0
     cmp.l     d0,d1           ;Read data and expect x
     bne       @101
     move.l    d2,(a4)         ;Write to all y
     move.l    (a4)+,d0        ;Read and expect y
     cmp.l     d0,d2
     bne       @102
     cmpa.l    a4,a3
     bne       @3
;
     movea.l   a2,a4
@5   move.l    (a4)+,d0        ;Make sure at all y
     cmp.l     d0,d2           ;Make sure at all y
     bne       @102
     cmpa.l    a4,a3
     bne       @5
;
@10  rts5
;
; Make failing address in a4, expect data in d3, actual data in d4
;
@100 suba.l    #4,a4      ;Failing Address
     move.l    d1,d3      ;Expect
     move.l    d0,d4      ;Actual
     bra       @200
;
```

0-F d3 + d4

and.l #$00FF00FF, d0
move.l d1(a1,d2), d3
cmp.l d0, d3

```
@101 move.l    d1,d3      ;Expect
     move.l    d0,d4      ;Actual
     bra       @200
;
@102 suba.l    #4,a4      ;Failing Address
     move.l    d2,d3      ;Expect
     move.l    d0,d4      ;Actual
;
; Displays the long word value currently in d5
@200    move.l  a4,d5           ;Failing location
        bsr6    Dsp6LHex
;
        lea     FExpMsg,a3      ;Point to message ' failed, exp = '
        bsr6    PWrite6Line     ;...write it out the SCC
;
        move.l  d3,d5           ;Get expected data
        bsr6    Dsp6LHex
;
        lea     ActMsg,a3       ;Point to message ' act = '
        bsr6    PWrite6Line     ;...write it out the SCC
;
        move.l  d4,d5           ;Get actual data
        bsr6    Dsp6LHex
        bsr6    Return6         ;Send CR
        rts7
;
```

```
; Edit date: 02/05/85
;
; File: March2.text
; Function: Marching pattern
; History: 02/05/85 GRC, Original.
;
; Enter with:
;   a2 = starting address
;   a3 = ending address
;
M2CMD
        bsr6    Return6         ;Send CR
;
        bsr7    M2Test
;
        clr.l   d4
        bra     TopCommand
;
;
;
M2Test
        lea     M2HeadMsg,a3    ;Point to message
        bsr6    PWrite6Line     ;...write it out the SCC
        move.l  #RAMStart,d5    ;Display start memory address
        move.l  d5,a2           ;Start location
        bsr6    Dsp6LHex
        move.b  #'t',d0
        bsr5    Out5Char
        move.b  #'o',d0
        bsr5    Out5Char
        move.b  #' ',d0
        bsr5    Out5Char
        move.l  #RamEnd,d5      ;Display end memory address
        move.l  d5,a3           ;Ending location
        bsr6    Dsp6LHex
        move.b  #'<',d0
        bsr5    Out5Char
        move.b  #' ',d0
        bsr5    Out5Char
;
        move.w  #$FFFF,d6       ;Set error flag in case get out early
        clr.l   d5              ;Clear error counter
;
        clr.l   d1              ;Background pattern
        move.l  #$FFFFFFFF,d2   ;Marching pattern
        bsr5    Walk3it
;
        clr.l   d2              ;Marching pattern
        move.l  #$FFFFFFFF,d1   ;Background pattern
        bsr5      Walk3it
;
        bsr6    Pass6           ;Send PASS message
        bsr6    Return6         ;Send CR
        clr.w   d6              ;Clear the error flag
        rts7
;
; Enter with:
;   d1 = background pattern
;   d2 = marching pattern
;   a2 = starting address, preserved
;   a3 = ending address, preserved
;
Walk3it
        movea.l a2,a4
@1      move.l  d1,(a4)+        ;Write to all x
        cmpa.l  a4,a3
        bne     @1
;
        movea.l a2,a4
@2      move.l  (a4)+,d0        ;Make sure at all x
        cmp.l   d0,d1           ;Make sure at all x
        bne     @100
        cmpa.l  a4,a3
        bne     @2
;
        movea.l a2,a4
@3      move.l  (a4),d0         ;Read data and expect x
        cmp.l   d0,d1           ;Read data and expect x
        bne     @101
        move.l  d2,(a4)+        ;Write to y
        cmpa.l  a4,a3
        bne     @3
;
        movea.l a2,a4
@4      move.l  (a4),d0         ;Read data and expect y
        cmp.l   (a4),d2         ;Read data and expect y
        bne     @102
        move.l  d1,(a4)+        ;Write to x
        cmpa.l  a4,a3
        bne     @4
;
;
;
        movea.l a3,a4
@5      move.w  d1,-(a4)        ;Write to all x
        cmpa.l  a4,a2
        bne     @5
;
        movea.l a3,a4
@6      move.l  -(a4),d0        ;Make sure at all x
        cmp.l   d0,d1           ;Make sure at all x
        bne     @103
        cmpa.l  a4,a2
```

6-F d3,d4

and.l #$00FF00FF,d0
move.l d1(a,d2)
a(a4), d3
and.l #$00FF00FF,d3
cmp d0, d3

```
        bne     @6
;
        movea.l a3,a4
@7      move.l  -(a4),d0        ;Read data and expect x
        cmp.l   d0,d1           ;Read data and expect x
        bne     @103
        move.l  d2,(a4)         ;Write to y
        cmpa.l  a4,a2
        bne     @7
;
        movea.l a3,a4
@8      move.l  -(a4),d0        ;Read data and expect y
        cmp.l   d0,d2           ;Read data and expect y
        bne     @104
        move.l  d1,(a4)         ;Write to x
        cmpa.l  a4,a2
        bne       @8
;
        movea.l a3,a4
@9      move.l  -(a4),d0        ;Read data and expect x
        cmp.l   d0,d1           ;Read data and expect x
        bne     @103
        cmpa.l  a4,a2
        bne     @9
;
@10     rts5
;
; Make failing address in a4, expect data in d3, actual data in d4
;
@100 suba.l     #4,a4       ;Failing Address
     move.l     d1,d3       ;Expect
     move.l     d0,d4       ;Actual
     bra        @200
;
@101 move.l     d1,d3       ;Expect
     move.l     d0,d4       ;Actual
     bra        @200
;
@102 move.l     d2,d3       ;Expect
     move.l     d0,d4       ;Actual
     bra        @200
;
@103 adda.l     #4,a4       ;Failing Address
     move.l     d1,d3       ;Expect
     move.l     d0,d4       ;Actual
     bra        @200
;
@104 adda.l     #4,a4       ;Failing Address
     move.l     d2,d3       ;Expect
     move.l     d0,d4       ;Actual
     bra        @200
;
; Displays the long word value currently in d5
@200    move.l  a4,d5           ;Failing location
        bsr6    Dsp6LHex
;
        lea     FExpMsg,a3      ;Point to message ' failed, exp = '
        bsr6    PWrite6Line     ;...write it out the SCC
;
        move.l  d3,d5           ;Get expected data
        bsr6    Dsp6LHex
;
        lea     ActMsg,a3       ;Point to message ' act = '
        bsr6    PWrite6Line     ;...write it out the SCC
;
        move.l  d4,d5           ;Get actual data
        bsr6    Dsp6LHex
        bsr6    Return6         ;Send CR
        rts7
;
;
```

```
;
; Edit date: 02/05/85 GRC, Original.
; File: ByteInv.TEXT
; Function: This is by the ROM based debugger, it uses only CPU registers and
;         the I/O space of the SCC.  No memory or stack is required.
;         This command checks the RAMs for addressing and data, this tests
;         all locations.
;
;
;================================================================================
;================================================================================
;
;--------------------------------------------------------------------
;
; RAM Byte Moving Inversions DIAGNOSTIC
;
BICMD
     bsr6   Return6        ;Send CR
;
     bsr7   BInvTest
;
     clr.l  d4
     bra    TopCommand
;
;
;
BInvTest
     lea    BIHeadMsg,a3   ;Point to message
     bsr6   PWrite6Line    ;...write it out the SCC
     move.l #RAMStart,d5   ;Display start memory address
     move.l d5,a2          ;Start location
     bsr6   Dsp6LHex
     move.b #'t',d0
     bsr5   Out5Char
     move.b #'o',d0
     bsr5   Out5Char
     move.b #' ',d0
     bsr5   Out5Char
     move.l #RamEnd,d5     ;Display end memory address
     move.l d5,a3          ;Ending location
     bsr6   Dsp6LHex
     move.b #'<',d0
     bsr5   Out5Char
     move.b #' ',d0
     bsr5   Out5Char
;
     move.l a2,a4          ;Starting address
@1   clr.b  (a4)+          ;set the memory to zero.
     cmp.l  a4,a3          ;...at end of memory
     bne    @1
;
; a4 = Working address        d1 = Address increment value
; a5 = Address increment
;
; TOP OF FORWARD LOOP
     clr.l  d1
     move.w #1,d1          ;Address increment, start at 1
@2   move.l #0,a5          ;Address offset value
;
; TOP OF ADDRESS INCREMENT LOOP
@3   move.l a2,a4          ;Starting address
     adda.l a5,a4          ;...plus offset
     cmp.l  a3,a4          ;See if out of bounds
     bpl    @10
     clr.b  d4             ;Expect pattern
;
; TOP OF WORKING ADDRESS LOOP
@5   move.b (a4),d3        ;Read
     cmp.b  d3,d4          ;...same as expected?
     bne    @100           ;...no, flag as error
     move.b #$FF,d3        ;Set the byte
     move.b d3,(a4)        ;...write, with test bits set
     move.b (a4),d5        ;...read again
     cmp.b  d5,d3          ;...same as written?
     bne    @101           ;...no, flag as error
     adda.l d1,a4
     cmp.l  a3,a4          ;At end of memory?
     bmi    @5
; BOTTOM OF WORKING ADDRESS LOOP
;
;
```

```
;
     move.b  #$FF,d4        ;Expect pattern
     move.l  a2,a4          ;Starting address
     adda.l  a5,a4          ;...plus offset
;
; TOP OF WORKING ADDRESS LOOP
@7   move.b  (a4),d3        ;Read
     cmp.b   d3,d4          ;...same as expected?
     bne     @100           ;...no, flag as error
     clr.b   d3             ;Clear the test byte
     move.b  d3,(a4)        ;...write, with test byte set
     move.b  (a4),d5        ;...read again
     cmp.b   d5,d3          ;...same as written?
     bne     @101           ;...no, flag as error
     adda.l  d1,a4
     cmp.l   a3,a4          ;At end of memory?
     bmi     @7
; BOTTOM OF WORKING ADDRESS LOOP
;
     adda.l  #1,a5          ;Next address base offset
     cmp.l   d1,a5          ;...done all offsets?
     bne     @3             ;...no, continue
; BOTTOM OF ADDRESS INCREMENT LOOP
;
     asl.l   #1,d1          ;New Address increment value
     cmp.l   #$20000,d1     ;...at end?
     bne     @2             ;...no, continue
; BOTTOM OF FORWARD LOOP

     bra     @10
;
@101 move.b  d3,d4          ;Expect data
     move.b  d5,d3          ;...same as written?
;
; Displays the long word value currently in d5
@100 move.l  a4,d5
     bsr6    Dsp6LHex
;
     lea     FExpMsg,a3     ;Point to message ' failed, exp = '
     bsr6    PWrite6Line    ;...write it out the SCC
;
     move.l  d4,d5          ;Get expected data
     bsr6    Dsp6BHex
;
     lea     ActMsg,a3      ;Point to message ' act = '
     bsr6    PWrite6Line    ;...write it out the SCC
;
     move.l  d3,d5          ;Get actual data
     bsr6    Dsp6BHex
     bra     @11
;
; Displays the long word value currently in d5
@99  move.l  a4,d5          ;Failing memory location
     bsr6    Dsp6LHex
;
     lea     FExpMsg,a3     ;Point to message ' failed, exp = '
     bsr6    PWrite6Line    ;...write it out the SCC
;
     move.l  d3,d5          ;Get expected data
     bsr6    Dsp6BHex
;
     lea     ActMsg,a3      ;Point to message ' act = '
     bsr6    PWrite6Line    ;...write it out the SCC
;
     move.l  d4,d5          ;Get actual data
     bsr6    Dsp6BHex
     move.w  #$FFFF,d6      ;...Set the error flag
     bra     @11
;
@10  bsr6    Pass6          ;Send PASS message
     clr.w   d6             ;...clear the error flag
;
@11  bsr6    Return6        ;Send CR
@20  rts7
;
;
;-------------------------------------------------------------------------
```

```
;
; Edit date: 02/05/85 GRC, Original.
; File: WordInv.TEXT
; Function: This is by the ROM based debugger, it uses only CPU registers and
;           the I/O space of the SCC.  No memory or stack is required.
;           This command checks the RAMs for addressing and data, this tests
;           all locations.
;
;
;=============================================================================
;=============================================================================
;
;-----------------------------------------------------------------------
;
; RAM Word Moving Inversions DIAGNOSTIC
;
WICMD
        bsr6    Return6         ;Send CR
;
        bsr7    WInvTest
;
        clr.l   d4
        bra     TopCommand
;
;
;
WInvTest
        lea     WIHeadMsg,a3    ;Point to message
        bsr6    PWrite6Line     ;...write it out the SCC
        move.l  #RAMStart,d5    ;Display start memory address
        move.l  d5,a2           ;Start location
        bsr6    Dsp6LHex
        move.b  #'t',d0
        bsr5    Out5Char
        move.b  #'o',d0
        bsr5    Out5Char
        move.b  #' ',d0
        bsr5    Out5Char
        move.l  #RamEnd,d5      ;Display end memory address
        move.l  d5,a3           ;Ending location
        bsr6    Dsp6LHex
        move.b  #'<',d0
        bsr5    Out5Char
        move.b  #' ',d0
        bsr5    Out5Char
;
        move.l  a2,a4           ;Starting address
@1      clr.w   (a4)+           ;set the memory to zero.
        cmp.l   a4,a3           ;...at end of memory
        bne     @1
;
; a4 = Working address          d1 = Address increment value
; a5 = Address increment
;
; TOP OF FORWARD LOOP
        clr.l   d1
        move.w  #2,d1           ;Address increment, start at 2
@2      move.l  #0,a5           ;Address offset value
;
; TOP OF ADDRESS INCREMENT LOOP
@3      move.l  a2,a4           ;Starting address
        adda.l  a5,a4           ;...plus offset
        cmp.l   a3,a4           ;See if out of bounds
        bpl     @10
        clr.w   d4              ;Expect pattern
;
; TOP OF WORKING ADDRESS LOOP
@5      move.w  (a4),d3         ;Read
        cmp.w   d3,d4           ;...same as expected?
        bne     @100            ;...no, flag as error
        move.w  #$FFFF,d3       ;Set the word
        move.w  d3,(a4)         ;...write, with test bits set
        move.w  (a4),d5         ;...read again
        cmp.w   d5,d3           ;...same as written?
        bne     @101            ;...no, flag as error
        adda.l  d1,a4
        cmp.l   a3,a4           ;At end of memory?
        bmi     @5
; BOTTOM OF WORKING ADDRESS LOOP
;
;
;
        move.w  #$FFFF,d4       ;Expect pattern
        move.l  a2,a4           ;Starting address
        adda.l  a5,a4           ;...plus offset
;
; TOP OF WORKING ADDRESS LOOP
@7      move.w  (a4),d3         ;Read
        cmp.w   d3,d4           ;...same as expected?
        bne     @100            ;...no, flag as error
        clr.w   d3              ;Clear the test word
        move.w  d3,(a4)         ;...write, with test word set
        move.w  (a4),d5         ;...read again
        cmp.w   d5,d3           ;...same as written?
        bne     @101            ;...no, flag as error
        adda.l  d1,a4
        cmp.l   a3,a4           ;At end of memory?
        bmi     @7
; BOTTOM OF WORKING ADDRESS LOOP
;
        adda.l #2,a5            ;Next address base offset
        cmp.l  d1,a5            ;...done all offsets?
        bne    @3               ;...no, continue
```

```
; BOTTOM OF ADDRESS INCREMENT LOOP
;
        asl.l   #1,d1           ;New Address increment value
        cmp.l   #$20000,d1      ;...at end?
        bne     @2              ;...no, continue
; BOTTOM OF FORWARD LOOP

        bra     @10
;
@101    move.w  d3,d4           ;Expect data
        move.w  d5,d3           ;...same as written?
;
; Displays the long word value currently in d5
@100    move.l  a4,d5
        bsr6    Dsp6LHex
;
        lea     FExpMsg,a3      ;Point to message ' failed, exp = '
        bsr6    PWrite6Line     ;...write it out the SCC
;
        move.l  d4,d5           ;Get expected data
        bsr6    Dsp6WHex
;
        lea     ActMsg,a3       ;Point to message ' act = '
        bsr6    PWrite6Line     ;...write it out the SCC
;
        move.l  d3,d5           ;Get actual data
        bsr6    Dsp6WHex
        bra     @11
;
; Displays the long word value currently in d5
@99     move.l  a4,d5           ;Failing memory location
        bsr6    Dsp6LHex
;
        lea     FExpMsg,a3      ;Point to message ' failed, exp = '
        bsr6    PWrite6Line     ;...write it out the SCC
;
        move.l  d3,d5           ;Get expected data
        bsr6    Dsp6WHex
;
        lea     ActMsg,a3       ;Point to message ' act = '
        bsr6    PWrite6Line     ;...write it out the SCC
;
        move.l  d4,d5           ;Get actual data
        bsr6    Dsp6WHex
        move.w  #$FFFF,d6       ;...Set the error flag
        bra     @11
;
@10     bsr6    Pass6           ;Send PASS message
        clr.w   d6              ;...clear the error flag
;
@11     bsr6    Return6         ;Send CR
@20     rts7
;
;
;-------------------------------------------------------------------------
```

```
;
; Edit date: 02/05/85 GRC, Original.
; File: LWordInv.TEXT
; Function: This is by the ROM based debugger, it uses only CPU registers and
;           the I/O space of the SCC.  No memory or stack is required.
;           This command checks the RAMs for addressing and data, this tests
;           all locations.
;
;
;=============================================================================
;=============================================================================
;
;-----------------------------------------------------------------------
;
; RAM Long Word Moving Inversions DIAGNOSTIC
;
LICMD
        bsr6    Return6         ;Send CR
;
        bsr7    LWInvTest
;
        clr.l   d4
        bra     TopCommand
;
;
;
LWInvTest
        lea     LIHeadMsg,a3    ;Point to message
        bsr6    PWrite6Line     ;...write it out the SCC
        move.l  #RAMStart,d5    ;Display start memory address
        move.l  d5,a2           ;Start location
        bsr6    Dsp6LHex
        move.b  #'t',d0
        bsr5    Out5Char
        move.b  #'o',d0
        bsr5    Out5Char
        move.b  #' ',d0
        bsr5    Out5Char
        move.l  #RamEnd,d5      ;Display end memory address
        move.l  d5,a3           ;Ending location
        bsr6    Dsp6LHex
        move.b  #'<',d0
        bsr5    Out5Char
        move.b  #' ',d0
        bsr5    Out5Char
;
        move.l  a2,a4           ;Starting address
@1      clr.L   (a4)+           ;set the memory to zero.
        cmp.l   a4,a3           ;...at end of memory
        bne     @1
;
; a4 = Working address          d1 = Address increment value
; a5 = Address increment
;
; TOP OF FORWARD LOOP
        clr.l   d1
        move.w  #4,d1           ;Address increment, start at 4
@2      move.l  #0,a5           ;Address offset value
;
; TOP OF ADDRESS INCREMENT LOOP
@3      move.l  a2,a4           ;Starting address
        adda.l  a5,a4           ;...plus offset
        cmp.l   a3,a4           ;See if out of bounds
        bpl     @10
        clr.l   d4              ;Expect pattern
;
; TOP OF WORKING ADDRESS LOOP
@5      move.l  (a4),d3         ;Read
        cmp.l   d3,d4           ;...same as expected?
        bne     @100            ;...no, flag as error
        move.l  #$FFFFFFFF,d3   ;Set the long word
        move.l  d3,(a4)         ;...write, with test bits set
        move.l  (a4),d5         ;...read again
        cmp.l   d5,d3           ;...same as written?
        bne     @101            ;...no, flag as error
        adda.l  d1,a4
        cmp.l   a3,a4           ;At end of memory?
        bmi     @5
; BOTTOM OF WORKING ADDRESS LOOP
;
;
;
        move.l  #$FFFFFFFF,d4   ;Expect pattern
        move.l  a2,a4           ;Starting address
        adda.l  a5,a4           ;...plus offset
;
; TOP OF WORKING ADDRESS LOOP
@7      move.l  (a4),d3         ;Read
        cmp.l   d3,d4           ;...same as expected?
        bne     @100            ;...no, flag as error
        clr.l   d3              ;Clear the test long word
        move.l  d3,(a4)         ;...write, with test word set
        move.l  (a4),d5         ;...read again
        cmp.l   d5,d3           ;...same as written?
        bne     @101            ;...no, flag as error
        adda.l  d1,a4
        cmp.l   a3,a4           ;At end of memory?
        bmi     @7
; BOTTOM OF WORKING ADDRESS LOOP
;
        adda.l #4,a5            ;Next address base offset
        cmp.l  d1,a5            ;...done all offsets?
        bne    @3               ;...no, continue
```

```
; BOTTOM OF ADDRESS INCREMENT LOOP
;
        asl.l   #1,d1           ;New Address increment value
        cmp.l   #$20000,d1      ;...at end?
        bne     @2              ;...no, continue
; BOTTOM OF FORWARD LOOP

        bra     @10
;
@101    move.l  d3,d4           ;Expect data
        move.l  d5,d3           ;...same as written?
;
; Displays the long word value currently in d5
@100    move.l  a4,d5
        bsr6    Dsp6LHex
;
        lea     FExpMsg,a3      ;Point to message ' failed, exp = '
        bsr6    PWrite6Line     ;...write it out the SCC
;
        move.l  d4,d5           ;Get expected data
        bsr6    Dsp6LHex
;
        lea     ActMsg,a3       ;Point to message ' act = '
        bsr6    PWrite6Line     ;...write it out the SCC
;
        move.l  d3,d5           ;Get actual data
        bsr6    Dsp6LHex
        bra     @11
;
; Displays the long word value currently in d5
@99     move.l  a4,d5           ;Failing memory location
        bsr6    Dsp6LHex
;
        lea     FExpMsg,a3      ;Point to message ' failed, exp = '
        bsr6    PWrite6Line     ;...write it out the SCC
;
        move.l  d3,d5           ;Get expected data
        bsr6    Dsp6LHex
;
        lea     ActMsg,a3       ;Point to message ' act = '
        bsr6    PWrite6Line     ;...write it out the SCC
;
        move.l  d4,d5           ;Get actual data
        bsr6    Dsp6LHex
        move.w  #$FFFF,d6       ;...Set the error flag
        bra     @11
;
@10     bsr6    Pass6           ;Send PASS message
        clr.w   d6              ;...clear the error flag
;
@11     bsr6    Return6         ;Send CR
@20     rts7
;
;
;-------------------------------------------------------------------------
```

```
; Edit date: 02/13/85
;
; File: VIA.TEXT
;
; Function: Included into the ROMTALK.TEXT program, tests the VIA (6522) and
;           most of the lines connected to it.
;           Note that these tests use the stack memory, exception routines
;           have to use the stack.
;
; History: 02/13/85 GRC, Original.
;
;_____________________________________________________________________________
;
; VIA tests.
;
; These are the basic tests of the VIA on the Main Logic board.
;_____________________________________________________________________________
;
; Register test, for reg ORB/DDRB/T1LL/T1LH/SR/ACR/PCR
;
;
VTCMD                           ; Via Test command entry point
        bsr6    Return6         ;Send CR
        bsr6    Via6Test
        clr.l   d4
        bra     TopCommand
;--
;       Run all the VIA tests and ignore any errors

Via6Test                                ; test all 3 VIA's
        move.l  a6, d7                  ; save the return address
        lea     VLEV1INT,a4
        move.l  a4,LEV3VCT              ; alter MAC compatable interrupt vector
        move.l  #MacVIA,a2              ; Pass address of Mac VIA
        lea     VRHead3Msg,a3           ;Point to message
        bsr4    VIAREGISTERS
        lea     VIHead3Msg,a3
        bsr4    VIAINTERRUPT
        lea     VT1Head3Msg,a3
        bsr4    VIA1TIMER
        lea     VT2Head3Msg,a3
        bsr4    VIA2TIMER

        lea     VLEV1INT,a4
        move.l  a4,LEV2VCT               ; level 2 interrupt vector
        move.l  #MiscVIA,a2              ; Pass address of Misc VIA
        lea     VRHead2Msg,a3           ;Point to message
        bsr4    VIAREGISTERS
        lea     VIHead2Msg,a3
        bsr4    VIAINTERRUPT
        lea     VT1Head2Msg,a3
        bsr4    VIA1TIMER
        lea     VT2Head2Msg,a3
        bsr4    VIA2TIMER

        lea     VLEV1INT,a4
        move.l  a4,LEV1VCT              ; level 1 interrupt vector
        move.l  #XtraVIA,a2             ; General purpose VIA
        lea     VRHead1Msg,a3           ;Point to message
        bsr4    VIAREGISTERS
        lea     VIHead1Msg,a3
        bsr4    VIAINTERRUPT
        lea     VT1Head1Msg,a3
        bsr4    VIA1TIMER
        lea     VT2Head1Msg,a3
        bsr4    VIA2TIMER

RestoreOk                               ; alternate entry from end of Via6Abort
        lea       LEV1INT,a4            ; restore the 3 vectors
        move.l    a4,LEV1VCT
        lea       LEV2INT,a4
        move.l    a4,LEV2VCT
        lea       LEV3INT,a4
        move.l    a4,LEV3VCT
        move.l  d7, a6                  ; restore the return address
        rts6                            ; back to caller

;--
;       Test all VIA's and abort on any error

Via6Abort                               ; test all 3 VIA's
        move.l  a6, d7                  ; save the return address
        lea     VLEV1INT,a4
        move.l  a4,LEV3VCT              ; alter MAC compatable interrupt vector
        move.l  #MacVIA,a2              ; Pass address of Mac VIA
        lea     VRHead3Msg,a3           ;Point to message
        bsr4    VIAREGISTERS
        cmp.w   #0,d6                   ;...check the error flag
        bne     RestoreAbort
        lea     VIHead3Msg,a3
        bsr4    VIAINTERRUPT
        cmp.w   #0,d6
        bne     RestoreAbort
        lea     VT1Head3Msg,a3
        bsr4    VIA1TIMER
        cmp.w   #0,d6
        bne     RestoreAbort
        lea     VT2Head3Msg,a3
        bsr4    VIA2TIMER
        cmp.w   #0,d6
        bne     RestoreAbort

        lea     VLEV1INT,a4
```

```
        move.l  a4,LEV2VCT              ; level 2 interrupt vector
        move.l  #MiscVIA,a2             ; Pass address of Misc VIA
        lea     VRHead2Msg,a3           ;Point to message
        bsr4    VIAREGISTERS
        cmp.w   #0,d6
        bne     RestoreAbort
        lea     VIHead2Msg,a3
        bsr4    VIAINTERRUPT
        cmp.w   #0,d6
        bne     RestoreAbort
        lea     VT1Head2Msg,a3
        bsr4    VIA1TIMER
        cmp.w   #0,d6
        bne     RestoreAbort
        lea     VT2Head2Msg,a3
        bsr4    VIA2TIMER
        cmp.w   #0,d6
        bne     RestoreAbort

        lea     VLEV1INT,a4
        move.l  a4,LEV1VCT              ; level 1 interrupt vector
        move.l  #XtraVIA,a2             ; General purpose VIA
        lea     VRHead1Msg,a3           ;Point to message
        bsr4    VIAREGISTERS
        cmp.w   #0,d6
        bne     RestoreAbort
        lea     VIHead1Msg,a3
        bsr4    VIAINTERRUPT
        cmp.w   #0,d6
        bne     RestoreAbort
        lea     VT1Head1Msg,a3
        bsr4    VIA1TIMER
        cmp.w   #0,d6
        bne     RestoreAbort
        lea     VT2Head1Msg,a3
        bsr4    VIA2TIMER
        cmp.w   #0,d6
        beq     RestoreOk               ; go to normal restore & rts6

RestoreAbort                            ; restore then abort
        lea       LEV1INT,a4            ; restore the 3 vectors
        move.l    a4,LEV1VCT
        lea       LEV2INT,a4
        move.l    a4,LEV2VCT
        lea       LEV3INT,a4
        move.l    a4,LEV3VCT
        bra     TopCommand              ; back to top level command loop

;--
;       first VIA test

VIAREGISTERS
        bsr6    PWrite6Line     ;...write it out the SCC
;
        move.l  a2,a5           ;Use a5 as base register to increment
        lea     AVAILABLE,a3    ;Registers that can be tested
        move.b  #$7F,IER(a5)    ;turn off 6522 interrupts
        move.b  #$00,ACR(a5)    ;Disable port latching.
        clr.w   d4              ;Init counter
;
@2      move.b  (a3)+,d2        ;See if legal register
        beq     @5              ;...skip if zero
;
        move.w  #$FF,d3         ;d3 holds data to write to 6522
@3      move.b  d3,(a5)         ;try writing to the 6522
        move.b  (a5),d2         ;...and reading back
        cmp.b   d3,d2           ;...was the data the same?
        bne     @11
;
@4      dbra    d3,@3           ;go thru data combinations to check
        move.b  d3,(a5)         ;Set to 0 on exit
;
@5      adda.l  #VRegIncr,a5    ;Next register address
        add.w   #1,d4           ;Increment register counter
        cmp.w   #14,d4
        bmi     @2
;
        bsr6    Pass6           ;Send PASS message
        clr.w   d6              ;...clear the error flag
;
@10     bsr6    Return6         ;Send CR
        rts4
;
@11     lea     FRegMsg,a3      ;Point to message ' failed, register '
        bsr6    PWrite6Line     ;...write it out the SCC
;
        move.l  d4,d5           ;Get register number
        bsr6    Dsp6BHex
;
        move.w  #$FFFF,d6       ;Set the error flag
        bra     @10
;
;
;       0   1    2    3    4    5    6    7    8    9 10  11  12  13  14  15
;     ORB,ORA,DDRB,DDRA,T1CL,T1CH,T1LL,T1LH,T2CL,T2CH,SR,ACR,PCR,IFR,IER,ORA
AVAILABLE
     .BYTE     0,0,1,0,0,0,1,1,0,0,1,1,1,0,0,0
;                    3     6 7    101112
;
;_____________________________________________________________________
;_____________________________________________________________________
;_____________________________________________________________________
;
```

```
; Check can get interrupt from I/O board.
;
VIAINTERRUPT
        clr.w   d6              ;clear the error flag
        bsr6    PWrite6Line     ;...write it out the SCC
;
        move.b  #$7F,IER(a2)    ;turn off 6522 interrupts
        move.b  #$00,ACR(a2)    ;Set so Timer 1 does one shot
;
        clr.w   d0              ;Do this test 10 times to assure reliability
@2      add.w   #1,d0           ;Times done - counter
        cmp.w   #11,d0          ;...Done 10 times yet?
        bpl     @3
;
        MOVE    #IntMask,SR     ; disable all interrupts
        move.b  #$C0,IER(a2)    ;...enable timer 1 interrupts
        move.b  #$01,T1CL(a2)   ;...and set a short (1 usec) timeout interval
        move.b  #$00,T1CH(a2)
        NOP                     ;wait a few microseconds for all the interrupts
        NOP                     ;...to be asserted
        mulu      #1,d6
        mulu      #1,d6
        mulu      #1,d6
        mulu      #1,d6
        clr.l     d3            ;D3 should come back a 1
;
        move    #$2000,sr       ;let the 6522 interrupt come thru
        NOP
        NOP                     ;Allow a little time to start processing
        mulu      #1,d6
        mulu      #1,d6
        cmp.w     #1,d3         ;Did we get the interrupt?, should be a 1
        beq       @2            ;...yes, good
;
        move.w    #$FFFF,d6     ;...no, set error flag
;
@3      MOVE    #IntMask,SR     ; disable all interrupts
        move.b  #$7F,IER(a2)    ; turn off 6522 timer interrupts
        move.b  #$7F,IFR(a2)    ; Clear any pending
;
        cmp.w   #0,d6           ;See if an error or not
        beq     @9
        bsr6    Fail6           ;Send Fail message
        bra     @10

@9      bsr6    Pass6           ;Send PASS message
;
@10     bsr6    Return6         ;Send CR
        rts4
;
;
;_____________________________________________________________________________
;_____________________________________________________________________________
;_____________________________________________________________________________
;
;
; Timer test, timer 1.
;
;Check timers on 6522(VIA)
;
VIA1TIMER
        bsr6    PWrite6Line     ;...write it out the SCC
        clr.l   d6              ;Clear error flag and counter of test values
;
@1      lea     HighValue,a3    ;Look for upper timer value to write
        add.l   d6,a3           ;...with offset for this test
        move.b  (a3),d1
;
        swap    d1
        lea     LowValue,a3     ;Look for lower timer value to write
        add.l   d6,a3           ;...with offset for this test
        move.b  (a3),d1
;
        lea     TLExpValue,a3   ;Look for expect value
        add.l   d6,a3           ;...with offset for this test
        add.l   d6,a3           ;...with offset for this test
        add.l   d6,a3           ;...with offset for this test
        add.l   d6,a3           ;...with offset for this test
        move.l  (a3),d2         ;Expected count
;
        lea     THExpValue,a3   ;Look for expect value
        add.l   d6,a3           ;...with offset for this test
        add.l   d6,a3           ;...with offset for this test
        add.l   d6,a3           ;...with offset for this test
        add.l   d6,a3           ;...with offset for this test
        move.l  (a3),d3         ;Expected count
;
        bsr     DoTimer1        ;Go do timer 1 test
        cmp.w   #$FFFF,d6       ;...see if a failure
        beq     TMRFAIL
;
        add.l   #1,d6           ;Go to next test value
        cmp.l   #21,d6          ;...see if all done
        bne     @1
;
TMRPASS
        clr.w   d6              ;Clear any errors
        bsr6    Pass6           ;Send PASS message
        bsr6    Return6         ;Send CR
        rts4
;
TMRFAIL
        bsr6    Return6         ;Send CR
```

```
;
        lea     FExpMsg,a3      ;Point to message
        bsr6    PWrite6Line     ;...write it out the SCC
        move.l  d2,d5           ;Get expected value
        bsr6    Dsp6LHex        ;Displays d5
;
        lea     ActMsg,a3       ;Point to message
        bsr6    PWrite6Line     ;...write it out the SCC
        move.l  d4,d5           ;Get Actual value
        bsr6    Dsp6LHex
;
        lea     LByteMsg,a3     ;Point to message
        bsr6    PWrite6Line     ;...write it out the SCC
        swap    d6              ;Move offset into lower half of register
        and.l   #$FFFF,d6       ;...clear error flag
        lea     LowValue,a3     ;Look for lower timer value to write
        add.l   d6,a3           ;...with offset for this test
        move.b  (a3),d5         ;Lower timer value
        bsr6    Dsp6BHex
;
        lea     UByteMsg,a3       ;Point to message
        bsr6    PWrite6Line     ;...write it out the SCC
        lea     HighValue,a3    ;Look for upper timer value to write
        add.l   d6,a3           ;...with offset for this test
        move.b  (a3),d5         ;Upper timer value
        bsr6    Dsp6BHex
;
        bsr6    Fail6           ;Send Fail message
        bsr6    Return6         ;Send CR
        rts4
;
;
;_____________________________________________________________________
;
;
; Timer test, timer 2.
;
;Check timers on 6522(VIA)
;
VIA2TIMER
        bsr6    PWrite6Line     ;...write it out the SCC
        clr.l   d6              ;Clear error flag and counter of test values
;
T2Top   lea     HighValue,a3    ;Look for upper timer value to write
        add.l   d6,a3           ;...with offset for this test
        move.b  (a3),d1
;
        swap    d1
        lea     LowValue,a3     ;Look for lower timer value to write
        add.l   d6,a3           ;...with offset for this test
        move.b  (a3),d1
;
        lea     TLExpValue,a3   ;Look for expect value
        add.l   d6,a3           ;...with offset for this test
        add.l   d6,a3           ;...with offset for this test
        add.l   d6,a3           ;...with offset for this test
        add.l   d6,a3           ;...with offset for this test
        move.l  (a3),d2         ;Expected count
;
        lea     THExpValue,a3   ;Look for expect value
        add.l   d6,a3           ;...with offset for this test
        add.l   d6,a3           ;...with offset for this test
        add.l   d6,a3           ;...with offset for this test
        add.l   d6,a3           ;...with offset for this test
        move.l  (a3),d3         ;Expected count
;
        bsr     DoTimer2        ;Go do timer 1 test
        cmp.w   #$FFFF,d6       ;...see if a failure
        beq     TMRFAIL
;
        add.l   #1,d6           ;Go to next test value
        cmp.l   #21,d6          ;...see if all done
        bne     T2Top
        bra     TMRPASS

;
LowValue        ;Lower timer value to write
        .byte   $01,$01,$02,$04,$08
        .byte   $10,$20,$40,$80,$FF
        .byte   $00,$00,$00,$00,$00
        .byte   $00,$00,$00,$FF,$00
        .byte   $FF
;
HighValue       ;Upper timer value to write
        .byte   $00,$00,$00,$00,$00
        .byte   $00,$00,$00,$00,$00
        .byte   $01,$02,$04,$08,$10
        .byte   $20,$40,$80,$00,$FF
        .byte   $FF
;
TLExpValue      ;Expect value
        .long   $00000000,$00000000,$00000000,$00000000,$00000001
        .long   $00000002,$00000004,$00000007,$00000011,$00000023
        .long   $00000023,$00000047,$00000090,$00000120,$00000240
        .long   $00000480,$00000940,$00001280,$00000023,$00002510
        .long   $00002530
;
THExpValue      ;Expect value
        .long   $00000004,$00000004,$00000004,$00000004,$00000004
        .long   $00000009,$0000000F,$0000001E,$00000035,$00000043
        .long   $00000053,$00000085,$000000F4,$00000184,$000002C8
        .long   $000005C9,$00000A80,$000014CE,$00000063,$00002C40
        .long   $00002C6A
```

```
;
;
DoTimer1
        move.b  #$7F,IER(a2)    ;...turn off 6522 interrupts
        move.b  #$7F,IFR(a2)    ;...clear pending interrupts
;
        move.b  #0,PCR(a2)      ;Set CA1 for neg active edge
        move.b  #0,ACR(a2)      ;Set timers for one-shot interrupt
        move.b  #$C0,IER(a2)    ;Enable Timer #1
;
        move.w  #$2000,sr       ;let the 6522 interrupt come thru
;
        clr.w   d3              ;Clear flag that says interrupt came thru
        move.b  d1,T1CL(a2)     ;Load counter low
        swap    d1
        move.b  d1,T1CH(a2)     ;Load upper and start counter
;
        clr.l   d0              ;Time expired counter
@2      cmp.w   #1,d3           ;Did timer #1 interrupt happen?
        beq     @3              ;...yes, ok
        add.l   #1,d0           ;...no, increment timeout timer
        cmp.l   #$FFFF,d0       ;...see if timeout or not
        bne     @2              ;...loop until timeout
;
@3      MOVE    #IntMask,SR     ; disable all interrupts
        move.b  #$7F,IER(a2)    ;...turn off 6522 interrupts
        move.b  #$7F,IFR(a2)    ;...clear pending interrupts
;
        move.l  d0,d4           ;Move to data register for compares
        lea     TLExpValue,a3   ;Look for expect value
        add.l   d6,a3           ;...with offset for this test
        add.l   d6,a3           ;...with offset for this test
        add.l   d6,a3           ;...with offset for this test
        add.l   d6,a3           ;...with offset for this test
        move.l  (a3),d2         ;Expected low count
        cmp.l   d2,d4           ;Compare against expected value
        bmi     @4              ;...negative if less than limit
;
        lea     THExpValue,a3   ;Look for expect value
        add.l   d6,a3           ;...with offset for this test
        add.l   d6,a3           ;...with offset for this test
        add.l   d6,a3           ;...with offset for this test
        add.l   d6,a3           ;...with offset for this test
        move.l  (a3),d2         ;Expected low count
        cmp.l   d4,d2           ;...negative if more than limit
        bmi     @4
        bra     @5
@4      swap    d6              ;Move offset into upper half of register
        move.w  #$FFFF,d6       ;...Set error flag
@5      rts
;
;_____________________________________________________________________
;
DoTimer2
        move.b  #$7F,IER(a2)    ;...turn off 6522 interrupts
        move.b  #$7F,IFR(a2)    ;...clear pending interrupts
;
        move.b  #0,PCR(a2)      ;Set CA1 for neg active edge
        move.b  #0,ACR(a2)      ;Set timers for one-shot interrupt
        move.b  #$A0,IER(a2)    ;Enable Timer #2
;
        move.w  #$2000,sr       ;let the 6522 interrupt come thru
;
        clr.w   d3              ;Clear flag that says interrupt came thru
        move.b  d1,T2CL(a2)     ;Load counter low
        swap    d1
        move.b  d1,T2CH(a2)     ;Load upper and start counter
;
        clr.l   d0              ;Time expired counter
@2      cmp.w   #2,d3           ;Did timer #2 interrupt happen?
        beq     @3              ;...yes, ok
        add.l   #1,d0           ;...no, increment timeout timer
        cmp.l   #$FFFF,d0       ;...see if timeout or not
        bne     @2              ;...loop until timeout
;
@3      MOVE    #IntMask,SR     ; disable all interrupts
        move.b  #$7F,IER(a2)    ;...turn off 6522 interrupts
        move.b  #$7F,IFR(a2)    ;...clear pending interrupts
;
        move.l  d0,d4           ;Move to data register for compares
        lea     TLExpValue,a3   ;Look for expect value
        add.l   d6,a3           ;...with offset for this test
        add.l   d6,a3           ;...with offset for this test
        add.l   d6,a3           ;...with offset for this test
        add.l   d6,a3           ;...with offset for this test
        move.l  (a3),d2         ;Expected low count
        cmp.l   d2,d4           ;Compare against expected value
        bmi     @4              ;...negative if less than limit
;
        lea     THExpValue,a3   ;Look for expect value
        add.l   d6,a3           ;...with offset for this test
        add.l   d6,a3           ;...with offset for this test
        add.l   d6,a3           ;...with offset for this test
        add.l   d6,a3           ;...with offset for this test
        move.l  (a3),d2         ;Expected low count
        cmp.l   d4,d2           ;...negative if more than limit
        bmi     @4
        bra     @5
@4      swap    d6              ;Move offset into upper half of register
        move.w  #$FFFF,d6       ;...Set error flag
@5      rts
;
;_____________________________________________________________________
```

```
;
;
;
; Level 1, VIA (6522), interrupt handler.
;  Expects
;     a2 = Via base address
;  Uses
;     d3 = 1 for timer 1 interrupt, 2 for timer 2 interrupt, 0 for other
;     d4 = Scratch
;
VLEV1INT
     move.b    IFR(a2),d4     ;Get which interrupt flags are set.
     move.b    IER(a2),d3     ;Get which interrupts are enabled.
     and.b     d3,d4          ;...Mask for possible interrupt source
     clr.w     d3
;
@1   btst      #6,d4          ;See if Timer 1 flag
     beq       @2             ;...not timer if flag zero
     move.b    T1CL(a2),d4    ;Clear interrupt
     move.w    #1,d3
     bra       @10
;
@2   btst      #5,d4          ;See if Timer 2 flag
     beq       @9             ;...not timer if flag zero
     move.b    T2CL(a2),d4    ;Clear interrupt
     move.w    #2,d3
     bra       @10
;
@9   move.b    #$7F,IER(a2)   ;turn off unknown 6522 interrupt
     move.b    #$7F,IFR(a2)   ;clear any pending interrupts

@10  rte
```

```
; Edit date: 02/14/85
;
; File: SCC.TEXT
;
; Function: Included into the ROM.TEXT program, tests the SCC (8530) and
;           most of the lines connected to it.
;
; History: 02/14/85 GRC, Original.
;          02/19/85 twm, Added SCCdelay Macro for Yacc hardware timing
;
;_____________________________________________________________________________
;
;
; SCC tests.
;
; These are the basic tests of the SCC on the Main Logic board.
;
;_____________________________________________________________________________
;
STCMD
        bsr6    Return6         ;Send CR
        clr.l   d4
        bra     TopCommand
;
        bsr4    SCCREGISTERS
;
        bsr4    SCCCRESET
;
        bsr4    SCCINTERNAL
;
        bsr4    SCCGINTERRUPT
;
        clr.l   d4
        bra     TopCommand
;
;_____________________________________________________________________________
;
; Register test, for registers 2, 12, 13, and 15.
;
;                                       d1 - Data written
;                                       d2 - Scratch/Data read back
; a3 - Registers to test mask address   d3 - Register number counter (0-15)
;
SCCREGISTERS
     lea        SCCRHeadMsg,a3  ;Point to message
     bsr6       PWrite6Line     ;...write it out the SCC
;
     clr.w      d6
     bsr6       Reset6SCC        ; reset the scc
;
     MOVE.L     #SCCRBase+2,A0 ;Pointer to SCC base read address A
     MOVE.L     #SCCWBase+2,A1 ;...generate SCC write address
     lea        AAVAILABLE,a3  ;...Registers that can be tested, and masks
     bsr        TestSCCRegs
;
     MOVE.L     #SCCRBase,A0   ;Pointer to SCC base read address B
     MOVE.L     #SCCWBase,A1   ;...generate SCC write address
     lea        BAVAILABLE,a3  ;...Registers that can be tested, and masks
     bsr        TestSCCRegs
;
     bsr6       Return6        ;Send CR
     rts4
;
;--
; twm - reset the scc via a reset command into the chip

Reset6SCC
     MOVE.L  #<SCCRBASE+2>,A0   ;initialize scc (point to channel A) read
     Move.B  #SCCReg9K, (A0)    ; point to WR9
     SCCDelay                   ;Delay
     Move.B  #SCCRstK, (A0)     ; the reset constant
     SCCDelay
     SCCDelay
     rts6
;_____________________________________________________________________________
;
; Test registers, expects:
;    a3 - Mask of registers that can be tested, 3 layer table start.
;
; Uses:
;    a5 - Pattern table.
;
TestSCCRegs
; Register loop
@1   move.b     (a3),d2        ;See if legal register
     beq        @5             ;...skip if zero
     cmp.b      #$12,d2        ;See if end of list, flag of $12
     beq        @6
;
     move.b     (a3),d1        ;Get mask for test bits
     move.b     17(a3),d2      ;Get register address
     MOVE.B     SCCREAD(A0),d0 ;Read to make sure SCC is sync'ed up
;
; Pattern loop
     lea        SCCPat,a5      ;Patterns to be used in the testing
@2   move.b     d2,WCtrl(A1)   ;Access correct register
     SccDelay                  ;Delay
     move.b     (a5),d3        ;Get pattern to write
     and.b      d1,d3          ;...Mask for valid bits
     move.b     d3,WCtrl(A1)   ;...Try writing to the SCC register
;
     SccDelay                  ;Delay
     move.b     d2,WCtrl(A1)   ;Access correct register
```

```
     SccDelay                  ;Delay
     move.b    RCtrl(A0),d4    ;...Try reading back
     SccDelay                  ;Delay
@3   and.b     d1,d4           ;...mask for valid bits
     cmp.b     d3,d4           ;...was the data the same?
     bne       @11
;
@4   move.b    (a5),d3         ;Last pattern?
     cmp.b     #0,d3           ;...expect a 0 for the last pattern
     beq       @5              ;...yes, exit on last pattern
     adda.l    #1,a5           ;go to next pattern
     bra       @2
;
@5   adda.l    #1,a3           ;Next register
     bra       @1
;
@6   move.l    a7,a6           ;save a7
     bsr7      Init7SCC        ;Initialize SCC port
     move.l    a6,a7           ;restore a7
;
     bsr6      Pass6            ;Send PASS message
     clr.w     d6               ;...clear the error flag
@7
     rts
;
;
@11  move.w    d2,d4
     move.l    a7,a6           ;save a7
     bsr7      Init7SCC        ;Initialize SCC port
     move.l    a6,a7           ;restore a7
;
     lea       FRegMsg,a3      ;Point to message ' failed, register '
     bsr6      PWrite6Line     ;...write it out the SCC
;
     move.l    d4,d5           ;Get register number
     bsr6      Dsp6BHex
;
     move.b    #' ',d0         ;Send space
     bsr5      Out5Char
;
     move.w    #$FFFF,d6       ;Set the error flag
     bra       @7
;
; NOTE: Data must be in this specific order. Each group is 17 bytes long
;   1st group is a mask of which registers to test, 1 means test it.
;   2nd group is a mask of which register bits can be tested.
;   3ed group is the register address to send to the control register.
;
;                    2                    12  13    15
AAVAILABLE
     .BYTE     0,0,$FF,0,0,0,0,0,0,0,0,0,$FF,$FF,0,$FA,$12
     .BYTE     0,0,$02,0,0,0,0,0,0,0,0,0,$0C,$0D,0,$0F,$12
;                                          12  13    15
BAVAILABLE
     .BYTE     0,0,0,0,0,0,0,0,0,0,0,0,$FF,$FF,0,$FA,$12
     .BYTE     0,0,0,0,0,0,0,0,0,0,0,0,$0C,$0D,0,$0F,$12
;
; Patterns must end with a $00
SCCPat
     .BYTE     $FF,$01,$02,$04,$08,$10,$20,$40,$80,$55,$AA,$00
;
;-------------------------------------------------------------------------
;
; Reset channels test, for registers 0, 1, 3, and 10.
;
;                                        d1 - Data written
;                                        d2 - Scratch/Data read back
; a3 - Registers to test mask address    d3 - Register number counter (0-15)
;
SCCCRESET
     lea       SCCRtHeadMsg,a3 ;Point to message
     bsr6      PWrite6Line      ;...write it out the SCC
;
     bsr6      Reset6SCC         ; reset the scc
;
     MOVE.L    #SCCRBase+2,A0 ;Pointer to SCC base read address A
     MOVE.L    #SCCWBase+2,A1 ;...generate SCC write address
     lea       EXPRESET,a3
     bsr       CKSCCRESET
;
     MOVE.L    #SCCRBase,A0   ;Pointer to SCC base read address B
     MOVE.L    #SCCWBase,A1   ;...generate SCC write address
     lea       EXPRESET,a3
     bsr       CKSCCRESET
;
     bsr6      Return6        ;Send CR
     rts4
;
;-------------------------------------------------------------------------
;
CKSCCRESET
     move.b    (a3),d1        ;Get mask for test bits
     cmp.b     #$12,d1        ;...end flag?
     beq       @1
     move.b    5(a3),d2       ;Get register address
     move.b    10(a3),d3      ;Get expect data
;
     move.b    d2,WCtrl(A1)   ;Access correct register
     SccDelay                 ;Delay
     move.b    RCtrl(A0),d4   ;...Try reading back
     SccDelay                 ;Delay
     and.b     d1,d4          ;...mask for valid bits
     cmp.b     d4,d3          ;compare to expected
```

```
     bne       @11
     adda.l    #1,a3          ;Next register
     bra       CKSCCRESET
;
@1   move.l    a7,a6          ;save a7
     bsr7      Init7SCC       ;Initialize SCC port
     move.l    a6,a7          ;restore a7
;
     bsr6      Pass6           ;Send PASS message
     clr.w     d6              ;...clear the error flag
@10  rts
;
@11  move.w    d2,d4
     move.l    a7,a6          ;save a7
     bsr7      Init7SCC       ;Initialize SCC port
     move.l    a6,a7          ;restore a7
;
     bsr6      Fail6          ;Send Fail message
;
     move.w    #$FFFF,d6      ;Set the error flag
     bra       @10
;
; Table contains a) mask for bits to test
;                b) register address
;                c) value expected
;  All fields terminated by a flag of $12, table length is critical to code
EXPRESET
     .BYTE     $C3,$FF,$FF,$FF,$12
     .BYTE     $00,$01,$03,$0A,$12
     .BYTE     $40,$06,$00,$00,$12,0
;          7 6 5 4 3 2 1 0
; 0 expect 0 1 . . . 1 0 0
; 1        0 0 0 0 0 1 1 1
; 3        0 0 0 0 0 0 0 0
;10        0 0 0 0 0 0 0 0
;
;_____________________________________________________________________________
;
; Internal loopback test.
;
;                                        d1 - Data written
;                                        d2 - Scratch/Data read back
; a3 - Registers to test mask address    d3 - Register number counter (0-15)
;
SCCINTERNAL
     lea       SCCILBHeadMsg,a3 ;Point to message
     bsr6      PWrite6Line     ;...write it out the SCC
;
     bsr6      Reset6SCC        ; reset the scc
;
     clr.w     d6
     bsr       ASCCInternal
;
     bsr       BSCCInternal
;
     bsr6      Return6        ;Send CR
     rts4
;
;_____________________________________________________________________________
;
ASCCInternal
     MOVE.L    #SCCRBase+2,A0      ;Pointer to SCC base read address A
     MOVE.L    #SCCWBase+2,A1      ;...generate SCC write address
;
     move.l    #$00800009,d1       ;9,Reset Channel A, disable interrupts
     bsr       SCCRWRITE
;
     bsr       SCCTEST
;
     rts
;
;_____________________________________________________________________________
;
BSCCInternal
     MOVE.L    #SCCRBase,A0        ;Pointer to SCC base read address B
     MOVE.L    #SCCWBase,A1        ;...generate SCC write address
;
     move.l    #$00400009,d1       ;9,Reset Channel B, disable interrupts
     bsr       SCCRWRITE
;
     bsr       SCCTEST
;
     rts
;
;_____________________________________________________________________________
;
SCCTEST
     SccDelay                      ;Delay
     SccDelay                      ;Delay
;
     move.l    #$004D0004,d1  ; 4,x16 clk,8 bit sync,2 stop bits/char,even par
     bsr       SCCRWRITE
     move.l    #$0050000B,d1  ;11,Rec clk = Transmit clk = BR gen output
     bsr       SCCRWRITE
     move.l    #$0000000F,d1  ;15,Interrupts disabled
     bsr       SCCRWRITE
     move.l    #$00000000E,d1 ;14,BR gen disabled
     bsr       SCCRWRITE
     move.l    #$000A000C,d1  ;12,9600 Baud
     bsr       SCCRWRITE
     move.l    #$00000000D,d1 ;13,9600 Baud
     bsr       SCCRWRITE
     move.l    #$0013000E,d1  ;14,BR gen enabled, local loopback mode
```

```
     bsr       SCCRWRITE
     move.l    #$0000000A,d1  ;NRZ mode
     bsr       SCCRWRITE
     move.l    #$00C10003,d1  ; 3,Rx 8 bits/char, Rx enable
     bsr       SCCRWRITE
     move.l    #$006A0005,d1  ; 5,Tx 8 bits/char, Tx enable, RTS enable
     bsr       SCCRWRITE
;
     TST.B     SCCREAD(A0)    ; CLEAR RX BUFFER
;
     move.l    #$FF005581,d0  ;Test data pattern, per byte, FF, 00, 55, 81
;
@1   move.b    #$30,WCtrl(A1) ;Reset any errors
;
     move.w    #$1000,d2      ;Timeout
@2   btst      #2,RCtrl(A0)   ;Tx Buffer empty?
     bne       @3             ;...yes, go to next step
     SccDelay                 ;Delay
     dbra      d2,@2          ;...Check for timeout
     move.w    #1,d4
     bra       @100
;
@3   move.b    d0,SCCWrite(A1) ;Write data
     move.w    #$5000,d2      ;Timeout
@4   move.w    #$01,d1        ;Read register #1
     bsr       SCCRREAD
     btst      #0,d1          ;All sent?
     bne       @5             ;...yes, continue
     dbra      d2,@4          ;...Check for timeout
     move.w    #2,d4
     bra       @100
;
@5   move.w    #$5000,d2      ;Timeout
@6   btst      #0,RCtrl(A0)   ;Character available?
     bne       @7             ;...yes, continue
     SccDelay                 ;Delay
     dbra      d2,@6          ;...Check for timeout
     move.w    #3,d4
     bra       @100
;
@7   move.b    SCCREAD(A0),d1 ;Read the data
@77  cmp.b     d1,d0          ;Same as written?
     beq       @8
     move.w    #4,d4          ;Data written <> data read
     bra       @100
;
@8   move.w    #$01,d1        ;Read register #1
     bsr       SCCRREAD
     and.b     #$70,d1        ;Check for Parity, Rx overrun, CRC/Framing errors
     cmp.b     #0,d1          ;...OK?
     beq       @9
     move.w    #5,d4          ;Error at end of transmission
     bra       @100

@9   rol.l     #8,d0          ;Go to next data
     cmp.b     #$81,d0        ;Last data?
     bne       @1
;
     move.l    a7,a6          ;save a7
     bsr7      Init7SCC       ;Initialize SCC port
     move.l    a6,a7          ;restore a7
;
     bsr6      Pass6           ;Send PASS message
;
@10
     rts
;
@100
     move.l    a7,a6          ;save a7
     bsr7      Init7SCC       ;Initialize SCC port
     move.l    a6,a7          ;restore a7
;
     lea       StepMsg,a3     ;Point to message
     bsr6      PWrite6Line    ;...write it out the SCC
;
     move.l    d4,d5          ;Get register number
     bsr6      Dsp6BHex
;
     move.b    #' ',d0        ;Send space
     bsr5      Out5Char
;
     bsr6      Fail6           ;Send Fail message
;
     move.w    #$FFFF,d6      ;Set the error flag
     bra       @10
;
;_____________________________________________________________________
;
; Write to an SCC register
; Expects a) Control write address in a1
;         b) Register address in lower byte of lower half of d1
;         c) Register data    in lower byte of upper half of d1
;
SCCRWRITE
     move.b    d1,WCtrl(A1)   ;Access correct register
     SccDelay                 ;Delay
     swap      d1
     move.b    d1,WCtrl(A1)   ;Write data to register
     SccDelay                 ;Delay
     swap      d1
     rts
;
;_____________________________________________________________________
```

```
;
; Reads from an SCC register
; Expects a) Control write address in a1
;         b) Control read  address in a0
;         c) Register address in lower byte of lower half of d1
; Returns
;         a) Register data    in lower byte of lower half of d1
;         b) Register address in lower byte of upper half of d1
;
SCCRREAD
     move.b    d1,WCtrl(A1)    ;Access correct register
     SccDelay                  ;Delay
     swap      d1
     move.b    RCtrl(A0),d1    ;Read data from register
     SccDelay                  ;Delay
     and.w     #$00FF,d1       ;...mask read data so lower word is clean
     rts
;
;-------------------------------------------------------------------------
;
;
; Check can get interrupt from SCC chip.
;
SCCGINTERRUPT
     clr.w     d6              ;clear the error flag
     lea       SCCINTHeadMsg,a3    ;Point to message
     bsr6      PWrite6Line     ;...write it out the SCC
;
     lea       SLEV2INT,a6     ;Put in our vector for interrupts from SCC
     move.l    a6,LEV2VCT
;
     bsr6      Reset6SCC         ; reset the scc
;
     MOVE.L    #SCCRBase+2,A0       ;Pointer to SCC base read address A
     MOVE.L    #SCCWBase+2,A1       ;...generate SCC write address
     move.w    #10,d0               ;Number of times to do interrupt test
;
@0   move.l    #$00800009,d1  ; 9,Reset Channel A, disable interrupts
     bsr       SCCRWRITE
     move.l    #$000A0009,d1  ; 9,Master Interrupt Enabled, No Vector
     bsr       SCCRWRITE
     move.l    #$00FF000C,d1  ;12,Set Baud rate to Max
     bsr       SCCRWRITE
     move.l    #$0000000D,d1  ;13,Set Baud rate to min
     bsr       SCCRWRITE
     move.l    #$0002000F,d1  ;15,Zero Count Interrupt Enabled
     bsr       SCCRWRITE
     move.b    #$10,WCtrl(A1) ; 0,Reset external status interrupts
     move.l    #$00010001,d1  ; 1,External/Status interrupt enable
     bsr       SCCRWRITE
     move.l    #$0003000E,d1  ;14,BR Gen Source, BR Gen Enable
     bsr       SCCRWRITE
;
     clr.w     d3
     move      #$2000,sr      ;let the SCC interrupt come thru
;
     move.w    #$1000,d1      ;Timeout value
@1   cmp.w     #1,d3          ;Did we get the interrupt
     beq       @2             ;...yes
     dbra      d1,@1          ;...no, check timeout
     bra       @12
;
@2   move.w    #IntMask,SR    ;disable all interrupts
     dbra      d0,@0          ;Redo interrupt test for specified # times
;
; Passed
@10  move.w    #IntMask,SR    ; disable all interrupts and restore interrupt
;
     move.l    #$00800009,d1  ; 9,Reset Channel A, disable interrupts
     bsr       SCCRWRITE
;
     lea       LEV2INT,a6     ;Restore original vector address
     move.l    a6,LEV2VCT
;
     move.l    a7,a6          ;save a7
     bsr7      Init7SCC       ;Initialize SCC port
     move.l    a6,a7          ;restore a7
;
     bsr6      Pass6           ;Send PASS message
;
@11  bsr6      Return6        ;Send CR
     rts4
;
; Failed
@12  move.w    #IntMask,SR    ; disable all interrupts and restore interrupt
;
     move.l    #$00800009,d1  ; 9,Reset Channel A, disable interrupts
     bsr       SCCRWRITE
;
     lea       LEV2INT,a6     ;Restore original vector address
     move.l    a6,LEV2VCT
;
     move.l    a7,a6          ;save a7
     bsr7      Init7SCC       ;Initialize SCC port
     move.l    a6,a7          ;restore a7
;
     bsr6      Fail6           ;Send Fail message
;
     move.w    #$FFFF,d6      ;Set the error flag
     bra       @11
;
;-------------------------------------------------------------------------
;
```

```
; Level 2, SCC (8530), interrupt handler.
; assumes that registers a0 and a1 are setup pointing to the A port.
; Sets d3=1 for port A interrupt
;      d3=2 for port B interrupt
;
SLEV2INT
     movem.l   d0-d1/a0-a1,-(sp)  ;Save everyone
     move.b    #$03,WCtrl(A1) ;Read register #3
     SccDelay                 ;Delay
     move.b    RCtrl(A0),d1   ;Read data from register
     SccDelay                 ;Delay
     clr.w     d3             ;Set up to no interrupt source known
     btst      #0,d1          ;Channel B?
     bne       @1
     btst      #3,d1          ;Channel A?
     bne       @2
     move      #IntMask,sr    ;disable all interrupts, interrupt unknown
     bra       @10
;
@1   sub.l     #2,a1          ;Set up to B port interrupt
     move.w    #2,d3          ;Set up to B port interrupt
     bra       @10
;
@2   move.w    #1,d3          ;Set up to A port interrupt
;
@10  move.b    #$10,WCtrl(a1) ; 0,Reset external status interrupts
     movem.l   (sp)+,d0-d1/a0-a1   ;Restore the world
     rte
;
;_____________________________________________________________________________
;_____________________________________________________________________________
;
```

```
;-------------------------------------------------------------------------
;
;    File:     Messages.text
;    This contains all the messages that can be printed to the serial port

MyInitMsg
;                 12345678901234567890123456789012345678901234567890123456789012
        .ascii  'ROMTALK, Rev 0.1x, Initialization complete'
        .byte   0,0
ErrBitsMsg
;                 12345678901234567890123456789012345678901234567890123456789012
        .ascii  ' are failing bits (31 to 0).'
        .byte   0,0
FExpMsg
;                 1234567890123
        .ascii  'failed, exp='
        .byte   0,0
StepMsg
;                 12345678
        .ascii  '  Step= '
        .byte   0,0
ActMsg
;                 1234
        .ascii  'act='
        .byte   0,0
LByteMsg
;                 1234
        .ascii  'LByte='
        .byte   0,0
UByteMsg
;                 1234
        .ascii  'UByte='
        .byte   0,0
FRegMsg
;                 1234567890123456789012345678
        .ascii  'failed, register $'
        .byte   0,0
;
;                 1234
HUH     .ascii  'Huh?'
        .byte   0,0
;
DLHeadMsg
;                 123456789012
        .ascii  'Data lines  '
        .byte   0,0
SCHeadMsg
;                 123456789012
        .ascii  'Stuck cell  '
        .byte   0,0
ALHeadMsg
;                 123456789012
        .ascii  'Addr lines  '
        .byte   0,0
FPHeadMsg
;                 123456789012
        .ascii  'Fixed pat   '
        .byte   0,0
M1HeadMsg
;                 123456789012
        .ascii  'March 1     '
        .byte   0,0
M2HeadMsg
;                 123456789012
        .ascii  'March 2     '
        .byte   0,0
BIHeadMsg
;                 123456789012
        .ascii  'Byte M Inv  '
        .byte   0,0
WIHeadMsg
;                 123456789012
        .ascii  'Word M Inv  '
        .byte   0,0
LIHeadMsg
;                 123456789012
        .ascii  'LWord M Inv '
        .byte   0,0
VRHead3Msg
;                 1234567890123456
        .ascii  'Mac VIA Regs    '
        .byte   0,0
VIHead3Msg
;                 1234567890123456
        .ascii  'Mac VIA Inter   '
        .byte   0,0
VT1Head3Msg
;                 1234567890123456
        .ascii  'Mac VIA Timer#1 '
        .byte   0,0
VT2Head3Msg
;                 1234567890123456
        .ascii  'Mac VIA Timer#2 '
        .byte   0,0
VRHead2Msg
;                 12345678901234567
        .ascii  'Misc VIA Regs    '
        .byte   0,0
VIHead2Msg
;                 12345678901234567
        .ascii  'Misc VIA Inter   '
        .byte   0,0
VT1Head2Msg
```

change to red.

```
;                12345678901234567
        .ascii  'Misc VIA Timer#1 '
        .byte   0,0
VT2Head2Msg
;                12345678901234567
        .ascii  'Misc VIA Timer#2 '
        .byte   0,0
VRHead1Msg
;                12345678901234567
        .ascii  'Xtra VIA Regs    '
        .byte   0,0
VIHead1Msg
;                12345678901234567
        .ascii  'Xtra VIA Inter   '
        .byte   0,0
VT1Head1Msg
;                12345678901234567
        .ascii  'Xtra VIA Timer#1 '
        .byte   0,0
VT2Head1Msg
;                12345678901234567
        .ascii  'Xtra VIA Timer#2 '
        .byte   0,0
SCCRHeadMsg
;                123456789012
        .ascii  'SCC Regs    '
        .byte   0,0
SCCRtHeadMsg
;                123456789012
        .ascii  'SCC Reset   '
        .byte   0,0
SCCILBHeadMsg
;                123456789012
        .ascii  'SCC IntLpbk '
        .byte   0,0
SCCINTHeadMsg
;                123456789012
        .ascii  'SCC Inter   '
        .byte   0,0
PHeadMsg
;                12345678901234
        .ascii  'Pass number  $'
        .byte   0,0

;_____________________________________________________________________________
;
;    These are Exception messages that should be accessed only upon an
;    Exception error.

EMISCMsg
;                12345678901234
        .ascii  'Misc exception'
        .byte   0,0
EBUSSMsg
;                1234567890
        .ascii  'Buss error'
        .byte   0,0
EADRMsg
;                12345678901234
        .ascii  'Address error '
        .byte   0,0
EILLMsg
;                12345678901234
        .ascii  'Illegal instr '
        .byte   0,0
EDIV0Msg
;                123456789012
        .ascii  'Divide by 0 '
        .byte   0,0
ECHKMsg
;                1234567890
        .ascii  'CHK instr '
        .byte   0,0
ETRAPVMsg
;                123456789012
        .ascii  'TRAPV instr '
        .byte   0,0
EPRIVMsg
;                123456789012
        .ascii  'Priv instr  '
        .byte   0,0
ETRACEMsg
;                123456789012
        .ascii  'Trace instr '
        .byte   0,0
E1010Msg
;                1234567890
        .ascii  'Line 1010 '
        .byte   0,0
E1111Msg
;                1234567890
        .ascii  'Line 1111 '
        .byte   0,0
EUIVMsg
;                12345678901234567890
        .ascii  'Uninit inter vector '
        .byte   0,0
ESpurMsg
;                123456789012345678
        .ascii  'Spurious interrupt'
        .byte   0,0
EL1Msg
;                123456789012345678
```

```
        .ascii  'Level 1 interrupt '
        .byte   0,0
EL2Msg
;                123456789012345678
        .ascii  'Level 2 interrupt '
        .byte   0,0
EL3Msg
;                123456789012345678
        .ascii  'Level 3 interrupt '
        .byte   0,0
EL4Msg
;                123456789012345678
        .ascii  'Level 4 interrupt '
        .byte   0,0
EL5Msg
;                123456789012345678
        .ascii  'Level 5 interrupt '
        .byte   0,0
EL6Msg
;                123456789012345678
        .ascii  'Level 6 interrupt '
        .byte   0,0
EL7Msg
;                123456789012345678
        .ascii  'Level 7 interrupt '
        .byte   0,0
```