```
;
; Edit Date: 07/21/83
;
;_______________________________________________________________________________
;
;    File: BootROM.TEXT
; Purpose: This is the main flow for the Lisa 1.75 Boot ROM
;
;
          .PROC     BootROM,0
          .ref      MakeDesk
;
ROM0           EQU  $00000000    ;Location of ROM
ROM1           EQU  $00020000    ;Location of second ROM
ROMSize        EQU  $00007FFF    ;Size of each ROM
VideoMemory    EQU  $00000000    ;Location of Video Memory
VidSize        EQU  $00020000    ;Size of Video Memory
LEDon          EQU  $0000300D    ;Turn on  LED on CPU BOARD, disable Serial Port A
LEDoff         EQU  $0000300C    ;Turn off LED on CPU BOARD, enable Serial Port A
;
          move.L    (SP)+,A0         ;save return address
          movem.l   d0-d7/a0-a6,-(sp)  ;Save everyone
;
; Set exception vector pointer to defaults in ROM
; Set stack to video memory
;
          tst.b     LEDon            ;Turn on CPU board LED
; Set Lisa Video mode.
;
          move.l    #ROM0,a1         ;Address of ROM for Checksum test
          move.l    #ROMSize,d1      ;Length of ROM
          lea       ROM0Done,a0      ;Return address, can not use the stack
          bra       CheckSum         ;Go verify the checksum of first ROM
ROM0Done
;
          move.l    #ROM1,a1         ;Address of ROM for Checksum test
          move.l    #ROMSize,d1      ;Length of ROM
          lea       ROM1Done,a0      ;Return address, can not use the stack
          bra       CheckSum         ;Go verify the checksum of second ROM
ROM1Done
;
          move.l    #VideoMemory,a1  ;Address of Video memory
          move.l    #VidSize,d1      ;Length of Video memory
          lea       VidMDone,a0      ;Return address, can not use the stack
          bra       VidMem           ;Go do video memory test
VidMDone
;
          move.l    #VideoMemory,a1  ;Address of Video memory
          move.l    #VidSize,d1      ;Length of Video memory
          lea       VidPMDone,a0     ;Return address, can not use the stack
          bra       VidPMem          ;Go do video memory parity test
VidPMDone
;
; Write screen area to all ones
; Rest of screen memory to all zeros
;
          jsr       MakeDesk         ;Make a desktop
;
          move.b    #1,d0            ;Timer #1 test (level 6).
          bsr       Timers
;
          bsr       RS232            ;RS232 tests (Level 6).
;
          bsr       IExpansion       ;Ck for expansion and slot 4 inter (5 to 2).
;
          move.b    #1,d0            ;Timer #0 test.
          bsr       Timers
;
          move.b    #1,d0            ;Timer #2 test.
          bsr       Timers
;
          bsr       COPSTest         ;Verify basic COPS operation.
;
;    SetContrast  - Set new contrast value.
;    SetVolume    - Set speaker volume.
;    Silence      - Turn off speaker.
;    Beep         - Tones for speaker.
;    Poll         - Polling mode on COPS.
;    Keyboard     - Get keyboard I.D.
;    KeybdEvent   - Get a keyboard event, must also handle COPS error codes.
;    KeybdPeek    - Examine keyboard queue.
;
          bsr       SizeMemory       ;Size memory, find all memory
;
          bsr       MMUBasics        ;MMU read/write & address test.
;
;
          bsr       MEMPatterns      ;Main memory pattern tests.
;
          bsr       MEMParity        ;Main memory parity circuit test.
;
          bsr       MMUFunctional    ;MMU functional test.
;
;    Floppy driver - Read a sector.
;    Floppy driver - Eject a disk.
;    Floppy driver - See if disk is in.
;    Floppy driver - Debug commands required by Field Service.
          bsr       IWMChip        ;IWM, floppy driver chip, test.
;
;    Built-in hard disk driver - Read status of selftest.
;    Built-in hard disk driver - See if disk is ready yet.
;    Built-in hard disk driver - Read a sector.
          bsr       BuiltIn        ;Builtin hard disk port test
;
          bsr       Test1Expansion ;Execute expansion card status program
;
          bsr       Test2Expansion ;Execute expansion card status program
;
          bsr       Test3Expansion ;Execute expansion card status program
;
          bsr       Test4Expansion ;Execute expansion card status program
;
          movem.l   (sp)+,d0-d7/a0-a6   ;Restore the world
          jmp       (a0)
;
;__________________________________________________________________________
;__________________________________________________________________________
;__________________________________________________________________________
;
; Function - Compute a checksum on the memory pointed to.
;
; On entry expects
;   a0 = return address after test is done
;   a1 = start address to check
;   d1 = number of bytes to test
; On exit
;   d0 = 0 for checksum OK, and non-zero for bad checksum
;   d1 is destroyed
;   d2 = Expected checksum
;   d3 = Actual checksum
;
CheckSum
;
          jmp       (a0)
;
;__________________________________________________________________________
;
;Function - Perform memory tests on the video memory
;
; On entry expects
;   a0 = return address after test is done
;   a1 = start address to check
;   d1 = number of bytes to test
; On exit
;   d0 = 0 for memory OK, and non-zero for bad memory
;   Memory is left at all zeros
VidMem
;
          jmp       (a0)
;
;__________________________________________________________________________
;
; Function - Perform parity tests on video memory
;
; On entry expects
;   a0 = return address after test is done
;   a1 = start address to check
;   d1 = number of bytes to test
; On exit
;   d0 = 0 for memory OK, and non-zero for bad memory parity
;   Memory is left at all zeros
VidPMem
;
          jmp       (a0)
;
;__________________________________________________________________________
;
; Function - Perform timer chip tests
;
; On entry expects
;   d0 = timer number to test, byte (0 to 2)
; On exit
;   d0 = 0 for timer OK, and non-zero for bad timer
;   a0 = Detailed error table
Timers
;
          rts
;
;__________________________________________________________________________
;
; Function - Perform RS232 port tests
;
; On entry expects
;   nothing expected
; On exit
;   d0 = 0 for timer OK, and non-zero for bad timer
;   a0 = Detailed error table
RS232
;
          rts
;
;__________________________________________________________________________
;
; Function - Check for expansion slot and slot 4 interrupts
;
; On entry expects
;   nothing expected
; On exit
;   d0 = 0 for interrupts OK, and non-zero for stray interrupts coming in
;   a0 = Detailed error table
IExpansion
;
          rts
; ______________________________________________________________________
;
; Function - COPS test, turns on the port, brings in any codes, reads the clock,
;    uses special register read commands to verify COPS (Checksum?), sends
;    keyboard reset command and gets keyboard I.D. to check against previous
;    I.D.  Handles COPS error codes coming in.
;
; On entry expects
;   nothing expected
; On exit
;   d0 = 0 for COPS OK, and non-zero for bad values from COPS
;   a0 = Detailed error table
COPSTest
;
          rts
;
; ______________________________________________________________________
;
; Function - Find memory on other boards.
;
; On entry expects
;   nothing expected
; On exit
;   d0 = 0 for found memory OK, and non-zero for non memory found
;   a0 = Detailed error table
;  Places memory data in table in video memory
SizeMemory
;
          rts
;
; ______________________________________________________________________
;
; Function - MMU read/write & address test.
;
; On entry expects
;   Expects that a memory board exists
; On exit
;   d0 = 0 for MMU OK, and non-zero for bad MMU Ram
;   a0 = Detailed error table
;  Leaves MMU in a state that ......
MMUBasics
;
          rts
; ______________________________________________________________________
;
; Function -  Main memory pattern tests.
;
; On entry expects
;   Expects that a memory board exists
; On exit
;   d0 = 0 for MMU OK, and non-zero for bad Ram
;   a0 = Detailed error table
;  Leaves Memory written to all zeros.
MEMPatterns
;
          rts
; ______________________________________________________________________
;
; Function - Main memory parity circuit test.
;
; On entry expects
;   Expects that a memory board exists
; On exit
;   d0 = 0 for MMU OK, and non-zero for bad MMU Ram
;   a0 = Detailed error table
;  Leaves MMU in a state that ......
MEMParity
;
          rts
; ______________________________________________________________________
;
; Function - MMU functional test.
;
; On entry expects
;   Expects that a memory board exists
; On exit
;   d0 = 0 for MMU OK, and non-zero for bad MMU Ram
;   a0 = Detailed error table
;  Leaves MMU in a state that ......
MMUFunctional
;
          rts
; ______________________________________________________________________
;
; Function - IWM, floppy driver chip, test.
;
; On entry expects
;   Expects nothing
; On exit
;   d0 = 0 for IWM OK, and non-zero for bad IWM
;   a0 = Detailed error table
;  Leaves IWM in a state that ......
IWMChip
;
         rts
;_______________________________________________________________________
;
; Function - Built-in hard disk port test
;
; On entry expects
;   Expects nothing
; On exit
;   d0 = 0 for port OK, and non-zero for bad port
;   a0 = Detailed error table
;  Leaves port in a state that ......
BuiltIn
;
         rts
;_______________________________________________________________________
;
; Function - Execute expansion card 1 status program
;
; On entry expects
;   Expects nothing
; On exit
;   d0 = 0 for card OK, and non-zero for bad card
;   a0 = Detailed error table
Test1Expansion
;
         rts
;
;_______________________________________________________________________
;
; Function - Execute expansion card 2 status program
;
; On entry expects
;   Expects nothing
; On exit
;   d0 = 0 for card OK, and non-zero for bad card
;   a0 = Detailed error table
Test2Expansion
;
         rts
;
;_______________________________________________________________________
;
; Function - Execute expansion card 3 status program
;
; On entry expects
;   Expects nothing
; On exit
;   d0 = 0 for card OK, and non-zero for bad card
;   a0 = Detailed error table
Test3Expansion
;
         rts
;
;_______________________________________________________________________
;
; Function - Execute expansion card 4 status program
;
; On entry expects
;   Expects nothing
; On exit
;   d0 = 0 for card OK, and non-zero for bad card
;   a0 = Detailed error table
Test4Expansion
;
         rts
;
;_______________________________________________________________________
;
        .PROC    MakeDesk,0
        .ref     AIcon_Draw,ADialog,DeskTop,Paint_String,Paint_Ch
;
        move.L   (SP)+,A0          ;save return address
        movem.l  d0-d7/a0-a6,-(sp) ;Save everyone
;
        jsr      DeskTop           ;Make a blank desktop
;
        move.W   #60,-(SP)         ;x1
        move.W   #30,-(SP)         ;y1
        move.W   #640,-(SP)        ;x2
        move.W   #90,-(SP)         ;y2
        jsr      ADialog           ;Draw dialog box for main screen
;
        move.W   #90,-(SP)         ;x1
        move.W   #50,-(SP)         ;y1
        move.W   #8,-(SP)          ;Icon code, LISA picture
        jsr      AIcon_Draw        ;Draw LISA picture in box
;
        move.W   #170,-(SP)        ;x1
        move.W   #50,-(SP)         ;y1
        move.W   #1,-(SP)          ;Icon code, Big board picture
        jsr      AIcon_Draw        ;Draw Big board picture in box
        move.W  #172,-(SP)          ;x1
        move.W  #60,-(SP)           ;y1
        lea     CPU,a1
        move.L  a1,-(SP)            ;string address
        jsr     Paint_String
;
        move.W  #250,-(SP)          ;x1
        move.W  #50,-(SP)           ;y1
        move.W  #2,-(SP)            ;Icon code, memory board picture
        jsr     AIcon_Draw          ;Draw memory board picture in box
;
        move.W  #330,-(SP)          ;x1
        move.W  #50,-(SP)           ;y1
        move.W  #3,-(SP)            ;Icon code, Expansion card 1 picture
        jsr     AIcon_Draw          ;Draw Expansion card picture in box
        move.W  #340,-(SP)          ;x1
        move.W  #60,-(SP)           ;y1
        move.w  #'1',-(sp)          ;Character
        jsr     Paint_Ch            ;Place character on the screen
;
        move.W  #410,-(SP)          ;x1
        move.W  #50,-(SP)           ;y1
        move.W  #3,-(SP)            ;Icon code, Expansion card 2 picture
        jsr     AIcon_Draw          ;Draw Expansion card picture in box
        move.W  #420,-(SP)          ;x1
        move.W  #60,-(SP)           ;y1
        move.w  #'2',-(sp)          ;Character
        jsr     Paint_Ch            ;Place character on the screen
;
        move.W  #490,-(SP)          ;x1
        move.W  #50,-(SP)           ;y1
        move.W  #3,-(SP)            ;Icon code, Expansion card 3 picture
        jsr     AIcon_Draw          ;Draw Expansion card picture in box
        move.W  #500,-(SP)          ;x1
        move.W  #60,-(SP)           ;y1
        move.w  #'3',-(sp)          ;Character
        jsr     Paint_Ch            ;Place character on the screen
;
        move.W  #570,-(SP)          ;x1
        move.W  #50,-(SP)           ;y1
        move.W  #1,-(SP)            ;Icon code, Expansion card 4
        jsr     AIcon_Draw          ;Draw Expansion card picture in box
        move.W  #580,-(SP)          ;x1
        move.W  #60,-(SP)           ;y1
        move.w  #'4',-(sp)          ;Character
        jsr     Paint_Ch            ;Place character on the screen
;
        movem.l (sp)+,d0-d7/a0-a6    ;Restore the world
        jmp     (a0)
;
;
;
CPU     .Byte   4
        .ASCII  'CPU '
        .Byte   0
;
        .END
```

To: Gary Martin, Paul Baker, Ann Nunziate 05/20/83
From: George Cossey
Subject: Lisa 1.75 ERS, 2nd pass

The following is a list of questions, comments, and request for more information regarding the Lisa 1.75 system.

Lisa 1.75

///// TIMER ///////

8253 Timer chip.
Three timers are not enough.
What about 2 timer chips?

/////// BOOT ROM ///////

Where is there a better place for the serial number?
Being tied to the Boot Rom makes updates for the Boot Roms VERY hard to do.

In which Rom is the hardware interface module?

/////// PARALLEL PORT /////////

Need details on the parallel port driving the hard disk.

////// SERIAL PORTS ////////

Are the serial ports to be IDENTICAL to the current ones? Every part of the circuit.

Add buffer that can be disabled to output of serial ports. This way the self test in the SCC chip can be used.

///////// 871 /////////

IWM chip. Not my area, need for Gary P. to look at it.

//////// CPU BOARD ////////

Reorder of interrupt levels? If doing disk transfers and the serial bus asks for attention, then trouble.
Why is SCC so high?

Get more of an address for a parity error. Currently only get MSB bits, to do better diagnostics and error correction the complete address is needed.

Where are the following status bits?
Soft error (ECC)
Hard Error (parity)
Bus timeout
Horz sync.

DIAG1 and DIAG2 equivalents?

DMA allowed? Where is DMA register address?

Is the Contrast/Volume latch readable? Reverse contrast value to normal 0 is dark?

Why not latch in the physical address on a parity error? MUCH easier for software.

Where is the reset switch?

//////// EXPANSION ////////

Are all current expansion slot boards compatible? Two modes, one for new fast boards and one for Lisa 1 boards?

Three expansion slots?

/////// TEST CARDS ////////

A Test card designed for the two "new" slots?

CPU board to have connector designed for equivalent to IOPTC? Connector for voltage measurements? Other connections for diagnostic purposes?

Test card as part of the design?

/////// COPS ////////

Differences in current 6522/COPS combination and new COPS?

COPS will hold a boot device. New commands to set and read it?

What does the Power Fail line to the COPS do?

Add capability of any COPS event to be NMI?

Use 871 button as the SW2 input line for the COPS?

Add capability of reading NMI key?

Add capability of reading keyboard indicators?

Which registers can be read and of what use is that?

What is in returned keyboard status? (command 0000 0100)

Are mouse and clock codes the same?

Handle NMI condition even though FIFO is full? Currently, if FIFO is
full then NMI condition is ignored.

COPS can lose commands sent to it? 7us.

The 1 second limit before an NMI is generated, NO don't do this to us.
NO don't reset the system automaticelly.

Questions on flow chart.

```
program LisaTalk;

uses  {$U UNITS:UNITLIB}  STDunit, SCCunit, ASMunit, TASKunit;

const
  RegBase = $400;               {reg save area}

type
  aRegset =
    packed record
      Dregs : array [0..7] of long;
      Aregs : array [0..7] of long;
      IVtype, flags : byte;
      SR : word;
      PC : long;
      FC : word;                   {function code on BusError}
      AA : long;                   {address on error}
      IR : word;                   {instruction reg}
      USP : long;                  {user stack ptr}
      CSP : long                   {call stack ptr}
    end;
  TTalkCmd = ( tEcho, tGo, tLoad, tDump,
               tRdBytes, tWrBytes, tRdWords, tWrWords, tRdLongs, tWrLongs );
  TTalkData =
    packed record case integer of
       0:  ( regs : aRegset );
       8:  ( byts : packed array [0..127] of byte );
      16:  ( wrds : array [0..63] of word );
      32:  ( lngs : array [0..31] of long )
    end;
  TDataLength = ( tlByte, tlWord, tlLong );
  aTalkPkt =
    record
      tCmd : TTalkCmd;
      tCount : integer;
      tAddr : long;
      tData : TTalkData
    end;
  charset = set of char;
  aMessage = string[39];

var
  cmdD : integer;  CameBack, ACKed : boolean;
  CmdSet, OctSet : charset;  theDataLength : TDataLength;
  SendPkt, RecvPkt : aTalkPkt;
  theRegs : aRegset;

procedure Wait;
var  C : char;  i : integer;
begin
  CameBack := false;  ACKed := false;
  repeat
    until AvailB or KeyPress;
   if KeyPress
    then
     getC(C,false);
   if RecvBC(C)
    then
     begin
     CameBack := true;
      if C = ACK
       then
        ACKed := true
       else
        begin
        write( 'bad response' );  NL;
        for i := 1 to 1000 do  ;
        FlushB
        end
     end
    else
     begin  write( 'no response' );  NL  end
end;

function cdiv( A, B : integer ) : integer;
begin
  cdiv := (A+B-1) div B
end;

function getH( var L : long; f : boolean ) : boolean;
var  d : nybl;
begin
  L := 0;  cmdD := 0;
   if f
    then
     begin  write( ': ' );  ReadCmdS  end;
   while cmdC = SP do  nxtC;
   if getH1(d)
     then
       begin
       repeat
         cmdD := cmdD+1;
         L := L*$10 + d;
         until not getH1(d);
       getH := true
       end
    else
      getH := false
end;

procedure Init;
var i : integer;
begin
  STDinit( 'LisaTalk 1.5 (18-Nov-82)' );
  CmdSet := [ 'D', 'G', 'L', 'R', 'S', 'Q' ];
  OctSet := [ '0'..'7' ];
  SCCinit(9600,2)
end;

procedure Fini;
begin
  SCCfini
end;

procedure getCinSet( var C : char;  S : aString;  Valid : charset );
begin
  repeat
    write( S, ': ' );  getC(C,true);  NL;
    if C = CR
     then
      exit(getCinSet)
    until C in Valid
end;

function getAddr( var A : long ) : boolean;
begin
  write( 'address' );
  A := 0;  theDataLength := tlByte;
  if getH( A, true )
   then
    begin
    getAddr := true;
    if cmdC in [ 'W', 'L' ]
     then
      begin
      A := Land( A, -2 );
      case cmdC of
        'W':  theDataLength := tlWord;
        'L':  theDataLength := tlLong
        end;
      nxtC
      end
    end
   else
    getAddr := false
end;

function getCount( var C : integer ) : boolean;
begin
  write( 'count: ' );  readCmdS;
  if cmdL > 0
   then
    getCount := getH4( C )
   else
    getCount := false
end;

procedure doLoad;
var
  i, l, Len : integer;  A : long;  found : boolean;  P : TblockP;
  Lname : aMessage;  Lfile : file;
  image : packed array [0..16383] of byte;
begin
  write( 'enter filename[.OBJ] - ' );  readln( Lname );  L := length(Lname);
  if L = 0
   then
    exit(doLoad);
  Lname := concat( Lname, '.OBJ' );
  {$I-}  reset( Lfile, Lname );  {$I+}
  if IOresult <> 0
   then
    begin  write( 'error opening ', Lname );  exit(doLoad)  end;
  Len := blockread( Lfile, image, 32 ) * 512 -1;
  i := 0;  found := false;
  repeat
    l := image[i+2]*$100 + image[i+3];
    if ( l <= 0 ) or ( (i+l) > Len )
     then
      begin  write( 'bad file' );  exit(doLoad)  end
    else
    if image[i] = $85
     then
      found := true
     else
      i := i + l
    until found;
  P := pointer( ord4(@image) + i + 8 );
  write( 'load address' );
  if not getH(A,true)
   then
    begin  write(BEL);  exit(doLoad)  end;
  with SendPkt do
```

```
    begin  tCmd := tLoad;  tCount := 1-8;  tAddr := A;  end;
  PutPktB( @SendPkt, 8 );
  Wait;
  if ACKed
   then
    begin
    PutPktB( P, 1-8 );
    Wait
    end
end;

procedure doDisplay;
var
  C : char;  rcount, i, K : integer;  A : long;
  data : TTalkData;
begin
  write( 'Display' );  NL;
  if getAddr(A)
   then
    SendPkt.tAddr := A
   else
    exit(doDisplay);
  with SendPkt do
    begin
    if not getCount(tCount)
     then
      tCount := 128
     else
      if tCount > 128
       then
        tCount := 128;
    case theDataLength of
      tlByte:  begin  tCmd := tRdBytes;  K := tCount  end;
      tlWord:  begin  tCmd := tRdWords;  K := cdiv(tCount,2)  end;
      tlLong:  begin  tCmd := tRdLongs;  K := cdiv(tCount,4)  end
      end
    end;
  PutPktB( @SendPkt, 8 );
  repeat
    if KeyPress
     then
      begin  getC(C,false);  exit(doDisplay)  end
    until GetPktB( @Data, rcount );
  if rcount <> SendPkt.tCount
   then
    begin  write( 'wrong length' );  NL  end;
  with Data do
    for i := 0 to K-1 do
      case theDataLength of
        tlByte:
          begin
          if (i mod 16)=0
           then
            begin  NL;  putH8( A+i, ':' )  end;
          write(' ');  putH2( byts[i], NUL )
          end;
        tlWord:
          begin
          if (i mod 8)=0
           then
            begin  NL;  putH8( A+i*2, ':' )  end;
          write( ' ' );  putH4( wrds[i], NUL )
          end;
        tlLong:
          begin
          if (i mod 4)=0
           then
            begin  NL;  putH8( A+i*4, ':' )  end;
          write( ' ' );  putH8( lngs[i], NUL )
          end
        end(case C);
  NL
end;

procedure getRegs;
var  rcount : integer;
begin
  with SendPkt do
    begin
    tCmd := tRdWords;  tCount := $58;  tAddr := RegBase
    end;
  PutPktB( @SendPkt, 8 );
  if not GetPktB( @theRegs, rcount )
   then
    begin  write( 'timeout' );  NL;  exit(getRegs)  end
   else
    if rcount <> $58
     then
      begin  write( 'wrong length, ', rcount );  NL  end
end;

procedure doRegs;
var  rcount, i : integer;
begin
  getRegs;
  with theRegs do
```

```
    begin
    write('PC='); putH8(PC,SP);  write('SR='); putH4(SR,SP);
    write('US='); putH8(USP,CR);
    for i := 0 to 7 do
      begin
      write('D',i:1,'='); putH8(Dregs[i],SP);
      if (i mod 4)=3 then NL
      end;
    for i := 0 to 7 do
      begin
      write('A',i:1,'='); putH8(Aregs[i],SP);
      if (i mod 4)=3 then NL
      end
    end
end;

procedure setReg;
var
  C, C1, C2 : char;  Rname : string[3];
  i : integer;  R : long;  ok : boolean;
begin
  write( 'setReg' ); NL; ok := false;
  nxtC; C1 := cmdC;  nxtC; C2 := cmdC;
  Rname := 'xx'; Rname[1] := C1;  Rname[2] := C2;
  with theRegs do
    begin
    case C1 of
      'A','D':
        if C2 in OctSet
         then
          begin
           i := ord(C2)-ord('0');
           if getH(R,true)
            then
             begin
              if C1 = 'D'
               then
                Dregs[i] := R
               else
                Aregs[i] := R;
             ok := true
             end
          end;
      'P':
        if C2 = 'C'
         then
          if getH(R,true)
           then
            begin PC := R;  ok := true  end;
      'S':
        if C2 = 'R'
         then
          if getH(R,true)
           then
            if cmdD = 4
             then
              begin SR := R;  ok := true  end;
      'U':
        if C2 = 'S'
         then
          if getH(R,true)
           then
            begin USP := R;  ok := true  end
      end{case C1};
    end{with theRegs};
  if not ok
   then
    begin write(BEL); exit(setReg) end;
  with SendPkt do
    begin
    tCmd := tWrWords;  tCount := $58;  tAddr := RegBase;
    with tData.regs do
      begin
      for i := 0 to 7 do
        begin
        Dregs[i] := theRegs.Dregs[i];
        Aregs[i] := theRegs.Aregs[i]
        end;
      SR := theRegs.SR;  PC := theRegs.PC;  USP := theRegs.USP
      end
    end;
  PutPktB( @SendPkt, $60 );
  Wait
end;

procedure doSet;
type
  Tkludge =
    record case integer of
      32: ( l : long );
      16: ( w0, w1 : word );
       8: ( b : packed array [0..3] of byte )
    end;
var
  C : char;  rcount, i, K : integer;  A : long;  X : Tkludge;
begin
```

```
  write( 'Set' );  NL;
  if not getAddr(A)
   then
    begin
    if cmdC = 'R'
     then
      setReg
     else
      write(BEL);
    exit(doSet)
    end;
  with SendPkt do
    begin
    tAddr := A;  k := 0;
    while getH( X.l, k=0 ) do  with tData do
      begin
      case theDataLength of
        tlByte:
          begin
          if cmdD > 6
           then
            begin  byts[k] := X.b[0];  k := k+1  end;
          if cmdD > 4
           then
            begin  byts[k] := X.b[1];  k := k+1  end;
          if cmdD > 2
           then
            begin  byts[k] := X.b[2];  k := k+1  end;
          byts[k] := X.b[3]
          end;
        tlWord:
          begin
          if cmdD > 4
           then
            begin  wrds[k] := X.w0;  k := k+1  end;
          wrds[k] := X.w1
          end;
        tlLong:  lngs[k] := X.l
        end;
      k := k + 1
      end;
    case theDataLength of
      tlByte:  begin  tCmd := tWrBytes;  tCount := k  end;
      tlWord:  begin  tCmd := tWrWords;  tCount := k*2  end;
      tlLong:  begin  tCmd := tWrLongs;  tCount := k*4  end
      end
    end;
  PutPktB( @SendPkt, 8 + SendPkt.tCount );
  Wait
end;

procedure doGo;
var  C : char;  A : long;
begin
  write( 'Go' );  NL;
  with SendPkt do
    begin
    if getH(A,true)
     then
      tAddr := A
     else
      tAddr := 0;
    tCmd := tGo
    end;
  PutPktB( @SendPkt, 8 );
  Wait;
  if ACKed
   then
    doRegs
end;

procedure doCommands;
var
  done : boolean;
  C : char;
begin
  done := false;
  repeat
    NL;
    if AvailA
     then
      begin
      write( '[' );
      while RecvAC( C ) do  write(C);
      write( ']' );  NL
      end;
    getCinSet( C, 'D(isplay) G(o) L(oad) Q(uit) R(egs) S(et)', CmdSet );
    flushB;
    if C = CR
     then
     else
      case C of
        'D':  doDisplay;
        'G':  doGo;
        'L':  doLoad;
        'R':  doRegs;
        'S':  doSet;
        'Q':  done := true
        end
    until done
end;

begin
  Init;
  doCommands;
  Fini
end.
```

```
UNIT  Menu;

{   Edit date:  05/10/83 G.Cossey
       05/10/83 G.Cossey - Changed alert format, now buttons are CONTINUE
              and CANCEL with each one described in the alert.  }


   INTERFACE
uses {$U obj:XYGRAPHICS.obj} XYGRAPHICS,
     {$U obj:hwint.obj} hwint;
{ Also needs hwintl,DRAWIT, and PDRAWIT for linking }

const
   BUTTONS = 64;
   ABUTTONS = 5;
   MBUTTON = $06;


var
   x1,x2,y1,y2,Selected :array[1..BUTTONS] of integer;
   Ax1,Ax2,Ay1,Ay2,ASelected :array[1..ABUTTONS] of integer;
   ATopButton,TopButton,MX,MY,Current_Item: integer;
   GOTBUTTON,EVENT:boolean;
   KEY:keyevent;


      Procedure ClearButtons;
      Procedure SelectButton(var Item:integer);
      Procedure AddButton(Number_Button,xx1,yy1,xx2,yy2:integer);
      Procedure SubButton(Number_Button:integer);
      Procedure InvertButton(Number_Button:integer);
      Procedure RemoveButton(First_Button,Last_Button:integer);
      Procedure WhiteButton(First_Button,Last_Button:integer);
      Procedure AAddButton(Number_Button,xx1,yy1,xx2,yy2:integer);
      Procedure AClearButtons;
      Procedure ASelectButton(var Item:integer);
      procedure ALERT(var AError:integer; MainLine:string255);


   IMPLEMENTATION

      procedure PSaveAlert;external;
      procedure PRestoreAlert;external;

{_____________________________________________________________}

Procedure ClearButtons;
{ Clear any buttons in use }
var
   INDEX: integer;

begin
TopButton:= 1;
for INDEX:= 1 to BUTTONS do
   begin
      X1[INDEX]:= 0;  X2[INDEX]:= 0;
  end;
end;

{_____________________________________________________________}
Procedure CheckMark;
var
   INDEX: integer;
   mask,cursor:array[1..12] of integer;

begin
for INDEX:= 1 to 12 do
   mask[INDEX]:=0;
cursor[1] := $0001;  cursor[2] := $0003;  cursor[3] := $0006;  cursor[4] := $000C;
cursor[5] := $0018;  cursor[6] := $0030;  cursor[7] := $C060;  cursor[8] := $60C0;
cursor[9] := $3180;  cursor[10]:= $1800;  cursor[11]:= $0E00;  cursor[12]:= $0400;
CursorImage(4,9,12,@cursor[1],@mask[1]);
end;

{______________________________________________________}

Procedure SelectButton;
var
   INDEX,Current,Ix1,Ix2,Iy1,Iy2: integer;
   Ck_Active:boolean;


   Procedure  Do_Cursor;
   var
      INDEX: integer;

   begin
   Current:= 0;

   {See if over any buttons}
   INDEX:= 1;
   repeat
     if (MX>(X1[INDEX])-10) and (MX<(X2[INDEX])+10) then
        begin
           if (MY>(Y1[INDEX])-10) and (MY<(Y2[INDEX])+5) then
              Current:= INDEX;
```

```
         end;
      INDEX:= INDEX+1;
   until (Current<>0) or (INDEX>TopButton);

   if (Current=0) and Ck_Active then
      begin
         ArrowMouse;
         Ck_Active:= false;
      end
   else if (Current<>0) and not(Ck_Active) then
      begin
         Ck_Active:= true;
         CheckMark;
      end;
   end;



   Procedure HighLight;
   { Highlight any buttons pressed }
   var
      INDEX: integer;

   begin
   Current:= 0;

   {See if over any buttons}
   INDEX:= 1;
   repeat
      if (MX>X1[INDEX]) and (MX<X2[INDEX]) then
         begin
            if (MY>Y1[INDEX]) and (MY<Y2[INDEX]) then
               Current:= INDEX;
         end;
      INDEX:= INDEX+1;
   until (Current<>0) or (INDEX>TopButton);

   {If not over any buttons then turn off all highlights}
   if (Current=0) then
      begin
         for INDEX:= 1 to TopButton do
            begin
               if (SELECTED[INDEX]<>0) then
                  begin
                     Ix1:=X1[INDEX]+1;  Ix2:=X2[INDEX]-1;
                     Iy1:=Y1[INDEX]+1;  Iy2:=Y2[INDEX]-1;
                     InvertArea(Ix1,Iy1,Ix2,Iy2);
                     SELECTED[INDEX]:= 0;
                  end;
            end;
      end
   else
      begin
         if (SELECTED[Current]=0) then
            begin
               Ix1:=X1[Current]+1;  Ix2:=X2[Current]-1;
               Iy1:=Y1[Current]+1;  Iy2:=Y2[Current]-1;
               InvertArea(Ix1,Iy1,Ix2,Iy2);
               SELECTED[Current]:= 1;
            end;
      end;
   end;



   Procedure ClearLight;
   var
      INDEX,Ix1,Ix2,Iy1,Iy2: integer;

   begin
   for INDEX:= 1 to TopButton do
      begin
         if (SELECTED[INDEX]<>0) then
            begin
               Ix1:=X1[INDEX]+1;  Ix2:=X2[INDEX]-1;
               Iy1:=Y1[INDEX]+1;  Iy2:=Y2[INDEX]-1;
               InvertArea(Ix1,Iy1,Ix2,Iy2);
               SELECTED[INDEX]:= 0;
            end;
       end;
   end;



begin
for INDEX:= 1 to TopButton do
   SELECTED[INDEX]:= 0;
Ck_Active:= false;

repeat
   EVENT:= KeyBdEvent(FALSE,FALSE,KEY);  {Flush que}
until not(EVENT);

ArrowMouse;
Item:= 0;
```

```
repeat
   GOTBUTTON:= FALSE;
   repeat
      MouseLocation(MX,MY);
      Do_Cursor;
      EVENT:= KeyBdEvent(FALSE,FALSE,KEY);   {Get button down}
      GOTBUTTON:= (MBUTTON = KEY.key);
      if EVENT and (Key.State=8) and (Key.ASCII='.') then
         begin
            Item:= 1;
            Exit(SelectButton);
         end;
   until GOTBUTTON;

   GOTBUTTON:= FALSE;
   repeat
      MouseLocation(MX,MY);
      Do_Cursor;
      HighLight;
      EVENT:= KeyBdEvent(FALSE,FALSE,KEY);   {Get button up}
      if EVENT then
         begin
            if (ord(KEY.ascii)=1) then
               GOTBUTTON:= (MBUTTON = KEY.key);
         end;
   until GOTBUTTON;

   for INDEX:= 1 to TopButton do
      if (SELECTED[INDEX]<>0) then
         Item:= INDEX;

   if (Item=0) then
      ClearLight;
until (Item<>0);
Current_Item:= Item;
HourGlass;
end;

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

Procedure AddButton;

begin
if (Number_Button > BUTTONS) then
   Number_Button:= BUTTONS;
if (Number_Button > TopButton) then
   TopButton:= Number_Button;

x1[Number_Button]:= xx1;  x2[Number_Button]:= xx2;
y1[Number_Button]:= yy1;  y2[Number_Button]:= yy2;
MakeBox(xx1,yy1,xx2,yy2);
end;

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

Procedure SubButton;

begin
if (Number_Button > BUTTONS) then
   Number_Button:= BUTTONS;

x1[Number_Button]:= 0;  x2[Number_Button]:= 0;
end;

{****************************************************************************}
Procedure RemoveButton;
var
   INDEX:integer;

begin
for INDEX:= First_Button to Last_Button do
   begin
      x1[INDEX]:= 0;  x2[INDEX]:= 0;
   end;
end;

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

Procedure WhiteButton;
var
   INDEX,Ix1,Ix2,Iy1,Iy2: integer;

begin
for INDEX:= First_Button to Last_Button do
   begin
      Ix1:=X1[INDEX]+1;  Ix2:=X2[INDEX]-1;
      Iy1:=Y1[INDEX]+1;  Iy2:=Y2[INDEX]-1;
      if (INDEX<>Current_Item) then
         WhiteArea(Ix1,Iy1,Ix2,Iy2);
      SELECTED[INDEX]:= 0;
   end;
end;

{****************************************************************************}
Procedure InvertButton;
```

```
var
   Ix1,Ix2,Iy1,Iy2: integer;

begin
Iy1:=Y1[Number_Button]+1;  Iy2:=Y2[Number_Button]-1;
Ix1:=X1[Number_Button]+1;  Ix2:=X2[Number_Button]-1;
{** InvertArea(Ix1,Iy1,Ix2,Iy2);  **}
end;

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

Procedure AAddButton;

begin
if (Number_Button > ABUTTONS) then
   Number_Button:= ABUTTONS;
if (Number_Button > ATopButton) then
   ATopButton:= Number_Button;

Ax1[Number_Button]:= xx1;  Ax2[Number_Button]:= xx2;
Ay1[Number_Button]:= yy1;  Ay2[Number_Button]:= yy2;
MakeBox(xx1,yy1,xx2,yy2);
end;

{*************************************************************************}
Procedure AClearButtons;
{ Clear any buttons in use }
var
   INDEX: integer;

begin
ATopButton:= 1;
for INDEX:= 1 to ABUTTONS do
   begin
      AX1[INDEX]:= 0;  AX2[INDEX]:= 0;
  end;
end;

{______________________________________________________________}

Procedure ASelectButton;
var
   INDEX,Current,Ix1,Ix2,Iy1,Iy2: integer;


   Procedure HighLight;
   { Highlight any buttons pressed }
   var
      INDEX: integer;

   begin
   Current:= 0;

   {See if over any buttons}
   INDEX:= 1;
   repeat
      if (MX>AX1[INDEX]) and (MX<AX2[INDEX]) then
         begin
            if (MY>AY1[INDEX]) and (MY<AY2[INDEX]) then
               Current:= INDEX;
         end;
      INDEX:= INDEX+1;
   until (Current<>0) or (INDEX>ATopButton);

   {If not over any buttons then turn off all highlights}
   if (Current=0) then
      begin
         for INDEX:= 1 to ATopButton do
            begin
               if (ASELECTED[INDEX]<>0) then
                  begin
                     Ix1:=AX1[INDEX]+1;  Ix2:=AX2[INDEX]-1;
                     Iy1:=AY1[INDEX]+1;  Iy2:=AY2[INDEX]-1;
                     InvertArea(Ix1,Iy1,Ix2,Iy2);
                     ASELECTED[INDEX]:= 0;
                  end;
            end;
      end
   else
      begin
         if (ASELECTED[Current]=0) then
            begin
               Ix1:=AX1[Current]+1;  Ix2:=AX2[Current]-1;
               Iy1:=AY1[Current]+1;  Iy2:=AY2[Current]-1;
               InvertArea(Ix1,Iy1,Ix2,Iy2);
               ASELECTED[Current]:= 1;
            end;
      end;
   end;



   Procedure ClearLight;
   var
      INDEX,Ix1,Ix2,Iy1,Iy2: integer;
   begin
   for INDEX:= 1 to ATopButton do
      begin
         if (ASELECTED[INDEX]<>0) then
            begin
               Ix1:=AX1[INDEX]+1;  Ix2:=AX2[INDEX]-1;
               Iy1:=AY1[INDEX]+1;  Iy2:=AY2[INDEX]-1;
               InvertArea(Ix1,Iy1,Ix2,Iy2);
               ASELECTED[INDEX]:= 0;
            end;
        end;
   end;



begin
for INDEX:= 1 to ATopButton do
   ASELECTED[INDEX]:= 0;

repeat
   EVENT:= KeyBdEvent(FALSE,FALSE,KEY);   {Flush que}
until not(EVENT);

ArrowMouse;
Item:= 0;
repeat
   GOTBUTTON:= FALSE;
   repeat
      EVENT:= KeyBdEvent(FALSE,TRUE,KEY);   {Get button down}
      GOTBUTTON:= (MBUTTON = KEY.key);
   until GOTBUTTON;

   GOTBUTTON:= FALSE;
   repeat
      MouseLocation(MX,MY);
      HighLight;
      EVENT:= KeyBdEvent(FALSE,FALSE,KEY);   {Get button up}
      if EVENT then
         begin
            if (ord(KEY.ascii)=1) then
               GOTBUTTON:= (MBUTTON = KEY.key);
         end;
   until GOTBUTTON;

   for INDEX:= 1 to ATopButton do
      if (ASELECTED[INDEX]<>0) then
         Item:= INDEX;

   if (Item=0) then
      ClearLight;
until (Item<>0);
Current_Item:= Item;
HourGlass;
end;


{_______________________________________________________________
 *
        Alert screen.   }
procedure ALERT;
Const
   X_Up_Left = 45;
   Y_Up_Left = 50;

var
  Size,Index,AAx1,Ax1,AAy1,Ay1,AAy2,Ay2,Item: integer;
  Line1,Line2,Line3,Line4: string[45];


begin
Line1:= ' '; Line2:= ' '; Line3:= ' '; Line4:= ' ';

Size:= length(MainLine);
if (Size<46) then
   Line1:= copy(MainLine,1,Size)
else if (Size<91) then
   begin
      Line1:= copy(MainLine,1,45);
      Size:= Size - 45;
      Line2:= copy(MainLine,46,Size);
   end
else if (Size<136) then
   begin
      Line1:= copy(MainLine,1,45);
      Size:= Size - 90;
      Line2:= copy(MainLine,46,45);
      Line3:= copy(MainLine,91,Size);
   end
else
   begin
      Line1:= copy(MainLine,1,45);
      Size:= Size - 135;
      Line2:= copy(MainLine,46,45);
      Line3:= copy(MainLine,91,45);
      Line4:= copy(MainLine,136,Size);
   end;

CursorHide;

AClearButtons;
PSaveAlert;
WhiteArea(X_Up_Left,Y_Up_Left,X_Up_Left+630,Y_Up_Left+110);
MakeBox(X_Up_Left,Y_Up_Left,X_Up_Left+630,Y_Up_Left+110);
AAddButton(1,X_Up_Left+545,Y_Up_Left+10,X_Up_Left+620,Y_Up_Left+25);
TypeText(X_Up_Left+556,Y_Up_Left+15,'Continue');
AAddButton(2,X_Up_Left+545,Y_Up_Left+85,X_Up_Left+620,Y_Up_Left+100);
TypeText(X_Up_Left+562,Y_Up_Left+90,'Cancel');
TypeText(X_Up_Left+40,Y_Up_Left+50,'S T O P');
InvertArea(X_Up_Left+35,Y_Up_Left+30,X_Up_Left+95,Y_Up_Left+80);
Ax1:= X_Up_Left+34;  Ay1:=  Y_Up_Left+31;  Ay2:= Y_Up_Left+79;
AAx1:=X_Up_Left+96;  AAy1:= Ay1;  AAy2:= Ay2;
for Index:= 1 to 10 do
   begin
      VLine(Ax1,Ay1,Ay2);  Ax1:= Ax1-1;
      VLine(Ax1,Ay1,Ay2);  Ax1:= Ax1-1;
      Ay1:= Ay1+1;  Ay2:= Ay2-1;
      VLine(AAx1,AAy1,AAy2);  AAx1:= AAx1+1;
      VLine(AAx1,AAy1,AAy2);  AAx1:= AAx1+1;
      AAy1:= AAy1+1;  AAy2:= AAy2-1;
   end;
TypeText(X_Up_Left+162,Y_Up_Left+20,LINE1);
TypeText(X_Up_Left+162,Y_Up_Left+30,LINE2);
TypeText(X_Up_Left+162,Y_Up_Left+40,LINE3);
TypeText(X_Up_Left+162,Y_Up_Left+50,LINE4);
TypeText(X_Up_Left+162,Y_Up_Left+80,'To proceed with the tests, click Continue.');
TypeText(X_Up_Left+162,Y_Up_Left+90,'To cancel the tests, click Cancel.');
CursorDisplay;
ASelectButton(AError);
CursorHide;
PRESTOREALERT;
CursorDisplay;
end;


END.
```

```
; Edit Date: 07/14/83
;
;________________________________________________________________________
;
; File:      RBASIC.ASM.TEXT
;
; Function: Assembly graphics routines, called by PASCAL unit BASICS or
;           directly from the Boot ROM.
;
; Input Parameters:
;      Refer to function to be called.
;
; Output Parameters:
;      Refer to function to be called.
;
; Use:  This file is to be used as an extension of PASCAL to provide
;       graphic features from LISA PASCAL.  It is to be assembled and
;       linked to the PASCAL program along with the unit BASICS.
;       Refer to the functions used for the correct PASCAL external declaration.
;
;
;===========================================================================
;
        .PROC    JUMPTABLE,0
        .ref     DeskTop,HLINE,VLINE,ALINE,ASquared_Box
        .ref     AWhite_Box,ABlack_Box,AGrey_Box,ALGrey_Box,AInvert_Box
        .ref     Paint_String,Paint_Ch,ASize_String,ADraw_Integer,ADraw_Hex
        .ref     ADraw_LHex,AFolder,ADialog,AIcon_Draw
;
        jmp      DeskTop
        jmp      HLINE
        jmp      VLINE
        jmp      ALINE
        jmp      ASquared_Box
        jmp      AWhite_Box
        jmp      ABlack_Box
        jmp      AGrey_Box
        jmp      ALGrey_Box
        jmp      AInvert_Box
        jmp      Paint_String
        jmp      Paint_Ch
        jmp      ASize_String
        jmp      ADraw_Integer
        jmp      ADraw_Hex
        jmp      ADraw_LHex
        jmp      AFolder
        jmp      ADialog
        jmp      AIcon_Draw
;
;
;===========================================================================
;===========================================================================
;
        .PROC     RBASIC,0
        .def      DeskTop,HLINE,VLINE,ALINE,ASquared_Box
        .def      AWhite_Box,ABlack_Box,AGrey_Box,ALGrey_Box,AInvert_Box
        .def      Paint_String,Paint_Ch,ASize_String,ADraw_Integer,ADraw_Hex
        .def      ADraw_LHex,AFolder,ADialog,AIcon_Draw
;
XLegal            ;Make legal X value
;    a4 = x value to test and make legal
;    called directly from assembly
;
        movem.l d0-d1/a1,-(SP)  ;save required regs
        move.l  a4,d0           ;X to working register
        and.l   #$000003ff,d0   ;x, Make legal for gross errors, $2cf is 719.
        move.w  #719,d1         ;Get right allowed edge
        cmp.w   d1,d0           ;To right of allowed edge?
        bmi     @1              ;...no, let go as is
        move.w  d1,d0           ;...yes, make legal
;
@1      move.l  d0,a4
        movem.l (SP)+,d0-d1/a1     ;...restore the reg
        rts
;
;==================================================================
;
YLegal                          ;Make legal Y value
;    a4 = y value to test and make legal
;    called directly from assembly
;
        movem.l d0-d1/a1,-(SP)  ;save required regs
        move.l  a4,d0           ;Y to working register
        and.l   #$000001ff,d0   ;y, Make legal for gross errors
        move.w  #363,d1         ;Get bottom allowed edge
        cmp.w   d1,d0           ;Below allowed edge?
        bmi     @1              ;...no, let go as is
        move.w  d1,d0           ;...yes, make legal
;
@1      move.l  d0,a4
        movem.l (SP)+,d0-d1/a1     ;...restore the reg
        rts
;
;==================================================================
;
DeskTop                    ; Draw a blank desktop
```

```
; Name:  DESKTOP
;  Procedure DeskTop;
;
;  Function: Draws a blank desktop
;
;
        MOVE.L  (SP)+,A0        ;save return address
;
        movem.l d0-d7/a0-a6,-(sp)  ;Save everyone
;
        MOVE.W  #0,-(SP)        ;x1
        MOVE.W  #20,-(SP)       ;y1
        MOVE.W  #719,-(SP)      ;x2
        MOVE.W  #363,-(SP)      ;y2
        bsr     AGrey_Box
;
        MOVE.W  #0,-(SP)        ;x1
        MOVE.W  #0,-(SP)        ;y1
        MOVE.W  #719,-(SP)      ;x2
        MOVE.W  #20,-(SP)       ;y2
        bsr     AWhite_Box
;
        move.w  #20,-(sp)           ;Ly1
        move.w  #0,-(sp)            ;X1
        move.w  #719,-(sp)          ;X2
        bsr     HLine               ;HLine(Ly1,X1,X2)
;
        movem.l (sp)+,d0-d7/a0-a6   ;Restore the world
        jmp     (a0)
;
;==================================================================
;
HLINE                   ; Draw a horizontal line.
; Name:  HLINE
;  Procedure HLine(Y1,X1,X2:integer);
;
;  Function: Draws a line, horz
;
;
        MOVE.L  (SP)+,A0        ;save return address
        MOVE.W  (SP)+,d2        ;x2
        MOVE.W  (SP)+,d1        ;x1
        MOVE.W  (SP)+,d0        ;y1
;
        movem.l d0-d7/a0-a6,-(sp)  ;Save everyone
;
; Draw a horizontal line.
;    d0 = y value, y1
;    d1 = starting x value, x1
;    d2 = ending x value, x2
;
        move.l  $160,a5                 ;Get video page address
        move.w  d0,a4
        bsr     YLegal
        move.l  a4,d0                   ;Y1 is now legal
        move.w  d1,a4
        bsr     XLegal
        move.l  a4,d1                   ;X1 is now legal
        move.w  d2,a4
        bsr     XLegal
        move.l  a4,d2                   ;X2 is now legal
        cmp.w   d1,d2
        bpl     @1
        move.w  d1,d4                   ;Swap
        move.w  d2,d1
        move.w  d4,d2
;
@1      mulu    #90,d0                  ;Find starting row address, in a5
        adda    d0,a5                   ;... y1*90 + Video Page start
;
        clr.l   d5
        clr.l   d3                      ;Init Pixel counter
@2      clr.l   d0                      ;Init Pixel per byte counter
        move.b  #$07,d0
@3      cmp.w   d1,d3                   ;In window on left side?
        bmi     @4
        cmp.w   d3,d2                   ;In window on right side?
        bmi     @4
        bset    d0,(a5)                 ;Set bit since in window
@4      sub.w   #1,d0                   ;Go to next bit in the byte
        add.w   #1,d3                   ;...next pixel count
        cmp.w   #0,d0                   ;...End of this byte?
        bmi     @5                      ;......yes, go to next byte
        bra     @3                      ;......no, continue in byte
;
@5      adda    #1,a5                   ;Go to next byte
        add.w   #1,d5                   ;...increment byte counter
        cmp.w   #90,d5                  ;...at end of row?
        bne     @2                      ;...no, continue
;
        movem.l (sp)+,d0-d7/a0-a6   ;Restore the world
        jmp     (a0)
;
;========================================================================
;
VLINE                       ; Draw a vertical line.
```

```
; Name:  VLINE
;  Procedure VLine(X1,Y1,Y2: integer);EXTERNAL;
;
;  Function: Draws a line, vertical
;
;
        MOVE.L  (SP)+,A0         ;save return address
        MOVE.W  (SP)+,d2         ;y2
        MOVE.W  (SP)+,d1         ;y1
        MOVE.W  (SP)+,d0         ;x1
;
        movem.l d0-d7/a0-a6,-(sp)
;
; Draw a vertical line.
;    d0 = x value
;    d1 = starting y value
;    d2 = ending y value
;
        move.l  $160,a5                    ;Get video page address
        move.w  d0,a4
        bsr     XLegal
        move.l  a4,d0                      ;X1 is now legal
        move.w  d1,a4
        bsr     YLegal
        move.l  a4,d1                      ;Y1 is now legal
        move.w  d2,a4
        bsr     YLegal
        move.l  a4,d2                      ;Y2 is now legal
        cmp.w   d1,d2
        bpl     @1
        move.w  d1,d4                      ;Swap
        move.w  d2,d1
        move.w  d4,d2
;
@1      move.l  a5,a4
        mulu    #90,d1                     ;Find starting row address, in a5
        adda    d1,a5
        mulu    #90,d2                     ;...find ending row address, in a4
        adda    d2,a4
;
        divs    #8,d0                      ;Find column offset
        move.w  d0,d4                      ;...get quotient
        swap    d0
        and.l   #$7,d0                     ;...get remainder
        not.b   d0
        and.l   #$7,d0
;
@2      cmp.l   a5,a4                      ;To last row yet?
        bmi     @10                        ;...yes, exit
        move.l  a5,a3
        adda    d4,a3
        bset    d0,(a3)                    ;Set bit
        adda    #90,a5                     ;Go to next row
        bra     @2
;
@10     movem.l (sp)+,d0-d7/a0-a6   ;Restore the world
        jmp     (a0)
;
;
;==========================================================================
;
;
ALINE
; Name:  ALINE
;   Procedure ALine(X1,Y1,X2,Y2,Line_Width: integer);
;
;  Function: Draws a line, horz or vertical, else does nothing.
;
;
        MOVE.L  (SP)+,A0         ;save return address
        MOVE.W  (SP)+,A2         ;Line Width
        MOVE.W  (SP)+,d2         ;y2
        MOVE.W  (SP)+,a1         ;x2
        MOVE.W  (SP)+,d1         ;y1
        MOVE.W  (SP)+,d0         ;x1
;
        movem.l d0-d7/a0-a6,-(sp)
;
        move.w  a2,d3            ;Line Width check
        and.w   #$00ff,d3        ;Max allowed is 255 for safety.
        beq     @5
;
        move.w  d0,d4                ;X1
        move.w  d1,d5                ;Y1
        move.w  a1,d6                ;X2
        move.w  d2,d7                ;Y2
;
        move.w  #1,d3                ;Doing_Line:= 1
;
        cmp.w   d4,d6                ; if (X1=X2) then
        bne     @2                   ;    begin
;
@1                                   ;repeat
        move.w  d4,-(sp)             ;Lx1
        move.w  d5,-(sp)             ;Y1
        move.w  d7,-(sp)             ;Y2
        bsr     VLine               ;VLine(Lx1,Y1,Y2);
        add.w   #1,d4               ; Lx1:= Lx1+1;
        sub.w   #1,d3               ; Doing_Line:= Doing_Line+1
        cmp.w   #0,d3
        bne     @1                  ;until (Doing_Line>Line_Width)
        bra     @5
;
@2      cmp.w   d5,d7               ;else if (Y1=Y2) then
        bne     @4                  ;    begin
;
@3                                  ;repeat
        move.w  d5,-(sp)            ;Ly1
        move.w  d4,-(sp)            ;X1
        move.w  d6,-(sp)            ;X2
        bsr     HLine               ;HLine(Ly1,X1,X2)
        add.w   #1,d5               ;Ly1:= Ly1+1
        sub.w   #1,d3               ;Doing_Line:= Doing_Line+1
        cmp.w   #0,d3
        bne     @3                  ;until (Doing_Line>Line_Width)
        bra     @5
;
@4                                  ;else
        move.w  #5,d0
        move.w  d0,-(sp)            ;Ly1
        move.w  #10,d0
        move.w  d0,-(sp)            ;X1
        move.w  #600,d0
        move.w  d0,-(sp)            ;X2
        bsr     HLine               ;HLine(5,10,600);  (Error result)
;
@5
        movem.l (sp)+,d0-d7/a0-a6   ;Restore the world
        jmp  (a0)
;
;
;============================================================================
;
ASquared_Box
; Name:  ASquared_Box
;       Procedure ASquared_Box(X1,Y1,X2,Y2,Line_Width:integer);
;
;  Function: Draw a delta line to new position from current position.
;

;
        MOVE.L  (SP)+,A0            ;save return address
        MOVE.W  (SP)+,a1            ;Line Width
        MOVE.W  (SP)+,a2            ;y2
        MOVE.W  (SP)+,d2            ;x2
        MOVE.W  (SP)+,d1            ;y1
        MOVE.W  (SP)+,d0            ;x1
;
        movem.l d0-d7/a0-a6,-(sp)
;
        move.w  d0,d4
        move.w  d1,d5
        move.w  d2,d6
        move.w  a2,d7
        move.w  a1,a4
;
        MOVE.W  d4,-(SP)            ;x1
        MOVE.W  d5,-(SP)            ;y1
        MOVE.W  d6,-(SP)            ;x2
        MOVE.W  d5,-(SP)            ;y1
        MOVE.W  a4,-(SP)            ;Line Width
        bsr     ALine               ;Upper-left to upper-right
;
        MOVE.W  d4,-(SP)            ;x1
        MOVE.W  d7,-(SP)            ;y2
        MOVE.W  d6,-(SP)            ;x2
        MOVE.W  d7,-(SP)            ;y2
        MOVE.W  a4,-(SP)            ;Line Width
        bsr     ALine               ;Upper-right to Lower-right
;
        MOVE.W  d4,-(SP)            ;x1
        MOVE.W  d5,-(SP)            ;y1
        MOVE.W  d4,-(SP)            ;x1
        MOVE.W  d7,-(SP)            ;y2
        MOVE.W  a4,-(SP)            ;Line Width
        bsr     ALine               ;Lower-right to Lower-Left
;
        MOVE.W  d6,-(SP)            ;x2
        MOVE.W  d5,-(SP)            ;y1
        MOVE.W  d6,-(SP)            ;x2
        MOVE.W  d7,-(SP)            ;y2
        MOVE.W  a4,-(SP)            ;Line Width
        bsr     ALine               ;Lower-left to Upper-left
;
        movem.l (sp)+,d0-d7/a0-a6   ;Restore the world
        jmp     (a0)
;
;
;============================================================================
;
AWhite_Box
; Name:  AWhite_Box
;       Procedure AWhite_Box(X1,Y1,X2,Y2: integer );
;
;  Function: Fill an area with white.
;
;
        move.w  #$0000,d0
        move.w  #$0000,d1
        bra     FillArea
;
;==========================================================================
;
ABlack_Box
; Name:  ABlack_Box
;       Procedure ABlack_Box(X1,Y1,X2,Y2: integer );
;
;  Function: Fill an area with black
;
;
        move.w  #$00ff,d0
        move.w  #$00ff,d1
        bra     FillArea
;
;==========================================================================
;
AGrey_Box
; Name:  AGrey_Box
;       Procedure AGrey_Box(X1,Y1,X2,Y2: integer );
;
;  Function: Fill an area with grey.
;
;
        move.w  #$00aa,d0
        move.w  #$0055,d1
        bra     FillArea
;
;==========================================================================
;
ALGrey_Box
; Name:  ALGrey_Box
;       Procedure ALGrey_Box(X1,Y1,X2,Y2: integer );
;
;  Function: Draw a delta line to new position from current position.
;
;
        move.w  #$00cc,d0
        move.w  #$0033,d1
        bra     FillArea
;
;================================================
;
FillArea                        ;fills an area
; Name:  FillArea
;
;
        lea     F1Mask,a1
        move.w  d0,(a1)
        lea     F2Mask,a1
        move.w  d1,(a1)
;
        MOVE.L  (SP)+,A0        ;save return address
        MOVE.W  (SP)+,a1        ;y2
        MOVE.W  (SP)+,d1        ;x2
        MOVE.W  (SP)+,d2        ;y1
        MOVE.W  (SP)+,d0        ;x1
;
        movem.l a0-a6/d0-d7,-(SP)      ;save required regs
        move.l  $160,a5                  ;Get video page
        move.w  a1,d3                    ;y2
        move.w  d0,a4
        bsr     XLegal
        move.l  a4,d0                    ;x1
        move.w  d1,a4
        bsr     XLegal
        move.l  a4,d1                    ;x2
        move.w  d2,a4
        bsr     YLegal
        move.l  a4,d2                    ;y1
        move.w  d3,a4
        bsr     YLegal
        move.l  a4,d3                    ;y2
        cmp.w   d0,d1                    ;See if x1<x2
        bpl     @1
        move.w  d0,d4                    ;...Swap
        move.w  d1,d0
        move.w  d4,d1
@1      cmp.w   d2,d3                    ;See if y1<y2
        bpl     @2
        move.w  d2,d4                    ;...Swap
        move.w  d3,d2
        move.w  d4,d3
;
@2      clr.l   d4
        move.w  d0,d4                    ;X1, find 1st part byte offset
        divs    #8,d4                    ;x1/8= which byte to start in
        clr.l   d7
        move.w  d4,d7                    ;Start offset
```

```
;
        swap    d4
        and.l   #$f,d4                  ;Get remainder for which bit
;
        lea     StartByte,a1
        adda    d4,a1
        move.b  (a1),d6                 ;Get starting byte mask
;
        clr.l   d4
        move.w  d1,d4                   ;X2, find 2nd part byte offset
        divs    #8,d4                   ;x2/8= which byte to end in
        clr.l   d5
        move.w  d4,d5
;
        swap    d4
        and.l   #$f,d4                  ;Get remainder for which bit
;
        lea     EndByte,a1
        adda    d4,a1
        move.b  (a1),d4                 ;Get ending byte mask
;
        move.l  d2,d0
        mulu    #90,d0                  ;Find starting row address, in a5
        adda    d0,a5
        sub.w   d2,d3                   ;y1-y2= number of rows to do
        move.w  d3,d2
        add.w   #1,d2
;
        lea     F2Mask,a1
        move.w  (a1),d3
        move.w  d3,a2
        lea     F1Mask,a1
        move.w  (a1),d3
        move.w  d3,a1
;
        move.w  d5,d3                   ;Get # bytes to do
        sub.w   d7,d3                   ;Only one byte?
        beq     @6                      ;...yes
        sub.w   #1,d3
;
;   Main clear loop
@3      cmp.w   #0,d2                   ;All rows done?
        beq     @11                     ;...yes, exit
;
        move.w  a1,a4                   ;Swap 55 with aa
        move.w  a2,a1
        move.w  a4,a2
;
        move.l  a5,a3
        adda    d7,a3                   ;First byte
        move.b  (a3),d0                 ;...get original
        and.b   d6,d0                   ;...mask
        move.b  d0,(a3)                 ;...replace
        not.b   d6
        move.w  a2,d0                   ;Get gray pattern
        and.b   d6,d0                   ;Mask it
        not.b   d6
        or.b    (a3),d0                 ;Place in memory
        move.b  d0,(a3)
;
        move.l  a5,a3
        adda    d5,a3                   ;Last byte
        move.b  (a3),d0                 ;...get original
        and.b   d4,d0                   ;...mask
        move.b  d0,(a3)                 ;...replace
        not.b   d4
        move.w  a4,d0                   ;Get gray pattern
        and.b   d4,d0                   ;Mask it
        not.b   d4
        or.b    (a3),d0                 ;Place in memory
        move.b  d0,(a3)
;
        clr.l   d1
        move.w  d3,d1                   ;Byte counter
        move.l  a5,a3
        adda    d7,a3                   ;First byte
        adda    #1,a3                   ;second byte
;
@4      cmp.w   #0,d1                   ;At end?
        beq     @42
        move.w  a2,d0                   ;Mask for this byte
        move.b  d0,(a3)+                ;...place mask
        sub.w   #1,d1                   ;Next byte
        bra     @4
;
@42     move.w  a2,d0
        and.b   d6,d0
        or.b    (a3),d0
        move.b  d0,(a3)
;
@5      adda    #90,a5                  ;Go to next row
        sub.w   #1,d2                   ;Decrement row counter
        bra     @3
;
;
@6      and.b   d6,d4                   ;Get mask for remainding bits
```

```
;    Main clear  loop
@7      cmp.w   #0,d2                    ;Row done?
        beq     @11                      ;...yes, exit
        move.l  a5,a3
;
        adda    d7,a3                    ;First byte
        move.b  (a3),d0                  ;...get original
        and.b   d4,d0                    ;...mask
        move.b  d0,(a3)                  ;...replace
;
@8      adda    #90,a5                   ;Go to next row
        sub.w   #1,d2                    ;Decrement row counter
        bra     @7
;
;
@11     movem.l (SP)+,a0-a6/d0-d7        ;...restore the reg
        jmp     (a0)
;
StartByte .byte  $00,$80,$c0,$e0,$f0,$f8,$fc,$fe
EndByte   .byte  $7F,$3F,$1F,$0F,$07,$03,$01,$00
;
;
;
;============================================================================
;
AInvert_Box
; Name:   AInvert_Box
;       Procedure AInvert_Box(X1,Y1,X2,Y2:integer);
;
;  Function: Invert an area.
;
;
        MOVE.L  (SP)+,A0        ;save return address
        MOVE.W  (SP)+,a1        ;y2
        MOVE.W  (SP)+,d1        ;x2
        MOVE.W  (SP)+,d2        ;y1
        MOVE.W  (SP)+,d0        ;x1
;
        movem.l a0-a6/d0-d7,-(SP)      ;save required regs
        move.w  a1,d3
        move.l  $160,a5
        move.w  d0,a4
        bsr     XLegal
        move.l  a4,d0                    ;x1
        move.w  d1,a4
        bsr     XLegal
        move.l  a4,d1                    ;x2
        move.w  d2,a4
        bsr     YLegal
        move.l  a4,d2                    ;y1
        move.w  d3,a4
        bsr     YLegal
        move.l  a4,d3                    ;y2
        cmp.w   d0,d1                    ;See if x1<x2
        bpl     @1
        move.w  d0,d4                    ;...Swap
        move.w  d1,d0
        move.w  d4,d1
@1      cmp.w   d2,d3                    ;See if y1<y2
        bpl     @2
        move.w  d2,d4                    ;...Swap
        move.w  d3,d2
        move.w  d4,d3
;
@2      move.l  a5,a4
        move.l  d2,d5
        mulu    #90,d5                   ;Find starting row address, in a5
        adda    d5,a5
        mulu    #90,d3                   ;...find ending row address, in a4
        adda    d3,a4
;
        move.w  d1,d4                    ;Get number of columns to invert
        sub.w   d0,d4
        add.w   #1,d4
;
        divs    #8,d0                    ;Find column offset
        move.w  d0,d2                    ;...get quotient
        swap    d0
        and.l   #$7,d0                   ;...get remainder
        not.b   d0
        and.l   #$7,d0
        move.w  d0,d7
;
;    Main clear  loop
@3      cmp.l   a5,a4                    ;To last row yet?
        bmi     @11                      ;...yes, exit
        move.l  a5,a3
;
        adda    d2,a3
        clr.l   d7                       ;Init Pixel per byte counter
        move.b  d0,d7
        clr.l   d5
        bra     @6
;
@4      clr.l   d7
        move.b  #$07,d7
```

```
@6      bchg    d7,(a3)                 ;Change bit since in window
        add.w   #1,d5                   ;...next pixel count
        cmp.w   d4,d5
        beq     @10
        sub.w   #1,d7                   ;Go to next bit in the byte
        bmi     @7                      ;......yes, go to next byte
        bra     @6                      ;......no, continue in byte
@7      adda    #1,a3                   ;Go to next byte
        bra     @4
;
@10     adda    #90,a5                  ;Go to next row
        bra     @3
@11     movem.l (SP)+,a0-a6/d0-d7       ;...restore the reg
        jmp     (a0)
;
;
;==========================================================================
;
SETCRSR                         ;Change x,y to page address
; Name: SETCRSR
;
; Function: Finds address in video page by x1 and y1 coordinates.
;           Callable from assembly language only by another procedure.
;
; Input Parameters:
;       d6 = x1
;       d7 = y1
;
; Output Parameters:
;       a1 = address
;
;

        move.l  d0,-(sp)        ;Save register
        move.l  $160,a1         ;Start at top of page
;
        clr.l   d0              ;...clear for use
        move.w  d7,d0           ;Get row coordinate
        mulu    #90,d0          ;...compute byte offset
        adda    d0,a1           ;...add in real screen address
;
        clr.l   d0
        move.w  d6,d0           ;Get column coordinate
        divs    #8,d0           ;Find column offset
        and.l   #$ffff,d0       ;...get quotient
        adda    d0,a1           ;...add in column offset
;
        mulu    #8,d0
        move.w  d0,d6           ;Send back column used
        move.L  (sp)+,d0        ;Restore register
        rts
;
;==========================================================================
;
Paint_String                            ;Print text string on screen
; Name: Paint_String
;
; Procedure Paint_String(X1,Y1:integer;var DStr:String255);
;
;  Function: Draw text on screen.
;
; Input Parameters:
;       x1      = Horzontial position, 0-719
;       y1      = Starting y position, 0-359.
;       Dstr    = string message to paint.
;
; Output Parameters:
;       None
;
;
        move.l  (a7)+,a0        ;Save return address
        move.l  (a7)+,a2        ;Get TITLE address
        move.w  (a7)+,d1        ;Get y1
        move.w  (a7)+,d0        ;Get x1
        movem.l d0-d7/a0-a6,-(sp) ;Save the world
;
        move.w  d0,d6
        move.w  d1,d7           ;Get video page address
        bsr     SETCRSR
;
        clr.w   d3
        move.b  (a2)+,d3        ;Get string length
;
        clr.l   d1              ;Init character counter
@1      clr.l   d2              ;...clear for use
        move.b  (a2)+,d2        ;Get character to display
        bsr     OSPVAL          ;display it
        add.w   #1,d1
        cmp.w   d3,d1
        bne     @1
@2      movem.L (sp)+,d0-d7/a0-a6   ;Restore register
        jmp     (a0)
;
; Requires  character code in d2, d5 is font wanted, A1 is screen location
OSPVAL  movem.l d2/a4,-(sp)         ;Save the world
        and.l   #$FF,d2
        sub.b   #$20,d2             ;Convert character code for table index
        lsl     #3,d2                   ;...for 8 long table
        lea     FONTTABLE,a4            ;Get pointer to character font table
        adda    d2,a4
        move.B  (A4)+,(A1)              ;Place character on screen
        move.B  (A4)+,90(A1)
        move.B  (A4)+,180(A1)
        move.B  (A4)+,270(A1)
        move.B  (A4)+,360(A1)
        move.B  (A4)+,450(A1)
        move.B  (A4)+,540(A1)
        move.B  (A4)+,630(A1)
        adda    #1,a1                   ;Bump cursor column coordinate for next char
        movem.L (sp)+,d2/a4             ;Restore register
        rts
;
;======================================================================
;
Paint_Ch                                ;Print a character on screen
; Name: Paint_Ch
;
;   Procedure Paint_Ch(X1,Y1:integer; Ch:char);
;
;  Function: Draw character on screen.
;
; Input Parameters:
;       x1      = Horzontial position, 0-719
;       y1      = Starting y position, 0-359.
;       Ch      = character to paint.
;
; Output Parameters:
;       None
;
        move.l  (a7)+,a0        ;Save return address
        move.w  (a7)+,d2        ;Get character
        move.w  (a7)+,d1        ;Get y1
        move.w  (a7)+,d0        ;Get x1
        movem.l d0-d7/a0-a6,-(sp) ;Save the world
;
        move.w  d0,d6
        move.w  d1,d7           ;Get video page address
        bsr     SETCRSR
;
        bsr     DSPVAL          ;display it
;
        movem.L (sp)+,d0-d7/a0-a6   ;Restore register
        jmp     (a0)
;
;======================================================================
;
ASize_String
; Name: ASize_String
;
;   Procedure ASize_String(X1,Y1:integer;var Str:String255;
;                 var SX1:integer);
;
;  Function: Calculate new X if string is printed.
;
        move.l  (a7)+,a0        ;Save return address
        move.l  (a7)+,a2        ;Get address of SX1
        move.l  (a7)+,a1        ;Get address of string
        move.w  (a7)+,d1        ;Get y1
        move.w  (a7)+,d0        ;Get x1
        movem.l d0-d7/a0-a6,-(sp) ;Save the world
;
        clr.l   d3
        move.b  (a1),d3         ;Length of string
        mulu    #8,d3
;
        move.w  d0,d6
        move.w  d1,d7           ;Get video page address
        bsr     SETCRSR
;
        add.w   d3,d6
;
        move.w  d6,(a2)         ;Send back new x value
        movem.l d0-d7/a0-a6,-(sp) ;Save the world
        jmp     (a0)
;
;======================================================================
;
ZeroDigitArea
;
; Requires  d0=x1, d1=y1, d3=number of digits
        movem.l d0-d7/a0-a6,-(sp) ;Save the world
        move.w  d0,d6
        move.w  d1,d7           ;Get video page address
        bsr     SETCRSR
;
@1      move.w  #$30,d2         ;Zero character
        bsr     DSPVAL          ;Clear area for new number
        sub.w   #1,d3
        cmp.w   #0,d3
        bne     @1
        movem.L (sp)+,d0-d7/a0-a6   ;Restore register
        rts
;======================================================================
;
ADraw_Integer
; Name: ADraw_Integer
;
;    Procedure ADraw_Integer(X1,Y1:integer; Num,Digits:integer;
;             Right_Justify:boolean);
;
;  Function: Draw integer on screen, POSITIVE integers only.
;
        move.l  (a7)+,a0            ;Save return address
        move.w  (a7)+,d2            ;Get Right_Justify
        move.w  (a7)+,a2            ;Get Digits
        move.w  (a7)+,a1            ;Get integer
        move.w  (a7)+,d1            ;Get y1
        move.w  (a7)+,d0            ;Get x1
        movem.l d0-d7/a0-a6,-(sp)   ;Save the world
;
        move.w  d2,a5               ;Right_Justify
        move.w  a2,d3               ;Number of digits
        move.w  a1,d4               ;Number
;
        cmp.w   #0,d3
        beq     @100
;
        move.w  d0,d6
        move.w  d1,d7               ;Get video page address
        bsr     SETCRSR
;
@1      move.w  #$20,d2             ;Space character
        bsr     DSPVAL              ;Clear area for new number, to all spaces
        sub.w   #1,d3
        cmp.w   #0,d3
        bne     @1
;
        move.w  a2,d3               ;Number of digits
        move.w  d0,d6
        move.w  d1,d7               ;Get video page address
        bsr     SETCRSR
;
        lea     DIGITS,a6
        clr.l   (a6)+               ;Clear to default of 0
        clr.w   (a6)
        lea     DIGITS,a6
;
        clr.l   d7
        clr.l   d6
        move.w  d4,d6                    ;Get number
        divs    #10000,d6                ;Check for 5th digit
        beq     @5                       ;...none
        move.b  d6,1(a6)                 ;...save in table
        muls    #10000,d6
        sub.w   d6,d4
        cmp.w   #0,d7
        bne     @5
        move.w  #5,d7                    ;5 Digits
;
@5      clr.l   d6
        move.w  d4,d6
        divs    #1000,d6                 ;Check for 4th digit
        beq     @6                       ;...none
        move.b  d6,2(a6)                 ;...save in table
        muls    #1000,d6
        sub.w   d6,d4
        cmp.w   #0,d7
        bne     @6
        move.w  #4,d7                    ;4 Digits
;
@6      clr.l   d6
        move.w  d4,d6
        divs    #100,d6                  ;Check for 3rd digit
        beq     @7                       ;...none
        move.b  d6,3(a6)                 ;...save in table
        muls    #100,d6
        sub.w   d6,d4
        cmp.w   #0,d7
        bne     @7
        move.w  #3,d7                    ;3 Digits
;
@7      clr.l   d6
        move.w  d4,d6
        divs    #10,d6                   ;Check for 2nd digit
        beq     @8                       ;...none
        move.b  d6,4(a6)                 ;...save in table
        muls    #10,d6
        sub.w   d6,d4
        cmp.w   #0,d7
        bne     @8
        move.w  #2,d7                    ;2 Digits
;
@8      clr.l   d6
        move.w  d4,d6                    ;Always do at least one digit
        move.b  d6,5(a6)                 ;...save in table
        cmp.w   #0,d7
        bne     @9
        move.w  #1,d7                    ;1 Digit
;
```

```
@9      move.w  a5,d0                   ;Right justify?
        cmp.w   #0,d0                   ;...no
        beq     @10
        adda    d3,a1                   ;Move to last digit
        suba    #1,a1
        bra     @11
@10     adda    d7,a1
        suba    #1,a1
@11     cmp.w   d7,d3                   ;Overflow area?
        bmi     @12                     ;...yes, Fill with ????
        bra     @13                     ;...no, go do it
;
@12     move.w  #$3F,d2                 ;? character
        bsr     DSPVAL                  ;Write area to overflow value
        suba    #2,a1
        sub.w   #1,d3
        cmp.w   #0,d3
        bne     @12
        bra     @100
;
@13     lea     LastDigit,a4
@14     clr.l   d2
        move.b  (a4),d2                 ;Digit
        add.w   #$30,d2
        bsr     DSPVAL                  ;Write number
        suba    #1,a4
        suba    #2,a1
        sub.w   #1,d7
        cmp.w   #0,d7
        bne     @14
;
@100    movem.L (sp)+,d0-d7/a0-a6   ;Restore register
        jmp     (a0)
;
;
;======================================================================
;
ADraw_Hex
; Name: ADraw_Hex
;
;    Procedure ADraw_Hex(X1,Y1:integer; Num:integer);
;
;  Function: Draw hex number on screen.
;
        move.l  (a7)+,a0        ;Save return address
        move.w  (a7)+,d2        ;Get integer
        move.w  (a7)+,d1        ;Get y1
        move.w  (a7)+,d0        ;Get x1
        movem.l d0-d7/a0-a6,-(sp) ;Save the world
;
        lea     Digits,a6       ;Init area
        move.l  #0,(a6)+
        move.l  #0,(a6)
        lea     Digits,a6
;
        move.w  #4,d3           ;Number of digits
;
; Requires  d0=x1, d1=y1, d3=number of digits
        bsr     ZeroDigitArea    ;Place Zeros in digit area
;
        move.w  d2,d7
        move.w  d2,d6           ;Save number
;
        and.w   #$F000,d6       ;Get highest digit
        ror.w   #8,d6           ;Can only rotate 8 at a time
        ror.w   #4,d6
        move.w  d6,(a6)+
;
        move.w  d7,d6
        and.w   #$0F00,d6       ;Get next digit
        ror.w   #8,d6
        move.w  d6,(a6)+
;
        move.w  d7,d6
        and.w   #$00F0,d6       ;Get next digit
        ror.w   #4,d6
        move.w  d6,(a6)+
;
        move.w  d7,d6
        and.w   #$000F,d6       ;Get last digit
        move.w  d6,(a6)+
;
        move.w  d0,d6
        move.w  d1,d7           ;Get video page address
        bsr     SETCRSR
;
        lea     Digits,a6
        move.w  #0,d4
@1      move.w  (a6)+,d3        ;Get hex digit
        lea     HexDigits,a5
        adda    d3,a5
        move.b  (a5),d2
        and.l   #$ff,d2
        bsr     DSPVAL
        add.w   #1,d4
        cmp.w   #4,d4
        bne     @1
;
@2      movem.l (sp)+,d0-d7/a0-a6       ;Restore registers
        jmp     (a0)
;
HexDigits .Byte $30,$31,$32,$33,$34,$35,$36,$37,$38,$39
          .Byte $41,$42,$43,$44,$45,$46
;
;==================================================================
;
ADraw_LHex
; Name: ADraw_LHex
;
;     Procedure ADraw_LHex(X1,Y1:integer; Num:longint);
;
;   Function: Draw hex number on screen.
;
        move.l  (a7)+,a0        ;Save return address
        move.l  (a7)+,d2        ;Get long integer
        move.w  (a7)+,d1        ;Get y1
        move.w  (a7)+,d0        ;Get x1
        movem.l d0-d7/a0-a6,-(sp) ;Save the world
;
        lea     Digits,a6       ;Init area
        move.l  #0,(a6)+
        move.l  #0,(a6)+
        move.l  #0,(a6)
        lea     Digits,a6
;
        move.w  #8,d3
; Requires  d0=x1, d1=y1, d3=number of digits
        bsr     ZeroDigitArea    ;Place Zeros in digit area
;
        move.l  d2,d7
        move.l  d2,d6           ;Save number
;
        and.l   #$F0000000,d6   ;Get highest digit
        ror.l   #8,d6
        ror.l   #8,d6
        ror.l   #8,d6
        ror.l   #4,d6
        move.b  d6,(a6)+
;
        move.l  d7,d6
        and.l   #$0F000000,d6   ;Get next digit
        ror.l   #8,d6
        ror.l   #8,d6
        ror.l   #8,d6
        move.b  d6,(a6)+
;
        move.l  d7,d6
        and.l   #$00F00000,d6   ;Get next digit
        ror.l   #8,d6
        ror.l   #8,d6
        ror.l   #4,d6
        move.b  d6,(a6)+
;
        move.l  d7,d6
        and.l   #$000F0000,d6   ;Get next digit
        ror.l   #8,d6
        ror.l   #8,d6
        move.b  d6,(a6)+
;
        move.l  d7,d6
        and.l   #$0000F000,d6   ;Get next digit
        ror.l   #8,d6
        ror.l   #4,d6
        move.b  d6,(a6)+
;
        move.l  d7,d6
        and.l   #$00000F00,d6   ;Get next digit
        ror.l   #8,d6
        move.b  d6,(a6)+
;
        move.l  d7,d6
        and.l   #$000000F0,d6   ;Get next digit
        ror.l   #4,d6
        move.b  d6,(a6)+
;
        move.l  d7,d6
        and.l   #$0000000F,d6   ;Get last digit
        move.b  d6,(a6)+
;
;
        move.w  d0,d6
        move.w  d1,d7           ;Get video page address
        bsr     SETCRSR
;
        lea     Digits,a6
        move.w  #0,d4
@1      move.b  (a6)+,d3        ;Get hex digit
        and.l   #$F,d3          ;Make legal
        lea     HexDigits,a5
        adda    d3,a5
        move.b  (a5),d2
        and.l   #$ff,d2
        bsr     DSPVAL
        add.w   #1,d4
        cmp.w   #8,d4
        bne     @1
;
@2      movem.l (sp)+,d0-d7/a0-a6       ;Restore registers
        jmp     (a0)
;
;=====================================================================
;==============================================================
;==========================================================================
;
;
AFolder
; Name:   AFOLDER
;   Procedure AFolder(X1,Y1,X2,Y2:integer; var DStr:String255);External;
;
;  Function: Draw a folder
;
;
        MOVE.L  (SP)+,A0        ;save return address
        MOVE.L  (SP)+,a1        ;string address
        MOVE.W  (SP)+,d3        ;y2
        MOVE.W  (SP)+,d2        ;x2
        MOVE.W  (SP)+,d1        ;y1
        MOVE.W  (SP)+,d0        ;x1
;
        movem.l d0-d7/a0-a6,-(sp)
;
        move.l  a1,a6
        move.w  d0,d4
        move.w  d1,d5
        move.w  d2,d6
        move.w  d3,d7
;
        move.w  d0,-(sp)        ;X1
        move.w  d1,-(sp)        ;Y1
        move.w  d2,-(sp)        ;X2
        move.w  d3,-(sp)        ;Y2
        bsr     AWhite_Box      ;(X1,Y1,X2,Y2:integer);
;
        move.w  d4,-(sp)        ;X1
        move.w  d5,-(sp)        ;Y1
        move.w  d6,-(sp)        ;X2
        move.w  d7,-(sp)        ;Y2
        move.w  #1,-(sp)        ;line width
        bsr     ASquared_Box    ;(X1,Y1,X2,Y2,Line_Width:integer);
;
        move.w  d5,d0           ;Y1
        add.w   #14,d0          ;+14
        move.w  d0,-(sp)        ;Y1+14
        move.w  d4,-(sp)        ;X1
        move.w  d6,-(sp)        ;X2
        bsr     HLine           ;(Y1,X1,X2:integer);
;
;
        clr.l   d3
        move.b  (a6),d3         ;Length of string
        mulu    #8,d3           ;...each char is 8 pixels wide
        move.w  d3,a4           ;string length
;
        clr.l   d0
        move.w  d6,d0           ;X2
        sub.w   d4,d0           ;X2-X1
        divs    #2,d0           ;(X2-X1)/2
        add.w   d4,d0           ;Center of window
;
        divs    #2,d3           ;text length / 2
        sub.w   d3,d0           ;text start
;
        move.w  d0,a5           ;String start
        move.w  d0,-(sp)        ;Get x1
        move.w  d5,d2
        add.w   #3,d2
        move.w  d2,-(sp)        ;Get y1
        move.l  a6,-(sp)        ;Get TITLE address
        bsr     Paint_String    ;Print text string on screen
;
        move.w  a4,d3           ;total length
        add.w   #40,d3          ;add for border
        move.w  a5,d0
        sub.w   #20,d0          ;start 10 left
;
        move.w  d5,d7
        add.w   #14,d7
;
        move.w  d0,d4
        add.w   #1,d4
        bsr     DoLine
;
        move.w  d0,d4
        add.w   #3,d4
        bsr     DoLine
;
        move.w  d0,d4
        add.w   #5,d4
        bsr     DoLine
;
        move.w  d0,d4
        add.w   #8,d4
        bsr     DoLine
;
        move.w  d0,d4
        add.w   #12,d4
        bsr     DoLine
;
;
;
        move.w  d0,d4
        add.w   d3,d4
        sub.w   #1,d4
        bsr     DoLine
;
        move.w  d0,d4
        add.w   d3,d4
        sub.w   #3,d4
        bsr     DoLine
;
        move.w  d0,d4
        add.w   d3,d4
        sub.w   #5,d4
        bsr     DoLine
;
        move.w  d0,d4
        add.w   d3,d4
        sub.w   #8,d4
        bsr     DoLine
;
        move.w  d0,d4
        add.w   d3,d4
        sub.w   #12,d4
        bsr     DoLine
;
; Invert box containing the text.
        move.w  d0,d1
        add.w   d3,d1
        move.w  d0,-(sp)        ;x1
        move.w  d5,d2
        add.w   #1,d2
        move.w  d2,-(sp)        ;y1
        move.w  d1,-(sp)        ;x2
        move.w  d5,d2
        add.w   #13,d2
        move.w  d2,-(sp)        ;y2
        bsr     AINVERT_BOX
;
;
@10     movem.l (sp)+,d0-d7/a0-a6   ;Restore the world
        jmp     (a0)
;
DoLine  movem.l d0-d3/a0-a3,-(sp)
        move.w  d4,-(SP)        ;x1
        MOVE.W  d5,-(SP)        ;y1
        MOVE.W  d7,-(SP)        ;y2
        bsr     VLINE
        movem.l (sp)+,d0-d3/a0-a3
        rts
;==============================================================
;=====================================================================
;
ADialog
; Name:   ADialog
;    Procedure ADialog(X1,Y1,X2,Y2: integer);External;
;
;   Function: Draw a dialog box
;
;
        MOVE.L  (SP)+,A0        ;save return address
        MOVE.W  (SP)+,d3        ;y2
        MOVE.W  (SP)+,d2        ;x2
        MOVE.W  (SP)+,d1        ;y1
        MOVE.W  (SP)+,d0        ;x1
;
        movem.l d0-d7/a0-a6,-(sp)
;
        move.w  d0,d4
        move.w  d1,d5
        move.w  d2,d6
        move.w  d3,d7
;
        move.w  d0,-(sp)        ;X1
        move.w  d1,-(sp)        ;Y1
        move.w  d2,-(sp)        ;X2
        move.w  d3,-(sp)        ;Y2
        bsr     AWhite_Box      ;(X1,Y1,X2,Y2: integer);
;
        move.w  d4,-(sp)        ;X1
        move.w  d5,-(sp)        ;Y1
        move.w  d6,-(sp)        ;X2
        move.w  d7,-(sp)        ;Y2
        move.w  #1,-(sp)        ;line width
        bsr     ASquared_Box    ;(X1,Y1,X2,Y2,Line_Width: integer);
```

```
;
        move.w  d7,d3           ;Y2
        add.w   #1,d3           ;+1
        move.w  d4,d0
        add.w   #5,d0
        move.w  d6,d1
        add.w   #1,d1
        move.w  d3,-(sp)        ;Y2+1
        move.w  d0,-(sp)        ;X1
        move.w  d1,-(sp)        ;X2
        bsr     HLine           ;(Y1,X1,X2:integer);
;
        move.w  d6,d3           ;X2
        add.w   #1,d3           ;+1
        move.w  d5,d0
        add.w   #5,d0
        move.w  d7,d1
        add.w   #1,d1
        move.w  d3,-(sp)        ;X2+1
        move.w  d0,-(sp)        ;Y1
        move.w  d1,-(sp)        ;Y2
        bsr     VLine           ;(X1,Y1,Y2:integer);
;
        movem.l (sp)+,d0-d7/a0-a6   ;Restore the world
        jmp     (a0)
;
;
;==========================================================================
;
AIcon_Draw
;  Procedure AIcon_Draw(X1,Y1,Icon_Code:integer);EXTERNAL;
;

        MOVE.L  (SP)+,A0        ;save return address
        MOVE.w  (SP)+,d2        ;save Icon_Code
        MOVE.w  (SP)+,d1        ;save Y1
        MOVE.w  (sp)+,d0        ;save X1
        movem.l d0-d7/a0-a6,-(SP)  ;save required regs
;
        move.w  d0,d6
        move.w  d1,d7
        bsr     SETCRSR
        move.l  a1,d6
        and.l   #$FFFFFFFE,d6   ;Has to be on a word boundary
        move.l  d6,a1
;
        bsr     FindIcon
;
@1      clr.l   d0
        move.w  (a2)+,(a1)+     ;data
        move.w  (a2)+,(a1)+     ;data
        move.w  (a2)+,(a1)+     ;data
;
        moveq   #$5A,d0
        add.l   d0,a1
        subq    #6,a1
;
        dbf     d3,@1
;
        movem.l (SP)+,d0-d7/a0-a6     ;...restore the reg
        jmp     (a0)
;
;************************************************************
;
FindIcon
;
        clr.l   d4
        move.w  d2,d5           ;Calculate offset to correct icon
        lea     ICONDATA,a2
@1      move.w  (a2)+,d0         ;Get Leading count
        add.l   #2,d4
        move.w  d0,d7
        clr.l   d6
        move.w  d0,d6
        sub.w   #1,d5
        cmp.w   #0,d5
        beq     @10
@2      move.l  (a2)+,d1        ;two words
        move.w  (a2)+,d1        ;third word
        add.l   #6,d4
        sub.w   #1,d6           ;to next icon line
        cmp.w   #0,d6
        bne     @2
        move.w  (a2)+,d3        ;Get trailing count
        add.l   #2,d4
        cmp.w   d3,d7           ;Leader = trailer?
        bne     @10
        bra     @1
;
@10     lea     ICONDATA,a2
        adda    d4,a2
        move.w  d0,d3          ;set heigth
        sub.w   #1,d3
        rts
;
;==========================================================================
;
F1Mask   .Word 0
F2Mask   .Word 0
X1Edge   .WORD 0    ;Left edge boundary in absolutes
Y1Edge   .WORD 0    ;Top edge boundary in absolutes
X2Edge   .WORD 719  ;Right edge boundary in absolutes
Y2Edge   .WORD 363  ;Bottom edge boundary in absolutes
Digits   .Word 0,0
         .byte 0
LASTDIGIT .byte 0
         .Word 0,0,0,0,0
;
;=============================================================
;------------------------------------------------------------------
; CHARACTER FONT TABLE
;------------------------------------------------------------------
FONTTABLE
  .BYTE  $00,$00,$00,$00,$00,$00,$00,$00  ;  (space)    $20
  .BYTE  $10,$10,$10,$10,$00,$00,$10,$00  ;  !
  .BYTE  $48,$48,$48,$00,$00,$00,$00,$00  ;  "
  .BYTE  $48,$48,$FC,$48,$FC,$48,$48,$00  ;  #
  .BYTE  $10,$3C,$50,$38,$14,$78,$10,$00  ;  $
  .BYTE  $00,$C4,$C8,$10,$20,$4C,$8C,$00  ;  %
  .BYTE  $60,$90,$90,$60,$94,$88,$74,$00  ;  &
  .BYTE  $08,$10,$20,$00,$00,$00,$00,$00  ;  '
  .BYTE  $08,$10,$20,$20,$20,$10,$08,$00  ;  (
  .BYTE  $40,$20,$10,$10,$10,$20,$40,$00  ;  )
  .BYTE  $10,$54,$38,$7C,$38,$54,$10,$00  ;  *
  .BYTE  $00,$10,$10,$7C,$10,$10,$00,$00  ;  +
  .BYTE  $00,$00,$00,$00,$00,$30,$30,$60  ;  ,
  .BYTE  $00,$00,$00,$FC,$00,$00,$00,$00  ;  -
  .BYTE  $00,$00,$00,$00,$00,$30,$30,$00  ;  .
  .BYTE  $00,$04,$08,$10,$20,$40,$80,$00  ;  /
  .BYTE  $78,$84,$8C,$B4,$C4,$84,$78,$00  ;  0          $30
  .BYTE  $10,$30,$50,$10,$10,$10,$7C,$00  ;  1
  .BYTE  $78,$84,$04,$18,$60,$80,$FC,$00  ;  2
  .BYTE  $78,$84,$04,$38,$04,$84,$78,$00  ;  3
  .BYTE  $08,$18,$28,$48,$FC,$08,$08,$00  ;  4
  .BYTE  $FC,$80,$F0,$08,$04,$88,$70,$00  ;  5
  .BYTE  $38,$40,$80,$F8,$84,$84,$78,$00  ;  6
  .BYTE  $FC,$84,$08,$10,$20,$20,$20,$00  ;  7
  .BYTE  $78,$84,$84,$78,$84,$84,$78,$00  ;  8
  .BYTE  $78,$84,$84,$7C,$04,$08,$70,$00  ;  9
  .BYTE  $00,$00,$30,$30,$00,$30,$30,$00  ;  :
  .BYTE  $00,$00,$30,$30,$00,$30,$30,$60  ;  ;
  .BYTE  $08,$10,$20,$40,$20,$10,$08,$00  ;  <
  .BYTE  $00,$00,$F8,$00,$F8,$00,$00,$00  ;  =
  .BYTE  $40,$20,$10,$08,$10,$20,$40,$00  ;  >
  .BYTE  $78,$84,$04,$18,$20,$00,$20,$00  ;  ?
  .BYTE  $38,$44,$94,$AC,$98,$40,$3C,$00  ;  @          $40
  .BYTE  $30,$48,$84,$FC,$84,$84,$84,$00  ;  A
  .BYTE  $F8,$44,$44,$78,$44,$44,$F8,$00  ;  B
  .BYTE  $78,$84,$80,$80,$80,$84,$78,$00  ;  C
  .BYTE  $F8,$44,$44,$44,$44,$44,$F8,$00  ;  D
  .BYTE  $FC,$80,$80,$F0,$80,$80,$FC,$00  ; /E
  .BYTE  $FC,$80,$80,$F0,$80,$80,$80,$00  ;  F
  .BYTE  $78,$84,$80,$9C,$84,$84,$78,$00  ;  G
  .BYTE  $84,$84,$84,$FC,$84,$84,$84,$00  ;  H
  .BYTE  $38,$10,$10,$10,$10,$10,$38,$00  ;  I
  .BYTE  $1C,$08,$08,$08,$08,$88,$70,$00  ;  J
  .BYTE  $84,$88,$90,$E0,$90,$88,$84,$00  ;  K
  .BYTE  $80,$80,$80,$80,$80,$80,$FC,$00  ;  L
  .BYTE  $84,$CC,$B4,$B4,$84,$84,$84,$00  ;  M
  .BYTE  $84,$C4,$A4,$94,$8C,$84,$84,$00  ;  N
  .BYTE  $78,$84,$84,$84,$84,$84,$78,$00  ;  O
  .BYTE  $F8,$84,$84,$F8,$80,$80,$80,$00  ;  P          $50
  .BYTE  $78,$84,$84,$84,$94,$88,$74,$00  ;  Q
  .BYTE  $F8,$84,$84,$F8,$90,$88,$84,$00  ;  R
  .BYTE  $78,$84,$80,$78,$04,$84,$78,$00  ;  S
  .BYTE  $7C,$10,$10,$10,$10,$10,$10,$00  ;  T
  .BYTE  $84,$84,$84,$84,$84,$84,$78,$00  ;  U
  .BYTE  $84,$84,$84,$48,$48,$30,$30,$00  ;  V
  .BYTE  $84,$84,$84,$B4,$B4,$CC,$84,$00  ;  W
  .BYTE  $84,$84,$48,$30,$48,$84,$84,$00  ;  X
  .BYTE  $44,$44,$44,$38,$10,$10,$10,$00  ;  Y
  .BYTE  $FC,$04,$08,$30,$40,$80,$FC,$00  ;  Z
  .BYTE  $78,$40,$40,$40,$40,$40,$78,$00  ;  [
  .BYTE  $00,$80,$40,$20,$10,$08,$04,$00  ;  \
  .BYTE  $78,$08,$08,$08,$08,$08,$78,$00  ;  ]
  .BYTE  $10,$28,$44,$00,$00,$00,$00,$00  ;  ^
  .BYTE  $00,$00,$00,$00,$00,$00,$00,$FE  ;  _
  .BYTE  $20,$10,$08,$00,$00,$00,$00,$00  ;             $60
  .BYTE  $00,$00,$70,$08,$78,$88,$74,$00  ;  a
  .BYTE  $80,$80,$B8,$C4,$84,$C4,$B8,$00  ;  b
  .BYTE  $00,$00,$78,$80,$80,$80,$78,$00  ;  c
  .BYTE  $04,$04,$74,$8C,$84,$8C,$74,$00  ;  d
  .BYTE  $00,$00,$78,$84,$FC,$80,$78,$00  ;  e
  .BYTE  $18,$24,$20,$F8,$20,$20,$20,$00  ;  f
  .BYTE  $00,$00,$74,$8C,$8C,$74,$04,$78  ;  g
  .BYTE  $80,$80,$B8,$C4,$84,$84,$84,$00  ;  h
  .BYTE  $10,$00,$30,$10,$10,$10,$38,$00  ;  i
  .BYTE  $08,$00,$18,$08,$08,$08,$88,$70  ;  j
  .BYTE  $80,$80,$88,$90,$A0,$D0,$88,$00  ;  k
  .BYTE  $30,$10,$10,$10,$10,$10,$38,$00  ;  l
  .BYTE  $00,$00,$E8,$54,$54,$54,$54,$00  ;  m
  .BYTE  $00,$00,$F8,$44,$44,$44,$44,$00  ;  n
  .BYTE   $00,$00,$38,$44,$44,$44,$38,$00   ;  o
  .BYTE   $00,$00,$B8,$C4,$C4,$B8,$80,$80   ;  p          $70
  .BYTE   $00,$00,$74,$8C,$8C,$74,$04,$04   ;  q
  .BYTE   $00,$00,$B8,$C4,$80,$80,$80,$00   ;  r
  .BYTE   $00,$00,$7C,$80,$78,$04,$F8,$00   ;  s
  .BYTE   $20,$20,$F8,$20,$20,$24,$18,$00   ;  t
  .BYTE   $00,$00,$84,$84,$84,$8C,$74,$00   ;  u
  .BYTE   $00,$00,$84,$84,$84,$48,$30,$00   ;  v
  .BYTE   $00,$00,$44,$44,$54,$54,$6C,$00   ;  w
  .BYTE   $00,$00,$84,$48,$30,$48,$84,$00   ;  x
  .BYTE   $00,$00,$84,$84,$8C,$74,$04,$78   ;  y
  .BYTE   $00,$00,$FC,$08,$30,$40,$FC,$00   ;  z
  .BYTE   $00,$04,$08,$08,$10,$08,$08,$04   ;  (
  .BYTE   $20,$20,$20,$20,$20,$20,$20,$20   ;  |
  .BYTE   $00,$10,$08,$08,$04,$08,$08,$10   ;  )
APPLICON
  .BYTE   $04,$08,$77,$FE,$FE,$7F,$3E,$00   ; apple icon
  .BYTE   $FE,$FE,$FE,$FE,$FE,$FE,$FE,$FE   ;  rubout
  .BYTE   $00,$02,$06,$0e,$1f,$0e,$06,$02   ;left arrow   $80
  .BYTE   $00,$40,$60,$70,$f8,$70,$60,$40   ;right arrow  $81
  .BYTE   $00,$00,$08,$1c,$3e,$7f,$08,$08   ;up arrow     $82
  .BYTE   $08,$08,$7f,$3e,$1c,$08,$00,$00   ;down arrow   $83
;
;
;
;
;*
ICONDATA
;* 1  Big board
   .word 23
   .word $03FF,$FFFF,$FFE0,$0200,$0000,$0020,$0200,$0000,$0020
   .word $0200,$0000,$0020,$0200,$0000,$0020,$0200,$0000,$0020,$0200,$0000,$0020
   .word $0200,$0000,$0020,$0200,$0000,$0020,$0200,$0000,$0020,$0200,$0000,$0020
   .word $0200,$0000,$0020,$0200,$0000,$0020,$0200,$0000,$0020,$0200,$0000,$0020
   .word $0200,$0000,$0020,$0200,$0000,$0020,$0200,$0000,$0020,$0200,$0000,$0020
   .word $0200,$0000,$0020,$03AA,$AAAF,$FFE0,$00AA,$AAA8,$0000,$00FF,$FFF8,$0000
   .word 23
;* 2  memory board
   .word 17
   .word $0000,$1FFF,$FFC0
   .word $0000,$1000,$0040,$07FF,$F000,$0040,$0400,$0000,$0040,$0400,$0000,$0040
   .word $0400,$0000,$0040,$0400,$0000,$0040,$0400,$0000,$0040,$0404,$5E88,$0040
   .word $0406,$D0D8,$0040,$0405,$5CA8,$0040,$0404,$5088,$0040,$0404,$5E88,$0040
   .word $0400,$0000,$0040,$07FF,$D557,$FFC0,$0000,$5554,$0000,$0000,$7FFC,$0000
   .word 17
;* 3  Expansion card
   .word 23
   .word $0001,$FFFF,$8000,$0001,$0000,$8000,$0001,$0000,$8000
   .word $0001,$0000,$8000,$0001,$0000,$8000,$0001,$0000,$8000,$0001,$0000,$8000
   .word $0001,$0000,$8000,$0001,$0000,$8000,$0001,$0000,$8000,$0001,$0000,$8000
   .word $0001,$0000,$8000,$0001,$0000,$8000,$0001,$0000,$8000,$0001,$0000,$8000
   .word $0001,$0000,$8000,$0001,$0000,$8000,$0001,$0000,$8000,$0001,$0000,$8000
   .word $0001,$0003,$8000,$0001,$EAAE,$0000,$0000,$2AA8,$0000,$0000,$3FF8,$0000
   .word 23
;* 4  diskette
   .word 24
   .word $00FD,$FFFF,$FE00,$0087,$0000,$0200,$0080,$0000,$3380,$0080,$0180,$0080
   .word $0080,$0180,$0080,$0080,$0180,$0080,$0080,$0180,$0080,$0080,$0180,$0080
   .word $0080,$0000,$0080,$0080,$03C0,$0080,$0080,$07E0,$0080,$0080,$0FF0,$0080
   .word $0080,$0FF0,$0080,$0080,$07E0,$0080,$0080,$03C0,$0080,$0080,$0000,$0080
   .word $0080,$0180,$0080,$0080,$0180,$0080,$0080,$0180,$0080,$0080,$0180,$0080
   .word $0080,$0180,$0080,$0080,$0000,$0080,$0080,$0000,$0080,$00FF,$FFFF,$FF80
   .word 24
;* 5  drive
   .word 9
   .word $0000,$07F8,$2000,$0000,$0408,$6000,$3FFF,$FC08,$A000,$2000,$0009,$3FFC
   .word $2000,$000A,$0004,$3FFF,$FC09,$3FFC,$0000,$0408,$A000,$0000,$07F8,$6000
   .word $0000,$0000,$2000
   .word 9
;* 6  insert disk
   .word 23
   .word $0000,$0000,$3FC0,$0000,$0000,$2040,$0001,$FFFF,$E040
   .word $0001,$0000,$0040,$0001,$07FF,$F840,$0001,$EFFF,$FC40,$0000,$1FC1,$FEC0
   .word $0000,$3F80,$FF00,$0000,$7FC1,$FF80,$0000,$FFFF,$FFC0,$0001,$FFFF,$FFE0
   .word $0003,$FFFF,$FFF0,$0000,$0000,$0000,$0000,$0008,$0000,$0000,$0014,$0000
   .word $0000,$0022,$0000,$0000,$0041,$0000,$0000,$0080,$8000,$0000,$0100,$4000
   .word $0000,$03E3,$E000,$0000,$0022,$0000,$0000,$0022,$0000,$0000,$003E,$0000
   .word 23
;* 7  profile
   .word 10
   .word $1FFF,$FFFF,$FFF8,$2000,$0000,$0004,$2000,$0000,$0004,$2000,$0000,$0004
   .word $237F,$0000,$0184,$237F,$0000,$0184,$2000,$0000,$0004,$2000,$0000,$0004
   .word $1FFF,$FFFF,$FFF8,$0300,$000C,$00C0
   .word 10
;* 8  lisa
   .word 24
   .word $007F,$FFFF,$FFFC,$0080,$0000,$0002,$011F,$FFFF,$03F1,$0120,$0000,$8001
   .word $0120,$0000,$83F1,$0120,$0000,$8001,$0120,$0000,$8001,$0120,$0000,$8001
   .word $0120,$0000,$8001,$0120,$0000,$8001,$0120,$0000,$87F1,$0120,$0000,$8001
   .word $011F,$FFFF,$0001,$0080,$0000,$0002,$007F,$FFFF,$FFFC,$0000,$0000,$0000
   .word $007F,$FFFF,$FFFC,$0080,$0000,$0002,$0095,$5555,$52A2,$008A,$AAAA,$A142
   .word $0081,$5555,$02A2,$0080,$0000,$0002,$0080,$0000,$0002,$007F,$FFFF,$FFFC
   .word 24
;* 9  keybd
   .word 13
   .word $1FFF,$FFFF,$FFFC
   .word $2000,$0000,$0002,$26DB,$6DB6,$0DB2,$2000,$0000,$0002,$26DB,$6DB6,$0DB2
   .word $2000,$0000,$0002,$26DB,$6DB6,$0DB2,$2000,$0000,$0002,$26FF,$FFF6,$0DB2
   .word $2000,$0000,$0002,$1FFF,$FFFF,$FFFC,$0000,$0000,$0000,$0000,$0000,$0000
   .word 13
;* 10  mouse
   .word 15
   .word $0000,$7FFC,$0000,$0000,$8002,$0000,$0000,$9FF2,$0000
   .word $0000,$9012,$0000,$0000,$9FF2,$0000,$0000,$8002,$0000,$0000,$8002,$0000
   .word $0000,$8002,$0000,$0000,$8002,$0000,$0000,$8002,$0000,$0000,$8002,$0000
   .word $0000,$8002,$0000,$0000,$8002,$0000,$0000,$8002,$0000,$0000,$7FFC,$0000
   .word 15
;* 11  keyboard out
   .word 29
   .word $0000,$0000,$01C0,$0000,$0000,$01C0,$0000,$0000,$01C0,$0000,$0000,$0FF8
   .word $0000,$0000,$07F0,$0000,$0000,$03E0,$0000,$0000,$01C0,$0000,$0000,$0080
   .word $0000,$0000,$0000,$0000,$0000,$01C0,$0000,$0000,$01C0,$0000,$0000,$0080
   .word $1B6D,$B6DB,$6080,$0DB6,$DB6D,$BF80,$36DB,$6DB6,$D800,$3000,$0000,$0000
   .word $3000,$0000,$0000,$3000,$0000,$0000,$7FFF,$FFFF,$FFF0,$8000,$0000,$0008
   .word $9B6D,$B6D8,$36C8,$8000,$0000,$0008,$9B6D,$B6D8,$36C8,$8000,$0000,$0008
   .word $9B6D,$B6D8,$36C8,$8000,$0000,$0008,$9BFF,$FFD8,$36C8,$8000,$0000,$0008
   .word $7FFF,$FFFF,$FFF0
   .word 29
;* 12  mouse out
   .word 29
   .word $00E0,$0000,$0000,$00E0,$0000,$0000,$00E0,$0000,$0000,$07FC,$0000,$0000
   .word $03F8,$0000,$0000,$01F0,$0000,$0000,$00E0,$0000,$0000,$0040,$0000,$0000
   .word $0000,$0000,$0000,$01E0,$0000,$0000,$01E0,$0000,$0000,$00C0,$0000,$0000
   .word $00C0,$0000,$0000,$00FF,$FF80,$0000,$0000,$0180,$0000,$0000,$0180,$0000
   .word $0000,$0180,$0000,$0000,$7FFC,$0000,$0000,$8002,$0000,$0000,$9FF2,$0000
   .word $0000,$9012,$0000,$0000,$9FF2,$0000,$0000,$8002,$0000,$0000,$8002,$0000
   .word $0000,$8002,$0000,$0000,$8002,$0000,$0000,$8002,$0000,$0000,$8002,$0000
   .word $0000,$7FFC,$0000
   .word 29
;* 13  2 port card point to port 1
   .word 24
   .word $03FF,$FF00,$0000,$0200,$0100,$0000,$0200,$0100,$0000,$0200,$0100,$0000
   .word $0200,$0100,$4000,$0200,$0100,$C000,$0200,$0101,$C000,$0200,$0103,$4000
   .word $0200,$01E6,$7FE0,$0200,$01EC,$0020,$0200,$01E6,$7FE0,$0200,$01E3,$4000
   .word $0200,$0101,$C000,$0200,$0100,$C000,$0200,$01E0,$4000,$0200,$01E0,$0000
   .word $0200,$01E0,$0000,$0200,$01E0,$0000,$0200,$0100,$0000,$0200,$0700,$0000
   .word $0200,$0400,$0000,$03AA,$BC00,$0000,$00AA,$A000,$0000,$00FF,$E000,$0000
   .word 24
;* 14  2 port card point to port 2
   .word 24
   .word $03FF,$FF00,$0000,$0200,$0100,$0000,$0200,$0100,$0000,$0200,$0100,$0000
   .word $0200,$0100,$0000,$0200,$0100,$0000,$0200,$0100,$0000,$0200,$0100,$0000
   .word $0200,$01E0,$0000,$0200,$01E0,$0000,$0200,$01E0,$2000,$0200,$01E0,$6000
   .word $0200,$0100,$E000,$0200,$0101,$A000,$0200,$01E3,$2000,$0200,$01E6,$3FE0
   .word $0200,$01EC,$0020,$0200,$01E6,$3FE0,$0200,$0103,$2000,$0200,$0701,$A000
   .word $0200,$0400,$E000,$03AA,$BC00,$6000,$00AA,$A000,$2000,$00FF,$E000,$0000
   .word 24
;* 15  wait
   .word 24
   .word $001F,$FFFF,$F800,$0010,$0000,$0800,$001F,$FFFF,$F800,$000E,$0000,$7000
   .word $000B,$0000,$D000,$0009,$8001,$9000,$0008,$CFF3,$1000,$0008,$67E6,$1000
   .word $0008,$33CC,$1000,$0008,$1998,$1000,$0008,$0DB0,$1000,$0008,$05A0,$1000
   .word $0008,$05A0,$1000,$0008,$0DB0,$1000,$0008,$1998,$1000,$0008,$318C,$1000
   .word $0008,$6186,$1000,$0008,$C183,$1000,$0009,$87E1,$9000,$000B,$0FF0,$D000
   .word $000E,$1FF8,$7000,$001F,$FFFF,$F800,$0010,$0000,$0800,$001F,$FFFF,$F800
   .word 24
;* 16  checkmrk
   .word 32
   .word $0000,$0000,$0780,$0000,$0000,$0900,$0000,$0000,$1200,$0000,$0000,$2400
   .word $0000,$0000,$4800,$0000,$0000,$9000,$0000,$0001,$2000,$0000,$0002,$4000
   .word $0000,$0004,$8000,$0000,$0009,$0000,$0000,$0012,$0000,$0000,$0024,$0000
   .word $0000,$0048,$0000,$0000,$0090,$0000,$0000,$0120,$0000,$0000,$0240,$0000
   .word $0000,$0480,$0000,$0000,$0900,$0000,$0000,$1200,$0000,$0000,$2400,$0000
   .word $0000,$4800,$0000,$0000,$9000,$0000,$0001,$2000,$0000,$0002,$4000,$0000
   .word $F804,$8000,$0000,$2409,$0000,$0000,$1212,$0000,$0000,$0924,$0000,$0000
   .word $04C8,$0000,$0000,$0210,$0000,$0000,$0120,$0000,$0000,$00C0,$0000,$0000
   .word 32
;* 17  Arrow down
   .word 8
   .word $0000,$01C0,$0000,$0000,$01C0,$0000,$0000,$01C0,$0000,$0000,$0FF8,$0000
   .word $0000,$07F0,$0000,$0000,$03E0,$0000,$0000,$01C0,$0000,$0000,$0080,$0000
   .word 8
;* 18  Arrow up
   .word 8
   .word $0000,$0080,$0000,$0000,$01C0,$0000,$0000,$03E0,$0000,$0000,$07F0,$0000
   .word $0000,$0FF8,$0000,$0000,$01C0,$0000,$0000,$01C0,$0000,$0000,$01C0,$0000
   .word 8
;* 19  Arrow Left
   .word 11
   .word $0000,$4000,$0000,$0000,$C000,$0000,$0001,$C000,$0000,$0003,$C000,$0000
   .word $0007,$FFE0,$0000,$000F,$FFE0,$0000,$0007,$FFE0,$0000,$0003,$C000,$0000
   .word $0001,$C000,$0000,$0000,$C000,$0000,$0000,$4000,$0000
   .word 11
;* 20  Arrow Right
   .word 11
   .word $0000,$0002,$0000,$0000,$0003,$0000,$0000,$0003,$8000,$0000,$0003,$C000
   .word $0000,$07FF,$E000,$0000,$07FF,$F000,$0000,$07FF,$E000,$0000,$0003,$C000
   .word $0000,$0003,$8000,$0000,$0003,$0000,$0000,$0002,$0000
   .word 11
;*
;
;
```

```
;
  END
```

Requirements on last review 05/03/83

1. Evaluate the Lisa 1.75 Hardware ERS and prepare written suggestions for
   a) Possible in circuit diagnostic aids, and
   b) improvements or fixes to Lisa 1.0 hardware problems.
2. Fix any Daisy Wheel, Dot Matrix Printer or Mouse Diagnostic bugs or enhancements submitted by NPR.
3. Finish your evaluation of Lisa and Macintosh QuickDraw modules and prepare a plan of action to convert LisaTest and Printer/Mouse diagnostic graphics over to use QuickDraw as a base.
4. Collaborate with Gary Phillips on the Lisa 1.75 Motherboard and combination CPU / I/O board diagnostics.
5. Add more data logging capability to your diagnostics for use with LisaTest level2 mode.
6. Continue to be involved and get more involved with the Advanced Development Store and Forward voice project.
7. In addition to your involvement with QuickDraw, you should become thoroughly familiar with the Applications Programmer's Handbook, the Workshop enviroment and the OS.

```
Function: Assembly graphics routines, called by PASCAL unit BASICS or
          directly from the Boot ROM.


==================================================================

           Name: DeskTop
       Function: Draw a blank desktop
Inputs required: None
        Outputs: None
Calling sequence:
   Assembly -      JSR  DeskTop

   Pascal   -      DeskTop
==================================================================

           Name: HLine
       Function: Draw a horizontal line.
Inputs required: HLine(Y1,X1,X2: integer)
        Outputs: None
Calling sequence:
   Assembly -      move.W  d1,-(SP)          ;y1
                   move.W  d2,-(SP)          ;x1
                   move.W  d3,-(SP)          ;x2
                   JSR     HLine

   Pascal   -      HLine(Y1,X1,X2);

==================================================================

           Name: VLine
       Function: Draw a vertical line.
Inputs required: VLine(X1,Y1,Y2: integer)
        Outputs: None
Calling sequence:
   Assembly -      move.W  d1,-(SP)          ;x1
                   move.W  d2,-(SP)          ;y1
                   move.W  d3,-(SP)          ;y2
                   JSR     VLine

   Pascal   -      VLine(X1,Y1,Y2);

==================================================================

           Name: ALine
       Function: Draws a line, horz or vertical, else does nothing.
Inputs required: ALine(X1,Y1,X2,Y2,Line_Width: integer);
        Outputs: None
Calling sequence:
   Assembly -      move.W  d1,-(SP)          ;x1
                   move.W  d2,-(SP)          ;y1
                   move.W  d3,-(SP)          ;x2
                   move.W  d4,-(SP)          ;y2
                   move.W  d5,-(SP)          ;Line_Width
                   JSR     ALine

   Pascal   -      ALine(X1,Y1,X2,Y2,Line_Width);

==================================================================

           Name: ASquared_Box
       Function: Draw a delta line to new position from current position.
Inputs required: ASquared_Box(X1,Y1,X2,Y2,Line_Width: integer);
        Outputs: None
Calling sequence:
   Assembly -      move.W  d1,-(SP)          ;x1
                   move.W  d2,-(SP)          ;y1
                   move.W  d3,-(SP)          ;x2
                   move.W  d4,-(SP)          ;y2
                   move.W  d5,-(SP)          ;Line_Width
                   JSR     ASquared_Box

   Pascal   -      ASquared_Box(X1,Y1,X2,Y2,Line_Width);

==================================================================

           Name: AWhite_Box
       Function: Fill an area with white.
Inputs required: AWhite_Box(X1,Y1,X2,Y2: integer);
        Outputs: None
Calling sequence:
   Assembly -      move.W  d1,-(SP)          ;x1
                   move.W  d2,-(SP)          ;y1
                   move.W  d3,-(SP)          ;x2
                   move.W  d4,-(SP)          ;y2
                   JSR     AWhite_Box

   Pascal   -      AWhite_Box(X1,Y1,X2,Y2);

==================================================================
          Name: ABlack_Box
      Function: Fill an area with black.
Inputs required: ABlack_Box(X1,Y1,X2,Y2: integer);
       Outputs: None
Calling sequence:
   Assembly -       move.W  d1,-(SP)            ;x1
                    move.W  d2,-(SP)            ;y1
                    move.W  d3,-(SP)            ;x2
                    move.W  d4,-(SP)            ;y2
                    JSR     ABlack_Box

   Pascal   -       ABlack_Box(X1,Y1,X2,Y2);

===================================================================

          Name: AGrey_Box
      Function: Fill an area with Grey.
Inputs required: AGrey_Box(X1,Y1,X2,Y2: integer);
       Outputs: None
Calling sequence:
   Assembly -       move.W  d1,-(SP)            ;x1
                    move.W  d2,-(SP)            ;y1
                    move.W  d3,-(SP)            ;x2
                    move.W  d4,-(SP)            ;y2
                    JSR     AGrey_Box

   Pascal   -       AGrey_Box(X1,Y1,X2,Y2);

===================================================================

          Name: ALGrey_Box
      Function: Fill an area with Light Grey.
Inputs required: ALGrey_Box(X1,Y1,X2,Y2: integer);
       Outputs: None
Calling sequence:
   Assembly -       move.W  d1,-(SP)            ;x1
                    move.W  d2,-(SP)            ;y1
                    move.W  d3,-(SP)            ;x2
                    move.W  d4,-(SP)            ;y2
                    JSR     ALGrey_Box

   Pascal   -       ALGrey_Box(X1,Y1,X2,Y2);

===================================================================

          Name: AInvert_Box
      Function: Invert an area.
Inputs required: AInvert_Box(X1,Y1,X2,Y2: integer);
       Outputs: None
Calling sequence:
   Assembly -       move.W  d1,-(SP)            ;x1
                    move.W  d2,-(SP)            ;y1
                    move.W  d3,-(SP)            ;x2
                    move.W  d4,-(SP)            ;y2
                    JSR     AInvert_Box

   Pascal   -       AInvert_Box(X1,Y1,X2,Y2);

===================================================================

          Name: Paint_String
      Function: Print text string on screen
Inputs required: Paint_String(X1,Y1: integer;var DStr:String255);
       Outputs: None
Calling sequence:
   Assembly -       move.W  d1,-(SP)            ;x1
                    move.W  d2,-(SP)            ;y1
                    lea     Dstr,a1
                    move.l  a1,-(SP)            ;address of DStr
                    JSR     Paint_String
   Note: example of string has number of characters as first byte.
DStr    .Byte    40
        .ASCII  'This line is 40 characters long.........'
        .Byte   0

   Pascal   -       DStr:= 'This is a sample string.';
                    Paint_String(X1,Y1,DStr, );

===================================================================

          Name: Paint_Ch
      Function: Print a character on screen
Inputs required: Paint_Ch(X1,Y1: integer; Ch:char);
       Outputs: None
Calling sequence:
   Assembly -       move.W  d1,-(SP)            ;x1
                    move.W  d2,-(SP)            ;y1
                    move.w  d3,-(SP)            ;character
                    JSR     Paint_Ch

   Pascal   -       Ch:= 'A';
                    Paint_Ch(X1,Y1,Ch);

===================================================================

          Name: ASize_String
       Function: Calculate new X if string is printed.
 Inputs required: ASize_String(X1,Y1:integer;var Str:String255;
                      var SX1:integer);
          Outputs: None
 Calling sequence:
    Assembly -      move.W  d1,-(SP)          ;x1
                    move.W  d2,-(SP)          ;y1
                    lea     Str,a1
                    move.l  a1,-(SP)          ;address of Str
                    lea     SX1,a1
                    move.l  a1,-(SP)          ;address of SX1
                    JSR     ASize_String

    Pascal   -      Str:= 'Where will X be after this line?';
                    ASize_String(X1,Y1,Str,SX1);

===================================================================

             Name: ADraw_Integer
         Function: Draw integer on screen, POSITIVE integers only.
 Inputs required: ADraw_Integer(X1,Y1:integer; Num,Digits:integer;
                    Right_Justify:boolean);
          Outputs: None
 Calling sequence:
    Assembly -      move.W  d1,-(SP)          ;x1
                    move.W  d2,-(SP)          ;y1
                    move.W  d3,-(SP)          ;Num
                    move.W  d4,-(SP)          ;Digits
                    move.W  d5,-(SP)          ;Right_Justify
                    JSR     ADraw_Integer

    Pascal   -      Num:= 123; Digits:= 4; Right_Justify:= True;
                    ADraw_Integer(X1,Y1,Num,Digits,Right_Justify);

===================================================================

             Name: ADraw_Hex
         Function: Draw Hex integer on screen, for word values.
 Inputs required: ADraw_Hex(X1,Y1:integer; Num:integer);
          Outputs: None
 Calling sequence:
    Assembly -      move.W  d1,-(SP)          ;x1
                    move.W  d2,-(SP)          ;y1
                    move.W  d3,-(SP)          ;Hex Num
                    JSR     ADraw_Hex

    Pascal   -      Num:= $FA0123;
                    ADraw_Hex(X1,Y1,Num);

===================================================================

             Name: ADraw_LHex
         Function: Draw Hex integer on screen, for long word values.
 Inputs required: ADraw_LHex(X1,Y1:integer; Num:longint);
          Outputs: None
 Calling sequence:
    Assembly -      move.W  d1,-(SP)          ;x1
                    move.W  d2,-(SP)          ;y1
                    move.l  d3,-(SP)          ;Hex Num
                    JSR     ADraw_LHex

    Pascal   -      Num:= $FA0123;
                    ADraw_LHex(X1,Y1,Num);

===================================================================

             Name: AFolder
         Function: Draw a folder.
 Inputs required: AFolder(X1,Y1,X2,Y2:integer; var DStr:String255);
          Outputs: None
 Calling sequence:
    Assembly -      move.W  d1,-(SP)          ;x1
                    move.W  d2,-(SP)          ;y1
                    move.W  d3,-(SP)          ;x2
                    move.W  d4,-(SP)          ;y2
                    lea     DStr,a1
                    move.l  a1,-(SP)          ;address of DStr
                    JSR     AFolder

    Pascal   -      DStr:= 'Folder Name';
                    AFolder(X1,Y1,X2,Y2,DStr);

===================================================================

             Name: ADialog
         Function: Draw a dialog box.
 Inputs required: ADialog(X1,Y1,X2,Y2:integer);
          Outputs: None
 Calling sequence:
    Assembly -      move.W  d1,-(SP)          ;x1
                    move.W  d2,-(SP)          ;y1
                    move.W  d3,-(SP)          ;x2
                    move.W  d4,-(SP)          ;y2
                    JSR     ADialog

    Pascal   -      ADialog(X1,Y1,X2,Y2);
```

```
===================================================================

           Name: AIcon_Draw
       Function: Draw an Icon.
 1  Big board          2  memory board       3  Expansion card      4  diskette
 5  drive              6  insert disk        7  profile             8  lisa
 9  keybd             10  mouse             11  keyboard out       12  mouse out
13  2 port card point to port 1             14  2 port card point to port 2
15  wait              16  checkmrk          17  Arrow down         18  Arrow up
19  Arrow Left        20  Arrow Right
 Inputs required: AIcon_Draw(X1,Y1,Icon_Code: integer);
         Outputs: None
Calling sequence:
   Assembly -      move.W  d1,-(SP)          ;x1
                   move.W  d2,-(SP)          ;y1
                   move.W  d3,-(SP)          ;Icon_Code
                   JSR     AIcon_Draw

   Pascal   -      AIcon_Draw(X1,Y1,Icon_Code: integer);

===================================================================
        .title  'SCC talk program for ROM'

;***    define symbols for use in code
SCCb    .equ    $FCD201         ; B-port base
ctrlR   .equ    0               ; offset of control
dataR   .equ    4               ;   "      " data

RCA     .equ    0               ; ReceiveCharacterAvail (RR0)
TBE     .equ    2               ; TransmitBufferEmpty (RR0)
ACK     .equ    $06             ; acknowlege byte

;***    definitions of reg save area.
tRegs   .equ    $0500           ; for debugging
tDregs  .equ    tRegs+$00
tAregs  .equ    tRegs+$20
tSSP    .equ    tAregs+$1C
tIV     .equ    tRegs+$40       ; reason for interrupt
tFlags  .equ    tRegs+$41
tSR     .equ    tRegs+$42       ; Status Reg
tPC     .equ    tRegs+$44       ; Program Counter
tFC     .equ    tRegs+$48       ; Function Code (error)
tAA     .equ    tRegs+$4A       ; error Address
tIR     .equ    tRegs+$4C       ; Instruction Register
tUSP    .equ    tRegs+$50       ; User StackPtr
tCSP    .equ    tRegs+$54       ; SP used for call

pCount  .equ    tRegs+$60       ; packet length
pCksm   .equ    tRegs+$62       ; checksum
RcvPkt  .equ    tRegs+$70       ; packet area
;***    offsets of data within packets
Cmd     .equ    0               ; command itself (byte)
Byte    .equ    1               ; param byte
Count   .equ    2               ; length (integer)
Addr    .equ    4               ; address (long)
Data    .equ    8               ; data (start)

        .proc   ROMtalk,0
;*************************************************************************
;*      ROMtalk - a slightly fancy version of the old reliable TALK program
;* which runs on a ROM and "talks" over the SCC portB to a Lisa.

;**     intitialize my ptrs, etc.
        lea     xF0,A0
        move.l  A0,$002C        ; fake our only one of interest
        lea     xFFFF,A0        ; and fake a dummy
        move.l  A0,tPC
        move    #$2700,tSR
        lea     ROMtalk,A0
        suba    #$100,A0        ; reserve room
        move.l  A0,tSSP

;***    initialize the SCC
        move.l  #SCCb,A6        ; base of chip
        tst.b   (A6)            ;  force reg-ptr to 0
        lea     IDatB,A0        ; point to init table

@1      move    (A0)+,D0        ; get the next init word
        beq.s   GetCmd          ;  skip out at end of list
        move.b  D0,(A6)         ; send reg# to chip
        lsr     #8,D0           ; get data value (and add delay)
        move.b  D0,(A6)         ;  send data to chip
        bra.s   @1              ; loop (and delay)

        .page
;*************************************************************************
;***    Main command loop here.

GetCmd  move    #RcvPkt,A3      ; point to the data area
        bsr     GetPkt          ;  and get it
        bcs.s   GetCmd          ; loop if not correct

        clr     D0              ; fetch command
        move.b  Cmd(A3),D0      ; the command itself

        cmpi    #9,D0           ; in proper range?
        bgt.s   GetCmd          ; no, ignore it
        add     D0,D0           ; else, convert to branch table offset
        move    CmdTable(D0),D0
        lea     ROMtalk,A5
        move    Count(A3),D1    ; preset count and
        move.l  Addr(A3),A1     ;  address for commands
        lea     Data(A3),A0     ; and pointer to data field
        jmp     0(A5,D0)        ; then, index off to command

CmdTable                        ;*** table offsets here folks
        .word   Echo
        .word   Go
        .word   Load
        .word   Dump
        .word   RdBytes
        .word   WrBytes
        .word   RdWords
        .word   WrWords
        .word   RdLongs
        .word   WrLongs
        .page
;*********************************************************************
;*      response codes

Reply   move    #ACK,D0
ReplyX  bsr     putB            ; send acknowledgement
        bra.s   GetCmd

Send    move    #RcvPkt,A0      ; echo the stuff
        move    Count(A0),pCount
        lea     Data(A0),A3
        bsr     PutPkt          ; send back as our response
        bra.s   GetCmd          ;  and loop

        .page
;*********************************************************************
;*      Byte (immediate) read/write routines.  The data is contained in the
;* second byte of the packet.


Go      ;** "call" program.
        movem.l tRegs,D0-D7/A0-SP       ; "restore" regs
        move.l  tPC,-(SP)               ; create RTE data
        move    tSR,-(SP)
        clr     tIV                     ; clear reason for return
        rte                             ;  and off to it

xF0     move.b  #$2C,tIV        ; tell how we got here
        move    (SP)+,tSR       ; save stuff
        move.l  (SP)+,tPC
        movem.l D0-D7/A0-SP,tRegs
        lea     ROMtalk,SP
        move.l  #SCCb,A6        ; establish addressing to SCC again
        move    #ACK,D0
        bsr     putB            ; issue response
        bra     GetCmd          ; and wait
xFFFF   .word   $FFFF,$FFFF     ; dummy to force call

Echo    ;** shared entry
        bra     Reply

Load    ;**     load an image;  the count and address should be used
        move.l  A1,A3           ; copy address for get
        move    #ACK,D0         ; send ACK to tell we're ready
        bsr     PutB
        bsr     GetPkt          ;  and read the thing
        bra     Reply           ; go back to main loop

Dump    ;**     dump an image

        .page
;*********************************************************************
;*      Byte reads/writes.  The operations are always done as single byte
;* instructions.

RdBytes ;**
@1      move.b  (A1)+,(A0)+     ; do one at a time
        subq    #1,D1
        bgt.s   @1
        bra     Send

WrBytes ;**
@1      move.b  (A0),(A1)
        move.b  (A1)+,(A0)+     ; provide echoed data
        subq    #1,D1
        bgt.s   @1
        bra     Reply

RdWords ;* do word ops
@1      move    (A1)+,(A0)+
        subq    #2,D1
        bgt.s   @1
        bra     Send

WrWords ;** word writes
@1      move    (A0),(A1)
        move    (A1)+,(A0)+
        subq    #2,D1
        bgt.s   @1
        bra     Reply

RdLongs ;* do long ops
@1      move.l  (A1)+,(A0)+
        subq    #4,D1
        bgt.s   @1
        bra     Send

WrLongs ;** long writes
@1      move.l  (A0),(A1)
        move.l  (A1)+,(A0)+
        subq    #4,D1
        bgt.s   @1
        bra     Reply

        .page
;*********************************************************************
```

```
;*      GetPkt - receive a packet into storage pointed to by A0.  Carry is
;* set on exit if an error (or timeout) occurred.

GetPkt                          ;*** entry point
        tst.b   (A6)            ;  and force reg
        move.l  A3,A0           ; get incoming ptr

;***    "noise" retry here
gp0     move.b  #$30,(A6)       ; reset errors
        moveq   #-1,D2          ; initial checksum
        clr     D0              ;  and byte value
        clr.l   D1              ; show count not yet received

;***    first word in is presumed to be the count
@0      bsr.s   get8            ; get one from chip
        bcs.s   @9              ; keep retrying on time-out
        move    D0,D1           ; the packet byte counter(hi)
        lsl     #8,D1
        bsr.s   get8
        bcs.s   @9
        move.b  D0,D1           ; count (lo)
        move    D1,pCount       ; save for caller

;***    now, get the data part of packet
@2      bsr.s   get8            ; get next byte
        bcs.s   @9              ; if time-out, retry whole thing
        move.b  D0,(A0)+        ; else, stash to buffer
        subq    #1,D1
        bgt.s   @2              ; loop over count

;***    now, get checksum and check it
        move    D2,pCksm        ;  and save our computed checksum

        bsr.s   get8            ; now, get checksum from line
        lsl     #8,D0
        bsr.s   get8
        swap    D1              ; get count to D1
        move    pCksm,D2        ; get checksum back
        eor     D2,D0           ;  checksum a match?
        beq.s   @9
        .word   $003C,0001      ; ori #1,CCR

@9      rts                     ; leave CC with result

        .page
;***************************************************************************
;*      get8    - get next byte and accumulate checksum

get8    move    #1000,D3        ; time-out counter (~ 1 msec)
@0      btst    #RCA,(A6)       ; do we have it?
        bne.s   @1              ;  yes, get the byte
        dbra    D3,@0           ; loop (if not timed out)
;       ori     #1,CCR          ; set C (to show time out)
        .word   $003C,0001
        rts

@1      add     D2,D2           ; while we delay, do part of checksum
        move.b  dataR(A6),D0
        addx    D0,D2           ;  and do the rest
        tst     D0              ; clear C (to show byte in)
        rts                     ; then, leave

        .page
;***************************************************************************
;*      PutPkt - send a packet (A0 ptr)

PutPkt                          ;*** entry point
        tst.b   (A6)            ;  force reg
        moveq   #-1,D2          ; initial checksum
        move.l  A3,A0           ; copy for transfer

;***    start transmission here
        clr     D0
        move.b  pCount,D0       ; do the count (hi first)
        bsr.s   put8
        move    pCount,D1       ; now, rest of count
        move.b  D1,D0           ; lo-part last
        bsr.s   put8


;***    send packet
@1      move.b  (A0)+,D0
        bsr.s   put8
        subq    #1,D1
        bgt.s   @1

;***    and checksum
        move    D2,D0
        ror     #8,D0           ; hi-byte first
        bsr.s   put8
        ror     #8,D0           ;  then, lo-byte
        bsr.s   put8

;***    return stuff
        rts
        .page
;***************************************************************************
;*      putB    - put a byte and calc checksum

putB    btst    #TBE,(A6)       ; ready yet?
        bne.s   @1              ;  yep, send it out
        bra.s   putB            ; else, wait

@1      add     D2,D2           ; while we delay, do checksum
        addx    D0,D2
        move.b  D0,dataR(A6)    ;  and send it
        rts

        .page
;***************************************************************************
;*      data storage, etc.

IdatB   .word   $4209           ; reset channel B
        .word   $4404           ; 16x, 1-stop (async)
        .word   $000A           ; NRZ
        .word   $500B           ; BR gen for Tx, Rx
        .word   $010C           ; constant for 125K baud
        .word   $000D
        .word   $030E           ; start BR
        .word   $0001           ; allow intr's on Rx, Tx
        .word   $C103           ; 8-bits, RxEN
        .word   $6A05           ; 8-bits, TxEN, DTR, RTS (422 mode)
        .word   0               ; end-of-list

        .end
```

```
;
; Edit Date: 07/21/83
;
;    File: SERVICE.TEXT
; Purpose: This is the debugger, service mode, for the Lisa 1.75 Boot ROM
;
;___________________________________________________________________________
;
        .PROC   SERVICEMODE,0
        .ref    AGrey_Box,AWhite_Box,Paint_Ch,Paint_String,ADraw_LHex,ADraw_Hex
        .ref    AFolder,ADialog,VLINE,HLINE,DeskTop,AInvert_Box,ROMTalk
        .DEF    OFFSERVICE,SERVICE
;
;
XCOMMAND        .EQU    144
;
LEV2VCT         .EQU    $0068
;
;  Equates for COPS use
;
VIABASE         .EQU    $FCDD81         ;BASE ADDR FOR 6522
PORTB           .EQU    0               ;PORT B DATA REG
PORTA           .EQU    2               ;PORT A DATA REG
DDRB            .EQU    4               ;PORT B DATA DIRECTION REG
DDRA            .EQU    6               ;PORT A DATA DIRECTION REG
ACR             .EQU    22              ;AUXILIARY DATA REG
PCR             .EQU    24              ;PERIPHERAL CONTROL REG
IFR             .EQU    26              ;INTERRUPT FLAG REG
IER             .EQU    28              ;INTERRUPT ENABLE REG
;
CCOL            .EQU    0               ;In KEYBOARD table
XXXXXX          .EQU    2
FLAG            .EQU    4               ;Flag word
DONE            .EQU    0               ;Bit 0 of FLAG, set means to exit
CONT            .EQU    1               ;Bit 1 of FLAG, set means to exit cont
COPSWATCH       .EQU    2               ;Bit 2 of FLAG, set means to monitor COPS
WCONT           .EQU    3               ;Bit 3 of FLAG, set means DCW, not DCB
ODDADDR         .EQU    4               ;Bit 4 of FLAG, set means odd address
BUGMODE         .EQU    5               ;Bit 5 of FLAG, set in LISA DEBUG
MOUSECNT        .EQU    6               ;Counter for mouse codes
ROW             .EQU    8               ;Row in folder display
COPSROW         .EQU    10              ;Row in dialog display
COPSCOL         .EQU    12              ;Col in dialog display
;
        move.L  (SP)+,A0        ;save return address
;
        movem.l d0-d7/a0-a6,-(sp)  ;Save everyone
;
        bsr     NewService      ;New service desktop
;
        bsr     SETINTERRUPTS           ;Setup interrupt vectors
        bsr     HARDWARE                ;Setup hardware
;
@2      btst    #DONE,FLAG(a5)          ;Ready for command
        beq     @2
;
        bclr    #DONE,FLAG(a5)          ;Ready for next command
        lea     LKBDQ,a4
        move.b  (a4)+,d7
        cmp.b   #$7E,d7
        bne     @100
        move.b  (a4),d7
        cmp.b   #"X",d7                 ;Get out?
        beq     @99
        cmp.b   #"1",d7                 ;Lisa debug?
        beq     @5
        cmp.b   #"2",d7                 ;Call program?
        beq     @6
        cmp.b   #"3",d7                 ;Execute a test?
        beq     @6
        cmp.b   #"4",d7                 ;Loop on a test?
        beq     @6
        cmp.b   #"5",d7                 ;Adjust L video?
        beq     @9
        cmp.b   #"6",d7                 ;Adjust M video?
        beq     @10
        cmp.b   #"7",d7                 ;Power Cycle?
        beq     @10
        cmp.b   #"8",d7                 ;Help?
        beq     @12
        cmp.b   #"9",d7                 ;Talk, Serial B?
        beq     @14
        bra     @100
;
@5      bsr     ONDEBUG
        bset    #BUGMODE,FLAG(a5)       ;Bit 5 of FLAG, set in LISA DEBUG
        bsr     Service
        bclr    #BUGMODE,FLAG(a5)       ;Bit 5 of FLAG, set in LISA DEBUG
        bsr     NewService
        bra     @2
;
@6      bsr     DoCALL
        bsr     NewService
        bra     @2
;
@9      bsr     DoLVideo
```

```
        bsr     NewService
        bra     @2
;
@10     bsr     DoMVideo
        bsr     NewService
        bra     @2
;
@12     bsr     SFolder                 ;Place service folder on the screen
        bsr     Help
@13     btst    #DONE,FLAG(a5)          ;Ready for command
        beq     @13
;
        bclr    #DONE,FLAG(a5)     ;Ready for next command
        bsr     NewService
        bra     @2
;
@14     bsr     SFolder            ;Place service folder on the screen
        move.W  #200,-(SP)         ;x1
        move.W  #220,-(SP)         ;y1
        lea     TTalk,a1
        move.L  a1,-(SP)           ;string address
        jsr     Paint_String
        jsr     ROMTalk
        bsr     NewService
        bra     @2
;
@99     bsr     RESTORE                 ;Restore hardware
;
        movem.l (sp)+,d0-d7/a0-a6       ;Restore the world
        jmp     (a0)
;
@100    move.w  #10,d6
        bsr      What
        bra      @2
;
;
;
;
NewService         ;New service desktop
        jsr     DeskTop            ;Make a blank desktop
;
        move.W  #10,-(SP)          ;x1
        move.W  #10,-(SP)          ;y1
        lea     Options,a1         ;Dstr:= 'OPTIONS';
        move.L  a1,-(SP)           ;string address
        jsr     Paint_String
;
        move.W  #5,-(SP)           ;x1
        move.W  #20,-(SP)          ;y1
        move.W  #170,-(SP)         ;x2
        move.W  #160,-(SP)         ;y2
        jsr     ADialog            ;Make "pull down" tab
;
        lea     Level,a5
        lea     Selections,a6
;
        clr.w   d7
@1      move.W  #10,-(SP)          ;x1
        move.W  (a5)+,-(SP)        ;y1
        move.L  a6,-(SP)           ;string address
        jsr     Paint_String
;
        adda    #22,a6
        add.w   #1,d7
        cmp.w   #10,d7
        bne     @1
;
        bsr     InitQueues
        rts
;
;
InitQueues
        lea      KEYBOARD,a5            ;Variable table
        move.w   #0,CCOL(a5)            ;Init command column
        move.w   #0,FLAG(a5)
        move.w   #0,MOUSECNT(a5)
        move.w   #0,ROW(a5)
;
        lea      KBDQ,a0                ;CHARACTER QUEUE
        lea      LKBDQ,a1               ;Last command
        clr.l    d0
        clr.l    d1
@1      move.b   d1,(a0)+
        move.b   d1,(a1)+
        add.w    #1,d0
        cmp.w    #40,d0
        bne      @1
        rts
;
;
Options  .Byte    8
         .ASCII   'OPTIONS'
         .Byte    0
;
TTalk    .Byte    26
         .ASCII   'TALK thru Serial Port B...'
        .Byte   0
;
;
Level   .word   25,35,50,60,75,85,100,115,130,145
Selections
        .Byte   20
        .ASCII  ' Lisa debug       '
        .Byte   $7E
        .ASCII  '1 '
        .Byte   0
        .Byte   20
        .ASCII  '*Call program     '
        .Byte   $7E
        .ASCII  '2 '
        .Byte   0
        .Byte   20
        .ASCII  '*Execute a test   '
        .Byte   $7E
        .ASCII  '3 '
        .Byte   0
        .Byte   20
        .ASCII  '*Loop on a test   '
        .Byte   $7E
        .ASCII  '4 '
        .Byte   0
        .Byte   20
        .ASCII  ' Adjust L video   '
        .Byte   $7E
        .ASCII  '5 '
        .Byte   0
        .Byte   20
        .ASCII  ' Adjust M video   '
        .Byte   $7E
        .ASCII  '6 '
        .Byte   0
        .Byte   20
        .ASCII  '*Power Cycle      '
        .Byte   $7E
        .ASCII  '7 '
        .Byte   0
        .Byte   20
        .ASCII  ' Help             '
        .Byte   $7E
        .ASCII  '8 '
        .Byte   0
        .Byte   20
        .ASCII  ' Talk, Serial B   '
        .Byte   $7E
        .ASCII  '9 '
        .Byte   0
        .Byte   20
        .ASCII  ' Exit             '
        .Byte   $7E
        .ASCII  'X '
        .Byte   0
;
;
;
;-------------------------------------------------------------------------
;
DoMVideo
        MOVE.W  #0,-(SP)                ;x1
        MOVE.W  #0,-(SP)                ;y1
        MOVE.W  #719,-(SP)              ;x2
        MOVE.W  #363,-(SP)              ;y2
        jsr     AWhite_Box
;
        move.w  #8,d5                   ;******** On Mac, to 12   Number of lines
        clr.l   d6                      ;Y value
        move.w  #45,d7                  ;Increment
@1      MOVE.W  d6,-(SP)                ;y1
        MOVE.W  #0,-(SP)                ;x1
        MOVE.W  #719,-(SP)              ;x2
        jsr     HLINE
        add.w   d7,d6                   ;Next y value
        dbeq    d5,@1
;
        move.w  #16,d5                  ;Number of lines
        clr.l   d6                      ;Y value
        move.w  #45,d7                  ;Increment
@2      MOVE.W  d6,-(SP)                ;x1
        MOVE.W  #0,-(SP)                ;y1
        MOVE.W  #363,-(SP)              ;y2
        jsr     VLINE
        add.w   d7,d6                   ;Next y value
        dbeq    d5,@2
;
        MOVE.W  #0,-(SP)                ;x1
        MOVE.W  #0,-(SP)                ;y1
        MOVE.W  #719,-(SP)              ;x2
        MOVE.W  #363,-(SP)              ;y2
        jsr     AInvert_Box
;
@3      btst    #DONE,FLAG(a5)  ;Ready for command
        beq     @3
;
```

```
        bclr      #DONE,FLAG(a5)  ;Ready for next command
;
         rts
;_____________________________________________________________________
;
DoLVideo
        MOVE.W  #0,-(SP)             ;x1
        MOVE.W  #0,-(SP)             ;y1
        MOVE.W  #719,-(SP)           ;x2
        MOVE.W  #363,-(SP)           ;y2
        jsr     AWhite_Box
;
        move.w  #13,d5               ;Number of lines
        clr.l   d6                   ;Y value
        move.w  #28,d7               ;Increment
@1      MOVE.W  d6,-(SP)             ;y1
        MOVE.W  #0,-(SP)             ;x1
        MOVE.W  #719,-(SP)           ;x2
        jsr     HLINE
        add.w   d7,d6                ;Next y value
        dbeq    d5,@1
;
        move.w  #16,d5               ;Number of lines
        clr.l   d6                   ;Y value
        move.w  #45,d7               ;Increment
@2      MOVE.W  d6,-(SP)             ;x1
        MOVE.W  #0,-(SP)             ;y1
        MOVE.W  #363,-(SP)           ;y2
        jsr     VLINE
        add.w   d7,d6                ;Next y value
        dbeq    d5,@2
;
        MOVE.W  #0,-(SP)             ;x1
        MOVE.W  #0,-(SP)             ;y1
        MOVE.W  #719,-(SP)           ;x2
        MOVE.W  #363,-(SP)           ;y2
        jsr     AInvert_Box
;
@3      btst      #DONE,FLAG(a5)  ;Ready for command
        beq       @3
;
        bclr      #DONE,FLAG(a5)  ;Ready for next command
;
         rts
;_____________________________________________________________________
;
DoCall
         bsr      ONDEBUG            ;Make folder and command line
;
         rts
;_____________________________________________________________________
;
SFolder
         move.W  #30,-(SP)              ;x1
         move.W  #90,-(SP)              ;y1
         move.W  #690,-(SP)             ;x2
         move.W  #355,-(SP)             ;y2
         lea     SHeader,a1
         move.L  a1,-(SP)               ;string address
         jsr     AFolder                ;Place service folder on the screen
         rts
;
SHeader  .Byte    12
         .ASCII   'Service Mode'
         .Byte    0
;_____________________________________________________________________
;
ONDEBUG
         movem.l d0-d7/a0-a6,-(sp)  ;Save everyone
;
         bsr      ClrCommand            ;Clear command line
;
         bsr      SFolder               ;Place service folder on the screen
;
         move.W  #117,-(SP)             ;y1
         move.W  #30,-(SP)              ;x1
         move.W  #690,-(SP)             ;x2
         jsr     HLine
;
         move.W  #40,-(SP)              ;x1
         move.W  #108,-(SP)             ;y1
         lea     SCommand,a1
         move.L  a1,-(SP)               ;string address
         jsr     Paint_String
;
         movem.l (sp)+,d0-d7/a0-a6    ;Restore the world
         rts
;
SCommand .Byte    8
         .ASCII   'Command:'
         .Byte    0
;
;*********************************************************************
;
OFFSERVICE
         movem.l d0-d7/a0-a6,-(sp)  ;Save everyone
```

```
;
        bsr     ClrCommand          ;Clear command line
;
        move.W  #30,-(SP)           ;x1
        move.W  #100,-(SP)          ;y1
        move.W  #690,-(SP)          ;x2
        move.W  #355,-(SP)          ;y2
        jsr     AGrey_Box           ;Clear folder off the screen
;
        movem.l (sp)+,d0-d7/a0-a6   ;Restore the world
        rts
;
;*********************************************************************
ClrCommand
        move.W  #40,-(SP)           ;x1
        move.W  #110,-(SP)          ;y1
        move.W  #685,-(SP)          ;x2
        move.W  #118,-(SP)          ;y2
        jsr     AWhite_Box          ;Clear command line
        rts
;
;*********************************************************************
;
;
;--------------------------------------------------------------------------
;
SERVICE
        movem.l d0-d7/a0-a6,-(sp)       ;Save everyone
;
        bsr     DoService
;
        movem.l (sp)+,d0-d7/a0-a6       ;Restore the world
        rts
;
;--------------------------------------------------------------------------
;
DoService
;
@2      btst    #DONE,FLAG(a5)  ;Ready for command
        beq     @2
;
        bclr    #DONE,FLAG(a5)  ;Ready for next command
        lea     LKBDQ,a4
        move.b  (a4),d7
        cmp.b   #'Q',d7
        beq     @100
;
        clr.l   d1
        move.w  ROW(a5),d1      ;Clear next 4 lines
        mulu    #10,d1
        add.w   #120,d1         ;Start of first row
        move.w  d1,d2
        move.w  d1,d6
        add.w   #10,d6          ;Next row
        bsr     Clr4            ;Clear 4 command lines below
;
        move.w   ROW(a5),d5
        cmp.w    #19,d5
        bmi      @3
        move.w   #0,ROW(a5)         ;Restart at top of page
        move.w   #120,d2
        move.w   #120,d1            ;Start of first row
        move.w   #130,d6
        bsr     Clr4            ;Clear 4 command lines below
;
@3
        move.W  #60,-(SP)           ;x1
        move.W  d2,-(SP)            ;y1
        lea     SLINE,a1
        move.L  a1,-(SP)            ;string address
        jsr     Paint_String
;
        add.w   #1,ROW(a5)
;
        bsr     CLRResponse
;
        cmp.b   #'D',d7
        bne     @4
        bsr     Display
        bra     @2
;
@4      cmp.b   #'R',d7
        bne     @5
        bsr     ReadRegRb
        bra     @2
;
@5      cmp.b   #'S',d7
        bne     @6
        bsr     Setit
        bra     @2
;
@6      cmp.b   #'T',d7
        bne     @7
        bsr     TestTimer
        bra     @2
;
```

```
@7      cmp.b     #'C',d7
        bne       @8
        bsr       COPSit
        bra       @2
;
@8      cmp.b     #'M',d7
        bne       @9
        bsr       MONCOPS
        bra       @2
;
@9      cmp.b     #'X',d7
        bne       @10
        bsr       ExitCOPS
        bra       @2
;
@10     cmp.b     #'H',d7
        bne       @11
        bsr       Help
        bra       @2
;
@11     bsr       What
        bra       @2

@100      rts
;
;===================================================================
;
What
         move.W   #280,-(SP)          ;x1 normally 80
         move.W   d6,-(SP)            ;y1
          lea     Invalid,a1
         move.L   a1,-(SP)            ;string address
          jsr     Paint_String
;
         add.w    #1,ROW(a5)
         rts
;
;===================================================================
Display
        lea       Response,a0
        adda      #11,a0
        lea       Dispchar,a1
        move.b    (a1),(a0)
;
        lea       LKBDQ,a4
        adda      #1,a4
        move.b    (a4)+,d7
        cmp.b     #' ',d7
        beq       @1
        cmp.b     #'W',d7
        bne       @2
        move.b    (a4)+,d7
        cmp.b     #' ',d7
        bne       @50
@1      bsr       GetAddress
        bsr       PrtResponse
        move.w    (a3),d0
;
        move.w    #180,-(SP)        ;x1
        move.W    d1,-(SP)          ;y1
        move.w    d0,-(SP)          ; integer
        jsr       ADraw_Hex
        bra       @100
;
@2      cmp.b     #'L',d7
        bne       @3
        move.b    (a4)+,d7
        cmp.b     #' ',d7
        bne       @50
        bsr       GetAddress
        bsr       PrtResponse
        move.l    (a3),d0
;
        move.w    #180,-(SP)        ;x1
        move.W    d1,-(SP)          ;y1
        move.l    d0,-(SP)          ; integer
        jsr       ADraw_LHex
        bra       @100
;
@3      cmp.b     #'S',d7
        bne       @4
        move.b    (a4)+,d7
        cmp.b     #' ',d7
        bne       @50
        bsr       GetAddress
;
        bsr       ClrScreen
        clr.w     d6                ;Row
@31     move.w    d6,ROW(a5)
        bsr       CLRResponse       ;Clear response line
        move.l    #180,d7           ;column word
        bsr       PrtResponse       ;Line header
;
@32     move.w    (a3)+,d0          ;Read a word of data
        move.w    d7,-(SP)          ;x1
        move.W    d1,-(SP)          ;y1
        move.w  d0,-(SP)          ; integer
        jsr     ADraw_Hex
;
        add.w   #48,d7
        cmp.w   #620,d7          ;All the way across
        bmi     @32
        add.w   #1,d6
        cmp.w   #15,d6           ;Do 15 rows
        bmi     @31
        bra     @100
;
@4      cmp.b   #'C',d7          ;Display continuous, DCx
        bne     @50
        bset    #WCONT,FLAG(a5)  ;Default to word size
        bclr    #CONT,FLAG(a5)   ;Init continuous exit flag off
        move.b  (a4)+,d7
        cmp.b   #' ',d7          ;Default DC means DCW
        beq     @41
        cmp.b   #'W',d7          ;Or select DCW
        beq     @41
        bclr    #WCONT,FLAG(a5)
        cmp.b   #'B',d7          ;Select DCB
        beq     @41
        bra     @50

@41     bsr     GetAddress       ;Write address displaying
        btst    #WCONT,FLAG(a5)  ;Word or byte, set means word?
        bne     @43
;
        btst    #ODDADDR,FLAG(a5)   ;Bit 4 of FLAG, set means odd address
        beq     @42
        adda    #1,a3
@42     bsr     PrtResponse
        move.b  (a3),d7          ;...read first byte
        bra     @44

@43     bsr     PrtResponse
        move.w  (a3),d7          ;...read first word
;
@44     move.w  #180,-(SP)        ;x1
        move.W  d1,-(SP)          ;y1
        move.w  d7,-(SP)          ; integer
        jsr     ADraw_Hex         ;Print first one
;
        bclr    #COPSWATCH,FLAG(a5)  ;Turn off MONCOPS mode
        bsr     ClrDialog          ;Clear dialog box
        move.w  #80,COPSCOL(a5)
        move.w  #0,COPSROW(a5)
;
@5       btst   #WCONT,FLAG(a5)  ;Word or byte, set means word?
         beq    @52
;
@51      btst   #CONT,FLAG(a5)   ;Check flag for exiting continuous
         bne    @100
         move.w (a3),d0
         cmp.w  d0,d7
         beq    @51
         bra    @53
;
@52      btst   #CONT,FLAG(a5)   ;Check flag for exiting continuous
         bne    @100
         move.b (a3),d0
         cmp.b  d0,d7
         beq    @52
;
@53      move.w d0,d7            ;Save code
         move.w COPSCOL(a5),d6   ;Column in dialog display
         cmp.w  #30,d6
         bmi    @100
         move.w d6,-(SP)         ;x1
         move.w COPSROW(a5),d5   ;Row in dialog display
         add.w  #40,d5
         move.W d5,-(SP)         ;y1
         move.w d7,-(SP)         ; integer
         jsr    ADraw_Hex
;
         btst   #WCONT,FLAG(a5)
         bne    @54
         bsr    PaintBlank
         add.w  #8,d6
         bsr    PaintBlank
;
@54      add.w  #40,d6
;
         bsr    PaintBlank
         add.w  #8,d6
         bsr    PaintBlank
         add.w  #8,d6
         bsr    PaintBlank
         add.w  #8,d6
         bsr    PaintBlank
;
         sub.w  #24,d6
         move.w d6,COPSCOL(a5)
;
         cmp.w  #600,d6          ;Past edge?
        bmi     @5              ;...no,  continue
        move.w  #120,COPSCOL(a5)
        move.w  COPSROW(a5),d5
        add.w   #10,d5          ;Next row
        move.w  d5,COPSROW(a5)
        cmp.w   #41,d5
        bmi     @5
        move.w  #0,COPSROW(a5)
        bra     @5
;
@50     bra     What
@100    rts
;
;======================================================================
;
PaintBlank
        move.W  d6,-(SP)        ;x1
        move.W  d5,-(SP)        ;y1
        move.w  #' ',-(sp)      ;Character
        jsr     Paint_Ch        ;Place character on the screen
        rts
;
;======================================================================
;
ReadRegRb
        bclr    #CONT,FLAG(a5)  ;Clear flag for exiting continuous
        lea     LKBDQ,a4
        adda    #1,a4
        move.b  (a4)+,d7
        cmp.b   #'C',d7         ;See if RC address
        bne     @10
        move.b  (a4)+,d7
        cmp.b   #' ',d7
        bne     @50
        bsr     GetAddress
        bsr     PrtResponse
        move.w  (a3),d0
;
        move.w  #180,-(SP)       ;x1
        move.W  d1,-(SP)         ;y1
        move.w  d0,-(SP)         ;integer
        jsr     ADraw_Hex
;
@1      move.w  (a3),d5
        btst    #CONT,FLAG(a5)  ;Check flag for exiting continuous
        beq     @1
        bra     @100
;
@10
;
@50     bra     What
@100    rts
;======================================================================
Setit
        lea     Response,a0
        adda    #11,a0
        lea     Setchar,a1
        move.b  (a1),(a0)
;
        lea     LKBDQ,a4
        adda    #1,a4
        move.b  (a4)+,d7
        cmp.b   #' ',d7
        beq     @1
        cmp.b   #'W',d7
        bne     @2
        move.b  (a4)+,d7
        cmp.b   #' ',d7
        bne     @50
@1      bsr     SetWord
        move.w  d5,(a3)
        bra     @100
;
@2      cmp.b   #'L',d7
        bne     @3
        move.b  (a4)+,d7
        cmp.b   #' ',d7
        bne     @50
        bsr     SetLong
        move.l  d5,(a3)
        bra     @100
;
@3      cmp.b   #'B',d7
        bne     @4
        move.b  (a4)+,d7
        cmp.b   #' ',d7
        bne     @50
        bsr     SetByte
        move.b  d5,(a3)
        bra     @100
;
@4      cmp.b   #'C',d7         ;See if continuous
        bne     @50             ;...no, must be an error
        bclr    #CONT,FLAG(a5)  ;Clear flag for exiting continuous
        move.b  (a4)+,d7        ;...yes, what form?
        cmp.b   #' ',d7         ;Default to word
        beq     @5
        cmp.b   #'W',d7            ;Was word selected?
        bne     @7                 ;...no, try another
        move.b  (a4)+,d7           ;...yes, followed by a blank?
        cmp.b   #' ',d7
        bne     @50                ;......no, must be error
@5      bsr     SetWord
@6      move.w  d5,(a3)
        btst    #CONT,FLAG(a5)     ;Check flag for exiting continuous
        beq     @6
        bra     @100
;
@7      cmp.b   #'L',d7            ;Continuous Long?
        bne     @9                 ;...no, try another
        move.b  (a4)+,d7
        cmp.b   #' ',d7            ;Followed by a blank?
        bne     @50                ;...no, must be an error
        bsr     SetLong
@8      move.l  d5,(a3)
        btst    #CONT,FLAG(a5)     ;Check flag for exiting continuous
        beq     @8
        bra     @100
;
@9      cmp.b   #'B',d7            ;Is it continuous byte?
        bne     @50
        move.b  (a4)+,d7
        cmp.b   #' ',d7
        bne     @50
        bsr     SetByte
@10     move.b  d5,(a3)
        btst    #CONT,FLAG(a5)     ;Check flag for exiting continuous
        beq     @10
        bra     @100
;
@50     bra     What
@100    rts
;
;
SetWord
        bsr     GetAddress
        bsr     PrtResponse
        bsr     GetData
        move.w  #180,-(SP)          ;x1
        move.W  d1,-(SP)            ;y1
        move.w  d5,-(SP)            ;integer
        jsr     ADraw_Hex
        rts
;
SetLong
        bsr     GetAddress
        bsr     PrtResponse
        bsr     GetData
        move.w  #180,-(SP)          ;x1
        move.W  d1,-(SP)            ;y1
        move.l  d5,-(SP)            ;integer
        jsr     ADraw_LHex
        rts
;
SetByte
        bsr     GetAddress
        btst    #ODDADDR,FLAG(a5)    ;Bit 4 of FLAG, set means odd address
        beq     @1
        adda    #1,a3
@1      bsr     PrtResponse
        bsr     GetData
        move.w  #180,-(SP)          ;x1
        move.W  d1,-(SP)            ;y1
        move.w  d5,-(SP)            ;integer
        jsr     ADraw_Hex
        move.W  #180,-(SP)          ;x1
        move.W  d1,-(SP)            ;y1
        move.w  #' ',-(sp)          ;Character
        jsr    Paint_Ch             ;Place character on the screen
        move.W  #188,-(SP)          ;x1
        move.W  d1,-(SP)            ;y1
        move.w  #' ',-(sp)          ;Character
        jsr    Paint_Ch             ;Place character on the screen
        rts
;
;==================================================================
TestTimer
        lea     Response,a0
        adda    #11,a0
        lea     Setchar,a1
        move.b  (a1),(a0)
;
        bclr    #CONT,FLAG(a5)     ;Clear flag for exiting continuous
        lea     LKBDQ,a4
        adda    #1,a4
        move.b  (a4)+,d7
        cmp.b   #'S',d7            ;Is this a TS instruction?
        bne     @4
        move.b  (a4)+,d7           ;Is it TS or TSR?
        cmp.b   #' ',d7
        bne     @2
        bsr     GetAddress
        bsr      PrtResponse
        move.W   #180,-(SP)           ;x1
        move.W   d1,-(SP)            ;y1
        lea      TSPAT,a1
        move.L   a1,-(SP)              ;string address
        jsr      Paint_String
@1      move.w   #$0000,(a3)
        move.w   #$FFFF,(a3)
        move.w   #$AAAA,(a3)
        move.w   #$5555,(a3)
        move.w   #$AAAA,(a3)
        move.w   #$5555,(a3)
        btst     #CONT,FLAG(a5)  ;Check flag for exiting continuous
        beq      @1
        bra      @100
;
@2      cmp.b    #'R',d7         ;Is it TSR?
        bne      @50
        bsr      GetAddress
        bsr      PrtResponse
        move.W   #180,-(SP)          ;x1
        move.W   d1,-(SP)            ;y1
        lea      TSPAT,a1
        move.L   a1,-(SP)              ;string address
        jsr      Paint_String
@3      move.w   #$0000,(a3)
        move.w   #$FFFF,(a3)
        move.w   #$AAAA,(a3)
        move.w   #$5555,(a3)
        move.w   #$AAAA,(a3)
        move.w   #$5555,(a3)
        move.w   (a3),d0
        btst     #CONT,FLAG(a5)  ;Check flag for exiting continuous
        beq      @3
        bra      @100
;
@4
        cmp.b    #'L',d7          ;Is this a TLOC instruction?
        bne      @6
        move.b   (a4)+,d7         ;Is it TLOC?
        cmp.b    #'O',d7
        bne      @50
        move.b   (a4)+,d7         ;Is it TLOC?
        cmp.b    #'C',d7
        bne      @50
        bsr      GetAddress
        bsr      PrtResponse
        bsr      GetData
        move.w   d5,d0
        bsr      GetData
@5      move.w   d0,(a3)
        move.w   d5,(a3)
        move.w   (a3),d1
        btst     #CONT,FLAG(a5)  ;Check flag for exiting continuous
        beq      @5
        bra      @100
;
@6
@50     bra      What
@100    rts
;===================================================================
;
COPSit
         lea      LKBDQ,a4
         adda     #1,a4
         move.b   (a4)+,d7
         cmp.b    #'O',d7         ;Is this a COPS instruction?
         bne      @50
         move.b   (a4)+,d7
         cmp.b    #'P',d7
         bne      @50
         move.b   (a4)+,d7
         cmp.b    #'S',d7
         bne      @50
         move.b   (a4)+,d7
         cmp.b    #' ',d7
         bne      @50
;
         clr.l    d1
         move.w   ROW(a5),d1      ;Calculate where row is
         mulu     #10,d1
         add.w    #120,d1
         move.w   #80,-(SP)         ;x1
         move.W   d1,-(SP)          ;y1
         lea      COPSHEAD,a1
         move.L   a1,-(SP)             ;string address
         jsr      Paint_String
;
         add.w    #1,ROW(a5)
;
        bsr      GetData
;
        move.w   #180,-(SP)         ;x1
        move.W   d1,-(SP)           ;y1
        move.w   d5,-(SP)           ; integer
        jsr      ADraw_Hex
        move.W  #180,-(SP)          ;x1
        move.W  d1,-(SP)            ;y1
        move.w  #' ',-(sp)          ;Character
        jsr     Paint_Ch            ;Place character on the screen
        move.W  #188,-(SP)          ;x1
        move.W  d1,-(SP)            ;y1
        move.w  #' ',-(sp)          ;Character
        jsr     Paint_Ch            ;Place character on the screen
;
        move.w  d5,d2
        bsr     LCOPSCMD
        bra     @100
;
@50     bra     What
@100    rts
;
;==================================================================
;
MONCOPS
        lea     LKBDQ,a4
        adda    #1,a4
        move.b  (a4)+,d7
        cmp.b   #'O',d7             ;Is this a MON instruction?
        bne     @50
        move.b  (a4)+,d7
        cmp.b   #'N',d7
        bne     @50
;
        move.w  #0,COPSROW(a5)      ;Row in dialog display
        move.w  #80,COPSCOL(a5)     ;Col in dialog display
;
        bsr     ClrDialog           ;Clear dialog box
        bset    #COPSWATCH,FLAG(a5)     ;Bit 2 of FLAG, set means to monitor COPS
        bra     @100
;
@50     bra     What
@100    rts
;==================================================================
;
ExitCOPS
        bclr    #CONT,FLAG(a5)  ;Check flag for exiting continuous
;
        lea     LKBDQ,a4
        adda    #1,a4
        move.b  (a4)+,d7
        cmp.b   #' ',d7             ;Is this a X     instruction?
        beq     @100
        cmp.b   #'M',d7             ;Is this a XMON instruction?
        bne     @50
        move.b  (a4)+,d7
        cmp.b   #'O',d7             ;Is this a XMON instruction?
        bne     @50
        move.b  (a4)+,d7
        cmp.b   #'N',d7
        bne     @50
;
        bclr    #COPSWATCH,FLAG(a5)
        bra     @100
;
@50     bra     What
@100    rts
;==================================================================
;
Help
         bsr    ClrScreen
;
         move.W #80,-(SP)            ;x1
         move.W #120,-(SP)           ;y1
         lea    Help1,a1
         move.L a1,-(SP)             ;string address
         jsr    Paint_String
;
         move.W #80,-(SP)            ;x1
         move.W #130,-(SP)           ;y1
         lea    Help2,a1
         move.L a1,-(SP)             ;string address
         jsr    Paint_String
;
         move.W #80,-(SP)            ;x1
         move.W #140,-(SP)           ;y1
         lea    Help3,a1
         move.L a1,-(SP)             ;string address
         jsr    Paint_String
;
         move.W #80,-(SP)            ;x1
         move.W #150,-(SP)           ;y1
         lea    Help4,a1
         move.L a1,-(SP)             ;string address
         jsr    Paint_String
;
         move.W #80,-(SP)            ;x1
         move.W #160,-(SP)           ;y1
         lea    Help5,a1
         move.L a1,-(SP)             ;string address
         jsr    Paint_String
;
```

```
        move.W  #80,-(SP)           ;x1
        move.W  #170,-(SP)          ;y1
        lea     Help6,a1
        move.L  a1,-(SP)            ;string address
        jsr     Paint_String
;
        move.W  #80,-(SP)           ;x1
        move.W  #190,-(SP)          ;y1
        lea     Help7,a1
        move.L  a1,-(SP)            ;string address
        jsr     Paint_String
;
        move.W  #80,-(SP)           ;x1
        move.W  #200,-(SP)          ;y1
        lea     Help8,a1
        move.L  a1,-(SP)            ;string address
        jsr     Paint_String
;
        move.W  #80,-(SP)           ;x1
        move.W  #220,-(SP)          ;y1
        lea     Help9,a1
        move.L  a1,-(SP)            ;string address
        jsr     Paint_String
;
        move.W  #80,-(SP)           ;x1
        move.W  #230,-(SP)          ;y1
        lea     Help10,a1
        move.L  a1,-(SP)            ;string address
        jsr     Paint_String
;
        move.W  #80,-(SP)           ;x1
        move.W  #240,-(SP)          ;y1
        lea     Help11,a1
        move.L  a1,-(SP)            ;string address
        jsr     Paint_String
;
        move.W  #80,-(SP)           ;x1
        move.W  #250,-(SP)          ;y1
        lea     Help12,a1
        move.L  a1,-(SP)            ;string address
        jsr     Paint_String
;
        move.W  #80,-(SP)           ;x1
        move.W  #260,-(SP)          ;y1
        lea     Help13,a1
        move.L  a1,-(SP)            ;string address
        jsr     Paint_String
;
        move.w   #15,ROW(a5)
;
        rts
;
;=====================================================================
;
Clr4
        move.W  #40,-(SP)           ;x1
        move.W  d1,-(SP)            ;y1
        move.W  #670,-(SP)          ;x2
        add.w   #28,d1
        move.W  d1,-(SP)            ;y2
        jsr     AWhite_Box          ;Clear 4 command lines below
        rts
;=====================================================================
;
CLRResponse
       clr.w    d2
       move.b   #' ',d1
       lea      Response,a1        ;Clear response string
       adda     #14,a1
@1     move.b   d1,(a1)+
       add.w    #1,d2
       cmp.w    #46,d2
       bne      @1
       rts
;
;=====================================================================
;
ClrDialog
       move.W  #65,-(SP)           ;x1
       move.W  #31,-(SP)           ;y1
       move.W  #635,-(SP)          ;x2
       move.W  #88,-(SP)           ;y2
       jsr     ADialog             ;Clear dialog box
       rts
;=====================================================================
;
ClrScreen
        move.W  #40,-(SP)           ;x1
        move.W  #120,-(SP)          ;y1
        move.W  #670,-(SP)          ;x2
        move.W  #340,-(SP)          ;y2
        jsr     AWhite_Box          ;Clear screen
        rts
;
;=====================================================================
;
```

```
GetAddress
        clr.l     d6
        clr.l     d7
@1      move.b    (a4)+,d7        ;Skip past any leading blanks
        cmp.b     #' ',d7
        bne       @3              ;...Not a blank, go after address
        add.w     #1,d6
        cmp.w     #30,d6          ;...No digits found safety counter
        bmi       @1
@2      clr.l     d6              ;Number not found, default to address 0
        move.l    d6,a3
        bra       @100
;
@3      clr.l     d6              ;Init address
        move.l    d6,a3
        bra       @5
;
@4      move.b    (a4)+,d7        ;Get next character
        cmp.b     #' ',d7
        beq       @100
;
@5      btst      #7,d7           ;Out of range?
        bne       @2
        cmp.b     #$30,d7         ;Below 0?
        bmi       @6
        cmp.b     #$39,d7         ;Above 9?
        bgt       @6
        and.b     #$F,d7          ;Isolate digit
        bra       @7
;
@6      cmp.b     #$41,d7         ;Below A?
        bmi       @2
        cmp.b     #$46,d7         ;Above F?
        bgt       @2
        and.b     #$F,d7          ;Isolate
        add.b     #$9,d7          ;...and convert
;
@7      adda      d7,a3           ;Add bit to address
        move.b    (a4),d7         ;See if address end
        cmp.b     #' ',d7
        beq       @100
        move.l    a3,d6
        rol.l     #4,d6           ;Open a space for the next digit
        move.l    d6,a3
        bra       @4
;
@100
        bclr      #ODDADDR,FLAG(a5)   ;Bit 4 of FLAG, set means odd address
        move.l    a3,d0
        move.l    d0,d1
        and.l     #$FFFFFFFE,d0
        move.l    d0,a3
        cmp.b     d1,d0
        beq       @101
        bset      #ODDADDR,FLAG(a5)   ;Bit 4 of FLAG, set means odd address
@101    rts
;
;==================================================================
;
GetData
        clr.l     d6
        clr.l     d7
@1      move.b    (a4)+,d7        ;Skip past any leading blanks
        cmp.b     #' ',d7
        bne       @3              ;...Not a blank, go after data
        add.w     #1,d6
        cmp.w     #30,d6          ;...No digits found safety counter
        bmi       @1
@2      clr.l     d5              ;Number not found, default to data 0
        bra       @100
;
@3      clr.l     d5              ;Init data
        bra       @5
;
@4      move.b    (a4)+,d7        ;Get next character
        cmp.b     #' ',d7
        beq       @100
;
@5      btst      #7,d7           ;Out of range?
        bne       @2
        cmp.b     #$30,d7         ;Below 0?
        bmi       @6
        cmp.b     #$39,d7         ;Above 9?
        bgt       @6
        and.b     #$F,d7          ;Isolate digit
        bra       @7
;
@6      cmp.b     #$41,d7         ;Below A?
        bmi       @2
        cmp.b     #$46,d7         ;Above F?
        bgt       @2
        and.b     #$F,d7          ;Isolate
        add.b     #$9,d7          ;...and convert
;
@7      add.l     d7,d5           ;Add bit to data
        move.b    (a4),d7         ;See if data end
        cmp.b   #' ',d7
        beq     @100
        rol.l   #4,d5           ;Open a space for the next digit
        bra     @4
;
@100    rts
;
;================================================================
;
; Expects address in a3, a5 points to variable table
;
PrtResponse
        clr.l   d1
        move.w  ROW(a5),d1      ;Calculate where row is
        mulu    #10,d1
        add.w   #120,d1
        move.w  #80,-(SP)       ;x1
        move.W  d1,-(SP)        ;y1
        lea     Response,a1
        move.L  a1,-(SP)           ;string address
        jsr     Paint_String
;
        add.w   #1,ROW(a5)
;
        move.w  #88,-(SP)       ;x1
        move.W  d1,-(SP)        ;y1
        move.l  a3,-(SP)        ;integer
        jsr     ADraw_LHex
        rts
;
;================================================================
;
;       Level 2 interrupt handler.  Handles COPS interrupts.
;
LVL2INT                                 ; Ye olde entry point.
        movem.l A0-A6/D0-D7,-(SP)       ; save required regs
        movea.l #VIABASE,A0             ; and set 6522 port base
        lea      KEYBOARD,a5    ;Variable table

        clr.l   d0
        move.b  PORTA(A0),D0            ; get the data
        bsr.s   ENQKBD                  ; QUEUE IT
        movem.l (SP)+,A0-A6/D0-D7       ; ...restore the reg
        rte                             ; ...and exit
;
;= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
;  ENQKBD - ADD CHARACTER TO QUEUE, character in d0
;
ENQKBD
        btst    #COPSWATCH,FLAG(a5)     ;Bit 2 of FLAG, set means to monitor COPS
        beq     @10
        bsr     WATCHIT
@10     clr.l   d1                      ;Initialize
        cmp.b   #$7F,d0                 ;Apple Up, treat as a return
        beq     @41
        btst    #7,d0                   ;Assure that down key was pressed
        beq     @100                    ;...else ignore and get out
        and.b   #$7F,d0
;
@11     move.w  MOUSECNT(a5),d1         ;Ignoring mice movement?
        beq     @1                      ;...none to ignore
        sub.w   #1,MOUSECNT(a5)         ;...Ignore and
        bra     @100                    ;......get out
;
@1      cmp.b   #$00,d0                 ;Start of mouse movement code sequence?
        bne     @2                      ;...no
        move.w  #2,MOUSECNT(a5)         ;...yes, expect two mode codes
        bra     @100
;
@2      lea     CONVERTCODE,a0          ;Ascii table
        move.b  (a0),d2                 ;Get a space for later compares
        adda    d0,a0                   ;...offset for character got
        clr.l   d1
        move.b  (a0),d1                 ;Ascii code
;
        cmp.b   d2,d1                   ;A space?
        bne     @3
        cmp.b   #$5C,d0                 ;...from the Space Bar?
        beq     @3                      ;...yes, ok
        bra     @100                    ;...no, exit
;
@3      cmp.b   #$45,d0                 ;Backspace?
        bne     @4                      ;...no
        move.w  CCOL(a5),d0             ;...yes, which column at?
        cmp.w   #0,d0
        beq     @100                    ;at end, ignore
;       clear last character printed and remove from que
        sub.w   #1,CCOL(a5)             ;Removed from que
;
        clr.l   d0
        move.w  CCOL(a5),d0
        mulu    #8,d0
        add.w   #XCOMMAND,d0
        move.W  d0,-(SP)                ;x1
        btst    #BUGMODE,FLAG(a5)       ;Bit 5 of FLAG, set in LISA DEBUG
        bne     @31
        move.W  #10,-(SP)               ;y1
        bra     @32
@31     move.W  #108,-(SP)              ;y1
@32     move.w  d2,-(sp)                ;Space Character
        jsr     Paint_Ch                ;Place character on the screen
        bra     @100

@4      cmp.b   #$48,d0                 ;Return?
        bne     @7                      ;...no
        bra     @42
@41     cmp.b   #$7E,d1                 ;Treat Apple up as a return
        beq     @7                      ;...no
@42     move.w  CCOL(a5),d0             ;Any command?
        beq     @100                    ;...No, ignore
        lea     KBDQ,a0                 ;CHARACTER QUEUE
        lea     LKBDQ,a1                ;Last command
        move.b  (a0),d0                 ;See if exit continuous loop
        cmp.b   #'X',d0
        bne     @43
        bset    #CONT,FLAG(a5)          ;Set flag for exiting continuous
@43     clr.l   d0
        move.w  CCOL(a5),d1
@5      move.b  (a0)+,(a1)+             ;Move to command buffer
        add.w   #1,d0
        cmp.w   d0,d1
        bne     @5
        move.b  #' ',d1                 ;Clear rest of command buffer
@6      move.b  d1,(a1)+
        add.w   #1,d0
        cmp.w   #40,d0
        bne     @6
        bset    #DONE,FLAG(a5)
        move.w  #0,CCOL(a5)
;
        move.W  #112,-(SP)              ;x1
        btst    #BUGMODE,FLAG(a5)       ;Bit 5 of FLAG, set in LISA DEBUG
        bne     @61
        move.W  #10,-(SP)               ;y1
        bra     @62
@61     move.W  #108,-(SP)              ;y1
@62     move.W  #600,-(SP)              ;x2
        btst    #BUGMODE,FLAG(a5)       ;Bit 5 of FLAG, set in LISA DEBUG
        bne     @63
        move.W  #18,-(SP)               ;y1
        bra     @64
@63     move.W  #116,-(SP)              ;y1
@64     jsr     AWhite_Box              ;Clear command line
        bra     @100
;
@7      lea     KBDQ,A2                 ;Set queue address
        clr.l   d4
        move.w  CCOL(a5),d4             ;find position in queue
        cmp.w   #38,d4
        beq     @100                    ;overflow
        adda    d4,a2
        move.b  d1,(a2)                 ;Place ascii in queue
;
        clr.l   d0
        move.w  CCOL(a5),d0
        mulu    #8,d0
        add.w   #XCOMMAND,d0
        move.W  d0,-(SP)                ;x1
        btst    #BUGMODE,FLAG(a5)       ;Bit 5 of FLAG, set in LISA DEBUG
        bne     @8
        move.W  #10,-(SP)               ;y1
        bra     @9
@8      move.W  #108,-(SP)              ;y1
@9      move.w  d1,-(sp)                ;Character
        jsr     Paint_Ch                ;Place character on the screen
;
        add.w   #1,CCOL(a5)
;
@100    rts
;
CONVERTCODE
        .ASCII  '       ↑         '   ;0x
        .ASCII  '                 '   ;1x
        .ASCII  ' /*=789-456+.23  '   ;2x
        .ASCII  '                 '   ;3x
        .ASCII  '-+↑>P↑  %0  /1   '   ;4x
        .ASCII  '90UIJK[]ML;   ,.O'   ;5x
        .ASCII  'E6785RTY\FGHVCBN'    ;6x
        .ASCII  'A2341QSW ZXD     '   ;7x
        .Byte   $7E
;               0123456789ABCDEF
;
;______________________________________________________________
;
WatchIt
        move.w  d0,d7           ;Save keycode
        move.w  COPSCOL(a5),d6  ;Column in dialog display
        cmp.w   #100,d6
        bmi     @100
        move.w  d6,-(SP)        ;x1
        move.w  COPSROW(a5),d5  ;Row in dialog display
        add.w   #40,d5
        move.W  d5,-(SP)            ;y1
        move.w  d7,-(SP)            ;integer
        jsr     ADraw_Hex
;
        bsr     PaintBlank
        add.w   #8,d6
        bsr     PaintBlank
;
        add.w   #40,d6
;
        bsr     PaintBlank
        add.w   #8,d6
        bsr     PaintBlank
;
        sub.w   #24,d6
        move.w  d6,COPSCOL(a5)
;
        cmp.w   #600,d6             ;Past edge?
        bmi     @100                ;...no,  continue
        move.w  #80,COPSCOL(a5)
        move.w  COPSROW(a5),d5
        add.w   #10,d5              ;Next row
        move.w  d5,COPSROW(a5)
        cmp.w   #41,d5
        bmi     @100
        move.w  #0,COPSROW(a5)
;
@100    move.w  d7,d0
        rts
;
;-------------------------------------------------------------------------------
;
;       Setup interrupt vector 2 for COPS interrupt handling
; by "linking" to current handler
;
SETINTERRUPTS
        move.w  #$2700,sr           ;Disable interrupts
        lea     SAVELEV2,a0
        move.l  lev2vct,a1          ;Save original level 2 vector
        move.l  a1,(a0)
        lea     LVL2INT,a1          ;Address of our handler
        move.l  a1,lev2vct          ;...into vector
        move.w  #$2000,SR           ;Enable interrupts
        rts
;-------------------------------------------------------------------------------
;
;       Initialize the COPS 6522 chip for appropriate interrupt,
;
HARDWARE        ;Setup hardware
        lea     VIABASE,a2          ; Setup our COPS base address
        lea     SAVEVIA,a1
        move.b  DDRB(a2),d0         ;Save port B
        move.b  d0,(a1)
        and.b   #$A0,d0             ;...save bits 5(PRES),7(CRES)
        ori.b   #$0E,d0             ;...add in PB1-PB3
        move.b  d0,DDRB(a2)         ;Make PB1-PB3 outputs for volume.
        move.b  #$A0,PORTB(a2)      ;Init value to port B, volume at zero.
        move.b  ACR(a2),1(a1)
        move.b  #$01,ACR(a2)        ; ...PortA latch enable
        move.b  PCR(a2),2(a1)
        move.b  #$c9,PCR(a2)        ; Handshake setup, CB2 in manual output mode.
        move.b  IER(a2),3(a1)
        move.b  #$7F,IER(a2)        ; ...clear all interrupt enables
        move.b  IFR(a2),4(a1)
        move.b  #$7F,IFR(a2)        ; ...and clear any existing bits
;
        move.b  #0,d2
        bsr     LCOPSCMD
        rts
;
;-------------------------------------------------------------------------------
;
LCOPSCMD movem.l d0-d4/a2-a3,-(sp)  ;Save registers used
;
        move.w  sr,-(sp)            ;Disable interrupts
        ori.w   #$0700,sr
;
        movea.l #$00FCDD80,a2       ;Keyboard VIA address
        clr.b   7(a2)               ;...Close port in case it was left open(DDRA)
        move.b  d2,31(a2)           ;...Place command on port (PORTA2)
;
        move.l  a2,a3
        adda    #$01,a3             ;Set address reg to access (ORB) fast
;
;1) wait for CRDY to go down
;
;  first assure donnt catch CRDY down toward end of cycle
        move.w  #$0897,d1           ;(12)set timeout
@1      subq.w  #1,d1               ;(4) decrement counter
        beq.s   @6                  ;(8) ...exit on timeout
        btst    #6,(a3)             ;Look at CRDY, wait until down
        bne.s   @1                  ;(8)
;
;  now space out into up area and look for next CRDY down
        mulu    #1,d0               ;Kill time, position for next CRDY
        move.w  #$0897,d1           ;(12)set timeout
```

```
@2      subq.w  #1,d1           ;(4)decrement counter
        beq.s   @5              ;(8)...exit on timeout
        btst    #6,(a3)         ;Wait until CRDY down, ready
        bne.s   @2              ;(8)
;
;2) setup valid data for COPS to accept
;
        move.b  #$ff,7(a2)      ;Open port to allow command to COPS,(DDRA)
;
;3) look for CRDY going up
        move.w  #$0830,d1       ;Set timeout
@3      subq.w  #1,d1           ;Decrement counter
        beq.s   @6              ;...exit on timeout
        move.b  (a3),d0         ;Wait for COPS ready
        btst    #6,d0
        beq.s   @3
;
;   hold data for COPS to read, hold time
        move.b  #10,d0
@4      subq.b  #1,d0           ;Delay for COPS to accept data
        bgt.s   @4
;
;4) turn off data
;
        clr.b   7(a2)           ;Close port, command taken by COPS, (DDRA)
        move.b  #$82,29(a2)     ;Enable CA1 for next interrupt, (IER)
        bra.s   @8              ;...get out with OK status
;
@5      move.w  #2,d7           ;COPS not responding (PB6 not high), error flag
        bra.s   @7
;
@6      move.w  #1,d7           ;COPS indicating never ready, error flag
;
; Get out with error
@7      move.w  (sp)+,sr        ;Enable interrupts
        movem.l (sp)+,d0-d4/a2-a3  ;restore registers
        move.w  #1,d0           ;error code, COPS failure
        bra     @10
;
; Get out OK
@8      move.w  (sp)+,sr        ;Enable interrupts
        movem.l (sp)+,d0-d4/a2-a3  ;restore registers
        clr.w   d0              ;OK code
;
@10     rts
;
;==========================================================================
;
; Restore vectors and conditions to pre-test state
;
RESTORE move.w  SR,d0
        move.w  #$2700,sr       ;Disable interrupts
        lea     SAVELEV2,a5
        move.l  (a5),a1         ;Restore original level 2 vector
        move.l  a1,lev2vct
        move.w  d0,SR
;
        lea     VIABASE,a2      ; Setup our COPS base address
        lea     SAVEVIA,a1
        move.b  (a1),DDRB(a2)
        move.b  1(a1),ACR(a2)     ;Restore original value
        move.b  2(a1),PCR(a2)
        move.b  3(a1),IER(a2)
        move.b  4(a1),IFR(a2)
        move.w  #$2000,sr       ;Enable interrupts
        rts
;
;=====================================================================
Invalid* .Byte   4
         .ASCII  '????'
         .Byte   0
Help1    .Byte   58
         .ASCII  'Lisa debug >    Quit       RB       Fail Buffer at xxxxxxxx'
         .Byte   0
Help2    .Byte   62
         .ASCII  'SB addr xx    SW addr/xxxx    SL addr xxxxxxxx  X (exits cont)'
         .Byte   0
Help3    .Byte   56
         .ASCII  'SCB addr xx   SCW addr xxxx   SCL addr xxxxxxxx  TS addr'
         .Byte   0
Help4    .Byte   62
         .ASCII  'COPS   xx     OW addr xxxx    OL addr xxxxxxxx   OS addr xxxx '
         .Byte   0
Help5    .Byte   58
         .ASCII  'MONCOPS       RC addr         DCB addr           DCW addr '
         .Byte   0
Help6    .Byte   60
         .ASCII  'XMONCOPS      TMODE x y       TLOC addr xxxx xxxx  TSR addr'
         .Byte   0
Help7    .Byte   62
         .ASCII  '00002000 IWM   00000000 MScreen  00006000  SCC   00000000 ROM'
         .Byte   0
Help8    .Byte   62
         .ASCII  '00004001 Timer  00001FA4 LScreen  xxxxxxxx  Stat  00007000 Err'
         .Byte   0
Help9    .Byte   62
```

```
          .ASCII  'Call program > JSR addr    Execute a test > Selected test once'
          .Byte   0
Help10    .Byte    42
          .ASCII  'Loop on a test > Selected test until fail.'
          .Byte    0
Help11    .Byte    60
          .ASCII  'Adjust L video > 720 x 364        Adjust M video > 720 x 544'
          .Byte    0
Help12    .Byte    42
          .ASCII  'Talk, Serial B > Low level external debug.'
          .Byte    0
Help13    .Byte    36
          .ASCII  'Power Cycle > Diagnostics repeatily.'
          .Byte    0

COPSHEAD .Byte     12
          .ASCII  ' COPS      > '
          .Byte   0
;
Response .Byte     60
          .ASCII  ' xxxxxxxx >
          .Byte   0
TSPAT     .Byte    30
          .ASCII  '0000 FFFF AAAA 5555 AAAA 5555 '
          .Byte   0
Dispchar .ASCII '>'
Setchar   .ASCII '<'
;
KEYBOARD        .word 0                     ;CCOL, Column, for command line
                .word 0                     ;XXXXXX
                .word 0                     ;FLAG, Flag word
                .word 0                     ;MOUSECNT, counter for mouse codes
                .word 0                     ;ROW, in folder display
                .word 0                     ;COPSROW,Row in dialog display
                .word 0                     ;COPSCOL,Col in dialog display
;
SAVEVIA         .word 0,0,0,0,0,0,0,0,0,0,0,0,0,0
;
SAVELEV2        .WORD 0,0
;
                .WORD $1234
KBDQ              .BLOCK  40,0     ;CHARACTER QUEUE
;
                .WORD $5678
SLINE           .BYTE 38
LKBDQ           .BLOCK  40,0       ;Last command queue
;
         .END
```

Size requirement estimates for Lisa 1.75 Boot ROM

Talk	$0280	640
AppleBus boot	$0280	640
Graphics w/o menu	$1800	6144
Menu	$xxxx	xxxx
Debug mode	$1055	4181
other debug (floppy)	$0200	512
Other service	$xxxx	xxxx
Extended tests	$xxxx	xxxx
Main boot flow	$xxxx	xxxx
Alternate boot & Monitor	$0AA0	2720
Fatal error handler	$xxxx	xxxx
Non-fatal error handler	$xxxx	xxxx
Default exception handlers	$xxxx	xxxx
Diagnostics		
Checksum	$0040	64
Video memory test	$xxxx	xxxx
Parity, Video memory	$05AC	1452
Video circuitry	$0456	1110
Timer #0	$0C00	3072
Timer #1	$0400	1024
Timer #2	$0200	512
COPS	$0200	512
RS232A	$0400	1024
RS232B	$0180	384
Size memory	$0140	320
MMU read/write,address	$0512	1298
Main memory std.	$xxxx	xxxx
Parity, main mem	$0200	512
MMU functional	$085A	2138
IWM	$xxxx	xxxx
Builtin hard port	$0300	768
Expansion cards	$0200	512
TOTAL	$7363	29539

```
Program Test;
type
   string255 = string[255];

var
   DStr:String255;


Procedure ROMTalk;EXTERNAL;

Procedure DeskTop;EXTERNAL;
Procedure Paint_String(X1,Y1:integer;var DStr:String255);EXTERNAL;
Procedure AFolder(X1,Y1,X2,Y2:integer;  var DStr:String255);EXTERNAL;
Procedure AIcon_Draw(X1,Y1,Icon_Code:integer);EXTERNAL;


begin
DeskTop;

DStr:= 'ROM Talk';
AFolder(60,70,680,250,DStr);

DStr:= 'This LISA is executing the ROM Talk program.';
Paint_String(200,140,DStr);

AIcon_Draw(90,140,8);

ROMTalk;

end.
```