

```

{ ****
*
*                               V D A C P . P A S
*
**-----**
*   Task                       : Demonstrates programming of the DAC color
*                               register graphics mode of the VGA card, using
*                               256 colors. This program requires the V3240PA.
*                               ASM assembly language module.
*
**-----**
*   Author                     : MICHAEL TISCHER
*   Developed on                : 01/02/91
*   Last update                 : 03/03/92
*
*****}

```

program VDACP;

uses dos, crt;

```

{-- Type declarations -----}

```

```

type DACREG = record
    { Describes a DAC register }
    case integer of
        0 : ( Red, Green, Blue : BYTE );{RGB color components}
        1 : ( RGB : array[ 1..3] of BYTE );
    end;
    DACARRAY = array [0..255] of DACREG;    { Complete DAC table }

```

```

{-- External references to the assembler routines -----}

```

```

{$L v3240pa}                                { Link assembler module }

```

procedure init320400; external;

procedure setpix( x, y : integer; pcolor : byte ); external;

```

function getpix( x, y: integer ) : byte ; external;
procedure setpage( page : byte ); external;
procedure showpage( page : byte ); external;

{-- Constants -----}

const MAXX = 319;           { Maximum X- and Y-coordinates }
      MAXY = 399;

      BWIDTH  = 10;          { Width of a color block in pixels }
      BHEIGHT = 20;          { Height of a color block in pixels }
      SPACING = 2;           { Distance between the blocks }
      TOWIDTH = 16 * BWIDTH + ( 15 * SPACING );      { Total width }
      TOHEIGHT = 16 * BHEIGHT + ( 15 * SPACING );    { Total height }
      STARTX  = ( MAXX - TOWIDTH ) div 2;           { Upper left block corner }
      STARTY  = ( MAXY - TOHEIGHT ) div 2;

{*****}
*  IsVga : Determines whether a VGA card is installed.  *
**-----**
*  Input   : None                                         *
*  Output  : TRUE or FALSE                               *
*****}

function IsVga : boolean;

var Regs : Registers;      { Processor registers for interrupt call }

begin
  Regs.AX := $1a00;         { Function 1AH applies to VGA only }
  Intr( $10, Regs );
  IsVga := ( Regs.AL = $1a );

```

end;

```
{ *****
*   PrintChar : Writes a character to the screen while in graphic mode.*
**-----**
*   Input      :   THECHAR = Character to be written                      *
*                 X, Y      = X- and Y-coordinates of upper-left corner  *
*                 FG        = Foreground color                          *
*                 BK        = Background color                          *
*   Info       :   Character is created in an 8x8 matrix, based on the   *
*                 8x8 ROM font.                                          *
* ***** }
```

procedure PrintChar( thechar : char; x, y : integer; fg, bk : byte );

type FDEF = array[0..255,0..7] of byte;                                 { Font array }  
TPTR = ^FDEF;   { Pointer to font }

var Regs : Registers;   { Registers for interrupt call }  
ch : char;   { Individual pixels in character }  
i, k,   { Loop counter }  
BMask : byte;   { Bit mask for character design }

const fptr : TPTR = NIL;   { Pointer to font in ROM }

begin  
  if fptr = NIL then   { Pointer to font already set? }  
    begin   { NO }  
      Regs.AH := \$11;   { Call video BIOS function 11H, }  
      Regs.AL := \$30;   { sub-function 30H }  
      Regs.BH := 3;   { Get pointer to 8x8 font }  
      intr( \$10, Regs );

```

    fptr := ptr( Regs.ES, Regs.BP );           { Set pointers }
end;

if ( bk = 255 ) then                          { Drawing transparent characters? }
  for i := 0 to 7 do                          { Yes --> Set foreground pixels only }
    begin
      BMask := fptr^[ord(thechar),i];{ Get bit pattern for one line }
      for k := 0 to 7 do
        begin
          if ( BMask and 128 <> 0 ) then      { Pixel set? }
            setpix( x+k, y+i, fg );          { Yes }
          BMask := BMask shl 1;
        end;
      end;
    else                                     { No --> Consider background as well }
      for i := 0 to 7 do                     { Execute lines }
        begin
          BMask := fptr^[ord(thechar),i];{ Get bit pattern for one line }
          for k := 0 to 7 do
            begin
              if ( BMask and 128 <> 0 ) then    { Foreground? }
                setpix( x+k, y+i, fg );        { Yes }
              else
                setpix( x+k, y+i, bk );        { No --> Background }
              BMask := BMask shl 1;
            end;
          end;
        end;
      end;
    end;
  end;
end;

```

```

{ *****
*   Line: Draws a line based on the Bresenham algorithm.   *
**-----**

```

```

*   Input   : X1, Y1 = Starting coordinates (0 - ...)           *
*               X2, Y2 = Ending coordinates                     *
*               LPCOL  = Color of line pixels                   *
*****}

procedure Line( x1, y1, x2, y2 : integer; lpcol : byte );

var d, dx, dy,
    aincr, bincr,
    xincr, yincr,
    x, y                : integer;

{-- Procedure for swapping two integer variables -----}

procedure SwapInt( var i1, i2: integer );

var dummy : integer;

begin
    dummy := i2;
    i2     := i1;
    i1     := dummy;
end;

{-- Main procedure -----}

begin
    if ( abs(x2-x1) < abs(y2-y1) ) then          { X- or Y-axis overflow? }
        begin                                   { Check Y-axis }
            if ( y1 > y2 ) then                  { y1 > y2? }
                begin
                    SwapInt( x1, x2 );           { Yes --> Swap X1 with X2 }
                end
            end
        end
    end
end

```

```

        SwapInt( y1, y2 );
    end;

    if ( x2 > x1 ) then xincr := 1
        else xincr := -1;

    dy := y2 - y1;
    dx := abs( x2-x1 );
    d := 2 * dx - dy;
    aincr := 2 * (dx - dy);
    bincr := 2 * dx;
    x := x1;
    y := y1;

    setpix( x, y, lpcol );
    for y:=y1+1 to y2 do
        begin
            if ( d >= 0 ) then
                begin
                    inc( x, xincr );
                    inc( d, aincr );
                end
            else
                inc( d, bincr );
                setpix( x, y, lpcol );
            end;
        end
    else
        begin
            if ( x1 > x2 ) then
                SwapInt( x1, x2 );
        end
    end;

```

```

{          and Y1 with Y2  }

```

```

{ Set X-axis increment }

```

```

{ Set first pixel }
{ Execute line on Y-axes }

```

```

{ Check X-axes }

```

```

{ x1 > x2? }

```

```

{ Yes --> Swap X1 with X2 }

```

```

        SwapInt( y1, y2 );
    end;

    if ( y2 > y1 ) then yincr := 1
        else yincr := -1;

    dx := x2 - x1;
    dy := abs( y2-y1 );
    d := 2 * dy - dx;
    aincr := 2 * (dy - dx);
    bincr := 2 * dy;
    x := x1;
    y := y1;

    setpix( x, y, lpcol );
    for x:=x1+1 to x2 do
        begin
            if ( d >= 0 ) then
                begin
                    inc( y, yincr );
                    inc( d, aincr );
                end
            else
                inc( d, bincr );
                setpix( x, y, lpcol );
            end;
        end;
    end;

    { *****
    *   GrfxPrint: Displays a formatted string on the graphic screen.   *
    *-----**

```

```

*   Input   : X, Y   = Starting coordinates (0 - ...)           *
*               FG    = Foreground color                         *
*               BK    = Background color (255 = transparent)     *
*               STRING = String with format information           *
*****}

```

```

procedure GrfxPrint( x, y : integer; fg, bk : byte; strt : string );

var i : integer;                                { Loop counter }

begin
  for i:=1 to length( strt ) do
    begin
      printchar( strt[i], x, y, fg, bk );    { Display using PrintChar }
      inc( x, 8 );                          { Move X to next character position }
    end;
  end;
end;

```

```

{ *****
*   GetDac: Gets the contents of a specific number of DAC registers. *
**-----**
*   Input   : FIRST = Number of first DAC register (0-255)       *
*               NUM  = Number of DAC registers                    *
*               BUF  = Buffer, in which the contents of the DAC    *
*                   registers are to be loaded. It must be a      *
*                   DACREG type variable or an array of this type. *
*   Info    : The passed buffer must have three bytes reserved for *
*               DAC register, in which the red, green and blue parts *
*               of each color are recorded.                       *
*****}

```

```

procedure GetDac( First, Num : integer; var Buf );

```



```

var Regs : Registers;          { Processor registers for interrupt call }

begin
  Regs.AX := $1017;            { Function and sub-function number }
  Regs.BX := First;            { Number of first DAC register }
  Regs.CX := Num;              { Number of registers to be loaded }
  Regs.ES := seg( Buf );       { Load pointers to buffer }
  Regs.DX := ofs( Buf );
  intr( $10, Regs );          { Call BIOS video interrupt }
end;

{ *****
* SetDac: Loads a specific number of DAC registers *
**-----**
* Input   : FIRST = Number of first DAC register (0-255) *
*          NUM    = Number of DAC registers *
*          BUF    = Buffer, from which the contents of the DAC *
*                  registers are to be taken. Must be a variable *
*                  of DACREG type or an array of this type. *
* Info    : See GetDac *
*****}

procedure SetDac( First, Num : integer; var Buf );

var Regs : Registers;          { Processor registers for interrupt call }

begin
  Regs.AX := $1012;            { Function and sub-function number }
  Regs.BX := First;            { Number of first DAC register }
  Regs.CX := Num;              { Number of registers to be loaded }
  Regs.ES := seg( Buf );       { Load pointer to buffer }

```

```

    Regs.DX := ofs( Buf );
    intr( $10, Regs );
    { Call BIOS video interrupt }
end;

```

```

{ *****
*   PrintDac: Displays the contents of a DAC register on the screen   *
*   and sets the color in the DAC register 255.                       *
*   -----
*   Input   : DREG  = DAC register                                     *
*             NO    = The number of this register                     *
*             COLOR = Output color                                     *
***** }

```

```

procedure PrintDac( DReg : DACREG; No, Color : BYTE );

```

```

var nrstr,
    rstr,
    gstr,
    bstr : string[3];
    { String for register number }
    { String for red part }
    { String for green part }
    { String for blue part }

```

```

begin
    SetDac( 255, 1, DReg );
    str( No : 3, nrstr );
    str( DReg.Red : 2, rstr );
    str( DReg.Green : 2, gstr );
    str( DReg.Blue : 2, bstr );
    GrfxPrint( 60, MAXY-10, Color, 0, 'DAC:' + nrstr + ' R:' + rstr +
    ' G:' + gstr + ' B:' + bstr );

```

```

end;

```

```

{ *****
*   Frame   : Draws a frame around one of the color boxes           *
***** }

```

```

**-----**
*   Input   : X       = X-coordinate of color box (0-15)   *
*             Y       = Y-coordinate of color box (0-15)   *
*             COLOR = Color of border                       *
*   Info    : Line thickness is one pixel, regardless of distance *
*             of color boxes.                               *
*****}

procedure Surround( X, Y, Color : BYTE );

var sx, sy,                { Upper-left corner of frame }
    ex, ey : integer;      { Lower-right corner of frame }

begin
    sx := STARTX + X * (BWIDTH + SPACING) - 1;  { Compute corner      }
    ex := sx + BWIDTH + 1;                      { coordinates of frame }
    sy := STARTY + Y * (BHEIGHT + SPACING) - 1;
    ey := sy + BHEIGHT + 1;
    Line( sx, sy, ex, sy, Color );               { Draw frame }
    Line( ex, sy, ex, ey, Color );
    Line( ex, ey, sx, ey, Color );
    Line( sx, ey, sx, sy, Color );
end;

{ *****
*   ChangeDacReg: Changes the contents of a DAC register in memory *
*                  and in the DAC table of the video card and then *
*                  displays them on the screen.                     *
*-----**
*   Input   : DREG    = DAC register to be changed               *
*             NO       = Number of DAC register                 *
*             COMP     = Number of components to be changed (1-3) *

```

```

*           1 = Red, 2 = Green, 3 = Blue                               *
*           INCR = Increment for these components                       *
*****}

```

```

procedure ChangeDacReg( var DReg : DACREG; No, Comp : BYTE;
                      Incr : integer );

```

```

begin
  inc( DReg.RGB[ Comp ], Incr );           { Increment components }
  if DReg.RGB[ Comp ] > 63 then             { Greater than 63? }
    DReg.RGB[ Comp ] := 0;                 { Yes, set to zero }
    SetDac( No, 1, DReg );                 { Load DAC register }
    PrintDac( DReg, No, 15 );              { Output new contents }
end;

```

```

{*****}
* Demo: Demonstrates the programming of the DAC register and the      *
* color model of the VGA card.                                         *
**-----**
* Input : None                                                         *
*****}

```

```

procedure Demo;

```

```

var x, y,
    ix, jx,
    iy, jy,
    k, f : integer;           { Loop counter }
    ch : char;                 { Key }
    dacbuf : DACARRAY;        { Array with complete DAC table }
    DReg : DACREG;            { Current DAC register }

```

```

begin
  {-- Create screen -----}

  SetPage( 0 );                                { Process page 0 }
  ShowPage( 0 );                               { Display page 0 }
  GetDac( 0, 256, dacbuf );                     { Load complete DAC table }

  GrfxPrint( 10, 0, 255, 0,
    'VDACP - (c) 1992 by Michael Tischer' );

  {-- Create block out of 16x16 color boxes -----}

  iy := STARTY;                                { Starting point on the screen }
  jy := STARTY + BHEIGHT - 1;
  f := 0;
  for y := 0 to 15 do                          { Execute 16 block lines }
    begin
      ix := STARTX;
      jx := STARTX + BWIDTH - 1;
      for x := 0 to 15 do                      { Execute 16 block columns }
        begin
          for k := iy to jy do { Make block out of individual lines }
            Line( ix, k, jx, k, f );
            inc( ix, BWIDTH + SPACING );        { Next block right }
            inc( jx, BWIDTH + SPACING );
            inc( f );                          { Increment color for next block }
          end;
          inc( iy, BHEIGHT + SPACING );          { Output position for next }
          inc( jy, BHEIGHT + SPACING );          { block line }
        end;
      end;
    end;

  {-- Read user input and respond to it -----}

```

```

ix := 0;           { Begin in upper-left corner with the color 0 }
iy := 0;
jx := 0;
jy := 0;
k := 0;
GetDac( 0, 1, DReg );           { Get color 0 }
Surround( 0, 0, 15 );          { Frame color box }
PrintDac( DReg, 0, 15 );        { and output contents }
repeat
  ch := ReadKey;                { Read key }
  if ( ch <> #0 ) then          { Not an extended key? }
    case ch of
      'r' : ChangeDacReg( DReg, k, 1, +1 );      { No, evaluate }
      'g' : ChangeDacReg( DReg, k, 2, +1 );      { r = Red + }
      'b' : ChangeDacReg( DReg, k, 3, +1 );      { g = Green + }
      'R' : ChangeDacReg( DReg, k, 1, -1 );      { b = Blue + }
      'G' : ChangeDacReg( DReg, k, 2, -1 );      { R = Red - }
      'B' : ChangeDacReg( DReg, k, 3, -1 );      { G = Green - }
      ' ' : begin                               { B = Blue - }
        DReg := dacbuf[ k ];
        ChangeDacReg( DReg, k, 1, 0 );
        end;
        { Space = Set original value }
    end
  else
    case ReadKey of
      #72 : if ( iy = 0 ) then
        jy := 15
        else
          jy := iy - 1;
        #80 : if ( iy = 15 ) then

```

```

        jy := 0
    else
        jy := iy + 1;

#75 : if ( ix = 0 ) then                                { Cursor left }
    jx := 15
    else
        jx := ix - 1;

#77 : if ( ix = 15 ) then                                { Cursor right }
    jx := 0
    else
        jx := ix + 1;

end;

if ( ix <> jx ) or ( iy <> jy ) then                    { New cursor position? }
begin                                                    { Yes }
    Surround( ix, iy, 0 );                                { Fade out frame in old position }
    Surround( jx, jy, 15 );                                { Draw frame around new position }
    ix := jx;                                              { Store new color box }
    iy := jy;
    k := iy*16+ix;                                         { Compute number of new box }
    GetDac( k, 1, DReg );                                  { Load DAC register }
    PrintDac( DReg, k, 15 );                               { and output }
end;
until ch = #13;                                           { Repeat until RETURN is pressed }

SetDac( 0, 256, dacbuf );                                { Restore DAC table }
end;

{ ***** }
{ ***** M A I N P R O G R A M ***** }

```

```

{*****}

begin
  if IsVga then
    begin
      init320400;
      Demo;
      Textmode( C080 );
    end
  else
    writeln( 'VDACP - (c) 1992 by Michael Tischer'#13#10#10 +
      'This program requires a VGA card'#13#10 );
  end.

```