

Listing: DFP.PAS

```
{*****}
{*                D F P . P A S                *}
{*-----*}
{* Task          : Formats 3.5" and 5.25" diskettes *}
{*-----*}
{* Author        : Michael Tischer                *}
{* Developed on   : 08/23/91                        *}
{* Last update    : 04/07/95                        *}
{*-----*}
{*****}
```

program DFP;

Uses Dos; { Add Crt and Dos units }

{-- Constants -----}

```
const NO_DRIVE    = 0; { No drive }
      DD_525      = 1; { Drive: 5.25" DD }
      HD_525      = 2; { Drive: 5.25" HD }
      DD_35       = 3; { Drive: 3.5" DD }
      HD_35       = 4; { Drive: 3.5" HD }
      MaxNumTries = 5; { Maximum number of tries }
```

{-- Type declarations -----}

```
type DdptType = array[ 0..10 ] of byte; { Structure for DDPT }
      DdptPtr = ^DdptType; { Pointer to DDPT }
```

```
PhysDataType = record { Physical format parameters }
  DSides, { Desired number of sides for diskette }
  STrax, { Number of tracks per side }
```



```

{-- Actual load program -----}

$FA,          { 0037  CLI          }
$B8, $30, $00, { 0038  MOV    AX,0030 }
$8E, $D0,      { 003B  MOV    SS,AX }
$BC, $FC, $00, { 003D  MOV    SP,00FC }
$FB,          { 0040  STI          }
$0E,          { 0041  PUSH     CS }
$1F,          { 0042  POP      DS }
$BE, $66, $7C, { 0043  MOV    SI,7C66 }
$B4, $0E,      { 0046  MOV    AH,0E }
$FC,          { 0048  CLD          }
$AC,          { 0049  LODSB       }
$0A, $C0,      { 004A  OR      AL,AL }
$74, $04,      { 004C  JZ      0052 }
$CD, $10,      { 004E  INT     10 }
$EB, $F7,      { 0050  JMP     0049 }
$B4, $01,      { 0052  MOV    AH,01 }
$CD, $16,      { 0054  INT     16 }
$74, $06,      { 0056  JZ      005E }
$B4, $00,      { 0058  MOV    AH,00 }
$CD, $16,      { 005A  INT     16 }
$EB, $F4,      { 005C  JMP     0052 }
$B4, $00,      { 005E  MOV    AH,00 }
$CD, $16,      { 0060  INT     16 }
$33, $D2,      { 0062  XOR    DX,DX }
$CD, $19 );    { 0064  INT     19 }

```

```

BootMes : string =
  #13#10'DFP - (C) 1992 by Michael Tischer'+ #13#10 +
  #13#10'Defective diskette or non-system diskette'#13#10 +

```

```
'Please change diskettes and press any key . . .' +  
#13#10;
```

```
{-- Non-initialized global variables -----}
```

```
var CurDrive      : byte;      { Number of drive to be formatted 0, 1 }  
CurDriveType    : byte;      { Current disk drive type }  
PData           : PhysDataType; { Physical format information }  
LData           : LogDataType;  { Logical format information }  
POldDDPT        : pointer;     { Pointer to old DDPT }  
OK              : boolean;     { Flag for program flow }  
ExitCode        : word;       { Return value to calling process }  
Param           : string;     { for evaluation of command line }
```

```
{*****}  
* GetDriveType : Gets disk drive type. *  
* Input       : DRIVE = Drive number (0, 1 etc.) *  
* Output      : Drive code as constant (DD_525, HD_525 etc.) *  
{*****}
```

```
function GetDriveType( Drive : byte ) : byte;
```

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

```
begin
```

```
  Regs.ah := $08;          { Function: Determine drive type }  
  Regs.dl := Drive;        { Drive number }  
  intr( $13, Regs );       { Call BIOS interrupt }  
  if ( Regs.flags and fcarry = 0 ) then {Call completed without error?}  
    GetDriveType := Regs.bl      { Drive type }  
  else  
    GetDriveType := DD_525;      { Function 08H of interrupt does }
```

```

end;                                     { not exist => Computer type = XT }

{*****}
{ * ResetDisk      : Disk reset on all drives.                * }
{ * Input          : None                                       * }
{ * Output         : None                                       * }
{ * Info           : Regardless of drive number loaded in DL, reset * }
{ *               : executed on all drives.                    * }
{*****}

procedure DiskReset;

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

begin
  with Regs do
    begin
      ah := $00;           { Function number for interrupt call }
      dl := 0;             { Drive a: (see Info) }
    end;
    intr( $13, Regs );     { Interrupt call }
  end;

{*****}
{ * GetFormatParameter: Determines the logical and physical    * }
{ *                   : parameters necessary for formatting.    * }
{ * Input             : FORMSTRING = Desired capacity as string * }
{ *                   : "360", "1200", "720", "1440"           * }
{ *                   : DRIVETYPE = Drive code as returned from * }
{ *                   : GetDriveType                           * }
{ *                   : PDATA = Loaded by procedure with the    * }
{ *                   : specifications of the physical         * }
{*****}

```



```

PHYS_1200 : PhysDataType = ( DSides   : 2; STrax : 80;
                             TSectors : 15; DDPT  : @DDPT_1200);
PHYS_1440 : PhysDataType = ( DSides   : 2; STrax : 80;
                             TSectors : 18; DDPT  : @DDPT_1440);
PHYS_720  : PhysDataType = ( DSides   : 2; STrax : 80;
                             TSectors : 9;  DDPT  : @DDPT_720 );

```

```
begin
```

```

  if ( FormString = '1200' ) then           { 1.2 Meg on 5.25"? }
    if ( DriveType = HD_525 ) then { Format compatible with drive? }
      begin                                { Yes, set parameter }
        PData := PHYS_1200;
        LData := LOG_1200;
        GetFormatParameter := true;        { End without error }
      end
    else
      GetFormatParameter := false { Drive and format incompatible }
    else if ( FormString = '360' ) then      { 360K? }
      if ( DriveType = HD_525 ) or ( DriveType = DD_525 ) then
        begin                                { Format and drive compatible, set parameter }
          PData := PHYS_360;
          LData := LOG_360;
          GetFormatParameter := true;        { End without error }
        end
      else
        GetFormatParameter := false { Drive and format incompatible }
    else if ( FormString = '1440' ) then      { 1.44 Meg on 3.5"? }
      if ( DriveType = HD_35 ) then { Format compatible with drive? }
        begin                                { Yes, set parameters }
          PData := PHYS_1440;
          LData := LOG_1440;
          GetFormatParameter := true;        { End without error }
        end
      end

```

```

        end
    else
        GetFormatParameter := false { Drive and format incompatible }
    else if ( FormString = '720' ) then { 720K on 3.5"? }
        if ( DriveType = HD_35 ) or ( DriveType = DD_35 ) then
            begin { Format and drive compatible, set parameters }
                PData := PHYS_720;
                LData := LOG_720;
                GetFormatParameter := true; { End without error }
            end
        else
            GetFormatParameter := false { Drive and format incompatible }
        else
            GetFormatParameter := false; { Invalid format specified }
        end;
    end;

```

```

{*****}
{ * DiskPrepare : Prepare drive, set data transfer rate. * }
{ * Input : DRIVE = Drive number * }
{ * PDATA = Physical parameters * }
{ * Output : None * }
{*****}

```

```

procedure DiskPrepare( Drive : byte; PData : PhysDataType );

```

```

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

```

```

begin

```

```

    {-- Set media type for format call -----}

```

```

    with Regs do

```

```

        begin

```



```

    ah := $18;           { Function number for interrupt call }
    ch := PData.STrax - 1; { Number of tracks per side }
    cl := PData.TSectors; { Number of sectors per track }
    dl := Drive;          { Drive number }
end;
intr( $13, Regs );      { Interrupt call }
end;

```

```

{ ***** }
{ * FormatTrack : Formats a track. * }
{ * Input : See below * }
{ * Output : Error status * }
{ ***** }

```

```

function Formattrack( DriveNum, { The disk drive number }
                     SideNum,   { The side number }
                     TrackF,    { Track to be formatted }
                     SecPTR : byte ) : byte; { Sectors per track }

```

```

type FormatTyp = record
    DTrack, DSideNum, DCounter, DLength : byte;
end;

```

```

var Regs      : Registers; { Processor registers for interrupt call }
    DataField : array[ 1..18 ] of FormatTyp; { Maximum 18 sectors }
    Counter   : byte;      { Loop counter }
    Attempts  : byte;      { Maximum number of tries }

```

```

begin
    for Counter := 1 to SecPTR do
        with DataField[ Counter ] do
            begin

```

```

    DTrack := TrackF;                { Track number }
    DSideNum := SideNum;              { Diskette side }
    DCounter := Counter;              { Sector number }
    DLength := 2;                     { Number of bytes per sector (512) }
end;
Attempts := MaxNumTries;              { Set maximum number of tries }
repeat
    with Regs do
        begin
            ah := 5;                  { Function number for interrupt call }
            al := SecPtr;              { Number of sectors for one track }
            es := Seg( DataField );    { Address of data field }
            bx := Ofs( DataField );    { to register es:bx }
            dh := SideNum;              { Side number }
            dl := DriveNum;            { Drive number }
            ch := TrackF;              { Track number }
        end;
        intr( $13, Regs );            { Call BIOS interrupt }
        if ( Regs.flags and fcarry = 1 ) then { Error? }
            DiskReset;                { Yes --> Disk reset before next try }
        dec( Attempts );
    until ( Regs.flags and fcarry = 0 ) or ( Attempts = 0 );
    Formattack := Regs.ah;            { Read error status }
end;

{ ***** }
{ * VerifyTrack : Verify track * }
{ * Input : Drive, side, track, sector number * }
{ * Output : Error code (0=OK) * }
{ ***** }

```

```

function VerifyTrack( DriveNum, SideNum, TrackF, TSectors : byte ) : byte;

```

```

var Attempts      : byte;                                { Maximum number of tries }
Regs              : Registers; { Processor registers for interrupt call }
TrackBuffer       : TrackBfType;                          { Memory for a track }

begin
  Attempts := MaxNumTries;                                { Set maximum number of tries }
  repeat
    with Regs do
      begin
        ah := $04;                                         { Function number for interrupt call }
        al := TSectors;                                    { Number of sectors per track }
        ch := TrackF;                                       { Track number }
        cl := 1;                                           { Start at sector 1 }
        dl := DriveNum;                                    { Drive number }
        dh := SideNum;                                     { Side number }
        es := Seg( TrackBuffer );                          { Address for buffer }
        bx := Ofs( TrackBuffer );
      end;
      intr( $13, Regs );                                    { Call BIOS interrupt }
      if ( Regs.flags and fcarry = 1 ) then                 { Error? }
        DiskReset;                                         { Yes --> Disk reset before next try }
      dec( Attempts );
    until ( Regs.flags and fcarry = 0 ) or ( Attempts = 0 );
    VerifyTrack := Regs.ah;
  end;

  { ***** }
  { * WriteTrack      : Write track * }
  { * Input           : Drive, side, track, start sector, number, data * }
  { * Output          : Error code (0=OK) * }
  { ***** }

```

```

function WriteTrack(      DriveNum, SideNum, TrackF,
                        Start, SecPtr      : byte;
                        var Buffer ) : byte;

var Attempts : byte;           { Maximum number of tries }
    Regs      : Registers; { Processor registers for interrupt call }

begin
    Attempts := MaxNumTries;      { Set maximum number of tries }
    repeat
        with Regs do
            begin
                ah := $03;          { Function number for interrupt call }
                al := SecPtr;        { Number of sectors per track }
                ch := TrackF;        { Track number }
                cl := Start;         { Start at sector 1 }
                dl := DriveNum;      { Drive number }
                dh := SideNum;       { Side number }
                es := Seg( Buffer ); { Address for buffer }
                bx := Ofs( Buffer );
            end;
            intr( $13, Regs );      { Call BIOS interrupt }
            if ( Regs.flags and fcarry = 1 ) then { Error? }
                DiskReset;          { Yes --> Disk reset before next try }
            dec( Attempts );
        until ( Regs.flags and fcarry = 0 ) or ( Attempts = 0 );
        WriteTrack := Regs.ah;
    end;

{ ***** }
{ * PhysicalFormat: Physical formatting of the diskette (Division * }

```

```

{ *                into tracks, sectors).                * }
{ * Input          : DRIVE  = Drive code                  * }
{ *                PDATA   = Physical parameters          * }
{ *                VERIFY = TRUE, If verify is to be executed * }
{ * Output         : FALSE if error, otherwise TRUE        * }
{ ***** }

```

```

function PhysicalFormat( Drive  : byte;
                        PData   : PhysDataType;
                        Verify  : boolean ) : boolean;

```

```

var Attempts : byte;                { Maximum number of tries }
    Regs      : Registers; { Processor registers for interrupt call }
    TrackF,   { Current track }
    SideNum,  { Current side }
    Stat      : byte;                { Return value of called functions }

```

```

begin
    {-- Format a diskette track by track -----}

    for TrackF := 0 to PData.STrax - 1 do          { Execute all tracks }
        for SideNum := 0 to PData.DSides - 1 do    { Execute all sides }
            begin
                Write( #13'Track: ', TrackF: 2, ' Side: ', SideNum: 2 );
                {-- A maximum of 5 tries to format a track -----}

                Attempts := MaxNumTries;            { Set maximum number of tries }
                repeat
                    Stat := FormatTrack( Drive, SideNum, TrackF, PData.TSectors );
                    if ( Stat = 3 ) then             { Diskette write/protected? }
                        begin

```

```

PhysicalFormat := false; { End procedure with error }
WriteLn( #13'Diskette is write/protected' );
exit; { End procedure }
end;
if ( Stat = 0 ) and Verify then
  Stat := VerifyTrack( Drive, SideNum, TrackF, PData.TSectors );
dec( Attempts );
if ( Stat > 0 ) then { Format unsuccessful }
  DiskReset;
until ( Stat = 0 ) or ( Attempts = 0 );
if ( Stat > 0 ) then { Error during formatting }
  begin
    PhysicalFormat := false; { End procedure with error }
    WriteLn( #13'Track defective ' );
    exit; { End procedure }
  end;
end;
PhysicalFormat := true; { Procedure ended without error }
end;

{*****}
* LogicalFormat : Logical formatting of diskette (Writing boot *
* sector, FAT and root directory) *
* * *
* Input : DRIVE = Drive number *
* * *
* PDATA = Physical formatting information *
* * *
* LDATA = Logical formatting information *
* * *
* Output : TRUE, if no error occurs *
{*****}

function LogicalFormat( Drive : byte;
  PData : PhysDataType;
  LData : LogDataType ) : boolean;

```

```

var Stat          : byte;          { Feedback of called functions }
TotalNoSectors   : word;          { Total number of sectors }
i                : byte;          { Loop counter }
CurSector,
CurSide,
CurTrack        : byte;
SecPtr          : integer;        { Number of tracks to be written }
TrackBuffer      : TrackBfType;   { Memory for a track }

begin
  fillchar( TrackBuffer, word( PData.TSectors ) * 512, 0 );{ Empty buf}

  {-- Bootsector: Fixed part -----}

  move( BootMask, TrackBuffer, 102 );      { Copy boot sector mask }
  move( BootMes[1], TrackBuffer[ 1, 103 ], { Copy boot texts }
        ord(BootMes[0]) );
  TrackBuffer[ 1, 511 ] := $55;            { End marker of boot sector }
  TrackBuffer[ 1, 512 ] := $AA;

  {-- Bootsector: Variable part -----}

  TotalNoSectors := PData.STrax * PData.TSectors * PData.DSides;
  TrackBuffer[ 1, 14 ] := LData.Cluster;   { Cluster size }
  TrackBuffer[ 1, 18 ] := LData.RootSize; { Num. entries in root dir. }
  TrackBuffer[ 1, 20 ] := lo( TotalNoSectors );{ Total number sectors }
  TrackBuffer[ 1, 21 ] := hi( TotalNoSectors );{ on the diskette }
  TrackBuffer[ 1, 22 ] := LData.Media;     { Media descriptor }
  TrackBuffer[ 1, 23 ] := LData.FAT;       { Size of FAT }
  TrackBuffer[ 1, 25 ] := PData.TSectors;  { Sectors per track }
  TrackBuffer[ 1, 27 ] := PData.DSides;    { Number of sides }

```

```

{-- Make FAT and FAT copy (Contents 00)-----}

TrackBuffer[ 2, 1 ] := LData.Media;           { Create 1st FAT }
TrackBuffer[ 2, 2 ] := $FF;
TrackBuffer[ 2, 3 ] := $FF;
TrackBuffer[ LData.FAT + 2, 1 ] := LData.Media; { Create 2nd FAT }
TrackBuffer[ LData.FAT + 2, 2 ] := $FF;
TrackBuffer[ LData.FAT + 2, 3 ] := $FF;

{-- Write boot sector and FAT -----}

Stat := WriteTrack( Drive, 0, 0, 1, PData.TSectors, TrackBuffer );
if Stat <> 0 then
    LogicalFormat := FALSE

{-- No error, write root directory -----}

else
    begin
        fillchar( TrackBuffer, 512, 0 );           { Empty sector }
        CurSector := PData.TSectors; { Write first track completely }
        CurTrack := 0;                             { Current track }
        CurSide := 0;                               { Current side }

        {-- Determine number of remaining sectors and write -----}

        SecPtr := LData.FAT * 2 + ( LData.Rootsize * 32 div 512 ) +
            1 - PData.TSectors;

        i := 1;
        repeat

```



```

inc( CurSector );                                { Next sector }
if ( CurSector > PData.TSectors ) then            { Track ended? }
begin
    CurSector := 1;                               { Continue with sector 1 }
    inc( CurSide );                               { Next side? }
    if ( CurSide = PData.DSides ) then            { Side 2 already? }
    begin
        CurSide := 0;                             { Back to side 0 }
        inc( CurTrack );
    end;
end;
Stat := WriteTrack( Drive, CurSide, CurTrack,
                    CurSector, 1, TrackBuffer );

inc( i );
until ( i > SecPTr ) or ( Stat <> 0 );
LogicalFormat := ( Stat = 0 )
end;
end;

{*****}
{ *                MAIN PROGRAM                * }
{*****}

begin
    WriteLn( 'DFP - (c) 1992 by Michael Tischer'#13#10 );
    if paramcount > 1 then                        { Parameters specified? }
    begin { Yes, evaluate }
        Param := paramstr( 1 );                  { Determine drive ( 0 = a:, 1 = b: ) }
        CurDrive := ord( upcase( Param[ 1 ] ) ) - 65;
        CurDriveType := GetDriveType( CurDrive ); { Type of current driv }
        if ( CurDriveType > 0 ) then                { Drive available? }
        begin
            { Yes --> Program can be continued }
        end
    end
end

```

```

if GetFormatParameter( paramstr( 2 ), CurDriveType,
                      PData, LData) then
begin
    { Format and drive are compatible }
    DiskPrepare( CurDrive, PData );
    GetIntVec( $1E, POldDDPT );          { Store old DDPT }
    SetIntVec( $1E, PData.DDPT );        { Set new DDPT }

    Param := paramstr( 3 );
    ok := PhysicalFormat( CurDrive, PData,
                        upcase( Param[ 1 ] ) <> 'N' );

    if ok then
    begin
        Write( #13'Write boot sector and FAT          ' );
        ok := LogicalFormat( CurDrive, PData, LData )
        end;

    {-- Evaluation of formatting process -----}

    if ok then
    begin
        WriteLn( #13'Formatting OK                      );
        ExitCode := 0;
    end
    else
    begin
        WriteLn( #13'Error - format cancelled' );
        ExitCode := 1;
    end;
    SetIntVec( $1E, POldDDPT );          { Restore old DDPT }
end
else
begin

```

```

        WriteLn( 'This drive does not support that format' );
        ExitCode := 2;      { Return value to calling process }
    end
end
else
    begin
        WriteLn( 'The specified disk drive does not exist' );
        ExitCode := 3;      { Return value to calling process }
    end
end
else
    begin
        writeln( 'Syntax: DFP Drive Format [ NV ]' );
        writeln( '          _____' );
        writeln( '          A: or B:          ' );
        writeln( '          _____' );
        writeln( '          360, 720, 1200, 1440 ' );
        writeln( '          _____' );
        writeln( '          NV = No verify' );
        ExitCode := 4;      { Return value to calling process }
    end;
    Halt( ExitCode );
end.

```