

Pascal listing: EXTP.PAS

```
{*****
*
*                               E X T P . P A S
*
**-----**
*  Demmonstration of accessing the Extended-Memory using the BIOS-
*  Functions of Interrupts 15h taking into consideration any RAN disks.*
**-----**
*  Author      : MICHAEL TISCHER
*  Developed on : 05/18/1889
*  last update on : 04/07/1995
*****}
```

program ExTP;

uses Dos;

{-- globale variables-----}

```
var RdLen      : integer;           { Size of the RAM disk in KB }
    ExtAvail   : boolean;           { Extended Memory available? }
    ExtStart   : longint;           { Start address of EXT-Memory as linear Adr. }
    ExtLen     : integer;           { size of extended Memory in KByte }
```

```
{*****
*  ExtAdrConv : convert a pointer into a 32-Bit large linear addresss
*              in the form of a LONGINTS return value
*
**-----**
*  Input      : Adr = of the convertedpointer
*  Output     : the converted address
*****}
```

function ExtAdrConv (Adr : pointer) : longint;

```

type PTRREC = record
    Ofs : word;           { for accessing the }
    Seg : word;           { contents of any }
                           { pointer you wish }
end;

begin
    ExtAdrConv := longint( PTRREC( Adr ).seg ) shl 4 + PTRREC( Adr ).ofs;
end;

{*****}
*  ExtCopy : copy data between any buffers within the      *
*            16-MB address range of the 80286/386/486.      *
*-----*
*  Input   : Start   = address of start buffers as 32-Bit linear Adr. *
*            Target  = address of target buffer as 32-Bit linear Adr. *
*            Len     = Number of bytes to be copied          *
*  Info    : - The number of bytes to be copied must be an even number. *
*            - This procedure is only intended for use      *
*            within this unit                                *
{*****}

procedure ExtCopy( Start, Target : longint; Len : word );

{-- Data structure for accessing the extended RAM -----}

type SDES = record
    Length : word;           { Segment descriptor }
    AdrLo  : word;           { Length of segment in bytes }
    AdrHi  : byte;           { Bit 0 to 15 of Segment adr. }
    Attribut : byte;         { Bit 16 to 23 of Segment adr. }
    Res    : word;           { Segment attribute }
                           { reserved for 80386 }

```

```

        end;

GDT = record                                { Global Descriptor Table }
    Dummy : SDES;
    GDTS : SDES;
    Start : SDES;                            { copy from ... }
    Target : SDES;                           { ... to }
    Code : SDES;
    Stack : SDES;
end;

LI = record                                { This accesses the content of the }
    LoWord : word;                           { LongInts of the linear 32-Bit- }
    HiByte : byte;                           { addresses }
    dummy : byte;
end;

var GTab : GDT;                             { Global Descriptor Table }
    Regs : Registers;                       { Processor regs. for interrupt call }
    Adr : longint;                          { for conversion of the address }

begin
    FillChar( GTab, SizeOf( GTab ), 0 );    { all fields to 0 }

    {-- Create segment descriptor of start segments -----}

    GTab.Start.AdrLo := LI( Start ).LoWord;
    GTab.Start.AdrHi := LI( Start ).HiByte;
    GTab.Start.Attribut := $92;
    GTab.Start.Length := Len;

    {-- Create segment descriptor of target segments -----}

```

```

GTab.Target.AdrLo      := LI( Target ).LoWord;
GTab.Target.AdrHi      := LI( Target ).HiByte;
GTab.Target.Attribut   := $92;
GTab.Target.Length     := Len;

{-- Copy memory range with help from function $87 the cassette- ----}
{-- interrupts $15 -----}

Regs.AH := $87;                { Function number for 'Memory copying' }
Regs.ES := seg( GTab );        { address of GDT }
Regs.SI := ofs( GTab );        { to ES:SI }
Regs.CX := Len shr 1;          { Number of copied words to CX }
intr( $15, Regs );             { call function }
if ( Regs.AH <> 0 ) then        { Error? }
  begin                        { Yes AH contains erro rcode }
    writeln('Error accessing extended RAM (', Regs.AH,')!');
    RunError;                  { Abort program with Run-Time-Error }
  end;
end;

{
*****
*   ExtRead : read the specified number of bytes fro extended      *
*               memory into main memory.                            *
*-----*
*   Input   :  ExtAdr = Source address in extended-RAM (linear address) *
*               BuPtr  = Pointer to the Target buffer in main memory   *
*               Len    = Number of bytes to bve copied                 *
*****}

procedure ExtRead( ExtAdr : longint;  BuPtr : pointer;  Len : word );

```

```

begin
  ExtCopy( ExtAdr, ExtAdrConv( BuPtr ), len );
end;

{ *****
*   ExtWrite : write the specified number of bytes from main      *
*               memory into extended memory.                      *
*   -----*
*   Input      : BuPtr = Pointer to source buffer in main memory  *
*               ExtAdr = Target address in extended RAM (linear address) *
*               Len   = Number of bytes to be copied              *
*   *****}

procedure ExtWrite( BuPtr : pointer; ExtAdr : longint; Len : word );

begin
  ExtCopy( ExtAdrConv( BuPtr ), ExtAdr, len );
end;

{ *****
*   ExtGetInfo : Determine the start address of the extended RAM and *
*               its size considering any RAM disks that may be present *
*   -----*
*   Input      : none                                             *
*   Ausgabe    : none                                             *
*   Globals    : ExtAvail/W, ExtStart/W, ExtLen/W                *
*   *****}

procedure ExtGetInfo;

type NAME_TYP    = array [1..5] of char;
type BOOT_SECTOR = record           { Boot-Sector of RAM disk }

```

```

        dummy1      : array [1..3] of byte;
                      Name      : NAME_TYP;
        dummy2      : array [1..3] of byte;
        BpS         : word;
        dummy3      : array [1..6] of byte;
        Sectors     : word;
        dummy4      : byte;           { fill to even length }
    end;

const VdiskName : NAME_TYP = 'VDISK';

var BootSec : BOOT_SECTOR;           { takes the supposed Boot Sector }
    Lastlp  : boolean;               { mark loop end }
    Regs    : Registers;             { Processor regs. for interrupt call }

begin
    {-- Determine size of extended memory and whether is is available ---}

    Regs.ah := $88;                  { Function nr.: "Determine size of Extended-RAM" }
    intr( $15, Regs );               { call cassette interrupt }
    if ( Regs.AX = 0 ) then
        begin
            ExtAvail := FALSE;       { no extended RAM available }
            ExtLen    := 0;
            ExtStart  := 0;
            exit;                   { return to caller }
        end;

    ExtAvail := TRUE;                { extended Memory available }
    ExtLen    := Regs.AX;             { extended RAM existing, mark size }

    {-- search for RAM disks of Typ VDISK -----}

```

```

ExtStart := $100000;                                { start at 1 MB }
Lastlp := FALSE;                                     { start for RAM-Disk }
repeat                                                { loop start }
  ExtRead( ExtStart, @BootSec, SizeOf( BootSec ) );
  with BootSec do
    if Name = VDiskName then                        { is boot sector a RAM disk? }
      inc( ExtStart, longint( Sectors ) * BpS ) {Yes, beyond RAM disk}
    else
      Lastlp := TRUE;                               { no RAM disk is found }
until Lastlp;

{-- Calculate size of RAMdisk from free extended RAM -----}

dec( ExtLen, integer( (ExtStart - longint($100000)) shr 10 ) );
end;

{*****
* CheckExt : Examine the consistency of the free extended RAM *
*****}

procedure CheckExt;

var AdrTest      : longint;                          { Address of test blocks }
    i, j         : integer;                          { loop counter }
    WriteBuf,    : array [1..1024] of byte;          { Test block }
    ReadBuf      : array [1..1024] of byte;
    Error        : boolean;                          { Pointer to memory error }

begin
  Randomize;                                         { initialize random number generator }
  AdrTest := ExtStart;

```

```

for i := 1 to ExtLen do { check memory in 1 KB-Blocks }
begin
  for j := 1 to 1024 do { Fill block with random value }
    WriteBuf[ j ] := Random( 255 );

  write(#13, AdrTest ); { output address of the checked KB-blocks. }

  {-- Write the buffer and then read the buffer -----}

  ExtWrite( @WriteBuf, AdrTest, 1024 );
  ExtRead( AdrTest, @ReadBuf, 1024 );

  {-- Determine identity of WriteBuf and ReadBuf -----}

  for j := 1 to 1024 do
    if WriteBuf[j] <> ReadBuf[j] then { Buffer content identical? }
      begin { No, Error! }
        writeln( ' Error! Memory part ',
                  AdrTest + longint(j-1) );
        Ferror := TRUE;
      end;

      inc( AdrTest, longint( 1024 ) ); { AdrTest of next KB-Block }
    end; { set }

  writeln;
  if not( Ferror ) then { did an error occur? }
    writeln( 'All o.k.!' ); { No }
end;

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

```



```

begin
  writeln( #13#10'EXTDEMO - (c) 1989 by Michael Tischer'#13#10);
  ExtGetInfo;      { Determine availability and size of extended memory }
  if ExtAvail then { is extended RAM available? }
  begin           { Yes }
    RdLen := integer( (ExtStart - longint( $100000 ) ) shr 10 );
    if ( RdLen = 0 ) then { are RAM disks installed? }
    begin             { no }
      writeln( 'There are no RAM disks installed. ');
      writeln( 'The free extended RAM begins with the ',
        '1 MB memory boundary. ');
    end
  else               { Yse RAM disks present }
  begin
    writeln( 'One or more RAM disks have reserved ', RdLen,
      ' KB of extended RAM. ');
    writeln( 'The free extended RAM begins ', RdLen,
      ' KB after ther 1 MB n\memory boundry. ');
    end;
    writeln( 'The size of the free extended RAM is ',
      ExtLen, ' KB. ');
    writeln( #13#10'The extended RAM has also been checked for',
      ' consistency...'#13#10);
    CheckExt;
  end
else
  writeln( 'There is no extended RAM installed in this computer!');
end.

```