

```

{
***** E X T P . P A S *****
**
* Demonstration of accessing the Extended-Memory using the BIOS-
* Functions of Interrupts 15h taking into consideration any RAM disks.*
**
* Author      : MICHAEL TISCHER
* Developed on : 05/18/1889
* last update on : 02/19/1992
*****}

program ExtP;

uses Dos;

{-- global 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 }

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

function ExtAddrConv ( Adr : pointer ) : longint;

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

begin
    ExtAddrConv := 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                  { Segment descriptor }
    Length : word;                  { Length of segment in bytes }
    AdrLo   : word;                  { Bit 0 to 15 of Segment adr. }
    AdrHi   : byte;                  { Bit 16 to 23 of Segment adr. }
    Attribut : byte;                 { Segment attribute }
    Res      : word;                 { 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 contents 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;

{-- Creat 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 error code }
    writeln('Error accessing extended RAM (', Regs.AH, ')!');
    RunError;                   { Abort program with Run-Time-Error }
end;

end;

{*****}
* ExtRead : read the specified number of bytes from 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 be 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 *
* Output     : 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 }
    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; { Get applicable boot sector }
    Lastlp  : boolean; { mark loop end }
    Regs    : Registers; { Processor regs. for interrupt call }

```

```

begin
  {-- Determine size of extended memory and whether it 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;
      ExtLen   := 0;
      ExtStart := 0;
      exit;
    end;
    { return to caller }

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

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

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

  {-- Calculate size of RAM disk 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;
    i, j      : integer;
    WriteBuf, ReadBuf : array [1..1024] of byte;
    Error     : boolean;

begin
  Randomize;
  AdrTest := ExtStart;
  for i := 1 to ExtLen do
    begin
      for j := 1 to 1024 do
        WriteBuf[ j ] := Random( 255 );
      write(#13, AdrTest );
    end;
    { Address of the test blocks }
    { loop counter }
    { Test block }
    { Pointer to memory error }

    {-- 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
        begin
          writeln( ' Error! Memory part ',
                  AdrTest + longint(j-1) );
          Error := TRUE;
        end;
        { Buffer contents identical? }
        { No, Error! }

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

    writeln;
    if not( Error ) then
      writeln( 'All o.k.!' );
    else
      writeln( 'Error! Memory part ',
              AdrTest + longint(j-1) );
    end;
    { did an error occur? }
    { 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
begin
    RdLen := integer( (ExtStart - longint( $100000 ) ) shr 10 );
    if ( RdLen = 0 ) then
    begin
        writeln( 'There are no RAM disks installed. ');
        writeln( 'The free extended RAM begins with the ',
            '1 MB memory boundary. ');
    end
    else
    begin
        { Yes RAM disks present }
        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.

```