27 Systemnahe Programmierung in Pascal

27.1 Definition 2
27.2 Operationen mit Bitmustern. Beispiel Zahlenkonvertierung dez <-> bin 2
27.3 Speicheradressierung. Segment und Offset 9
27.4 Zugriff auf Speicheradressen in Turbo-Pascal 11
27.5 Inline-Code und Assembler-Code in Turbo-Pascal 12
27.6 Die Turbo-Pascal-Unit DOS 14
27.7 Die Interrupts im Überblick 16
27.8 Beispiel: Zeichenausgabe über Pascal, DOS, BIOS und Hardware 32

27.9 Diverse Demo-Programme A
27.9.1 Tastatur-Statusbytes, Umschalttasten (BIOS-Interrupt 16h) 36
27.9.2 BIOS-Interrupt 11h, Konfiguration feststellen 39
27.9.3 Maus-Interrupt 33h 41
27.9.4 BIOS-Interrupt 10h, Bildschirm, Cursorposition, Zeichen und Attribut 45
27.9.5 Absolute Speicheradressierung, Bildschirm in Datei speichern 47
27.9.6 BIOS-Interrupt 13h, Diskette, Platte 49

27.10 Diverse Demo-Programme B
27.10.1 BIOS-Interrupt 12h, Speicherkapazität abfragen 52
27.10.2 BIOS-Interrupt 10h, Cursor 53
27.10.3 BIOS-Interrupt 17h, Druckerstatus 54
27.10.4 ROM-Basic-Interrupt 18h, ROM-Basic (nicht bei allen PCs) 55
27.10.5 Speicherauszug (Hex-Dump) 56

Durch die Entwicklung des Betriebssystems MS-Windows sind einige Abschnitte dieses Kapitels nicht mehr so von Bedeutung wie früher. Für das Verständnis eines Betriebssystems sind sie dennoch hilfreich. In den Lehrveranstaltungen wird nur eine Auswahl aus diesem Kapitel behandelt.

27.1 Definition

Unter systemnaher Programmierung in einer höheren Programmiersprache versteht man das Programmieren von Anweisungen oder Funktionen, in denen unmittelbar auf Speicherstellen (Adressen) des Rechners oder auf Register des Prozessors zugegriffen wird, den Aufruf von Interrupts, z.B. für die Mausprogrammierung und auch die Definition bzw. Aufruf von (kleineren) Programmen in der Maschinensprache mit den Mitteln der höheren Programmiersprache.

Systemnahe Programmierung ist z.B. bei zeitkritischen Programmteilen angebracht. Die Übertragbarkeit von Pascal-Programmen mit systemnahen Programmteilen auf andere Rechner kann problematisch werden.

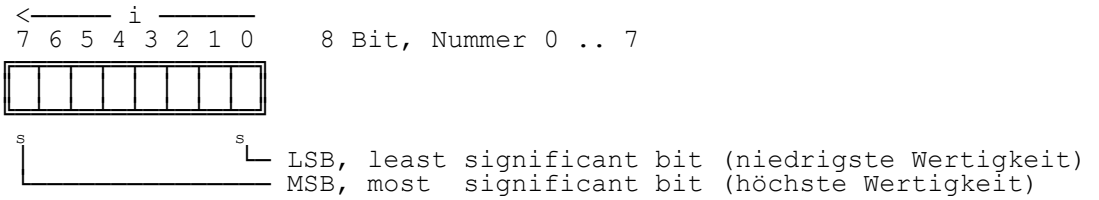
Turbo-Pascal gestattet den direkten Zugriff auf Speicherstellen; außerdem können Prozessorbefehle mit **`inline`** direkt in den Quelltext eingegeben werden, ab Turbo-Pascal 6.0 mittels **`asm`** auch in den mnemotechnischen Bezeichnungen der Assemblersprache.

27.2 Operationen mit Bit-Mustern

Bei systemnaher Programmierung sind oft Operationen mit Bit-Mustern notwendig. Unter Bit-Muster versteht man eine aus den Zeichen '0' und '1' bestehende Zeichenfolge, entsprechend dem binären Zahlensystem. Für die Operationen werden die logischen Operatoren **`and`**, **`or`**, **`not`** und **`xor`**, sowie die Schiebeoperatoren **`shl`** (shift left) und **`shr`** (shift right) verwendet. Diese Operatoren stehen mit gleichen Bezeichnungen sowohl in Pascal als auch im Assembler zur Verfügung.

Die folgenden Beispiele beziehen sich auf ein Bit-Muster mit der Länge 8, entsprechend einem Byte. Die Zählung der Bits beginnt zweckmäßigerweise rechts mit 0.

Für die späteren Demonstration der Bit-Muster-Operationen wird ein beliebiges Bit-Muster namens **`VOR`** angenommen. Durch die Operationen wird das Bit-Muster namens **`NACH`** erzeugt.



27.2.1 Zu den Shift-Operatoren shl und shr

Bei den meisten der späteren Bit-Operationen wird zur Veranschaulichung die Potenzschreibweise 2^i benutzt. Bekanntlich gibt es in Pascal die Potenzfunktion nicht als Standardfunktion. Die Nachstellung mit Exponential- und Logarithmusfunktion wäre

für die anstehenden Aufgaben zu aufwendig. Wesentlich eleganter und effizienter läßt sich das Problem mit dem Shift-Left-Operator **shl** oder dem Shift-Right-Operator **shr** lösen. Die folgende Tabelle zeigt Anwendungen des Shift-Left- und des Shift-Right-Operators.

Es gilt:

```
2^0  = 1 shl 0  =   1          2^7  = 128 shr 0 = 128
2^1  = 1 shl 1  =   2          2^6  = 128 shr 1 =  64
2^2  = 1 shl 2  =   4          2^5  = 128 shr 2 =  32
2^3  = 1 shl 3  =   8          2^4  = 128 shr 3 =  16
2^4  = 1 shl 4  =  16          2^3  = 128 shr 4 =   8
2^5  = 1 shl 5  =  32          2^2  = 128 shr 5 =   4
2^6  = 1 shl 6  =  64          2^1  = 128 shr 6 =   2
2^7  = 1 shl 7  = 128          2^0  = 128 shr 7 =   1
-------------------            --------------------
2^8  = 1 shl 8  = 256 (0)      2^-1 = 128 shr 8 =    0
2^9  = 1 shl 9  = 512 (0)      2^-2 = 128 shr 9 =    0
-------------------            --------------------
usw.                           usw.
```

```
Bytetyp, i = 0..7:    2^i = 1 shl i =   128 shr ( 7 - i)
Wordtyp, i = 0..15:   2^i = 1 shl i = 32768 shr (15 - i)
```

Bei Benutzung von Variablen für die Ergebnisse bei Shift-Operationen ist auf den Definitionsbereich des Datentyps zu achten, z.B. 0..255 bei Typ *Byte*. Bei Konstanten steht in Turbo-Pascal der Bereich des Typs *LongInt* zur Verfügung. Beim "Hinausschieben" nach rechts wird als Ergebnis 0 geliefert, da Schiebeoperatoren nur für Ganzzahlen definiert sind.

Das folgende Demo-Programm zeigt die Wirkung der Schiebeoperatoren und mögliche Fehler:

```
program Pas27021;  { Die Schiebe-Operatoren shl und shr }
uses
  CRT;
var
  i:          ShortInt;
  ByteLlinks,
  ByteRechts: Byte;
  WordLinks,
  WordRechts: Word;
begin
  ClrScr;
  WriteLn('   Datentyp Byte. Shift richtig für i = 0..7');
  WriteLn('   2 hoch  i = [1 shl i] = [128 shr (7 - i)]');
  WriteLn('   ----------------------------------------');
  for i := -2 to 9 do  { Richtig nur für i = 0..7, Byte }
    begin
      ByteLinks  := (1   shl i      );
      ByteRechts := (128 shr (7 - i));
      WriteLn('   2 hoch ', i:2, ' = ', ByteLinks:6, ByteRechts:14);
    end;
```

```
  WriteLn; Write(  '   Weiter mit Return ... '); ReadLn; WriteLn;

  WriteLn('   Datentyp Word. Shift richtig für i = 0..15  ');
  WriteLn('   2 hoch  i = [1 shl i] = [32768 shr (15 - i)]');
  WriteLn('   ----------------------------------------');
  for i := -2 to 17 do   { Richtig nur für i = 0..15, Word }
    begin
      WordLinks  := (1     shl i       );
      WordRechts := (32768 shr (15 - i));
      WriteLn('   2 hoch ', i:2, ' = ', WordLinks:6, WordRechts:16);
    end;
  repeat
  until ReadKey <> '';
end.
```

Die Ausgabe für den ersten Programmteil:

```
   Datentyp Byte. Shift richtig für i = 0..7
   2 hoch  i = [1 shl i] = [128 shr (7 - i)]
   ----------------------------------------
   2 hoch -2 =      0             0
   2 hoch -2 =      0             0
   2 hoch  0 =      1             1
   2 hoch  1 =      2             2
   2 hoch  2 =      4             4
   2 hoch  3 =      8             8
   2 hoch  4 =     16            16
   2 hoch  5 =     32            32
   2 hoch  6 =     64            64
   2 hoch  7 =    128           128
   2 hoch  8 =      0             0
   2 hoch  9 =      0             0
```

27.2.2 Bit-Muster um i-Stellen nach links verschieben

```
NACH := VOR  shl  i
```

i = 0 ... 7. Von rechts
shl: shift left

Das Bit-Muster wird rechts mit Nullen aufgefüllt. Die Verschiebung um eine Stelle nach links verdoppelt den Wert des Bit-Musters. Bits, die "hinausgeshiftet" werden, gehen verloren. In Turbo-Pascal jedoch Abbruch mit Fehlermeldung, wenn die Obergrenze des vereinbarten Datentyps, z.B. bei Byte = 255 überschritten wird.

1. Beispiel: *i* = 2, VOR = 0001 1111, dez 31, hex H1F
NACH = 0111 1100, dez 124, hex H7C

31 **shl** 2 ==> 124

2. Beispiel: *i* = 1, VOR = 1000 0000, dez 128, hex H80
NACH = 0000 0000, dez 0, hex H00

```
128 shl 1  ==>  0  (Zu Pascal siehe oben)
```

27.2.3 Bit-Muster um i-Stellen nach rechts verschieben

```
NACH := VOR  shr  i
```

i = 0 ... 7. Von rechts.
shr: shift right

Das Bit-Muster wird links mit Nullen aufgefüllt. Die Verschiebung um eine Stelle nach rechts halbiert den Wert des Bit-Musters. Beim "Hinausschieben" aller Bits wird das Ergebnis 0 geliefert.

1. Beispiel:

```
i = 2,  VOR  = 1001 0000,  dez 144, hex H90
        NACH = 0010 0100,  dez  36, hex H48
```

```
144 shr 2  ==>  36
```

2. Beispiel:

```
i = 1,  VOR  = 0000 0001,  dez   1, hex H01
        NACH = 0000 0000,  dez   0, hex H00
```

```
1 shr 1  ==>  0
```

27.2.4 Das i-te Bit setzen, die anderen Bits nicht verändern

```
NACH := VOR  or  2^i
```

i = 0 ... 7. Von rechts.

1. Beispiel:

```
i = 5,  VOR = 0101 1111,  dez  95, hex H5F
        2^5 = 0010 0000,  dez  32, hex H20
```

```
NACH = VOR or 2^5  = 0111 1111,  dez 127, hex H7F
95 or 32  ==>  127
```

2. Beispiel:

```
i = 5,  VOR = 0111 1111,  dez 127, hex H7F
        2^5 = 0010 0000,  dez  32, hex H20
```

```
NACH = VOR or 2^5  = 0111 1111,  dez 127, hex H7F
127 or 32  ==> 127  (Achtung: 127 or 34  ==> 127)
```

27.2.5 Das i-te Bit löschen, die anderen Bits nicht verändern

```
NACH := VOR  and  (not  2^i)
```

i = 0 ... 7. Von rechts
Klammern nicht notwendig

1. Beispiel:

```
i = 3,   2^3  = 0000 1000,  dez   8, hex H08
      not 2^3  = 1111 0111,  dez 247, hex HF7

         VOR  = 0101 1101,  dez  93, hex H5D
      not 2^3  = 1111 0111
```

```
NACH = VOR and (not 2^3)  = 0101 0101,  dez  85, hex H55
93 and (not 8) ==>  85

2. Beispiel:   i = 3,   2^3  = 0000 1000,  dez   8, hex H08
                      not 2^3  = 1111 0111,  dez 247, hex HF7

                         VOR  = 0101 0101,  dez  85, hex H55
                      not 2^3  = 1111 0111
NACH = VOR and (not 2^3)  = 0101 0101,  dez  85, hex H55
85 and (not 8)  ==>  85
```

27.2.6 Das i-te Bit invertieren, die anderen Bits nicht verändern

```
NACH := VOR xor 2^i
```

i = 0 ... 7. Von rechts
xor: Exklusives Oder

```
1. Beispiel:   i = 6,  VOR = 0001 1111,  dez  31, hex H1F
                       2^6 = 0100 0000,  dez  64, hex H40
NACH = VOR xor 2^6  = 0101 1111,  dez  95, hex H5F
31 xor 64  ==>  95

2. Beispiel:   i = 6,  VOR = 0101 1111,  dez  95, hex H5F
                       2^6 = 0100 0000,  dez  64, hex H40
NACH = VOR xor 2^6  = 0001 1111,  dez  31, hex H1F
95 xor 64  ==>  31
```

27.2.7 Testen, ob das i-te Bit gesetzt ist. Ergebnistyp Boolean

```
BitIgesetzt := ( (VOR and 2^i ) = 2^i )
```

i = 0 ... 7. Von rechts
Das innere Klammerpaar ist notwendig!

```
1. Beispiel:   i = 6,  VOR = 0101 1111,  dez  95, hex H5F
                       2^6 = 0100 0000,  dez  64, hex H40
(VOR and 2^6 ) = 0100 0000,  dez  64, hex H40
(VOR and 2^6 ) = 2^6  ==>   Bit6gesetzt = True

2. Beispiel:   i = 6,  VOR = 0001 1111,  dez  31, hex H1F
                       2^6 = 0100 0000,  dez  64, hex H40
```

```
(VOR and 2^6 )  = 0000 0000,  dez   0, hex H00
(VOR and 2^6 ) <> 2^6  ==>  Bit6gesetzt = False
```

27.2.8 Bit-Muster löschen (alle Bits auf 0)

```
NACH := VOR  xor  VOR
```

xor: Exklusives Oder
Alternative zu: NACH := 0

Beispiel:

```
                    VOR = 0001 1111,  dez  31, hex H1F
                    VOR = 0001 1111,  dez  31, hex H1F
          ------------------------------------------------
NACH = VOR xor VOR = 0000 0000,  dez   0, hex H00
31 xor 31  ==>  0
```

27.2.9 Anwendung: Zahlenkonvertierung

a) Konvertierung dezimal in binär

```
program Pas27022; { Zahlenkonvertierung "dezimal-binär" }
                  { K. Haller }
uses
  CRT;

type
  Str16 = string[16];
  Str21 = string[21];

var
  Dezimalzahl:             Word;
  BinaerString:            Str16;
  BinaerString Formatiert: string[21];

function BinStr(Dezimalzahl: Word): Str16; { ---------------- }
var                                                          { }
  Temp: Str16;                                               { }
  i:    Byte;                                                { }
begin                                                        { }
  if Dezimalzahl > 255                                       { }
     then Temp := '0000000000000000'                         { }
     else Temp := '        00000000';                        { }
  for i := 0 to 15 do                                        { }
    if Dezimalzahl and (1 shl i) = (1 shl i)                 { }
       then Temp[Length(Temp) - i] := '1';                   { }
  BinStr := Temp;                                            { }
end; {-------------------------------------------------------- }

function Formatierung(BinaerString: Str16): Str21; { ------- }
var                                                          { }
  Temp: Str21;                                               { }
begin                                                        { }
  Temp := BinaerString;                                      { }
  Insert(' ', Temp,  5);                                     { }
```

```
  Insert(' ', Temp, 14);                                    { }
  Insert('', Temp, 10);                                     { }
  Insert('', Temp,  1);                                     { }
  Insert('', Temp, 21);                                     { }
  Formatierung := Temp;                                     { }
end; { ──────────────────────────────────────────────────────┘ }
begin
  ClrScr;
  WriteLn('Zahlenkonvertierung dezimal-binär');
  WriteLn('Man achte auf Fehler, wenn Eingabe nicht im ',
          'Word-Bereich liegt.');
  WriteLn;
  WriteLn('Eingabe dezimal, Ende mit 0.    Binär unformatiert   ',
          ' Binär formatiert');
  WriteLn('-------------------------------------------------------',
          '----------------------');
  repeat
    Write('Dezimal (0..65535): ');
    {$R- Range-Prüfung ausnahmsweise auf AUS }
    ReadLn(Dezimalzahl);
    {$R+}
    BinaerString            := BinStr(Dezimalzahl);
    BinaerString Formatiert := Formatierung(BinaerString);
    GotoXY(33, WhereY - 1);
    WriteLn(BinaerString, '      ', BinaerString Formatiert);
  until Dezimalzahl = 0;

  repeat
  until ReadKey <> '';
end.
```

Eine mögliche Bildschirmausgabe:

```
Zahlenkonvertierung "dezimal-binär"
Man achte auf Fehler, wenn Eingabe nicht im Word-Bereich liegt.

Eingabe dezimal, Ende mit 0.    Binär unformatiert    Binär formatiert
-------------------------------------------------------------------------
Dezimal (0..65535): 1           ________00000001      |____ ____|0000 0001|
Dezimal (0..65535): 2           ________00000010      |____ ____|0000 0010|
Dezimal (0..65535): 4           ________00000100      |____ ____|0000 0100|
Dezimal (0..65535): 8           ________00001000      |____ ____|0000 1000|
Dezimal (0..65535): 16          ________00010000      |____ ____|0001 0000|
Dezimal (0..65535): 32          ________00100000      |____ ____|0010 0000|
Dezimal (0..65535): 64          ________01000000      |____ ____|0100 0000|
Dezimal (0..65535): 128         ________10000000      |____ ____|1000 0000|
Dezimal (0..65535): 129         ________10000001      |____ ____|1000 0001|
Dezimal (0..65535): 254         ________11111110      |____ ____|1111 1110|
Dezimal (0..65535): 255         ________11111111      |____ ____|1111 1111|
Dezimal (0..65535): 256         0000000100000000      |0000 0001|0000 0000|
Dezimal (0..65535): 65534       1111111111111110      |1111 1111|1111 1110|
Dezimal (0..65535): 65535       1111111111111111      |1111 1111|1111 1111|
Dezimal (0..65535): 65536       ________01100100      |____ ____|0110 0100|
Dezimal (0..65535): 65537       ________00000001      |____ ____|0000 0001|
Dezimal (0..65535): 65538       ________00000010      |____ ____|0000 0010|
```

b) Konvertierung binär in dezimal

```
program Pas27023; { Zahlenkonvertierung "binär-dezimal" }
                  { K. Haller }
uses
  CRT;
```

```
var
  Dezimalzahl:   Word;
  BinaerString:  string;
  i, Zeile:      Byte;
  Fehlerfrei:    Boolean;
begin
  ClrScr;
  WriteLn('Zahlenkonvertierung binär-dezimal');
  WriteLn;
  WriteLn('Eingabe binär, 1..16 Stellen. Ende mit 0.  Dezimal');
  WriteLn('-------------------------------------------------');

  repeat
    Zeile := WhereY;
    repeat
      Fehlerfrei := True;
      GotoXY(1, Zeile);
      Write('Eingabe binär: ................'); ClrEoL;
      GotoXY(16, Zeile);
      ReadLn(BinaerString);
      if (Length(BinaerString) <  1) or
         (Length(BinaerString) > 16)
         then Fehlerfrei := False;
      if Fehlerfrei
         then for i := 1 to Length(BinaerString) do
                if (BinaerString[i] <> '0')  and
                   (BinaerString[i] <> '1')
                   then Fehlerfrei := False;
    until Fehlerfrei;

    while Length(BinaerString) < 16 do
      BinaerString := '0' + BinaerString;

    Dezimalzahl := 0;
    for i := 0 to 15 do
      if BinaerString[16 - i] = '1'
         then Dezimalzahl := Dezimalzahl + 1 shl i;

    GotoXY(44, WhereY - 1);
    WriteLn(Dezimalzahl);
  until Dezimalzahl = 0;
  repeat
  until ReadKey <> '';
end.
```

27.3 Speicheradressierung. Segment und Offset

Der Stammvater der Intel-Mikroprozessoren für PCs, der Intel 8086 (auch 8088) besitzt 20 Adreßleitungen. Damit lassen sich 2^{20} = 1 048 576 Speicherstellen adressieren, das sind 1024 KByte oder 1 MByte. Jüngere Prozessoren besitzen mehr Adreßleitungen und somit einen größeren Adreßbereich (i80286 mit 24 Adreßleitungen, ab i80386 mit 32 Adreßleitungen). Aus Kompatibilitätsgründen laufen aber auf diesen Prozessoren die üblichen MS-Programme im sog. Real-Mode, der nur den Adreßbereich bis 1 MByte verwaltet. Nur mit besonderen Maßnahmen kann der zusätzliche Adreßraum genutzt

werden (Einsatz des Betriebssystems OS/2, Windows, Nutzung als Plattencache oder RAM-Disk, Emulation eines Expansionsspeichers usw. Siehe Kap. Betriebssystem).

Die folgenden Ausführungen beziehen sich auf den 8086 bzw. den Real-Mode der neueren Intel-Prozessoren mit dem gemeinsamen Merkmal, daß die Speicheradressen nicht fortlaufend (linear) durchgezählt werden, im Gegensatz zu Prozessoren anderer Hersteller, wie z.B. Motorola. Vielmehr wird der Adreßbereich in Segmente unterteilt, die an beliebigen Vielfachen von 16 Byte (= 1 Paragraph) beginnen können und maximal 64 KByte groß sein können. Die relative Adresse zum Segmentbeginn nennt man Offset, der aber nicht über eine maximale Segmentgröße hinausgehen kann. Für eine Adresse außerhalb dieser Grenze muß eine Segmentumschaltung vorgenommen werden.

Die segmentierte Speicheradresse wird in folgender Notation angegeben:

```
segment:offset
```

Es ist allgemein üblich (aber nicht notwendig), diese Angaben in hexadezimaler Notation zu machen. Da der Segmentbeginn immer ein Vielfaches von 16 darstellt, ergibt sich als letzte Stelle für den Segmentbeginn eine hexadezimale Null, die vereinbarungsgemäß weggelassen wird. Bei der Bildung der physischen Adresse wird die Segmentadresse um vier Bit nach links geschoben, was einer Multiplikation mit dem Faktor 16 entspricht, anschließend wird der Offset addiert.

Durch diese Technik bedingt, können sowohl Segment als auch Offset nur Werte zwischen 0 und 65535 (in hex: h0000 und hFFFF) annehmen und somit in den 16-Bit-Registern des Prozessors 8086 gespeichert werden.

Man beachte, daß in Pascal das Dollarzeichen $ als Hex-Vorsatzzeichen dient.

Beispiel 1: Physikalische Adresse des ersten Tastatur-Statusbytes, in üblicher Hex-Notation 0040:0017, in Pascal $0040:$0017

```
16 * (0*4096 + 0*256 + 4*16 + 0*1) +
     (0*4096 + 0*256 + 1*16 + 7*1)    =  1047
```

Beispiel 2: Physikalische Adresses des Beginns des Farbbildschirmspeichers, in üblicher Hex-Notation B800:0000 (Mono: B000:0000)

```
16 * (11*4096 + 8*256) = 753664
```

Beispiel 3: Die höchste physikalische Adresse, die rein rechnerisch mit den höchsten Werten von Segment und Offset gebildet werden kann, in üblicher Hex-Notation FFFF:FFFF (in Pascal $FFFF:$FFFF)

```
16 * (15*4096 + 15*256 + 15*16 + 15) +
      15*4096 + 15*256 + 15*16 + 15  =  1 114 095
```

Diese Adresse kann aber nicht mehr mit den 20 Adreßleitungen (A0 bis A19, 2^{20} = 1 048 576) des Intel 8086 dargestellt werden. Mit MS-DOS ab Version 5.0 und ab Prozessor 80286 kann aber dieser über 1 MByte hinausgehende und 64 KByte große Speicherbereich im Real-Mode

adressiert werden. Details siehe Kap. Betriebssystem (HMA, High Memory Area, Behandlungsroutine für die Adreßleitung A20).

Wichtig: Die gleiche physikalische Adresse kann aus vielen Kombinationen von **`Segment:Offset`** gebildet werden.

27.4 Zugriff auf Speicheradressen in Turbo-Pascal

Den Zugriff auf Speicheradressen (Speicherstellen) gestattet der in Turbo-Pascal vordefinierte (Pseudo-) Array `Mem[..]` (Mem steht für Memory) in den drei Varianten:

`Mem[`*`segment`*`:`*`offset`*`]`	Datentyp *Byte*, 1 Byte
`MemW[`*`segment`*`:`*`offset`*`]`	Datentyp *Word*, 2 Byte
`MemL[`*`segment`*`:`*`offset`*`]`	Datentyp *LongInt*, Doppelwort, 4 Byte

`segment``:`*`offset`* Beide Ausdrücke mit Datentyp Word. Konstanten in der Regel in Hex-Notation.

Der Zugriff kann mit diesem vordefiniertem Turbo-Pascal-Array sowohl lesend (RAM- und ROM-Speicherstellen) als auch schreibend (nur RAM-Speicherstellen) sein. In anderen Programmiersprachen gibt es getrennte Sprachelemente für Lesen und Schreiben, in Basic und C z.B. *Peek* für Lesen und *Poke* für Schreiben.

Beispiel:

```
....
var
  B: Byte;
  W: Word;
....
begin
  ....
  B := Mem[$0040:$0017];      { Lesen, 1 Byte     }
   ....
  W := MemW[$0040:$0017];     { Lesen, 2 Byte     }
   ....
  Mem[$0040:$0017] := 156;    { Schreiben, 1 Byte }
   ....
  MemW[$0040:$0017] := 64156; { Schreiben, 2 Byte }
  ....
end.
```

Beispiel: Beschreiben der Bildschirmspeicherstellen

```
program Pas27041;  { Bildschirmspeicher }
   { Zum Bildschirmspeicher: B800:0000 (color) bzw. B000:0000 (mono)
     Jeder Schreibstelle sind im Bildschirmspeicher 2 Bytes zuge-
     ordnet. Im ersten Byte (geradzahlige Adresse) steht das Zeichen;
```

```
     im zweiten Byte (ungeradzahlige Adressen) steht das Attribut des
     Zeichens (Farbe Vordergrund/Hintergrund, blinkend oder nicht).
     Das folgende Demo-Programm schreibt über den vordefiniertem Array
     »Mem[segment:offset]« direkt in den Bildschirmspeicher. Es werden
     alle 256 ASCII-Zeichen geschrieben und zwar in der linken oberen
     Ecke beginnend (Offset = 0 für das erste Zeichen). Das Attribut
     wird von Zeichen zu Zeichen gewechselt, der Einfachheit halber
     mit dem Wert der Laufvariablen.

     •  Mit »Mem[segment:offset]« können beliebige Speicherstellen
        byteweise angesprochen werden.
     •  Mit »MemW[segment:offset]« werden Speicherstellen mit 2 Bytes
        (Typ Word) angesprochen.

     •  Mit »MemL[segment:offset] werden Speicherstellen mit 4 Bytes
        (Typ LongInt) angesprochen.

     In allen Fällen sind »segment« und »offset« Ausdrücke mit dem
     Datentyp Word.

     Man sei bei direkten Speicherzugriffen vorsichtig!
   }
var
  i: Integer;

begin
  for i := 0 to 255 do
    begin
      Mem[$b800:2*i]     := i;     { Das erste Byte:  Das Zeichen  }
      Mem[$b800:2*i + 1] := i;     { Das zweite Byte: Das Attribut }
    end;                           { Zur Demo: Jedes Zeichen mit   }
                                   { anderem Attribut.             }
  repeat until ReadKey <> '';
end.
```

27.5 Inline-Code und Assembler-Code in Turbo-Pascal

a) Inline-Codes

... sind Prozessor-Codes (Bytezahlen 0..255, $00..$ff), die in den Pascal-Quelltext eingebaut werden, üblicherweise in Hex-Notation, d.h. in Pascal mit vorausgestellten Dollarzeichen. Die Codes kann man sich z.B. durch Disassemblieren eines (kleinen) COM-Files mit dem Hilfsprogramm DEBUG verschaffen. Siehe Kap. 30. Inline-Codes werden nach dem reservierten Word **inline** und einer öffnenden runden Klammer byteweise eingegeben. Als Trennzeichen dient der Schrägstrich /. Beendet werden Inline-Codes mit der schließenden runden Klammer. Die Eingabe der Codes ist ansonsten formatfrei.

Format für Inline-Codes:

inline (*code*[/*code*]... **)**

b) Assembler-Codes (ab Turbo-Pascal 6.0)

... sind Codes in den mnemotechnischen Bezeichnungen der Assemblersprache. Mit gewissen Einschränkungen können diese Codes ab Turbo-Pascal 6.0 assembliert werden. Details siehe Handbuch.

Format für Assembler-Codes:

```
asm
   asm-Anweisung  { Wenn mehrere Assembler-Anweisungen, dann    }
                  { Trennzeichen Semikolon oder Zeilenvorschub }
end;
```

1. Beispiel

```
program Pas27051;   { Print Screen (Hardcopy) über Interrupt h05 }
                    { Turbo-Pascal, "inline" und "asm" }
begin
  WriteLn;
  WriteLn('    Hardcopy mit Inline-Code »int h05« ');
  Write(  '    Wenn Drucker bereit, Taste Return: ');  ReadLn;

  inline ( $CD/$05 );   { Im Assembler: int 05 ; Interrupt 05 }
                        { = Auslösen einer Hardcopy           }

  WriteLn;
  WriteLn('    Hardcopy mit Asm-Code (Pascal 6.0) ');
  Write(  '    Wenn Drucker bereit, Taste Return: ');  ReadLn;

  asm           { »asm« Assembler-Code, ab Turbo-Pascal 6.0 }
    int $05     { Bei mehreren Assembler-Anweisungen als }
  end;          { Trennzeichen Semikolon oder neue Zeile }
end.
```

2. Beispiel:

```
program Pas27052;   { ASCII-Zeichensatz mit inline-Code   }
                    { Turbo-Pascal 5.0/6.0                }

begin
  WriteLn; WriteLn; WriteLn('ASCII-Zeichensatz über Inline-Code:  ');

  inline (
    $b1/$ff/     { 01: mov CL, ff   ; CL mit hex FF = dez 255      }
    $88/$ca/     { 02: mov DL, CL   ; Wert von CL in DL            }
    $b4/$02/     { 03: mov AH, 02   ; Funktion 02 bei Interrupt 21 }
    $cd/$21/     { 04: int 21       ; = Zeichen in DL anzeigen     }
    $fe/$c9/     { 05: dec CL       ; CL dekrementieren            }
    $75/$f6/     { 06: jnz 0102     ; jump if not zero. Siehe unten}
    $90/         { 07: nop          ; no operation (überflüssig)   }
    $b4/$08/     { 08: mov AH, 08   ; Funktion 08 bei Interrupt 21 }
    $cd/$21/     { 09: int 21       ; = Zeicheneingabe ohne Echo   }
    $b4/$4c/     { 10: mov AH, 4c   ; Funktion 4C bei Interrupt 21 }
    $cd/$21/     { 11: int 21       ; = Programm beenden           }
    $90/         { 12: nop          ; no operation (überflüssig)   }
    $90 );       { 13: nop          ; dto.                         }

    (* • Zur Sprungdistanz $f6 in Zeile 06:  $f6 = 246. Das höchste
         Bit ist gesetzt, da > 127. Somit negative Sprungdistanz
```

```
          (255 - 246) = 9 Byte rückwärts auf erstes Byte in Zeile 02
          (Adresse h0102 bei COM-File).
          Maximale Sprünge: -128 und +127. Bei größeren Distanzen
          über Zwischenwert und von dort weiter.
       •  Das Programm gibt den Ascii-Code rückwärts aus. Einige
          Steuerzeichen werden interpretiert. Welche? BEL, BS, HT, LF
    *)
end.
```

27.6 Die Turbo-Pascal-Unit DOS

In der Unit DOS sind in Turbo-Pascal zahlreiche Konstanten, Datentypen, Variablen, Prozeduren und Funktionen definiert. Nachstehend nur eine Auswahl für systemnahe Programmierung:

a) Record-Typ *Registers*

Der Record-Typ `Registers` ist ein vordefinierter Record, der auschließlich aus varianten Teilen besteht. Er ist in der Unit DOS wie folgt definiert:

```
type
  Registers =  record
    case Integer of
      0: (AX, BX, CX, DX, BP, SI, DI, DS, ES, Flags: Word);
      1: (AL, AH, BL, BH, CL, CH, DL, DH:            Byte);
    end;
```

Wenn beispielsweise im Pascal-Programm eine Variable mit dem freien aber sinnvoll gewählten Bezeichner "`Reg`" (Registervariable) mit dem vordefinierten Record-Datentyp `Registers` (aus Unit DOS) definiert ist, dann kann auf die einzelnen Register über die Felder der Registervariable nach einem Interrupt-Aufruf wie folgt zugegriffen werden:

```
var
  Reg: Registers;
```

`Reg.AX`	16-Bit-Register `AX`
`Reg.AL`	8-Bit-Register `AL`
`Reg.BH`	8-Bit-Register `BH`
`Reg.Flags`	16-Bit-Register `Flags`

Beispiel:

```
if Reg.AL <> 0 then ....
....
WriteLn(Reg.AX);
```

In ähnlicher Weise können die Registervariablen mit einem Wert für einen folgenden Interrupt-Aufruf belegt werden.

Beispiel: `Reg.AL := 23;`

b) Flag-Konstanten:

`FCarry = $0000;`	Bit 0:	Carry-Flag. Übertrags-Flag
`FParity = $0004;`	Bit 2:	Paritäts-Flag
`FAuxiliary = $0010;`	Bit 4:	Auxiliary-Flag. Hilfsübertrags-Flag
`FZero = $0040;`	Bit 6:	Zero-Flag. Null-Flag
`FSign = $0080;`	Bit 7:	Sign-Flag. Vorzeichen-Flag
`FOverflow = $0800;`	Bit 11:	Overflow-Flag. Überlauf-Flag

Die vorstehenden sechs Flags sind Status-Flags. Sie signalisieren das Ergebnis einer arithmetischen oder logischen Operation.

Die Bits 1, 3, 5, 12, 13, 14, 15 des Flag-Registers werden beim Intel 8086 nicht benutzt. Die Bits 8 (Trap, Einzelschritt-Flag), 9 (Interrupt-Flag) und 10 (Direction, Richtungs-Flag) sind Steuerflags um die Arbeitsweise des Prozessors zu beeinflussen.

Wenn beispielsweise im Pascal-Program mit `Reg` ein Record-Datentyp `Registers` (aus Unit DOS) definiert ist, dann können die einzelnen Flags z.B. wie folgt abgefragt werden:

```
if Reg.Flags and FCarry <> 0 then ...
```
(True, wenn Carry-Flag gesetzt, False wenn nicht gesetzt)

```
if Reg.Flags and FZero = 0 then ...
```
(True, wenn Zero-Flag nicht gesetzt. False wenn gesetzt)

c) Prozedur *Intr* **Format:** `Intr(`*`interruptNr, registervariable`*`)`

Diese Prozedur führt einen Interrupt aus. Siehe Beispiel in 27.8

d) Prozedur *MsDOS* **Format:** `MsDOS(`*`registervariable`*`)`

Diese Prozedur führt einen DOS-Funktionsaufruf aus (= Interrupt 21h). Siehe Beispiel in 27.8

e) Prozedur *GetIntVec* **Format:** `GetIntVec(`*`interruptNr, zeigervariable`*`)`

Diese Prozedur ermittelt die Adresse, auf die ein Interrupt-Vektor zeigt.

f) Prozedur *SetIntVec* **Format:** `SetIntVec(`*`interruptNr, zeiger`*`)`

Diese Prozedur setzt einen Interrupt-Vektor auf eine bestimmte Adresse.

g) Funktion *DosVersion*

Diese Funktion liefert die DOS-Version im Datentyp Word. Die beiden Bytes müssen getrennt interpretiert werden. Im niederwertigen Byte (Low-Byte) steht Hauptnummer,

im höherwertigen Byte (High-Byte) die Unternummer. Mit den Standardfunktion `Lo(..)` und `Hi(..)` können die beiden Bytes getrennt angesprochen werden.

Beispiel:

```
...
WriteLn('DOS-Version: ', Lo(DosVersion), '.', Hi(DosVersion));
...
```

27.7 Die Interrupts im Überblick

Dieses Kapitel enthält einen Überblick über die Interrupts. Eine vollständige Behandlung ist an dieser Stelle nicht möglich. Es werden nur die Interrupts bzw. die Funktionen mit ihren Registerbelegungen genauer erklärt, die bei den Praktikumsaufgaben bzw. Demo-Programmen verwendet werden.

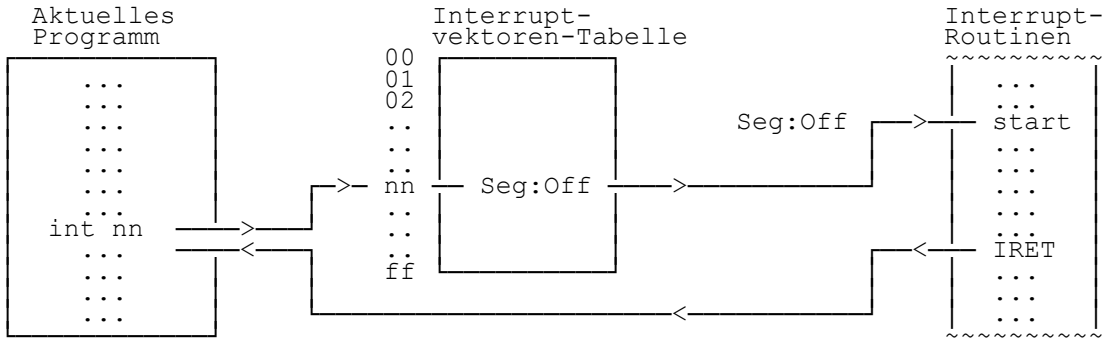
Literaturhinweise:

[1] Microsoft MS-DOS Programmer's Reference. Version 3.3, Microsoft Corporation, 1988. Nur DOS-Interrupts.
[2] Microsoft Mouse, Programmmer's Reference Guide, Microsoft Corporation, 1986. Nur Maus-Interrupt 33h.
[3] W. Höfs "MS-DOS", Sybex-Ratgeber, Sybex-Verlag, 1986. Nur DOS-Interrupts
[4] P. Norton "Neues Programmierhandbuch für IBM PC & PS/2", Microsoft Press Vieweg-Verlag, 1989. Kurzbeschreibung der meisten Interrupts
[5] M. Tischer "PC Intern 3.0", Data Becker Verlag, 1993. Ausführliche Beschreibung aller Interrupts.

27.7.1 Die Interrupt-Vektoren

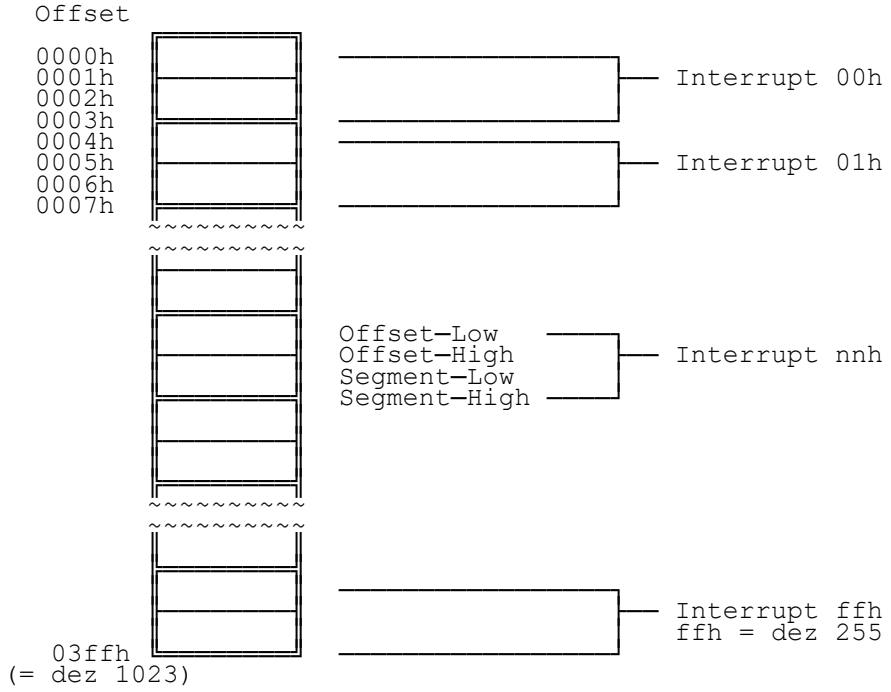
Das Betriebssystem (von Microsoft) und das BIOS (Basic Input Output System des Rechnerherstellers) stellen ihre Dienste in einer Vielzahl von Interrupt-Routinen zur Verfügung. Diese Routinen sind wie Pascal-Prozeduren auzufassen. Am Ende steht der Befehl **IRET** (interrupt return, Maschinencode CFh = dez 207), von dort aus wird an das aktuelle Programm zurückgekehrt und mit der nächsten Anweisung fortgesetzt. An die Interrupt-Routinen können über Register Werte übergeben werden, ähnlich den Parametern in Pascal. In ähnlicher Weise können Interrupts auch Werte über Register zurückliefern. Der Zugriff auf die Interrupt-Routinen erfolgt nicht direkt durch die Angabe der Segment:Offset-Adresse, sondern über eine Tabelle, in der die Segment:Offset-Adressen (Interrupt-Vektoren) in der Reihenfolge der Interrupt-Nummern gespeichert sind. Die Tabelle befindet sich am Anfang des RAM-Speichers und wird beim Booten angelegt. Da RAM-Speicherstellen überschrieben werden können, ist es auch möglich, die Segment:Offset-Adressen so zu ändern ("Interrupt-Vektor verbiegen"), daß sie auf andere oder eigene Routinen weisen. Die alten Werte sollte man aber später wieder reaktivieren.

Das folgende Schema zeigt den Ablauf bei einem Interrupt-Aufruf:



Da in der üblichen Intel-Notation Segment und Offset durch je 2 Byte dargestellt werden, sind für einen Vektor 4 Byte in der Tabelle notwendig. Die Interrupt-Nummer wird durch eine 1-Byte-Zahl ausgedrückt. Somit sind 256 Interrupt-Nummern möglich (wenn auch nur der kleinere Teil vergeben ist). Die Tabelle hat somit einen Umfang von 256 * 4 Byte = 1024 Byte = 1 KByte. Sie befindet sich ganz am Anfang des RAM-Speichers, also Segment-Adresse 0 und Offset-Adressen von 0 bis 1023, in hex: 0000h bis 03ffh). Bei jedem Eintrag wird zuerst die Offset- und dann die Segment-Adresse aufgeführt. Es ist weiter zu beachten, daß bei einem (16-bit-) Wort das niederwertige Byte (Low-Byte) zuerst gespeichert wird (Intel-Notation).

Interruptvektoren-Tabelle im Segment 0000h:



Demo-Programm: Auslesen der Interrupt-Vektoren

```
program Pas27071; { Kap: 27.7: Interrupt-Vektoren         }
                  { K. Haller  }
 { Hinweis: Über die hier gezeigten Methoden hinaus gibt es
   in Turbo-Pascal noch die DOS-Prozedur   "GetIntVec(int_nr,
```

```
   pointer variable)". Siehe auch DOS-Prozedur "SetIntVec(int nr,
   pointer)"
  }
uses
  CRT, DOS;
type
  String5 = string[5];
var
  Ch:              Char;
  i:               Integer;
  Dez:             LongInt; { Nur für Einzug }
  Segment,
  Offset:          Word;
  Physik Adresse,
  Segment LongInt: LongInt;
  Zeiger:          Pointer; { Pointer = vordefinierter Datentyp
                              für untypisierten Zeiger }
function Dez HexStr(x: Word): String5;
var
  TempStr: String5;

begin
  TempStr := '';
  while x <> 0 do
    begin
      TempStr := Copy('0123456789ABCDEF',
                      (x mod 16) + 1, 1) + TempStr;
      x := x div 16;
    end;
  while Length(TempStr) < 4 do
    TempStr := '0' + TempStr;
  Dez HexStr := '$' + TempStr;
end;
begin
  ClrScr;
  New(Zeiger);
  GetIntVec(5, Zeiger);  { Interrupt 5: Hardcopy }
  WriteLn('  Mit "Ctrl-F4" Evaluate die Pointer-Variable "Zeiger" ');
  WriteLn('  für Interrupt-Vektor für BIOS-Interrupt 5 = Hardcopy ');
  Write  ('  = $F000:$FF54 anzeigen (j/n): ');
repeat
    Ch := UpCase(ReadKey);
  until (Ch = 'J') or (Ch = 'N');
  WriteLn(Ch);
  WriteLn;

  if Ch = 'J' then Halt; { >>>>>>>>>>>>>>>> }

  repeat
    Write('Eingabe Interrupt-Vektor (dez oder $hex, Ende mit -1): ');
    ReadLn(i);
    if (i >= 0) and (i <= 255) then
       begin
         Offset  := MemW[0:i*4    ];
         Segment := MemW[0:i*4 + 2];
         Segment LongInt := Segment;  { sonst Fehler bei "16 *" }
         Physik Adresse  := 16 * Segment LongInt + Offset;
         WriteLn('  Interrupt dez ', i:3, '  hex $',
                    Copy(Dez_HexStr(i), 4, 2), '   ',
```

```
                    Dez HexStr(Segment), ':',
                    Dez HexStr(Offset),
                    '  Physikalisch: ', Physik Adresse);
        WriteLn;
        WriteLn('  Jetzt über Mem[..] statt MemW[..]: ');
        Offset  := Mem[0:i*4 + 0] + 256 * Mem[0:i*4 + 1];
        Segment := Mem[0:i*4 + 2] + 256 * Mem[0:i*4 + 3];
        Segment LongInt := Segment; { Sonst Fehler bei "16 *" }
        Physik Adresse  := 16 * Segment LongInt + Offset;
        WriteLn('  Interrupt dez ', i:3, '  hex $',
                    Copy(Dez HexStr(i), 4, 2), '   ',
                    Dez HexStr(Segment), ':',
                    Dez HexStr(Offset),
                    '  Physikalisch: ', Physik Adresse);
        WriteLn;
      end;
  until i < 0;
end.
```

Demo-Programm: "Verbiegen" eines Interrupts

```
program Pas27072;  { Interrupt-Vektor verbiegen }
                   { Turbo-Pascal, K. Haller    }
 { Hinweise:
   • Mit "MemW[segment:offset]" kann man Datentyp Word im richtigen
     Format abspeichern und auch auslesen, d.h. es wird berück-
     sichtigt, daß das Low-Byte zuerst gespeichert wird. Auf diese
     Möglichkeit wurde hier bewußt verzichtet; die Vektorkomponenten
     werden im Programm mit "Mem[segment:offset]" byteweise ange-
     sprochen und erst dann zusammengesetzt mit:
          Word = LowByte + 256 * HighByte
   • Auf die elegantere, aber nicht so durchsichtige Lösung der
     gestellten Aufgaben mittels der Turbo-Pascal-DOS-Prozeduren
     "GetIntVec(int nr, pointervariable)"  und "SetIntVec(int nr,
     pointer)" wurde hier verzichtet.
 }
uses
  CRT, DOS;
const
  i = 4;    { Interrupt 04h = Overflow. Vektor wird zeit-
                              weise auf 05h verbogen      }
  j = 5;    { Interrupt 05h = Hardcopy }
var
  Segment, Segment Alt,
  Offset,  Offset Alt:    Word;
  Byte0, Byte0 Alt,
  Byte1, Byte1 Alt,
  Byte2, Byte2 Alt,
  Byte3, Byte3 Alt:       Byte;
  Reg:                    Registers;  { Recordtyp aus Unit DOS }
begin
  ClrScr;
  WriteLn; WriteLn;
  WriteLn('  Hardcopy-Interruptvektor ermitteln und "verbiegen" ');
  WriteLn; WriteLn; WriteLn; WriteLn;
```

```
  Byte0 Alt   := Mem[0:i*4 + 0];  { Offset-Low   }
  Byte1 Alt   := Mem[0:i*4 + 1];  { Offset-High  }
  Byte2 Alt   := Mem[0:i*4 + 2];  { Segment-Low  }
  Byte3 Alt   := Mem[0:i*4 + 3];  { Segment-High }
  Offset Alt  := Byte0 Alt + 256 * Byte1 Alt;
  Segment Alt := Byte2 Alt + 256 * Byte3 Alt;
  WriteLn('  Interrupt 0', i, 'h = Overflow. Segment:Offset = ',
             Segment Alt:5, ':', Offset Alt:5, ' (in dez)');
  WriteLn;
  WriteLn('  Jetzt wird Interrupt 04h auf 05h = Hardcopy "verbogen"');
  Mem[0:i*4 + 0] := Mem[0:j*4 + 0];  { Offset-Low   }
  Mem[0:i*4 + 1] := Mem[0:j*4 + 1];  { Offset-High  }
  Mem[0:i*4 + 2] := Mem[0:j*4 + 2];  { Segment-Low  }
  Mem[0:i*4 + 3] := Mem[0:j*4 + 3];  { Segment-High }
  Byte0   := Mem[0:j*4 + 0];  { Offset-Low   }
  Byte1   := Mem[0:j*4 + 1];  { Offset-High  }
  Byte2   := Mem[0:j*4 + 2];  { Segment-Low  }
  Byte3   := Mem[0:j*4 + 3];  { Segment-High }
  Offset  := Byte0 + 256 * Byte1;
  Segment := Byte2 + 256 * Byte3;

  WriteLn('  Interrupt 0', i, 'h = Hardcopy. Segment:Offset = ',
             Segment:5, ':', Offset:5, ' (in dez)');
  WriteLn;
  WriteLn('  Jetzt wird über Software-Aufruf des Interrupts 04h ');
  Write('  Hardcopy ausgelöst. Wenn Drucker bereit, Taste Return ');
  repeat
  until ReadKey = #13;
  Intr($04, Reg);
  WriteLn; WriteLn;
  WriteLn('  Jetzt wird der alte Interruptvektor reaktiviert: ');
  Mem[0:i*4 + 0] := Byte0 Alt;  { Offset-Low   }
  Mem[0:i*4 + 1] := Byte1 Alt;  { Offset-High  }
  Mem[0:i*4 + 2] := Byte2 Alt;  { Segment-Low  }
  Mem[0:i*4 + 3] := Byte3 Alt;  { Segment-High }
  Byte0   := Mem[0:i*4 + 0];    { Offset-Low   }
  Byte1   := Mem[0:i*4 + 1];    { Offset-High  }
  Byte2   := Mem[0:i*4 + 2];    { Segment-Low  }
  Byte3   := Mem[0:i*4 + 3];    { Segment-High }
  Offset  := Byte0 + 256 * Byte1;
  Segment := Byte2 + 256 * Byte3;
  WriteLn('  Interrupt 0', i, 'h = Overflow. Segment:Offset = ',
             Segment:5, ':', Offset:5, ' (in dez)');
  repeat
  until ReadKey <> '';
end.
```

27.7.2 Die Interrupts

Inter-rupt	Typ	Bemerkungen
00h	CPU	Division durch null
01h	CPU	Einzelschritt
02h	CPU	NMI. Nicht maskierbarer Interrupt
03h	CPU	Breakpoint

04h	CPU	Überlauf
05h	BIOS	Hardcopy
08h	CPU	Zeitgeber
09h	CPU	Tastatur
10h	BIOS	Bildschirm, 18 Funktionen
11h	BIOS	Konfiguration
12h	BIOS	Feststellen der Speichergröße
13h	BIOS	Disketten/Platten (formatieren, schreiben, lesen, usw.) Für Disketten 9 Funktionen, für Festplatten 15 Funktionen
14h	BIOS	Serielle Schnittstelle, 4 Funktionen
15h	BIOS	Diverses für AT (alter Kassetteninterrupt), 8 Funktionen
16h	BIOS	Tastatur, 3 Funktionen
17h	BIOS	Parallele Drucker-Schnittstelle (Centronics), 3 Funktionen
18h	BIOS	ROM-Basic (IBM)
19h	BIOS	Booten des Rechners
1Ah	BIOS	Datum und Zeit, 8 Funktionen
1Bh	BIOS	Tastatur: Break-Taste betätigt
1Ch	BIOS	Periodischer Interrupt
1Dh	BIOS	Video-Tabelle
1Eh	BIOS	Laufwerkstabelle
1Fh	BIOS	Zeichentabelle, nur Pointer
20h	DOS	Programm beenden (besser über Int 21h, Funktion 4Ch)
21h	**DOS**	**Allgemeine DOS-Funktionen** Über 100 Funktionen und Unterfunktionen
22h	DOS	Programm beenden. Siehe Int 20h.
23h	DOS	Break-Taste betätigt
24h	DOS	Kritischer Fehler
25h	DOS	Absolutes Lesen (Platte, Diskette)
26h	DOS	Absolutes Schreiben (Platte/Diskette)
27h	DOS	Programm beenden, aber im Speicher belassen
33h	Maus	Maus oder Lichtgriffel, über 30 Funktionen
67h	EMS	Expanded Memory System nach LIM, 11 Funktionen

Nachfolgend werden die Interrupts kurz erläutert. Nur bei einigen ausgewählten Anwendungen werden genauere Informationen gegeben.

Zum Interrupt 00h: Division durch null

Der zugehörige Vektor wird von DOS auf eine Routine gelegt, die eine Fehlermeldung ausgibt. Nach dem abschließenden Interrupt-Return-Befehl **IRET** wird das Programm mit dem Befehl fortgesetzt, der auf den fehlerhaften Divisionsbefehl folgt.

Zum Interrupt 01h: Einzelschritt

Wird von der CPU dann aufgerufen, wenn das Trap-Bit des Flag-Registers gesetzt ist. Dann wird das Programm schrittweise ausgeführt. Das BIOS setzt den Interrupt-Vektor aber auf den Befehl IRET, so daß beim Setzen des Trap-Bits außer einer Verlangsamung nichts passiert. Sinnvoll nur bei Testprogrammen wie DEBUG um Programmablauf und Registerbelegung verfolgen zu können. DEBUG "verbiegt" den Interrupt-Vektor auf eine eigene Routine, in der das Trap-Bit auch wieder gelöscht werden kann.

Zum Interrupt 02h: NMI, nicht maskierbarer Interrupt

Dieser Interrupts kann im Gegensatz zu allen anderen *nicht* mit dem Befehl **CLI** (clear interrupts, Löschen des Interrupt-Flags **IF** im Flag-Register) gesperrt werden. Beim Auftreten von RAM-Fehlern wird auf diesen Interrupt verzweigt, der das System anhält.

Zum Hardware-Interrupt 03h: Breakpoint

Innerhalb eines Testprogramms (z.B. DEBUG) Unterbrechungspunkte setzen um Registerinhalte anzuzeigen.

Zum Interrupt 04h: Überlauffehler

Der Interrupt wird unter bestimmten Umständen aufgerufen, wenn das Ergebnis einer Operation nicht mehr in die dafür vorgesehene Registerbreite paßt, was z.B. bei einer Multiplikation der Fall sein kann. DOS setzt aber den Interruptvektor standardmäßig auf den Befehl IRET, so daß der Interrupt nicht zur Wirkung kommt.

Zum Interrupt 05h: Hardcopy

Nach Drücken der Taste PrtSc (deutsch Druck) wird eine Hardcopy des Text-Bildschirms auf den Drucker ausgegeben. Bei Graphik-Bildschirmen muß vorher das Programm GRAPHICS.COM geladen werden. Die Grafik-Ausgabe ist aber nur bei IBM-Grafik-Druckern oder dazu kompatiblen Druckern fehlerfrei.

Zum Interrupt 08h: Zeitgeber

Der Schwingquarz des Timer-Bausteins arbeitet mit einer Frequenz von 1.193.180 Hz. Nach $2^{16} = 65536$ Schwingungen erzeugt der Timer-Baustein einen Aufruf des Interrupts 08h, d.h. in einer Sekunde 18,20648193 mal (ca. 18,2mal). Diese Frequenz ist *unabhängig* von der Taktfrequenz des Mikroprozessors (8 MHz, 16 MHz, 25 MHz, 33 MHz, 40 MHz usw.).

Zum Interrupt 09h: Tastatur

Ein eigener Tastatur-Prozessor überwacht die Tastatur. Der Interrupt wird ausgelöst, wenn eine Taste gedrückt oder losgelassen wird. Die weitere Verarbeitung erfolgt über nachgeschaltete BIOS-Tastatur-Routinen.

Zum Interrupt 10h: Bildschirm

Funktionsnummer
Unterfunktion

		Bemerkungen
00h		Video-Modus setzen
01h		Gestalt des Cursors definieren
02h		Cursor positionieren
03h		Cursorposition ermitteln
05h		Bildschirmseite auswählen
06h		Textzeilen noch oben scrollen
07h		Textzeilen nach unten scrollen
08h		Zeichen/Attribut an Cursorstelle lesen
09h		Zeichen/Attribut an Cursorstelle schreiben, ohne Cursorversatz
0Ah		Zeichen an Cursorstelle schreiben, altes Attribut, ohne Cursorversatz
0Bh	00h	Auswahl Farbe für Rahmen und Hintergrund
0Bh	01h	Auswahl Farbpalette für Graphik 320 * 200
0Ch		Graphikpunkt schreiben

0Dh		Graphikpunkt lesen
0Eh		Zeichen an Cursorstelle schreiben, altes Attribut, mit Cursorversatz
0Fh		Video-Modus auslesen
10h	xxh	Nur für EGA/VGA: Unterfunktionen xx = 00, 01, 02, 03, 07, 10, 12 13, 15, 17, 18, 19, 1A, 1B
11h	xxh	Nur für EGA/VGA: Unterfunktionen xx = 00, 01, 02, 03, 10, 11, 12, 14, 30
12h	xxh	Nur für EGA/VGA: Unterfunktionen xx = 10, 20, 30, 31, 32, 33, 34, 36
13h		Ausgabe Zeichenkette an bestimmter Cursorposition
1Ah		Nur VGA: Code für Emulation einer anderen Bildschirmkarte ermitteln

Zur Funktion 02h des Bildschirm-Interrupts 10h, Cursor positionieren

Eingabe: AH = 02h
BH = Nummer der Bildschirmseite (0, 1, ..)
DH = Bildschirmzeile (0..24, Text)
DL = Bildschirmspalte (0..79, Text)
Ausgabe: keine

Zur Funktion 03h des Bildschirm-Interrupts 10h, Cursorposition ermitteln

Eingabe: AH = 03h
BH = Nummer der Bildschirmseite (0, 1, ..)
Ausgabe: DH = Bildschirmzeile (0..24, Text)
DL = Bildschirmspalte (0..79, Text)
CH = Anfangszeile des Cursors
CL = Endzeile des Cursors

Zur Funktion 09h des Bildschirm-Interrupts 10h, Zeichen und Attribut schreiben

Eingabe: AH = 09h
AL = (ASCII-) Code des Zeichens
BL = Attribut des Zeichens
CX = Anzahl der Wiederholungen der Zeichenausgabe
BH = Nummer der Bildschirmseite (0, 1, ..)
Ausgabe: keine

Bei dieser Funktion werden Steuerzeichen *nicht* interpretiert. Der Cursor wird mit Ausnahme der Wiederholungen nicht versetzt; er muß deshalb mit der Funktion 02h versetzt werden.

Zum Interrupt 11h: Konfiguration

Der Interrupt 11h hat keinen weiteren Eingabeparameter. Das Ergebnis wird im Register AX zurückgeliefert, wobei die Bits einzeln interpretiert werden müssen und zwar unterschiedlich bei PC/XT- und bei AT-Rechnern. Es interessiert nur der AT-Rechner:

Für AT-Rechner gilt:

```
<———————— Bit—Nr ————————————————— | Bit 0..7: AL,  Bit 8..15: AH
1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0     |
5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0     | Bedeutung (für AT und PS—2)
═══════════════════════════════════ ═══════════════════════════════
x x . . . . . . . . . . . . . . .   | Anzahl der parallelen Drucker
. . x . . . . . . . . . . . . . .   | Nicht verwendet
. . . x . . . . . . . . . . . . .   | Nicht verwendet
. . . . x x x . . . . . . . . . .   | Anzahl serielle Schnittstellen
. . . . . . . x . . . . . . . . .   | Nicht verwendet
. . . . . . . . x x . . . . . . .   | Anzahl Diskettenlaufwerke — 1
                                    |    0 0:  1 Diskettenlaufwerk
                                    |    0 1:  2 Diskettenlaufwerke
```

```
                                    |    1 0:  3 Diskettenlaufwerke
                                    |    1 1:  4 Diskettenlaufwerke
. . . . . . . . . . x x . . . .     | Bildschirmmodus beim Booten
                                    |    0 0:  Nicht verwendet
                                    |    1 0:  Color,     80 * 25
                                    |    0 1:  Color,     40 * 25
                                    |    1 1:  Monochrom, 80 * 25
. . . . . . . . . . . . x . . .     | Nicht verwendet
. . . . . . . . . . . . . x . .     | Zeigegerät (Maus) installiert
. . . . . . . . . . . . . . x .     | Coprozessor installiert
. . . . . . . . . . . . . . . x     | Diskettenlaufwerk(e) vorhanden
====================================|=================================
5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0     |
1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0     |
```

Anmerkung: Der Bildschirmmodus kann nach dem Booten verändert worden sein. Der aktuelle Modus kann nicht mit dem Interrupt 11h abgefragt werden, sondern muß mit der Funktion 0Fh des Interupts 10h ausgelesen werden. Das Ergebnis steht dann im Register AL.

Zum Interrupt 12h: Speicher unter 1 MByte ermitteln

Nach Aufruf dieses Interrupts steht im Register AX die Speichergröße in KByte. Die Speichergröße *über* 1 MByte kann mit der Funktion 88h des Interrupts 15h (Diverses für AT) abgefragt werden.

Zum Interrupt 13h: Diskette/Platte

Die Laufwerke werden intern mit folgenden Nummern angesprochen:

00h Diskettenlaufwerk A:
01h Diskettenlaufwerk B:
80h Plattenlaufwerk C:
81h Plattenlaufwerk D:
82h Plattenlaufwerk E: usw.

Funktionsnummer
Unterfunktion

		Bemerkungen
00h		Reset
01h		Status lesen
02h		Lesen
03h		Schreiben
04h		Verifizieren
05h		Formatieren
08h		Nur Platte: Format ermitteln
09h		Nur Platte: Anpassung fremder Laufwerke
0Ah		Nur Platte: Erweitertes Lesen
0Bh		Nur Platte: Erweitertes Schreiben
0Dh		Nur Platte: Reset
10h		Nur Platte: Laufwerk bereit
11h		Nur Platte: Rekalibrierung des Laufwerks
14h		Nur Platte: ControllerÄDiagnose
15h		Feststellen des Laufwerktyps. Nur AT
16h		Nur Diskette: Fesstellen des Diskettenwechsels. Nur AT
17h		Nur Diskette: Diskettenformat festlegen. Nur AT

Zum Interrupt 14h: Serielle Schnittstelle

Funktionsnummer
Unterfunktion

		Bemerkungen
00h		Initialisierung
01h		Ein Zeichen senden
02h		Ein Zeichen empfangen

03h		Status ermitteln

Zum BIOS-Interrupt 15h: Diverses für AT

Dieser Interrupt diente früher als Kassetteninterrupt und wurde mit der Einführung der AT-Rechner (Mikroprozessor Intel 80286) geändert.

Funktionsnummer
Unterfunktion

		Bemerkungen
83h		Flag nach Zeitintervall setzen
84h	00h	Status-Abfrage der Joystick-Feuerknöpfe
84h	01h	Position der Joysticks abfragen
85h		Taste SysReq (S-Abf) betätigt
86h		Warten bis bestimmte Zeit verstrichen
87h		Speicherbereich über 1 MByte verschieben
88h		Speichergröße über 1 MByte ermitteln
89h		Umschaltung in Protected Mode

Hinweis zur Funktion 88h: Die Speichergröße *unter* 1 MByte kann mit dem Interrupt 12h abgefragt werden.

Zur Funktion 88h des AT-Interrupts 15h:

Eingabe: AH = 88h
Ausgabe: AX = Größe des Speichers (über 1 MByte) in KByte

Zum Interrupt 16h: Tastatur

Funktionsnummer
Unterfunktion

		Bemerkungen
00h		Ein Zeichen aus Tastaturpuffer lesen
01h		Abfrage, ob Zeichen im Tastaturpuffer
02h		Status der Tastatur ermitteln

Zur Funktion 02h des Tastatur-Interrupts 16h:

Eingabe: AH = 02h
Ausgabe: AL = Statusbyte der Tastatur nach folgender Tabelle:

<—— Bit—Nr —— 7 6 5 4 3 2 1 0	Bit 0..7, Byte 0040:0017 Bedeutung
x	1 = Insert an
. x	1 = Caps Lock an
. . x	1 = Num Lock an
. . . x	1 = Scroll Lock an
. . . . x . . .	1 = Alt—Taste bet„tigt
. . . . . x . .	1 = Ctrl—Taste bet„tigt
. . . . . . x .	1 = linke Shift—Taste bet„tigt
. . . . x	1 = rechte Shift—Taste bet„tigt

Hinweise:

- Das von der Funktion 02h zurückgelieferte Statusbyte wird aus dem ersten Tastatur-Statusbyte im RAM gelesen, das sich an folgender Segment:Offset-Adresse befindet: 0400:0017 (hex). Da sich

diese Adresse im RAM befindet, kann der Inhalt auch geändert werden; in Turbo-Pascal z.B. mit "Mem[segmet:offset] := ...". Auf diese Weise ist es z.B. möglich, die Num-Lock-Taste per Software zu aktivieren, in dem man das Bit 5 auf 1 setzt. Es kann nicht immer davon ausgegangen werden, daß das Tastaturstatusbyte von allen Anwenderprogrammen im ursprünglichen Sinn interpretiert wird.

- In der nächsten RAM-Speicherstelle, also 0040:0018 ist ein weiteres Tastatur-Statusbyte abgelegt, das wie folgt zu interpretieren ist:

<—— Bit—Nr —— 7 6 5 4 3 2 1 0	Bit 0..7: Byte 0040:0018 Bedeutung
x	1 = Insert gedr•ckt
. x	1 = Caps Lock gedr•ckt
. . x	1 = Num Lock gedr•ckt
. . . x	1 = Scroll Lock gedr•ckt
. . . . x . . .	1 = Pause oder Ctrl—Num Lock gedr•ckt
. . . . . x . .	1 = Sys Req oder S Abf gedr•ckt
. . . . . . x .	1 = linke Alt—Taste bet„tigt
. . . . x	1 = linke Ctrl—Taste bet„tigt

Siehe auch Tastatur-Demoprogramm.

Zum Interrupt 17h: Drucker (parallel, Centronics)

Funktionsnummer
Unterfunktion

		Bemerkungen
00h		Ein Zeichen auf Drucker ausgeben
01h		Drucker initialisieren
02h		Druckerstatus ermitteln

Zur Funktion 00h des Drucker-Interrupts 17h, Zeichen auf Drucker ausgeben:

Eingabe: AH = 00h
AL = (ASCII-) Code des Zeichens
DX = Nummer des Druckers, wobei 0000h = LPT1:, 0001h = LPT2:
Ausgabe: AH = Statusbyte des Druckers nach folgender Tabelle:

<—— Bit—Nr —— 7 6 5 4 3 2 1 0	Bit 0..7: Drucker—Status Bedeutung
x	0 = Drucker ist besch„ftigt
. x	1 = Empfang best„tigt
. . x	1 = Papier aus
. . . x	1 = Drucker auf On Line
. . . . x . . .	1 = šbertragungsfehler
. . . . . x . .	nicht verwendet
. . . . . . x .	nicht verwendet
. . . . x	1 = Time—Out—Fehler

Zur Funktion 01h des Drucker-Interrupts 17h, Drucker intialisieren:

Eingabe: AH = 01h
DX = Nummer des Druckers, wobei 0000h = LPT1:, 0001h = LPT2:
Ausgabe: AH = Statusbyte des Druckers nach vorstehender Tabelle:

Zur Funktion 02h des Drucker-Interrupts 17h, Druckerstatus abfragen:

Eingabe: AH = 02h
DX = Nummer des Druckers, wobei 0000h = LPT1:, 0001h = LPT2:
Ausgabe: AH = Statusbyte des Druckers nach vorstehender Tabelle:

Zum Interrupt 18h: ROM-Basic

Falls ROM-Basic vorhanden ist, z.B. bei IBM-PCs, wird es mit diesem Interrupt gestartet. Nach Beenden des Interrupts ist aber keine Rückkehr zum aufrufenden Programm möglich. Warm- oder Kaltstart erforderlich. Der Interrupt 18h hat keinen Parameter.

Zum Interrupt 19h: Booten des Rechners

Nach Aufruf dieses Interrupts wird der Rechner gebootet. Er wird auch von der Tastaturroutine bei der Tastenkombination **Ctrl+Alt+Del** (Affengriff Strg+Alt+Entf) aufgerufen. Der Interrupt 19h hat keinen Parameter.

Zum BIOS-Interrupt 1Ah: Datum und Zeit. Nur AT

Der Zeitzähler wird in der Sekunde 18,2 mal inkrementiert, genauer: 18,2064819336 mal.

Funktionsnummer
Unterfunktion

		Bemerkungen
00h		Zeitzähler auslesen
01h		Zeitzähler setzen
02h		Uhrzeit auslesen
03h		Uhrzeit setzen
04h		Datum auslesen
05h		Datum setzen
06h		Alarmzeit setzen
07h		Alarmzeit löschen

Zum Interrupt 1Bh: Break-Taste

Mit diesem Interrupt wird beim Betätigen der Tastenkombination **Ctrl+Break** wird zunächst nur ein Flag gesetzt. Erst wenn über eine DOS-Funktion Zeichen ein- oder ausgegeben werden, wird das Programm abgebrochen. Der Interrupt 1Bh hat keinen Parameter.

Zum Interrupt 1Ch: Periodischer Interrupt

Der Timer-Baustein ruft den (Hardware-) Interrupt 08h in der Sekunde 18,2mal auf. Am Ende des Interrupts 08h wird der Interrupt 1Ch aufgerufen, dessen Vektor normalerweise auf einen Interrupt-Return-Befehl IRET zeigt, so daß keine Aktion erfolgt. Durch "Verbiegen" des Vektors auf eine eigene Routine könnte man aber z.B. auf dem Bildschirm immer die aktuelle Uhrzeit anzeigen. Der Interrupt 1Ch hat keinen Parameter.

Zum BIOS-Interrupt 1Dh: Zeiger auf Videotabelle

Der Interruptvektor zeigt nicht auf eine ausführbare Routine, sondern auf eine Tabelle, die Informationen über die eingesetzte Video-Karte enthält.

Zum BIOS-Interrupt 1Eh: Zeiger auf Laufwerkstabelle

Der Interruptvektor zeigt ebenfalls nicht auf eine ausführbare Routine, sondern auf eine Tabelle, die Informationen über den eingesetzten Disketten-Controller enthält.

Zum BIOS-Interrupt 1Fh: Zeiger auf Zeichentabelle

Der Interruptvektor zeigt ebenfalls nicht auf eine ausführbare Routine, sondern auf eine Tabelle, die Informationen über die Bitmuster (Bitmap) der Zeichen mit den Code-Nummern > 127 enthält. Die Bitmaps werden vom Befehl GRAFTABL angelegt. Die Bitmaps der Zeichen <= 127 sind dagegen fest im ROM abgelegt.

Zum DOS-Interrupt 20h: Programm beenden

Statt dieses Interrupts sollte man besser die Funktion 4Ch des DOS-Interrupts 21h benutzen, weil diese die Rückgabe eines Exit-Codes an das aufrufende Programm gestattet.

Zum DOS-Interrupt 21h: Allgemeine DOS-Funktionen

DOS-Version 3.xx

Funktionsnummer **DOS-Interrupt 21h**
Unterfunktion

		Bemerkungen (P/D = Platte/Diskette)
00h		Programm beenden. Besser Funktion 4Ch
01h		Zeicheneingabe mit Echo
02h		Ausgabe eines Zeichens
03h		Empfang eines Zeichens von serieller Schnittstelle
04h		Ausgabe eines Zeichens auf serielle Schnittstelle
05h		Ausgabe eines Zeichens auf Drucker
06h		Direkte Zeichenein-/-ausgabe. Ohne Prüfung Ctrl-C
07h		Direkte Zeicheneingabe ohne Echo. Ohne Prüfung Ctrl-C
08h		Zeicheneingabe ohne Echo
09h		Ausgabe einer Zeichenkette
0Ah		Eingabe einer Zeichenkette
0Bh		Eingabestatus lesen
0Ch		Eingabepuff. lösch. u. Eingabefunktion aufrufen (01,06,07,08)
0Dh		Inhalt Blockpuffer auf P/D schreiben
0Eh		Aktuelles Laufwerk definieren
0Fh		Datei öffnen
10h		Datei schließen
11h		Ersten Datei-Eintrag im FCB (File Control Block) suchen
12h		Nächsten Datei-Eintrag im FCB suchen
13h		Datei löschen
14h		Sequentielles Lesen aus Datei
15h		Sequentielles Schreiben in Datei
16h		Neue Datei anlegen und öffnen
17h		Datei umbenennen
19h		Nummer des aktuellen Laufwerks ermitteln (0 = A, 1 = B ...)
1Ah		Verlegung der Disk Transfer Area (DTA)
1Bh		Kenngrößen aktuelles Laufwerk ermitteln (u.a. Typ, Sektoren)
1Ch		Kenngrößen eines bestimmtes Laufwerk ermitteln
21h		Wahlfreies Lesen P/D
22h		Wahlfreies Schreiben P/D
23h		Dateigröße ermitteln
24h		Positionszeige für wahlfreien Zugriff setzen
25h		Interruptvektor auf anderen Wert setzen (Interrupt verbiegen)
26h		Programm-Segment-Präfix (PSP) an andere Adresse kopieren
27h		Wahlfreies Lesen mehrerer Datensätze

28h		Wahlfreies Schreiben mehrerer Datensätze
29h		Dateinamen in File Control Block (FCB) schreiben
2Ah		Datum ermitteln
2Bh		Datum setzen
2Ch		Uhrzeit ermitteln
2Dh		Uhrzeit setzen
2Eh		Verify-Flag bei Schreiben P/D setzen
2Fh		Adresse der Data Transfer Arera (DTA) ermitteln
30h		DOS-Versionsnummer ermitteln
31h		Programm beenden, aber im Speicher belassen
33h	00h	Lesen Break-Flag (ob auf Ctrl-C geprüft werden soll)
33h	01h	Setzen Break-Flag
35h		Interrupt-Adresse ermitteln
36h		Freie Kapazität P/D ermitteln
38h	00h	Landesspezifische Symbole und Formate ermitteln
38h	01h	Landesspezifische Symbole und Formate setzen
39h		Unterverzeichnis erstellen
3Ah		Unterverzeichnis löschen
3Bh		Unterverzeichnis wählen
3Ch		Neue Datei erstellen bzw. vorhandene leeren
3Dh		Datei öffnen
3Eh		Datei schließen
3Fh		Bestimmte Anzahl von Zeichen von Datei lesen
40h		Bestimmte Anzahl von Zeichen in Datei schreiben
41h		Datei löschen
42h		Positionszeiger für wahlfreien Zugriff P/D setzen
43h	00h	Attribut einer Datei ermitteln
43h	01h	Attribut einer Datei setzen
44h	00h	Attribut eines Zeichentreibers ermitteln
44h	01h	Attribut eines Zeichentreibers setzen
44h	02h	Daten von Zeichentreiber empfangen
44h	03h	Daten an Zeichentreiber übergeben
44h	04h	Daten von Blocktreiber empfangen
44h	05h	Daten an Blocktreiber übergeben
44h	06h	Eingabestatus eines Gerätetreibers ermitteln
44h	07h	Ausgabestatus eines Gerätetreibers ermitteln
44h	08h	Test, ob Datenträger gewechselt werden kann (D = j, P = n).
44h	09h	Netzwerk: Test, ob Laufwerk auf anderem Rechner
44h	0Ah	Netzwerk: Test, ob Datei auf anderem Rechner
44h	0Bh	Netzwerk: Anzahl der Zugriffswiederholungen setzen
45h		Kanal duplizieren
46h		Kanal auf zweiten Kanal kopieren
47h		Pfadnamen des aktuellen Verzeichnisses ermitteln
48h		RAM-Speicher reservieren
49h		RAM-Speicher freigeben
4Ah		Größe des reservierten RAM-Speichers ändern
4Bh	00h	Anderes Programm laden und ausführen, dann zurück
4Bh	03h	Anderes Programm laden ohne Ausführung (Overlay)
4Ch		Programm mit Exit-Code (im Register AL) beenden
4Dh		Exit-Code eines anderen Programms ermitteln
4Eh		Ersten Eintrag einer Datei im Verzeichnis suchen
4Fh		Nächsten Eintrag einer Datei im Verzeichnis suchen
54h		Verify-Flag lesen
56h		Datei umbenennen
57h	00h	Datum und Uhrzeit der letzten Datei-Modifikation ermitteln
57h	01H	Datum und Uhrzeit der letzten Datei-Modifikation setzen
58h	00h	Strategie für Speicherzuteilung ermitteln
58h	01h	Strategie für Speicherzuteilung setzen
59h		Erweiterte Fehlercodes ermitteln
5Ah		Temporäre Datei anlegen
62h		Adresse des Programmsegment-Prefixs (PSP) ermitteln

Bei allen Aufrufen des Interrupts 21h ist die Funktionsnummer in das Register AH zu schreiben.

Zur Funktion 02h des DOS-Interrupts 21h, Ausgabe eines Zeichens:

Eingabe: AH = 02h
DL = (ASCII-) Code des Zeichens
Ausgabe: keine

Die Bildschirmausgabe erfolgt an der aktuellen Cursorstelle mit dem alten Attribut. Der Cursor wird versetzt. Steuerzeichen werden bei der Bildschirmausgabe interpretiert.

Zur Funktion 4Ch des DOS-Interrupts 21h, Programm mit EXIT-Code beenden:

Eingabe: AH = 4Ch
AL = Exit-Code nach eigener Vergabe (00h..ffh)
Ausgabe: keine

Diese Funktion sollte vorzugsweise für die Beendigung von Programmen eingesetzt werden. Auf den optionale Exit-Code kann in Batch-Programmen mit ERRORLEVEL zugegriffen werden, siehe Kap. 30. Üblicherweise setzt man den EXIT-Code bei fehlerfreier Programmbeendigung auf null, sonst auf einen anderen Wert.

Zum Interrupt 22h: Routine zur Programm-Beendigung

Diese Routine darf nicht direkt aufgerufen werden. Sie wird von allen anderen Interrupts zur Beenden eines Programms automatisch aufgerufen (Interrupt 20h, Funktionen 00h, 21h und 4Ch des Interrupts 21h).

Zum Interrupt 23h: Break-Taste betätigt

Die Routine wird aufgerufen, wenn die Break-Taste oder Ctrl+C betätigt wird. Sie darf aber nicht direkt aufgerufen werden.

Zum Interrupt 24h: Kritischer Fehler

Die Routine wird aufgerufen, wenn bei einem Hardware-Zugriff ein kritischer Fehler entdeckt wird. Sie darf nicht direkt aufgerufen werden.

Zum Interrupt 25h: Absolutes Lesen

Mit diesem Interrupt können logisch aufeinanderfolgende Sektoren von Disketten/Platten eingelesen werden. Als Eingabeparameter braucht der Interrupt u.a. die Laufwerksnummer, die Nummer des ersten zu lesenden Sektors und die Anzahl der Sektoren.

Zum DOS-Interrupt 26h: Absolutes Schreiben

Analog wie 26h, lediglich Schreiben statt Lesen.

Zum Interrupt 27h: Programmende ohne Speicherfreigabe

Bei diesem Interrupt verbleibt das Programm nach dem Ende resident im Speicher verfügbar (z.B. Gerätetreiber). Nur bei COM-Programmen möglich. Besser Funktion 31h des DOS-Interrupts 21h benutzen.

Zum Maus-Interrupt 33h: Maus

In einem späteren Demo-Programm werden einige Funktionen des Maus-Interrupts behandelt. Für den Maus-Interrupt muß in der Datei Config.SYS mit device der Maustreiber aufgeführt sein. Beispiel für Eintrag in Datei **Config.SYS**:

. . .

```
device = C:\MAUS\Mouse.SYS
...
```

Funktions-Nr des Maus-Interrupts 33h und Kurzerklärung

00h	Initialisierung und Reset des Maustreibers
01h	Mauscursor anzeigen
02h	Mauscursor ausblenden
03h	Mausposition und Status der Mausknöpfe ermitteln
04h	Mauscursor positionieren
05h	Wie oft wurde ein Mausknopf gedrückt?
06h	Wie oft wurde ein Mausknopf losgelassen?
07h	Horizontale Größe des Maus-Fensters festlegen
08h	Vertikale Größe des Maus-Fensters festlegen
09h	Gestalt des Maus-Cursors im Graphik-Mode festlegen
0Ah	Gestalt des Maus-Cursors im Text-Mode festlegen
0Bh	Entfernung der aktuellen Maus-Position von der letzten
0Ch	Zusätzliche Benutzer-Interruptroutine installieren
0Dh	Emulation des Lichgriffels anschalten
0Eh	Emulation des Lichtgriffels abschalten
0Fh	Maus-Empfindlichkeit einstellen
10h	Ausschlußbereich für Maus festlegen
12h	Gestalt des großen Graphik-Mauscursors festlegen
13h	Schwelle für Verdoppelung der Mausgeschwindigkeit festlegen
14h	Austauschen der Benutzer-Interruptroutine
15h	Größe des Maus-Statuspuffers ermitteln
16h	Maus-Status sichern
17h	Maus-Status restaurieren
18h	Alternative Benutzer-Interruptroutine installieren
19h	Adresse der alternativen Benutzer-Interruptroutine ermitteln
1Ah	Maus-Empfindlichkeit einstellen. Kombination von 0Fh und 13h
1Bh	Maus-Empfindlichkeit ermitteln
1Ch	Häufigkeit der Maus-Abfrage einstellen (0..200/sec)
1Dh	Bildschirmseite für Maus-Cursor setzen
1Eh	Bildschirmseite für Maus-Cursor ermitteln
1Fh	Maustreiber deakivieren
20h	Maustreiber wieder aktivieren
21h	Reset des Maustreibers
24h	Maustyp ermitteln (Bus, seriell, Inport, PS/2, ...)

Zum EMS-Interrupt 67h: EMS-Speicher

EMS = Expanded Memory System. Ergänzungspeicher nach Spezifikation der Firmen Lotus, Intel und Microsoft (LIM-Standard). Neben EMS gibt es noch den erweiterten EMS (EEMS) der Firmen Ast, Quadram und Aston Tate. Die Software-Schnittstelle zu EMS hat die Bezeichnung EMM (Expanded Memory Manager).

Expanded Memory (Ergänzungsspeicher) darf nicht mit Extended Memory (Erweiterungsspeicher) verwechselt werden. Letzter liegt im Adreßbereich über 1 MByte und ist nur von den Prozessoren Intel 80286, 80386 und höheren im Protected Mode erreichbar, was aber im normalen DOS-Betrieb nicht möglich ist. Der Erweiterungsspeicher kann aber unter gewissen Voraussetzungen für RAM-Disk, Platten-Cache, Emulation von Ergänzungsspeicher genutzt werden. Details siehe Kap. Betriebssystem MS-DOS.

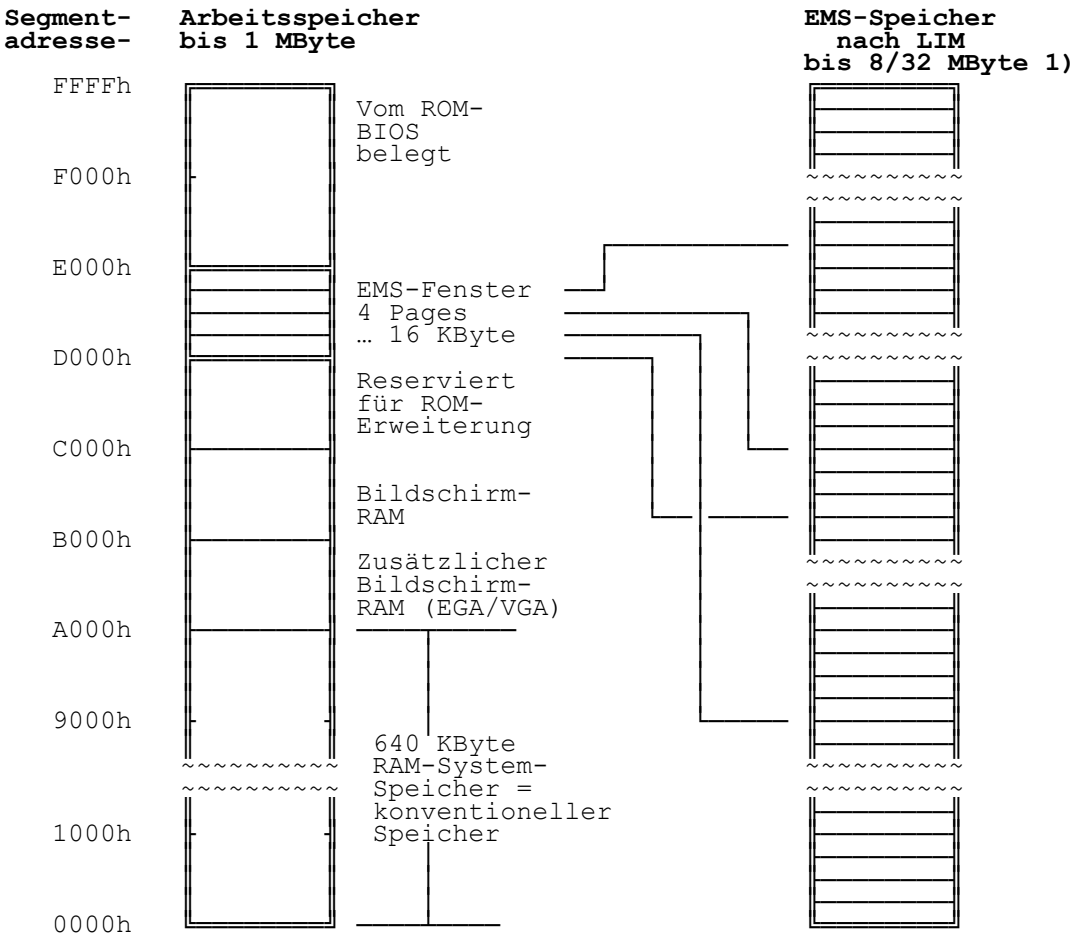
Mit EMS bzw. EMM können nacheinander Speichersegmente (d.h. jeweils 64 KByte groß) aus dem bis zu 8 MByte großen Ergänzungsspeicher in einen freien Adreßbereich zwischen 640 KByte und 1 MByte eingeblendet werden, der normaler für DOS reserviert ist. Da die physischen Adressen somit unter 1 MByte liegen, kann EMS somit auf allen MS-DOS-Rechnern eingesetzt werden. Voraussetzung ist ist das Einbinden des EMM-Treibers in die Datei Config.SYS mit "device = ..." und natürlich Software, die den Interrupt 67h auch verwendet.

Das Einblenden geschieht nicht durch Kopieren des Segmentinhaltes des EMS-Speichers in den Speicherbereich unter 1 MByte, sondern durch Umprogrammieren der Adreßleitungen mittels Zusatz-Hardware, die sich entweder auf der EMS-Karte oder bereits auf der Hauptplatine befindet. Der Vorgang wird auch mit *Bank Switching* oder *Memory Mapping* bezeichnet.

Gegenüber früheren Bank-Switching-Verfahren ist EMS noch verbessert: Es muß nicht immer ein ganzes Segment (64 KByte) umgeschaltet werden, sondern Pages (Seiten) mit 16 KByte können einzeln umgeschaltet werden. Die einzelnen Seiten können beliebig weit voneinander entfernt liegen.

Das Problem besteht zuerst darin, im Speicherbereich zwischen 640 KByte und 1 MByte einen nicht benutzten Adreßbereich von 64 KByte Größe für EMS zu finden. Der Bereich zwischen 640 KByte und 1 MByte ist zwar für normale DOS-Anwendungen reserviert, aber glücklicherweise nicht vollständig vergeben. Üblicherweise ist das Segment mit der Adresse D000h, das ursprünglich für ROM-Cartridges vorgesehen war, nicht belegt und kann somit für EMS benutzt werden.

Das folgende Schema zeigt den Speicheraufbau und das Einblenden von EMS:



1) Ab LIM 4.0: EMS-Speicher bis 32 MByte

Der EMS-Interrupt 67h besitzt eine größere Anzahl von Funktionen. Auf die eingangs erwähnte Spezialliteratur wird verwiesen.

27.8 Beispiel: Zeichenausgabe über Pascal, DOS, BIOS und Hardware

Das folgende Demo-Programm gibt den ASCII-Zeichensatz aus mit vier verschiedenen Methoden:

- Pascal
- Aufruf DOS-Interrupt h21, Funktion h02
- Aufruf BIOS-Interrupt h10, Funktionen h02 und h09
- Schreiben in Bildschirmspeicher

```
program Pas27081;  { Kap. 27.8: Ascii-Satz in Pascal, DOS, BIOS
                                und Hardware                    }
                   { K. Haller }
uses
  CRT, DOS;

var
  Reg: Registers;
     { Registers ist ein in der Unit DOS definierter (varianter)
       Recordtyp mit den Feldern
       •  AX, BX, CX, DX, BP, DI, DS, ES, Flags (16-Bit-Register)
       •  AL, AH, BL, BH, CL, CH, DL, DH        (8-Bit-Register )  }

procedure ReturnTaste;
begin
  GotoXY(10, 25); ClrEoL;
  GotoXY(10, 25);
  Write('Weiter mit Taste Return: ');
  repeat
  until ReadKey = #13;
end;

procedure DOS_Version;
begin
  ClrScr;
  GotoXY(10, 2);
  WriteLn('DOS-Version mit Funktion "DosVersion" aus Unit DOS');
  GotoXY(10, 5);
  WriteLn('Die DOS-Version: ', Lo(DosVersion), '.', Hi(DosVersion));
  ReturnTaste;
end;

procedure AsciiZeichen_mit_Pascal;
var
 i: Byte;

begin
  ClrScr;
  GotoXY(10, 2);
  WriteLn('Ascii-Zeichen mit Pascal');
  for i := 0 to 255 do
    begin
      GotoXY(10 + i mod 64, 5 + i div 64);
      Write(Char(i));
```

```
    end;
  ReturnTaste;
end;

procedure AsciiZeichen_mit_DOS_Interrupt_h21_Funktion_h02;
var
  i: Byte;
begin
  ClrScr;
  GotoXY(10, 2);
  WriteLn('Ascii-Zeichen mit DOS-Interrupt h21, Funktion h02');

  for i := 0 to 255 do
    begin
      Reg.AH := $02;
      Reg.DL := i;
      GotoXY(10 + i mod 64, 5 + i div 64);
      MsDOS(Reg);  { Hat gleiche Wirkung wie "Intr($21, Reg)" }
      { Steuerzeichen BEL, BS, CR und LF werden interpretiert }
    end;
  ReturnTaste;
end;

procedure AsciiZeichen_mit_BIOS_Interrupt_h10_Funktion_h09;
   { Der Interrupt-Aufruf h10, Funktion h09: Zeichen ausgeben
     Steuerzeichen werden nicht interpretiert. Cursor wird
     nicht versetzt. Der Aufruf verändert u.a. Register AX,
     wogegen die Register BX und CX nicht verändert werden }

const
  Wiederholungen = 4;

var
  i: Byte;

begin
  ClrScr;
  GotoXY(10, 2);
  WriteLn('Ascii-Zeichen mit BIOS-Interrupt h10, Funktion h09');

  Reg.BH := 0;            { Die Bildschirmseite, 0 = Standard   }
  Reg.CX := Wiederholungen;  { Anzahl der Zeichenwiederholungen }

  for i := 0 to 127 do
    begin
      Reg.AL := i;    { Der Ascii-Code des auszugebenden Zeichens }
      Reg.BL := i;    { Das Zeichen-Attribut. Für Demo jedes
                        Zeichen mit anderem Attribut darstellen   }

      Reg.AH := $09;  { Die Funktion h09 }

      GotoXY(10 + i*Wiederholungen mod 64,
             5 + i*Wiederholungen div 64);
      { Cursor "von Hand" versetzen }

      Intr($10, Reg); { Der Interrupt-Aufruf }
    end;
  ReturnTaste;
end;
```

```
procedure AsciiZeichen_mit_BIOS_Interrupt_h10_Funktionen_h02_h09;
   { Der Interrupt-Aufruf h10, Funktion h02: Cursor positionieren
     und Kombination mit Funktion h09: Zeichen ausgeben
     Steuerzeichen werden  n i c h t  interpretiert.
     Die Aufrufe verändern u.a. Register AX,
     wogegen die Register BX und CX nicht verändert werden }
var
  i: Byte;
begin
  ClrScr;
  GotoXY(10, 2);
  WriteLn('Ascii-Zeichen mit BIOS-Interrupt h10');
  GotoXY(10, 3); WriteLn('Funktion h02: Cursor positioniern und ');
  GotoXY(10, 4); WriteLn('Funktion h09: Zeichen ausgeben');

  Reg.BH := 0;    { Die Bildschirmseite, 0 = Standard   }
  Reg.CX := 1;    { Anzahl der Zeichenwiederholungen    }

  for i := 0 to 255 do
    begin
      Reg.AH := $02; { Funktion h02: Cursor postionieren }
      Reg.DL :=  5 + i*Reg.CX mod 64; { Bildschirmspalte }
      Reg.DH :=  8 + i*Reg.CX div 64; { Bildschirmzeile  }

      Intr($10, Reg); { Cursor positionieren }

      { --- und jetzt Zeichen ausgeben: --------- }
      Reg.AL := i;    { Der Ascii-Code des auszugebenden Zeichens }
      Reg.BL := i;    { Das Zeichen-Attribut. Für Demo jedes
                        Zeichen mit anderem Attribut darstellen   }

      Reg.AH := $09;  { Die Funktion h09 }

      Intr($10, Reg); { Der Interrupt-Aufruf }
    end;

  ReturnTaste;
end;

procedure AsciiZeichen_in_Bildschirmspeicher;
const
  BildschirmSegment = $b800;  { Farbe: $b800, Mono: $b000 }
  { Segmentadresse des Bildschirmspeichers }
  Startzeile = 5;
var
  i:      Byte;
  Offset: Word;

begin
  ClrScr;
  GotoXY(10, 2);
  WriteLn('Ascii-Zeichen mit Attribut in Bildschirmspeicher');
  GotoXY(10, 3);
  WriteLn('Mit vordefin. Array-Variablen "Mem[Segment:Offset]" ');

  for i := 0 to 255 do
    begin
```

```
      Offset := Startzeile * 80 * 2  + 2*i;
      Mem[BildschirmSegment:Offset    ] := i; { Zeichen  }
      Mem[BildschirmSegment:Offset + 1] := i; { Attribut }
    end;

  ReturnTaste;
end;

procedure Hardcopy;
   { Der Interrupt-Aufruf für Hardcopy: h05 (Hardware-Interrupt) }
var
  Ch: Char;

begin
  GotoXY(10, 25);
  Write('Hardcopy mit Hardware-Interrupt h05 (j/n): n');
  GotoXY(WhereX - 1, WhereY);
  repeat
    Ch := UpCase(Readkey);
    if Ch = #13 then Ch := 'N';
  until (Ch = 'J') or (Ch = 'N');
  Write(Ch);

  if Ch = 'J'
     then Intr($05, Reg);
                { »Reg« wird hier nur wegen der Syntax gebraucht }

  ReturnTaste;
end;

begin
  DOS_Version;
  AsciiZeichen_mit_Pascal;
  AsciiZeichen_mit_DOS_Interrupt_h21_Funktion_h02;
  AsciiZeichen_mit_BIOS_Interrupt_h10_Funktion_h09;
  AsciiZeichen_mit_BIOS_Interrupt_h10_Funktionen_h02_h09;
  AsciiZeichen_in_Bildschirmspeicher;
  Hardcopy;
end.
```

27.9 Weitere Demo-Programme A

```
program Pas27091; { **** Umschalttasten abfragen ****************** }
                  {  Turbo-Pascal,                         K. Haller }

        { In diesem Programm werden die Tastatur-Statusbytes
          $0040:$0017 und $0040:$0018 durch Zugriff mit
          Mem[segment:offset] gelesen und auch überschrieben.

          Das Lesen des ersten Tastatur-Statusbyte wäre auch
          mit der Funktion $02 des BIOS-Interrupts $16 möglich.
          Das Ergebnis wird im Register AL geliefert. }

uses
  CRT;
```

```
const
  Bit0 =  1; Bit1 =  2; Bit2 =  4; Bit3 =   8;
  Bit4 = 16; Bit5 = 32; Bit6 = 64; Bit7 = 128;    { Bit-Wertigkeiten }
          {  oder z.B.  Bit6 = 1 shl 6  }

var
  ByteH17,
  ByteH18:    Byte;
  BinaerH17,
  BinaerH18:  string[9];
  Taste:      Char;
  T17, T18:   string;

function Dez_BinStr(Dez: Word): string; { ---------------------- }
const
  BasisBin = 2;
  Blank    = ' ';
  Bitmuster_in_Viererbloecken = True;
     { ggf. ändern »Tue« <---> »False«  }

var
  BinaerString:    string[19];
  BinStringLaenge: Byte;
  Dez_Temp:        Word;

begin
  BinaerString := '';
  Dez_Temp     := Dez;

  while Dez_Temp <> 0 do
    begin
      if (Dez_Temp mod BasisBin) = 0
         then  BinaerString := '0' + BinaerString
         else  BinaerString := '1' + BinaerString;
      Dez_Temp := Dez_Temp div BasisBin;
    end;

  if Dez <= 255
     then BinStringLaenge :=  8
     else BinStringLaenge := 16;

  while Length(BinaerString) < BinStringLaenge do
    BinaerString := '0' + BinaerString;

  if Bitmuster_in_Viererbloecken
     then if Dez <= 255
             then Insert(Blank, BinaerString, 5)
             else begin
                    Insert(Blank, BinaerString,  5);
                    Insert(Blank, BinaerString, 10);
                    Insert(Blank, BinaerString, 15);
                  end;

  Dez_BinStr := BinaerString;
end;  { ------------------------------------------------------------ }

begin
  ClrScr;
```

```
GotoXY(10, 5); Write('Man achte auf die Leuchtdioden. Weiter ... ');
{ Bei den Bit-Manipulationen ist darauf zu achten, daß nur das
  Einzel-Bit verändert wird, die anderen Bits dürfen nicht verändert
  werden!     }
repeat
  Mem[$0040:$0017] := Mem[$0040:$0017] xor Bit4;
                      { Bit 4 invertieren, Operator xor }
  Mem[$0040:$0017] := Mem[$0040:$0017] xor Bit5;
                      { Bit 5 invertieren, Operator xor }
  Mem[$0040:$0017] := Mem[$0040:$0017] xor Bit6;
                      { Bit 6 invertieren, Operator xor }
  Delay(100);  { 100 ms warten }
until KeyPressed;

Mem[$0040:$0017] := Mem[$0040:$0017] and not Bit4;
                    { Bit 4 löschen,   Operatoren and not }
Mem[$0040:$0017] := Mem[$0040:$0017] or  Bit5;
                    { Bit 5 setzen,    Operator or      }
Mem[$0040:$0017] := Mem[$0040:$0017] and not Bit6;
                    { Bit 6 löschen,   Operatoren and not }

ClrScr;
GotoXY(10, 5);
Write('Umschalttasten abfragen, Ende mit »Ctrl-Break«  (kha)');

TextColor(Green);
GotoXY(17,  8); Write('Bit-Nr  7654 3210');
GotoXY(17,  9); Write('        |||| ||||');
GotoXY(10, 10); Write('Byte H17: ');
GotoXY(17, 11); Write('        |||| ||||');
GotoXY(10, 12); Write('Byte H18: ');

repeat
  ByteH17   := Mem[$0040:$0017];
  ByteH18   := Mem[$0040:$0018];

  BinaerH17 := Dez_BinStr(ByteH17);
  BinaerH18 := Dez_BinStr(ByteH18);

  GotoXY(21, 10); TextColor(Green);  Write(ByteH17:3);
                  TextColor(Yellow); Write(BinaerH17:10); ClrEoL;
  GotoXY(21, 12); TextColor(Green);  Write(ByteH18:3);
                  TextColor(Yellow); Write(BinaerH18:10); ClrEoL;
                  TextColor(Green);

  { Jetzt wird getestet, ob ein bestimmtes Bit gesetzt ist }
  T17 := ''; { ---- Meldung T17 für Byte $17 -------------------- }
  if ByteH17 and Bit0 = Bit0
     then T17 := T17 + ' Shift-R ';      { Shift-R   }
  if ByteH17 and Bit1 = Bit1
     then T17 := T17 + ' Shift-L ';      { Shift-L   }
  if ByteH17 and Bit2 = Bit2
     then T17 := T17 + ' Ctrl ';
  if ByteH17 and Bit3 = Bit3
     then T17 := T17 + ' Alt ';
  if ByteH17 and Bit4 = Bit4
     then T17 := T17 + ' Scroll ';       { On/Off }
```

```
    if ByteH17 and Bit5 = Bit5
       then T17 := T17 + ' NumLock ';      { On/Off }
    if ByteH17 and Bit6 = Bit6
       then T17 := T17 + ' CapsLock ';     { On/Off }
    if ByteH17 and Bit7 = Bit7
       then T17 := T17 + ' Insert-Mode ';  { On/Off }

    T18 := '';  { ---- Meldung T18 für Byte $18 -------------------- }
    if ByteH18 and Bit0 = Bit0
       then T18 := T18 + ' Ctrl-L ';       { Ctrl-Links     }
    if ByteH18 and Bit1 = Bit1
       then T18 := T18 + ' Alt-L ';        { Alt-Links      }
    if ByteH18 and Bit2 = Bit2
       then T18 := T18 + ' SysReq ';       { System-Abfrage }
    if ByteH18 and Bit3 = Bit3
       then T18 := T18 + ' Halt ';         { Ctrl-NL/Paus   }
    if ByteH18 and Bit4 = Bit4
       then T18 := T18 + ' Scroll ';       { während Tastendruck }
    if ByteH18 and Bit5 = Bit5
       then T18 := T18 + ' NumLock ';      { während Tastendruck }
    if ByteH18 and Bit6 = Bit6
       then T18 := T18 + ' CapsLock ';     { während Tastendruck }
    if ByteH18 and Bit7 = Bit7
       then T18 := T18 + ' Insert ';       { während Tastendruck }

    GotoXY(35, 10); Write(T17); ClrEol;
    GotoXY(35, 12); Write(T18); ClrEol;

    Delay(100);

  until False;  { Endlosschleife. Abbruch nur mit »Ctrl+Break« }
end.
```

```
program Pas27092; { BIOS-Interrupt h11: Konfiguration abfragen    }
                  { Turbo-Pascal, K. Haller    }
  {
   Nach dem Aufruf von Interrupt h11 wird im Register AX die
   Konfiguration abgelegt. Die Bedeutung der einzelnen Bits
   beim AT und PS/2 (bei einfachen PCs z.T. anders):

   <──────── Bit-Nr ──────────────  | Bit 0..7: AL,  Bit 8..15: AH
   1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 |
   5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 | Bedeutung bei AT/PS-2
   ═══════════════════════════════ ═══════════════════════════════
   x x . . . . . . . . . . . . . . | Anzahl der parallelen Drucker
   . . x . . . . . . . . . . . . . | Reserviert
   . . . x . . . . . . . . . . . . | Reserviert
   . . . . x x x . . . . . . . . . | Anzahl serielle Schnittstellen
   . . . . . . . x . . . . . . . . | Nicht verwendet
   . . . . . . . . x x . . . . . . | Anzahl Diskettenlaufwerke - 1
                                   |    0 0:  1 Diskettenlaufwerk
                                   |    0 1:  2 Diskettenlaufwerke
                                   |    1 0:  3 Diskettenlaufwerke
                                   |    1 1:  4 Diskettenlaufwerke
   . . . . . . . . . . x x . . . . | Bildschirmmodus beim Booten
                                   |    0 0:  Nicht verwendet
                                   |    1 0:  Color,     80 * 25
                                   |    0 1:  Color,     40 * 25
                                   |    1 1:  Monochrom, 80 * 25
   . . . . . . . . . . . . x . . . | Nicht verwendet
```

```
   . . . . . . . . . . . . . . x . .  | Zeigegerät (Maus) installiert
   . . . . . . . . . . . . . . . x .  | Coprozessor installiert
   . . . . . . . . . . . . . . . . x  | Diskettenlaufwerk(e) vorhanden
   ================================== ================================
   5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0    |
   1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0    |

   Anmerkung: Der Bildschirmmodus kann nach dem Booten verändert
              worden sein. Der aktuelle Modus kann nicht mit dem
              Interrupt h11 abgefragt werden, sondern muß mit der
              Funktion h0F des Interupts h10 ausgelesen werden.
              Das Ergebnis steht dann im Register AL.
  }
uses
  CRT, DOS;    { Unit DOS für Interrupt-Aufruf }

type
  String16 = string[16];

var
  Reg: Registers;  { "Registers": Record-Typ aus Unit DOS }
  AX:  Word;
  Serielle_Schnittstellen:  Byte;  { RS-232, V.24 (COM1:, COM2: usw) }
  Parallele_Schnittstellen: Byte;  { paralleler Drucker (Centronics) }

function Bit(Register: Word; i: Byte): Boolean;
begin  { Testet ob Bit i gesetzt }
  if Register and (1 shl i) = 1 shl i
     then Bit := True
     else Bit := False
end;

function BitMuster16(Register: Word): String16;
var
  i: 0..15;
  s: String16;
begin
  for i := 0 to 15 do
    if Register and (1 shl 15 shr i) = (1 shl 15 shr i)
       then s[i + 1] := '1'
       else s[i + 1] := '0';
  BitMuster16 := s;
end;

begin          { Die folgende Interpretation gilt für AT-Rechner    }
               { und höhere,  n i c h t  aber für normale PC und XT }
  ClrScr;

  {                                                                  }
  {  }  Intr($11, Reg);  {  Interrupt h11: Konfiguration abfragen    }
  {                                                                  }

  AX := Reg.AX;

  WriteLn('---- Interrupt h11: Konfiguration -----');
  WriteLn('Register AX in dez: ', AX);
  WriteLn;
  WriteLn('Bit-Nr:        5432109876543210');
  WriteLn('Das Bitmuster: ', BitMuster16(AX));
  WriteLn('--------------------------------------');
```

```
  if Bit(AX, 0)
     then WriteLn('Ein oder mehrere Diskettenlaufwerke')
     else WriteLn('Kein Diskettenlaufwerk vorhanden');

  if Bit(AX, 1)
     then WriteLn('Coprozessor vorhanden')
     else WriteLn('Kein Coprozessor vorhanden');

  if Bit(AX, 2)
     then WriteLn('Zeigegerät (Maus) installiert')
     else WriteLn('Kein Zeigegerät (Maus) installiert');

  if Bit(AX, 0) then
     begin
       Write('a) Anzahl der Diskettenlaufwerke:     ');
       if not Bit(AX, 7) and not Bit(AX, 6)  then WriteLn('1');
       if not Bit(AX, 7) and     Bit(AX, 6)  then WriteLn('2');
       if     Bit(AX, 7) and not Bit(AX, 6)  then WriteLn('3');
       if     Bit(AX, 7) and     Bit(AX, 6)  then WriteLn('4');
       { --------- Eleganter mit Shift-Operatoren: --------- }
       Write('b) Anzahl der Diskettenlaufwerke:     ');
       WriteLn(AX shl 8 shr 14  + 1);
     end;

  Serielle_Schnittstellen := 0;
  if Bit(AX,  9) then Inc(Serielle_Schnittstellen, 1);
  if Bit(AX, 10) then Inc(Serielle_Schnittstellen, 2);
  if Bit(AX, 11) then Inc(Serielle_Schnittstellen, 4);
  WriteLn('Anzahl der seriellen Schnittstellen:  ',
           Serielle_Schnittstellen);

  Parallele_Schnittstellen := 0;
  if Bit(AX, 14) then Inc(Parallele_Schnittstellen, 1);
  if Bit(AX, 15) then Inc(Parallele_Schnittstellen, 2);
  WriteLn('Anzahl der parallelen Schnittstellen: ',
           Parallele_Schnittstellen);

  repeat
  until ReadKey <> '';
end.
```

```
program Pas27093;  { Maus-Interrupt 33h.  Demo: Maus im Text-Mode }
                   { Turbo-Pascal 5.0     7002119       K. Haller }
 {   In der Datei "Config.SYS" muß mit "device" der (Microsoft-)
   Maustreiber"Mouse.SYS" mit seinem Zugriffspfad aufgeführt sein.
   Beispiel:       device = C:\Maus\Mouse.SYS

   Funktions-Nr des Maus-Interrupts 33h und Kurzerklärung
   ──────────────────────────────────────────────────────────────
   00h  Initialisierung und Reset des Maustreibers
   01h  Mauscursor anzeigen
   02h  Mauscursor ausblenden
   03h  Mausposition und Status der Mausknöpfe ermitteln
   04h  Mauscursor positionieren
   05h  Wie oft wurde ein Mausknopf gedrückt?
```

```
   06h  Wie oft wurde ein Mausknopf losgelassen?
   07h  Horizontale Größe des Maus-Fensters festlegen
   08h  Vertikale Größe des Maus-Fensters festlegen
   09h  Gestalt des Maus-Cursors im Graphik-Mode festlegen
   0Ah  Gestalt des Maus-Cursors im Text-Mode festlegen
   0Bh  Entfernung der aktuellen Maus-Position von der letzten
   0Ch  Zusätzliche Benutzer-Interruptroutine installieren
   0Dh  Emulation des Lichgriffels anschalten
   0Eh  Emulation des Lichtgriffels abschalten
   0Fh  Maus-Empfindlichkeit einstellen
   10h  Ausschlußbereich für Maus festlegen
   12h  Gestalt des großen Graphik-Mauscursors festlegen
   13h  Schwelle für Verdoppelung der Mausgeschwindigkeit festlegen
   14h  Austauschen der Benutzter-Interruptroutine
   15h  Größe des Maus-Statuspuffers ermitteln
   16h  Maus-Status sichern
   17h  Maus-Status restaurieren
   18h  Alternative Benutzer-Interruptroutine installieren
   19h  Adresse der alternativen Benutzer-Interruptroutine ermitteln
   1Ah  Maus-Empfindlichkeit einstellen. Kombination von 0Fh und 13h
   1Bh  Maus-Empfindlichkeit ermitteln
   1Ch  Häufigkeit der Maus-Abfrage einstellen (0..200/sec)
   1Dh  Bildschirmseite für Maus-Cursor setzen
   1Eh  Bildschirmseite für Maus-Cursor ermitteln
   1Fh  Maustreiber deakivieren
   20h  Maustreiber wieder aktivieren
   21h  Reset des Maustreibers
   24h  Maustyp ermitteln (Bus, seriell, Inport, PS/2, ...)
  ------------------------------------------------------------------
 }

uses
  CRT, DOS;

var
  Reg:           Registers;  { Recordtyp aus Unit DOS }
  Spalte, Zeile: Byte;

procedure WriteXY(Spalte, Zeile: Byte; Meldung: string);
begin
  GotoXY(Spalte, Zeile);
  Write(Meldung);
end;

procedure Taste;
begin
  WriteXY(9, 25, 'Weiter mit Tastendruck ... ');
  Write(ReadKey);
end;

procedure Maustreiber_initialisieren_und_Reset;
begin
  ClrScr;
  WriteXY(9, 3, 'Maustreiber initialisieren. Funktion 00h ');
  Reg.AX := $0000; Intr($33, Reg);
  if Reg.AX = $ffff
```

```
     then begin
             WriteXY(9, 4, 'Maustreiber ist installiert. ' +
                           'Die Maus hat ');
             Write(Reg.BX, ' Knöpfe ');
          end
     else WriteXY(9, 4, 'Kein Maustreiber installiert');
  Taste;
end;

procedure Mauscursor_anzeigen;
begin
  ClrScr;
  WriteXY(9, 3, 'Mauscursor anzeigen. Funktion 01h. ' +
                'Maus bewegen ... ');
  Reg.AX := $0001; Intr($33, Reg);
  Taste;
end;


procedure Mauscursor_ausblenden;
begin
  ClrScr;
  WriteXY(9, 3, 'Mauscursor ausblenden. Funktion 02h');
  Reg.AX := $0002; Intr($33, Reg);
  Taste;
end;

procedure Mausposition_und_Mausknoepfe;
const
  KlickSpalte = 49;
  KlickZeile  =  5;

begin
  ClrScr;
  WriteXY(9, 3, 'Mausposition und Mausstatus. Funktion 03h ');
  WriteXY(9, 4, 'Mausknöpfe links, rechts, beide. Auch ziehen ... ');
  WriteXY(9, 5, 'Ende: Mausknopf links auf diesem Punkt: ');
  WriteXY(KlickSpalte, KlickZeile, '•');

  repeat
    repeat
      GotoXY(3, 7); ClrEoL;
      GotoXY(3, 8); ClrEoL;
      GotoXY(3, 9); ClrEoL;

      Reg.AX := $0003; Intr($33, Reg);

      Spalte := 1 + Reg.CX div  8;  { Divisor 8 für Text- }
      Zeile  := 1 + Reg.DX div  8;  { bildschirm 25 * 80  }

      if (Reg.BX and 1) = 1 then
          begin
            WriteXY(9, 7, 'Bit 0: Mausknopf links ');
            Write('   Spalte: ', Spalte:2, '   Zeile: ',  Zeile);
          end;

      if (Reg.BX and 2) = 2 then
          begin
```

```
            WriteXY(9, 8, 'Bit 1: Mausknopf rechts');
            Write('   Spalte: ', Spalte:2, '   Zeile: ',  Zeile);
          end;

      if (Reg.BX and 4) = 4 then
          begin
            WriteXY(9, 9, 'Bit 2: Mausknopf mitte ');
            Write('   Spalte: ', Spalte:2, '   Zeile: ',  Zeile);
          end;

      until (Reg.BX and 1 = 1) or (Reg.BX and 2 = 2) or
            (Reg.BX and 4 = 4);

     Delay(100);  { Sonst Textanzeige auf Bildschirm zu langsam }
  until (Spalte = KlickSpalte) and
        (Zeile  = KlickZeile ) and ( (Reg.BX and 1) = 1);
  Taste;
end;


procedure Mauscursor_positionieren;
const
  Spalte = 40;
  Zeile  =  5;
  Mausspalte = (Spalte - 1) * 8;   { Multiplikator 8 für    }
  Mauszeile  = (Zeile  - 1) * 8;   { Textbildschirm 25 * 80 }
begin
  ClrScr;
  WriteXY(9, 3, 'Mauscursor positionieren. Funktion 04h ');
  WriteXY(9, 4, 'Der Mauscursor müßte auf dem Punkt stehen. ' +
                'Bewegen ... ');
  WriteXY(Spalte, Zeile, '•');
  Reg.CX := Mausspalte;
  Reg.DX := Mauszeile;
  Reg.AX := $0004; Intr($33, Reg);
  Taste;
end;

procedure Mausfenster_und_malen;
const
  Dauer = 5; { Diese Zeit in Sekunden warten. Nur für Demo }
  SpMin = 9;   SpMax = 60;
  ZeMin = 9;   ZeMax = 20;
  MausspalteMin = (SpMin - 1) * 8;   { Multiplikator 8 für    }
  MausspalteMax = (SpMax - 1) * 8;   { Textbildschirm 25 * 80 }
  MauszeileMin  = (ZeMin - 1) * 8;
  MauszeileMax  = (ZeMax - 1) * 8;
var
  i, j: Byte;

begin
  ClrScr;
  for i := ZeMin to ZeMax do
    for j := SpMin to SpMax do
      WriteXY(j, i, '·');

  WriteXY(9, 3, 'Mausfenster definieren. Funktionen 07h und 08h ');
```

```
  WriteXY(9, 4, 'Ende nach ');
  Write(Dauer, ' sec. Beide Maustasten. Maus bewegen ... ');
  Reg.CX := MausspalteMin;
  Reg.DX := MausspalteMax;
  Reg.AX := $0007; Intr($33, Reg);
  Reg.CX := MauszeileMin;
  Reg.DX := MauszeileMax;
  Reg.AX := $0008; Intr($33, Reg);
  Delay(Dauer * 1000);    { Umrechnung in ms }
  WriteXY(9, 4, #7 + 'Jetzt Fensterfunktion 07h kombiniert ' +
                'mit Funktion 03h ');
  WriteXY(9, 5, 'Links = Punkt setzen, rechts = Punkt löschen, ' +
                'beide = Ende ');
  repeat
    Reg.AX := $0003; Intr($33, Reg);  { Mausstatus und -position }
    if (Reg.BX and 1) = 1 then  { linke Maustaste }
        begin
          Spalte := 1 + Reg.CX div 8;
          Zeile  := 1 + Reg.DX div 8;
          WriteXY(Spalte, Zeile, '•');
        end;
    if (Reg.BX and 2) = 2 then  { rechte Maustaste }
        begin
          Spalte := 1 + Reg.CX div 8;
          Zeile  := 1 + Reg.DX div 8;
          WriteXY(Spalte, Zeile, '·');
        end;
  until ( (Reg.BX and 1) = 1) and ( (Reg.BX and 2) = 2);
end;

begin { ------------------------------ }
  Maustreiber_initialisieren_und_Reset;
  Mauscursor_anzeigen;
  Mauscursor_ausblenden;
  Mauscursor_anzeigen;
  Mausposition_und_Mausknoepfe;
  Mauscursor_positionieren;
  Mausfenster_und_malen;
end.  { ------------------------------ }
```

```
program Pas27094; { BIOS-Interrupt $10: Bildschirm }
                  { Turbo-Pascal, K. Haller        }
   { ---- Interrupt $10 hat folgende Funktionen:
     $00  Videomodus setzen.
     $01  Cursorform definieren.
     $02  Cursor positionieren.
     $03  Cursorposition auslesen.
     $04  Position des Lichtgriffels auslesen (falls vorhanden).
     $05  Auswahl der aktuellen Bildschirmseite.
     $06  Textzeilen nach oben scrollen.
     $07  Textzeilen nach unten scrollen.
```

```
     $08  Auslesen des Zeichens und des Attributs.
     $09  Schreiben eines Zeichens und des Attributs.
          Cursor wird nicht versetzt. Steuerzeichen werden
          nicht interpretiert.
     $0A  Schreiben eines Zeichens. Attribut wird beibehalten
          Cursor wird nicht versetzt. Steuerzeichen werden
          nicht interpretiert.
     $0B  Auswahl von Vordergrund- und Hintergrundfarbe und der
          Farbpalette. Unterfunktionen 0 und 1.
     $0C  Grafikpunkt setzen (Koordinaten, Farbe).
     $0D  Farbe des Grafikpunktes lesen.
     $0E  Schreiben eines Zeichens. Attribut wird beibehalten.
          Cursor wird versetzt. Steuerzeichen werden interpretiert.
     $0F  Auslesen des aktuellen Videomodus.
     $13  Zeichenkette mit Attribut ausgeben. Steuerzeichen
          werden interpretiert.
    -----------
    Es werden hier demonstriert:
    • Funktion $02 (Cursor positionieren) und
    • Funktion $09 (Zeichen mit Attribut ausgeben)

    Zum Attribut-Byte bei Farbbildschirmen:
    Bit 0, 1, 2:    Farbe des Zeichens. 3 Bit = 8 Farben, 0..7
    Bit 3:          Hellere Farbe des Zeichens (Farben 0..15).
    Bit 4, 5, 6:    Farbe des Hintergrundes. 8 Farben.
    Bit 7:          Blinkende Darstellung wenn Bit gesetzt.

    Zum Attribut-Byte bei monochromen Bildschirmen:
    Bit 0, 1, 2:    Helligkeit/Unterstreichung des Zeichens.
                    Nur die drei Kombinationen "000" (schwarz),
                    "001" (unterstrichen weiß) und "111" (weiß)
                    werden ausgeführt.
    Bit 3:          Bei gesetztem Bit 3 größere Zeichen-Helligkeit.
    Bit 4, 5, 6:    Helligkeit des Hintergrundes. Nur die zwei
                    Kombinationen "000" (schwarz) und "111" (weiß)
                    werden ausgeführt.
    Bit 7:          Blinkende Darstellung wenn Bit gesetzt.
   }
uses
  CRT, DOS;           { Unit DOS für Interrupt-Aufrufe }
const
  Wiederholungen = 5;  { Jedes Zeichen so oft ausgeben }
var
  Reg:           Registers;  { Record-Typ aus Unit DOS }
  i:             Byte;
  Zeile, Spalte: Byte;
begin
  ClrScr;
  Zeile  := 5;
  Spalte := 1;
  for i := 0 to 255 do
    begin
      Reg.AH := $02;  { Funktion $02: Positionierung des Cursors }
      Reg.BH := 0;
```

```
      Reg.DL := Spalte - 1;   { Interne Zählung ab 0 }
      Reg.DH := Zeile  - 1;   { Interne Zählung ab 0 }
      Intr($10, Reg);
      ;
      Reg.AH := $09;  { Funktion $09: Zeichen mit Attribut ausgeben }
      Reg.BH := 0;    { Bildschirmseite         }
      Reg.CX := Wiederholungen;
      Reg.AL := i;    { Ascii-Code des Zeichens }
      Reg.BL := i;    { Jedes Zeichen mit eigenem Attribut }
      Intr($10, Reg);
      ;
      Inc(Spalte, Wiederholungen);
      if Spalte > 80 then
         begin
            Spalte := 1;
            Inc(Zeile);
         end;
      ;
    end;

  repeat
  until KeyPressed;
  ClrScr;
end.
```

```
program Pas27095; { Bildschirmspeicher, Turbo-Pascal, K. Haller }
                  { Farbbildschirm }
uses
  CRT;

const
  ZeilenMax       = 15;
  SpaltenMax      = 80;
  iMax            = ZeilenMax * SpaltenMax;
  Waagrechtstrich = '-';     { #196 }
  Senkrechtstrich = '|';     { #179 }

type
  Bildschirm_Typ = array[1..ZeilenMax, 1..SpaltenMax, 0..1] of Char;
                   { Zeilen, Spalten, Zeichen [0] und Attribut [1] }
var
  Bildschirm_1,
  Bildschirm_2: Bildschirm_Typ  absolute $B800:$0000;
  Bildschirm_3: Bildschirm_Typ; { Farbe: $B800:$0000 }
                                {  Mono: $B000:$0000 }
  Datei_1,
  Datei_2:      file of Bildschirm_Typ;
  Nr:           Char;

procedure Bildschirm_aufbauen(Nr: Byte; Zeichen: Char;
                              Farbe: Byte; iMax: Word);
var
  i: Word;

begin                           { Im Datenteil des BIOS steht in     }
  ClrScr;                       { der Adresse 0040:0049 die Kennung  }
```

```
  TextColor(Farbe);             { des Bildschirms; Zugriff über      }
                                { Pseudo-Array »Mem[segment:offset]« }
  for i := 1 to iMax do
    Write(Zeichen);

  GotoXY(20, 3);
  Write(' Bildschirm Nr ', Nr, ' ');
  GotoXY(20, 5);
  Write(' Die Bildschirm-Kennung: ',
          Mem[$0040:$0049]);  { »3« bei IBM 8513 }
  GotoXY(20, 7);
  Write(' Die ASCII-Nr des ersten Zeichens: ',
          Ord(Bildschirm_1[1, 1, 0]):4);
  GotoXY(20, 8);
  Write(' Das Attribut des ersten Zeichens: ',
          Ord(Bildschirm_1[1, 1, 1]):4);
  GotoXY(20, 14);
  Write(' Ende mit beliebigem Tastendruck ... ');
end;

procedure Bildschirm_3_aufbauen;
const
  Meldung_3: string = ' Bildschirm Nr 3 ';
var
  i, j: Word;

begin
  ClrScr;
  for i := 1 to ZeilenMax do
    for j := 1 to SpaltenMax do

      begin
         Bildschirm_3[i, j, 0] := Chr(((i - 1)*80 + j) mod 256);
                                { ... das Zeichen  }
         Bildschirm_3[i, j, 1] := Chr(((i - 1)*80 + j) mod 256);
                                { ... das Attribut }
      end;

  for i := 1 to Length(Meldung_3) do
    Bildschirm_3[3, 19 + i, 0] := Meldung_3[i];
end;

begin
  Assign(Datei_1, 'Screen-1.DAT');
  Assign(Datei_2, 'Screen-2.DAT');

  { -------------- Nr 1 ------------------------------------------- }
  Bildschirm_aufbauen(1, Waagrechtstrich, LightCyan, iMax);

  Rewrite(Datei_1);
  Write(  Datei_1, Bildschirm_1);
  Close(  Datei_1);

  { -------------- Nr 2 ------------------------------------------- }
  Bildschirm_aufbauen(2, Senkrechtstrich, Yellow, iMax);

  Rewrite(Datei_2);
  Write(  Datei_2, Bildschirm_1);
  Close(  Datei_2);
```

```
  { -------------- Nr 3 ------------------------------------------- }
  Bildschirm_3_aufbauen;

  { ---------------------------------------------------------------- }

  repeat
    GotoXY(15, 24); Write('Bildschirm 1, 2, 3 (Ende mit 0): ');
    repeat
      Nr := ReadKey;
    until Nr in ['0'..'3'];

    case Nr of
    '1': begin
           Reset(Datei_1);
           Read( Datei_1, Bildschirm_1);
           Close(Datei_1);
           repeat
           until KeyPressed;
         end;
    '2': begin
           Reset(Datei_2);
           Read( Datei_2, Bildschirm_2);
           Close(Datei_2);
           repeat
           until KeyPressed;
         end;
    '3': begin
           Bildschirm_1 := Bildschirm_3;
           repeat
           until ReadKey <> '';
         end;
    end;

  until Nr = '0';
end.
```

```
program Pas27096; { BIOS-Interrupt 13H: Diskette/Festplatte         }
                  { Turbo-Pascal, K. Haller }
   { ---- Der Interrupt 13H hat folgende Funktionen: ------------
          (Die Fortsetzungspunkte ... deuten an, daß weitere
          Informationen benötigt werden)

     00H  Reset. Eingabe: Funktionsnummer 00H in AH und Laufwerks-
          nummer in DL.

          Die Laufwerksnummern sind wie folgt festgelegt:
          •  bei Disketten:   00H, 01H, usw.
          •  bei Festplatten: 80H, 81H, usw.
          Gilt auch für andere Funktionen des Interupts 13H.

          Wirkung: Zurückversetzen des Controllers und des Laufwerks
          in in Einschaltzustand; die Schreib-/Leseköpfe werden auf
          eine definierte Spur gesetzt. Beim Festplatten-Reset wird
          aber auch ein Disketten-Reset durchgeführt. Falls nicht
          erwünscht, dann den mit der Funktion 0DH alternativen
          Festplatten-Reset aufrufen.
```

```
      Ausgabe: Flag in CF (carry flag), wobei 0 = Erfolg,
      1 = Fehler. In AH Statusbyte wie bei Funktion 01H.

01H   Status lesen. Eingabe (außer 01H in AH) Laufwerks-
      nummer in DL (Diskette 00H, 01H. Festplatte 80H, 81H).
      ---- Ausgabe Statusbyte in AH: ---------
      AH = 00H: Kein Fehler.
      AH = 01H: Nicht erlaubte Funktionsnummer.
      AH = 02H: Adreßmarkierung nicht gefunden.
      AH = 03H: Schreibversuch auf schreibgeschützter Diskette.
      AH = 04H: Sektor nicht gefunden.
      AH = 05H: Reset bei Festplatte nicht möglich.
      AH = 06H: Diskette entfernt.
      AH = 07H: Fehlerhafte Festplatten-Parametertabelle.
      AH = 08H: DMA-Überlauf. DMA = direct memory access.
      AH = 09H: DMA über 64 KByte.
      AH = 0AH: Fehlerhafte Sektor-Flag der Festplatte.
      AH = 0BH: Fehlerhafter Zylinder der Festplatte.
      AH = 0CH: Falscher Diskettentyp.
      AH = 0DH: Fehlerhafte Sektorenzahl im Format Festplatte.
      AH = 0EH: Kontrolldaten-Adreßmarkierung bei Festplatte
                gefunden.
      AH = 0FH: DMA-Level außerhalb gültigem Bereich. Festplatte.
      AH = 10H: Mit CRC oder ECC Lesefehler festgestellt. Prüf-
                summenverfahren CRC = cyclical redundancy check.
                Fehlerkorrekturverfahren ECC = error correction
                code.
      AH = 11H: Fehler in ECC-korrigierten Daten. Festplatte.
      AH = 20H: Controller-Defekt.
      AH = 40H: Positionierfehler.
      AH = 80H: Time-out-Fehler. Laufwerk reagiert nicht in
                Zeitspanne.
      AH = AAH: Bei Festplatte: Laufwerk nicht bereit.
      AH = BBH: Unbekannter Festplattenfehler.
      AH = CCH: Schreibfehler Festplatte.
      AH = E0H: Statusfehler Festplatte.
      AH = FFH: Prüfoperation nicht möglich.

02H   Sektoren lesen ...
03H   Sektoren schreiben ...
04H   Sektoren prüfen ...
05H   Spur (Zylinder) formatieren. Für IBM PS/2 mit ESDI-
      Laufwerken nicht diese, sondern Funktion 1AH verwenden.
      Siehe dort. Durch exotische Formatierangaben läßt sich
      "Kopierschutz" erreichen ...
06H   Nur bei PC/XT: Festplattenspur formatieren ...
07H   Nur bei PC/XT: Festplatte ab bestimmten Zylinder
      formatieren ...
08H   Parameter des Laufwerkes lesen.
      Eingaben (außer 08H in AH) Laufwerksnummer in DL,
      wobei: 80H = erste Festplatte, 81H = zweite Festplatte,
             00H = erste Disk,       01H = zweite Disk usw.
      Ausgaben:
      AH: Statusbyte wie bei Funktion 01H.
      CF: 0 = Erfolg, 1 = Fehler.
```

```
          DL: Anzahl der Laufwerke (an einem Controller)
          DH: Anzahl der Schreib-/Leseköpfe, 0 = erster Kopf usw.
          CH: Bit 0 bis 7 der maximale Zylindernummer (Bit 0 bis 9).
          CL: Bit 6 und 7: Bit 8 und 9 der maximalen Zylindernummer
              Bit 0 bis 5: maximale Sektornummer
          Da ein Sektor standardmäßig 512 Byte umfaßt, kann man die
          Gesamtkapazität der Festplatte nach folgender Formel
          berechnen:
          ‖ Kapazität = Köpfe * Zylinder * Sektoren * 512 Byte ‖
          Mit den max. 10 Bits der Zylindernummern (Bit 0 bis 9)
          ergeben sich max. 2^10 = 1024 Zylinder (Original-DOS).
          Die max. 6 Bits der Sekorennummern (Bit 0 bis 5) ergeben
          max. 2^6 = 64 Sektoren à 512 Byte. Mit einem Kopf somit
          max. (1024 * 64 * 512 Byte) = 33 554 432 Byte = 32 MByte.
    09H   Parameter einer (fremden) Festplatte anpassen.
          Interrupt 41H zeigt auf Tabelle für Laufwerk 80H,
          Interrupt 46H zeigt auf Tabelle für Laufwerk 81H.
          Siehe dort. Nicht für ESDI-Laufwerke bei IBM PS/2-
          Rechnern verwenden ...
    0AH   "Lange" Sektoren lesen. Ein langer Sektor besteht aus
          einem Sektor mit Daten und einem 4 oder 6 Bytes langen
          Fehlerkorrekturcode (ECC) ...
    0BH   "Lange" Sektoren schreiben ...
    0CH   Auf Zylinder der Festplatte positionieren ...
    0DH   Alternativer Festplatten-Reset.
          Siehe auch Funktion 00H ...
    10H   Festplattenlaufwerk bereit? ...
    11H   Festplattenlaufwerk neu kalibrieren ...
    14H   Diagnose Controller ...
    15H   Feststellen des Laufwerk-Typs ...
    16H   Diskettenwechsel erkennen ...
    17H   Diskettentyp für Formatieren festlegen
          5,25-Zoll: 360 KByte, 1,20 MByte,
          3,50-Zoll: 720 KByte, 1,44 MByte ...
    18H   Disketten-/Plattenparameter für Formatieren festlegen ...
    19H   Festplattenköpfe auf Parkspur. Nur für IBM PS/2.
          Eingabe: 19H in AL, Laufwerksnummer (80H, 81H) in DL.
          Ausgabe: Fehlerflag in CF, mit 0 = Erfolg, 1 = Fehler.
                   Statusbyte in AH.
    1AH   ESDI-Laufwerk formatieren. Nur für IBM PS/2 mit "Enhanced
          Small Device Interface"-Adapter ...
   ------------------------------------------------------------------

   Es wird hier demonstriert:

   • Funktion 08H: Parameter des Laufwerks abfragen

  }
uses
  CRT, DOS;           { Unit DOS für Interrupt-Aufrufe }

var
  Reg:       Registers;  { Record-Typ aus Unit DOS }
  Laufwerke: Byte;
  Ch:        Char;
```

```
  Laufwerk:  Byte;

  Zylinder,
  Sektoren,
  Koepfe,
  Speicher:  LongInt;

function Bit(Acht_Bit_Register: Byte; i: Byte): Boolean;
begin
  if Acht_Bit_Register and (1 shl i) = 1 shl i
     then Bit := True
     else Bit := False
end;

begin
   ClrScr;
   WriteLn('--- Interrupt 13H, Funktion 08H ---');

   Write('Eingabe Laufwerk (A, B, C, D):    ');
   repeat
    Ch := UpCase(ReadKey);
   until Ch in ['A'..'D'];
   WriteLn(Ch);

   case UpCase(Ch) of
   'A': Laufwerk := $00;
   'B': Laufwerk := $01;
   'C': Laufwerk := $80;  { Bei Festplatten ist das   }
   'D': Laufwerk := $81;  { höchste Bit auf 1 gesetzt }
   end;

   Reg.DL := Laufwerk;
   Reg.AH := $08;  { Funktion 08H: Parameter der Festplatte lesen }
   Intr($13, Reg);

   { Nachfolgend wird die Konstante "FCarry" aus der Unit DOS
     benutzt. Sie ist dort wie folgt definiert: FCarry = $0001
     »Flags« ist ein Feldbezeichner für den ebenfalls in DOS
     definierten Record-Typ »Registers« }

   WriteLn('Das Statusbyte (0 = in Ordnung): ', Reg.AH:2);

   if Reg.Flags and FCarry = 1      { Carry-Flag abfragen }
      then WriteLn('Carry-Flag auf 1. Fehler.')
      else begin
             WriteLn('Carry-Flag auf 0. Kein Fehler');

             Laufwerke := Reg.DL;

             Koepfe   := Reg.DH + 1;    { da Zählung ab 0 }

             Zylinder := Reg.CH;
             if Bit(Reg.CL, 6) then Zylinder := Zylinder + 256;
             if Bit(Reg.CL, 7) then Zylinder := Zylinder + 512;
             Zylinder := Zylinder + 1;  { da Zählung ab 0 }

             Sektoren := (Reg.CL shl 2) shr 2; { Zählung ab 1 }

             WriteLn('Anzahl der Laufwerke: ', Laufwerke:13);
             WriteLn('Anzahl der Köpfe:     ', Koepfe:13  );
```

```
            WriteLn('Anzahl der Zylinder:  ', Zylinder:13);
            WriteLn('Anzahl der Sektoren:  ', Sektoren:13);
            Speicher := Koepfe * Zylinder * Sektoren * 512;
            WriteLn('Speicher in KByte: ', Speicher div 1024:16);
          end;
   WriteLn('-----------------------------------');
   repeat
   until ReadKey <> '';
end.
```

27.10 Weitere Demo-Programme B

```
program Pas27101; { Interrupt $12: Speicherkapazität abfragen }
                  { Turbo-Pascal                  K. Haller  }
  { Mit diesem Interrupt kann nur die Größe des konventionellen
    Speichers (Basisspeicher, 0..639 KByte) ermittelt werden.
    Ausgabe in KByte.
   }
uses
  CRT, DOS;
var
  Reg: Registers;  { Record-Typ aus Unit DOS }
begin
  ClrScr;
  Intr($12, Reg);
  WriteLn('Der konventionelle Speicher in KByte: ', Reg.AX);
  repeat
  until ReadKey <> '';
end.
```

```
program Pas27102;  { Cursor ON/OFF mit Interrupt $10 }
                   { Turbo-Pascal               kha }
 { Im Interrupt $10  (dez 016) wird die Farbpalette des Bildschirms
   festgelegt. Der Interrupt hat 16 Funktionen. Die Funktion $03
   ermittelt die aktuelle Cursorposition und die Größe des Cursors,
   wogen mit der Funktion $01 die Cursorgröße bestimmt werden kann.
 }
uses
  CRT, DOS;           { Unit DOS für Interrupt-Aufruf }
var
  Cursor_LinieBeginn,
  Cursor_LinieEnde,
  Zeile, Spalte:      Byte;
  Register:           Registers; { Datentyp aus Unit DOS }
procedure Cursor_Ermitteln;     { • Funktion $03: Position und Größe }
begin                           {   des Curors auslesen              }
```

```
  Register.AH := $03;           { • Die Cursorlinien werden in der   }
  Intr($10, Register);          {   Zeichenmatrix von oben gezählt.  }
  Cursor_LinieBeginn := Register.CH;
  Cursor_LinieEnde   := Register.CL;
  Zeile              := Register.DH + 1;  { plus 1, da Zählung ab 0 }
  Spalte             := Register.DL + 1;  { plus 1, da Zählung ab 0 }
end;

procedure Cursor_EIN;
begin
  Register.AH := $01;
  Register.CH := Cursor_LinieBeginn;
  Register.CL := Cursor_LinieEnde;
  Intr($10, Register);
end;

procedure Cursor_AUS; { • Funktion $01: Größe Blink-Cursor setzen    }
begin                 { • Normale Werte: mono:  0..13, $00..$0C      }
  Register.AH := $01; {                    color: 0..7,  $00..$07      }
  Register.CH := $0D; { • Durch "unmögliche" Werte verschwindet      }
  Register.CL := $00; {   Cursor. Auch möglich: Mit Funktion $02 des }
                      {   Intr $10 außerhalb Bildschirm setzen.      }
  Intr($10, Register);
end;

procedure Cursor_Spezial;
begin
  Register.AH := $01;
  Register.CH := 2;
  Register.CL := 7;
  Intr($10, Register);
end;

begin
  ClrScr;

  Cursor_Ermitteln;

  Cursor_AUS;
  Write('Cursor unsichtbar. Drücke Return ...  '); ReadLn;

  Cursor_EIN;
  Write('Cursor sichtbar.   Drücke Return ...  '); ReadLn;

  Cursor_Spezial;
  Write('Cursor spezial.    Drücke Return ...  '); ReadLn;

  Cursor_EIN;
  Write('Cursor normal.     Drücke Return ...  '); ReadLn;
end.
```

```
program Pas27103;     { Drucker-Interrupt $17   }
                      { Turbo-Pascal, K. Haller }

   { Der Druckerstatus wird im Register AH zurückgeliefert.
     Die Bits 1 und 2 werden nicht benutzt. Die Bedeutung
     der anderen Bits wird im Programm erklärt. }

uses
```

```
  CRT, DOS;

type
  Str8 = string[8];

var
  BinaerString:  Str8;
  Reg:           Registers;  { »Registers«: ein Record-Typ
                               aus der Unit »DOS« }

function BinStr(Dezimalzahl: Byte): Str8; { ──────────────┐ }
var                                                       { │ }
  Temp: Str8;                                             { │ }
  i:    Byte;                                             { │ }
begin                                                     { │ }
  Temp := '00000000';                                     { │ }
  for i := 0 to 7 do                                      { │ }
    if Dezimalzahl and (1 shl i) = 1 shl i                { │ }
       then Temp[8 - i] := '1';                           { │ }
  BinStr := Temp;                                         { │ }
end; {────────────────────────────────────────────────────┘ }


begin
  ClrScr;

  Reg.DX := $0000;  { $0000 = Drucker 1, (paraller Drucker)
                      $0001 = Drucker 2 usw. }
  Reg.AH := $02;    { Funktion $02 des Drucker-Interrupt $17 }
                    { = Druckerstatus abfragen               }

  Intr($17, Reg);

  WriteLn('Der Druckerstatus: ', Reg.AH, '   in binär: ',
          BinStr(Reg.AH));
  WriteLn;

  if Reg.AH and (1 shl 0) =  1 shl 0   { Test Bit 0 }
     then WriteLn('Statusbit 0: Time-Out-Fehler')
     else WriteLn('Statusbit 0: Kein Time-Out-Fehler');

  WriteLn(        'Statusbit 1: Nicht benutzt');
  WriteLn(        'Statusbit 2: Nicht benutzt');

  if Reg.AH and (1 shl 3) =  1 shl 3   { Test Bit 3 }
     then WriteLn('Statusbit 3: Übertragungsfehler')
     else WriteLn('Statusbit 3: Kein Übertragungsfehler');

  if Reg.AH and (1 shl 4) =  1 shl 4   { Test Bit 4 }
     then WriteLn('Statusbit 4: Drucker On-Line')
     else WriteLn('Statusbit 4: Drucker nicht On-Line');

  if Reg.AH and (1 shl 5) =  1 shl 5   { Test Bit 5 }
     then WriteLn('Statusbit 5: Drucker hat kein Papier')
     else WriteLn('Statusbit 5: Drucker hat Papier');

  if Reg.AH and (1 shl 6) =  1 shl 6   { Test Bit 6 }
     then WriteLn('Statusbit 6: Empfang bestätigt')
     else WriteLn('Statusbit 6: Empfang nicht bestätigt');

  if Reg.AH and (1 shl 7) =  1 shl 7   { Test Bit 7 }
```

```
    then WriteLn('Statusbit 7: Drucker ist nicht beschäftigt')
    else WriteLn('Statusbit 7: Drucker ist beschäftigt');

  repeat
  until ReadKey <> '';
end.
```

```
program Pas27104;       { ROM-Basic mit Interrupt 18h starten  }
  { ROM-Basic ist eine Minimalversion von Basic. Wie der Name
    besagt, befindet sich dieses Basic fest in ROM, aber nur bei
    IBM PCs und einigen Clones. Wenn ROM-Basic nicht existiert,
    führt der Aufruf von Interrupt 18h zu einem Systemabsturz.
    Im Gegensatz zum ladbaren Basic kann ROM-Basic nicht mit dem
    sonst dafür vorgesehenen Basic-Befehl "SYSTEM" verlassen
    werden; es ist ein Kaltstart erforderlich.

    Das Beispiel »ROM-Basic« wurde nur der Kuriosität wegen in die
    Sammlung aufgenommen.
    Turbo-Pascal, K. Haller }

uses
  CRT, DOS;

var
  Reg: Registers;  { »Registers« vordefinierter Recordtyp
                     aus der Unit DOS }

begin
  ClrScr;
  GotoXY(5, 5);
  WriteLn(#7, 'Achtung: Nach folgenden Interrupt-Aufruf 18h für');
  GotoXY(5, 6);
  WriteLn('Start ROM-Basic  k e i n e  Rückkehr am Basic-Ende.');
  GotoXY(5, 7);
  Write('Kaltstart notwendig! Drücke Taste Return ... ');
  repeat
  until ReadKey = #13;

  Intr($18, Reg);

end.
```

```
program Pas27105; { früher "Mem_Dump" Speicherauszug }
                  { Dr. K. Haller }

uses
  CRT;

const
  SegmentMax                  = 1 shl 16;      { =    65.536     }
  AdressePhysischMax20Addr    = 1 shl 20 - 1;  { = 1.048.576 - 1 }
       { Max. Adresse bei 20 Adressleitungen, "A0" bis "A19"     }
       { (interne Zählung ab 0), z.B. beim i8086. Bei mehr als   }
       { 20 Adressleitungen (alle Prozessoren ab i80286) kann    }
       { mit der segmentierten Adressierung (Format "ssss:oooo") }
```

```
      { und der 21. Adressleitung "A20" noch ein weiteres 64-   }
      { KByte-Segment (HMA = High Memory Area) adressiert       }
      { werden. Bei der Adresse                                 }
      {        "FFFF:FFFF" = 16*($FFFF) + $FFFF = 1.114.095     }
      { ist aber mit der DOS-Adressierung (max. je vier Hex-    }
      { Ziffern = 16 bit für Segment und Offset) Schluss.       }

      { Das ROM-BIOS (Basic Input Output System) liegt immer    }
      { im Segment "F000" und endet mit "F000:FFFF", also der   }
      { höchsten Speicherstelle. In den letzten Bytes steht im  }
      { Format "mm/tt/jj" das Freigabedatum, dann folgen noch   }
      { drei Bytes. Der Beginn des ROM-BIOS ist hersteller-     }
      { spezifisch. BIOS-Erweiterungen liegen in den Segmenten  }
      { "C000", "D000" und "E000".                              }
  AdressePhysischMaxMitHMA    = 16*$FFFF + $FFFF;  { = 1.114.095 }
  BildschirmSegment           = $B800;     { Text-Farbbildschirm }

var
  Spalte, Zeile:    Byte;
  BildschirmOffset: Word;
  AdressePhysisch:  LongInt;
  Segment:          LongInt; { Nur für Fehlerprüfung, sonst "Word" }
  OffsetStart,
  OffsetParagraph,
  i, IOFehler:      Word;
  B:                Byte;    { Inhalt der Speicherzelle }
  Ch:               Char;

procedure WriteXY(Spalte, Zeile: Byte; Meldung: string);
begin
  GotoXY(Spalte, Zeile);
  Write(Meldung);
end;

function Dez_HexStr(x: Word; L: Byte): string;
var
  TempStr: string;
begin
  TempStr := '';
  while x <> 0 do
    begin
      TempStr := Copy('0123456789ABCDEF', (x mod 16) + 1, 1) +
                      TempStr;
      x       := x div 16;
    end;
  while Length(TempStr) < L do
    TempStr := '0' + TempStr;
  Dez_HexStr := TempStr;
end;

procedure SpeicherInfos(AdressePhysisch: LongInt);

procedure Info(s1, s2: string);
begin
  TextColor(Yellow);
  s1 := 'Hintergrund der Hex-Codes ' + s1;
```

```
  WriteXY(3,                    3, s1); ClrEol;
  GotoXY( 3, 4); ClrEoL;
  WriteXY(3 + Pos(':', s1) + 1, 4, s2); ClrEoL;
end;

begin
  if (AdressePhysisch >=    0) and  { Interrupt-Vektoren }
     (AdressePhysisch <  1024) then { in anderer Farbe   }
      begin
        Info('cyan: Arbeitsspeicher bis 1 KByte',
                    'Interruptvektoren');
        TextBackground(Cyan);
      end;
  if (AdressePhysisch >=     1024) and
     (AdressePhysisch <  640*1024) then
      begin
        Info('grün: Arbeitsspeicher von 1 KByte bis 640 KByte', '');
        TextBackground(Green);
      end;
  if (AdressePhysisch >= 640*1024) and
     (AdressePhysisch <  16*BildschirmSegment) then
      begin
        Info('braun: Hoher Speicherbereich UMA von 640 KByte',
                     'bis Bildschirmspeicher');
        TextBackground(Brown);
      end;
  if (AdressePhysisch >= 16*BildschirmSegment) and
     (AdressePhysisch <= 16*BildschirmSegment + 4000) then
      begin
        Info('hellgrau: Bildschirmspeicher',
                       'Ab Segment $B800, 4000 Byte');
        TextBackground(LightGray);
      end;
  if (AdressePhysisch >= 16*BildschirmSegment + 4000) and
     (AdressePhysisch <= 16*$C000                )    then
      begin
        Info('braun: Weiterer Video-RAM-Bereich', '');
        TextBackground(Brown);
      end;
  if (AdressePhysisch >= 16*$C000) and
     (AdressePhysisch <= AdressePhysischMax20Addr)    then
      begin
        Info('magenta: Hoher Speicherbereich UMA nach Bildschirm-',
                      'speicher. ROM-BIOS und BIOS-Erweiterungen');
        TextBackground(Magenta);
      end;
  if (AdressePhysisch > AdressePhysischMax20Addr) then
      begin
        Info('rot: High Memory Area HMA, 64 KByte ab 1024 KByte',
                   'Nur mit 21. Adressleitung "A20"');
        TextBackGround(Red);
      end;
end;
```

```
procedure Vorspann;
begin
  TextColor(White); TextBackground(Blue); ClrScr;
  WriteXY(10, 5, 'Programm zum Betrachten des DOS-Speichers');
  WriteXY(10, 7, 'Bei Hex-Eingaben  Pascal-Vorsatzzeichen $');
  TextColor(Yellow);
  WriteXY(50, 7, '$');
  TextColor(White);
  repeat
    WriteXY(10, 9, 'Eingabe Start-Segment (Beispiel ');
    TextColor(Yellow); Write('$F000');
    TextColor(White);  Write('): ');   ClrEoL;
    {$I-}
    ReadLn(Segment);
    IOFehler := IOResult;
    {$I+}
    if IOFehler = 0 then if Segment < 0
                         then IOFehler := 4711;
    if IOFehler = 0 then if Segment >= SegmentMax
                         then IOFehler := 4711;
  until IOFehler = 0;
end;

begin { ------------- Hauptprogramm -------------------- }
  Vorspann;

  ClrScr;
  TextColor(Yellow); WriteXY(1, 5, '  Segm:Offs ');
  TextColor(LightGray);

  WriteXY(13, 5, ' 00 01 02 03 04 05 06 07 08 09' +
                 ' 0A 0B 0C 0D 0E 0F  0123456789ABCDEF');
  TextColor(LightGray);
  WriteXY(3, 25, 'Ende mit Esc, weiter mit ' +
                 'beliebiger Taste ... ');
  TextColor(Yellow); WriteXY(12, 25, 'Esc');

  TextColor(White); TextBackground(Blue);
  WriteXY( 1,  1, 'FH München, Stg Druckereitechnik, Dr. K. Haller');
  WriteXY(77, 25, 'XXX');

  repeat
    GotoXY(3, 6);
     ;
    for OffsetParagraph := 0 to 4095  do    { 0 bis 65535 Byte }
      begin
        OffsetStart := OffsetParagraph * 16;

        TextColor(LightGray);
        WriteXY(3, WhereY, Dez_HexStr(Segment, 4) + ':');
        Write(Dez_HexStr(OffsetStart, 4));

        for i := 0 to 15 do  { 16 Bytes in einer Zeile darstellen }
          begin
            Spalte := WhereX;
            Zeile  := WhereY;
            B      := Mem[Segment:OffsetStart + i];
```

```
            AdressePhysisch := 16 * Segment + OffsetStart + i;
            ;
            SpeicherInfos(AdressePhysisch);
            ;
            GotoXY(Spalte, Zeile);
            ;
            WriteXY(14 + 3*i, WhereY, Dez_HexStr(B, 2));
            if i < 15 then Write(' ');
            ;
            { Alle Zeichen in Bildschirmspeicher }
            BildschirmOffset := (WhereY - 1)*160 + (62 + i)*2;
            Mem[BildschirmSegment:BildschirmOffset    ] := B;
               { Code des Zeichens     }
            Mem[BildschirmSegment:BildschirmOffset + 1] := Yellow;
               { Attribut des Zeichens }
            TextBackground(Blue);
          end;
        if WhereY < 22 then WriteLn;

        if WhereY = 22 then
           begin
             TextColor(Yellow);
             WriteXY(59, 22, '¦¦                    ');
             WriteXY(59, 23, '+- Physische Adresse ');
             GotoXY(62, 24); Write('dez ', AdressePhysisch);
             GotoXY(49, 25);
             Ch := ReadKey;
             if Ch = #27
                then Halt(0)
                else GotoXY(1, 6);
           end;
      end;
    ;
    Segment := Segment + (1 shl 12);
                       { (1 shl 12) = 4096 = 65536 div 16 }
    if Segment > 65535        { Notwendige Minus-1-Korrektur für }
       then Segment := 65535; { Segment, das in HMA hineinreicht }
  until AdressePhysisch = AdressePhysischMaxMitHMA;

  GotoXY( 3, 25); ClrEoL;
  WriteXY(3, 25, #7 + 'Beenden mit beliebigem ' +
                      'Tastendruck ...        ');
  GotoXY(WhereX - 7, WhereY);
  repeat
  until ReadKey <> '';
end.
```