Text im Grafikmodus

Eigene Zeichensätze schnell darstellen

Das Beschriften von Grafiken ist eine Aufgabe, die oft auf einen Programmierer zukommt. Zahlreiche Möglichkeiten, Zeichen im Grafikmodus auszugeben, bietet das BIOS. Wem diese Funktionen nicht genügen, dem stellen wir ein schnelles und erweiterbares Assemblermodul vor, um beliebige Zeichen in Bildschirmgrafiken zu setzen.

Von der CGA über die EGA bis hin zur VGA bietet das Video-BIOS immer mehr Möglichkeiten, Texte im Grafikmodus auszugeben und hierzu vorhandene oder eigene Zeichensätze zu benutzen.

Im Unterschied zum Textmodus, in dem der Zeichencode und das Attributbyte aus dem Bildschirmspeicher automatisch in die entsprechenden Bildpunkte umgesetzt werden, müssen bei der Zeichenausgabe im Grafikmodus die Position und Abfolge der Pixel einzeln bestimmt werden. Ihre Adressen im Videospeicher und ihre Programmierung sind zudem abhängig vom Grafikmodus selbst.

Tabelle 1. Interrupt 10h – Aufrufe zur Darstellung von Text im Grafikmodus

ah	Beschreibung	Parameter für den Aufruf
2	Set Cursor Location Auch in den Grafikmodes gelten Textkoordinaten!	bh: Bildschirmseite dh: Cursorzeile dl: Cursorspalte
9	Write Character & Attribute at Cursor Location	al: ASCII-Code bh: Bildschirmseite (Hintergrundfarbe bei Modus 13h, 320×200 in 256 Farben) bl: Vordergrundfarbe (Attribut in den Textmodes) cx: Wiederholungsfaktor
0Ah	Write Character at Cursor Location	al: ASCII-Code bh: Bildschirmseite (Hintergrundfarbe bei Modus 13h, 320×200 in 256 Farben) bl: Vordergrundfarbe (nur in den Grafikmodes) cx: Wiederholungsfaktor
0Eh	Display Character in Teletype Mode	al: ASCII-Code bh: Bildschirmseite bl: Vordergrundfarbe (nur in den Grafikmodes)
11h	Character Generator Interfaces	siehe Tabelle 2
13h	Display Character String	al = 0: Attribut in bl, Cursor fest 1: Att. in bl, Cursor verschieben 2: Att. im String, Cursor fest 3: Att. im String, Cursor verschieben bh: Bildschirmseite bl: Attribut, siehe al cx: Länge des Strings dh: Cursorzeile dl: Cursorspalte es:bp: Zeiger auf Beginn des Strings

In den *Tabellen 1* und *2* sind die Funktionsaufrufe des Softwareinterrupts 10h zusammengefaßt, mit denen man Zeichen in einem Grafikmodus ausgeben kann. Diese Funktionen haben eine sinngemäße Bedeutung in den Textmodi.

Tabelle 2. Interrupt 10h, ah = 11h: Character Generator Interface		
al	Beschreibung	Parameter für den Aufruf
0	Load alphanumeric character definition Benutzerdefinierbarer Zeichensatz	bh: Höhe bzw. Bytes pro Zeichen bl: Table im Character Gen. RAM cx: Anzahl der definierten Zeichen dx: ASCII-Code des ersten Zeichens es:bp: Zeiger auf Beginn der Matrizen
1	8×14-Matrix des ROM-BIOS	bl: Table im Character Gen. RAM
2	8×8-Matrix des ROM-BIOS	bl: Table im Character Gen. RAM
3	Character Map Select Register setzen	bl: Neuer Wert für das Character Map Select Register der EGA/VGA
4	8×16-Matrix des ROM-BIOS Load alphanumeric character definition and program the CRT	bl: Table im Character Gen. RAM
10h	Benutzerdefinierter Zeichensatz	bh: Höhe bzw. Bytes pro Zeichen bl: Table im Character Gen. RAM cx: Anzahl der definierten Zeichen dx: ASCII-Code des ersten Zeichens es:bp: Zeiger auf erste Matrix
11h	8×14-Matrix des ROM-BIOS	bl: Table im Character Gen. RAM
12h	8×8-Matrix des ROM-BIOS	bl: Table im Character Gen. RAM
14h	8×16-Matrix des ROM-BIOS Load graphics character definition	bl: Table im Character Gen. RAM
20h	Benutzerdefinierter Zeichensatz mit 8×8-Matrix für INT 1Fh	es:bp: Zeiger auf erste Matrix
21h	Benutzerdefinierter Zeichensatz	bl = 0: Textzeilen pro Seite in dl 1: 14 Textzeilen pro Seite 2: 25 Textzeilen pro Seite cx: Anzahl der definierten Zeichen dl: Textzeilen pro Seite, siehe bl es:bp: Zeiger auf erste Matrix
22h	8×14-Matrix des ROM-BIOS	bl = 0: Textzeilen pro Seite in dl 1: 14 Textzeilen pro Seite 2: 25 Textzeilen pro Seite 3: 43 Textzeilen pro Seite dl: Textzeilen pro Seite, siehe bl
23h	8×8-Matrix des ROM-BIOS	bl = 0: Textzeilen pro Seite in dl 1: 14 Textzeilen pro Seite 2: 25 Textzeilen pro Seite 3: 43 Textzeilen pro Seite dl: Textzeilen pro Seite, siehe bl
24h	8×16-Matrix des ROM-BIOS	bl = 0: Textzeilen pro Seite in dl 1: 14 Textzeilen pro Seite 2: 25 Textzeilen pro Seite 3: 43 Textzeilen pro Seite dl: Textzeilen pro Seite, siehe bl

30h	Informationen über den derzeitigen Character Generator	bh = 0: Int 1Fh-Zeiger 1: Int 43h-Zeiger 2: Zeiger auf 8×14-Matrizen im ROM 3: Zeiger auf 8×8-Matrizen im ROM 4: Zeiger auf ASCII-Codes 128 bis 255 der 8×8-Matrizen 5: Zeiger auf alternative 9×14-Matrizen im ROM 6: Zeiger auf 8×16-Matrizen im ROM 7: Zeiger auf alternative 9×16-Matrizen im ROM zurückgegebene Werte: cx: Höhe der Zeichenmatrix dl: Anzahl der Textzeilen pro Seite – 1 es:bp: Zeiger auf den Character Definition Table

Das BIOS hat's

Den Ort, an dem Text mit einer BIOS-Funktion ausgegeben werden kann, legt der Cursor fest. Seine Position ist in der Video Display Data Area (*Tabelle 3*) für alle anwählbaren Bildschirmseiten gespeichert und kann mit Interrupt 10h, Funktion 2, auf eine neue Stelle gesetzt werden. Die Funktionsnummer wird dem Interrupt im Register ah übergeben. So wie die Cursorposition werden alle Systemvariablen in der Video Display Data Area gespeichert und von den jeweiligen Interrupt-10h-Aufrufen abgefragt oder verändert.

Jene BIOS-Aufrufe, die Zeichen in einem Grafikmodus darstellen können, verwenden einen Zeichensatz, der nach der Anwahl des Grafikmodus ausdrücklich bestimmt werden muß. Das besorgt Funktion 11h, die wegen ihrer besonderen Bedeutung und ihres Parameterumfangs in der Tabelle 2 herausgehoben ist.

Ein Aufruf mit den Parametern al = 0 bis 4 legt einen Zeichensatz für einen Textmodus fest, bei den Parametern al = 10h bis 14h wird zudem der Cathode Ray Tube Controller (CRTC) für die neue Zeilen- und Spaltenzahl umprogrammiert. Der Interrupt-10h-Aufruf mit ax = 1112h, bl = 0 schaltet beispielsweise eine VGA vom Textmode 80×25- in die 80×50-Darstellung.

Der Zeichensatz mit der 8×8-Matrix stammt aus der CGA. Die erste Hälfte seiner Zeichenmatrizen, das sind die Zeichencodes 0 bis 127, sind an einer festen Speicheradresse (F000:FA6Eh) im ROM des PC abgelegt. Das MS-DOS-Programm Graftabl installiert den zweiten Teil (Zeichencodes 128 bis 255) resident im RAM und richtet den Interruptvektor 1Fh auf diese Zeichenmatrizen. Wenn dieser Zeiger nicht korrekt gesetzt ist – beispielsweise noch mit 0000:0000h initialisiert – werden bei Ausgabe der Zeichen 128 bis 255 im Grafikmodus unsinnige Pixelanreihungen aus den ersten 1024 Byte des Hauptspeichers ausgegeben, auf dessen Anfang der Vektor 0000:0000h ja zeigt.

Tabelle 3. Video Display Data Area

Adresse	Typ	Beschreibung für den Aufruf
0040:0040h	Word	Größe des verwendeten Videospeichers in Bytes
0040:0050h	Word*8	Cursorpositionen für maximal 8 Bildschirmseiten, Zeile im MSB, Spalte im LSB
0040:0060h	Word	Oberste (MSB) und unterste (LSB) Linie des Cursorblocks
0040:0084h	Byte	Angezeigte Textzeilen pro Bildschirmseite – 1
0040:0085h	Word	Höhe der Zeichenmatrix

Mit der EGA kam die 8×14-Matrix hinzu, und mit der MCGA und der VGA die 8×16-Matrix. Natürlich ergeben bei den jeweiligen Grafikkarten nur die Funktionsaufrufe aus Tabelle 2 sinnvolle Ergebnisse, die sich auf vorhandene Eigenschaften der Grafikadapter beziehen. Im ROM-BIOS der neueren Videokarten sind alle Zeichensätze komplett enthalten, also auch die CGA-kompatiblen 8×8-Matrizen. Die Adressen lassen sich mit Int 10h, ax = 1130h und bh laut Tabelle 2 ermitteln.

Für die Zeichenausgaben in den Grafikmodi sind die Aufrufe al = 20h bis 24h des Interrupt 10h, mit ah = 11h wichtig. Beim VGA-Adapter kann zum Beispiel eine der vordefinierten 8×8-, 8×14- oder 8×16-Matrizen ausgewählt oder ein eigener Zeichensatz eingebunden werden. Das Format für eigene Zeichenmatrizen läßt sich aus dem Prinzip,

das in *Bild 1* für die 8×8-ROM-Matrix dargestellt ist, leicht ableiten. Einzelne Bytes repräsentieren das Zeichen zeilenweise, gesetzte Bits entsprechen Pixeln in der Vordergrundfarbe.

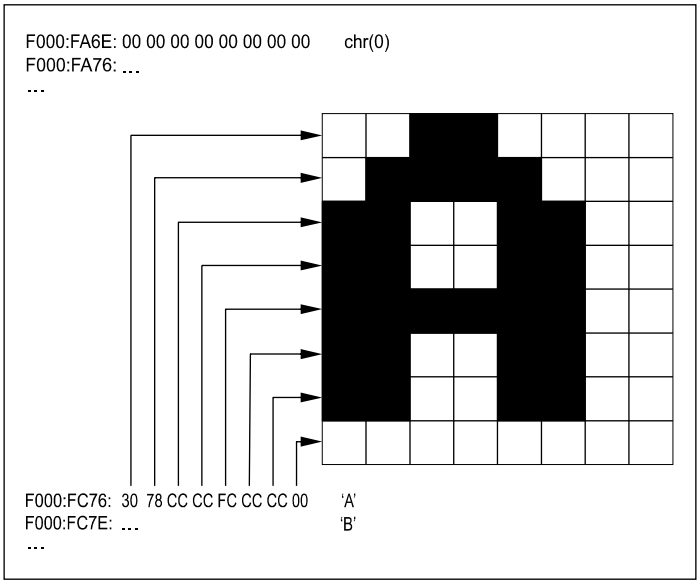

Bild 1. Die Zeichenmatrizen sind byteweise im Speicher abgelegt

Das Character Generator Interface (CGI) ist flexibel und bietet eine bequeme Möglichkeit, einen neuen Zeichensatz für die Bildschirmausgabe zu bestimmen. Die Änderungen in der Video Data Display Area werden automatisch durchgeführt. Hierbei wird im Eintrag 0040:0085h die Höhe der Zeichenmatrix festgehalten, die bei den grundsätzlich ein Byte breiten Matrizen der PC-Zeichensätze gleichbedeutend ist mit der Größe einer Zeichenmatrix in Byte. Das Wort an der Adresse 0040:0084h enthält die Nummer der untersten Textzeile einer Bildschirmseite. Der Zeiger des Interrupt 43h wird auf den Beginn der Zeichenmatrizen im Speicher gerichtet.

Einzelne Zeilen...

Zurück zu den Textausgabefunktionen aus Tabelle 1. Die mit Interrupt 10h, ah = 11h installierte Zeichenmatrix wird von den BIOS-Routinen ah = 9, 0Ah, 0Eh und 13h zur Zeichenausgabe verwendet. Die Aufrufe 9 und 0Ah arbeiten im Grafikmodus identisch, im Textmodus läßt sich mit Funktion 0Ah zusätzlich auch das Attribut verändern. Beide BIOS-Routinen geben ein Zeichen, das mehrfach wiederholt werden kann, an der Cursorposition aus, ohne diese zu aktualisieren.

Die Zeichenmatrix wird dabei so in den Grafikbildschirm übertragen, daß die Hintergrundfarbe auf 0 gesetzt wird, oder – sofern Bit 7 der Vordergrundfarbe im Aufruf gesetzt ist – das Zeichen mit dem ursprünglichen Inhalt des Videospeichers XOR-verknüpft wird. Hierbei bleibt im Bereich des Hintergrundes der Bildschirminhalt erhalten, die Farbe der einzelnen Pixel des neuen Zeichens können sich aus der gewählten Vordergrund- und der bisherigen Pixelfarbe jedoch unterschiedlich ergeben. Mit den BIOS-Routinen ist es daher in den Grafikmodi nicht möglich, Zeichen unter Beibehaltung des Hintergrundes auf jeden Fall in einer bestimmten Vordergrundfarbe – unabhängig vom bisherigen Bildschirminhalt – auszugeben. Dies ist ein erhebliches Manko der BIOS-Funktionen.

Die Aufrufe ah = 0Eh und 13h verwenden die eben besprochene Routine 0Ah zur Ausgabe ihrer Zeichen, die Verarbeitung der Farben bleibt also die gleiche. Die Int-10h-Funktion ah = 0Eh gibt ein Zeichen im Teletype-Mode aus. Der Cursor wird hierbei mitbewegt, und die Steuercodes 7 (Glocke), 8 (Rückschritt), 10 (Zeilenvorschub) und 13 (Wagenrücklauf) werden ausgewertet, leider jedoch nicht ASCII 9, der Tabulator.

...oder ganze Strings

Funktion 13h gibt gleich eine ganze Zeichenkette aus, wobei die oben genannten Steuerzeichen ausgewertet werden, der Cursor wahlweise weiterbewegt und die Attribute ebenfalls wahlweise im String jedem einzelnen Zeichen mitgegeben werden.

Eine Besonderheit ergibt sich bei PC/AT mit EGA oder MCGA: Zeilenvorschub und Wagenrücklauf werden immer auf die angezeigte Bildschirmseite bezogen, egal welche Bildschirmseite man für die gesamte Zeichenkette im bh-Register angibt.

Die BIOS-Routinen können Zeichen nur an Cursorpositionen in Textkoordinaten ausgeben, also nicht auf ein beliebiges Pixel genau in Grafikkoordinaten. Die Cursorsteuerung arbeitet byteorientiert – alle ROM-Zeichensätze sind 8 Bit breit – und orientiert sich an den Byte-Adressen im Videospeicher. Das bedeutet für die Ausgabe im Grafikmode, daß eine Zeile, die über den rechten Bildschirmrand hinausgehen würde, im Speicher kontinuierlich abgelegt und auf dem Monitor links eine Pixelreihe tiefer als neuer Zeilenanfang fortgesetzt wird. Unter Umständen werden dort bereits vorhandene Zeichen überschrieben und unleserlich. Wer die BIOS-Routinen verwenden will, muß also selbst auf die Zeichenumbrüche achten.

Die vorhandenen Ausgabefunktionen sind unkompliziert geschrieben und erfüllen ihre Aufgabe, ohne sich durch besondere Raffinessen auszuzeichnen. *Listing 1* enthält ein Assemblermodul für den EGA-640×350-und den VGA-640×480-Grafikmodus, das eine Zeichenkette ausgibt und zwei Hauptnachteile der BIOS-Funktionen abstellt. Zum einen werden alle Zeichen in ihrer Vordergrundfarbe vor unverändertem Hintergrund dargestellt, wobei alle mit dem Character Generator Interface installierten Zeichensätze verarbeitet werden können. Zum anderen lassen sich die Zeichen über Grafikkoordinaten unabhängig vom Textcursor an jede beliebige Stelle des Bildschirmes schreiben. So können auch Diagramme in hoher Auflösung exakt und auf das Pixel genau beschriftet werden.

Kursiv, genau und schnell

Als Zugabe läßt sich der Text auch kursiv ausgeben, wobei die gesamte Zeichenmatrix von der untersten zur obersten Rasterzeile um jeweils ein Pixel nach rechts verschoben wird. Das Assemblermodul kann Zeichen bis zu einer Höhe von 17 Pixel bei einer Breite von grundsätzlich 8 Bit kursiv darstellen. Das Programm in *Listing 2* dient als Beispiel für das Einbinden des Moduls aus Listing 1.

Die Assemblerroutine GRSTRING.INC enthält mit den Adreßberechnungen und der Programmierung der EGA/VGA-Ports das Grundgerüst für individuelle Erweiterungen. Neue Möglichkeiten gibt es genug, hier einige Anregungen.

Die Vordergrund-, wie auch die Hintergrundfarbe brauchen nicht überschrieben zu werden, sondern können auch mit den bisherigen Farben über AND, OR oder XOR verknüpft werden. Das Clipping für ein bestimmtes Grafikfenster kann eingefügt werden, wobei die Zeichen entweder ganz oder pixelweise dem Clipping unterworfen werden. Dieses teilweise Clippen ist weitaus aufwendiger. Hier bietet sich das Verwenden eines Hilfsspeichers an, in den Bereiche der Matrix ausgeblendet werden, bevor der gesamte Hilfsspeicher als Block in den Bildschirmspeicher übertragen wird.

Der Zeilenumbruch am rechten Rand kann mit einem Zeilenvorschub verbunden werden, so daß der Text in der neuen Reihe eine wirkliche Textzeile tiefer steht. Beliebig breite und hohe Zeichensätze können berücksichtigt werden, ebenso wie ein proportionaler Zeichensatz durch eine Breitentabelle. Hierzu muß die horizontale Byte-Grenze softwaremäßig umgangen werden, wie im Listing durch die Shift-Befehle vorgestellt. Durch Bitmanipulationen lassen sich auch eine Fettschrift oder eine vertikale Zeichenorientierung erzeugen. Last but not least kann für die Grafikmodi auch die Grenze der 256 darstellbaren Zeichen gesprengt werden.

Das Assemblermodul in Listing 1 kann individuell und aufwendig ergänzt werden, ganz nach Ihren Wünschen. Doch je mehr Zusätze man macht, und je komplizierter der Quellcode, um so inkompatibler werden die Ausgaberoutinen zu den BIOS-Funktionen.

Bitbewegungen in Assembler

Routinen zum pixelweisen Darstellen von Zeichen sind immer zeitkritisch, daher sollten sie in Assembler programmiert werden. Darüber hinaus sollte immer maximale Kompatibilität zur Vorgehensweise des BIOS angestrebt werden, zum Beispiel das Installieren neuer Zeichensätze mit Interrupt 10h, ah = 11h, damit alle notwendigen Zeiger gesetzt werden, und die Video Display Data Area sinnvolle Informationen enthält.

Nun noch einige Tips, wie Sie an neue Zeichensätze – genauer gesagt neue Zeichenmatrizen – kommen. Einen eigenen Assembler-Quellcode mit einzelnen Define Bytes-Anweisungen (DB) einzutippen, ist sehr mühevoll. Ein

Beispiellisting hierzu hätte den Beitrag ziemlich aufgebläht, und wer will schon 2048 Byte, die allein für einen Zeichensatz mit 256 verschiedenen 8×8-Matrizen notwendig sind, per Hand eintippen?

Wenn Sie die im BIOS-ROM vorhandenen Zeichensätze nur wenig verändern wollen, können sie diese in den Hauptspeicher kopieren und gezielt verändern. Wie Sie die Matrizen im ROM aufstöbern, wissen sie ja jetzt. Sollte Ihnen jedoch ein völlig neues Outfit für Ihre Zeichen vorschweben, so bietet der Public-Domain-Markt eine reichhaltige Auswahl an Editoren – zur bequemen Eingabe der Pixelmatrizen per Maus – oder fertige Fonts. Es lassen sich übrigens auch Softfonts für Laserdrucker zweckentfremden.

Ingo Eickmann/ed

Literatur

[1] *Wilton, Richard:* The Programmer's guide to PC and PS/2, Video Systems, Microsoft Press, 1987.

Listing 1. Das Include-File GRSTRING.INC gibt Texte im Grafikmode normal oder kursiv vor beliebigem Hintergrund aus

```
; ---------------------------------------------
;     GRSTRING.INC
; ---------------------------------------------
;     Include-Datei zur Ausgabe von Text
;     in den EGA/VGA-Grafikmodes 10h und 12h
; ---------------------------------------------

BIOSSeg      equ  0040h
GRC          equ   3CEh
GRSeg        equ  0A000h
INT43h       equ    43h shl 2
LineLength   equ    80
PointsOfs    equ    85h

OUTDX   macro X           ; Makro zur Ausgabe
                          ; eines Wortes an
        mov ax,X          ; den Port in DX
        out dx,ax
        endm
; ---------------------------------------------

        .DATA             ; Beginn des Daten-
                          ; Segments

Color    db 0             ; Variable der Sub-
                          ; routine GRString
Func     db 0
Points   dw 0
TabSeg   dw 0
VAddr    dw 0
; ---------------------------------------------

         .CODE            ; Beginn des Code-
                          ; Segments
; =============================================
; Subroutine GRString: Ausgabe eines Strings im
;                      Grafikmode
;          ax:    X-Grafikkordinate
;          bx:    Y-Grafikkordinate
;          dh:    Farbe
;          dl:    Funktion
;                 0: Normal
;                 1: Kursiv (max Höhe: 17 Pts.)
;          es:si: Zeiger auf String, letztes
;                 Zeichen muß 00h sein
; =============================================
GRString proc near
        mov [Func],dl     ; Funktion zwi-
                          ; schenspeichern
        push ax           ; X retten

        mov ah,dh
        xor al,al
        mov dx,GRC        ; Graphics Controller
```

```
                        ; programmieren
        out dx,ax       ; Farbe in das
                        ; Set/Reset Register
        OUTDX 0F01h     ; Enable Set Reset
                        ; Register
        OUTDX 0003h     ; Data Rotate/
                        ; Function
                        ; Select Register
        mov ax,LineLength ; Offset im Video-
                        ; speicher berechnen:
        mul bx          ; Adr = Y * 80
                        ; Byte/Line
        pop bx
        mov ch,bl
        and ch,00000111b  ; Anzahl der hori-
                        ; zontalen Shifts
        mov cl,3
        shr bx,cl       ; Adr = Adr + X div 8
        mov cl,ch
        xor ch,ch
        add ax,bx
        mov [VAddr],ax  ; Offset im Video-
                        ; speicher ablegen
        push ds         ; Datensegment retten
        mov ax,BIOSSeg  ; BIOS-Datensegment
        mov bx,INT43h   ; Offset des INT 43h-
                        ; Vektors
        mov ds,ax
        mov dl,ds:[PointsOfs]; Höhe der aktu-
                        ; ellen Zeichenmatrix
        xor ax,ax       ; aus der Video
                        ; Display Data Area
        mov ds,ax
        lds di,ds:[bx]  ; Zeiger DS:DI auf
                        ; aktuellen Font
        mov bx,ds       ; laut INT 43h
        pop ds
        mov [TabSeg],bx ; ermittelte Werte
                        ; zwischenspeichern
        mov byte ptr [Points],dl
        cmp [Func],1    ; Kursiv ?
        jnz NextChr2
        add cl,dl       ; JA: Versatz für
                        ; oberste Scan Line
        dec cl
NextChr2:
; ===============================================
        push ds         ; äußere Schleife für
                        ; nächstes Zeichen
        push es
        pop ds
        cld
        lodsb           ; nächstes ASCII-
                        ; Zeichen auf String
        pop ds
        or al,al        ; Endmarke 00h ge-
                        ; funden?
        jnz NoEnd       ; NEIN => Rausspringen
```

```
        jmp Raus          ; JA => Rausspringen
NoEnd:  push [VAddr]      ; Adresse und Versatz
                          ; für nächstes
        push cx           ; Zeichen retten
        xor ah,ah
        mov bx,[Points]
        mul bx            ; Eintrag in Matrizen-
                          ; tabelle berechnen
        push bx
        mov bx,ax
NextLine:
; ---------------------------------------------
        push ds           ; innere Schleife für
                          ; nächste Scan Line
        mov ax,TabSeg
        mov ds,ax         ; nächste Scan Line
                          ; des Zeichens aus der
        mov ah,[di+bx]    ; Tabelle holen
        pop ds
        inc bx            ; Zeiger bx erhöhen
        push bx           ; Register für das Be-
                          ; schreiben des Video-
        push di           ; speichers retten
        xor bx,bx
        push cx
        and cx,0FFh       ; LSB extrahieren
        jz Weiter         ; Verschiebung = 0 ?
Rotate: shr ax,1          ; Bitmuster pixelweise
                          ; horz. verschieben
        rcr bx,1
        loop Rotate
Weiter: mov ch,al         ; 2. Byte zwischen-
                          ; speichern
        mov al,8          ; Bit Mask Register des
                          ; Graphic-Controllers
        mov dx,GRC        ; programmieren
        out dx,ax
        mov di,[VAddr]
        push ds
        mov ax,GRSeg      ; Segment des Video-
                          ; speichers im Grafik-
        mov ds,ax         ; mode herstellen
        mov cl,[di]       ; Pixel des linken
                          ; Byte setzen
        mov [di],cl
        mov al,8          ; Bit Mask Register
                          ; adressieren
        IRP n,<ch,bh,bl>  ; die restlichen
                          ; 3 Byte schreiben
           inc di         ; Videospeicheradresse
                          ; des Bytes
           mov ah,&n
           or ah,ah       ; Pixel setzen ?
           jz Zero&n
           out dx,ax      ; JA: Bitmaske in
                          ; Graphic Controller
           mov cl,[di]    ; Pixel dieser Scan-
                          ; Line setzen
```

```
           mov [di],cl
Zero&n     label near
        endm
        add di,LineLength - 3
                          ; Adresse fur nächste
        pop ds            ; Scan Line berechnen
        mov [VAddr],di    ; und ablegen
        pop cx
        cmp [Func],1      ; Kursiv ?
        jnz Norm
        dec cl            ; JA: Versatz verrin-
                          ; gern
Norm:   pop di            ; Register wieder her-
                          ; stellen
        pop bx
        pop ax
        dec ax
        jz NextChr1       ; letzte Scan Line er-
                          ; reicht?
        push ax           ; NEIN => weiter zur
                          ; nächsten Scan Line
        jmp NextLine      ; Sprung zum Schlei-
                          ; fenanfang
; ---------------------------------------------
NextChr1:
        pop cx            ; Versatz der obersten
                          ; Scan Line
        pop ax            ; Adresse im Video
                          ; speicher für nächs-
        inc ax            ; tes Zeichen erhöhen
        mov [VAddr],ax
        jmp NextChr2      ; Sprung zum
                          ; Schleifenanfang
; =============================================
Raus:                     ; Standardwerte für
                          ; den Graphics Contr.
        OUTDX 0000h       ; Set/Reset Register
        OUTDX 0F01h       ; Enable Set Reset Re-
                          ; gister
        OUTDX 0003h       ; Data Rotate/Function
                          ; Select Register
        OUTDX 0FF08h      ; Bit Mask Register
        ret
GRString endp
```

Listing 2. Das Programm GRDEMO.ASM zeigt die Möglichkeiten der Include-Datei

```
; ------------------------------------------
;     GRDEMO.ASM
; ------------------------------------------
;     Assemblieren mit MASM 5.x :
;            MASM grdemo;
;            LINK grdemo;
; ------------------------------------------
;  Beispielprogramm zu GRSTRING.INC fur EGA/VGA.
;  Farbziffern werden normal und kursiv,
;  horizontal und vertikal um einzelne Pixel
;  versetzt ausgegeben.
; ------------------------------------------

        PAGE   65,80
        TITLE  GRDEMO

Font8        equ  1123h  ; ax-Register für
                         ; CGI (8x16-Matrix
Font14       equ  1122h  ; nur bei VGA)
Font16       equ  1124h
HorzOff      equ   100   ; horizontaler Offset
                         ; für '(kursiv)'
Rows8        equ     3   ; bl-Register für CGI
Rows14       equ     2   ; (8x16-Matrix nur
Rows16       equ     2   ; bei VGA)
VertOff      equ   175   ; Versatz fur untere
                         ; Bildhälfte

        DOSSEG
        .MODEL SMALL     ; Standard Memory
                         ; Model
        .STACK 100h
        INCLUDE GRSTRING.INC
; ------------------------------------------
        .DATA            ; Beginn des Daten-
                         ; Segments
Paras   dw 0             ; Zwischenspeicher für
                         ; dh und dl
Str8    db '8x8 Matrix',0
                         ; String für CGA-
                         ; Zeichenmatrix
Str14   db '8x14 Matrix',0
                         ; String für EGA-
                         ; Zeichenmatrix
Str16   db '8x16 Matrix',0
                         ; String für VGA-
                         ; Zeichenmatrix
StrKurs db '(kursiv)',0  ; String für kursiv
StrZ    dw 0             ; Platzhalter für
                         ; Ziffern
; ------------------------------------------
        .CODE            ; Beginn des Code Seg-
                         ; ments
Start:
        mov ax,@DATA     ; Datensegment in DS
                         ; laden
```

```
        mov ds,ax
        mov es,ax         ; für GRString auch ES
                          ; mit DS laden
        mov ax,0010h      ; Grafikmode 10h
                          ; (650x350, 16 colors)
        int 10h

        xor dl,dl         ; 1. Normal ausgeben
        xor bp,bp         ; 2. Kursiv (dl=1) mit
                          ; Versatz (bp)
Lop3:   mov cx,10h        ; Laufvariable für 16
                          ; Farben
        push dx
        mov ax,Font8      ; 8x8-Matrix anwählen
        mov bl,Rows8      ; maximal 43 Zeilen zu-
                          ; lassen
        int 10h
        pop dx
Lop1:   push cx           ; Farbe retten
        mov ax,cx         ; X- und Y-Grafik-
        mov bx,cx         ; koordinate berechnen
        dec cx            ; Farbnummer ermitteln
        mov dh,cl
        add cl,'0'        ; in ASCII umwandeln
        cmp cl,'9'
        jle Lop2
        add cl,'A'-'9'-1 ; Farben 0Ah bis 0Fh
                          ; berücksichtigen
Lop2:  mov byte ptr [StrZ],cl
                          ; Farbnummer als String
                          ; ablegen
       mov si,offset StrZ
                          ; Zeiger ES:SI auf den
                          ; String richten
       mov cl,3
       shl ax,cl          ; X-Koordinate für
                          ; horizontale Reihe
       push ax            ; Parameter für 2. Auf-
                          ; ruf speichern
       push bx
       add bx,bp          ; vertikalen Versatz
                          ; addieren
       push dx
       push si
       call GRString    ; Farbnummer in horiz.
                        ; Reihe ausgeben
       pop si           ; Parameter für 2. Auf-
                        ; ruf neu laden
       pop dx
       pop ax           ; X- und Y-Koordinate
                        ; vertauschen
       pop bx
       add bx,bp        ; vertikalen Versatz
                        ; addieren
       push dx
       call GRString    ; Farbnummer in vertik.
                        ; Reihe ausgeben
       pop dx
```

```
       pop cx
       loop Lop1        ; nächster Farbwert

       mov dh,7         ; Pixelfarbe weiß
       mov [Paras],dx
       IRP n,<8,14>     ; Bei VGA: <8,14,16>
          mov ax,Font&n
          mov bl,Rows&n
          int 10h       ; Font für Matrix an-
                        ; wählen
          mov ax,160    ; Grafikkoordinaten für
          mov bx,10 * &n; die Strings berechnen
          add bx,bp     ; vertikalen Versatz
                        ; addieren
          mov dx,[Paras]
          cmp dl,1      ; kursive Ausgabe ?
          jnz Norm&n    ; NEIN => nur normaler
                        ; String
          push ax       ; JA  => '(kursiv)'
                        ; ausgeben
          push bx
          add ax,HorzOff;horizontalen Offset
                        ; addieren
          mov si,offset StrKurs
          call GRString ; String ausgeben
          pop bx
          pop ax
          mov dx,[Paras]
Norm&n:   mov si,offset Str&n
                        ; ES:SI auf String
                        ; richten
          call GRString ; String ausgeben
       endm

       mov bp,VertOff   ; Versatz für 2. Durch-
                        ; lauf
       mov dx,[Paras]
       inc dl           ; Funktionsparameter
                        ; erhöhen
       cmp dl,1         ; noch kursiv ?
       jg DOS           ; NEIN => Ende Ausgabe
       jmp Lop3         ; JA   => Anfang der
                        ; Ausgabe
DOS:   mov ah,0         ; auf Tastatureingabe
                        ; warten
       int 16h
       mov ax,0003      ; Textmode 3 (80x25,
                        ; 16 colors)
       int 10h
       mov ax,4C00h     ; Programmende und
                        ; Rücksprung ins DOS
       int 21h

       END Start        ; Ende von GRDemo.ASM
```