Praktikumsprotokoll – Mikrorechentechnik I
Versuch „Assembler“ (A 03)

Gruppe 63:
Fabian Kurz, Alexander Eder
Stephan Stiebitz, Phillip Burker

4. Dezember 2004

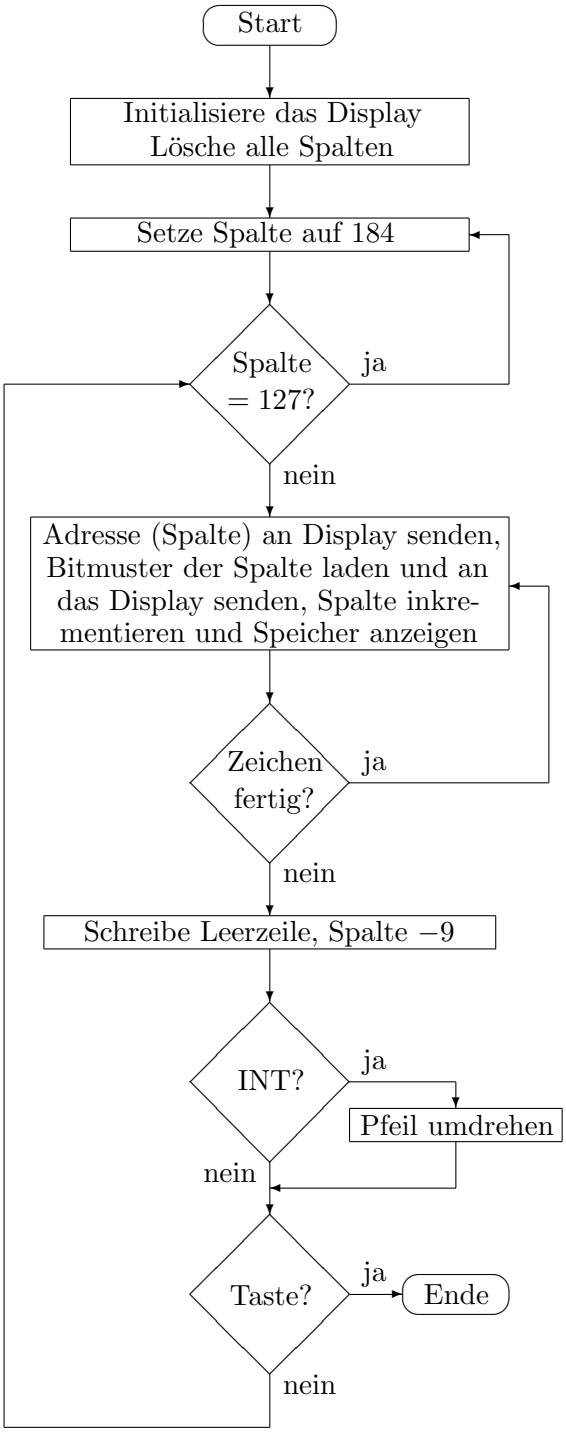
Inhaltsverzeichnis

1 Aufgabenstellung

Stellen Sie am unteren Rand des Matrix-Displays einen horizontal von rechts nach links wandernden Pfeil dar. Wenn das Symbol den linken Rand erreicht hat, soll die Bewegung am rechten Rand wieder beginnen. Durch Betätigen der `INT`-Taste soll zwischen zwei verschiedenen Pfeilen ($\rightarrow$ und $\leftarrow$) umgeschaltet werden. Das Programm soll durch eine beliebige Tastatureingabe beendet werden.

2 Lösungsprozess

2.1 Programmablauf



Der Programmablauf ist im nebenstehenden Flußdiagramm dargestellt. Nach dem Starten wird das Display initialisiert und komplett gelöscht, da am Anfang nicht davon ausgegangen werden kann, daß das Display leer ist. Danach wird die Spaltennummer auf 184 gesetzt, dies entspricht der ersten Spalte eines 8 Spalten breiten Zeichens, das sich am rechten Rand des Displays befindet.
Es wird dann überprüft, ob die aktuelle Spalte gleich 127 ist, also die Position unten links im Display, an die wieder zurückgesprungen werden muß. Ist die Überprüfung positiv, so springt das Programm zurück und setzt die aktuelle Spalte wieder auf 184, also der Ausgangsposition. Verläuft der Test negativ, wird mit die aktuelle Adresse / Spalte an das Display gesendet und in den Zwischenspeicher übernommen. Danach wird das Bitmuster der ersten Spalte des Zeichens gelesen, an das Display übermittelt und in den Zwischenspeicher gerbacht und letztendlich angezeigt.
Die Position der aktuellen Spalte wird inkrementiert und überprüft, ob schon alle 8 Byte / Spalten des Zeichens auf das Display übertragen wurden. Ist dies nicht der Fall, wird der Vorgang des Zeichnens der einzelnen Spalten wiederholt, andernfalls wird noch eine Leerspalte eingefügt (um den Rest des vorhergehenden Pfeils zu löschen). Danach wird die Spaltenposition um 9 dekrementiert, da der nächste Pfeil eine Position weiter nach links verschoben werden soll, das Zeichnen der einzelnen 8 Spalten allerdings nach rechts hin erfolgte.
Zuletzt werden noch die Interrupt-Taste und eine Tastatureingabe abgefragt; falls die Interruptabfrage positiv verläuft, wird der Pfeil umgedreht, wenn die Tastaturabfrage positiv verläuft, wird das Programm beendet, andernfalls die Hauptschleife wiederholt.

2.2 Datenübergabe

Die Übergabe der Daten an das Display erfolgt über den Parallel-Port des Computers. Je ein 8-bit breites *Datenregister* `dport` und *Steuerregister* `seport` beinhalten die zur Ansteuerung nötigen Werte. Dessen Inhalte werden mittels des `out`-Befehls an die entsprechenden PC-Adressen, 378 h und 378 h + 2 respektive, gesendet.
Um eine Anzeige auf dem Display zu erzeugen muß zuerst das Bit 2 (`ENABLE`) des Steuerregisters auf 1 gesetzt werden, da sonst das Display dunkelgetastet ist.
Die eigentliche Übergabe der anzuzeigenden Werte, Adresse und Bitmuster der Spalte, erfolgt über `dport`, die Steuerung erfolgt über `seport`. Die einzelnen Schritte werden im Folgenden genau erläutert:

Beschreibung der Displayansteuerung

1. Senden der Spalte / Adresse (0–191) an `dport`.
2. Setzen von Bit 3 an `seport` auf 1, durch die 0–1–Flanke wird die im ersten Schritt übergebene Adresse in den Zwischenspeicher für die Displayadresse übernommen.
3. Senden des Bitmuster für die in 1. angebenene Adresse an `dport`.
4. Zurücksetzen von Bit 3 an `seport` auf 0, durch die 0–1–Flanke wird der in 3. übergebene Wert in den Zwischenspeicher für Daten übernommen.
5. Setzen von Bit 0 an `seport` auf 1, durch die 0–1–Flanke werden die Werte aus den Zwischenspeichern für Displayadresse und Daten an den Dualport-RAM des Displays übergeben und somit angezeigt.

2.3 Unterprogramme

Zur besseren Gliederung und Wiederverwendbarkeit wiederkehrender Codeabschnitte wurde zur Realisierung der Aufgabe eine Reihe von Unterprogrammen geschrieben, welche im Folgenden erläutert werden.

2.3.1 Initseport

Das Unterprogramm `initseport` initialisiert das Steuerregister. Bit 0 (`CE-RAM`) wird auf 0 gesetzt. Eine 0–1–Flanke dieses Bits bewirkt eine Übernahme der Zwischenspeicher in den Dualport-RAM und somit eine Aktualisierung des Displays; dies wird später benötigt.

Quelltext

```
initseport proc
        push ax              ; Retten der Register ax und dx auf Stack
        push dx

        mov al,00001000b     ; Setze al auf Initialisierungswert
        mov dx,seport        ; Lade Adresse von seport in dx
        out dx,al            ; geben Initialisierungswert an seport aus

        pop dx               ; Wiederherstellen der Register in entgegen-
        pop ax               ; gesetzer Reihenfolge, da FILO-Stack
        ret                  ; Return
initseport endp
```

2.3.2 Datendport

Das Unterprogramm `datendport` übergibt das in der Variable `daten` gespeicherte Bitmuster an `dport`.

Quelltext

```
datendport proc
        push ax              ; Retten der Register ax und dx
        push dx

        mov al, daten        ; Laden des Datums in al
        mov dx, dport        ; Laden der Adresse von dport in dx
        out dx, al           ; Ausgabe von daten an dport

        pop dx               ; Wiederherstellen der Register dx und ax
        pop ax
        ret
datendport endp
```

2.3.3 Dispspaldport

Das Unterprogramm `dispspaldport` übergibt die in der Variable `dispspal` gepeicherte Displayposition/Adresse an `dport`.

Quelltext

```
dispspaldport proc
        push ax              ; Retten der Register ax und dx
        push dx

        mov al, dispspal     ; Laden der aktuellen Displayspalte in al
        mov dx, dport        ; Laden der Adresse von dport in dx
        out dx, al           ; Ausgabe der Displaysspalte an dport

        pop dx               ; Wiederherstellen der Register ax und dx
        pop ax
        ret                  ; Zurueck zum Programm
dispspaldport endp
```

2.3.4 Dportds

Die Umschaltung von Bit 3 des Steuerregisters `seport` dient zur Übernahme des Datenregisters `dport` in den Zwischenspeicher für Daten und wird im Unterprogramm `dportds` realisiert.

Quelltext

```
dportds proc
        push ax              ; Register ax auf Stack sichern
        push dx              ;          dx
```

```
        mov  al, 00001000b   ; Bit 3 = 1, Rest 0 auf das Register kopiert
        mov  dx, seport      ; Adresse des seports in dx laden
        out  dx, al          ; 00001000b an seport, Übernimmt dport in
                             ; den Datenspeicher

        pop dx               ; dx zurückholen (FILO!)
        pop ax               ; ax zurückholen
        ret                  ; zurück zum Hauptprogramm
dportds endp
```

2.3.5 Dportas

Das Gegenstück zu `dportds` (2.3.4) ist `dportas`, welches das Bit 3 des Steuerregisters auf 0 setzt. Durch die entstehenden 1–0–Flanke werden die Daten von `dport` in den Zwischenspeicher für Adressen übernommen.

Quelltext

```
dportas proc
        push ax              ; Register ax auf Stack sichern
        push dx              ;          dx

        mov  al, 00000000b   ; Bit 3 = 0  auf das Register kopiert
        mov  dx, seport      ; Adresse des seports in dx laden
        out  dx, al          ; 00000000b an seport, Übernimmt dport in
                             ; den Adressspeicher

        pop dx               ; dx zurückholen (FILO!)
        pop ax               ; ax zurückholen
        ret                  ; zurück zum Hauptprogramm
dportas endp
```

2.3.6 Anzeigen

Das Unterprogramm `anzeigen` setzt das Bit 0 von `seport` auf 1 und erzeugt somit die 0–1–Flanke, welche zur Übernahme der Werte aus den Daten– und Adress–Zwischenspeichern in den Dualport-RAM und somit zum Display benötigt wird.

Quelltext

```
anzeigen proc
        push ax              ; Retten der Register ax und dx
        push dx

        or  al, 00000001b    ; Bit 0 von al auf 1
        mov dx, seport       ; Adresse von seport in dx laden
        out dx, al           ; 00000001b an seport senden, Uebernahme der Daten

        pop dx               ; Wiederherstellen der Register ax und dx
        pop ax
        ret                  ; Zurueck zum Hauptprogramm
anzeigen endp
```

2.3.7 Inttest

Die Abfrage der INT-Taste und das Umdrehen des Pfeils erfolgt über das Unterprogramm `inttest`. Dazu wird `saport` eingelesen und dessen Bit 3 ausgewertet. Ist dieses 0, so ist die INT-Taste gedrückt und die Pfeilrichtung muss geändert werden. Dies geschieht durch Umdrehen des Arrays, in dem das Bitmuster für den Pfeil gespeichert ist. Der Einfachheit halber wird das Bitmuster komplett neu gesetzt und nicht mittels eines Algorithmus Byte für Byte gespiegelt. Dieses Unterprogramm wird im Hauptprogramm nach jedem Durchlauf der Schleife zum Zeichnen des Pfeils aufgerufen.

Quelltext

```
inttest proc
        push ax
        push dx
        mov al,richtung      ; richtung in al
        mov dx,saport        ; Adresse des saports in dx speichern
        in al,dx             ; Einlesen des Saports in al
        not al               ; al invertieren
        and al,00001000b     ; alles außer Bit 3 auf Null setzen
        cmp al,0             ; vergleiche mit Null
        je  weiter1          ; wenn Null, springe weiter
        not richtung         ; (sonst) invertiere richtung

        call warte           ; Warten um die Taste zu entprellen
        call warte
        call warte
        call warte
        call warte
        call warte
        call warte

        cmp richtung,0       ; Vergleiche Richtung mit 0 (links)
        je plinks            ; falls null, springe nach Pfeil-links
                             ; sonst zeigt der Pfeil nach rechts
        call nullung         ; loesche Display
        mov char_lst[7],00011000b  ; nach rechts zeigenden Pfeil
        mov char_lst[6],00111100b  ; im Array speichern
        mov char_lst[5],01011010b
        mov char_lst[4],10011001b
        mov char_lst[3],00011000b
        mov char_lst[2],00011000b
        mov char_lst[1],00011000b
        mov char_lst[0],00011000b
        jmp weiter1                ; fertig, Sprung zum Ende
plinks:
        call nullung               ; loesche Display
        mov char_lst[0],00011000b  ; nach links zeigenden Pfeil
        mov char_lst[1],00111100b  ; im Array speichern
        mov char_lst[2],01011010b
        mov char_lst[3],10011001b
        mov char_lst[4],00011000b
        mov char_lst[5],00011000b
```

```
        mov char_lst[6],00011000b
        mov char_lst[7],00011000b
weiter1:
        pop dx
        pop ax
        ret
inttest endp
```

2.3.8 Tasttest

Die Überprüfung der Abbruchbedingung des Programms, also ein Tastendruck auf der Tastatur erfolgt im Unterprogramm `tasttest`. Genau wie `inttest` erfolgt der Aufruf in der Hauptschleife.

Quelltext

```
tasttest proc
        push ax             ; Register sichern
        mov ah,01h          ; Tastaturabfrage wartet nicht auf Eingabe
        int 16h             ; Tastaturabfrage ueber DOS Int 16h
        jz j5               ; falls Zeichen im Puffer Zero Flag 0 ansonsten 1
        mov ende,1          ; Abbruchvariable verschieden von 0 setzen
j5:     pop ax              ; gesicherte Register zurueckholen
        ret
tasttest endp
```

2.3.9 Warte

Die Warteschleife im Unterprogramm `warte` besteht aus drei ineinandergeschachtelten Schleifen, die je von einem vorher festzulegenden Wert bis Null herunterzählen. Die resultierende Verzögerung ist von der Geschwindigkeit und ggf. der sonstigen Auslastung des Prozessors abhängig.

Quelltext

```
warte proc
weiterwarten3:              ; auesserste Warteschleife
        dec warten3
weiterwarten2:              ; auessere Warteschleife
        dec warten2
weiterwarten:               ; innere Warteschleife
        dec warten          ; inneren Wartezaehler dekrementieren
        cmp warten,0        ; Zaehler mit null vergleichen
        jne weiterwarten    ; Falls nicht null weitermachen
        mov warten,255      ; nachdem Schleife beendet den Zaehler fuer den
                            ; naechsten Lauf auf 255 setzen

        cmp warten2,0       ; analog zur inneren Warteschleife
        jne weiterwarten2
        mov warten2,255

        cmp warten3,0       ; dito
```

```
        jne weiterwarten3
        mov warten3,50
        ret
warte endp
```

---

3 Probleme

Da das Programm wärend der Entwicklung (zu Hause) nicht Schritt für Schritt am Matrix-Display getestet werden konnte, gestaltete sich die Fehlersuche hier schwierig. Im Wesentlichen konnten nur syntaktische Fehler durch den Assembler und die Funktion des Unterprogramms zum Abbruch durch eine Tastatureingabe einfach überprüft werden. Der Rest mußte im Kopf durchgespielt werden (das Programm mittels des Debuggers zu untersuchen würde, wenn man nicht nach einem speziellen Fehler sucht keinen Sinn machen, bzw. extrem Zeitaufwändig sein). Die Implementierung des Programms zu Hause bereitete Dank der Vorbereitung durch das Praktikum zum NEUMANN-Rechner und die Vorlesung keine Probleme. Das Beispielprogramm zur Ansteuerung des Displays in den Praktikumsunterlagen war dem Verständnis ebenfalls sehr dienlich.
Beim Praktikum selbst traten dann folgende Fehler auf:

- Der Pfeil war viel zu schnell, auf dem Display war eine konstante Leuchtspur zu erkennen. Durch Einfügen einer Warteschleife konnte das Programm soweit heruntergebremst werden, daß der Pfeil eine niedrigere Geschwindigkeit bekam und somit auch als solcher auf dem Display zu erkennen war. Die Geschwindigkeit mit der die Schleifen abgearbeitet werden ist proportional zur Prozessorgeschwindigkeit.

- Der Pfeil wurde nach dem Sprung von links nach rechts nicht gelöscht. Dieses konnte durch einen Aufruf des Unterprogramms zur Nullung des Displays nach diesem Sprung behoben werden.

- Die Umschaltung des Pfeils mittels der `INT`-Taste wird in jedem Durchgang überprüft und führt somit bei längerem Drücken der Taste zu einem ständigen Wechsel der Richtung, bzw. kann auch bei einem einzelnen kurzen Druck mehrfach erfolgen, da während der Zeit des Tastendrucks mehrfach abgefragt wird. Dieses Problem konnte schnell dadurch behoben werden, daß direkt nach dem Registrieren des `INT` eine Warteschleife durchlaufen wird, die Länger als ein Tastendruck dauert. Somit wird nur noch eine Umschaltung, bzw. bei dauerndem Druck der `INT`-Taste nur noch ein paar Mal pro Sekunde umgeschaltet (Abhängig von der Prozessorgeschwindigkeit).

Nachdem diese Fehler beseitigt waren lief das Programm wie gefordert.

A Quelltext

```
; Festlegungen fuer den Assembler

.286                          ; Benutze Befehlssatz des 286ers
dosseg                        ; Segmentreihenfolge..
.model small                  ; Speichermodell
.stack 200h                   ; Groesze des Stacks

.data                         ; Datensegment
char_lst db 00011000b         ; Bitmuster des Pfeils in Afangsposition (links)
         db 00111100b
         db 01011010b
         db 10011001b
         db 00011000b
         db 00011000b
         db 00011000b
         db 00011000b
char_l      db 8              ; Laenge des Pfeils ist 8
ba          =378h             ; Basisadresse des Matrixdisplays
dport       =ba+0             ; Adresse des Datenregisters dport
saport      =ba+1             ;                            saport
seport      =ba+2             ;                            seport
warten   db 255               ; Laufindex der Warteschleife
warten2  db 255               ; Laufindex der Warteschleife 2
warten3  db 50                ; Laufindes der Warteschleife 3
daten    db ?                 ; Hier wird das Bitmuster einer Spalte abgelegt
dispspal db 184               ; Hier wird die Adresse abgelegt, Start bei 184
richtung db 0                 ; Richtung des Pfeiles. 0 = <-, 1 = ->^M
ende     db 0                 ; Abbruchbedingung, bei 1 bricht Programm ab



.code

; Unterprogramm zur anfaenglichen Initialisierung von seport, setzt
; Bit 0 auf 0, da zum Anzeigen hier eine 0-1-Flanke erwartet wird
; Bit 2 auf 0, so daß das Display hellgetastet ist
; Bit 3 auf 1 falls zuerst die Displayadresse eingelesen wird, fuer
; deren Uebernahme eine 1-0-Flanke benoetigt wird

initseport proc
        push ax               ; Retten der Register ax und dx auf den Stack
        push dx

        mov al,00001000b      ; Setze al auf Initialisierungswert
        mov dx,seport         ; Lade Adresse von seport in dx
        out dx,al             ; geben Initialisierungswert an seport aus

        pop dx                ; Wiederherstellen der Register in entgegen-
        pop ax                ; gesetzer Reihenfolge, da FILO-Stack
        ret                   ; Return
```

```
initseport endp


; Unterprogramm welches seport Bit 3 auf 1 schaltet, somit wird dport in den
; Datenspeicher übernommen

dportds proc
        push ax              ; Register ax auf Stack sichern
        push dx              ;          dx

        mov  ah, 00001000b   ; Bit 3 = 1, rest 0 auf das Register kopiert
        mov  al, ah          ; ah kann nicht fuer out benutzt werden
        mov  dx, seport      ; adresse des sexpoprts in dx laden
        out  dx, al          ; 00001000b an seport, Übernimmt dport in
                             ; den Datenspeicher

        pop dx               ; dx zurückholen (FILO!)
        pop ax               ; ax zurückholen
        ret                  ; zurück zum Hauptprogramm
dportds endp

;  Unterprogramm welches seport Bit 3 auf 0 schaltet, somit wird dport in den
;  Adressspeicher uebernommen

dportas proc
        push ax              ; Register ax auf Stack sichern
        push dx              ;          dx

        mov  ah, 00000000b   ; Bit 3 = 0  auf das Register kopiert
        mov  al, ah          ; ah kann nicht fuer out benutzt werden
        mov  dx, seport      ; Adresse des sexpoprts in dx laden
        out  dx, al          ; 00000000b an seport, Übernimmt dport in
                             ; den Adressspeicher

        pop dx               ; dx zurückholen (FILO!)
        pop ax               ; ax zurückholen
        ret                  ; zurück zum Hauptprogramm
dportas endp


; Unterprogramm welches den Inhalt der Variable daten an das Displayregister
; dport übergibt

datendport proc
        push ax
        push dx

        mov al, daten        ; Laden des Datums in al
        mov dx, dport        ; Laden der Adresse von dport in dx
        out dx, al           ; Ausgabe von daten an dport
```

```
        pop dx
        pop ax
        ret
datendport endp


; Unterprogramm welches die Display-Adresse an dport uebermittelt
; Adresse gespeichert in dispspal

dispspaldport proc
        push ax
        push dx

        mov al, dispspal     ; Laden der aktuellen Displayspalte in al
        mov dx, dport        ; Laden der Adresse von dport in dx
        out dx, al           ; Ausgabe der Displaysspalte an dport

        pop dx
        pop ax
        ret
dispspaldport endp


; Unterprogramm welches die Daten der beiden Zwischenspeicher fuer
; Adresse und Datum zur Anzeige uebermittelt
; dazu muß Bit 0 des seports von 0 auf 1 springen

anzeigen proc
        push ax
        push dx

        or  al, 00000001b    ; Bit 0 von al auf 1
        mov dx, seport       ; Adresse von seport in dx laden
        out dx, al           ; 00000001b an seport senden, Uebernahme der Daten

        pop dx
        pop ax
        ret
anzeigen endp

; Unterprogramm zur Abfrage der INT-Taste. Falls diese gedrueckt wird,
; aendert sich die Richtung

inttest proc
        push ax
        push dx
        mov al,richtung      ; richtung in al
        mov dx,saport        ; Adresse des saports in dx speichern
        in al,dx             ; Einlesen des Saports in al
        not al               ; al invertieren
        and al,00001000b     ; alles außer Bit 3 auf Null setzen
```

```
        cmp al,0             ; vergleiche mit Null
        je  weiter1          ; wenn Null, springe weiter
        not richtung         ; (sonst) invertiere richtung

        call warte           ; Warten um die Taste zu entprellen
        call warte
        call warte
        call warte
        call warte
        call warte
        call warte

        cmp richtung,0       ; Vergleiche Richtung mit 0 (links)
        je plinks            ; falls null, springe nach Pfeil-links
                             ; sonst zeigt der Pfeil nach rechts
        call nullung         ; loesche Display
        mov char_lst[7],00011000b  ; nach rechts zeigenden Pfeil
        mov char_lst[6],00111100b  ; im Array speichern
        mov char_lst[5],01011010b
        mov char_lst[4],10011001b
        mov char_lst[3],00011000b
        mov char_lst[2],00011000b
        mov char_lst[1],00011000b
        mov char_lst[0],00011000b
        jmp weiter1                ; fertig, Sprung zum Ende
plinks:
        call nullung               ; loesche Display
        mov char_lst[0],00011000b  ; nach links zeigenden Pfeil
        mov char_lst[1],00111100b  ; im Array speichern
        mov char_lst[2],01011010b
        mov char_lst[3],10011001b
        mov char_lst[4],00011000b
        mov char_lst[5],00011000b
        mov char_lst[6],00011000b
        mov char_lst[7],00011000b
weiter1:
        pop dx
        pop ax
        ret
inttest endp

; Unterprogram welches auf ein Zeichen im Tastaturpuffer testet
; falls dies zutrifft wird die Variable ende auf 1 gesetzt

tasttest proc
        push ax              ; Register sichern
        mov ah,01h           ; Tastaturabfrage wartet nicht
        int 16h              ; Tastaturabfrage
        jz j5                ; falls Zeichen im Puffer Zero Flag 0 ansonsten 1
        mov ende,1           ; Abbruchvariable verschieden von 0 setzen
j5:     pop ax               ; gesicherte Register zurueckholen
```

```
        ret
tasttest endp

; Unterprogramm zum loeschen des Displays durch Ueberschreiben mit Nullen

nullung  proc
        mov ah,dispspal      ; alte Display-Position merken
        push ax
        push bx              ; Register sichern

        mov  daten,00000000b ; Leere Spalte
        mov bl,192           ; Zaehler/Adresse auf 192 (letzte Spalte+1) setzen
nullweiter:
        dec bl               ; Zaehler dekrementieren
        mov dispspal,bl      ; Zaehler => Displayspalte
        call dispspaldport   ; Aufruf zur Uebermittlung der Adresse
        call dportas         ; Aufruf zum Speichern der Adresse
        call datendport      ; Aufruf zur Uebermittlung des Datums
        call dportds         ; Aufruf zum Speichern des Datums
        call anzeigen        ; Display-Anzeige aktualisieren

        cmp bl,0             ; Vergleiche ob die Adresse schon Null ist
        jne nullweiter       ; wenn noch nicht, wiederhole
        pop bx
        pop ax
        mov dispspal,ah      ; Wiederherstellen der alten Display-Position
        ret
nullung endp

; Unterprogramm welches ein Delay erzeugt

warte proc
weiterwarten3:               ; auesserste warteschleife
        dec warten3
weiterwarten2:               ; auessere warteschleife
        dec warten2

weiterwarten:                ; innere Warteschleife
        dec warten           ; inneren Wartezaehler dekrementieren
        cmp warten,0         ; Zaehler mit Null vergleichen
        jne weiterwarten     ; Falls nicht Null weitermachen
        mov warten,255       ; nachdem Schleife beendet den Zaehler fuer den
                             ; naechsten Lauf auf 255 setzen

        cmp warten2,0        ; analog zur inneren Warteschleife
        jne weiterwarten2
        mov warten2,255

        cmp warten3,0        ; dito
        jne weiterwarten3
        mov warten3,50
```

```
        ret
warte endp



; ################## HAUPTPROGRAMM ######################

beginn: mov dx,@data          ; Anfangsadresse des Datensegment ins Register
        mov ds,dx
        call initseport       ; initialisiere seport

rechts: call nullung          ; Display loeschen^M
        mov dispspal,184      ; setze Position auf 184 (ganz rechts)

anfang:
        cmp dispspal,127      ; vergleiche Position mit 128 (= Minimum)
        jbe rechts            ; falls linkes Ende erreicht, springe nach rechts

        mov al,char_l         ; in al die Laenge des Zeichens (8) laden
        mov bh,0              ; Laufindex bx, obere 8 bit auf 0
        mov bl,0              ; Laufindex bx, untere 8 bit auf 0

weiter:                       ; weitere Spalte muss geschrieben werden
        call dispspaldport    ; lade Displayposition in dport
        call dportas          ; dport in Adressspeicher uebernehmen
        mov dh,char_lst[bx]   ; Lade bxte Spalte nach daten
        mov daten,dh
        call datendport       ; daten in dport laden
        call dportds          ; dport in Datenspeicher laden
        call anzeigen         ; Werte aus Speichern anzeigen
        inc dispspal          ; gehe eine Spalte weiter nach rechts
        inc bl                ; Laufvariable erhoehen
        cmp bl,al             ; wurden alle Spalten geschrieben?
        jne weiter            ; sonst naechste Spalte schreiben

        call dispspaldport    ; lade displayposition in dport
        call dportas          ; dport in Adressspeicher uebernehmen
        mov daten,00000000b   ; leere Zeile zum loeschen des letzten Pfeilteils
        call datendport       ; zum dport schicken
        call dportds          ; in den Datenspeicher damit
        call anzeigen         ; Werte aus Speichern anzeigen (leere Zeile)
        sub dispspal,9        ; 9 nach links springen

        call warte            ; Verzoegerung damit Pfeil nicht zu schnell ist
        call nullung          ; Display loeschen

        call inttest          ; ueberpruefe int-Taste
        call tasttest         ; ueberpruefe ob Taste auf Tastatur gedrueckt
        cmp ende,1            ; vergleiche ende mit 1
        je  schluss           ; springe zum schluss wenn dies erfuellt ist
        jmp anfang            ; (sonst) springe zum anfang, neuer pfeil wird
```

```
                              ; gezeichnet

schluss: mov ah,4Ch           ; Vorbereitung des Interruptaufrufs
          int 21h             ; Int 21h beendet das Programm

end beginn
```