DTI

Stand: 2.4.92

Dies Handbuch soll die Anwendung des Display- und Tastaturinterfaces unterstützen und die Funktionsweise der Schaltung erklären. Sollte Ihnen ein Fehler auffallen, so verständigen Sie bitte den Verfasser:

Dipl.-Ing. Michael Schmidt
analoge und digitale Elektronik

Aureliusstr.. 22
52072 Aachen

Tel.: 02 41/ 2 05 22
Fax.: 02 41/ 40 89 58

Inhaltsverzeichnis

Funktionsbeschreibung DTI V1.4

Ein EC-Bus-System stellt eine hervorragende Basis zur Entwicklung autarker Meß-, Steuer- und Regelgeräte dar. Was nützt jedoch der leistungsfähigste Rechnerkern, wenn der Anwender keine Möglichkeit hat, interaktiv in den Programmablauf einzugreifen. Aufgrund dieser Überlegungen ist das Display- und Tastaturinterface DTI V1.4 entstanden.

Mit dem Interface können Sie eine Matrixtastatur mit maximal 16 Tasten oder auch einzelne Taster an die Controllerkarte koppeln. Eine Interruptsteuerung des Rechners durch die Tastatur ist möglich. Damit zeitkritische Programmteile nicht von den Eingaben unterbrochen werden, läßt sich die Interruptsteuerung während der Laufzeit ein- und ausschalten. Durch den verwendeten integrierten Tastaturbausteins MM74C922 ist es nicht mehr nötig per Software die Tasten zu entprellen. Nach einer Unterbrechungsanforderung wird der Tastencode (Dualcode zwischen 0 und 15) einfach unter der Tastaturenadresse ausgelesen.

Schaltungsbeschreibung:
Auf der Controllerkarte 8052-ECB/BasiControl ist der Adreßraum von F800h bis FF7Fh für eine aufsteckbare Erweiterung reserviert. Das Display- und Tastaturinterface DTI V1.4 belegt den Bereich ab FF00h bis FF7Fh. Der NAND-Baustein 7430 (IC1) dekodiert mit acht Eingängen das obere Adreßbyte (FFh). Die feinere Adreßauswahl nimmt der GAL-Baustein 16V8 (IC3) vor. Auf den Anschlüssen der Controllerkarte stehen die gemultiplexten Adreß- und Datenbits zur Verfügung. Der achtfach Latch-Baustein 74HCT573, IC2, demultiplext die Adressen für das GAL und das Display.
Die Adressen FF00h bis FF03h werden für die Steuerung des LCD-Displays benötigt. Die entsprechende Freigabeleitung ist mit E (enable) bezeichnet.
Aus dem Tastaturcontroller lassen sich die Daten unter der Adresse FF40h auslesen. Dazu gibt die Steuerleitung /OE (output enable) die Tristate-Ausgänge des ICs frei.
Um den Interrupt des Tastaturchips ein- und auszuschalten wurde mit dem GAL ein Flipflop realisiert. Ein Schreibzugriff auf die Adressen FF42h und FF43h erzeugt ein Low-Signal auf dem GAL-Ausgang CLKO (clock out, Pin 17) mit der Dauer des Write-Signals. Die Leitung ist rückgekoppelt auf den Takteingang des Bausteins (Pin 1). Der Wert des Adreßbits A0 wird nun in einem D-Flipflop im GAL gespeichert und steht am Ausgang /INTEN (interrupt enable, Pin 19) zur Verfügung. Das bedeutet, daß ein Schreibzugriff auf die Adresse FF42h den Ausgang /INTEN auf Null setzt und ein Zugriff auf FF43h ihn wieder auf High-Pegel setzt. Das Datum spielt dabei keine Rolle.

Adresse	Zugriff		Aktion
0FF40h	lesen	/RD=0	Datum aus Tastatur lesen
0FF42h	schreiben	/WR=0	Interrupt freigeben (Datum beliebig)
0FF43h	schreiben	/WR=0	Interrupt sperren (Datum beliebig)

Die Dekodierung der Adressen ist unvollständig. Die Leitungen A3, A4 und A5 werden in dem GAL-Baustein nicht ausgewertet, so daß einige Spiegeladressen existieren. Da der Bereich des Interfaces nicht anderweitig benutzt wird ist diese Tatsache jedoch bedeutungslos.

Tastaturdecoder:
Der Tastaturchip MM74C922, IC5, ist ein schon etwas betagter Baustein, zu dem es jedoch keine modernere Alternative gibt. Das CMOS-IC enthält die gesammte Logik, um eine Gruppe von bis zu 16 Tasten in einen reinen Dualcode zu kodieren. Die Tasten, angeordnet in einer 4*4 Matrix, werden sequentiell abgetastet. Die Taktrate ist durch den externen Kondensator C2 bestimmt. Die Schaltung entprellt automatisch die Schalter. Ist ein Tastendruck registriert, so wird der Oszillator für kurze Zeit angehalten. In dieser Haltezeit, werden alle weiteren Schaltspiele ignoriert. Die Entprellzeit ist durch den Kondensator C1 bestimmt. C1 soll laut Hersteller die zehnfache Kapazität von C2 haben. Die Kondensatoren können je nach Prelldauer den verwendeten Tasten angepaßt werden.
Solange eine Taste gedrückt wird liegt am Ausgang DA (data available, Pin 12) High-Pegel. Damit wird das Gate des V-MOS-FET V1 angesteuert. Ist, wie oben beschrieben, der Interrupt freigegeben, liegt am Source-Anschluß logisch Null und der Transistor leitet. Je nach dem, welcher Jumper gesetzt ist, wird ein Interrupt am Prozessor ausgelöst.
Der Ausgang DA geht nach dem Loslassen einer Taste auch dann auf Null, wenn noch eine andere Taste gedrückt ist. Der Ausgang nimmt dann, nach einer Entprellperiode, wieder High-Pegel an, um die neue Eingabe zu kennzeichnen. Dieses Zwei-Tasten-Rollover besteht zwischen allen Tasten der Matrix. Interne Latches speichern die jeweils letzte Eingabe auch dann, wenn die Taste losgelassen wurde. Die Tristate-Ausgänge stellen den vier-Bit-Code auf dem Prozessorbus zur Verfügung wenn /OE auf Null geht.

Bei der Programmierung ist zu beachten, daß die oberen Datenbits undefiniert sind. Sie werden deshalb maskiert. Die Taste zwischen den Anschlüssen X1 (Pin 10) und Y1 (Pin 9) erzeugt den Code 0. Dies entspricht nicht der Information "keine Taste gedrückt". Es wird immer der Code der letzten Taste ausgegeben. Ob eine Eingabe erfolgte, kann man also nur durch den Interrupt feststellen. Der Jumper J1 legt den Interrupt auf den Prozessoreingang /INT1. Die Unterbrechungsanforderung an diesem Pin läßt sich auch in Basic ausnutzen. Das Beispiel zeigt, wie eine Abfrage realisiert werden kann:

```
10        REM Hauptprogramm
20        REM Interrupt freigeben:
30        ONEX1 1000
40        XBY(0FF42h) = 0
50        PRINT" Kaffee(1) oder Tee(2)?"
60        FOR i = 0 TO 1000: NEXT i
70        REM Sperren des Interrupts:
80        XBY(0FF43h) = 0
90        REM Auswertung:
100       IF tast = 1 THEN PRINT " Kaffee"
110       IF tast = 2 THEN PRINT " Tee"
          ...
```

Für eine weitere Abfrage ist der Interrupt erneut freizugeben wie in Zeile 40 gezeigt.

```
1000      REM Unterprogramm zur Tastaturabfrage
1010      REM Code lesen und maskieren:
1020      tast = XBY(0FF40h) .AND. 0Fh
1030      REM Rücksprung:
1040      RETI
```

Display:
Zur Anzeige werden heute überwiegend die preiswerten LCD-Displays mit dem integrierten Hitachi-Controller HD 44780 eingesetzt. Sie sind mit Anzeigegrößen von 1 Zeile * 16 Zeichen bis zu 2 Zeilen * 40 Zeichen verfügbar und können mit dem Interface ebenfalls an den Controller gekoppelt werden. Dabei ist es möglich, die Daten und Befehle auch aus dem Display zurückzulesen. Die Programmierung wird dadurch erheblich erleichtert.

Außer einer Betriebsspannung von +5V benötigen einige Displaytypen (je nach Hersteller) zusätzlich eine negative Hilfsspannung zur Kontrasteinstellung. Diese Spannung wird von dem Interface bereitgestellt. Dazu ist ein integrierter Spannungswandler (Ladungspumpe) ICL7660, IC4 vorgesehen. Am Ausgang des Wandler-ICs stehen -5V zur Verfügung. Das nachgeschaltete Trimmpoti läßt eine Variation der Kontrastspannung zwischen -5V und +5V zu.

Neben dem 8-Bit-Datenbus sind noch einige Steuerleitungen notwendig:
Der Anschluß R/W (read/write) unterscheidet zwischen Schreib- (R/W = 0) und Lesezugriffen (R/W = 1).
Das Display hat intern verschiedene Register: Für die angezeigten Daten das DD-RAM, für selbstdefinierte Zeichen das CG-RAM und einige Befehls- oder Instruktionsregister (IR) für die Steuerbefehle. Die Umschaltung erfolgt mit der Funktionsauswahlleitung RS (register select). RS = 0 wählt den IR-Speicher aus. Hier werden die Betriebsarten eingestellt und die Startadressen für DD- oder CG-RAM initialisiert. Ein Lesezugriff mit RS = 0 liefert das Busy-Flag und den Wert des Adreßzählers. RS = 1 aktiviert die Datenregister, DD- und CG-RAM, je nach voreingestellter Adresse. Hier werden die anzuzeigenden ASCII-Werte oder das Bitmuster selbstdefinierter Zeichen eingeschrieben. Die Daten lassen sich auch zurücklesen.
Das Display wird mit der E-Leitung (enable) aktiviert. Der Anschluß ist vergleichbar mit den üblichen CS-Eingängen eines Speicherbausteins, jedoch logisch Eins aktiv.
Die LCD-Displays erfordern ein etwas anderes Timing bezüglich der Steuerleitungen, als es bei den Intel- und Zilog-Prozessoren üblich ist. Die Funktionsauswahlleitungen RS und R/W müssen als erste in einen stabilen Zustand schalten. Frühestens 140ns später darf die Freigabeleitung E logisch Eins annehmen, womit der Baustein aktiv wird. Dieser Zustand muß mindestens 450ns stabil bleiben. Die Daten sollen 200ns vor der 1-0-Flanke an E, also vor dem Ende eines Schreibzyklus, oder früher anliegen. RS, R/W und die Daten sollen auch eine kurze Zeit (10ns) nach Ende des Enable-Signals gültig bleiben.
Es kann also nicht einfach die Leitung R/W mit RD des Prozessors verbunden werden, da das Schreib- oder Lesesignal der CPU immer erst nach der Bausteinauswahl, nach Ausgabe der Adressen, aktiv wird. Umgekehrt darf der E-Eingang des Displays nicht mit einem üblichen Adreßdecoder angesteuert werden. Ein einfacher Trick führt aus diesem Dilemma: R/W und RS werden mit den Adreßleitungen A0 und A1 verbunden. Das Enable-Signal wird nur gleichzeitig mit /RD oder /WR der CPU aktiv, also hinreichend verzögert und in einem Zeitraum, in dem die Datenleitungen gültig sind. Das Timing wird eingehalten, solange die Taktfrequenz des 8052 unter 12MHz bleibt.
Das Adreßdecoder-GAL verhindert gleichzeitig einen Schreibzugriff auf eine Adresse unter der nur Lesen gestattet ist und umgekehrt. Es ergibt sich folgende Adressierung:

Adresse	R/W	RS	Zugriff		Aktion
0FF00h	0	0	schreiben	/WR=0	Befehlsregister schreiben
0FF01h	1	0	lesen	/RD=0	Befehlsregister lesen
0FF02h	0	1	schreiben	/WR=0	Datenregister schreiben
0FF03h	1	1	lesen	/RD=0	Datenregister lesen

Der Anschluß des Displays an das Interface erfolgt über einen 14poligen Pfostenstecker und Flachbandkabel. Beachten Sie bitte, daß teilweise die Durchnummerierung der Pins am Display umgekehrt verläuft oder einige Anschlüsse vertauscht sind. Die Angaben im Schaltplan beziehen sich auf die Displays von Sharp LM16255 und Hitachi LM016; Pin 14 befindet sich am linken Rand der Anzeigeplatine und ist mit der Datenleitung D7 verbunden.

Für das Hitachi-Display LM016 gibt es auch Datenblätter, in denen die Nummerierung der Anschlüsse vertauscht wurde. Pin 1 liegt dann außen, Pin 14 in der Mitte der Platine, die Datenleitung D7 ist jedoch immer am Platinenrand.

Die Firma Epson hat es geschafft, bei Ihren Displays die Anschlußreihenfolge und -zählrichtung zyklisch zu verändern und gelegentlich auch doppelreihige Verbindungen vorzusehen. Seiko und Densitron verwenden ebenfalls doppelreihige Anschlüsse mit unterschiedlicher Zählrichtung. Beim Seiko M1632 sind zusätzlich die +5V- und Masse-Anschlüsse vertauscht.

Verwenden Sie also bitte keine Displays, deren Anschlußbelegung Ihnen nicht bekannt ist! Sowohl das Display als auch die Controllerkarte könnten Schaden nehmen.

Beispiel für eine Befehlserweiterung des MCS BASIC-52 zur Displayansteuerung

Zum Testen des Display- und Tastaturinterfaces kann man z.B. das unten angegebene Programm assemblieren. Es wurde mit dem Shareware-Programm TASM erstellt. Verwendet man einen anderen Crossasembler sind evtl. einige Anpassungen im Quellcode vorzunehmen.
Das abgedruckte Maschinenprogramm soll verdeutlichen, wie man eigene Basicbefehle zur Unterstützung des DTIs erstellt. Das Hauptprogramm PRJ.ASM bleibt, auch für andere Befehlserweiterungen, immer gleich. Es teilt dem Basicinterpreter die Existenz der neuen Befehle mit. Als Beispiel ist hier ein Initialisierungsbefehl und eine Ausgaberoutine für das Display abgedruckt.

Die Befehlserweiterung DISPINIT steuert eine LCD-Anzeige vom Typ LM 16255 o.ä mit 2 Zeilen und 16 Zeichen/Zeile. Folgende Aktionen werden ausgeführt:

Display löschen und Cursor in Grundstellung
Bewegungsrichtung des Cursors nach rechts
Anzeige wird nicht verschoben
Anzeige ein, Cursor aus, kein Blinken des Zeichens über dem Cursor
5*7 Dots Standardzeichensatz

Dabei wird das Busyflag abgefragt, so daß der Rechner auf die Ausführungszeit des Displays wartet.

Die Befehlserweiterung PRINT@ funktioniert entsprechend einer normalen PRINT-Anweisung, leitet aber die Ausgabe auf das Display um. Auch LIST@ und PH0.@. arbeiten auf den gleichen Pfad. PRINT@, LIST@ und PH0.@ arbeiten sowohl im Kommando- als auch im Programm-Modus. Zusätzlich sind folgende Befehle implementiert:

Befehle:
\1 Ausgabe am Anfang der ersten Zeile
\2 Ausgabe am Anfang der zweiten Zeile
\n Ausgabe am Anfang der nächsten Zeile
\\ Ausgabe eines Zeichens ähnlich dem \
CR Ausgabe am Anfang der selben Zeile
LF Ausgabe am Anfang der nächsten Zeile

1 und 2 sind die ASCII-Zeichen!
Alle anderen Zeichen werden direkt an das Display ausgegeben.
TAB, SPC und USING funktioniert wie bisher.

```
Beispiel: 10 PRINT@ "\1dies ist die \2zweite Zeile"
          20 PRINT@ "\1dies wird wohl",CR,
          30 PRINT@ "überschrieben"
```

```
;    PRJ.ASM   Projektdatei zum Einbinden diverser
;    Basicerweiterungen.               9.5.91
;
;-------------------------------------------------------
;    Damit ein EPROM-Programmierer ein voll-
;    ständiges File von 16KByte Länge erhält
;    ist es nötig, daß die erste und letzte
;    Adresse auftaucht (siehe -f00 Zusatz):
;
          .ORG 0000h
;
;-------------------------------------------------------
;    Um die Special Funktion Register des 8051 zu
;    nutzen müssen zusätzliche Mnemonics aus der
;    Datei 8051.H eingebunden werden:
;
#include  "8051.H"
;
;-------------------------------------------------------
;    Die Register des Displays:
;
;    Befehlsregister schreiben: Adresse 0FF00h
WR_CTRL   .EQU 0FF00h
;
;    Befehlsregister lesen    : Adresse 0FF01h
RD_CTRL   .EQU 0FF01h
;
;    Datenregister schreiben  : Adresse 0FF02h
WR_DATA   .EQU 0FF02h
;
;    Datenregister lesen      : Adresse 0FF03h
RD_DATA   .EQU 0FF03h
;
;
;    Wie sag ich's meinem Rechner?
;-------------------------------------------------------
;    Steht im Programmspeicher an der Adresse 2002h
;    das Datum 5Ah so erwartet der Basicinterpreter
;    eine Befehlserweiterung. Das stellt er wohl
;    schon beim Einschalten fest...
;
BAS_INIT  .ORG 2002h
          .DB 5Ah
;
;-------------------------------------------------------
;    Der Interpreter führt das Programm an der
;    Adresse 2048h im Codespeicher aus. Die Routine
;    setzt das Bit 45, das Option-Bit. (Adresse
;    2Dh, das 5te Bit des internen bitadressierbaren
;    Speichers Adresse 37 dez.) Damit erwartet der
;    Basicinterpreter eine Befehlserweiterung.
;
S_OPT_BIT .ORG 2048h
          setb 45
;
;    und zurück:
;
          ret
;
;-------------------------------------------------------
;    Nachdem Bit 45 gesetzt ist, testet der Inter-
;    preter beim Tokenisieren einer Zeile jedesmal
;    den Lookup table. Dessen Adresse wird in einem
;    Programm an der Adresse 2078h in den Data-
;    Pointer (DPTR) geladen:
;
          .ORG 2078h
```

```
          mov  DPTR, #LOOKUP_T
          ret
;
;-------------------------------------------------------
;    Die Adresse für das Label LOOKUP_T wird im
;    folgenden festgelegt:
;
LOOKUP_T  .ORG 2100h
;
;    Die verfügbaren Tokens sind 10h bis incl.
;    1Fh. Die Tabelle beginnt mit dem Token
;    (Nummer), dem dazugehörigen Namen (Befehl,
;    ASCII) und einer Null (00h) als Zeichen, das
;    weitere Tokens folgen. Der letzte Name wird
;    mit 0FFh abgeschlossen. Die Reihenfolge ist
;    beliebig.
;
          .DB 10h        ;Token
          .DB "DISPINIT" ;Display initialisieren
          .DB 0FFh       ;Ende der Tabelle
;
;-------------------------------------------------------
;    Nun kennt 8052-Basic die neuen Tokens, nimmt
;    die Befehle an und gibt sie auch wieder aus.
;    Nur die Ausführung hapert noch...
;
;    Möchte der Interpreter einen unserer neuen
;    Befehle ausführen (nach run oder im Command-
;    Modus), so testet er zuerst das Bit 45 (Option-
;    Bit, siehe oben), dann wird ein Programm an
;    der Adresse 2070h ausgeführt, welches die
;    Adresse der Vektor Tabelle übergibt. Die
;    Adresse wird in den DPTR geschrieben.
;
          .ORG 2070h
          mov DPTR,#VEKTOR_T
          ret
;
;-------------------------------------------------------
;    Die Vektor Tabelle wiederum besteht aus
;    den Einsprungadressen für die Basic-
;    erweiterungen (Assemblerroutinen). Die
;    Reihenfolge hier muß mit der in dem Lookup-
;    table übereinstimmen!
;
```

```
VEKTOR_T  .ORG 2200h
;
;    Diese Adresse wurde wieder frei gewählt,
;    und dort steht:
;
          .MSFIRST
;    in der richtigen Reihenfolge (MSB-LSB)
;
          .DW DISPINIT
;
;    usw.
          .LSFIRST
;    ...sonst stimmt nichts mehr
;
;-------------------------------------------------------
;    Nun kommt der eigentliche Code für die
;    Basicerweiterungen:
;         (das wurde auch Zeit...)
;
;    Display initialisieren:
;
DISPINIT  .ORG 2300h
#include "DISPINIT.ASM"
;
;-------------------------------------------------------
;    Einbinden der PRINT@-Erweiterung:
;    Die Ausgaberoutine soll bei Adresse 403Ch an-
;    fangen (32kByte EPROM), oder bei Adresse 003Ch
;    (16kByte EPROM)
;
PRINT     .ORG 003Ch
#include  "PRINT.ASM"
;
;-------------------------------------------------------
;    Ende des Quälcodes und auffüllen auf
;    16KByte Länge:
;
          .ORG 3FFFh
;
          .END


;    Basicerweiterung DISPINIT.ASM         6.5.91
;
;    Routine zum Initialisieren des LCD-Displays
;    Typ: LM 16255 uam.  2 Zeilen, 16 Zeichen
;
;-------------------------------------------------
;    Die Register des Displays:
;
;    Befehl schreiben: WR_CTRL
;
;    Befehl lesen    : RD_CTRL
;
;    Daten schreiben : WR_DATA
;
;    Daten lesen     : RD_DATA
;
;-------------------------------------------------
```

```
;    und was kann es...
;    Befehl schreiben:
;
;    Display löschen und Cursor in Grundstellung
;    0 0 0 0 0 0 0 1 bin, 01h,          t=1,6ms
;    wird an dritter Stelle ausgeführt. Code 01h
;
;    Cursor in Grundstellung, verschobene Anzeige
;    in Ausgangsposition, DD-RAM bleibt erhalten:
;    0 0 0 0 0 0 1 x bin, 02h,          t=1,6ms
;    wird hier nicht ausgeführt.
;
;    Bewegungsrtg. des Cursors, Anzeige verschieben
;    (wird während des Lesens der Daten ausgeführt):
;    0 0 0 0 0 1 i/d s bin,             t=40us
;    i/d 1:increment, 0:decrement (Richtung)
;     s  1:scroll,    0:no scroll (Anzeige)
;    wird an zweiter Stelle ausgeführt. Code 06h:
;    increment, no scroll
;
;    Anzeige an/aus, Cursor an/aus, Blinken des
;    Zeichens über dem Cursor:
;    0 0 0 0 1 d c b bin,               t=40us
;     d  1:Anzeige an, 0:Anzeige aus
;     c  1:Cursor an,  0:Cursor aus
;     b  1:Blinken an, 0:Blinken aus
;    wird an vierter Stelle ausgeführt. Code 0Ch:
;    Anzeige an, kein Cursor, kein Blinken
;
;    Bewegen des Cursors und Verschieben der
;    Anzeige ohne DD-RAM zu ändern:
;    0 0 0 1 s/c r/l x x bin,           t=40us
;    s/c 1:Display schieben, 0:Cursor schieben
;    r/l 1:nach rechts,      0:nach links
;    wird hier nicht ausgeführt.
;
;    Datenwortlänge, Anzahl der Zeilen und
;    Zeichenvorrat:
;    0 0 1 dl n f x x bin,              t=40us
;    dl  1:8Bit       0:4Bit
;     n  1:2 Zeilen   0:1 Zeile
;     f  1:5*10 Dots  0:5*7 Dots
;    wird als erstes ausgeführt. Code 38h:
;    8 Bit, 2 Zeilen, 5*7 Dots
;
;    CG-RAM Adresse setzen vor dem Schreiben:
;    0 1 a5 a4 a3 a2 a1 a0 bin,         t=40us
;    8Byte/Zeichen * 8 Zeichen
;    wird hier nicht ausgeführt, nur interner
;    Zeichensatz.
;
;    DD-RAM Adresse setzen vor dem Lesen:
;    1 a6 a5 a4 a3 a2 a1 a0 bin,        t=40us
;    80h Startadresse erste Zeile
;    C0h Startadresse zweite Zeile
;    wird als fünftes ausgeführt und setzt
;    Adresse auf die erste Zeile 80h.
;
;-------------------------------------------------
```

```
;    Befehl lesen:
;
;    Busyflag und Adresse lesen:
;    bf a6 a5 a4 a3 a2 a1 a0 bin,    t=40us
;    bf 1:beschäftigt 0:ready
;
;-------------------------------------------------
;    Daten schreiben:
;
;    8Bit ACSII ähnlich                 t=40us
;
:-------------------------------------------------
;    Daten lesen:
;
;    8Bit w.o.                          t=40us
;
;-------------------------------------------------
;    schön wenn Sie's verstanden haben,
;    here we go:
;
;    Mitteilung an den Basicinterpreter, daß eine
;    anwenderspezifische Ausgaberoutine existiert.
;    (in der Datei PRINT.ASM)
;    Wenn dieser Programmpunkt noch nicht ausgeführt
;    wurde, gibt Basic eine Ausgabe mit Print@ an
;    das Terminal. Also erst Dispinit und danach
;    Print@.
;
;    Setzen von Bit 27h (39dez.), das 7te Bit des
;    internen bitorientierten Speicher 24h (36dez.):
;
          setb 39
;-------------------------------------------------
;    Reservieren und Null setzen eines internen
;    Bytes für Kontrollzwecke in PRINT@ Adresse 20h:
;    Benötigt wird Bit 07h (Adr 20h, 32dez,
;    das 7te Bit)
;
          mov 20h, #00h
;
;-------------------------------------------------
;    Datenformat bestimmen
;    8Bit, 2 Zeilen, 5*7 Dots =38h:
;
          mov DPTR, #WR_CTRL
          mov A, #38h
          movx @DPTR, A
;
;    und dann warten bis dies verdaut ist:
;
          acall WAIT
;
;-------------------------------------------------
```

```
;    Cursor nach rechts, Anzeige steht 06h:
;
          mov DPTR, #WR_CTRL
          mov A, #06h
          movx @DPTR, A
;
          acall WAIT
;
;-------------------------------------------------
;    Display löschen, Cursor-Grundstellung 01h:
;
          mov DPTR, #WR_CTRL
          mov A, #01h
          movx @DPTR, A
;
          acall WAIT
;
;-------------------------------------------------
;    Anzeige an, Cursor aus, kein Blinken 0Ch:
;
          mov DPTR, #WR_CTRL
          mov A, #0Ch
          movx @DPTR, A
;
          acall WAIT
;
;-------------------------------------------------
;    wenn wir nicht schieben reicht dies schon,
;    nun in die erste Zeile 80h:
;
          mov DPTR, #WR_CTRL
          mov A, #80h
          movx @DPTR, A
;
          acall WAIT
;
;-------------------------------------------------
;    und fertig!
;
          ret
;
;-------------------------------------------------
WAIT
;    Die Warteschlange für das LCD-Display
;    benutzt DPTR und A:
;
          mov DPTR, #RD_CTRL
          movx A, @DPTR
;
;    nun enthält der Akku in seinem 7ten Bit das
;    Busy-Flag des Displays. Springe wenn Eins:
;
          jb ACC.7, WAIT
;
;    geht weiter wenn Flag gleich Null
;
          ret
;
```

```
;    Basicerweiterung PRINT@, PRINT.ASM       9.5.91
;
;    Ausgabe auf LCD-Display. Der Befehl ist bereits
;    enthalten und muß nicht im Lookup-table auf-
;    geführt werden.
;
;-------------------------------------------------
;    Befehle:
;    \1   Ausgabe am Anfang der ersten Zeile
;    \2   Ausgabe am Anfang der zweiten Zeile
;    \n   Ausgabe am Anfang der nächsten Zeile
;    \\   Ausgabe eines Zeichens ähnlich dem \
;    CR   Ausgabe am Anfang der selben Zeile
;    LF   Ausgabe am Anfang der nächsten Zeile
;
;    1 und 2 sind die ASCII-Zeichen!
;    Alle anderen Zeichen werden direkt an das
;    Display ausgegeben.
;
;-------------------------------------------------
;    Die Ausgaberoutine soll bei Adresse 403Ch an-
;    fangen (32kByte EPROM), oder bei Adresse 003Ch
;    (16kByte EPROM).
;    Mitteilung an den Basicinterpreter, daß eine
;    anwenderspezifische Ausgaberoutine existiert:
;    Setzen von Bit 27h (39dez.), das 7te Bit des
;    internen bitorientierten Speicher 24h (36dez.)
;    Dies geschieht in der Routine "DISPINIT.ASM".
;    (Wenn nicht gibt Basic an die Console aus)
;
;-------------------------------------------------
;    Die Register des Displays sind in der Haupt-
;    datei "PRJ.ASM" angegeben:
;
;    Befehl schreiben: WR_CTRL
;
;    Befehl lesen    : RD_CTRL
;
;    Daten schreiben : WR_DATA
;
;    Daten lesen     : RD_DATA
;
;-------------------------------------------------
;    Teste ob Bit 07h (Adr 20h, 32dez, das 7te Bit)
;    gesetzt ist, wenn nicht gehts bei SLASH weiter:
;
          jnb 07h, SLASH
;
;-------------------------------------------------
;    Wenn Bit gesetzt, setze es zurück:
;
          clr 07h
;
;-------------------------------------------------
ASCII1
;    Wenn Akkuinhalt ungleich "1" ASCII ist, springe
;    zu ASCII2:           ASCII 1 = 31h
;
          cjne A, #'1', ASCII2
;
```

```
;    Akku hat eine "1", also schreibe 80h in das
;    Controllregister (erste Zeile, 80h > WR_CTRL):
;
          mov DPTR, #WR_CTRL
          mov A, #80h
          movx @DPTR, A
;
;    springe dann zur Warteschlange WAITPR
;
          sjmp WAITPR
;
;-------------------------------------------------
ASCII2
;    Wenn Akkuinhalt ungleich "2" ASCII ist, springe
;    zu ASCIIS:           ASCII 2 = 32h
;
          cjne A, #'2', ASCIIS
;
;    Akku hat eine "2", also schreibe 0C0h in das
;    Controllregister (zweite Zeile, 0C0h > WR_CTRL):
;
          mov DPTR, #WR_CTRL
          mov A, #0C0h
          movx @DPTR, A
;
;    springe dann zur Warteschlange WAITPR
;
          sjmp WAITPR
;
;-------------------------------------------------
ASCIIS
;    Wenn Akkuinhalt ungleich "\" ASCII ist, springe
;    zu ASCIIN:          ASCII \ = 5Ch
;
          cjne A, #5Ch, ASCIIN
;
;    Akku hat ein Slash. Entsprechend der C-Konvention
;    geben wir nun auch dieses Zeichen aus. Code 0A4h
;    kommt dem \ am Nächsten:
;
          mov A, #0A4h
;
;    und zur Ausgabe:
;
          sjmp OUTD
;
;-------------------------------------------------
ASCIIN
;    Wenn Akkuinhalt ungleich "n" ASCII ist, springe
;    zur Erweiterung oder zur Ausgabe:
;    ASCII n = 6Eh
;
          cjne A, #'n', OUTD
;
;    Akku hat n. Setze Cursor in die nächste Zeile,
;    Routine NEXTL:
;
          sjmp NEXTL
;
```

```
;    hier evtl. Erweiterung für weitere Steuerzeichen..
;
;    Das war der Programmteil für die Steuersequenzen.
;-------------------------------------------------
SLASH
;    Teste ob Akku den Backslash enthält und springe,
;    wenn nicht, zur Abfrage CR:   "\" = 5Ch
;
          cjne a, #5Ch, CR
;
;    Da der Backslash kam, setze Bit 07h:
;
          setb 07h
;
;    Return ohne Warteschlange:
;
          ret
;
;-------------------------------------------------
CR
;    Teste ob Akku "CR" enthält, wenn nicht springe
;    zur Abfrage LF:               "CR" = 0Dh
;
          cjne A, #0Dh, LF
;
;-------------------------------------------------
OLDL
;    Nun soll der Cursor an den Anfang der Zeile. Lese
;    Adresse:
;
          mov DPTR, #RD_CTRL
          movx A, @DPTR
;
;    Ist das 6te Bit gesetzt, so war der Cursor in der
;    zweiten Zeile, ist es Null, war er in der Ersten.
;    Setze das 7te Bit:
;
          setb ACC.7
;
;    und das 5te bis 0te Bit auf Null:
;
          anl A, #11000000b
;
;    dies ist die neue Adresse:
;
          mov DPTR, #WR_CTRL
          movx @DPTR, A
;
;    Springe zur Warteschlange
;
          sjmp WAITPR
;
;-------------------------------------------------
LF
;    Teste ob Akku "LF" enthält, wenn nicht springe
;    zur Ausgabe OUTD:             "LF" = 0Ah
;
          cjne A, #0Ah, OUTD
;
;-------------------------------------------------
```

```
NEXTL
;    Der Cursor soll an den Anfang der nächsten Zeile.
;    Lese aktuelle Adresse:
;
          mov DPTR, #RD_CTRL
          movx A, @DPTR
;
;    Ist das 6te Bit gesetzt, so war der Cursor in der
;    zweiten Zeile, ist es Null, war er in der Ersten.
;    Setze das 7te Bit:
;
          setb ACC.7
;
;    invertiere das 6te Bit:
;
          cpl ACC.6
;
;    und das 5te bis 0te Bit auf Null:
;
          anl A, #11000000b
;
;    dies ist die neue Adresse:
;
          mov DPTR, #WR_CTRL
          movx @DPTR, A
;
;    Springe zur Warteschlange
;
          sjmp WAITPR
;
;-------------------------------------------------
OUTD
;    Ausgabe auf dem LCD-Display,
;    aus dem Basic frei übersetzt:
;
          mov DPTR, #WR_DATA
          movx @DPTR, A
;
;-------------------------------------------------
WAITPR
;    Die Warteschlange für das LCD-Display
;    benutzt DPTR und A:
;
          mov DPTR, #RD_CTRL
          movx A, @DPTR
;
;    nun enthält der Akku in seinem 7ten Bit das
;    Busy-Flag des Displays. Springe wenn Eins:
;
          jb ACC.7, WAITPR
;
;    geht weiter wenn Flag gleich Null
;
          ret
```

Steckverbindungen DTI V1.4

Controlleranschluß links ST1

Pin	1	l	+5V
Pin	3	l	+5V
Pin	24	l	/INT0
Pin	26	l	/INT1
Pin	32	l	/WR
Pin	34	l	/RD
Pin	36	l	GND
Pin	38	l	GND
Pin	39	l	GND
Pin	40	l	GND

Interrupt-eingänge

Controlleranschluß rechts ST2

Pin	1	r	+5V
Pin	2	r	AD0
Pin	3	r	AD1
Pin	4	r	AD2
Pin	5	r	AD3
Pin	6	r	AD4
Pin	7	r	AD5
Pin	8	r	AD6
Pin	9	r	AD7
Pin	10	r	Latch
Pin	13	r	A15
Pin	14	r	A14
Pin	15	r	A13
Pin	16	r	A12
Pin	17	r	A11
Pin	18	r	A10
Pin	19	r	A9
Pin	20	r	A8

Daten- und Adressen gemultiplext

Adressen

Displayanschluß (Stiftleiste) ST3

Pin 1	GND	
Pin 2	+5V	
Pin 3	Vo	Hilfsspannung
Pin 4	RS	0= Befehle, 1= Daten
Pin 5	R//W	0= schreiben, 1= lesen
Pin 6	E	Display-Freigabe
Pin 7	D0	
Pin 8	D1	
Pin 9	D2	Datenleitungen
Pin 10	D3	
Pin 11	D4	
Pin 12	D5	
Pin 13	D6	
Pin 14	D7	

Tastaturanschluß (Stiftleiste) ST4

Pin 3	Y4	
Pin 4	X4	X = Spaltenleitungen
Pin 5	Y3	
Pin 6	X3	Y = Reihenleitungen
Pin 7	Y2	
Pin 8	X2	
Pin 9	Y1	
Pin 10	X1	

resultierende Tastaturcodes

Verbindung	D3	D2	D1	D0
X1-Y1	0	0	0	0
X2-Y1	0	0	0	1
X3-Y1	0	0	1	0
X4-Y1	0	0	1	1
X1-Y2	0	1	0	0
X2-Y2	0	1	0	1
X3-Y2	0	1	1	0
X4-Y2	0	1	1	1
X1-Y3	1	0	0	0
X2-Y3	1	0	0	1
X3-Y3	1	0	1	0
X4-Y3	1	0	1	1
X1-Y4	1	1	0	0
X2-Y4	1	1	0	1
X3-Y4	1	1	1	0
X4-Y4	1	1	1	1

Bauteileliste für DTI V1.4

1*		Platine DTI V1.4	
1*	ST1	Präzisionssockel, doppelreihig 2,54mm, 40polig	
		Buchsenleiste, doppelreihig 2,54mm, 8mm hoch, 40polig	
1*	ST2	Präzisionssockel, einreihig 2,54mm, 20polig	
		Buchsenleiste, einreihig 2,54mm, 8mm hoch, 20polig	
1*	ST3	Stiftleiste, gerade 2,54mm, doppelreihig, 14polig	
		Wanne oder Wanne mit Auswerfern	
1*	ST4	Stiftleiste, gerade 2,54mm, doppelreihig, 10polig	
		Wanne oder Wanne mit Auswerfern	
1*		Stiftleiste, gerade, doppelreihig, 4polig	
1*	J1 oder J2	Jumper	
10*		Lötbrücke entsprechend Layout	
1*	TR1	Trimmpoti stehend 10k PIHER PT 10 Lh	
1*	C2	Keramikkondensator 4n7...100n RM2,54/5,08	
1*	C1	Keramikkondensator 47n ... 1u RM2,54	C1 = 10 * C2
4*	C3,C4,C5,C6	Keramikkondensator u1 RM2,54	
2*	C7,C8	Elko 10u 16V RM2,54	
1*	V1	V-MOS-FET BS 170	
1*	IC1	74LS30	Es können auch HCT-Typen
1*	IC2	74LS573	verwendet werden!
1*	IC3	GAL 16V8 -25	programmiert DTI V1.4
1*	IC4	ICL 7660	
1*	IC5	MM 74C922	National Semiconductor
1*		Präz.sockel 8pol.	
1*		Präz.sockel 14pol.	
1*		Präz.sockel 18pol.	
2*		Präz.sockel 20pol.	

```
·-  GAL-Dokumentation   DTI V1.4 ------------------·
|                                                  |
·--------------------------------------------------·


Jedec-Datei für DTI V1.4
Dateiname:     DTI_V14.LCI
Datum:         3.1.91
Kommentare in Hochkommata

%ID
               DTI_V1.4

%TYP
               GAL16V8

%PINS
     'Bezeichnung der Pins in der Reihenfolge 1...20    '
     'INTEN hier ohne Invertierung dargestellt          '
     'GE heißt GalEnable und gibt den Ausgang INTEN frei'

     CLKI A0   A1   A2   !Y   !RD  !WR  A6   A7      'GND'
     GE   E    NC   NC   NC   NC   CLKO !OE  INTEN   '+'

%IN
     A0

%OUT
     INTEN

%LOGIC
     ' in positiver Logikschreibe                              '

     'E ist Freigabe für Display:                               '
     'Adresse FF00h für Schreibzugriffe Befehlsreg.(/WR = 0)'
     '    und FF01h für Lesezugriffe    Befehlsreg.(/RD = 0)'
     '    und FF02h für Schreibzugriffe Datenreg.  (/WR = 0)'
     '    und FF03h für Lesezugriffe    Datenreg.  (/RD = 0)'

     E =   Y * !A7 * !A6 * !A2 * !A0 * WR
         + Y * !A7 * !A6 * !A2 *  A0 * RD;

     '!OE ist die Ausgangsfreigabe für den Tastaturchip:    '
     'Adresse FF40h und /RD = 0 (read only)                 '

     OE = Y * !A7 * A6 * !A2 * !A1 * !A0 * RD;

     'CLKO ist der Takt für das D-FF im GAL und low solange '
     ' /WR = 0 ist und die Adresse FF42h oder FF43h anliegt '
     'FF42h gibt INTEN frei, ff43h sperrt INTEN, siehe unten'

     CLKO = Y * !A7 * A6 * !A2 * A1 * WR;

%FUNCTION
     ' A0 -> INTEN, beachte pos. Logik!                      '

     0 -> 0
     1 -> 1

%END
```