8052-ECB V1.1

BasiControl

Stand: 20.3.92

Dies Handbuch soll die Anwendung der Controllerkarte 8052-ECB unterstützen. Die vorliegende Beschreibung entstand, wie wohl jedes Handbuch, in großer Eile. 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

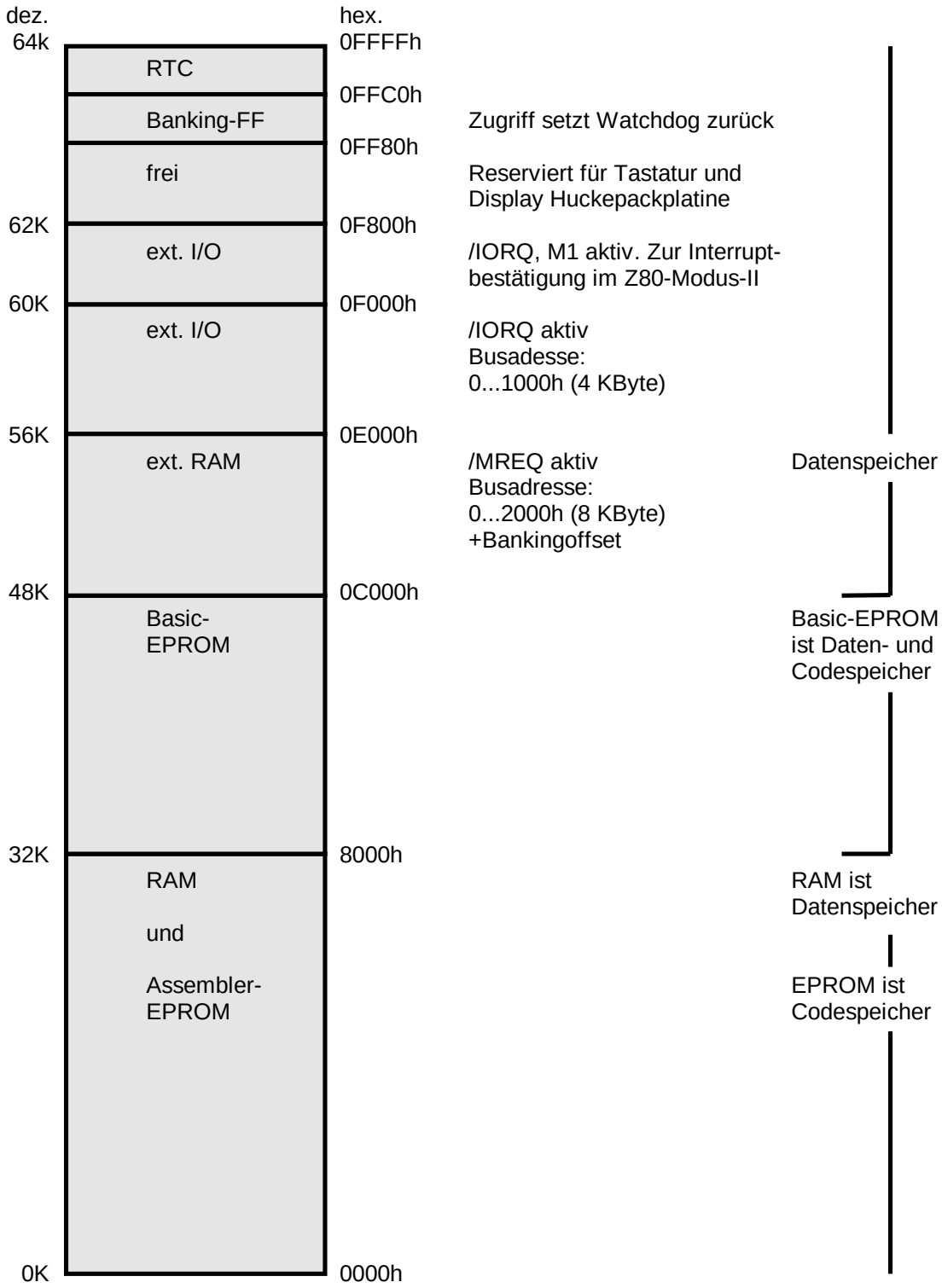
Funktionsbeschreibung 8052-ECB V1.1

8051-Speicherverwaltung:
Die Intel-Prozessoren der 8051-Familie können generell drei unabhängige Adreßbereiche verwalten: Einen Code- oder Programmspeicher, einen internen und einen externen Datenspeicher. Der Codespeicher kann 64 kByte umfassen, wovon beim 8052 die unteren 8 kByte auf dem Chip integriert sind (Basicinterpreter). Der Codespeicher kann nur gelesen werden und enthält Assemblerprogramme. Die Leitung /EA des Prozessors erlaubt den internen Codespeicher abzuschalten und die gesammten 64 kByte extern anzuschließen. Der externe Datenspeicher darf ebenso 64 kByte lang sein, der interne Datenspeicher hat beim 8052/32-Prozessor eine feste Länge von 256 Byte (128 Byte im 8051/31). Im externen Datenspeicher werden Basicprogramme und Daten abgelegt. Diese Prozessoren adressieren ihren Speicher also nicht linear, wie sonst üblich, sondern teilweise parallel. Die Bereiche werden mit unterschiedlichen Befehlen angesprochen.

Adreßbereiche:
Auf der Controllerkarte 8052-ECB sind 32 kByte RAM als externer Datenspeicher und ein Sockel für max. 32 kByte EPROM, externer Codespeicher vorhanden. Beide Speicher liegen im Adreßbereich 0000h bis 7FFFh.
Der folgende Bereich 8000h bis BFFFh (16 kByte) beherbergt ein EPROM für Basicprogramme. Dieser Speicher ist auf der Karte programmierbar.
Um die Möglichkeiten eines ECB-Systems auszuschöpfen, sollte auch externer Speicher auf anderen Steckkarten ansprechbar sein. Im Bereich C000h bis DFFFh (8 kByte) ist das Bussignal /MREQ aktiv. Durch Banking adressiert der Rechner 64 kByte externen Speicher in acht Blöcken. Hier können sowohl Basicprogramme als auch Daten abgelegt werden.
Der ECB-Bus läßt eine Unterscheidung zwischen Ein-, Ausgabe- und Speicherbaugruppen zu. Zum Ansprechen von I/O-Karten dient das Bussignal /IORQ. Es wird von der Controllerkarte im Adreßbereich E000h bis F7FFh (6 kByte) generiert.
Um eine Interruptroutine entsprechen dem Z80-Modus-II durchzuführen, wird für Adressen zwischen F000h und F7FFh (2 kByte) gleichzeitig mit /IORQ auch das Signal /M1 logisch Null. Damit kann der Controller die Herkunft einer Unterbrechungsanforderung feststellen und entsprechende Programme ausführen (vektorisierter Interrupt mit mehreren Quellen).
Der Adreßbereich F800h bis FF7Fh ist für spätere Erweiterungen auf einer Huckepackplatine reserviert. Ab Adresse FF80h wird das Banking-Flipflop gesetzt. Nach einer Schreiboperation liegen an den Adreßleitungen A15, A14 und A13 des ECB-Systems die Datenbits D7, D6 und D5 an. Damit selektiert der Controller die oben beschriebenen acht Speicherblöcke. Gleichzeitig setzt jeder (Schreib- oder Lese-) Zugriff auf diese Adresse den Watchdog zurück.
Die Echtzeituhr ist im Bereich FFC0h bis FFFFh aktiv. Die einzelnen Register der RTC sind in einem eigenen Kapitel beschrieben.

Adreßdekoder-GAL:
Zur Dekodierung der Adressen der diversen Bausteine und der Signale /MREQ, /IORQ und /M1 wurde ein GAL-Baustein 16V8 verwendet. Dieses IC erhält als Eingangsvariablen die Adreßleitungen A6 bis A15. Der Baustein ist programmierbar und kann vom Anwender den Bedürfnissen angepaßt werden.

Bitmap 8052-ECB V1.1



Bussignale:
Zum Betrieb benötigt die Rechnerkarte eine Betriebsspannung von 5 V bei etwa 200 mA Stromaufnahme. Ein Akku für RAM und Echtzeituhr ist auf der Karte vorhanden. Er kann auch externe Baugruppen über den Busanschluß 24a (5V Batt.) puffern. Ist diese Belegung unerwünscht, so wird der Jumper J10 gezogen.
Um das Basic-EPROM auf der Platine zu programmieren ist, je nach EPROM-Typ, eine Spannung von 12,5 V oder 21 V notwendig. Sie kann ebenfalls über den ECB-Bus zugeführt werden. Dazu wurde der Anschluß 13a (+12V/Vpp) ausgewählt. Auch dieser Busanschluß kann, mit dem Jumper J7, abgeschaltet werden.
Der Resetgenerator läßt sich über den Busanschluß 31c (RESET) mit einem Low-Signal triggern. Zum Initialisieren anderer Steckkarten dient der Ausgang 26c (/PCL, power on clear).
Die Daisy-chain Anschlüsse IEI und IEO sind durchgeschleift.
Ein DMA-Betrieb ist nicht implementiert.
Die RTC kann ein programmierbares periodisches Interruptsignal liefern. Dies steht via Jumper J19 am ECB-Bus bereit. Der Ausgang ist an Anschluß 26a (/Ci, calling indicator) geführt, welcher sonst nicht belegt ist. Dieser Interrupt wird auch ohne Betriebsspannung generiert und kann z.B. ein geeignetes Netzteil einschalten (Weckfunktion).
Die ECB-Interrupteingänge /NMI (Pin 20c) und /INT (Pin 21c) können mit den Interruptports des Prozessors, über die Jumper J1 bis J6, verbunden werden. Die Möglichkeiten sind weiter unten aufgelistet.
Ebenso kann auch der /WAIT-Eingang (Pin 10a) die Controllerkarte steuern. (Die 8051-Prozessoren haben keine Waitfunktion implementiert.)
Wartet der Prozessor auf eine Unterbrechung (Idle-Modus), so geht der Busanschluß /HALT (Pin 25c) auf Low.
Die Adreßleitungen sind ständig aktiv, die Datenleitungen dagegen nur während eines Buszugriffs. Datenleitungen enthalten gemultiplext auch die niederwertigen Adressen. Für einige Anwendungen kann dies vorteilhaft sein, in Verbindung mit normalen ECB-Baugruppen stört es nicht. Zum Demultiplexen ist das ALE-Signal des Prozessors auf dem Anschluß 29c (CLK) verfügbar.

Terminalport:
An der Frontseite der Europakarte befindet sich der serielle Terminalanschluß (ST2). Hier wurde ein 9poliger D-Sub Stecker verwendet, wie auch bei PCs üblich. Die Pinbelegung und die Pegel entsprechen der RS232-C. Der Anschluß ist als DTE (data terminal equipment) oder dtsch. Daten-Endeinrichtung ausgeführt, so daß ein handelsübliches Nullmodemkabel verwendet werden kann. Als Pegelkonverter wurde ein MAX 232 verwendet. Dieses IC enthält einen integrierten Spannungswandler, so daß keine zusätzlichen Betriebsspannungen notwendig sind. Handshakeleitungen sind nicht vorhanden, jedoch kann man mit dem Jumper J15 die Betriebsspannung auf den Anschluß DTR (Pin 4) legen um evtl. angeschlossene Schaltungen oder das Terminal mit Spannung zu versorgen. Die Grundeinstellung der Schnittstelle ist asynchron, 1 Startbit, 8 Datenbits, 1 Stopbit, kein Pariätsbit, XON/XOFF-Handshake. Der Basicinterpreter startet nach dem Einschalten zunächst eine automatische Baudratenerkennung. Dazu ist es nötig am Terminal die Space-Taste zu betätigen, die Übertragungsgeschwindigkeit wird ausgewertet und der Rechner gibt eine Meldung zurück:

```
*MCS-51(tm) BASIC V1.1*
READY
>
```

Prinzipiell sind Baudraten bis zu 19200 Bd möglich jedoch hat sich eine Übertragung mit 9600 Bd als sicherer erwiesen. Eine Unterbrechung laufender Programme erreicht man mit Control-C.

Druckerport:
Weiterhin wurde ein Druckerport (ST3) vorgesehen. Der 8052-Prozessor kennt spezielle Basicbefehle mit denen eine Ausgabe über diesen seriellen Anschluß möglich ist. So können z.B. ständig Meßwerte protokolliert oder Programmlistings ausgegeben werden. Der Druckerport ist unidirektional konzipiert und nutzt kein Handshake. Es kann aber, unterstützt von einem Maschinenspracheprogramm, die Leitung RXD oder CTS abgefragt werden. Dazu ist mit den Jumpern J16, J17, J14 eine Verbindung zum Port T2 (P1.0) des Prozessors herzustellen. Wie schon beim Terminal beschrieben kann auch hier der DTR-Anschluß mit Betriebsspannung verbunden werden (J18). Die Baudrate des Druckerports kann per Software eingestellt werden.

Schallwandler:
Zum Anschluß eines piezokeramischen Schallwandlers (SBD) wurde der PWM-Ausgang des Prozessors gepuffert ausgeführt. Das Signal steht am Anschluß ST6 auf der Platine zur Verfügung. Mit einfachen Programmanweisungen ist es möglich Warn- und Signaltöne auszugeben.

Watchdog:
In der Entwicklungsphase eines Programms ist es sinnvoll den Ablauf mittels eines Watchdogs zu überwachen. Hat sich das Programm in einer Endlosschleife verfangen, so wird von der Watchdog-Hardware nach spätestens 10 Sekunden ein Reset ausgelöst. Das Programm startet dann von vorn. Es wäre möglich die auftretenden Fehler mit einem Signalton zu melden oder deren Anzahl zu zählen. Fehlerhafte Interruptroutinen und Ähnliches können so nur begrenzten Schaden anrichten und der Programmierer wird rechtzeitig aufmerksam. Die Zuverlässigkeit der Watchdogfunktion hängt weitgehend von der Struktur des Programms ab. Der relativ lange Zeitraum von 10 Sekunden wurde gewählt um auch eine Nutzung unter Basicprogrammen zu ermöglichen. Die Watchdogzeit kann durch Austausch eines Kondensators in weiten Grenzen variiert werden.
Zur Realisierung wurde ein freilaufender Oszillator mit integriertem Zähler verwendet. Nach 214 Impulsen des Oszillators, entsprechend 10 Sekunden, resultiert ein Resetsignal. Findet vorher ein Schreib- oder Lesezugriff auf das Bankingflipflop statt (Adresse 0FF80h...0FFBFh), erhält der Zähler ein Rücksetzsignal und die Wartezeit beginnt von vorn. Wird die Watchdogfunktion nich benötigt, so kann über einen Jumper (J13) dem Zähler regelmäßig ein Rücksetzsignal zugeführt werden.

Resetgenerator:
Als Resetgenerator wurde ein integrierter Baustein TL7705 verwendet. Nach Anlegen der Betriebsspannung wird der Prozessor und angeschlossene Peripherie in einen definierten Startzustand versetzt. Ein Resetsignal wird durch Absinken der Betriebsspannung, durch den Watchdog, durch Betätigen des Tasters an der Frontseite des Rechners oder durch Anlegen eines Lowsignals am RESET-Eingang des ECB-Bus erzeugt. Ferner hat die Controllerkarte einen Initialisierungsausgang für angeschlossene Buskarten. Der Ausgang /PCL führt während eines Resets etwa 10ms lang Lowpegel. Für diesen Zeitraum wird auch das batteriegepufferte RAM und die Programmierlogik für das Basic-EPROM deselektiert, so daß ein Überschreiben der gespeicherten Informationen durch undefinierte Pegel unterbleibt.

Echtzeituhr:
Für viele Aufgaben der Meßtechnik ist es unabdingbar Uhrzeit und Datum sofort nach dem Einschalten verfügbar zu haben. Diese Daten können dann zusammen mit Meßwerten abgespeichert werden und später die Auswertung erleichtern. Zwar besitzt der 8052-Prozessor eine interne Uhrenfunktion, jedoch muß diese jedesmal nach dem Einschalten gestellt werden. Aus diesen Gründen wurde auf der Controllerkarte ein integrierter Uhrenbaustein RTC72421 vorgesehen. Dieses IC enthält im Gehäuse einen Schwingquarz und ermöglicht so einen äußerst platzsparenden Aufbau. Zudem ist die Uhr direkt buskompatibel und vereinfacht damit die Programmierung (kein zusätzlicher Portbaustein notwendig). Der Uhrenchip wird ebenso wie das RAM batteriegepuffert, was aufgrund des äußerst geringen Ruhestroms von etwa 1,5 uA einen Datenerhalt über etwa 800 Tage erlaubt. Der Fehler durch Temperaturschwankungen liegt bei max. 1...2 Sekunden pro Tag. Alle gewöhnlichen Zeit- und Datumsfunktionen sind vorhanden. Zusätzlich ist ein programmierbarer Interruptausgang vorgesehen. Dieser Anschluß kann regelmäßig einen Lowimpuls generieren und damit z.B. ein Netzteil und damit den Rechner einschalten. Der Interruptausgang kann mit dem ECB-Bus verbunden werden und ist hier mit /Ci calling indicator bezeichnet.

Programmiereinrichtung:
Auf der vorliegenden Controllerkarte ist eine EPROM-Programmiereinrichtung vorgesehen. Damit können Basic-Programme, die zuvor im RAM-Bereich getestet wurden, dauerhaft abgelegt werden. Die Programme können dann z.B. automatisch, nach Anlegen der Betriebsspannung, ausgeführt werden. Ferner kann man im EPROM die Baudrate und den Wert für MTOP ablegen. Damit die Programmierung onboard stattfinden kann, ist es notwendig die Programmierspannung und Programmierimpulse am EPROM anzuschalten. Ferner muß das Adreßlatch abgeschaltet werden (Dies ist eine Eigenart des Prozessors, nähere Informationen siehe Intel-Hand-buch). Leider belegt die Programmiereinrichtung drei Prozessorports, die sonst für andere Anwendungen auf einer Huckpackplatine zur Verfügung stünden. Um dem Benutzer die Funktionen wahlweise zu ermöglichen ist die Programmiereinrichtung abschaltbar (entfernen von Jumper J12), die Ports sind dann frei verfügbar. Ferner wurde in die Logik das Reset-Signal mit eingebunden. So ist sichergestellt, daß beim langsamen Absinken der Betriebsspannung kein versehentliches Überschreiben des Speichers durch undefinierte Pegel möglich ist.

Huckepackerweiterung:
Für verschiedene Anwendungen ist ein Display und ein Tastenfeld am Rechner unerläßlich. Damit für diese Aufgabe keinen Bussteckplatz verwendet werden muß, kann eine Erweiterungskarte direkt auf den Rechner gesteckt werden. An den Pfostenverbindern, links (ST5) und rechts (ST4) neben der CPU, stehen alle Prozessor-Signale zur Verfügung. Zusätzlich wurden hier auch die /RESET-Leitung und das Latch-Signal bereitgestellt.

ECB-BELEGUNG (ST1) 8052-ECB V1.1

Pin nr.:		Fkt:	I/O:	Anmerkung:
1a		+5V	I	I_c=350mA f. NMOS
2a		D5	I/O	
3a		D6	I/O	Multiplexsignal:
4a		D3	I/O	Daten und Adressen
5a		D4	I/O	
6a		A2	O	
7a		A4	O	
8a		A5	O	
9a		A6	O	
10a		/WAIT	I 10k	Interrupteingang
11a		/BUSRQ	-	nicht implementiert
12a		-		
13a	!	+12V/Vpp	I	Programmierspg. 12V/21V; jump.
14a		-		
15a		-5V	-	reserviert
16a		-		
17a		-		
18a		A14	O	banking
19a	!	+15V	-	reserviert
20a		/M1	O	Adreßbereich
21a		-		
22a		-		
23a		-		
24a		5V Batt.	O	3,6V Akku für RAM,RTC; jump.
25a		-		
26a	!	/Ci	O	RTC-Impuls open Drain; jump.
27a		/IORQ	O	Adreßbereich
28a		-		
29a		A13	O	banking
30a		A9	O	
31a		/BUSAK	-	nicht implementiert
32a		GND		

Pin nr.:		Fkt:	I/O:	Anmerkung:
1c		+5V	I	
2c		D0	I/O	
3c		D7	I/O	
4c		D2	I/O	
5c		A0	O	
6c		A3	O	
7c		A1	O	
8c		A8	O	
9c		A7	O	
10c		-		
11c		IEI		verbunden mit IEO
12c		-		
13c		-		
14c		D1	I/O	Der war betrunken!?
15c	!	-15V	-	reserviert
16c		IEO		verbunden mit IEI
17c		A11	O	
18c		A10	O	
19c		-		
20c		/NMI	I 10k	Interrupteingang
21c		/INT	I 10k	Interrupteingang
22c		/WR	O	
23c		-		
24c		/RD	O	
25c		/HALT	O	IDLE
26c		/PCL	O	/RESET
27c		A12	O	
28c		A15	O	banking
29c		CLK	O	ALE; 1,8MHz
30c		/MREQ	O	Adreßbereich
31c		/RESET	I 10k	/RESIN
32c		GND		

!	Belegung entspricht nicht dem Standardbus
I/O	Ein-, Ausgänge der Prozessorkarte
-	Wird von der Prozessorkarte nicht benutzt
jump.	Pin ist mittels Jumper freischaltbar
10k	Pullup-Widerstand auf der Prozessorkarte

Steckverbindungen 8052-ECB V1.1

Terminalport (DTE) 9pol. D-Sub Stecker ST2

Pin 2	RXD	Empfangsdaten
Pin 3	TXD	Sendedaten
Pin 4	DTR	via Jumper an +5V
Pin 5	GND	Masse

Druckerport (DTE) 9pol. D-Sub Stecker ST3

Pin 2	RXD	via Jumper an Prozessoreingang T2
Pin 3	TXD	Sendedaten
Pin 4	DTR	via Jumper an +5V
Pin 5	GND	Masse
Pin 8	CTS	via Jumper an Prozessoreingang T2

PWM-Ausgang gepuffert für Piezoschwinger ST6

Pfostenverbinder ST4
rechts vom Prozessor

Pin:	Fkt:
1	+5V
2	AD0
3	AD1
4	AD2
5	AD3
6	AD4
7	AD5
8	AD6
9	AD7
10	Latch (ALE & /DIS)
11	ALE
12	/PSEN
13	A15
14	A14
15	A13
16	A12
17	A11
18	A10
19	A9
20	A8

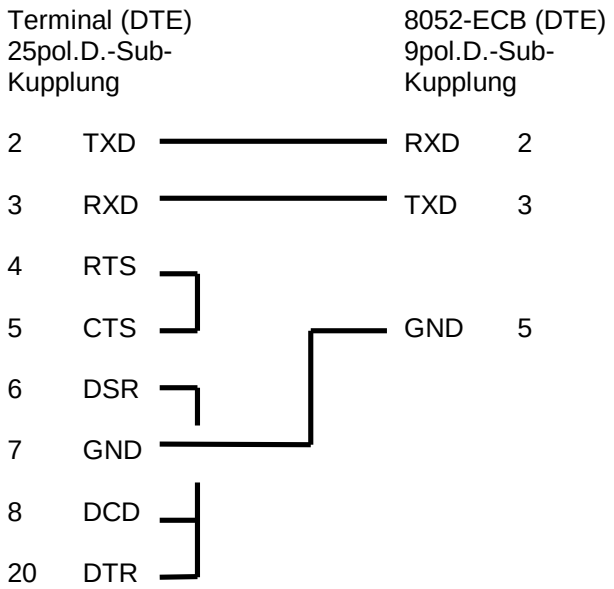
Pfostenverbinder ST5
links vom Prozessor

Pin:	Fkt:	Pin:	Fkt:
1	+5V	2	T2
3	+5V	4	T2EX
5	+5V	6	PWM
7	nc	8	/ALED
9	nc	10	/PRP
11	nc	12	/PREN
13	nc	14	/DMACK
15	nc	16	LPOUT
17	nc	18	RESET
19	nc	20	/RXD
21	nc	22	/TXD
23	nc	24	/INT0
25	nc	26	/INT1
27	nc	28	T0
29	nc	30	T1
31	nc	32	/WR
33	nc	34	/RD
35	RESET	36	GND
37	/RESET	38	GND
39	GND	40	GND

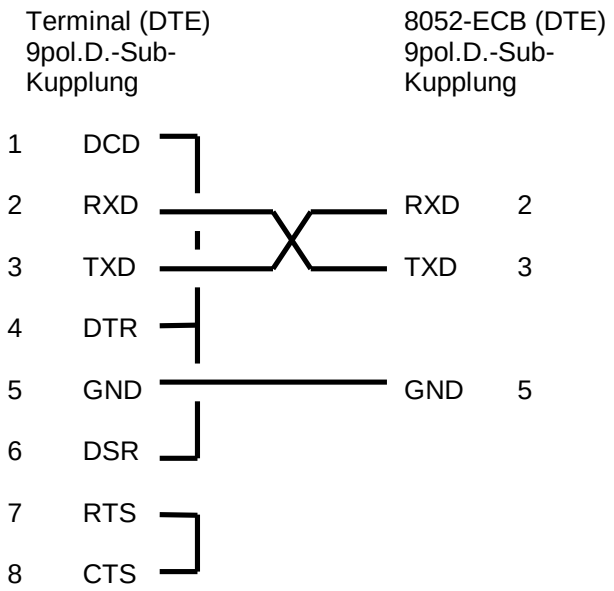
Schnittstellenkabel

Verbindungskabel für die Komunikation zwischen 8052-ECB und Terminal oder 8052-ECB und seriellem Drucker:

25poliger RS232-Anschluß am Terminal:



9poliger RS232-Anschluß am Terminal:



Jumperfunktionen 8052-ECB V1.1

J1...J6	ECB-Interrupteingänge an Interrupteingänge-8052 (setzen mehrerer Jumper bewirkt wired-and)	
J1	/WAIT Pin 10a an /INT1	
J2	/NMI Pin 20c an /INT1	/INT1 ist in Basic oder
J3	/INT Pin 21c an /INT1	Assembler bedienbar
J4	/WAIT Pin 10a an /INT0	
J5	/NMI Pin 20c an /INT0	/INT0 ist nur in Assembler
J6	/INT Pin 21c an /INT0	nutzbar
J7	Programmierspannung (12 V oder 21 V) aus +12V/Vpp Pin 13a ECB	
J8	Batterie/Akku puffert RAM und RTC	
J9	Akku wird im Betrieb des Rechners geladen (Q/100)	
J10	Batterie/Akku-Anschluß von RAM und RTC mit 5V Batt. Pin 24a ECB verbunden	
J11	/EA, internes ROM (Basicinterpreter) des Prozessors abgeschaltet	
J12	Programmierung des Basic-EPROMS freigegeben	
J13	Watchdog abgeschaltet	
J14	Empfangsleitung des Druckerports an T2-Eingang des Prozessors	
J15	DTR (Pin4 ST2) Terminalport an +5V	
J16	RXD (Pin2 ST3) Druckerport ist Empfangsleitung	
oder:		
J17	CTS (Pin8 ST3) Druckerport ist Empfangsleitung	
J18	DTR (Pin4 ST3) Druckerport an +5V	
J19	/Ci an Pin 26a ECB	

GAL-Dokumentation 8052-ECB V1.1

Jedec-Datei für 8052-ECB V1.1
Dateiname: 8052_V11.LCI
Datum: 15.12.90
Kommentare in Hochkommata

```
%ID
        32k_V1.1

%TYP
        GAL16V8

%PINS
        'Bezeichnung der Pins in der Reihenfolge 1...20'

        A6      A15     A14     A13     A12     A11     A10     A9      A8      'GND'
        A7      !IORQ   !MREQ   !M1     !CSRAM !CSRTC  !CSP    !CSBAS  !CSASS  '+'

%LOGIC
        'Darstellung in positiver Logic!'

'RAM            Adr.: 0000h...7FFFh'
        CSRAM   = !A15;

'Assembler-ROM  Adr.: 0000h...7FFFh'
        CSASS   = !A15;

'Basic-EPROM    Adr.: 8000h...BFFFh'
        CSBAS   = A15 * !A14;

'extern RAM     Adr.: C000h...DFFFh'
        MREQ    = A15 * A14 * !A13;

'extern I/O     Adr.: E000h...EFFFh exklusiv    '
'               und  F000h...F7FFh mit M1-Signal'
        IORQ    = A15 * A14 * A13 * !A12
                + A15 * A14 * A13 * A12 * !A11;

'M1-Signal      Adr.: F000h...F7FFh nur mit IORQ'
        M1      = A15 * A14 * A13 * A12 * !A11;

'Adresslatch    Adr.: FF80h...FFBFh'
        CSP     = A15 * A14 * A13 * A12 * A11 * A10 * A9 * A8 * A7 * !A6;

'RTC            Adr.: FFC0h...FFFFh'
        CSRTC   = A15 * A14 * A13 * A12 * A11 * A10 * A9 * A8 * A7 * A6;

' Adressen F800h bis FF7Fh sind frei für Erweiterungen!'

%END
```

Bauteileliste 8052-ECB V1.1

1*		Platine 8052-ECB V1.1
1*	ST1	Messerleiste DIN 41.612 Bauform C, 64polig, gewinkelt, a+c bestückt (DIN BEZ.: C64M-C1A)
2*	ST2,3	D-Sub.-Stiftleiste, 9 polig (DB9), gewinkelte Lötstifte
2*		Befestigungswinkel zu D-Sub., 9 pol. (Kunstofformteil)
4*		Adapter-Gewindebolzen UNC 4-40/M3-05/08 oder D-Sub-Stiftleiste komplett
1*		Kunststoffwinkel für Frontplatte
6*		Zylinderkopfschraube M3*8
10*		Mutter sechskant M3
4*		Zahnscheibe M3
2*		Zylinderkopfschraube M2,5*10
2*		Zylinderkopfschraube M2,5*8
4*		Mutter sechskant M2,5
1*	S1	Resettaster Mentor 1.840 Miniaturtaster f.Leiterpl. mit Knopf (Bürklin 12 G 2781 und 12 G 2785)
1*		Stiftleiste gerade 2,54mm einreihig 42 polig
1*		Stiftleiste gerade 2,54mm zweireihig 58 polig
13*		Jumper
1*	RN1	Widerstandsarray SIL 9* 10k
1*	RN2	Widerstandsarray SIL 8* 10k
1*	R3	Widerstand Metallfilm 1k3
3*	R4,R7,R8	Widerstand Metallfilm 4k7
6*	R1,R5,R6,R9, R11,R15	Widerstand Metallfilm 10k
1*	R13	Widerstand Metallfilm 27k
1*	R10	Widerstand Metallfilm 47k
4*	R2,R12,R14,R16	Widerstand Metallfilm 100k
2*	C1,C2	Keramikkondensator 33p RM2,54/5,08
1*	C15	Keramikkondensator 1n RM2,54/5,08
1*	C14	Kondensator 10n RM2,54/5,08 (WD-Zeit 1n = 1s)
18*	C3...C8, C10...C13,C17, C23,C25...C30	Keramikkondensator u1 RM2,54 (Z5U o.ä.)
1*	C9	Elko 1u 10V RM2,54 (Reset-Zeit)
5*	C16,C18...C21	Elko 10u 16V RM2,54 Da<5,08mm
2*	C22,C24	Elko 47u/100u 10V RM2,54 Da<7,62mm
5*	D1,D3...D6	Diode 1N4148 o.ä.
1*	D2	Schottky-Diode SD 101 o.ä.
1*	V1	Transistor NPN BC 547B o.ä.
1*	V3	Transistor PNP BC 557B o.ä.
1*	V2	V-MOS-FET BS 170
1*	IC5	TL 7705
1*	IC2	MAX 232
1*	IC6	4060
1*	IC7	4093
1*	IC18	74LS 07
1*	IC9	74HCT08
1*	IC8	74HCT32

4*	IC15,IC16, IC17,IC19	74HCT245
1*	IC3	74HCT373
1*	IC11	74HCT377
1*	IC10	RTC 72421
1*	IC4	GAL 16V8-25 (Q) (programmiert 32k_V1.1)
1*	IC1	P 8052AH-BASIC Version 1.1
1*	IC12	RAM 43256-150
1*	IC13	EPROM 27C128 250ns, 12,5V für Basicprog.
1*	X	Quarz 11,0592 MHz HC-18/U
1*		Präz.sockel 8pol.
4*		Präz.sockel 14pol.
2*		Präz.sockel 16pol.
1*		Präz.sockel 18pol.
7*		Präz.sockel 20pol.
3*		Präz.sockel 28pol.
1*		Präz.sockel 40pol.
1*		Akku Varta 3,6V 60mAh oder Lithiumbatterie Best.nr.:Varta 53306 603 059
optional:		
1*	IC14	EPROM 27C64, 27C128 oder 27C256 für Assemblerprog.
1*	SBD	Keramikschwinger z.B. Signalgeber F/EE 17 P Digisound (Bürklin 36 M 572)
1*		Nullkraftsockel 28pol. z.B. NIF 28

Funktionsbeschreibung RTC-72421

Die RTC-72421 ist eine buskompatible Echtzeituhr für den Einsatz in Mikroprozessorsystemen. Sie beinhaltet einen Quarzoszillator und ist in stromsparen der C-MOS-Technik ausgeführt. Der eingebaute Schwingquarz macht externe Komponenten überflüssig und erlaubt eine einfache und platzsparende Schaltungsauslegung. Alle üblichen Zeit und Datumsfunktionen incl. Schaltjahr und 12/24 Stunden Format sind implementiert.

Leistungsdaten:
- ALE-Eingang für die gemultiplexten Bussysteme der 8048/51/85-Prozessoren
- Separate 4-Bit Adress- und Datenbusse
- C-MOS-Schaltung mit geringer Stromaufnahme
- Batteriepufferung bis minimal UB=2V
- 18-poliges Dual-In-Line-Gehäuse
- Pin- und Funktionskompatibel zu SMC 5242C
- Drei Kontrollregister für Stell- und Interruptfunktionen

Standardimpuls-Ausgang:
Das Uhrenmodul hat einen programmierbaren Interruptausgang, mit dem z.B. eine Stromversorgung für die Rechnerkarte zu bestimmten Zeiten eingeschaltet werden kann oder eine regelmäßige Meßwertaufnahme gesteuert wird. Der Ausgang erzeugt Low-Impulse, die auf dem Bus verfügbar sind. Sie werden auch generiert wenn die Betriebsspannung fehlt und die Uhr batteriegepuffert wird. Über dem Jumper J19 ist der Ausgang mit dem ECB-Bus, Pin 26a, /Ci (Calling indicator) verbunden. Ein 100 kOhm Pullup-Widerstand hält diesen Anschluß auf Batteriespannung. Der maximale Strom bei Lowpegel sollte 2 mA nicht überschreiten. Die Impulse können in Abständen von 1/64s, 1s, 1min oder 1std. erzeugt werden. Der Standardimpuls-Ausgang wird mit Kontrollregister E gesteuert.

Resetsteuerung:
Das Uhrenmodul hat zwei Chip-select-Eingänge. Der /CS0-Eingang aktiviert den Baustein ab Adresse FFC0h. Eingang CS1 ist mit der Reset-Schaltung des Rechners verbunden und ermöglicht eine saubere Batteriepufferung.

Vorsichtsmaßnahmen:
Die RTC sollte nicht über 150°C gelagert werden. Der Baustein ist nur von Hand oder im Schwallbad einzulöten (max. 10s bei 260°C). Durch den integrierten Schwingquarz ist die Uhr empfindlich gegen Stoß und Schock. Reinigen Sie die Platine bitte nicht im Ultraschallbad wenn das IC montiert ist. Die üblichen Vorsichtsmaßnahmen im Umgang mit C-MOS-Schaltkreisen sind zu beachten. Starke Magnetfelder in der direkten Umgebung können zu Fehlfunktionen führen.

Genauigkeit:
Im Temperaturbereich von 0°C...50°C zeigt die RTC eine Abweichung von df/f = 0...-20ppm, wobei die Zeitdifferenz sich ergibt zu dt = df/f*60*60*24 in s/tag. Z.B. im Bereich 0...50°C etwa 1.8s/tag im Bereich 10...40°C etwa 0,7s/tag.

Stromaufnahme:
Im Standby-Betrieb nimmt die RTC etwa 1,5uA (typ.) auf. Zusammen mit dem gepufferten RAM wird die Batterie des Rechners mit ca. 3uA belastet. Mit der Batteriekapazität von 60mAh ergibt sich eine theoretische Speicherzeit von 800 Tagen. Die Frequenzabweichung der RTC nimmt mit abnehmender Batteriespannung bis etwa 1ppm bei 2V zu.

Registertabelle:

Adresse	D3	D2	D1	D0	Registername	Wert	Beispiel
0 (FFC0h)	S8	S4	S2	S1	Sekunden Einer	0...9	5
1 (FFC1h)	-	S40	S20	S10	Sekunden Zehner	1...5	10s
2 (FFC2h)	M8	M4	M2	M1	Minuten Einer	1...9	2
3 (FFC3h)	-	M40	M20	M10	Minuten Zehner	1...5	20m
4 (FFC4h)	H8	H4	H2	H1	Stunden Einer	1...9	8
5 (FFC5h)	-	AM/PM	H20	H10	Stunden Zehner	0...2	10h
6 (FFC6h)	D8	D4	D2	D1	Tag Einer	0...9	5
7 (FFC7h)	-	-	D20	D10	Tag Zehner	0...3	10.
8 (FFC8h)	M8	M4	M2	M1	Monat Einer	0...9	2
9 (FFC9h)	-	-	-	M10	Monat Zehner	0,1	10.
A (FFCAh)	Y8	Y4	Y2	Y1	Jahr Einer	0...9	0
B (FFCBh)	Y80	Y40	Y20	Y10	Jahr Zehner	0...9	90
C (FFCCh)	-	W4	W2	W1	Wochentag	0...6	6
D (FFCDh)	30ADJ	IRQ	BUSY	HOLD	Register D		
E (FFCEh)	t1	t0	I/S	MASK	Register E	Samstag 15.12.90	
F (FFCFh)	TEST	24/12	STOP	RES	Register F	18Uhr 22min 15sec	

Anmerkungen:

AM/PM	1 = PM, muß im 24 Stunden Modus maskiert werden
HOLD	write only
BUSY	read only
IRQ	Interrupt request flag, kann nur Null gesetzt werden
I/S	Interrupt/Standard 1 = Interrupt
RES	Reset
12/24	1 = 24
-	wird bei Schreiboperationen ignoriert
Adresse	relativ (absolut in 8052-ECB)
Wochentag	0=Sonntag ... 6=Samstag

Schaltjahr nach Hindu-Kalender
Das Schreiben eines ungültigen Datums ist zu vermeiden

Register D: Adresse D relativ (FFCDh absolut)

1. Hold-bit (D0)
Zum Einstellen oder Lesen der Uhrzeit muß die RTC angehalten werden, so daß sich die Registerinhalte nicht verändern. Die Vorteiler und der Oszillator laufen weiter. Solange das Hold-bit für weniger als eine Sekunde gesetzt bleibt, ergibt sich kein Fehler. Deshalb sollten zum Stellen oder Lesen der Uhr alle Interrupteingänge des Prozessors gesperrt werden. Nach Setzen des Hold-bits auf Eins geht Busy auf Null; Hold kann nicht gelesen werden. Sind die Operationen abgeschlossen, muß Hold wieder auf Null gesetzt werden. Wenn in der Zwischenzeit ein Übertrag aus den Vorteilern der Uhr resultiert, wird das S1-Register nach dem Rücksetzen des Hold-bits aktualisiert. CS1 = Null (Reset oder power down) setzt Hold auf Null.

2. Busy-bit (D1)
Die Zeitregister dürfen nur gestellt werden, wenn Busy gleich Null ist. Busy ist read only und kann nur mit Hold beeinflußt werden. Der Hersteller der RTC empfiehlt, nach dem Setzen des Hold-bits, noch Busy abzufragen.

3. IRQ Interrupt request flag (D2)
Das IRQ wird vom programmierbaren Standardimpuls-Ausgang gesteuert. Solange der Standardimpuls aktiv (Null) ist, ist IRQ logisch Eins. Im Interrupt-Modus des Standardimpulses bleibt der Ausgang Null bis IRQ rückgesetzt wird. Im Standard-Modus bleibt der Ausgang für maximal 7,8 ms Null oder bis IRQ zurückgesetzt wird. Ein Setzen des IRQ-Bits auf Eins hat keinen Einfluß auf den Standardimpuls-Ausgang. Nach dem Verändern von t0 oder t1 (siehe unten) muß IRQ zurückgesetzt werden.

4. 30ADJ (D3)
Durch Schreiben einer Eins in dieses Bit, wird das Sekundenregister Null gesetzt. War der Inhalt des Sekundenregisters vorher größer als 29 wird ein Übertrag ins Minutenregister generiert. Das Rücksetzen des 30ADJ-Bits erfolgt automatisch nach 76,3 us.

Register E: Adresse E relativ (FFCEh absolut)

1. MASK (D0)
Eine Null im MASK-Bit aktiviert den Standardimpuls-Ausgang. Ein laufender Standardimpuls kann durch Schreiben einer Eins in MASK auch unterbrochen werden.

2. I/S Interrupt/Standard (D1)
Kontrolliert die Funktionsweise des Standardimpuls-Ausgangs:
I/S = 0: Standard Modus, der Ausgang bleibt für 7,8125ms Null oder bis er durch IRQ = Null zurückgesetzt wird.
I/S = 1: Interrupt Modus, der Ausgang bleibt Null, bis eine Null in das IRQ geschrieben wird.

3. t0 (D2), t1 (D3)
Diese Bits steuern die Periodizität des Standardimpulses:

t1	t0	Periodendauer
0	0	1/64 s
0	1	1 s
1	0	1 min
1	1	1 std

Register F Adresse F relativ (FFCFh absolut)

1. RES Reset (D0)
Dies Bit setzt alle Zählstufen unter 1 Hz zurück und stoppt den Zählvorgang. Zeit- und Datumsregister werden davon nicht beeinflußt. Das Reset-Bit muß wieder zurückgesetzt werden. CS1 = Null (Reset oder power down) setzt RES ebenfalls zurück.

2. STOP (D1)
Dieses Bit hält den 8192Hz Teiler an, solange es Eins gesetzt ist. Der Standardimpuls-Ausgang verweilt auf dem letzten Pegel.

3. 24/12 (D2)
Mit diesem Bit wird zwischen 12- und 24-Stunden-Modus umgeschaltet. D2 gleich Eins entspricht dem 24-Stunden-Modus. AM/PM wird dann ignoriert.

4. TEST (D3)
Dies Bit ist normalerweise Null und nicht näher erläutert.

MCS® BASIC-52

Basic- quick and dirty?
Es gibt viele Softwareentwickler, die keine Lust haben in die Tiefen der Assemblerprogrammierung herabzusteigen. Wenn man sich nach alternativen Möglichkeiten für die Kommunikation mit einem Mikrocontroller umsieht, findet man eine Menge Produkte, die einem die Arbeit erleichtern. Die angebotenen Crosscompiler ermöglichen die Programmierung in einer Hochsprache, z.B. "C", seltener auch "Pascal". Der Compiler wird auf einem PC gestartet und das Programm für die Zielhardware auch dort geschrieben. Danach wird alles so übersetzt, eben in die Maschinensprache des Zielsystems, daß der Controller uns versteht. Es besteht dann nur noch das Problem, den so aufbereiteten Code dem Prozessor vor die Pins zu legen. Dazu benutzt man in der Regel einen EPROM-Simulator, ein Gerät, was sich dem PC als EPROM-Brenner vorstellt und dem Prozessor als EPROM. Dann erfolgt ein häufig kurzer Test, in dem man sich von der Fehlerhaftigkeit des Programms überzeugt und alles beginnt von vorne. Wenn hoffentlich irgendwann 'mal alles funktioniert, wird mit einem Programmiergerät ein EPROM gebrannt und in den dafür vorgesehenen Sockel gesteckt. Fertig! Unterstützend zu diesem Verfahren kann ein "in circuit emulator" die Fehlersuche erleichtern. Damit ist der Programmierer in der Lage, dem Prozessor während der Abarbeitung des Programms zuzusehen und sich den Kopf über die Geschehnisse zu zerbrechen. Das hat natürlich alles seinen Preis und der liegt in der kommerziellen Ausführung weit über den Kosten für die Zielhardware inclusive 386er AT. Außerdem können Sie eigentlich auch direkt in Maschinensprache programmieren, wenn Sie "C" schon beherrschen. Es ist kaum kryptischer. Aber auch wenn Sie äußerst günstige Entwicklungswerkzeuge erstanden haben, eine schnelle "mal eben" Software gelingt nur selten. Wem dies alles nicht liegt, der findet hier Abhilfe:

Sie nehmen ein beliebiges Terminal (oder einen mit einem Terminalprogramm versehenen Computer), den 8052AH-BASIC-Rechner und verbinden beides mit einem seriellen Schnittstellenkabel. Wenn Sie nun auf dem Terminal die Leertaste drücken, meldet sich der Controller mit:

```
*MCS-51(tm)BASIC V1.1
READY
>
```

und wartet auf Ihre Aufgaben. Die verwendete Baudrate erkennt der Controller selbstständig, deshalb auch die Leertaste. Sie können nun entweder im Direkt- oder Komandomodus arbeiten, d.h., daß die Eingabe "PRINT (12*14)" Ihnen das Ergebnis (168) liefert, oder unter Zuhilfenahme von Zeilennummern Basicprogramme kreieren. Ein einfacher Zeileneditor erleichtet die Eingabe. Der Interpreter legt die Zeilen, codiert, im externen RAM ab und führt nach dem Befehl "RUN" das Programm aus.

Einige Vor- und Nachteile:
Programme können auf der Rechnerkarte (geeignete Programmierspannung vorausgesetzt) in ein, für Basic reserviertes EPROM gebrannt werden und von dort aus ablaufen. Mit "RAM" und "ROMn" werden die Programme aktiviert und können dann mit "RUN" gestartet oder mit "LIST" angesehen werden. (n ist die Nummer des Programms im ROM, hier können mehrere, im RAM nur ein Programm abgelegt werden.) Das EPROM wird mit dem Befehl "PROG" gebrannt. Ferner gibt es Befehle, um ein Löschen der Programme im RAM nach dem Einschalten zu unterbinden oder den Interpreter anzuweisen, direkt mit der Ausführung eines Programms zu beginnen (PROGx, für Anwendungen ohne Terminal).
Mit "CALL" werden selbstgeschriebene Maschinenspracheprogramme aufgerufen, und auch die Kreation eigener Basicbefehle ist möglich. Damit stellt der Interpreter gleichzeitig eine einfache Testumgebung für die Assemblerprogrammierung dar.
"ONEX1" ist eine Sprunganweisung, die bei einem Interrupt ausgeführt wird. Mit "ONTIME" wird ein zeitgesteuerter Interrupt ausgeführt. Eine Echtzeituhr befindet sich auf dem Chip, sie vergißt leider die Zeit mit dem Abschalten der Betriebsspannung. Der Befehl "IDLE" legt den Prozessor schlafen bis eine Unterbrechungsanforderung eintritt.
"PUSH" und "POP" sind Befehle zum Beschreiben und Lesen des Basicstacks, mit "PWM" wird eine Rechteckspannung auf einem Pin ausgegeben. Neben diversen Befehlen zum Steuern der Timer und Ports, läßt sich mit "XBY","DBY" und "CBY" auf internes-, externes RAM und ROM zugreifen (Die Befehle PEEK und POKE gibt es hier nicht).

Wie vielleicht schon aus dieser kurzen Beschreibung hervorgeht, hat dieses Basic eine Menge Unvollkommenheiten, aber auch viele Möglichkeiten die Hardware zu steuern. Es läßt sich damit jedenfalls alles programmieren, und es ist relativ schnell in der Ausführung.

Es folgt nun zunächst eine Beschreibung der verwendeten Begriffe und einiger Besonderheiten. Dann schließt eine Auflistung der Befehle an, so daß sich ein erfahrener Programmierer schnell eine Übersicht verschaffen kann. Auf den nachfolgenden Seiten finden Sie auch eine alphabetisch sortierte Beschreibung aller Befehle. Hier wurde in erster Linie Wert darauf gelegt, eine Anwendung unter Basic zu ermöglichen. Befehle zur Kopplung von Basic und Maschinensprache wurden bewußt kurz erklärt, da dies den Rahmen sprengen würde.

Begriffsbestimmung:
Das Basic des 8052 läßt zwei verschiedene Betriebsmodi zu: Den Kommando-Modus (Command-mode) und den Programm-Modus (Run-mode). Wer einen C-64 (Commodore) oder ähnlichen Homecomputer hat wird die Unterscheidung kennen. Um ein Betriebssystem einzusparen, führt der Rechner im Kommando-Modus die eingegebenen Befehle direkt aus. Notwendig sind z.B. die Befehle zum Löschen des Speichers (NEW) und Starten eines Programms (RUN). In der alphabetischen Auflistung weiter unten sind diese Steuerbefehle alle mit "Command" gekennzeichnet. Startet man ein Programm, so wechselt der Interpreter in den Programm-Modus. Die innerhalb eines Programms auszuführenden Befehle heißen im Englischen "Statements" (allerdings auch die Programmzeilen als Ganzes). Ebenso können Operatoren im Programm ausgeführt werden. Dazu gehören mathematische Operatoren wie +, -, * usw., als auch Befehle, die direkt auf die Hardware des Rechners wirken. Einige der Statements und alle Operatoren lassen sich auch im Kommando-Modus ausführen. Beispielsweise zeigt PRINT a (CR) den Wert der Variable a auf dem Terminal an. In der alphabetischen Auflistung ist jeweils vermerkt, in welchem Modus sich das Statement anwenden läßt.

Jede Programmzeile beginnt mit einer Zeilennummer und ist mit Return (CR) abzuschließen. Label, wie in moderneren Basic-Dialekten, sind nicht zulässig. Bei Sprunganweisungen muß folglich auch immer die Zielzeile durch eine Nummer angegeben werden. Die Zeilennummern liegen im Bereich von 0 bis 65535 und dürfen natürlich nur einmal vorkommen. Die Reihenfolge der Zeilen ist durch ihre Nummern bestimmt, die Reihenfolge ihrer Eingabe ist beliebig. Verwendet man eine Zeilennummer zum zweiten Mal, so wird die alte Zeile durch die neue ersetzt. In einer Zeile dürfen 79 Zeichen stehen. Gibt man, im Eifer des Gefechts, doch mehr Zeichen ein, ertönt ein Warnsignal am Terminal. Leerzeichen werden vom 8052 beim Speichern unterdrückt und beim Listen wieder eingefügt. Einrückungen im Programmlisting sind nicht möglich. In einer Zeile sind auch mehrere Statements zulässig, wenn man sie durch Doppelpunkte trennt. Nach einem Doppelpunkt ist keine neue Zeilennummer zulässig.

Zeileneditor:
Das MCS BASIC-52 stellt Ihnen einen einfachen Zeileneditor zur Verfügung. Die Zeichen dürfen mit großen oder kleinen Buchstaben eingegeben werden. Beim Listen erscheinen nur große Buchstaben, außer hinter REM und in den Anführungszeichen eines PRINT-Statements. Das jeweils letzte Zeichen in der aktuellen Zeile läßt sich mit Delete (7Fh) löschen, Control-D löscht die ganze Zeile. Backspace ist leider ohne Funktion (kann DOS mehr?). Jede Zeile muß mit Carriage return (CR) abgeschlossen werden. Ein X-ON, X-OFF Handshake ist möglich. Dabei wird die Ausgabe auf das Terminal angehalten, wenn der Interpreter das Zeichen Control-S empfängt und wieder gestartet durch die Eingabe von Control-Q. Diese Funktion arbeitet zusammen mit dem LIST- und PRINT-Statement und wirkt jeweils am Ende einer Zeile. Einfache Programme lassen sich so nur mit einem einfachen Terminal eingeben. Umfangreiche Abläufe programmiert man am besten mit einem Editor und sendet die erstellte Datei (ASCII mit Zeilennummern) über die serielle Schnittstelle an den Interpreter. Nach jedem Carrige return (CR) benötigt der Rechner eine kurze Zeit, um die Zeile zu tokenisieren und im RAM abzulegen. Ist das erledigt, meldet sich der 8052 mit einem ">"-Zeichen und ist bereit die nächste Zeile zu empfangen. Entweder überwacht man diesen Vorgang mit einem speziellen Programm oder man läßt das Terminalprogramm nach jeder Zeile etwa 1/2 Sekunde warten. Eine elegantere Lösung bietet das Programm "COMPRETER-52", welches dann auch gleichzeitig eine komplette Entwicklungsumgebung bietet und Variablenkonflikte mit dem Basic überwacht.

Datentypen:
Der 8052 kann Zahlen im Bereich von +/-1 * 10^{-127} bis +/-0,99999999 * 10^{+127} verarbeiten. Die Genauigkeit beträgt acht Stellen und die Variablenwerte sind entsprechend gerundet. Folgende Ein- und Ausgabeformate sind möglich:

Integer	Beispiel:	127
Dezimal		16,08
Hexadezimal		0FFh
Dezimal mit Zehnerexponent		1,43E -4 (=1,43 * 10^{-4})

Ist einer Variable ein Wert zugewiesen, so benötigt das immer gleich viel Speicher, unabhängig vom Format des Wertes.

Integer:
Einige Operationen sind nur mit Integerzahlen zulässig. Eine Integerzahl hat keine Nachkommastellen und liegt stets im Bereich von 0 bis 65535 oder 0FFFFh. Integerzahlen können immer in dezimaler oder hexadezimaler Form eingegeben werden. Die Darstellungsform bei der Ausgabe auf dem Terminal hängt von dem verwendeten Befehl (PRINT, PH0. oder PH1.) ab. Integerwerte müssen immer (!) mit einer Dezimalziffer in der ersten Stelle beginnen, deshalb auch die Null in 0FFFFh. Diese Regel ist eine häufige Fehlerursache. Basic erkennt an der führenden Dezimalziffer, daß es sich um eine Zahl und nicht um eine Variable handelt. Operatoren, die Integerzahlen verarbeiten, schneiden die Nachkommastellen ab, wenn versehentlich eine Dezimalzahl eingegeben wird.

Variablenerkennung:
Variablennamen dürfen aus max. acht Buchstaben und/oder Ziffern bestehen und werden durch den ersten, den letzten und die Anzahl der Buchstaben unterschieden. Der Unterstrich ist ebenfalls im Namen zulässig. Ferner gibt es String- und (eindimensional) dimensionierte Variablen (siehe STRING- und DIM-Statement). Basicschlüsselworte dürfen nicht in dem Namen enthalten sein. "FORTRAN" und "FOR_MAN" sind also nicht zu unterscheidende Variablen, die beide zu einem Fehlverhalten führen, da das Schlüsselwort "FOR" enthalten ist. Je länger der Variablenname ist, um so größer wird die Verarbeitungszeit, dimensionierte Variablen werden ebenso etwas langsamer verarbeitet. Ein wesentlicher Fehler des Interpreters zeigt sich bei der Verarbeitung von Variablen mit "U" und "F" am Anfang oder Ende des Namens. Die Buchstaben werden häufig (nicht immer) verschluckt, tauchen also im Listing nicht mehr auf. Diese Effekt wird durch Befehlssatzerweiterungen in Assembler, evtl. auch auf andere Buchstaben ausgedehnt.

Strings:
Strings oder Zeichenketten können beliebige Buchstaben, Zeichen und Ziffern enthalten. Das MCS BASIC-52 läßt maximal 255 eindimensionale Stringvariablen zu. Die Strings sind, mit Null beginnend, durchnummeriert. Stringvariablen bestehen immer aus dem "Dollarzeichen" und der Nummer in Klammern, "$(x)". Die Verwendung von Zeichenketten ist z.B. nützlich, wenn ein Programm einen Text mehrmals ausgeben soll. Er wird dann einfach anhand seiner Nummer aufgerufen, z.B. mit der Anweisung PRINT $(2). Vor der Verwendung von Zeichenketten muß man mit der STRING-Anweisung Speicherplatz reservieren. Weder NEW noch CLEAR geben den reservierten Speicher wieder frei, sondern nur die Anweisung STRING 0,0. Jede STRING-Anweisung löscht den gesammten Variablenspeicher, da sich Variablen und Strings hintereinander im RAM befinden. Die Texte lassen sich mit den Anweisungen ASC() und CHR() beeinflussen und mit INPUT über die Tastatur einlesen.

Rangfolge der Operatoren:
Operatoren innerhalb einer Programmzeile werden in einer bestimmten, mathematisch sinnvollen Reihenfolge ausgewertet. So kommt man mit einem Minimum an Klammern aus. Der Rechner wertet den Ausdruck von links nach rechts aus und berechnet einen Term erst, wenn kein Ausdruck mit höherer Rangfolge gefunden wird. Die Operatoren in ihrer Reihenfolge:

1. Klammerausdrücke ()

2. Potenzen (**)

3. Negation (-)

4. Multiplikation (*) und Division (/)

5. Addition (+) und Subtraktion (-)

6. Relative Ausdrücke (=, <>, >, >=, <, <=)

7. Logisches UND (.AND.)

8. Logisches ODER (.OR.)

9. Logisches Exklusiv-ODER (.XOR.)

Fehlermeldungen:
Laufzeitfehler werden vom Basic in äußerst komfortabler Form angezeigt:

```
ERROR: typ - IN LINE num
num statement
-----------X
```

Der Fehlertyp wird angegeben und nachfolgend die vermutlich fehlerhafte Zeile (num statement). Zusätzlich gibt der Interpreter aber auch durch das Kreuz die ungefähre Position des Fehlers an. Im Kommando-Modus meldet der Interpreter nur den Fehlertyp, da keine Zeilennummern existieren. Die Fehlertypen sind:

BAD SYNTAX	unverständliche Anweisung oder unbekannter Operator
BAD ARGUMENT	Argument außerhalb des erlaubten Zahlenbereichs
ARITH. UNDERFLOW	Rechenergebnis der Operation zu klein (Fließkommazahl < |+/-1E -127|)
ARITH. OVERFLOW	Rechenergebnis der Operation zu groß (Fließkommazahl > |+/- 99999999E +127|)
DIVIDE BY ZERO	Division durch Null
NO DATA	READ-Anweisung findet keine Daten mehr Keine DATA-Zeilen oder schon alle Daten gelesen
CAN'T CONTINUE	CONT-Anweisung kann nicht ausgeführt werden Programmende erreicht oder Programm verändert
PROGRAMMING	Fehler bei EPROM-Programmierung, EPROM defekt oder kein Speicherplatz mehr, keine Programmierspannung
A-STACK	Argumentstack läuft über oder ist leer, zu viele PUSH- oder POP-Anweisungen
C-STACK	Zu viele Schleifen verschachtelt, RETURN ohne GOSUB, WHILE oder UNTIL ohne DO, NEXT ohne FOR
I-STACK	Zu viele arithmetische Operationen in einer Zeile oder fehlerhafte Maschinenspracheroutine
ARRAY SIZE	Feldvariable liegt außerhalb der vereinbarten Dimensionierung
MEMORY ALLOCATION	String zu groß, MTOP auf ungültigen Wert gesetzt

Control-C:
Der Ablauf eines Basic-Programms läßt sich durch die Eingabe von Control-C anhalten. Damit kann man häufig das Programm retten, wenn der Rechner in einer Endlosschleife hängt. Der Griff zum Reset-Taster bleibt einem erspart. Nun kann die Unterbrechung in einem zeitkritischen Programmteil aber unerwünscht sein. Dies gilt z.B. für das Einstellen oder Abfragen der Uhrzeit aus dem RTC-Baustein. Die Control-C-Funktion ist abschaltbar, wenn man Bit 48 (30h) im internen RAM (bitadressierbar) auf Eins setzt. Das Bit findet sich im Speicher unter der Adresse 38.0 (26.0h). Die Anweisung:

```
DBY(38) = DBY(38) .OR. 01h
```

setzt das Bit. Das kann sowohl im Programm-, als auch im Kommando-Modus geschehen. Die Funktion wird durch folgende Anweisung wieder aktiv:

```
DBY(38) = DBY(38) .AND. 0FEh
```

Fehler im Interpreter:
Fehlfunktionen können auftreten, wenn auf eine Zeilennummer ohne Leerzeichen der Buchstabe "H" folgt. Der Interpreter wertet die Zeilennummer als hexadezimale Angabe.
Steht vor einer ELSE-Anweisung der Buchstabe "I", so muß er durch ein Leerzeichen getrennt werden. Anderenfalls liest der Interpreter daraus die IE-Anweisung.
In den Klammern einer ASC-Anweisung darf kein Leerzeichen stehen. Der Interpreter entfernt die Leerzeichen beim Tokenisieren der Zeile und die Klammer ist dann leer. Zur Laufzeit meldet der Rechner einen BAD ARGUMENT ERROR. Die ASC-Anweisung kann jedoch Leerzeichen aus Strings lesen.
Variablennamen, die die Buchstaben "I" und "U" enhalten, werden nicht immer richtig gespeichert. Verwendet man eigene Befehlserweiterungen, so dehnt sich dieser Fehler auch auf andere Buchstaben aus.

Control Stack:
Mit dem Controlstack verwaltet der Rechner Schleifen- und GOSUB-Anweisungen. Die Rücksprungadressen sind hier gespeichert. Der Stack belegt den Speicher 60h bis 0FEh (158 Byte) im externen RAM (XBY). Die Startadresse ist 0FEh und der Stack wächst nach unten. FOR-NEXT-Schleifen belegen 17 Byte, DO-Schleifen 3 Byte.

Argument Stack:
Die Konstanten der aktuellen Operation speichert der Interpreter auf dem Argumentstack. Bei einer Addition werden z.B. zwei Speicherplätze für den ersten und zweiten Operanden benutzt. Auch das Ergebnis wird hier abgelegt. Außerdem benutzen die PUSH- und POP-Anweisungen diesen Speicher. Jeder Wert benötigt 6 Byte, unabhängig davon, ob es sich um eine Fließkommazahl oder einen Integerwert handelt. Der Stack liegt im externen RAM auf den Adressen 12Dh bis 1FEh und wächst, bei 1FEh beginnend, nach unten.

Internal Stack:
Den internen Stack benutzt der Interpreter bei der Abarbeitung der Basic- Befehle. Der Anwender hat keinen Einfluß auf diesen Speicher, solange der Rechner nur in Basic genutzt wird. Der Stackpointer (SP) ist ein Special- Funktion-Register und in Basic nicht zugänglich. Er wird auf 4Dh (internes RAM) initialisiert und wächst nach oben.

Special-Function-Register:
Die Special-Function-Register sind Speicherplätze im Controllerbaustein, die direkt die integrierten Funktionen steuern. Dazu gehören z.B. die Speicher für Ausgangsleitungen (Port 0...3), Interrupt-Prioritäts- und Freigabe-Register, Timer-Register, aber auch der Akkumulator und andere mehr. Solange der Anwender nur in Basic programmiert, sind die SFR weitgehend bedeutungslos. Alle Funktionen werden vom Basic überwacht. Möchte man eigene Befehle hinzufügen oder direkt in Assembler programmieren, so ist es unbedingt nötig, sich mit dem internen Aufbau der 8051-Controller zu befassen. Beschreibungen zu diesen Prozessoren finden sich in jeder gut sortierten Buchhandlung.

Initialisierung:
Nach einem Reset oder dem Einschalten erledigt der Interpreter die folgenden Aufgaben:

1. Das interne RAM im 8052 wird gelöscht.
2. Interne Register und Stackpointer werden initialisiert.
3. Der Arbeitsspeicher (RAM hier 43256, IC12, Adresse 0000h bis 7FFFh) wird getestet und gelöscht. Dabei wird auch die Speichergröße festgehalten.
4. Einige Variablen wie MTOP und der Wert der Quarzfrequenz XTAL werden dem System zugewiesen.
5. Die Adresse 8000h im EPROM wird gelesen. Hier kann ein Wert für die Baudrate gespeichert sein. Wird kein Wert gefunden wartet der Interpreter auf ein Space (Leerzeichen) von seriellen Port um die Baudrate selbstständig zu bestimmen.
6. An das Terminal wird die Startmeldung ausgegeben.

Der Anwender kann nach dem Reset überprüfen, ob die Initialisierung korrekt erfolgte.

Die Eingabe von:	PRINT XTAL, TMOD, TCON, T2CON (CR)
zeigt:	11059200 16 244 52 (hoffentlich)

Das bedeutet, daß das System (serielle Schnittstelle, RAM und Takterzeugung) einwandfrei funktioniert.

Auf den folgenden Seiten soll ein kurzer Überblick über die Leistungsfähigkeit des Basic-Interpreters im 8052AH 1.1 gegeben werden. Die Darstellung der Befehle erhebt keinen Anspruch auf Vollständigkeit. Genaue und vollständige Informationen entnehmen Sie bitte dem Intel-Handbuch:

[1] INTEL: MCS BASIC-52 User's Manual. INTEL, 1989, ISBN: 1555-12-0857

Weiterführende Informationen finden Sie in:

[2] J.P.M. Steeman: 8052 AH-BASIC. Elektor-Verlag, 1989, ISBN: 3-921608-72-4
[3] INTEL: Embedded Controller Handbook, Vol. 1, 8-Bit. INTEL, 1987, ISBN: 1-55512-072-5
[4] VALVO: Die 8bit-Mikrocontroller-Familie 8051, Bd1 Eigenschaften. Verlag Boysen + Maasch, 1984, ISBN: 3-87095-260-1
[5] VALVO: Die 8bit-Mikrocontroller-Familie 8051, Bd2 Befehlsvorrat. Verlag Boysen + Maasch, 1984, ISBN: 3-87095-260-X

Die Gliederung ist alphabetisch und bezieht sich nur auf die Interpreter Version 1.1

Basic-Interpreter 8052AH Version 1.1 - Befehlsübersicht

Commands	Statements	Operators
RUN	BAUD	+
CONT	CALL	/
LIST	CLEAR	*
LIST#	CLEARS	**
LIST@	CLEARI	-
NEW	CLOCK0	.AND.
NULL	CLOCK1	.OR.
RAM	DATA	.XOR.
ROM	READ	ABS()
XFER	RESTORE	INT()
PROG	DIM	SGN()
PROG1	DO-WHILE	SQR()
PROG2	DO-UNTIL	RND
PROG3	END	LOG()
PROG4	FOR-TO-STEP	EXP()
PROG5	NEXT	SIN()
PROG6	GOSUB	COS()
FPROG	RETURN	TAN()
FPROG1	GOTO	ATN()
FPROG2	ON-GOTO	=,>,<,>=,<=,<>
FPROG3	ON-GOSUB	ASC()
FPROG4	IF-THEN-ELSE	CHR()
FPROG5	INPUT	CBY()
FPROG6	LET	DBY()
	ONERR	XBY()
	ONEX1	GET
	ONTIME	IE
	PRINT	IP
	PRINT#	PORT1
	PRINT@	PCON
	PH0.	RCAP2
	PH0.#	T2CON
	PH0.@	TCON
	PH1.	TMOD
	PH1.#	TIME
	PH1.@	TIMER0
	PGM	TIMER1
	PUSH	TIMER2
	POP	XTAL
	PWM	MTOP
	REM	LEN
	RETI	FREE
	STOP	PI
	STRING	NOT()
	UI0	
	UI1	
	UO0	
	UO1	
	LD@	
	ST@	
	IDLE	
	RROM	

***** Operator
Kommando- und Programm-Modus
Multiplikation Beispiel: 3 * 5

/ Operator
Kommando- und Programm-Modus
Division Beispiel: 4 / 2

****** Operator
Kommando- und Programm-Modus
Potenzen Beispiel: 2 ** 3 (zwei hoch drei)
Diese Notation wurde gewählt, da ein Terminal den Pfeil "^" evtl. als Steuerzeichen auffaßt.

+ Operator
Kommando- und Programm-Modus
Addition Beispiel: 3 + 7

- Operator
Kommando- und Programm-Modus
Subtraktion Beispiel: 8 - 2

.AND. Operator
Kommando- und Programm-Modus
Logisches UND. Die bitweise Verknüpfung ist nur auf Integer-Werte im Bereich 0 bis 65535 (0FFFFh) anwendbar. Fließkommazahlen in diesem Bereich werden vor dem Komma abgeschnitten. Die Punkte grenzen den Befehl gegen Variablen davor oder dahinter ab.
Beispiel: 22h .AND. 0Fh (= 02h)

.OR. Operator
Kommando- und Programm-Modus
Logisches ODER. Die bitweise Verknüpfung ist nur auf Integer-Werte im Bereich 0 bis 65535 (0FFFFh) anwendbar. Fließkommazahlen in diesem Bereich werden vor dem Komma abgeschnitten. Die Punkte grenzen den Befehl gegen Variablen davor oder dahinter ab.
Beispiel: 01h .OR. 20h (= 21h)

.XOR. Operator
Kommando- und Programm-Modus
Logisches EXKLUSIV-ODER. Die bitweise Verknüpfung ist nur auf Integer-Werte im Bereich 0 bis 65535 (0FFFFh) anwendbar. Fließkommazahlen in diesem Bereich werden vor dem Komma abgeschnitten. Die Punkte grenzen den Befehl gegen Variablen davor oder dahinter ab.
Beispiel: 06h .XOR. 05h (= 03h)

=, >, <, >=, <=, <> Operator
Kommando- und Programm-Modus
Die relativen Ausdrücke testen in Schleifen und Verzweigungen auf Identität, Größer, Kleiner, Größer-Gleich, Kleiner-Gleich und Ungleich. Das Ergebnis ist entweder Wahr (0FFFFh) oder Falsch (0h). Die Operatoren lassen sich mit den logischen Funktionen verknüpfen.
Beispiel: IF a = b .AND. a <> c THEN...

ABS([expr]) Operator

Kommando- und Programm-Modus

Bestimmt den Absolutwert (vorzeichenfrei) des auf ABS folgenden Ausdrucks.

ASC([expr]) Operator

Kommando- und Programm-Modus

Der ASC-Operator liefert den ASCII-Wert (Integer 0h...0FFh) des eingeklammerten Buchstabens [expr]. ASC kann auch einzelne ASCII-Werte aus Strings bestimmen. In der Klammer muß dann erst die String-Nummer und, durch ein Komma getrennt, der gewünschte Buchstabe stehen. Die Buchstaben eines Strings sind mit Eins beginnend durchnummeriert.

Beispiel: $(1) = "Hallo"

ASCII : ASC($(1),1) (=72, "H")

Außerdem kann ASC einer bestimmten Position innerhalb eines Strings einen neuen Buchstaben zuweisen. Der Operator steht dazu links vom Gleichheitszeichen.

Beispiel: ASC($(2),5) = ASC(A)

setzt auf die fünfte Position im String Nummer 2 den Buchstaben "A".

Vgl.: STRING, CHR

ATN([expr]) Operator

Kommando- und Programm-Modus

Berechnet den Arcus Tangens des Arguments. Das Ergebnis, der Winkel, ist im Bogenmaß mit den Hauptwerten zwischen +/- Pi/2 angegeben. Die Berechnung erfolgt mit einer Taylor-Reihe und liefert 7 signifikante Stellen.

BAUD Statement

Syntax: BAUD [expr]

Kommando- und Programm-Modus

BAUD stellt die durch [expr] angegebene Baudrate für die serielle Druckerschnittstelle ein. Das Übertragungsformat ist stets 1 Startbit, 8 Datenbit, 2 Stopbit, kein Paritätsbit. Die Einstellung ist nur dann korrekt, wenn der Interpreter die Quarzfrequenz kennt. Der Defaultwert ist 11,0592 MHz. Wird eine andere Frequenz verwendet, so gibt man den Wert mit dem Operator XTAL an. Wurde vor der Ausgabe mit PRINT# der Wert BAUD nicht eingestellt, so sendet der 8052 mit etwa 1 Baud. Das dauert sehr, sehr lange, und man könnte denken, der Rechner ist tot. Bitte unbedingt zuerst initialisieren! Ein Wert von <15 für BAUD führte immer zu einer Fehlermeldung, Nachkommastellen sind offenbar ebenfalls nicht zulässig. Ein unsinniger Wert von z.B. 1 Million wird akzeptiert, das Resultat wurde aber nicht erprobt.

Der Druckerport ist unidirektional, kann also nur senden und unterstützt keinen Handshake. Eine Erweiterung zum bidirektionalen Betrieb mit RXD oder CTS ist mit der Hardware des 8052-ECB möglich. Dazu muß in einer Maschinenspracheroutine der Port P1.0 (T2) eingelesen werden.

Vgl.: XTAL, LIST#, PRINT#

CALL Statement

Syntax: CALL [int]

Kommando- und Programm-Modus

Call startet ein Maschinenspracheprogramm. Das Programm muß im Assembler-
EPROM (oder im EPROM-Emulator) ab der Adresse [int] stehen. Call läd den Wert [int] in den prozessorinternen Befehlszähler und rettet vor der Verzweigung die Zeilennummer des auf CALL folgenden Statements auf den Stack. Wenn die Maschinenroutine mit RET endet, wird das Basicprogramm danach fortgesetzt. So ist es möglich, den ohnehin sehr leistungsfähigen Befehlssatz des 8052 zu erweitern und der angeschlossenen Peripherie anzupassen.

Für Argumente [int] kleiner 80h berechnet der Interpreter eine vektorisierte Startadresse. Diese ergibt sich zu ADR = 4100h + 2 * [int] . Mit CALL 0 wird ein Programm ab Adresse 4100h gestartet, CALL 1 führt zu 4102h usw.

Die Programme können auch im Basic-EPROM des 8052-ECB eingebracht werden, da dieser Bereich sowohl Code- als auch Datenspeicher ist.

Vgl.: PRINT@, LIST@

CBY([expr]) Operator

Kommando- und Programm-Modus

CBY liest ein Byte aus dem Programmspeicherbereich des 8052. Die Adresse ist durch [expr] angegeben, Integer-Werte zwischen 0 und 65535 (0FFFFh) sind zulässig. Zum Programmspeicher gehört der prozessorinterne Basicinterpreter (0...1FFFh), das Assembler-EPROM (IC14)(0...7FFFh) und auch das Basic-EPROM (IC13)(8000h...0BFFFh). Aus dem Programmspeicher kann der 8052 nur lesen, d.h., CBY kann nur rechts von Zuweisungszeichen stehen.

Beispiel: a = CBY(8000h)

Vgl.: XBY, DBY

CHR([expr]) Operator

Kommando- und Programm-Modus

CHR liefert den Buchstaben zu dem in Klammern angegebenen ASCII-Wert [expr]. Der Operator ist das Gegenstück zu ASC (siehe oben) und die Syntax ist ähnlich. CHR ergibt nur in Verbindung mit der PRINT-Anweisung einen Sinn (was soll ein Rechner auch sonst mit Buchstaben anfangen?).

Beispiel: PRINT CHR(65) ("A")
PRINT CHR($(2),5) (5ter Buchst. aus String Nr.2)

Anders als mit ASC kann CHR keinen Buchstaben zuweisen. Der Ausdruck CHR($(1),1) = A ist unzulässig, hier muß man den ASC-Operator anwenden.

Vgl.: STRING, ASC, PRINT

CLEAR Statement

Syntax: CLEAR

Kommando- und Programm-Modus

Das CLEAR-Statement weist allen Variablen den Wert Null zu und setzt den Basic-Stackpointer zurück. CLEAR sperrt alle Basic-Interrupts, d.h., ONEX1, ONTIME und ONERR sind bei Bedarf nach der CLEAR-Anweisung erneut auszuführen. Die Echtzeituhr im 8052 und der für Strings reservierte Speicher bleiben unbeeinflußt.

Vgl.: CLEARI, CLEARS, ONTIME, ONEX1, ONERR

CLEARI Statement

Syntax: CLEARI

Kommando- und Programm-Modus

CLEARI sperrt alle Basic-Interrupts. Die Uhrenfunktion im 8052 wird nicht beeinflußt. Mit diesem Befehl kann in einem zeitkritischen Programmteil jede Unterbrechung von außen verhindert werden. Das ist z.B. wichtig beim Lesen oder Stellen des RTC-Bausteins (72421, IC10). Eine Interruptroutine, die den Auslesevorgang für länger als eine Sekunde unterbricht, würde die Uhr verstellen. Nach der Abarbeitung dieses zeitkritischen Teilprogramms sind gegebenenfalls die Befehle ONEX1 und ONTIME erneut auszuführen. CLEARI setzt die Bits 2 und 3 im Special Function Register IE zurück und löscht das Statusbit. Im Listing (LIST) erscheint CLEARI als CLEAR I.

Vgl.: CLEAR, CLEARS, ONTIME, ONEX1, ONERR

CLEARS Statement

Syntax: CLEARS

Kommando- und Programm-Modus

CLEARS initialisiert alle Basic-Stacks. Dazu gehören der Argument-Stack (siehe PUSH und POP) und der Control-Stack für die Organisation von Unterprogrammen und Schleifen. In den Stackpointer (Special Function Register SP) wird der Inhalt der Speicherzelle 3Eh (internes RAM) kopiert. Mit dem CLEARS-Befehl ist es möglich FOR-NEXT- und DO-Schleifen ungeschoren zu verlassen, bevor sie beendet sind. Das Programm wird dadurch jedoch nicht gerade übersichtlicher und man sollte unbedingt die Wirkung auf den Argument-Stack beachten. Im Listing (LIST) erscheint CLEARS als CLEAR S.

Vgl.: CLEAR, CLEARI

CLOCK0 Statement

Syntax: CLOCK0
Kommando- und Programm-Modus
Der Befehl CLOCK0 hält die Uhrenfunktion des 8052 an. Zwei Bits im Register IE werden gelöscht (siehe CLOCK1) und der Wert der Variable TIME bleibt fortan konstant. Der Interrupt durch Timer/Counter 0 kann nach CLOCK0 auch in eigenen Assemblerroutinen genutzt werden. CLOCK0 ist der einzige Befehl zum Anhalten der Uhr, sie wird durch CLEAR und CLEARI nicht beeinflußt.
Vgl.: CLOCK1, TIME, ONTIME

CLOCK1 Statement

Syntax: CLOCK1
Kommando- und Programm-Modus
CLOCK1 startet die Echtzeituhr im 8052. Die Uhrenfunktion ist mit Hilfe des internen Timer/Counter 0 im 13 Bit-Modus realisiert. Der Timer erzeugt alle 5 ms einen Interrupt und inkrementiert damit den Wert des Special-FunctionOperator TIME. Die Uhr läuft also in 5 ms Schritten. Dabei ist vorrausgesetzt, daß die Variable XTAL den korrekten Wert der Quarzfrequenz enthält. Der Defaultwert ist 11,0592 MHZ. Die Variable TIME zählt von 0 bis 65535,995 Sekunden, der Überlauf setzt den Wert auf Null zurück. CLOCK1 startet zwar die Uhr, beeinflußt aber nicht den vorhandenen Wert von TIME. Die Uhrenfunktion benötigt 0,4% der Rechenzeit des 8052. Durch die Uhr ist der Timer~ /Counter 0 und der dazugehörige Interrupt belegt und kann nicht mehr in einer Assemblerroutine benutzt werden. Der CLOCK1-Befehl setzt unter anderem die Bits 2 und 7 im Special-Function-Register IE. Die Uhr ist natürlich nach einem Spannungsausfall außer Funktion gesetzt und "vergißt" die Zeit. Deshalb ist auf dieser Controllerkarte eine zusätzliche RTC vorhanden. Dennoch läßt sich die Funktion sinnvoll nutzen, um z.B. eine zeitinterrupt gesteuerte Erfassung von Meßdaten durchzuführen.
Vgl.: CLOCK0, TIME, ONTIME

CONT Command

Syntax: CONT (CR)
Kommando-Modus
Wurde ein Programm durch den Empfang von Control-C oder durch die Anweisung STOP angehalten, so startet es nach der Eingabe von CONT (CR) an der gleichen Stelle. Das funktioniert nicht, wenn ein Laufzeitfehler auftrat, der Interpreter also ERROR ... auf das Terminal ausgibt. Während der Unterbrechung des Programms kann man sich Variablenwerte mit PRINT oder PH0. anzeigen lassen oder auch verändern. Das Programm selbst darf nicht verändert werden. Durch die STOP-Anweisung ("Breakpoints") sind Fehler leichter aufzufinden.
Vgl.: STOP, PRINT, PH0.

COS([expr]) Operator

Kommando- und Programm-Modus
Berechnet den Cosinus des Winkels. Das Argument [expr] soll im Bogenmaß ange- geben werden und im Bereich +/- 200000 liegen. Das Ergebnis wird mit einer Taylor-Reihe berechnet und liefert 7 signifikante Stellen. Der Interpreter reduziert den Ausdruck [expr] zuvor auf Werte zwischen +/- Pi/2. Durch die dabei auftretenden Rundungsfehler ist es günstig, das Arument so klein wie möglich zu halten.

DATA-READ-RESTORE Statement

Syntax: DATA [expr1], [expr2], ..., [expr n]
READ [var1], [var2], ..., [var m]
RESTOR
Programm-Modus
DATA legt nummerische Ausdrücke oder Strings [expr x] im Speicher ab und reserviert den nötigen Platz. Mehrere Ausdücke sind duch Kommata zu trennen. Es dürfen mehrere DATA-Anweisungen an beliebigen Stellen im Programm stehen, um mehr Speicher zu reservieren. DATA-Zeilen werden nicht ausgeführt.
READ weist den nachfolgenden Variablen [var x] die aufeinanderfolgenden Werte aus den DATA-Zeilen zu. Mehrere Variable sind auch hier durch Kommata zu trennen. Mehrere READ-Zeilen lesen die DATA-Werte in ihrer Reihenfolge ein. Werden mehr READ-Zuweisungen ausgeführt als Daten vorhanden sind, (m>n) erfolgt eine Fehlermeldung.

Welche Daten an die folgende READ-Zuweisung übergeben werden, wird durch einen Zeiger auf den jeweils nächsten DATA-Wert gesteuert. RESTORE setzt diesen Zeiger zurück, so daß eine nachfolgende READ-Anweisung wieder den ersten DATA-Ausdruck liest.
Mit dieser Befehlsfolge kann man relativ übersichtlich die Adressen der angesteuerten Peripherieschaltungen an das Programm übergeben. Eventuelle Änderungen sind dann nur innerhalb der DATA-Zeilen nötig. Ebenso ist eine Konvertierungstabelle realisierbar, um z.B. die Daten einer Tastatur in ASCII-Werte zu wandeln.

DBY([expr]) Operator

Kommando- und Programm-Modus
Mit DBY ist der Zugriff auf das prozessorinterne RAM möglich. Da der Speicher nur 256 Byte groß ist, muß die Adresse [expr] im Bereich 0 bis 255 (0FFh) liegen. Das interne RAM kann gelesen und beschrieben werden, DBY darf also links und rechts vom Zuweisungszeichen stehen.
Beispiel:
```
a = DBY(38)
DBY(38) = DBY(38) .OR. 01h
```
Vgl.: XBY, CBY

DIM Statement

Syntax: DIM [var]([int])
Kommando- und Programm-Modus
DIM reserviert Platz für eindimensionale Felder. Der 8052 kann eindimensionale Feldvariablen mit einer maximalen Größe von 254 Elementen [int] verwalten. Eine einmal in einem Programm vereinbarte Dimensionierung darf nicht nachträglich verändert werden, d.h., der Befehl DIM darf für eine Variable [var] nur einmal in einem Programm auftauchen. Anderenfalls meldet der Interpreter einen "ARRAY SIZE ERROR". Innerhalb einer DIM-Vereinbarung können auch mehrere Variablen dimensioniert werden, sie sind dazu durch Kommata zu trennen. Wird in einem Programm eine Feldvariable verwendet ohne zuvor eine Dimensionierung mit DIM durchzuführen, so setzt der Interpreter selbstständig einen Defaultwert von zehn Elementen fest. Die Ausführung eines NEW-, RUN- oder CLEAR-Kommandos setzt die Feldgröße auf Null zurück. Der benötigte Speicherplatz für jedes vereinbarte Feld beträgt ([int] + 1) * 6 Byte.

DO-UNTIL Statement

Syntax:
```
DO
...
UNTIL [rel expr]
```
Programm-Modus
DO-UNTIL ist eine Schleifeninstruktion. Alle Anweisungen zwischen DO und UNTIL werden ausgeführt bis der relative Ausdruck [rel expr] hinter UNTIL wahr ist. Die Bedingung wird am Ende der Schleife überprüft, die Anweisungen in der Schleife werden also mindestens einmal ausgeführt. Die Schleifen können verschachtelt werden.
Vgl.: DO-WHILE, FOR-NEXT

DO-WHILE Statement

Syntax:
```
DO
...
WHILE [rel expr]
```
Programm-Modus
DO-WHILE ist eine Schleifeninstruktion. Alle Anweisungen zwischen DO und WHILE werden ausgeführt solange der relative Ausdruck [rel expr] hinter WHILE wahr ist. Die Bedingung wird am Ende der Schleife überprüft, die Anweisungen in der Schleife werden also mindestens einmal ausgeführt. Die Schleifen können verschachtelt werden.
Vgl.: DO-UNTIL, FOR-NEXT

END Statement

Syntax: END
Programm-Modus
Die END-Anweisung schließt die Programmausführung ab und ist die letzte logische Anweisung im Programm. CONT zeigt danach keine Wirkung mehr, der Rechner ist wieder im Kommandomodus. Die END-Anweisung kann auch entfallen, wenn in der letzten Zeile die letzte logische Anweisung steht. Sollten noch Unterprogramme folgen, ist END notwendig. Anderenfalls werden auch die Unterprogramme abgearbeitet und es kommt zu einem Laufzeitfehler, da der Befehl RETURN keinen Sinn ergibt.
Vgl.: GOSUB-RETURN

EXP([expr]) Operator

Kommando- und Programm-Modus
Berechnet e hoch [expr] ($e^{[expr]}$), mit e = 2,7182818.

FOR-TO-(STEP)-NEXT Statement

Syntax: FOR [var]=[expr 1] TO [expr 2] (STEP [expr 3])
...
NEXT [var]
(STEP [expr 3]) ist optional
Kommando und Programm-Modus
In dieser Schleife werden die zwischen den Zeilen FOR... und NEXT... stehenden Anweisungen mehrfach ausgeführt. Beim Eintritt in die Schleife wird der Variable [var] (Schleifenzähler) der Wert [expr 1] zugewiesen. Nach der Abarbeitung des Schleifenkörpers erhöht [var] jeweils um die Schrittweite STEP [expr 3]. Fehlt die STEP-Anweisung ist die Schrittweite Eins. Der NEXT-Befehl inkrementiert nicht nur, sondern es findet auch eine Abfrage statt. Ist [var] kleiner oder gleich [expr 2], wird der Schleifenkörper erneut durchlaufen. Ist [var] größer als [expr 2], geht es mit der nächsten Anweisung nach NEXT weiter. Es reicht aus, in der letzten Zeile nur NEXT ohne Angabe des Variablennamens [var] anzugeben. Bei verschachtelten Schleifen gehören dann jeweils die inneren FOR- und NEXT-Anweisungen zusammen. Hinter STEP darf auch ein negativer Wert [expr 3] stehen. Eine FOR-NEXT-Schleife läßt sich auch im Kommando-Modus ausführen. Die ganze Schleife incl. Schleifenkörper muß dann in einer Zeile stehen. Z.B. listet der Befehl:

```
FOR i=8000h TO 80FFh: PH0.XBY(i),: NEXT i
```

den Inhalt der ersten 256 Bytes aus dem Basic-EPROM auf dem Terminal.

FPROG Command

Kommando-Modus
Wie PROG, nur das hier ein schnellerer (INTELligenter) Programmieralgorithmus verwendet wird. Dazu wäre es nötig, das EPROM mit 6V Betriebsspannung zu versorgen. Auf der Platine 8052-ECB ist das nicht vorgesehen.
Vgl.: PROG

FPROGx Command

Kommando-Modus
Wie PROGx, nur das hier ein schnellerer (INTELligenter) Programmieralgorithmus verwendet wird. Dazu wäre es nötig, das EPROM mit 6V Betriebsspannung zu versorgen. Auf der Platine 8052-ECB ist das nicht vorgesehen.
Vgl.: PROGx

FREE Operator

Kommando- und Programm-Modus
FREE gibt an, wieviel Arbeitsspeicher für Basic frei ist. Die Angabe bezieht sich immer auf das RAM. Der Speicher oberhalb von MTOP wird nicht mitgezählt. Fragt man FREE im RAM-Modus ab, so gilt:
FREE = MTOP - LEN - 511. (511 Byte sind für Basic reserviert.)
Vgl.: MTOP, LEN

GET Operator

Programm-Modus

GET holt ein Zeichen aus dem Puffer der seriellen Schnittstelle, in der Regel vom Terminal. Ist kein Zeichen im Puffer liefert GET den Wert Null. Der Operator wartet nicht auf die Eingabe, sondern macht eine Momentaufnahme. Dadurch ist GET nur im Programm-Modus sinnvoll. Nach einer Zuweisung (z.B. a = GET) ist der Puffer ausgelesen und der Operator gelöscht. Eine neue Zuweisung liefert einen neuen Wert.

Vgl.: INPUT

GOSUB Statement

Syntax: GOSUB [ln num]
...
RETURN

Programm-Modus

Nach der Ausführung von GOSUB setzt der Interpreter den Ablauf direkt an der durch [ln num] gegebenen Adresse fort. Zusätzlich speichert der Interpreter die Adresse des auf GOSUB folgenden Statements auf dem Control-Stack. Das durch [ln num] gegebene Unterprogramm wird nun abgearbeitet und mit der Anweisung RETURN gelangt man in das Hauptprogramm zurück. Dabei wird der Control-Stack gelesen und zurückgesetzt. Nicht abgeschlossene FOR-NEXT-Schleifen in einem Unterprogramm erzeugen keinen Fehler, wenn nach der Schleife RETURN folgt. Der Befehl wirkt offenbar ähnlich einem lokalen CLEARS. GOSUB-RETURN Befehle können auch verschachtelt werden, so daß ein Unterprogramm das Nächste aufruft. Ordnet man die Unterprogramme hinter dem Hauptprogramm an, so wird das Listing lesbarer. Das logische Programmende muß dann mit END gekennzeichnet sein.

Vgl.: END, RETI

GOTO Statement

Syntax: GOTO [ln num]

Kommando- und Programm-Modus

GOTO bewirkt einen unbedingten Sprung zu der angegebenen Zeilennummer [ln num]. Dieser Befehl kann bestenfalls dazu dienen, Programme unleserlich und wartungsunfreundlich zu gestalten (Spagetticode).

GOTO bewirkt im Kommando-Modus die Programmausführung ab der angegebenen Zeilennummer. Die Variablen werden dabei im Gegensatz zu RUN, nicht gleich Null gesetzt. Wurde das Programm vor dem Start mit GOTO verändert, löscht auch dieser Befehl die Variablen. Dies ist notwendig, da eine Veränderung auch den Variablenspeicher zerstören kann.

Vgl.: GOSUB, RUN

IDLE Statement

Syntax: IDLE

Programm-Modus

Der IDLE-Befehls hält die Ausführung des Programms an und läßt den Interpreter auf einen Interrupt warten. Die Unterbrechung kann zeitgesteuert (ONTIME) oder extern (ONEX1) eintreten. Die Interrupts müssen natürlich vorher freigegeben sein. Anderenfalls ist mit dem Rechner nicht mehr viel los, da er für immer den IDLE-Befehl ausführt und wartet. Mit dem Eintreffen einer Unterbrechung springt der Interpreter in das entsprechende Unterprogramm und setzt danach die Arbeit mit dem auf IDLE folgenden Befehl fort. Während der Warte~ zeit setzt der 8052 seinen Anschluß /DMACK (Port 1.6, Pin 7) und damit auch den EC-Bus-Anschluß /HALT (ST1, Pin 25c) auf logisch Null. Damit wird die angeschlossen Peripherie informiert, daß der Rechner wartet. Die Ausführung des IDLE-Befehls schließt auch durch eine Interruptverarbeitung in Assembler ab.

Vgl.: ONTIME, ONEX1

IE Operator

Kommando- und Programm-Modus

Der Operator verändert oder liest das Special-Function-Register IE des 8052. Das Register überwacht die Interrupt-Freigabe und wird durch CLOCK0, CLOCK1, ONEX1, CLEAR und CLEARI beeinflußt. Genauere Informationen entnehmen Sie bitte dem Handbuch: [1] INTEL, MCS Basic-52 oder einer Beschreibung der 8051-Prozessoren.

IF-THEN-ELSE Statement

Syntax: IF [rel expr] THEN [Anweisung] ELSE [Anweisung]
Programm-Modus
Das Programm führt abhängig von der logischen Aussage [rel expr] verschiedene Anweisungen aus. Ist die Aussage wahr, wird die Anweisung nach THEN abgearbeitet. Der Ablauf geht in der nächsten Zeile weiter. Ist sie falsch, so wird die Anweisung nach ELSE ausgeführt und das Programm in der folgenden Zeile fortgesetzt. Die Folge IF-THEN-ELSE muß immer in einer Zeile stehen. Soll bei einer falschen Aussage nichts geschehen, so darf ELSE [Anweisung] entfallen. Nach THEN und ELSE können auch mehrere durch Doppelpunkt getrennte Anweisungen stehen. Folgt auf THEN nur eine Anweisung, so kann THEN entfallen.
Bsp: IF a=0 PRINT a
Ist eine Anweisung ein unbedingter Sprungbefehl GOTO [ln num], so darf GOTO entfallen. In diesem Fall muß THEN und/oder ELSE stehenbleiben.
Bsp: IF a=0 THEN 100 ELSE 200

INPUT Statement

Syntax: INPUT "[Text]",[var 1],[var 2],..,[var n]
Programm-Modus
Input fordert eine oder mehrere Eingaben vom Terminal an. Es kann zuvor ein Text [Text] ausgegeben werden, um den Benutzer anzuweisen. Die Eingaben werden den nachfolgenden Variablen [var] zugeordnet. Sind mehrere Werte gefordert, so müssen sie bei der Eingabe durch Kommata getrennt werden. Läßt man das Komma vor der ersten Variable weg, so wird ein Fragezeichen ausgegeben. Gibt der Anwender nicht genügend- oder ungültige Daten ein, erfolgt die Fehlermeldung "Try again", ohne daß das Programm dadurch unterbricht. Mit INPUT können auch Strings eingelesen werden. Bei der Eingabe muß jeder String mit Return (CR) abschließen.
Vgl.: PRINT

INT([expr]) Operator

Kommando- und Programm-Modus
Bildet den Integerwert des nachfolgenden Ausdrucks.

IP Operator

Kommando- und Programm-Modus
Der Operator verändert oder liest das Special-Function-Register IP des 8052. Das Register überwacht die Interrupt-Prioritäten. Genauere Informationen entnehmen Sie bitte dem Handbuch: [1] INTEL, MCS Basic-52 oder einer Beschreibung der 8051-Prozessoren.

LD@ Statement

Syntax: LD@ [adr]
Kommando- und Programm-Modus
Der Befehl LD@ holt Fließkommawerte von beliebigen Stellen im RAM auf den Argument-Stack. Die zu lesende Adresse im Speicher steht hinter dem Befehl [adr]. Der Befehl LD@ ist das Gegenstück zu ST@ (siehe unten). LD@ eignet sich z.B. um Daten und Meßwerte aus einer Memory-card zu lesen. Da Fließkommazahlen im 8052-Basic sechs Byte lang sind, werden mit LD@ auch sechs Byte aus dem RAM gelesen. Das höchstwertige Byte findet sich unter der Adresse [adr], die folgenden Bytes unter [adr] - 1, [adr] - 2 usw.
Vgl.: ST@, PUSH, POP

LEN Operator

Kommando- und Programm-Modus
Die Systemvariable LEN gibt Auskunft über die Länge des aktuellen Programms. Der Wert kann sich auf das RAM oder ein Programm im EPROM beziehen. Der Variable LEN kann man keinen Wert zuweisen, sie ist nur zu lesen. Steht kein Programm im Speicher, so ist LEN = 1. Dies entspricht einem Byte zum Kennzeichnen des Programmendes.
Vgl.: MTOP, FREE

LET Statement

Syntax: LET [var] = [expr]
Kommando- und Programm-Modus
Dieser Befehl weist der Variablen [var] einen Wert [expr] zu, was aber auch ohne die Anweisung LET geschehen kann. Extrem anachronistisch.

LIST Command

Syntax: LIST (CR)
Kommando-Modus
Nach der Eingabe von LIST im Kommando-Modus gibt der Interpreter das jeweils aktive Programm (RAM oder ROMn) auf dem Terminal aus. Der Vorgang bricht mit der Eingabe von Control-C ab. Nach der Zeilennummer und vor und nach jedem Basic-Befehl werden bei der Ausgabe Leerzeichen eingefügt, die im eigentlichen Programm nicht enthalten sind. Dadurch ist das Listing etwas übersichtlicher. Wurde das Programm mit Einrückungen oder zusätzlichen Leerzeichen eingegeben, so hat dies leider keinen Effekt. Die Leerzeichen werden beim Tokenisieren (Aufbereiten des Codes zur Ablage im Speicher) leider entfernt, um Platz zu sparen.
Optionen:
LIST 22 (CR) zeigt den Code ab Zeilennummer 22 an,
LIST 100-120 (CR) dagegen von Zeile 100 bis Zeile 120.
Vgl.: RAM, ROM, LIST#, LIST@

LIST# Command

Syntax: LIST# (CR)
Kommando-Modus
Das Listing erscheint nicht auf dem Terminal, sondern wird über die zweite serielle Schnittstelle z.B. auf einen Drucker ausgegeben. Die Baudrate auf dieser Leitung legt man mit dem Befehl BAUD fest. Die Anweisung NULL (siehe unten) fügt Wartezyklen in die Ausgabe ein.
Optionen wie unter LIST.
Vgl.: BAUD, NULL, LIST

LIST@ Command

Syntax: LIST@ (CR)
Kommando-Modus
LIST@ leitet die Ausgabe auf eine durch den Benutzer definierte Schnittstelle um. Eine Maschinensprache-Routine im Assembler-EPROM muß die Ausgabe unterstützen. Denkbar sind z.B. Listings über zusätzliche serielle Schnittstellenkarten am EC-Bus oder auf einem LCD-Display. LIST@ lenkt die Ausgabe auf den gleichen Pfad, wie auch PRINT@ und PH0.@. Näheres zur Programmierung entnehmen Sie bitte dem Handbuch: [1] INTEL, MCS 52-Basic.
Optionen wie unter LIST.
Vgl.: PRINT@, PH0.@

LOG([expr]) Operator

Kommando- und Programm-Modus
Berechnet den natürlichen Logarithmus des Arguments, $\log_e$[expr] = ln [expr]. Das Ergebnis hat 7 signifikante Stellen. Nur positive Argumente [expr] sind zulässig.

MTOP Operator

Kommando- und Programm-Modus
Nach einem Reset bestimmt der 8052 die Größe seines Arbeitsspeichers oder übernimmt diesen Wert aus dem EPROM (siehe PROGx). Die Adresse der letzten verfügbaren Speicherzelle legt der Interpreter in der Variable MTOP ab. Beschreibt man MTOP mit einem kleineren Wert, so ist der Speicherplatz darüber nur in Assembler oder mit dem XBY-Befehl nutzbar. Das Basic benutzt fortan keinen Speicher oberhalb dieser Adresse. MTOP bezieht sich immer auf das RAM.
Vgl.: LEN, FREE, PROGx

NEW Command

Syntax: NEW (CR)
Kommando-Modus
NEW eingegeben im Kommando-Modus löscht das im Arbeitsspeicher (RAM) stehende Programm. Alle Variablen werden gleich Null gesetzt. Strings und Interrupts sind nach NEW gelöscht. Die Uhr im 8052, der Wert des Stackpionters und die Größe des Stringspeichers werden nicht verändert.
Vgl.: CLEAR, CLEARI, CLEARS, CLOCK0, CLOCK1

NEXT siehe FOR-TO-(STEP)-NEXT

NOT([expr]) Operator

Kommando- und Programm-Modus
Bildet das Einerkomplement der auf NOT folgenden Integer-Zahl. Bei Fließkommazahlen wird nur der ganzzahlige Teil beachtet. Das Einerkomplement entspricht in dualer Schreibweise der Inversion jedes einzelnen Bits, für dezimale Werte ist es die Differenz zu 65535.

NULL Command

Syntax: NULL [int] (CR)
Kommando-Modus
Das NULL-Kommando veranlaßt den Interpreter seine Ausgabe sowohl an das Terminal als auch an den Druckerport zu bremsen. Nach jeder gesendeten Zeile (Carriage return) gibt der 8052 eine Anzahl von NULL-Zeichen (00h, i.d.R. ohne Wirkung) aus. Die Anzahl der Nullen kann zwischen 0 (Defaultwert) und 255 betragen. Das Kommando NULL war wichtig für den Anschluß rein mechanischer Drucker. Da jedoch die meisten Terminals und seriellen Drucker heute einen Empfangspuffer besitzen, dürfte sich die Anwendung erübrigt haben. Während der Programmausführung kann die Anzahl der gesendeten Nullen durch Beschreiben des internen RAMs (Adresse 15h, DBY(15h)=[int]) verändert werden.
Vgl.: BAUD

ON-GOSUB Statement

Syntax: ON [expr] GOSUB [ln num 0] [ln num 1] [ln num 2]...
Programm-Modus
ON-GOSUB ermöglicht bedingte Verzweigungen zu Unterprogrammen. Der Wert des auf ON folgenden Ausdrucks [expr] bestimmt, zu welcher Zeilennummer das Programm verzweigt. Ist der Wert [expr] gleich Null, so wird das Unterprogramm in Zeilennummer [ln num 0] fortgesetzt, [expr] = 1 führt zu [ln num 1] usw. Die Zeilennummern können dabei natürlich beliebige Werte haben. Die Zahl [expr] darf nicht negativ oder größer als die Anzahl der Zeilennummern sein. Der Unterprogrammaufruf ist identisch mit GOSUB-RETURN (siehe oben).
Vgl.: GOSUB-RETURN

ON-GOTO Statement

Syntax: ON [expr] GOTO [ln num 0] [ln num 1] [ln num 2]...
Programm-Modus
ON-GOTO ermöglicht bedingte Verzweigungen. Der Wert des auf ON folgenden Ausdrucks [expr] bestimmt, zu welcher Zeilennummer das Programm springt. Ist der Wert [expr] gleich Null, so wird das Programm in Zeilennummer [ln num 0] fortgesetzt, [expr] = 1 führt zu [ln num 1] usw. Die Zeilennummern können dabei natürlich beliebige Werte haben. Die Zahl [expr] darf nicht negativ oder größer als die Anzahl der Zeilennummern sein. Der Sprungbefehl ist identisch mit GOTO (siehe oben).
Vgl.: GOTO

ONERR Statement

Syntax: ONERR [ln num]
Programm-Modus
Tritt ein Laufzeitfehler im Programm auf, so bricht der Interpreter die Ausführung ab und geht mit einer Fehlermeldung in den Kommando-Modus über. Mit ONERR unterbleibt der Abbruch bei arithmetischen Fehlern und das Programm verzweigt statt dessen an die Zeilennummer [ln num]. In der dort stehenden Routine sollte eine Fehlerbehandlung erfolgen. Die Art des Fehlers kann durch Auslesen der Adresse 101h im externen RAM (XBY 101h) ermittelt werden. Folgende Fehlercodes sind möglich:

Code 10- Division durch Null
20- Overflow
30- Underflow
40- Falsches Argument

Die ONERR-Anweisung wirkt erst nach der Ausführung, sollte also am Programmanfang stehen.

ONEX1 Statement

Syntax: ONEX1 [ln num]
Programm-Modus
Nach der Ausführung von ONEX1 reagiert der Interpreter auf einen Hardware-Interrupt am Prozessoreingang /INT1 (für gewöhnlich /INT am EC-Bus, Pin 21c). Geht /INT1 auf Null, so verzweigt der Programmablauf an die durch [ln num] gegebene Adresse. Die Verzweigung entspricht einem GOSUB, mit dem Befehl RETI (return interrupt) gelangt man wieder zurück in das Hauptprogramm. Solange RETI nicht erfolgt, sind alle weiteren Interrupts an /INT1 gesperrt. Der Interpreter reagiert erst dann auf einen Interrupt, wenn der letzte Befehl abgearbeitet wurde. Das kann bis zu einigen zehn Millisekunden dauern und die Interruptquelle muß den Pin /INT1 solange auf Null halten. Deshalb sind besser "kurze" Befehle zu verwenden, die relativ schnell abgearbeitet werden. Z.B. blockiert eine PWM-Anweisung den Interpreter u.U. für einige Sekunden und auch Ausgaben an das Terminal nehmen einige Zeit in Anspruch. Der ONTIME-Interrupt hat eine höhere Priorität und kann einen ONEX1-Interrupt unterbrechen. Dagegen kann ein ONTIME-Interrupt nicht durch ONEX1 unterbrochen werden. ONEX1 setzt die Bits 3 und 7 im Register IE.
Vgl.: ONTIME, GOSUB-RETURN, IDLE

ONTIME Statement

Syntax: ONTIME [expr], [ln num]
Programm-Modus
Nach der Ausführung von ONTIME verzweigt der Programmablauf zu einer bestimmten Zeit an die durch [ln num] gegebene Adresse. Der Sprung in das Unterprogramm erfolgt, wenn der Special-Function-Operator TIME den Wert [expr] erreicht oder überschritten hat. Zuvor muß also die interne Uhrenfunktion mit CLOCK1 gestartet werden. TIME kann Werte zwischen 0 und 65535,995 annehmen, wobei der ganzzahlige Teil die Zeit seit dem Start der Uhr in Sekunden angibt. TIME wird in 5 ms-Intervallen incrementiert, die ONTIME-Anweisung vergleicht jedoch nur den ganzzahligen Teil von [expr] mit dem ganzzahligen Teil von TIME. Die Verzweigung entspricht einem Unterprogrammaufruf mit GOSUB und muß mit RETI (return interrupt) abgeschlossen werden. Solange RETI nicht erfolgt ist, sind alle weiteren Interrupts gesperrt. Der Interpreter reagiert erst dann auf einen Interrupt, wenn der letzte Befehl abgearbeitet wurde. Das kann bis zu einigen zehn Millisekunden dauern. Deshalb sind besser "kurze" Befehle zu verwenden, die relativ schnell abgearbeitet werden. Z.B. blockiert eine PWM-Anweisung den Interpreter u.U. für einige Sekunden und auch Ausgaben an das Terminal nehmen einige Zeit in Anspruch. Der ONTIME-Interrupt hat eine höhere Priorität als ONEX1 und kann einen ONEX1-Interrupt unterbrechen.
Vgl.: ONEX1, IDLE, CLOCK1, CLOCK0, GOSUB-RETURN

PCON Operator

Kommando- und Programm-Modus
Der Operator verändert oder liest das Special-Function-Register PCON des 8052. Mit dem höchstwertigen Bit des Registers überwacht das Basic die Baudrate. Ist das Bit gesetzt, so verdoppelt sich die mit Timer/Counter 1 erzeugte Taktfrequenz. Die anderen Bits werden von dem Basic nicht benutzt. Genauere Informationen entnehmen Sie bitte dem Handbuch: [1] INTEL, MCS Basic-52 oder einer Beschreibung der 8051-Prozessoren.

PGM Statement

Syntax: PGM
Kommando- und Programm-Modus
PGM kann zur Laufzeit eines Programms das Basic-EPROM brennen. Damit ist es möglich, den Rechner z.B. als EPROM-Programmiergerät zu verwenden oder bestimmte Parameter eines Programms dauerhaft zu speichern. Im Gegensatz zu PROG speichert PGM nicht nur die tokenisierten Basic-Zeilen, sondern auch beliebige Daten. Leider sind zuvor einige Werte in den Registern des 8052 abzulegen, die den Rechner z.B. über die Herkunfts- und Zieladresse der Daten und über die Programmierzeiten für das EPROM informieren. Eine genaue Beschreibung, wie PGM zu benutzen ist, würde hier den Rahmen sprengen. Vollständige Informationen finden Sie im Handbuch: [1] INTEL, MCS Basic-52.

PH0. Statement

Syntax: PH0. [expr], [expr],..., [expr]
Kommando- und Programm-Modus
PH0. gibt Texte, Strings, Werte von Ausdrücken und Variablen wie PRINT auf dem Terminal aus. Jedoch zeigt dieser Befehl alle Zahlen im Bereich 0 bis 65535 in hexadezimaler ganzzahliger Form an. Nachkommastellen werden abgeschnitten. Hinter den Zahlen wird immer der Buchstabe "H" für hexadezimal ausgegeben. PH0. unterdrückt die führenden Nullen bei Werten bis 0FFh. Für allen anderen Zahlen erfolgt die Ausgabe wie unter PRINT beschrieben.
Vgl.: PRINT

PH0.# Statement

Syntax: PH0.# [expr], [expr],..., [expr]
Kommando- und Programm-Modus
Ausgabe wie bei PH0., nur an die serielle Druckerschnittstelle.
Vgl.: PH0., PRINT, PRINT#

PH0.@ Statement

Syntax: PH0.@ [expr], [expr],..., [expr]
Kommando- und Programm-Modus
Ausgabe wie bei PH0., nur an eine benutzerdefinierte Schnittstelle.
Vgl.: PH0., PRINT, PRINT@

PH1. Statement

Syntax: PH1. [expr], [expr],..., [expr]
Kommando- und Programm-Modus
PH1. gibt Texte, Strings, Werte von Ausdrücken und Variablen wie PRINT auf dem Terminal aus. Jedoch zeigt dieser Befehl alle Zahlen im Bereich 0 bis 65535 in hexadezimaler ganzzahliger Form an. Nachkommastellen werden abgeschnitten. Hinter den Zahlen wird immer der Buchstabe "H" für hexadezimal ausgegeben. Die hexadezimale Anzeige erfolgt im Gegensatz zu PH0. immer mit vier Stellen. Für allen anderen Zahlen erfolgt die Ausgabe wie unter PRINT beschrieben.
Vgl.: PRINT

PH1.# Statement

Syntax: PH1.# [expr], [expr],..., [expr]
Kommando- und Programm-Modus
Ausgabe wie bei PH1., nur an die serielle Druckerschnittstelle.
Vgl.: PH1., PRINT, PRINT#

PH1.@ Statement

Syntax: PH1.@ [expr], [expr],..., [expr]
Kommando- und Programm-Modus
Ausgabe wie bei PH1., nur an eine benutzerdefinierte Schnittstelle.
Vgl.: PH1., PRINT, PRINT@

PI Operator

Kommando- und Programm-Modus

Die Zahl Pi, eine Konstante. Pi ist im 8052-Basic mit dem Wert 3,1415926 gespeichert. Genauer wäre in der letzten Stelle eine 7. Für die Berechnung der trigonometrischen Funktionen benötigt der Interpreter sehr oft Pi/2. Der kleinste Rechenfehler entsteht, wenn die Beziehung Pi/2 + Pi/2 = Pi genau erfüllt ist, und dies kann nur mit einer geraden Zahl geschehen.

POP Statement

Syntax: POP [var], [var],...,[var]

Kommando- und Programm-Modus

POP ordnet den Variablen in der Reihenfolge von links nach rechts die auf dem Argument-Stack abgelegten Werte zu. Es können mehrere Variablen, getrennt durch Kommata, innerhalb einer Zeile übergeben werden. Der Stackpionter decrementiert jeweils um sechs Byte. Dies entspricht dem Speicherplatz für eine Fließkommavariable. Die Routine ermöglicht die Übernahme von Werten in Assembler- oder Basicunterprogramme. Den letzten auf den Stack "gepushten" Wert liest POP als erstes wieder zurück. Ist der Stack leer, folgt auf POP eine Fehlermeldung.

Meines Wissens ist POP die einzige Möglichkeit, von einer Basic-Befehlssatzerweiterung einen Wert zurück zu erhalten. (Wenn Sie einen anderen Weg wissen, bitte ich unbedingt um Nachricht!)

Vgl.: PUSH

PORT1 Operator

Kommando- und Programm-Modus

Der Operator verändert oder liest das Special-Function-Register PORT1 des 8052. Das Register entspricht direkt den Anschlüssen des Ports 1 (Pin 1 bis 8). Das Basic belegt die Pins teilweise schon mit besonderen Aufgaben, wie z.B. der EPROM-Programmiereinrichtung und der PWM-Funktion. Kann man darauf verzichten, so ist es gestattet, den Port für beliebige I/O-Operationen zu nutzen. Dazu sind die Anschlüsse beim 8052-ECB auf die Stiftleiste ST5 gelegt. Mit dem Jumper J12 läßt sich auch die Programmiereinrichtung abschalten. Bei der Programmierung ist zu beachten, das ein Bit nur als Eingangsleitung verwendbar ist, wenn zuvor eine Eins auf die Adresse geschrieben wurde. Anderenfalls wird der Anschluß auf Null geklemmt. Genauere Informationen entnehmen Sie bitte dem Handbuch: [1] INTEL, MCS Basic-52 oder einer Beschreibung der 8051-Prozessoren.

PRINT, P., ? Statement

Syntax: PRINT [expr], [expr],..., [expr]

Kommando- und Programm-Modus

PRINT gibt Texte, Strings, Werte von Ausdrücken und Variablen auf dem Terminal aus. Die Befehle PRINT, P. und ? haben identische Funktionen. Genauso wie im Terminalbetrieb ist die Ausgabe von Umlauten und ß nicht möglich. Innerhalb einer PRINT-Zeile sind mehrere Ausgaben gestattet, wenn man die Variablen und/oder Strings [expr] durch Kommata trennt. Auf dem Terminal werden dabei jeweils zwei Leerzeichen zwischen den Werten eingefügt. Texte sind in Anführungsstrichen einzufassen, Strings dagegen nicht. Innerhalb der Anführungsstriche darf auch ein Doppelpunkt stehen.

Am Ende einer Ausgabe mit PRINT erfolgt ein Zeilenvorschub. Der Interpreter fügt selbstständig (CR) und (LF) (0Dh, 0Ah) ein. Eine PRINT-Anweisung ohne Argumente erzeugt folglich nur einen Zeilenvorschub.

Optionen:

Der Zeilenvorschub kann durch Anfügen eines Kommas an die Liste der Variablen unterbunden werden. Der Cursor steht dann am Ende der Zeile hinter dem zuletzt ausgegebenen Wert. Die nächste PRINT-Anweisung schreibt in der gleichen Zeile weiter.

"CR" setzt den Cursor an den Anfang der Zeile zurück, erzeugt aber keinen Zeilenvorschub. Mit Komma und "CR" am Ende der PRINT-Anweisung steht der Cursor nach der Ausgabe in der gleichen Zeile am linken Rand. Die nächste Ausgabe überschreibt die Vorherige.

Beispiel: PRINT "Es ist :",std,min,sec,CR

zeigt an: Es ist : 12 52 35 (Der Cursor steht unter "E".)

TAB ([n]) setzt die nachfolgende Ausgabe in die durch [n] spezifizierte Spalte. Ist der Cursor bereits auf oder rechts neben der gewünschten Position, so wird TAB ignoriert.

SPC ([n]) fügt zwischen zwei Ausgaben [n] Leerzeichen ein. SPC (0) führt zu zwei Leerzeichen, SPC(1) zu drei Zwischenräumen usw., da der Interpreter ohnehin immer zwei Spaces einfügt.

Die Option USING formatiert die Ausgabe von Zahlenwerten. Das durch USING angegebene Format wird gespeichert, so daß es für alle nachfolgenden PRINT-Anweisungen gilt. USING muß also nur dann in

PRINT-Zeilen auftauchen, wenn das Format geändert werden soll. USING kann durch U. abgekürzt werden. Zwischen USING und der nachfolgenden Klammer darf kein Leerzeichen stehen.
USING(#.#) bewirkt Ausgaben mit fester Anzahl von Vor- und Nachkommastellen. Die Anzahl der #-Zeichen vor dem Punkt bestimmt die Anzahl der Stellen für den ganzzahligen Teil, die Anzahl #-Zeichen nach dem Punkt die Zahl der Nachkommastellen. Ist nur die Angabe des Integerwertes gefordert, so kann der Punkt entfallen. Das Basic zeigt maximal acht Stellen an, entsprechend maximal acht Kreuzen in der Klammer. Paßt das gewählte Format nicht auf die Zahl, so wird ein Fragezeichen vor die Ausgabe gestellt und die Zahl im Basic-eigenen Format (USING(0) siehe unten) angegeben.
Beispiel: PRINT USING (###.##),22
zeigt an: 022.00
USING(Fn) erzeugt eine Ausgabe im Fließkommaformat mit Angabe des Zehnerexponenten. Die Zahl n legt die Anzahl der signifikanten Stellen fest. Weniger als drei Stellen sind nicht möglich, so das (F1), (F2) und (F3) gleiche Wirkung zeigen. USING(F0) gibt ebenfalls die Werte im Fließkommaformat mit Exponenten aus, unterdrückt aber die nachfolgenden Nullen.
Beispiel: PRINT USING (F4),1
zeigt an: 1.000 E 0
USING(0) ist das Standardformat nach einem Reset. Der Interpreter legt dabei das Ausgabeformat und die Anzahl der Stellen selbständig fest. Liegt der Zahlenwert zwischen 99 999 999 und 0,1 so erscheint die Zahl als Fließkommawert, außerhalb dieser Grenzen wird der Zehnerexponent mit angegeben.

PRINT#, P.#, ?# Statement
Syntax: PRINT# [expr], [expr],..., [expr]
Kommando- und Programm-Modus
PRINT# gibt Texte, Strings, Werte von Ausdrücken und Variablen auf den Druckerport, die zweite serielle Schnittstelle, aus. Die Befehle PRINT#, P.# und ?# haben identische Funktionen. Die Ausgabe funktioniert wie unter PRINT beschrieben (siehe oben). Zuvor ist die Übertragungsgeschwindigkeit mit BAUD einzustellen.
Vgl.: BAUD, NULL, LIST

PRINT@, P.@, ?@ Statement
Syntax: PRINT@ [expr], [expr],..., [expr]
Kommando- und Programm-Modus
PRINT@ leitet die Ausgabe auf eine durch den Benutzer definierte Schnittstelle um. Die Ausgabe muß durch eine Maschinensprache-Routine im Assembler-EPROM unterstützt werden. Denkbar sind z.B. Ausgaben über zusätzliche serielle Schnittstellenkarten am EC-Bus oder auf einem LCD-Display. PRINT@ lenkt die Ausgabe auf den gleichen Pfad wie auch LIST@ und PH0.@. Näheres zur Programmierung entnehmen Sie bitte dem Handbuch: [1] INTEL, MCS 52-Basic.
Optionen wie unter PRINT aufgeführt.
Vgl.: LIST@, PH0.@

PROG Command
Syntax: PROG (CR)
Kommando-Modus
Der Befehl programmiert das aktive Programm ins Basic-EPROM. Dabei ist es egal ob gerade ein Programm im RAM oder im EPROM selektiert ist. Letzteres ist allerdings sinnlos und füllt höchstens den Speicher auf. Vorraussetzung zum Brennen der EPROMs ist natürlich, daß die entsprechenden Jumper für die Programmierung gesetzt sind und 12,5V oder 21V Programmierspannung (je nach EPROM-Typ) an der Platine anliegen. Nach dem Vorgang erscheint die laufende Nummer des Programms auf dem Terminal. Ihr Rechner ist nicht abgestürzt, das Brennen kann, je nach Länge des Programms, eine ganze Zeit dauern. Vgl.: RAM, ROM, XFER

PROG1 Command
Syntax: PROG1 (CR)
Kommando-Modus
Mit PROG1 wird die aktuelle Baudrate im EPROM verewigt. Damit entfällt die automatische Baudratenerkennung und das Eingeben eines Leerzeichens nach dem Reset. Die Startmeldung (*MCS-52 BASIC...) wird sofort ausgegeben. Die Vorraussetzungen wie unter PROG zum Brennen des EPROMs, müssen natürlich gegeben sein. Soll die Baudrate variiert werden, so ist zuvor das EPROM zu löschen.

PROG2 Command

Syntax: PROG2 (CR)
Kommando-Modus
Wie PROG1, nur daß der Interpreter nach einem Reset sofort das erste Programm im EPROM ausführt. Der Rechner kommt also gar nicht erst in den Kommando-Modus. Damit ist der Einsatz bei Anwendungen ohne Terminal möglich.
Vgl.: PROG1

PROG3 Command

Syntax: PROG3 (CR)
Kommando-Modus
Wie bei PROG1 wird auch hier die aktuelle Baudrate und zusätzlich der Wert für MTOP (Speichergrenze) im EPROM festgehalten. Der Interpreter löscht nach einem Reset oder dem Anlegen der Betriebsspannung den Arbeitsspeicher bis zu der in MTOP angegebenen Adresse. Für gewöhnlich wird der Wert MTOP vorher getestet, das Ergebnis ist 7FFFh bei einem 32kByte RAM. Setzt man den Wert auf z.B. 3FFFh herab und führt danach den PROG3 Befehl aus, so wird der Bereich von 4000h bis 7FFFh nicht mehr gelöscht. Da das RAM batteriegepuffert ist, können in diesem Teil des Arbeitsspeichers (im Beispiel 16kByte) Meßwerte o.ä. dauerhaft abgelegt werden.
Vgl.: MTOP, PROG1

PROG4 Command

Syntax: PROG4 (CR)
Kommando-Modus
PROG4 ist die Kombination aus PROG2 und PROG3, verhindert also das Löschen des RAMs oberhalb von MTOP und startet das erste Programm im EPROM.
Vgl.: MTOP, PROG2, PROG3

PROG5 Command

Syntax: PROG5 (CR)
Kommando-Modus
PROG5 sichert ebenfalls die Baudrate und der Wert für MTOP im EPROM gesichert, wie unter PROG3 beschrieben.
Nach dem Anlegen der Betriebsspannung oder nach einem Reset, testet der Interpreter die Adresse 5Fh im externen Datenspeicher (RAM). Hat man hier vorher den Wert 0A5h abgelegt, so ist der gesammte Speicher geschützt. Damit lassen sich nicht nur Daten oberhalb von MTOP, sondern auch Basicprogramme im RAM verwahren. Ein zu testendes Programm muß also nicht extra in das EPROM gebrannt werden, um den nächsten Reset zu überleben.
Zusätzlich kann man den Interpreter veranlassen, das Programm im Arbeitsspeicher auszuführen und nicht erst in den Kommando-Modus zu gelangen. Die Funktion ist also ähnlich wie unter PROG2 beschrieben, nur wird hier das Programm im RAM! gestartet. Diese Option ist aktiv, wenn man unter der Adresse 5Eh im RAM das Byte 34h ablegt. Dies nennt sich dann RUNTRAP-MODE von englisch trap = Falle.
Diese Funktion erweist sich häufig wirklich als Falle, da ein Programmabbruch oder -ende und damit der Einstieg in den Kommandomodus sofort ein erneutes Starten des Programms zur Folge hat. Ist beispielsweise gar kein Programm im RAM, so gibt der Interpreter ununterbrochen READY an das Terminal aus. Diesen Vorgang kann der entnervte User nur durch Löschen des RAMs oder zumindest der Speicherstelle 5Eh unterbrechen. Also muß man die Batterie abklemmen (5min warten) oder das RAM aus dem Sockel ziehen.
Ist man nicht sicher, ob das Programm fehlerfrei läuft, so sollte man eine Möglichkeit vorsehen, den Wert der Adresse 5Eh irgendwie zu überschreiben, z.B. nach der Abfrage eines Codewortes (siehe GET). So hat man eine reelle Chance, in den Kommando-Modus zu gelangen. Der eigentliche Sinn ist, in einer autarken Anwendung nach dem Einschalten oder einem Reset, sofort und ohne Terminal das Programm zu starten.
Vgl.: PROG2, MTOP, XBY, GET

PROG6 Command

Syntax: PROG6 (CR)
Kommando-Modus
Nach der Ausführung geschieht zunächst das Gleiche wie bei PROG5, nur daß der Interpreter immer nach dem Start ein Maschinenspracheprogramm ausführt. Dies entspricht in der Wirkung einem CALL-Befehl. Die Startadresse der Assemblerroutine ist 4039h im externen Programmspeicher (Assembler-EPROM IC14). Steht hier kein Programm, so wird dies einen Systemabsturz zur Folge haben, also bitte nicht einfach erproben. Außerdem sollte man wie oben beschrieben den Speicherschutz für den Arbeitsspeicher aktivieren.
Damit ist die Möglichkeit vorhanden, eigene Resetroutinen zu kreieren. Folgende Optionen sind vorhanden:

1. Wenn das Carry-Bit nach dem Verlassen der neuen Resetroutine zurückgesetzt ist, startet die automatische Baudratenerkennung. Vom Terminal sollte der Interpreter ein Leerzeichen erhalten. Er gibt dann die Startmeldung aus.
2. Ist das Carry-Bit gesetzt und der Inhalt des Akku-Bit-0 (ACC.0) gleich Null, so wird die Baudrate aus dem EPROM eingestellt. Der Interpreter meldet sich mit der Startmeldung.
3. Ist das Carry-Bit gesetzt und der Inhalt des Akku-Bit-0 gleich Eins, so startet nach der Resetroutine automatisch das erste Programm im Basic-EPROM.

Vgl.: PROG5, MTOP

PUSH Statement

Syntax: PUSH [expr], [expr],..., [expr]
Kommando- und Programm-Modus
Der auf Push folgende arithmetische Ausdruck oder die Konstante wird auf den "Argument Stack" abgelegt. Es können mehrere Variablen, getrennt durch Kommata, gleichzeitig übergeben werden. Der Stackpionter incrementiert jeweils um sechs Byte. Dies entspricht dem Speicherplatz für eine Fließkommavariable. Der letzte auf den Stack "gepushte" Wert, wird mit POP als erstes wieder gelesen. Der Befehl kann z.B. Werte an Assembler- oder Basicunterprogramme übergeben. Es ist auch möglich die Werte zweier Variablen zu vertauschen, wie das Beispiel zeigt:

```
Beispiel:   10 a=1: b=2
            20 PUSH a,b
            30 POP a,b
            40 PRINT "a=",a,"b=",b
zeigt  :    a=  2  b=  1
```

Vgl.: POP

PWM Statement

Syntax: PWM [n on], [n off], [n per]
Kommando- und Programm-Modus
PWM ist die Abkürzung für pulse width modulation, Pulsbreitenmodulation. Der Befehl erzeugt eine Rechteckschwingung am Prozessorport P 1.2 und an der Stiftleiste ST6. Ein angeschlossener piezokeramischer Schallwandler (SBD) gibt dann einen Ton von sich. Die Frequenz leitet sich aus der Taktfrequenz des Prozessors (11,0592 MHz) ab. Ein Maschinenzyklus dauert zwölf Takte, entsprechend einer Frequenz von f = 921,6 kHz und T = 1,085 us. Der Ausdruck [n on] gibt die Anzahl der Maschinenzyklen an, während der das Signal logisch Eins ist, der Ausdruck [n off] bestimmt die Zyklenzahl für Null-Pegel. Die dritte Zahl [n per] bestimmt die Anzahl der Perioden. Werte von 0 bis 0FFFFh sind möglich, [n on] und [n off] müssen mindestens 25 sein. Der Befehl PWM 461, 461, 500 erzeugt einen Ton von 1 kHz, symmetrisches Rechteck, für eine Dauer von 500 Perioden entsprechend 0,5 s. In der Ausführungszeit kann der Prozessor nichts anderes bearbeiten, also auch nicht auf Interrupts reagieren!

RAM Command

Syntax: RAM (CR)
Kommando-Modus
Im Arbeitsspeicher und im EPROM dürfen verschiedene Programme stehen. Der Befehl RAM schaltet den Interpreter auf den Schreib-, Lesespeicher. Das dort stehende Programm kann editiert, gelistet und gestartet werden. Die Startadresse für Basicprogramme ist 200h. Im RAM kann nur ein Programm stehen.
Vgl.: ROM, XFER

RCAP2 Operator

Kommando- und Programm-Modus

RCAP2 liest oder beschreibt die Special-Function-Register RCAP2H und RCAP2L. Beide Register zu einem 16 Bit Operator zusammengefaßt. RCAP2 bestimmt die Baudrate für die serielle Schnittstelle. Genauere Informationen entnehmen Sie bitte dem Handbuch: [1] INTEL, MCS Basic-52 oder einer Beschreibung der 8051-
Prozessoren.

READ siehe DATA

REM Statement

Syntax: REM Text

Kommando- und Programm-Modus

REM gestattet dem Programmierer nachfolgend Kommentare (Text) anzugeben. Alles was hinter REM steht hat keinen Einfluß auf die Programmausführung. Ein Doppelpunkt innerhalb einer REM-Zeile ist wirkungslos. Mit Kommentaren sollte man nicht sparen, die Programme sind besser lesbar. Erstellt man die Programme mit einem Editor und nicht direkt im Terminalbetrieb, so kann man die Zeilennummern vor REM auch weglassen. Die Zeilen werden beim Download zwar gesendet, aber vom Interpreter ignoriert, da sie keine Nummer haben. So spart man erheblich Speicherplatz im RAM. Die Zeilen tauchen beim Listen nicht mehr auf. Dies empfielt sich natürlich nur, wenn der Quelltext immer mit einem Editor erstellt wird.

RESTORE siehe DATA

RETI Statement

Programm-Modus

RETI schließt Unterprogramme zur Interruptbearbeitung ab. Der Befehl hat die gleiche Wirkung wie RETURN, der Interpreter springt also in das Hauptprogramm zurück. RETI gibt aber auch die gesperrten Interrupts wieder frei. Siehe ONEX1 und ONTIME.

RETURN Statement

Programm-Modus

RETURN schließt ein durch GOSUB aufgerufenes Unterprogramm ab. Der Ablauf setzt mit dem auf GOSUB folgenden Befehl fort. Siehe GOSUB.

RND Operator

Kommando- und Programm-Modus

Der Befehl gibt eine Zufallszahl zwischen 0 und 1 zurück. Ein Ausdruck hinter RND ist im Gegensatz zu anderen Basic-Dialekten verboten. Die Zufallszahlen wiederholen sich erst nach 2^ 16 Möglichkeiten.

ROM Command

Syntax: ROM [int] (CR)

Kommando-Modus

Im Arbeitsspeicher und im EPROM können verschiedene Programme stehen. Der Befehl ROM [int] aktiviert das Programm Nr.: [int] im EPROM. Das Programm kann gelistet und gestartet werden, editieren ist nicht möglich. Es können mehrere Basicprogramme im ROM stehen. Wird keine Nummer angegeben ist automatisch das erste Programm aktiv. Eine zu hohe Programmnummer, z.B. ROM 6 (CR) bei nur vier gespeicherten Programmen, führt zur Fehlermeldung "ERROR: PROM MODE, ebenso der Versuch ein Programm im ROM zu editieren. Die Adresse für das erste Programm im EPROM ist 8010h.
Vgl.: ROM, XFER

RROM Statement

Syntax: RROM [int] (CR)
Kommando- und Programm-Modus
RROM startet die Ausführung des Programms im ROM mit der Nummer [int]. Dies kann sowohl im Kommando-Modus, als auch aus einem laufenden Programm heraus geschehen. Dabei werden alle Variablen und Strings gelöscht, so daß man keine Werte an das zu startende Programm übergeben kann. Der Aufruf eines nicht existenten Programms führt zu keiner Fehlermeldung. Leider ist der RROMBefehl eine Einbahnstraße. So kann aus einem Programm im RAM ein Programm im ROM gestartet werden, der umgekehrte Weg ist aber nicht möglich.
Vgl.: RUN, ROM, RAM

RUN Command

Syntax: RUN (CR)
Kommando-Modus
Nachdem der 8052 den Befehl RUN und anschließend CR empfängt, löscht er zunächst alle Variablenwerte. Der Interpreter wechselt dadurch vom Kommando- in den Programm-Modus und startet die Ausführung des Programms. RUN startet immer in der ersten Zeile, GOTO dagegen auch ab einer beliebigen Zeilennummer. Der RUN-Befehl ist sowohl auf Programme im Daten-, als auch im BasicEPROM-Speicher wirksam; jeweils das Programm, welches mit LIST angesehen werden kann. Control-C bricht die Ausführung des Programms ab.
Vgl. CONT, GOTO

SGN([expr]) Operator

Kommando- und Programm-Modus
Bestimmt das Vorzeichen der nachfolgenden Zahl. Ist die Zahl größer Null so ist das Ergebnis +1, kleiner Null führt zu -1 und ist der Wert Null, so gibt SGN Null zurück.

SIN([expr]) Operator

Kommando- und Programm-Modus
Berechnet den Sinus des Winkels. Das Argument [expr] wird im Bogenmaß angegeben und liegt im Bereich +/- 200000. Das Ergebnis wird durch eine Taylor-Reihe berechnet. Sie liefert 7 signifikante Stellen. Der Interpreter reduziert den Ausdruck [expr] zuvor auf Werte zwischen +/- Pi/2. Durch die dabei auftretenden Rundungsfehler ist es günstig das Arument so klein wie möglich zu halten.

SQR([expr]) Operator

Kommando- und Programm-Modus
Bestimmt die Quadratwurzel des nachfolgenden Ausdrucks. Der Wert von [expr] darf nicht negativ sein. (Wie, keine komplexe Rechnung?). Die Genauigkeit des Resultats ist +/- 5 Digit.

ST@ Statement

Syntax: ST@ [adr]
Kommando- und Programm-Modus
Der Befehl ST@ speichert Fließkommawerte an beliebigen Stellen im RAM. Der zu speichernde Wert muß vor der Operation auf dem Argument-Stack liegen und wird mit ST@ an die Adresse [adr] gebracht. Das Gegenstück zu ST@ ist der Befehl LD@ (siehe oben). ST@ eignet sich z.B. um Daten und Meßwerte in einer Memory-card abzulegen. Da Fließkommazahlen im 8052-Basic sechs Byte lang sind, werden mit ST@ auch sechs Byte im RAM gespeichert. Das höchstwertige Byte findet sich unter der Adresse [adr], die folgenden Bytes unter [adr] - 1, [adr] - 2 usw.
Vgl.: LD@, PUSH, POP

STEP siehe FOR-TO-(STEP)-NEXT

STOP Statement

Syntax: STOP
Programm-Modus
Bei Erreichen einer STOP-Anweisung unterbricht der Interpreter die Programmausführung. CONT setzt den Ablauf fort. STOP ermöglicht es dem Programmierer zwischendurch die Werte von Variablen zu überprüfen oder zu verändern. Das vereinfacht das "debugging" der Programme. Bei jedem STOP gibt der Interpreter eine Meldung mit der Zeilennummer aus. Die Nummer bezieht sich immer auf die nächste Programmzeile!
Vgl.: CONT

STRING Statement

Syntax: STRING [t spc], [s spc]
Kommando- und Programm-Modus
Mit der STRING-Anweisung reserviert der 8052 Speicherplatz für Zeichenketten. Dies muß unbedingt vor der ersten Definition eines Strings geschehen, da nach einem Reset kein Platz reserviert ist. Der zweite Ausdruck hinter STRING ([s spc]) gibt die Anzahl der Bytes pro Zeichenkette an, entsprechend der Anzahl der Buchstaben und Leerzeichen. Der erste Ausdruck [t spc] bestimmt den gesammten Speicherplatz. Dies ist nicht das Produkt aus Anzahl der Zeichenketten und deren Länge, da das Basic etwas Platz für die Verwaltung benötigt. Pro String ist ein zusätzliches Byte erforderlich und ein weiteres Byte unabhängig von der Anzahl der Zeichenketten. Der Speicher berechnet sich zu:
[t spc] = 1 + (Anzahl der Strings * ([s spc] + 1)).
Beachten Sie, daß die Nummerierung der Zeichenketten mit Null beginnt. $(0) ist eine gültige String-Variable. Weder NEW noch CLEAR geben den reservierten Speicher wieder frei, sondern nur die Anweisung STRING 0,0. Jede STRING-Anweisung löscht den gesammten Variablenspeicher, da sich Variablen und Strings hintereinander im RAM befinden. STRING hat also die gleiche Wirkung wie CLEAR. Es sollte deshalb eine der ersten Anweisungen im Programm sein und kein zweites Mal vorkommen.
Vgl.: ASC, CHR

T2CON Operator

Kommando- und Programm-Modus
Der Operator verändert oder liest das Special-Function-Register T2CON des 8052. Das Register steuert die Funktionsweise des Timer/Counter 2 und bestimmt, ob Timer 1 oder Timer 2 die Baudrate der seriellen Schnittstelle erzeugt. Genauere Informationen entnehmen Sie bitte dem Handbuch: [1] INTEL, MCS Basic-52 oder einer Beschreibung der 8051-Prozessoren.

TAN([expr]) Operator

Kommando- und Programm-Modus
Berechnet den Tangens des Winkels. Das Argument [expr] wird im Bogenmaß angegeben und liegt im Bereich +/- 200000. Das Ergebnis wird durch eine Taylor-Reihe berechnet. Sie liefert 7 signifikante Stellen. Der Interpreter reduziert den Ausdruck [expr] zuvor auf Werte zwischen +/- Pi/2. Durch die dabei auftretenden Rundungsfehler ist es günstig das Arument so klein wie möglich zu halten.

TCON Operator

Kommando- und Programm-Modus
Der Operator verändert oder liest das Special-Function-Register TCON des 8052. Das Register steuert die Timer0 und 1, sowie deren Interrupts. Genauere Informationen entnehmen Sie bitte dem Handbuch: [1] INTEL, MCS Basic-52 oder einer Beschreibung der 8051-Prozessoren.

TIME Operator

Kommando- und Programm-Modus
Der Operator oder das Register TIME enthält den Wert der Software-Uhr im 8052. Time läßt sich lesen und beschreiben. Mit CLOCK1 (siehe oben) startet die Uhr, CLOCK0 hält sie an. Beide Befehle beeinflussen aber nicht den vorhandenen Wert von TIME. Die Funktion ist mit Hilfe des internen Timer/-Counter 0 im 13 Bit-Modus realisiert. Der Timer inkrementiert alle 5 ms den Wert von TIME. Dabei ist vorrausgesetzt, daß die Variable XTAL den korrekten Wert der Quarzfrequenz enthält. TIME zählt von 0 bis 65535,995 Sekunden, der Überlauf setzt den Wert auf Null zurück. Nach einem Reset hat TIME den Wert Null und die Uhrenfunktion ist ausgeschaltet. Weist man TIME einen Wert zu, so ändert sich nur der Integeranteil. Möchte man die Uhr alle 12 Stunden (= 43200 s) zurücksetzen, so laufen die Sekunden-

bruchteile präzise weiter unabhängig von der Ausführungszeit der Befehle. So läßt sich auch in Basic eine genaue Uhr programmieren. Die Nachkommastellen lassen sich durch Beschreiben der Speicherstelle 47h im internen RAM (DBY 47h) verändern. Der dort stehende Wert multipliziert mit 5 ms repräsentiert die Sekundenbruchteile.
Vgl.: CLOCK1, CLOCK0

TIMER0 Operator

Kommando- und Programm-Modus
TIMER0 liest oder beschreibt die Register TH0 (high Byte) und TL0 (low Byte) des 8052. Der Timer 0 steuert in Basic die Echtzeituhr. Nur wenn man auf diese Funktion verzichtet, darf man Timer 0 mit anderen Aufgaben belegen. Genauere Informationen entnehmen Sie bitte dem Handbuch: [1] INTEL, MCS Basic-52 oder einer Beschreibung der 8051-Prozessoren.

TIMER1 Operator

Kommando- und Programm-Modus
TIMER1 liest oder beschreibt die Register TH1 (high Byte) und TL1 (low Byte) des 8052. Der Timer 1 steuert in Basic die serielle Druckerschnittstelle, die EPROM-Programmierung und die PWM-Anweisung. Nur wenn man auf diese Funktionen verzichtet, darf man Timer 1 mit anderen Aufgaben belegen. Genauere Informationen entnehmen Sie bitte dem Handbuch [1] INTEL, MCS Basic-52 oder einer Beschreibung der 8051-Prozessoren.

TIMER2 Operator

Kommando- und Programm-Modus
TIMER2 liest oder beschreibt die Register TH2 (high Byte) und TL2 (low Byte) des 8052. Der Timer 2 erzeugt in Basic die Baudrate der seriellen Schnittstelle. Nur wenn man auf diese Funktion verzichtet, darf man Timer 2 mit anderen Aufgaben belegen. Genauere Informationen entnehmen Sie bitte dem Handbuch [1] INTEL, MCS Basic-52 oder einer Beschreibung der 8051-Prozessoren.

TMOD Operator

Kommando- und Programm-Modus
Der Operator verändert oder liest das Special-Function-Register TMOD des 8052. Das Register steuert die Timer0 und 1. Genauere Informationen entnehmen Sie bitte dem Handbuch [1] INTEL, MCS Basic-52 oder einer Beschreibung der 8051-Prozessoren.

TO siehe FOR-TO-(STEP)-NEXT

UI0, UI1 Statement

Kommando- und Programm-Modus
Nach der Ausführung von UI1 bezieht der 8052 die Eingaben nicht mehr vom Terminal, sondern aus einer anderen Quelle. Dies gilt nicht nur für die Befehle GET oder INPUT, vielmehr wird auch im Kommando-Modus die neue Eingabemöglichkeit benutzt. Dazu muß nach festgelegten Regeln eine Maschinenspracheroutine die neue Tastatur oder Schnittstelle unterstützen. Mit dem Befehl UI0 schaltet der Interpreter in den normalen Betrieb, auf das Terminal, zurück. Innerhalb eines Programms sind die beiden Befehle mehrfach benutzbar und schalten beliebig zwischen den Quellen um. Denkbar wäre z.B. eine PC-Tastatur am Rechner zu betreiben und auf ein Terminal zu verzichten. Näheres zur Programmierung entnehmen Sie bitte dem Handbuch: [1] INTEL, MCS Basic-52.
Vgl.: UO0, UO1

UO0, UO1 Statement

Kommando- und Programm-Modus
Nach der Ausführung von UO1 sendet der 8052 die Ausgaben nicht mehr an das Terminal, sondern zu einem anderen Ausgabegerät. Dies gilt nicht nur für den PRINT-Befehl, vielmehr benutzt der Interpreter auch im Kommando-Modus die neue Ausgabemöglichkeit. Dazu muß nach festgelegten Regeln eine Maschinenspracheroutine das neue Anzeigegerät unterstützen. Mit dem Befehl UO0 schaltet der Interpreter in den normalen Betrieb, auf das Terminal, zurück. Die beiden Befehle kann man innerhalb eine

Programms mehrfach benutzen und beliebig zwischen den Ausgaben umschalten. Näheres zur Programmierung entnehmen Sie bitte dem Handbuch: [1] INTEL, MCS Basic-52.
Vgl.: UI0, UI1

XBY([expr]) Operator

Kommando- und Programm-Modus
Mit XBY wird ein Byte aus dem Datenspeicherbereich des 8052 gelesen oder geschrieben. Die Adresse ist durch [expr] angegeben, Integer-Werte zwischen 0 und 65535 (0FFFFh) sind zulässig. Zum Programmspeicher gehört das externe RAM (IC12)(0...7FFFh), das Basic-EPROM (IC13)(8000h...0BFFFh write only), die ECB-I/O und Speicherbereiche (0C000h...0F7FFh), Banking-FF und Watchdog (0FF80h) und die RTC (0FFC0h...0FFFFh). Alle Transferaktionen auf dem EC-Bus werden mit XBY gestaltet. Der Datenspeicher kann gelesen und beschrieben werden. XBY darf also links und rechts vom Zuweisungszeichen stehen.
Beispiel: a = XBY(0E000h)
XBY(0FF80h) = 0F0h
Vgl.: DBY, CBY

XFER Command

Syntax: XFER (CR)
Kommando-Modus
XFER kopiert das aktive Programm aus dem EPROM (z.B. ROM 2) in den Arbeitsspeicher und schaltet danach in den RAM-Modus um. Das Programm läßt sich danach editieren. Der Befehl funktioniert aber auch im RAM-Modus: Das Programm wird auf sich selbst kopiert, womit der Interpreter einige Zeit beschäftigt ist.
Vgl.: RAM, ROM

XTAL Operator

Syntax: XTAL = [expr]
Kommando- und Programm-Modus
Der Wert von XTAL informiert den Interpreter über die Taktfrquenz der Rechnerkarte. Die Zahl repräsentiert die Frequenz in Hertz. Alle zeitgesteuerten Vorgänge unter Basic sind von XTAL abhängig. Der Defaultwert ist 11059200 Hz. Verwendet man eine andere Quarzfrequenz, so ist im Programm an XTAL ein neuer Wert zuzuweisen.

Demoprogramme

Einstellen und ständiges Anzeigen der Uhrzeit auf dem Terminal:
Bis Zeile 350 wird in einer Abfrage die Uhrzeit und das Datum eingestellt. In den Zeilen 360 bis 460 wird in einer Endlosschleife Zeit und Datum ständig ausgegeben. Die Unterprogramme zum Setzen und Lesen von Zeit und Datum folgen ab Zeile 470.

```
10     REM  Initialisierung:
20     REM  Run-Trapp-Modus ausschalten
30    XBY(5EH)=00H
40     REM  Speichertest ausschalten
50    XBY(5FH)=0A5H
60     REM  Baudrate fuer Printerport = 9600 Bd
70     BAUD 9600
80     REM  interne Uhr einschalten
90     CLOCK 1
100    REM  obere Adressenbits des Busses auf Null setzen:
110   XBY(0FF80H)=00H
120    REM  Fertig    pieps, pieps
130    PRINT " Setup beendet " :  PRINT
140    PWM 400,400,200
150    PRINT " Uhrzeit und Datum neu stellen? "
160    PRINT " 0 = Nein, sonst doch "
170    INPUT A
180    IF A=0 GOTO 370
190    REM   Datum einstellen:
200    PRINT " Bitte aktuelles Datum eingeben! "
210    PRINT " Format : tt , mm , jj (CR)"
220    INPUT TAG,MNT,JAHR
230    PUSH TAG,MNT,JAHR
240    GOSUB 470
250    REM   Datum holen:
260    GOSUB 740
270    REM   und anzeigen:
280    POP TAG,MNT,JAHR
290    PRINT TAG,".",MNT,".19",JAHR
300    REM   Uhrzeit einstellen:
310    PRINT " Bitte aktuelle Uhrzeit eingeben! "
320    PRINT " Format : hh , mm , ss (CR)"
330    INPUT STD,MIN,SEC
340    PUSH SEC,MIN,STD
350    GOSUB 950

360    REM   Uhrzeit, Datum holen und anzeigen, endlos:
370    GOSUB 1300
380    GOSUB 740
390    REM   und anzeigen:
400    POP TAG,MNT,JAHR
410    POP SEC,MIN,STD
420    PRINT  CR ,STD,":",MIN,":",SEC,"  ",TAG,".",MNT,". 19",JAHR,
430    REM  warten:
440    FOR TM=0 TO 1000 :  NEXT TM
450    GOTO 370
460    END
```

```
470    REM  Unterprogramm zum Setzen des Datums
480    REM  Startadresse:
490   A=0FFC0H
500    REM  Uebernahme der Werte:
510    POP JAHR,MNT,TAG
520   B=INT(JAHR/10) : C=JAHR-10*B
530   D=INT(MNT/10) : E=MNT-10*D
540   G=INT(TAG/10) : H=TAG-10*G
550    REM  Sperren von Control-C:
560   DBY(26H)=DBY(26H).OR.01H
570    REM  Vorbereiten zum Schreiben:
580    REM  IRQ zuruecksetzen und HOLD = 1:
590   XBY(A+0DH)=5
591    REM Zeitschleife bis BUSY = 0
592   DO
593       BUSY=XBY(RTC+0DH).AND.2
594   UNTIL BUSY=0
600    REM  Datum stellen:
610   XBY(A+0BH)=B
620   XBY(A+0AH)=C
630   XBY(A+09H)=D
640   XBY(A+08H)=E
650   XBY(A+07H)=G
660   XBY(A+06H)=H
670    REM  Uhr disablen:
680    REM  HOLD = 0, IRQ = 1:
690   XBY(A+0DH)=4
700    REM  Sperre fuer Control-C wieder entfernen:
710   DBY(26H)=DBY(26H).AND.0FEH
720    REM  und zurueck:
730    RETURN


740    REM  Unterprogramm zum Lesen des Datums
750    REM  Startadresse:
760   A=0FFC0H
770    REM  Sperren von Control-C:
780   DBY(26H)=DBY(26H).OR.01H
790    REM  Vorbereiten zum Lesen:
800    REM  IRQ zuruecksetzen und HOLD = 1:
810   XBY(A+0DH)=5
811    REM Zeitschleife bis BUSY = 0
812   DO
813       BUSY=XBY(RTC+0DH).AND.2
814   UNTIL BUSY=0
820    REM  Auslesen:
830   JAHR=(0FH.AND.XBY(A+0BH))*10+(0FH.AND.XBY(A+0AH))
840   MNT=(0FH.AND.XBY(A+09H))*10+(0FH.AND.XBY(A+08H))
850   TAG=(0FH.AND.XBY(A+07H))*10+(0FH.AND.XBY(A+06H))
860    REM  Uhr disablen:
870    REM  HOLD = 0, IRQ = 1:
880   XBY(A+0DH)=4
890    REM  Sperre fuer Control-C wieder entfernen:
900   DBY(26H)=DBY(26H).AND.0FEH
910    REM  Uebergabe ans Hauptprogramm:
920    PUSH JAHR,MNT,TAG
930    REM  und zurueck:
940    RETURN
```

```
950    REM  Unterprogramm zum Setzen der Uhrzeit
960    REM  Startadresse:
970   A=0FFC0H
980    REM  Uebernahme der Werte:
990    POP STD,MIN,SEC
1000  B=INT(STD/10) : C=STD-10*B : B=03H.AND.B
1010  D=INT(MIN/10) : E=MIN-10*D
1020  G=INT(SEC/10) : H=SEC-10*G
1030   REM  Sperren von Control-C:
1040  DBY(26H)=DBY(26H).OR.01H
1050   REM  Vorbereiten zum Schreiben:
1060   REM  24-Stunden-Modus:
1070  XBY(A+0FH)=4
1080   REM  IRQ zuruecksetzen und HOLD = 1:
1090  XBY(A+0DH)=5
1091   REM Zeitschleife bis BUSY = 0
1092  DO
1093      BUSY=XBY(RTC+0DH).AND.2
1094  UNTIL BUSY=0
1100   REM  Standardimpuls-Periodendauer:
1110   REM  0  = 1/64 sec.
1120   REM  4  = 1 sec.    hier!
1130   REM  8  = 1 min.
1140   REM  12 = 1 std.
1150  XBY(A+0EH)=4
1160   REM  Uhrzeit stellen:
1170  XBY(A+05H)=B
1180  XBY(A+04H)=C
1190  XBY(A+03H)=D
1200  XBY(A+02H)=E
1210  XBY(A+01H)=G
1220  XBY(A+00H)=H
1230   REM  Uhr disablen
1240   REM  HOLD = 0, IRQ = 1:
1250  XBY(A+0DH)=4
1260   REM  Sperre fuer Control-C wieder entfernen:
1270  DBY(26H)=DBY(26H).AND.0FEH
1280   REM  und zurueck:
1290   RETURN


1300   REM  Unterprogramm zum Lesen der Uhrzeit
1310   REM  Startadresse:
1320  A=0FFC0H
1330   REM  Sperren von Control-C:
1340  DBY(26H)=DBY(26H).OR.01H
1350   REM  Vorbereiten zum Lesen:
1360   REM  IRQ zuruecksetzen und HOLD = 1:
1370  XBY(A+0DH)=5
1371   REM Zeitschleife bis BUSY = 0
1372  DO
1373      BUSY=XBY(RTC+0DH).AND.2
1374  UNTIL BUSY=0
1380   REM  Auslesen:
1390  STD=(03H.AND.XBY(A+05H))*10+(0FH.AND.XBY(A+04H))
1400  MIN=(0FH.AND.XBY(A+03H))*10+(0FH.AND.XBY(A+02H))
1410  SEC=(0FH.AND.XBY(A+01H))*10+(0FH.AND.XBY(A+00H))
1420   REM  Uhr disablen
1430   REM  HOLD = 0, IRQ = 1:
1440  XBY(A+0DH)=4
1450   REM  Sperre fuer Control-C wieder entfernen:
1460  DBY(26H)=DBY(26H).AND.0FEH
1470   REM  Uebergabe ans Hauptprogramm:
1480   PUSH STD,MIN,SEC
1490   REM  und zurueck:
1500   RETURN
```

Ausgabe der Uhrzeit auf LCD-Display:
Das Display wird zunächst initialisiert und ein Teststring wird ausgegeben. Danach wird ständig die Uhrzeit gelesen und angezeigt. Die Initialisierungsroutine beginnt in Zeile 520, das Unterprogramm zur Ausgabe eines Strings in Zeile 720 und die Uhrzeit wird ab Zeile 880 gelesen.

```
10     REM  Initialisierung:
20     GOSUB 520
30     REM  Platz fuer zehn Strings a 16 Zeichen:
40     STRING 171,16
50     REM  Strings:
60    $(0)=">Hello world !!<"
70    $(1)="0123456789abcdef"
80    $(2)="                "
90    $(3)="  Uhr   Minuten "
100   $(4)="und   Sekunden  "
110   $(5)="                "
120   $(6)="                "
130   $(7)="                "
140   $(8)="                "
150   $(9)="                "
160    REM  Ausgabe erster String, erste Zeile
170    PUSH 0,0
180    GOSUB 720
190    REM  Ausgabe zweiter String, zweite Zeile
200    PUSH 1,1
210    GOSUB 720
220    REM  warten:
230    FOR N=0 TO 1000 :  NEXT N
240    REM  Display loeschen umstaendlich:
250    PUSH 2,0
260    GOSUB 720
270    PUSH 2,1
280    GOSUB 720

290    REM  Endlosschleife:
300    DO
310    REM  Uhrzeit holen:
320    GOSUB 880
330    POP ESEC,ZSEC,EMIN,ZMIN,ESTD,ZSTD
340    REM  Bei Bedarf hier auf Terminal ausgeben:
350    REM  print zstd,estd,zmin,emin,zsec,esec
360    REM  Einfuegen in den String 3 und 4:
370   ASC($(3),1)=ZSTD.OR.30H
380   ASC($(3),2)=ESTD.OR.30H
390   ASC($(3),7)=ZMIN.OR.30H
400   ASC($(3),8)=EMIN.OR.30H
410   ASC($(4),5)=ZSEC.OR.30H
420   ASC($(4),6)=ESEC.OR.30H
430    REM  Ausgabe String drei in der ersten Zeile:
440    PUSH 3,0
450    GOSUB 720
460    REM  Ausgabe String vier in der zweiten Zeile:
470    PUSH 4,1
480    GOSUB 720
490    REM  ohne Abbruchbedingung:
500    UNTIL 0
510    END
```

```
520    REM  Unterprogramm zum Initialisieren des LCD-Displays:
530    REM  Fuer LM016  16Zeichen, zwei Zeilen.
540    REM  Das Befehlsregister hat die Adresse 0FF00h.
550    REM  Vor der Initialisierung ist eine Zeile hell, die
560    REM  Andere dunkel. So kann der Kontrast justiert werden.
570    REM
580    REM  Keine Uebergabeparameter notwendig, ohne Cursor, ohne
590    REM  blinken des Zeichens ueber dem Cursor, ohne scrollen.
600    REM  Display loeschen:
610   XBY(0FF00H)=01H
620    REM  Datenformat 8bit, zwei Zeilen, 5*7 Matrix:
630   XBY(0FF00H)=38H
640    REM  Cursor rechts, Anzeige steht:
650   XBY(0FF00H)=06H
660    REM  Anzeige an, Cursor aus, Blinken aus:
670   XBY(0FF00H)=0CH
680    REM  Startadresse erste Zeile:
690   XBY(0FF00H)=80H
700    REM zurueck
710    RETURN


720    REM  Unterprogramm zur LCD-Ausgabe eines Strings:
730    REM  Fuer LM016  16Zeichen-String
740    REM  Befehlsregister 0FF00h
750    REM  Datenregister   0FF02h
760    REM  Uebergabe der Stringnummer und der gewuenschten
770    REM  Ausgabezeile (0/1) auf dem Stack
780    REM  a=Zeilennr. (0=oben, 1=unten), b=Stringnr.:
790    POP A,B
800    REM  Einstellen der Zeilenstartadresse:
810    IF A=0 THEN XBY(0FF00H)=80H ELSE XBY(0FF00H)=0C0H
820    REM  Ausgabeschleife:
830    FOR N=1 TO 16
840   XBY(0FF02H)=ASC($(B),N)
850    NEXT N
860    REM  zurueck
870    RETURN


880    REM  Unterprogramm zum Lesen der Uhrzeit
890    REM  hier 4-Bitweise fuer Display o.ae.
900    REM  Startadresse:
910   A=0FFC0H
920    REM  Sperren von Control-C:
930   DBY(26H)=DBY(26H).OR.01H
940    REM  Vorbereiten zum Lesen:
950    REM  IRQ ruecksetzen und HOLD = 1:
960   XBY(A+0DH)=5
961    REM Zeitschleife bis BUSY = 0
962   DO
963       BUSY=XBY(RTC+0DH).AND.2
964   UNTIL BUSY=0
970    REM  Auslesen:
980   ZSTD=03H.AND.XBY(A+05H) : ESTD=0FH.AND.XBY(A+04H)
990   ZMIN=0FH.AND.XBY(A+03H) : EMIN=0FH.AND.XBY(A+02H)
1000  ZSEC=0FH.AND.XBY(A+01H) : ESEC=0FH.AND.XBY(A+00H)
1010   REM  Uhr disablen
1020   REM  HOLD = 0, IRQ = 1:
1030  XBY(A+0DH)=4
1040   REM  Sperre fuer Control-C wieder entfernen:
1050  DBY(26H)=DBY(26H).AND.0FEH
1060   REM  Uebergabe ans Hauptprogramm:
1070   PUSH ZSTD,ESTD,ZMIN,EMIN,ZSEC,ESEC
1080   REM  und zurueck:
1090   RETURN
```