

Systemhandbuch Zelle

Burkhard Kainka

Vorwort

Die Arbeit mit dem Computer ist in den allgemeinbildenden Schulen heute weit verbreitet. So wird z.B. im naturwissenschaftlichen Unterricht die Ausführung von Messungen von Computern unterstützt. Immer mehr wird dabei die Messung von Umweltdaten auch außerhalb der Schule interessant. Dazu sind geeignete Datenerfassungsgeräte erforderlich.

In einer Arbeitsgruppe am Landesinstitut für Schule und Weiterbildung des Landes Nordrhein-Westfalen in Soest wurde der Plan zum Bau eines autonomen Meß- und Steuersystems entwickelt. Es wurde "Zelle" genannt, weil es die intelligente Kernzelle in unterschiedlichsten Versuchen darstellt.

Die Zelle eignet sich einerseits als tragbares Datenerfassungsgerät für die unterschiedlichsten Aufgaben, andererseits aber auch als Element, das in einen größeren Aufbau als Steuerzelle eingesetzt werden kann.

Dieses Handbuch beschreibt die Hardware und Software der Zelle. Nach einer Einführung in die Grundlagen wird die Schaltung und das Betriebssystem der Zelle vorgestellt. Es folgt die Dokumentation der verschiedenen Software-Ebenen und schließlich Möglichkeiten der Hardware-Erweiterung.

Inhalt

| | |
|---|----|
| Vorwort..... | 3 |
| Inhalt | 5 |
| 1. Mikrocontroller-Grundlagen..... | 7 |
| 1.1 Die 8048-Familie | 7 |
| 1.2 Die Hardware des 8048-Mikrocontrollers | 8 |
| 1.3 Befehlsvorrat..... | 12 |
| 2. Die Hardware der Zelle | 19 |
| 2.1 Speichermodell | 21 |
| 2.2 Der A/D-Wandler..... | 23 |
| 2.3 I/O-Ports und weitere externe Anschlüsse..... | 25 |
| 3. Das Betriebssystem..... | 29 |
| 4. Ansteuerung der Zelle aus Hochsprachen | 33 |
| 5. Der SIMPEL-Compiler..... | 37 |
| 5.1 Die Simpel-Grundbefehle | 39 |
| 5.2 Prozeduren | 43 |
| 5.3 Erweiterte Kontrollstrukturen | 44 |
| 5.4 Spracherweiterungen..... | 49 |
| 5.5 Zugriff auf Systemparameter | 51 |
| 5.6 Wahlfreier Zugriff auf das RAM | 53 |
| 6. Hardware-Erweiterungen für die Zelle..... | 55 |
| 6.1 Der I/O-Bus..... | 56 |
| 6.2 Ansteuerung einer LCD-Anzeige..... | 57 |
| Anhang..... | 63 |
| Literatur..... | 63 |
| Platine, Bestückungsplan, Teileliste | 63 |
| 8048-Programmierskarte | 64 |
| Assemblerlisting des Zelle-Betriebssystems..... | 65 |
| PROGRAM Simpel_2; | 77 |

1. Mikrocontroller-Grundlagen

Ein Mikrocontroller ist ein Mikroprozessor, der zusammen mit ROM, RAM, I/O-Ports, Timern und eventuell noch weiteren Elementen auf einen Chip gebaut wurde. Damit lassen sich Ein-Chip-Computer für feste Aufgaben aufbauen, da alle wesentlichen Elemente eines Computers bereits vorhanden sind. Selbst wenn weitere externe Elemente wie z.B. zusätzliches RAM, A/D-Wandler usw. angeschlossen werden, ergibt sich für kleinere Steueraufgaben meist ein wesentlich geringerer Hardware-Aufwand als dies mit üblichen Mikroprozessoren (8085, Z80, 6502 usw.) möglich wäre.

Hier sollen nun die Mikrocontroller der 8048-Familie vorgestellt werden, um sie dann in der autonomen Meß- und Steuerzelle einzusetzen.

1.1 Die 8048-Familie

Der Einchip-Mikrocontroller 8048 wurde von der Industrie als universell einsetzbarer Steuercomputer für Konsumgüter entwickelt. Er findet sich heute in zahlreichen Geräten, von der Computertastatur bis zur Hifi-Anlage. In den meisten Fällen arbeitet der 8048 ohne jeden weiteren Baustein als selbständiger Computer. Dies ist möglich, da er bereits alle Elemente eines Rechners hat: Die CPU, RAM, ROM, Ein-/Ausgabeports und sogar einen Zähler bzw. Timer. Das ROM wird bei der Herstellung des Chips mit dem Programm des Anwenders fest programmiert. Diese Maskenprogrammierung bringt einen erheblichen Investitionsaufwand mit sich, so daß sie erst bei großen Stückzahlen angewandt werden kann. Für die Programmentwicklung und für kleine Serien wurden aber auch zum 8048 kompatible Bausteine (8748) entwickelt, die das interne ROM als EPROM enthalten und individuell programmiert und gelöscht werden können. Daneben gibt es auch noch eine ROM-lose Version (8035), die mit einem externen Programmspeicher arbeitet.

Der 8048 enthält einen Programmspeicher mit einer Größe von 1 K und ein RAM mit 64 Bytes. Weil sich dies für manche Anwendungen als zu wenig erwiesen hat, wurden der 8049 mit doppelter und der 8050 mit vierfacher RAM- und ROM-Größe gebaut. Auch hierzu sind ROM-lose und EPROM-Versionen erhältlich. Alle Versionen, auch die maskenprogrammierten, können mit externem Programmspeicher bis zu 4 K arbeiten. Deshalb ist auch ein fest programmierter 8048 oder 8049 für den Hobbyanwender nicht wertlos. Für eigene Versuche ist der Einstieg mit einem externen EPROM als Programmspeicher am wirtschaftlichsten. Ein einfaches System läßt sich leicht mit einem Bauteileaufwand unter 20 DM aufbauen. Wenn man bedenkt, daß damit die unterschiedlichsten Steueraufgaben lösbar sind, versteht man leicht die große Attraktivität dieser Mikrocontroller, die sich in zahlreichen Anwendungen widerspiegelt.

Die folgende Tabelle zeigt die verschiedenen Typen der 8048-Familie:

| RAM | maskenprogrammiert | programmierbar | ROM-los |
|----------|--------------------|-----------------|---------|
| 64 Byte | 8048, ROM 1 K | 8748, EPROM 1 K | 8035 |
| 128 Byte | 8049, ROM 2 K | 8749, EPROM 2 K | 8039 |
| 256 Byte | 8050, ROM 4 K | - | 8040 |

1.2 Die Hardware des 8048-Mikrocontrollers

Die Prozessoren der 8048-Familie besitzen einen 8-bit-Akku und zahlreiche Arbeitsregister. Weiterhin stehen 8-bit-breite I/O-Ports und ein 8-bit-Zähler/Timer zur Verfügung.

Ein interner Taktozillator bestimmt über einen angeschlossenen Quarz die Arbeitsgeschwindigkeit des Prozessors. Befehle werden mit einer Frequenz von einem fünfzehntel der Quarzfrequenz ausgeführt. Ein Ein-Byte-Befehl benötigt bei einem Quarz von 6 MHz immer 2,5 μ s, ein Befehl, der noch einen Parameter aus dem Programmspeicher benötigt, braucht die doppelte Zeit.

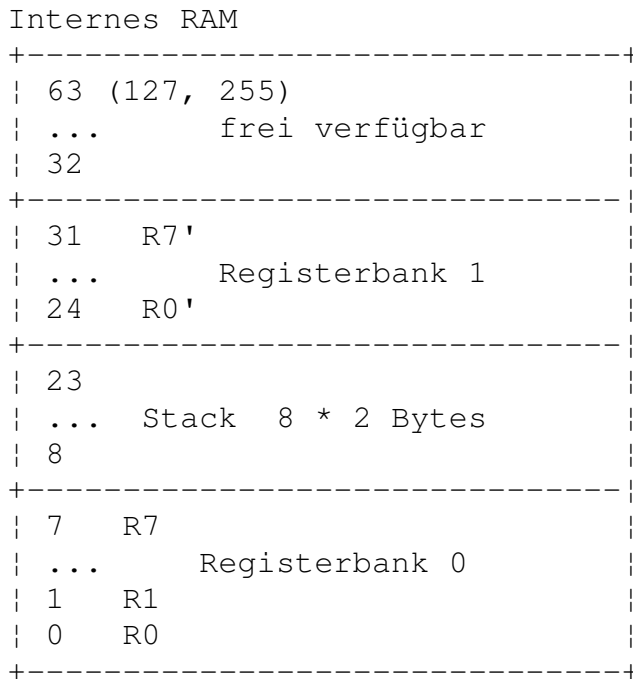
Der 8048 verwaltet getrennte Adressen für das interne RAM von 128 bis 256 Bytes, den Programmspeicher von 1024 bis 4096 Bytes und einen eventuell angeschlossenen externen Datenspeicher.

Dieselbe Adresse kann also gleich dreimal vorkommen. Insbesondere die Datenadressen des internen und externen RAMs müssen sorgfältig unterschieden werden. Auf sie wird mit unterschiedliche Befehlen zugegriffen.

| Programmspeicher | |
|------------------|--------------------------|
| 4096 | |
| ... | Speicherbank 1 |
| 2048 | |
| 2047 | |
| ... | Speicherbank 0 |
| 8 | |
| 7 | Timmer-Interruptvektor |
| 6 | |
| 5 | |
| 4 | |
| 3 | Interruptvektor |
| 2 | |
| 1 | |
| 0 | Programmstart nach Reset |

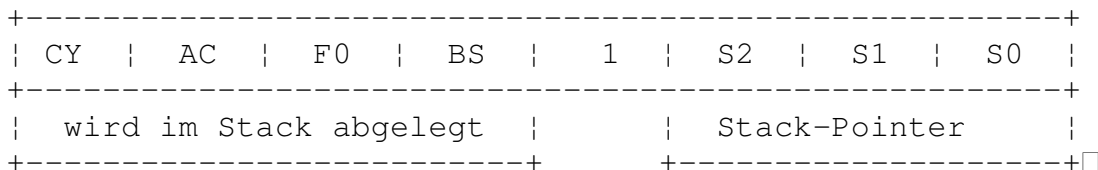
Der Programmspeicher kann im Prozessor (z.B. 8048, 8748) oder außerhalb (z.B. 8035) liegen. Da nur ein 12-bit-breiter Adreßzähler zur Verfügung steht, kann maximal 2K an einem Stück verwendet werden. Weitere 2 K stehen nach einer Bankumschaltung zur Verfügung. Prozessoren mit einem internen ROM von nur 1K greifen bei Adressen oberhalb 1023 zwangsweise auf den externen Speicher zurück, auch wenn die Leitung External Access (EA) low ist.

Ein externes RAM verwendet den selben Adreß- und Datenbus wie der externe Programmspeicher. Es wird jedoch mit den Steuerleitungen /WR und /RD angesprochen, während der externe Programmspeicher mit /PSEN aktiviert wird. Der Prozessor kann eigentlich nur 256 Adressen des externen RAMs verwalten. Wenn man jedoch die höherwertigen Adreßleitungen separat steuert, sind auch größere Speicher verwendbar.



Das interne RAM enthält die Arbeitsspeicher R0...R7 und R0'...R7', den Stack für die Verwaltung der Rücksprungadressen aus Unterprogrammen und zahlreiche Speicherplätze die über die Register R0 und R1 adressiert werden können.

Der allgemeine Status des Prozessors wird in einem besonderen Programm-Status-Register (PSW) festgehalten. Hier wird neben einigen Flags auch die aktuelle Stackadresse aufgehoben. Die vier oberen Bits des PSW werden selbst auf dem Stack abgelegt, wenn ein Unterprogramm aufgerufen wird. Sie werden jedoch nicht bei einem Rücksprung mit dem Return-Befehl RET, sondern nur nach einem Return-Restore (RETR) wiederhergestellt. Ein Aufruf benötigt jeweils 2 Bytes, da der Adreßzähler mit 12 Bits und das halbe PSW mit vier Bits zusammen gespeichert werden.



CY: Carry-Flag

AC: Auxiliary Carry

F0: Flag 0

BS: Registerbankwahl

Der Controller enthält einen 8-Bit-Aufwärtszähler, der sich als Timer oder als Zähler für externe Ereignisse einsetzen läßt. Der Zählerüberlauf kann über ein eigenes Flag abgefragt werden oder einen Interrupt auslösen, wobei das Programm bei Adresse 7 fortgeführt wird.

Als Timer erhält der Zähler Impulse von einem internen Vorteiler, der die Quarzfrequenz durch 480 teilt. Bei einer Quarzfrequenz von 6 MHz wird 12,5 kHz bzw. 80 µs gezählt.

Die Ports 1 und 2 können als Ein- und Ausgänge dienen. Sie werden als quasi-bidirektionale Ports bezeichnet. Nach dem System-Reset sind sie als Ausgänge hochgelegt, und können, da sie in diesem Zustand hochohmig sind, gleichzeitig als Eingänge verwendet werden. Dies ist möglich, weil die Portanschlüsse über Open-Drain-Anschlüsse aktiv nur nach Masse gezogen werden können und an jedem Anschluß Pull-Up-Widerstände von ca. 50 kOhm besitzen. Offene Eingänge werden daher als gesetzt erkannt. Es genügen einfache Schalter nach Masse, um den Zustand der Eingänge zu verändern. Wenn andererseits ein Ausgang benutzt werden soll, um ein low-Signal auszugeben, dann ist der entsprechende Portanschluß als Eingang ebenfalls low und sollte von außen nicht hochgesetzt werden, da der low-Zustand der Portanschlüsse niederohmig ist. Wird dies trotzdem versucht, treten Kurzschlußströme von ca. 30mA pro Portanschluß auf, die den Prozessor gefährden können. Sollen Eingänge verwendet werden, dann müssen also die entsprechenden Bits des Ports zuvor als Ausgänge hochgesetzt werden.

Die unteren vier Bits des Ports 2 nehmen eine Sonderstellung ein, wenn man mit externem Programmspeicher arbeitet. Dann werden nämlich hier die oberen Adreßbits für den Programmspeicher ausgegeben, allerdings nur während des Lesezyklus für den Programmspeicher. In der übrigen Zeit haben sie den Zustand der Port-2-Ausgänge. Benötigt man diese Bits als stehende Ausgänge, dann kann man die Daten zwischenspeichern, indem man sie mit der ansteigenden Flanke der ALE-Impulse in ein Datenlatch übernimmt. Außerdem können über diese Leitungen mehrere Erweiterungs-Portbausteine 8243 mit je 4 mal 4 Portanschlüssen betrieben werden.

Soll ein externer Programm- oder Datenspeicher verwendet werden, dann müssen dafür einige Portleitungen geopfert werden. Daten und Adressen werden nacheinander (gemultiplext) über die selben Anschlüsse geleitet. Dieses Verfahren spart einerseits Anschlüsse, andererseits ist neben dem ROM bzw. EPROM noch ein weiterer Baustein zur Zwischenspeicherung der Adressen (Adreßlatch) erforderlich. Die folgende Tabelle zeigt alle Anschlüsse des Controllers:

| Pin | Bezeichnung | Funktion |
|-----|-------------|--|
| 1 | T0 | Eingang zur Steuerung bedingter Sprungbefehle oder Ausgang für Clock-Signal |
| 2 | XTAL 1 | Anschluß für den Quarz (Eingang) |
| 3 | XTAL 2 | Anschluß für den Quarz (Ausgang) |
| 4 | RESET | Reset-Eingang |
| 5 | SS | Single-Step-Eingang |
| 6 | /INT | Interrupt-Eingang |
| 7 | EA | External Access. Wird an +5 V gelegt, um mit externem ROM zu arbeiten. |
| 8 | /RD | Read. Gibt Leseimpulse an ein externes RAM oder Peripheriebausteine aus. |
| 9 | /PSEN | Program Strobe Enable. Gibt Leseimpulse für den externen Programmspeicher aus. |
| 10 | /WR | Write. Gibt Schreibimpulse an ein externes RAM oder Peripheriebausteine aus. |
| 11 | ALE | Adress Latch Enable. Gibt Steuerimpulse zur Adressen-Zwischenspeicherung aus. |
| 12 | P01 | Bidirektionaler Port 0 oder |
| | bis | bidirektionaler Datenbus für externes |
| 19 | P07 | ROM, RAM und Peripherie. Überträgt auch die unteren 8 Bit der Adressen. |
| 20 | VSS | Masseanschluß |
| 21 | P20 | Die unteren vier Bits von Port 2. |
| | bis | Zugleich die oberen Adreßleitungen für |
| 24 | P23 | das externe ROM oder RAM. |
| 25 | PROG | Eingang für EPROM-Programmierimpuls und Strobe-Ausgang für 8243-Erweiterungsport |
| 26 | VDD | +5 V, beim Programmieren (8748) +21 V |
| 27 | P10 | Port 1. Alle Bits können als Ausgänge und als Eingänge frei verwendet werden. |
| | bis | |
| 34 | P17 | |
| 35 | P24 | Die oberen vier Bits von Port 2. Können beliebig als Ein- und Ausgänge verwendet werden. |
| | bis | |
| 38 | P27 | |
| 39 | T1 | Eingang für den internen Zähler und zur Steuerung bedingter Sprungbefehle |
| 40 | VCC | +5 V |

Der Eingang EA (Pin 7) entscheidet darüber, ob der Prozessor mit internem oder mit externem Programmspeicher arbeitet. Er muß bei den ROM-losen Versionen grundsätzlich an +5 V liegen. Die übrigen Versionen können wahlweise mit internem oder externem ROM arbeiten.

1.3 Befehlsvorrat

In diesem Kapitel werden die Befehle des 8048 vorgestellt. Sie finden sich übersichtlich noch einmal in der Programmierkarte im Anhang.

Alle Register und der Akku können verwendet werden, um Zwischenergebnisse zu speichern, Rechenoperationen auszuführen oder Zähler für allgemeine Aufgaben zu bilden. Eine Sonderstellung nehmen die Register R0 und R1 ein. Sie dienen unter anderem als 8-Bit-Adreßzeiger, über die sich Speicheradressen im internen oder externen RAM adressieren lassen.

Ähnlich den Registern können auch die Ports über direkte Befehle beschrieben und ausgelesen werden. Daneben gibt es einen 8-Bit-Zähler, der ebenfalls geladen und ausgelesen werden kann. Sein Takteingang kann entweder mit prozessoreigenen Taktimpulsen oder mit externen Zählimpulsen belegt werden. Speicher anderer Art sind die Flags. Hierbei handelt es sich um 1-Bit-Speicher, die bestimmte Zustände im Prozessor anzeigen, wie z.B. den Zählerüberlauf und den aufgetretenen Übertrag bei einer Rechnung.

Direkte Ladebefehle:

| Befehl | Mnemonic | Takte | Fuktion |
|---------|---------------|-------|-------------------------------|
| 27 | CLR A | 1 | A := 0 |
| 23 | MOV A, #data | 2 | A := data |
| B8...BF | MOV R, #data | 2 | R0...R7 := data |
| B0, B1 | MOV @R, #data | 2 | (R0,R1) := data, internes RAM |

Der Ausdruck #data steht hier für ein direkt im Programmspeicher stehendes Datenbyte. 23 FF lädt also z.B. den Akku mit dem Wert FF (dez. 255). Genauso lassen sich die acht Arbeitsregister direkt laden, wobei jedes Register einen eigenen Befehl hat. B9 FF lädt etwa das Register R1 mit FF.

Das direkte Laden eines Bytes in irgendeine Adresse des internen RAMs ist über die Befehle B0 und B1 möglich. Dabei wird die Speicheradresse indirekt über das Register 0 oder 1 adressiert, das also zuvor mit der Adresse geladen wurde. Die indirekte Adressierung wird in Assembler durch den "Klammeraffen" @ ausgedrückt, in der Funktionsbeschreibung durch Klammern. (R1) meint also die Speicherzelle, die durch R1 adressiert wird.

Datentransportbefehle:

| Befehl | Mnemonic | Takte | Fuktion |
|---------|-------------|-------|----------------------------|
| F8...FF | MOV A, R | 1 | A := R0...R7 |
| A8...AF | MOV R, A | 1 | R0...R7 := A |
| F0, F1 | MOV A, @R | 1 | A := (R0,R1), internes RAM |
| A0, A1 | MOV @R, A | 1 | (R0,R1) := A, internes RAM |
| 80, 81 | MOVX A, @R | 2 | A := (R0,R1), externes RAM |
| 90, 91 | MOVX @R, A | 2 | (R0,R1) := A, externes RAM |
| A3 | MOVP A, @A | 2 | A := (A), ROM |
| E3 | MOVP3 A, @A | 2 | A := (A), ROM, Seite 3 |
| D7 | MOV PSW, A | 1 | PSW := A |
| C7 | MOV A, PSW | 1 | A := PSW |

Der Datentransport läuft hier grundsätzlich über den Akku. Neben dem direkten Transport zwischen Akku und Registern ist auch das Laden und Lesen des internen RAMs möglich, wobei die Register 0 oder 1 wieder zur indirekten Adressierung verwendet wird. In gleicher Weise kann ein eventuell angeschlossenes externes RAM bzw. irgendwelche am Datenbus angeschlossene Peripherie angesprochen werden. Der Adreßumfang ist dabei nur jeweils eine Seite (256 Byte). Die Umschaltung der Seiten geschieht über das direkte Setzen der

Portleitungen P20 bis P23. Die Befehle MOVX A,@R und MOVX @R,A bilden den einzigen Zugang zu einem externen RAM oder angeschlossener Peripherie. In allen anderen Fällen adressieren die Register 0 und 1 das interne RAM des Prozessors.

Die Möglichkeit, das ROM indirekt über den Akku zu adressieren, ist besonders hilfreich beim Auslesen von Datenfeldern, die im ROM angelegt wurden. Dabei muß das Datenfeld bei Verwendung des Befehls MOVP in der gerade aktiven Seite stehen, bei MOVP3 grundsätzlich auf Seite 3. Das Prozessor-Statuswort PSW beinhaltet interne Informationen des Prozessors, die gelesen und verändert werden können.

Datenaustauschbefehle:

| Befehl | Mnemonic | Takte | Fuktion |
|---------|-----------|-------|--|
| 28...2F | XCH A,R | 1 | A < > (R0...R7) |
| 20, 21 | XCH A,@R | 1 | A < > (R0,R1), internes RAM |
| 30, 31 | XCHD A,@R | 1 | A(1...3) < > (R0,R1,1...3) internes RAM, je untere 4 Bits |
| 47 | SWAP A | 1 | A(1...3) < > A(4...7) |

Hier werden die Inhalte des Akkus und der Register bzw. des internen RAMs miteinander vertauscht. Der Befehl XCHD betrifft dabei nur die unteren vier Bits eines Bytes. SWAP tauscht die unteren und die oberen vier Bits im Akku.

Bitmanipulationen:

| Befehl | Mnemonic | Takte | Fuktion |
|---------|-------------|-------|-------------------------------|
| 58...5F | ANL A,R | 1 | A := A AND R0...R7 |
| 48...4F | ORL A,R | 1 | A := A OR R0...R7 |
| D8...DF | XRL A,R | 1 | A := A EXOR R0...R7 |
| 50,51 | ANL A,@R | 1 | A := A AND (R0,R1), int. RAM |
| 40,41 | ORL A,@R | 1 | A := A OR (R0,R1), int. RAM |
| D0,D1 | XRL A,@R | 1 | A := A EXOR (R0,R1), int. RAM |
| 53 | ANL A,#data | 2 | A := A AND data |
| 43 | ORL A,#data | 2 | A := A OR data |
| D3 | XRL A,#data | 2 | A := A EXOR data |
| 37 | CPL A | 1 | A := NOT A |

Alle diese logischen Operationen verknüpfen den Inhalt des Akkus mit einer zweiten Quelle, die ein Register, das indirekt adressierte interne RAM oder ein direkt geladenes Byte sein kann. CPL A invertiert den Akkuinhalt.

Additionsbefehle:

| Befehl | Mnemonic | Takte | Fuktion |
|---------|--------------|-------|--------------------------------|
| 68...6F | ADD A,R | 1 | A := A + R0...R7 |
| 78...7F | ADDC A,R | 1 | A := A + R0...R7 + C |
| 60,61 | ADD A,@R | 1 | A := A + (R0,R1), int. RAM |
| 70,71 | ADDC A,@R | 1 | A := A + (R0,R1) + C, int, RAM |
| 03 | ADD A,#data | 2 | A := A + data |
| 13 | ADDC A,#data | 2 | A := A + data + C |
| 57 | DA A | 1 | Dezimalabgleich |

Auch die Additionsbefehle benutzen grundsätzlich den Akku. Sie beeinflussen das Übertragsflag C, d.h. C wird gesetzt, wenn das Ergebnis einer Addition 255 übersteigt. Es kann mit Dezimalzahlen (BCD) gerechnet werden, wenn nach der Addition der Dezimalausgleich angewandt wird. Der Akku enthält dann in den unteren und oberen vier Bits je eine Dezimalziffer (0...9).

Incrementieren und Decrementieren:

| Befehl | Mnemonic | Takte | Fuktion |
|---------|----------|-------|---------------------------------|
| 18...1F | INC R | 1 | R0...R7 := R0...R7 + 1 |
| C8...CF | DEC R | 1 | R0...R7 := R0...R7 - 1 |
| 17 | INC A | 1 | A := A + 1 |
| 07 | DEC A | 1 | A := A - 1 |
| 10,11 | INC @R | 1 | (R0,R1) := (R0,R1) + 1, int.RAM |

Diese Befehle erlauben das Erhöhen und Erniedrigen von Bytes um Eins.

Ein- Ausgabebefehle

| Befehl | Mnemonic | Takte | Fuktion |
|--------|-------------|-------|-------------------------|
| 09,0A | IN A,P | 2 | A := P1,P2 |
| 39,3A | OUTL P,A | 2 | P1,P2 := A |
| 99,9A | ANL P,#data | 2 | P1,P2 := P1,P2 AND data |
| 89,8A | ORL P,data | 2 | P1,P2 := P1,P2 OR data |

Busbefehle nur ohne externen Speicher (nicht für 8035/39):

| | | | |
|----|---------------|---|-------------------|
| 08 | INS A,BUS | 2 | A := P0 |
| 02 | OUTL BUS,A | 2 | P0 := A |
| 98 | ANL BUS,#data | 2 | P0 := P0 AND data |
| 88 | ORL BUS,#data | 2 | P0 := P0 OR data |

Portbefehle für den Erweiterungsport 8243:

| | | | |
|---------|----------|---|--------------------------|
| 0C...0F | MOVD A,P | 2 | A := P4...P7 |
| 3C...3F | MOVD P,A | 2 | P4...P7 := A |
| 9C...9F | ANLD P,A | 2 | P4...P7 := P4...P7 AND A |
| 8C...8F | ORLD P,A | 2 | P4...P7 := P4...P7 OR A |

Die Ports 1 und 2 lassen sich mit den Daten des Akkus laden und auch vom Akku lesen. Außerdem lassen sie sich über ODER- und UND-Befehle beeinflussen. Auf diese Weise kann man einzelne Bits gezielt setzen oder zurücksetzen, ohne die anderen zu beeinflussen.

Die BUS-Befehle betreffen den Port 0, der allerdings in vielen Fällen für den Anschluß eines externen ROMs belegt ist. Sie sind deshalb nur für maskenprogrammierte oder EPROM-Mikrocontroller verwendbar.

Der Portbaustein 8243 enthält vier weitere 4-Bit-Ports (P4...P7). Alle entsprechenden Befehle arbeiten mit den unteren vier Bits des Akkus, wobei die oberen ohne Einfluß bleiben bzw. beim Lesebefehl MOVXD A,P auf Null gesetzt werden.

Zähler-Befehle:

| Befehl | Mnemonic | Takte | Fuktion |
|--------|-----------|-------|-----------------------------|
| 42 | MOV A, T | 1 | A := T |
| 62 | MOV T, A | 1 | T := A |
| 55 | STRT T | 1 | startet den Timer |
| 45 | STRT CNT | 1 | startet den Ereigniszähler |
| 65 | STOP TCNT | 1 | stoppt Timer/Ereigniszähler |
| 25 | EN TCNTI | 1 | erlaubt Zähler-Interrupt |
| 35 | DIS TCNTI | 1 | verbietet Zähler-Interrupt |

Der interne 8-Bit-Zähler kann entweder als Timer oder als Ereigniszähler gestartet werden. Das Zählerregister kann beliebig geladen oder ausgelesen werden. Beim Betrieb als Timer zählt der Zähler mit einem festen Takt von 1/32 des Befehlszyklus vom geladenen Startwert aufwärts. Ein Überlauf kann einen Interrupt auslösen, wenn EN TCNTI ausgeführt wurde. Der Prozessor unterbricht dann das gerade bearbeitete Programm und springt zur Speicheradresse 7. Dort steht im allgemeinen ein Sprungbefehl zu einer Interrupt-Serviceroutine. Eine andere Möglichkeit, den Überlauf auszuwerten, besteht in der Verwendung des bedingten Sprungbefehls JTF, der direkt das Überlauf-Flag des Zählers auswertet und wieder zurücksetzt.

In gleicher Weise läßt sich der Zähler als Ereigniszähler einsetzen. Dabei wird der Pin T1 als Eingang verwendet. Die Zählimpulse dürfen 1/3 der Befehlszyklus-Frequenz haben.

Flag-Befehle:

| Befehl | Mnemonic | Takte | Fuktion |
|--------|----------|-------|--------------|
| 97 | CLR C | 1 | C := 0 |
| A7 | CPL C | 1 | C := NOT C |
| 85 | CLR F0 | 1 | F0 := 0 |
| 95 | CPL F0 | 1 | F0 := NOT F0 |
| A5 | CLR F1 | 1 | F1 := 0 |
| B5 | CPL F1 | 1 | F1 := NOT F1 |

Neben dem Übertragsflag C gibt es die beiden Benutzerflags F1 und F2, die man im Programm beliebig zurücksetzen oder invertieren kann. Alle drei Flags können über bedingte Sprungbefehle abgefragt werden. Das Übertragsflag C wird auch bei den Additionsbefehlen weiterverwendet.

Verschiebefehle:

| Befehl | Mnemonic | Takte | Fuktion |
|--------|----------|-------|-----------------------------|
| E7 | RL A | 1 | Akku-Bits links schieben |
| 77 | RR A | 1 | Akku-Bits rechts schieben |
| F7 | RLC A | 1 | Bits links über C schieben |
| 67 | RRC A | 1 | Bits rechts über C schieben |

Mit diesen Befehlen läßt sich die Funktion eines Schieberegisters auf den Akku anwenden. Man kann die Verschieberichtung wählen und nach Bedarf über das Übertragsflag schieben. Damit lassen sich die einzelnen Bits eines Bytes im Akku nacheinander auswerten.

Sprungbefehle:

| Befehl | Mnemonic | Takte | Fuktion |
|--------|----------|-------|---------|
|--------|----------|-------|---------|

Sprungbefehle innerhalb der aktiven Seite (256 Byte):

| | | | | |
|---------|------|------|---|---|
| C6 | JZ | Adr. | 2 | Sprung, wenn A = 0 |
| 96 | JNZ | Adr. | 2 | Sprung, wenn A <> 0 |
| F6 | JC | Adr. | 2 | Sprung, wenn C = 1 |
| E6 | JNC | Adr. | 2 | Sprung, wenn C = 0 |
| 36 | JT0 | Adr. | 2 | Sprung, wenn Pin T0 = 1 |
| 26 | JNT0 | Adr. | 2 | Sprung, wenn Pin T0 = 0 |
| 56 | JT1 | Adr. | 2 | Sprung, wenn Pin T1 = 1 |
| 46 | JNT1 | Adr. | 2 | Sprung, wenn Pin T1 = 0 |
| B6 | JF0 | Adr. | 2 | Sprung, wenn F0 = 1 |
| 76 | JF1 | Adr. | 2 | Sprung, wenn F1 = 1 |
| 86 | JNI | Adr. | 2 | Sprung, wenn Pin INT = 0 |
| 16 | JTF | Adr. | 2 | Sprung, wenn Zähler-Überlauf |
| B3 | JMPP | @A | 2 | Sprung nach (A) |
| 12...F2 | JBb | Adr. | 2 | Sprung, wenn Akkubit 0...7 = 1 |
| E8...EF | DJNZ | Adr. | 2 | R0...R7 := R0...R7 - 1 und Sprung, wenn R0...R7 <> 0 |

Sprungbefehle innerhalb 8 Seiten (2 K):

| | | | | |
|---------|------|------|---|--------------------------------|
| 04...E4 | JMP | Adr. | 2 | Sprung zu Adr., Seite 0...7 |
| 14...F4 | Call | Adr. | 2 | Unterprogrammaufruf, S.0...7 |
| 83 | RET | | 2 | Rücksprung v. Unterprogramm |
| 93 | RETR | | 2 | Rücksprung v. Interruptroutine |

Die meisten Sprungbefehle erreichen nur Sprungziele innerhalb der gerade aktiven Seite des Programmspeichers. Es lassen sich viele unterschiedliche Bedingungen dafür angeben, ob der Sprung ausgeführt werden soll. Dabei können die Prozessoranschlüsse T1 und T2 sowie /INT direkt abgefragt werden. Ein Sprung kann auch an die Bedingung geknüpft werden, ob ein bestimmtes Bit im Akku gesetzt ist. Die DJNZ-Befehle erlauben den einfachen Aufbau von Zählschleifen, wobei jedes der Register R0 bis R7 als Schleifenzähler eingesetzt werden kann. Der Sprungbefehl JMPP @A erlaubt das Anspringen berechneter Sprungziele.

Die acht möglichen JMP-Befehle ermöglichen das Anspringen jedes Ziels innerhalb des Bereichs von 2 K, wobei für jede Seite ein eigener Befehl existiert. Das gleiche gilt für den CALL-Befehl zum Aufrufen von Unterprogrammen. Will man den vollen Adreßumfang von 4 K ausnutzen, dann muß vor einem Sprung in den jeweils anderen 2-K-Bereich (Speicherbank) der Befehl SEL MB1 oder SEL MB0 ausgeführt werden. Die Rücksprungbefehle RET und RETR arbeiten ebenfalls in einem Bereich von 2 K. RETR restauriert die oberen vier Bits des PSW und quittiert gleichzeitig einen Interrupt.

Prozessor-Steuerung:

| Befehl | Mnemonic | Takte | Fuktion |
|--------|----------|-------|-------------------------------|
| 05 | ENI | 1 | erlaubt Interrupt von Pin INT |
| 15 | DISI | 1 | verbietet Interrupt |
| C5 | SEL RBO | 1 | selektiert Registerbank 0 |
| D5 | SEL RB1 | 1 | selektiert Registerbank 1 |
| E5 | SEL MB0 | 1 | selektiert Speicherbank 0 |
| F5 | SEL MB1 | 1 | selektiert Speicherbank 1 |
| 75 | ENTO CLK | 1 | Taktausgabe an Pin T0 |
| 00 | NOP | 1 | Wartebefehl 1 Takt |

Ein Interrupt kann direkt von einem Low-Pegel am Pin INT ausgelöst werden, wenn er durch den Befehl ENI freigegeben wurde. Der Prozessor unterbricht dann das laufende Programm und springt zur Adresse 3. Hier steht üblicherweise ein Sprungbefehl zu einer Interrupt-Serviceroutine, die mit RETR endet.

Neben der Programmspeicher-Bankumschaltung lassen sich auch zwei völlig unabhängige Registerbänke wählen. Während die Register R0...R7 in der ersten Bank die internen RAM-Adressen 0 bis 7 belegen, ist ihnen in der zweiten Bank der Bereich 32 bis 40 zugewiesen.

Der Testpin T0 kann mit dem Befehl ENTO CLK als Ausgang für das Clocksignal (1/3 der Quarzfrequenz) verwendet werden. Dieser Zustand wird nur durch einen Reset wieder rückgängig gemacht.

Assembler für 8048-Controller sind als Public-Domain bzw. Shareware-Programme erhältlich. Besonders geeignet ist das Shareware-Programm TASM (Table-driven Assembler) von Thomas N. Anderson. Die Zelle kann als allgemeines Entwicklungssystem eingesetzt werden, da assemblierte Programme ins RAM geladen und gestartet werden können. Ausgetestete Programme lassen sich dann auf Ein-Chip-Systeme übertragen. Eine Alternative zur Programmierung in Assembler stellt die Steuersprache SIMPEL dar, die weiter unten vorgestellt wird.

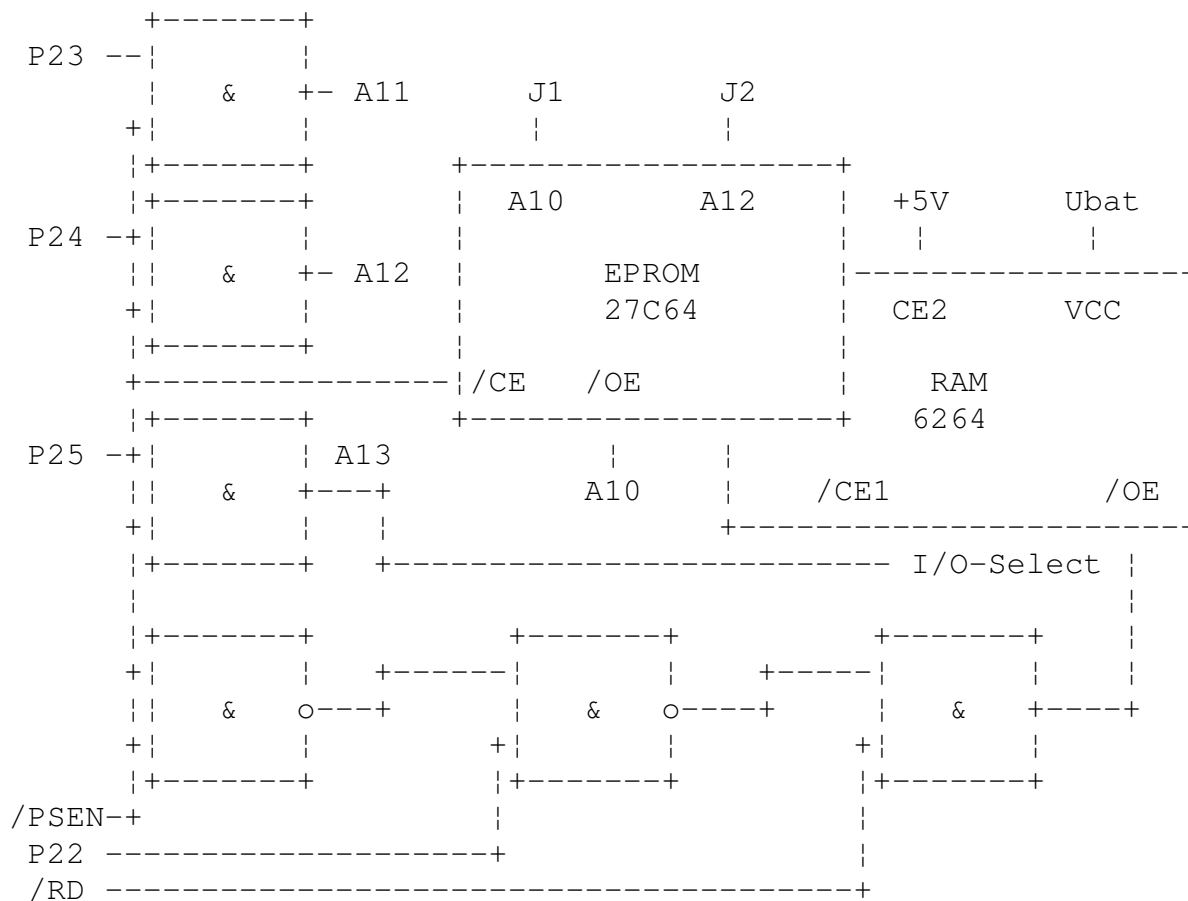
Wandler μ PD7001 ist über über fünf Portleitungen direkt mit dem Prozessor verbunden. Auch die serielle Schnittstelle wird softwaremäßig über Portleitungen realisiert. Zusätzliche Peripherie ist über den Expansionsport an den Daten- und Adreßbus anschließbar.

Hier die technischen Daten des Systems im Überblick:

| | |
|------------------|---|
| Prozessor: | 80C39, ROM-lose C-MOS-Version aus der 8048-Familie |
| Betriebssystem: | Größe 1k, über Jumper J1 und J2 sind bis zu vier Versionen im 8-k-EPROM wählbar |
| RAM: | 8K, davon 6K für Daten, 1k für SIMPEL-Programme |
| AD-Wandler: | 4 Kanäle, 8 Bit, 0...2,5V, geschützt bis +/-10V |
| Geschwindigkeit: | 1 ms/Messung bis 1 min/Messung |
| Versorgung: | Batteriespannung 4V...6V, keine Spannungsregelung, ca. 15mA im Betrieb, 1 μ A in Stellung "AUS" (Datenpufferung) |
| Anschlüsse: | vier Analogeingänge, 0-2,5V, 8 Bit vier digitale Ein/Ausgänge (P14...P17) ein einzelner digitaler Ein/Ausgang (P26) Systembus (=Expansions-Port) |
| Bedienelemente: | EIN/AUS-Schalter, RESET-Taster, START-Taster, eine LED zeigt A/D-Wandlungen. |
| Expansion-Port: | Daten- und Adreßbus für externe Peripherie unter 256 Adressen. Kann über den RAM-Sockel angeschlossen werden. |

2.1 Speichermodell

Da die Mikrocontroller der 8048-Familie intern nur einen 11-Bit-großen Adreßzähler für ihren Programmereich besitzen und außerdem physikalisch getrennte Programm- und Datenspeicher verwenden, ist der Aufwand, zusätzlich ein gemeinsames RAM für Daten und Programme anzuschließen, etwas höher. Zunächst einmal müssen die Leitungen /PSEN für die Freigabe des Programmspeichers und /WR für die Freigabe des Datenspeichers miteinander verknüpft werden, um das RAM in beiden Situationen über seinen Anschluß /OE ansprechen zu können. Dazu genügt im Prinzip ein UND-Gatter (1/4 4081). Zusätzlich wird das /PSEN-Signal jedoch über zwei NAND-Gatter abgeschaltet, solange die Adreßleitung A10 low ist. So kann das RAM nur im Adreßbereich 1K...2K als Programmspeicher arbeiten, während es im ganzen Adreßbereich als Datenspeicher ausgelesen werden kann. Im Bereich 0...1K liegt das EPROM als Adreßspeicher. Zwar liegt seine /CE-Leitung immer an /PSEN, doch wird bei hochgelegter Adreßleitung A10 die Ausgabe über /OE gesperrt. Die höheren Adreßleitungen A11 und A12 werden statisch über Lötbrücken oder Jumper angeschlossen, so daß das ganze EPROM in vier 2K-Blöcke aufgeteilt wird, von denen jeweils nur das untere Kilobyte ausgenutzt wird. Man kann so vier Versionen des Betriebssystems bzw. andere Programme im EPROM bereithalten.



Die Portleitungen P20 bis P25 des Prozessors bilden die höherwertigen Adreßleitungen. Bei Zugriffen auf das Daten-RAM müssen die Adreßleitungen A8 bis A12 durch Portbefehle statisch gesetzt werden. Während der Zugriffe auf den Datenspeicher, also während PSEN low ist, schalten sich die Pegel der Leitungen P20 bis P22 des Prozessor entsprechend dem Zustand des Programm-Counters automatisch um. Damit kann ein Adreßbereich bis zu 2K ohne zusätzlichen Aufwand genutzt werden. Da hier jedoch ein größerer Adreßraum gefordert wird, müssen die höheren Adreßleitungen A11 (P23) bis A13 (P25) durch drei zusätzliche UND-Gatter zurückgesetzt werden, solange PSEN aktiv ist. Nur so kann erreicht werden, daß der Prozessor

Programm-Bytes im unteren Adreßbereich bis 2K findet, während mit Daten im höheren Adreßbereich gearbeitet wird.

Die Adreßleitung A13 dient als Freigabeleitung für den I/O-Bereich, der vom Prozessor wie das RAM mit /WR und /RD angesprochen wird. Deshalb wird das RAM über /CE1 gesperrt, wenn externe Peripherie aktiviert ist. Insgesamt ergibt sich damit folgende Aufteilung des Adreßraums:

| Adressen | Programm | Daten |
|---------------|----------|-------|
| 0000h...0400h | EPROM | RAM |
| 0400h...07FFh | RAM | RAM |
| 0800h...1FFFh | - | RAM |
| 2000h...3FFFh | - | I/O |

Das RAM ist über seinen Anschluß VCC immer mit der Batterie verbunden, während alle übrigen Bausteine ihre Betriebsspannung über den Ein/Aus-Schalter erhalten. Damit wird eine permanente Datenpufferung erreicht. Sobald die Betriebsspannung abgeschaltet wird, wird die Freigabeleitung CE2 des RAMs über die +5V-Leitung und den Betriebsspannungsumschalter an Masse gelegt und das RAM damit gesperrt. Alle Daten bleiben so bei einem Verbrauch von ca 1µA erhalten. In eingeschaltetem Zustand braucht die Zelle dagegen ca 15mA.

2.2 Der A/D-Wandler

Der AD-Wandler μ PD7001 wird über die vier Portleitungen P10 bis P14 gesteuert und über die Leseleitung T1 abgefragt. Er arbeitet nach dem Prinzip der sukzessiven Approximation und führt eine Wandlung in ca 100 μ s aus. Zusätzlich enthält der Baustein einen Vierkanal-Multiplexer, so daß insgesamt vier Meßkanäle mit einem Bereich von 0 bis 2,5V zur Verfügung stehen. Eine extern angeschlossene Bandgap-Referenzdiode LM336 garantiert die Genauigkeit des Wandlers. Der μ PD7001 wird seriell mit Informationen versehen und ausgelesen. Deshalb werden nur wenige Leitungen des Prozessors belegt.

| Pin | Funktion | Anschluß an: |
|-----|-----------------------------|------------------------------|
| 1 | EOC, End of Conversion | frei |
| 2 | DL, Data Latch | P13 |
| 3 | SI, Serial Input | P12 |
| 4 | /SCK, Serial Clock | P21 |
| 5 | SO, Serial Output | T1 mit Pull-Up 10 k Ω |
| 6 | /CS, Chip Select | P20 + LED gegen +5V |
| 7 | C10 Clock | R/C-Netzwerk |
| 8 | C11 Clock | R/C-Netzwerk |
| 9 | Vss, Digital Ground | Masse |
| 10 | A0 | A0 |
| 11 | A1 | A1 |
| 12 | A2 | A2 |
| 13 | A3 | A3 |
| 14 | GND, Analog Ground | Masse |
| 15 | Vref, Referenzspannung 2,5V | LM336 |
| 16 | V+, Betriebsspannung | +5V |

Eine Wandlung mit dem μ PD7001 umfaßt zwei Phasen:

- Übertragen der Kanalnummer 0...3 und Start:
 /CS low
 /SCLK low
 Datenbits D0 und D1 an SI anlegen und jeweils mit steigender Flanke von /SCLK eintakten
 Impuls an DL
 /CS high
 ca. 140 μ s warten oder warten bis /EOC low
- Ergebnis auslesen:
 /CS low
 8 mal: /SCLK low, Datenbit an SO lesen (MSB zuerst), /SCLK high
 /CS high

Die folgende Routine zur Ansteuerung des A/D-Wandlers ist Teil des Betriebssystems der Zelle:

```

0151          ;UPD7001-Messen, Übergabe des Kanals (1...4) im Akku,
0151          ;Ergebnisabgabe im Akku, benötigt ca 0,5ms, verwendet r2
0151          ;Datenübertragung seriell getaktet. Schieberegister über P11 an
0151          ;SCK, Steuerdaten über P12 an SI, Meßdaten über T1 von SO,
0151          ;Freigabe über P10 an CS, Steuern/Ausleseumschaltung über P13
0151          ;an DL. End of Conversion (EOC) wird nicht benutzt, statt
0151          ;dessen feste Warteschleife
0151
0151 99 F0 Messen      anl      p1,#0F0h  ;P10...P13 Null
0153 BA 02          mov      r2,#02   ;Zählschleife 2 Bits
0155 07            dec      a        ;Byte 0...3 für Kanal 1...4

```

```

0156 77          rr      a          ;Bits 0/1 nach 6/7
0157 77          rr      a
0158 F7          rlc      a          ;Kanalbit in c
0159 E6 5D      S11      jnc      Strobe
015B 89 04          orl      p1,#04      ;wenn c=1, Si = 1
015D 89 02      Strobe  orl      p1,#02      ;Clock = 1
015F 99 FD          anl      p1,#0FDh     ;Clock = 0
0161 99 FB          anl      p1,#0FBh     ;Si = 0
0163 EA 58          djnz     r2,S11      ;2 mal
0165 89 08          orl      p1,#008h     ;DL=1, Kanal gesendet
0167 99 F7          anl      p1,#0F7h     ;DL=0
0169 89 01          orl      p1,#1        ;Cs=1, Messung starten
016B BA 1E          mov      r2,#30      ;ca 150 µs/6MHz
016D EA 6D      S12      djnz     r2,S12      ;Warteschleife
016F 99 FE          anl      p1,#0FEh     ;Cs=0
0171 89 08          orl      p1,#08h     ;DL=1, Auslesen
0173 BA 08          mov      r2,#08      ;Schleife 8 Bits
0175 97          S13      clr      c          ;c=0
0176 46 79          jnt1    T1aus      ;T1=0?
0178 A7          cpl      c          ;wenn T1=1, dann c=1
0179 F7          T1aus  rlc      a          ;Datenbit reinschieben
017A 89 02          orl      p1,#02      ;Clock=1
017C 99 FD          anl      p1,#0FDh     ;Clock=0
017E EA 75          djnz     r2,S13      ;acht mal
0180 89 0F          orl      p1,#0Fh     ;P10...P13 = 1
0182 83          ret

```

Der Protanschuß P20 des Prozessors zu Ansteuerung der Freigabeleitung CS des A/D-Wandlers steuert gleichzeitig eine Leuchtdiode. Jeder Zugriff auf den A/D-Wandler kann so optisch verfolgt werden, was oft Hinweise auf die ordnungsgemäße Funktion des Geräts erlaubt. Zusätzlich kann die LED aber auch von einem Programm ohne Zugriff auf den A/D-Wandler benutzt werden.

Alle Analogeingänge sind über Schutzwiderstände von $10k\hat{U}$ an die Anschlußbuchsen gelegt. Zusammen mit den internen Schutzdioden des Wandlers ergibt sich damit ein gewisser Schutz gegen Überspannungen. Der Wandler kann sicher in einem Bereich von -10V bis +10V betrieben werden. Der vierte Analogeingang (A3) ist zusätzlich über einen hochohmigen Spannungsteiler mit der Betriebsspannung verbunden. Dies erlaubt die Kontrolle des Batteriezustands durch den Prozessor. Übliche niederohmige Meßwandler oder Vorverstärker können trotz des Spannungsteilers problemlos an A3 betrieben werden.

2.3 I/O-Ports und weitere externe Anschlüsse

Die beiden Datenleitungen RXD und TXD der seriellen Schnittstelle sind über einfache CMOS-Inverter direkt mit dem Prozessor verbunden. Zwar sind so keine RS232-Normpegel möglich, jedoch kann eine übliche RS232 z.B. eines PCs problemlos direkt angeschlossen werden. Der Prozessor verfügt über keine eingebaute RS232-Hardware-Schnittstelle. Geeignete Unterprogramme müssen deshalb die Datenübertragung selbst steuern. Als Ausgabeleitung wurde P27 gewählt. Die Eingabeleitung ist die Interrupt-Leitung INT, die bei nicht freigegebenem Hardware-Interrupt über bedingte Sprungbefehle im Programm abgefragt werden kann. Damit ergibt sich aber auch die Möglichkeit, über die Datenleitung RXD einen externen Interrupt auszuführen, um etwa ein laufendes Programm anzuhalten und die Kontrolle an einen Hostrechner zu übergeben. Auf diese Weise ist es möglich, die Datenübertragung zwischen Zelle und Hostrechner über eine Dreidraht-Leitung mit GND, RXD und TXT abzuwickeln. Die folgenden RS232-Routinen sind Teil des Betriebssystems der Zelle:

```

0100      .org 100h
0100      ;RS232 Senden, verwendet r2...r3
0100      ;9600 Baud, Sendesignal liegt invertiert an P27
0100      ;Interrupt ist für die Zeit der Ausgabe gesperrt
0100      ;Das Stopbit ist sehr kurz, weil nie zwei Bytes
0100      ;sofort hintereinander ausgegeben werden
0100
0100 15      Senden      dis      i          ;Interrupt gesperrt
0101 BA 08      mov      r2,#8      ;Zählschleife 8 Bits
0103 9A 7F      anl      p2,#7Fh    ;P27=0, Startbit
0105      ;          mov      r3,#162   ;1200Baud
0105 BB 10      mov      r3,#16      ;9600 Baud
0107 EB 07      Sch1      djnz     r3,Sch1  ;Warteschleife Startbit
0109 97          Sp1      clr      c
010A 67          rrc      a          ;Datenbit in C schieben
010B F6 11      jc      Sp2
010D 9A 7F      anl      p2,#7Fh    ;wenn c=0, dann P27=0
010F 24 15      jmp      Sp3
0111 8A 80      Sp2      orl      p2,#80h    ;wenn c=1, dann P27=1
0113 8A 80      orl      p2,#80h
0115      ;Sp3      mov      r3,#160   ;1200Baud
0115 BB 0F      Sp3      mov      r3,#15      ;9600 Baud
0117 EB 17      Sch2      djnz     r3,Sch2  ;Warteschleife Datenbit
0119 EA 09      djnz     r2,Sp1    ;8 mal wiederholen
011B 23 00      mov      a,#0
011D 23 00      mov      a,#0
011F 8A 80      orl      p2,#80h    ;P27=1, Stopbit
0121      ;          mov      r3,#108   ;1200 Baud
0121 BB 01      mov      r3,#1      ;kurzes Stopbit, 9600 Baud
0123 EB 23      Sch3      djnz     r3,Sch3  ;Warteschleife Stopbit
0125 05          en      i          ;Interrupt wieder frei
0126 83          ret

```



```

0127      ;rs232 Empfangen, verwendet r2...r4
0127      ;Eingang ist INT mit Abfrage über jni. Baudrate 9600 bei 6MHz
0127      ;Hardware-Interrupt ist für die Zeit der Eingabe gesperrt
0127      ;Das Label "Startbit" darf auch extern angesprungen werden,
0127      ;wenn ein Signal an INT erkannt wurde.
0127
0127 15      Empf      dis      i          ;Interrupt sperren
0128 86 2C      jni      Startbit ;Int = 0?

```

```

012A 24 27      jmp      Empf      ;wenn Int = 1
012C           ;Startbit      mov      r2,#225   ;1200 Baud
012C BA 19      Startbit      mov      r2,#25    ;9600 Baud
012E EA 2E      Sch6          djnz     r2,Sch6   ;Warten 1,5 Startbit
0130 BB 08      mov      r3,#8     ;Schleife 8 Bits
0132 97         Sp6           clr      c         ;c=0
0133 86 43      jni      Null      ;Int = 0 ?
0135 00         nop                     ;Zeitausgleich
0136 A7         cpl      c         ;C=1
0137 2C         xch     a,r4       ;Byte von r4 holen
0138 67         rrc     a         ;c in a schieben
0139 2C         xch     a,r4       ;Byte wieder nach r4
013A           ;           mov      r2,#162   ;1200 Baud
013A BA 0F      mov      r2,#15    ;9600 Baud
013C EA 3C      Sch7          djnz     r2,Sch7   ;Warten Datenbit
013E EB 32      djnz     r3,Sp6    ;8 Bits empfangen?
0140 FC         mov      a,r4      ;empfangenes Byte in a
0141 05         en      i         ;Interrupt wieder frei
0142 83         ret
0143 00         Null          nop                     ;Schleife für Nullbits:
0144 97         clr      c         ;c=0
0145 2C         xch     a,r4       ;Byte von r4 holen
0146 67         rrc     a         ;c in a schieben
0147 2C         xch     a,r4       ;Byte wieder nach r4
0148           ;           mov      r2,#162   ;1200 Baud
0148 BA 0F      mov      r2,#15    ;9600 Baud
014A EA 4A      Sch8          djnz     r2,Sch8   ;Warten Datenbit
014C EB 32      djnz     r3,Sp6    ;8 Bits empfangen?
014E FC         mov      a,r4      ;empfangenes Byte in a
014F 05         en      i         ;Interrupt wieder frei
0150 83         ret

```

Für viele Anwendungen ist eine Start-Taste sinnvoll, die vom Prozessor gelesen werden kann. Sie wurde hier an die Eingangsleitung T0 angeschlossen. Da diese Leitung hochohmig ist, muß zusätzlich ein Pull-Up-Widerstand verwendet werden. Eine weitere Taste befindet sich am RESET-Eingang des Prozessors. Mit dieser Taste kann der Prozessor wie nach einem Einschalten der Betriebsspannung in den Grundzustand versetzt werden.

Alle noch freien Portleitungen des 80C39 werden herausgeführt. Die Portleitungen P14 bis P17 werden als digitale I/O-Leitungen D0...D3 bezeichnet, während die freie Leitung P26 als digitale Steuerleitung ST eingesetzt wird. Die fünf digitalen Leitungen werden nicht besonders geschützt, damit sie sowohl als Eingänge als auch als Ausgänge TTL-kompatibel bleiben.

Der Ruhezustand aller fünf digitalen Portleitungen nach dem Einschalten der Zelle ist high. In diesem Zustand sind sie hochohmig und dürfen als digitale Eingänge benutzt werden. Schalter können wie bei Standard-TTL-Bausteinen direkt gegen Masse angeschlossen werden. Wird ein Portanschluß durch ein Programm zurückgesetzt, dann wird er niederohmig, so daß ohne weitere Treiber kleine Lasten mit Strömen bis zu 10 mA direkt getrieben werden können. Z.B. läßt sich eine Leuchtdiode mit einem Vorwiderstand von 330Ω direkt gegen +5V anschließen. Sie wird damit in negativer Logik betrieben.

Werden Portanschlüsse als Eingänge verwendet und von außen hochgesetzt, gleichzeitig aber fälschlich durch einen Ausgabebefehl von innen zurückgesetzt und damit niederohmig gemacht, dann können Kurzschlußströme von ca. 30mA pro Portanschluß fließen. Um diesen Fall auszuschließen, sollte man nie Schalter oder andere niederohmige Elemente direkt gegen +5V anschließen. Schalter werden im einfachsten Fall in umgekehrter Logik gegen Masse angeschlossen. Legt man Wert auf positive Logik, kann der Schalter mit einem Vorwiderstand von 1kΩ gegen +5V angeschlossen werden. Zusätzlich ist in diesem Falle aber

noch ein Pull-Down-Widerstand von ca $10\text{k}\hat{\Omega}$ erforderlich, damit der Eingang im Ruhezustand des Schalters low ist.

3. Das Betriebssystem

Das Betriebssystem steuert alle wichtigen Funktionen der Zelle. Es ist im EPROM des Systems untergebracht und hat eine Größe von unter einem Kilobyte. Bei der Entwicklung wurde ein Kompromiß zwischen Einfachheit und Vollständigkeit angestrebt. Dabei stand der Einsatz als universelles Meß- und Steuersystem im Vordergrund. Folgende Grundfunktionen müssen vorhanden sein:

- Autonomes Durchführen von Meßserien mit einstellbaren Parametern: (Länge, Kanäle, Zeiten)
- Datenaustausch zwischen Daten-RAM und Host-Rechner
- Direktes Ansprechen von A/D-Wandler und I/O-Leitungen vom Host-Rechner aus
- Nachladen von Zusatzprogrammen in das RAM
- Jederzeit mögliche Unterbrechung durch den Hostrechner

Alle diese Funktionen sollen durch eine RS232-Schnittstelle ansprechbar sein. Zwischen Host und Zelle wird ein Steuerprotokoll festgelegt, das über Ein-Byte-Kommandos bestimmte Funktionen der Zelle aufruft. Innerhalb jeder Funktion kann wieder ein Datenaustausch nach festgelegtem Muster erfolgen. Die Grundstruktur des Programms sieht damit so aus:

```

+----> Kommando empfangen
|
|      Kommando dekodieren
|
|      Funktion aufrufen
|
|      Funktion:
|      z.B. Daten empfangen
|           Daten verarbeiten
|           Daten zurücksenden
|
+-----+

```

Die Funktion im Beispiel führt nur eine Aktion durch. Dies ist z.B. beim Ansprechen des A/D-Wandlers der Fall. Die Zelle muß zunächst den gewünschten Kanal erfahren, dann eine Messung über diesen Kanal ausführen und schließlich das Ergebnis zurücksenden. Es gibt aber auch Fälle, in denen eine Funktion aus einer beliebigen Anzahl von Wiederholungen derselben Unterfunktion besteht. Ein Beispiel dafür ist das sequenzielle Auslesen des Daten-RAMs. In einem solchen Fall muß zusätzlich eine Vereinbarung über die Beendigung des Vorgangs festgelegt werden. Sinnvoll ist z.B. die Festlegung eines Unterkommandos "0" für "Ende".

Das Betriebssystem kennt drei Hauptzustände, die im batteriegepufferten RAM als Startinformation vermerkt werden:

| Modus | Startinformation |
|---------------------------------|------------------|
| Bereitschaft für neue Kommandos | 0 |
| Meßmodus | 129 |
| Programmmodus | 130 |

Jeder dieser Modi wird nach dem Einschalten oder nach einem Reset automatisch wieder aktiviert, wenn er vorher eingeschaltet war. Das bedeutet, man kann die Zelle in den Meßmodus versetzen, dann ausschalten und erst am Meßort wieder einschalten, wobei dann die Messung aktiviert wird. Genauso kann ein

Userprogramm geladen werden, wobei die Zelle in den Programm-Modus gelangt. Das Programm läuft dann nach jedem Einschalten oder nach einem Reset erneut ab. Sowohl der Meßmodus als auch der Programmmodus kann durch einen Interrupt über die Leitung RXD abgebrochen werden. Dazu wird einfach das Zeichen "0" an die Zelle gesendet.

Nach jedem Einschalten oder Reset beginnt das Betriebssystem bei Adresse Null. Hier wird neben anderen Grundeinstellungen auch der externe Interrupt freigegeben. Dann wird im "System-RAM" bei Adresse 03FF die Startinformation gelesen, ausgewertet und eine entsprechende Verzweigung eingeleitet. So ist der alte Betriebsmodus wieder aktiv. Ein Interrupt über die Leitung RXD führt in jedem Falle in die Interrupt-Servocsroutine "Inter". Hier wird die Startinformation "0" in das System-RAM geschrieben und außerdem das Betriebssystem mit zurückgesetztem Stack neu gestartet. Dabei werden alle Rücksprungadressen aus eventuellen Unterprogrammen vergessen, so daß eine Unterbrechung aus beliebigen Programmsituationen möglich ist.

Wird vom Betriebssystem oder von einem User-Programm die serielle Schnittstelle benutzt (Unterprogramme Empf oder Sende), dann wird für die Zeit des seriellen Datenaustauschs der Interrupt gesperrt. Dies kann aber dazu führen, daß ein nachgeladenes Programm, das die serielle Schnittstelle verwendet, nicht mehr ohne weiteres abgebrochen werden kann. Für diesen Fall wurde eine zweite Art zum zwangsweisen Einschalten des Befehlsmodus vorgesehen: Wird nach einem Reset für ca 5s die Starttaste gedrückt, befindet sich das System wieder im Grundzustand. Kürzere Zeiten führen in die beiden anderen Modi. Mit einer Sekunde gelangt man in den Meßmodus, mit drei Sekunden in den Programmmodus. So ist es auch möglich, gleichzeitig Parameter für eine Messung und ein Userprogramm im RAM bereitzuhalten und ohne Hilfe eines Hostrechners einzuschalten.

Das vollständige Assembler-Listing ist in der Datei ZELLE.LST enthalten.

Von einem Hostrechner aus können die folgenden Kommandos gegeben werden:

1. Start einer Messung.
Eingabe der Blocklänge für Messungen als Highbyte und Lowbyte. Eingabe von vier Meßkanälen 1...4, oder "aus" (=0). Eingabe von Highbyte und Lowbyte der Intervallzeit minus Anzahl der Kanäle in ms. Nach den insgesamt acht Steuer-bytes kann der Rechner für jeden Block durch START gestartet werden. Statt der Taste "START" kann auch ein Byte "9" vom Hostrechner die nächste Serie starten. Der Meßmodus kann über die Schnittstelle durch ein Byte "0" des Hostrechners unterbrochen werden. Alle acht Eingabeparameter werden an den Anfang des Daten-RAMs ab Adresse 0800h kopiert. Es folgen die gewonnenen Meßwerte in aufsteigender Reihenfolge. Nach dem Kommando ist eine Wartezeit von 1ms erforderlich.
2. Auslesen des RAMs über die RS232.
Der Hostrechner sendet für jedes gewünschte Byte eine "1" und erhält jeweils das nächste Byte aus dem Daten-RAM zurück. Eine "0" vom Hostrechner unterbricht den Modus. Die Daten werden aus dem Datenbereich des RAMs ab Adresse 0800h sukzessive ausgelesen. Die ersten acht Bytes sind die eingestellten Meßparameter. Die folgenden Bytes sind Meßwerte. Nach dem Kommando ist eine Wartezeit von 1ms erforderlich.
3. Beschreiben des RAMs durch den Hostrechner.
Der Hostrechner wartet für jedes Byte zunächst ein Byte der Zelle ab. Die Zelle sendet nur zu Timing-Zwecken ein zufälliges Byte ohne weitere Bedeutung. Darauf sendet der Hostrechner eine "1" als Zeichen, daß ein Byte folgt oder eine "0" als Zeichen für das Ende der Übertragung. Die Daten werden im Datenbereich des RAMs ab Adresse 0800h abgelegt. Nach dem Kommando ist eine Wartezeit von 1ms erforderlich.
4. Direktes Lesen der Analogeingänge.

Der Hostrechner sendet jeweils die gewünschte Kanalnummer (1...4) und erhält als Antwort den Meßwert als Byte. Die Leseaktion kann beliebig oft wiederholt werden. Beendigung dieses Modus geschieht durch Senden einer "0" statt der Kanalnummer. (Vgl. auch Kommando 7)

5. Laden eines eigenen Programms.
Die Übertragung entspricht genau der beim Kommando 3, mit dem Unterschied, daß die übertragenen Daten in den Programmbereich des RAMs ab Adresse 0400h gelangen. Das Programm wird durch das Kommando "6" oder durch RESET bzw. Aus- und Einschalten gestartet. Nach dem Kommando ist eine Wartezeit von 1ms erforderlich.
6. Start eines eigenen Programms ab 0400h.
Es läuft zunächst nur einmal ab, kann aber durch die RESET-Taste erneut gestartet werden. Unterbrechung des Programmmodus erfolgt durch Senden eines Bytes "0". Neustart durch das Kommando "6" ist ohne neues Laden möglich.
7. Einmaliges direktes Lesen eines Analogeingangs.
Wie Kommando 4, aber mit dem Unterschied, daß nur eine Messung ausgeführt wird. Danach besteht wieder Bereitschaft für ein neues Kommando.
8. Interface-Modus nach dem Protokoll des "Parallelen Interface-Bus":
externe Peripherie kann direkt über die RS232 unter den Adressen 0...127 angesprochen werden. Das Bit 7 gibt die Datenrichtung an, 0=Schreiben, 1=Lesen. Nach einer Schreib- oder Leseaktion beendet sich dieser Modus selbst.
9. Start
Dieses Kommando kann an Stelle der Taste "START" die nächste Serie des laufenden Meßprogramms starten. Aus der allgemeinen Befehlsbereitschaft heraus hat es keine Wirkung.
10. Start einer Serienmessung mit den zuletzt benutzten Parametern.
Ein Block der Messung wird danach jeweils durch "START" oder das Kommando "9" gestartet.
11. Beschreiben des Digitalports 1
Die Zelle erwartet als nächstes Byte den gewünschten Zustand des Ports. Bits 0...3 gehören zum A/D-Wandler und sollen high bleiben. Bit 0 kann low gesetzt werden, um die LED anzuschalten. Bits 4...7 sind als Digitaleingänge herausgeführt. Der Zustand aller Leitungen ist nach dem Einschalten high. Anschlüsse, die als Eingänge genutzt werden sollen, müssen high gesetzt bleiben. Nach Aufruf einer Messung sind die unteren vier Portbits high, die oberen vier bleiben unverändert.
12. Auslesen des Digitalports 1
Die Zelle sendet den Zustand des Ports als ein Byte zurück. Die oberen vier Bits sind als Digitale Ein/Ausgänge zugänglich, die unteren vier Bits gehören zum A/D-Wandler. Ein Portanschluß kann nur dann als Eingang genutzt werden, wenn er nicht vorher auf Null gesetzt wurde. Offene Eingänge sind nach einem Neustart grundsätzlich high.
13. Lesen des Zustands der START-Taste
Die Zelle antwortet mit einer "0" für "gedrückt" oder einer "1" für "nicht gedrückt".
14. Beschreiben des Digitalports 2
Die Zelle erwartet eine "1" oder eine "0" als Steuerbit für die einzelne Ausgangsleitung P26.
15. Auslesen des Digitalports 2
Die Zelle antwortet mit "1" oder "0", je nach Zustand der Leitung P26. Soll der Anschluß als Eingang genutzt werden, darf er vorher nicht zurückgesetzt werden.
16. Auslesen eines Prozessorregisters

Die Zelle erwartet die gewünschte Adresse und antwortet mit dem Inhalt der Adresse. Das Kommando kann z.B. verwendet werden, um nach dem Stop einer Messung die Anzahl der gemessenen Werte zu erfragen.

17. Beschreiben eines Prozessorregisterns

Die Zelle erwartet die Adresse (0...127) der gewünschten Zelle des internen Prozessor-RAMs und das Datenbyte. Das Kommando kann zur gezielten Parameterübergabe an eigene Programme verwendet werden.

18. Direktes Auslesen einer beliebigen RAM-Adresse

Die Zelle erwartet Highbyte und Lowbyte der RAM-Adresse und antwortet mit dem Inhalt.

19. Direktes Beschreiben einer beliebigen RAM-Adresse

Die Zelle erwartet Highbyte und Lowbyte der RAM-Adresse und das Datenbyte.

Die Datenübertragung zum Hostrechner erfolgt mit 9600 Baud, 8 Bit, NO Parity, 2 Stopbits. Nach dem Einschalten des Systems kann es sich zufällig oder noch von der letzten Benutzung her im Programmmodus oder im Meßmodus befinden. Es muß daher immer mit einem Byte "0" zurückgesetzt werden. Eine Serie von zehn Bytes "0" holt die Zelle aus jedem Zustand in den Startzustand zurück, also auch dann, wenn sich das System durch Fehlbedienung in einem falschen Zustand befindet. Zwischen den einzelnen "0"-Bytes sollen 2ms Wartezeit liegen, weil dies die maximale Bearbeitungszeit zufällig noch aktiver Funktionen sicher überschreitet.

Fast alle Kommandos dürfen mit der maximal möglichen Geschwindigkeit zusammen mit den erforderlichen Daten gesendet werden, da die zugehörigen Funktionen innerhalb der Zeit für zwei Stopbits beendet sind. Eine Ausnahme bilden die Kommandos, die ein Einrichten des Adreßpointers auf den erforderlichen Speicherbereich bewirken: Nach den Kommandos 1, 2, 3 und 5 muß eine Pause von 1ms gelassen werden, bevor die weitere Datenübertragung beginnen kann.

4. Ansteuerung der Zelle aus Hochsprachen

Das Betriebssystem der Zelle ist ganz und gar auf die Zusammenarbeit mit einem Hostrechner ausgelegt. Alle Funktionen können durch Kommandos über die RS232-Schnittstelle aufgerufen werden. Im Normalfall wird es für die einzelnen Anwendungsbereiche Programme geben, die in einer Hochsprache geschrieben sind. Außer der Datenkopplung können sie je nach Anwendungsfall auch die Auswertung von Daten, die Konvertierung von Daten oder komplexe Steuerungen auf der Ebene des Hostrechners enthalten. Dabei spielt die Wahl der verwendeten Sprache im Prinzip keine Rolle. Die folgenden Beispielprogramme demonstrieren die Verwendung der Zelle-Kommandos in Basic.

Das erste Beispiel zeigt das direkte Auslesen aller vier Analogkanäle in einer Endlosschleife:

```
10 REM Direkte Messung an vier Kanälen
20 OPEN "COM1:9600 ,N,8,1,cs,ds" AS #1
30 FOR N = 1 TO 10: PRINT#1, CHR$(0);: NEXT N: REM System_Reset
40 CLS
50 FOR N = 1 TO 4
60 PRINT#1, CHR$(7); CHR$(N);           : REM Messen Kanal N
70 A = ASC (INPUT$(1,1))                : REM Ergebnis lesen
80 LOCATE N+5,10
90 PRINT "Kanal ";N;" : "; A/100; " V "
100 NEXT N
110 IF INKEY$ = "" THEN GOTO 50
```

Das zweite Beispiel zeigt die Verwendung der Serien-Meßfunktion der Zelle in Basic. Es wird mit fest eingestellten Parametern gearbeitet, und die Meßergebnisse werden der Einfachheit halber nur am Bildschirm ausgegeben:

```
10 REM Meßserie an Kanal 1 und 2 mit 100 Messungen und 10 ms/Messung
20 OPEN "COM1:9600 ,N,8,1,cs,ds" AS #1
30 FOR N = 1 TO 10: PRINT#1, CHR$(0);: NEXT N : REM System_Reset
40 PRINT#1, CHR$(1); CHR$(0); CHR$(100);      : REM Serienmessung, 100 mal
50 PRINT#1, CHR$(1); CHR$(2); CHR$(0);CHR$(0);: REM Kanal 1 und 2
60 PRINT#1, CHR$(0); CHR$(10);                : REM 100 ms
70 CLS
80 PRINT " Bitte Messung starten. Taste drücken zum Auslesen. "
90 IF INKEY$="" THEN GOTO 90
100 FOR N = 1 TO 10: PRINT#1, CHR$(0);: NEXT N : REM System_Reset
110 PRINT#1, CHR$(2);                          : REM Auslesen
120 FOR N = 1 TO 8
130 PRINT#1, CHR$(1);                          : REM Anforderung
140 A$ = INPUT$(1,1): NEXT N                  : REM Parameterblock
150 FOR N= 1 TO 100
160 PRINT#1, CHR$(1);                          : REM Anforderung
170 A = ASC (INPUT$(1,1))                      : REM Ergebnis lesen
180 PRINT#1, CHR$(1);                          : REM Anforderung
190 B = ASC (INPUT$(1,1))                      : REM Ergebnis lesen
200 PRINT N; ": Kanal 1 : "; A/100; " V      Kanal 2 : "; B/100; " V"
210 NEXT N
220 PRINT "Nächster Block? j/n"
230 INPUT C$
240 IF C$="j" THEN GOTO 150
250 PRINT#1,CHR$(0);                          : REM Auslesen Ende
```

Für Pascal und Comal wurden Prozeßsprachen in Form von Spracherweiterungen entwickelt, die den Umgang mit der Zelle erleichtern. Die Hardware-nahe Erzeugung von Zelle-Kommandos und die Übertragung von Parametern und Meßergebnissen wurden in einfach handhabbare Prozeduren verpackt.

Dadurch werden die obigen Programme z.B. in Pascal wesentlich kürzer und übersichtlicher. Die folgenden Beispielprogramme demonstrieren die Verwendung der Prozeßsprache in Form der Unit "Zelle".

Program Vierkanal_Direktmessung;

```
Uses Crt, Zelle;
```

```
Procedure Messen;
```

```
var n : Integer;
```

```
begin
```

```
  for n:=0 to 3 do begin
```

```
    GotoXY (10,n+5);
```

```
    write ('Kanal ',n+1,': ',UEin(n):3:2,' V');
```

```
  end;
```

```
end;
```

```
begin
```

```
  Init;
```

```
  CLrScr;
```

```
  Repeat
```

```
    Messen
```

```
  until KeyPressed;
```

```
end.
```

Program Messserie;

```
Uses Crt, Zelle;
```

```
Procedure Messung;
```

```
Var m, n : Integer;
```

```
  Ch : Char;
```

```
begin
```

```
  BlockDef (100,3,10);
```

```
  Writeln ('Bitte Messung Starten und Taste drücken für Ausgabe');
```

```
  Repeat until KeyPressed;
```

```
  Ch := ReadKey;
```

```
  writeln ('Auslesen der Daten, bitte warten.');
```

```
  BlockEin(10);
```

```
  Repeat
```

```
    m:= 9;
```

```
    for n:=1 to 100 do begin
```

```
      write (n,' Kanal 1: ',Dat_Puf[m+2*n]/100:3:2,' V ');
```

```
      writeln (' Kanal 2: ',Dat_Puf[m+2*n+1]/100:3:2,' V ');
```

```
    end;
```

```
    Write ('Nächster Block j/n ');
```

```
    Readln (Ch);
```

```
    m:=m+200;
```

```
  until Ch <> 'j';
```

```
end;
```

```
begin
```

```
  Init;
```

```
ClrScr;  
Messung;  
end.
```


5. Der SIMPEL-Compiler

Die Mikrocontroller der 8048-Familie sind nicht sonderlich für die Implementierung höherer Programmiersprachen geeignet, weil sie nur kleine Programme bis maximal 4 K aufnehmen können und weil sie mit einem kleinen Stack von 16 Bytes arbeiten. Deshalb war bisher keine Sprache für diesen Prozessor erhältlich.

Bei der Entwicklung der Zelle kam der Wunsch nach einer einfachen höheren Programmiersprache auf, die dem Anwender in den meisten Fällen das mühsame Erlernen des Umgangs mit Assembler ersparen sollte. Diese Sprache mußte einfach zu erlernen sein und sollte möglichst leicht zu implementieren sein. Gleichzeitig mußte sie sehr sparsam mit dem Speicherplatz umgehen. Dafür war es nicht erforderlich, alle Möglichkeiten eines Assemblers zu erreichen. Eine Interpretersprache kam nicht in Frage, da sie zu viel Speicherplatz benötigen würde.

Daher sollte ein Cross-Compiler entwickelt werden, der einen Quelltext im PC in 8048-Maschinencode übersetzt, der dann als fertiges Maschinenprogramm in die Zelle übertragen werden kann. Die Sprache dieses Compilers wurde "SIMPEL" genannt, weil sie einerseits leicht erlernbar ist, andererseits aber auch nur eingeschränkte Möglichkeiten hat.

Die Grundidee für die Sprache Simpel besteht darin, Befehle zu definieren, die als einfache Aufrufe von Unterprogrammen des Betriebssystems übersetzt werden können. Zum Beispiel kann der Befehl EinRS232 zum Lesen eines Bytes von der seriellen Schnittstelle aus dem Assemblerbefehl Call 0127 bestehen (Siehe Listing des Betriebssystems der Zelle im Anhang). Nach dem Rücksprung aus dem Unterprogramm liegt das Ergebnis im Akku vor. Umgekehrt übergibt der Befehl AusRS232 ein zu sendendes Bytes im Akku an die entsprechende Systemroutine.

Allgemein soll immer der Akku das Ergebnis einer Operation erhalten. Damit hat man die schwierige Umsetzung eines Variablen-Konzepts umgangen. In diesem Punkt ist Simpel einem Assembler sehr nahe. Der Akku bekommt in Simpel den Namen "X" und stellt die einzige wirkliche Variable dar. Daneben muß es noch Hilfsvariablen geben. Das Register R7 wurde Y genannt und ist für Rechenoperationen und Vergleiche mit X bestimmt. Zusätzlich wurden die Adressen 32 bis 52 des internen RAMs als allgemeine Zwischenspeicher festgelegt und "Merkspeicher" genannt. Als Zähler für Schleifen wurden die Register R0 bis R7 der zweiten Registerbank eingesetzt. Sie erhalten die Namen "A" bis "H". Insgesamt arbeitet Simpel also nur mit X, Y, Merk 1 ... Merk 20 und A...H. Damit werden viele Schwierigkeiten der Assemblerprogrammierung vom Benutzer ferngehalten:

- Er braucht den Aufbau des internen Speichers nicht zu kennen.
- Es gibt keine Bankumschaltung.
- Doppelverwendung eines Registers als Variable und als Schleifenzähler ist ausgeschlossen.
- Die wenigen zugelassenen Merkspeicher können nicht mit dem Betriebssystem kollidieren.
- Die Register R0 bis R6 können vom Programmierer nicht verändert werden und sind dem Betriebssystem vorbehalten.

Der Simpel-Compiler sollte selbst einfach und überschaubar sein. Deshalb wurde als Grundstrategie der Übersetzung festgelegt, daß jede Befehlszeile direkt über eine Tabelle interpretierbar und übersetzbar sein muß. Dies hat Simpel mit Assembler gemeinsam. Deshalb kann eine Programmzeile maximal aus einem Befehl und einem Parameter bestehen. Das einfachste Beispiel ist das Laden von X mit einer Zahl:

```
in Simpel: X 3          ;      In Assembler  mov  a, #3
```

In beiden Fällen lautet die Übersetzung 23h 03h. Neben Unterprogrammaufrufen aus dem Betriebssystem und einfachen Maschinenbefehlen enthält die Tabelle aber auch kurze Programmsequenzen, die für einen

Befehl eingesetzt werden. Ein Beispiel ist der Befehl Minus, der nicht als Maschinenbefehl vom Prozessor bereitgestellt wird, sondern mit mehreren Befehlen programmiert werden muß.

Die vollständige Befehlsübersicht, Grundlagen der Programmierung in Simpel und einige Programmbeispiele befinden sich in der Bedienungsanleitung zum Simpel-Compiler. Weitere Hilfen kann man der Simpel-Programmsammlung entnehmen. An dieser Stelle soll der Aufbau der Befehle und des Compilers erklärt werden.

5.1 Die Simpel-Grundbefehle

Hier werden zunächst alle Befehle vorgestellt, die sich ohne besondere Berücksichtigung ihrer Adresslage erzeugen lassen. Die Tabelle SIMPEL.TAB des Simpel-Compilers (Siehe Anhang) legt jeweils den Befehl und die zugehörige Übersetzung fest. Hier wird zuerst das Assembler-Listing der Grundbefehle gezeigt. In der Befehltabelle werden neben den hexadezimalen OP-Codes auch besondere Zeichen oberhalb FF als Anweisungen an den Compiler benutzt:

- XX** Das zweite Wort der Programmzeile ist eine Zahl und wird hier eingesetzt.
- RR** Das zweite Wort in der Programmzeile ist eine Zahl und kennzeichnet einen Merkspeicher. Sie wird mit 32 addiert und hier eingesetzt.
- PA** Optionale Parameterübergabe: Wird hinter dem Befehl ein Wert angegeben, wird dieser zuvor in X geladen.

Um den Aufbau der Befehle zu verdeutlichen, wurde als Parameter für **XX** jeweils der Wert 05 eingesetzt, für den Merkspeicher **RR** jeweils das Register 32 (= Merk 0). Das Zeichen **PA** für die optionale Parameterübergabe wurde nicht mit in das Assemblerlisting übernommen. Es dient dazu, einfache Befehle, die einen Wert in X verarbeiten, mit und ohne Parameter verwenden zu können. Wird hinter dem Befehl ein Wert angegeben, so fügt der Compiler vor dem Befehl die Zuweisung "X Wert" ein. Im folgenden Assemblerlisting wird dagegen davon ausgegangen, daß der Wert schon im Akku vorliegt. Der Zusatz PA ist in Klammern im Kommentar angegeben.

Parameter dürfen dezimal, hexadezimal oder binär angegeben werden. Das Programm prüft jeweils die Art der Angabe und führt entsprechende Berechnungen durch.

Das Register R6 wird zur Zwischensicherung des Akkus verwendet, wenn ein Befehl wie z.B. AusRS232 einen Wert in X verarbeitet, der nachher unverändert vorliegen soll, obwohl die aufgerufene Systemroutine den Akku verändert.

Während fast alle Befehle direkt eingesetzt werden oder aus einem Unterprogrammaufruf bestehen, verwendet der Befehl "Stop" einen Sprungbefehl auf eine Endlosschleife im Betriebsprogramm. Eine Rückkehr aus diesem Zustand ist nur durch Reset oder Inerrupt möglich.

Assemblerlisting der Grundbefehle

```

-----
23 05      X          mov     a, #05

BF 05      Y          mov     r7, #05

AF          XnachY    mov     r7, a

FF          YnachX    mov     a, r7

B9 20      Merk       mov     r1, #32
A1                    mov     @r1, a

B9 20      LiesMerk   mov     r1, #32
F1                    mov     a, @r1

03 20      LiesMerk(X) add     a, #32
A9                    mov     r1, a
F1                    mov     a, @r1

AE          SchreibeRAM mov     r6, a    ;X sichern (PA)
34 8F                    call    018Fh
FE                    mov     a, r6    ;X zurücladen

34 AB      LiesRAM    call    01ABh

34 83      AnfangRAM  call    0183h

34 51      Messen     call    0151h    ; (PA)

39          AusPort   outl   p1, a     ; (PA)

09          EinPort   in     a, P1

8A 40      StAn       orl    p2, #40h

9A BF      StAus      anl    p2, #0BFh

0A          EinSt     in     a, P2
E7                    rl     a
E7                    rl     a
53 01                    anl    a, #01

B9 05      AusI/O     mov     r1, #05 ; (PA)
8A 20                    orl    p2, #20h
91                    movx   @r1, a
9A DF                    anl    p2, #0DFh

B9 05      EinI/O     mov     r1, #05

```

```

8A 20      orl      p2,#20h
81         movx    a,@r1
9A DF      anl      p2,#0DFh

AE         AusRS232  mov    r6,a      ;X sichern,  (PA)
34 00      call    0100h
FE         mov    a,r6      ;X zurückladen

34 27      EinRS232  call    0127h

03 05      Plus     add    a,#05

6F        PlusY    add    a,r7

BE 05      Minus    mov    r6,#05
37         cpl    a
6E        add    a,r6
37         cpl    a

37        MinusY   cpl    a
6F        add    a,r7
37        cpl    a

53 05      AND      anl    a,#05

5F        ANDY     anl    a,r7

43 05      OR       orl    a,#05

4F        ORY     orl    a,r7

D3 05      XOR      xrl    a,#05

DF        XORY     xrl    a,r7

37        NOT     cpl    a

E7        LinksSchieben  rl    a

77        Rechtsschieben rr    a

24 F4      Stop    jmp    01F4h

74 1F      Taste   call   031Fh

54 D7      WarteTaste call  02D7h

55        UhrStart strt  t

```

Die Umsetzung eines Compilers für die Grundbefehle ist relativ einfach. Im Anhang findet sich der Quelltext

zu einer einfachen Version des Compilers (SIMPEL_2.PAS) ohne Editor und Schnittstelle, die den Kern der integrierten Entwicklungsumgebung SIMPEL darstellt.

Zunächst wird die Befehlstabelle SIMPEL.TAB in die Arrays "Befehl" und "Opcode" eingelesen, indem jede Zeile der Tabelle in das Schlüsselwort und den entsprechenden Maschinencode aufgeteilt wird. Dabei werden die erlaubten Befehle gezählt. Die maximale Anzahl erlaubter Befehle wird in "MaxBef" geführt.

Die Hauptprozedur "Compiler" des Programms liest den Quelltext und teilt jede Zeile in "Wort1" und "Wort2". Dabei werden Leerzeichen als Trennung zwischen den Worten gelesen oder als Einrückzeichen interpretiert. Ein drittes Wort würde als Kommentar überlesen. Die beiden Worte einer Zeile können z.B. ein Befehl und ein Parameter sein und müssen im folgenden übersetzt werden. Dies übernimmt die Prozedur "Translate".

In Translate wird die Befehlsliste nach dem Vorkommen des "Wort1" abgesucht. Wird es nicht gefunden, gibt es eine Fehlermeldung. Ansonsten stellt der parallele Eintrag in Opcode die Übersetzungsanweisung dar. Wenn sich hier nur hexadezimale Zahlen zwischen 00 und FF finden, werden sie einfach in das Ausgabearray Hexcode [Adr] kopiert, wobei die gültige Adresse mit hochgezählt wird. Alle Zeichen über FF stellen besondere Anweisungen an den Compiler dar, die in der Prozedur "Sonder" behandelt werden. Dort werden z.B. entsprechend der Anweisung XX Werte eingesetzt.

5.2 Prozeduren

Ein Simpel-Programm besteht im allgemeinen aus mehreren Prozeduren und einem Hauptprogramm. In die ersten beiden Adressen des Programmspeichers (0400h und 0401h) trägt der Compiler einen Sprungbefehl zum Anfang des Hauptprogramms ein, so daß alle Prozeduren zunächst übersprungen werden.

Die Prozeduren sind Unterprogramme, die mit dem Rücksprungbefehl `ret` abgeschlossen werden. Für den Namen der Prozedur muß im weiteren ein Unterprogrammaufruf der Anfangsadresse der Prozedur eingesetzt werden. Während des Compilierens wird `MaxBef` für jede neue Prozedur um eins erhöht und zugleich die Arrays `Befehl` und `Opcode` erweitert. Der Name der neuen Prozedur ist dann gleichwertig mit einem schon vordefinierten Befehl. In der Befehlstabelle `SIMPEL.TAB` finden sich folgende Steueranweisungen für den Compiler:

PD Definition einer Prozedur. Der Prozedurname wird zusammen mit dem Befehltext zu seinem Aufruf an die Befehlsliste angefügt.

HA Hauptprogramm-Anfang. Ein Sprungbefehl auf die aktuelle Adresse wird nachträglich an den Anfang des Programms gesetzt.

Assemblerlisting der Programmstruktur

```
-----
                ;Prozedur Name                ; (PD)
83              ProzedurEnde    ret                ; (SO)
                ;Anfang                        ; (HA)
24 F4          Ende            jmp      01F4h      ; (SO)
```

5.3 Erweiterte Kontrollstrukturen

Zusätzlich zu den einfachen Befehlen gibt es in Simpel die folgenden Kontrollstrukturen:

- Zählschleifen: Amal ...
- Endlosschleife: Immer ...
- Begingte Aufrufe: WennX=0 ...
- Bedingte Programmschleifen: Solange...
- ...
- SolangeEnde

Diese Strukturen erfordern vom Compiler verschiedene Adressenberechnungen für Sprungbefehle. Es ist zwischen den kurzen Sprüngen der bedingten Sprungbefehle auf der selben Seite und langen Sprüngen des jmp-Befehls zu unterscheiden. Die Berechnung der Adressen wird durch besondere Zeichen über FF in der Übersetzungstabelle SIMPEL.TAB gesteuert. Die folgende Liste zeigt alle definierten Anweisungen:

- XX Das zweite Wort der Programmzeile ist eine Zahl und wird hier eingesetzt.
- RR Das zweite Wort in der Programmzeile ist eine Zahl und kennzeichnet einen Merkspeicher. Sie wird mit 32 addiert und hier eingesetzt.
- PA Optionale Parameterübergabe: Wird hinter dem Befehl ein Wert angegeben, wird dieser zuvor in X geladen.
- PD Definition einer Prozedur. Der Prozedurname wird zusammen mit dem Befehltext zu seinem Aufruf an die Befehlsliste angefügt.
- HA Hauptprogramm-Anfang. Ein Sprungbefehl auf die aktuelle Adresse wird nachträglich an den Anfang des Programms gesetzt.
- HH Es wird kein Byte eingesetzt, aber der Compiler mekt sich die aktuelle Adresse für einen späteren (kurzen) Rücksprung.
- SS An dieser Stelle wird das Lowbyte der bei HH gespeicherten Adresse als Rücksprungadresse eingetragen.
- PR Das zweite Wort der Programmzeile ist ein Prozedurname oder ein Befehl. An dieser Stelle werden ein oder mehrere Bytes dieses Befehls eingesetzt.
- NN Das zweite Wort der Programmzeile ist ein Prozedurname oder ein Befehl. Es wird die Länge des Maschinencodes festgestellt und die Vorwärts-Sprungadresse zum bedingten Überspringen des Befehls an dieser Stelle eingesetzt.
- JP An dieser Stelle wird der JMP-Befehl 84, A4, C4 oder E4 je nach aktueller Seite (4...7) des Programmspeichers eingesetzt.
- HS Der Schleifenzähler wird incrementiert, und die Aktuelle Adresse wird für einen späteren Rücksprung in einer bedingten Programmschleife gespeichert.
- JS Es werden die Befehle zum Rücksprung an die mit HS bezeichnete Stelle eingesetzt.
- J3 Es wird die Adresse für einen drei Byte weiten Vorwärtssprung berechnet und hier eingesetzt.
- JZ In die Aktuelle Adresse wird das vorläufige Sprungziel "00" engesetzt, und die Adresse wird für das spätere Einsetzen der richtigen Sprungadresse gespeichert.
- JE Die aktuelle Sprungadresse wird in die bei "JZ" gespeicherte Adresse eingesetzt. Der Schleifenzähler wird decrementiert.
- SO Es wird am Ende einer Prozedur oder des Hauptprogramms geprüft, ob noch Schleifen geöffnet sind.
- HX Es werden die folgenden hexadezimalen Bytes in das Programm eingesetzt.

Alle besonderen Anweisungen an den Compiler werden in der Prozedur "Sonder" interpretiert und bearbeitet, wobei für jeden Sonderfall eine eigene Prozedur zuständig ist.

Die übergeordnete Prozedur Translate übernimmt u.a. die Überwachung der Adressen des entstehenden Maschinenprogramms auf mögliche Wechsel der Seiten. Dies ist wichtig, da bei 8048-Prozessoren bedingte Sprünge nicht über die Grenzen einer Seite möglich sind. Der Einfachheit halber werden beim Erreichen einer Adresse, die um 16 unter einer neuen Seite liegt, also bei F0h, 1F0h, 2F0h usw, nop-Befehle "00" eingesetzt, bis die neue Seite erreicht ist.

Die im Programm geführte aktuelle Adresse (Adr) beginnt bei Null, während die tatsächliche Adresse im Zielsystem bei 0400h beginnt. Die wirkliche Areßlage wird bei allen Sprungbefehlen beachtet. Adr dient dem Aufbau des Ausgabe-Arrays Hexcode [Adr]. Hier wird das übersetzte Programm in hexadezimaler Form abgelegt. Die Übersetzung beginnt mit Adresse 2, weil bei der Übersetzung des Anfangs des Hauptprogramms (Anweisung : "Anfang") ein absoluter Sprungebefehl auf den dann bekannten Programmanfang in die Adressen 0 und 1 eingesetzt wird. Ein ähnlicher Rückgriff auf kleinere Adressen findet bei der Übersetzung der Solange-Schleifen statt. Hier muß eine Sprungadresse eingesetzt werden, die erst am Ende der Schleife bekannt ist.

Die Zählschleifen können mit acht verschiedenen Zählern A bis H gebildet werden, die die Register r0 bis r7 der zweiten Registerbank (Bank 1) verwenden. Während Simpel-Programme sonst nur die Registerbank 0 verwenden, werden für die Zählschleifen die Register r0 bis r7 in Bank 1 benutzt. Die Register können direkt geladen oder über den Akku geladen und ausgelesen werden. Das folgende Listing zeigt die Verwendung des Registers r0 für den Zähler A. Die Zähler B bis H verwenden entsprechend die Register r1 bis r7. Der Inhalt der eigentlichen Zählschleife kann entweder die Programmsequenz eines Befehls oder ein Prozeduraufruf sein. Hier wurde zur Verdeutlichung die Prozedur "Name" verwendet.

Assemblerlisting der Wiederholschleifen

```

0402 83          Name          ret

0403 D5          A              sel      rb1
0404 B8 05       mov      r0,#05
0406 C5          sel      rb0

0407 D5          XnachA        sel      rb1
0408 A8          mov      r0,a
0409 C5          sel      rb0

040A D5          AnachX        sel      rb1
040B F8          mov      a,r0
040C C5          sel      rb0

040D C5          Amal          sel      rb0      ;HH
040E 94 02       call     Name
0410 D5          sel      rb1
0411 E8 0D       djnz    r0,Amal  ;SS
0413 C5          sel      rb0

0414 94 02       Immer          call     Name
0416 84 14       jmp     Immer

0418 AC          Warte          mov      r4,a      ; (PA)
0419 34 C1       HH              call     01C1h□
041B EC 19       djnz    r4,HH     ;SS

```

Im Gegensatz zu den Zählschleifen führt die Verwendung der Immer-Schleife zu einer endlosen Wiederholung, die nur durch Reset oder einen Interrupt abgebrochen werden kann.

Der Warte-Befehl besteht aus einer Wiederholschleife unter Verwendung des Registers r4 der Registerbank 0. Der Aufruf der Systemroutine zum Warten bis zur nächsten abgelaufenen Millisekunde ist fest eingefügt.

Bedingte Prozeduraufrufe greifen auf die bedingten Sprungbefehle des Prozessors zurück. Wenn die Bedingung an einen Vergleich mit Y gebunden ist, dann wird zuerst eine Subtraktion durchgeführt. Der einzusetzende Befehl bzw. der Prozeduraufruf wird bedingt übersprungen. Die Sprungadresse zeigt also auf den nach der Wenn-Struktur stehenden Programmcode.

Assemblerlisting der Wenn-Strukturen

```

0402 83          Name          ret

0403 96 07      WennX=0      jnz      NN1
0405 94 02          call      Name
0407          NN1          ...

0408 C6 0C      WennX>0      jz       NN2
040A 94 02          call      Name
040C          NN2          ...

040D AE          WennX>=Y     mov      r6,a    ;X sichern
040E 37          cpl      a
040F 6F          add      a,r7
0410 37          cpl      a
0411 FE          mov      a,r6
0412 F6 16      jc       NN3
0414 94 02          call      Name
0416          NN3          ...

0417 AE          WennX<Y     mov      r6,a    ;X sichern
0418 37          cpl      a
0419 6F          add      a,r7
041A 37          cpl      a
041B FE          mov      a,r6
041C E6 20      jnc      NN4
041E 94 02          call      Name
0420          NN4          ...

0421 AE          WennX=Y     mov      r6,a    ;X sichern
0422 37          cpl      a
0423 6F          add      a,r7
0424 37          cpl      a
0425 03 FF      add      a,#255
0427 FE          mov      a,r6
0428 F6 2C      jc       NN5
042A 94 02          call      Name
042C          NN5          ...

```

Während die bisherigen Kontrollstrukturen nur einen Befehl oder einen Prozeduraufruf erlauben, schließt die letzte unter Verwendung der Solange-Befehle mehrere Programmzeilen ein. Sie ist daher besonders aufwendig umzusetzen und erfordert eine Ausnahme von der Strategie, Zeile für Zeile direkt zu übersetzen. Es sind nämlich Sprungadressen einzusetzen, die erst im weiteren Verlauf bekannt werden.

Solange-Schleifen dürfen bis zu achtfach ineinander geschachtelt werden. Man kann so durch komplexere Prozeduren Prozeduraufufe einsparen, um den Prozessor-Stack weniger zu belasten. Geöffnete Schleifen werden im Compiler in der Variablen Schleife_Nr gezählt. Beim Beenden einer Prozedur und am Ende des Hauptprogramms wird überprüft, ob alle geöffneten Schleifen auch wieder geschlossen wurden.

Die SolangeNichtFertig-Schleife verwendet das Prozessor-Flag F0 als Bedingung für die Ausführung der Schleife. Es kann durch eigene Befehle beliebig gesetzt oder gelöscht werden.

Assemblerlisting der Solange-Schleifen

```
-----
0402 83          Name          ret
0403 85          Fertig        clr      F0      ;F0=0
0404 85          NichtFertig    clr      F0
0405 95          cpl      F0      ;F0=1

0406            SolangeNichtFertig
0406 B6 0A      HS            jF0      J3
0408 84 0D      jmp      JZ
040A            J3            ...      ;Programm-
                                ...      ;zeilen
040B 84 06      SolangeEnde    jmp      HS
040D            JZ            nop
```

Die übrigen Solange-Schleifen unterscheiden sich nur durch den verwendeten bedingten Sprungbefehl, der jeweils direkt den Akku oder den Zustand der Leitung T0 abfragt.

| Schleife | Befehl | Opcode |
|--------------------|--------|--------|
| SolangeNichtFertig | jF0 | B6 |
| SolangeX=0 | jz | C6 |
| SolangeX>0 | jnz | 96 |
| SolangeTaste | jnT0 | 26 |
| SolangeNichtTaste | jT0 | 36 |

5.4 Spracherweiterungen

Simpel kann von jedem fortgeschrittenen Anwender leicht verändert oder erweitert werden. Da alle Befehle über die Tabelle SIMPEL.TAB definiert sind, ist es auch möglich, die Schlüsselworte komplett auszutauschen, um sie z.B. anderen Sprachkonzepten anzupassen oder in eine andere Landessprache zu übertragen.

Erweiterungen der Befehltabelle sind ohne weiteres möglich, indem das gewünschte Schlüsselwort und der Programmcode in einer neuen Zeile an die Liste angehängt werden. Bei einer Durchsicht der vorhandenen 8048-Maschinenbefehle fallen noch zahlreiche Möglichkeiten ins Auge, die bisher aus Gründen der Einfachheit der Sprache nicht umgesetzt wurden. Beispiele sind:

- anl p2,#data und orl p2,#data zur direkten Anwendung auf die digitalen Ein/Ausgabeleitungen
- inc a und dec a zum Erhöhen und Verkleinern des Akkus um Eins
- Verwendung des zweiten Prozessor-Flags F1 zum Aufbau einer weiteren Solange-Struktur mit globalem Flag
- jb0 bis jb7 zum Aufbau von Wenn-Befehlen
- swap a zum Tauschen von Halbbytes
- retr zum Wiederherstellen von F1 (Fertig/NichtFertig) nach einem Rücksprung

Die folgenden Befehle wurden nach der ersten ausgelieferten Version des Simpel-Compilers angefügt.

Zusätzliche Grundbefehle

| | | | |
|----|------------------|------|------------|
| 00 | Nichts | nop | |
| 2F | TauscheXY | xch | a, r7 |
| 47 | TauscheHalbbytes | swap | a |
| 83 | ProzedurAusgang | ret | ; ohne SO□ |
| 93 | ProzedurAbschluß | retr | ; (SO) |

ProzedurAusgang kann in einer Prozedur mit einer Bedingung verknüpft werden. ProzedurAbschluß ersetzt ProzedurEnde mit der Wirkung, daß das Fertig-Flag (F0) den alten Zustand erhält.

Damit kann die SolangeNichtFertig-Schleife mehrfach und unabhängig voneinander in verschiedenen Prozeduren verwendet werden. Fertig/NichtFertig gilt also nicht mehr global, sondern nur noch lokal.

Zusätzliche Wenn-Strukturen

```

-----
0403 36 07      WennTaste      jT0      NN1
0405 94 02      call      Name
0407           NN1           ...

0408 26 0C      WennNichtTaste jnT=     NN2
040A 94 02      call      Name
040C           NN2           ...

040D AE        WennX<>Y      mov      r6,a    ;X sichern
040E 37        cpl      a
040F 6F        add      a,r7
0410 37        cpl      a
0411 FE        mov      a,r6
0412 E6 16     jnc      NN3
0414 94 02     call      Name
0416           NN3           ...

```

Wenn neue Befehle implementiert werden sollen, kann man sie zunächst mit der Code-Anweisung im Programm ausprobieren. Bei Erfolg können sie mit dem Editor des Simpel-Compilers an die Befehlsliste SIMPEL.TAB angefügt werden. Um die Tabelle einzuladen, muß das Datei/Laden-Menü mit <Esc> verlassen werden. Man kann dann einen Dateinamen eingeben, der nicht die Endung .SIM hat. Nach dem Speichern der veränderten Tabelle muß Simpel zunächst verlassen werden, damit die Tabelle neu eingelesen wird.

Eine andere Möglichkeit zum Implementieren neuer Befehle liegt in der ausschließlichen Verwendung der Code-Anweisung. Mit ihr kann wie mit der Inline-Anweisung in Turbo Pascal direkter Maschinencode in das Programm eingesetzt werden. Im allgemeinen wird man eine Prozedur für jeden neuen Befehl benutzen, um Namen vergeben zu können. Diese Möglichkeit ist besonders für sehr spezielle Befehle geeignet, die nur für ein Programm gebraucht werden (vgl. Kap 5.6, Wahlfreier Zugriff auf das RAM). Ein Nachteil gegenüber der Implementierung über die Befehlstabelle ist, daß mehr Prozeduraufrufe benötigt werden. Es können wegen der begrenzten Größe des Prozessor-Stack nur acht ineinander geschachtelte Aufrufe erfolgen.

5.5 Zugriff auf Systemparameter

Die zwanzig Merkspeicher in Simpel nehmen die Adressen 33 bis 52 im internen RAM des Mikrocontrollers ein. Ein Blick in das Betriebssystem zeigt, daß die Adressen 51 und 52 (Merk 19, 20) als Zähler für die Blocklänge bei Serienmessungen verwendet werden. Ein Konflikt ist durch diese Doppelbelegung nicht zu erwarten, da nur entweder die Betriebssystemebene oder ein Simpelprogramm aktiv sein kann.

Andererseits können darüberliegende Adressen ebenfalls wie Merkspeicher angesprochen werden, wodurch sich erweiterte Möglichkeiten ergeben. Insbesondere die beiden Speicheradressen des Pointers auf das Daten-RAM können sinnvoll verändert werden:

| interne RAM-Adresse | Merkspeicher | Funktion |
|---------------------|--------------|------------------|
| 61 | 29 | Pointer-Highbyte |
| 62 | 30 | Pointer-Lowbyte |

Der Befehl AnfangRAM stellt den Adreßpointer auf 0800h ein. Jeder Zugriff auf das RAM mit SchreibeRAM oder LiesRAM erhöht den Pointer, so daß im Normalfall mit sequenziellen Daten gearbeitet wird. Durch Wertzuweisungen an die Merkspeicher 29 und 30 kann jede beliebige Adresse im gesamten RAM eingestellt werden, also auch der unbenutzte Bereich von 0000h bis 03FEh, der Systemspeicher bei 03FFh und der Programmbereich ab 0400h.

Das Beispielprogramm "Systemparameter" zeigt, wie die Betriebsebenen der Zelle willkürlich beeinflußt werden können. Normalerweise führt jeder Reset bei eingeschaltetem Programmmodus zu einem Neustart des Simpel-Programms. Ein endgültiger Abbruch ist nur durch einen Interrupt über die RS232 oder durch langes Drücken der Starttaste nach Reset möglich. Die Prozedur systemram ermöglicht jedoch die Vorgabe anderer Startinformationen in der Adresse 03FFh.

Weitere Eingriffe in Funktionen des Betriebssystems können durch Einfügen von Maschinencode mit dem Befehl Code erreicht werden. Im Hauptprogramm des Beispiels wird zunächst der Interrupt gesperrt, so daß der Hostrechner keinerlei Einfluß mehr auf die Zelle hat. Nach Beendigung des Programms durch die Starttaste wird der Ausgangszustand der Zelle durch direktes Anspringen der Interrupt-Routine hergestellt. Insgesamt wird so erreicht, daß das Programm nur einmal ablaufen kann und daß sowohl Reset als auch die Starttaste wieder in den Befehlsmodus führen.

;Programm Systemparameter

Prozedur systemram

```

Merk 1           ;X zwischenspeichern
X C3h           ;beachte: P26 u.P27 high (C0h)
Merk 29         ;Adreßpointer high (03h + C0h)
X FFh
Merk 30         ;Adreßpointer low
LiesMerk 1      ;X zurückladen
SchreibeRAM     ;in 03FFh schreiben
ProzedurEnde

```

Anfang

```

systemram 0     ;Kommandobereitschaft nach Reset
Code 15        ;dis i, Interrupt abschalten
SolangeNichtTaste
AusPort 254    ;LED an

```

```
SolangeEnde
AusPort 255          ;LED aus
Code 04C2           ;jmp 00C2h, Interrupt-Routine
Ende
```

Das interne RAM des 80C39 hat noch wesentlich mehr Speicherzellen als bisher verwendet wurden. Die zweite Hälfte des Speichers von Adresse 64 bis Adresse 127 wurde nicht verwendet, um kompatibel zu den kleineren Prozessortypen 8048, 8035 und 8748 zu bleiben, da Simpelprogramme prinzipiell auch in kleinen Ein-Chip-Systemen lauffähig sind. Wenn dies aber nicht angezielt wird, können zusätzlich die Merkspeicher 32 bis 95 verwendet werden.

5.6 Wahlfreier Zugriff auf das RAM

Die Befehle LiesRAM und SchreibeRAM arbeiten sequentiell und sind nach AnfangRAM nur im Datenbereich ab 0800h benutzbar. Der untere RAM-Bereich von einem Kilobyte (parallel zum Betriebssystem im EPROM) blieb bisher ungenutzt. Er eignet sich aber sehr gut zur Zwischensicherung von Daten oder für Tabellen. Deshalb werden hier Prozeduren vorgestellt, mit denen jede RAM-Adresse wahlfrei erreicht werden kann. Y wird als Adresspointer verwendet. Die angezielte Seite (256-Byte-Bereich) des Speichers muß extra gesetzt werden.

Vergleichbare Prozeduren werden auch für das interne RAM vorgestellt. Damit erhält man einen anderen Zugriff auf Merkspeicher, der sich besonders bei berechneten Adressen bewährt.

;Programm RAM-Zugriff

```
Prozedur InTRAMschreiben ;X Datum, Y Adresse
TauscheXY
Code A8 ; mov r0,x
TauscheXY
Code A0 ; mov @r0,a
ProzedurEnde
```

```
Prozedur InTRAMlesen ; Y Adresse
TauscheXY
Code A8 ; mov r0,x
TauscheXY
Code F0 ; mov a,@r0
ProzedurEnde
```

```
Prozedur RAMSeite ; X Seite
Plus C0h
Code 3A ; outl p2,a
ProzedurEnde
```

```
Prozedur RAMschreiben ; X Daten, Y Adresse
TauscheXY
Code A8 ; mov r0,x
TauscheXY
Code 90 ; movx @r0,a
ProzedurEnde
```

```
Prozedur RAMlesen ; X Daten, Y Adresse
TauscheXY
Code A8 ; mov r0,x
TauscheXY
Code 80 ; movx a,@r0
ProzedurEnde
```

```
Y 250  
RAMlesen  
Y 32  
inRAMschreiben  
Ende
```

Das Hauptprogramm kopiert Adresse 250 des RAMs in die Adresse 32 (= Merkspeicher 1) des internen RAMs. Die Prozeduren lassen sich auch vermischt mit den Befehlen LiesRAM und SchreibeRAM einsetzen. Allerdings verändern sie die aktuelle Seite, sodaß sie wieder neu gesetzt werden muß.

6. Hardware-Erweiterungen für die Zelle

Für viele Aufgaben fehlen der Zelle eine ausreichende Zahl kräftiger digitaler Ausgänge. Wenn vier oder fünf Ausgänge genügen, kann man sich im einfachsten Fall mit Darlington-Treiberbausteinen ULN 2003 oder ähnlichen behelfen, die direkt an die Leitungen D0...D3 und St angeschlossen werden. Sie benötigen jedoch einen Steuerstrom von ca 1mA und müssen daher mit Pull-Up-Widerständen von 2,2kΩ angeschlossen werden.

Wenn man mehr Ausgänge braucht und über den Sammelstecker gehen will, dann empfiehlt sich die Verwendung von Schieberegistern, die jeweils mit einer Datenleitung und einer Taktleitung angesteuert werden. Die Steuersoftware kann in Simpel geschrieben werden. Eine andere Alternative ist die Verwendung eines 8243-Erweiterungsbausteins.

Für umfangreiche Erweiterungen der Zelle sollte man den I/O-Bus verwenden. Er stellt 128 Adressen für externe Peripherie zur Verfügung, die sowohl über die Prozeßsprachen als auch aus Simpel heraus direkt beschrieben und gelesen werden können.

6.1 Der I/O-Bus

Im Adreßbereich ab 2000h (oberhalb 8kB) ist sowohl das RAM als auch das EPROM der Zelle gesperrt. Greift der 80C39 mit /WR oder /RD auf diesen Bereich zu, dann ist die Portleitung P25 high. Im Prinzip kann man weitere 8K adressieren. Allerdings ist die Software nur auf 256 (Simpel) bzw 128 Adressen (Prozeßsprache) ausgelegt, was für Peripheriebausteine durchaus ausreicht.

Der Erweiterungsbus ist über den RAM-Sockel erreichbar. Mit einer Zwischenfassung können alle Anschlüsse über ein Flachbandkabel an die Erweiterung geführt werden:

| Pin | Anschluß | Pin | Anschluß |
|-----|----------|-----|---------------------|
| 1 | /RD | 28 | Ubat (direkt) |
| 2 | (A12) | 27 | /WE |
| 3 | A7 | 26 | +5V (über Schalter) |
| 4 | A6 | 25 | (A8) |
| 5 | A5 | 24 | (A9) |
| 6 | A4 | 23 | (A11) |
| 7 | A3 | 22 | (OE) |
| 8 | A2 | 21 | (A10) |
| 9 | A1 | 20 | I/O-Select |
| 10 | A0 | 19 | D7 |
| 11 | D0 | 18 | D6 |
| 12 | D1 | 17 | D5 |
| 13 | D2 | 16 | D4 |
| 14 | GND | 15 | D3 |

(Anschlüsse in Klammern werden nicht für externe Peripherie benötigt.)

Mit seinen Datenleitungen, Adreßleitungen und den Steuerleitungen /RD und /WR kann der Erweiterungsbus an alle üblichen Bausteine wie 8255, 8253, Latches wie 74LS574, A/D-Wandler wie μ PD7002, A/D-Wandler wie den ZN428 und zahlreiche weitere busorientierte Peripherie angeschlossen werden. Es muß allerdings beachtet werden, daß das I/O-Select-Signal umgekehrte Polarität zum sonst üblichen /CS-Signal besitzt. Invertiert man es z.B. mit einem CMOS-Baustein, dann können alle oben genannten Bausteine angeschlossen werden. Verwendet man z.B. nur einen 8255, so erhält man 24 weitere bidirektionale Portleitungen. Ohne weitere Bausteine werden die Adressen des 8255 mehrfach gespiegelt, was jedoch problemlos ist, wenn man nur die Adressen 0, 1, 2 und 3 anspricht.

Für größere Projekte mit mehreren externen Bausteinen kann es nötig werden, einen Adreßdecoder einzusetzen. Für nähere Einzelheiten verweise ich auf mein Buch "Messen, Steuern und Regeln über die RS232-Schnittstelle" im Franzis-Verlag.

6.2 Ansteuerung einer LCD-Anzeige

Die Zelle kann über ihren Expansionsbus eine intelligente Standard-LCD-Anzeige ansteuern. Das folgende Simpel-Programm zeigt die grundlegende Steuerung und einige nützliche Prozeduren für erweiterte Anwendungen. Es ist als "Steinbruch" für eigene Versuche zu verstehen.

```
;Ansteuerung des LCD-Displays LM-16251 von SHARP
;E an Y4 (HC138-Ausgang, invertiert), Adressen 64...79
;R/W an A0 (ungerade Adressen nur Lesen)
;RS an A1 (Befehle: Adressen 64/65, Daten: Adressen 66/67)
```

```
Prozedur bereitschaft ;Jede Aktion erfordert zuerst ein
NichtFertig ;Feststellen der Bereitschaft des
SolangeNichtFertig ;Displays, sonst Zerstörungsfahr.
EinI/O 65
AND 128
WennX=0 Fertig
SolangeEnde
ProzedurEnde
```

```
Prozedur befehl ;Ausgabe an Adresse 64
Merk 1 ;Übergabe in X
bereitschaft
LiesMerk 1
AusI/O 64
ProzedurEnde
```

```
Prozedur daten ;Datenausgabe über Adresse 66
Merk 1 ;Übergabe in X
bereitschaft
LiesMerk 1
AusI/O 66
ProzedurEnde
```

```
Prozedur init ;Erforderlich nach dem Einschalten
befehl 38h
befehl 1
befehl 0Ch ;Cursor aus, 0E= Cursor an
befehl 6
ProzedurEnde
```

```
Prozedur cursor ;Cursor-Position ansteuern
Plus 128
befehl
ProzedurEnde
```

```
Prozedur abc ;Erzeugen aufsteigender ASCII-Zeichen
LiesMerk 2
daten
```

LiesMerk 2
Plus 1
Merk 2
ProzedurEnde

Prozedur **schreibe** ;Eine Reihe A..P schreiben (Test)
B 16
X 65
Merk 2
Bmal abc
cursor 64 ;Zweite Reihe Zeichen schreiben
B 16
X 81
Merk 2
Bmal abc
ProzedurEnde

Prozedur **abisf** ;Hilfsprozedur für Hexadezimalzeichen
Plus 7
ProzedurEnde

Prozedur **hexcode** ;Erzeugen von Hex_Zeichen
Merk 3
LinksSchieben
LinksSchieben
LinksSchieben
LinksSchieben
AND 15
Y 10
WennX>=Y abisf
Merk 4 ;Ausgabe an Position 2/3
cursor 2
LiesMerk 4
Plus 48
daten
LiesMerk 3
AND 15
Y 10
WennX>=Y abisf
Plus 48
daten
ProzedurEnde

Prozedur **dezimalcode** ;Dezimalzahlen ausgeben
Merk 6
Merk 3
X 0
Merk 10

```
NichtFertig
Y 100
LiesMerk 3
WennX<Y Fertig
SolangeNichtFertig
LiesMerk 3
Minus 100
WennX<Y Fertig           ;Hunderter
Merk 3
LiesMerk 10
Plus 1
Merk 10
SolangeEnde
X 0
Merk 11
NichtFertig
Y 10
LiesMerk 3
WennX<Y Fertig
SolangeNichtFertig
LiesMerk 3
Minus 10
WennX<Y Fertig           ;Zener
Merk 3
LiesMerk 11
Plus 1
Merk 11
SolangeEnde
X 0
Merk 12
NichtFertig
Y 1
LiesMerk 3
WennX<Y Fertig
SolangeNichtFertig
LiesMerk 3
Minus 1                   ;Einer
WennX<Y Fertig
Merk 3
LiesMerk 12
Plus 1
Merk 12
SolangeEnde
LiesMerk 10
Plus 48
daten
LiesMerk 11
Plus 48
daten
```

LiesMerk 12
Plus 48
daten
LiesMerk 6
ProzedurEnde

Prozedur **zählen** ;Hochzählen (Test)
LiesMerk 5
hexcode
cursor 10
LiesMerk 5
dezimalcode
LiesMerk 5
Plus 1
Merk 5
Warte 255
Warte 255
ProzedurEnde

Prozedur **vierkanäle** ;4-Kanal-Digitalvoltmeter
cursor 0
Messen 1
Dezimalcode
cursor 4
Messen 2
dezimalcode
cursor 8
Messen 31
dezimalcode
cursor 12
Messen 4
dezimalcode
X 250
Warte
ProzedurEnde

Prozedur **rechnen** ;Beispiel für Rechnung
Cursor 5
X 1
Merk 7
NichtFertig
SolangeX>0
cursor 5
dezimalcode 200
daten 32
LiesMerk 7
dezimalcode
daten 32
LiesMerk 7

```
XnachY
X 200
MinusY
dezimalcode
Warte 255
Warte 255
LiesMerk 7
Plus 1
Merk 7
SolangeEnde
Stop
ProzedurEnde
```

```
Anfang
init
X 0
Merk 5
Immer rechnen           ;Aufruf der gewünschten Aktion ...
;vierkanäle
;zählen
Ende
```


Anhang

Literatur

Platine, Bestückungsplan, Teileliste

8048-Programmierkarte

Register-Operationen

| Register | R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7 |
|-------------|----|----|----|----|----|----|----|----|
| ADD A,R | 68 | 69 | 6A | 6B | 6C | 6D | 6E | 6F |
| ADDC A,R | 78 | 79 | 7A | 7B | 7C | 7D | 7E | 7F |
| ANL A,R | 58 | 59 | 5A | 5B | 5C | 5D | 5E | 5F |
| ORL A,R | 48 | 49 | 4A | 4B | 4C | 4D | 4E | 4F |
| XRL A,R | D8 | D9 | DA | DB | DC | DD | DE | DF |
| INC R | 18 | 19 | 1A | 1B | 1C | 1D | 1E | 1F |
| DEC R | C8 | C9 | CA | CB | CC | CD | CE | CF |
| DJNZ R,adr | E8 | E9 | EA | EB | EC | ED | EE | EF |
| MOV A,R | F8 | F9 | FA | FB | FC | FD | FE | FF |
| MOV R,A | A8 | A9 | AA | AB | AC | AD | AE | AF |
| MOV R,#data | B8 | B9 | BA | BB | BC | BD | BE | BF |
| XCH A,R | 28 | 29 | 2A | 2B | 2C | 2D | 2E | 2F |

Register relativ

| Register | R0 | R1 |
|--------------|----|----|
| ADD A, @R | 60 | 61 |
| ADDC A, @R | 70 | 71 |
| ANL A, @R | 50 | 51 |
| ORL A, @R | 40 | 41 |
| XRL A, @R | D0 | D1 |
| INC @R | 10 | 11 |
| MOV A, @R | F0 | F1 |
| MOV @R,A | A0 | A1 |
| MOV @R,#data | B0 | B1 |
| XCH A, @R,# | 20 | 21 |
| XCHD A, @R | 30 | 31 |
| MOVX A, @R | 80 | 81 |
| MOVX @R,A | 90 | 91 |

Sprungbefehle

| Bit 5-7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---------|----|----|----|----|----|----|----|----|
| JMP | 04 | 24 | 44 | 64 | 84 | A4 | C4 | E4 |
| JBb | 12 | 32 | 52 | 72 | 92 | B2 | D2 | F2 |
| CALL | 14 | 34 | 54 | 74 | 94 | B4 | D4 | F4 |

Ein/Ausgabe

| Port | 1 | 2 |
|---------------|--------|----|
| IN A,P | 09 | 0A |
| OUTL p,A | 39 | 3A |
| ANL p,#data | 99 | 9A |
| ORL P,#data | 89 | 8A |
| MOVD A,p | 0C..0F | |
| MOVD p,A | 3C..3F | |
| ANLD p,A | 9C..9F | |
| ORLD p,A | 8C..8F | |
| INS a,Bus | | 08 |
| OUTL Bus,A | | 02 |
| ANL Bus,#data | | 98 |
| ORL Bus,#data | | 88 |

Bedingte Sprungbefehle

| | | | | | |
|------|----|----------|----|------|----|
| JC | F6 | JMPP a A | B3 | JT0 | 36 |
| JNC | E6 | JZ | C6 | JNT0 | 26 |
| JT1 | 56 | JNZ | 96 | JTF | 16 |
| JNT1 | 46 | JF0 B6 | | JNI | 86 |
| JF1 | 76 | RET | 83 | RETR | 93 |

Arithmetische Befehle

| | | | | | | | |
|--------------|----|-------------|----|-------|----|-------|----|
| ADD A,#data | 03 | ANL A,#data | 53 | INC A | 17 | RLA | E7 |
| ADDC A,#data | 13 | ORL A,#data | 43 | DEC A | 07 | RLC A | F7 |
| DA A | 57 | XRL A,#data | D3 | CLR A | 27 | RRA | 77 |
| SWAP A | 47 | | | CPL A | 37 | RRC A | 67 |

Timer-Befehle

| | |
|-----------|----|
| MOV A,T | 42 |
| MOV T,A | 62 |
| STRT T | 55 |
| STRT CNT | 45 |
| STOP TCNT | 65 |
| EN TCNTI | 25 |
| DIS TCNTI | 35 |

Flags

| | |
|--------|----|
| CLR C | 97 |
| CPL C | A7 |
| CLR F0 | 85 |
| CPL F0 | 95 |
| CLR F1 | A5 |
| CPL F1 | B5 |

Interne Steuerung

| | |
|----------|----|
| ENI | 05 |
| DISI | 15 |
| SEL RB0 | C5 |
| SEL RB1 | D5 |
| SEL MB0 | E5 |
| SEL MB1 | F5 |
| ENTO CLK | 75 |

Sonstige

| | |
|-------------|----|
| MOV A,#data | 23 |
| MOV A,PSW | C7 |
| MOV PSW,A | D7 |
| MOVP A,@A | A3 |
| MOVP3 A,@A | E3 |
| NOP | 00 |

Assemblerlisting des Zelle-Betriebssystems

```

0001 0000          ;Betriebssystem für die 8035-Zelle, 0...1kB im EPROM
0002 0000          ;Version 2.5.90
0003 0000          ;
0004 0000          ;Besondere Adressen des internen RAM:
0005 0000          ;61,62 Hi,Lo- aktuelle RAM-Adresse
0006 0000          ;59,60 Hi,Lo- Wartezeit im ms
0007 0000          ;55..58 aktive Kanäle 1...4, 0=aus
0008 0000          ;53,54 Hi,Lo- Länge eines Datenblocks
0009 0000          ;51,52 Hi,Lo- Zähler für Blocklänge
0010 0000          ;Besonders reservierte Register:
0011 0000          ;R7 = Y in SIMPEL
0012 0000          ;R6 = PUSH/POP-Speicher in SIMPEL
0013 0000
0014 0000          ;Nach dem Programmstart wird ein Kommando über die RS232
0015 0000          ;erwartet, das zu den verschiedenen Grundfunktionen führt.
0016 0000          ;Befindet sich die Autostart-Information "129" oder "130" in
0017 0000          ;der Speicherzelle 03A0 des batteriegepufferten RAMs, dann wird
0018 0000          ;sofort eine Messung bzw ein User-Programm gestartet.
0019 0000          ;5-Sekunden-Drücken der Starttaste nach RESET bewirkt ein
0020 0000          ;ALL-RESET
0021 0000
0022 0000          .org 00h
0023 0000 04 05    Anfa          jmp          Anfang
0024 0003          .org 03h
0025 0003 04 A3    Anfang          jmp          Inter
0026 0005 00          nop                               ;Abfrage Status
0027 0006 05    Anf          en          i
0028 0007 35          dis          tcnti
0029 0008 55          strt          t
0030 0009 26 B6          jnt0         Allreset
0031 000B 8A FF          orl          p2,#255
0032 000D 89 FF          orl          p1,#255
0033 000F 54 CB    Befehle          call         SystemRAM      ;Autostart?
0034 0011 34 AB          call         LiesRAM
0035 0013 F2 90          jb7          Start
0036 0015 34 27    BefRS          call         Empf          ;Befehlsinterpreter:
0037 0017 C6 15          jz          BefRS
0038 0019 07          dec          a
0039 001A C6 54          jz          Bef1
0040 001C 07          dec          a
0041 001D C6 56          jz          Bef2          ;Kommando "2" erkannt
0042 001F 07          dec          a
0043 0020 C6 58          jz          Bef3          ;usw.
0044 0022 07          dec          a
0045 0023 C6 5A          jz          Bef4          ;
0046 0025 07          dec          a
0047 0026 C6 5C          jz          Bef5          ;
0048 0028 07          dec          a
0049 0029 C6 64          jz          Bef6          ;
0050 002B 07          dec          a
0051 002C C6 6C          jz          Bef7          ;
0052 002E 07          dec          a
0053 002F C6 6E          jz          Bef8          ;
0054 0031 07          dec          a
0055 0032 C6 70          jz          Bef9          ;
0056 0034 07          dec          a
0057 0035 C6 72          jz          Bef10         ;
0058 0037 07          dec          a
0059 0038 C6 7A          jz          Bef11         ;
0060 003A 07          dec          a
0061 003B C6 7C          jz          Bef12         ;
0062 003D 07          dec          a
0063 003E C6 7E          jz          Bef13         ;

```

```

0064 0040 07          dec    a
0065 0041 C6 84      jz     Bef14          ;
0066 0043 07          dec    a
0067 0044 C6 86      jz     Bef15          ;
0068 0046 07          dec    a
0069 0047 C6 88      jz     Bef16          ;
0070 0049 07          dec    a
0071 004A C6 8A      jz     Bef17          ;
0072 004C 07          dec    a
0073 004D C6 8C      jz     Bef18          ;
0074 004F 07          dec    a
0075 0050 C6 8E      jz     Bef19          ;
0076 0052 04 15      jmp    BefRS          ;Ende Interpreterschleife
0077 0054
0078 0054 24 D4      Bef1   jmp    Messparameter
0079 0056 44 00      Bef2   jmp    Lesen
0080 0058 44 0E      Bef3   jmp    Schrb
0081 005A 44 1E      Bef4   jmp    Direkt
0082 005C 54 CB      Bef5   call   SystemRAM
0083 005E 23 82          mov    a,#130
0084 0060 34 8F      call   SchreibeRAM
0085 0062 64 50      jmp    Proglad
0086 0064 54 CB      Bef6   call   SystemRAM
0087 0066 23 82          mov    a,#130
0088 0068 34 8F      call   SchreibeRAM
0089 006A 64 65      jmp    Progsta
0090 006C 44 2A      Bef7   jmp    Direkt1
0091 006E 64 68      Bef8   jmp    PIB1
0092 0070 04 15      Bef9   jmp    BefRS          ;reserviert für "Start"
0093 0072 54 CB      Bef10  call   SystemRAM
0094 0074 23 81          mov    a,#129
0095 0076 34 8F      call   SchreibeRAM
0096 0078 44 33      jmp    Serie
0097 007A 64 00      Bef11  jmp    Port1
0098 007C 64 05      Bef12  jmp    LiesPort1
0099 007E 74 1F      Bef13  call   LiesT0
0100 0080 34 00      call   Senden
0101 0082 04 15      jmp    BefRS
0102 0084 64 0A      Bef14  jmp    Port2
0103 0086 64 16      Bef15  jmp    LiesPort2
0104 0088 64 26      Bef16  jmp    LiesIRAM
0105 008A 64 2E      Bef17  jmp    SchrbIRAM
0106 008C 64 36      Bef18  jmp    LiesERAM
0107 008E 64 43      Bef19  jmp    SchrbERAM
0108 0090
0109 0090
0110 0090
0111 0090          ;Autostart: der gelesene Startwert war über 128 und
0112 0090          ;wird hier ausgewertet. 129: Serienmessung, 130: Programmstart
0113 0090          ;alle anderen Werte führen zu einem Neustart über die
0114 0090          ;Interrupt-Prozedur
0115 0090
0116 0090 05      Start  en    i
0117 0091 53 7F      anl   a,#7fh
0118 0093 07          dec    a
0119 0094 C6 9B      jz     SerieStart    ;129
0120 0096 07          dec    a
0121 0097 C6 9F      jz     ProgrammStart ;130
0122 0099 04 A3      jmp    Inter
0123 009B
0124 009B 54 33      SerieStart call   Serie
0125 009D 04 00      jmp    Anfa
0126 009F 64 65      ProgrammStart jmp   Progsta
0127 00A1 04 00      jmp    Anfa
0128 00A3
0129 00A3

```

```

0130 00A3
0131 00A3 ;Interrupt-Serviceroutine, darf auch direkt angesprungen
0132 00A3 ;werden. Stellt aus jeder Situation den Ausgangszustand des
0133 00A3 ;Programms wieder her.
0134 00A3
0135 00A3 15 Inter dis i
0136 00A4 54 CB call SystemRAM
0137 00A6 23 00 mov a,#0 ;Startinfo abschalten
0138 00A8 34 8F call SchreibeRAM
0139 00AA 34 2C call Startbit ;ein Byte abwarten
0140 00AC 14 B5 call EndeInter
0141 00AE 23 00 mov a,#00
0142 00B0 D7 mov psw,a ;Stack auf Null setzen
0143 00B1 23 00 mov a,#0
0144 00B3 04 05 jmp Anfang
0145 00B5
0146 00B5 93 EndeInter retr ;Interrupt bestätigen
0147 00B6
0148 00B6 ;Auswerten der Starttaste nach RESET: 0s<t<1s: nichts,
0149 00B6 ;1s<t<3s: Messen, 3s<t<5s: Programm, 5s<t<7s: ALL-RESET
0150 00B6
0151 00B6 BA 32 Allreset mov r2,#50
0152 00B8 BB 14 Warte20 mov r3,#20
0153 00BA 34 C1 Wartel call Warten ;1ms
0154 00BC EB BA djnz r3,Wartel
0155 00BE 36 05 jt0 Anfang ;Taste frei?
0156 00C0 EA B8 djnz r2,Warte20 ;50 mal 20ms
0157 00C2 BA 64 mov r2,#100
0158 00C4 BB 14 Warte20b mov r3,#20
0159 00C6 34 C1 Wartelb call Warten ;1ms
0160 00C8 EB C6 djnz r3,Wartelb
0161 00CA 36 72 jt0 Bef10 ;Taste frei?: Serie
0162 00CC EA C4 djnz r2,Warte20b ;100 mal 20ms
0163 00CE BA 64 mov r2,#100
0164 00D0 BB 14 Warte20c mov r3,#20
0165 00D2 34 C1 Wartelc call Warten ;1ms
0166 00D4 EB D2 djnz r3,Wartelc
0167 00D6 36 64 jt0 Bef6 ;Taste frei?: Programm
0168 00D8 EA D0 djnz r2,Warte20c ;250 mal 20ms
0169 00DA 04 A3 jmp Inter ;wenn >6s gedr ü ckt
0170 00DC
0171 0100 .org 100h
0172 0100 ; rs232 Senden, verwendet r2...r3
0173 0100 ;9600 Baud, Sendesignal liegt invertiert an P27
0174 0100 ;Interrupt ist für die Zeit der Ausgabe gesperrt
0175 0100 ;Das Stopbit ist sehr kurz, weil nie zwei Bytes
0176 0100 ;sofort hintereinander ausgegeben werden
0177 0100
0178 0100 15 Senden dis i ;Interrupt gesperrt
0179 0101 BA 08 mov r2,#8 ;Zählschleife 8 Bits
0180 0103 9A 7F anl p2,#7Fh ;P27=0, Startbit
0181 0105 ; mov r3,#162 ;1200Baud
0182 0105 BB 10 mov r3,#16 ;9600 Baud
0183 0107 EB 07 Sch1 djnz r3,Sch1 ;Warteschleife Startbit
0184 0109 97 Sp1 clr c
0185 010A 67 rrc a ;Datenbit in C schieben
0186 010B F6 11 jc Sp2
0187 010D 9A 7F anl p2,#7Fh ;wenn c=0, dann P27=0
0188 010F 24 15 jmp Sp3
0189 0111 8A 80 Sp2 orl p2,#80h ;wenn c=1, dann P27=1
0190 0113 8A 80 orl p2,#80h
0191 0115 ;Sp3 mov r3,#160 ;1200Baud
0192 0115 BB 0F Sp3 mov r3,#15 ;9600 Baud
0193 0117 EB 17 Sch2 djnz r3,Sch2 ;Warteschleife Datenbit
0194 0119 EA 09 djnz r2,Sp1 ;8 mal wiederholen
0195 011B 23 00 mov a,#0

```

```

0196 011D 23 00          mov     a,#0
0197 011F 8A 80          orl     p2,#80h          ;P27=1, Stopbit
0198 0121                ;      mov     r3,#108      ;1200 Baud
0199 0121 BB 01          mov     r3,#1          ;kurzes Stopbit, 9600 Baud
0200 0123 EB 23          Sch3   djnz   r3,Sch3      ;Warteschleife Stopbit
0201 0125 05            en      i                ;Interrupt wieder frei
0202 0126 83            ret
0203 0127
0204 0127
0205 0127                ;rs232 Empfangen, verwendet r2...r4
0206 0127                ;Eingang ist INT mit Abfrage über jni. Baudrate 9600 bei 6MHz
0207 0127                ;Hardware-Interrupt ist für die Zeit der Eingabe gesperrt
0208 0127                ;Das Label "Startbit" darf auch extern angesprungen werden,
0209 0127                ;wenn ein Signal an INT erkannt wurde.
0210 0127
0211 0127 15            Empf   dis     i                ;Interrupt sperren
0212 0128 86 2C          jni     Startbit        ;Int = 0?
0213 012A 24 27          jmp     Empf            ;wenn Int = 1
0214 012C                ;Startbit   mov     r2,#225          ;1200 Baud
0215 012C BA 19          Startbit mov     r2,#25          ;9600 Baud
0216 012E EA 2E          Sch6   djnz   r2,Sch6      ;Warten 1,5 Startbit
0217 0130 BB 08          mov     r3,#8          ;Schleife 8 Bits
0218 0132 97            Sp6    clr     c                ;c=0
0219 0133 86 43          jni     Null            ;Int = 0 ?
0220 0135 00            nop
0221 0136 A7            cpl     c                ;C=1
0222 0137 2C            xch     a,r4            ;Byte von r4 holen
0223 0138 67            rrc     a                ;c in a schieben
0224 0139 2C            xch     a,r4            ;Byte wieder nach r4
0225 013A                ;      mov     r2,#162          ;1200 Baud
0226 013A BA 0F          mov     r2,#15          ;9600 Baud
0227 013C EA 3C          Sch7   djnz   r2,Sch7      ;Warten Datenbit
0228 013E EB 32          djnz   r3,Sp6          ;8 Bits empfangen?
0229 0140 FC            mov     a,r4            ;empfangenes Byte in a
0230 0141 05            en      i                ;Interrupt wieder frei
0231 0142 83            ret
0232 0143 00            Null   nop
0233 0144 97            clr     c                ;c=0
0234 0145 2C            xch     a,r4            ;Byte von r4 holen
0235 0146 67            rrc     a                ;c in a schieben
0236 0147 2C            xch     a,r4            ;Byte wieder nach r4
0237 0148                ;      mov     r2,#162          ;1200 Baud
0238 0148 BA 0F          mov     r2,#15          ;9600 Baud
0239 014A EA 4A          Sch8   djnz   r2,Sch8      ;Warten Datenbit
0240 014C EB 32          djnz   r3,Sp6          ;8 Bits empfangen?
0241 014E FC            mov     a,r4            ;empfangenes Byte in a
0242 014F 05            en      i                ;Interrupt wieder frei
0243 0150 83            ret
0244 0151
0245 0151                ;UPD7001-Messen, Übergabe des Kanals (1...4) im Akku,
0246 0151                ;Ergebnisausgabe im Akku, benötigt ca 0,5ms, verwendet r2
0247 0151                ;Datenübertragung seriell getaktet. Schiebetakt über P11 an
0248 0151                ;SCK, Steuerdaten über P12 an SI, Meßdaten über T1 von SO,
0249 0151                ;Freigabe über P10 an CS, Steuern/Ausleseumschaltung über P13
0250 0151                ;an DL. End of Conversion (EOC) wird nicht benutzt, statt
0251 0151                ;dessen feste Warteschleife
0252 0151
0253 0151 99 F0          Messen  anl     p1,#0F0h        ;P10...P13 Null
0254 0153 BA 02          mov     r2,#02          ;Zählschleife 2 Bits
0255 0155 07            dec     a                ;Byte 0...3 für Kanal 1...4
0256 0156 77            rr      a                ;Bits 0/1 nach 6/7
0257 0157 77            rr      a
0258 0158 F7            S11    rlc     a                ;Kanalbit in c
0259 0159 E6 5D          jnc     Strobe
0260 015B 89 04          orl     p1,#04          ;wenn c=1, Si = 1
0261 015D 89 02          Strobe orl     p1,#02          ;Clock = 1

```

```

0262 015F 99 FD          anl    p1,#0FDh          ;Clock = 0
0263 0161 99 FB          anl    p1,#0FBh          ;Si = 0
0264 0163 EA 58          djnz  r2,S11             ;2 mal
0265 0165 89 08          orl    p1,#008h          ;DL=1, Kanal gesendet
0266 0167 99 F7          anl    p1,#0F7h          ;DL=0
0267 0169 89 01          orl    p1,#1             ;Cs=1, Messung starten
0268 016B BA 1E          mov    r2,#30            ;ca 150 µs/6MHz
0269 016D EA 6D          S12   djnz  r2,S12             ;Warteschleife
0270 016F 99 FE          anl    p1,#0FEh          ;Cs=0
0271 0171 89 08          orl    p1,#08h           ;DL=1, Auslesen
0272 0173 BA 08          mov    r2,#08            ;Schleife 8 Bits
0273 0175 97          S13   clr    c                ;c=0
0274 0176 46 79          jnt1  T1aus              ;T1=0?
0275 0178 A7          cpl    c                ;wenn T1=1, dann c=1
0276 0179 F7          T1aus rlc    a                ;Datenbit reinschieben
0277 017A 89 02          orl    p1,#02            ;Clock=1
0278 017C 99 FD          anl    p1,#0FDh          ;Clock=0
0279 017E EA 75          djnz  r2,S13             ;acht mal
0280 0180 89 0F          orl    p1,#0Fh           ;P10...P13 = 1
0281 0182 83          ret
0282 0183
0283 0183          ;Setzen des RAM-Adreßpointers (61/62) auf den Anfang des
0284 0183          ;Datenbereichs. 0400h (2k). P27 u. P25 bleiben hochgesetzt.
0285 0183          ;P26 bleibt unverändert.
0286 0183
0287 0183 B9 3D          AnfangRAM  mov    r1,#61            ;Adresse Hi
0288 0185          ;          mov    @r1,#0C4h        ;page4, 1k
0289 0185 0A          in     a,p2
0290 0186 53 C0          anl    a,#0C0h           ;P26 u. P27 bleiben
0291 0188 43 08          orl    a,#08h            ;
0292 018A A1          mov    @r1,a             ;page8, 2k
0293 018B 19          inc    r1                ;Adresse Lo
0294 018C B1 00          mov    @r1,#0
0295 018E 83          ret
0296 018F
0297 018F          ;Ein Byte in den Datenbereich schreiben und Adreßpointer weiter
0298 018F          ;setzen. Beim Überschreiten des RAM-Bereichs wird das gesamte
0299 018F          ;Programm abgebrochen. P25 (I/O-Select) wird immer nur kurz
0300 018F          ;auf Null gesetzt, um das RAM überwiegend zu deaktivieren
0301 018F          ;verwendet r1,r2,r6
0302 018F
0303 018F B9 3D          SchreiberRAM  mov    r1,#61            ;High
0304 0191 AE          mov    r6,a              ;PUSH
0305 0192 F1          mov    a,@r1
0306 0193 3A          outl  p2,a               ;A8...A12
0307 0194 19          inc    r1                ;62, Low
0308 0195 F1          mov    a,@r1
0309 0196 A8          mov    r0,a
0310 0197 FE          mov    a,r6              ;POP
0311 0198 90          movx  @r0,a              ;speichern
0312 0199 8A 20          orl    p2,#20h           ;RAM inaktiv, Stromsparen
0313 019B F1          mov    a,@r1
0314 019C 17          inc    a                 ;Adresse erhöhen
0315 019D A1          mov    @r1,a
0316 019E 96 AA          jnz   spr1              ; Übertrag?
0317 01A0 C9          dec    r1                ;62, High
0318 01A1 F1          mov    a,@r1
0319 01A2 17          inc    a
0320 01A3 A1          mov    @r1,a             ;Adreßbyte high erhöhen
0321 01A4          ;          anl    a,#8           ;RAM-Obergrenze (2k)
erreicht?
0322 01A4 53 20          anl    a,#32            ;RAM-Obergrenze (8k)
erreicht?
0323 01A6 C6 AA          jz    spr1
0324 01A8 24 F4          jmp   Endlos            ;Neustart durch Reset erlaubt
0325 01AA 83          spr1  ret

```

```

0326 01AB
0327 01AB ;Lesen eines Bytes aus dem Daten-RAM und erhöhen des
0328 01AB ;Adreßpointers (61/62)
0329 01AB ;verwendet r0,r1,r6
0330 01AB
0331 01AB B9 3D LiesRAM mov r1,#61 ;High
0332 01AD F1 mov a,@r1
0333 01AE 3A outl p2,a ;A8...A12
0334 01AF 19 inc r1 ;62, Low
0335 01B0 F1 mov a,@r1
0336 01B1 A8 mov r0,a
0337 01B2 80 movx a,@r0 ;lesen
0338 01B3 8A 20 orl p2,#20h ;RAM inaktiv, Stromsparen
0339 01B5 AE mov r6,a
0340 01B6 F1 mov a,@r1
0341 01B7 17 inc a ;Adresse erhöhen
0342 01B8 A1 mov @r1,a
0343 01B9 96 BF jnz sprul ;Übertrag?
0344 01BB C9 dec r1 ;61, High
0345 01BC F1 mov a,@r1
0346 01BD 17 inc a
0347 01BE A1 mov @r1,a ;Adreßbyte high erhöhen
0348 01BF FE sprul mov a,r6 ;POP
0349 01C0 83 ret
0350 01C1
0351 01C1
0352 01C1 ;Wartezeit bis zur nächsten abgelaufenen Millisekunde abwarten.
0353 01C1 ;Feinabgleich nötig, da bei 6MHz keine glatte ms möglich.
0354 01C1 16 C5 Warten jtf endlms ;Timer-Überlauf?
0355 01C3 24 C1 jmp Warten
0356 01C5 AE endlms mov r6,a
0357 01C6 23 F4 mov a,#244 ;1ms/6MHz
0358 01C8 62 mov t,a
0359 01C9 23 00 mov a,#0 ;Feinabgleich
0360 01CB 23 00 mov a,#0
0361 01CD 23 00 mov a,#0
0362 01CF 23 00 mov a,#0
0363 01D1 FE mov a,r6
0364 01D2 55 str t ;Timer-Neustart
0365 01D3 83 ret
0366 01D4
0367 01D4
0368 01D4 ;Empfang und Abspeicherung von 8 Steuerbytes für die Serien-
0369 01D4 ;messung. Die Steuerbytes werfen in die Adressen 53...60
0370 01D4 ;geschrieben und zusätzlich an den Anfang des Daten-RAMs
0371 01D4 ;kopiert. Nach Ende der Übertragung wird die Startinformation
0372 01D4 ;"129" im RAM (Adr. 03A0) abgelegt und die Serienmessung
0373 01D4 ;angesprungen. Sie kann mit der Starttaste gestartet werden.
0374 01D4
0375 01D4 B9 34 Messparameter mov r1,#52
0376 01D6 BD 08 mov r5,#8 ;8 Parameter
0377 01D8 19 Mal8 inc r1
0378 01D9 34 27 call Empf
0379 01DB A1 mov @r1,a
0380 01DC ED D8 djnz r5,Mal8
0381 01DE 54 CB call SystemRAM ;Startinfo:
0382 01E0 23 81 mov a,#129 ;Serienmessung
0383 01E2 34 8F call SchreibeRAM
0384 01E4 34 83 call AnfangRAM ;Anfang 2k
0385 01E6 BA 34 mov r2,#52
0386 01E8 BB 08 mov r3,#8 ;Parameter an den
0387 01EA 1A copy inc r2 ;RAM-Anfang schreiben
0388 01EB FA mov a,r2
0389 01EC A9 mov r1,a
0390 01ED F1 mov a,@r1
0391 01EE 34 8F call SchreibeRAM

```

```

0392 01F0 EB EA          djnz   r3,copy
0393 01F2 44 33          jmp    Serie           ;Warten auf START
0394 01F4
0395 01F4 24 F4          Endlos  jmp    Endlos       ;bis Start durch Reset
0396 01F6
0397 01F6
0398 0200                .org 200h
0399 0200                ;Auslesen des Daten-RAMs. Der Hostrechner fordert jeweils
0400 0200                ;mit einer "1" ein Byte an und bricht mit "0" ab.
0401 0200
0402 0200 34 83          Lesen   call   AnfangRAM      ;Anfang 2k
0403 0202 34 27          Leseloop call  Empf             ;"1" oder "0" erwartet
0404 0204 C6 0C          jz      Leseende       ;Ende, wenn "0"
0405 0206 34 AB          call   LiesRAM         ;auslesen
0406 0208 34 00          call   Senden          ;Datenbyte zurück
0407 020A 44 02          jmp    Leseloop
0408 020C 04 15          Leseende jmp   BefRS           ;nächstes Kommando
0409 020E
0410 020E
0411 020E                ;Beschreiben des Daten-RAMs. Die Zelle fordert jeweils ein
0412 020E                ;Byte durch Senden eines beliebigen Bytes an.
0413 020E                ;Der Hostrechner antwortet mit "1" und dem Datenbyte oder
0414 020E                ;mit "0" für "Ende"
0415 020E
0416 020E 34 83          Schrb   call   AnfangRAM      ;Anfang 2k
0417 0210 34 00          Schrloop call  Senden           ;Anforderung
0418 0212 34 27          call   Empf             ;"0" oder "1" erwartet
0419 0214 C6 1C          jz      Schrende       ;Ende, wenn "0"
0420 0216 34 27          call   Empf             ;Datenbyte
0421 0218 34 8F          call   SchreibeRAM     ;abspeichern
0422 021A 44 10          jmp    Schrloop
0423 021C 04 15          Schrende jmp   BefRS
0424 021E
0425 021E
0426 021E                ;Direkt Messung über den A/D-Wandler ohne Abspeichern.
0427 021E                ;Es wird die Kanalnummer empfangen und das Ergebnis gesendet.
0428 021E                ;Abbruch, wenn die Kanalnummer "0" erscheint
0429 021E
0430 021E 34 27          Direkt  call   Empf             ;Kanalnummer
0431 0220 C6 28          jz      DirektEnde     ;Ende, wenn "0"
0432 0222 34 51          call   Messen
0433 0224 34 00          call   Senden
0434 0226 44 1E          jmp    Direkt
0435 0228 04 15          DirektEnde jmp   BefRS
0436 022A
0437 022A                ;Eine einzelne direkt Messung. Wie die vorige Routine, aber
0438 022A                ;ohne Wiederholungsschleife
0439 022A
0440 022A 34 27          Direkt1  call   Empf
0441 022C 34 51          call   Messen
0442 022E 34 00          call   Senden
0443 0230 27          clr    a
0444 0231 04 15          jmp    BefRS
0445 0233
0446 0233                ;Serienmessung für die eingestellten Kanäle und Intervallzeiten
0447 0233                ;Die Anzahl wird für jede Serie in den aktuellen Zähler (51/52)
0448 0233                ;kopiert, den das Unterprogramm "Block" verwendet.
0449 0233
0450 0233 34 83          Serie   call   AnfangRAM
0451 0235 BA 34          mov    r2,#52
0452 0237 BB 08          mov    r3,#8
0453 0239 34 AB          copyback call  LiesRAM
0454 023B AE          mov    r6,a
0455 023C 1A          inc    r2              ;Infos vom RAM-Anfang
0456 023D FA          mov    a,r2           ;lesen
0457 023E A9          mov    r1,a

```

```

0458 023F FE          mov     a,r6
0459 0240 A1          mov     @r1,a
0460 0241 EB 39      djnz   r3,copyback
0461 0243 54 D7          call  WarteT0          ;Startsignal abwarten
0462 0245 B8 36      NextSerie  mov     r0,#54          ;Anzahl/Serie 53/54 n. 51/52
0463 0247 B9 34          mov     r1,#52
0464 0249 F0          mov     a,@r0
0465 024A 96 4D      jnz    Nicht0
0466 024C 17          inc     a              ;wenn Lowbyte = 0
0467 024D A1      Nicht0    mov     @r1,a
0468 024E C9          dec     r1
0469 024F C8          dec     r0
0470 0250 F0          mov     a,@r0
0471 0251 17          inc     a
0472 0252 A1          mov     @r1,a
0473 0253 05          en     i              ;Interrupt frei
0474 0254 23 F8      mov     a,#248        ;Timer-Start
0475 0256 62          mov     t,a
0476 0257 55          strt   t
0477 0258 B9 37      Messloop  mov     r1,#55        ;1. aktiver Kanal
0478 025A F1          mov     a,@r1
0479 025B C6 63      jz     Nr2
0480 025D 34 51      call   Messen
0481 025F 34 8F      call   SchreibeRAM    ;Meßwert abspeichern
0482 0261 34 C1      call   Warten
0483 0263 B9 38      Nr2      mov     r1,#56        ;2. aktiver Kanal
0484 0265 F1          mov     a,@r1
0485 0266 C6 6E      jz     Nr3
0486 0268 34 51      call   Messen
0487 026A 34 8F      call   SchreibeRAM    ;Meßwert abspeichern
0488 026C 34 C1      call   Warten
0489 026E B9 39      Nr3      mov     r1,#57        ;3. aktiver Kanal
0490 0270 F1          mov     a,@r1
0491 0271 C6 79      jz     Nr4
0492 0273 34 51      call   Messen
0493 0275 34 8F      call   SchreibeRAM    ;Meßwert abspeichern
0494 0277 34 C1      call   Warten
0495 0279 B9 3A      Nr4      mov     r1,#58        ;4. aktiver Kanal
0496 027B F1          mov     a,@r1
0497 027C C6 84      jz     Nr5
0498 027E 34 51      call   Messen
0499 0280 34 8F      call   SchreibeRAM    ;Meßwert abspeichern
0500 0282 34 C1      call   Warten
0501 0284 44 A4      Nr5      jmp    Block
0502 0286 B9 3B      BlockOK  mov     r1,#59        ;Adr 59: Wait-high
0503 0288 F1          mov     a,@r1
0504 0289 AD          mov     r5,a          ;warten high
0505 028A 1D          inc     r5
0506 028B ED 8F      spr2    djnz   r5,spr3
0507 028D 44 97      jmp    spr4
0508 028F BC 00      spr3    mov     r4,#0
0509 0291 34 C1      wait1   call   Warten
0510 0293 EC 91      djnz   r4,wait1
0511 0295 44 8B      jmp    spr2
0512 0297 B9 3C      spr4    mov     r1,#60
0513 0299 F1          mov     a,@r1
0514 029A AC          mov     r4,a
0515 029B 1C          inc     r4
0516 029C EC A0      spr5    djnz   r4,wait2      ;Warteschleife
0517 029E 44 58      jmp    Messloop
0518 02A0 34 C1      wait2   call   Warten
0519 02A2 44 9C      jmp    spr5
0520 02A4
0521 02A4          ;Zählt die Messungen mit und geht in erneute Meßbereitschft,
0522 02A4          ;wenn eine Serie abgelaufen ist.
0523 02A4

```

```

0524 02A4 B9 34      Block      mov     r1,#52
0525 02A6 F1          mov     a,@r1
0526 02A7 07          dec     a
0527 02A8 A1          mov     @r1,a
0528 02A9 96 C1          jnz    B11
0529 02AB C9          dec     r1
0530 02AC F1          mov     a,@r1
0531 02AD 07          dec     a
0532 02AE A1          mov     @r1,a
0533 02AF 96 C1          jnz    B11
0534 02B1 B8 35          mov     r0,#53
0535 02B3 F0          mov     a,@r0
0536 02B4 17          inc     a
0537 02B5 A1          mov     @r1,a
0538 02B6 19          inc     r1
0539 02B7 18          inc     r0
0540 02B8 F0          mov     a,@r0
0541 02B9 96 BC          jnz    NoNull
0542 02BB 17          inc     a
0543 02BC A1          NoNull  mov     @r1,a
0544 02BD 54 D7          Endlos2 call   WarteT0      ;bis weiter durch Start
0545 02BF 44 45          jmp     NextSerie
0546 02C1 44 86          B11     jmp     BlockOK
0547 02C3
0548 02C3              ;Endlosschleife mit abgeschalteter Startinformation
0549 02C3
0550 02C3 54 CB          Stop    call   SystemRAM   ;Startinfo = 0
0551 02C5 23 00          mov     a,#0
0552 02C7 34 8F          call   SchreibeRAM
0553 02C9 44 C3          jmp     Stop
0554 02CB
0555 02CB              ;Stellt die Adre pointer auf 03A0 ein.
0556 02CB              ;L sst P26 unver ndert
0557 02CB
0558 02CB B9 3D          SystemRAM mov   r1,#61      ;Adresse Hi
0559 02CD 0A          in     a,p2
0560 02CE 53 C3          anl    a,#0C3h
0561 02D0 43 03          orl    a,#03h
0562 02D2 A1          mov     @r1,a      ;page3
0563 02D3 19          inc     r1          ;Adresse Lo
0564 02D4 B1 64          mov     @r1,#0A0   ;Adresse 03A0
0565 02D6 83          ret
0566 02D7
0567 02D7              ;Wartet auf die Taste "Start" oder auf ein Byte "9" vom
0568 02D7              ;Hostrechner. Wird eine "0" empfangen, erfolgt ein Sprung
0569 02D7              ;zur Interrupt-Routine und damit ein Neustart.
0570 02D7
0571 02D7 15          WarteT0 dis   i          ;Interrupt abschalten
0572 02D8 86 E2          jni    RSweiter    ;Startbit?
0573 02DA 26 D7          jnt0   WarteT0     ;wenn noch gedr ckt
0574 02DC 86 E2          T0high jni    RSweiter    ;Startbit?
0575 02DE 36 DC          jt0    T0high      ;wenn nicht gedr ckt
0576 02E0 05          en     i          ;Interrupt wieder frei
0577 02E1 83          ret
0578 02E2 34 2C          RSweiter call   Startbit   ;Byte empfangen
0579 02E4 C6 E7          jz     Schluss
0580 02E6 83          ret
0581 02E7 04 A3          Schluss jmp    Inter
0582 02E9
0583 02E9
0584 0300              .org 300h
0585 0300              ;Ausgabe eines Bytes vom Hostrechner an Port 1
0586 0300
0587 0300 34 27          Port1  call   Empf
0588 0302 39          outl   p1,a
0589 0303 04 15          jmp    BefRS

```

```

0590 0305
0591 0305 ;Auslesen des Ports 1 durch den Hostrechner
0592 0305
0593 0305 09 LiesPort1 in a,p1
0594 0306 34 00 call Senden
0595 0308 04 15 jmp BefRS
0596 030A
0597 030A ;Setzen (1) oder Rücksetzen (0) von P26
0598 030A
0599 030A 34 27 Port2 call Empf
0600 030C 12 12 jnb0 Port2an ;"1"
0601 030E 9A BF anl p2,#0BFh ;wenn "0"
0602 0310 04 15 jmp BefRS
0603 0312 8A 40 Port2an orl p2,#40h
0604 0314 04 15 jmp BefRS
0605 0316
0606 0316 ;Auslesen der Leitung P26 durch den Hostrechner
0607 0316 ;Zurückgesendet wird "0" oder "1"
0608 0316
0609 0316 0A LiesPort2 in a,p2
0610 0317 E7 rl a ;Bit verschieben
0611 0318 E7 rl a
0612 0319 53 01 anl a,#1
0613 031B 34 00 call Senden
0614 031D 04 15 jmp BefRS
0615 031F
0616 031F
0617 031F ;Auslesen des Zustands der Starttaste durch den Hostrechner
0618 031F
0619 031F 23 00 LiesT0 mov a,#0
0620 0321 26 25 jnt0 T0Ende
0621 0323 23 01 mov a,#1
0622 0325 83 T0Ende ret
0623 0326
0624 0326
0625 0326 ;Auslesen einer beliebigen Adressen des internen RAMs durch
0626 0326 ;den Hostrechner.
0627 0326
0628 0326 34 27 LiesIRAM call Empf ;Adresse
0629 0328 A8 mov r0,a
0630 0329 F0 mov a,@r0
0631 032A 34 00 call Senden ;Datum
0632 032C 04 15 jmp BefRS
0633 032E
0634 032E ;Beschreiben einer beliebigen Adressen des internen RAMs durch
0635 032E ;den Hostrechner.
0636 032E
0637 032E 34 27 SchrbIRAM call Empf ;Adresse
0638 0330 A8 mov r0,a
0639 0331 34 27 call Empf ;Datum
0640 0333 A0 mov @r0,a
0641 0334 04 15 jmp BefRS
0642 0336
0643 0336 ;Auslesen einer beliebigen Adressen des externen RAMs durch
0644 0336 ;den Hostrechner.
0645 0336
0646 0336 34 27 LiesERAM call Empf ;Adresse, high
0647 0338 43 C0 orl a,#0C0h ;P26...P27 high
0648 033A 3A outl p2,a
0649 033B 34 27 call Empf ;Adresse, low
0650 033D A8 mov r0,a
0651 033E 80 movx a,@r0
0652 033F 34 00 call Senden ;Datum
0653 0341 04 15 jmp BefRS
0654 0343
0655 0343 ;Beschreiben einer beliebigen Adressen des externen RAMs durch

```

```

0656 0343 ;den Hostrechner.
0657 0343
0658 0343 34 27 SchrbERAM call Empf ;Adresse, high
0659 0345 43 C0 orl a,#0C0h
0660 0347 3A outl p2,a
0661 0348 34 27 call Empf ;Adresse, low
0662 034A A8 mov r0,a
0663 034B 34 27 call Empf ;Datum
0664 034D 90 movx @r0,a
0665 034E 04 15 jmp BefRS
0666 0350
0667 0350 ;Laden eines User-Programms: Jedes Byte wird erst durch
0668 0350 ;ein beliebiges Byte der Zelle angefordert. Der
0669 0350 ;Hostrechner sendet danach jeweils eine "1" und das
0670 0350 ;Programmbyte. Die Programmbytes werden ab C400 (1k)
0671 0350 ;abgespeichert. Beendigung der Übertragung erfolgt
0672 0350 ;durch ein Steuerbyte "0" des Hostrechners.
0673 0350
0674 0350 B9 3D Proglad mov r1,#61 ;Adresse Hi
0675 0352 B1 C4 mov @r1,#0C4h ;page4, 1k
0676 0354 19 inc r1 ;Adresse Lo
0677 0355 B1 00 mov @r1,#0
0678 0357 34 00 Progloop call Senden ;Anforderung
0679 0359 34 27 call Empf ;"1" oder "0" erwartet
0680 035B C6 63 jz Progladende ;Ende, wenn "0"
0681 035D 34 27 call Empf ;Programmbyte holen
0682 035F 34 8F call SchreibeRAM ;Wert ins RAM
0683 0361 64 57 jmp Progloop
0684 0363 04 15 Progladende jmp BefRS ;Warte Befehl oder Reset
0685 0365
0686 0365 ;Sprung zum Userprogramm ab 1k, Interrupt freigegeben, so daß
0687 0365 ;es mit einem Interrupt abgeschaltet werden kann
0688 0365
0689 0365 05 Progsta en i
0690 0366 84 00 jmp 400h
0691 0368
0692 0368 ;Paralleler Interface-Bus:
0693 0368 ;Zufriff auf die Adressen 0...127 des I/O-Bus
0694 0368 ;Auslesen, wenn Bit 7 gesetzt ist, sonst Schreiben
0695 0368 ;Ende nach einer I/O-Aktion
0696 0368
0697 0368 8A 20 PIB1 orl p2,#20h ;I/O-Bereich aktiv
0698 036A 34 27 call Empf
0699 036C F2 75 jb7 Read1 ;Lesen
0700 036E A8 mov r0,a ;Adresse nach r0
0701 036F 34 27 call Empf ;Datum holen
0702 0371 90 movx @r0,a ;zum I/O-Bus
0703 0372 27 clr a
0704 0373 04 15 jmp BefRS ;neues Kommando
0705 0375 53 7F Read1 anl a,#7Fh ;Bit 7 zurücksetzen
0706 0377 A8 mov r0,a ;Adresse nach R0
0707 0378 80 movx a,@r0 ;I/O-Adresse auslesen
0708 0379 34 00 call Senden ;Datum zurücksenden
0709 037B 27 clr a
0710 037C 04 15 jmp BefRS ;neues Kommando
0711 037E
0712 037E .end

```

```

ADDR 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
-----
0000 04 05 FF 04 A3 00 05 35 55 26 B6 8A FF 89 FF 54
0010 CB 34 AB F2 90 34 27 C6 15 07 C6 54 07 C6 56 07
0020 C6 58 07 C6 5A 07 C6 5C 07 C6 64 07 C6 6C 07 C6
0030 6E 07 C6 70 07 C6 72 07 C6 7A 07 C6 7C 07 C6 7E
0040 07 C6 84 07 C6 86 07 C6 88 07 C6 8A 07 C6 8C 07
0050 C6 8E 04 15 24 D4 44 00 44 0E 44 1E 54 CB 23 82

```

0060 34 8F 64 50 54 CB 23 82 34 8F 64 65 44 2A 64 68
0070 04 15 54 CB 23 81 34 8F 44 33 64 00 64 05 74 1F
0080 34 00 04 15 64 0A 64 16 64 26 64 2E 64 36 64 43
0090 05 53 7F 07 C6 9B 07 C6 9F 04 A3 54 33 04 00 64
00A0 65 04 00 15 54 CB 23 00 34 8F 34 2C 14 B5 23 00
00B0 D7 23 00 04 05 93 BA 32 BB 14 34 C1 EB BA 36 05
00C0 EA B8 BA 64 BB 14 34 C1 EB C6 36 72 EA C4 BA 64
00D0 BB 14 34 C1 EB D2 36 64 EA D0 04 A3 FF FF FF FF
00E0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00F0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0100 15 BA 08 9A 7F BB 10 EB 07 97 67 F6 11 9A 7F 24
0110 15 8A 80 8A 80 BB 0F EB 17 EA 09 23 00 23 00 8A
0120 80 BB 01 EB 23 05 83 15 86 2C 24 27 BA 19 EA 2E
0130 BB 08 97 86 43 00 A7 2C 67 2C BA 0F EA 3C EB 32
0140 FC 05 83 00 97 2C 67 2C BA 0F EA 4A EB 32 FC 05
0150 83 99 F0 BA 02 07 77 77 F7 E6 5D 89 04 89 02 99
0160 FD 99 FB EA 58 89 08 99 F7 89 01 BA 1E EA 6D 99
0170 FE 89 08 BA 08 97 46 79 A7 F7 89 02 99 FD EA 75
0180 89 0F 83 B9 3D 0A 53 C0 43 08 A1 19 B1 00 83 B9
0190 3D AE F1 3A 19 F1 A8 FE 90 8A 20 F1 17 A1 96 AA
01A0 C9 F1 17 A1 53 20 C6 AA 24 F4 83 B9 3D F1 3A 19
01B0 F1 A8 80 8A 20 AE F1 17 A1 96 BF C9 F1 17 A1 FE
01C0 83 16 C5 24 C1 AE 23 F4 62 23 00 23 00 23 00 23
01D0 00 FE 55 83 B9 34 BD 08 19 34 27 A1 ED D8 54 CB
01E0 23 81 34 8F 34 83 BA 34 BB 08 1A FA A9 F1 34 8F
01F0 EB EA 44 33 24 F4 FF FF FF FF FF FF FF FF FF
0200 34 83 34 27 C6 0C 34 AB 34 00 44 02 04 15 34 83
0210 34 00 34 27 C6 1C 34 27 34 8F 44 10 04 15 34 27
0220 C6 28 34 51 34 00 44 1E 04 15 34 27 34 51 34 00
0230 27 04 15 34 83 BA 34 BB 08 34 AB AE 1A FA A9 FE
0240 A1 EB 39 54 D7 B8 36 B9 34 F0 96 4D 17 A1 C9 C8
0250 F0 17 A1 05 23 F8 62 55 B9 37 F1 C6 63 34 51 34
0260 8F 34 C1 B9 38 F1 C6 6E 34 51 34 8F 34 C1 B9 39
0270 F1 C6 79 34 51 34 8F 34 C1 B9 3A F1 C6 84 34 51
0280 34 8F 34 C1 44 A4 B9 3B F1 AD 1D ED 8F 44 97 BC
0290 00 34 C1 EC 91 44 8B B9 3C F1 AC 1C EC A0 44 58
02A0 34 C1 44 9C B9 34 F1 07 A1 96 C1 C9 F1 07 A1 96
02B0 C1 B8 35 F0 17 A1 19 18 F0 96 BC 17 A1 54 D7 44
02C0 45 44 86 54 CB 23 00 34 8F 44 C3 B9 3D 0A 53 C3
02D0 43 03 A1 19 B1 64 83 15 86 E2 26 D7 86 E2 36 DC
02E0 05 83 34 2C C6 E7 83 04 A3 FF FF FF FF FF FF FF
02F0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0300 34 27 39 04 15 09 34 00 04 15 34 27 12 12 9A BF
0310 04 15 8A 40 04 15 0A E7 E7 53 01 34 00 04 15 23
0320 00 26 25 23 01 83 34 27 A8 F0 34 00 04 15 34 27
0330 A8 34 27 A0 04 15 34 27 43 C0 3A 34 27 A8 80 34
0340 00 04 15 34 27 43 C0 3A 34 27 A8 34 27 90 04 15
0350 B9 3D B1 C4 19 B1 00 34 00 34 27 C6 63 34 27 34
0360 8F 64 57 04 15 05 84 00 8A 20 34 27 F2 75 A8 34
0370 27 90 27 04 15 53 7F A8 80 34 00 27 04 15 FF FF

PROGRAM Simpel_2;

```

USES    Crt;

CONST  Copyright = '(c) Burkhard Kainka';
       Version   = '1.0';
       Update    = '17.4.91';

VAR
  ArbeitsDatei : Text;
  DateiName    : STRING;
  Ausgabefile  : Text;
  C            : CHAR;

Var
  Befehl       : ARRAY[1..500] OF STRING[20];
  Opcode       : ARRAY[1..500] OF STRING[20];
  HexCode      : ARRAY[0..1024] OF STRING[2];
  Adr          : Integer;
  MaxBef       : Integer;
  MaxInit      : Integer;
  Zeile        : Integer;
  Fehler       : Integer;
  RuecksprungAdresse : Array[0..9] of Integer;
  EndeSprungadresse : Array[0..9] of INTEGER;
  Schleife_Nr  : Byte;
  Sprungadresse : Integer;
  Wort1, Wort2 : string[20];
  Fehlermeldung : string[45];

PROCEDURE LadeTabelle;
  VAR   i      : LongInt;
        B, j   : BYTE;
        Bef, Opc, Spc : Boolean;
        Nr     : INTEGER;
        SeitenZahl : INTEGER;
        TabellenDatei : FILE OF BYTE;
  BEGIN
    Assign (TabelleDatei, 'simpl.tab');
    (*$I-*) Reset (TabelleDatei); (*$I+*)
    IF IOResult = 0
    THEN BEGIN
      Nr:=1; Bef:= true;Opc:=false;
      Befehl[Nr]:='';Opcode[Nr]:='';
      FOR i := 1 TO FileSize(TabelleDatei) DO BEGIN
        Read (TabelleDatei,B);
        If (Bef and (b>32)) then Befehl[Nr]:=concat
          (Befehl[Nr],upcase(Chr(b)));
        If (Bef and (B=32)) then Bef:=false;
        if ((Not Bef) and (B>32)) then Opc:=true;
        if (Opc and (b>32)) then Opcode[Nr]:=concat
          (Opcode[Nr],upcase(Chr(b)));
        if (Opc and (b=13)) then begin

```

```

        Bef := true; Opc := false; Nr := Nr +1;
        Befehl[Nr]:= ''; Opcode[Nr]:= ''; END
    END;
    END;
    Close (TabellenDatei);
    Nr := Nr-1; MaxBef := Nr;
    writeln;
END (* LadeTabelle *);

PROCEDURE Byte_Hex (Zahl:Integer; Var Hex : STRING);
Var Low, High : INTEGER;
Begin
    If ((Zahl<0) Or (Zahl>255)) then Fehler :=7;
    High := Zahl div 16;
    Low := Zahl mod 16;
    High := High+48; if (High>57) then High:=High+7;
    Low := Low+48; if (Low>57) then Low:=Low+7;
    Hex := concat (chr(High),chr(Low));
End (* of Byte_Hex *);

PROCEDURE Hex_Byte (Hex : STRING; Var Zahl:Integer);
Var N, Low, High : INTEGER;
    Ch : Char;
Begin
    Ch := Hex[1];
    If NOT (Ch in ['0'..'9','A'..'F']) then Fehler :=7;
    High := Integer (Ch);
    If High >64 then High := High-7 ;
    High := High -48;
    Ch := Hex[2];
    If NOT (Ch in ['0'..'9','A'..'F']) then Fehler :=7;
    Low := Integer (Ch);
    If Low >64 then Low := Low-7 ;
    Low := Low -48;
    Zahl := 16*High + Low;
End (* of Hex_Byte *);

PROCEDURE Prozedur_Anfang;
Var Low, High, I, N : INTEGER;
    Hex : String;
Begin
    i:=0;
    FOR N := 1 to MaxBef DO BEGIN
        if Wort2 = Befehl[n] then i:=n;
    END;
    if i>0 then Fehler:=10;
    MaxBef := MaxBef + 1;
    Befehl [MaxBef] := Wort2;
    High := Adr div 256;
    Low := Adr mod 256;
    Byte_Hex (Low,HEX);
    Opcode [MaxBef] := concat ('PA94',Hex);
    if High = 1 then Opcode [MaxBef] := concat ('PAB4',Hex);
    if High = 2 then Opcode [MaxBef] := concat ('PAD4',Hex);
    if High = 3 then Opcode [MaxBef] := concat ('PAF4',Hex);
End;

```

```

Procedure Zahl (Wort2: String; var Wert: Integer);
var Laenge, Position, N      : Integer;
    Ch      : Char;
    Wort    : String;
Begin;
  Laenge := length (Wort2);
  Ch := Wort2[Laenge];
  If Ch in ['0'..'9'] then begin
    Val (Wort2, Wert, n); end;
  If Ch = 'H' then begin
    Wort := copy (Wort2,1,(Laenge-1));
    If (Laenge > 3) then Fehler := 7;
    If (length(Wort) = 1) then wort := concat ('0', Wort);
    Hex_Byte (Wort,Wert); end;
  If Ch = 'B'then begin;
    n := 1; wert := 0;
    Position := Laenge;
    while Position > 1 do begin
      Position := Position -1;
      if (Wort2[Position] = '1') then wert := wert + n;
      if (Wort2[Position] = 'I') then wert := wert + n;
      if NOT (Wort2[Position] in ['1','0','I','O']) then fehler := 7;
      n := n*2;
    end; end;
end;

PROCEDURE Parameter_Option;
var Wert : Integer;
    Hex : String;
begin
  if (length(Wort2) > 0) then begin
    if (Wort2[1] IN ['0'..'9','A'..'F','a'..'f','o','O','i','I']) then begin
      Zahl(Wort2,Wert);
      Byte_hex (Wert, Hex);
      Hexcode[Adr] := '23'; write ('23');
      Adr := Adr+1;
      Hexcode[Adr] := Hex; write (Hex);
      Adr := Adr+1;
    end;
  end;
end;

PROCEDURE Anfang;
Var N,I      : Integer;
    Hex,Hex1 : STRING;
Begin
  if Hi(Adr)=0 then Hexcode[0] := '84';
  if Hi(Adr)=1 then Hexcode[0] := 'A4';
  if Hi(Adr)=2 then Hexcode[0] := 'C4';
  if Hi(Adr)=3 then Hexcode[0] := 'F4';
  Byte_Hex (Lo(Adr),Hex);
  Hexcode[1] := Hex;
END;

Procedure Code;
var Hex      : String;

```

```

    N, Laenge, Wert    : Integer;

begin
  Laenge := length (Wort2);
  n:= 1;
  while n < Laenge do begin
    while (Wort2[n]=' ') do n:=n+1;
    Hex := copy (Wort2,n,2);
    n := n+2;
    Hex_Byte (Hex,Wert);
    Byte_Hex (Wert,Hex);
    Hexcode[Adr]:= Hex; write (Hex);
    Adr := Adr +1;
  end;
end;

Procedure Aufruf_Einsetzen;
var I, N    : Integer;
    Hex     : String;
Begin
  i:=0;
  FOR N := 1 to MaxBef DO BEGIN
    if Wort2 = Befehl[n] then i:=n;
  END;
  if i=0 then Fehler:=3 else begin
    n:=1;
    REPEAT
      Hex := COPY (Opcode[i],n,2);
      if not (Hex = '') then begin
        If ((Hex>'FF') AND NOT (Hex='PA')) then begin
          Fehler := 2;
          End;
        if NOT (Hex='PA') then begin
          Hexcode[Adr] := Hex; write (Hex);
          Adr := Adr+1;
        end;
        n:= n+2;
      end;
    UNTIL Hex = '';
  end;
end;

Procedure VorwaertsAdresse (Var Adressbyte : String);
var I, N, Adresse  : Integer;
    Hex            : String;
begin
  I := 0;
  FOR N := 1 to MaxBef DO BEGIN
    if Wort2 = Befehl[n] then i:=n;
  END;
  if i=0 then Fehler :=3 else begin
    n:=1;
    REPEAT
      Hex := COPY (Opcode[i],n,2);
      if not (Hex = '') then begin
        If ((hex>'FF') AND NOT (Hex='PA')) then Fehler:=2;
        n:= n+2;
      end;
    UNTIL Hex = '';
  end;
end;

```

```

    end;
    UNTIL Hex = '';
    Hex := Copy (Opcode[i],1,2);
    if Hex = 'PA' then n := n-2;
    Adresse := Adr + ((n+1) div 2);
    Byte_Hex (Lo(Adresse),Adressbyte);
    end;
end;

```

```

Procedure Ruecksprung_merken;
begin;
  Schleife_Nr := Schleife_Nr +1;
  If (Schleife_Nr>8) then Fehler := 4;
  Ruecksprungadresse[Schleife_Nr] := Adr;

```

```
end;
```

```

Procedure Ruecksprung_einsetzen;
var Hex : String;
begin;
  If (Schleife_Nr<1) then Fehler := 5;
  If Hi(RuecksprungAdresse[Schleife_Nr])=0 then begin
    Hexcode[Adr] := '84' ; write ('84'); end;
  If Hi(RuecksprungAdresse[Schleife_Nr])=1 then begin
    Hexcode[Adr] := 'A4' ; write ('A4'); end;
  If Hi(RuecksprungAdresse[Schleife_Nr])=2 then begin
    Hexcode[Adr] := 'C4' ; write ('C4'); end;
  If Hi(RuecksprungAdresse[Schleife_Nr])=3 then begin
    Hexcode[Adr] := 'E4' ; write ('E4'); end;
  Adr := Adr+1;
  Byte_Hex (Lo(RuecksprungAdresse[Schleife_Nr]),Hex);
  Hexcode[Adr] := Hex; write (Hex);
  Adr := Adr+1;
end;

```

```

Procedure Zieladresse_Einsetzen;
var Hex : String;
    SprungCodeAdr : Integer;
begin;
  Byte_Hex (Lo(Adr),Hex);
  SprungCodeAdr := EndeSprungAdresse[Schleife_Nr];
  Hexcode[SprungCodeAdr] := Hex; write (' 00:',Hex);
  SprungCodeAdr := SprungCodeAdr - 1;
  if Hi(Adr)=0 then Hexcode[SprungCodeAdr] := '84';
  if Hi(Adr)=1 then Hexcode[SprungCodeAdr] := 'A4';
  if Hi(Adr)=2 then Hexcode[SprungCodeAdr] := 'C4';
  if Hi(Adr)=3 then Hexcode[SprungCodeAdr] := 'F4';
  If (Schleife_Nr=0) then Schleife_Nr := 1;
  Schleife_Nr := Schleife_Nr - 1;
end;

```

```

PROCEDURE Sonder (Hex: String; Var Hexout:String);
  Var Wert, N : Integer;
  Begin
    If Hex = 'XX' then begin
      If (Wort2='') then Fehler := 8;

```

```

    Zahl (Wort2, Wert);
    Byte_Hex (Wert,Hexout);
end;
If Hex = 'RR' then begin
    Val (Wort2, Wert, n);
    Wert := Wert+32;
    Byte_Hex (Wert,Hexout);
end;
If Hex = 'HH' then begin
    Sprungadresse := Adr;
    Hexout := '**';
end;
If Hex = 'SS' then begin
    Byte_Hex (lo(Sprungadresse),HexOut);
end;
If Hex = 'PR' then begin
    Aufruf_einsetzen;
    Hexout := '**';
end;
If Hex = 'PD' then begin
    Prozedur_Anfang;
    Hexout := '**';
end;
If Hex = 'PA' then begin
    Parameter_Option;
    Hexout := '**';
end;
If Hex = 'HX' then begin
    Code;
    Hexout := '**';
end;
If Hex = 'JP' then begin
    If Hi(Adr)=0 then Hexout := '84';
    If Hi(Adr)=1 then Hexout := 'A4';
    If Hi(Adr)=2 then Hexout := 'C4';
    If Hi(Adr)=3 then Hexout := 'E4';
end;
If Hex = 'NN' then Vorwaertsadresse (Hexout);
If Hex = 'HS' then begin
    Ruecksprung_merken; Hexout := '**';
end;
If Hex = 'JS' then begin
    Ruecksprung_einsetzen; Hexout := '**'; end;
If Hex = 'J3' then Byte_Hex (Lo(Adr+3),Hexout);
if Hex = 'JZ' then begin
    EndeSprungadresse[Schleife_Nr] := Adr-1; Hexout := '**'; end;
if Hex = 'JE' then begin;
    Zieladresse_Einsetzen; Hexout := '**'; end;
if Hex = 'SO' then begin
    if (Schleife_Nr>0) then Fehler :=9; Hexout := '**'; end;
if Hex = 'HA' then begin
    Anfang; Hexout := '**'; end;
End;

```

```
PROCEDURE Translate;
```

```

    Var  N,I           : Integer;
         Hex,Hex1     : STRING;
         Flag         : BOOLEAN;

```

```

        Teilwort      :String;
Begin
If Lo(Adr) > 239 then
  repeat
    Hexcode[Adr] := '00';
    write ('00');
    Adr := Adr+1;
  until Lo(Adr)=0;
Flag := true;
Teilwort := copy (Wort1,1,1);
if Teilwort = ';' then Flag := false;
if Wort1 = '' then Flag := false;
If Flag then Begin
  i:=0;
  FOR N := 1 to MaxBef DO BEGIN
    if Wort1 = Befehl[n] then begin
      i:=n; N:= MaxBef; end;
  END;
  if i=0 then Fehler := 1 else begin
    n:=1;
    REPEAT
      Hex := COPY (Opcode[i],n,2);
      if not (Hex = '') then begin
        if Hex > 'FF' then begin
          Sonder(Hex,Hex1);
          Hex := Hex1; End;
        n:= n+2;
        if NOT (Hex = '**') then begin
          Hexcode[Adr] := Hex;
          Adr := Adr+1;
          write (Hex);
        end;
      end;
    UNTIL Hex = '';
  END;
END;
Wort1:=''; Wort2:='';
END; (* Translate *)

PROCEDURE Fehlermelden;
var Fehlernr, Zeilennr      :   String;
Begin
  if Fehler>0 then begin
    Fehlermeldung := ('Fehler ');
    str (Fehler,Fehlernr);
    Fehlermeldung := concat (Fehlermeldung,Fehlernr);
    Fehlermeldung := concat (Fehlermeldung,' in Zeile ');
    str (Zeile,Zeilennr);
    Fehlermeldung := concat (Fehlermeldung,Zeilennr);
    Fehlermeldung := concat (Fehlermeldung,' :');
    Case Fehler of
      1 : Fehlermeldung := concat (Fehlermeldung,'Befehl unbekannt ');
      2 : Fehlermeldung := concat (Fehlermeldung,'Befehl paßt nicht ');
      3 : Fehlermeldung := concat (Fehlermeldung,'Prozedur unbekannt ');
      4 : Fehlermeldung := concat (Fehlermeldung,'zu viele Schleifen ');
      5 : Fehlermeldung := concat (Fehlermeldung,'keine Schleife geöffnet ');
      6 : Fehlermeldung := concat (Fehlermeldung,'Befehl unbekannt ');
      7 : Fehlermeldung := concat (Fehlermeldung,'Unerlaubter Wert ');
      8 : Fehlermeldung := concat (Fehlermeldung,'Wert fehlt ');
    end;
  end;
end;

```

```

    9 : Fehlermeldung := concat (Fehlermeldung, 'Schleife noch geöffnet ');
    10: Fehlermeldung := concat (Fehlermeldung, 'Name existiert bereits ');
end;
write (Fehlermeldung);
end;
end;

```

```
PROCEDURE Anhalten;
```

```

Begin
  if Keypressed then begin
    C := ReadKey;
    repeat until KeyPressed;
    C := ReadKey;
  END;END;

```

```
PROCEDURE Compiler;
```

```

var Zustand      : INTEGER;
    Quelltextdatei : FILE OF BYTE;
    i,n,Byt      : INTEGER;
    B            : Byte;
    Hex          : String;
    OutputDateiname : String;
BEGIN
  MaxBef := MaxInit;
  Clrscr;
  Assign (QuelltextDatei,Dateiname);
  (*$I-*) Reset (QuelltextDatei); (*$I+*)
  IF IOResult = 0
  THEN BEGIN
    Schleife_Nr := 0;
    Adr := 2;Zeile :=1; Fehler:=0;
    Zustand := 0;
    Wort1:='';Wort2:='';
    FOR i := 1 TO FileSize(QuelltextDatei) DO BEGIN
      Read (QuelltextDatei,B); if b>20 then Write (chr(b));
      (* ---- Abtrennen des 1. und 2. Wortes im Quelltext ---- *)
      if ((Zustand=0) and (B>32)) then Zustand := 1;
      if ((Zustand=1) and (B>32)) then Wort1:=concat
        (Wort1,upcase (Chr(b)));
      If ((Zustand=1) and (B=32)) then Zustand:=2;
      if ((Zustand=2) and (B>32)) then Zustand :=3;
      if ((Zustand=3) and (B>32)) then Wort2:=concat
        (Wort2,upcase (Chr(b)));
      if ((Zustand=3) and (B=32)) then Zustand :=4;
      if ((b=13) or (i=FileSize(QuelltextDatei))) then begin
        Zustand :=0;
        Byte_Hex (Lo(Adr),Hex);
        GotoXY (40,WhereY);
        if Hi(Adr)>0 then write (hi(Adr)) else write (' ');
        Byte_Hex (lo(Adr),Hex);
        write (Hex,' ');
        if Fehler=0 then Translate;
        if Fehler=0 then Zeile:= Zeile+1;
        if Fehler>0 then i:=FileSize(QuelltextDatei);
        Anhalten;
        writeln;
      end;
    end;
  end;

```

```

        Wort1:=''; Wort2:=''; END
    END;
    Close (QuelltextDatei);
    END;
    Fehlermelden;
    delay (1000);
    Anhalten;
ClrScr;
    if Fehler =0 then begin
    writeln;
    writeln ('          00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ');
    For N:= 0 to (Adr-1) do begin
        Wort1 := copy (Hexcode[n],1,2);
        if (N Mod 16) = 0 then begin
            writeln;
            Byte_Hex ((N div 16),Hex);
            write (' ',Hex,' ');
        end;
        write (Wort1,' ');
        END;
        delay (2000);
        Anhalten;
    end;
    END (*Compiler *);

procedure CompilatSpeichern;
    var OutputDateiname : String;
        N , Byt      : Integer;
begin
    n:=1;
    repeat
        n:=n+1
    until DateiName[n] = '.';
    OutputDateiname := copy (Dateiname,1,n-1);
    OutputDateiname := concat (OutputDateiname, '.BIN');
    gotoXY (1,1); write ('Dateiname: ',OutputDateiname);
    delay (500);
    assign (Ausgabefile,OutputDateiname);
    rewrite (Ausgabefile);
    For N:= 0 to (Adr-1) do begin
        Wort1 := copy (Hexcode[n],1,2);
        Hex_Byte (Wort1,Byt);
        write (Ausgabefile,Chr(Byt));
    END;
    close (Ausgabefile);
end;

BEGIN
    ClrScr;
    Write ('Dateiname? ');
    Readln (Dateiname);
    LadeTabelle;
    MaxInit := MaxBef;
    Adr := 0;
    Fehler := 0;
    ClrScr;
    Compiler;

```

```
    CompilatSpeichern;  
END.
```