Technische Beschreibung

Einchipmikrorechner

U 883

veb mikroelektronik ›karl marx‹ erfurt
stammbetrieb

DDR – 5010 Erfurt, Rudolfstraße 47 Telefon 580 Telex 061306

Technische Beschreibung

Einchipmikrorechner

U 883

veb mikroelektronik »karl marx« erfurt
stammbetrieb

DDR – 5010 Erfurt, Rudolfstraße 47 Telefon 580 Telex 061 386

Vorbemerkungen

Diese Beschreibung soll in knapper Form alles Wichtige über die im U883 enthaltene Software vermitteln. Das betrifft sowohl die Initialisierungsroutine nebst einiger kurzer Routinen als auch den BASIC-Interpreter.

Folgende Vorraussetzungen sollten beim Leser erfüllt sein:

- Kenntnis der techn. Beschreibung des U881
- Kenntnis irgendeiner BASIC-Variante.

INHALT

INIT

1. Verhalten unmittelbar nach Reset

Nach Reset wird zunächst getestet, ob am Port 3 zwischen Bit 2 und 5 ein Schluß besteht. Es handelt sich dabei um einen Eingang und einen Ausgang, die man normalerweise nicht miteinander verbinden wird. Besteht ein Schluß, dann geht der EMR in ein Testprogramm (s. Abschn. 2), andernfalls werden Port 0 und 1 für den Anschluß von externem Speicher initialisiert. Deshalb muß P32 grundsätzlich mit einem definierten Pegel beschaltet sein, um eine Fehlauslösung des Testprogramms zu verhindern. Danach wird getestet, ob auf der Adresse %812 im externen Speicher RAM ist, indem auf diese Adresse geschrieben und anschließend gelesen wird. Sollte das so sein, dann folgt ein Sprung auf die Adresse %E000; wenn nicht, dann wird auf %812 gesprungen.

Die Interruptvektoren zeigen auf die Adressen %800, %803, %806, %809, %80C und %80F. Damit hat man jeweils 3 Bytes für einen Sprungbefehl in die eigentliche Serviceroutine.

2. Hardware - Testmöglichkeiten

Das im ersten Abschnitt erwähnte Testprogramm setzt zunächst die Ports 0 und 1 auf Output und initialisiert alle Bits mit high. Am Pin 40 (Tout) wird der Systemtakt ausgegeben. Danach läuft eine Testschleife, in der folgendes geschieht:

Auf Bit 5 von Port 3 wird ein Triggerimpuls ausgegeben. Danach werden der Reihe nach P10 bis P17 und P00 bis P07 kurzzeitig auf low gelegt. Danach folgt der nächste Triggerimpuls usw. Damit ist es möglich, bereits mit einem einfachen Serviceoszillografen den häufigsten Fehler, nämlich Schlüsse auf den Daten- und Adresseleitungen schnell zu lokalisieren.

Ein weiterer Test wird begonnen, wenn man kurzzeitig P32 mit einem der Pins vom Port 0 oder 1 kurzschließt. Dann nämlich werden die Leitungen der Ports 0 und 1 sowie die Ausgänge AS, DS und R/W hochohmig und es besteht die Möglichkeit, einen Testadapter

auf den eingelöteten Schaltkreis aufzusetzen, um damit den externen Speicher zu testen. Das ist besonders für die Inbetriebnahme von Geräten in der Serienfertigung interessant.

Wenn einmal sichergestellt ist, daß der externe Speicher funktioniert, dann kann man eventuell dort lokalisierte Testprogramme für den Rest der Schaltung aufrufen.

3. Direkter Speicherzugriff

Der U883 hat keinen BUSREQ-Eingang wie der U880. Dennoch sind mit einer internen Interruptserviceroutine DMA-Transfers möglich. Als Nachteil muß hingenommen werden, daß die Zeit bis zur Busfreigabe verhältnismäßig lang ist. (82 Taktzyklen = 20,5 µs bei 8MHz) Beim eigentlichen Datentransfer gibt es keine Einschränkungen.

Als Busrequest-Eingang dient P32, als Busacknowledge-Ausgang dient P35. Beide Signale sind low-aktiv.

Man beachte, daß außer den sonst für Interrupts notwendigen Initialisierungen noch das Register %7f mit dem Byte, mit dem das Port0/1-Mode-Register geladen wurde, zu initialisieren ist. Das ist notwendig, weil dieses Steuerregister modifiziert werden muß, aber nicht lesbar ist.

Das gesamte Testprogramm ist im Anhang C zu finden.

INTERPRETER

1. Grundlegende Ideen

In diesem Abschnitt sollen die Motive erläutert werden, die zur Entwicklung eines Einchipmikrorechners mit residentem BASIC-Interpreter geführt haben.

Die ursprüngliche Absicht bestand darin, auch Anwendern, die nur über bescheidene oder gar keine Entwicklungstechnik verfügen, eine Möglichkeit in die Hand zu geben, selbst effektiv Software zu entwickeln. Im Nachgang stellte sich jedoch heraus, daß auch der Anwender, der ein Entwicklungssystem mit einem leistungsfähigen Assembler hat, bei der Programmentwicklung mit BASIC schneller ist.

Deshalb entstand im "veb mikroelektronik 'karl marx' erfurt" die Tiny BASIC-Variante "Tiny MPBASIC". Der Name MPBASIC soll zwei Dinge andeuten: MP steht für Mikroprozessor, d.h. der Interpreter wurde ganz allgemein für die Programmierung von Mikroprozessoren konzipiert. Zweitens soll angedeutet werden, daß es sich um eine eigenständige Variante des mme handelt, die in einigen Details vom üblichen Tiny BASIC abweicht.

Bei einem Interpreter für den nur knapp 2 Kbyte Programmspeicher zur Verfügung stehen, gibt es natürlich harte Restriktionen bezüglich der implentierbaren Funktionen. Um dennoch eine akzeptable Leistungsfähigkeit zu erreichen, wurde davon ausgegangen, daß die nur für die Programmentwicklung notwendigen BASIC-Kommandos wie z.B. LIST und RUN durchaus in einem externen Programm untergebracht werden können. In einem Gerät, in dem mit einem fertigen Programm gearbeitet wird, würden sie nur unnötig Speicherplatz belegen.

Tiny MPBASIC wurde deshalb in zwei Teile zerlegt:

a) den Interpreter und
b) den Editor/Debugger.

Der Interpreter ist der Teil, der die 2 Kbyte des internen ROM's belegt, während der Editor/Debugger entweder im externen Speicherbereich oder in einem Wirtsrechner untergebracht ist.

Obwohl Tiny MPBASIC für den U883 entwickelt wurde, ist diese Sprache auf jeden anderen Prozessor übertragbar. Das wird dadurch erreicht, daß der Benutzer anwendungsspezifische Prozeduren selbst in Maschinensprache schreiben und über einen Namen aufrufen kann.

Diese Dokumentation beschreibt den Interpreter, der in der Lage ist, ein fertiges und fehlerfreies BASIC-Programm abzuarbeiten.

2. Allgemeines über Tiny MPBASIC

Wie in Tiny BASIC Üblich, sind die Variablen A ... Z zulässig. Sie sind intern 16 Bit breit und werden im Zweierkomplemet abgespeichert. Damit wird ein Bereich von -32767 (8001 Hex) bis 32767 (7FFF Hex) erfaßt. Zahlen können auch hexadezimal dargestellt werden. Zur Kennzeichnung von Hexzahlen wird % benutzt. Die Variablen werden beim U883 im Registersatz ab Register %20 abgelegt. Der Rest des Registersatzes ab %54 ist z.B. als Stack nutzbar. Die arithmetischen Operatoren interpretieren die 16 Bit als Zahlen im Zweierkomplement, während die logischen Operatoren bitweise arbeiten.

3. Prozeduren und Funktionen

3.1 Definitionen und Anwendung

Eine Prozedur ist ein in Maschinensprache geschriebenes Programm, das einen gewissen Satz von Eingabedaten vom Interpreter übergeben bekommt und einen gewissen Satz von Ausgabedaten an denselben zurückgibt. Eine Prozedur kann auch ohne Übergabedaten aufgerufen werden. Wieviele Daten maximal übergeben werden können, hängt von der Belastbarkeit des Stacks ab.

Prozeduren können mit einem Namen aufgerufen werden. Das er-

leichtert die Programmierung ganz erheblich, da sich der Programmierer statt einer Adresse nur einen Namen merken muß. Die Adresse einer Prozedur kann geändert, oder erst nachträglich festgelegt werden, ohne daß das BASIC-Programm geändert werden muß. Außerdem wird es dadurch möglich, systemunabhängige Programme zu schreiben.

Eine Funktion ist eine Prozedur, die genau einen Wert an den Interpreter zurückgibt. Sie darf auch in Ausdrücken verwendet werden, während Prozeduren nur mit der dafür geschaffenen PROCedure-Anweisung aufgerufen werden können.

Der Interpreter kennt einige Standardprozeduren und -funktionen die vom Benutzer durch beliebig viele eigene Prozeduren ergänzt werden können. Damit ist es möglich, Tiny MPBASIC optimal einer gegebenen Hardwarekonfiguration anzupassen.

Die Zuordnung des Prozedurnamens zur Prozeduradresse erfolgt über eine Tabelle die folgendes enthält:

1) die Länge des Namens (1 byte)
2) den Namen (max. 255 byte)
3) die Adresse der Prozedur (2 byte)

Das Ende der Tabelle wird mit %FF gekennzeichnet.
Wo sich diese Tabelle befindet, ist in die Register 8 und 9 einzuschreiben. Existiert solch eine Tabelle nicht, dann müssen diese Register mit 0 initialisiert werden.

3.2 Datenübergabe bei einer Prozedur (beim U883)

Wird eine Prozedur aufgerufen, an die die Werte X1 ... Xn übergeben werden und die die Werte Y1 ... Ym zurückliefert, dann befindet sich der Stack in folgendem Zustand:

SP+2n+2m	reserviert für Ym-1
	.
	.
	.
SP+2n+5	reserviert für Y3
SP+2n+2	reserviert für Y1
SP+2n	(wird vom Interpreter benötigt)
SP+2n-2	X1
	.
	.
	.
SP+4	Xn-2
SP+2	Xn-1
SP	Rückkehradresse zu Interpreter

Ym wird in den Arbeitsregistern R4 und R5 und Xm in R2 und R3 übergeben.

Vor dem Return zum Interpreter soll der Stackpointer um 2n-2 erhöht worden sein (,wenn n>1,) und auf die Rückkehradresse zum Interpreter zeigen.

4. Ausdrücke

Die Verknüpfung der Daten (Variablen, Konstanten und Funktionen) erfolgt in den Ausdrücken mit Hilfe der arithmetischen und der logischen Operatoren. Diese sind:

a) + für Addition
b) - für Subtraction
c) * für Multiplikation

d) / für Division
e) $MOD für modulo
f) $AND für logisches UND
g) $OR für logisches ODER und
h) $XOR für exklusives ODER.

Ausdrücke werden streng von links nach rechts abgearbeitet. Es besteht jedoch die Möglichkeit, Klammern zu setzen. Diese dürfen auch ineinander verschachtelt sein. Die zulässige Tiefe hängt vom verfügbaren Stack ab. Wird der interne Stack benutzt, so ist bei zwei geöffneten Klammern ein Überlaufen i. allg. nicht zu befürchten.

Wo überall Ausdrücke auftreten dürfen, ist der Beschreibung der Anweisungen zu entnehmen.

5. Das Basicprogramm

Das Basicprogramm besteht aus Zeilen, die mit einer Marke beginnen, und beliebig viele, durch Semikolons getrennte Anweisungen enthalten dürfen:

```
marke anweisung ; ... ; anweisung
```

Es muß wenigstens eine Anweisung auf jeder Zeile stehen.

Bei der Marke handelt es sich um eine positive Zahl zwischen 0 und 32766. Die Zeilen müssen in aufsteigender Folge markiert werden, da sonst Programmverzweigungen nicht ordnungsgemäß ausgeführt werden können. Das Sortieren übernimmt gewöhnlich der EDITOR. Ist ein solcher nicht vorhanden, dann ist noch zu beachten, daß die Anweisungsnamen abgekürzt werden (siehe Syntaxübersicht), und daß eine Anweisung außer in Kommentaren und Texten keine Leerzeichen enthalten darf. Es wird ansonsten im ASCII-Code abgespeichert.

6. Anweisungen

Eine Anweisung besteht aus ihrem Namen und den Argumenten. Soll eine Anweisung mehrmals hintereinander mit verschiedenen Argumenten ausgeführt werden, dann genügt es, den Namen einmal aufzuschreiben und die verschiedenen Argumente durch Kommas zu trennen.
Die folgenden zwei Zeilen sind äquivalent:

```
10 LET A=0; LET B=1
10 LET A=0, B=1
```

6.1 LET

LET dient der Wertzuweisung für eine Variable.

Syntax:

'LET' variable '=' ausdruck

6.2 GOTO

GOTO dient der Programmverzweigung.

Syntax:

'GOTO' ausdruck

Die Zieladresse kann also berechnet werden. Dadurch ist es z.B. möglich, in einem Unterprogramm gewisse Programmteile in Abhängigkeit von den Notwendigkeiten des rufenden Programms abzuarbeiten.

Man kann sich leicht überlegen, daß es keinen Sinn hat, mehrere durch Kommas getrennte Ausdrücke anzugeben.

6.3 GOSUB

Mit GOSUB können Unterprogramme aufgerufen werden.

Syntax:

'GOSUB' ausdruck

Ein Unterprogramm kann wieder ein anderes Unterprogramm aufrufen. Die maximale Tiefe für das Verschachteln von GOSUB-Anweisungen wurde auf 15 beschränkt, um ein versehentliches Überlaufen des Stacks zu verhindern.

6.4 RETURN

RETURN markiert das Ende eines Unterprogramms und bewirkt die Rückkehr ins rufende Programm und zwar zu der dem GOSUB folgenden Anweisung. Es hat also keinen Sinn, z.B.

```
GOSUB 100, 400
```

zu programmieren.

6.5 IF ... THEN, ELSE

Mit IF ... THEN kann eine Zeile in Abhängigkeit von einem Testergebnis abgearbeitet oder übersprungen werden.

Syntax:

'IF' bedingung 'THEN' anweisung {; anweisung}*
'ELSE' anweisung {; anweisung}*

Eine Bedingung hat die Form

ausdruck rel_op ausdruck

und rel_op kann

=, >, <, >=, <= oder <>

sein. Ist die Bedingung erfüllt, so wird anweisung abgearbeitet; andernfalls werden anweisung und der Rest der Zeile übersprungen.

Die einer Zeile mit IF folgende Zeile kann mit ELSE beginnen. Sie wird nur dann abgearbeitet, wenn die Bedingung nicht erfüllt war.

6.6 PROCedure

Die Prozedur-Anweisung dient dem Aufruf einer Prozedur (s. Abschn. 4).

Syntax:

'PROC' {variablenliste} =
prozedurname {parameterliste}

Die Variablenliste sind in eckige Klammern gesetzte und durch Kommas getrennte Variablen. Bei der Parameterliste handelt es sich um in eckige Klammern gesetzte und durch Kommas getrennte Ausdrücke.

6.7 Consoleneingabe- und ausgabe

Ist eine Console angeschlossen, sei es nun als Terminal, Fernschreiber, LED- oder LCD-Anzeige und Tastatur, so kann diese mit INPUT, PRINT und PRINTHEX bedient werden.

Da von vornherein nicht klar ist, welche Schnittstelle benutzt werden soll, muß der Anwender zwei Treiberprogramme, get_char und put_char selbst schreiben. Bei diesen Programmen wird je ein

Zeichen von der Console übernommen bzw. an sie übergeben. Diese Routinen müssen auf den Adressen %815 bzw. %818 beginnen. Das Zeichen wird in R3 (get_char) bzw. in R5 (put_char) übergeben. Der Registerzeiger steht auf %10 und muß am Schluß wieder mit diesem Wert geladen sein. Andere Register aus diesem Registersatz dürfen nicht benutzt werden. Beispiele für get_char und put_char sind im Anhang B zu finden.

6.7.1 PRINT

Die PRINT-Anweisung druckt auf die Console.

Syntax:

'PRINT' {'"' text '"'} {ausdruck}

Dabei wird der Wert von ausdruck in dezimaler Form ausgegeben. Der Text darf kein '"'enthalten. Sowohl text als auch ausdruck dürfen weggelassen werden. Wird beides weggelassen, so schickt PRINT lediglich ein Carriage Return <CR> (%0D) an die Console. Wird PRINT mit .Komma abgeschlossen, dann unterbleibt die Ausgabe von <CR>.

6.7.2 PRINTHEX

PRINTHEX unterscheidet sich von PRINT nur dadurch, daß das Ergebnis von ausdruck hexadezimal ausgegeben wird.

Syntax:

'PRINTHEX' {'"' text '"'} {ausdruck}

6.7.3 INPUT

Syntax:

'INPUT' {'"' text '"'} variable

INPUT druckt zunächst text auf die Console und erwartet danach die Eingabe eines Wertes. Die Eingabe kann sowohl dezimal als auch hexadezimal (mit % vor dem Wert) erfolgen. Bei fehlerhaften Eingaben wird ein Fragezeichen ausgegeben und eine neue Eingabe erwartet.

INPUT ist auch als Funktion verfügbar. (s. Abschn. 7)

6.8 STOP

Mit STOP kann das Programm angehalten, später aber weiter abgearbeitet werden. Die Programmzeile, in der 'STOP' auftauchte, wird noch fertig abgearbeitet.

Diese Anweisung dient in Verbindung mit dem EDITOR/DEBUGGER dem Setzen von Unterbrechungspunkten, ist also für die Programmtestung interessant.

6.9 END

Sollte das Ende des BASIC-Programms nicht mit der letzten Anweisung zusammenfallen, dann kann es mit 'END' gekennzeichnet werden.

END bewirkt also ein Verlassen des Interpreters. Der Rest der

Zeile wird auch hier noch abgearbeitet.

6.10 REMark

REM gestattet es, Anmerkungen im Programm unterzubringen.

Syntax:

'REM' {kommentar}

Der Interpreter überliest diese Anweisung.

Bei der Anwendung von REM sollte man allerdings bedenken, daß damit u.U. eine erhebliche Menge Speicherplatz belegt wird.

6.11. WAIT

Syntax:

'WAIT'ausdruck

Der Ausdruck wird berechnet, und anschließend wird eine Softwarewarteschleife durchlaufen(ohne Timer). Wie oft sie durchlaufen wird, liegt am Ergebnis von ausdruck. Dieses wird ohne Vorzeichen interpretiert, d.h. beim Ergebnis -1 (= %FFFF) z.B., wird die Schleife 65535 mal durchlaufen und bei +1 einmal. Ein Durchlauf dauert eine Millisekunde bei einer Taktfrequenz von intern 4 MHz. Da auch für den Aufruf der Zeitschleife Zeit benötigt wird, muß man damit rechnen, daß eine etwas längere Zeit als programmiert vergeht. Für Zeitmessungen ist diese Anweisung also nicht geeignet.

Sie kann aber z.B. zum Entprellen von Tasten benutzt werden.

6.12 CALL

Syntax:

'CALL' ausdruck

CALL ruft ein Maschinenprogramm auf, dessen Adresse durch das Ergebnis von ausdruck geliefert wird. Dieses Programm muß mit RET enden, was eine Rückkehr zum Interpreter bewirkt.

6.13 TRAP

Syntax:

'TRAP' bedingung 'TO' ausdruck

Von dem Augenblick, an dem TRAP gegeben wurde, wird bedingung vor der Abarbeitung jeder neuen Zeile überprüft. Sobald die Bedingung erfüllt ist, geht der Interpreter in die Traproutine, deren Anfangsadresse durch ausdruck angegeben wird. Die Trapbedingung wird vorher gelöscht. Die Traproutine ist ein gewöhnliches Unterprogramm, das mit RETURN endet.

Sobald eine Trapbedingung aktiv ist, verlangsamt sich natürlich die Programmabarbeitung.

6.14 CLRTRP

Syntax:

'CLRTRP'

CLRTRAP löscht die Trapbedingung.

7. Standardprozeduren und -funktionen

Einige elementare Prozeduren und Funktionen sind standardmäßig im Interpreter vorhanden. Es sind dies zunächst die allgemeinen, d.h. nicht auf den U883 zugeschnittenen Funktionen:

ABS[par] , absoluter Betrag von par
NOT[par] , logische (bitweise) Negation von par
RL[par] , par links rotieren
RR[par] , par rechts rotieren
GTC , Zeichen mittels get_char holen
INPUT , Zahl mittels get_char holen

Dazu kommt die Prozedur

PTC[par] , par mittels put_char ausgeben.

Die anderen Funktionen und Prozeduren dienen dazu, auf die Register und auf den externen Datenspeicher zuzugreifen.

- GETR[reg] liefert den Inhalt von reg; die höherwertigen 8 Bits sind 0.
- GETRR[reg] liefert den Inhalt von reg (höherwertige 8 Bits) und reg+1 (niederwertige 8 Bits)

Ganz analog arbeiten die Funktionen

- GETEB[adr] (Byte aus externem Datenspeicher holen) und
- GETEW[adr] (Wort aus externem Datenspeicher holen).

Sinngemäß können auch Register und Bytes (Worte) im externen

Datenspeicher gesetzt werden. Dazu gibt es die Prozeduren:

- SETR[reg, wert] , Register setzen
- SETRR[reg, wert] , Doppelregister setzen
- SETEB[adr, wert] , externes Byte setzen
- SETEW[adr, wert] , externes Wort setzen.

Damit sind die Möglichkeiten des Interpreters dargestellt. Weitere Details können der Syntaxbeschreibung (Anhang A) entnommen werden.

8. Beispielprogramm und Anwendungshinweise

8.1. Echtzeitbetrieb

Bei Steuerungen und Regelungen ist es natürlich notwendig, die Reaktionszeiten des Rechners zu kennen. Im Gegensatz zur Assemblerprogrammierung ist es nicht möglich, für jede Anweisung eine exakte Abarbeitungszeit anzugeben. Man möge jedoch berücksichtigen, daß auch bei der Programmierung in Maschinensprache die theoretische Ermittlung der Verarbeitungszeit so aufwendig werden kann, daß sie praktisch unmöglich wird und man mit Ausprobieren wesentlich schneller zum Ziel kommt.

Die Möglichkeit des Ausprobierens besteht natürlich bei BASIC auch. Dabei ist es wichtig zu wissen, daß die Abarbeitungszeit für einen Ausdruck nicht von der aktuellen Belegung der Variablen abhängt, sofern nur die intern enthaltenen Funktionen benutzt werden. Damit jedoch abgeschätzt werden kann, ob der Interpreter für eine spezielle Steuerung prinzipell geeignet ist, sei folgendes gesagt:

Für die Abarbeitung einer Anweisung kann man etwa 2 ... 3 ms veranschlagen. Wenn also in einer Anlage die Zeitabläufe in Sekunden gemessen werden, dann kann BASIC prinzipiell eingesetzt werden. Sollten jedoch Reaktionszeiten benötigt werden, die im kritischen Bereich, also bei einigen Millisekunden und darunter liegen, so gibt es die Möglichkeit, Maschinenprogramme in das Basicprogramm einzubinden, oder mit Interrupts zu arbeiten.

Nehmen wir z.B. an, es soll ein EPROM U555 programmiert werden. Dann benötigt man einen Programmierimpuls mit einer Dauer von 0,1 ... 1 ms. Das läßt sich in BASIC nicht programmieren, ist aber mit einer Prozedur, die z.B. "PULSE" heißen könnte, sehr leicht zu bewerkstelligen.

Wenn man das eben Gesagte berücksichtigt, dann ist Tiny MPBASIC also durchaus "echtzeitfähig".

8.2. Beispiel

Das folgende Beispiel demonstriert, wie ein Programm - das Auflisten eines Speicherinhalts - , das gewöhnlich in Assembler geschrieben wird, auch in Tiny MPBASIC abgefaßt werden kann.

```
 10 REM RAM DUMP
 20 INPUT "ADRESSE: " A, "LAENGE: " I
 25 IF I<=0 THEN END
 30 IF I $MOD %10<>0 THEN LET I = I+(%10-(I $MOD %10))
 40 REM EINE ZEILE
 50 LET J=0; PRINTHEX A, "   ",; REM ADRESSE
 60 PRINTHEX " " GETEW  A ,
 70 LET I=I-2, J=J+2, A=A+2
 80 IF J<16 THEN GOTO 60
 90 LET A = A-J
100 PRINT "     ",
105 REM ASCII
110 LET C=GETEB[A]
120 IF C>%1F THEN IF C<%7F THEN PROC PTC[C]
130 ELSE PROC PTC[%2E]
140 LET J=J-1, A=A+1
150 IF J>0 THEN GOTO 110
160 PRINT
170 IF I>0 THEN GOTO 50
```

8.3. Felder

Die Verarbeitung von Feldern ist zwar nicht implementiert, aber trotzdem relativ leicht möglich.

Will man systemunabhängig sein, dann sollte man sich eine Funktion

```
adr = ALLOC[ len ]
```

schreiben, die mit Hilfe des Speicherverwalters im Betriebssystem (sofern vorhanden) Speicher der gewünschten Länge len von Bytes reserviert. Diese Funktion liefert dann die Adresse adr, ab der Speicher reserviert wurde. Mit adr = 0 kann angezeigt werden, daß nicht genügend Speicher vorhanden war.

Dann kann man mit

```
LET A = ALLOC[2N]
```

die im Standard-BASIC implementierte Anweisung

```
DIM A%(N)
```

zum Definieren eines Feldes von ganzzahligen Variablen nachvollziehen. Statt

```
LET A%(I) = X
```

kann nunmehr

```
PROC SETEW[A+(2*I), X]
```

geschrieben werden. Die Indexvariable I läuft, statt von 1 bis N, von 0 bis N-1. A hat die Funktion eines Zeigers.

Wenn das Programm nicht auf andere Systeme übertragbar sein muß, kann man natürlich auch per Hand einen Plan zu dem benötigten externen Speicher anfertigen und auf ALLOC verzichten.

ANHANG

****

S Y N T A X - Ü B E R S I C H T

für

Tiny MPBASIC (U883)

Vers. 3.0

****

Es bedeutet:

/ ->	oder
{...} ->	... darf weggelassen werden
{...} + ->	... darf beliebig oft dastehen, wenigstens aber einmal
{...} +n ->	... darf maximal n-mal, muss aber mindestens einmal dastehen
{...} * ->	... darf weggelassen werden oder beliebig oft dastehen
{...} *n ->	... darf weggelassen werden oder bis zu n-mal dastehen
' ... ' ->	... ist in ASCII

SYNTAX:

ARITHMETIK_OPERATOR => '+'/'-'/'*'/'/'/'$MOD'

KOMMENTAR => beliebiges ASCII-String ohne <CR> und ohne ';'

BEDINGUNG => AUSDRUCK REL_OP AUSDRUCK

KONSTANTE => HEX_KONST/DEZ_KONST

ANHANG

```
ZIFFER =>          '1'/'2' ... '8'/'9'/'0'

DEZ_KONST =>     {-} {ZIFFER}+5

AUSDRUCK =>    WERT { OPEARATOR WERT}*

FUNKTIONS_NAME =>          IDENTIFIER

FUNKTION =>      FUNKTIONS_NAME  {PARAMETER_LISTE}

HEX_KONST =>     '%'{ZIFFER/'A'/'B'/'C'/'D'/'E'/'F'} +4

IDENTIFIER =>    LETTER {LETTER/ZIFFER}+254

MARKE =>         {ZIFFER}+5

LETTER =>        'A'/'B'/'C' ...
                                  'X'/'Y'/'Z'

ZEILE =>          MARKE ANWEISUNG { ';' ANWEISUNG } *

LOGIK_OPERATOR =>          '$AND'/'$OR'/'$XOR'

OPERATOR =>      ARITHMETIK_OPERATOR/LOGIK_OPERATOR

PARAMETER_LISTE =>  '[' AUSDRUCK {','AUSDRUCK}* ']'

PROZEDUR_NAME =>          IDENTIFIER

PROGRAMM =>          {ZEILE} +32767

REL_OP =>          '>'/'<'/'='/'>='/'<='/'<>'

EINFACHER_WERT => VARIABLE_NAME/KONSTANTE/FUNKTION
```

ANHANG

```
ANWEISUNG =>     ANWEISUNGS_NAME  ANWEISUNGS_ARGUMENTE
                                 {',' ANWEISUNGS_ARGUMENTE}*

ANWEISUNGS_ARGUMENTE =>  siehe Beschreibung der Anweisungen

ANWEISUNGS_NAME =>        'LET' / 'PROC' / 'GOTO' /
                          'IF' / 'ELSE' / 'RETURN' /
                          'GOSUB' / 'WAIT' / 'REM' /
                          'CALL' / 'STOP' / 'END' /
                          'TRAP' / 'CLRTRP' /
                          'PRINT' / 'PRINTHEX' / 'INPUT'

TEXT =>          beliebiges ASCII-String,
                 das nicht '"' und nicht <CR> enthaelt

WERT =>         EINFACHER_WERT / '(' AUSDRUCK ')'

VARIABLEN_LISTE =>  '[' VARIABLEN_NAME {',' VARIABLEN_NAME}* ']'

VARIABLEN_NAME =>         LETTER
```

Die Variablen sind 16-bit INTEGER.
Bei BYTES wird nur das niederwertige Byte benutzt,
waehrend die restl. 8 bit bei Schreiboperationen
Null gesetzt werden.

ANWEISUNGEN:

```
'LET' VARIABLEN_NAME '=' AUSDRUCK
'GOTO' AUSDRUCK
'IF' BEDINGUNG 'THEN' ANWEISUNG
'ELSE' ANWEISUNG
'GOSUB' AUSDRUCK
'RETURN'
```

ANHANG

'PROC' {VARIABLEN_LISTE}'=' PROZEDUR_NAME{PARAMETER_LISTE}
'TRAP' BEDINGUNG 'TO' AUSDRUCK
'CLRTRP'
'INPUT' {'"' TEXT '"'} VARIABLEN_NAME
'PRINT' {'"' TEXT '"'} {AUSDRUCK}
'PRINTHEX' {'"' TEXT '"'} {AUSDRUCK}
'STOP'
'END'
'REM' {KOMMENTAR}
'WAIT' AUSDRUCK
'CALL' AUSDRUCK

STANDARD PROZEDUREN:

allgemein:

- PTC	! PUT CHARACTER !

nur für U883

- SETR	! SETZE REGISTER !
- SETRR	! SETZE DOPPELREGISTER !
- SETEB	! SET EXTERNAL BYTE !
- SETEW	! SET EXTERNAL WORD !

STANDARD FUNKTIONEN

allgemein:

- ABS	! ABSOLUTER BETRAG !
- NOT	! LOGISCHE NEGATION !
- GTC	! GET CHARACTER !
- INPUT	
- RL	! ROTATE LEFT !
- RR	! ROTATE RIGHT !

nur für U883:

- GETR ! GET REGISTER !
- GETRR ! GET DOPPELREGISTER !
- GETEB ! GET EXTERNAL BYTE !
- GETEW ! GET EXTERNAL WORD !

ABKUERZUNGEN

MARKE -> 16-bit INTEGER, positiv,
jedoch bit 15 gesetzt

FUER ANWEISUNGEN:

LET -> L
GOTO -> G
IF ... THEN -> F ... ,
ELSE -> >;
GOSUB -> S
RETURN -> R
PROC -> O
TRAP ... TO -> ! ... ,
CLRTRP -> /
INPUT -> I
PRINT -> P
PRINTHEX -> H
STOP -> T
END -> E
WAIT -> W
CALL -> C
REM -> M

FUER OPERATOREN:

$AND -> $A
$OR -> $O
$XOR -> $X
$MOD -> $M

Der restliche Text wird ohne Leerzeichen in ASCII abgespeichert.
Das Programmende wird mit 00 gekennzeichnet.

Definieren einer Prozedur durch den Benutzer:

1. Tabelle:

```
'LAENGE DES NAMENS (1 BYTE)
        N
        A
        M
        E,
ADRESSE DER PROZEDUR (2 BYTES),

        .
        .
        .
weitere Definitionen
        .
        .
        .

%FF AM ENDE'
```

2. Datenübergabe über Stack (bei U883):

ANHANG

```
SP: RET-ADRESSE ZU INTERPRETER
SP+2 ... SP+2n-2: N-1 PARAMETER, WENN N>1
        LETZTER PARAMETER AUS LISTE IST IN
        ARBEITSREGISTERN 4 UND 5.
SPn+2: WIRD VON INTERPRETER BENOETIGT
SP+2n+2 ... SP+2n+2m: PLATZ FUER M-1
        ERGEBNISVARIABLE, WENN M>1
        DER WERT FUER DIE LETZTE VARIABLE
        IST IN DIE ARBEITSREGISTER 2 UND 3
        ZU LADEN
```

```
                                1
                                2
                                3          !****************************
                                4
                                5                  Beispiel für
                                6
                                7          start sowie
                                8          put_char und get_char
                                9
                               10          ****************************!
                               11
                               12
                               13 constant
                               14               ! Register !
                               15               dsth := r2
                               16               dstl := r3
                               17               srch := r4
                               18               srcl := r5
                               19
                               20
                               21
                               22 internal
P 0000                         23               start procedure
                               24               entry
                               25               $abs %800
                                                ! Interruptsprünge !
P 0800 30  54                  26               jp @%54
P 0802 FF                      27               nop
P 0803 30  56                  28               jp @%56
P 0805 FF                      29               nop
P 0806 30  58                  30               jp @%58
P 0808 FF                      31               nop
P 0809 30  5A                  32               jp @%5a
P 080B FF                      33               nop
P 080C 30  5C                  34               jp @%5c
P 080E FF                      35               nop
```

```
P 080F 30  5E          36          jp @%5e
P 0811 FF              37          nop
                                   ! Eintrittspunkte !
P 0812 8D  0843        38          jp hinit
P 0815 8D  081B        39          jp get_char
P 0818 8D  082F        40          jp put_char
P 081B                 41          end start
                       42
                       43
                       44
                       45          ! Zeichen von SIO in dst !
                       46 global
P 081B                 47          get_char procedure
                       48          entry
P 081B 56  FA  F7      49          and irq,#%f7     ! Altes Request
                                                      rücks. !
P 081E 76  FA  08      50 gtc10:   tm irq,#8
P 0821 6B  FB          51          jr z,gtc10       ! noch kein
                                                    Zeichen da !
P 0823 56  FA  F7      52          and irq,#%f7     ! Req. rücks. !
P 0826 38  F0          53          ld dstl,sio
P 0828 56  E3  7F      54          and dstl,#%7f    ! Par. rücks. !
P 082B B0  E2          55          clr dsth
P 082D 58  E3          56          ld srcl,dstl     ! Echo !
P 082F                 57          end get_char
                       58
                       59
                       60          ! Zeichen aus srcl in sio !
P 082F                 61          put_char procedure
                       62          entry
P 082F 76  FA  10      63          tm irq,#%10      ! letztes Zeichen
                                                        'raus ? !
P 0832 6B  FB          64          jr z,put_char
P 0834 56  FA  EF      65 ptc10:   and irq,#%ef     ! Request rücks. !
P 0837 59  F0          66          ld sio,srcl
P 0839 A6  E5  0D      67          cp srcl,#'%r'
```

```
P 083C 6B  01          68            jr z,ptc20
P 083E AF              69            ret
P 083F 5C  0A          70 ptc20:     ld srcl,#'%1'
P 0841 8B  EC          71            jr put_char
P 0843                 72            end put_char
                       73
                       74
                       75
                       76            ! Initialisierung !
                       77 internal
P 0843                 78            hinit procedure
                       79            entry
P 0843 31  F0          80            srp #%f0
P 0845 8C  96          81            ld r8,#%96        ! Normal Timing !
P 0847 EC  00          82            ld r14,#0
P 0849 FC  7F          83            ld r15,#%7f
P 084B 4C  02          84            ld r4,#2          ! 9600 Baud !
P 084D 7C  49          85            ld r7,#%49
P 084F 5C  0D          86            ld r5,#%d
P 0851 1C  43          87            ld r1,#%43
P 0853 E6  FB  02      88            ld imr,#%2
P 0856 9F              89            ei
P 0857 31  10          90            srp #%10
P 0859 5C  0D          91            ld srcl,#'%r'     ! CR ausgeben !
P 085B D6  0834        92            call ptc10
P 085E                 93            end hinit
                       94
                       95            ! Beginn des Anwenderprogramms !

                                     ...
```

```
                                                    1
                                                    2
                                                    3          !**********************************
                                                    4
                                                    5                   Bootstrap für U883
                                                    6
                                                    7          **********************************!
                                                    8
                                                    9
                                                   10
                                                   11 $section basic program
                                                   12
                                                   13 internal
P 0000 0800  0803                                  14          intvect array 6 word  :=
P 0004 0806  0809
P 0008 080C  080F
                                                   15                    [ %800, %803, %806,
                                                   16                      %809, %80c, %80f ]
                                                   17
P 000C                                             18          init procedure
                                                   19          entry
P 000C 31  00                                      20          srp #0
P 000E 3C  0F                                      21          ld r3,#%0f        ! Test, ob P32
                                                   22                  und P35 gebrückt sind !
P 0010 FF                                          23          nop
P 0011 76  E3  04                                  24          tm r3,#4
P 0014 3C  FF                                      25          ld r3,#%ff
P 0016 EB  05                                      26          jr nz,init10
P 0018 76  E3  04                                  27          tm r3,#4
P 001B EB  20                                      28          jr nz,test        ! Wenn ja
                                                                                 => Testmode !
                                                   29
                                                   30 init10:
                                                   31          ! extended external memory timing !
P 001D E6  F8  B6                                  32          ld p01m,#%(2)10110110
P 0020 E6  F7  08                                  33          ld p3m,#%(2)00001000
```

```
P 0023 4C  08          34          ld r4,#8          ! Test,
                                                     ob auf %812 RAM !
P 0025 5C  12          35          ld r5,#%12
P 0027 C2  64          36          ldc r6,$rr4
P 0029 60  E6          37          com r6
P 002B D2  64          38          ldc $rr4,r6
P 002D C2  74          39          ldc r7,$rr4
P 002F 60  E6          40          com r6
P 0031 D2  64          41          ldc $rr4,r6
P 0033 B2  67          42          xor r6,r7
P 0035 31  F0          43          srp #%f0
P 0037 ED  E000        44          jp nz,%e000       ! Wenn ja =>
                                         Sprung auf %E000 !
P 003A 8D  0812        45          jp %812           ! sonst auf
                                                     %812 springen !
P 003D                 46          end init
                       47
                       48
                       49
                       50          ! Testhilfe zum Auffinden von
                                   Schlüssen auf
                       51            den Daten-/Adressleitungen !
P 003D                 52          test procedure
                       53          entry
P 003D E6  F8  04      54          ld p01m,#4        ! p01 output !
P 0040 E6  F1  C0      55          ld tmr,#%c0       ! internen Takt
                                                     ausgeben !
P 0043 0C  FF          56          ld r0,#%ff
P 0045 1C  FF          57          ld r1,#%ff
P 0047 CF              58          rcf
                       59
P 0048 56  E3  DF      60 test10:  and r3,#%df       ! Triggerimpuls !
P 004B 46  E3  20      61          or r3,#%20
P 004E 10  E1          62 test20:  rlc r1
P 0050 10  E0          63          rlc r0
P 0052 76  E3  04      64          tm r3,#4          ! tristate ? !
```

```
P 0055 6B  04          65          jr z,tristate
P 0057 7B  F5          66          jr c,test20
                       67
P 0059 8B  ED          68          jr test10
                       69
                       70 tristate:
P 005B E6  F8  7F      71          ld p01m,#%7f
P 005E 8B  FE          72 halt:    jr halt
                       73
P 0060                 74          end test
                       75
                       76
                       77
                       78          ! Serviceroutine für BUSREQ
                       79          (BUSREQ an P32, BUSACK an P35;
                       80                  beides low-aktiv )
                       81          p01m muß in Reg. %7f verfügbar
                       82          sein, da p01m-Reg. nicht lesbar !
                       83 global
P 0060                 84          busreq procedure
                       85          entry
P 0060 46  7F  18      86          or %7f,#%18      ! tristate !
P 0063 E4  7F  F8      87          ld p01m,%7f      ! 3-state !
P 0066 56  03  DF      88          and 3,#%df       ! BUSACK geben !
P 0069 76  03  04      89 busreq10: tm 3,#4
P 006C 6B  FB          90          jr z,busreq10    ! Ende abwarten !
P 006E 56  7F  F7      91          and %7f,#%f7     ! Code für
                                                    AD0 ... AD7 !
P 0071 46  03  20      92          or 3,#%20        ! BUSACK
                                                     zurücknehmen !
P 0074 E4  7F  F8      93          ld p01m,%7f      ! wieder auf
                                                     Bus gehen !
P 0077 FF              94          nop              ! Ausf. abwarten!
P 0078 BF              95          iret
P 0079                 96          end busreq
```

ANHANG

Aufruf des Interpreters

Liegt ein syntaktisch einwandfreies Programm vor, dann kann dies folgendermaßen abgearbeitet werden:

1. Startadresse des Basicprogramms in die Register 6 und 7 laden.
2. Startadresse der Prozedurtabelle in die Register 8 und 9 laden.
   Ist eine solche Tabelle nicht vorhanden, dann Register 8 und 9 löschen.
3. Aufruf des Interpreters mit

```
srp #%10
call %7fd.
```

(siehe auch Anhang B.)

Erstellt am: 29.03.2020
Elmar Klinder

Ra 409/85 V/6/15