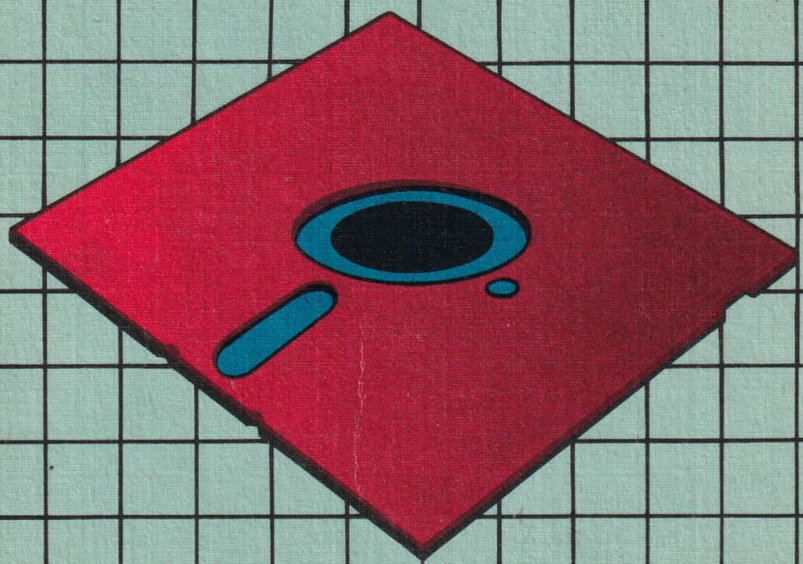
Technische Informatik

Claßen · Oefler

Wissensspeicher Mikrorechner-programmierung

Neu: CIO und SCC



VEB Verlag Technik Berlin

Claßen · Oefler

Wissensspeicher Mikrorechnerprogrammierung

Technische Informatik

Herausgegeben von

Dr.-Ing. Ludwig Claßen

Dr.-Ing. Dieter Nedo

Dr. rer. nat. Gerhard Paulin

Wissensspeicher Mikrorechner-programmierung

Dr.-Ing. Ludwig Claßen
Dipl.-Math. Ulrich Oefler

4., stark bearbeitete Auflage

VEB VERLAG TECHNIK BERLIN

Claßen, Ludwig
Wissensspeicher Mikrorechnerprogrammierung / Ludwig Claßen ; Ulrich Oefler. - 4., stark bearb. Aufl. - Berlin : Verl. Technik, 1989. - 240 S. : 51 Bilder, 22 Taf.
ISBN 3-341-00712-1
NE: Oefler, Ulrich

ISBN 3-341-00712-1
ISSN 0863-0860

4., stark bearbeitete Auflage

Lizenz 201 • 370/70/89
Printed in the German Democratic Republic
Druck und buchbinderische Verarbeitung:
Druckerei "Neues Deutschland", Berlin
Lektor: Jürgen Reichenbach
Einbandgestaltung: Gabriele Schwesinger
LSV 3054 • VT 3/5847-4
Bestellnummer: 554 103 8
02200

Vorwort

Zur 1. Auflage

Die Bereitstellung von modernen Mikroprozessorbauelementen und zugehörigen programmierbaren Peripherieschaltkreisen hat sich in den letzten Jahren in der DDR außerordentlich stürmisch entwickelt. Es steht heute eine ganze Palette solcher Bauelemente mit spezifischen Leistungsparametern für unterschiedlichste Einsatzfälle zur Verfügung. Dieser Entwicklung entsprechend ist die Anwendungsbreite von Mikroprozessoren in allen Bereichen der Volkswirtschaft gewachsen. Es gibt heute wohl kaum einen Bereich, wo nicht durch ihren Einsatz die Arbeits- und Lebensbedingungen rationeller und effektiver oder auch leichter und umweltfreundlicher gestaltet worden sind.
Ein ganz wesentlicher Aspekt des Einsatzes von Mikroprozessoren sind Fragen der Software. Ohne die anwendungsfallspezifische Programmierung handelt es sich bei einem Mikroprozessorsystem immer nur um eine nutzlose Anhäufung von hochintegrierten Halbleiterbauelementen. Erst durch die Programmierung wird spezifiziert, welche Anweisungen aus dem Befehlsrepertoire des Mikroprozessors in welcher Reihenfolge auszuführen sind, um das vorliegende Problem zu lösen. Die besondere Bedeutung der Software bei der Applikation der Mikroprozessoren ist nachdrücklich zu unterstreichen.
Dieser Band der Fachbuchreihe TECHNISCHE INFORMATIK soll in kurzgefaßter Form möglichst umfassend über die Programmierung der Mikroprozessorbausteine der DDR-Produktion informieren. Schwerpunkt ist die durch die Software mögliche Einsatzvariation der Schaltkreise und die Nutzung der verfügbaren Programmentwicklungswerkzeuge:

System U880	U880-CPU; U855-PIO; U856-SIO; U857-CTC; U858-DMA
System U8000	U8000-CPU; U8010-MMU
System U881/U882	U881/U882-EMR

PLZ/ASM Assemblerprogrammierung U880, U8000 und U881/U882
UDOS Floppy-Disk Programmentwicklungsbetriebssystem
Programmiersprachen U880-BASIC, U883-BASIC und U880-PASCAL

Die Programmiermöglichkeiten der Mikroprozessor- und Peripheriebausteine werden durch Befehlslisten, Programmiertabellen und Bilder beschrieben. Der Aufbau von PLZ/ASM-Programmen sowie die Handhabung der Übersetzerprogramme unter dem Betriebssystem UDOS werden an Hand von Syntaxdarstellungen erläutert. Die Beschreibungen der höheren Programmiersprachen haben das Ziel, die Anwendungsbreite zu verdeutlichen, um potentielle Einsatzfälle abschätzen zu können. Allgemeine Erläuterungen werden auf das durch den Rahmen dieses Bandes der Fachbuchreihe TECHNISCHE INFORMATIK gegebene Maß beschränkt, bzw. es wird auf sie ganz verzichtet und auf weiterführende Literatur verwiesen.

Der Band hat den Charakter eines zusammenfassenden Nachschlagebuches mit

hoher Informationsdichte. Er wendet sich an einen breiten Leserkreis, an Programmierer, an Mitarbeiter in Rechenzentren, an Entwicklungsingenieure, Studenden und Lehrkräfte - an alle, die sich mit Softwarefragen für die in der DDR gefertigten Mikroprozessorsysteme beschäftigen.
Dieser Band baut auf den Bänden 192 /1/ und 201 /2/ der REIHE AUTOMATISIERUNGSTECHNIK sowie der ersten, zweiten und dritten Auflage des vorliegenden 'Wissensspeichers Mikrorechnerprogrammierung' des VEB Verlag Technik auf, deren Inhalt aus heutiger Sicht an vielen Stellen durch die rasche Entwicklung stark ergänzungs- und überarbeitungsbedürftig war.
Trotz großer Sorgfalt bei der Manuskripterarbeitung (zur Manuskriptfortschreibung und zur Herstellung der Druckvorlage wurde ein modernes U880-Textverarbeitungssystem benutzt) ist es bei der Fülle des Stoffes nicht vollständig auszuschließen, daß sich Druck- oder Sachfehler eingeschlichen haben. Die Autoren sind deshalb jederzeit für Hinweise dankbar.
Besonderer Dank gilt dem Herausgeber, Herrn Dr. G. Paulin, für viele wertvolle Hinweise, dem verantwortlichen Lektor des VEB Verlag Technik, Herrn J. Reichenbach, für die außerordentlich konstruktive Zusammenarbeit sowie allen Mitarbeitern der Abteilung Basissoftware des ZFT/KEAW, die durch vielfältige Diskussionen und durch gemeinsame Arbeit zum Zustandekommen dieses Bandes beigetragen haben.

Zur 4. Auflage

Der 'Wissensspeicher Mikrorechnerprogrammierung' in der Fachbuchreihe TECHNISCHE INFORMATIK aus dem VEB Verlag Technik hat seit 1986 einen breiten Leserkreis gefunden. Die jetzt vorliegende 4. ergänzte und überarbeitete Auflage wurde notwendig, weil 1988 vom VEB Mikroelektronik 'Karl Marx' Erfurt zwei neue hochintegrierte Peripheriebausteine - U82530-/U8030-SCC und U82536-/U8036-CIO - vorgelegt wurden. Sie ergänzen und vervollständigen das 16-Bit-Mikroprozessorsystem U8000, können aber auch sehr effektiv im 8-Bit-Mikroprozessorsystem U880 oder im 16-Bit-Mikroprozessorsystem K1810WM86 eingesetzt werden.
Die anderen Kapitel wurden, entsprechend den aktuellen Erfordernissen, überarbeitet und es wurden eine ganze Reihe kleinerer Druck- und Sachfehler - auch Dank der Hilfe aufmerksamer Leser - beseitigt. Diesen Fachkollegen sei hier ausdrücklich gedankt.

Ludwig Claßen Ulrich Oefler

Inhaltsverzeichnis

1. Einleitung

Moderne Technologien bei der Herstellung von Halbleiterbauelementen ermöglichen die Anordnung umfangreicher logischer Strukturen auf einem Kristallplättchen (Chip) von wenigen Quadratmillimetern Grundfläche. Dieser hohe Integrationsgrad hat zur Folge, daß die für eine kostengünstige Produktion solcher LSI-Bauelemente (LSI large scale integration) notwendigen Stückzahlen von den wenigsten Anwendern erreicht werden, solange es sich um spezielle Kundenwunschlogik handelt. Ein Ausweg wurde hier in der Einführung programmierbarer Schaltkreise gefunden.
Der Mikroprozessor (MP) stellt heute die flexibelste Stufe solcher programmierbarer digitaler LSI-Logikkomplexe dar. Sein Aufbau ist nicht auf eine spezielle Funktion zugeschnitten, sondern hat den universellen Charakter der zentralen Verarbeitungseinheit (ZVE) eines Digitalrechners. Träger der speziellen Funktion in einem Einsatzfall ist ein Programm (Software), das aus einzelnen Befehlen des jeweiligen Mikroprozessortyps zusammengesetzt ist und in dem Programmspeicher des Mikrorechners abgelegt wird. Zur Speicherung von Daten, die bei der Programmabarbeitung im MP benutzt werden, wird ein Datenspeicher verwendet. Als Programm- und Datenspeicher finden hochintegrierte Halbleiterspeicherbauelemente Verwendung: Lesespeicherbauelemente ROM/EPROM (ROM read only memory / EPROM electrically programmable read only memory) oder Lese-/Schreibspeicherbauelemente RAM (RAM random access memory).
Entsprechend dem im Programm implementierten Algorithmus werden über angeschlossene Ein-/Ausgabeperipheriebauelemente vom Mikroprozessor Daten mit der Prozeß- und Bedienperipherie ausgetauscht. Ein-/Ausgabeperipheriebauelemente von Mikroprozessorsystemen sind ganz verschiedenartig einsetzbar. Sie werden durch Steuerbefehle auf eine spezielle Aufgabe zugeschnitten (programmiert).
Moderne MP-Systeme enthalten zusätzliche LSI-Funktionskomponenten, zum Beispiel zur Interruptbehandlung (interrupt controler), zur effektiven Verwaltung großer Speicherräume (memory management unit), zur Bussteuerung (bus arbiter) und für anderes mehr /3/./4/./5/.

2. 8-Bit-Mikroprozessorsystem U 880

Das 8-Bit-Mikroprozessorsystem U880 besteht im wesentlichen aus dem folgenden Grundbausteinsortiment (Bild 2.1.):

U880-CPU	zentrale Verarbeitungseinheit (central processor unit)
U855-PIO	paralleler Ein-/Ausgabebaustein (parallel input output)
U856-SIO	serieller Ein-/Ausgabebaustein (seriel input output)
U857-CTC	Zähler-/Zeitgeberbaustein (counter timer circuit)
U858-DMA	Baustein für direkten Speicherzugriff (direct memory access)

Der Mikroprozessor U880 und die Ein-/Ausgabeperipheriebausteine PIO, SIO, CTC und DMA werden in N-Kanal-MOS-Technologie hergestellt. Ein 2,5 MHz (U880) bzw. 4 MHz (UA880) Einphasentakt wird von den Schaltkreisen als externe Taktfrequenz benötigt /1/./3/./6/./7/./8/./9/.

2.1. Zentrale Verarbeitungseinheit U 880-CPU

Die U880-CPU übernimmt als Kernstück des gesamten U880-Bausteinsortimentes die Systemüberwachung, die Befehlsabarbeitung und den überwiegenden Teil der Interruptsteuerung. Es handelt sich hier um einen Digitalrechnerkern, der mit Befehlswörtern variabler Länge - 1, 2, 3 oder 4 Byte - standardmäßig Datenwörter von 8 Bit, entsprechend einem Byte, bearbeiten kann aber auch einzelne Bits in einem Byte, zwei 4-Bit-BCD-Zahlen in einem Byte und 16-Bit-Worte behandelt. Die Befehlszykluszeit liegt zwischen 1,2 und 1,6 µs (2,5 MHz) bzw. 0,75 und 1,0 µs (4 MHz). Die Möglichkeit, Bitoperationen, BCD-Operationen (BCD binary code decimal) und 16-Bit-Operationen ausführen zu können, ist eines der wesentlichen Merkmale des Systems U880. In den Mikroprozessor U880 integriert sind große Teile des Vektorinterruptsystems. Der Rest befindet sich in den Ein-/Ausgabebausteinen PIO, SIO, CTC und DMA. Die Interruptreaktionszeit des U880 liegt bei 8 µs (2,5 MHz) bzw. bei 5 µs (4 MHz).

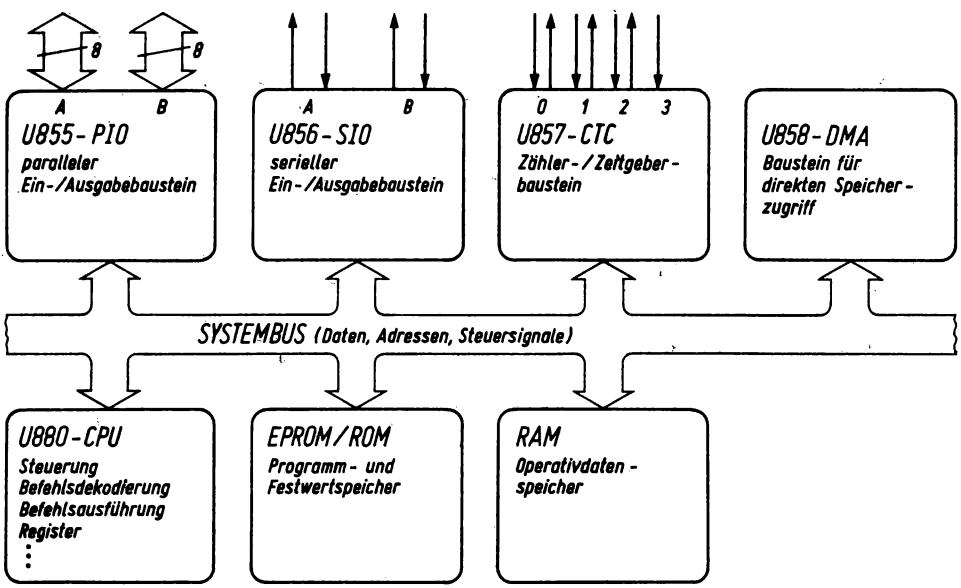


Bild 2.1. Grundstruktur eines U880-Mikrorechners

2.1.1. Adreßraum

Der Mikroprozessor U880 hat einen linear durchgängigen Adreßraum von 65536 Byte entsprechend 64 KByte (0000H bis OFFFFH / H hexadezimal).

2.1.2. Registersatz

Es existiert im U880 ein zweigeteilter Registersatz - Hauptregister/Zweitregister - mit je sechs allgemeinen Registern, die wahlweise als 8-Bit-

Register oder als 16-Bit-Registerpaare benutzt werden können, mit je einem Akkumulator, in dem die arithmetischen und logischen Operationen ausgeführt werden und mit je einem Flagregister zur Statusidentifikation.
Neben diesem Notizblockspeicherregister existieren zwei 16-Bit-Indexregister (IX und IY) zur Realisierung indizierter Adressierungsabläufe, ein 16-Bit-Stackpointer (SP) zur Lokalisierung des LIFO-Stackspeicherbereichs (LIFO last in first out) im RAM für Unterprogramm- bzw. Interruptserviceroutinenabläufe u.a.m., ein Befehlszähler (PC program counter), ein Interruptvektorregister (I) (s. a. Abschnitt 2.1.4.) und ein Speicherrefreshregister (R), das für den Programmierer allerdings ohne Bedeutung ist (Bild 2.2.).

2.1.3. Flagbits

Im Ergebnis der Befehlsausführung werden - bis auf einige Befehle, die die Flags nicht beeinflussen - je nach Operation alle oder ein Teil der im F-Register (F'-Register) des Mikroprozessors U880 als Statusindikatoren zusammengefaßten Flagbits aktualisiert.

F(F')-Register

D7	D6	D5	D4	D3	D2	D1	D0
S	Z	x	H	x	P/V	N	C

Bei der Programmabarbeitung können vier dieser Bits (C, Z, P/V und C) durch die Verwendung bedingungsabhängiger (TRUE/FALSE) Sprung-, Call- und Returnbefehle abgetestet werden. Dadurch wird ein vom Ergebnis der Operation abhängiger Programmablauf organisiert. Zwei Flags (H und N) dienen zur Realisierung der BCD-Arithmetik und zwei Bitpositionen (D3 und D5) im F(F')-Register sind nicht besetzt.

C Übertragsflag (carry)
Z Nullflag (zero)
P/V Paritäts-/Überlaufsflag (parity/overflow)
S Vorzeichenflag (sign).
H Halbübertragsflag (half carry)
N Additions-/Subtraktionsflag (add/subtract).

C-Flag - Das Übertragsflag (C oder auch CY) eröffnet dem Programmierer die Möglichkeit, Additions- und Subtraktionsroutinen höherer Genauigkeit aufzubauen, die über mehrere Bytes laufen. Der Übertrag bei der Addition (und das Borgen bei der Subtraktion) von der Bitposition D7 eines niederwertigen Datenbytes auf die Bitposition D0 eines höherwertigen Datenbytes läuft immer über das Übertragsflag.

Z-Flag - Das Zeroflag (Z) signalisiert, ob im Ergebnis einer Operation der Wert Null entstanden ist (Z=1) oder nicht (Z=0).

P/V-Flag - Das P/V-Flag wird unterschiedlich, in Abhängigkeit von bestimmten Befehlsgruppen, benutzt.
Bei den logischen und Verschiebebefehlen zeigt es die Parität des Ergebnisses an. Ist die Anzahl der gesetzten Bits innerhalb eines Bytes gerade (even), wird P/V=1; ist sie ungerade (odd), wird P/V=0 gesetzt.
Bei den arithmetischen Befehlen wird das P/V-Flag als Vorzeichenüberlaufskennzeichen benutzt. Es zeigt an, ob das Ergebnis der Operation zweier vorzeichenbehafteter ganzer Zahlen (1 Bit Vorzeichen / 7 Bit Daten) innerhalb des zulässigen Zahlenbereichs liegt. Ist das Resultat einer arithmetischen Operation größer als die größtmögliche positive Zahl (+127) oder kleiner als die kleinstmögliche negative Zahl (-128), wird dieses Flag gesetzt (P/V=1), anderenfalls rückgesetzt (P/V=0).

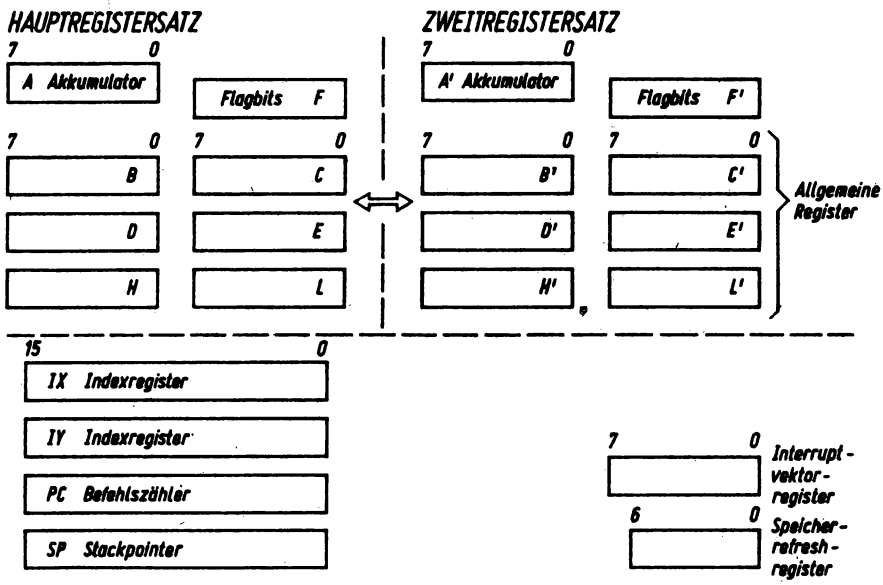


Bild 2.2. Registerstruktur des Mikroprozessors U880

S-Flag - Das Vorzeichenflag (S) enthält nach den arithmetischen und logischen Operationen den Inhalt des höchstwertigen Akkumulatorbits - also des Vorzeichenidentifikators.

H-Flag - Genauso wie das C-Flag nach der Ausführung von arithmetischen Befehlen einen Übertrag vom Bit D7 auf das nicht vorhandene Akkumulatorbit D8 signalisiert, zeigt das Halbübertragsflag (H) einen Übertrag vom Bit D3 auf Bit D4 im Akkumulator an.

N-Flag - Das N-Flag kennzeichnet, ob eine Addition oder Subtraktion als letzter Befehl ausgeführt wurde. Dadurch kann von einem nachfolgenden BCD-Normalisierungsbefehl (decimal adjust) entschieden werden, ob ein gesetztes H-Flag einen Überlauf (ADD) oder ein Borgen (SUB) darstellt.

2.1.4. Interruptsystem

Das im Mikroprozessorsystem U880 implementierte Interruptsystem hat zwei voneinander unabhängige Interruptsignaleingänge:

NMI nichtmaskierbarer Interrupt höchster Priorität
INT maskierbarer Interrupt mit drei verschiedenen Betriebsarten

Ein auf dem INT-Signal aufbauendes Interruptsystem kann vom Programm über dafür bereitstehende Befehle ein- (EI enable interrupt) und ausgeschaltet (DI disable interrupt), sowie auf drei auswählbare Betriebsarten (Mode 0, Mode 1 oder Mode 2) zugeschnitten werden (Bild 2.3.).

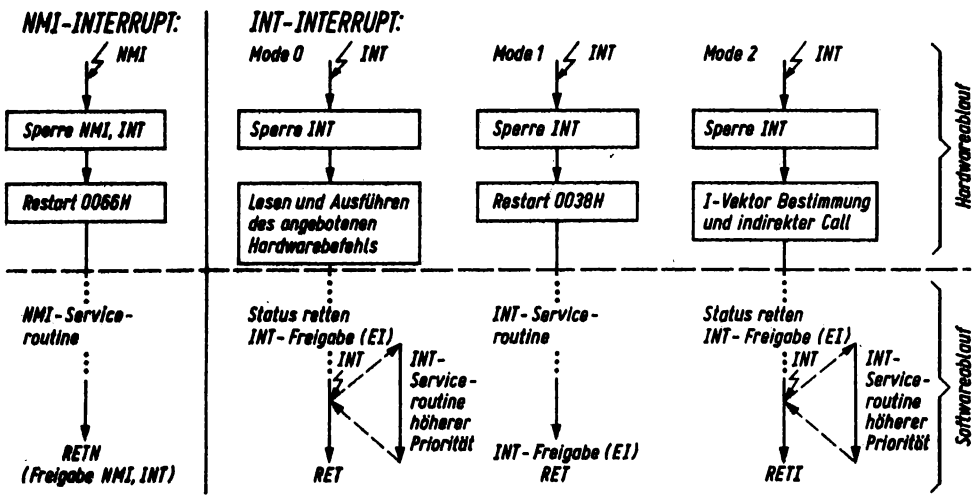


Bild 2.3. Interruptstruktur des Mikroprozessorsystems U880

Die speziell für die LSI-Bausteine des Mikroprozessorsystems U880 vorgesehene Betriebsart ist der Interruptmode 2. Der auf das Interruptsignal folgende Ansprung einer Interruptserviceroutine erfolgt hier über einen 16-Bit-Vektor. Dieser Zeiger setzt sich aus dem Inhalt des I-Registers der U880-CPU als High-Teil (obere 8 Bit) und aus einem variablen, vom interruptauslösenden PIO, SIO, CTC oder DMA kommenden Low-Teil (untere 8 Bit) zusammen. Er stellt die indirekte Ansprungadresse der ausgewählten Interruptserviceroutine dar. Alle Interruptansprungadressen werden in einer Tabelle zusammengefaßt. Diese Startadressentabelle wird durch den Inhalt des Interruptvektorregisters I im Speicher lokalisiert und kann, da für jede Startadresse zwei Byte benötigt werden, maximal 128 verschiedene Interruptserviceroutinen umfassen (Bild 2.4.).

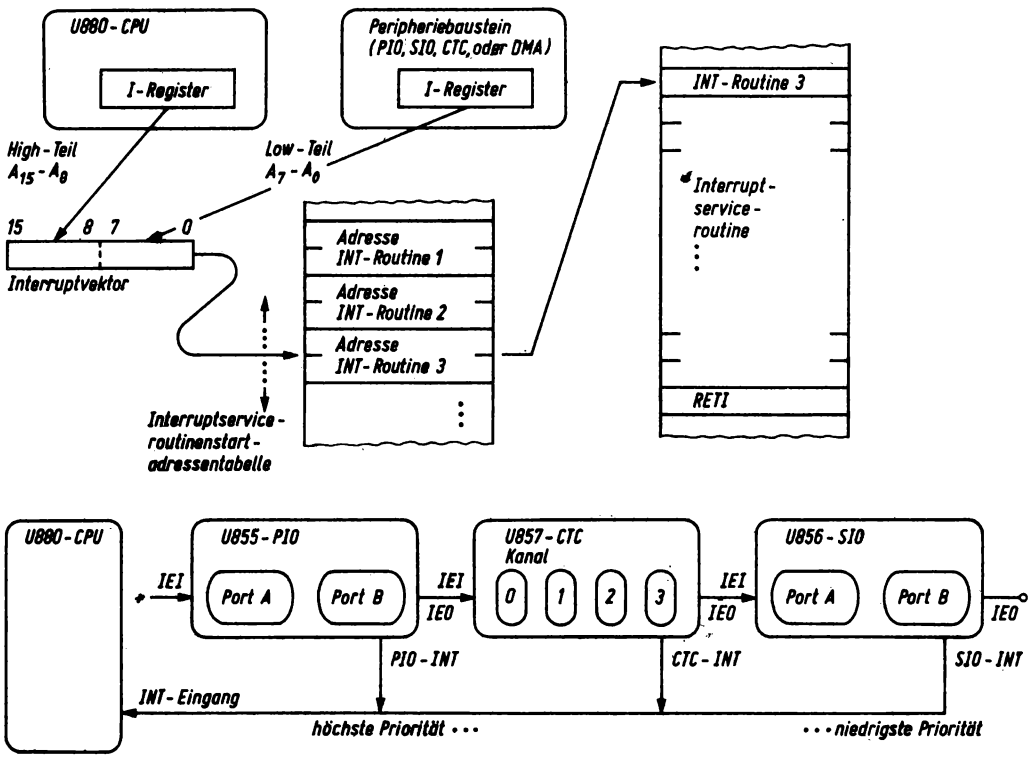


Bild 2.4. Auswahl der Interruptserviceroutine im Interruptmode 2 und Beispiel einer Interruptprioritätskaskade von Peripheriebausteinen

Die Festlegung der Vorrangstruktur der Interruptquellen erfolgt im Mode 2 über Prioritätscontroller in den Peripheriebausteinen, die in einer Prioritätskette (daisy chain) zusammengeschaltet werden. Wird ein Baustein über ein Interruptsignal in dieser Kette aktiv, können nur Bausteine höherer Priorität das entsprechende Behandlungsprogramm erneut unterbrechen.

2.1.5. Adressierungsarten

Der Befehlsatz des Mikroprozessors U880 beinhaltet sechs Adressierungsarten:

direkte Adressierung	Der Operationskode beinhaltet vollständig die Operandenadresse.
implizite Adressierung	Der Operationskode bezieht sich fest auf bestimmte Operanden (Speicherplätze oder Register).
unmittelbare Adressierung	Der Operationskode beinhaltet unmittelbar eine 8- oder 16-Bit-Operandenkonstante.
indirekte Adressierung	Die vollständige 16-Bit-Adresse des Operanden befindet sich in einem CPU-Registerpaar.
indizierte Adressierung	Der Operationskode beinhaltet ein vorzeichenbehaftetes Datenbyte (7 Bit Daten / 1 Bit Vorzeichen), das sogenannte Displacement, das, zum Inhalt des Indexregisters IX oder IY addiert, die vollständige 16-Bit-Adresse des Operanden ergibt (Basisadresse-128 bis Basisadresse+127).
relative Adressierung	Der Operationskode beinhaltet ein vorzeichenbehaftetes Datenbyte (7 Bit Daten / 1 Bit Vorzeichen), die sogenannte Distanzadresse, das, zum Inhalt des Befehlszählers PC addiert, die vollständige 16-Bit-Adresse des Operanden ergibt (PC-126 bis PC+129).

2.1.6. Befehlssatz

Die Leistungsfähigkeit eines Mikroprozessors wird neben der Operationsgeschwindigkeit und der Verarbeitungsbreite wesentlich bestimmt durch die Struktur und die Flexibilität seines Befehlssatzes (Tafel 2.1.). Der Befehlssatz des U880 läßt sich in 13 Basisgruppen einteilen:

- 8-Bit-Ladebefehle
- 16-Bit-Ladebefehle
- Stack- und Stackpointerbefehle
- Exchangebefehle
- Blocktransfer- und Blocksuchbefehle
- 8-Bit-Arithmetik-/Logikbefehle
- 16-Bit-Arithmetikbefehle
- Rotations- und Verschiebebefehle
- Bitbefehle
- Sprungbefehle
- Call- und Returnbefehle
- Ein-/Ausgabebefehle
- CPU-Steuerbefehle

Die Ladebefehle transportieren Daten von Register zu Register oder zwischen Register und Speicher. Alle Ladebefehle müssen durch die Angabe einer Datenquelle (source) und eines Datenziels (destination) spezifiziert werden. Die Stack- und Stackpointerbefehle beziehen sich auf den LIFO-

Stackspeicherbereich (LIFO last in - first out) für die Unterprogramm- und Interruptorganisation. Die Exchangebefehle erschließen dem Programmierer neben dem Hauptregistersatz auch den Zweitregistersatz der U880-CPU. Die Blocktransferbefehle ermöglichen den Transport eines in seiner Größe spezifizierten Datenblockes von einem Speicherbereich in einen anderen Speicherbereich. Die Blocksuchbefehle gestatten das Durchmustern eines Speicherbereiches auf ein bestimmtes Datenbyte.
Die Arithmetik-/Logikbefehle arbeiten mit einem Akkumulator (A, HL, IX oder IY), der einen der beiden Operanden und nach Ausführung des Befehls, das Ergebnis enthält. Der zweite Operand kann aus einem Register oder aus dem Speicher kommen. Im Ergebnis der Operation werden die Flagbits verändert.
Eine besonders wichtige Befehlsgruppe stellen die Rotations- und Verschiebebefehle dar. Jeder Multiplikations- oder Divisionsalgorithmus, viele Konvertierungsprogramme und alle aufeinanderfolgenden Bittestoperationen in einem Datenbyte werden sich dieser Befehle bedienen. Die Bitmanipulationsbefehle erlauben, mit einem einzigen Befehl ein zu spezifizierendes Bit in einem Notizblockregister der CPU oder im Speicher zu setzen (=1), rückzusetzen (=0) oder abzutesten. Sprung- ebenso wie Call- und Returnbefehle werden verwendet, um jeweils bestimmte Programmsequenzen im Lesespeicher zur Arbeit zu aktivieren. Call und Return sind dabei der Unterprogrammorganisation vorbehalten.
Die Ein-/Ausgabebefehle ermöglichen dem Mikroprozessorsystem die Kommunikation mit seiner Umwelt, sei es nun Datenverarbeitungsperipherie oder Prozeßperipherie. Die Steuerbefehle schließlich gestatten dem Programmierer, das Interruptsystem des U880 seinen speziellen Softwareanforderungen anzupassen.

2.1.7. Ein-/Ausgabe

Es existieren zwei verschiedene Möglichkeiten, Ein-/Ausgabeperipheriegeräte an ein U880-Mikroprozessorsystem anzuschließen. Zum einen kann einem Peripheriegerät eine der 65536 (64K) möglichen Speicheradressen zugeordnet werden (memory mapped). Datenwörter sind dann mit Speicherlese- oder Speicherschreibbefehlen (Signal: MREQ memory request) ein- bzw. auszugeben. Die zweite Möglichkeit ist mit den INPUT/OUTPUT-Befehlen (Signal: IORQ input/output request) gegeben. Diese Befehle wenden sich an standardmäßig 256 (maximal 65536) Ein-/Ausgabekanäle (E/A-Ports).

Tafel 2.1. Befehlsliste des Mikroprozessors U880

ZEICHENERLÄUTERUNG

r,r'	8-Bit-Register	A=111 / B=000 / C=001 D=010 / E=011 H=100 / L=101
dd, qq, pp, rr	16-Bit-Registerpaar	dd BC=00 / DE=01 / HL=10 / SP=11 qq BC=00 / DE=01 / HL=10 / AF=11 pp BC=00 / DE=01 / IX=10 / SP=11 rr BC=00 / DE=01 / IY=10 / SP=11
t	r, (HL), (IX+d) oder (IY+d)	
n	8-Bit-Binärzahl / nn 16-Bit-Binärzahl	
d	8-Bit-Displacement (-128 bis +127)	
e	8-Bit-Distanzadresse (-126 bis +129)	
b	Bitposition	D0=000 / D1=001 / D2=010 / D3=011 D4=100 / D5=101 / D6=110 / D7=111
cc	Sprungbedingung	NZ=000 (non zero) Z=001 (zero) NC=010 (non carry) C=011 (carry) PO=100 (parity odd) PE=101 (parity even) P=110 (sign positive) M=111 (sign negative)
p	Restartadressen	p=00H p=08H p=10H p=18H p=20H p=28H p=30H p=38H
Flagbitbeeinflussung		. unverändert X unbestimmt ? entsprechend dem Ergebnis gesetzt V Overflowfunktion des Paritätsflagbit P Parityfunktion des Paritätsflagbit IF Inhalt des IFF2 Interruptspeicherbits bei NMI
TxZ=Befehlsausführungszeit in ns (Z=400 2,5MHz U880 / Z=250 4MHz UA880)		

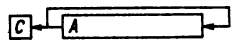
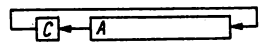
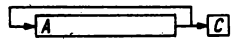
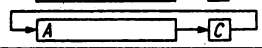
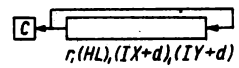
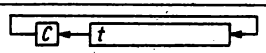
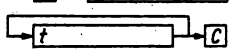
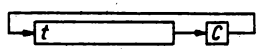
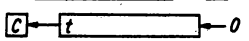
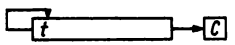
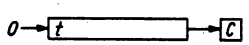
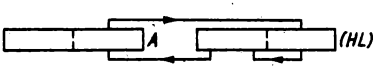
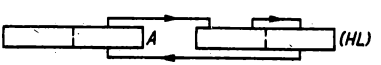
8-BIT-LADEBEFEHLE

Befehl	T	Kode	Erläuterung	C Z P S N H
LD r,r'	4	01 r r'	r:=r'	. . . . .
LD r,(HL)	7	01 r 110	r:=(HL)	. . . . .
LD (HL),r	7	01 110 r	(HL):=r	. . . . .
LD r,n	7	00 r 110 - n -	r:=n	. . . . .
LD (HL),n	10	00 110 110 - n -	(HL):=n	. . .
LD r,(IX+d)	19	11 011 101 01 r 110 - d -	r:(IX+d)	. . . . .
LD r,(IY+d)	19	11 111 101 01 r 110 - d -	r:=(IY+d)	. . . . .
LD (IX+d),r	19	11 011 101 01 110 r - d -	(IX+d):=r	. . . . .
LD (IY+d),r	19	11 111 101 01 110 r - d -	(IY+d):=r	. . . . .
LD (IX+d),n	19	11 011 101 00 110 110 - d - - n -	(IX+d):=n	. . . . .
LD (IY+d),n	19	11 111 101 00 110 110 - d - - n -	(IY+d):=n	. . . . .

Befehl	T	Kode	Erläuterung	C Z P S N H
LD A,(BC)	7	00 001 010	A:=(BC)	
LD A,(DE)	7	00 011 010	A:=(DE)	
LD A,(nn)	13	00 111 010 - n - - n -	A:=(nn)	
LD (BC),A	7	00 000 010	(BC):=A	
LD (DE),A	7	00 010 010	(DE):=A	
LD (nn),A	13	00 110 010 - n - - n -	(nn):=A	
LD A,I	9	11 101 101 01 010 111	A:=I	. ? IF? 0 0
LD A,R	9	11 101 101 01 011 111	A:=R	. ? IF? 0 0
LD I,A	9	11 101 101 01 000 111	I:=A	
LD R,A	9	11 101 101 01 001 111	R:=A	
16-BIT-LADEBEFEHLE				
LD dd,nn	10	00 dd0 001 - n - - n -	dd:=nn	
LD IX,nn	14	11 011 101 00 100 001 - n - - n -	IX:nn	
LD IY,nn	14	11 111 101 00 100 101 - n - - n -	IY:=nn	
LD HL,(nn)	16	00 101 010 - n - - n -	H:=(nn+1) L:=(nn)	. . . . .
LD dd,(nn)	20	11 101 101 01 dd1 011 - n - - n -	dd_H:=(nn+1) dd_L:=(nn)	
LD IX,(nn)	20	11 011 101 00 101 010 - n - - n -	IX_H:=(nn+1) IX_L:=(nn)	
LD IY,(nn)	20	11 111 101 00 101 010 - n - - n -	IY_H:=(nn+1) IY_L:=(nn)	
LD (nn),HL	16	00 100 010 - n - - n -	(nn+1):=H (nn):=L	
LD (nn),dd	20	11 101 101 01 dd0 011 - n - - n -	(nn+1):=dd_H (nn):=dd_L	
LD (nn),IX	20	11 011 101 00 100 010 - n - - n -	(nn+1):=IX_H (nn):=IX_L	
LD (nn),IY	20	11 111 101 00 100 010 - n - - n -	(nn+1):=IY_H (nn):=IY_L	

STACK- UND STACKPOINTERBEFEHLE				
LD SP,HL	6	11 111 001	SP:=HL	
LD SP,IX	10	11 011 101 11 111 001	SP:=IX	
LD SP,IY	10	11 111 101 11 111 001	SP:=IY	
PUSH qq	11	11 qq0 101	$(SP-2):=qq_L$ $(SP-1):=qq_H$	. . . . .
PUSH IX	15	11 011 101 11 100 101	$(SP-2):=IX_L$ $(SP-1):=IX_H$	. .
PUSH IY	15	11 111 101 11 100 101	$(SP-2):=IY_L$ $(SP-1):=IY_H$	
POP qq	10	11 qq0 001	$qq_H:=(SP+1)$ $qq_L:=(SP)$	
POP IX	14	11 011 101 11 100 001	$IX_H:=(SP+1)$ $IX_L:=(SP)$	
POP IY	14	11 111 101 11 100 001	$IY_H:=(SP+1)$ $IY_L:=(SP)$	
EXCHANGEBEFEHLE				
EX DE,HL	4	11 101 011	DE:==:HL	
EX AF,AF'	4	00 001 000	AF:==:AF'	
EXX	4	11 011 001	BC:==:BC' DE:==:DE' HL:==:HL'	
EX (SP),HL	19	11 100 011	H:==:(SP+1) L:==:(SP)	
EX (SP),IX	23	11 011 101 11 100 011	IX_H:==:(SP+1) IX_L:==:(SP)	
EX (SP),IY	23	11 111 101 11 100 011	IY_H:==:(SP+1) IY_L:==:(SP)	. . . .
BLOCKTRANSFER- UND BLOCKSUCHBEFEHLE				
LDI	16	11 101 101 10 100 000	(DE):=(HL) dann DE:=DE+1 HL:=HL+1 BC:=BC-1	. . ? . 0 0 BC-1=0 P=0 BC-1≠0 P=1
LDIR	21 (16)	11 101 101 10 110 000	(DE):=(HL) dann DE:=DE+1 HL:=HL+1 BC:=BC-1 bis BC=0	. 0 . 0 0
LDD	16	11 101 101 10 101 000	(DE):=(HL) dann DE:=DE-1 HL:=HL-1 BC:=BC-1	. . ? . 0 0 BC-1=0 P=0 BC-1≠0 P=1
LDDR	21 (16)	11 101 101 10 111 000	(DE):=(HL) dann DE:=DE-1 HL:=HL-1 BC:=BC-1 bis BC=0	. . 0 . 0 0
CPI	16	11 101 101 10 100 001	A-(HL)=? dann HL:=HL+1 BC:=BC-1	. ? ? ? 1 ? A=(HL) Z=1 A≠(HL) Z=0 BC-1=0 P=0 BC-1≠0 P=1
CPIR	21 (16)	11 101 101 10 110 001	A-(HL)=? dann HL:=HL+1 BC:=BC-1 bis BC=0 oder A=(HL)	. ? ? ? 1 ? A=(HL) Z=1 A≠(HL) Z=0 BC-1=0 P=0 BC-1≠0 P=1
CPD	16	11 101 101 10 101 001	A-(HL)=? dann HL:=HL-1 BC:=BC-1	. ? ? ? 1 ? A=(HL) Z=1 A≠(HL) Z=0 BC-1=0 P=0 BC-1≠0 P=1
CPDR	21 (16)	11 101 101 10 111 001	A-(HL)=? dann HL:=HL-1 BC:=BC-1 bis BC=0 oder A=(HL)	. ? ? ? 1 ? A=(HL) Z=1 A≠(HL) Z=0 BC-1=0 P=0 BC-1≠0 P=1

Befehl	T	Kode	Erläuterung	C Z P S N H
8-BIT-ARITHMETIK-/LOGIKBEFEHLE				
ADD A,r	4	10 000 r	A:=A+r	? ? V ? 0 ?
ADD A,n	7	11 000 110 - n -	A:=A+n	? ? V ? 0 ?
ADD A,(HL)	7	10 000 110	A:=A+(HL)	? ? V ? 0 ?
ADD A,(IX+d)	19	11 011 101 10 000 110 - d -	A:=A+(IX+d)	? ? V ? 0 ?
ADD A,(IY+d)	19	11 111 101 10 000 110 - d -	A:=A+(IY+d)	? ? V ? 0 ?
ADC A,s		001	A:=A+s+CY s kann sein:	? ? V ? 0 ?
SUB s		010	A:=A-s r,n,(HL),	? ? V ? 1 ?
SBC A,s		011	A:=A-s-CY (IX+d) oder (IY+d)	? ? V ? 1 ?
AND s		100	A:=A AND s Kodierung	0 ? P ? 0 1
OR s		110	A:=A OR s entspricht ADD	0 ? P ? 0 0
XOR s		101	A:=A XOR s 000 wird	0 ? P ? 0 0
CP s		111	A-s=? substituiert	? ? V ? 1 ?
INC r	4	00 r 100	r:=r+1	. ? V ? 0 ?
INC (HL)	11	00 110 100	(HL):=(HL)+1	. ? V ? 0 ?
INC (IX+d)	23	11 011 101 00 110 100 - d -	(IX+d):=(IX+d)+1	. ? V ? 0 ?
INC (IY+d)	23	11 111 101 00 110 100 - d -	(IY+d):=(IY+d)+1	. ? V ? 0 ?
DEC t		101	t:=t-1 t kann sein: r,(HL),(IX+d) oder (IY+d) Kodierung entspricht INC 100 wird substituiert	. ? V ? 1 ?
DAA	4	00 100 111	Konvertierung A in BCD DECIMAL ADJUST	? ? P ? . ?
CPL	4	00 101 111	A:=A negiert	. . . . 1 1
NEG	8	11 101 101 01 000 100	A:=-A	? ? V ? 1 ?
CCF	4	00 111 111	C-Flag:=C-Flag negiert	? . . . 0 X
SCF	4	00 110 111	C-Flag:=1	1 . . . 0 0
16-BIT-ARITHMETIKBEFEHLE				
ADD HL,dd	11	00 dd1 001	HL:=HL+dd	? . . . 0 X
ADC HL,dd	15	11 101 101 01 dd1 010	HL:=HL+dd+CY	? ? V ? 0 X
SBC HL,dd	15	11 101 101 01 dd0 010	HL:=HL-dd-CY	? ? V ? 1 X
ADD IX,pp	15	11 011 101 00 pp1 001	IX:=IX+pp	? . . . 0 X
ADD IY,rr	15	11 111 101 00 rr1 001	IY:=IY+rr	? . . . 0 X
INC dd	6	00 dd0 011	dd:=dd+1	. . . . . .
INC IX	10	11 011 101 00 100 011	IX:=IX+1	. . . . . .
INC IY	10	11 111 101 00 100 011	IY:=IY+1	. . . . . .
DEC dd	6	00 dd1 011	dd:=dd-1	. . . . . .
DEC IX	10	11 011 101 00 101 011	IX:=IX-1	. . . . . .
DEC IY	10	11 111 101 00 101 011	IY:=IY-1	. . . . . .

ROTATIONS- UND VERSCHIEBEBEFEHLE				
RLCA	4	00 000 111		? . . . 0 0
RLA	4	00 010 111		? . . . 0 0
RRCA	4	00 001 111		? . . . 0 0
RRA	4	00 011 111		? . . . 0 0
RLC r	8	11 001 011 00 000 r		? ? P ? 0 0
RLC (HL)	15	11 001 011 00 000 110		? ? P ? 0 0
RLC (IX+d)	23	11 011 101 11 001 011 - d - 00 000 110	 r;(HL),(IX+d),(IY+d)	? ? P ? 0 0
RLC (IY+d)	23	11 111 101 11 001 011 - d - 00 000 110		? ? P ? 0 0
RL t		010		? ? P ? 0 0
RRC t		001		? ? P ? 0 0
RR t		011		? ? P ? 0 0
SLA t		100		? ? P ? 0 0
SRA t		101		? ? P ? 0 0
SRL t		111		? ? P ? 0 0
RLD	18	11 101 101 01 101 111		. ? P ? 0 0
RRD	18	11 101 101 01 100 111		. ? P ? 0 0
BITBEFEHLE				
BIT b,r	8	11 001 011 01 b r	$Z:=r_b$ negiert	? X X 0 1
BIT b,(HL)	12	11 001 011 01 b 110	$Z:=(HL)_b$ negiert	. ? X X 0 1
BIT b,(IX+d)	20	11 011 101 11 001 011 d 01 b 110	$Z:=(IX+d)_b$ negiert	. ? X X 0 1
BIT b,(IY+d)	20	11 111 101 11 001 011 - d - 01 b 110	$Z:=(IY+d)_b$ negiert	. ? X X 0 1
SET b,r	8	11 001 011 11 b r	$r_b:=1$	. . . . .
SET b,(HL)	15	11 001 011 11 b 110	$(HL)_b:=1$	
SET b,(IX+d)	23	11 011 101 11 001 011 - d - 11 b 110	$(IX+d)_b:=1$	

Befehl	T	Kode	Erläuterung	C Z P S N H
SET b,(IY+d)	23	11 111 101 11 001 011 - d - 11 b 110	$(IY+d)_b$:=1	. . . . .
RES b,t		10	t_b:=0 t kann sein: r, (HL), (IX+d) oder (IY+d) Kodierung entspr. SET Befehl 11 wird substituiert	. . . . .
SPRUNGBEFEHLE				
JP nn	10	11 000 011 - n - - n -	PC:=nn	. . . . .
JP cc,nn	10	11 cc 010 - n - n -	PC:=nn wenn Bedingung cc erfüllt PC:=PC+3 wenn Bedingung cc nicht erfüllt	. . . . .
JR e	12	00 011 000 - e-2 -	PC:=PC+e	. . . . .
JR C,e	12 (7)	00 111 000 - e-2 -	PC:=PC+e wenn C=1 PC:=PC+2 wenn C=0	.
JR NC,e	12 (7)	00 110 000 - e-2 -	PC:=PC+e wenn C=0 PC:=PC+2 wenn C=1	
JR Z,e	12 (7)	00 101 000 - e-2 -	PC:=PC+e wenn Z=1 PC:=PC+2 wenn Z=0	.
JR NZ,e	12 (7)	00 100 000 - e-2 -	PC:=PC+e wenn Z=0 PC:=PC+2 wenn Z=1	
JP (HL)	4	11 101 001	PC:=HL	. . . . .
JP (IX)	8	11 011 101 11 101 001	PC:=IX	. . . . .
JP (IY)	8	11 111 101 11 101 001	PC:=IY	
DJNZ e	13 (8)	00 010 000 - e-2 -	B:=B-1 wenn dann B=0 dann PC:=PC+2 wenn dann B≠0 dann PC:=PC+e	. . . . .
CALL- UND RETURNBEFEHLE				
CALL nn	17	11 001 101 - n - - n -	(SP-1):=PC_H (SP-2):=PC_L PC:=nn	.
CALL cc,nn	17 (10)	11 cc 100 - n - - n -	(SP-1):=PC_H (SP-2):=PC_L PC:=nn wenn cc TRUE PC:=PC+3 wenn cc FALSE	
RET	10	11 001 001	PC_L:=(SP) PC_H:=(SP+1)	
RET cc	11 (5)	11 cc 000	PC_L:=(SP) PC_H:=(SP+1) PC:=PC+1 wenn cc TRUE	
RETI	14	11 101 101 01 001 101	RETURN vom INT	. . . . .
RETN	14	11 101 101 01 000 101	RETURN vom NMI	
RST p	11	11 a 111	(SP-1):=PC_H (SP-2):=PC_L PC_H:=0 PC_L:=p	

EIN-/AUSGABEBEFEHLE				
IN A,(n)	11	11 011 011 - n -	A:=(n) ADR:=A/n	. . . . .
IN r,(C)	12	11 101 101 01 r 000	r:=(C) ADR:=B/C	. ? P ? o ?
INI	16	11 101 101 10 100 010	(HL):=(C) ADR:=B/C dann B:=B-1 HL:=HL+1	. ? X X 1 X B-1=o Z=1 B-1≠o Z=o
INIR	21 (16)	11 101 101 10 110 010	(HL):=(C) ADR:=B/C dann B:=B-1 HL:=HL+1 bis B=o	. 1 X X 1 X
IND	16	11 101 101 10 101 010	(HL):=(C) ADR:=B/C dann B:=B-1 HL:=HL-1	. ? X X 1 X B-1=o Z=1 B-1≠o Z=o
INDR	21 (16)	11 101 101 10 111 010	(HL):=(C) ADR:=B/C dann B:=B-1 HL:=HL-1 bis B=o	. 1 X X 1 X
OUT (n),A	11	11 010 011 - n -	(n):=A ADR:=A/n	. . . . .
OUT (C),r	12	11 101 101 01 r 001	(C):=r ADR:=B/C	. . . . .
OUTI	16	11 101 101 10 100 011	(C):=(HL) ADR:=B/C dann B:=B-1 HL:=HL+1	. ? X X 1 X B-1=o Z=1 B-1≠o Z=o
OTIR	21 (16)	11 101 101 10 110 011	(C):=(HL) ADR:=B/C dann B:=B-1 HL:=HL+1 bis B=o	. 1 X X 1 X
OUTD	16	11 101 101 10 101 011	(C):=(HL) ADR:=B/C dann B:=B-1 HL:=HL-1	. ? X X 1 X B-1=o Z=1 B-1≠o Z=o
OTDR	21 (16)	11 101 101 10 111 011	(C):=(HL) ADR:=B/C dann B:=B-1 HL:=HL-1 bis B=o	. 1 X X 1 X
CPU-STEUERBEFEHLE				
NOP	4	00 000 000	NO OPERATION	. . . . .
HALT	4	01 110 110	HALT	. . . . .
DI	4	11 110 011	Interrrupt AUS	. . . . .
EI	4	11 111 011	Interrrupt EIN	. . . . .
IM 0	8	11 101 101 01 000 110	Interrupt Mode 0	. . . . .
IM 1	8	11 101 101 01 010 110	Interrupt Mode 1	. . . . .
IM 2	8	11 101 101 01 011 110	Interrupt Mode 2	. . . . .

2.2. Paralleler Ein-/Ausgabebaustein U 855-PIO

Der Anschluß einer Vielzahl von Datenverarbeitungs- und Prozeßperipheriegeräten an einen Mikrorechner kann über ein Parallelinterface erfolgen. Das gilt z.B. für Lochstreifen-Ein-/Ausgabegeräte, Drucker, Tastaturen, Magnatbandspeicher und digitale oder analoge Prozeß-Ein-/Ausgabebaugruppen. Diesen recht unterschiedlichen Geräten ist gemeinsam, daß auf einem parallelen Datenbus Informationen angeboten (Eingabegeräte) oder empfangen (Ausgabegeräte) werden. Die Details des Datenaustausches ebenso wie die den Datenstrom begleitenden Steuersignale sind meist recht unterschiedlich. Im System U880 steht zum Anschluß solcher Ein-/Ausgabekanäle der U855-PIO zur Verfügung. Er dient als Interfaceadapter zwischen Peripherie und U880. Durch die Programmiermöglichkeiten des U855 wird eine dem jeweiligen Einsatzfall entsprechende Modifikation der Arbeitsweise erreicht /6/,/7/,/10/,/11/.
Der parallele Ein-/Ausgabebaustein U855-PIO übernimmt beim Datenaustausch zwischen Mikroprozessor und Peripherie folgende Funktionen:

- Zwischenspeicherung von Ein-/Ausgabedaten, so daß U880 und Ein-/Ausgabegerät nicht starr (synchron) gekoppelt werden müssen.
- Generierung eines Quittungsbetriebsablaufs (handshake) bei der Datenübergabe und bei der Datenübernahme.
- Erzeugung von Interruptanforderungen an die U880-CPU einschließlich der Bereitstellung eines Interruptvektors und der Prioritätsermittlung.

2.2.1. Aufbau

Der U855-PIO besteht im wesentlichen aus zwei unabhängigen 8-Bit-Ports (Bild 2.5.), die den Ein-/Ausgabedatentransfer zwischen CPU und Peripherie in vier verschiedenen Betriebsarten abwickeln:

Byte-Ausgabe (Mode 0)
Byte-Eingabe (Mode 1)
Byte-Ein-/Ausgabe bidirektional (Mode 2)
Bit-Ein-/Ausgabe (Mode 3)

Die Programmierung des U855-PIO erfolgt über vier Adressen: Port A Daten (data), Port B Daten (data), Port A Steuerwörter (control) und Port B Steuerwörter (control).
Dem Programmierer steht beim U855-PIO der interruptgesteuerte Ein-/Ausgabebetrieb und der zeitlich synchronisierte Ein-/Ausgabebetrieb zur Verfügung. Die Anwendung des Abfrageverfahrens (polling) ist nicht möglich, da kein der Software zugängliches Informationsbit existiert, das Auskunft über den Abschluß des Datentransfers PIO - Ein-/Ausgabegerät gibt.

Ausgabemode (Mode 0): Die Betriebsart 'Byte-Ausgabe' dient der Ausgabe eines 8-Bit-Datenwortes vom U880 über den parallelen Ein-/Ausgabebaustein U855 an ein Peripheriegerät. Die Datenübergabe vom U855 an das Peripheriegerät erfolgt nach der Ausgabe der Information von der CPU an den PIO selbständig. Das bei der Datenübergabe zwischen U855 und Ausgabegerät ablaufende Handshakesignalspiel stellt den ordnungsgemäßen Datentransfer sicher. Den Abschluß der Datenübergabe meldet der PIO an die CPU über eine Interruptfertigmeldung.

Eingabemode (Mode 1): Die Betriebsart 'Byte-Eingabe' dient zur Eingabe eines 8-Bit-Datenwortes von einem Peripheriegerät über den parallelen Ein-/Ausgabebaustein U855 in den Mikroprozessor U880. Die Datenübernahme vom Peripheriegerät in den U855 erfolgt wie im Ausgabemode selbständig. Ist die Datenübernahme abgeschlossen, erfolgt im Interruptbetrieb eine INT-Fertigmeldung vom PIO und das im U855 zwischengespeicherte Datenbyte kann von der CPU abgeholt werden.

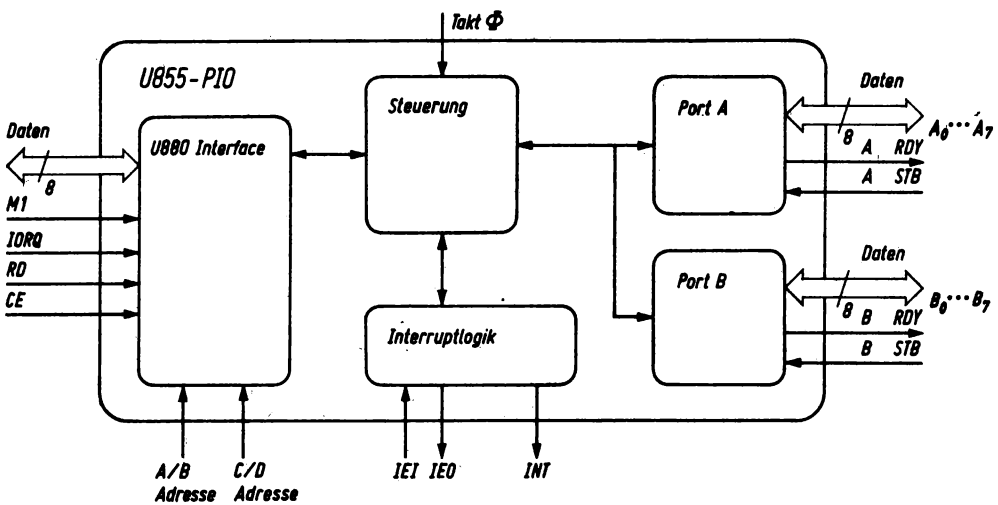


Bild 2.5. Aufbau und Signalstruktur des parallelen Ein-/Ausgabebausteins U855-PIO

Ein-/Ausgabemode (Mode 2): Die Betriebsart 'Byte-Ein-/Ausgabe' dient zur bidirektionalen Eingabe und Ausgabe von 8-Bit-Datenwörtern über den Kanal A des U855. Für die Abwicklung des Datenaustausches werden alle vier Handshakesignalleitungen (RDY A/B und STB A/B) eines PIO-Schaltkreises benötigt. Für die Organisation des Datenaustausches gilt das beim Mode 0 und Mode 1 Gesagte.

Bit-Ein-/Ausgabemode (Mode 3): Die Betriebsart 'Bit-Ein-/Ausgabe' dient zur Eingabe und Ausgabe von Datenwörtern, deren Bits einzeln - je nach Bedarf - in Eingaberichtung oder in Ausgaberichtung geschaltet werden können. Von den in Eingaberichtung geschalteten Bits eines Datenbytes lassen sich bei einer Flankenänderung des anliegenden Signals ('logisch 0' auf 'logisch 1' (HIGH) oder 'logisch 1' auf 'logisch 0' (LOW)) Interruptmeldungen (INT) ableiten.

In der Betriebsart 'Bit-Ein-/Ausgabe' gilt dann:

Fall 1 HIGH/AND	Das letzte auf 1 gehende Eingabebit löst den INT aus.
Fall 2 LOW/AND	Das letzte auf 0 gehende Eingabebit löst den INT aus.
Fall 3 HIGH/OR	Jedes auf 1 gehende Eingabebit löst den INT aus.
Fall 4 LOW/OR	Jedes auf 0 gehende Eingabebit löst den INT aus.

2.2.2. Programmierung

Die Arbeitsweise des U855-PIO in einem speziellen Einsatzfall wird vom Programmierer über eine Reihe von Steuerwortausgaben an die Steuerwortadresse (control) des jeweiligen U855 Kanales festgelegt. Dabei ist zu beachten, daß auch ein einmal programmierter PIO durch erneute Steuerwortausgaben teilweise oder vollständig umprogrammiert werden kann.
Die verschiedenen Steuerwörter des U855-PIO unterteilen sich in zwei Gruppen:

- Steuerwörter, die in einem oder mehreren Bits ein Kennzeichen (Adresse) enthalten, aus dem der PIO entnehmen kann, um was für ein Steuerwort es sich handelt.
- Steuerwörter, die sequentiell auf ein bestimmtes Steuerwort der ersten Gruppe folgen müssen.

Steuerwörter mit Adressenkennzeichen können jederzeit, ohne eine bestimmte Reihenfolge einhalten zu müssen, ausgegeben werden. Steuerwortausgaben mit sequentieller Kennzeichnung müssen vom Programmierer genauestens eingehalten werden. Anderenfalls kommt der PIO-Programmiervorgang vollständig durcheinander. Der Aufbau und die Bitbelegung der Steuerwörter ist der Tafel 2.2. zu entnehmen.

Tafel 2.2. Steuerwortstruktur des U855-PIO

Interruptvektor / Kennzeichen: D0=0

V7	V6	V5	V4	V3	V2	V1	0

Steuerwort Modeauswahl / Kennzeichen: D3...D0=1 (D5,D4 ohne Bedeutung)

M1	M0	X	X	1	1	1	1

M1,M0=00 Mode 0 Byte-Ausgabe
M1,M0=01 Mode 1 Byte-Eingabe
M1,M0=10 Mode 2 Byte-Ein-/Ausgabe
M1,M0=11 Mode 3 Bit-Ein-/Ausgabe

Steuerwort Ein-/Ausgabebits (folgt nur im Mode 3 auf die Modeauswahl)

E/A	E/A	E/A	E/A	E/A	E/A	E/A	E/A

E/A=0 Ausgabebit E/A=1 Eingabebit

Interruptsteuerwort / Kennzeichen: D3=0; D2...D0=1 (D6,D5,D4 nur im Mode 2)

EI/DI	AND/OR	HIGH/LOW	MASKS	0	1	1	1

EI=1 DI=0; AND=1 OR=0; HIGH=1 LOW=0; MASKS=1 Maskenwort folgt

Achtung: In jeder Betriebsart führt das Setzen von D4=1 zum Rücksetzen eines angemeldeten Interrupts !

Interruptmaskenwort (folgt auf das Interruptsteuerwort wenn D4=1)

MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0

MB7, MB6 ... MB0=0 Interrupt ist auf diesem Bit generierbar.
MB7, MB6 ... MB0=1 Interrupt ist auf diesem Bit nicht generierbar.

Interrupt Ein/Aus / Kennzeichen: D3,D2=0; D1,D0=1 (D6,D5,D4 ohne Bedeutung)

DI/EI	X	X	X	0	0	1	1

DI/EI=1 Interrupt Ein DI/EI=0 Interrupt Aus

X = 0 oder 1 - ohne Bedeutung

2.3. Serieller Ein-/Ausgabebaustein U 856-SIO

Der U856-SIO ist ein programmierbarer Interfacebaustein zum Anschluß von seriellen Datenübertragungskanälen an das Mikroprozessorsystem U880. Er ist vorgesehen für die Umwandlung von 8 Bit paralleler in 1 Bit serielle Information in Eingangs- und Ausgangsrichtung über zwei unabhängige 'voll duplex' Ein-/Ausgabekanalpaare /6/,/7/,/12/,/13/.

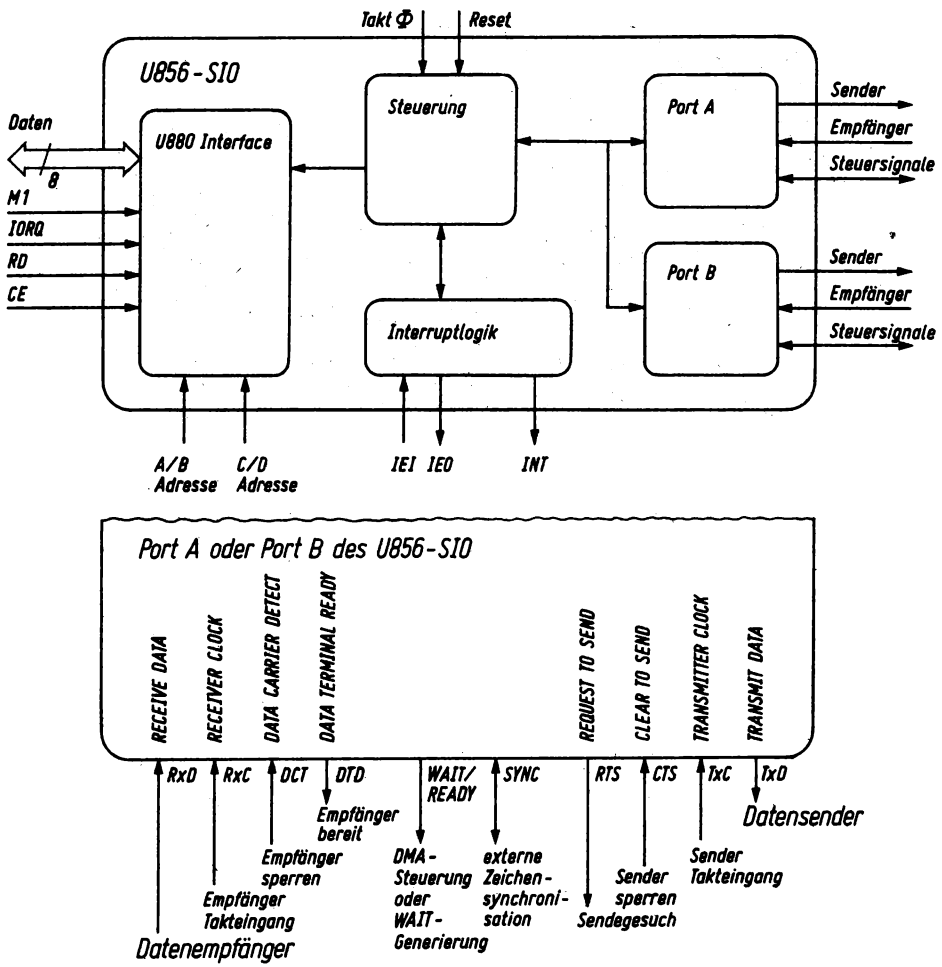


Bild 2.6. Aufbau und Signalstruktur des seriellen Ein-/Ausgabebausteins U856-SIO

Der serielle Schnittstellenbaustein U856-SIO erlaubt die asynchrone und die synchrone serielle Datenübertragung nach verschiedenen Normen, je nach der vom Programm eingestellten Betriebsart. Er ist, wie alle anderen U880-Peripheriebausteine, voll in die Vektorinterruptstruktur dieses MP-Systems eingebunden.

2.3.1. Aufbau

Um die Arbeitsweise des seriellen Ein-/Ausgabebausteins U856-SIO verstehen zu können und um aus der großen Vielfalt der möglichen Betriebsarten die für den gegebenen Einsatzfall zutreffende Initialprogrammierfolge auswählen zu können, muß vor allem die Signalstruktur dieses Bausteins bekannt sein (Bild 2.6.). Neben dem U880-Interface, bestehend aus acht Datenleitungen, zwei Adreßleitungen und einigen Steuersignalen, existiert ein Systemtakteingang, ein Rücksetzsignal (RESET), ein Interruptsignal (INT), zwei Signale (IEI, IEO) zum Einbinden des SIO-Bausteins in die Prioritätskaskade und das in zwei gleiche Teile (Port A, Port B) zerfallende Peripherieinterface. Die zwei Adreßleitungen kodieren vier interne SIO-Adressen, die beiden Datenkanaladressen Port A (data) und Port B (data) und zwei Steuerwortkanaladressen Port A (control) und Port B (control). Durch die Einbindung in die Prioritätskaskade eines konkreten Systems wird dem SIO-Baustein eine bestimmte Priorität in der Vorrangstruktur übertragen. Intern hat der Kanal A die höhere Priorität gegenüber dem Kanal B. Die Priorität innerhalb eines Kanals wurde in einer festen Reihenfolge (Empfänger - Sender - Extern/Status) vergeben.
Das Peripherieinterface jedes SIO-Ports besteht aus je einem Anschluß für den seriellen Datensender (TxD) und den seriellen Datenempfänger (RxD). Dieser Sende- und Empfangsleitung sind je drei weitere Signale zugeordnet, ein Sender- und Empfängertakteingang (TxC, RxC), ein Sperrsignal für Sender und Empfänger (CTS, DCD) und ein Sendegesuchsignal (RTS) bzw. ein Empfängerbereitsignal (DTR). Der Sender- und der Empfängertakteingang wird in der asynchronen Betriebsart intern um den Faktor 1, 16, 32 oder 64 vorgeteilt. Das Sperrsignal blockiert in der Betriebsart 'auto enable' den Datensender des entsprechenden Kanals, ebenso wie das Empfängersperrsignal den Datenempfänger des zugehörigen Kanals ausschaltet. Sendergesuchsignal und Empfängerbereitsignal sind Mitteilungen des Programmierers an die externen Geräte, daß Daten abzusetzen sind bzw. daß Daten erwartet werden.

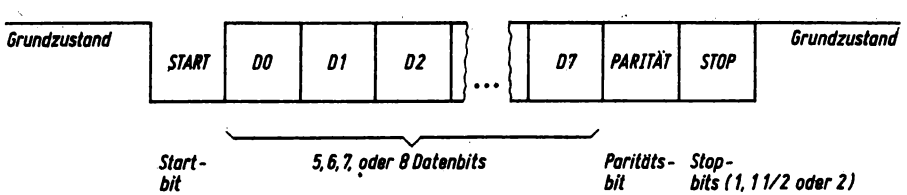


Bild 2.7. Aufbau des seriellen Datentelegramms im Asynchronbetrieb

Neben diesen dem Sender bzw. dem Empfänger eines Kanals zugeordeten Signalen existieren zwei weitere Anschlüsse. Das Signal WAIT/READY kann zum Anschluß einer Steuerung für den direkten Speicherzugriff (U858-DMA) unter Umgehung der CPU oder zur WAIT-Signalgenerierung für die U880-CPU verwendet werden. Das Signal SYNC schließlich wird in der Betriebsart 'externe Synchronisation' zum Zeichenaufbau verwendet.

Asynchrone serielle Datenübertragung: Der U856-SIO kann für zwei grundverschiedene Betriebsarten programmiert werden, die asynchrone und die synchrone Betriebsart. Für die asynchrone Betriebsart ist kennzeichnend, daß, unregelmäßig, in Abhängigkeit von dem zu übertragenden Informationsaufkommen, eine serielle Datenübertragung nur dann stattfindet, wenn beim Sender bzw. Empfänger eine Datenausgabe- bzw. Dateneingabeanforderung vorliegt. Findet keine Datenübertragung statt, ist der Signalweg im Grundzustand (Ruhezustand) (Bild 2.7.). Die Mitteilung, daß die Datenübertragung eines Zeichens auf der Seite des Senders oder Empfängers abgeschlossen ist, kann über eine Abfrageschleife (polling) oder über eine Interruptmeldung erfolgen.

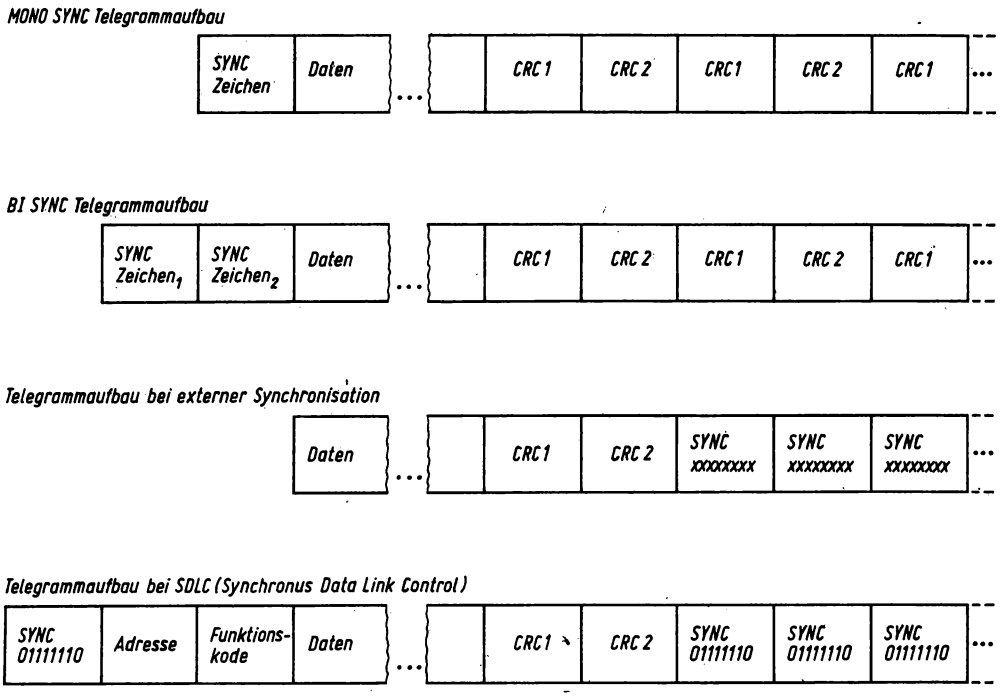


Bild 2.8. Aufbau verschiedener serieller Datentelegramme im Synchronbetrieb

Synchrone serielle Datenübertragung: Im Gegensatz zur asynchronen seriellen Datenübertragung findet in der synchronen Betriebsart des U856-SIO ein ständiger, ununterbrochener Datenaustausch zwischen Sender und Empfänger statt. Liegen keine zur Übertragung an den SIO ausgegebenen Informationszeichen in dem Sendepufferspeicher vor, werden auf der Datenleitung zwischen Sender und Empfänger automatisch erzeugte Synchronisationszeichen übertragen. Die Datenrate bei der synchronen, seriellen Datenübertragung ist mit der an den Schaltkreis angelegten Taktfrequenz (TxC, RxC) identisch. Die verschiedenen mit dem U856-SIO im synchronen Mode erzeugbaren Telegrammstrukturen, sind dem Bild 2.8. zu entnehmen. Sie unterscheiden sich im wesentlichen durch unterschiedliche Blockformate (Framestrukturen). Der die eigentlichen Datenzeichen umgebende Rahmen besteht dabei aus den SYNC-Zeichen und der CRC-Kontrollsumme (CRC zyklischer redundanter Kode). Die SYNC-Zeichen dienen der Koordinierung der Arbeit von Sender und Empfänger. Die CRC-Kontrollsumme ermöglicht die Überprüfung der übertragenen Daten auf einen eventuellen Informationsverlust.

2.3.2. Programmierung

Insgesamt acht Schreibregister und drei Leseregister stehen dem Programmierer zur Betriebsartenauswahl und zur Steuerung des Datenaustausches zur Verfügung. Zur Adressierung der Schreib- und Leseregister dient ein Pointer. Er ist vor jeder Ausgabe an ein Schreibregister und vor jeder Eingabe von einem Leseregister an die Steuerwortadresse (control) des spezifizierten Datenkanals Port A oder Port B auszugeben. Der Pointer belegt die unteren drei Bits im Schreibregister Null. Die erste Ausgabe eines Steuerwortes nach dem Einschalten ebenso wie eine Ausgabe nach dem Lesen oder Schreiben eines der Register - ausgenommen Register Null - geht immer zum Schreibregister Null (Tafel 2.3.).

Tafel 2.3. Registerstruktur U856-SIO

Pointer 0 Schreibregister 0 : Kommandoregister

00000000	CRC1	CRC0	CMD2	CMD1	CMD0	0	0	0

Pointer 1 Schreibregister 1 : Interruptmoderegister

00000001	WAIT RDY2	WAIT RDY1	WAIT RDY0	REC MODE1	REC MODE0	STATUS VEKTOR	TRANSM INT EN	EXTERN INT EN

Pointer 2 Schreibregister 2 : Interruptvektorregister

00000010	V7	V6	V5	V4	V3	V2	V1	V0

Pointer 3 Schreibregister 3 : Empfängersteuerungsregister

00000011	CHAR0	CHAR1	AUTO EN	HUNT	CRC EN	ADR SEA	SYNC	REC EN

Pointer 4 Schreibregister 4 : Empfänger-/Sendersteuerungsregister

00000100	CLOCK 1	CLOCK 0	SYNC1	SYNC0	STOP1	STOP0	EVEN ODD	PARITY ENABLE

Pointer 5 Schreibregister 5 : Sendersteuerungsregister

00000101	DTR	TRANS 1	TRANS 0	SEND BREAK	TRANSM ENABLE	CRC CODE	RTS	TRANS CRC EN

Pointer 6 Schreibregister 6 : Register SYNC-Zeichen (low)

00000110	SYN7	SYN6	SYN5	SYN4	SYN3	SYN2	SYN1	SYN0

Pointer 7 Schreibregister 7 : Register SYNC-Zeichen (high)

00000111	SYN15	SYN14	SYN13	SYN12	SYN11	SYN10	SYN9	SYN8

Pointer 0 Leseregister 0 : Interruptstatusregister

00000000	BREAK	TRANSM UNDER	CTS	SYNC HUNT	DCD	TRANS EMPTY	INT	REC AVAI

Pointer 1 Leseregister 1 : Fehlerstatusregister

00000001	END FRAME	CRC FRAME	OVER RUN	PARITY ERROR	RES2	RES1	RES0	ALL SEND

Pointer 2 Leseregister 2 : Interruptvektorregister

00000010	V7	V6	V5	V4	V3*	V2*	V1*	V0

```
Schreibregister 0:      Kommandoregister

CMD2 CMD1 CMD0          COMMAND BITS
   = 000  Kein Kommando (NOP / Verwendung Pointer setzen)
   = 001  SDLC-Abort senden (8 bis 13 Einsen)
   = 010  Rücksetzen 'Extern/Status Interrupts'      (Leseregister 0)
   = 011  Rücksetzen Port (Entspricht RESET-Port)
   = 100  Rücksetzen Empfängerinterrupt 'Erstes empfangenes Zeichen'
   = 101  Rücksetzen Senderinterrupt 'Jedes Zeichen'
   = 110  Rücksetzen Parity-/Overrunfehler           (Leseregister 1)
   = 111  RETI-Kommando (Nur Port A - wenn nicht in Daisy-Chain-Kaskade.)

CRC1 CRC0               CRC RESET CODE
   = 00   Kein Kommando (NOP / Verwendung Pointer setzen)
   = 01   Rücksetzen CRC-Empfangsfilter synchrone Betriebsart mit CRC
   = 10   Rücksetzen CRC-Sendegenerator synchrone Betriebsart mit CRC
   = 11   Rücksetzen CRC/SYNCS-Statusspeicher
```

```
Schreibregister 1:      Interruptmoderegister

EXTERN INT EN           EXTERNAL/STATUS INTERRUPT ENABLE
   = 0    externer Interrupt AUS
   = 1    externer Interrupt EIN (DCD; CTS; SYNC; Break/Abort; CRC-/Sync)

TRANSM INT              TRANSMITTER INTERRUPT ENABLE
   = 0    Senderinterrupt AUS
   = 1    Senderinterrupt EIN (Sender Puffer leer)

STATUS VEKTOR           STATUS EFFECTS VECTOR  (Nur Port B)
   = 0    Interruptvektor Kanal A und B wird nicht modifiziert.
   = 1    Interruptvektor Kanal A und B wird entsprechend Status modifiziert.

REC MODE1  REC MODE0    RECEIVE INTERRUPT MODES
   = 00   Empfängerinterrupt AUS
   = 01   Empfängerinterrupt 'Erstes Zeichen' (Sonst nur Interrupt bei
          Sonderbedingung: End of Frame SDLC, Overrun, Framing Error.)
   = 10   Empfängerinterrupt 'Jedes Zeichen' (Paritätsfehlerinterrupt
          ist eine spezielle Empfangsbedingung.)
   = 11   Empfängerinterrupt 'Jedes Zeichen' (Paritätsfehlerinterrupt
          ist keine spezielle Empfangsbedingung.)

WAIT/RDY0               WAIT/READY FUNCTION SELECTION
   = 0    WAIT aktiv - Sendepuffer voll / READY aktiv - Sendepuffer leer
   = 1    WAIT aktiv - Empfängerpuffer leer / READY aktiv - Empfänger-
          puffer voll

WAIT/RDY1               WAIT/READY FUNCTION SELECTION
   = 0    WAIT/READY wird als WAIT-Anschluß für die U880-CPU benutzt.
   = 1    WAIT/READY wird als READY-Anschluß für den U858-DMA benutzt.

WAIT/RDY2               WAIT/READY FUNCTION SELECTION
   = 0    WAIT/READY logisch 1 (READY-Mode) - logisch unbestimmt (WAIT-Mode)
   = 1    WAIT/READY wird entsprechend Betriebsartenauswahl aktiv.
```

```
Schreibregister 2:      Interruptvektorregister  (Nur Port B)

V7 V6 V5 V4   konstanter Teil des Interruptvektors

V3 = 1    Port A
   = 0    Port B

V2 V1
   = 00   Sendepuffer leer
   = 01   Statuswechsel                    (siehe Leseregister 0 und 1)
   = 10   Empfangenes Zeichen verfügbar
   = 11   Empfangssonderfall               (siehe Leseregister 0 und 1)
```

V0 = 0 (Konstant)

Anmerkung: V3, V2 und V1 werden, wenn STATUS EFFECTS VECTOR=1 (Schreibregister 1), bei einem Interrupt automatisch, wie angegeben, vom U856-SIO an die U880-CPU geliefert - anderenfalls ist V3, V2, und V1 genau wie V7, V6, V5, V4 und V0 konstant.

Schreibregister 3: Empfängersteuerungsregister

REC EN — RECEIVER ENABLE
- = 0 Empfänger ist ausgeschaltet.
- = 1 Empfänger ist eingeschaltet.

SYNC — SYNC CHARACTER LOAD INHIBIT
- = 0 Synchronisationszeichen werden beim Empfang nicht ausgeblendet.
- = 1 Synchronisationszeichen werden beim Empfang ausgeblendet.

ADR SEA — ADDRESS SEARCH MODE
- = 0 keine Telegrammadressenprüfung
- = 1 Es wird nur dann ein Zeichen empfangen und ein Interrupt abgesetzt, wenn die Telegrammadresse mit der programmierten Adresse (Schreibregister 6) bzw. mit der Adresse '11111111' übereinstimmt.

CRC EN — RECEIVER CRC ENABLE
- = 0 keine CRC-Polynomberechnung
- = 1 CRC-Polynomberechnung ab nächstem empfangenen Zeichen. (Ist vor jeder Zeicheneingabe neu zu definieren.)

HUNT — ENTER HUNT MODE
- = 0 keine Operation (NOP)
- = 1 Port geht in den HUNT-Mode (SYNC-Zeichensuche).

AUTO EN — AUTO ENABLES
- = 0 CTS- und DCD-Signal sind ohne Bedeutung für Datenübertragung.
- = 1 CTS- und DCD-Signal schalten den Sender und Empfänger EIN/AUS.

CHAR0 CHAR1 — RECEIVER BITS CHARACTERS
- = 00 5 Bit je übertragenen Zeichen
- = 01 6 Bit je übertragenen Zeichen
- = 10 7 Bit je übertragenen Zeichen
- = 11 8 Bit je übertragenen Zeichen

Schreibregister 4: Empfänger-/Sendersteuerungsregister

PARITY — PARITY
- = 0 Datenübertragung ohne Paritätsbit.
- = 1 Datenübertragung mit Paritätsbit.

EVEN/ODD — PARITY EVEN / ODD
- = 0 ungerade Parität
- = 1 gerade Parität

STOP1 STOP0 — STOP BITS
- = 00 SYNC-Mode
- = 01 1 Stop Bit
- = 10 1 1/2 Stop Bits
- = 11 2 Stop Bits

SYNC1 SYNC0 — SYNC MODE SELECT
- = 00 Monosync-Mode (8 Bit Initialszeichen)
- = 01 Bisync-Mode (16 Bit Initialzeichen)
- = 10 SDLC-Mode (SDLC-Flag '01111110' Initialzeichen)
- = 11 Extern-Sync-Mode (SYNC-Signal Initialisierung)

```
CLOCK1 CLOCK0          CLOCK RATE
   = 00   Datenrate x 1 = CLOCK-Rate TxC, RxC
   = 01   Datenrate x16 = CLOCK-Rate TxC, RxC
   = 10   Datenrate x32 = CLOCK-Rate TxC, RxC
   = 11   Datenrate x64 = CLOCK-Rate TxC, RxC
```

```
Schreibregister 5:     Sendersteuerungsregister

TRANS CRC EN           TRANSMIT CRC ENABLE
   = 0    keine CRC-Berechnung (Sender)
   = 1    CRC-Berechnung je Zeichen (Sender)

RTS                    REQUEST TO SEND
   = 0    RTS-Signal 0
   = 1    RTS-Signal 1

CRC CODE               CRC-16/SDLC
   = 0    SDLC-Mode x16+x12+x5+1 CRC-Kode für Sender und Empfänger
   = 1    CRC-16    x16+x15+x2+1 CRC-Kode für Sender und Empfänger

TRANSM ENABLE          TRANSMIT ENABLE
   = 0    Sender AUS
   = 1    Sender EIN

SEND BREAK             SEND BREAK
   = 0    Normalbetrieb
   = 1    SEND BREAK

TRANS1 TRANS0          TRANSMIT BITS
   = 00   5 Bit oder weniger je Zeichen
   = 01   7 Bit je Zeichen
   = 10   6 Bit je Zeichen
   = 11   8 Bit je Zeichen
          Die Sendezeichen sind im Datenbyte rechtsbündig einzutragen.
          Wenn fünf oder weniger Bits ausgegeben werden, gilt das Format:
          11110000  1 Bit
          11100000  2 Bit
          11000000  3 Bit
          10000000  4 Bit
          00000000  5 Bit

DTR                    DATA TERMINAL READY
   = 0    DTR-Signal 1
   = 1    DTR-Signal 0
```

```
Schreibregister 6:     Register SYNC-Zeichen (low)

SYN7, SYN6 ... SYN0
   Monosync-Mode       Monosync-Folge SENDEN
   Bisync-Mode         ersten 8 Bit der Bisync-Folge SENDEN und EMPFANGEN
   SDLC-Mode           Adresse (wenn verwendet) EMPFANGEN
   Extern-Sync-Mode    nicht benutzt (SENDEN)
```

```
Schreibregister 7:     Register SYNC-Zeichen (high)

SYN15, SYN14 ... SYN8
   Monosync-Mode       Monosync-Folge EMPFANGEN
   Bisync-Mode         zweiten 8 Bit der Bisync-Folge SENDEN und EMPFANGEN
   SDLC-Mode           01111110 EMPFANGEN
   Extern-Sync-Mode    nicht benutzt (EMPFANGEN)
```

Leseregister 0: Interruptstatusregister

REC AVAI	RECEIVE CHARACTER AVAILABLE / Dieses Bit wird gesetzt, wenn vom Empfänger ein Zeichen abgeholt werden kann. Wird verwendet bei der Arbeit ohne Interrupt.
INT	INTERRUPT PENDING / Das Bit spezifiziert, ob eine Interruptbedingung (Fehler/Datenübertragung) existiert oder nicht. Wird verwendet bei der Arbeit ohne Interrupt.
TRANS EMPTY	TRANSMITTER BUFFER EMPTY / Dieses Bit wird gesetzt, sobald der Pufferspeicher des Senders leer ist. Wird verwendet bei der Arbeit ohne Interrupt.
DCD	DATA CARRIER DETECT / Das DCD-Bit spiegelt den Zustand des negierten DCD-Signals wider.
SYNC/HUNT	SYNC/HUNT / In der asynchronen und extern-synchronen Betriebsart spiegelt das SYNC/HUNT-Bit den Zustand des SYNC-Signals wider. In der synchronen Betriebsart wird dieses Bit durch den Befehl 'Enter HUNT-Mode' gesetzt.
CTS	CLEAR TO SEND / Das CTS-Bit spiegelt den Zustand des negierten CTS-Signals wider.
TRANSM UNDER	TRANSMITTER UNDERRUN - END OF MESSAGE / Senderunterlauf bzw. Nachrichtenende
BREAK	BREAK / In der asynchronen Betriebsart wird dieses Bit gesetzt, wenn ein 'BREAK' erkannt wird. In der SDLC-Betriebsart wird dieses Bit durch eine Abbruchsequenz gesetzt und rückgesetzt.

Leseregister 1: Fehlerstatusregister

ALL SEND	ALL SEND / Alle Zeichen haben den Sender verlassen, wenn dieses Bit gesetzt ist.
RES2 RES1 RES0	RESIDUE CODES / Diese Bits kennzeichnen die Länge des I-Feldes im SDLC-Mode:
= 100	3 Bit I-Feld
= 010	4 Bit I-Feld
= 110	5 Bit I-Feld
= 001	6 Bit I-Feld
= 101	7 Bit I-Feld
= 011	8 Bit I-Feld
= 111	9 Bit I-Feld
= 000	10 Bit I-Feld
PARITY	PARITY ERROR / Paritätsfehler erkannt.
OVERRUN	RECEIVE OVERRUN ERROR / Empfängerüberlaufsfehler erkannt.
CRC FRAME	CRC-FRAMING ERROR / Dieses Bit wird gesetzt bei 'FRAMING ERROR' (asynchroner Mode) und bei CRC-Prüffehler (synchroner Mode).
END FRAME	END OF FRAME / Dieses Bit zeigt im SDLC-Mode die Endesequenz des übertragenen Telegramms an.

Leseregister 2: Interruptvektorregister

Die Belegung des Leseregisters 2 entspricht der Belegung - ggf. mit Vektormodifikation V3*, V2*, V1* - des Schreibregisters 2.

2.4. Zähler-/Zeitgeberbaustein U 857-CTC

Der Zähler-/Zeitgeberbaustein U857-CTC übernimmt im Mikrorechnersystem U880 die verschiedensten Zähl- und Zeitgeberfunktionen. Der CTC-Baustein (CTC counter timer circuit) leitet dabei die Zähl- und Zeitgeberaktivitäten entweder vom 250 ns bzw. vom 400 ns Systemtakt oder aber von einem extern bereitgestellten alternierenden Signal ab /6/,/7/,/14/,/15/.

2.4.1. Aufbau

Auf der Peripherieseite des U857 stehen vier unabhängig oder verkettet betreibbare Zählkanäle zur Verfügung, die jeweils einen externen Zähltakteingang und - ausgenommen Kanal 3 - einen Nulldurchgangsausgang besitzen. Der externe Zähltakteingang wird nur benutzt, wenn nicht mit dem Systemtakt gearbeitet werden soll. Der Nulldurchgangsausgang wird immer dann aktiv, wenn der als Zeitkonstante bezeichnete, voreingestellte Zählwert durch den externen Zähl- oder durch den internen Systemtakt abgearbeitet wurde. Wird mit dem Systemtakt gearbeitet, ist vor die abzuarbeitende 8-Bit-Zeitkonstante ein Vorteiler (16 oder 256) geschaltet.
Die beiden möglichen Betriebsarten des U857-CTC 'Systemtakt zählen' und 'externen Takt zählen' werden als 'Zeitgebermode' und 'Zählermode' bezeichnet. Gleichgültig, ob die Kanäle verkettet oder einzeln betrieben werden, wird nach jedem Nulldurchgang die Zeitkonstante selbständig erneuert, indem der im Zeitkonstantenregister befindliche Wert in das Zählregister (Rückwärtszähler) umgeladen wird. Durch diesen automatischen Vorgang, der durch ein Hardwarereset, durch ein Softwarekanalreset oder durch ein neues Initialisieren des entsprechenden Kanals abgebrochen werden kann, entsteht eine zyklische Arbeitsweise der CTC-Kanäle.
Auf der Prozessorseite ist der CTC-Baustein u.a. durch den Datenbus mit dem Mikroprozessor verbunden. Er gestattet die Programmierung (Ausgabe) des Steuerwortes, der Zeitkonstante und des Interruptvektors. Über den Datenbus können außerdem die momentanen Kanalzählerstände vom U857 in den U880 eingelesen werden (Bild 2.9.).
Die Interruptsignalleitung meldet der CPU - entsprechend dem Nulldurchgangsausgang - den Zeitpunkt der Abarbeitung des Zählerregisters. Der dabei an die CPU übergebene Interruptvektor (Low-Teil) kennzeichnet den entsprechenden Kanal. Die Signale IEI und IEO gestatten die Einordnung des U857 in die Interruptprioritätskaskade eines konkreten Mikrorechnersystems. Intern hat der Kanal 0 die höchste und der Kanal 3 die niedrigste Priorität. Da der aktuelle, interne Zählerstand jedes Kanals auch vom U880 per Programm abgefragt werden kann, ist es möglich, den Zähler-/Zeitgeberbaustein U857 nicht nur im interruptgesteuerten Betrieb sondern auch im Abfragebetrieb (polling) zu betreiben.

Zeitgebermode: Im Zeitgebermode wird der Rückwärtszähler des CTC-Kanals durch den von einem Vorteiler untersetzten Systemtakt abgearbeitet. Der Vorteiler kann auf den Wert 16 oder 256 eingestellt werden. Die Zeit zwischen zwei Nulldurchgängen des Kanals berechnet sich dann aus: Zeitkonstante x Vorteiler x Systemtakt (in ns). Die Nulldurchgänge eines im Zeitgebermode arbeitenden CTC-Kanals werden über ein Interruptsignal an die U880-CPU gemeldet oder sind per Software abzufragen.

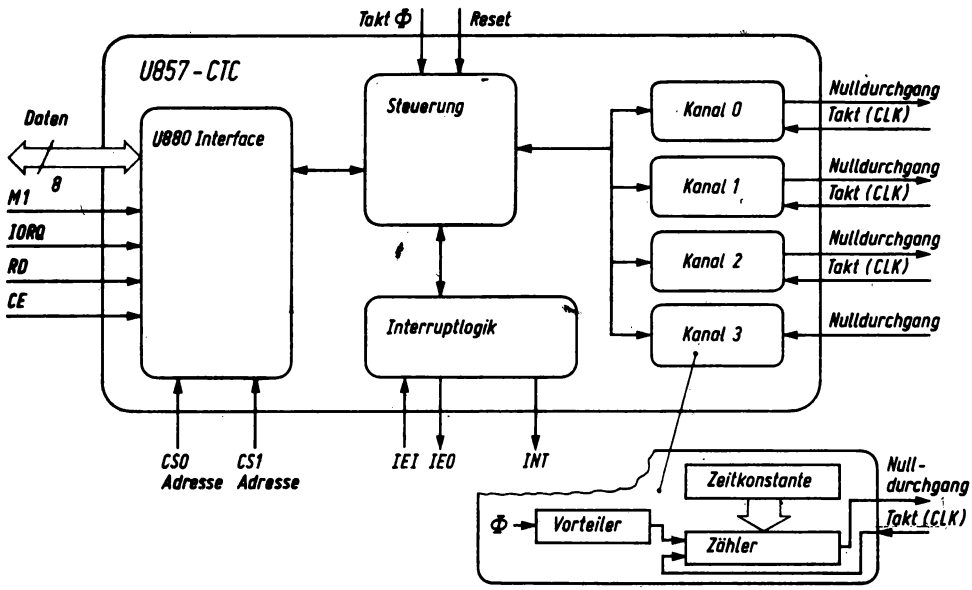


Bild 2.9. Aufbau und Signalstruktur des Zähler-/Zeitgeberbausteins U857-CTC

Zählermode: Im Zählermode wird der Rückwärtszähler des CTC-Kanals durch ein Taktsignal am externen Eingang (CLK) abgearbeitet. Der Vorteiler findet im Zählermode keine Berücksichtigung. Jeder Takt am CLK-Eingang dekrementiert einmal die Zeitkonstante. Ebenso wie im Zeitgebermode werden die Nulldurchgänge des Rückwärtszählers über Interrupt an den Mikroprozessor gemeldet oder über einen Rücklesebefehl per Programm abgefragt.

2.4.2. Programmierung

Die zwei Adreßbits des CTC-Bausteins CS0 und CS1 ermöglichen bei der Datenausgabe und bei der Dateneingabe zwischen U880 und U857 die Auswahl eines der vier CTC-Kanäle. Bei der Dateneingabe vom U857 in den CPU-Schaltkreis gelangen die aktuellen Zählerstände der einzelnen Kanäle in den Mikroprozessor. Der Zählvorgang wird dadurch nicht beeinflußt. Bei der Datenausgabe muß zwischen dem Interruptvektor (Low-Teil), dem eigentlichen U857-Steuerwort und der abzuzählenden Zeitkonstante unterschieden werden (Tafel 2.4.).
Der Interruptvektor wird vom Steuerwort durch ein rückgesetztes Bit in der Position D0 unterschieden, während zwischen Steuerwort und Zeitkonstante eine feste Ausgabesequenz festgelegt ist. Der Interruptvektor wird nur an die Kanaladresse CS0,CS1=00 ausgegeben und enthält in den Bitpositionen

D7...D3 einen für alle vier CTC-Kanäle gemeinsamen Teil. In die Bitpositionen D2 und D1 wird im Interruptfall automatisch die entsprechende Kanaladresse (00,01,10,11) eingeblendet. Zur Unterscheidung zwischen Interruptvektor und Steuerwort wird im Bit D0 des Steuerwortes generell eine 1 eingetragen. Die Steuerwortausgabe ist nicht an einen bestimmten Zeitpunkt gebunden. Sie kann sowohl auf einen rückgesetzten (RESET) als auch auf einen bereits arbeitenden Kanal erfolgen, ausgenommen der Kanal erwartet nach einer bereits erfolgten Steuerwortausgabe eine Zeitkonstante. Bei der Festlegung des CTC-Steuerwortes sind in den Bitpositionen D1, D2, D3 und D4 folgende Besonderheiten zu beachten:

D1=0 - Keine Veränderung (NOP) in der Arbeitsweise des CTC-Kanals.

D1=1 - Rücksetzen (Stop) des CTC-Kanals. Der Kanal bricht den laufenden Zählvorgang ab. Ist gleichzeitig das Bit D2=1, wartet der Kanal, bis eine Zeitkonstante eingegeben wird, und zählt dann weiter. Anderenfalls muß ein neues Steuerwort geladen werden, um den Kanal wieder zu aktivieren.

D2=0 - Auf das Kanalsteuerwort folgt keine Zeitkonstante. Durch ein Steuerwort mit D2=0 kann 'interrupt disable' (D7=0) gegeben werden, ohne eine Zeitkonstante mit ausgeben zu müssen.

D2=1 - Das nächste an die CTC-Kanaladresse ausgegebene Datenwort ist die Zeitkonstante. Sollte in dem adressierten Kanal zum Zeitpunkt der Zeitkonstantenausgabe ein Zählvorgang laufen, so wird der Abschluß (Nulldurchgang) dieses Zählvorganges abgewartet, bis die neue Zeitkonstante wirksam wird.

D3=0 - Die Abarbeitung des Rückwärtszählers beginnt mit der fallenden Flanke des zweiten Taktes in dem auf die Ausgabe der Zeitkonstante folgenden Maschinenzyklus (nur im Zeitgebermode von Bedeutung).

D3=1 - Die eigentliche Startfreigabe des Rückwärtszählers erfolgt, wenn eine Flanke (siehe D4) am externen Takteingang einläuft. Ist die Startfreigabe erfolgt, gilt das bei D3=0 Gesagte (nur im Zeitgebermode von Bedeutung).

D4=1 / Zeitgebermode - Eine positive Flanke am externen Takteingang erteilt die Startfreigabe bei D3=1.

D4=1 / Zählermode - Die positiven Flanken des am externen Takteingang anliegenden Signals arbeiten den Rückwärtszähler ab.

D4=0 - Wie D4=1, nur daß nicht die positiven, sondern die negativen Flanken des externen Takteingangs die Startfreigabe bzw. die Zählsignale darstellen.

Tafel 2.4. Registerstruktur des U857-CTC

Interruptvektor - nur Kanal 0 (V2,V1 Kanalmodifikation; V0=0)

V7	V6	V5	V4	V3	V2*	V1*	0

Der Interruptvektor gilt für die Kanäle 0 bis 3. Die Bitpositionen V2* und V1* sind bei der Interruptvektorvorgabe an den CTC-Schaltkreis ohne Bedeutung, sie werden bei der Interruptantwort eingeblendet:

V2*,V1*=00 CTC-Kanal 0
V2*,V1*=01 CTC-Kanal 1
V2*,V1*=10 CTC-Kanal 2
V2*,V1*=11 CTC-Kanal 3

Steuerwort (D5,D3 nur im Zeitgebermode von Bedeutung; D0=1)

EI/DI	MODE	16/256	SLOPE	TRIGGER	TIMECON	RESET	1

EI=1	Kanalinterrupt EIN	DI=0	Kanalinterrupt AUS
MODE=1	Zählermode	MODE=0	Zeitgebermode
16/256=1	Vorteiler 256	16/256=0	Vorteiler 16
SLOPE=1	positive Flanke	SLOPE=0	negative Flanke
TRIGGER=1	externer Start	TRIGGER=0	Zeitkonstantenstart
TIMECON=1	Zeitkonstante folgt	TIMECON=0	Zeitkonstante folgt nicht
RESET=1	Kanal RESET	RESET=0	keine Operation (NOP)

Zeitkonstante (Folgt bei D2=1 auf das Steuerwort.)

TC7	TC6	TC5	TC4	TC3	TC2	TC1	TC0

Binärer Wert der Zeitkonstante: 1 bis 255 (0 entspricht 256).

2.5. Baustein für direkten Speicherzugriff U 858-DMA

Die Schaltkreisfamilie des Mikroprozessorsystems U880 beinhaltet den DMA-Baustein U858 (DMA direct memory access) zur Abwicklung des direkten Datenaustausches zwischen Speicher und Ein-/Ausgabekanälen ohne Mitwirkung der U880-CPU. Dieser Schaltkreis findet immer dann Verwendung, wenn der Mikroprozessor von den routinemäßigen Datentransferaufgaben 'LESEN eines Datenbytes aus Speicher oder Eingabekanal' - 'SCHREIBEN eines Datenbytes in Ausgabekanal oder Speicher' entlastet werden soll oder wenn Datenübertragungsraten von mehr als 120 KByte (2,5 MHz U880-System) bzw. 190 KByte (4 MHz UA880-System) je Sekunde - die maximale Ein-/Ausgabegeschwindigkeit beim programmierten Datenaustausch mit IN/OUT-Befehlen - erreicht werden müssen.
Der U858-DMA ist in der Lage, Datenübertragungsraten von maximal 1,25 MByte (U858) bzw. 4 MByte (UA858) je Sekunde zu erreichen. Er ist speziell auf die beiden Funktionen LESEN und SCHREIBEN zugeschnitten und hat als einzige Aufgabe , den Transport von Daten zwischen den Ein-/Ausgabekanälen und dem Mikrorechnerspeicher zu übernehmen. Der DMA-Baustein unterdrückt dazu über ein speziell für diesen Zweck geschaffenes Signalspiel (BUSRQ bus request - BUSAK bus acknowledge) für die Zeit des Datentransfers kurzzeitig den Zugriff des Mikroprozessors auf den Systembus und führt dann seine speziellen Lese- und Schreiboperationen aus /6/./16/./17/./18/.

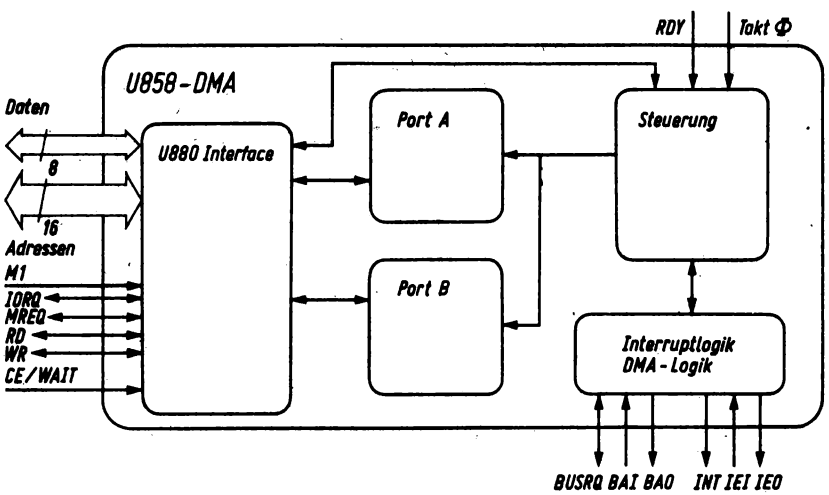


Bild 2.10. Aufbau und Signalstruktur des U858-DMA

2.5.1. Aufbau

Der U858-DMA muß vor seiner Aktivierung vom Mikroprozessor initialisiert werden. Dabei werden der Operationsmodus, das Zeitverhalten, das Inter-

ruptverhalten, die auszutauschenden Daten u.a. spezifiziert. Nach dieser Initialprogrammierung erfolgt der DMA-Start. Der DMA-Baustein generiert dann die gesamten zur Datenübertragung notwendigen Steuerungsabläufe vollkommen selbständig. Wärend der Dauer der DMA-Aktivitäten wird die Befehlsabarbeitung im Mikroprozessor ausgesetzt. Die Priorität einer DMA-Anforderung liegt über der des höchstpriorisierten Interruptsignals.
Ebenso wie bei der Prioritätsfestlegung in der Interruptkaskade (daisy chain / IEI-IEO) ist es möglich, bei Bedarf mehrere U858-DMA Schaltkreise in einer DMA- Kaskade (BAI-BAO) anzuordnen. Der DMA-Schaltkreis mit der höchsten Priorität befördet dann nicht nur die U880-CPU sondern auch alle anderen DMA-Bausteine mit dem Signal BUSRQ in 'Ruhestellung', bis seine Aktivitäten beendet sind.
Neben der Datenübertragung ist der U858-DMA auch in der Lage, die Suche nach einem spezifizierten Byte im Arbeitsspeicher oder Ein-/Ausgabebereich zu übernehmen oder beides, die Datenübertragung und die Datensuche zu kombinieren (Bild 2.10.).
Intern existieren im DMA-Schaltkreis zwei Ports (A und B), die vom Anwender wahlweise der DMA-Datenquelle und dem DMA-Datenziel zugeordnet werden. Datenquelle und Datenziel können sowohl Speicheradressen als auch Peripherieschnittstellen darstellen. Die Startadresse und die Blocklänge des DMA-Datenstromes werden bei der Steuerwortausgabe vorprogrammiert. Ebenso kann ein ggf. zu suchendes Byte - bei Bedarf mit Maske - dem U858 vorgegeben werden.
Bezogen auf den Zeitablauf beim DMA-Verkehr existieren drei mögliche Betriebsarten:

1-BYTE	Jeweils ein Byte wird bei einer DMA-Anforderung (BUSRQ) übertragen bzw. gesucht.
BURST	Die DMA-Datenübertragung(-suche) läuft so lange, wie das READY-Signal (RDY) aktiv ist - maximal jedoch bis zum Blockende.
CONTINOUS	Die DMA-Datenübertragung(-suche) läuft bis zum festgelegten Blockende.

Die Betriebsart '1-BYTE' hat den Vorteil, daß zwischen jeder Übertragung eines Datenbytes im DMA-Verkehr kurzzeitig der Mikroprozessor die Steuerung über den Systembus zurückerhält. Er hat dann die Möglichkeit, so lange seine Befehlsabarbeitung fortzusetzen, bis der DMA wieder den Systembus mit einer Busanforderung (BUSRQ) belegt. Die Betriebsart '1-BYTE' führt nicht zu einer Verringerung der DMA-Datenrate. Der Unterschied zwischen den Betriebsarten 'BURST' und 'CONTINOUS' liegt einzig in der Auswertung des Signals READY. Mit diesem Signal kann der U858-DMA mitten im Betrieb hardwaremäßig zur DMA-Datenübertragung (-suche) gesperrt bzw. wieder freigegeben werden.
Neben den zeitablaufbezogenen Betriebsarten existieren drei funktionsbezogene Betriebsarten des U858-DMA Bausteins - die Datenübertragung, die Datensuche und die Datenübertragung mit gleichzeitiger Datensuche:

DMA-Datenübertragung: In der Betriebsart 'Datenübertragung' werden vom U858 Datenbytes von einer Datenquelle gelesen und sofort in ein Datenziel geschrieben. Die Lese- und Schreibfunktionen werden jeweils einem DMA-Port (A oder B) zugeordnet. Bei der Programmierung wird festgelegt, ob einem DMA-Port der Systemspeicher (MREQ) oder eine Ein-/Ausgabeschnittstelle

(IORQ) zugeordnet wird. Dadurch ergeben sich folgende Möglichkeiten der Datenübertragung:

Speicher ——————————→ Speicher
Speicher ——————————→ E/A-Schnittstelle
E/A-Schnittstelle ——————————→ Speicher
E/A-Schnittstelle ——————————→ E/A-Schnittstelle

Das Ende der Datenübertragung wird der U880-CPU über eine Status- oder Interruptmeldung angezeigt.

DMA-Datensuche: In der Betriebsart 'Datensuche' findet kein Datentransfer statt. Ein Speicher- oder ein Ein-/Ausgabeadressenbereich wird nach einem dem DMA-Baustein vorgegebenen Datenbyte durchgemustert. Das vorgegebene Datenbyte (match byte) kann durch ein bei der Programmierung des U858-DMA festgelegtes Maskenbyte überdeckt werden, um einzelne Bitpositionen aus dem Vergleichsvorgang auszunehmen. Wird das gesuchte Byte gefunden, erfolgt eine Status- oder Interruptmeldung an die CPU - die weitere Suche wird abgebrochen.

DMA-Datenübertragung mit Datensuche: Die Betriebsart 'Datenübertragung mit Datensuche' stellt eine Kombination der beiden bereits beschriebenen DMA-Arbeitsregime dar. Transfer und Vergleich werden hier gleichzeitig ausgeführt. Wird die gesuchte Bitkombination gefunden, erfolgt eine Status- oder Interruptmeldung an die U880-CPU. Die weitere Datenübertragung und Datensuche wird abgebrochen oder fortgeführt je nach Programmierung.

2.5.2. Programmierung

Die Programmierung des U858-DMA erfolgt über Ein-/Ausgabebefehle an sieben Schreibregistergruppen (WR0, WR1...WR6) und sieben Leseregister (RR0, RR1 ... RR6).

Schreibregistergruppe WR0 (Tafel 2.5.) - Das Schreibregister WR0 spezifiziert in den Datenbits C0 und C1 die DMA-Betriebsarten (Datentransfer, Datensuche, Datentransfer mit Datensuche).
Ein Quellenportkennzeichen (SOURCE) legt die Übertragungsrichtung fest, d.h., ob über Port A gelesen und Port B geschrieben wird oder umgekehrt. Mit diesem Steuerbit kann ein Hin- und Hertransfer von Datenfeldern mit dem DMA organisiert werden, ohne bei jedem neuen Start alle Initialinformationen (Quellen-/Zieladressen, Blocklänge u.a.m.) neu vorgeben zu müssen.
Die Datenbits D3, D4, D5 und D6 spezifizieren mit einer '1' jeweils ein folgendes Datenbyte mit der Startadresse des DMA-Port A - die des DMA-Port B folgt auf das Schreibregister WR4 - und der Blocklänge der zu transferierenden Bytes. Startadresse und Blocklänge umfassen 16 Bit, die in einen Low-Teil (untere 8 Bit) und einen High-Teil (obere 8 Bit) aufgespalten werden. Startadresse und Blocklänge werden nur, wenn es notwendig ist, dem U858-DMA mehrmals vorgegeben. Eine einmal eingegebene Startadresse ebenso wie eine einmal eingegebene Blocklänge werden im DMA-Schaltkreis bis zum nächsten Hardware- oder Softwaresystemrücksetzen (RESET) gespeichert.

Tafel 2.5. Schreibregister WR0 des U858-DMA

Schreibregister WR0

0	BLOCK HIGH	BLOCK LOW	ADRA HIGH	ADRA LOW	SOURCE	C1	C0

Eine '1' spezifiziert: Blocklänge high/low folgt
Startadresse Port A high/low folgt

SOURCE
= 1 Port A Lesen - B Schreiben
= 0 Port B Lesen - A Schreiben

C1 C0
= 00 verboten
= 01 Datentransfer
= 10 Datensuche
= 11 Datentransfer und Datensuche

Startadresse Port A - Blocklänge

A7	A6	A5	A4	A3	A2	A1	A0
A15	A14	A13	A12	A11	A10	A9	A8
BL7	BL6	BL5	BL4	BL3	BL2	BL1	BL0
BL15	BL14	BL13	BL12	BL11	BL10	BL9	BL8

Schreibregistergruppe WR1 (Port A) und WR2 (Port B) (Tafel 2.6.) - Die Schreibregister WR1 und WR2 dienen im wesentlichen zur Spezifizierung der beiden DMA-Ports A und B (IORQ/MREQ, Inkrement/Dekrement Adresse, Zeitverhalten). Es muß zusammen mit dem ggf. nachfolgenden Zeitsteuerbyte einmal für den Port A (WR1) und einmal für den Port B (WR2) ausgegeben werden - festgelegt durch das Datenbit D2.
Das Datenbit D3 spezifiziert, ob es sich bei dem angeschlossenen Ein-/Ausgabemodul um ein Peripheriegerät (IORQ-Zugriff - input output request) oder um einen Speicher (MREQ-Zugriff - memory request) handelt.
Die Datenbits D4 und D5 schließlich legen fest, ob die im Anschluß an das Schreibregister WR0 (Port A) bzw. die im Anschluß an das Schreibregister WR4 (Port B) ausgegebenen Portadressen - nach einem übertragenen Byte - inkrementiert oder dekrementiert werden oder ob sie für alle Datenbytes fest sind.

Tafel 2.6. Schreibregister WR1 und WR2 des U858-DMA

Schreibregister WR1 und WR2

0	TIME BYTE	ADR FIX	INC DEC	IORQ MREQ	PORT A/B	0	0

TIME BYTE
= 1 Zeitsteuerbyte folgt
= 0 Zeitsteuerbyte folgt nicht

ADR FIX
= 1 Adresse bleibt fest
= 0 Adresse wird inkrementiert oder dekrementiert

INC DEC
= 1 Adresse wird inkrementiert
= 0 Adresse wird dekrementiert

IORQ MREQ
= 1 IORQ-Zugriff
= 0 MREQ-Zugriff

PORT A/B
= 1 Port A
= 0 Port B

Zeitsteuerbyte

WR	RD	0	0	MREQ	IORQ	T1	T0

T1 T0
= 00 4 X T
= 01 3 X T (Standard)
= 10 2 x T
= 11 verboten

Falls D2, D3, D6 oder D7 mit '0' spezifiziert werden, endet das entsprechende Steuersignal (WR, RD, MREQ oder IORQ) eine halbe Taktzeit vor dem Zyklusende.

Das Datenbit D6 im Schreibregister WR1 und WR2 dient zur Kennzeichnung, ob der Ausgabe des Schreibregisters (WR1/WR2) die Festlegung des Zeitsteuerbytes (unmittelbar) folgt oder nicht. Das Zeitsteuerbyte dient der Anpassung der jeweiligen DMA-Ports (A oder B) an das Zeitverhalten der angeschlossenen Ein-/Ausgabemodule. Die Datenbits D0 und D1 im Zeitsteuerbyte legen fest, aus wieviel U880-Systemtakten T (2, 3 oder 4) ein DMA-Zyklus (Lesen bzw. Schreiben) besteht. Die Datenbits D2, D3, D6 und D7 spezifizieren mit einer '0', daß die U880-Systemsignale IORQ, MREQ, RD und WR bei der Generierung durch den U858-DMA jeweils um einen halben Takt verkürzt werden.

Schreibregistergruppe WR3 (Tafel 2.7.) - Das Schreibregister WR3 dient der Interruptsteuerung, der Masken- und Matchbytespezifizierung, der Freigabe des DMA-Verkehrs und es enthält das Stopkennzeichen.
Eine '1' in der Bitposition STOP bewirkt, daß in der Betriebsart 'Datentransfer mit Datensuche' der DMA-Verkehr abgebrochen wird, sobald das

Vergleichsbyte (Matchbyte) gefunden wurde. Ist dieses Stopbit nicht gesetzt, erfolgt der Abbruch des DMA-Verkehrs erst am Ende des Blockes. Ein im Block gefundenes Matchbyte wird dann über den Interruptvektor V2=1, V1=1, ein nicht gefundenes Matchbyte über den Interruptvektor V2=1, V1=0 signalisiert, vorausgesetzt, im Schreibregister WR4 wurde 'status effects interrupt vector' programmiert.
Die Datenbits D5 (IE interrupt enable) und D6 (CE chip enable) geben das Interruptsystem des U858 und den DMA-Verkehr frei. Ist WR3 das letzte ausgegebene Schreibregister, ist die Chipfreigabe (CE) mit dem Start des U858-DMA identisch. Die Datenbits 'interrupt enable' und 'chip enable' im Schreibregister WR3 entsprechen den äquivalenten Kommandos im Schreibregister WR6.
Die Datenbits D3 und D4 signalisieren, daß das Maskenbyte und das Vergleichsbyte (Matchbyte) unmittelbar auf das Schreibregister WR3 folgen. Das Vergleichsbyte stellt das zu suchende Datenbyte in der Betriebsart 'Datensuche' und 'Datentransfer mit Datensuche' dar. Das Maskenbyte dient zur Ausschließung (Dx=1) einzelner Bitpositionen aus diesem Vergleichsprozeß.

Tafel 2.7. Schreibregister WR3 des U858-DMA

Schreibregister WR3

1	CE	IE	MATCH	MASKE	STOP	0	0

Eine '1' spezifiziert:

CE	U858-DMA Freigabe (chip enable)
IE	Interruptfreigabe (interrupt enable)
MATCH	Matchbyte folgt
MASK	Maskenbyte folgt
STOP	Stop bei Vergleich (stop on match)

Maskenbyte

M7	M6	M5	M4	M3	M2	M1	M0

Eine '0' in den einzelnen Bitpositionen führt zur Einbeziehung des Datenbits in den Vergleich.

Matchbyte (Vergleichsbyte)

S7	S6	S5	S4	S3	S2	S1	S0

In der Betriebsart 'Datensuche' und 'Datentransfer und Datensuche' wird dem U858-DMA über das Matchbyte das zu suchende Bitmuster vorgegeben.

Schreibregistergruppe WR4 (Tafel 2.8.) - Das Schreibregister WR4 dient im wesentlichen der Festlegung der Betriebsart (1-BYTE, CONTINOUS oder BURST), bezogen auf den Zeitablauf des DMA-Verkehrs. Außerdem wird über das Schreibregister WR4 das Interruptverhalten festgelegt.
Eine '1' in den Bitpositionen D2 und D3 spezifiziert eine folgende Startadresse für den DMA-Port B - äquivalent den Bitpositionen D3 und D4 im

Schreibregister WR0 für den DMA-Port A. Auch hier gilt, daß diese Adresse nur einmal vorgegeben werden muß. Der DMA speichert sie für beliebig viele Datenübertragungen bis zum nächsten RESET, auch wenn der DMA-Baustein im Zeitverhalten, im Interruptverhalten oder in anderen Parametern umprogrammiert wird.
In den Bitpositionen D5 und D6 des Schreibregisters wird die Übertragungsablaufsbetriebsart programmiert.
Die Bitposition D4 im Schreibregister WR4 dient zur Kennzeichnung eines folgenden Interruptsteuerbytes (interrupt control byte). Es eröffnet, obwohl selber sequentiell adressiert, eine neue Steuerwortsequenz für den Interruptvektor und den Pulszähler. Der Interruptvektor muß nur dann ausgegeben werden, wenn der U858-DMA im Interruptbetrieb arbeiten soll.
Das Datenbit D0 im Interruptsteuerbyte spezifiziert, daß ein Interrupt generiert werden soll, sobald das Vergleichsbyte (Matchbyte) gefunden wurde, während das Datenbit D1 eine Interruptgenerierung am Blockende zur Folge hat. Die Bitposition D2 (Pulserzeugung einschalten) und D3 (Pulszähler folgt) dient der Steuerung der Pulserzeugung. Der auf des Interruptsteuerbyte auszugebende Pulszähler wird beim DMA-Verkehr mit den unteren acht Bit des Bytezählers verglichen. Bei Übereinstimmung wird ein Impuls auf der Interruptleitung generiert. (Wird von der U880-CPU wegen des laufenden DMA-Verkehrs nicht ausgewertet.)
Die Bitposition D4 zeigt einen auf das Interruptsteuerbyte und ggf. auf den Pulszähler folgenden Interruptvektor an. Die Bitposition D5 dient der Interruptvektormodifikation (V2,V1).
Das Datenbit D6 im Interruptsteuerbyte bewirkt eine Interruptauslösung, bevor der DMA-Schaltkreis den U880-Systembus für seine Zwecke über das Signal BUSRQ (bus request) anfordert.

Tafel 2.8. Schreibregister WR4 des U858-DMA

Schreibregister WR4

1	M1	M0	IC BYTE	ADR B HIGH	ADR B LOW	0	1

M1 M0
= 00 1-BYTE-Mode
= 01 CONTINOUS-Mode
= 10 BURST-Mode
= 11 ohne Bedeutung

IC BYTE
= 1 Interruptsteuerbyte folgt
= 0 Interruptsteuerbyte folgt nicht

ADR B HIGH/LOW
= 1 Startadresse Port B high/low folgt
= 0 Startadresse Port B high/low folgt nicht

Startadresse Port B

A7	A6	A5	A4	A3	A2	A1	A0
A15	A14	A13	A12	A11	A10	A9	A8

Interruptsteuerbyte

0	INTER BEFORE	STATUS AFFECT	INTER VEKTOR	PULS COUNT	PULS GENE	INTER BLOCK	INTER MATCH

Eine '1' spezifiziert:
- Interrupt vor einer DMA-Anforderung (before requesting bus)
- Interruptvektormodifikation (status affects interrupt vektor)
- Interruptvektor folgt
- Pulszähler folgt (puls counter)
- Pulsgenerierung einschalten
- Interrupt nach Blockende
- Interrupt, wenn Matchbyte gefunden

Pulszähler

P7	P6	P5	P4	P3	P2	P1	P0

Interruptvektor

V7	V6	V5	V4	V3	V2*	V1*	V0

Die Bitpositionen V7, V6 ... V3 sind der konstante Teil des Interruptvektors. Die Bitpositionen V2 und V1 werden bei 'Status Affects Interrupt-Vektor' (Interruptsteuerbyte / D5=1) modifiziert:

V2 V1 = 00 Interrupt bei READY
= 01 Interrupt, wenn Matchbyte gefunden
= 10 Interrupt bei Blockende
= 11 Interrupt bei Blockende und wenn Matchbyte gefunden

Schreibregistergruppe WR5 (Tafel 2.9.) - Das Schreibregister WR5 dient zur Festlegung des aktiven Pegels (low/high) vom DMA-Signal READY, der Spezifizierung des CE-Anschlußes (CE chip enable) und ggf. der Autorepeatfunktion (automatisches Wiederholen der zuletzt programmierten Funktion).

Tafel 2.9. Schreibregister WR5 des U858-DMA

Schreibregister WR5

1	0	STOP	CE	READY	0	1	0

STOP = 0 Stop am Blockende
= 1 Autorepeat am Blockende

CE = 0 Chip Enable
= 1 Chip Enable / Wait - Multiplex

READY = 0 Ready aktiv 'low'
= 1 Ready aktiv 'high'

Schreibregistergruppe WR6 (Tafel 2.10.) - Im Schreibregister WR6 wird ein Zustandsschlüssel F4, F3, F2, F1 und F0 kodiert. Im einzelnen lassen sich über diesen Zustandsschlüssel folgende Kommandos kodieren:

C3	RESET	Stop des laufenden DMA-Verkehrs und Rücksetzen des U858-DMA (Ausschalten Interrupt und BUSRQ-Logik, Reset Interrupt-Latches, Freigabe FORCE READY, Reset Autorepeat WR5, Reset WAIT WR5, Reset Zeitsteuerbyte WR1/WR2 T=3).
C7(CB)	RESET PORT TIME	Die Zykluszeit im Zeitsteuerbyte wird auf T=3 gesetzt (C7=Port A / CB=Port B).
CF	LOAD	Laden der Startadresse des Datenquellenports A oder B (Neuübernehmen der alten oder Einschreiben einer neuen Startadresse - High- und Low-Teil WR0/WR4 - des Datenquellenkanals in den zugehörigen Abarbeitungsadreßzähler und Löschen des Bytezählers des U858-DMA zur Vorbereitung einer neuen DMA-Operation ohne Autorepeat. Eine ggf. neue Startadresse muß vor LOAD über das Schreibregister WR0/WR4 vorgegeben werden. (LOAD nur für Datenquellenport geben)
D3	CONTINUE	Neustart mit altem Initialisierungsstand (Autorepeat).
AB	ENABLE INT	Das Interruptsystem wird freigegeben.(Verwendung nicht in U880-Systemen).
AF	DISABLE INT	Das Interruptsystem wird gesperrt.(Verwendung nicht in U880-Systemen.)
A3	RESET INT	Rücksetzen und Freigabe des Interruptsystems für erneute Unterbrechungen. (Verwendung bei Interrupt ohne RETI nicht in U880-Systemen).
87	ENABLE DMA	Alle Operationen des U858-DMA mit Ausnahme des Interrupts werden freigegeben.
83	DISABLE DMA	Alle Operationen des U858-DMA mit Ausnahme des Interrupts werden gesperrt.
BB	READ MASK	Spezifizierung, daß das folgende ausgegebene Datenbyte die Lesemaske darstellt. Die Lesemaske dient zur Kennzeichnung des Inhalts mehererer in der Folge einzulesender Bytes.
A7	RESET READ	Das Kommando BB (Read Mask / Lesemaske folgt) wird zurückgesetzt. Der Lesezeiger zeigt danach auf das Statusregister.
BF	READ STATUS	Das nächste vom DMA in die CPU eingelesene Byte stellt das Statusregister dar.
B3	FORCE READY	'Erzwungenes READY' - wenn der DMA-Verkehr trotz des inaktiven READY ablaufen soll (gilt auch im BURST-Mode).
B7	ENABLE RETI	Der DMA sendet erst wieder ein Bus Request nachdem ein RETI-Befehl abgearbeitet wurde.
8B	INIT STATUS BYTE	Re-Initialisierung (Setzen / =1) der Bits D4 und D5 im Statusbyte des U858-DMA (BLOCK END; MATCH).

Auf das Schreibregister WR6 folgt ggf. die Ausgabe der Lesemaske. Sie dient der Kennzeichnung von bis zu sieben in der Folge einzulesenden Bytes aus dem Leseregistern RR0 bis RR6:

Leseregister RR0	Statusbyte
Leseregister RR1	Blocklänge (low)
Leseregister RR2	Blocklänge (high)
Leseregister RR3	Startadresse Port A (low)
Leseregister RR4	Startadresse Port A (high)
Leseregister RR5	Startadresse Port B (low)
Leseregister RR6	Startadresse Port B (high)

Das über das Statuslesegesuch (Kommando BF) oder über die Lesemaske (Kommando BB) im Schreibregister WR6 angeforderte Statusbyte wird vor allem Verwendung finden, wenn der DMA-Verkehr nicht im Interrupt-, sondern im Abfragebetrieb (polling) erfolgen soll.

Tafel 2.10. Schreibregister WR6 des U858-DMA

Schreibregister WR6

1	F4	F3	F2	F1	F0	1	1

F4 F3 F2 F1 F0

=10000	U858-DMA Reset
=10001	Zeitverhalten Port A Reset
=10010	Zeitverhalten Port B Reset
=10011	Bytezähler auf Null setzen / Laden Startadressen A/B
=10100	Restart mit momentanem Adressenstand
=01010	Interruptfreigabe
=01011	Interruptsperre
=01000	Interrupt zurücksetzen
=00001	U858-DMA freigeben
=00000	U858-DMA sperren
=01110	Lesemaske folgt
=01001	Rücksetzen Lesemaske
=01111	Lesen des Status
=01100	Erzwungenes READY-Signal
=01101	Freigabe der Busanforderung nach RETI
=00010	Rücksetzen des Statusbyte (BLOCK END; MATCH)

Lesemaske

0	PORT B	PORT B	PORT A	PORT A	BYTE HIGH	BYTE LOW	STATUS BYTE

Über die Lesemaske wird ein Lesegesuch für die einzelnen Leseregister (Dx=1) an den U858-DMA ausgegeben. Die entsprechenden Datenbytes können danach vom U858-DMA in der folgenden Reihenfolge gelesen werden: Statusbyte, Blocklänge (Bytezähler) low/high, Adresse Port A low/high, Adresse Port B low/high.

Statusbyte

X	X	BLOCK END	MATCH	INT PENDING	X	READY AKTIV	CYC

Eine '0' spezifiziert:
- Blockende erreicht
- Vergleichsbyte (Matchbyte) gefunden
- Nicht abgeschlossener Interrupt liegt an
- READY aktiv
- U858-DMA Zyklus ist nicht beendet

X = 0 oder 1 - ohne Bedeutung

3. 16-Bit-Mikroprozessorsystem U 8000

Das hochmoderne 16-Bit-Mikroprozessorsystem U8000 besteht aus drei Grundbausteinen:

U8001-CPU	zentrale Verarbeitungseinheit mit 64 KByte Adreßraum
U8002-CPU	zentrale Verarbeitungseinheit mit 128 x 64 KByte Adreßraum
U8010-MMU	Speicherverwaltungsbaustein (MMU memory management unit)

Alle drei Bausteine werden in N-Kanal-MOS Technologie hergestellt. Ein 4-MHz-Einphasentakt wird von den Schaltkreisen als externe Taktfrequenz benötigt. Durch spezielle Schaltungsmaßnahmen sind die U880-Peripherieschaltkreise U855-PIO, U856-SIO, U857-CTC und U858-DMA auch im System U8000 verwendbar /19/,/20/,/21/,/22/,/23/.

3.1. Zentrale Verarbeitungseinheit U 8000-CPU

Ebenso wie im 8-Bit-System U880 übernimmt in einem 16-Bit-System U8000 die CPU die gesamte Systemüberwachung, die Befehlsabarbeitung (ggf. geteilt mit EPU Co-Prozessoren / EPU extended processing unit / z.B.: Gleitkommaarithmetik, Netzwerkinterface ...) und die Abwicklung der Interruptanforderungen. Die mit den U8000-Befehlen bearbeitbaren Datentypen sind: Bits in einem Byte, Bits in einem 16-Bit-Wort, BCD-Zahlen (4 Bit) in einem Byte, Bytes, 16-Bit-Worte, 32-Bit-Langworte, Ketten von bis zu 65536 Bytes und Ketten von bis zu 32768 16-Bit-Worten. Mit dem gegenüber dem Mikroprozessor U880 weit umfangreicheren und homogeneren Befehlssatz des U8000 besteht die Möglichkeit, 32-Bit-Festkommamultiplikationsbefehle und 32-Bit-Festkommadivisionsbefehle hardwaremäßig ausführen zu können - eine von vielen Erweiterungen gegenüber dem U880. Die durch die 16-Bit-Struktur verdoppelte Verarbeitungsbreite führt zu einer Erhöhung der Verarbeitungsgeschwindigkeit. Die Möglichkeit, die U8000-CPU im 'Normalmode' oder im 'Systemmode' zu betreiben, eröffnet neue Möglichkeiten der Gestaltung der Software (Anwenderprogramm im Normalmode / Betriebssystem im Systemmode) mit hoher Systemzuverlässigkeit und vom Anwenderprogramm nicht beeinflußbaren Schutzmechanismen. (Befehle, die den Mikrorechnerzustand ändern, wie Ein-/Ausgabeoperationen, Veränderung von Statusregisterinhalten u.a.m. können ohne Intervention der CPU nur im Systemmode abgearbeitet werden - s.a. Befehlsliste in Tafel 3.1.: 'priviligierte Befehle'.) Ein hochorganisiertes Interruptsystem, Multi-Mikroprozessorsystemunterstützung (multi micro control), getrennte Programm-, Daten- und Stackbereiche und vieles andere mehr ergeben zusammengenommen einen Qualitätssprung beim Übergang von einem 8-Bit-U880-System auf ein 16-Bit-U8000-System, so daß ein direkter Vergleich beider Familien, der hauptsächlich die Verdopplung der Verarbeitungsbreite beinhaltet, in keiner Weise mehr verhältnismäßig erscheint.

3.1.1. Adreßraum

Die U8000-CPU wird in zwei Ausführungen, der U8001-CPU (48-Pin-Version) und der U8002-CPU (40-Pin Version), angeboten. Der grundlegende Unterschied zwischen beiden Schaltkreisen besteht ausschließlich im adressierbaren Speicherbereich:

U8001-CPU segmentierte Version 128 x 64 KByte Adreßraum
(7 Segmentleitungen und 16 Adreßleitungen)
U8002-CPU nichtsegmentierte Version 64 KByte Adreßraum
(16 Adreßleitungen)

Während der U8002 einen linear durchgängigen Adreßraum von 65536 Byte, entsprechend 64 KByte (Adresse 0000H bis FFFFH) besitzt, hat der U8001 einen in 128 Segmente aufgeteilten 'segmentierten' Adreßraum von 128 x 65536 Byte entsprechend 8 MByte (00H bis 7FH x 0000H bis 0FFFFH).

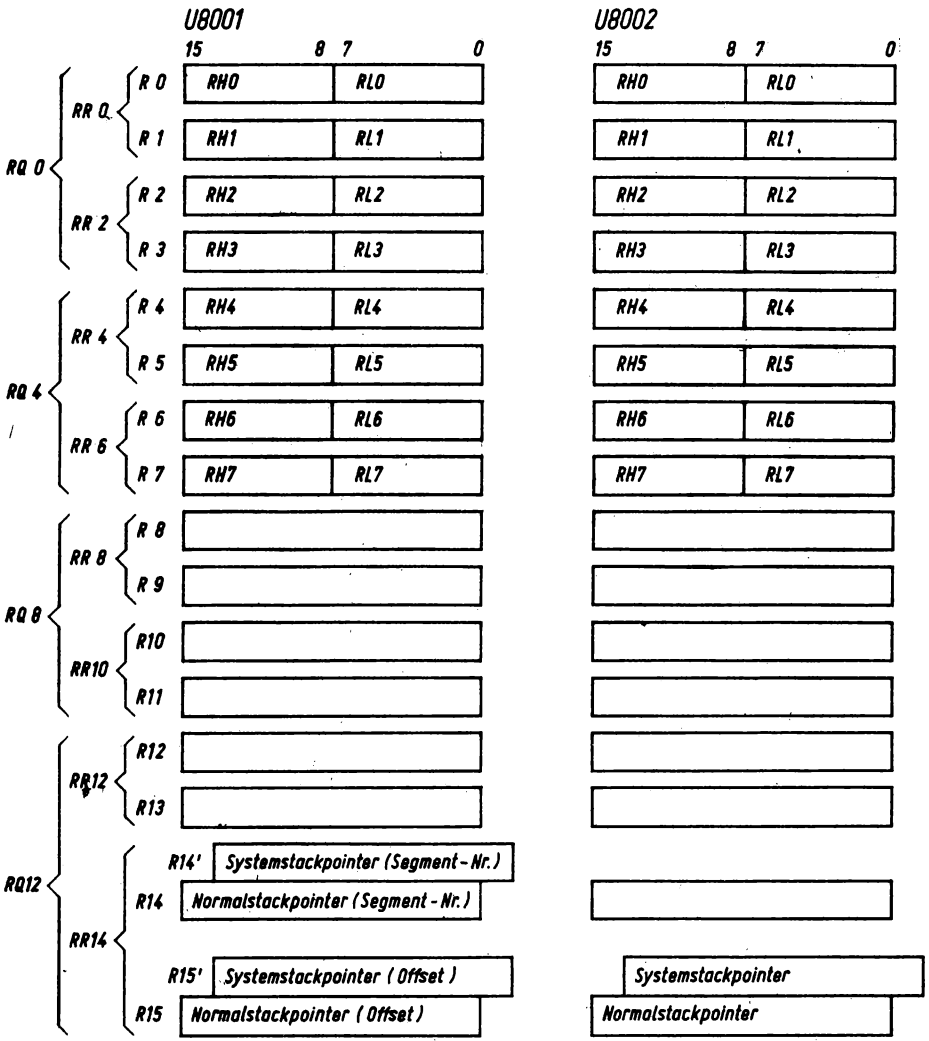


Bild 3.1. Registersatz des Mikroprozessors U8001/U8002

Berücksichtigt man, daß 'Normalmode' und 'Systemmode' ebenso wie 'Programmzugriff', 'Datenzugriff' und 'Stackzugriff' über die entsprechenden Hardwaresignale auf unterschiedliche Speicherräume gelegt werden können, ergibt sich - bezogen auf den angegebenen Adreßbereich von 64 KByte bzw. 8 MByte - eine Versechsfachung des theoretisch adressierbaren Speicherraumes auf 384 KByte beim U8002 bzw. 48 MByte beim U8001.
Wichtig ist, daß auch die segmentierte Version der U8000-CPU, der U8001-Schaltkreis, zur Entlastung des Programmspeichers - die in den Befehlen kodierten Adressen sind im nichtsegmentierten Mode mit 16 Bit wesentlich kürzer als im segmentierten Mode mit 32 Bit - durch Softwareprogrammierung jeweils in einem Speichersegment, im nichtsegmentierten Mode, betrieben werden kann. Zusätzlich ist zu beachten, daß bei der Programmierung der U8001-CPU im segmentierten Mode zwei unterschiedliche Adreßformate existieren. Standardformat ist eine 32-Bit-Adresse (Segmented Long Offset / 16-Bit-Segmentoffset und 7-Bit-Segmentnummer) mit der der gesamte U8001-Adreßbereich überstrichen werden kann. Daneben existiert im segmentierten Mode ein 16-Bit-Adreßformat (Segmented Short Offset / 8-Bit-Segmentoffset Low-Teil und 7-Bit-Segmentnummer) bei dem die oberen 8 Bit des Segmentoffset immer Null sind und das zur programmspeichersparenden Adressierung innerhalb eines 256 Adressen umfassenden Teilbereichs in einem U8001-Segment verwendet werden kann (s. Befehlsliste in Tafel 3.1. und Speicherverwaltung im System U8000 im Abschnitt 3.2.).

3.1.2. Registersatz

Im Mikroprozessor U8000 existieren sechzehn allgemeine 16-Bit-Register, die wahlweise als 8-Bit-Register (RH0 ...RH7 und RL0 ...RL7), als 16-Bit-Register (R0 ...R16), als 32-Bit-Register (RR0, RR2 ...RR14) oder als 64-Bit-Register (RQ0, RQ4, RQ8 und RQ12) aufgeteilt bzw. gruppiert werden können (Bild 3.1.).
Alle Register des allgemeinen Registersatzes (ausgenommen R14/R14' und R15/R15') könnnen die Funktion eines Akkumulators übernehmen. Das gleiche gilt für die Indexregisterfunktion (ausgenommen R0/RR0). Der U8001-Registersatz unterscheidet sich vom U8002-Registersatz nur durch die Größe und die Funktion des Registers R14 - beim U8002 16 Bit lang ohne Zusatzfunktion, übernimmt es beim U8001 zweimal 16 Bit lang die Funktion, die Segmentnummer des Normal- und Systemstackpointers zu speichern. Als Stackpointer selbst dient grundsätzlich das Register R15 (U8001 nur Offset des Stackpointers / U8002 kompletter Stackpointer). Der Stackpointer ist zweimal vorhanden, einmal für den Normalmode und einmal für den Systemmode, um beide Betriebsarten möglichst weitgehend zu entkoppeln. Im Normalmode ist nur der Normalstackpointer in den Registern R14/R15 für den U8001 und im Register R15 für den U8002 nutzbar. Im Systemmode wird der Systemstackpointer in den Registern R14'/R15' (U8001) und R15' (U8002) abgelegt.
Neben den allgemeinen Registern und dem noch zu behandelnden Programmstatusregister existiert ein Befehlszählerregister (PC) und ein Refreshregister (R) für den Anschluß dynamischer RAM-Speicherbauelemente (RE=0 refresh disable / RE=1 refresh enable / Rate=1...64 entsprechend 1 bis 64 µs Refreshrate / ROW-Adressierung von bis zu neun Speicherspalten - wird nach jedem Refreshimpuls zweimal inkrementiert / Bild 3.2.)

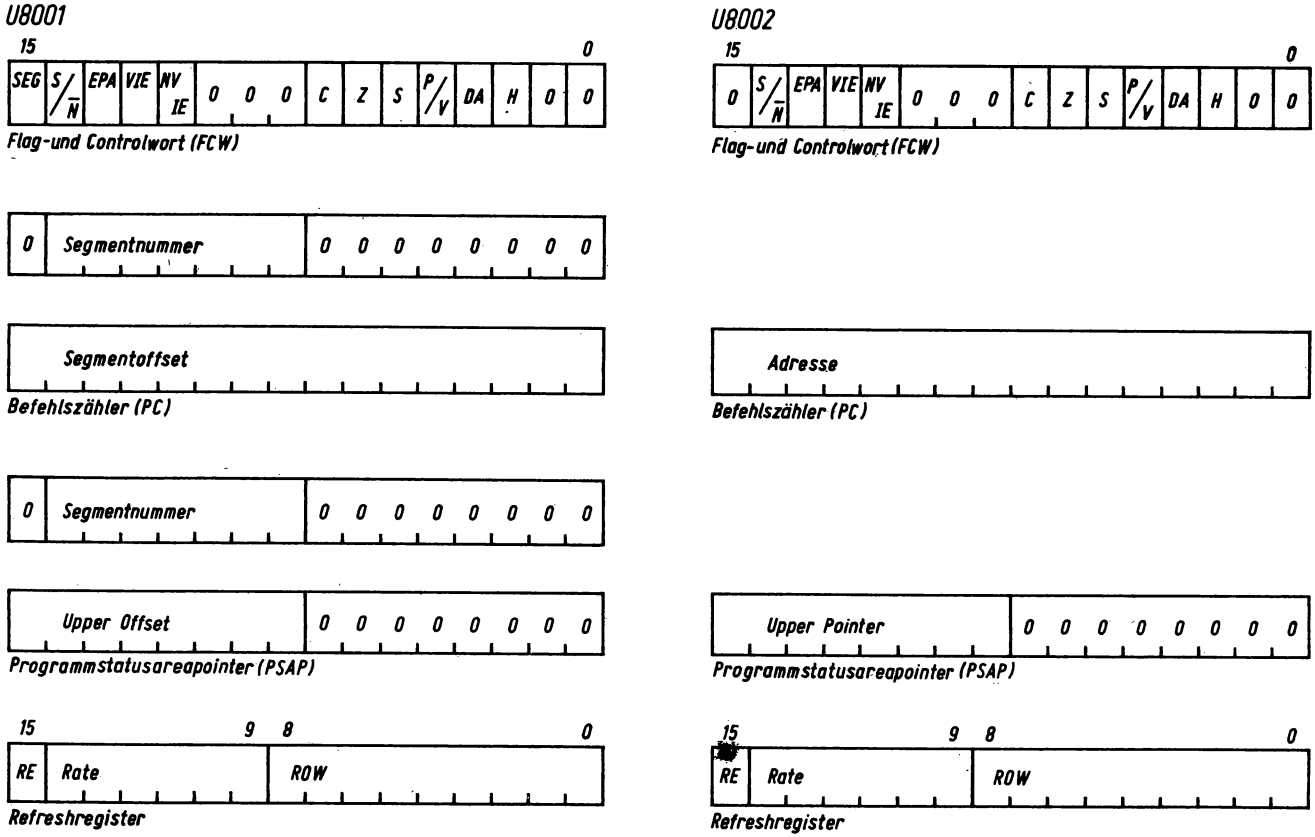


Bild 3.2. Programmstatusregister des Mikroprozessors U8001/U8002

3.1.3. Programmstatusregister

Die Programmstatusregister (Bild 3.2.) enthalten das Flag- und Controlwort (FCW), den Befehlszähler (PC) und den Programmstatusareapointer (PSAP). Das Flag- und Controlwort enthält beim U8001 fünf und beim U8002 vier Steuerbits SEG (segmentation mode), S/N (system/normal mode), EPA (extended processor architectur mode), VIE (vectored interrupt enable) und NVIE (none vectored interrupt enable), die jeweils durch Setzen oder durch Rücksetzen (vom Programmierer) die folgenden Steuerfunktionen übernehmen:

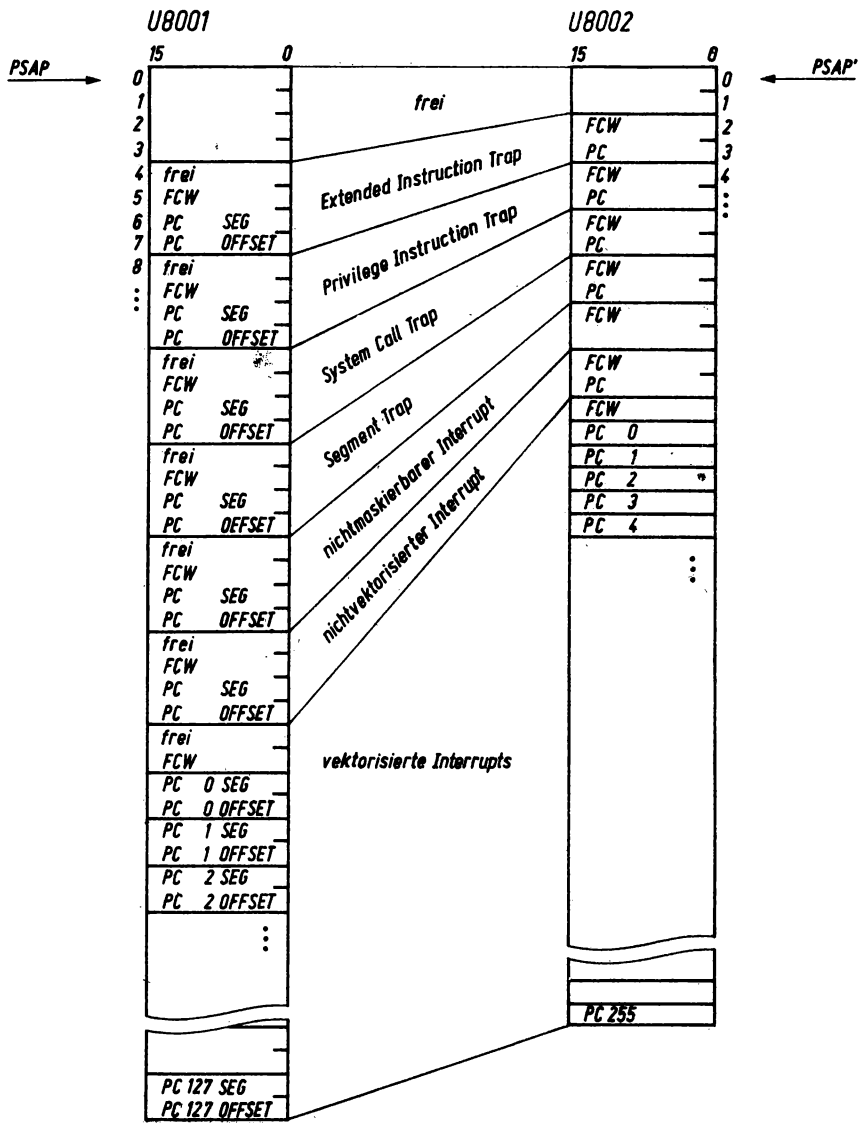


Bild 3.3. Aufbau der Programmstatusarea beim U8001/U8002

U8001-CPU arbeitet im segmentierten (SEG=1) oder im nichtsegmentierten (SEG=0) Mode, U8001/2-CPU arbeitet mit (EPA=1) oder ohne (EPA=0) Co-Prozessor EPU und U8001/2-CPU arbeitet mit (VIE=1 bzw. NVIE=1) oder ohne (VIE=0 bzw. NVIE=0) vektorisierten bzw. nichtvektorisierten Interrupt.
Neben diesen vier Steuerbits enthält das FCW die schon vom U880 bekannten (s. Abschnitt 2.1.3.) Statusidentifikatorbits Übertragsflag C (carry), Nullflag Z (zero), Paritäts-/Überlaufsflag (P/V), Vorzeichenflag S (sign), Additions-/Subtraktionsflag DA (decimal adjust) und das Halbübertragsflag H (half carry).
Der Befehlszähler unterscheidet sich in seiner Funktion gegenüber dem Befehlszähler beim U880 in keiner Weise. Beim U8001 ist allerdings, aus dem segmentierten Adressraum resultierend, ein etwas modifizierter Aufbau zu beachten (Segmentnummer und Offset - also Segmentauswahl und Versatz bezogen auf den Segmentanfang).
Der Programmstatusareapointer (PSAP) zeigt auf einen Speicherbereich, in dem die Programmzustände (FCW- und PC-Werte) für die verschiedenen Trap- und Interruptquellen (siehe Abschnitt 3.1.4.) abgelegt werden (Bild 3.3.).
Der Aufbau der Programmstatusarea erfolgt in der Regel in der Programminitialisierungphase vom System- oder Anwenderprogramm. Ihr Inhalt kann je nach Aufgabenstellung fest sein oder im Laufe der Programmabarbeitung modifiziert werden.
Zu beachten ist beim U8001 und U8002, daß nach dem Systemreset bei einem Neuanlauf des Mikroprozessors in den ersten vier (U8001) bzw. drei (U8002) Speicherworten der zu ladende Initialprogrammstatus steht. Die eigentliche Programmabarbeitung beginnt bei der hier angegebenen Befehlszähleradresse mit dem auf der Speicherwortadresse 1 stehenden Flag- und Controlwort (FCW).

Adresse:	U8001:	U8002:
0	frei	frei
1	FCW	FCW
2	Segmentnummer	Befehlszähler
3	Segmentoffset	erster ausführbarer Befehl
4	erster ausführbarer Befehl	

3.1.4. Interruptsystem/Trapsystem

Das Mikroprozessorsystem U8000 besitzt ein außerordentlich flexibles und leistungsfähiges Interrupt- und Trapsystem.
Interrupts sind von der U8000-CPU Befehlsabarbeitung vollkommen unabhängige, also asynchrone Meldungen von externen Quellen an den Mikroprozessor. Traps sind vom momentanen Systemzustand des Mikrorechners und von der Abarbeitung bestimmter Befehle abhängige, also synchrone, Meldungen des Mikrorechners an sich selbst, die vom System genau wie Interruptmeldungen behandelt werden.

Grundsätzlich kennt der Mikroprozessor U8000 drei voneinander unabhängige Interruptsignaleingänge und vier unterschiedliche Trapsignale:

U8000-Interrupts	NMI	nichtmaskierbarer Interrupt
	VI	maskierbarer Vektorinterrupt (Sammelinterrupt)
	NVI	maskierbarer Interrupt ohne Vektor / Non-Vektor (Einzel- oder Sammelinterrupt)
U8000-Traps		System Call Trap (U8001/U8002-CPU Trap)
		Extended Instruction Trap (U8001/U8002-CPU Trap)
		Privileged Instruction Trap (U8001/U8002-CPU Trap)
		Segment Trap (U8010-MMU Trap)

Der nichtmaskierbare Interrupt NMI wird dem externen Interrupteingang mit der höchsten Priorität zugeordnet, der jederzeit unbedingt erkannt und akzeptiert werden muß. Der Unterschied zwischen dem Vektor- und dem Non-Vektorinterrupt ist nur in der Auflösung des einen (VI) in unterschiedliche Serviceroutinen über den Interruptvektor zu suchen, während der andere (NVI) nur eine einzige Serviceroutine (für alle ggf. hinter diesem Signal stehende Quellen) initiiert.
Der System Call Trap ist unmittelbar an die Abarbeitung des SC-Befehls gebunden. Er ermöglicht einen definierten Übergang vom Normalmode in den Systemmode. Der Extended Instruction Trap wird ausgelöst, wenn ein EPA-Co-Prozessorbefehl abgearbeitet wird, ohne daß ein EPA-Co-Prozessor vorhanden ist (EPA-Bit im FCW =0). Dadurch kann ersatzweise zum Beispiel eine Gleitkommabefehlsfolge abgearbeitet werden, wenn kein Gleitkommaprozessor vorhanden ist. Der Privileged Instruction Trap meldet die Abarbeitung eines priviligierten Befehls im Normalmode (S/N-Bit im FCW =0). Ein Segment Trap ist auf ein Ereignis in der U8010-MMU zurückzuführen (s. Abschnitt 3.2.), das definitionsgemäß eine Trapauslösung zur Folge hat (z.B.:Speicherzugriffsverletzung).

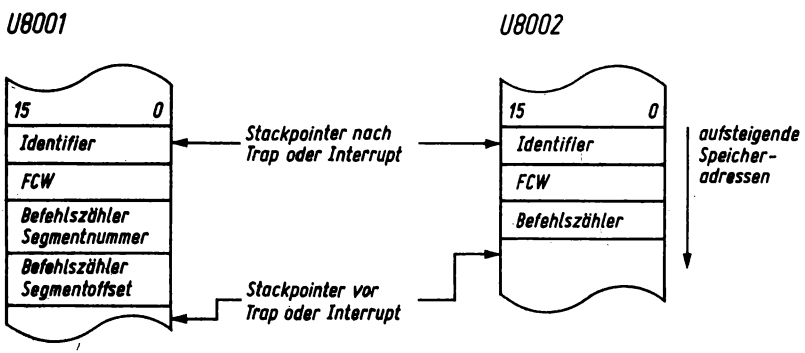


Bild 3.4. Aufbau des Programmstatus beim U8001/U8002 im Stack

Die Priorität der Interrupt- und Trapsignale ist in der folgenden Weise vergeben:

die höchste Priorität haben die drei U8001/U8002-CPU Traps,
gefolgt vom NMI Interrupt,
dem U8010-MMU Trap,
dem VI Interrupt und am Schluß
der NVI Interrupt.

Die Prioritätsverteilung der verschiedenen maskierbaren Vektorinterruptquellen untereinander, wird (wie im System U880) durch eine Prioritätskaskade (daisy chain) der peripheren Ein-/Ausgabebausteine festgelegt.
Sowohl bei der Abarbeitung eines Interrupts als auch bei der Abarbeitung eines Traps läuft immer folgender Vorgang ab:

1. Abspeichern des momentanen Programmstatus bestehend aus dem FCW, dem Befehlszähler und dem 16-Bit-Interrupt-/Trapidentifier im System- oder Normalstack (Bild 3.4.). (Identifier: Bei einem U8001/U8002-CPU Trap das erste Wort des Befehls, der den Trap auslöste - bei einem U8010-MMU-Trap und bei externen Interrupts der MMU-Trap- bzw. Interruptvektor, d.h. der Wert auf dem Datenbus, der von der U8000-CPU während des Interruptanerkennungszyklus gelesen wird.)
2. Automatisches Laden des aktuellen neuen Programmstatus (FCW und PC) aus der Programmstatusarea, festgelegt durch den Inhalt des PSAP.
3. Abarbeitung der zugehörigen Trap- oder Interruptserviceroutine.
4. Rückladen des im Stack befindlichen alten Programmstatus und Rücksprung in das unterbrochene Programm.

Dabei ist zu beachten, daß für alle vektorisierten Interruptquellen nur ein einheitliches Flag- und Controlwort in der Programmstatusarea (s. Bild 3.3.) existiert, der jeweilige neue Befehlszähler aber über einen für jede vektorisierte Interruptquelle spezifischen 8-Bit-Interruptvektor (U8001: 0,2,4... 254; U8002: 0,1,2... 255) aus einer hintereinanderliegenden Folge von 128 (beim U8001) und 256 (beim U8002) Befehlzählerwerten - sie entsprechen den Ansprungadressen der jeweiligen Serviceroutinen - ausgewählt werden.
Zu beachten ist ferner, daß beim U8001 der Interruptanerkennungsvorgang automatisch immer so abläuft, als würde sich der Prozessor im segmentierten Mode befinden. Daraus folgt, egal ob im Moment die U8001-CPU im segmentierten Mode oder im nichtsegmentierten Mode arbeitet, daß bei einem Trap oder Interrupt immer der Inhalt des Registers R14 (Segmentnummer) im Stack mit abgespeichert wird. Es muß deshalb durch programmtechnisches Hin- und Herschalten - segmentierter Mode / nichtsegmentierter Mode - gewährleistet werden, daß die Rückkehr aus einer Serviceroutine (mit dem Befehl RETI) ebenfalls nur im segmentierten Mode erfolgt.

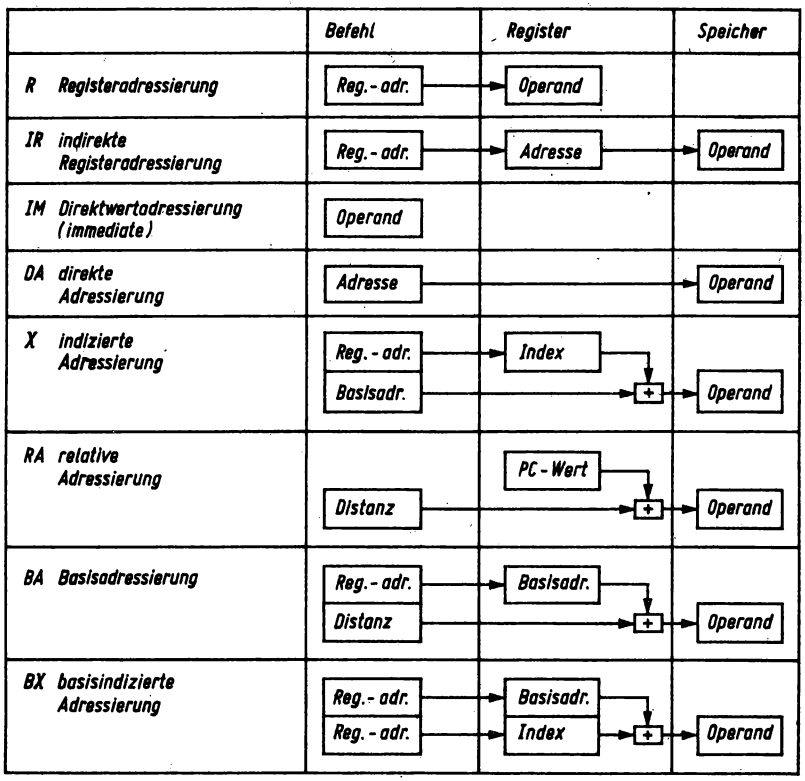


Bild 3.5. Adressierungsarten des Mikroprozessors U8001/U8002

3.1.5. Adressierungsarten

Der Befehlssatz des Mikroprozessors U8001/U8002-CPU beinhaltet acht Adressierungsarten (Bild 15).

R Registeradressierung: Operand ist der Inhalt eines Arbeitsregisters.

z.B.: ADDL RR2/RR4

IR indirekte Adressierung: Operand ist der Inhalt einer Adresse, die durch den Inhalt eines Registers ausgewählt wird.

z.B.: JP@R2

IM Direktwertadressierung: Operand ist im Befehl direkt enthalten.

z.B.: LDB RH0,#100

DA direkte Adressierung: Operand ist der Inhalt einer Adresse die im Befehl angegeben ist.

z.B.: JP C,%2000

X	indizierte Adressierung: z.B.: LD R10,TAB(R3)	Operand ist der Inhalt einer Adresse, deren Wert sich aus der im Befehl angegebenen Adresse (Basisadresse), addiert mit dem Inhalt eines Arbeitsregisters (Index) ergibt.
RA	relative Adressierung: z.B.: DJNZ R5,LOOP	Operand ist der Befehlszählerinhalt, versetzt (plus/minus) um den im Befehl angegebenen Distanzwert (displacement).
BA	Basisadressierung: z.B.: LDL RR2,R1(#255)	Operand ist der Inhalt einer Adresse, deren Wert sich aus der Adresse in einem Arbeitsregister (Basisadresse), versetzt (plus/minus) um den im Befehl angegebenen Distanzwert (displacement) ergibt.
BX	basisindizierte Adressierung: z.B.: LD R3,R8(R4)	Operand ist der Inhalt der Adresse, deren Wert sich aus der Adresse in einem Arbeitsregister (Basisadresse), addiert mit dem Inhalt eines weiteren Arbeitsregisters (Index) ergibt.

3.1.6. Befehlssatz

Der Befehlssatz des Mikroprozessors U8001/U8002-CPU ist in Tafel 3.1. wiedergegeben. Er umfaßt die Befehlsgruppen:

Lade-/Exchangebefehle
Logikbefehle
Bitbefehle
Arithmetikbefehle
Programmsteuerbefehle
Blocktransferbefehle/Zeichenkettenbefehle
Ein-/Ausgabebefehle
CPU-Steuerbefehle
Rotations-/Verschiebebefehle
EPU-Befehle

Mit Hilfe der Ladebefehle wird der Transport von Daten zwischen dem Mikrorechnerspeicher und den Registern der U8000-CPU organisiert (Datentransport vom Speicher zu einem oder mehreren aufeinander folgenden Registern und umgekehrt sowie Laden von Direktwerten). Die Adreßbestimmungsbefehle dienen der Bestimmung von Adreßwerten zur Vorbereitung von Zugriffsoperationen auf komplexe Datenstrukturen. Das Laden von Konstanten in ein Register und das Löschen von Speicherstellen oder Registern vervollständigt die Palette der Ladebefehle. Der wechselseitige Austausch von Daten zwischen verschiedenen Registern sowie zwischen dem Speicher und den Registern ebenso wie der Datentransport von und zum Stackspeicherbereich von Register- und Speicherwerten stellen ganz spezielle Ladebefehlsformen dar.
Die Logikbefehle dienen der logischen Bit-für-Bit-Verknüpfung zweier Operanden - eines Operanden im Register und eines Operanden im Speicher. Ein einzelner Operand im Register oder Speicher kann logisch negiert werden und es kann abgetestet werden, ob alle Bitpositionen eines Operan-

den den Wert Null besitzen. Ein spezieller Testbefehl (test condition code) übernimmt die Abfrage (TRUE oder FALSE) der aktuellen Flagbitbelegung und die Übermittlung des Ergebnisses dieser Abfrage als einzelnen Bitwert (1 oder 0) in das angegebene CPU-Register. Bei der Abarbeitung der logischen Befehle (ausgenommen TCC/TCCB) werden das S-, Z- und P-Flag (Parität) entsprechend dem Ergebnis gesetzt.
Die Bitmanipulationsbefehle gestatten das Abtesten, Setzen (=1) oder Rücksetzen (=0) einer einzelnen Bitposition eines Registers oder Speicherwertes. Die Bitposition ist dabei entweder eine Konstante oder auch ein variabler Wert (in einem Register), der sich im Laufe der Programmabarbeitung erst ergibt (Bitposition: 0...7 bei Byteoperanden, 0...15 bei Wortoperanden). Ein ganz spezieller Bitbefehl (test and set) dient der Unterstützung der Synchronisation von Prozessen in Multiprogramming-Softwareumgebung (z.B: Besetztkennzeichen). Er kopiert erst das MSB-Bit des Register- oder Speicheroperanden in das S-Flag und setzt danach alle seine Bitpositionen auf den Wert Eins. Die Ausführung dieses Befehls ist auch nicht durch einen DMA-Verkehr (BUSRQ) unterbrechbar.
Die Arithmetikbefehle ermöglichen die Ausführung von Festkommaoperationen mit Operanden im Register und Speicher im 2-er-Komplementformat. Festkommamultiplikation und - division (auch von 32-Bit-Long-Wortoperanden) sind dabei besonders bemerkenswert. Das Ergebnis der Operation wird in der Regel in einem Register abgelegt - die dem Ergebnis entsprechenden Flagbitbelegung in das C-, Z-, S- und P/V-Flag (overflow) eingetragen. Durch die Anwendung der multiplen Additions- und Subtraktionsbefehle (...with carry) und durch die Anwendung des Vorzeichenausdehnungsbefehls ist der Aufbau von Arithmetikroutinen beliebiger Genauigkeit möglich. Die Addition und Subtraktion von Bytewerten (ADCB, ADDB, SBCB, SUBB) wird durch den DAB-Befehl auch im gepackten BCD-Format unterstützt. Das Dekrementieren und Inkrementieren eines Wertes in einem Register oder in einer Speicherstelle um einen Wert von 1 bis 16 schließen diese Befehlsgruppe ab.
Die Programmsteuerbefehle dienen dem Aufbau von Programmverzweigungen, von Programmschleifen, von Unterprogrammstrukturen und dem Aufbau eines Interruptsystems. Sie entsprechen weitestgehend in ihrer Wirkungsweise beim U8000 den U880-Befehlen. Der SC-Befehl (system call) dient dem Zugriff aus dem Anwenderprogramm (Normalmode) zu den Ressourcen des Betriebssystems (Systemmode). Seine Abarbeitung löst einen Trap aus (s. a. Abschnitt 3.1.4.), der über das Abspeichern des momentanen Programmstatus (PC und FCW) und das Abspeichern des Befehls selbst (Identifier) zu einer Trapbehandlungsroutine für den Systemcall führt. Die Systemcallnummer (src-Wert), ein Wert zwischen 0 und 255, ermöglicht - aus dem Stack entnommen - in der Trapbehandlungsroutine die Softwarerufauflösung. Die Rückkehr in das aufrufende Programm erfolgt wie bei Interruptserviceroutinen mit dem IRET-Befehl.
Die Blocktransferbefehle dienen - der Name sagt es bereits - dem byteweisen oder wortweisen Transfer und dem Vergleich (Durchmustern) von ganzen Datenblöcken mit einer Länge von bis zu 64 KByte. Die Befehle existieren in Varianten mit Inkrementierung oder Dekrementierung der Adresse(n), als Einzel- oder Kettenbefehle und mit oder ohne automatische Wiederholung. Dadurch können sowohl einzelne Byte- oder Wortketten transferiert und durchgemustert als auch zwei Byte- oder Wortketten miteinander verglichen werden. Zusätzlich kann durch die Anwendung der Übersetzungsbefehle eine alte Bytekette in eine neue Bytekette umgesetzt werden, indem jedes alte Bytekettenelement als Versatzelement in einer spezifischen Übersetzungstabelle benutzt wird. Eine Reihe weiterer Befehle dieser

Gruppe schließlich stülpen über die Übersetzungstabellenbytes eine Suchgruppe, um speziell interessierende Bytewerte herausfiltern zu können.
Die Ein-/Ausgabebefehle existieren jeweils in einer 'Normalvariante' und in einer 'Spezialvariante' - letztere auf die U8010-MMU bezogen. Beide Varianten ermöglichen den Ein-/Ausgabetransfer sowohl einzelner Byte- oder Wortwerte als auch ganzer Byte- oder Wortketten. Alle Befehle dieser Gruppe sind priviligierte Befehle, die ohne Trapintervention durch die CPU-Steuerung nur im Systemmode abgearbeitet werden können.
Die Gruppe der CPU-Steuerbefehle enthält eine Reihe von Standardbefehlen zur Flagbitmanipulation, zur Interruptsteuerung sowie einen Befehl zum Laden der U8000-CPU Steuerregister (FCW, PSAP, NSP und REFRESH) und einen NOP- und einen HALT-Befehl. Ein Befehl zum Laden des Programmstatus, bestehend aus dem FCW- und dem PC-Inhalt, kann besonders effektiv beim Übergang vom Systemmode in den Normalmode oder bei der Abarbeitung von Programmen im nichtsegmentierten Mode auf der segmentierten U8001-CPU verwendet werden. Die Multi-Micro-Befehle dieser Gruppe schließlich dienen der Synchronisation des Zugriffs mehrerer U8000-Prozessoren zu gemeinsamen Ressourcen in einem Multiprozessorsystem.
Die Rotations- und Verschiebebefehle ermöglichen die Ausführung arithmetischer und logischer Rotations- und Verschiebeoperationen. Die Rotationsoperationen können um ein oder zwei Bitpositionen, die Verschiebeoperationen um 0 bis 8 (Byteoperanden), 0 bis 16 (Wortoperanden) oder 0 bis 32 (Long-Wortoperanden) Bitpositionen erfolgen. Die Rotations- und Verschiebeanzahl kann dabei teilweise dynamisch über Registerwerte vorgegeben werden. Zwei Rotationsbefehle dienen vorwiegend der Behandlung von BCD-Zahlen.
Die EPU-Befehlsgruppe ermöglicht den Hardware-/Softwareanschluß von EPU-CO-Prozessoren an die U8000-CPU und die koordinierte Zusammenarbeit einschließlich der Steuerung der wechselseitigen Datenübergabe.

3.1.7. Ein-/Ausgabe

Der Mikroprozessor U8000 besitzt zwei unterschiedliche Ein-/Ausgabeadreßräume, den 'Standard Ein-/Ausgabeadreßraum' (Standard Ein-/Ausgabebefehle) und den 'Spezial Ein-/Ausgabeadreßraum' (Spezial Ein-/Ausgabebefehle), beide jeweils eine 16-Bit-Adresse mit 65536 einzelnen Ein-/Ausgabekanälen umfassend. Der spezielle Ein-/Ausgabeadreßraum ist für die Adressierung ausgewählter Ein-/Ausgabebausteine aus der U8000-Familie vorgesehen - insbesondere zum Anschluß des Speicherverwaltungsbausteins U8010-MMU. Der normale Ein-/Ausgabeadreßraum dient der Realisierung des Anschlusses der allgemeinen Ein-/Ausgabegeräte. Beide Adreßräume werden über unterschiedliche Ein-/Ausgabebefehle angesprochen und unterscheiden sich durch spezifische Hardwaresignalspiele.
Zusätzlich gilt natürlich auch beim U8000 das beim U880 (s. Abschnitt 2.1.7.) zur Ein-/Ausgabe über normale Speicheradressen Gesagte (memory mapped).

Tafel 3.1. Befehlsliste des Mikroprozessors U8001/U8002

Befehlserläuterung	Mnemonik	Datentyp	Adressierungsarten
Lade-/Exchangebefehle			
load (register)	LD	B, W, L	R, IM, IR DA, X BA, BX
load (memory register)	LD	B, W, L	IR, DA, X, BA, BX
load (memory immediate)	LD	B, W, L	IR, DA, X
load address (register)	LDA	W	DA, X, BA, BX
load address relative (register)	LDAR	W	RA
load constant (register)	LDK	W	IM
load multiple	LDM	W	IR, DA, X
load relative to PC (memory/register)	LDR	B, W, L	RA
clear	CLR	B, W	R, IR, DA, X
exchange (register)	EX	B, W	R, IR, DA, X
pop stack	POP	W, L	R IR, DA, X
push stack	PUSH	W, L	R, IM, IR, DA, X
Logikbefehle			
and	AND	B, W	R, IM, IR, DA, X
complement	COM	B, W	R, IR, DA, X
or	OR	B, W	R, IM, IR, DA, X
test	TEST	B, W, L	R, IR, DA, X
test condition code	TCC	B, W	R
exclusive or	XOR	B, W	R, IM, IR, DA, X
Bitmanipulationsbefehle			
bit test (static)	BIT	B, W	R IR, DA, X
bit test (dynamic)	BIT	B, W	R
bit reset (static)	RES	B, W	R, IR, DA, X
bit reset (dynamic)	RES	B, W	R
bit set (static)	SET	B, W	R, IR, DA, X
bit set (dynamic)	SET	B, W	R
bit test and set	TSET	B, W	R, IR, DA, X
Arithmetikbefehle			
add with carry	ADC	B, W	R
add	ADD	B, W, L	R, IM, IR, DA, X
compare (immediate)	CP	B, W	IR, DA, X
compare (register)	CP	B, W, L	R, IM, IR, DA, X
decimal adjust	DAB	B	R
decrement	DEC	B, W	R, IR, DA, X
extend sign	EXTS	B, W, L	R
divide	DIV	W, L	R, IM, IR, DA, X
increment	INC	B, W	R, IR, DA, X
multiply	MULT	W, L	R, IM, IR, DA, X
negate	NEG	B, W	R, IR, DA, X
subtract with carry	SBC	B, W	R
subtract	SUB	B, W, L	R, IM, IR, DA, X
Programmsteuerbefehle			
call procedure	CALL	-	IR, DA, X
call procedure relative	CALR	-	RA
decrement, jump if not zero	DJNZ	B, W	RA
interrupt return	IRET	-	-
jump	JP	-	IR, DA, X
jump relative	JR	-	RA
return from procedure	RET	-	-
system call	SC	-	IM

Blocktransferbefehle / Zeichenkettenbefehle			
compare and decrement	CPD	B, W	IR
compare, decrement and repeat	CPDR	B, W	IR
compare and increment	CPI	B, W	IR
compare, increment and repeat	CPIR	B, W	IR
compare string and decrement	CPSD	B, W	IR
compare string, decrement and repeat	CPSDR	B, W	IR
compare string and increment	CPSI	B, W	IR
compare string, increment and repeat	CPSIR	B, W	IR
load and decrement	LDD	B, W	IR
load, decrement and repeat	LDDR	B, W	IR
load and increment	LDI	B, W	IR
load, increment and repeat	LDIR	B, W	IR
translate and decrement	TRDB	B	IR
translate, decrement and repeat	TRDRB	B	IR
translate and increment	TRIB	B	IR
translate, increment and repeat	TRIRB	B	IR
translate, test and decrement	TRTDB	B	IR
translate, test, decrement and repeat	TRTDRB	B	IR
translate, test and increment	TRTIB	B	IR
translate, test, increment and repeat	TRTIRB	B	IR
Ein-/Ausgabebefehle			
input	IN	B, W	IR, DA
input and decrement	IND	B, W	IR
input, decrement and repaet	INDR	B, W	IR
input and increment	INI	B, W	IR
input, increment and repeat	INIR	B, W	IR
output	OUT	B, W	IR, DA
output and decrement	OUTD	B, W	IR
output, decrement and repeat	OUTDR	B, W	IR
output and increment	OUTI	B, W	IR
output, increment and repeat	OUTIR	B, W	IR
CPU-Steuerbefehle			
complement flag	COMFLG	-	-
disable interrupt	DI	-	-
enable interrupt	EI	-	-
halt	HALT	-	-
multi-bit test	MBIT	-	-
multi-micro request	MREQ	-	-
multi-micro set	MSET	-	-
multi-micro reset	MRES	-	-
no operation	NOP	-	-
reset flag	RESFLG	-	-
set flag	SETFLG	-	-
load control register	LDCTL	-	-
load program status	LDPS	-	IR, DA, X
Rotations- und Verschiebebefehle			
rotate left	RL	B, W	R
rotate left trough carry	RLC	B, W	R
rotate left digit	RLDB	B	R
rotate right	RR	B, W	R
rotate right through carry	RRC	B, W	R
rotate right digit	RRDB	B	R
shift dynamic arithmetic	SDA	B, W, L	R
shift dynamic logical	SDL	B, W, L	R
shift left arithmetic	SLA	B, W, L	R
shift left logical	SLL	B, W, L	R
shift right arithmetic	SRA	B, W, L	R
shift right logical	SRL	B, W, L	R

ZEICHENERLÄUTERUNG

dst	Datenziel (destination)
src	Datenquelle (source)
disp	Disdanzadresse (displacement)
%	Hexadezimalkennzeichen
P	privilegierter Befehl
(n:m)	nur die Bits n bis m werden betrachtet
R(n)	Wortregister
R(n,n+1)	zwei Wortregister (n=0,2,4...14)
R(n,n+1...n+3)	vier folgende Wortregister (n=0,4,8,12)
msb	höchstwertiges Bit (most significate bit)
opco-Feld	Operationskode
w-Feld	Wortoperation (=1) Byteoperation (=0) (Long-Wortoperationen 1 oder 0)
mo-Feld	Adressierungsmode
r-Feld	Registerkodierung
x-Feld	Indexregisteradresse
num-Feld	Direktwert
Operandenlänge: B	Byteoperand (8 Bit)
W	Wortoperand (16 Bit) Standard
L	Long-Wortoperand (32 Bit)

FLAGS	Flagbits
FCW	Flag- und Controlwort
REFRESH	Refreshregister
PSAP	Programmstatusareapointer
PSAPSEG	Programmstatusareapointer Segment
PSAPOFF	Programmstatusareapointer Offset
NSP	Normalstackpointer
NSPSEG	Normalstackpointer Segment
NSPOFF	Normalstackpointer Offset
VI	vektorisierter Interrupt (v-Feld)
NVI	nichtvektorisier. Interrupt (n-Feld)
CTLR	Steuerregister (control)
Zyklen	Befehlsausführungszyklen (250 nsec)

Flagbitbeeinflussung:

.	unverändert
x	unbestimmt
?	gesetzt (0 oder 1)
0/1	auf Null/Eins gesetzt

Flagbitbedingungskode (cc):

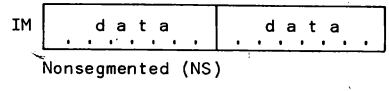
Z	zero (Null)	Z=1	0110
NZ	not zero (nicht Null)	Z=0	1110
C	carry (Übertrag)	C=1	0111
NC	not carry (kein Übertrag)	C=0	1111
PL	plus	S=0	1101
MI	minus	S=1	0101
NE	not equal (ungleich)	Z=0	1110
EQ	equal (gleich)	Z=1	0110
OV	overflow (Überlauf)	V=1	0100
NOV	no overflow (kein Überlauf)	V=0	1100
PE	parity even (Parität gerade)	P=1	0100
PO	parity odd (Parität ungerade)	P=0	1100
GE	greater than or equal (größer gleich)	S XOR V=0	1001
LT	less than (kleiner)	S XOR V=1	0001
GT	greater than (größer)	Z OR (S XOR V)=0	1010
UGE	unsigned greater than or equal	C=0	1111
ULT	unsigned (betragsmäßig) less than	C=1	0111
UGT	unsigned less than	(C=0) AND (Z=0)=1	1011
ULE	unsigned less or equal	C OR Z=1	0011

Registerkodierung:

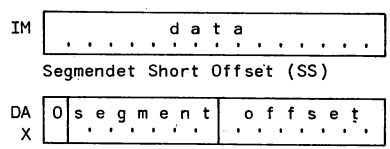
RQ0	RR0	R0	RH0	0000
		R1	RH1	0001
	RR2	R2	RH2	0010
		R3	RH3	0011
RQ4	RR4	R4	RH4	0100
		R5	RH5	0101
	RR6	R6	RH6	0110
		R7	RH7	0111
RQ8	RR8	R8	RL0	1000
		R9	RL1	1001
	RR10	R10	RL2	1010
		R11	RL3	1011
RQ12	RR12	R12	RL4	1100
		R13	RL5	1101
	RR14	R14	RL6	1110

Operandenformate

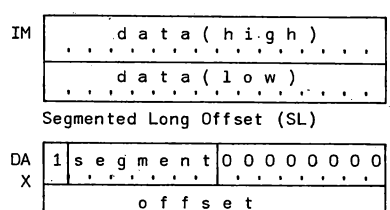
Bytebefehle:



Wortbefehle:



Long-Wortbefehle:



Befehlsformate:

F1.1

mo	opco	w	dst	*)

F1.2

mo	opco	w	dst	src

F1.3

mo	opco	dst	cc

F1.4

mo	opco	w	dst	cc

F2.1

mo	opco	w	src	dst

F2.2

mo	opco	w	dst	src

F2.3

mo	opco	src	*)

F3.1

opco	r	w	disp

F3.2

opco	dst	src

F3.3

opco	cc	disp

F3.4

opco	disp

F4.1

mo	opco	w	dst	src
0000	x	00000000		

F4.2

mo	opco	w	dst	src
displacement				

F4.3

mo	opco	w	src	dst
0000	x	00000000		

F4.4

mo	opco	w	src	dst
displacement				

F5.1

mo	opco	w	dst	*)
src				

F6.1

mo	opco	src	*)
0000	dst	0000	num

F6.2

mo	opco	dst	*)
0000	src	0000	num

F6.3

opco	w	*)	src
0000	dst	00000000	

F6.4

opco	w	src	*1)
0000	r	dst	*2)

F6.5

opco	w	src	*)
0000	r	dst	cc

F6.6

mo	opco	w	dst	src
0000	src	00000000		

F7.1

opco	w	dst	*)

F7.2

opco	w	src	*)

F7.3

opco	w	src	dst

F7.4

opco	w	dst	src

F8.1

opco	src	*)

F8.2

opco	dst	*)

F9.1

opco	CZSP	*)

F9.2

opco	*)	v	n

F9.3

opco	opco

F9.4

opco	0000	cc

F9.5

opco	src

*), *1), *2) spezielle Kennzeichenfelder
(src=0)* oder (dst=0)* src-Feld bzw. dst-Feld gleich Null

Lade-/Exchangebefehle

Mnemonik	Erläuterung	Format	Operanden	mo	opco	W/B-Zyklen: NS	SS	SL	L-Zyklen: NS	SS	SL	C Z S P D H
LD dst,src	dst:=src	F2.1	R,R	10	mo10 000w	3	-	-	5	-	-	
LDB			R,IR	00		7	-	-	11	-	-	
LDL			R,X	01	L: mo01 0100	10	10	13	13	13	16	
			R,DA (src=0)*	01		9	10	12	12	13	15	
			R,IM (src=0)*	00		7	-	-	11	-	-	
	nur LDB R,IM	F3.2	R,IM	00	1100	5	-	-				
		F4.4	R,BA	00	mo11 000w	14	-	-	17	-	-	
		F4.3	R,BX	01	L: mo11 0101	14	-	-	17	-	-	
		F2.2	IR,R	00	mo10 111w	8	-	-	11	-	-	
			DA,R (dst=0)*	01	L: mo01 1101	11	12	14	14	15	17	
			X,R	01		12	12	15	15	15	18	
		F5.1	IR,IM	00	mo00 110w	11	-	-	-	-	-	
			DA,IM (dst=0)*	01	*) 0101	14	15	17	-	-	-	
			X,IM	01		15	15	18	-	-	-	
		F4.2	BA,R	00	mo11 001w	14	-	-	17	-	-	
		F4.1	BX,R	01	L: mo11 0111	14	-	-	17	-	-	
LDA dst,src	dst:=Adresse src	F2.1	R,DA (src=0)*	01	mo11 0110	12	13	15				. .
			R,X	01		13	13	16				
		F4.4	R,BA	00	mo11 0100	15	-	-				
		F4.3	R,BX	01		15	-	-				
LDAR dst,src	dst:=Adresse src	F4.4	R,RA (src=0)*	00	mo11 0100	15	-	-				. . . . .
LDK	dst:=src (src 0...15)	F1.2	R,IM	10	mo11 110w	5	-	-				. . . . .
LDM dst,src,num	Register vom Speicher	F6.1	R,IR	00	mo01 1100	11+3*n	-	-				
	R,R+num-1 :=src		R,DA (src=0)*	01	*) 0001	14+3*n	15+3*n	17+3*n				
	(num 1...16)		R,X	01		15+3*n	15+3*n	18+3*n				
	Speicher vom Register	F6.2	IR,R	00	mo01 1100	11+3*n	-	-				
	dst:=R,R+num-1		DA,R (dst=0)*	01	*) 1001	14+3*n	15+3*n	17+3*n				
	(num 1...16)		X,R	01		15+3*n	15+3*n	18+3*n				
LDR dst,src	dst:=src	F4.4	R,RA (src=0)*	00	mo11 000w	14	-	-	17	-	-	. . . . .
LDRB					L: mo11 0101							
LDRL		F4.2	RA,R (src=0)*	00	mo11 001w	14			17			
					L: mo11 0111							
CLR dst	dst:=0	F1.1	R	10	mo00 110w	7	-	-				. . . . .
CLRB			IR	00	*) 1000	8	-	-				
			DA (dst=0)*	01		11	12	14				
			X	01		12	12	15				

EX dst,src EXB	dst:==:	F2.1	R,R R,IR R,DA (src=0)* R,X	10 00 01 01	mo10 110w	6 12 15 16	- 16 16	- 18 19				
POP dst,src POPL	dst:=src AUTOINC src(+2 bei Word +4 bei Long)	F2.1	R,IR IR,IR DA,IR (dst=0)* X,IR	10 00 01 01	mo01 011w L: mo01 0101	8 12 15 16	- - 16 16	- - 18 19	12 19 22 23	- - 23 23	- - 25 26	. . .
PUSH dst,src PUSHL	AUTODEC dst(-2 bei Word -4 bei Long) dst:=src	F2.2 F5.1	IR,R IR,IR IR,DA (src=0)* IR,X IR,IM	10 00 01 01 00	mo01 001w L: mo01 0001 mo00 110w *) 1001	9 13 13 14 12	- - 14 14 -	- - 16 17 -	12 20 20 21 -	- - 21 21 -	- - 23 24 -	.
Logikbefehle												
AND dst,src ANDB	dst:=dst AND src	F2.1	R,R R,IM (src=0)* R,IR R,DA (src=0)* R,X	10 00 00 01 01	mo00 011w	4 7 7 9 10	- - - 10 10	- - - 12 13				. ? ? . . . . ? ? ? . .
COM dst COMB	dst:=NOT dst	F1.1	R IR DA (dst=0)* X	10 00 01 01	mo00 110w *) 0000	7 12 15 16	- - 16 16	- - 18 19				. ? ? . . . . ? ? ? . .
OR dst,src ORB	dst:=dst OR src	F2.1	R,R R,IM (src=0)* R,IR R,DA (src=0)* R,X	10 00 00 01 01	mo00 010w	4 7 7 9 10	- - - 10 10	- - - 12 13				. ? ? . . . . ? ? ? . .
TEST dst TESTB TESTL	dst OR 0	F1.1	R IR DA (dst=0)* X	10 00 01 01	mo00 110w *) 0100 L: mo01 1100 *) 1000	7 8 11 12	- - 12 12	- - 14 15	13 13 16 17	- - 17 17	- - 19 20	. ? ? . . . . ? ? ? . . . ? ? x . .
TCC cc,dst TCCB	wenn cc TRUE dst(0):=1	F1.1	R	10	mo10 111w	5	-	-				. . . . .
XOR dst,src XORB	dst:=dst XOR src	F2.1	R,R R,IM (src=0)* R,IR R,DA (src=0)* R,X	10 00 00 01 01	mo00 100w	4 7 7 9 10	- - - 10 10	- - - 12 13				. ? ? . . . . ? ? ? . .

Bitbefehle

Mnemonik	Erläuterung	Format	Operanden	mo	opco	W/B-Zyklen: NS SS SL	L-Zyklen: NS SS SL	C Z S P D H
BIT dst,src BITB	Testen des durch src angegebenen Bit von dst - wenn Null Z:=1	F1.2	R,IM	10	mo10 011w	4 - -		. ? . .
			IR,IM	00		8 - -		
			DA,IM (dst=0)*	01		10 11 13		
			X,IM	01		11 11 14		
		F6.3	R,R		0010 011w	10 - -		
RES dst,src RESB	reset bit dst(src):=0	F1.2	R,IM	10	mo10 001w	4 - -		
			IR,IM	00		11 - -		
			DA,IM (dst=0)*	01		13 14 16		
			X,IM	01		14 14 17		
		F6.3	R,R		0010 001w *) 0000	10 - -		
SET dst,src SETB	set bit dst(src):=1	F1.2	R,IM	10	mo10 010w	4 - -		
			IR,IM	00		11 - -		
			DA,IM (dst=0)*	01		13 14 16		
			X,IM	01		14 14 17		
		F6.3	R,R		0010 010w *) 0000	10 - -		
TSET dst TSETB	test and set S-Flag:=dst(msb) dst(0:msb):=111...111	F1.1	R	10	mo00 110w	7 - -		? . . .
			IR	00	*) 0110	11 - -		
			DA (dst=0)*	01		14 15 17		
			X	01		15 15 18		
Arithmetikbefehle								
ADC dst,src ADCB	dst:=dst + src + c	F2.1	R,R	10	mo11 010w	5 - -		? ? ? ? . . ? ? ? ? 0 ?
ADD dst,src ADDB ADDL	dst:=dst + src	F2.1	R,R	10	mo00 000w	4 - -	8 - -	? ? ? ? . .
			R,IM (src=0)*	00	L: mo01 0110	7 - -	14 - -	? ? ? ? 0 ?
			R,IR	00		7 - -	14 - -	? ? ? ? . .
			R,DA (src=0)*	01		9 10 12	15 16 18	
			R,X	01		10 10 13	16 16 19	
CP dst,src CPB CPL	dst-src ?	F2.1	R,R	10	mo00 101w	4 - -	8 - -	? ? ? ? . .
			R,IM (src=0)*	00	L: mo01 0000	7 - -	14 - -	
			R,IR	00		7 - -	14 - -	
			R,DA (src=0)*	01		9 10 12	15 16 18	
			R,X	01		10 10 13	16 16 19	
		F5.1	IR,IM	00	mo00 110w	11 - -	- - -	
			DA,IM (dst=0)*	01	*) 0001	14 15 17	- - -	
			X,IM	01		15 15 18	- - -	

DAB dst	dst:=Dezimalkorrektur dst F1.1	R	10	B: mo11 000w *) 0000	5	-	-				? ? ? . . .
DEC dst,src DECB	dst:=dst - src F1.2 (src 1...16)	R,IM IR,IM DA,IM (dst=0)* X,IM	10 00 01 01	mo10 101w	4 11 13 14	- - 14 14	- - 16 17				. ? ? ? . .
EXTS dst EXTSB EXTSL	Vorzeichenbit des Low-Teils von dst wird in alle Bits des High-Teils kopiert F1.1	R	10	mo11 0001 W: *) 1010 B: *) 0000 L: *) 0111	11	-	-	11	-	-	.
DIV dst,src DIVL	R(n+1):=R(n,n+1) / src(W) R(n):=Rest F2.1 R(n+2,n+3):=R(n...n+3)/src(L) R(n,n+1):=Rest Zyklen 1. Zeile: Normalfall 2. Zeile: Divisor Null 3. Zeile: Absolutwert des High-Teils vom Dividenden ist größer als der Absolutwert des Divisors	R,R R,IM (src=0)* R,IR R,DA (src=0)* R,X	10 00 00 01 01	mo01 101w L: mo01 1010	107 13 25 107 13 25 107 13 25 108 14 26 109 15 27	- - - - - - - - - 108 15 27 109 15 27	- - - - - - - - - 111 17 28 112 18 29	744 30 51 744 30 51 744 30 51 745 31 52 746 32 53	- - - - - - - - - 746 32 53 746 32 53	- - - - - - - - - 748 34 55 749 35 56	? ? ? ? . . C:=1, falls Quotient kleiner oder größer (-)2 EXP 15 (-)2 EXP 31 Z:=1, falls Quotient oder Divisor Null P:=1, falls Divisor Null und Quotient zu groß
INC dst,src INCB	dst:=dst + src F1.2 (src 1...16)	R,IM IR IM DA,IM (dst=0)* X,IM	10 00 01 01	mo10 100w	4 11 13 14	- - 14 14	- - 16 17				. ? ? ? . .
MULT dst,src MULTL	R(n,n+1):=R(n+1) * src(W) R(n...n+3):=R(n+2,n+3) * src(L) F2.1 Zyklus 1. Zeile: n Anzahl der Bits, die in dst Null (Low-Byte) 2. Zeile: Ein Faktor ist Null	R,R R,IM R,IR R,DA (src=0)* R,X	10 00 00 01 01	W: mo01 1001 L: mo01 1000	70 18 70 18 70 18 71 19 72 20	- - - - - - 72 20 72 20	- - - - - - 74 22 75 22	282+7n 30 282+7n 30 282+7n 30 283+7n 31 284+7n 32	- - - - - - 284+7n 32 284+7n 32	- - - - - - 286+7n 34 287+7n 35	? ? ? 0 . . C:=1, falls Produkt kleiner od. größer (-)2 EXP 15 bzw. (-)2 EXP 31

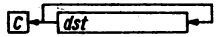
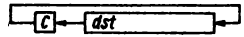
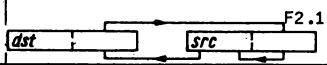
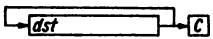
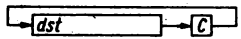
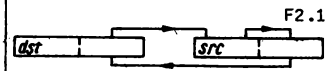
Mnemonik	Erläuterung	Format	Operanden	mo	opco	W/B-Zyklen: NS SS SL	L-Zyklen: NS SS SL	C Z S P D H
NEG dst NEGB	dst:= - dst	F1.1	R IR DA (dst=0)* X	10 00 01 01	mo00 110w *) 0010	7 - - 12 - - 15 16 18 16 16 19		? ? ? ? . . P:=1, falls Ergebnis %8000 (%80)
SBC dst,src SBCB	dst:=dst - src - c	F2.1	R,R	10	mo11 011w	5 - -		? ? ? ? . . ? ? ? ? 1 ?
SUB dst,src SUBB SUBL	dst:=dst - src	F2.1	R,R R,IM (src=0)* R,IR R,DA (src=0)* R,X	10 00 00 01 01	mo00 001w L: mo01 0010	4 - - 7 - - 7 - - 9 10 12 10 10 13	8 - - 14 - - 14 - - 15 16 18 16 16 19	? ? ? ? . . ? ? ? ? 1 ? ? ? ? ? . .
Programmsteuerbefehle								
CALL dst	PC:=dst Unterprogrammaufruf	F1.1	IR DA (dst=0)* X	00 01 01	mo01 1111 *) 0000	10 - 15 12 18 20 13 18 21		. .
CALR dst	PC:=PC-(2xdisp)	F3.4	RA (±2048)		1101	10 - 15		. . . . .
DJNZ r,dst DBJNZ	r:=r-1 falls r≠0 dann PC:=PC-(2xdisp)	F3.1	RA (0...127)		1111	11 - -		. . . . .
IRET P	return from interrupt	F9.3			0111 1011 *) 0000 0000	13 - 16		? ? ? ? ? ?
JP cc,dst	cc TRUE: PC:=dst	F1.3	IR DA (dst=0)* X	00 01 01	mo01 1110	Sprung 10 - 15 7 8 10 8 8 11	kein Sprung 7 - 7 7 8 10 8 8 11	. . . . .
JR cc,dst	cc TRUE: PC:=PC+(2xdisp)	F3.3	RA (±128)		1110	6 - -	6 - -	. . . . .
RET cc	cc TRUE: return	F9.4			1001 1110 *) 0000	10 - 13	7 - 7	. . . . .
SC src	Systemcall (src Request)	F9.5	IM		0111 1111	33 - 39		. . . . .
Blocktransferbefehle / Zeichenkettenbefehle								
CPD dst,src, r,cc CPDB	1. dst - src ? 2. src:=src-(B:-1; W:-2) 3. r:=r-1	F6.5	R,IR		1011 101w *) 1000	20 - -		x ? x ? . . Z:=1, falls cc TRUE

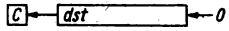
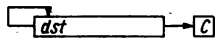
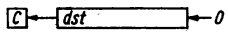
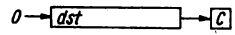
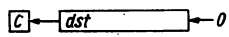
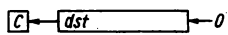
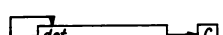
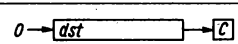
CPDR dst,src, CPDRB r,cc	wie CPD, zusätzlich: F6.5 repeat bis cc TRUE oder r=0	R,IR		1011 101w *) 1100	11+9*n - -	n Datenelementanzahl	x ? x ? . .
CPI dst,src, r,cc CPIB	1. dst - src ? F6.5 2. src:=src+(B:+1; W:+2) 3. r:=r-1	R,IR		1011 101w *) 0000	20 - -		x ? x ? . .
CPIR dst,src, CPIRB r,cc	wie CPI, zusätzlich: F6.5 repeat bis cc TRUE oder r=0	R,IR		1011 101w *) 0100	11+9*n - -	n Datenelementanzahl	x ? x ? . .
CPSD dst,src, r,cc CPSDB	1. dst - src ? F6.5 2. dst,src:=dst,src-(B:-1; 3. r:=r-1 W:-2)	IR,IR		1011 101w *) 1010	25 - -		x ? x ? . .
CPSDR dst,src, CPSDRB r,cc	wie CPSD, zusätzlich: F6.5 repeat bis cc TRUE oder r=0	IR,IR		1011 101w *) 1110	11+14*n - -	n Datenelementanzahl	x ? x ? . .
CPSI dst,src, r,cc CPSIB	1. dst - src ? F6.5 2. dst,src:dst,src+(B:+1; 3. r:=r-1 W:+2)	IR,IR		1011 101w *) 0010	25 - -		x ? x ? . .
CPSIR dst,src, CPSIRB r,cc	wie CPSI, zusätzlich: F6.5 repeat bis cc TRUE oder r=0	IR,IR		1011 101w *) 0110	11+14*n - -	n Datenelementanzahl	x ? x ? . .
LDD dst,src,r LDDB	1. dst:=src F6.4 2. dst,src:=dst,src-(B:-1; 3. r:=r-1 W:-2)	IR,IR		1011 101w *1) 1001 *2) 1000	20 - -		. x . ? . . P:=1, falls r=0
LDDR dst,src,r LDDRB	wie LDD, zusätzlich: F6.4 repeat bis r=0	IR,IR		1011 101w *1) 1001 *2) 0000	11+9*n - -	n Datenelementanzahl	. x . 1 . .
LDI dst,src,r LDIB	1. dst:=src F6.4 2. dst,src:=dst,src+(B:+1 3. r:=r-1 W:+2)	IR,IR		1011 101w *1) 0001 *2) 1000	20 - -		. x . ? .
LDIR dst,src,r LDIRB	wie LDI, zusätzlich: F6.4 repeat bis r=0	IR,IR		1011 101w *1) 0001 *2) 0000	11+9*n - -	n Datenelementanzahl	. x . 1 . .
TRDB dst,src,r	1. dst:=src(dst) F6.4 2. dst:=dst-1 3. r:=r-1 dst Adresse Index Translation	IR,IR		B: 1011 1000 *1) 1000 *2) 0000	25 - -		. x . ? . .
TRDRB dst,src,r	wie TRDB, zusätzlich: F6.4 repeat bis r=0	IR,IR		B: 1011 1000 *1) 1100 *2) 0000	11+14*n - -	n Datenelementanzahl	. x . ? . .

Mnemonik	Erläuterung Format	Operanden	mo	opco	W/B-Zyklen: NS SS SL	L-Zyklen: NS SS SL	C Z S P D H
TRIB dst,src,r	1. dst:=src(dst) F6.4 2. dst:=dst+1 3. r:=r-1 dst address index translation	IR,IR		B: 1011 1000 *1) 0000 *2) 0000	25 - -		. x . ? . .
TRIRB dst,src,r	wie TRIB, zusätzlich: F6.4 repeat bis r=0	IR,IR		B: 1011 1000 *1) 0100 *2) 0000	11+14*n - -	n Datenelementanzahl	. x . 1 .
TRTDB src1, src2,r	1. RH1:=src2(src1) F6.4 2. src1:=src1-1 3. r:=r-1 src1 address index translation	IR,IR		B: 1011 1000 *1) 1010 *2) 0000	25 - -		. ? . ? . . Z:=1, falls Wert Null
TRTDRB src1, src2,r	wie TRTDB, zusätzlich: F6.4 repeat bis RH1=0 oder r=0	IR,IR		B: 1011 1000 *1) 1110 *2) 1110	11+14*n - -	n Datenelementanzahl	. ? . ? . .
TRTIB src1, src2,r	1. RH1:=src2(src1) F6.4 2. src1:src1+1 3. r:=r-1 src1 address index translation	IR,IR		B: 1011 1000 *1) 0010 *2) 0000	25 - -		? ? . .
TRTIRB src1, src2,	wie TRTIB, zusätzlich: F6.4 repeat bis RH=0 oder r=0	IR,IR		B: 1011 1000 *1) 0110 *2) 1110	11+14*n - -	n Datenelementanzahl	. ? . ? .
Eingabe-/Ausgabebefehle							
IN dst,src INB P	dst:=src F7.3 F7.1	R,IR R,DA		0011 110w 0011 101w *) 0100	10 - - 12 - -		
IND dst,src,r INDB P	1. dst:=src F6.4 2. dst:=dst-(B:+1; W:-2) 3. r:=r-1	IR,IR		0011 101w *1) 1000 *2) 1000	21 - -		. x . ? . . P:=1, falls r=0
INDR dst,src,r INDRB P	wie IND, zusätzlich: F6.4 repeat bis r=0			0011 101w *1) 1000 *2) 0000	11+10*n - -	n Datenelementanzahl	. x . 1 . .
INI dst,src,r INIB P	1. dst:=src F6.4 2. dst:=dst+(B:+1; W:+2) 3. r:=r-1	IR,IR		0011 101w *1) 0000 *2) 1000	21 - -		. x . ? . .
INIR dst,src,r INIRB P	wie INI, zusätzlich: F6.4 repeat bis r=0	IR,IR		0011 101w *1) 0000 *2) 0000	11+10*n - -	n Datenelementanzahl	. x . 1

OUT dst,src OUTB P	dst:=src	F7.4 F7.2	IR,R DA,R		0011 111w 0011 101w *) 0110	10 12				
OUTD dst,src,r OUTDB P	1. dst:=src 2. src:=src-(B:-1; W:-2) 3. r:=r-1	F6.4	IR,IR		0011 101w *1) 1010 *2) 1000	21	-	-		. x . ? . .
OTDR dst,src,r OTDRB P	wie OUTD, zusätzlich: repeat bis r=0	F6.4	IR,IR		0011 101w *1) 1010 *2) 0000	11+10*n	-	-	n Datenelementanzahl	. x . 1 .
OUTI dst,src,r OUTIB P	1. dst:=src 2. src:=src+(B:+1; W:+2) 3. r:=r-1	F6.4	IR,IR		0011 101w *1) 0010 *2) 1000	21	-			. x . ? . .
OTIR dst,src,r OTIRB P	wie OUTI, zusätzlich: repeat bis r=0	F6.4	IR,IR		0011 101w *1) 0010 *2) 0000	11+10*n	-	-	n Datenelementanzahl	. x . 1 . .
SIN dst,src SINB P	dst:=src	F7.1	R,DA		0011 101w *) 0101	12				
SIND dst,src,r SINDB P	1. dst:=src 2. dst:=dst-(B:-1; W:-2) 3. r:=r-1	F6.4	IR,IR		0011 101w *1) 0001 *2) 1000	21	-			. x . ? . . P:=1, falls r=0
SINDR dst,src,r SINDRB	wie SIND, zusätzlich: repeat bis r=0	F6.4	IR,IR		0011 101w *1) 1001 *2) 0000	11+10*n		-	n Datenelementanzahl	. x . 1 . .
SINI dst,src,r SINIB P	1. dst:=src 2. dst:=dst+(B:+1; W:+2) 3. r:=r-1	F6.4	IR,IR		0011 101w *1) 1001 *2) 1000	21	-	-		. x . ? . .
SINIR dst,src,r SINIRB P	wie SINI, zusätzlich: repeat bis r=0	F6.4	IR,IR		0011 101w *1) 0001 *2) 0000	11+10*n	-	-	n Datenelementanzahl	. x . 1 . .
SOUT dst,src P SOUTB	dst:=src	F7.2	DA,R		0011 101w *) 0111	12	-	-		
SOUTD dst,src,r SOUTDB P	1. dst:=src 2. src:=src-(B:-1; W:-2) 3. r:=r-1	F6.4	IR,IR		0011 101w *1) 1011 *2) 1000	21	-	-		. x . ? . .
SOTDR dst,src,r SOTDRB P	wie SOUTD, zusätzlich: repeat bis r=0	F6.4	IR,IR		0011 101w *1) 1011 *2) 0000	11+10*n	-	-	n Datenelementanzahl	. x . 1 . .

Mnemonik	Erläuterung	Format	Operanden	mo	opco	W/B-Zyklen: NS SS SL	L-Zyklen: NS SS SL	C Z S P D H
SOUTI dst,src,r SOUTIB P	1. dst:=src 2. src:=src+(B:+1; W:+2) 3. r:=r-1	F6.4	IR,IR		0011 101w *1) 0011 *2) 1000	21 - -		. x . ? . .
SOTIR dsr,src,r SOTIRB P	wie SOUTI, zusätzlich: repeat bis r=0	F6.4	IR,IR		0011 101w *1) 0011 *2) 0000	11+10*n - -	n Datenelementanzahl	. x . 1 . .
CPU-Steuerbefehle								
COMFLG flag	Komplementierung der angegebenen Flags	F9.1	C,Z,S,P		1000 1101 *) 0101	7 - -		? ? ? ? . .
DI int P	Interruptsteuerbit Null setzen	F9.2	VI,NVI		0111 1100 0000 00vn	6 - -		. . . . .
EI int P	Interruptsteuerbit Eins setzen	F9.2	VI,NVI		0111 1100 0000 01vn	6 - -		. . . . .
HALT P	Befehlsabarbeitungshalt	F9.3			0111 1010 0000 0000	8+3*n - -	n Anzahl der Zyklen	. . . . .
MBIT P	test multi-micro input pin mI (S:=NOT mI)	F9.3			0111 1011 0000 1010	7 - -		. ? x . . .
MREQ dst P	multi-micro request dst:=dst-1 repeat bis dst=0	F8.2			0111 1011 *) 1101	12 - - 12+7*n	n Anzahl dst:=dst-1	. ? ? . . .
MRES P	m0:=0 (m0: multi-micro output pin)	F9.3			0111 1011 0000 1001	5 - -		. .
MSET P	m0:=1	F9.3			0111 1011 0000 1000	5 - -		. . .
NOP	no operation	F9.3			1000 1101 0000 0111	7 - -		. .
RESFLG flag	reset flag	F9.1	C,Z,S,P		1000 1101 *) 0011	7 - -		? ? ? ? . .
SETFLG flag	set flag	F9.1	C,Z,S,P		1000 1101 *) 0001	7 - -		? ? ? ? . .

LDCTL CTLR,src LDCTLB	FLAGS(2:7):=src(2:7)	F8.1	FLAGS,R		B: 1000 1100 *) 1001	7				? ? ? ? ? ?
P	FCW(2:7):=src(2:7) FCW(11:15):=src(11:15)	F8.1	FCW,R		W: 0111 1101 *) 1010	7				? ? ? ? ? ?
	REFRESH(1:15):=src(1:15)	F8.1	REFRESH,R		W: *) 1011	7	-	-		
	PSAPSEG(8:14):=src(8:14)	F8.1	PSAPSEG,R		W: *) 1100	7	-	-		
	PSAPOFF(8:15):=src(8:15)	F8.1	PSAPOFF,R		W: *) 1101	7	-	-		
	NSPSEG:=src	F8.1	NSPSEG,R		W: *) 1110	7	-	-		
	NSPOFF:=src	F8.1	NSPOFF,R		W: *) 1111	7	-	-		
LDCTL dst,CTLR LDCTLB	dst(2:7):=FLAGS(2:7)	F8.2	R,FLAGS		B: 1000 1100 *) 0001	7	-	-		
P	dst(2:7):=FCW(2:7) dst(11:15):=FCW(11:15)	F8.2	R,FCW		W: 0111 1101 *) 0010	7				. . . . .
	dst(1:8):=REFRESH(1:8)	F8.2	R,REFRESH		W: *) 0011	7	-	-		
	dst(8:14):=PSAPSEG(8:14)	F8.2	R,PSAPSEG		W: *) 0100	7	-	-		
	dst(8:15):=PSAPOFF(8:15)	F8.2	R,PSAPOFF		W: *) 0101	7	-	-		
	dst:=NPSEG	F8.2	R,NSPSEG		W: *) 0110	7	-	-		
	dst:=NSPOFF	F8.2	R,NSPOFF		W: *) 0111	7	-	-		
LDPS src	PS:=src (FCW und PC)	F2.3	IR DA X	00 01 01	mo11 1001 *) 0000	12 16 17	- 20 20	16 22 23		? ? ? ? ? ?
P										
Rotations-/Verschiebebefehle										
RL dst,src RLB		F1.1	R,IM	10	mo11 001w *) 00s0	src=1 (s=0): 6 - - (eine Operation)		src=2 (s=1): 7 - - (zwei Operationen)		? ? ? ? . .
RLC dst,src RLCB		F1.1	R,IM	10	mo11 001w *) 10s0	6 - -		7 - -		? ? ? ? . .
RLDB dst,src		F2.1	R,R	10	mo11 1110	9 - -				. ? ? .
RR dst,src RRB		F1.1	R,IM	10	mo11 001w *) 01s0	6 - -		7 - -		? ?
RRC dst,src		F1.1	R,IM		mo11 001w *) 11s0	6 - -		7 - -		? ? ? ? . .
RRDB dst,src		F2.1	R,R		mo11 1100	9 - -				. ? ? . . .

Mnemonik	Erläuterung	Format	Operanden	mo	opco	W/B-Zyklen: NS SS SL	L-Zyklen: NS SS SL	C Z S P D H
SLA dst,src SLAB SLAL		F5.1	R,IM	10	mo11 001w *) 1001 L: mo11 0011 *) 1101	13+3*n - - src(=n): Anzahl der Bitpositionen	13+3*n	
SDA dst, SDAB SDAL	negativer src-Wert:  positiver src-Wert: 	F6.6	R,R	10	mo11 001w *) 1011 L: mo11 0011 *) 1111	15+3*n	15+3*n	
SDL dst,src SDLB SDLL	negativer src-Wert:  positiver src-Wert: 	F6.6	R,R	10	mo11 001w *) 0011 L: mo11 0011 *) 0111	15+3*n	15+3*n	? ? ? x . .
SLL dst src SLLB SLLL		F5.1	R,IM	10	mo11 001w *) 0001 L: mo11 0011 *) 0101	13+3*n	13+3*n	? ? ? X . .
SRA dst,src SRAB SRAL		F5.1	R,IM	10	mo11 001w *) 1001 L: mo11 0011 *) 1101	13+3*n	13+3*n	? ? ? 0 . .
SRL dst,src SRLB SRLL		F5.1	R,IM	10	mo11 001w *) 0001 L: mo11 0011 *) 0101	13+3*n -	13+3*n	? ? ? x . .

EPU-Extended-Befehle							
	load memory from EPU dst:=EPU (n Wörter)	 IR X DA (dst=0)*	 00 00 01 01	 mo001111 dst11.. ..... n-1	 11+3*n 15+3*n 15+3*n 18+3*n 14+3*n 15+3*n 17+3*n		
	load EPU from memory EPU:=src (n Wörter)	IR X DA (src=0)*	00 01 01	mo001111 src01.. ..... n-1	11+3*n 15+3*n 15+3*n 18+3*n 14+3*n 15+3*n 17+3*n		
	load CPU register from EPU dst:=EPU (n Wörter)			100011110...00.. .... dst.... n-1	11+3*n		
	load EPU from CPU register EPU:=src (n Wörter)			100011110...10.. .... src.... n-1	11+3*n		
	load FCW from EPU Flags:=EPU(AD0 - AD7)			10001110....00.. ....0000....0000	14		? ? ? ? ? ?
	load EPU from FCW EPU:=Flags			10001110....10.. ....0000....0000	14		
	NOP Initialisieren EPU			10001110....01..	14		

3.2. Speicherverwaltungsbaustein U 8010-MMU (Memory Management Unit)

Der Speicherverwaltungsbaustein U8010-MMU dient der effektiven Verwaltung eines bis 4 MByte großen Adreßraumteils des segmentierten, insgesamt 8 MByte umfassenden Speicherraumes des 16-Bit-Mikroprozessors U8001-CPU (128 Segmente von 64 KByte - resultierend aus 7 Segmentleitungen und 16 Adreßleitungen). Er übernimmt in einem U8001-Mikrorechnersystem die physische Zuordnung der einzelnen logischen Adreßsegmente in einer realen Speicherkonfiguration. Diese Zuordnung kann durch die Programmiermöglichkeiten der U8010-MMU dynamisch verschieblich (relocatable) gestaltet werden. Programme und Daten sind mit dem Einsatz des MMU-Schaltkreises nicht mehr fest an eine bestimmte Adreßzuordnung im Gesamtspeicherbereich gebunden (gelinkt) sondern können dynamisch, in Abhängigkeit von der momentanen Speicherauslastung, in freien Segmenten abgelegt und abgearbeitet werden. Durch den Einsatz mehrerer MMU-Schaltkreise in einem U8001-System können außerdem unterschiedliche Speicherzuordnungstabellen für unterschiedliche Programme und Datenstukturen angelegt werden (z.B.: Programme - Daten - Stack - Normalmode - Systemmode). Neben dem Übersetzen logischer Adressen in physische Adressen kann die U8010-MMU eine Reihe von Speicherschutzfunktionen (protection violation) vor unautorisiertem Zugriff übernehmen. Das kleinste von einer U8010-MMU behandelbare Speichersegment ist 256 Byte, das größte 64 KByte groß /20/,/21/,/23/,/24/,/25/.

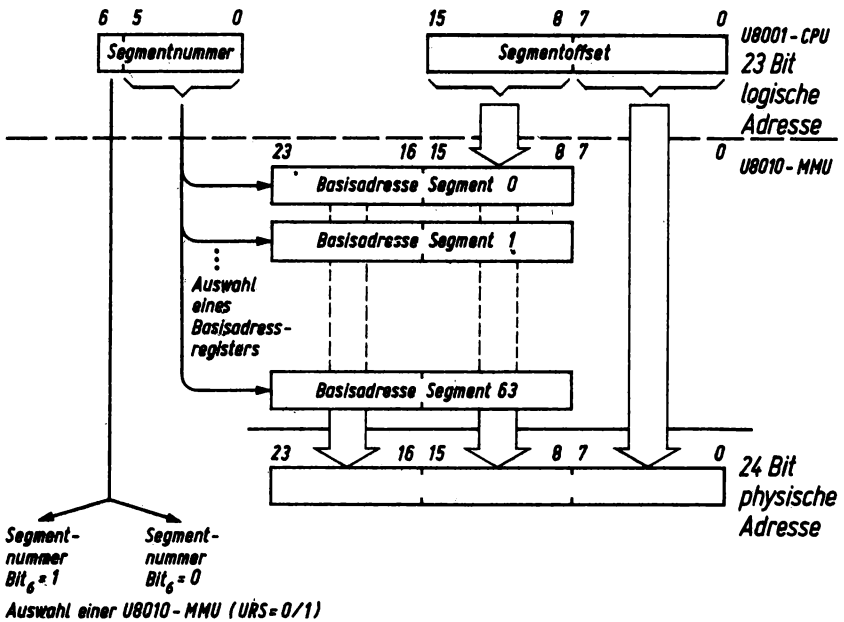


Bild 3.6. Generierung der physischen Speicheradresse durch die U8010-MMU aus der logischen U8001-CPU Adresse

3.2.1. Aufbau

Ein U8010-MMU-Schaltkreis ermöglicht die dynamische Speicherverwaltung von maximal 64 Speichersegmenten. Deren Anfangsadresse (Basisadresse), deren Größe und deren Speicherschutzeigenschaften werden durch den Inhalt von 64 Segmentbeschreibungsregistern (Segmentdeskriptorregister) bestimmt. Die Programmierung dieser Register erfolgt mit Hilfe eines Steuerregistersatzes (control register), die von der U8001-CPU über die speziellen Ein-/Ausgabebefehle erreicht werden können (Low-Byte der 16-Bit-Ein-/Ausgabeadresse, die Adresse der U8010-MMU - High-Byte die Spezifizierung des Kommandos).

Tafel 3.2. Aufbau des U8010-MMU Segmentdeskriptorregisterfeldes (SDR)

Basisadreßfeld Limitfeld Attributfeld

15 8 7 0 7 0 7 0

BAH0	BAL0	L0	A0	SDR0
BAH1	BAL1	L1	A1	SDR1
.	.	.	.	.
BAH63	BAL63	L63	A63	SDR63

Attributfeld

REF	CHG	DIRW	DMAI	EXC	CPUI	SYS	RD

RD (read only) - RD-Flag gesetzt: Das Segment kann nur gelesen werden; die Daten sind vor jedem Schreibversuch geschützt.
SYS (system only) - SYS-Flag gesetzt: Auf das Segment kann nur im Systemmode zugegriffen werden; ein Zugriff im Normalmode wird unterdrückt.
CPUI (CPU inhibit) - CPUI-Flag gesetzt: Die CPU kann nicht auf das Segment zugreifen; das Segment ist nur im DMA-Betrieb erreichbar.
EXC (execute only) - EXC-Flag gesetzt: Auf das Segment kann nur im Befehlslesezyklus (intruction fetch) zugegriffen werden (Programmkode).
DMAI (DMA inhibit) - DMAI-Flag gesetzt: Ein Zugriff im DMA-Betrieb wird unterdrückt; das Segment ist nur von der CPU erreichbar.
DIRW (direction and warning) - DIRW-Flag gesetzt: Das Segment kann nur in fallender Adreßrichtung beschrieben werden (Stacksegment). Jedes Schreiben in dieses Segment ist mit einer Prüfung verbunden, ob auf den letzten verfügbaren 256-Byteblock zugegriffen wird - wenn ja, so wird ein Segmenttrap zur Warnung (Stacküberlauf) generiert.
CHG (changed) - Das von der U8010-MMU gesetzte CHG-Flag signalisiert, daß das Segment beschrieben worden ist. Das Flag wird automatisch bei jedem ordnungsgemäßen Schreibzugriff gesetzt.
REF (referenced) - Das von der U8010-MMU gesetzte REF-Flag signalisiert, daß auf das Segment zugegriffen wurde.

Die Zuordnung der programmierlogischen Speicheradressen (Segmentnummer+Offset) zu den wahren (physischen) Speicheradressen erfolgt durch Addition der oberen acht Bit des von der CPU erzeugten Adreßoffset zu einer 16-Bit-Basisadresse. Die Basisadresse wird dabei aus dem Segmentdeskriptorregister (SDR) geliefert, das der von der U8001-CPU an die

U8010-MMU gelieferten Segmentnummer fest zugeordnet ist. Das Low-Byte des Offset wird von der MMU nicht beeinflußt. Die physische Adresse ist dann eine 24-Bit-Speicheradresse, die den Zugriff zu einem 8 MByte Speicherraum gestattet.

Tafel 3.3. Aufbau der U8010-MMU Steuerregister (CTL)

MSEN	TRNS	URS	MST	NMS	ID2	ID1	ID0	
								Moderegister
0	0	Segmentdeskriptornummer						SAR
0	0	0	0	0	0	DSC1	DSC0	DSC

ID (identification) - Wird bei einem MMU-Segmenttrap benutzt, um die Adreßleitung AD8+ID (3 Bit) als Identifier auf 'hign' zu setzen.

NMS (normal mode select) - Das NMS-Flag zeigt im Zusammenhang mit dem MST-Flag an, ob der N/S-Signalzustand 'high' (NMS=1) oder 'low' (NMS=0) zum Übersetzen der von der U8001-CPU gelieferten Adresse in einer Mehrfach-MMU-Konfiguration führt (anderenfalls sind die MMU-Adreßleitungen im hochohmigen Zustand).

MST (multi segment table) - MST-Flag gesetzt: Es sind mehrere MMU-Schaltkreise im Mikrorechner vorhanden; das N/S-Signal bestimmt, welche U8010-MMU die beim Zugriff gültige Übersetzungstabelle enthält.

URS (upper range select) - Das URS-Flag zeigt an, ob die MMU die Segmentdeskriptorregister für die Segmentnummern 0-63 oder für die Segmentnummern 64-127 enthält. Das höchstwertige Bit der Segmentnummer muß immer dem URS-Flag entsprechen (anderenfalls sind die MMU-Adreßleitungen hochohmig).

TRNS (translate) - Das TRNS-Flag zeigt an, ob die U8010-MMU die logische Adressen in physische Adressen übersetzen soll (TRNS=1) oder ob sie die MMU unverändert und ungeprüft passieren sollen (TRNS=0).

MSEN (master enable) - MSEN-Flag gesetzt: Die U8010-MMU kann aktiviert werden; d.h. sie kann, wenn entsprechend programmiert, die Adreßübersetzung und Adreßprüfung übernehmen.
MSEN-Flag rückgesetzt: Die MMU-Adreßleitungen sind hochohmig.
(MSEN=1 und TRNS=1 ist die Voraussetzung, daß eine Adreßübersetzung und eine Adreßüberprüfung stattfindet.)

SAR (Segmentadreßregister) - Das SAR-Register zeigt bei der MMU-Programmierung als Pointer auf eines der 64 Segmentdeskriptorregister der U8010-MMU. Jeder Zugriff auf ein Segmentdiskriptorregister (SDR) erfolgt über diesen Zeiger (automatische Erhöhung nach jedem Zugriff - Autoinkrement).

DSC (Deskriptorselektionszähler) - 2-Bit-Zähler, dessen Inhalt bestimmt, zu welchem Byte des angewählten SDR zugegriffen wird:
= 0 auf die oberen 8 Bit des Basisadreßfeldes,
= 1 auf die unteren 8 Bit des Basisadreßfeldes,
= 2 auf das Limitfeld,
= 3 auf das Attributfeld.

3.2.2. Programmierung

Der Speicherverwaltungsbaustein U8010-MMU enthält drei Registergruppen, die Segmentdeskriptorregister (SDR), die Steuerregister (CTL) und die Statusregister (STR).

Der Satz der 64 Segmentdiscriptorregister enthält die Informationen, die für die Wandlung der programmierlogischen Speicheradressen in die physischen Speicheradressen notwendig sind. Die Segmentnummer einer logischen Adresse bestimmt dabei, welches Segmentdeskriptorregister für die Übersetzung der logischen Adresse benutzt wird. Jedes SDR-Register enthält außerdem die notwendigen Informationen zur Prüfung, ob ein Speicherzugriff innerhalb der zulässigen Segmentgrenzen erfolgt und ob die Zugriffsart gestattet ist. Es signalisiert nach dem Zugriff, ob ein Segment beschrieben oder gelesen wurde.
Zusätzlich zu den 64 SDR-Registern existieren in jeder U8010-MMU drei 8-Bit-Steuerregister (CTL) für die Initialisierung des Bausteins und sechs 8-Bit-Statusregister (STR), die Informationen für Maßnahmen im Falle einer Zugriffsverletzung enthalten.
Die Programmierung der U8010-MMU erfolgt über Kommandos, die bei der MMU-Eingabe oder bei der MMU-Ausgabe im oberen Adreßbyte der für den Verkehr mit dem MMU-Baustein vorgesehenen speziellen U8001-Ein-/Ausgabebefehle kodiert werden. Das untere Adreßbyte dieser Befehle wird zur eigentlichen Adressierung der U8010-MMU benutzt. Dabei ist zu beachten, daß nur gerade MMU-Adressen zulässig sind und die Lese- und Schreibdaten auf den oberen acht Datenbits des Ausgabewortes (AD15-AD8) ausgetauscht werden (Tafel 3.5.).

Segmentdeskriptorregister (SDR) - Jedes der 64 SDR enthält ein 16-Bit-Basisadreßfeld, ein 8-Bit-Limitfeld und ein 8-Bit-Attributfeld. Die Basisadresse - in High-Teil und Low-Teil untergliedert - ist die jeweilige physische Segmentanfangsadresse. Das Limitfeld enthält als Vielfaches von 256 kodiert die Segmentblocklänge. Das Attributfeld enthält acht Flags. Fünf werden zur Kennzeichnung erlaubter Zugriffsarten vom Programmierer benutzt. Ein Flag - ebenfalls vom Programmierer zu setzen - kennzeichnet eine spezielle Segmentstruktur und zwei Flags - von der U8010-MMU als Rückantwort an den Programmierer vorgesehen - kennzeichnen die Zugriffsart, in der das Segment zuletzt angesprochen wurde (Tafel 3.2.).

Steuerregister - Die drei 8-Bit-Steuerregister (Moderegister, Segmentadressenregister, Deskriptorselektionszähler) steuern im einzelnen die Arbeitsweise der U8010-MMU.
Das Moderegister spezifiziert die Aktivierungsbedingungen einer U8010-MMU - auch in Mehrfach-MMU-Konfigurationen.
Das Segmentadressenregister (SAR) wählt ein bestimmtes Segmentdeskriptorregister, der Deskriptorselektionszähler (DSC) ein bestimmtes Byte innerhalb des angewählten Segmentdeskriptorregisters bei der MMU-Programmierung aus (Tafel 3.3.).

Statusregister - Die sechs 8-Bit-Statusregister (Violationsegmentnummer, Violationoffset, Buszyklusstatus, Befehlssegmentnummer, Befehlsoffset, Violationtyp) enthalten Informationen für den Fall einer Speicherzugriffsverletzung (violation) die zu einem U8010-MMU Segmenttrap geführt haben.
Das Violationtypregister beschreibt die genauen Umstände, der MMU-Trapgenerierung. Die Violationsegmentnummer und der Violationoffset zeichnen die Segmentnummer und die oberen acht Bit des Segmentoffset der programmierlogischen Adresse auf, die den Trap verursachte. Die Befehlssegmentnummer und der Befehlsoffset kennzeichnen die Segmentnummer und die oberen acht Bit des Segmentoffset des ersten Befehlswortes der letzten ausgeführten Instruktion vor dem Segmenttrap. Das Buszyklusstatusregister

schließlich, zeichnet den aktuellen Busstatus auf (CPU-Statuskode, Read/Write, Normal-/Systemmode), der zum Zeitpunkt der Adreßverletzung vorhanden war (Tafel 3.4.).

Tafel 3.4. Aufbau der U8010-MMU Statusregister (STR)

Violationsegment	0	Segmentnummer						
Violationoffset	Upper Offset							
Befehlszyklus	0	0	N/S	R/W	CPU-Status			
Befehlssegment	0	Segmentnummer						
Befehlsoffset	Upper Offset							
Violationtyp	FATL	SWW	PWW	EXCV	CPUIV	SLV	SYSV	RDV

RDV (read only violation) - Das RDV-Flag wird bei einem Zugriff auf ein schreibgeschütztes Segment gesetzt.
SYSV (system violation) - Das SYSV-Flag wird bei einem Zugriff im Normalmode auf ein für den Systemmode reserviertes Segment gesetzt.
SLV (segment length violation) - Das SLV-Flag wird gesetzt, wenn sich der Offset der logischen Adresse außerhalb der Segmentgrenzen bewegt.
CPUIV (CPU inhibit violation) - Das CPUIV-Flag wird bei einem Zugriff auf ein Segment gesetzt, dessen CPUI-Flag gesetzt (=1) ist.
EXCV (execute only violation) - Das EXCV-Flag wird bei einem Nicht-Befehlslesezugriff auf ein Segment gesetzt, dessen EXC-Flag gesetzt (=1) ist.
PWW (primary write warning) - Das PWW-Flag wird bei einem Schreibzugriff auf die ersten 256 Byte eines Segmentes gesetzt, dessen DIRW-Flag gesetzt (=1) ist (Warnung).
SSW (secondary write warning) - Das SSW-Flag wird bei einem Schreibzugriff auf die ersten 256 Byte eines Segmentes gesetzt, dessen DIRW-Flag und eines der Flags EXCV, CPUIV, SLV, SYSV, RDV oder PWW gesetzt (=1) ist. Ist das SSW-Flag gesetzt, dann wird bei weiteren Warnbedingungen kein Trap mehr ausgelöst.
FATL (fatal condition) - Das FATL-Flag wird gesetzt, wenn irgendein Flag des Violationregisters gesetzt ist und eine Adreßzugriffsverletzung oder Warnbedingung im Normalmode auftrat. Ist das FATL-Flag gesetzt, werden weitere Zugriffsverletzungen so lange nicht mehr beachtet, wie das Flag nicht über ein Kommando gelöscht wurde

Tafel 3.5. U8010-MMU Kommandos

Lesen/Scheiben des Segmentdeskriptorregisterfeldes (SDR)			
Kommando	08	Bedeutung	Lesen/Schreiben Basisadressenfeld
	09		Lesen/Schreiben Limitfeld
	0A		Lesen/Schreiben Attributfeld
	0B		Lesen/Schreiben Segmentdeskriptorregister (4 Byte mit Autoinkrement DSC: 0,1,2,3)
	0C		Lesen/Schreiben Basisadr.-feld - Inkrement SAR
	0D		Lesen/Schreiben Limitfeld - Inkrement SAR
	0E		Lesen/Schreiben Attributfeld - Inkrement SAR
	0F		Lesen/Schreiben Segmentdeskriptorregister - Autoinkrement SAR (4 Byte mit Autoinkrement DSC: 0,1,2,3)
Lesen/Schreiben der Steuerregister			
Kommando	00	Bedeutung	Lesen/Schreiben Moderegister
	01		Lesen/Schreiben Segmentadreßregister (SAR)
	20		Lesen/Schreiben Descriptorselektionszähler (DSC)
Lesen/Schreiben der Statusregister			
Kommando	02	Bedeutung	Lesen Violationtyperegister
	03		Lesen Violationsegmentnummer
	04		Lesen Violationoffset (High-Byte)
	05		Lesen Befehlszyklus
	06		Lesen Befehlssegmentnummer
	07		Lesen Befehlsoffset (High-Byte)
Setzen (1) / Rücksetzen (0)			
Kommando	15	Bedeutung	Setzen aller 64 CPUI-Flags im SDR-Feld
	16		Setzen aller 64 DMAI-Flags im SDR-Feld
	11		Rücksetzen des Violationtyperegisters
	13		Rücksetzen des SWW-Flag Violationtyperegister
	14		Rücksetzen des FATL-Flag Violationtyperegister

3.3. Zähler-/Zeitgeber- und paralleler Ein-/Ausgabebaustein 8 × 36-CIO

Die beiden am meisten eingesetzten Peripheriebausteine des 8-Bit-Mikroprozessorsystems U880 sind der U855-PIO für parallele Ein-/Ausgabeaktivitäten und der U857-CTC für Zähler-/Zeitgeberaufgaben (Abschn. 2.2. und 2.4.). Parallele Ein-/Ausgabedatenströme, Zählfunktionen und Zeitnormale werden wohl in nahezu jedem Standardmikrorechner Verwendung finden.
Im 16-Bit-Mikrorechnersystem U8000 stehen für parallele Ein-/Ausgabedatenströme und für Zähler-/Zeitgeberaufgaben der - beide Grundfunktionen (PIO und CTC) in einem hochintegrierten Bauelement vereinigende - U8036-CIO bzw. U82536-CIO (abgekürzt U8x36) zur Verfügung (CIO counter timer and input/output unit). Er kann für jede Art von Ein-/Ausgabeaktivitäten bei parallelen Datenströmen, für Zählaufgaben, für Zeitgeberaktivitäten, aber auch, davon abgeleitet, als Interruptsteuerschaltkreis (interrupt controler), zur Zeichenerkennung (pattern recognition) u.a.m. eingesetzt werden. Eine große Anzahl programmierbarer Optionen erlaubt die flexible Anpassung des U8x36-CIO an den jeweiligen Einsatzfall.
Insbesondere existieren beim U8x36-CIO:

- zwei voll programmierbare bidirektionale 8-Bit-Ein-/Ausgabekanäle, die, doppelt gepuffert, unabhängig oder auch verkettet (16-Bit-Port) arbeiten,

- ein bidirektionaler 4-Bit-Ein-/Ausgabekanal, dessen Leitungen zum Datentransfer oder als Handshake- bzw. Request/Wait-Signalleitungen für einen oder für beide 8-Bit-Ein-/Ausgabekanäle verwendet werden,

- eine Zeichenerkennungslogik, die, entsprechend programmiert, zur Nutzung des U8x36-CIO als Interruptcontroller verwendet werden kann,

- drei voll programmierbare, unabhängig oder verkettet betreibbare 16-Bit-Zähler-/Zeitgeberkanäle mit jeweils bis zu vier verschiedenen externen Zugriffsvarianten (count input, output, gate, trigger).

Die Programmierung des U8x36-CIO erfolgt über insgesamt 48 adressierbare Register, die eine kaum überschaubare Variationsbreite beim Einsatz bieten.

Wegen des begrenzt zur Verfügung stehenden Platzes ist eine erschöpfende Behandlung des U8x36-CIO an dieser Stelle nicht möglich. Der Leser wird in den nachfolgenden Abschnitten zum U8x36-CIO aber alle für die Programmierung notwendigen Informationsübersichten in gedrängter Form wiederfinden. In speziellen Einsatzfällen sind notwendigenfalls experimentelle Untersuchungen erforderlich. Außerdem sei auf die Datenbücher /52/, /53/ verwiesen.

Zu beachten ist, daß zwei Versionen des CIO-Schaltkreises existieren, die sich ausschließlich in der Busanpassung unterscheiden. Während der U8036-CIO einen gemultiplexten Adress-/Datenbus und zwei Chipselektsignale besitzt, kennt der U82536-CIO nur einen Datenbus und zwei Adreßauswahlsignale. Die Unterschiede der Busanpassung zwischen beiden Schaltkreisen finden hauptsächlich bei der Programmierung ihren Niederschlag.

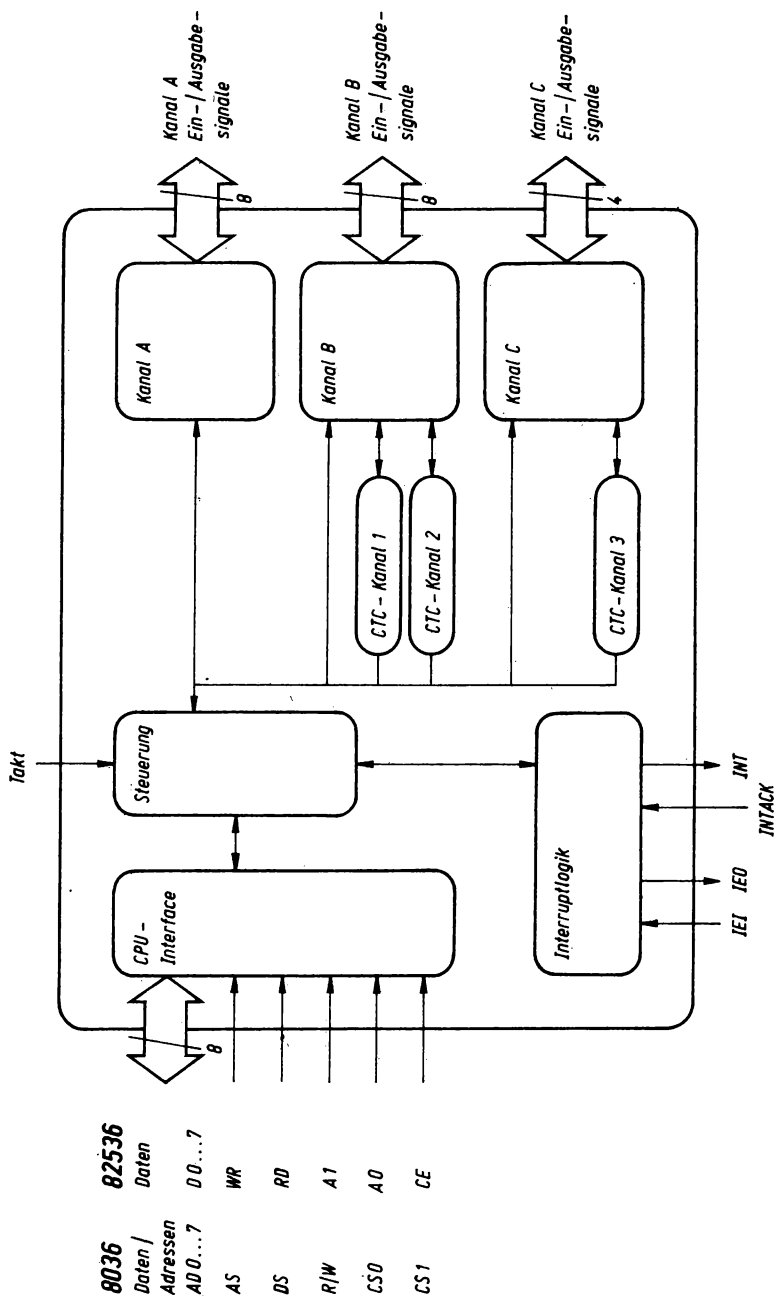


Bild 3.7. Aufbau und Signalstruktur des Bausteins für Zähler-/Zeitgeber-aufgaben und für parallele Ein-/Ausgaben U8x36-CIO

Anwendungsbezogen unterscheiden sich beide Schaltkreise bezogen auf den Einsatz in unterschiedlichen Mikroprozessor-Zielsystemen. Der U8036-CIO ist vorwiegend oder ausschließlich für den Einsatz in U8000-Systemen vorgesehen. Die bei dieser CIO-Variante vorhandene Busanpassung ist optimal auf diese Mikroprozessorfamilie zugeschnitten. Der U82536-CIO besitzt dagegen ein mehr universelles Mikrorechnerinterface, das ihn für den Einsatz in nahezu jedem modernen 8- oder 16-Bit-MP-Schaltkreissystem prädestiniert.

3.3.1. Aufbau

Der U8x36-CIO besteht aus zwei allgemein verwendbaren 8-Bit-Ein-/Ausgabeports, aus einem speziell verwendbaren 4-Bit-Ein-/Ausgabeport, aus drei 16-Bit-Zähler-/Zeitgeberports, aus einer Interruptsteuerlogik, einem Mikrorechnerinterface und aus der CIO-Schaltkreissteuerung (Bild 3.7.).
Die beiden 8-Bit-Ein-/Ausgabeports A und B des U8x36-CIO sind, bis auf die Tatsache, daß der Port B, entsprechend programmiert, dem externen Zugriff auf die CTC-Kanäle 1 und 2 dienen kann, identisch.
Die Handshake-Steuerung für Kanal A oder/und B erfolgt grundsätzlich über den 4-Bit-Ein-/Ausgabekanal C, über den auch ein - gegebenenfalls notwendiger - externer Zugriff des CTC-Kanals 3 abgewickelt wird.
Sowohl der Ein-/Ausgabekanal A als auch B sind mit einer Interrupt-Zeichenerkennungslogik ausgerüstet, die - wenn notwendig - die Eingabe eines vorher spezifizierten Bitmusters signalisiert. Außerdem ist es möglich, aus den beiden 8-Bit-Kanälen A und B einen 16-Bit-Kanal mit einem einzigen Handshake-Signalspiel zu formieren.
Die Funktion des 4-Bit-Ein-/Ausgabeports C hängt stark von der speziellen Programmierung der 8-Bit-Kanäle A und B sowie von der speziellen Nutzung der CTC-Kanäle ab. Ein notwendiges Handshake-Signalspiel, Request/Wait-Steuersignale zur Synchronisierung der Arbeit des U8x36-CIO mit CPU- oder DMA-Bausteinen und die externen Zugriffe des CTC-Kanals 3 laufen über diesen Port.
Den beiden 8-Bit-Ein-/Ausgabekanälen A und B sind jeweils 12, dem 4-Bit-Ein-/Ausgabekanal insgesamt 3 spezielle Steuer- und Statusregister zugeordnet, die die Programmierung und Steuerung der Ein-/Ausgabeports ermöglichen.
Die drei CTC-Kanäle des U8x36 sind grundsätzlich identisch. Jeder besteht aus einem 16-Bit-Zeitkonstantenregister, aus einem 16-Bit-Zählregister, aus einem 16-Bit-Momentanwertregister und aus einem 8-Bit-Steuer-/Statusregister.
Die Funktionsblöcke Interruptsteuerlogik, Mikrorechnerinterface und Schaltkreissteuerung des U8x36-CIO entsprechen in Aufbau und Funktion ähnlichen Funktionsblöcken bekannter Peripherieschaltkreise für Mikrorechner.

3.3.2. Programmierung

Die Programmierung des CIO-Schaltkreises erfolgt über insgesamt 48 interne Steuer- und Statusregister, die von dem programmierenden CPU-Schaltkreis über eine 6-Bit-Pointeradresse gelesen und/oder geschrieben werden können.
Die Übergabe der 6-Bit-Pointeradresse an den CIO-Schaltkreis erfolgt beim 8036-CIO und beim 8536-CIO in unterschiedlicher Weise.

Während beim U8036-CIO der gemultiplexte Adreß- und Datenbus eine direkte Kodierung der 6-Bit-Pointeradresse der Steuer- und Statusregister innerhalb des 8-Bit-Adreßbytes ermöglicht, erfolgt die Adressierung beim U82536-CIO über eine zweistufige Ausgabesequenz. Im ersten Schritt wird die 6-Bit-Pointeradresse des auszuwählenden Registers über die Datenbits D5, D4 ... D0 mit der Adreßbitbelegung A1=1; A0=1 an den U82536-CIO übergeben. Im zweiten Schritt kann dann über den 8-Bit-Datenbus mit der gleichen CIO-Adreßbitbelegung (A1=1; A0=1) das ausgewählte Register gelesen oder geschrieben werden.
Beim U82536-CIO gilt also grundsätzlich, daß alle Pointerausgaben und alle Lese-/Schreiboperationen, die sich an die 48 Register des U82536-CIO richten, über die Adreßbitbelegung A1=1; A0=1 erfolgen.

Für die Ausgabe der 6-Bit-Pointeradresse und für die Ein-/Ausgaben von den 48 internen Steuer- und Statusregistern existiert beim U82536-CIO ein Zustandsschema mit drei Grundzuständen, das bei der Programmierung genau zu beachten ist (Bild 3.8.).

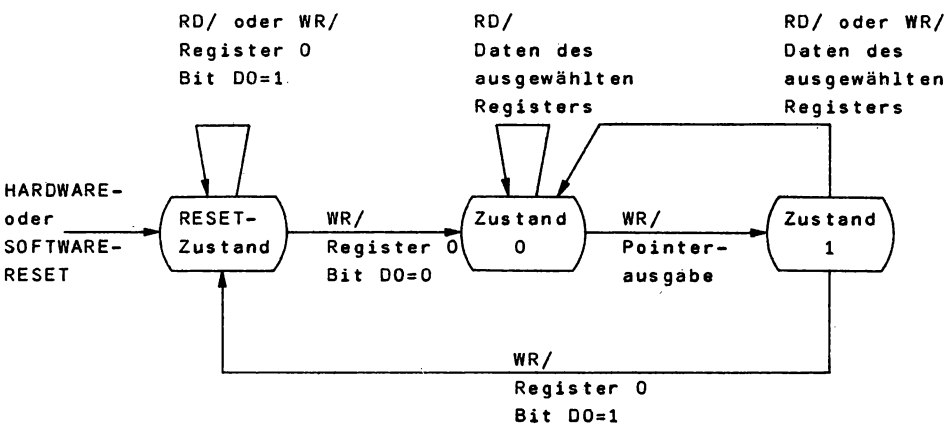


Bild 3.8. Zustandsschema des U82536-CIO bei der Programmierung der 48 Steuer- und Statusregister (RD/ steht für das Lesen - WR/ steht für das Schreiben von Daten)

Zu beachten ist außerdem, daß die Datenregister der CIO-Ports A, B und C beim U82536-CIO unter Nutzung der bei diesem Schaltkreis verbleibenden drei möglichen Kodierungen der CIO-Adreßleitungen (A1; A0) auch direkt adressiert werden können (gilt nur für den U82536-CIO !):

A1=1	A0=1	Adresse 6-Bit-Pointeradressenvorgabe
A1=1	A0=0	Adresse Port A
A1=0	A0=1	Adresse Port B
A1=0	A0=0	Adresse Port C

In der Tafel 3.6. ist eine Registerübersicht, in der Tafel 3.7. die genaue Registerbelegung beider CIO-Schaltkreisvarianten wiedergegeben.

Tafel 3.6. Registerübersicht U8036/U82536-CIO

Registerbezeichnung	Pointeradresse	Zugriffsarten
Hauptsteuerregister		
Interruptsteuerregister	000000	R/W
Konfigurationssteuerregister	000001	R/W
Port A Interruptvektor	000010	R/W
Port B Interruptvektor	000011	R/W
Counter/Timer-Interruptvektor	000100	R/W
Port C Datenpfadpolaritätsregister	000101	R/W
Port C Datenrichtungsregister	000110	R/W
Port C Ein-/Ausgabesteuerregister	000111	R/W
Hauptzugriffsregister		
Port A Kommando- und Statusregister	001000	
Port B Kommando- und Statusregister	001001	
Counter/Timer 1 Kommando- und Statusregister	001010	
Counter/Timer 2 Kommando- und Statusregister	001011	
Counter/Timer 3 Kommando- und Statusregister	001100	
Port A Daten **	001101	R/W
Port B Daten **	001110	R/W
Port C Daten **	001111	R/W
Counter/Timer-Register		
Counter/Timer 1 Momentanwertregister MS-Byte	010000	R
Counter/Timer 1 Momentanwertregister LS-Byte	010001	R
Counter/Timer 2 Momentanwertregister MS-Byte	010010	R
Counter/Timer 2 Momentanwertregister LS-Byte	010011	R
Counter/Timer 3 Momentanwertregister MS-Byte	010100	R
Counter/Timer 3 Momentanwertregister LS-Byte	010101	R
Counter/Timer 1 Zeitkonstantenregister MS-Byte	010110	R/W
Counter/Timer 1 Zeitkonstantenregister LS-Byte	010111	R/W
Counter/Timer 2 Zeitkonstantenregister MS-Byte	011000	R/W
Counter/Timer 2 Zeitkonstantenregister LS-Byte	011001	R/W
Counter/Timer 3 Zeitkonstantenregister MS-Byte	011010	R/W
Counter/Timer 3 Zeitkonstantenregister LS-Byte	011011	R/W
Counter/Timer 1 Betriebsartenauswahlregister	011100	R/W
Counter/Timer 2 Betriebsartenauswahlregister	011101	R/W
Counter/Timer 3 Betriebsartenauswahlregister	011110	R/W
Vektorrückleseregister	011111	R

Port A Betriebsartenauswahlregister		
Port A Betriebsartenauswahlregister	100000	R/W
Port A Handshake-Spezifikationsregister	100001	R/W
Port A Datenpfadpolaritätsregister	100010	R/W
Port A Datenrichtungsregister	100011	R/W
Port A Ein-/Ausgabesteuerregister	100100	R/W
Port A Zeichenpolaritätsregister	100101	R/W
Port A Zeichenübertragungsregister	100110	R/W
Port A Zeichenmaskenregister	100111	R/W

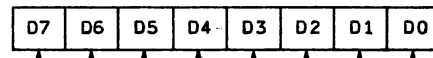
Port B Betriebsartenauswahlregister		
Port B Betriebsartenauswahlregister	101000	R/W
Port B Handshake-Spezifikationsregister	101001	R/W
Port B Datenpfadpolaritätsregister	101010	R/W
Port B Datenrichtungsregister	101011	R/W
Port B Ein-/Ausgabesteuerregister	101100	R/W
Port B Zeichenpolaritätsregister	101101	R/W
Port B Zeichenübertragungsregister	101110	R/W
Port B Zeichenmaske	101111	R/W

R Leseregister
W/R Lese-/Schreibregister

* Einige Bits nur lesbar - einige Bits lesbar und schreibbar.
** Diese Register sind, außer über die 6-Bit-Pointeradresse beim U82536-CIO, auch direkt über die Auswahl einer zugeordneten Zugriffsadresse lesbar und schreibbar.

Tafel 3.7. Bitbelegung der 48 Steuer- und Statusregister des U8036/U82536-CIO

Interruptsteuerregister — Adresse: 000000 (Lese- und Schreibregister)



MIE=0 CIO-Interruptsystem gesperrt
DLC=1 IEO=Low / Interruptkette gesperrt

NV =1 keine Interruptvektorausgabe

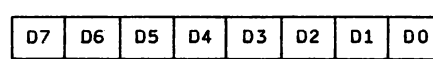
VIS=1 Port A - Interruptvektor enthält Status
VIS=1 Port B - Interruptvektor enthält Status
VIS=1 CTC - Interruptvektor enthält Status

RJA=0 Steuerregisteradresse verschoben AD6 - AD1
RJA=1 Steuerregisteradresse rechtsbündig AD5 - AD0

RESET=1 Softwarereset

Achtung: Bei der Programmierung des U8036-CIO ist bei der 6-Bit-Adressenkodierung der 48 Steuer- und Statusregister innerhalb des 8-Bit-Adreßbytes der Zustand des RJA-Bits besonders beim U8000-Byte-Ein-/Ausgabeverkehr mit ungeraden Adressen zu beachten.

Konfigurationssteuerregister — Adresse: 000001 (Lese- und Schreibregister)



PBE =1 Port B aktiviert

CT1E=1 CTC 1 aktiviert
CT2E=1 CTC 2 aktiviert
PCE/CT3E=1 Port C und CTC 3 aktiviert

PLC=1 Port A und B arbeiten verkettet

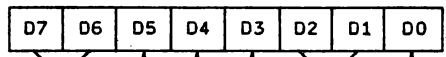
PAE=1 Port A aktiviert

LC1/LC0 CTC-Verkettungssteuerung

LC1	LC0	
0	0	CTC-Kanäle arbeiten unabhängig voneinander
0	1	CTC 1 (invertiert) ist Gate für CTC 2
1	0	CTC 1 (invertiert) ist Trigger für CTC 2
1	1	CTC 1 (invertiert) ist Zähler für CTC 2

Port A und B Betriebsartenauswahlregister Adresse: 100000 Port A
101000 Port B

(Lese- und Schreibregister)



PTS1/PTS0 Betriebsartenauswahl

PTS1	PTS0	
0	0	Bit-Port kein Handshake
0	1	Input-Port '1 aus 4' Handshake
1	0	Output-Port '1 aus 4' Handshake
1	1	Bidi-Port '1 aus 2' Handshake

ITB=1 Interrupt nach zwei Bytes

SB =1 Zeichen einfach gepuffert

IM0=1 Interrupt nachdem Match-Zeichen gefunden

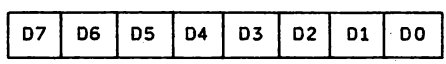
PMS1/PMS0 Spezifikation Match-Zeichensuche

PMS1	PMS0	
0	0	keine Zeichensuche
0	1	"AND" Mode
1	0	"OR" Mode
1	1	"OR PRIORITY ENCODED VECTOR" Mode

LPM=1 Suchmuster abspeichern (gilt im: Bit-Mode)
DTE=1 Deskew-Timer aktiviert (gilt im: Handshake-Mode)

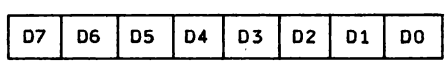
Port A und B Datenregister Adresse: 001101 Port A
001110 Port B

(Lese- und Schreibregister)



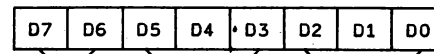
Port C Datenregister Adresse: 001111 Port C

(Lese- und Schreibregister)



Achtung: Die vier MS-Bits bilden die Ausgabemaske für die vier LS-Bits des Ein-/Ausgabeports C. Maskierte Bits (=1) werden bei der Ausgabe nicht verändert.

Port A und B Handshake-Spezifikationsregister Adresse: 100001 Port A
101001 Port B
(Lese- und Schreibregister)



HTS1/HTS0 Handshake-Typspezifikation

HTS1	HTS0	
0	0	Interlock-Handshake-Mode
0	1	Strobe-Handshake-Mode
1	0	Pulse-Handshake-Mode
1	1	3-Wire-Handshake-Mode

RWS2/RWS1/RWS0 Request/Wait-Spezifikationsbits

RWS2	RWS1	RWS0	
0	0	0	Request/Wait nicht aktiv
0	0	1	Output-Wait aktiv
0	1	1	Input-Wait aktiv
1	0	0	Special-Request aktiv
1	0	1	Output-Request aktiv
1	1	1	Input-Request aktiv

DTS3/DTS2/DTS1 Deskew-Time-Spezifikation

Spezifiziert werden die drei MS-Bits der Zeitkonstante des Dehnungszeitgebers (deskew timer) - das LS-Bit ist immer 1.

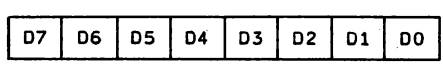
DTE*	DTS3	DTS2	DTS1	Dehnungszeit (in PCLK-Zyklen)
0	X	X	X	0
1	0	0	0	2
1	0	0	1	4
1	0	1	0	6
1	0	1	1	8
1	1	0	0	10
1	1	0	1	12
1	1	1	0	14
1	1	1	1	16

siehe Betriebsartenauswahlregister

Datenpfadpolaritätsregister Adresse: 100010 Port A
101010 Port B
000101 Port C (4 LSB)

(Lese- und Schreibregister)

DPP=1 invertiertes Datenbit
DPP=0 nichtinvertiertes Datenbit



Datenrichtungsregister

Adresse: 100011 Port A
101011 Port B
000110 Port C (4 LSB)

(Lese- und Schreibregister)

DD=1 Eingabebits
DD=0 Ausgabebits

D7	D6	D5	D4	D3	D2	D1	D0

Port A und B Kommando- und Statusregister

Adresse: 001000 Port A
001001 Port B

(Lese- und teilweise Schreibregister)

D7	D6	D5	D4	D3	D2	D1	D0

IUS Interrupt anerkannt
IE Interruptanmeldung erlaubt
IP Interrupt angemeldet

Funktionskode IUS, IE und IP (Schreiben)

Nullkode	0	0	0
Clear IP und IUS	0	0	1
Set IUS	0	1	0
Clear IUS	0	1	1
Set IP	1	0	0
Clear IP	1	0	1
Set IE	1	1	0
Clear IE	1	1	1

ERR=1 Zeichen (IP=1) gefunden (Read-Only-Bit)

ORE=1 Daten ausgegeben (Read-Only-Bit)

IRF=1 Daten eingelesen (Read-Only-Bit)

PMF=1 Zeichenmuster erkannt (Read-Only-Bit)

IOE=1 IP wird nach dem Rücksetzen bei ERR=1 wieder gesetzt
IOE=0 Zeichensuche wird bei ERR=1 nicht ausgeführt

Ein-/Ausgabe-Steuerregister

Adresse: 100100 Port A
101100 Port B
000111 Port C (4 LSB)

(Lese- und Schreibregister)

SIO=0 normale Ein- oder Ausgabebits
SIO=1 Ausgabebits: Open-Drain
Eingabebits: Impuls-Fangschalt.

D7	D6	D5	D4	D3	D2	D1	D0

Zeichen-Polaritätsregister (PP)

Adresses: 100101 Port A
101101 Port B
(Lese- und Schreibregister)

D7	D6	D5	D4	D3	D2	D1	D0

Zeichen-Übertragungsregister (PT)

Adresses: 100110 Port A
101110 Port B
(Lese- und Schreibregister)

D7	D6	D5	D4	D3	D2	D1	D0

Maskenregister (PM)

Adresses: 100111 Port A
101111 Port B
(Lese- und Schreibregister)

D7	D6	D5	D4	D3	D2	D1	D0

Register (PM)	Register (PT)	Register (PP)	Bedeutung
0	0	0	Bit maskiert
0	1	0	jede Flanke
1	0	0	Bit 0
1	0	1	Bit 1
1	1	0	fallende Flanke
1	1	1	steigende Flanke

Interruptvektorregister

Adresse: 000010 Port A
000011 Port B
000100 Counter/Timer

(Lese- und Schreibregister)

D7	D6	D5	D4	D3	D2	D1	D0

Vektorrückleseregister

Adresse: 011111

(Leseregister)

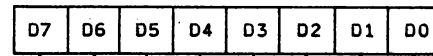
D7	D6	D5	D4	D3	D2	D1	D0

Momentan höchstpriorisierter Interrupt
(Liegt kein Interrupt an, dann alles 1 !)

CTC Betriebsartenauswahlregister

Adresse: 011100 Counter/Timer 1
011101 Counter/Timer 2
011110 Counter/Timer 3

(Lese- und Schreibregister)



CSC=1 Laden der Zeitkonstante
CSC=0 Stop Zählvorgang

EOE =1 externer Ausgang aktiv

ECE =1 Zählermode - Takteingang
ECE =0 Timermode - Takteingang

ETE =1 externer Start/Wiederstart des Zählers

EGE =1 externer Gate-Eingang

REB =1 Trigger verursacht Neustart des Zähler
REB =0 Trigger wird ignoriert

DCS1, DSC0 Retrigger-Aktivierungsbits

DSC1	DCS0	
0	0	Pulse-Output
0	1	One-Shot-Output
1	0	Squre-Wave-Output
1	1	nicht benutzt

Bitzuordnung des Counter/Timers bei externem Zugriff:

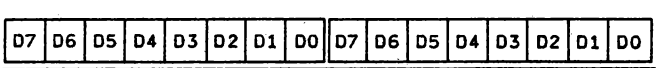
Funktion	Counter/ Timer 1	Counter/ Timer 2	Counter/ Timer 3
Counter Timer Output	Port B4	Port B0	Port C0
Counter Input	Port B5	Port B1	Port C1
Trigger Input	Port B6	Port B2	Port C2
Gate Input	Port B7	Port B3	Port C3

CTC Zeitkonstantenregister

Adresse: 010110 CTC 1 MS-Byte
010111 CTC 1 LS-Byte
011000 CTC 2 MS-Byte
011001 CTC 2 LS-Byte
011010 CTC 3 MS-Byte
011011 CTC 3 LS-Byte

höherwertiges Byte

niederwertiges Byte



CTC Momentanwertregister

Adresse: 010000 CTC 1 MS-Byte
010001 CTC 1 LS-Byte
010010 CTC 2 MS-Byte
010011 CTC 2 LS-Byte
010100 CTC 3 MS-Byte
010101 CTC 3 LS-Byte

höherwertiges Byte

D7	D6	D5	D4	D3	D2	D1	D0

niederwertiges Byte

D7	D6	D5	D4	D3	D2	D1	D0

CTC Kommando- und Statusregister

Adresse: 001010 Counter/Timer 1
001011 Counter/Timer 2
001100 Counter/Timer 3

(Lese- und teilweise Schreibregister)

D7	D6	D5	D4	D3	D2	D1	D0

IUS Interrupt anerkannt
IE Interruptanmeldung erlaubt
IP Interrupt angemeldet

Funktionskode IUS, IE und IP (Schreiben)

Nullkode	0	0	0
Clear IP und IUS	0	0	1
Set IUS	0	1	0
Clear IUS	0	1	1
Set IP	1	0	0
Clear IP	1	0	1
Set IE	1	1	0
Clear IE	1	1	1

ERR=1 Zeichen (IP=1) gefunden (Read-Only-Bit)

RCC=1 Lesezählersteuerung (Read-Only-Bit)

GCB=1 Gate offen
GCB=0 Gate geschlossen

TCB=1 Triggerstart des Zählers (Write-Only-Bit)
TCB=0 Startverhinderung des Zählers (Write-Only-Bit)

CIP=1 Zählvorgang läuft (Read-Only-Bit)

Achtung: Wurde in der Tafel eine Aussage nur für eine Bitbelegung angegeben (=1 oder =0), gilt für die invertierte Bitbelegung jeweils eine entgegengesetzte Aussage.

Port A/B Konfiguration	Port C Pin-Konfiguration			
	Pin C3	Pin C2	Pin C1	Pin C0
Port A und B sind Bit-Ports	Bit I/O	Bit I/O	Bit I/O	Bit I/O
Port A Input- oder Output-Port (Interlock-, Strobe- oder Pulse-Handshake) *	RFD oder DAV/	ACKIN/	Request-Wait/ oder Bit I/O	Bit I/O
Port B Input- oder Output-Port (Interlock-, Strobe- oder Pulse-Handshake) *	REQUEST-WAIT/ oder Bit I/O	Bit I/O	RFD oder DAV/	ACKIN/
Port A oder B Input-Port (3-Wire-Handshake)	RFD (Output)	DAV/ (Input)	Request-Wait/ oder Bit I/O	DAC (Output)
Port A oder B Output-Port (3-Wire-Handshake)	DAV/ (Output)	DAC (Input)	Request-Wait/ oder Bit I/O	RFD (Output)
Port A oder B Bidi-Port (Interlock-, Strobe- oder Pulse-Handshake)	RFD oder DAV/	ACKIN/	Request-Wait/ oder Bit I/O	IN-OUT/

* Beide Ports A und B können gleichzeitig als Input- oder Output-Ports mit Interlock-, Strobe- oder Pulse-Handshake und mit einem Request/Wait-Signalspiel spezifiziert werden.

Beispielprogrammierungsfolge U8036-CIO:

Register	**Adresse**	Daten (Hex)	
Interruptsteuerregister	X0000000*	01	Reset U8036-CIO
Interruptsteuerregister	X000000X	00	Löschen Reset
Port A Betriebsartenauswahlreg.	X100000X	60	Input-Port; Interrupt nach zwei Bytes
Port A Interruptvektorregister	X000010X	xx	Interruptvektor
Port A Kommando- und Statusreg.	X001000X	C0	Interrupt aktiv
Port C Datenrichtungsregister	X000110X	F4	PC2 Input von PC0; PC1 und PC3 Output
Port C Daten	X001111X	48	RFD Init-High; PC0 und PC1 Init-Low
Port B Betriebsartenauswahlreg.	X101000X	06	Bit-Port; Zeichensu.
Port B Datenrichtungsregister	X101011X	FE	PB0 Output; PB1-PB7 Input
Port B Zeichenpolaritätsregister	X101101X	FF	Interrupteingänge High aktiv
Port B Zeichenmaske	X101111X	BE	PB0 und PB6 off

Port B Interruptvektorregister	X000011X	xx	Interruptvektor
Port B Kommando- Und Statusreg.	X001001X	C0	Interrupt aktiv
CTC 1 Betriebsartenauswahlreg.	X011100X	14	Single-Cycle; Ext-Trigger aktiv Re-Trigger aktiv
CTC 1 Zeitkonstantenreg. (MS)	X010110X	xx	Zeitkonstante
CTC 1 Zeitkonstantenreg. (LS)	X010111X	xx	Zeitkonstante
CTC-Interruptvektorregister	X000100X	xx	Interruptvektor
CTC 1 Kommando- und Statusreg.	X001010X	C0	Interrupt aktiv
CTC 2 Betriebsartenauswahlreg.	X011101X	C2	Continuous; Ext-Output aktiv
CTC 2 Zeitkonstantenreg. (MS)	X011100X	xx	Zeitkonstante
CTC 2 Zeitkonstantenreg. (LS)	X011001X	xx	Zeitkonstante
CTC 3 Betriebsartenauswahlreg.	X011110X	80	Continuous
CTC 3 Zeitkonstantenreg. (MS)	X011010X	xx	Zeitkonstante
CTC 3 Zeitkonstantenreg. (LS)	X011011X	xx	Zeitkonstante
CTC 3 Kommando- und Statusreg.	X001100X	C0	Interrupt aktiv
Konfigurationssteuerregister	X000001X	F4	Port-Aktivierung
CTC 1 Kommando- und Statusreg.	X001010X	06	Trigger/Gate Komman.
CTC 2 Kommando- und Statusreg.	X001011X	06	Trigger/Gate Komman.
CTC 3 Kommando- und Statusreg.	X001100X	06	Trigger/Gate Komman.
Interruptsteuerregister	X000000X	8C	Interrupt aktiv; Port B und CTC enthält Status

* Wenn der Initialzustand des RJA-Bits unbekannt ist, muß der erste Zugriff auf das Interruptsteuerregister mit der Bitbelegung AD0=0 erfolgen.

Verwendete Abkürzungen:

ACKIN	acknowledge input signal
CIP	count in progress
CTx	counter/timer x enable
CSC	continous single cycle
DAV	data available
DCS	duty cycle selects
DD	data direction register
DLC	disable lower chain
DPP	data path polarity register
DTE	deskew timer enable
DTS	deskew timer specification
DTE/LPM	deskew timer enable/latch on pattern match
ECE	external count enable
EGE	external gate enable
EOE	external output enable
ERR	interrupt error
ETE	external trigger enable
GCB	gate command bit
HTS	handshake type specification bits
IE	interrupt enable
IMO	Interrupt on match only

IOE	interrupt on error
IP	interrupt pending
IRF	input register full
ITB	interrupt on two bytes
IUS	interrupt under service
LC	counter/timer link control
LPM	latch on pattern match
LSB	least-significant bit
MIE	master interrupt enable
MSB	most-significant bit
NV	no vector
OCS	output duty cycle selects
ORE	output register empty
PAE	port A enable
PBE	port B enable
PCE	port C enable
PCLK	peripheral clock
PLC	port link control
PM	pattern mask register
PMF	pattern match flag
PMS	pattern mode specification bits
PP	pattern polarity register
PT	pattern transition register
PTS	port type selects
RCC	read counter control
REB	retrigger enable bit
RFD	ready for data
RJA	right justified address
RWS	request/wait specification bits
SB	single buffered
SIO	special input/output
TCB	trigger command bit
VIS	counter/timer vector includes status

3.4. Serieller Kommunikations-Ein-/Ausgabebaustein 8 × 30-SCC

Alle seriellen Ein-/Ausgabeaktivitäten im 8-Bit-Mikroprozessorsystem U880 werden standardmäßig über den U856-SIO abgewickelt. Die Takterzeugung, mit der die Übertragungsrate der seriellen Datenübertragung bei diesem Baustein festgelegt wird, übernimmt dabei in der Regel ein Zähler-/Zeitgeberschaltkreis U857-CTC (Abschn. 2.3. und 2.4.). Serielle Datenübertragungsaufgaben werden in vielen Mikrorechneranwendungen eine wichtige Rolle spielen - zur Ankopplung von Datensichtgeräten ebenso, wie zur Rechnerkopplung, Datenfernübertragung u.v.a.m..

Im 16-Bit-Mikrorechnersystem U8000 steht für die Abwicklung der verschiedensten seriellen Ein-/Ausgabeaktivitäten der alle Grundfunktionen (SIO und CTC - einschließlich Baudratentakterzeugung) in einem hochintegrierten Bauelement vereinigende U8x30-SCC zur Verfügung (SCC serial communications controller). Er kann für die verschiedensten Aufgaben bei der seriellen Ein-/Ausgabeabwicklung verwendet werden, von einfachen asynchronen seriellen Datenstrukturen, über die verschiedensten Arten synchroner Datentelegramme, bis hin zur Ein-/Ausgabe von FM-kodierten Floppy-Disk-Laufwerksdatenströmen u.a.m.. Eine große Anzahl programmierbarer Optionen erlaubt die flexible Anpassung des U8x30 an den jeweiligen Einsatzfall.
Insbesondere existieren beim U8x30-SCC:

- zwei unabhängige, voll duplex arbeitende, serielle Ein-/Ausgabekanäle mit einer Übertragungsrate von bis zu 1 MBit pro Sekunde, jeder mit einer getrennten Takterzeugung für einen eigenständigen Baudratengenerator,

- programmgesteuerte Abwicklung von verschiedenen synchronen und asynchronen seriellen Datenströmen, einschließlich NRZ-, NRZI- oder FM-kodierter Telegramme durch eine integrierte digitale PLL-Schaltung (PLL phase locked loop),

- Übertragung asynchroner serieller Datentelegramme mit fünf bis 8 Datenbits, 1-, 1 1/2- oder 2-Stop-Bits pro Zeichen, Breaksignalüberwachung/-generierung, Paritäts-/Überlaufskontrolle (overrun), Framing-Error-Überwachung,

- Übertragung synchroner serieller Datentelegramme mit interner oder externer Zeichensynchronisation, einem oder zwei SYNC-Zeichen zur Sender-/Empfänger-Synchronisierung und CRC-16- bzw. CRC-CCITT-Fehlerüberwachung,

- Übertragung synchroner serieller Datentelegramme entsprechend der SDLC/HDLC-Norm mit Frome-Error-Überwachung, automatischer Null-Generierung/-Überwachung, I-Field-Handling, Abbruchgenerierung/-überwachung, CRC-Generierung/-überwachung,

- integrierter automatischer Auto-Echo- und Lokal-Mode.

Die Programmierung des U8x30-SCC erfolgt für jeden der beiden SCC-Kanäle über 9 Lese- und 14 Schreibregister, die eine sehr große Variationsbreite beim Einsatz dieses Schaltkreises bieten.

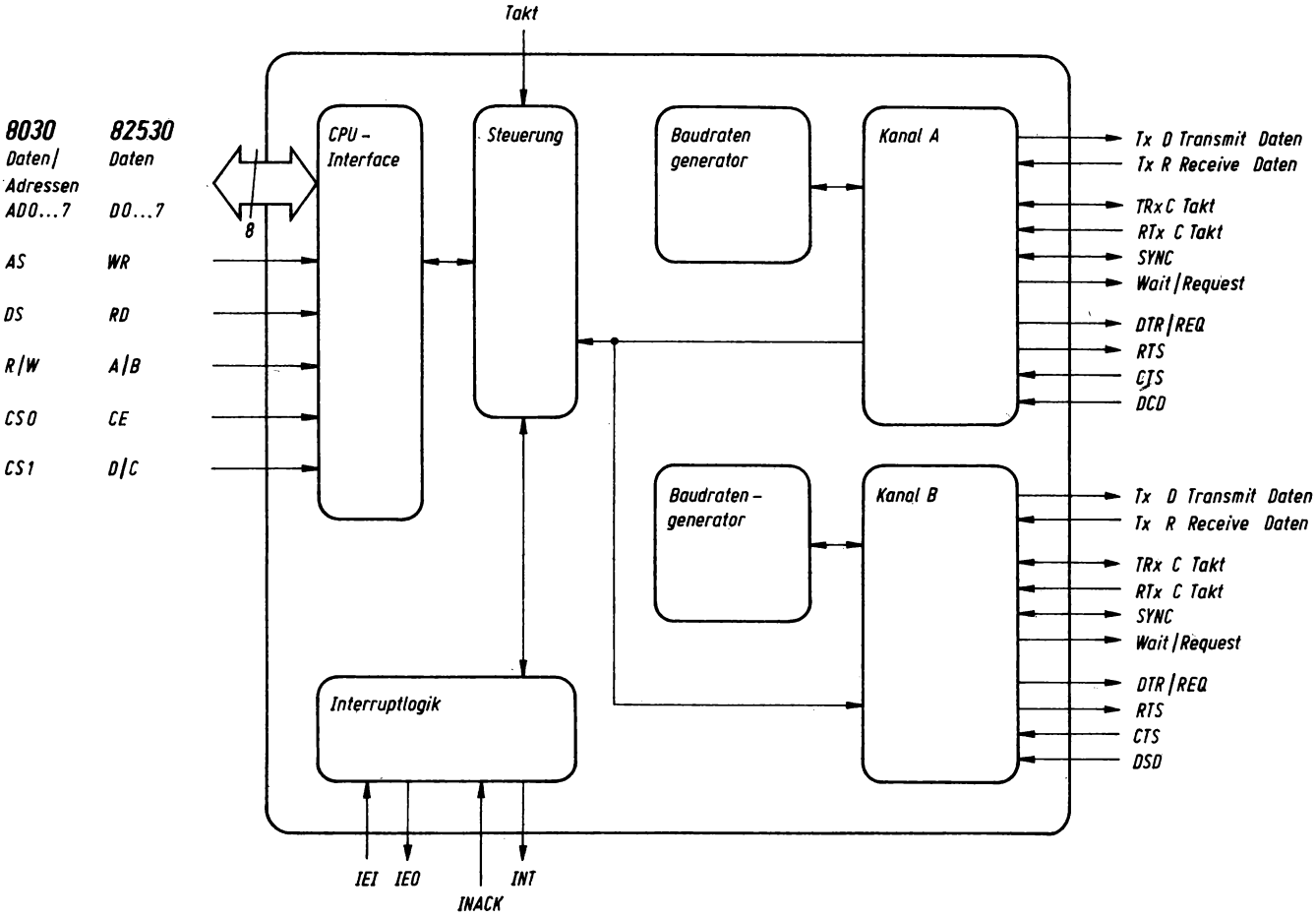


Bild 3.9. Aufbau und Signalstruktur des Bausteins für serielle Ein-/Ausgaben U8x30-SCC

Wegen des in diesem Band nur begrenzt zur Verfügung stehenden Platzes ist eine ausführliche Behandlung des U8x30-SCC hier nicht möglich. Es wurden aber alle für den Softwarebearbeiter wichtigen Tabellen und Informationsübersichten zusammengetragen, um den Leser bei der Programmierung dieses Schaltkreises möglichst umfassend zu unterstützen. In speziellen Einsatzfällen sind notwendigenfalls experimentelle Untersuchungen erforderlich. Außerdem sei auf die Kundendokumentation und auf die Datenbücher von zum U8x30-SCC kompatiblen Schaltkreisen verwiesen /54/, /55/, /56/.

Zu beachten ist, daß zwei Versionen des SCC-Schaltkreises existieren, die sich ausschließlich in der Busanpassung unterscheiden. Während der U8030-SCC einen gemultiplexten Adreß-/Datenbus und zwei Chipselektsignale besitzt, kennt der U82536-SCC einen Datenbus, ein A-/B-Kanalauswahlsignal, ein Chip-Enable-Aktivierungssignal und ein Signal zur Kennzeichnung, ob ein Daten- oder Steuerwortverkehr ablaufen soll.
Die Unterschiede der Busanpassung bei beiden Schaltkreisen finden hauptsächlich bei der Programmierung ihren Niederschlag.
Die Einsatzgebiete des U8030-SCC sind vor allem U8000-Mikrorechnersysteme, zu denen hier eine optimale Busanpassung existiert, während der U82530-SCC vor allem für den universellen Einsatz in beliebigen Mikroprozessorsystemen vorgesehen ist.

3.4.1. Aufbau

Der U8x30-SCC besteht aus zwei funktionell absolut identischen seriellen Ein-/Ausgabekanälen (A und B) mit jeweils einer seriellen Eingabe- und einer seriellen Ausgabesignalleitung für die Voll-Duplex-Datenübertragung und einer ganzen Reihe zugehöriger Steuersignalleitungen.

Jedem der beiden seriellen Ein-/Ausgabekanäle ist außerdem ein eigenständiger Baudratengenerator und ein Lese-/Schreibregistersatz zur Programmierung zugeordnet. Für beide serielle Datenkanäle gemeinsam existiert eine Interruptsteuerlogik, ein Mikrorechnerinterface und schließlich die SCC-Schaltkreissteuerlogik (Bild 3.9.), die im Aufbau und in der Funktion ähnlichen Funktionsblöcken bekannter Peripherieschaltkreise für Mikrorechner entsprechen.

3.4.2. Programmierung

Die Programmierung des SCC-Schaltkreise erfolgt über insgesamt 9 Lese- und 14 Schreibregister für jeden Kanal, die von dem programmierenden CPU-Schaltkreis über eine Adresse gelesen bzw. geschrieben werden können.
Die Leseregister des U8x30-CIO spiegeln den momentanen internen SCC-Status wider. Er ist abhängig von der eingestellten Betriebsart, von den Lese-/Schreibdaten und von dem Zugriff des CPU-Schaltkreises auf diese Daten.
Der Schreibregistersatz des U8x30-CIO besteht aus zehn Steuerregistern, aus zwei SYNC-Zeichenregistern, aus zwei Baudraten-Zeitkonstantenregistern, aus einem Interruptregister und aus einem Sendedatenregister. Der einzige Unterschied in dem Registersatz zwischen U8030-SCC und U82530-SCC besteht in einem leicht modifizierten Kommandoregister WR0 (s.a. Tafel 3.9.).

Der wichtigste Unterschied zwischen beiden Schaltkreisen besteht für den Programmierer in der unterschiedlichen Übergabe der Adresse des ausgewählten Lese- und Schreibregisters vom CPU-Schaltkreis an den U8x30-SCC. Während beim U8030-CIO der gemultiplexte Adreß- und Datenbus eine direkte Kodierung der 3-Bit-Registeradresse der Lese- und Schreibregister innerhalb des 8-Bit-Adreßbytes ermöglicht, erfolgt die Adressierung beim U82536-CIO über eine zweistufige Ausgabesequenz. Im ersten Schritt dieser Ausgabesequenz wird über den 8-Bit-Datenbus eine 3-Bit-Lese- oder Schreib-Registeradresse in die unteren drei Bit des Schreibregisters WR0 eingetragen. Im zweiten Schritt kann dann über den 8-Bit-Datenbus das ausgewählte Register gelesen oder geschrieben werden. Danach stellt sich der Lese-/Schreibpointer wieder automatisch auf Null, so daß das Leseregister RR0 und das Schreibregister WR0 immer direkt gelesen bzw. geschrieben werden kann.
Mit Hilfe eines speziellen Hardware-Signalspiels (D/C-Pin High) ist es beim U82530-SCC außerdem möglich, das Sende- und Empfangsdatenregister (RR8 bzw. WR8), unabhängig von einer Pointerregister-Ein-/ausgabe, in einer einstufigen, direkten Ein-/Ausgabe zu lesen bzw. zu schreiben.
In der Tafel 3.9 wird eine Übersicht über den U8x30-SCC Registersatz zusammen mit der Registeradressierung gegeben, die Tafel 3.10. beschreibt den Inhalt bzw. die Bedeutung jedes einzelnen der Lese-/Schreibregister und in der Tafel 3.11. ist der Registersatz des U8x30-SCC nach einem Hardware-Reset in Verbindung mit einer Standard-Initialisierungssequenz zu entnehmen.

Tafel 3.9. U8x30-SCC Registerübersicht und Registeradressierung

Schreibregister	
WR0	Kommandoregister mit Registerpointer (nur U82530); CRC-Initialisierung und Resetinformationen
WR1	Interruptbedingungen; Wait-/DMA-Steuerung
WR2	Interruptvektor
WR3	Empfangssteuerparameter; Anzahl der Bits pro Zeichen; Rx CRC-Aktivierung
WR4	Sende- und Empfangsparameter; Taktrate; Anzahl SYNC-Zeichen; Anzahl der Stop- und Paritätsbits
WR5	Sendesteuerparameter; Anzahl der Tx-Bits pro Zeichen; Tx CRC-Aktivierung
WR6	SYNC-Zeichen (erstes Byte) oder SDLC-Adressenfeld
WR7	SYNC-Zeichen (zweites Byte) oder SDLC-Flag
WR8	Senderegister
WR9	Interruptsteuerung; Reset-Bits; Daisy-Chain-Steuerung
WR10	Sende- und Empfangssteuerparameter; NRZI-, NRZ- und FM-Kodierung; CRC-Reset
WR11	Taktsteuerung; Rx- und Tx-Taktquellen
WR12	Zeitkonstante Baudratengenerator (untere 8 Bits)
WR13	Zeitkonstante Baudratengenerator (obere 8 Bits)
WR14	Steuerparameter des Baudratengenerators; PLL-Steuerung; Autoecho- und Lokalmode
WR15	External/Status-Interrupt; externe Interruptbedingungen

Leseregister	
RR0	Sende- und Empfangspufferstatus; externer Status
RR1	Status der speziellen Empfangsbedingungen; Fehlerbedingungen
RR2	Interruptvektor-Rückleseregister: Kanal B modifizierter Interruptvektor Kanal A nicht modifizierter Interruptvektor
RR3	Interruptanerkennungsbedingungen (nur Kanal A)
RR8	Empfangsregister
RR10	Statusparameter
RR12	Zeitkonstante des Baudratengenerators (untere 8 Bit)
RR13	Zeitkonstante des Baudratengenerators (obere 8 Bit)
RR15	Steuerinformationen External/Status-Interrupt

U8030-SCC Registeradressen

AD	AD	AD	AD	AD	Shift-Left-Adressierung AD5, AD4, AD3, AD2, AD1		Shift-Right-Adressierung AD4, AD3, AD2, AD1, AD0	
					Schreiben	Lesen	Schreiben	Lesen
0	0	0	0	0	WR0B	RR0B	WR0B	RR0B
0	0	0	0	1	WR1B	RR1B	WR0A	WR0A
0	0	0	1	0	WR2	RR2B	WR1B	WR1B
0	0	0	1	1	WR3B	RR3B	WR1A	RR1A
0	0	1	0	0	WR4B	(RR0B)	WR2	RR2B
0	0	1	0	1	WR5B	(RR1B)	WR2	RR2A
0	0	1	1	0	WR6B	(RR2B)	WR3B	RR3B
0	0	1	1	1	WR7B	(RR3B)	WR3A	RR3A
0	1	0	0	0	WR8B	RR8B	WR4B	RR0B
0	1	0	0	1	WR9	(RR13B)	WR4A	RR0A
0	1	0	1	0	WR10B	RR10B	WR5B	(RR1B)
0	1	0	1	1	WR11B	(RR15B)	WR5A	(RR1A)
0	1	1	0	0	WR12B	RR12B	WR6B	RR2B
0	1	1	0	1	WR13B	RR13B	WR6A	RR2A
0	1	1	1	0	WR14B	(RR10B)	WR7B	(RR3B)
0	1	1	1	1	WR15B	RR15B	WR7A	(RR3A)
1	0	0	0	0	WR0A	RR0A	WR8B	RR8B
1	0	0	0	1	WR1A	RR1A	WR8A	RR8A
1	0	0	1	0	WR2	RR2A	WR9	(RR13B)
1	0	0	1	1	WR3A	RR3A	WR9	(RR13A)
1	0	1	0	0	WR4A	(RR0A)	WR10B	RR10B
1	0	1	0	1	WR5A	(RR1A)	WR10A	RR10A
1	0	1	1	0	WR6A	(RR2A)	WR11B	(RR15B)
1	0	1	1	1	WR7A	(RR3A)	WR11A	(RR11A)
1	1	0	0	0	WR8A	RR8A	WR12B	RR12B
1	1	0	0	1	WR9	(RR13A)	WR12A	RR12A
1	1	0	1	0	WR10A	RR10A	WR13B	RR13B
1	1	0	1	1	WR11A	(RR15A)	WR13A	RR13A
1	1	1	0	0	WR12A	RR12A	WR14B	(RR10B)
1	1	1	0	1	WR13A	RR13A	WR14A	(RR10A)
1	1	1	1	0	WR14A	(RR10A)	WR15B	RR15B
1	1	1	1	1	WR15A	RR15A	WR15A	RR15A

U82530-SCC Registeradressen (Pointeradressierung)							
A/B	PNT2	PNT1	PNT0	Schreiben (Point-High=000)	Lesen (Point-High=000)	Schreiben (Point-High=001)	Lesen (Point-High=001)
0	0	0	0	WR0B	RR0B	WR8B	RR8B
0	0	0	1	WR1B	RR1B	WR9	RR13B
0	0	1	0	WR2	RR2B	WR10B	RR10B
0	0	1	1	WR3B	RR3B	WR11B	(RR15B)
0	1	0	0	WR4B	(RR0B)	WR12B	RR12B
0	1	0	1	WR5B	(RR1B)	WR13B	RR13B
0	1	1	0	WR6B	(RR2B)	WR14B	(RR10B)
0	1	1	1	WR7B	(RR3B)	WR15B	RR15B
1	0	0	0	WR0A	RR0A	WR8A	RR8A
1	0	0	1	WR1A	RR1A	WR9	(RR13A)
1	0	1	0	WR2	RR2A	WR10A	RR10A
1	0	1	1	WR3A	RR3A	WR11A	(RR15A)
1	1	0	0	WR4A	(RR0A)	WR12A	RR12A
1	1	0	1	WR5A	(RR1A)	WR13A	RR13A
1	1	1	0	WR6A	(RR2A)	WR14A	RR14A
1	1	1	1	WR7A	(RR3A)	WR15A	RR15A

Tafel 3.10. Bitbelegung der 9 Lese- und 14 Schreibregister des U8x30-SCC

Kommandoregister U8030-SCC	D7	D6	D5	D4	D3	D2	D1	D0
Nullkode	0	0						
Reset CRC-Überprüfung Empfänger	0	1						
Reset CRC-Generator Sender	1	0						
Reset Underrun/EOM-Latch Sender	1	1						
Nullkode			0	0	0			
Nullkode			0	0	1			
Reset External/Status-Interrupts			0	1	0			
Senden Abbruchsignalfolge (SDLC)			0	1	1			
Interrupt beim nächsten empfangenen Zeichen			1	0	0			
Reset Senderinterruptanmeldung			1	0	1			
Reset Fehlerbits RR1			1	1	0			
Reset höchstpriorisierter anerk. Interrupt			1	1	1			
0								
Nullkode							0	0
Nullkode							0	1
Auswahl Shift-Left-Adressierung (Tafel 3.9) *							1	0
Auswahl Shift-Right-Adressierung (Tafel 3.9) *							1	1

* s.a. RJA-Bit U8x36-CIO in Tafel 3.7.

Kommandoregister U82530-SCC

	D7	D6	D5	D4	D3	D2	D1	D0
Nullkode	0	0						
Reset CRC-Überprüfung Empfänger	0	1						
Reset CRC-Generator Sender	1	0						
Reset Underrun/EOM-Latch Sender	1	1						
Nullkode			0	0	0			
Point-High-Kommando (Tafel 3.9.)			0	0	1			
Reset External/Status-Interrupts			0	1	0			
Senden Abbruchsignalfolge (SDLC)			0	1	1			
Interrupt beim nächsten empfangenenZeichen			1	0	0			
Reset Senderinterruptanmeldung			1	0	1			
Reset Fehlerbits RR1			1	1	0			
Reset höchstpriorisierter anerk. Interrupt			1	1	1			
Register 0						0	0	0
Register 1						0	0	1
Register 2						0	1	0
Register 3						0	1	1
Register 4						1	0	0
Register 5						1	0	1
Register 6						1	1	0
Register 7						1	1	1
Register 8						0	0	0
Register 9						0	0	1
Register 10						0	1	0
Register 11						0	1	1
Register 12						1	0	0
Register 13						1	0	1
Register 14						1	1	0
Register 15 (* Point-High-Kommando = 001)						1	1	1

Schreibregister 1

	D7	D6	D5	D4	D3	D2	D1	D0
Wait/DMA-Pin aktiv=1 / inaktiv=0								
Wait/DMA-Pin Request=1 / Wait=0								
Wait/DMA-Pin Empfänger=1 / Sender=0								
Empfangsinterrupt bei spezieller Empfangsbedingung				1	1			
wie 11 und bei jedem empfangenen Zeichen				1	0			
wie 11 und beim ersten empfangenen Zeichen				0	1			
Empfangsinterrupt nicht aktiv				0	0			
Parität ist eine spez. Bedingung ja=1 / nein=0								
Sendeinterrupt aktiv=1 / inaktiv=0								
alle External/Status-Interrupts aktiv=1 / inaktiv=0								

Schreibregister 2

D7	D6	D5	D4	D3	D2	D1	D0

SCC-Interruptvektor V7, V6 ... V0

Schreibregister 3

D7	D6	D5	D4	D3	D2	D1	D0

8 Bits pro Empfangszeichen	1	1
6 Bits pro Empfangszeichen	1	0
7 Bits pro Empfangszeichen	0	1
5 Bits pro Empfangszeichen	0	0

Selbstaktivierung DCD-/CTS-Pins ja=1 / nein=0

Hunt-Mode aktiv=1 / inaktiv=0

CRC-Berechnung Empfänger aktiv=1 / inaktiv=0

SDLC-Adressensuchmode aktiv=1 / inaktiv=0

SYNC-Zeichen-Ladeverhinderung aktiv=1 / inaktiv=0

Empfänger aktiv=1 / inaktiv=0

Schreibregister 4

D7	D6	D5	D4	D3	D2	D1	D0

Taktrate (Clock) X64 Datenrate	1	1				
Taktrate (Clock) X32 Datenrate	1	0				
Taktrate (Clock) X16 Datenrate	0	1				
Taktrate (Clock) 1 Datenrate	0	0				
externer SYNC-Mode			1	1		
SDLC-Flag 01111110			1	0		
16 Bit SYNC-Zeichenmode			0	1		
8 Bit SYNC-Zeichenmode			0	0		
2 Stop Bit pro Zeichen					1	1
1 1/2 Stop Bit pro Zeichen					1	0
1 Stop Bit pro Zeichen					0	1
SYNC-Mode-Aktivierung					0	0

Paritätsüberwachung gerade=1 / ungerade=0

Paritätsüberwachung aktiv=1 / inaktiv=0

Schreibregister 5

D7	D6	D5	D4	D3	D2	D1	D0

DTR-Signalpin (data terminal ready) Low=1 / High=0

8 Bits pro Sendezeichen	1	1
6 Bits pro Sendezeichen	1	0
7 Bits pro Sendezeichen	0	1
5 Bits - oder weniger - pro Sendezeichen	0	0

D7	D6	D5	D4	D3	D2	D1	D0	
1	1	1	1	0	0	0	D	Senden von 1 Datenbit
1	1	1	0	0	0	D	D	Senden von 2 Datenbits
1	1	0	0	0	D	D	D	Senden von 3 Datenbits
1	0	0	0	D	D	D	D	Senden von 4 Datenbits
0	0	0	D	D	D	D	D	Senden von 5 Datenbits

Send-Break-Zeichenfolge aktiv=1 / inaktiv=0

Datensender aktiv=1 / inaktiv=0

CRC-Polynomberechnung Sender/Empfänger CRC-16=1 / SDLC=0

RTS-Signalpin (request to send) Low=1 / High=0

CRC-Polynomberechnung Sender aktiv=1 / inaktiv=0

Schreibregister 6

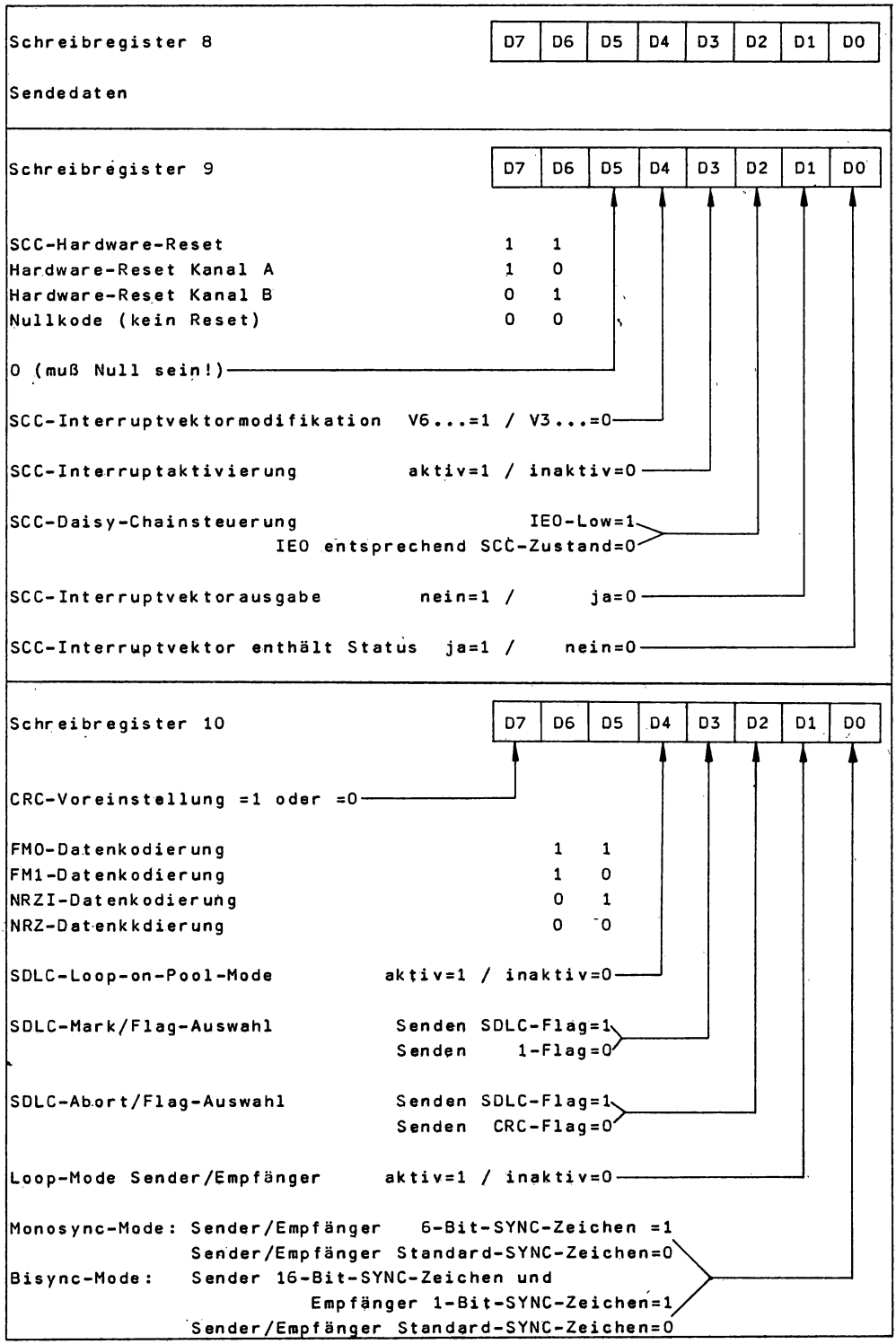
D7	D6	D5	D4	D3	D2	D1	D0

Monosync 8 Bit	Sync7	Sync6	Sync5	Sync4	Sync3	Sync2	Sync1	Sync0
Monosync 6 Bit	Sync5	Sync4	Sync3	Sync2	Sync1	Sync0	x	x
Bisync 16 Bit	Sync15	Sync14	Sync13	Sync12	Sync11	Sync10	Sync9	Sync8
Bisync 12 Bit	Sync11	Sync10	Sync9	Sync8	Sync7	Sync6	Sync5	Sync4
SDLC	0	1	1	1	1	1	1	0

Schreibregister 7

D7	D6	D5	D4	D3	D2	D1	D0

Monosync 8 Bit	Sync7	Sync6	Sync5	Sync4	Sync3	Sync2	Sync1	Sync0
Monosync 6 Bit	Sync1	Sync0	Sync5	Sync4	Sync3	Sync2	Sync1	Sync0
Bisync 16 Bit	Sync7	Sync6	Sync5	Sync4	Sync3	Sync2	Sync1	Sync0
Bisync 12 Bit	Sync3	Sync2	Sync1	Sync0	1	1	1	1
SDLC	ADR7	ADR6	ADR5	ADR4	ADR3	ADR2	ADR1	ADR0
SDLC (Adr. Range)	ADR7	ADR6	ADR5	ADR4	x	x	x	x


Schreibregister 8
D7 D6 D5 D4 D3 D2 D1 D0
Sendedaten
Schreibregister 9
D7 D6 D5 D4 D3 D2 D1 D0
SCC-Hardware-Reset 1 1
Hardware-Reset Kanal A 1 0
Hardware-Reset Kanal B 0 1
Nullkode (kein Reset) 0 0
0 (muß Null sein!)
SCC-Interruptvektormodifikation V6...=1 / V3...=0
SCC-Interruptaktivierung aktiv=1 / inaktiv=0
SCC-Daisy-Chainsteuerung IEO-Low=1
IEO entsprechend SCC-Zustand=0
SCC-Interruptvektorausgabe nein=1 / ja=0
SCC-Interruptvektor enthält Status ja=1 / nein=0
Schreibregister 10
D7 D6 D5 D4 D3 D2 D1 D0
CRC-Voreinstellung =1 oder =0
FM0-Datenkodierung 1 1
FM1-Datenkodierung 1 0
NRZI-Datenkodierung 0 1
NRZ-Datenkkdierung 0 0
SDLC-Loop-on-Pool-Mode aktiv=1 / inaktiv=0
SDLC-Mark/Flag-Auswahl Senden SDLC-Flag=1
Senden 1-Flag=0
SDLC-Abort/Flag-Auswahl Senden SDLC-Flag=1
Senden CRC-Flag=0
Loop-Mode Sender/Empfänger aktiv=1 / inaktiv=0
Monosync-Mode: Sender/Empfänger 6-Bit-SYNC-Zeichen =1
Sender/Empfänger Standard-SYNC-Zeichen=0
Bisync-Mode: Sender 16-Bit-SYNC-Zeichen und
Empfänger 1-Bit-SYNC-Zeichen=1
Sender/Empfänger Standard-SYNC-Zeichen=0

Schreibregister 11

D7	D6	D5	D4	D3	D2	D1	D0

Empfängertakteingang RTxC
Quarz (RTxC - SYNC)=1
TTL-Signalpegel=0

Empfängertakteingang = DPLL-Ausgang	1	1
Empfängertakteingang = BR-Generator-Ausgang	1	0
Empfängertakteingang = TRxC-Pin	0	1
Empfängertakteingang = RTxC-Pin	0	0

Sendertakteingang = DPLL-Ausgang	1	1
Sendertakteingang = BR-GeneratorAusgang	1	0
Sendertakteingang = TRxC-Pin	0	1
Sendertakteingang = RTxC-Pin	0	0

TRxC-Signalpin Ausgang=1 / Eingang=0

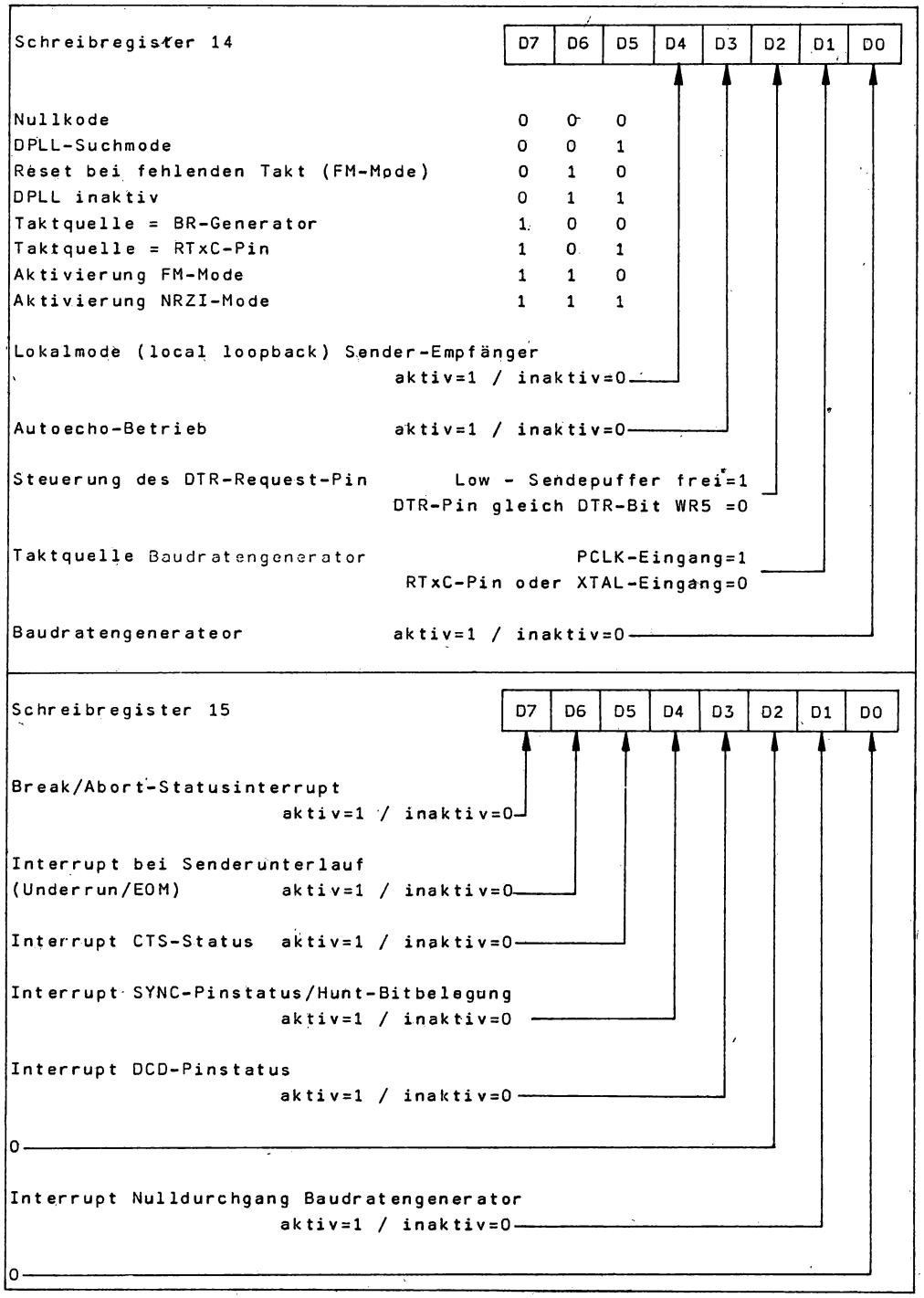
TRxC-Ausgang ist DPLL-Ausgang	1	1
TRxC-Ausgang ist BR-Generator-Ausgang	1	0
TRxC-Ausgang ist Sendertaktausgang	0	1
TRxC-Ausgang ist XTAL-Ausgang	0	0

Schreibregister 12 und 13

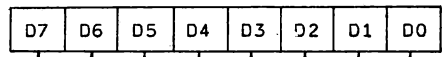
Zeitkonstantenregister	TC15, TC14 ... TC8,	TC7, TC6 ... TC0
16-Bit-Zähler	WR13	WR12

Zeitkonstante = (Taktfrequenz (2 * (Baudrate)) - 2
Baudrate (Taktfrequenz (2 * (Zeitkonstante + 2))

Beispiel mit 3.9936 MHz Takt: Baudrate	Zeitkonstante		Fehler
19200	102	(%0066)	0
9600	206	(%00CE)	0
4800	414	(%019E)	0
3600	553	(%0229)	.06%
2400	830	(%033E)	0
2000	996	(%03E4)	.04%
1800	1107	(%0453)	.03%
1200	1662	(%067E)	0
600	3326	(%0CFE)	0
300	6654	(%19FE)	0
150	13310	(%33FE)	0
110	18151	(%46E7)	.0015%
75	26622	(%67FE)	0
50	39934	(%9BFE)	0


Schreibregister 14
D7 D6 D5 D4 D3 D2 D1 D0
Nullkode 0 0 0
DPLL-Suchmode 0 0 1
Reset bei fehlenden Takt (FM-Mode) 0 1 0
DPLL inaktiv 0 1 1
Taktquelle = BR-Generator 1 0 0
Taktquelle = RTxC-Pin 1 0 1
Aktivierung FM-Mode 1 1 0
Aktivierung NRZI-Mode 1 1 1
Lokalmode (local loopback) Sender-Empfänger
aktiv=1 / inaktiv=0
Autoecho-Betrieb
aktiv=1 / inaktiv=0
Steuerung des DTR-Request-Pin
Low - Sendepuffer frei=1
DTR-Pin gleich DTR-Bit WR5 =0
Taktquelle Baudratengenerator
PCLK-Eingang=1
RTxC-Pin oder XTAL-Eingang=0
Baudratengenerateor
aktiv=1 / inaktiv=0
Schreibregister 15
D7 D6 D5 D4 D3 D2 D1 D0
Break/Abort-Statusinterrupt
aktiv=1 / inaktiv=0
Interrupt bei Senderunterlauf
(Underrun/EOM)
aktiv=1 / inaktiv=0
Interrupt CTS-Status
aktiv=1 / inaktiv=0
Interrupt SYNC-Pinstatus/Hunt-Bitbelegung
aktiv=1 / inaktiv=0
Interrupt DCD-Pinstatus
aktiv=1 / inaktiv=0
0
Interrupt Nulldurchgang Baudratengenerator
aktiv=1 / inaktiv=0
0

Leseregister 0



Indikator (=1) Breaksequenz-/Abortsequenz

Indikator (=1) Senderunterlauf (Underrun/EOM)

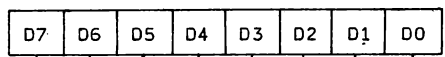
Indikator (=1) CTS-Pin
Indikator (=1) SYNC-Pin/Hunt-Bit
Indikator (=1) DCD-Pin

Indikator (=1) Sendepuffer frei

Indikator (=1) Nulldurchgang Baudratengenerator

Indikator (=1) empfangenes Zeichen verfügbar

Leseregister 1



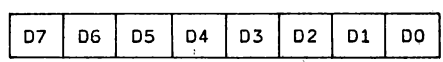
Indikator (=1) SDLC-End-of-Frame

Indikator (=1) Fehler: CRC-Framing-Error
Indikator (=1) Fehler: Senderüberlauf
Indikator (=1) Fehler: Parität

Indikator (=1) SDLC-Residuenkode 0
Indikator (=1) SDLC-Residuenkode 1
Indikator (=1) SDLC-Residuenkode 2

Indikator (=1) Sendedaten abgesetzt

Leseregister 2

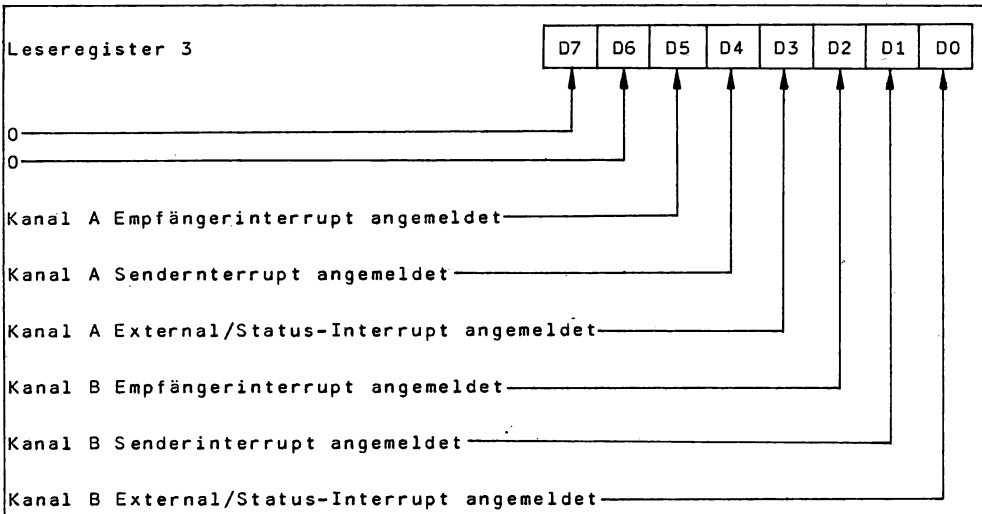


Interruptvektor V7, V6 ... V0

Interruptvektormodifikation

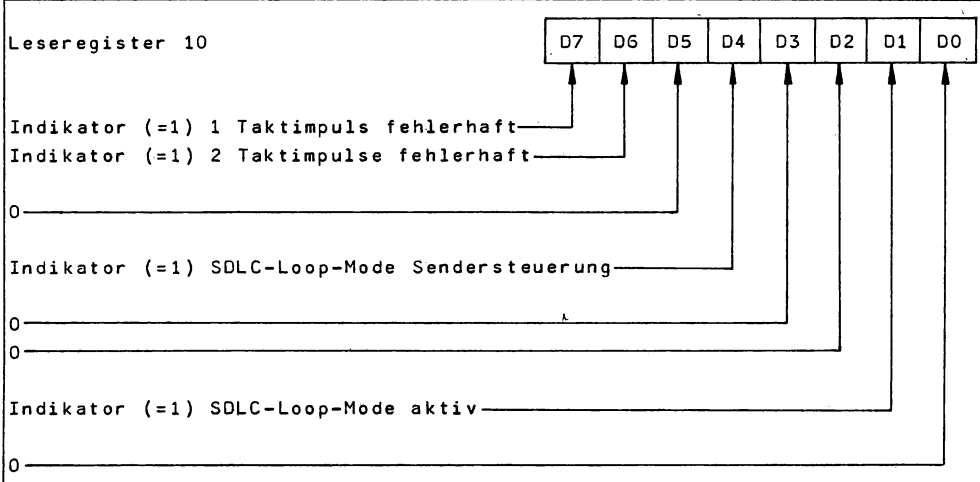
V3	V2	V1	Interruptvektormodafikation = 0 (WR9)
V4	V5	V6	Interruptvektormodifikation = 1 (WR9)
0	0	0	Kanal B Sendepuffer leer
0	0	1	Kanal B External/Status
0	1	0	Kanal B Empfangsdaten verfügbar
0	1	1	Kanal B spezielle Empfangsbedingung
1	0	0	Kanal A Sendepuffer leer
1	0	1	Kanal A External/Status
1	1	0	Kanal A Empfangsdaten verfügbar
1	1	1	Kanal A spezielle Empfangsbedingung

Leseregister 3



Anmerkung: Für jeden angemeldeten Interrupt, wird in das Leseregister RR3 eine 1 eingetragen.

Leseregister 10



Leseregister 12 und 13

Zeitkonstantenregister | TC15, TC14 ... TC8, | TC7, TC6 ... TC0 |

Leseregister 15

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |

Das Leseregister RR15 ist ein direktes Rückleseregister des Schreibregisters **WR15.**

Tafel 3.11. Registerzustand des U8x30-SCC nach einem Hardware-Systemreset in Verbindung mit einer Standard-Initialisierungssrequenz

	Register	Daten	Kommentar
MODE			
	WR9	11000000	Hardware-Reset
	WR0	000000XX	Auswahl Shift-Mode (nur U8030-SCC)
	WR4	XXXXXXXX	Sender/Empfängersteuerung / Async-Sync
	WR1	0XX00X00	Auswahl W/REQ (optional)
	WR2	XXXXXXXX	Interruptvektor
	WR3	XXXXXXX0	Auswahl Empfängersteuerung
	WR5	XXXX0XXX	Auswahl Sendersteuerung
	WR6	XXXXXXXX	Sync-Zeichen (optional)
	WR7	XXXXXXXX	Sync-Zeichen (optional)
	WR9	000X0XXX	Auswahl Interruptsteuerung
	WR10	XXXXXXXX	verschiedene Steuerfunktionen (optional)
	WR11	XXXXXXXX	Taktsteuerung
	WR12	XXXXXXXX	Zeitkonstante Low-Byte
	WR13	XXXXXXXX	Zeitkonstante High-Byte
	WR14	XXXXXXX0	verschiedene Steuerfunktionen
	WR14	XXXSSSSS	verschiedene Steuerfunktionen (optional)
AKTIVIERUNGEN			
	WR3	SSSSSSS1	Setzen von D0 (Rx aktiv)
	WR5	SSSS1SSS	Setzen von D3 (Tx aktiv)
	WR0	10000000	Rücksetzen von TxCRC
	WR14	000SSSS1	BR-Generator aktiv
	WR1	XSS00S00	Setzen D7 (DMA Aktivierung optional)
INTERRUPT			
	WR15	XXXXXXXX	Aktivierung externe Interrupts
	WR0	00010000	Rücksetzen External/Status
	WR1	SSSXXSXX	Aktivierung Interrupts
	WR9	000SXSSS	Aktivierung Masterinterruptbit D3

X Bitbelegung neu festlegen
S Bitbelegung unverändert lassen

Hardware-Reset								Kanal-Reset								
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	WR0
0	0	.	0	0	.	0	0	0	0	.	0	0	.	0	0	WR1
.							.								.	WR2
.					.	.	0						.	.	0	WR3
.			.	.	1	.		.			.	.	1	.		WR4
0		.	0	0	0	0		0		.	0	0	0	0		WR5
.																WR6
.		.		.	.					.						WR7
1	1	0	0	0	0	.	.		.	0	.	.	.	.	.	WR9
0	0	0	0	0	0	0	0	0		.	0	0	0	0	0	WR10
0	0	0	0	1	0	0	0									WR11
.																WR12
.		.	.	.	.	.	.			.		.	.			WR13
.	.	1	0	0	0	0	0	.	.	1	0	0	0	.	.	WR14
1	1	1	1	1	0	0	0	1	1	1	1	1	0	0	0	WR15
0	1	.	.	.	1	0	0	0	1	.	.	.	1	0	0	RR0
0	0	0	0	0	1	1	1	0	0	0	0	0	1	1	1	RR1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	RR3
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	RR10

4. Einchipmikrorechner U 881/U 882-EMR

Die Grundstruktur eines Mikrorechners ist im Bild 2.1. (Seite 12) wiedergegeben. Sie besteht aus vier Basisfunktionsblöcken - der zentralen Verarbeitungseinheit (CPU), dem Programmspeicher, dem Operativdatenspeicher und aus den Ein-/Ausgabekanälen. Die zentrale Verarbeitungseinheit übernimmt die Systemsteuerung und die Befehlsabarbeitung. Der Programmspeicher enthält die Maschinenbefehle des Mikrorechners, die Schritt für Schritt von der zentralen Verarbeitungseinheit aufgerufen und abgearbeitet werden. Im Operativdatenspeicher werden Daten, die bei der Programmabarbeitung anfallen, zwischengespeichert bzw. abgelegt. Über die Ein-/Ausgabekanäle eines Mikrorechners wird die Kommunikation mit der Umwelt sichergestellt. In den behandelten U880- und U8000-Mikroprozessorsystemen (Abschnitte 2. und 3.) werden diese Funktionsblöcke durch Zusammenschaltung unterschiedlicher Bauelemente realisiert. Bei dem Einchipmikrorechner U881/U882 sind alle vier Basisfunktionen (beim U882 drei Basisfunktionen) eines Mikrorechners

zentrale Verarbeitungseinheit	(CPU)
Programmspeicher (U882 externer EPROM)	(ROM)
Operativdatenspeicher	(RAM)
Ein-/Ausgabekanäle	(PIO, UART, CTC)

auf einem einzigen Siliziumplättchen (Chip) von wenigen Quadratmillimetern Grundfläche angeordnet (Bild 4.1.) /26/,/27/,/28/,/29/.

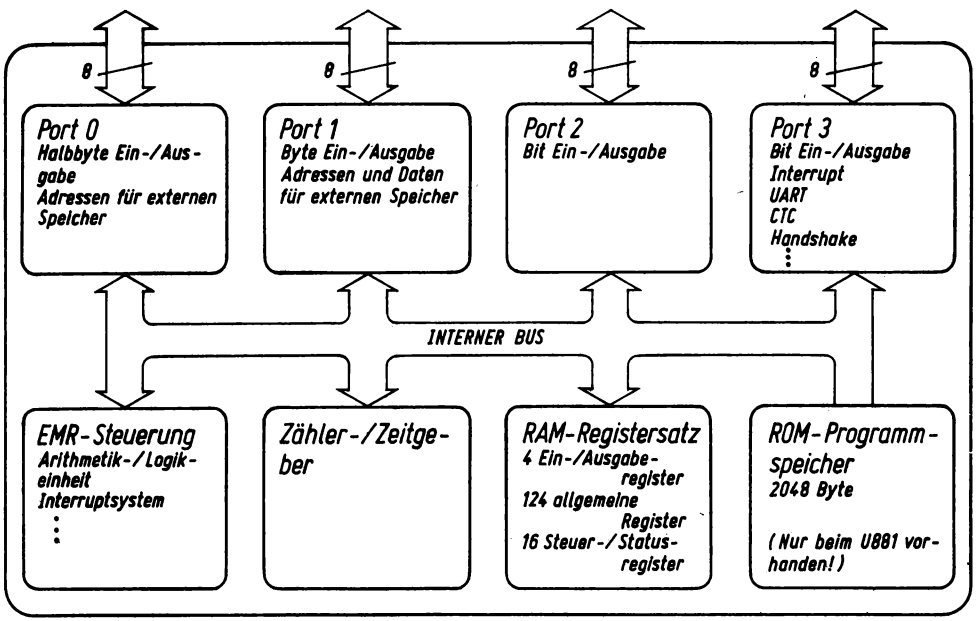


Bild 4.1. Interne Funktionskomponenten des Einchipmikrorechners U881/U882

Der U881, ein Schaltkreis mit 40 Signalanschlüssen (Pins), ist für den Einsatz in Geräten mit sehr großen Stückzahlen und unveränderlichem Programm vorgesehen. Der vom Anwender gewünschte Programmspeicherinhalt wird beim Bauelementeproduzenten /26/ fest und für alle Zeiten unveränderlich beim Herstellungsprozeß der Schaltkreise definiert (ROM-Maskenprogrammierung). Der U882 (64 Signalanschlüsse) besitzt keinen internen Programmspeicher. Die zusätzlichen Pins ermöglichen jedoch an Stelle des internen ROM den Anschluß eines externen Programmspeichers (EPROM). Die interne Struktur der beiden Schaltkreise U881 und U882 ist bis auf den beim U882 fehlenden internen Programmspeicher vollkommen identisch.
Bezogen auf die interne Struktur gilt das auch für eine dritte Variante des Einchipmikrorechners, den U883 (40 Pins), dessen interner ROM-Programmspeicher mit einem vom Anwender frei nutzbaren BASIC-Interpreter und Bootstraplader vorprogrammiert wurde.
Es ist hier anzumerken, daß der Einchipmikrorechner U881/U882 so konzipiert ist, daß er entweder als ein-/ausgabeintensiver oder als speicherintensiver Mikrorechner arbeiten kann. Das wird erreicht durch eine programmierbare Struktur der Ein-/Ausgabekanäle, die es ermöglicht, daß einige der Ein-/Ausgabekanalleitungen in einen im Multiplexverfahren betriebenen Adreßbus und Datenbus umgewandelt werden. Dadurch wird die Verwendung zusätzlicher externer Programm- und Operativdatenspeicherbereiche beim U881 und beim U882 möglich (s.a. Abschnitte 4.1.2. und 4.2.).

4.1. Zentrale Verarbeitungseinheit

4.1.1. Aufbau

Die zentrale Verarbeitungseinheit (CPU) des U881/U882 besteht im wesentlichen aus der Steuerung des Einchipmikrorechners, aus der Arithmetik-Logik-Einheit ALU (arithmetic logic unit) und aus einer ganzen Reihe von Einzelkomponenten, wie z.B. dem Vektorinterruptsystem, dem Befehlszähler PC und dem Befehlsdekoder IR (IR instruction register). Die Arithmetik-/Logik-Einheit realisiert die Befehlsausführung der 112 Basisinstruktionen. Sie ermöglicht die Behandlung von verschiedenen Datentypen (einzelne Bits in einem Byte, 4-Bit-BCD-Worte in einem Byte, Bytes und 16-Bit-Worte). Die EMR-Steuerung übernimmt die Steuerung und Überwachung des gesamten Geschehens im Einchipmikrorechner sowie die Signalgenerierung entsprechend den gegebenen Zeitabläufen für die Kommunikation mit der umgebenden Umwelt. Die interne Taktfrequenz liegt abhängig von der vorliegenden Bauelementeversion, zwischen 0,5 MHz und 4 MHz.

4.1.2. Adreßraum

Der interne Programmspeicher des Einchipmikrorechners umfaßt 2048 Byte (2 KByte). Er ist nur beim U881 aus wirklich internen, hier maskenprogrammierten ROM-Speicherzellen ausgebildet. Bei der U882-Variante des Ein-Chip-Mikroechners wird der 'interne' Speicherbereich durch einen externen Speicher, in der Regel ein EPROM-Bauelement (2K x 8 EPROM), ersetzt.
Es ist jedoch zu beachten, daß der Adreßraum des Einchipmikrorechners U881/U882 wesentlich über die 2 KByte internen Programmspeicher (ROM oder EPROM) hinausgeht. Durch entsprechende Maßnahmen kann ein externer Programmspeicherbereich von 62 KByte und zuzüglich ein ausschließlich Datenstrukturen zugeordneter externer Datenspeicherbereich von nochmals 62

KByte an den EMR-Schaltkreis über die beiden Ein-/Ausgabekanäle Port 0 und Port 1 angeschlossen werden (Bild 4.2.).

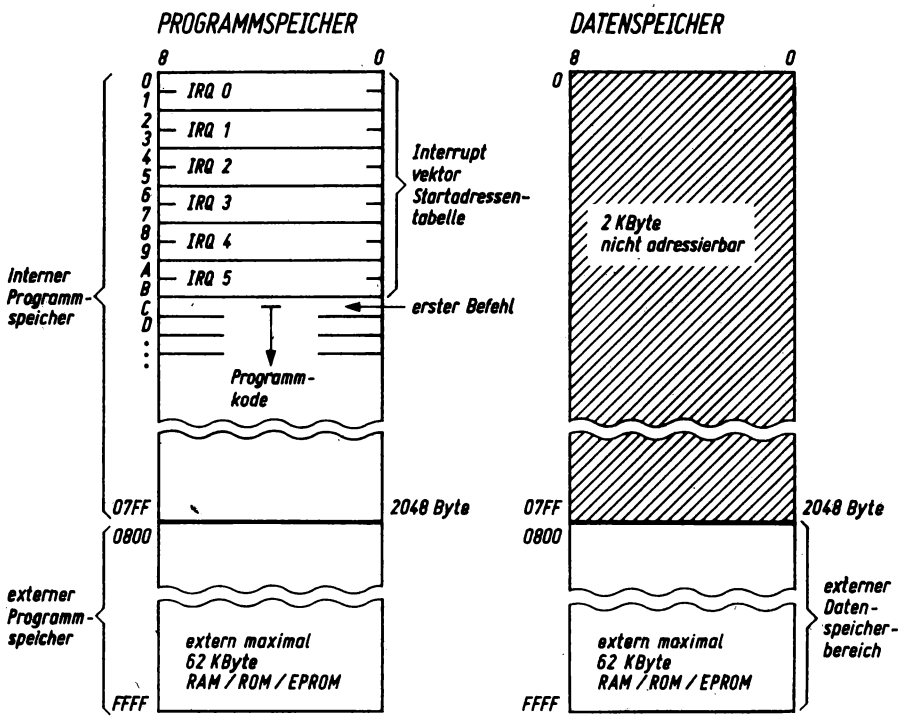


Bild 4.2. Struktur des internen und externen Programmspeichers und des externen Datenspeicherbereiches beim Einchipmikrorechner U881/U882

4.1.3. Registersatz

Der aus 144 Byte bestehende Registersatz des U881/U882 untergliedert sich in zwei Teilkomponenten, den allgemeinen Registersatz (128 Byte) und den Steuer- und Statusregistersatz (16 Byte).
Die 128 Byte des allgemeinen Registersatzes bestehen aus acht Gruppen zu je 16 Byte. Die unterste dieser Gruppen enthält auch die vier bei der Ein-/Ausgabe Verwendung findenden Ein-/Ausgabekanalregister R0, R1, R2 und R3 (Ein-/Ausgabeports) (Bild 4.3.). Sie können mit den gleichen Befehlen gelesen und geschrieben sowie arithmetisch und logisch verknüpft werden wie alle anderen Register. Spezielle Ein-/Ausgabebefehle zur Kommunikation mit der Umwelt wie beim U880 und U8000 (IN, OUT) existieren beim U881/U882 nicht. Die Einführung eines dem Programmierer zugeordneten Registerpointers (er zeigt auf eine der insgesamt neun 16 Byte Registergruppen der allgemeinen Register und der Steuer- und Statusregister) ermöglicht den

besonders schnellen und speicherplatzsparenden Zugriff innerhalb der aktuellen 16 Byte Gruppe. Diese vom Registerpointer ausgewählte Gruppe wird als Arbeitsregistergruppe bezeichnet.
Neben den Steuer- und Statusregistern existiert im U881/U882 ein 16-Bit-Befehlszählerregister PC (program counter).

4.1.4. Steuerregister/Statusregister

Die Steuer- und Statusregister (16 Byte) des U881/U882 dienen der Festlegung der Betriebsarten des Einchipmikrorechners und zur Vermittlung des EMR-Status an das laufende Programm. Das Lesen und Schreiben dieses Teiles des Registersatzes erfolgt in der gleichen Weise wie bei den allgemeinen Registern, einschließlich der Adressierungsmöglichkeit über den Registerpointer. Dabei ist zu beachten, daß ein Teil der Register (R243, R245, R246, R247, R248) nur geschrieben nicht aber gelesen werden kann (write only). Die Steuer- und Statusregistergruppe besteht aus folgenden Einzelregistern (Bild 4.3.):

Stackpointer R255(0FFH) SPL (Low-Teil) und R254(0FEH) SPH (High-Teil) - Der Stackpionter des U881/U882 enthält, wie beim U880 und U8000, die Adresse des im Speicher angeordneten Stackspeicherbereichs. Er dient zur Aufnahme der Rückkehradressen bei Unterprogrammaufrufen und Interruptserviceroutinen, kann allerdings mit Hilfe der Befehle PUSH und POP auch zur Aufbewahrung verschiedener bei der Programmabarbeitung anfallender Operativdaten, zur Parameterübergabe bei Unterprogrammen u.a.m. benutzt werden. Beim U881/U882 kann entweder mit einem internen Stack (innerhalb des allgemeinen Registersatzes) oder mit einem externen Stack (im externen Datenspeicherbereich) gearbeitet werden. Dementsprechend ist ein 8-Bit-Stackpointer (nur R255) ausreichend (R254 ist dann frei verfügbar) oder es muß ein 16-Bit-Stackpointer (R255 und R254) benutzt werden. Die Festlegung, ob ein interner oder ein externer Stack verwendet wird, erfolgt über das Datenbit D2 im Register R248 (Port 0 / Port 1 Moderegister).

Registerpionter R253(0FDH) RP - Der Registerpointer dient zur Auswahl einer als Arbeitsregister bezeichneten Gruppe von 16 Einzelregistern aus dem Gesamtregistersatz des Einchipmikrorechners. Der Inhalt der oberen vier Bit des Registerpointers zeigt bei diesem Arbeitsregisteradressierungsverfahren auf das erste Register jeweils einer Befehlsgruppe. Die unteren vier Bit werden als interne 4-Bit-Registeradresse dem aktuellen Befehlskode entnommen. Zusammen entsteht aus zwei 4-Bit-Gruppen eine 8-Bit-Adresse mit 256 Einzeladressen, genau den Registeradreßraum der U881/U882 überstreichend. Solange sich der Programmierer innerhalb einer 16 Byte umfassenden Arbeitsregistergruppe bewegt, bleibt der Registerpointer unverändert und es kann mit den ein Byte kürzeren, nur auf die jeweilige Arbeitsregistergruppe bezogenen Befehlen gearbeitet werden. Der Übergang von einer Arbeitsregistergruppe in eine andere erfolgt durch Umladen des Registerpointers (Bild 4.3.).

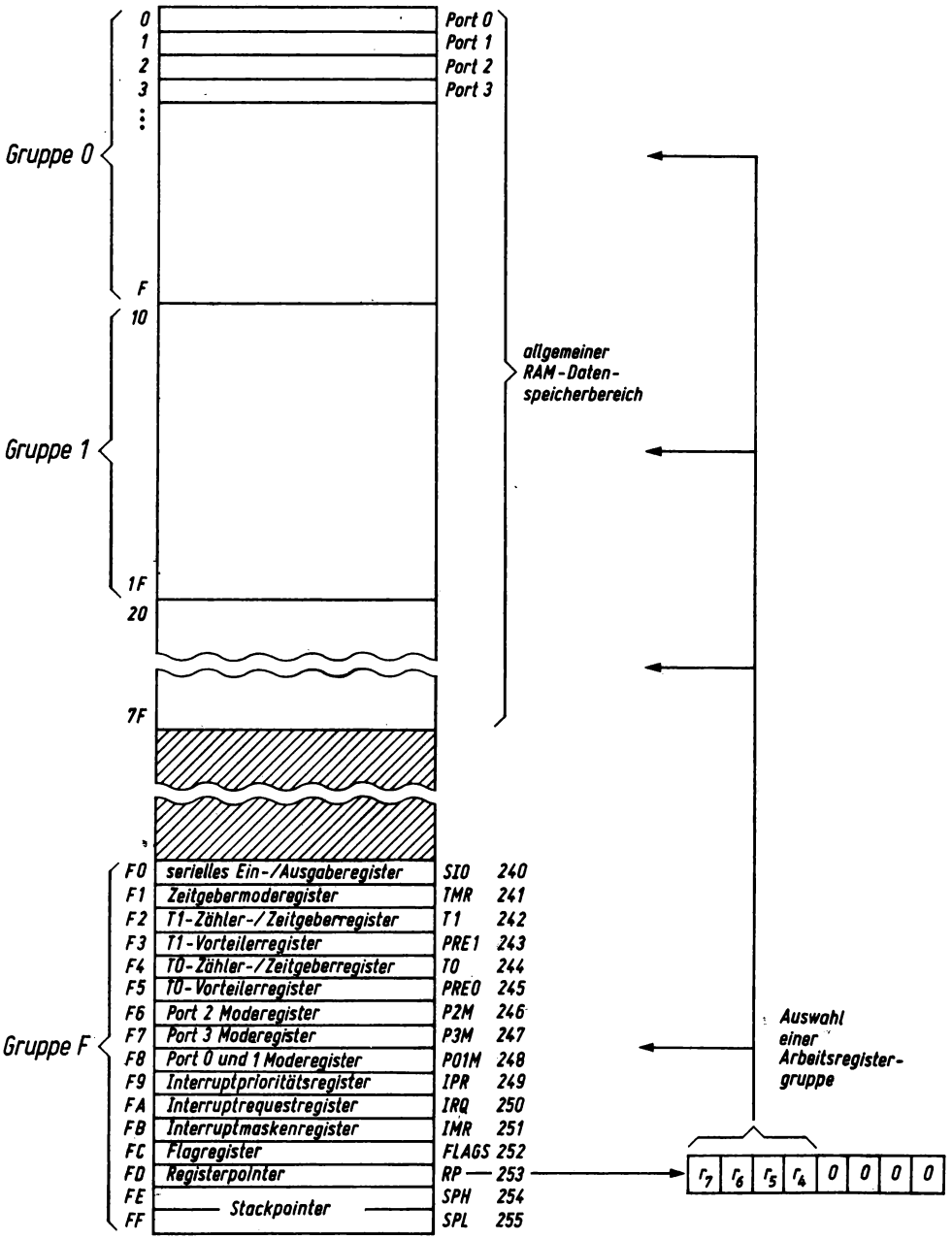


Bild 4.3. Aufbau des U881/U882-EMR Registersatzes

Flagregister R252(OFCH) FLAGS - Der Inhalt des Registers R252 gibt Auskunft über verschiedene im Ergebnis der Befehlsausführung im U881/U882 eingetretene Statusmodifikationen und dient zur bedingungsabhängigen Gestaltung des Programmablaufes. Die Funktionen der Flagbits C, Z, S, V, D und H entsprechen bei der EMR-Befehlsabarbeitung weitgehend der Funktion der äquivalenten U880-Flags (s.a. Abschn. 2.1.3. sowie EMR-Befehlsliste Tafel 4.1.). Zwei Flags F1 und F2 sind vollkommen frei vom Programmierer verwendbar (alle Bits des Flagregisters können sowohl gelesen als auch geschrieben werden).
Im einzelnen hat das Flagregister folgende Belegung:

C	Z	S	V	D	H	F2	F1

C-Flag Übertragsflag (carry)
Z-Flag Nullflag (zero)
S-Flag Vorzeichenflag (sign)
V-Flag Überlaufsflag (overflow)
D-Flag BCD-Normalisierungsflag (decimal adjust)
H-Flag Halbübertragsflag (half carry)
F2-Flag Anwenderflag 2 (user flag 2)
F1-Flag Anwenderflag 1 (user flag 1)

Interruptmaskenregister R251(OFBH) IMR - Das Interruptmaskenregister ermöglicht dem Programmierer, das Interruptsystem des U881/U882 insgesamt oder selektiv (sechs Einzelinterruptquellen) ein- bzw. auszuschalten. Dem Ein- und Ausschalten des gesamten EMR-Interruptsystems ist das Datenbit D7 zugeordnet. Es wird mit einem speziellen Befehl gesetzt EI (=1 / enable interrupt) und rückgesetzt DI (=0 / disable interrupt). Gesetzt wird dieses Flagbit außerdem automatisch bei der Abarbeitung des IRET-Befehls (return from interrupt) und rückgesetzt im Interruptanerkennungszyklus.

EI	0/1	IRQ5	IRQ4	IRQ3	IRQ2	IRQ1	IRQ0

Zu beachten ist, daß das Datenbit D7 im Interruptmaskenregister nach dem Einschalten des U881/U882 (Systemreset) mit dem DI-Befehl rückgesetzt werden muß, bevor es selbst und das in der Folge noch zu beschreibende Interruptprioritätsregister erstmalig beschrieben werden kann. Die Datenbits D0 bis D5 des Interruptmaskenregisters sind den sechs EMR-Interruptquellen IRQ0 bis IRQ5 zugeordnet. Werden die entsprechenden Datenbits gesetzt (=1), sind die korrespondierenden Einzelinterruptsignale zugelassen - werden sie rückgesetzt (=0), sind sie gesperrt. (D6 im IMR wird nicht benutzt.)

Interruptrequestregister R250(OFAH) IRQ - Das Interruptrequestregister speichert eine anstehende Interruptanforderung bis zu ihrer Abarbeitung. Trifft eine Einzelinterruptanforderung ein, wird das korrespondierende Datenbit D0...D5 entsprechend IRQ0...IRQ5 auf den Wert Eins gesetzt. Nach

seiner Akzeptierung im Interruptanerkennungszyklus (interrupt machine cycle) wird es automatisch rückgesetzt.

0/1	0/1	IRQ5	IRQ4	IRQ3	IRQ2	IRQ1	IRQ0

Da es sich beim IRQ-Register um ein Schreib-/Leseregister handelt, in dem auch vom Programmierer einzelne Datenbits gesetzt und rückgesetzt werden können, ergibt sich, daß durch Lesen dieses Registers ermittelt werden kann, ob einzelne Interruptanforderungen eingegangen sind oder nicht (polling). Durch Schreiben des IRQ-Registers kann der Programmierer Einzelinterruptsignale simulieren, in dem er das entsprechende Datenbit im Interruptrequestregister auf Eins setzt. (D7 und D6 im ITQ werden nicht benutzt.)

Interruptprioritätsregister R249(0F9H) IPR - Das Interruptprioritätsregister hat die Aufgabe, bei gleichzeitigem Eintreffen oder bei gleichzeitigem Anstehen (nach einer Interruptsperre) von mehreren Einzelinterruptsignalen zu entscheiden, welche der verschiedenen Interruptanforderungen zuerst bedient werden soll. Mit Hilfe dieses Registers können die sechs Einzelinterrupts in 48 verschiedene Prioritätsverteilungsarten aufgelöst werden. Die Interruptprioritätsermittlung über das Register R249 ist nur bei gleichzeitigem Anstehen mehrerer Interruptanforderungen aktiv, die prioritätsgesteuerte Verschachtelung (nesting) der Interruptbedienung wie beim U880 und U8000 über eine Prioritätskaskade (daisy chain) wird vom U881/U882 hardwaremäßig nicht unterstützt. (D7 und D6 im IPR werden nicht benutzt.)
Prioritätsfestlegung im IPR (A, B und C sind Prioritätsgruppen):

```
D4,D3,D0=000  ---                      Gruppe A:  D1=0  IRQ1 größer IRQ4
        =001  C größer A größer B                 =1  IRQ4 größer IRQ1
        =010  A größer B größer C      Gruppe B:  D2=0  IRQ2 größer IRQ0
        =011  A größer C größer B                 =1  IRQ0 größer IRQ2
        =100  B größer C größer A      Gruppe C:  D5=0  IRQ5 größer IRQ3
        =101  C größer B größer A                 =1  IRQ3 größer IRQ5
        =110  B größer A größer C
        =111  ----
```

(Steuer- und Statusregister R240, R241 ...R248 s. Abschnitt 4.2. Ein-/Ausgabe.)

4.1.5. Interruptsystem

Das Vektorinterruptsystem des U881/U882 erlaubt die Behandlung von sechs verschiedenen Interrupts aus acht unterschiedlichen Quellen (Bilder 4.4. und 4.5.).
Vier Interruptquellen (IRQ0...IRQ3) sind den vier Eingangssignalen des Ein-/Ausgabeports 3 zugeordnet. Sie können wahlweise den Eingangsbits dieses Ports (Flanke 'high' auf 'low'), den Quittungsbetriebssignalen (handshake) DAV0/RDY0, DAV1/RDY1 und DAV2/RDY2 vom Port 0, 1 oder 2 oder aber dem internen CTC- bzw. dem UART-Eingangssignal des Einchipmikrorechners zugeordnet werden. Die beiden Interruptquellen IRQ4 und IRQ5

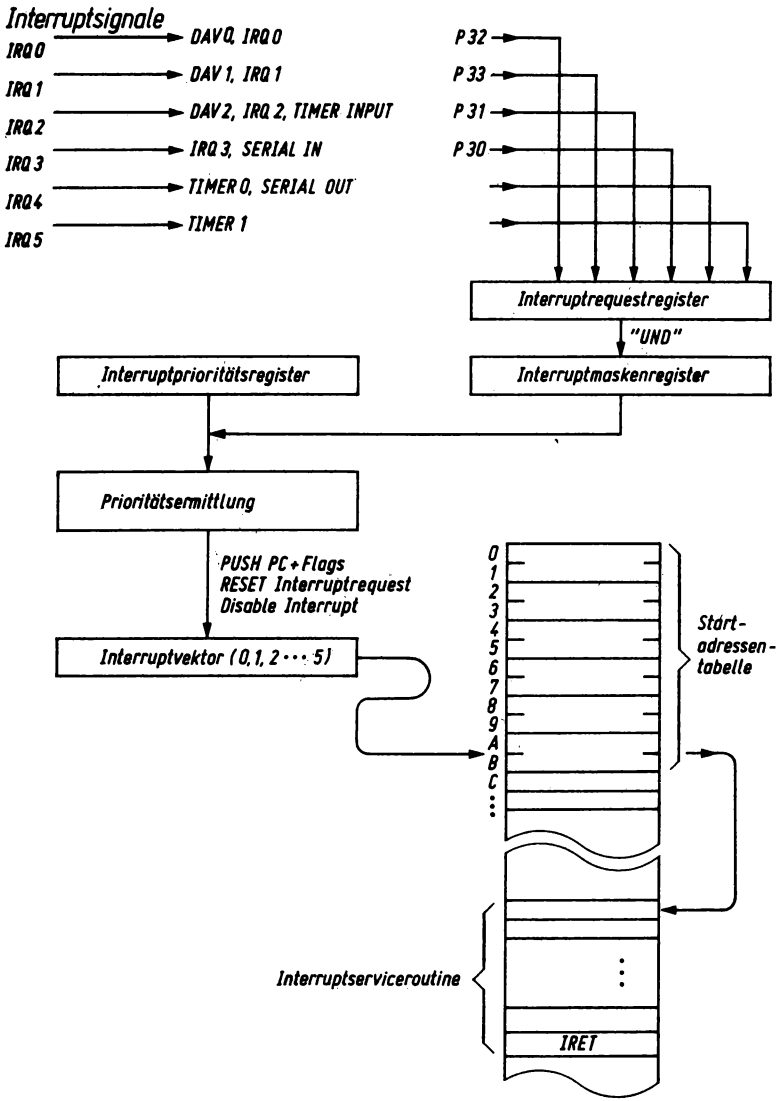


Bild 4.4. Struktur des U881/U882 Vektorinterruptsystems

entsprechen wahlweise den beiden CTC-Ausgangssignalen (T0 und T1) oder dem CTC-Ausgangssignal (T1) und dem UART-Ausgangssignal (UART universal asynchronous receiver transmitter / s. Abschnitt 4.2.). Die Programmierung welche Interruptquelle für ein bestimmtes Bit anzusetzen ist, erfolgt mit der Programmierung des Ein-/Ausgabeports 3 (Register R247) und mit der Programmierung der beiden CTC-Kanäle (Register R241, R242, R243 und R244). Der Ablauf einer Interruptbehandlung beim U881/U882 ist dem Bild 4.4. zu entnehmen, dabei ist der Einfluß der Bitbelegung des Interruptmaskenregisters (R249), des Interruptrequestregisters (R250) und des Interruptprioritätsregisters (R249) auf die Interruptbearbeitung zu beachten.
Zu beachten ist außerdem, daß bei einem Interrupt (ähnlich wie beim U8000 aber im Gegensatz zum U880) nicht nur der Befehlszählerinhalt, sondern auch der Inhalt des Flagregisters R252 automatisch in den Stack eingespeichert wird - zusammen drei Byte (s.a. IRET-Befehl Tafel 4.1.).

4.1.6. Adressierungsarten

Der Befehlssatz des Einchipmikrorechners U881/U882 beinhaltet fünf Adressierungsarten:

Registeradressierung	Der Operand ist der Inhalt eines Registers.
Indirekte Registeradressierung	Der Operand ist der Inhalt einer Adresse, die durch den Inhalt eines Registers ausgewählt wird.
Direktwert Adressierung	Der Operand ist Teil des Befehls.
Indizierte Adressierung	Der Operand ist der Inhalt einer Adresse, deren Wert sich aus dem Inhalt eines Registers, addiert um den Inhalt eines im Befehl angegebenen Arbeitsregisters (Index), ergibt.
Relative Adressierung	Der Operand ist der Befehlszählerinhalt, versetzt (plus/minus) um den im Befehl angegebenen Distanzwert (displacement).

4.1.7. Befehlssatz

Der Befehlssatz des Einchipmikrorechners U881/U882 umfaßt die Befehlsgruppen:

Ladebefehle,
Blocktransferbefehle,
Arithmetik-/Logikbefehle,
Rotations-/Verschiebebefehle,
Sprung-/Call-/Returnbefehle und
EMR-Steuerbefehle.

Die U881/U882-Befehlstypen ähneln den U880- und den U8000-Befehlstypen in vieler Hinsicht.
Die Befehle der Ladebefehlsgruppe transportieren Daten zwischen den EMR-Registern entsprechend den bei diesem Schaltkreis vorhandenen Adressierungsverfahren. Sie dienen dem Laden von Konstanten (immediate) in

die Register, dem Löschen von Registern und dem Ein- und Ausspeichern von Registerinhalten in den Stackspeicherbereich. Vier Befehle dieser Gruppe ermöglichen den Datentransport zwischen den EMR-Registern sowie dem externen Programm- und Datenspeicherbereich. Die Blocktransferbefehle (LDCI/LDEI) dienen dem Datentransfer von und zum Programm- und Datenspeicher mit automatischer Inkrementierung von zwei Zählern.
Die Arithmetik-/Logikbefehle bieten dem Programmierer die vom U880 und U8000 in ihrer Wirkungsweise weitgehend bekannten Standardbefehle zur Addition, zur Subtraktion, zum Inkrementieren/Dekrementieren (hier 8- und 16-Bit-Wörter), zum Vergleich sowie für die logischen Operationen AND, OR, XOR und die logische Negation an. Alle Befehle arbeiten mit 8-Bit-Operanden in einem EMR-Register. Ein Testbefehl für Bitwerte mit Hilfe einer Maske ergänzt diese Gruppe. Zu beachten ist hier die durchgängige Beeinflussung der Flagbits.
Die Rotations- und Verschiebebefehle gestatten die bitweise Rotation und Verschiebung von Registerinhalten.
Die Sprung-, Call- und Returnbefehle dienen dem Aufbau von Programmverzweigungen, von Programmschleifen, der Unterprogrammorganisation und der Rückkehr aus Interruptserviceroutinen. Die EMR-Steuerbefehle schließlich ermöglichen die Manipulation des C-Flagbit, des U881/U882-Interruptsystems und des Registerpointers.
Die Effektivität der Befehle sowohl in bezug auf die Abarbeitungsgeschwindigkeit (typisch 2,5 µs für ein 4 MHz U881/U882-System) als auch in bezug auf Befehlskodelänge (Arbeitsregisteradressierungsschema) ist sehr groß, so daß mit den im konkreten Einsatzfall meist begrenzten EMR-Ressourcen doch recht komplexe Aufgaben gelöst werden können. Durch den überlappenden Befehlsaufruf (instruction pipelining), bei dem sich die Abarbeitung des vorhergehenden Befehls mit dem Befehlslesen des nächsten teilweise überdeckt, wird zusätzlich eine Optimierung der Programmabarbeitungszeit erreicht.

Tafel 4.1. Befehlsliste des Einchipmikrorechenrs U881/U882

ZEICHENERLÄUTERUNG

r,r'	Arbeitsregister (Rn n=0...15)
R,R'	Register (reg 0...127,240...255)
RR	Registerpaar (reg 0,2...126,240,242...254)
Ir	Arbeitsregister indirekt (§Rn n=0...15)
IR	Register indirekt (§reg 0...127,240...255)
Irr	Arbeitsregisterpaar indirekt (§RRp p=0,2,4...14)
IRR	Registerpaar indirekt (§reg 0,2,4...126,240,242...254)
DA	direkte Adresse (16-Bit-Konstante)
RA	relative Adresse (8-Bit-Konstante)
IM	Direktwert (8-Bit-Konstante)

%	Hexadezimalkennzeichen
§	Indirektkennzeichen
*	Befehlszählerinhalt

Flagbitbedingungskode (cc):

	FALSE (falsch)	0000	----
	TRUE (richtig)	1000	----
Z	ZERO	0110	Z=1
NZ	NOT ZERO	1110	Z=0
C	CARRY	0111	C=1
NC	NOT CARRY	1111	C=0
PL	PLUS	1101	S=0
MI	MINUS	0101	S=1
NE	NOT EQUAL	0110	Z=0
EQ	EQUAL	1110	Z=1
OV	OVERFLOW	0100	V=1
NOV	NO OVERFLOW	1100	V=0
GE	GREATER THAN OR EQUAL	1001	S XOR V=0
LT	LESS THAN	0001	S XOR V=1
GT	GREATER THAN	1010	Z OR (S XOR V)=0
LE	LESS THAN OR EQUAL	0010	Z OR (S XOR V)=1
UGE	UNSIGNED GE	1111	C=0
ULT	UNSIGNED LT	0111	C=1
UGT	UNSIGNED GT	1011	(C=0) AND (Z=0)=1
ULE	UNSIGNED LE	0011	C OR Z=1

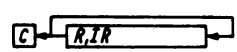
Flagbitbeeinflussung:
. unverändert
x unbestimmt
? gesetzt (0 Oder 1)
0 Null gesetzt
1 Eins gesetzt

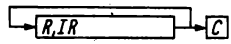
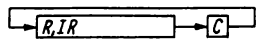
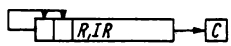
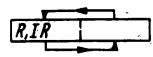
T1/T2 Befehlsausführungszyklen/Befehlspipelinezyklen (0,4 . .1,0 µs)

LADEBEFEHLE

Befehl	T1	T2	Kodierung	Erläuterung	C Z S V D H
LD r,R	6	5	--r-1000 ----R---	r:=R	. . . . .
LD R,r	6	5	--r-1001 ----R---	R:=r	. . . . .
LD r,Ir	6	5	11100011 --r--Ir-	r:=Ir	. . . . .
LD Ir,r	6	5	11110011 -Ir---r-	Ir:=r	. . . . .
LD R,R'	10	5	11100100 ----R'-- ----R---	R:=R'	. . . . .
LD R,IR	10	5	11100101 ---IR--- ----R---	R:=IR	. . .
LD IR,R	10	5	11110101 ----R--- ---IR---	IR:=R	. . . . .

Befehl	T1	T2	Kodierung	Erläuterung	C Z S V D H
LD r,IM	6	5	--r-1100 ---IM---	r:=IM	. . . . .
LD R,IM	10	5	11100110 ----R--- ---IM---	R:=IM	. . . . .
LD IR,IM	10	5	11100111 ---IR--- ---IM---	IR:=IM	
LD r,X	10	5	11000111 --r--r'- ----R---	r:=X indiziert X=reg(Rn)	. . . . .
LD X,r	10	5	11010111 --r--r'- ----R---	X:=r indiziert X=reg(Rn)	. . . . .
CLR R CLR IR	6 6	5 5	10110000 10110001 ---IR---	R:=0 IR:=0	
LDC r,Irr	12	0	11000010 --r-Irr-	r:=Irr Laden aus (in) dem (den)	. . . . .
LDC Irr,r	12	0	11010010 Irr--r-	Irr:=r Programmspeicher	
LDE r,Irr	12	0	10000010 --r-Irr-	r:=Irr Laden aus (in) dem (den)	
LDE Irr,r	12	0	10010010 Irr--r-	Irr:=r Datenspeicher	
PUSH R	10 12	1	01110000 ----R---	SP:=SP-1 (10 int.) §SP:=R (12 ext.)	
PUSH IR	12 14	1	01110001 ---IR---	SP:=SP-1 (12 int.) §SP:=IR (14 ext.)	
POP R	10	5	01010000 ----R---	R:=§SP SP:=SP+1	
POP IR	10	5	01010001 ---IR---	IR:§SP SP:=SP+1	
BLOCKTRANSFERBEFEHLE					
LDCI Ir,Irr	18	0	11000011 -Ir--Irr	Ir:=Irr Transfer aus r:=r+1 dem Programm- rr:rr+1 speicher	. . . . .
LDCI Irr,Ir	18	0	11010011 -Irr-Ir-	Irr:=Ir Transfer in r:=r+1 den Programm- rr:=rr+1 speicher	. . . . .
LDEI Ir,Irr	18	0	10000011 -Ir--Irr	Ir:=Irr Transfer aus r:=r+1 dem Daten- rr:=rr+1 speicher	. . . . .
LDEI Irr,Ir	18	0	10010011 -Irr-Ir-	Irr:=Ir Transfer in r:=r+1 den Daten- rr:=rr+1 speicher	
ARITHMETIK-/LOGIKBEFEHLE					
ADD r,r'	6	5	00000010 --r--r'-	r:=r+r'	? ? ? ? 0 ?
ADD r,Ir	6	5	00000011 --r--Ir-	r:=r+Ir	? ? ? ? 0 ?
ADD R,R'	10	5	00000100 ----R'-- ----R---	R:=R+R'	? ? ? ? 0 ?

ADD R,IR	10	5	00000101 ---IR--- ----R---	R:=R+IR		? ? ? ? 0 ?
ADD R,IM	10	5	00000110 ----R--- ---IM---	R:=R+IM		? ? ? ? 0 ?
ADD IR,IM	10	5	00000111 ---IR--- ---IM---	IR:=IR+IM		? ? ? ? 0 ?
ADC s,s' SUB s,s' SBC s,s' OR s,s' AND s,s' XOR s,s' CP s,s' TCM s,s' TM s,s'			0001 0010 0011 0100 0101 1011 1010 0110 0111	s:=s+s'+C s:=s-s' s:=s-s'-C s:=s OR s' s:=s AND s' s:=s XOR s' s=s' ? TEST s-Bit 0 (s neg UND s') TEST s-Bit 1 (s UND s')	s kann sein: r,Ir, R,IR oder IM Kodierung analog ADD 0000 wird substituiert	? ? ? ? 0 ? ? ? ? ? 1 ? ? ? ? ? 1 ? . ? ? 0 . . . ? ? 0 . . . ? ? 0 . . ? ? ? ? . . . ? ? 0 . . . ? ? 0 . .
COM R	6	5	01100000 ----R---	R:=R logisch negiert		. ? ? 0 . .
COM IR	6	5	01100001 ---IR---	IR:=IR logisch negiert		. ? ? 0 . .
DA R	8	5	01000000 ----R---	DEZIMAL ADJUST R	(BCD)	? ? ? x . .
DA IR	8	5	01000001 ---IR---	DEZIMAL ADJUST IR	(BCD)	? ? ? x . .
INC r	6	5	—r-1110	r:=r+1	8 Bit	. ? ? ? . .
INC R	6	5	00100000 ----R---	R:=R+1	8 Bit	. ? ? ? . .
INC IR	6	5	00100001 ---IR---	IR:=IR+1	8 Bit	. ? ? ? . .
DEC R	6	5	00000000 ----R---	R:=R-1	8 Bit	. ? ? ? . .
DEC IR	6	5	00000001 ---IR---	IR:=IR-1	8 Bit	. ? ? ? . .
INCW RR	10	5	10100000 ---RR---	RR:=RR+1	16 Bit	. ? ? ? . .
INCW IR	10	5	10100001 ---IR---	IR:=IR+1	16 Bit	. ? ? ? . .
DECW RR	10	5	10000000 ---RR---	RR:=RR-1	16 Bit	. ? ? ? . .
DECW IR	10	5	10000001 ---IR---	IR:=IR-1	16 Bit	. ? ? ? . .
ROTATIONS-/VERSCHIEBEBEFEHLE						
RL R	6	5	10010000 ----R---			? ? ? ? . .
RL IR	6	5	10010001 ---IR---			? ? ? ? . .
RLC R	6	5	00010000 ----R---			? ? ? ? . .
RLC IR	6	5	000100001 ---IR---			? ? ? ? . .

Befehl	T1	T2	Kodierung	Erläuterung	C Z S V D H
RR R	6	5	11100000 ----R---		? ? ? ? . .
RR IR	6	5	11100001 ---IR---		? ? ? ? . ? ? ? ? .
RRC R	6	5	11000000 ----R---		? ? ? ? . .
RRC IR	6	5	11000001 ---IR---		? ? ? ? . .
SRA	6	5	11010000 ----R---		? ? ? 0 . .
SRA IR	6	5	11010001 ---IR---		? ? ? 0 . .
SWAP R	8	5	11110000 ----R---		x ? ? x . .
SWAP IR	8 8	5	11110001 ---IR---		x ? ? x . .
SPRUNG-/CALL-/RETURNBEFEHLE					
JP IRR	8	0	00110000 --IRR---	PC:=IRR	. . . . .
JP cc,DA	12 10	0 0	-cc-1101 ---DA_H-- ---DA_L--	PC:=DA cc TRUE PC:=PC+3 cc FALSE	. . . . .
JR cc,RA	12 10	3 0	-cc-1011 ---RA---	PC:=PC+RA cc TRUE PC:=PC+2 cc FALSE	. . . . .
DJNZ RA	12	3	--r-1010 ---RA---	r:=r+1 wenn r≠0 PC:=PC+RA wenn r=0 PC:=PC+2	. . . . .
CALL DA	20	0	11010110 ---DA_H-- ---DA_L--	SP:=SP-2 §SP:=PC+3 und PC:=DA	. . . . .
CALL IRR	20	0	11010100 --IRR---	SP:=SP-2 §SP:=PC+2 und PC:=IRR	.
RET	14	0	10101111	PC:=§SP SP:=SP+2	. . . . .
IRET	16	0	10111111	FLAGS:=§SP PC:=§SP SP:=SP+3 und IMR_7:=1	? ? ? ? ? ?
EMR-STEUERBEFEHLE					
CCF	6	5	11101111	C:=C negiert	? . . . .
RCF	6	5	11001111	C:=0	0 . . . .
SCF	6	5	11011111	C:=1	1 . . . .
DI	6	1	10001111	IMR_7:=0	. . . . .
EI	6	1	10011111	IMR_7:=1	. . . . .
SRP IM	6	1	00110001 ---IM---	RP:=IM (Registerpointer)	. . . . .
NOP	6	0	11111111	no operation	. . . . .

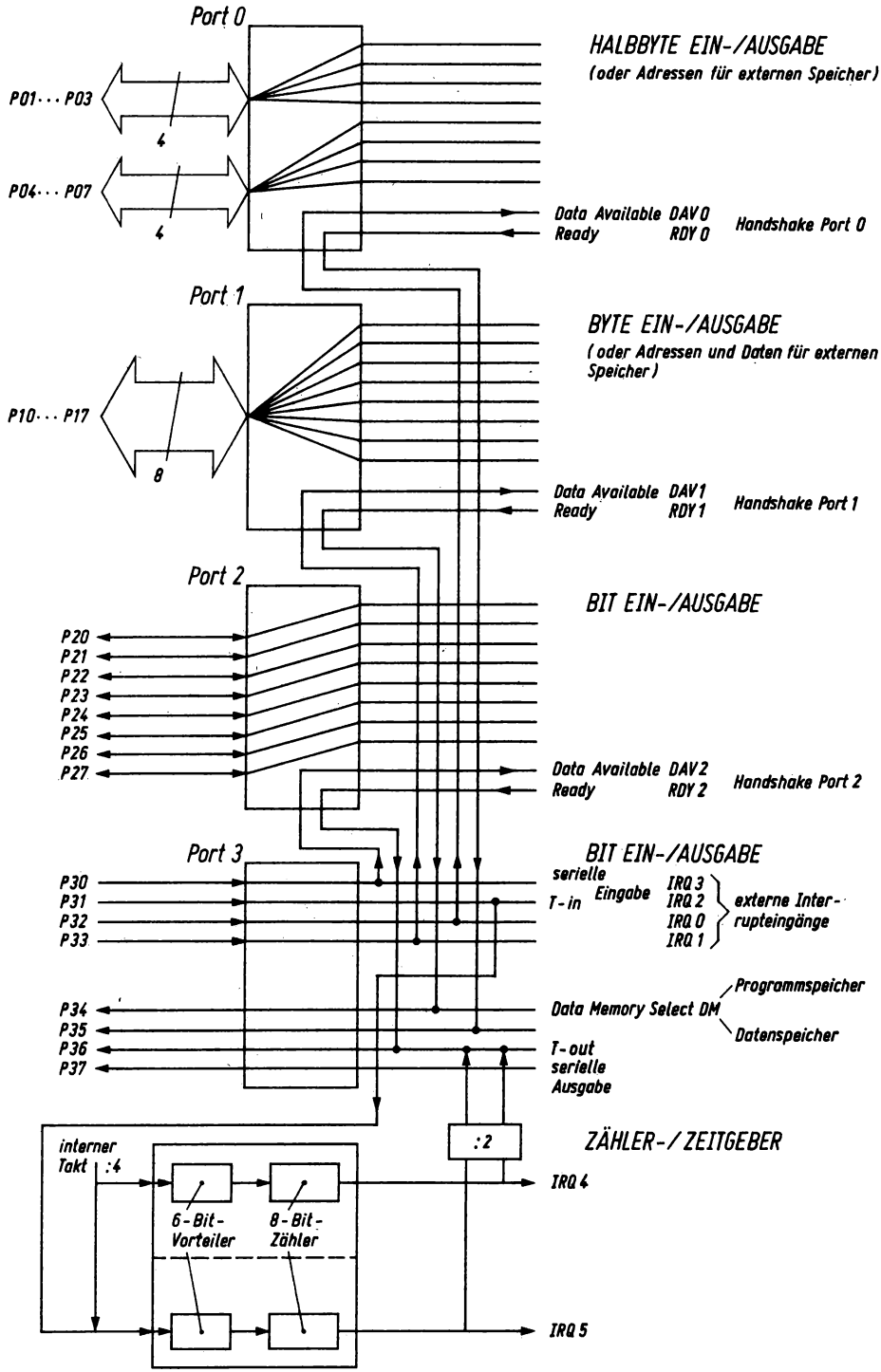


Bild 4.5. Ein-/Ausgabekanäle des Einchipmikrorechners U881/U882

4.2. Ein-/Ausgabekanäle

4.2.1. Aufbau

Insgesamt 32 Signalanschlüsse des Einchipmikrorechners U881/U882 werden zur Datenkommunikation mit der Umwelt verwendet (Bild 4.5.). Sie können zur parallelen Ein-/Ausgabe (Port 0, 1, 2 und 3), zur seriellen Ein-/Ausgabe (Port 3 Bit P30 serial in - P37 serial out), für Zähler-/Zeitgeberaktivitäten (Port 3 Bit P31 T-in - P36 T-out) oder auch zur Ankopplung externer Programm- und Datenspeicherbereiche (Port 0 und 1) genutzt werden.
Durch die spezielle Programmierung der U881/U882-Ein-/Ausgabekanäle stehen dem Anwender hier außerordentlich vielfältige Variationsmöglichkeiten offen. Die Ein- und Ausgabe von Daten über die vier E/A-Ports erfolgt durch Lesen (Eingabe) bzw. Schreiben (Ausgabe) der korrespondierenden Register R0, R1, R2 oder R3. Spezielle Ein-/Ausgabebefehle existieren beim U881/U882-EMR nicht.

Ein-/Ausgabeport 0 - Der Ein-/Ausgabeport 0 dient zur Eingabe oder Ausgabe von zwei 4-Bit-Datenströmen (Halbbyte Ein-/Ausgabe / nibble input output). Außerdem laufen gegebenenfalls über den Port 0, ebenso wie über den Port 1, je nach Programmierung vier oder acht Adreßsignalleitungen (A8...A11 oder A8...A15) bei Verwendung eines externen Programm- und (oder) Datenspeichers. Ein für die Übertragung von Adreßsignalen verwendetes Halbbyte dieses Datenkanals kann nicht mehr (oder nur durch externe Zusatzmaßnahmen und spezielle Software) für die Ein-/Ausgabe verwendet werden. Wird der Ein-/Ausgabeport 0 teilweise oder ausschließlich für den Ein-/Ausgabetransfer verwendet, können bei Bedarf über die Datenbits P32 und P35 des Ein-/Ausgabekanals 3 zwei Handshakesignalleitungen DAV0 und RDY0 (DAV data available / RDY ready) aktiviert werden (Bild 4.6.).

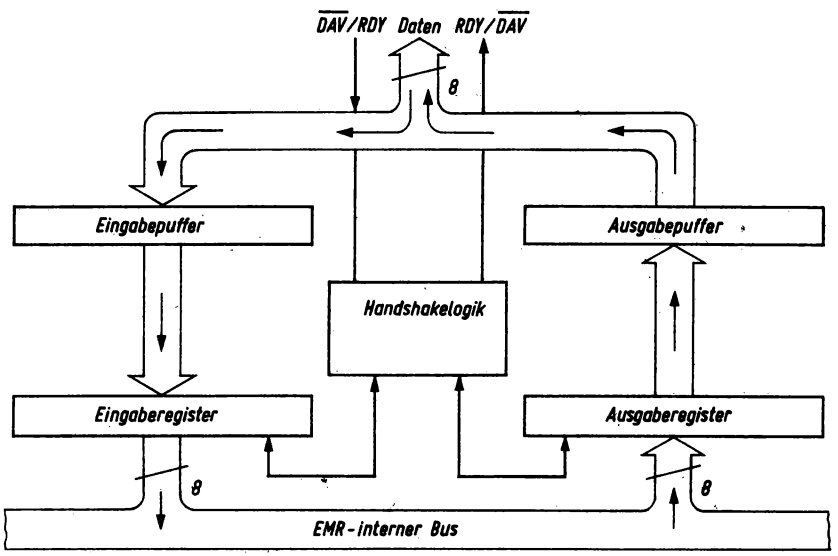


Bild 4.6. Interner Aufbau der Ein-/Ausgabekanäle 0, 1, und 2 des U881/U882

Ein-/Ausgabeport 1 - Der Ein-/Ausgabeport 1 dient zur Eingabe oder Ausgabe eines 8-Bit-Datenstromes (Byte Ein-/Ausgabe). Außerdem laufen gegebenenfalls über den Port 1 - ähnlich wie über den Port 0 - acht Adreßsignal- und Datensignalleitungen (AD0...AD7 / multiplex) bei Verwendung eines externen Programm- und (oder) Datenspeichers. Wird nur der Ein-/Ausgabeport 1 zum Anschluß externer Speicherbereiche benutzt, können maximal 256 Adressen (8 Adreßleitungen) angesprochen werden. Wird ein größerer Adreßbereich benötigt muß zusätzlich der Port 0 zur Adreßsignalvermittlung benutzt werden.
Auch dem Port 1 können bei Bedarf zwei Handshakesignalleitungen DAV1 und RDY1 analog denen des Ein-/Ausgabekanals 0 zugeordnet werden (Datenbits P33 und P34).

Ein-/Ausgabeport 2 - Der Ein-/Ausgabeport 2 dient zur Eingabe und Ausgabe von acht Einzelsignalen, die bitweise auf Eingabe oder Ausgabe geschaltet werden können.
Für die gegebenenfalls notwendige Handshakesteuerung - DAV2 und RDY2 - des Ein-/Ausgabeverkehrs gilt das beim Port 0 und 1 Gesagte (Datenbits P31 und P36).

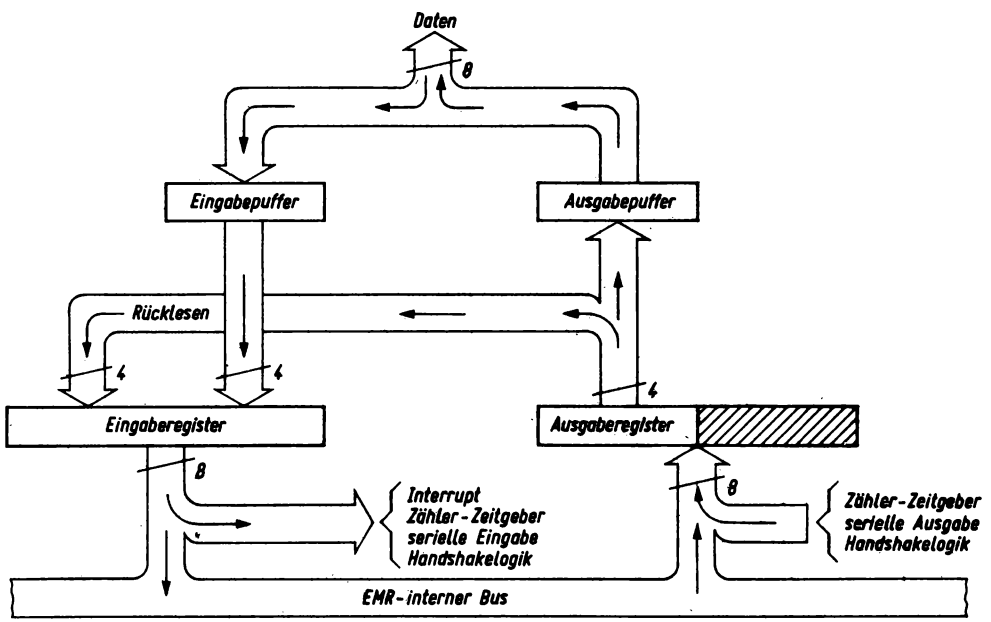


Bild 4.7. Interner Aufbau des Ein-/Ausgabekanals 3 des U881/U882

Ein-/Ausgabeport 3 - Der Ein-/Ausgabeport 3 dient, ebenso wie der Ein-/Ausgabeport 2 zur Eingabe und Ausgabe von acht Einzelsignalen, die allerdings fest aus einer 4-Bit-Eingabegruppe und einer 4-Bit-Ausgabegruppe bestehen (Bild 4.7.).

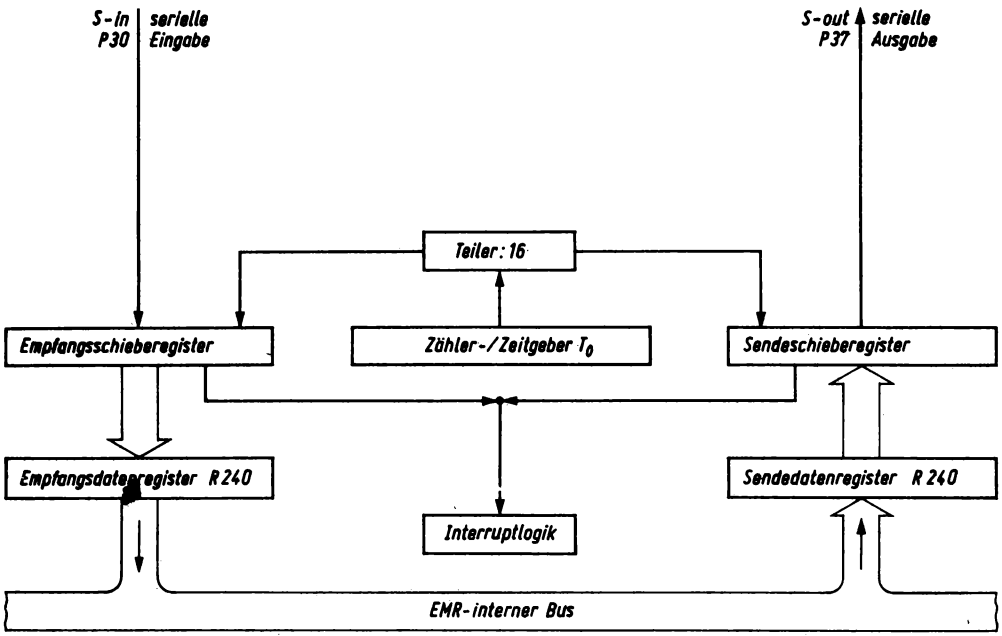


Bild 4.8. Interner Aufbau der seriellen Ein-/Ausgabe (UART) des U881/U882

Besonders zu beachten ist, daß der Ein-/Ausgabeport 3 bei Bedarf zur Vermittlung einer ganzen Reihe von speziellen Signalen verwendet werden kann:

Eingangsinterruptsignalleitungen (interrupt request)

P30	IRQ3
P31	IRQ2
P32	IRQ0
P33	IRQ1

Handshake-Quittungsbetriebssignale für die Steuerung des Ein-/Ausgabe-datenverkehrs Port 0, 1 und 2

P31	DAV2 negiert / RDY2
P32	DAV0 negiert / RDY0
P33	DAV1 negiert / RDY1
P34	RDY1 / DAV1 negiert
P35	RDY0 / DAV0 negiert
P36	RDY2 / DAV2 negiert

serielle Signale für den integrierten duplex UART-Baustein zur bit-seriellen Ein-/Ausgabe

P30 serielle Eingabe
P37 serielle Ausgabe

CTC-Ein-/Ausgabesignale 'Nulldurchgangsausgang' und 'Triggereingang'

P31 T-in
P36 T-out

DM-Signal (external data memory select) für den Anschluß externer Speicher

P34 DM negiert.

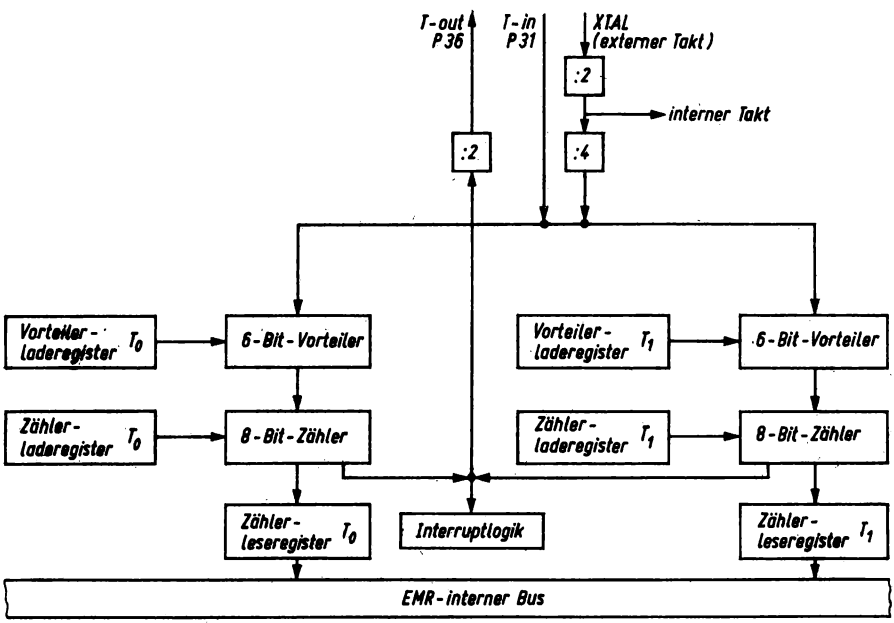


Bild 4.9. Interner Aufbau der Zähler-/Zeitgeberkanäle des U881/U882

Serielle Ein-/Ausgabe - Die bit-serielle UART-Ein-/Ausgabe mit dem U882/U882-EMR Baustein kann nur im Asynchronbetrieb mit oder ohne Paritätskontrolle mit folgenden Datentelegrammstrukturen erfolgen:

Sendedaten
1 Startbit - 8 Datenbits - 2 Stopbits
1 Startbit - 7 Datenbits - 1 Paritätsbit (odd) - 2 Stopbits

Empfangsdaten
1 Startbit - 8 Datenbits - 1 Stopbit
1 Startbit - 7 Datenbits - 1 Paritätsfehlerflag - 1 Stopbit

Die Datenbitrate für die serielle Ein-/Ausgabe wird über den CTC-Kanal T0 vorgegeben (wird intern noch dividiert durch 16 - d.h. Bitrate=XTAL:(2x4xPREOxT0x16)). Die serielle Dateneingabe und die serielle Datenausgabe selbst erfolgt vom Programm über Lese- und Schreibbefehle bezogen auf das Register R240 (Bild 4.8.).

Zähler-/Zeitgeber - Der CTC-Funktionsblock (CTC counter timer circuit) dient, je nach Initialprogrammierung, als Zähler oder als Zeitgeber, das heißt zum Zählen von externen Ereignissen über den Triggereingang oder zum Zählen des internen Systemtaktes. Er ist aus zwei adäquaten Kanälen aufgebaut, die jeweils aus einem 6-Bit-Vorteilerregister und aus einem 8-Bit-Zählregister bestehen. Die Betriebsart Zeitgeber dient insbesondere zum Aufbau einer EMR-internen Zeitbasis. Sie ist unabdingbare Voraussetzung für den Einsatz des Einchipmikrorechners in Echtzeitsystemen (Bild 4.9.).

4.2.2. Programmierung

Die Programmierung der Ein-/Ausgabekanäle des U881/U882-EMR erfolgt über die Register R240, R241 R248 (Tafel 4.2. und Bild 4.3.).

Tafel 4.2. Programmierung der Ein-/Ausgabeports des Einchipmikrorechners U881/U882

Für alle Register gilt der Aufbau:

D7	D6	D5	D4	D3	D2	D1	D0

Notation: Pxy Ein-/Ausgabebit y im Ein-/Ausgabeport x

Ein-/Ausgabeport 0 und 1 Moderegister R248(0F8H) P01M (Schreibregister)

```
D7,D6=00  P04...P07 Ausgabebits
D7,D6=01  P04...P07 Eingabebits
D7,D6=1x  P04...P07 Adreßbits A12...A15 (x Bit ohne Bedeutung)

   D5=0   normales Memory-Timing
   D5=1   erweitertes Memory-Timing (eine zusätzliche Clockperiode)

D4,D3=00  P10...P17 Byteausgabe
D4,D3=01  P10...P17 Byteeingabe
D4,D3=10  P10...P17 Adreß-/Datenbits AD0...AD7 (multiplex)
D4,D3=11  P10...P17 hochohmig AD0...AD7, AS, DS, R/W, A8...A11, A12...A15
                    - wenn entsprechend programmiert

   D2=0   externer Stack
   D2=1   interner Stack

D1,D0=00  P00...P03 Ausgabebits
D1,D0=01  P00...P03 Eingabebits
D1,D0=1x  P00...P03 Adreßbits A8...A11  (x Bit ohne Bedeutung)
```

Ein-/Ausgabeport 2 Moderegister R246(0F6H) P2M (Schreibregister)

D7,D6...D0 Das entsprechende Bit des Port 2 wird als Ausgabebit (=0) oder Eingabebit (=1) definiert.

Ein-/Ausgabeport 3 Moderegister R247(0F7H) P3M (Schreibregister)

```
      D7=0     Keine Paritätskontrolle bei der seriellen Datenübertragung
      D7=1     Paritätskontrolle bei der seriellen Datenübertragung

      D6=0     P30  Eingabebit             /  P37  Ausgabebit
      D6=1     P30  serielle Eingabe       /  P37  serielle Ausgabe

      D5=0     P31  Eingabebit (CTC T-in)  /  P36  Ausgabebit (CTC T-out)
      D5=1     P31  Port 2 DAV2 negiert    /  P36  Port 2 RDY2

   D4,D3=00    P33  Eingabebit             /  P34  Ausgabebit
   D4,D3=01/10 P33  Eingabebit             /  P34  DM negiert
   D4,D3=11    P33  Port 1 DAV1 negiert    /  P34  Port 1 RDY1

      D2=0     P32  Eingabebit             /  P35  Ausgabebit
      D2=1     P32  Port 0 DAV0 negiert    /  P35  Port 0 RDY0

      D1=x     nicht benutzt

      D0=0     Port 2 Ausgänge  (pull-ups: open drain)
      D0=1     Port 2 Ausgänge  (pull-ups: active)
```

Serielles Ein-/Ausgaberegister R240(0F0H) SIO (Lese-/Schreibregister)

```
   D7,D6...D0   UART-Daten (D0 niederwertigstes Bit)
                Lesen R240 - serielle Eingabe
                Schreiben R240 - serielle Ausgabe
```

Zeitgebermoderegister R241(0F1H) TMR (Lese-/Schreibregister)

```
   D7,D6=00  ohne Bedeutung (P36 ist ein Datenausgabebit / s. R247 D5)
   D7,D6=01  T0-Ausgabe (T-out)
   D7,D6=10  T1-Ausgabe (T-out)
   D7,D6=11  Systemtaktausgabe (:2 T-out)

   D5,D4=00  ext. Takteingang (T-in) / (ext. Takt:4 in Vorteiler)
   D5,D4=01  ext. Toreingang (T-in) / (ext. Takt:4 in Vorteiler / ext.
                                       Startflanke 'high to low')
   D5,D4=10  ext. Triggereingang (non-triggerable) (T-in) / (interner Takt:4
                                       in Vorteiler / einmalige  ext. Start-
                                       flanke 'high to low')
   D5,D4=11  ext. Triggereingang (retriggerable) (T-in) / (interner   Takt:4
                                       in Vorteiler / mehrmalige ext. Start-
                                       flanke 'high to low')

      D3=0   Ausschalten T1-Zählung
      D3=1   Einschalten T1-Zählung

      D2=0   nicht benutzt
      D2=1   Laden von T1 (Übernahme R243 und R242)

      D1=0   Ausschalten T0-Zählung
      D1=1   Einschalten T0-Zählung

      D0=0   nicht benutzt
      D0=1   Laden von T0 (Übernahme R245 und R244)
```

T1-Zähler-/Zeitgeberregister R242(0F2H) T1 (Lese-/Schreibregister)

```
D7,D6...D0  Zählerinitialwert beim Schreiben (1...255,256 / 01...FF,00)
            momentaner Zählerstand beim Lesen
```

T1-Vorteilerregister R243(0F3H) PRE1 (Schreibregister)

D7,D6,D5,D4,D3,D2= Vorteilerwert (1...63,64 / 01...FC,00)

D1=0 T1 externer Zähltakt (T-in Mode)
D1=1 T1 interner Zähltakt (interner Takt:4 in Vorteiler)

D0=0 Betriebsart Single-Pass (einmaliges Abzählen)
D0=1 Betriebsart Modulo-N (mehrmaliges Abzählen mit Autorestart)

T0-Zähler-/Zeitgeberregister R244(0F4H) T0 (Lese-/Schreibregister)

Die Bitbelegung des Zähler-/Zeitgeberregisters T0 (R244) entspricht der Bitbelegung des Zähler-/Zeitgeberregisters T1 (R242).

T0-Vorteilerregister R245(0F5H) PRE0 (Schreibregister)

Die Bitbelegung des Vorteilerregisters PRE0 (R245) entspricht der Bitbelegung des Vorteilerregisters PRE1 (R243) - ausgenommen Bitposition D1.

5. Programmiersprache PLZ/ASM

Der größte Teil der Mikroprozessorsoftware wird derzeit in Assemblersprache erstellt. Daneben gewinnen aber auch höhere Programmiersprachen für Mikroprozessoranwendungen immer mehr an Bedeutung. Die Programmiersprache PLZ/ASM stellt eine Erweiterung der bisher bekannten "reinen" Assemblersprache mit Elementen von höheren Programmiersprachen dar.

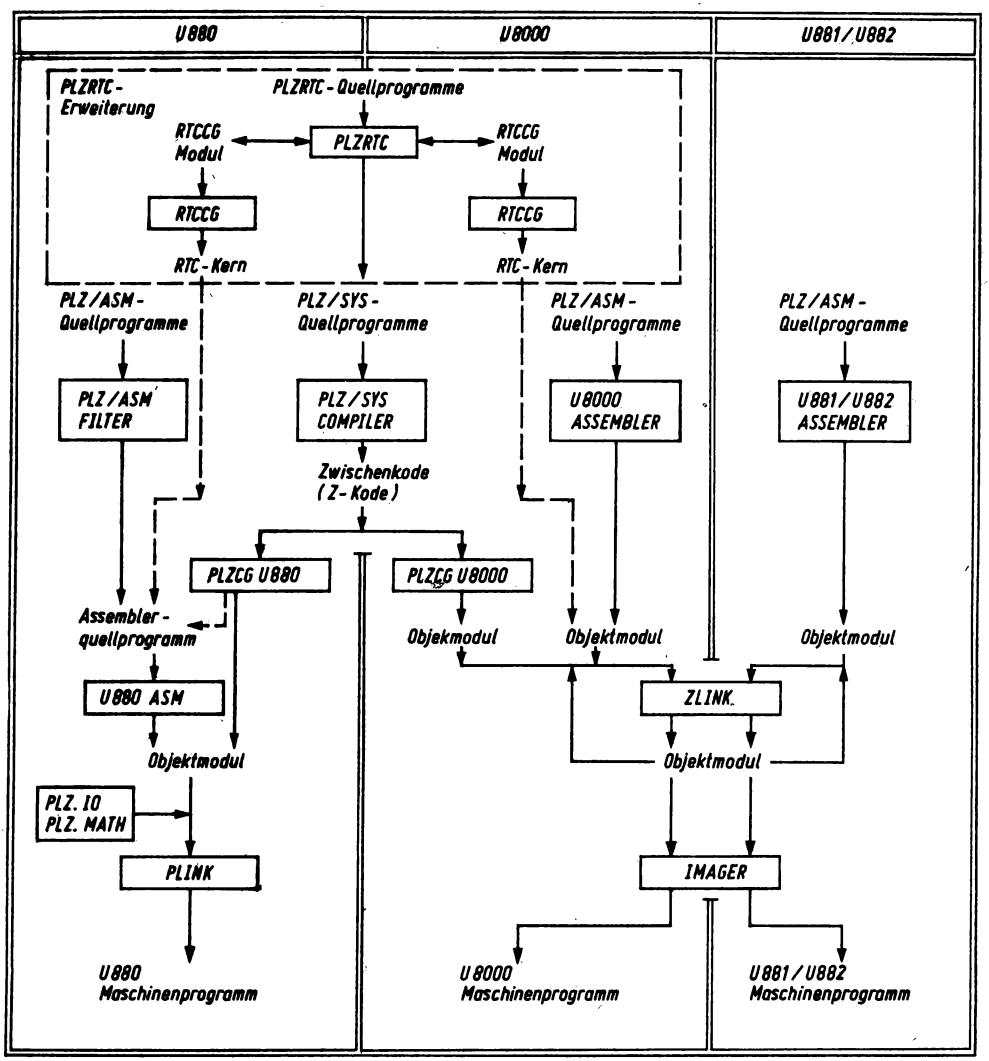


Bild 5.1. PLZ-Sprachfamilie

Anmerkung: Die Darstellung zum LINK- und IMAGER-Programm bedeutet nicht, daß U8000-Objektmodule mit U881/U882-Objektmodulen verbunden werden können, sondern daß die Programme LINK und IMAGER für beide Prozessoren identisch sind. Erläuterungen zu diesen Programmen und zur Handhabung werden im Abschn. 6.3. gegeben.

Diese Elemente unterstützen den Programmierer in den Fragen der Datenstrukturierung, der Programmablaufsteuerung und im modularen Programmaufbau. Da das Konzept von PLZ/ASM für alle drei Prozessoren (U880, U8000 und U881/U882) nahezu identisch ist und entsprechende Übersetzerprogramme unter dem Betriebssystem UDOS verfügbar sind, wird es im folgenden näher erläutert.
Die Programmiersprache PLZ/ASM ist Bestandteil der PLZ-Sprachfamilie. Innerhalb dieser Familie ist die höhere Programmiersprache PLZ/SYS hervorzuheben, die zur Entwicklung von Systemprogrammen (Compiler, Dateihandler, Betriebssystemkomponenten u.a.m.) entworfen wurde.
Auf der Basis dieser Sprache wurde an der Technischen Hochschule Karl-Marx-Stadt die Echtzeitprogrammiersprache PLZRTC entwickelt, mit deren Hilfe es möglich ist, für Echtzeitanwendungen optimale Echtzeitbetriebssystemkerne zu generieren. Durch einen ebenfalls an der TH Karl-Marx-Stadt, Sektion Informationstechnik, erarbeiteten PLZ/SYS-Kodegenerator für den U8000 wurde die Anwendungsbreite von PLZ/SYS und PLZRTC auf den U8000 erweitert.
Aus Platzgründen kann auf diese Komponenten nicht näher eingegangen werden. Für detaillierte Informationen sei auf die Literatur /35/,/36/,/37/ verwiesen. Die Übersicht im Bild 5.1. verdeutlicht das Zusammenspiel der verschiedenen Systemkomponenten für die einzelnen Prozessoren.

5.1. Sprachkonzept

Wie aus der Übersicht zur PLZ-Sprachfamilie (Bild 5.1.) deutlich wird, existiert beim U880 ein Vorübersetzerprogramm, das die höheren Sprachelemente in Assembleranweisungen auflöst. Beim U8000 und U881/U882 dagegen werden als Assemblerquellen nur Quellprogramme mit PLZ/ASM-Struktur akzeptiert. Im Vergleich zum U880 wurden für den U8000 und den U881/U882 die Möglichkeiten von PLZ/ASM erweitert (z.B.: Typdefinition). Das Konzept von PLZ/ASM für die letzteren beiden Prozessoren ist identisch. Es wird in den folgenden Punkten näher beschrieben. Für den U880 sei auf /2/ verwiesen.

Innerhalb der nachfolgenden Abschnitte wird zur syntaktischen Darstellung eine modifizierte Backus-Naur-Darstellung verwendet. Syntaktische Konstruktionen werden durch Texte, die in "<",">" eingeschlossen sind, gekennzeichnet. Mögliche Wiederholungen einer Konstruktion werden durch Anhängen von

\+ (ein- oder mehrmalige Wiederholung) oder
* (null- oder mehrmalige Wiederholung) angezeigt.

Basissymbole der Sprache werden entweder groß geschrieben (Schlüsselwörter) oder in Apostrophe (für Symbole) gesetzt.
Runde Klammern sind Metasymbole und werden benutzt, um eine Gruppe von Konstruktionen zusammenzufassen, auf die ein Wiederholungssymbol folgt. Die Möglichkeit des Weglassens einer Konstruktion wird durch Einschließen in die Metasymbole [und] angezeigt. Die Sprachelemente werden anhand von Beispielen erläutert. Vollständige Syntaxübersichten sind in /31/ und /32/ angegeben.

5.1.1. Strukturierung von PLZ/ASM-Programmen

PLZ/ASM-Programme bestehen aus einem oder mehreren getrennt übersetzbaren Modulen. Diese werden unter Benutzung des Verbinderprogramms (LINKER) zu einem ausführbaren Programm verbunden. Dabei enthält ein Modul das "Hauptprogramm". Dies ist eine als global gekennzeichnete Prozedur, die beim Verbinden als Eintrittspunkt angegeben wird.
Module bestehen aus Assembler- und höheren Sprachanweisungen, die entweder Daten vereinbaren, definieren oder Aktionen beschreiben. Ausführbare Anweisungen müssen innerhalb von Prozeduren erscheinen. Prozeduren erhalten ebenso wie Module Namen, so daß sie über CALL-Anweisungen aufrufbar sind. Eine Prozedur kann lokale Variablenvereinbarungen enthalten. Neben den Assembleranweisungen können innerhalb einer Prozedur zur Steuerung des Programmablaufs auch höhere Sprachelemente, wie: DO-Schleifen, REPEAT-, EXIT-, IF-, CASE-Anweisungen verwendet werden.
Auf Datenvereinbarungen, Marken und Prozeduren kann von einem anderen Modul aus zugegriffen werden. Sie müssen in dem Modul, in dem sie definiert sind, als GLOBAL gekennzeichnet sein und in dem Modul, in dem auf sie zugegriffen wird, als EXTERNAL. Wenn sie als INTERNAL vereinbart sind, so ist ihr Gültigkeitsbereich nur der Modul selbst. Der Gültigkeitsbereich lokaler Größen (LOCAL) ist die Prozedur, in der sie definiert sind.

Der größte Teil eines PLZ/ASM-Programmes werden natürlich Assembleranweisungen sein. Die höheren Sprachelemente dienen zur besseren Strukturierung und Selbstdokumentation der Quellprogramme. Höhere Sprachelemente realisieren zwei Grundfunktionen:

Definition der Programmstruktur (Module, Prozeduren)
Vereinbarung und Definition von Daten

Assemblersteueranweisungen steuern die Art der Übersetzung (absolut oder verschieblich, bedingte Übersetzung u.a.m., s. Abschn. 6.3.). Sie werden durch Angabe eines Zeichens "$" als erstes Zeichen gekennzeichnet und können an beliebiger Stelle innerhalb eines Moduls stehen. Beispiele, die die Struktur von PLZ/ASM-Programmen verdeutlichen werden im Abschn. 5.2. angegeben.

5.1.2. Datenvereinbarungen

Daten (Konstanten und Variablen) müssen vor ihrer Benutzung definiert oder vereinbart werden. Im allgemeinen verbindet eine Datendefinition einen Namen mit einem festen Wert oder Typ. Datenvereinbarungen führen einen Bezeichner als Namen einer Variablen ein und legen deren Typ und Gültigkeitsbereich fest. Folgende drei Anweisungen werden benutzt, um Daten zu definieren und zu vereinbaren:

Konstantendefinition (CONSTANT)
Typdefinition (TYPE)
Variablenvereinbarung

Durch eine Konstantendefinition erhält ein Name den Wert eines Konstantenausdrucks. Alle zur Definition verwendeten Namen müssen vorher definiert sein.

Der Gültigkeitsbereich einer Konstanten ist der Modul, in dem sie definiert wurde (INTERNAL).
Eine Konstantendefinition hat folgende Syntax:

```
CONSTANT
  (<Konstantenname> ':=' <konstanter_Ausdruck>)+
```

Beispiel:

```
CONSTANT
    Zeilen := 24
    Spalten := 80
    ANZAHL := Zeilen*Spalten
    Zeichen_x := 'x'
```

Datentypen werden mit einer Variablen verbunden, um deren Größe zu kennzeichnen, oder um einen Namen als Marke zu definieren. Datentypen können entweder direkt in einer Variablendeklaration verwendet werden, oder sie können über einen vom Anwender ausgewählten Typnamen, der in einer Typanweisung definiert wurde, zur Kennzeichnung von Variablen herangezogen werden.
Eine Typdefinition hat folgende Syntax:

```
TYPE
  (<Typname>  <Typ>)*
```

Es werden einfache und strukturierte Datentypen unterschieden. Einfache Datentypen sind:

SHORT_INTEGER oder BYTE	ein Byte (8 Bit)
INTEGER oder WORD	ein Wort (16 Bit)

Beim U8000 kommt noch dazu:

LONG_INTEGER oder LONG	zwei Worte (32 Bit)

Beispiele für einfache Datentypdefinitionen:

```
TYPE
  CHAR  BYTE
  DIGIT CHAR
  pointer WORD
```

An strukturierten Datentypen werden unterschieden:

ARRAY	Zusammenfassung von Komponenten eines Typs. Adressierung erfolgt über einen Index (0 bis N-1, bei N Elementen).
RECORD	Zusammenfassung von Komponenten (genannt Felder) unterschiedlichen Typs. Der Zugriff erfolgt über Feldnamen.

Beispiele für strukturierte Datentypvereinbarungen:

```
TYPE
  TEXT ARRAY [64 BYTE]

INTERNAL
  titel TEXT
  ! titel definiert ein 64 Byte grosses Feld !

TYPE
  adr RECORD [nr WORD  name TEXT]

INTERNAL
  s2 adr
  ! adr definiert einen 66 Byte grossen Bereich !
  ! auf die Feldelemente von s2 kann ueber die Notation
    s2.nr und s2.name zugegriffen werden !
```

Variablenvereinbarungen werden benutzt, um den Typ von Variablen zu spezifizieren und ihnen wahlweise Anfangswerte zuzuweisen. Eine Variablenvereinbarung hat folgende Syntax:

```
<Variablenname>+  <Typ> [':=' <Anfangswerte>]
```

Sie können in folgenden Kategorien erfolgen: GLOBAL, EXTERNAL, INTERNAL (im Modulniveau) oder LOCAL (im Prozedurniveau). Dieses Format ist jedoch durch den Gültigkeitsbereich und den angegebenen Typ beschränkt. Externe Variablenvereinbarungen dürfen keine Anfangswertzuweisung enthalten.
Für die Anfangswertzuweisung gibt es verschiedene Regeln und spezielle Symbole mit einer besonderen Bedeutung. Wenn eine einzelne Vereinbarung mehr als einen Variablennamen enthält, so müssen die aufgelisteten Anfangswerte in eckige Klammern eingeschlossen werden. Variablen einfachen Typs werden durch die Angabe von konstanten Ausdrücken initialisiert. Variablen strukturierten Typs werden durch einen sogenannten Konstruktor, einer Liste von in eckigen Klammern eingeschlossenen Werten, initialisiert. Dabei kennzeichnet immer ein Paar von eckigen Klammern ein Verschachtelungsniveau.
Das spezielle Symbol "..." kennzeichnet, daß der vorherige Wert oder Konstruktor für den Rest der Variablen im aktuellen Niveau wiederholt wird. Das spezielle Symbol "?" kann in einer Liste von Anfangswerten für Variablen oder Komponenten einfachen Typs verwendet werden, um dieser Stelle keinen Anfangswert zuzuweisen. Der leere Konstruktor "[]" kennzeichnet, daß die entsprechende strukturierte Variable keinen Anfangswert erhält.

Beispiele für einfache Variablenvereinbarungen:

```
INTERNAL
  end WORD := %FFFF
  nr,anz,step BYTE := [0...]
  A,B,C BYTE := ['x','y','z']
  D,E,F BYTE := [0,1]    ! F ist undefiniert !
```

Eine Variablenvereinbarung vom Typ ARRAY hat folgende Syntax:

```
<Name>+ ARRAY '['<Dimension>+ <Typ>']'[':='<Anfangswerte>]
```

oder

```
<Name>+ <Feldtyp> [':='<Anfangswerte>]
```

Dabei kennzeichnet <Dimension> die Anzahl der Dimensionen in der ARRAY-Struktur und die Anzahl der Elemente in jeder Dimension. <Typ> kann ein einfacher Typ, ein vorher definierter Typbezeichner oder eine ARRAY- oder RECORD-Typdefintion sein. <Feldtyp> muß ein vorher definierter ARRAY-Typbezeichner sein. Die Anfangswerte sind eine in eckigen Klammern eingeschlossene Liste von konstanten Ausdrücken. Die Initialisierung erfolgt von links nach rechts (z.B.: wird eine 2x2-Matrix in folgender Reihenfolge:

```
[0,0] [0,1] [1,0] [1,1]
```

initialisiert.
Bei einem eindimensionalen Feld kann anstelle der Dimension auch das Zeichen "*" verwendet werden. In diesem Fall wird die Feldlänge aus der Größe der Liste der Anfangswerte ermittelt. Dabei kann dann auch anstelle der in eckigen Klammern eingeschlossenen Anfangswerten eine in Hochkomma eingeschlossene Zeichenfolge verwendet werden.

Beispiele:

```
INTERNAL
  matrix  ARRAY [20 20 WORD]
  liste   ARRAY [8 BYTE] := [1,1,0,0]
  text    ARRAY [5 BYTE] := ['M','E','Y','E','R']
  tab1,tab2 ARRAY [2 BYTE] :=[[0...][1...]]
  liste1  ARRAY [* BYTE] := [1,1,0,0]
  ! liste1 ist ein Feld von 4 Bytes; das Feld liste ist ein
    Feld von 8 Bytes, von denen nur vier initialisiert sind !
  text1   ARRAY [* BYTE] := 'MUELLER'
```

Eine Variablenvereinbarung vom Typ RECORD hat folgende Syntax:

```
<Name>+ RECORD '['(<Feldname>+ <Typ>)+']'[':='<Anfangswerte>]
```

oder

```
<Name>+ <Recordtyp> [':='<Anfangswerte>]
```

<Typ> kann wiederum ein einfacher Typ, ein vorher definierter Typbezeichner oder eine ARRAY- oder RECORD-Typdefinition sein. <Recordtyp> muß ein vorher definierter RECORD-Typbezeichner sein. Falls ein ARRAY oder RECORD als Komponenten auftreten, so wird diese Verschachtelung durch Einschließen in eckige Klammern gekennzeichnet. Die Zuordnung der angegebenen Anfangswerte erfolgt von links nach rechts.

Beispiele:

```
GLOBAL
  person RECORD [alter,groesse,gewicht BYTE
                 geburt RECORD [tag,monat,jahr BYTE]
                 pkz WORD ]
  text RECORD [laenge BYTE zeichen ARRAY[64 BYTE]]:=[0[0]]
  ! die Laenge und das erste Zeichen des Feldes werden mit
   Null initialisiert !


TYPE
  patient RECORD [zimmer WORD
                  geburt RECORD [tag,monat,jahr BYTE]
                  alter,geschlecht BYTE]

INTERNAL
  weiblich ARRAY [150 patient] := [[?,[],?,'W']...]
  ! nur das Geschlecht eines jeden Records wird initialisiert !
```

Das Übersetzerprogramm (Assembler) sorgt bei Variablen einfachen Typs automatisch dafür, daß alle Werte größer 8 Bit (WORD, LONG) auf geraden Speicheradressen beginnen. Dies erfolgt ebenfalls mit Anweisungen innerhalb einer Prozedur. Bei strukturierten Variablen muß der Programmierer selber auf diese Ausrichtung achten!

Neben den bisher beschriebenen Vereinbarungen existiert noch eine Markenvereinbarung. Durch sie wird ein angegebener Name als Markenname in einem Programm gekennzeichnet. Er kann in seinem Gültigkeitsbereich für keinen anderen Zweck verwendet werden. Die Syntax einer Markenvereinbarung ist wie folgt:

```
<Name>+ LABEL
```

Bei der Definition wird dem Namen kein Doppelpunkt nachgestellt. Die Markenvereinbarung kann nicht dazu benutzt werden, den Marken absolute Adressen zuzuweisen. Eine Marke kann folgende Gültigkeitsbereiche haben: GLOBAL, EXTERNAL, INTERNAL oder LOCAL. Falls eine Marke in einem ausführbaren Teil verwendet wird, ohne daß sie durch eine Markenvereinbarung spezifiziert wurde, so ist ihr Gültigkeitsbereich der Modul, in dem sie definiert wurde (INTERNAL). Falls eine Marke in einer Prozedur lokal sein soll, so muß sie vor dem ENTRY-Schlüsselwort als lokale Marke vereinbart werden. Der Gültigkeitsbereich von Marken der Form $n (n-Dezimalzahl) ist immer nur die Prozedur, in der sie definiert wurden.

Beispiel:

```
GLOBAL
  m2 LABEL

  bsp PROCEDURE      ! Prozedur ist global !
     LOCAL
          .
          .
       m1 LABEL
     ENTRY
       m1:           ! m1 ist lokal zu bsp !
          .
          .
       m2:           ! m2 ist global !
          .
          .
       m3:           ! m3 ist intern zum Modul !
          .
          .
       $1:           ! $1 ist lokal zu bsp !
          .
          .
  END bsp
```

Dem Anwender steht noch ein spezieller unärer Operator (SIZEOF) zur Verfügung, der auf Typnamen angewendet werden kann und als Wert die Größe in Bytes liefert.

Beispiele:

```
TYPE
  char BYTE
  zahl char
  feld ARRAY [5 5 WORD]
  patient RECORD [alter,groesse BYTE
                  zimmer WORD]

                          Wert:
SIZEOF digit               1
SIZEOF matrix             50
SIZEOF patient             4
SIZEOF patient.zimmer      2
```

5.1.3. Prozedurvereinbarungen

Eine Prozedurvereinbarung definiert einen ausführbaren Programmteil und verbindet diesen mit einem Namen, so daß er über eine CALL-Anweisung aufgerufen werden kann. Im Prozedurkopf wird der Prozedurname spezifiziert. Er kennzeichnet die erste Anweisung der Prozedur und kann wie jede andere Programmarke verwendet werden. Prozeduren können in folgenden Kategorien definiert werden: GLOBAL, INTERNAL oder EXTERNAL. Bei

einer externen Prozedurvereinbarung wird nur der Prozedurname angegeben, da die Definition der Prozedur in einem anderen Modul erfolgt. Eine Prozedurvereinbarung kann auch lokale Variablenvereinbarungen enthalten. Der Gültigkeitsbereich dieser Variablen ist dann nur die Prozedur, in der sie definiert wurden.
Eine Prozedurvereinbarung hat folgende Syntax:

```
<Prozedurname> PROCEDURE
  [LOCAL
    (<Variablenname>+ <Typ>)* ]*
  [ENTRY
     <Anweisung>* ]
END <Prozedurname>
```

Für < Anweisung > kann dabei eine Assembleranweisung, eine DO-, IF-, EXIT-, oder REPEAT-Anweisung stehen. Das Schlüsselwort ENTRY trennt die Vereinbarung von lokalen Variablen vom ausführbaren Teil der Prozedur. Zu beachten ist, daß vor dem Ende der Prozedur eine RET-Anweisung zu programmieren ist. Dies ist ein wesentlicher Unterschied zu dem in /2/ beschriebenen PLZ/ASM-Vorübersetzerprogramm für den U880. Dort wird die RET-Anweisung automatisch zum Ende einer Prozedur generiert.
Zur Ablaufsteuerung innerhalb einer Prozedur existieren folgende höhere Anweisungen:

DO-Anweisung

Sie liefert die Möglichkeit Programmschleifen zu programmieren. Die zwischen den Schlüsselwörtern DO und OD eingeschlossenen Anweisungen werden so lange wiederholt, bis durch eine Schleifensteueranweisung ein Abbruch erfolgt. Eine solche Steueranweisung kann eine Assembler-sprunganweisung (z.B.: JP, DJNZ), eine EXIT- oder REPEAT-Anweisung sein. Eine DO-Anweisung hat folgende Syntax:

```
[<Marke>]*
DO
  <Anweisung>*
OD
```

Für < Anweisung > kann dabei eine Assembleranweisung, eine DO-, IF-, EXIT- oder REPEAT-Anweisung stehen.

EXIT-Anweisung

Die EXIT-Anweisung veranlaßt das Verlassen einer DO-Schleife. Sie liefert die Möglichkeit, mit der nach der DO-Schleife programmierten Anweisung fortzusetzen. Durch Angabe der Marke einer DO-Anweisung kann auch ein bestimmter zu verlassender DO-Block spezifiziert werden. Sie besitzt folgende Syntax:

```
EXIT [FROM <Marke>]
```

Das Übersetzerprogramm erzeugt dafür einen unbedingten Sprung zu der nach dem OD-Schlüsselwort folgenden Anweisung. Dabei wird je nach

Entfernung ein relativer (JR) oder absoluter Sprung (JP) erzeugt.

REPEAT-Anweisung

Die REPEAT-Anweisung veranlaßt einen Neubeginn der Schleife, in der sie erscheint. Durch Angabe der Marke einer DO-Anweisung kann auch ein bestimmter DO-Block spezifiziert werden, bei dem wieder begonnen werden soll. Die Syntax ist analog zur EXIT-Anweisung:

```
REPEAT [FROM <Marke>]
```

Das Übersetzerprogramm erzeugt dafür wiederum einen unbedingten Sprung zu dem entsprechenden DO-Schlüsselwort. Dabei wird je nach Entfernung ein relativer (JR) oder absoluter Sprung (JP) erzeugt.

Beispiel:

```
M1:  DO
          .
          .
       IF Z THEN REPEAT FI
       DO
          .
          .
          IF GE THEN EXIT FROM M1 FI
          .
          .
       OD
     OD
```

IF-Anweisung

Sie liefert die Möglichkeit, eine Programmverzweigung zu programmieren. Falls die nach IF folgende Bedingung wahr ist, so werden die zwischen THEN und ELSE (oder zwischen THEN und FI, falls der ELSE-Zweig fehlt) eingeschlossenen Anweisungen ausgeführt. Andernfalls werden die zwischen ELSE und FI eingeschlossenen Anweisungen ausgeführt. Falls die Bedingung nicht erfüllt ist und kein ELSE-Zweig vorhanden ist, so wird mit der nach FI folgenden Anweisung fortgesetzt. Die IF-Anweisung hat folgende Syntax:

```
IF <Bedingungskode>
THEN <Anweisung1>*
[ELSE <Anweisung2>*]
FI
```

Für < Bedingungskode > kann dabei eine der in den Befehlslisten (Abschnitte 3.1.6. und 4.1.7.) angegebenen Abkürzungen (z.B.: Z, NZ, C, NC u.a.) stehen.

Beispiel:

```
IF NZ
  THEN CALL UP1; JP Marke1
  ELSE CALL UP2
FI
```

IF/CASE-Anweisung

Die IF/CASE-Anweisung stellt eine Erweiterung der IF-Anweisung dar. Sie gestattet in Abhängigkeit vom Inhalt eines Selektorregisters eine Mehrfachverzweigung. In einer CASE-Gruppe können mehr als ein Ausdruck vorkommen. Nach jeder CASE-Gruppe wird ein unbedingter Sprung zum Ende der Select-Konstruktion eingefügt. Ein ELSE-Zweig kann als Alternative zu allen CASE-Fällen spezifiziert werden. Falls kein ELSE-Zweig angegeben ist und keine Übereinstimmung mit einer CASE-Gruppe auftritt, so wird mit der nach FI stehenden Anweisung fortgesetzt. Die IF/CASE-Anweisung hat folgende Syntax:

```
IF <Selektorregister>
  (CASE <Ausdruck>+ THEN <Anweisung>*)+
  [ELSE <Anweisung>*]
FI
```

Für <Selektorregister> steht dabei eine Registerbezeichnung. Der nach CASE anzugebende Ausdruck muß einen gültigen Operanden in einer Vergleichsanweisung darstellen.

Beispiel:

```
IF R3
  CASE #4  THEN CALL UP1
  CASE #2,R4,@R5  THEN CALL UP2
  ELSE JP error
FI
```

5.2. Programmbeispiele

Als Programmbeispiel wurde ein einfacher Sortieralgorithmus (in /32/ dargestellt) ausgewählt. Ausgangspunkt für die Sortierprogramme ist ein externes Feld von 10 Wörtern. Es wird das Feld 'list' (von list[0] bis list[9]) durchgemustert, wobei immer zwei benachbarte Elemente verglichen werden. Falls list[i] > list[i+1] ist, so erfolgt ein Austausch dieser beiden Elemente, und der Vergleichstest beginnt wieder bei list[0]. Dieser Prozeß läuft so lange ab, bis das gesamte Feld der Größe nach geordnet ist.

5.2.1. U 880

Die Länge des zu sortierenden Feldes wird der Prozedur 'sort' über das Register B mitgeteilt. Als Austauschanzeiger findet das Bit 0 des Registers C Verwendung.

```
simple_sort MODULE                      ;Modulvereinbarung

 CONSTANT                               ;Konstantenvereinbarung
   flag := 0

 EXTERNAL
   list ARRAY [10 WORD]                 ;zu sortierendes Feld

 INTERNAL

   sort PROCEDURE                       ;Prozedurvereinbarung
     ENTRY                              ;Beginn ausfuehrbarer Teil
     DO
       res flag,c                       ;Austauschanzeiger := nein
       ld ix,list                       ;Adresse list[0]
       ld d,b
       dec d                            ;Elementezaehler dekrementieren
       DO
          IF Z THEN EXIT FI             ;Feldende erreicht?
          ld h,(ix)
          ld l,(ix+1)                   ;hl:=list[i]
          ld a,(ix+2)
          ld e,a
          cp h
          ld a,(ix+3)                   ;ea:=list[i+1]
          IF Z THEN cp l FI             ;Vergleich byteweise
          IF C THEN                     ;wenn list[i]>list[i+1]
               ld (ix),e                ;Austausch der Elemente
               ld (ix+1),a
               ld (ix+2),h
               ld (ix+3),l
               set flag,c               ;Austauschanzeiger := ja
          FI
          inc ix
          inc ix                        ;naechstes Element (Wort)
          dec d                         ;Elementezaehler dekrementieren
       OD                               ;Ende der inneren DO-Schleife
       bit flag,c                       ;war Austausch?
       IF Z THEN EXIT FI
     OD                                 ;Ende der aeusseren DO-Schleife
   END sort                             ;Ende der Prozedur sort

 GLOBAL                                 ;neue Prozedurvereinbarung

   main PROCEDURE                       ;Hauptprogramm
     ENTRY                              ;keine lokalen Groessen
     ld b,10                            ;Feldgroesse
     call sort                          ;Aufruf der Prozedur sort
   END main                             ;Hauptprogrammende

END simple_sort
```

5.2.2. U 8000

Die Länge des zu sortierenden Feldes wird der Prozedur 'sort' über das Register R0 mitgeteilt. Der Austauschanzeiger ist eine lokale Größe (ein Byte) innerhalb der Prozedur 'sort':

```
simple_sort MODULE                          ! Modulvereinbarung !

 CONSTANT                                   ! Konstantenvereinbarungen !
   false:=0
   true :=1

 EXTERNAL
   list ARRAY [10 WORD]                     ! zu sortierendes Feld !

 INTERNAL

   sort PROCEDURE                           ! Prozedurvereinbarung !
     LOCAL                                  ! lokale Variablenvereinbarung !
       switch BYTE                          ! Austauschanzeiger !
     ENTRY                                  ! Beginn des ausfuehrbaren Teils !
     DO
       LDB switch,#false                    ! Initialisierung von switch !
       CLR R1                               ! Loeschen Feldzeiger i !
       DO
         CP R1,R0                           ! Feldende erreicht? !
         IF UGE THEN EXIT FI
         LD R2,R1                           ! Initialisierung Zeiger j !
         INC R2,#2                          ! j=i+1 (doppelt da Worte) !
         LD R4,list(R1)
         LD R6,list(R2)
         CP R4,R6                           ! wenn list[i]>list[j] dann !
         IF UGT THEN                        ! Austausch !
                LDB switch,#true
                LD list(R1),R6
                LD list(R2),R4
         FI
         INC R1,#2                          ! naechste Element (Wort) !
       OD                                   ! Ende innere DO-Schleife !
       CPB switch,#false                    ! war Austausch? !
       IF EQ THEN RET FI
     OD                                     ! Ende auessere DO-Schleife !
   END sort                                 ! Ende der Prozedur sort !

 GLOBAL                                     ! neue Prozedurvereinbarung !

   main PROCEDURE                           ! Hauptprogramm !
     ENTRY                                  ! keine lokalen Groessen !
     LD R0,#9*2                             ! Feldlaenge !
     CALL sort                              ! Aufruf der Prozedur sort !
     RET
   END main                                 ! Hauptprogrammende !

END simple_sort                             ! Modulende !
```

5.2.3. U 881/U 882

Die Länge des zu sortierenden Feldes wird der Prozedur 'sort' über die externe Größe limit mitgeteilt. Der Austauschanzeiger ist eine lokale Größe (ein Byte) innerhalb der Prozedur 'sort'.

```
simple_sort MODULE                          ! Modulvereinbarung !

 CONSTANT                                   ! Konstantenvereinbarungen !
   false:=0
   true :=1

 EXTERNAL
   list ARRAY [10 WORD]                     ! zu sortierendes Feld !
   limit BYTE                               ! Feldlaenge !

 INTERNAL

   sort PROCEDURE                           ! Prozedurvereinbarung !
     LOCAL                                  ! lokale Variablenvereinbarungen !
       i,j BYTE                             ! Elementezeiger !
       switch BYTE                          ! Austauschanzeiger !
     ENTRY                                  ! Beginn des ausfuehrbaren Teils !
       SRP #%10
       DO
         LD switch,#false                   ! Initialisierung Austauschanz. !
         CLR i                              ! Loeschen Feldzeiger i !
         DO
           CP i,limit                       ! Feldende erreicht? !
           IF UGE THEN EXIT FI
           LD j,i                           ! Initialisierung Zeiger j !
           INC j                            ! j=i+1 !
           LD R2,i
           ADD R2,R2
           LD R4,list(R2)
           LD R5,list+1(R2)                 ! R4/R5 = list[i] !
           LD R3,j
           ADD R3,R3
           LD R6,list(R3)
           LD R7,list+1(R3)                 ! R6/R7 = list[i+1] !
           CP R4,R6                         ! Vergleich byteweise !
           IF EQ THEN CP R5,R7 FI
           IF UGT THEN                      ! wenn list[i]>list[i+1] dann !
               LD switch,#true              ! Austausch !
               LD list(R2),R6
               LD list+1(R2),R7
               LD list(R3),R4
               LD list+1(R3),R5
           FI
           INC i                            ! naechstes Feldelement !
         OD                                 ! Ende der inneren DO-Schleife !
         CP switch,#false                   ! war Austausch? !
         IF EQ THEN RET FI
       OD                                   ! Ende der aeusseren DO-Schleife !
   END sort                                 ! Ende der Prozedur sort !

 GLOBAL                                     ! neue Prozedurvereinbarung !

   main PROCEDURE                           ! Hauptprogramm !
     ENTRY                                  ! keine lokalen Groessen !
       LD limit,#9                          ! Feldlaenge !
       CALL sort                            ! Aufruf der Prozedur sort !
       RET
   END main                                 ! Hauptprogrammende !

END simple_sort                             ! Modulende !
```

6. Programmentwicklungsbetriebssystem UDOS

In den vorherigen Abschnitten wurde die Programmierung von Mikroprozessoren sowie der dazugehörigen Peripheriebausteine und die Programmiersprache PLZ/ASM behandelt. Um komplette Programme für die oben angeführten Mikroprozessorsysteme erstellen zu können, wird eine Programmierumgebung benötigt. Dies kann sowohl ein Groß- oder Kleinrechner, als auch ein Mikrorechner mit der entsprechenden Softwareausstattung sein. Erfahrungen zeigen, daß ein ständig am Arbeitsplatz zur Verfügung stehendes Mikrorechnerentwicklungssystem mit der notwendigen Hard- und Softwareausstattung (IC-Emulator, Echtzeitspeicher, Entwicklungs- und Testsoftware u.a.m.) die besten Arbeitsmöglichkeiten bietet.
Eine besondere Rolle spielt hierbei das Betriebssystem, unter dem der Rechner arbeitet. Es bildet das Bindeglied zu den Ressourcen des Rechners und bestimmt weitgehend den Komfort und die Bedienerfreundlichkeit des Systems. Weiterhin beeinflußt das Betriebssystem ganz wesentlich die Portabilität von Programmen, d.h. die Übertragbarkeit auf andere Rechner, und den gegenseitigen Austausch von Programmen, Daten und Dokumentation auf Datenträgern (z.B.: Floppy-Disk oder Magnetband).
Internationale Beispiele für weitverbreitete Betriebssysteme auf dem Gebiet der Mikrorechner sind: CP/M, für die 8-Bit-Generation; MS/DOS, für 16-Bit-Personal-Computer und UNIX für 32-Bit-Arbeitsplatzstationen.
Im folgenden wird auf das Betriebssystem UDOS eingegangen, da es für U880-Rechner (z.B.: BC A5120, PC 1715 vom Kombinat Robotron und P8000 (8-Bit-Version) vom KEAW) verfügbar ist und da unter diesem Betriebssystem Entwicklungssoftware eines einheitlichen Konzepts für die Prozessoren: U880, U8000 und U881/U882 auf Basis der Sprache PLZ/ASM bis hin zu höheren Sprachen existiert.

6.1. Systemstruktur

Das Betriebssystem UDOS ist für eine U880-Rechnerkonfiguration mit Floppy-Disk als externem Speicher und einer Speicherausstattung von minimal 32KByte RAM konzipiert. Optimal sind dabei eine Ausstattung von 64 KByte RAM, da eine Reihe von Systemprogrammen (z.B.: Übersetzerprogramme für U8000, U881/U882, sowie Sprachübersetzer und Interpreter für höhere Sprachen) einen vollen Speicherausbau benötigen.
Alle wesentlichen Komponenten des Systems (auch das Betriebssystem selbst) liegen als Dateien auf Diskette vor und werden bei Bedarf in den RAM geladen und gestartet. Falls explizit keine andere Zuordnung angegeben wird, werden alle entstehenden (sowie temporären) Dateien auf Disketten abgelegt. Für die Eingabedaten wird vorausgesetzt, daß sie als Dateien bereits auf Disketten existieren. Beim Systemstart werden die für das Betriebssystem notwendigen Programme von einer Systemdiskette in den Speicher geladen und der Betriebssystemkern gestartet. Der ablaufende Initialisierungsprozeß ist vom Anwender an mehreren Stellen beeinflußbar. Die einfachste Art ist die Abänderung des Inhalts der Kommandodatei OS.INIT, deren Kommandos beim Systemstart automatisch ausgeführt werden. Dadurch ist eine elegante und schnelle Anpassung an besondere Gegebenheiten der Anwender möglich.
Der sich nach dem Systemstart ständig im Speicher befindliche Teil untergliedert sich in die folgenden drei Teile:

BFOS realisiert Monitorfunktionen und
den Systemstart; enthält E/A Grundmodule:
-Einzelzeichenein-/ausgabe
-Lesen und Schreiben ein Sektor
von/auf Diskette

OS Betriebssystemkern
organisiert Speicherverwaltung, variable
Zuordnung der E/A-Ströme:
enthält die internen Kommandos und den
logischen Bildschirmterminaltreiber

DOS eigenständiges System zur Dateiverwaltung,
realisiert Diskettenoperationen auf der Basis eines
Satzes von Standard Ein-/Ausgabeanforderungen

Diese drei Teile belegen nach dem Systemstart den Speicherbereich von 0 bis 3FFFH (hexadezimal). Zur Verwaltung der Diskettenbelegung organisiert DOS pro benutzter Diskette eine Belegungstabelle, die am Ende des zur Verfügung stehenden Speichers abgelegt wird. Daraus ergibt sich folgende Speicheraufteilung:

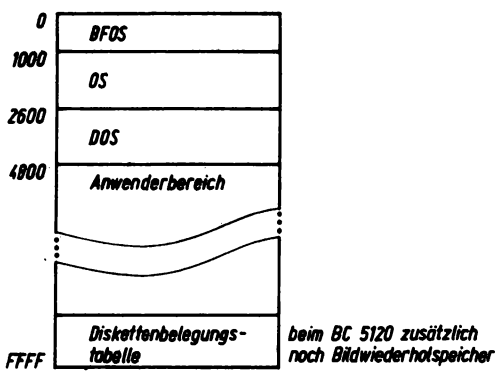


Bild 6.1. UDOS-Speicherbelegung

Eine Beschreibung der Monitorfunktionen und der internen Kommandos des Betriebssystemkerns wird im Abschn. 6.2.1. gegeben.
Alle weiteren Angaben beziehen sich auf die UDOS-Implementation zum System P8000 (8-Bit-Version). Unterschiede zu anderen Implementationen können im Diskettenformat und der DOS-Arbeitsweise (Zeigeranordnung) bestehen.
Das in UDOS verwirklichte Konzept der Dateiarbeit bringt für den Anwender eine Reihe von Vorteilen mit sich. Die gesamte Verwaltung der Daten auf den Disketten erfolgt automatisch (d.h., der Nutzer braucht sich über die Anordnung der Daten auf der Diskette nicht zu kümmern) auf der Basis von Dateinamen und einer Reihe von Standard-Ein-/Ausgabeanforderungen, wie z.B.:

Eröffnen von Dateien	(OPEN)
Lesen von Dateien	(READ)
Schreiben in Dateien	(WRITE)
Dateizeiger positionieren	(SKIP)
Streichen von Datensätzen,	(DELETE)
oder von Dateien	(ERASE)
Schließen von Dateien	(CLOSE)

Bei Dateioperationen wird ein Dateizeiger verwendet, der die aktuelle Position innerhalb der Datei, von der zu Lesen bzw. ab der zu Schreiben ist, kennzeichnet.
Dieses Prinzip wird konsequent bis hin zur Dateiarbeit innerhalb der höheren Programmiersprachen verwirklicht (s. Abschnitte 7.1.6. und 9.3.) und versetzt den Nutzer in die Lage, auf der Basis weniger Grundbausteine, umfangreiche Dateimanipulationen innerhalb seiner Anwenderprogramme relativ leicht realisieren zu können.

6.1.1. Dateien

Alle externen Daten und Programme werden in UDOS in Form von Dateien auf dem Speichermedium (Diskette) abgelegt und manipuliert. Eine Datei besteht aus mehreren Sätzen fester Länge. Ein Satz kann dabei einen oder mehrere Sektoren umfassen. Ein Sektor ist die kleinste Einheit, die von der Diskette gelesen oder auf sie geschrieben werden kann. Ein Sektor umfaßt 256 Datenbytes.
Ein Satz (auch Record genannt) kann sein:

1	Sektor	256	Zeichen	100 (hexadezimal)
2	Sektoren	512	"	200
4	"	1024	"	400
8	"	2048	"	800
16		4096	"	1000

Ein Satz wird mit einem Zugriff gelesen bzw. geschrieben, seine Länge wird im System als Recordlänge bezeichnet (Abkürzung: RL=100 bis RL=1000 hexadezimal). Sie kann vom Anwender beim Eröffnen einer Datei spezifiziert werden. Falls keine Angabe erfolgt, so wird mit der Recordlänge RL=100 gearbeitet. Die Recordlänge hat Einfluß auf die Zugriffszeiten, deshalb sollte bei großen Dateien mit einer Recordlänge größer 100 gearbeitet werden.

Der Zugriff erfolgt über die Angabe von Dateinamen, die vollständig oder abgekürzt angegeben werden können. Ein Dateiname unter UDOS hat folgendes Aussehen: (die eckigen Klammern kennzeichnen eine optionale Angabe)

Dateiname [[$Gerätetreiber:] Laufwerk/] Name

Gerätetreiber:
kennzeichnet das externe Medium (z.B.: Diskette: $NDOS, oder Magnetband: $MB, oder Treiber für andere Dateiformate auf Diskette: $MEOS, oder Drucker- bzw. Lochstreifentreiberprogramme); falls keine Angabe erfolgt wird $NDOS angenommen

Laufwerk:

kennzeichnet Laufwerksnummer (z.B.: 0, 1, 2), falls keine Angabe, so erfolgt Suche oder Ablage nach im System festgelegter Laufwerksreihenfolge

Name:

muß mit einem Buchstaben beginnen; kann aus Buchstaben, Ziffern und den Zeichen "." und "_" bestehen (max. 32 Zeichen), Groß- und Kleinbuchstaben werden unterschieden !
(z.B.: MEYER.PROJ.A_KONST.S)

Zusätzlich zum Dateinamen ist eine Datei durch Attribute gekennzeichnet, die in einem speziellen Beschreibungssatz (genannt: Descriptor-Record; immer 1 Sektor groß, unabhängig von der Recordlänge) abgelegt sind.
Dazu gehören:

Dateityp:		
'D'	Directory	(Inhaltsverzeichnis)
'A'	ASCII	(ASCII-Datei); Quellen, Texte u.a.
'B'	Binary	(Objektdateien); Daten, Objektkode
'P'	Procedure	(Maschinenkode); unter UDOS lauffähige Programme
Subtype:		
Zahl von 0 bis 15, zur Unterscheidung von Dateien eines Typs (z.B.: alle Ein-/Ausgabetreiber müssen vom Subtype 1 sein !)		
Dateieigenschaft:		
'W'	schreibgeschützt	(write protected)
'E'	löschgeschützt	(erase protected)
'L'	unveränderliche Eigenschaften	(properties locked)
'S'	geheime Datei	(secret)
'R'	wahlfreie Datei	(random)
'F'	Datei wird unabhängig von Speicherbelegung geladen (entspricht internem Force-Kommando)	

Die Organisation auf der Diskette wird im folgenden Abschnitt dargelegt.

6.1.2. Diskettenbelegung

Je nach eingesetzter UDOS-Implementation ist die Speicherkapazität pro Diskette unterschiedlich (bei 8"-Hardsector/einfache Aufzeichnungsdichte: 32 Sektoren pro Spur und 77 Spuren pro Diskette; bei 5,25" softsektor P8000: 32 Sektoren pro Spur (zweiseitig) und 80 Spuren pro Diskette).
Wesentlich bei UDOS ist die Verkettung der physischen Sätze (records) über zwei Zeiger (fore- und backpointer) und das Vorhandensein einer Belegungstabelle (bitmap) pro Diskette, in der die belegten Sektoren (pro Sektor ein Bit) gekennzeichnet sind. Dadurch wird erreicht, daß beim Streichen einer Datei die freiwerdenden Sektoren sofort für andere Dateien wieder zur Verfügung stehen. Durch das Prinzip der doppelt verbundenen Datensätze ist es durch Einführen einer Satzdistanz beim Schreiben möglich, optimale Zugriffszeiten zu erreichen.
Der Zugriff auf eine Datei erfolgt immer über das Inhaltsverzeichnis (directory) der Diskette, welches selbst wieder eine Datei ist und die Dateinamen sowie einen Verweis auf den entsprechenden Beschreibungssatz enthält. Die Inhaltsverzeichnisdatei ist 10 Sektoren groß und hat einen festen Platz auf allen Disketten.

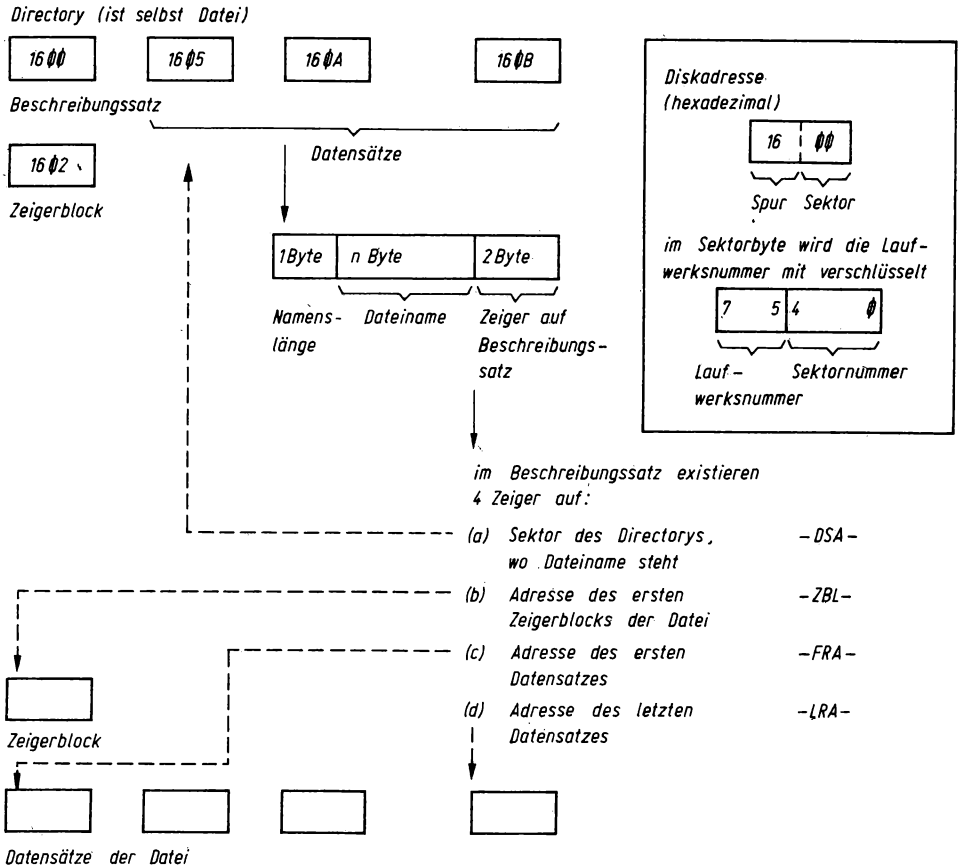
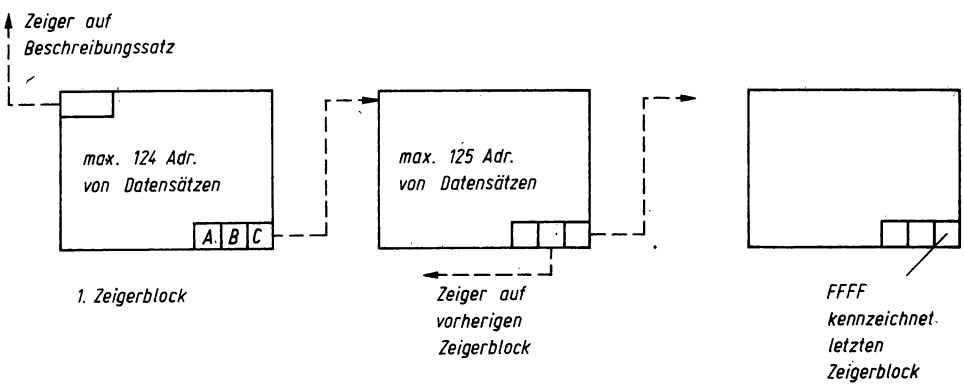


Bild 6.2. UDOS-Dateiorganisation



(A) relative Adresse der letzten Eintragung eines Datensatzes im Zeigerblock (zeigt auf 1. Byte der letzten Eintragung)

(B) Rückwärtszeiger auf vorherigen Zeigerblock (zeigt beim ersten Zeigerblock auf Beschreibungssatz)

(C) Vorwärtszeiger auf nächsten Zeigerblock (Eintrag: FFFF kennzeichnet letzten Zeigerblock)

Bild 6.3. NDOS-Zeigerblockorganisation

6.1.3. Ein-/Ausgabeanforderungen

Die Zuordnung der Ein-/Ausgabeströme erfolgt im Betriebssystem UDOS über logische Einheiten (Nr. 0 bis 20). Dadurch ist es z.B. möglich, für ein und dasselbe Programm zu verschiedenen Zeiten die Ausgabe auf verschiedene Geräte zu legen (z.B.: Bildschirm, Diskette, Drucker, ...). Die Umorganisation erfolgt über Betriebssystemkommandos (siehe dazu Abschn. 6.2.2. Kommandos: LADT, DEFINE, ACTIVATE und DEACTIVATE).
Die Zuordnung der logischen Einheit Nr. 0 kann nicht verändert werden. Nr. 1 dient als Konsoleingabe, Nr. 2 als Konsolausgabe und Nr. 3 ist als Druckerausgabe vorgesehen. Nr. 1 bis 3 sind nach dem Systemstart mit dem Terminaltreiber $CON verbunden (falls in der Initialisierungsdatei OS.INIT nichts anderes programmiert wurde!). Die restlichen Einheiten Nr. 4 bis 20 sind implizit mit dem Diskettentreiber $NDOS verbunden.
Die Verbindung des Systems mit einem Treiberprogramm erfolgt über einen Anforderungsvektor (Requestvektor). Das Register IY des U880 wird dabei als Zeiger auf diesen Vektor benutzt.

Aufbau des Anforderungsvektors

	Byte	
IY -->	0	logische Einheit (Nr.)
	1	Anforderungskode (request code) "Was ist zu tun ?"
	2,3	Datentransferadresse (Pufferbereich)
	4,5	Datenlänge
	6,7	Rückkehradresse nach Beendigung der Operation
	8,9	Fehlerrückkehradresse
	A	Fertigstellungskode (completion code) "Wie erfolgte Operation ?"
	B,C	Adresse eines Zusatzparameters, der weitere Spezifikationen enthält oder 2 Byte Zusatzinformation

Bit 0 des Anforderungskodes kennzeichnet zusätzlich die Art der Operation:
=0, d.h. Rückkehr erst nach Abschluß der Ein-/Ausgabeoperation
=1, d.h. Interrupt-Betriebsart.

Ein-/Ausgabeanforderungskodes (request codes)

0	Initialisierung	1A	Datei streichen
2	Zuweisung	1C	Lesen und Streichen
4	Eröffnung	1E	akt. Satz lesen
6	Schließen	20	vorherigen Satz lesen
8	Dateizeiger auf Beschreibungssatz	22	direktes Lesen
A	binäres Lesen	24	Zeiger vorwärts
C	ASCII-Lesen	26	Zeiger rückwärts
E	binäres Schreiben	28	Zeiger auf Dateiende
10	ASCII-Schreiben	2A	Umbenennung
12	akt. Satz schreiben	2C	Aktualisieren Attribute
14	direktes Schreiben	2E	Setzen von Attributen
16	Streichen	30	Abfrage Attribute
18	Rest streichen		

Beim Fertigstellungskode kennzeichnet das Setzen von Bit 6, daß ein Fehler auftrat. Die restlichen Bits geben die Fehlerart an. Falls die Operation fehlerfrei abgelaufen ist, so ist der Fertigstellungskode auf 80 (hexadezimal) zu setzen.

Fertigstellungskodes (completion codes)			
40	ungültiges Laufwerk	C6	Datenübertragungsfehler
41	ungültiges/inaktives Gerät	C7	Datei nicht gefunden
42	ungültige log. Einheit	C9	Dateiendefehler
43	Speicherschutzverletzung	CA	Zeigerfehler
44	fehlende/ungültige Operanden	CB	Datei nicht eröffnet
45	Systemfehler	CC	log. Einheit bereits offen
46	falscher Dateiname	CD	Zuweisungspuffer voll
47	nichtex. Kommando	CE	ungültige Laufwerksangabe
48	falscher Dateityp	CF	Tabelle log. Einheiten voll
49	Programmabbruch		
4A	nicht genügend Speicher	D0	Datei ex. bereits
4B	fehlende/ungültige Dateieigenschaften	D1	Diskettennamenfehler
		D2	ungültige Attribute
		D3	Diskette ist voll
80	fehlerfreie Operation	D4	Datei im falschen Satz
81	Formatfehler in Directory		der Directory
82	temp. Datei wurde erzeugt	D5	Dateibeginnfehler
83	Dateiname wurde verkürzt	D6	Datei bereits offen
84	Attributliste verkürzt	D7	ungültige Umbenennung einer temporären Datei
C1	ungültige Anforderung	D8	Dateieigenschaften sind
C2	log. Gerät nicht bereit		geschützt
C3	Schreibschutz	D9	ungültige Eröffnungs-
C4	Sektorlesefehler		anforderung
C5	Spurfehler		

Ein-/Ausgabetreiber die sich an dieses Konzept halten und die entsprechenden Anforderungen verarbeiten, sind relativ leicht ins Betriebssystem einzubinden. Beispiele dafür sind Lochband- und Druckertreiber sowie Transferprogramme für die Dateiübertragung über eine serielle Kopplung zu anderen Rechnersystemen.

6.2. Kommandos

UDOS besitzt zwei Arten von Kommandos: interne und externe. Während die internen Kommandos Bestandteil des Betriebssystemkerns OS sind, werden die externen Kommandos erst zur Abarbeitung geladen. Sie sind als lauffähige Programme (Dateien vom Typ "P") auf der Systemdiskette abgelegt. Somit kann der Kommandosatz jederzeit um selbstgeschriebene Funktionen erweitert werden.
Zum besseren Verständnis der Kommandos sollten sie mit unterschiedlichen Parameterkombinationen am Rechner ausprobiert werden, um deren Wirkung und mögliche Fehlermeldungen genau kennen zu lernen.
Das Zeichen ";" dient als Trennzeichen zwischen Kommandos, d.h. in einer Zeile können mehrere Kommandos hintereinander eingegeben werden. Ein Komma "," am Ende eines externen Kommandos bewirkt, daß es nur geladen wird, die Ausführung ist dann mit dem Xeq-Kommando zu erreichen. In der Zwischenzeit können interne Kommandos ausgeführt werden, beispielsweise die Initialisierung der Laufwerke nach einem Diskettenwechsel.

Bei der Angabe der Kommandosyntax besitzen folgende Zeichen eine besondere Bedeutung:

[...] in eckige Klammern eingeschlossene Teile können optional angegeben werden

(...)*	Teile, die mehrfach wiederholt oder weggelassen werden können
(...)+	Teile, die mehrfach wiederholt werden können aber mindestens einmal angegeben werden müssen
|	logisches Oder, es darf nur einer der beiden Teile angegeben werden

6.2.1. Interne Kommandos

Da die internen Kommandos Bestandteil des Betriebssystemkerns sind, können sie bei der Kommandoeingabe abgekürzt werden. Der unbedingt einzugebende Teil des Kommandos wird bei der Angabe des Kommandonamens groß geschrieben.

Allocate untere_Grenze obere_Grenze Blocklänge

Versucht einen Speicherblock der angegebenen Blocklänge zwischen den angegebenen Grenzen als besetzt zu kennzeichnen. Die Parameterwerte sind als Hexadezimalzahlen anzugeben, die Blocklänge wird auf Vielfache von 80h gerundet. (siehe dazu auch: DISPLAY-Kommando)

Brief

Umschalten in die sogenannte "kurze Betriebsart", d.h. keine Wiederholung des eingegebenen Kommandos. (siehe dazu auch: Verbose-Kommando)

Close * | E/A_Einheit

Bewirkt eine Abschluß-E/A-Anforderung an die angegebene logische Einheit, wird * angegeben, so werden alle logischen Einheiten abgeschlossen.

DEAllocate Blockadresse Blocklänge

Markiert einen Speicherblock der angegebenen Länge ab der spezifizierten Adresse als nicht besetzt. Die Parameterwerte sind als Hexadezimalzahlen anzugeben, die Blockadresse wird auf Vielfache von 80h abgerundet, die Blocklänge wird aufgerundet. (siehe dazu auch: DISPLAY-Kommando)

Debug

Bewirkt Übergang vom Betriebssystemniveau ins Monitorniveau. Auf dieser Ebene sind folgende Kommandos möglich: (UDOS P8000-Implementation !)

Q Rückkehr ins Betriebssytemniveau

O Start des Lade- und Initialisierungsprozesses für UDOS

R [Registerbezeichnung]
Anzeige aller U880 Register oder Anzeige eines Registerinhalts und Möglichkeit seiner Modifikation

D Adresse [Länge]
Anzeige eines Speicherinhalts und Möglichkeit zur Modifikation, oder Anzeige eines Speicherbereichs

M Zieladresse Quelladresse Länge
Transfer eines Speicherbereichs der angegebenen Länge von Quelladresse zur Zieladresse

PW Portadresse Bitmuster
Ausgabe des angegebenen Bitmusters (hexadezimalzahl) an die Portadresse

PR Portadresse
Lesen des Wertes, der an Portadresse anliegt

B [Prüfpunkt]
Unterbrechungspunkt löschen bzw. mit angegebener Adresse laden

N [Befehlsanzahl]
Ausführung von Befehlen im Einzelschritt

G [Adresse]
Sprung zur angegebenen oder im PC-Register eingetragenen Adresse, bei vorher gesetztem Unterbrechungspunkt wird angehalten

C Anfangsadr_1 Anfangsadr_2 Anzahl
Vergleich zweier Speicherbereiche

F Anfangsadr. Endadr. Bitmuster
Füllen eines Speicherbereichs

I Anzeige oder Ändern des Interruptstatus

GE Dateiname
Laden einer Procedure-Datei in den RAM

S Dateiname Anfangsadr. Endadr. (E=entry) (RL=Recordlänge)
Auslagern eines Speicherbereichs als Procedure-Datei

Force Kommandozeile
Bewirkt, daß alle in der Kommandozeile angegebenen Kommandos (bis zum ";" oder Zeilenende) geladen werden, auch wenn der entsprechende Speicherplatz belegt ist.

Initialize [Gerätename [Parameterliste]]
Sendet eine Initialisierungsanforderung an den angegebenen Gerätetreiber, oder ans Mastergerät (siehe MASTER-Kommando). Die Zusatzparameteradresse des Anforderungsvektors zeigt auf die Parameterliste. Nach jedem Diskettenwechsel unbedingt ausführen !

RELease
Freigeben des belegten Speicherbereichs durch ein zuvor geladenes Kommando.

Verbose
Umschalten in die sogenannte "lange Betriebsart", d.h. Wiederholung des eingegebenen Kommandos. Wichtig im Zusammenhang mit Kommandodateien (siehe DO-Kommando), da sonst auf der Konsole nicht sichtbar wird, welches Kommando gerade abgearbeitet wird.

Xeq [* | nn [Parameterliste]]
Bei Angabe von * : Start des zuletzt geladenen Programms mit der evtl. angegebenen Parameterliste; bei Angabe von nn Programmstart ab Adresse nn.

: Ausdruck
Berechnet hexadezimale Ausdrücke (max. vierstellig) von links nach rechts; es dürfen die Operatoren +, -, * und / verwendet werden.

6.2.2. Externe Kommandos

Bei der Angabe von Dateinamen als Parameter für die externen Kommandos kann unter Verwendung des Zeichens "*" eine abkürzende Schreibweise verwendet werden. (z.B.: * , d.h. alle Dateien; A*B, d.h. alle Dateien, die mit A beginnen und B enden; A*, d.h. alle Dateien, die mit A beginnen; *B entsprechend mit B enden)
Zur Abarbeitung der externen Kommandos muß eine Systemdiskette, die die Kommandos enthält aktiv sein. Eine Kommandoabkürzung ist hierbei nicht möglich, da es sich um Dateinamen handelt.

ACTIVATE $Gerätename [Eintrittspunkt]

Lädt das angegebene Gerätetreiberprogramm, führt eine Initialisierungsanforderung aus und trägt es in die interne Tabelle der aktiven Geräte ein. (siehe dazu auch LADT-Kommando)

CAT (Dateiname | T=Type | P=Eigensch. | D=Laufwerksnr. | F=L | L=Listingausgabe | DATE relop Datum | CDATE relop Datum)*

Ausgabe aller Dateien, die die spezifizierten Bedingungen erfüllen. Bei Angabe von F=L erfolgt Ausgabe im sogenannten "langen Format" (Größe, Recordlänge, Datum usw.). Bei Typen und Eigenschaften sind die unter 5.1.1. aufgeführten Abkürzungen in beliebiger Kombination verwendbar. Bei Eigenschaften kennzeichnet das Zeichen "&" in einer Kombination Dateien mit wenigstens einer dieser Eigenschaften und alleine angegeben alle Dateien. Mit L=... kann die Ausgabe umgelenkt werden (z.B.: L=CATLIST Erzeugen einer Datei, die Liste enthält oder L=$PRINTER Ausdruck der Liste). DATE steht für Datum der letzten Dateiänderung, CDATE für Erzeugungsdatum. Datum wird in der Form JJMMTT angegeben. Für relop kann stehen: =,<>,<,>,<=,>=.

COMPARE Datei1. Datei2

Byteweiser Vergleich der Datensätze der beiden Dateien. Unterschiede werden ausgegeben, falls keine Ausgabe erfolgt, liegt Identität vor.

COPY Datei1 Datei2 (A | U | O | RL=Recordlänge | T=Type)*

Kopieren einer Datei mit der Möglichkeit der Abänderung des Namens, der Veränderung der Recordlänge und des Typs. A steht für Anhängen an Datei2 (append), U für Einfügen am Anfang (update) und O für Überschreiben von Datei2 (open).

COPY.DISK [Quellaufwerksnr. TO Ziellaufwerksnr.] [V]

Physisches Kopieren des gesamten Diskette ninhalts von Quellaufwerk nach Ziellaufwerk (falls nichts angegeben wird: von 0 nach 1) mit anschließendem Vergleichslauf. Bei Angabe von V nur Ausführung des Vergleichslaufs.

DATE [JJMMTT]

Anzeige oder Neufestlegung des Systemdatums in der Reihenfolge Jahr, Monat, Tag.

DEACTIVATE $Gerätename

Bewirkt eine CLOSE-Anforderung an den Gerätetreiber, trägt ihn aus der internen Tabelle der aktiven Geräte aus und gibt den durch ihn belegten Speicherbereich wieder frei.

DEFINE (log_Einheit Dateiname | log_Einheit $Gerätename |
log_Einheit * | *)+ [A | O | U | I | NF | NO]

Verbindet eine logische Einheit (Nr.1 bis 20) mit aktivem Gerätetreiber bzw. Datei oder führt bei Angabe des Zeichens "*" zu dem Zustand nach der Systeminitialisierung. Die Optionen geben den Typ der Eröffnungsanforderung an, die dabei ausgeführt wird (A: append, O: output, U: update, I: input, NF: newfile und NO: no open-request).

DELETE (Dateiname | T=Type | P=Eigensch. | D=Laufwerksnr. |
Q=N | DATE relop Datum | CDATE relop Datum)*

Streichen von Dateien, die die spezifizierten Bedingungen erfüllen (analog CAT). Bei Angabe von Q=N erfolgt keine weitere Sicherheitsabfrage vor dem Streichen, sonst einzelne Abfrage, die zu quittieren ist.

DISPLAY

Ausgabe der Speicherbelegung. "A" steht dabei für belegter Sektor (128 Byte) und "." für einen freien Sektor.

DO Kommandodateiname (Parameter)*

Ausführung der in der Kommandodatei (ASCII-Datei) aufgeführten Kommandos. Innerhalb der Kommandodatei kann auf Parameter über #n zugegriffen werden (n = Position des Par.). Die bedingte Ausführung von Kommandos in Abhängigkeit von der Anzahl der Parameter wird in der Kommandodatei durch Einschließen von Kommandos in eckige Klammern erreicht. Das DO-Kommando kann rekursiv benutzt werden.

DUMP Dateiname [m [n]]

Ausgabe eines Speicherbelegungsbildes der angegebenen Datei. Mit m und n kann ein Teilbereich angegeben werden (von Satz m bis Satz n; Zahlenangabe dezimal !).

ECHO Zeichenkette

Bewirkt Ausgabe der angegebenen Zeichenkette über den Bildschirm. Wird in Zusammenhang mit Kommandodateien verwendet.

EXTRACT Dateiname

Ausgabe von Kenndaten des Beschreibungssatzes der Datei, wie Satzanzahl, Satzlänge u.a.m.

FORMAT (S | D=Laufwerksnr. | ID=Diskettenname | Q=N)*

Formatieren einer Diskette im angegebenen Laufwerk mit der Möglichkeit einen Diskettennamen zu vergeben. Bei Angabe von S wird sie als Systemdiskette formatiert (Q=N analog DELETE).

HELP (Kommando | *)*

Ausgabe von Erläuterungen zum angegebenen Kommando über den Bildschirm. Bei Angabe von * werden alle Schlüsselwörter aufgelistet, für die in HELP Erläuterungen existieren.

IMAGE Dateiname (erste_Speicheradr. letzte_Speicheradr.)+
[E=Eintrittspunkt] [RL=Satzlänge] [ST=Stackgröße]

Kopiert den Inhalt des angegebenen Speicherbereichs in die spezifizierte Datei (vom Typ "P"). Mit den Optionen können gleichzeitig Attribute der Datei gesetzt werden.

LADT

Listet die interne Tabelle der aktiven Geräte und die Zuordnung zu den logischen Einheiten aus.

MASTER [$Gerätename]

Ausgabe oder Setzen des sogenannten Mastergerätes, d.h. der Gerätetreiber mit dem gearbeitet wird, wenn bei Angabe des Dateinamens der Gerätetreiber fehlt.

MOVE (Dateiname | T=Type | P=Eigensch. | F=L |
D=Zielgerät | S=Quellgerät | L=Listingausgabe | Q=N |
DATE relop Datum | CDATE relop Datum)*

Kopieren von Dateien vom angegebenen Quellgerät (Laufwerksnr. oder Gerätetreiber) auf das angegebene Zielgrät, die die spezifizierte Bedingung erfüllen (analog CAT). Der Dateiname bleibt dabei unverändert.

PAUSE

Erwartet eine Tastatureingabe. Ist das eingegebene Zeichen "ESC", so wird der Rest der Kommandozeile ignoriert, sonst normales weitergehen. Wird in Kommandodateien angewendet.

RENAME (alter_Dateiname neuer_Dateiname |
$FLOPPY:Laufwerksnr. ID='neuer_Diskettenname')*

Umbenennung von Datei- bzw. Diskettennamen.

SET (Optionenliste)*

für Optionenliste kann stehen:
PROPERTIES OF Dateiname TO Eigenschaft
TYPE OF Dateiname TO Typ
SUBTYPE OF Dateiname TO Subtypnummer
LOW_ADDRESS OF Dateiname TO nnnn
HIGH_ADDRESS OF Dateiname TO nnnn
STACK_SIZE OF Dateiname TO nnnn

(nnnn - Hexadezimalzahl)
Weiterhin können verschiedene Systemparameter mit dem SET-Kommando gesetzt werden. Die vollständige Optionenliste kann über das HELP-Kommando ausgelistet werden.

STATUS [Laufwerksnr.]

Ausgabe der Diskettenstatistik (Belegung, Name usw.), bei Warnmeldungen muß die Diskette neu formatiert werden (vorher aber Dateien retten !).

An dieser Stelle sei nochmals darauf hingewiesen, daß nur durch ein bewußtes Verstehen der wichtigsten Vorgänge und Zusammenhänge innerhalb des Betriebssystems Fehlbedienungen und damit eventuell verbundene Programm- und Datenverluste vermieden werden können. Diese Kenntnis ist eine Voraussetzung für die umfassende Ausnutzung der vollen Leistungsfähigkeit des Systems.
Bei Erweiterungen des Systems oder bei selbst zu entwickelnder unter UDOS laufender Anwendersoftware trägt die konsequente Einhaltung der durch das Betriebssystem vorgegebenen Prinzipien wesentlich zur Transparenz und Zuverlässigkeit der Software bei.

6.3. Mikroprozessorentwicklungssoftware

Für die beschriebenen Mikroprozessortypen existiert unter dem Betriebssystem UDOS laufende Entwicklungssoftware (Assembler und Linker). In der vorliegenden Version gibt es für den U8000 und den U881/U882 nur Übersetzerprogramme, die die im Abschnitt 5. beschriebene PLZ/ASM-Struktur voraussetzen. Beim U880 werden dagegen über ein Vorübersetzerprogramm (FILTER) die höheren Sprachelemente von PLZ/ASM in Assembleranweisungen aufgelöst. Dieses so entstandene Quellprogramm muß dann mit dem U880-Assembler (ASM) übersetzt werden.

6.3.1. U 880

Das PLZ/ASM-Vorübersetzerprogramm für den U880 wird durch Eingabe von:

```
%FILTER Dateiname [(B)]
```

aufgerufen. Der PLZ/ASM-Quelltextdateiname muß die Endung ".ASM" haben. Die vom Filterprogramm erzeugte Assemblerquelltextdatei erhält die Endung ".S". Falls die Option B angegeben wird, so erscheinen die höheren PLZ/ASM-Anweisungen nicht mehr in der Ausgabedatei. Sonst werden sie als Kommentare mit übernommen.
Zu beachten ist, daß innerhalb des PLZ/ASM-Quelltextes die Marken mit einem Doppelpunkt beendet sein müssen, die reservierten Schlüsselwörter und die vom Filterprogramm erzeugten Markennamen nicht verwendet werden dürfen (siehe /2/).

Bei der Anwendung von PLZ/ASM zur Programmierung des U880 bilden die "reinen" Assemblerteile den Hauptteil des Programms. Deshalb werden im folgenden die wichtigsten Assemblervereinbarungen für den U880 angegeben.

Marken können aus Buchstaben, Zahlen, dem Fragezeichen (?) und dem Unterstreichungsstrich (_) zusammengesetzt sein. Sie müssen mit einem Buchstaben beginnen. Die ersten sechs Zeichen sind signifikant. Eine Marke kann, falls sie mit einem Doppelpunkt (:) abgeschlossen ist, in einer beliebigen Spalte beginnen. Wenn sie in Spalte 1 beginnt, dann braucht kein Doppelpunkt vorhanden zu sein.
Verschiedene Zahlendarstellungen werden durch Anhängen eines Zeichens gekennzeichnet: "B" für binär, "O" oder "Q" für oktal, "H" für hexadezimal und "D" oder ohne Zusatz für dezimal (z.B.: 10101100B, 17Q, 0AFH, 23).
Als Operanden können Zeichen und Zeichenketten, die in Hochkomma eingeschlossen sind, verwendet werden (z.B.: 'A' für ASCII-Wert 41H).
Innerhalb des Quellprogramms können auch Kleinbuchstaben verwendet werden. Dabei müssen jedoch Operationskodes und Schlüsselwörter entweder groß oder klein geschrieben werden. Markennamen, arithmetische Operatoren und Zahlen können gemischt dargestellt werden. Dabei sind dann z.B. Loop und LOOP zwei verschiedene Marken!
Das Assemblerprogramm bietet vielfältige Möglichkeiten, Ausdrücke zu bilden. Sie werden unter Berücksichtigung des Vorrangs der Operatoren von links nach rechts berechnet. Klammern können zur Beeinflussung der Berechnungsreihenfolge benutzt werden.

Operator	Funktion	Vorrang
+	unäres Plus	1
-	unäres Minus	1
.NOT. oder \	logische Negation	1
.RES.	Result	1
**	Potenzierung	2
*	Multiplikation	3
/	Division	3
.MOD.	Modulo	3
.SHR.	logische Rechtsverschiebung	3
.SHL.	logische Linksverschiebung	3
+	Addition	4
-	Subtraktion	4
.AND. oder &	logisches Und	5
.OR. oder !	logisches Oder	6
.XOR.	exklusives Oder	6
.EQ. oder =	gleich	7
.GT. oder >	größer als	7
.LT. oder <	kleiner als	7
.UGT.	vorzeichenloses größer als	7
.ULT.	vorzeichenloses kleiner als	7

Bei der relativen Adressierung ermittelt das Assemblerprogramm bei Angabe einer Marke automatisch die Differenz (z.B.: JR Z,MARKE). Das Symbol "$" kennzeichnet den augenblicklichen Stand des Befehlskodezählers (JR $-10).

Die Befehlsmnemoniken und die Operandennotation sind der Befehlsliste (Abschn. 2.1.6.) zu entnehmen. Zusätzlich dazu verarbeitet das Assemblerprogramm noch folgende Pseudooperationen:

[Marke:] DEFB Ausdruck
Generiert ein einzelnes Byte.

[Marke:] DEFW Ausdruck
Generiert ein einzelnes Wort (2 Byte).

[Marke:] DEFM 'Zeichenkette'
Generiert eine Folge von Bytes, die den ASCII-Wert eines jeden Zeichens der Zeichenkette widerspiegelt.

[Marke:] DEFT 'Zeichenkette'
Analog DEFM, zusätzlich wird am Anfang ein Byte eingefügt, das die Länge der Zeichenkette angibt.

[Marke:] DEFS Ausdruck
Reserviert die angegebene Anzahl von Bytes.

[Marke:] END
Kennzeichnet das Ende des Quellprogramms (kann auch weggelassen werden).

Name: EQU Ausdruck
Weist dem Namen den Wert des Ausdrucks zu. Der Name darf im Programm nicht neu definiert werden.

Name: DEFL Ausdruck
Analog EQU, dem gleichen Namen kann aber durch eine weitere DEFL-Anweisung ein neuer Wert zugewiesen werden.

GLOBAL Name1,Name2,...
Kennzeichnet die angegebenen Operanden als globale Größen, d.h. sie können in einem anderen Quelltextmodul als externe Größen verwendet werden.

EXTERNAL Name1,Name2,...
Kennzeichnet die angegebenen Operanden als externe Größen, d.h. sie sind in einem anderen Quelltextmodul definiert.

[Marke:] ORG Ausdruck
Setzt den Befehlskodezähler auf den Wert des Ausdrucks. Beachten Sie dabei den Unterschied von relativer und absoluter Übersetzung (siehe Option A beim Assembleraufruf).

[Marke:] COND Ausdruck
[Marke:] ENDC
Die zwischen COND und ENDC eingeschlossenen Quelltextzeilen werden nur übersetzt, wenn der Wert des Ausdrucks wahr, d.h. ungleich Null, ist (bedingte Übersetzung).

Makroname MACRO #P0,#P1,...,#Pn
[Marke:] ENDM
Die zwischen MACRO und ENDM eingeschlossenen Anweisungen bilden den zum Makronamen gehörenden Makrokörper. Makros müssen vor ihrem Aufruf definiert werden. Innerhalb des Körpers sind andere Makroaufrufe, einschließlich zu sich selber (Rekursion), erlaubt. (Beachten Sie in diesem Zusammenhang die Option M beim Assembleraufruf).

Zusätzlich existieren noch eine Reihe von Assemblersteueranweisungen. Diese haben als erstes Zeichen einen Stern ("*") und müssen in Spalte 1 beginnen.

*Eject	Veranlaßt einen Seitenvorschub im Listing.
*Heading Zeichenkette	Die angegebene Zeichenkette wird in jeder Kopfzeile des Listings untergebracht (max. 28 Zeichen).
*List OFF	Unterdrückt die Erzeugung eines Listings ab dieser Zeile.
*List ON	Ab dieser Zeile wird wieder ein Listing erzeugt.
*Maclist OFF	Innerhalb des Listings werden Makroexpansionen nicht ausgegeben. Nur die Makroaufrufe erscheinen.
*Maclist ON	Veranlaßt die Ausgabe der Makroexpansionen.
*Include Dateiname	Die angegebene Datei wird an der Stelle in den Quelltext eingefügt.

Die Pseudooperationen und die Steueranweisungen können sehr wirkungsvoll, vor allem in Kombinationen untereinander, zur besseren Gestaltung und Aufteilung der Quelltexte eingesetzt werden.

Der Aufruf des Assemblerprogramms erfolgt durch folgende Kommandozeile:

%ASM Dateiname+ [(Optionen)]

Es können ein oder mehrere Quelltextdateinamen angegeben werden. Diese müssen die Endung ".S" haben (sie braucht beim Aufruf nicht mit angegeben zu werden!). Das Assemblerprogramm erzeugt implizit eine Objektkodedatei (mit der Endung ".OBJ") und eine Listingdatei (Endung ".L"). Falls Optionen angegeben werden, so müssen sie in runde Klammern eingeschlossen sein. Die Reihenfolge ist dabei beliebig.
Folgende Optionen sind möglich:

M	Erlaubt Makrobearbeitung, ansonsten werden Makrodefinitionen und -aufrufe als Fehler angezeigt.
S	Es wird eine Symboltabelle erzeugt, die an den Objektmodul angefügt wird.
X	Veranlaßt die Erzeugung einer alphabetisch sortierten Crossreferenzliste am Ende der Listingdatei.
A	Veranlaßt die Übersetzung des Moduls mit absoluten Adressen.
NOL	Unterdrückt die Erstellung einer Listingdatei.
NOM	Unterdrückt die Ausgabe von Makroexpansionen.
NOO	Unterdrückt die Erstellung einer Objektkodedatei.
NOW	Unterdrückt die Ausgabe von Fehlerwarnmeldungen.
D=Zeichenkette	Die angegebene Zeichenkette (maximal 18 Zeichen) wird im Listingkopf jeder Seite untergebracht.
P	Veranlaßt die Ausgabe der Listingdatei über die logische Ein-/Ausgabeeinheit Nr. 3 des Betriebssystems.

Zusätzlich können noch die implizite Namensgebung von Ausgabedateien und temporären Dateien verändert werden:
I | L | M | O | T | X = Dateiname
(Zwischen-, Listing-, Makro-, Objektkode-, Symboltabellen- und Crossreferenzdatei)

Mit dem Verbinderprogramm (LINK) werden getrennt voneinander übersetzte Module zu einem lauffähigen Maschinenprogramm (Datei vom Typ "P") kombiniert. Der Aufruf erfolgt durch folgende Kommandozeile:

%LINK Modulliste+ [(Optionen)]

Dabei besteht die Modulliste aus einer Folge von Objektkodedateinamen (die Endung ".OBJ" braucht nicht mit angegeben zu werden), der optional eine absolute oder relative Adreßzuweisung der Form: $=nnnn oder $=$+nnnn (nnnn Hexadezimalzahl) vorangestellt sein kann. Falls ein Objektkodedateiname mit einem Minuszeichen beginnt, so wird dieser Modul nur zur Berechnung von externen Größen herangezogen (link only). Das entstehende Maschinenprogramm erhält, falls keine N-Option spezifiziert ist, den Namen des ersten angegebenen Moduls (ohne Endung). Zusätzlich wird eine Mapdatei (Endung ".MAP") angelegt, in der Adreßinformationen, Modulnamen, globale und externe Größen u.a. Linkinformationen zusammengefaßt sind. Folgende Optionen sind möglich:

E=n	Legt Startadresse des Maschinenprogramms fest (n Hexadezimalzahl oder globaler Name).
LET	Das Maschinenprogramm wird auch erzeugt, wenn Fehler beim Verbinden auftreten.
M=Dateiname	Legt Namen für Mapdatei fest.
N=Dateiname	Analog für entstehende Proceduredatei.
NOM	Unterdrückt die Erzeugung der Mapdatei.
NOW	Unterdrückt Warnmeldungen.
P	Veranlaßt die Ausgabe der Mapdatei über die logische Ein-/Ausgabeeinheit Nr. 3 des Betriebssystems.
RL=n	Legt Satzlänge für entstehende Proceduredatei fest.
ST=n	Legt die Größe des vom Maschinenprogramm benötigten Stackbereichs fest (implizit: 80h).
SY=Dateiname	
SY	Veranlaßt das Erstellen einer Symboldatei. Falls kein Dateiname spezifiziert ist, so wird der Datei die Endung ".SYM" gegeben.

6.3.2. U 8000

Der Aufbau von PLZ/ASM-Programmen und die Syntax wurde im Abschn. 5. erläutert. Die Befehlsmnemoniken und die Operandennotationen sind der Befehlsliste (Abschn. 3.1.6.) zu entnehmen. Weiterhin verarbeitet das Übersetzerprogramm noch folgende Assemblersteueranweisungen, die als erstes Zeichen ein "$" haben und in Spalte 1 beginnen müssen.

$ABS Adresse	Kennzeichnet den erzeugten Objektkode als absolut adressiert. Falls eine Adresse spezifiziert ist, so wird der Befehlskodezähler auf diesen Wert gesetzt (sonst 0). Die Anweisung ist bis zum Auftreten einer $REL-Anweisung gültig.
$REL Adresse	Kennzeichnet den erzeugten Objektkode als verschieblich. Falls eine Adresse angegeben ist, so kennzeichnet sie die relative Entfernung vom Anfang des aktuellen Bereichs (SECTION). Die Wirkung kann durch eine $ABS-Anweisung aufgehoben werden. (Implizit ist $REL 0 gesetzt!)
$SECTION Name	Verbindet den nachfolgend erzeugten Objektkode mit dem symbolischen Namen zur späteren Zuordnung in bestimmte Speicherbereiche (siehe ZLINK-Beschreibung). Die Anweisung ist bis zum Auftreten einer anderen $SECTION- oder der $DEFAULT-Anweisung gültig.
$DEFAULT	Beendet die Wirkung einer vorherigen $SECTION-Anweisung. Die implizite Speicherzuweisung (Datenvereinbarungen in Datenbereiche und Prozedurvereinbarungen in Programmbereiche) ist wieder wirksam.
$LISTON Ausdruck $LISTON	Die nach $LISTON folgenden Programmzeilen werden bis zum Auftreten einer weiteren $LISTON-Anweisung im Listing unterdrückt, falls der Wert des Ausdrucks wahr, d.h. ungleich Null, ist.

Zusätzlich können noch folgende erweiterte Befehle innerhalb einer Prozedur zur Erzeugung von Byte-, Wort- oder Longwortgrößen benutzt werden:

```
BVAL  konstanter_Ausdruck
WVAL  konstanter_Ausdruck
LVAL  konstanter_Ausdruck
```

Der Aufruf des Assemblerprogramms erfolgt durch folgende Kommandozeile:

```
%U8000ASM  Dateiname  [Optionen]
```

Der Quelltextdateiname muß die Endung ".S" haben (sie braucht beim Aufruf nicht mit angegeben zu werden). Das Assemblerprogramm erzeugt implizit eine Objektkodedatei (Endung ".OBJ") und eine Listingdatei (Endung ".L"). Folgende Optionen können nach dem Quellprogrammnamen in beliebiger Reihenfolge angegeben werden:

D=Zeichenkette	Die angegebene Zeichenkette (max. 18 Zeichen) wird im Listingkopf abgelegt (normalerweise für Datum u.ä. benutzt).
NOL	Unterdrückt die Erstellung einer Listingdatei.
P	Veranlaßt die Ausgabe der Listingdatei über die logische Ein-/Ausgabeeinheit Nr. 3 des Betriebssystems.

Es können noch die implizite Namensgebung der Zwischendatei (I=Dateiname), der Listingdatei (L=Dateiname) und der Objektdatei (O=Dateiname) verändert werden. Falls keine Objektdatei gewünscht wird, so ist O=$NULL anzugeben! Bei der Angabe des Objektkodes im Listing kennzeichnet das Hochkomma (') einen verschieblichen Wert und der Stern (*), daß der Wert von einer externen Größe abhängt.

Beim Erstellen der Quellprogramme sind die jeweiligen vom Übersetzerprogramm reservierten Worte (Operatornamen, Flagbezeichnungen, Assemblerbefehle, Bedingungskodes, Registerbezeichnungen, Schlüsselwörter der höheren Sprachelemente u.a.m.) und die Zeichen mit besonderer Bedeutung (z.B.: zur Kennzeichnung der Adressierungsart) zu beachten.
Bei Anwendung der Übersetzer- und Verbinderprogramme sind versionsbedingte Einschränkungen und Erweiterungen zu berücksichtigen, die den jeweiligen Handbüchern zu entnehmen sind.
Beim Arbeiten mit verschieblichen Modulen ist eine Neufestlegung der Adressen ohne nochmalige Übersetzung möglich. Weiterhin können die Programme in unabhängige Module aufgegliedert werden, die getrennt voneinander programmiert und übersetzt werden können. Dies verbessert erheblich die Strukturierung, die Dokumentation und die Wartung der Programme.

Das Verbinderprogramm (ZLINK) kombiniert separat erzeugte Objektmodule zu einem Lademodul (wieder ein Objektmodul) und erzeugt optional eine Mapdatei, die Informationen über den Linkprozeß enthält.
Dadurch ist ein sogenanntes "inkrementales" Verbinden möglich, d.h. eine Gruppe von Objektmodulen wird zu einem Modul verbunden, der wiederum als ein Objektmodul für weitere Linkanweisungen zur Verfügung steht.
Das Verbinderprogramm bietet weiterhin die Möglichkeit der Manipulation von Programm- und Datenbereichen. Durch die Steueranweisung $SECTION kann der Anwender Programmbereichen Namen geben, auf die beim Linkeraufruf Bezug genommen werden kann. Implizit separiert PLZ/ASM das Programm in die Bereiche "Modulname_D" für den Datenteil und "Modulname_P" für den Programmteil. Dieses Konzept ist für die Strukturierung von komplexen Programmen und zur Speicheraufteilung der Anwenderprogramme (z.B.: verschiedene Segmente beim U8001 oder Trennung in PROM- und RAM-Bereiche) nützlich.
Der Aufruf des Linkerprogramms erfolgt durch die Kommandozeile:

```
%ZLINK  Modulliste
```

Die Modulliste besteht aus Objektmodulnamen (Endung ".OBJ" braucht nicht mit angegeben zu werden!), Bereichsgruppierungsangaben und Optionen. Bereiche können durch Verwendung von runden Klammern zusammengefaßt werden.

```
(PROZESS_XY  DATEN_XY  GENERIERUNG_XY)
```

Solche Gruppen können wieder einen Namen erhalten, so daß sie in einem späteren Linklauf als einzelne Bereiche verwendet werden können.

```
ALLES_XY=( ... )
```

Zur Vereinfachung kann anstelle der vollständigen Angabe der Bereichsnamen

das Zeichen "*" zur Abkürzung verwendet werden.

```
ALLES_XY=(*_XY)
```

Die Anordnung der Gruppen und der Bereiche innerhalb der Gruppen entspricht der Reihenfolge der Angabe in der Kommandozeile.
Optionen werden durch ein Schlüsselzeichen gefolgt von einem Gleichheitszeichen gekennzeichnet.

L=Dateiname	Bewirkt das Erstellen einer Mapdatei mit dem spezifizierten Namen. (Implizit wird keine Mapdatei angelegt!)
E=Name	Festlegung des Startpunktes (Prozedurname oder Marke). Implizit ist es die Stelle im Kode, an der das erste ENTRY gefunden wird.
N=Dateiname	Festlegung des Dateinamens für den Lademodul (implizit ist es der erste Objektkodename, der ohne Endung ".OBJ" angegeben wurde).
T=Dateiname	Analog Assemblerbeschreibung für die temporäre Datei.
W=ON OFF	Bewirkt Ausgabe oder Unterdrückung von Warnmeldungen (implizit ist W=OFF gesetzt).
K= Namensliste K=ALL NONE K=ALLBUT Namensliste	Bewirkt, daß Informationen über die in der Namensliste angegebenen globalen Größen, über alle globalen Größen, über keine globalen Größen oder über alle nicht in der Namensliste aufgeführten globalen Größen im Lademodul abgelegt werden. Implizit ist K=NONE gesetzt.

Beispiel zur Anwendung von ZLINK:
Vorausgesetzt seien zwei Module mit folgenden Bereichen:

Modulname:	Dateiname:	Bereiche im Modul:
A	A.DATEI.OBJ	A_D
		STEUERUNG
B	B.DATEI.OBJ	B_D
		B_P

Dann werden durch:

```
%ZLINK A.DATEI B.DATEI  DATEN=(*_D) (STEUERUNG) L=* N=TEST.OBJ
```

die Datenbereiche zusammengefaßt und der Bereich STEUERUNG vom restlichen Programmbereich getrennt. Der Lademodul mit dem Namen TEST.OBJ besteht aus den Bereichen: DATEN (A_D und B_D), STEUERUNG und B_P. Zusätzlich wird die Mapdatei mit dem Namen A.DATEI.MAP angelegt.

Mit Hilfe des Imagerprogramms wird aus einem vom Assembler oder Linker erzeugten Objektmodul ein Maschinenprogramm mit absoluten Adressen erzeugt. Dieser Programmkode kann als Datei (vom Typ "P") abgelegt werden. Das Maschinenprogramm kann maximal 8000h Byte groß sein. Es verbleibt nach dem Imagerlauf im Speicher des Betriebssystems ab Adresse 4400H. Der Aufruf erfolgt durch folgende Kommandozeile:

```
%IMAGER  Dateiliste  Bereichsadressenliste  Fenster  [Optionen]
```

In der Dateiliste müssen die zu verarbeitenden Objektdateinamen exakt angegeben werden (es werden keine Endungen automatisch angehängt!). Die Bereichsadressenliste hat folgende Syntax:

[-][Segmentnummer=] ($=nnnn Bereichsliste)

Bei Angabe des Minuszeichens wird das folgende Segment nicht in den Speicher geladen. Die Segmentnummer hat nur für den U8001 eine Bedeutung. Mit $=nnnn erfolgt die Adreßzuweisung (hexadezimal) für die nachfolgend aufgeführten Bereiche. Falls einem Bereichsnamen ein Minuszeichen vorangestellt ist, so kennzeichnet dies, daß der Bereich nicht mit geladen wird. Das Fenster hat folgende Syntax:

{Beginnadresse Endadresse}

Damit wird der Bereich des zu ladenden Programms festgelegt. Falls die O-Option angegeben ist, so entsprechen die Werte der Low- und Highadresse der Prozedurdatei. Wenn kein Eintrittspunkt spezifiziert ist, so ist die Beginnadresse der Startpunkt.
Durch die Optionen: O=Dateiname und E=nnnn werden der Prozedurdateiname und der Eintrittspunkt spezifiziert.

Beispiel für Imageraufruf:

```
Durch     %IMAGER TEST.OBJ ($=100 DATEN $=500 STEUERUNG B_D)
           {100 1000} O=TEST E=500
```

erfolgt die Adreßzuweisung für die aufgeführten Bereiche (Daten ab Adresse 100H, Programmbereiche STEUERUNG und B_D liegen hintereinander ab Adresse 500H). Das erzeugte Maschinenprogramm (von Adresse 100H bis 1000H) wird in der Prozedurdatei TEST mit dem Eintrittspunkt 500H abgelegt.

6.3.3. U881/U882

Für diese Prozessoren erfolgt die Programmentwicklung analog dem vorherigen Punkt. Das Assemblerprogramm hat den Namen U8ASM. Bei den Assemblersteueranweisungen existieren einige Abweichungen (z.B.: $PAGE für Seitenvorschub und $IF... für eine bedingte Übersetzung - siehe /33/).
Der Assemblerbefehl LVAL ist im U8ASM nicht implementiert. Das Linker- und das Imagerprogramm sind die gleichen wie beim U8000. Bei der Separierung in Bereiche erfolgt durch den U8ASM eine Aufteilung in die drei Bereiche "PROGRAM", "REGISTER" und "DATA", deshalb muß bei der $SECTION-Anweisung der Bereich mit angegeben werden ($SECTION [Name] Bereich). Durch den hardwaremäßigen Unterschied des U881/U882 vom U8000 in bezug auf Programm- und Datenspeicher ist es beim U881/U882 nur gestattet, Datenbereiche mit anderen Datenbereichen, Registerbereiche mit anderen Registerbereichen und Programmbereiche mit anderen Programmbereichen zu kombinieren.

6.4. Utilitysoftware

Neben den Betriebssystemkommandos und den eben beschriebenen Übersetzerprogrammen für die drei Mikroprozessortypen existiert noch eine Vielzahl unter UDOS laufender Software für die verschiedensten Anwendungsfälle (z.B.: Reassemblerprogramme, Textverarbeitungssytem, Makroprozessor u.a.m.). Zwei für die Arbeit mit dem System wesentliche Hilfsprogramme sind das Editorprogramm und der Diskettenmonitor. Sie werden in den folgenden zwei Abschnitten beschrieben.

6.4.1. Editor

Das Editorprogramm dient zur Erstellung bzw. zur Korrektur von Quellprogrammen oder beliebigen Textdateien. Es ist zeilenorientiert (max. Zeilenlänge: 512 Zeichen) und arbeitet intern mit einem Zeilenzeiger, der auf die zuletzt angesprochene Zeile zeigt. Die Dateiarbeit erfolgt automatisch. Dabei wird mit einer selbsttätigen Pufferumschaltung gearbeitet, so daß es möglich ist Dateien beliebiger Größe zu verarbeiten. Ein sogenanntes Promptzeichen (">") zeigt an, daß man sich im Dialogniveau des Editors befindet. Im Eingabemodus wird kein Promptzeichen ausgegeben. Der Aufruf des Editors erfolgt durch folgende Kommandoeingabe:

```
%EDIT  [Dateiname] [Optionen]
```

Falls kein Dateiname angegeben wird, so erwartet das Programm als erstes die Namenseingabe (NAME?). Falls die angegebene Datei noch nicht existiert, so wird sie eröffnet und sofort in den Eingabemodus übergegangen. Falls die Datei bereits existiert, so wird eine Sicherheitskopie angelegt (mit dem gleichen Namen plus der Endung ".OLD") und der Editor meldet sich zum Dialog. Folgende Optionen können angegeben werden:

O=Dateiname1	die Sicherheitskopie erhält den angegebenen Namen
N	es wird keine Sicherheitskopie angelegt
RL=Satzlänge	die neue Datei wird mit der angegebenen Satzlänge (hexadezimal) eröffnet

Die Eingabe der 24 Editorkommandos kann entweder in Groß- oder Kleinbuchstaben erfolgen. Es gibt zwei Arten von Kommandoparametern:

eine Zahl n	Dezimalzahl, gibt an, wie oft die Operation zu wiederholen ist. (Ein "*" gibt an, daß die Operation über den gesamten Bereich beginnend mit der aktuellen Zeilenposition erfolgen soll)
eine Zeichenkette	zeigt an, daß ein Kommando so lange wiederholt werden soll, bis die Zeichenkette gefunden wurde. Begrenzer (definiert als erstes nichtleeres, nichtnumerisches Zeichen) müssen die Zeichenkette einschließen.

Kommandos und deren Parameter müssen wenigstens durch ein Leerzeichen getrennt werden. Das Editorprogramm kann in zwei Modi arbeiten, im sogenannten "brief-mode" (die Ausgabe einer Textzeile nach den Kommandos:

bottom, change, find, goto, get, locate, next und up wird unterdrückt), oder im "verbose-mode" (es erfolgt die Ausgabe der aktuellen Zeile nach den o.g. Kommandos).
Die kurze Betriebsart kann durch Eingabe des Brief-Kommandos, oder durch Anhängen eines Punktes an das Kommando (z.B.: F. /ab/) erreicht werden. Der unbedingt einzugebende Teil des Kommandos wird bei der folgenden Angabe wieder groß geschrieben.

Kommandobeschreibungen
Again [n] Wiederholt das vorherige Kommando n-mal. Ist n nicht angegeben, so wird das Kommando einmal wiederholt. Wird again nach join oder xecute benutzt, so wird nur das letzte Kommando wiederholt.
Bottom Setzt den aktuellen Zeilenzeiger auf die letzte Zeile.
BRief Bringt das Editorprogramm in die "kurze" Betriebsart (s.o.).
Change /alte_Zeichenkette/neue_Zeichenkette[/[n1[n2]]] Lokalisiert in dem durch n1 und n2 festgelegten Bereich die alte_Zeichenkette und ersetzt diese durch die neue_Zeichenkette. n1 legt die Anzahl der Zeilen fest, in denen die Ersetzung durchgeführt wird und n2 legt fest, wie oft pro Zeile die Änderung vollzogen werden soll (n1 n2 implizit = 1). Wenn keine neue_Zeichenkette angegeben ist, dann wird die alte_Zeichenkette gestrichen.
DElete [n | /Zeichenkette[/]] Streicht n Zeilen ab der aktuellen Zeile. Wenn eine Zeichenkette angegeben ist, so werden alle Zeilen bis zur Zeile, die die Zeichenkette enthält gestrichen.
Find /Zeichenkette[/] Setzt den aktuellen Zeilenzeiger auf die erste Zeile, die die Zeichenkette beginnend in Spalte 1 (z.B. als Marke) enthält.
GEt [Dateiname] Fügt den Inhalt der angegebenen Datei ein. Falls kein Dateiname spezifiziert wird, so wird die durch put oder putd erstellte temporäre Datei eingefügt.
Goto n Setzt den aktuellen Zeilenzeiger auf die durch n (Dezimalzahl) festgelegte Zeile.
Input [Textzeile] Falls eine Textzeile angegeben wird, so wird diese eingefügt. Ansonsten wird in den Eingabemodus übergangen, d.h. alle nachfolgenden Textzeilen werden übernommen. Durch Eingabe von zweimal CR wird der Eingabemodus verlassen.
Join &Kommando&[Kommando&]* Abarbeitung einer Folge von Kommandos. (entspricht einem Macro-Kommando sofort gefolgt von einem xecute-Kommando)

LIneno

Gibt die aktuelle Zeilennummer (dezimal) aus.

Macro &Kommando&[Kommando&]*

Lädt die angegebene Kommandofolge in den Makropuffer. Ausführung dieser Folge durch Eingabe des xecute-Kommandos. (Implizit enthält der Makropuffer: $U. 6$P 12$)

Next [n | /Zeichenkette[/]]

Setzt den aktuellen Zeilenzeiger n Zeilen nach unten, oder auf die erste Zeile, die die Zeichenkette enthält.

Print [n | /Zeichenkette[/]]

Gibt beginnend mit der aktuellen Zeile die nächsten n Zeilen, oder alle Zeilen bis zum ersten Auftreten der angegebenen Zeichenkette aus.

PUt [n | /Zeichenkette[/ [Dateiname [RL=m]]]]

Schreibt von der aktuellen Zeile ausgehend n Zeilen, oder alle Zeilen bis zum ersten Auftreten der angegebenen Zeichenkette in eine Datei. Wenn kein Dateiname angegeben wird, so wird eine temporäre Datei angelegt, wobei der vorherige Inhalt überschrieben wird.

PUTD [n | /Zeichenkette[/ [Dateiname [RL=m]]]]

Analog put, zusätzlich werden die übertragenen Zeilen im Arbeitsspeicher gestrichen.

QUIT

Schließt die Datei ab und die Steuerung geht wieder ins Betriebssystem über.

Replace [Textzeile]

Falls eine Textzeile angegeben ist, so wird die aktuelle Zeile durch diese ersetzt. Ansonsten wird in den Eingabemodus übergegangen, d.h. alle nachfolgend eingegebenen Zeilen werden anstelle der gestrichenen Zeile eingefügt.

Top

Setzt den aktuellen Zeilenzeiger an den Anfang der Datei.

Up [n | /Zeichenkette[/]]

Erniedrigt den Wert des Zeilenzeigers um die angegebene Zahl n, oder setzt den Zeilenzeiger auf die erste davorliegende Zeile, die die angegebene Zeichenkette enthält.

Verbose

Aufhebung des "brief-mode".

Window

Zeigt die erste und letzte Zeilennummer des momentan im Arbeitsspeicher befindlichen Blocks an.

Xecute

Bewirkt die Ausführung einer über das macro-Kommando aufgebauten Kommandofolge.

6.4.2. Diskettenmonitor

Dieses Programm mit dem Namen FILE.DEBUG hilft zerstörte Dateistrukturen (durch Übertragungsfehler entstanden oder durch Software produziert) zu reparieren. Durch nicht sachgemäßen (sorgfältigen) Gebrauch der im folgenden angeführten Kommandos ist es aber auch leicht möglich, eine Datei oder eine ganze Diskette zu zerstören. Richtig angewendet kann dieses Programm in fast aussichtslos aussehenden Situationen die letzte Rettung sein.

Das Programm stützt sich auf folgende Variablen:

DA	aktuelle Diskadresse (2 Byte, high: Spurnummer, low: Bit 0-4: Sektornr., Bit 5-7: Laufwerksnr.)
ZBL	Adresse des ersten Zeigersektors der Datei (siehe dazu Bilder 6.2 und 6.3)
RL	Satzlänge

Es finden vier interne Pufferbereiche Anwendung:

Schreib-/Lesepuffer	(Adr. = Linkadr. + 0C00h)
Maskierungspuffer	(Adr. = Linkadr. + 0900h)
Vergleichspuffer	(Adr. = Linkadr. + 0A00h)
Zeigerblockpuffer	(Adr. = Linkadr. + 0B00h)

Die Linkadresse ist meistens 4000h. Implizit wird mit dem Laufwerk Nr. 0 gearbeitet. Die Werte der Variablen sind als Hexadezimalzahlen anzugeben, bzw. werden als Hexadezimalzahlen ausgegeben.
Der Aufruf des Programms erfolgt durch Eingabe von:

```
%FILE.DEBUG
#
```

Die Ausgabe des Zeichens # zeigt an, daß eine Kommandoeingabe erwartet wird. Es reicht die Eingabe des ersten Buchstabens des Kommandos.

Kommandos
Read [DA [RL]] Liest einen durch den aktuellen Wert von FORE adressierten (oder von der angegebenen Diskadresse) Satz der angegebenen Länge (impl. 100h) in den Schreib-/Lesepuffer. Eingabe von CR (return) entspricht einem READ-Kommando ohne Parameter. Eingabe von (CR) entspricht einem READ-Kommando ohne Parameter, aber in umgekehrter Richtung (d.h. beginnend bei BACK).
Write [DA [RL]] Schreiben eines Satzes vom Schreib-/Lesepuffer auf die aktuelle Diskadresse (oder die angegebene Diskadresse). Vor der Schreiboperation erfolgt nochmals eine Sicherheitsabfrage, die mit Y zu quittieren ist.
Open Dateiname Sucht nach der angegebenen Datei und liest, falls vorhanden, ihren Beschreibungssatz in den Schreib-/Lesepuffer ein. Zusätzlich werden die Zeigerwerte DSA, FRA und LRA (siehe Bild 6.2.) ausgegeben.

Trace [Dateiname | ↑] Fortlaufende Leseoperation bis ein Zeigerwert Null ist oder ein Zeigerfehler auftritt. Wenn ein Dateiname angegeben wird, so erfolgt vor dem Lesen eine OPEN-Operation. Die Angabe eines bewirkt einen rückwärts ablaufenden Suchlauf.
List Gibt die aktuellen Werte von DA und RL aus.
Debug Übergang in das Monitorniveau. Dort ist die Möglichkeit der Veränderung der Pufferinhalte gegeben. Rückkehr wieder durch Eingabe von Quit.
Quit Rückkehr ins Betriebssystemniveau.
Search Sucht auf der Diskette nach Sektoren, deren Inhalt mit dem Inhalt des Vergleichspuffers identisch ist. Dabei werden nur die Bits verglichen, die im Maskierungspuffer gesetzt (=1) sind. Der Suchprozeß kann jederzeit durch Eingabe von ESC abgebrochen werden.

6.4.3. Hinweise auf UPROG und UFORM

An dieser Stelle sei noch auf zwei Programme kurz hingewiesen, die im Zusammenhang mit der Softwareentwicklung unter dem Betriebssystem UDOS genutzt werden können.

Zur Programmierung von verschiedenen EPROM-Typen dient das Programm UPROG. Das Quellprogramm besteht aus PLZ/SYS-Modulen, hardwareabhängige Teile sind als Assemblerprogramme eingebunden. Somit besteht die Möglichkeit der Anpassung an unterschiedliche EPROM-Programmierhardware.

Das Programm UFORM dient zur Textverarbeitung. Mit ihm können Textdateien formatiert werden (z.B.: Briefe, Dokumentationen). Der formatierte Text kann als Datei abgelegt werden. Die zur Formatierung notwendigen Steuerkommandos werden als sogenannte Punktkommandos (erstes Zeichen ein Punkt) in den Text integriert. Zur Formatsteuerung existieren vielfältige Möglichkeiten:

- Spezifizierung der Seitengröße (Länge, Breite, Rand)
- Angabe von Fuß- und Kopfzeilen (links-/rechtsbündig oder zentriert)
- Ausrichten des Textes (rechter Randausgleich)
- Auffüllen des Textes
- Zeilen einrücken (gesamte Text oder nur eine Zeile)
- Zeilen zentrieren
- Unterbrechen der Ausgabe (z.B. für Typenradwechsel)
- Einzelseitenausgabe

  u.a.m.

7. U 880 BASIC-Interpreter

Die Programmiersprache BASIC (Abkürzung für Beginner's All purpose Symbolic Instruction Code) ist eine der am weitesten verbreiteten Dialogsprachen. Neben den Programmanweisungen, in denen die Aufgabe formuliert wird, gibt es eine Reihe von Kommandos, die das Erstellen, Testen, Ändern und Abarbeiten der Programme im Dialogbetrieb ermöglichen. Dieses Konzept gestattet einen schnellen und unkomplizierten Einstieg in die Programmierung. Da das Programm mittels eines Interpreters ausgeführt wird, kann es ohne vorherige Übersetzung abgearbeitet werden.
Durch die Verarbeitung von arithmetischen Ausdrücken und der Möglichkeit der Zeichenkettenmanipulation eignet sich BASIC sowohl für wissenschaftlich-technische als auch für kommerzielle Aufgabenstellungen kleinen bis mittleren Umfangs. Die hier beschriebene BASIC-Version für den U880 unter dem Betriebssystem UDOS nutzt eine Vielzahl der Möglichkeiten des Systems (z.B.: Dateiarbeit, Zuweisung von Ein-/Ausgabeströmen u.a.m.) konsequent aus und gestattet daher dem Anwender ein relativ einfaches Einbinden dieser Fähigkeiten in seine Programme.

7.1. Sprachelemente

Ein BASIC-Programm ist zeilenorientiert. Jede Zeile beginnt mit einer Zeilennummer. Beim Start eines Programms arbeitet der Interpreter die Programmzeilen in der Reihenfolge ihrer Zeilennummern unabhängig von der Stellung im Quelltext ab (Ausnahmen: Sprünge, Unterprogramme, Schleifen). Falls eine Programmzeile geändert werden soll, so ist eine Zeile mit der gleichen Zeilennummer neu einzugeben. Bei der Eingabe der Programmzeilen erfolgt eine Prüfung auf syntaktische Richtigkeit. Sinnvollerweise werden die Zeilen mit einem Inkrement größer 1 numeriert, um die Möglichkeit des Einfügens von Befehlsfolgen zu erhalten. Die Eingabe nur einer Zeilennummer streicht die entsprechende Programmzeile. Die Anweisungen haben ein freies Format, d.h. Leerzeichen werden ignoriert. Der Aufruf des BASIC-Interpreters erfolgt vom UDOS aus durch Eingabe folgender Kommandozeile:

```
%BASIC
 (Meldung von BASIC)
 ...
 ...
>
```

Der Interpreter zeigt seine Dialogbereitschaft durch Ausgabe des Zeichens ">" an. Eine erwartete Eingabe während der Ausführung einer INPUT-Anweisung wird durch das Zeichen "?" angezeigt. Fehler- und Warnmeldungen werden durch eine Nummer gekennzeichnet. Die Erläuterungen dazu sind dem Anwenderhandbuch zu entnehmen.
Der BASIC-Interpreter arbeitet intern mit einem Gleitkommapaket, das eine 13stellige Mantisse realisiert.

Kommandos realisieren Steuerfunktionen, die Aktionen werden direkt ausgeführt. Anweisungen dagegen dienen zur Formulierung des Problems, sie werden erst bei Programmabarbeitung ausgeführt. Ähnlich den Kommandos

können aber auch einige Anweisungen für Testzwecke ebenfalls direkt abgearbeitet werden. Sie müssen dann ohne Zeilennummer eingegeben werden.

7.1.1. Kommandos

Die Kommandos unterteilen sich in drei Gruppen: Programmausführungs-, Editier- und diskettenorientierte Kommandos.

Programmausführungskommandos	
RUN[-Zeilennummer]	Start eines BASIC-Programms bei der ersten ausführbaren Anweisung, oder falls angegeben bei der spezifizierten Programmzeile.
CONTINUE[-Zeilennr]	Weiterlauf eines unterbrochenen Programms (durch eine STOP-Anweisung oder Betätigen der DEL-Taste) bei der nächsten Anweisung, oder falls angegeben bei der spezifizierten Programmzeile.
STEP	Weiterlauf des Programms bis zum Beginn der nächsten Anweisung. ("Einzelschrittbetrieb")
QUIT	Verlassen des BASIC-Interpreters und Rückkehr ins Betriebssystemniveau.

Die folgenden Editierkommandos beziehen sich immer auf das aktuelle Programm, das sich im Arbeitsspeicher (RAM) des BASIC-Interpreters befindet. Groß-/Kleinschreibung siehe Bemerkung auf Seite 131.

Editierkommandos	
LISt[-Bereich]	Vollständiges oder teilweises Auslisten des aktuellen Programms: LIS-n — nur der Programmzeile n LIS-n, — alle Programmzeilen ab n bis zum Ende LIS-,n — alle Programmzeilen von der ersten bis zur Zeile n (einschließlich) LIS-n,m — von Programmzeile n bis m (einschließlich) LIS-, — des gesamten Programms
NEW[-Pufferanz.]	Löschen des aktuellen Programms. Dabei können für die Dateiarbeit benötigte Puffer (0 bis 15) zu je 512 Byte reserviert werden.
DELete[-Bereich]	Streichen von Programmzeilen. Der Bereich kann analog zum LIST-Kommando angegeben werden.
RENUMBER[-neuerst[,delta[,alterst[,altletzt]]]]	Umnumerierung eines gesamten Programms, oder falls angegeben eines durch alterst und altletzt spezifizierten Programmbereichs. Dabei geben neuerst die neue erste Zeilennummer und delta die Schrittweite an. Falls sie nicht spezifiziert werden, so wird implizit mit den Werten 10 gearbeitet. Bei Ausführung dieses Kommandos muß das Programm RENUMBER.PROG.BP auf der Diskette verfügbar sein. Das Kommando erstellt die Dateien PROG.BEFORE.REN.BP (enthält das alte Programm) und NEW.REN.PROG.BP (enthält das umnumerierte Programm).

SIZE	Ausgabe des Status des aktuellen Programms. Dabei werden die Anzahl der verfügbaren Bytes (AVAIL=...); die Größe des durch das Programm (PROG=...), sowie durch Variable (VAR=...) belegten Speichers in Bytes und die Anzahl der angelegten Pufferbereiche ausgegeben.
CLEAR	Rücksetzen aller durch ein Programm realisierten Zuweisungen (Variable, Funktionsaufrufe, Schließen aller Dateien usw.). CLEAR wird vom RUN-Kommando automatisch generiert.

Um eingegebene BASIC-Programme für eine spätere Verwendung zu speichern und vorhandene Programme zu laden, existieren eine Reihe von diskettenbezogenen Kommandos. Die so erstellten Dateien erhalten zur zusätzlichen Kennzeichnung dabei automatisch die Endung ".BP". Beim Laden dieser Programme wird die Endung automatisch ergänzt, d.h. sie darf nicht mit angegeben werden.

Diskettenbezogene Kommandos	
SAVE-Programmname	Speichert eine Kopie des aktuellen Programms in verdichteter Form unter dem angegebenen Dateinamen auf Diskette ab. Diese Form ist kompakter als die lesbare Form des Quellprogramms und erlaubt somit ein schnelleres Laden.
ASAVE-Programmname	Speichert eine Kopie des aktuellen Programms in Quelltextdarstellung unter dem angegebenen Dateinamen auf Diskette ab. Die so erstellten Dateien können mit UDOS-Dienstprogrammen (z.B.: Editor) weiter bearbeitet werden.
RSAVE-Programmname	Analog SAVE und ASAVE, die angegebene Datei muß bereits existieren. Sie wird überschrieben.
GET-Programmname	Laden des angegebenen BASIC-Programms von Diskette in den Arbeitsbereich.
XEQ-Programmname	Laden des angegebenen BASIC-Programms und starten des Programms. Entspricht GET-Kommando gefolgt von RUN.
APPEND-Programmname	Anhängen des angegebenen Programms an das aktuelle Programm im Arbeitsspeicher. Das zu ladende Programm muß in Quelltextdarstellung (ASAVE-Form) vorliegen.

Um sich vom BASIC-Niveau aus einen Überblick über vorhandene Dateien und deren Inhalt verschaffen zu können, dienen die Programme CAT.BP (Auflisten von Dateien) und FLIST.BP (Ausgabe des Inhalts einer Datei). Die Programmlistings sind im Abschn. 7.3. angegeben. Die Programme können mit dem XEQ-Kommando vom BASIC-Interpreter aus abgearbeitet werden.

7.1.2. Ausdrücke

Ausdrücke kombinieren Konstanten, Variablen oder Funktionen mit Operatoren in einer geordneten Reihenfolge.

Als Konstanten können Zahl- oder Zeichenkettenkonstanten verwendet werden. Zahlkonstanten (Integer- oder Gleitkommazahlen) können im Bereich von 10 hoch -128 bis 10 hoch +128 liegen. Zur Darstellung von Gleitkommazahlen kann auch die E-Notation (z.B.: 12.34567E-89) verwendet werden.
Zeichenkettenkonstanten bestehen aus einer Folge von ASCII-Zeichen (außer " und <), die in Anführungszeichen eingeschlossen sind (z.B.: "HALLO").

Eine Variable ist ein Name, dem ein Wert zugewiesen werden kann. Es gibt numerische- und Zeichenkettenvariablen. Numerische Variablen unterteilen sich in reelle Variablen (durch einen einzigen Buchstaben A...Z oder einen Buchstaben direkt gefolgt von einer Ziffer A0...Z9 gekennzeichnet) und ganzzahlige Variablen (Kennzeichnung analog den reellen Variablen, aber mit dem Zusatz "%", z.B.: I1%). Zeichenkettenvariablen können als Wert eine Zeichenkette besitzen. Ihre Kennzeichnung ist analog den reellen Variablen, aber mit dem Zusatz "$", z.B.: A0$. Zeichenkettenvariablen müssen vor ihrer Benutzung durch eine DIM-Anweisung vereinbart werden.
Die Einschränkung der Variablennamen auf einen Buchstaben eventuell noch durch eine Ziffer ergänzt behindert die Selbstdokumentation der Variablenbezeichnungen. Deshalb sollten ausreichend Kommentare zur Variablenverwendung in die Quelltexte integriert werden.

Eine Variable kann ein Feld (maximal zweidimensional) kennzeichen. Felder können numerische Werte (ganzzahlig oder reell) oder Zeichenketten enthalten. Auf die Elemente kann über Indizierung zugegriffen werden. Als Index kann eine ganzzahlige Konstante, eine Variable oder ein Ausdruck, der auf einen ganzzahligen Wert gerundet wird, auftreten.
Eine einfache numerische Variable kann denselben Namen haben wie ein numerisches Feld, ohne daß dies zu Komplikationen führt. Numerische Felder müssen vor ihrer Benutzung durch eine DIM-Anweisung vereinbart werden, wenn die Dimension größer als 10 Zeilen (bei eindimensionalen Feldern) oder größer als 10 Zeilen und 10 Spalten (bei zweidimensionalen Feldern) ist. Der gerundete Wert eines Indizies muß ein Wert zwischen 1 und dem maximalen Zeilen- oder Spaltenwert sein.
Zeichenkettenfelder können nur eindimensional sein. Sie müssen immer durch eine DIM-Anweisung vereinbart werden. Der Name einer einfachen Zeichenkettenvariablen darf im Gegensatz zu den numerischen Variablen nicht mit dem Namen eines Zeichenkettenfeldes übereinstimmen!

Eine Funktion ist eine durch einen Namen gekennzeichnete Operation, die einen oder mehrere Parameterwerte benutzt, um **einen einzelnen** Wert zu berechnen.
Funktionsnamen können aus mehreren Buchstaben bestehen. Der Anhang "%" kennzeichnet eine ganzzahlige numerische Funktion und der Anhang "$" eine Zeichenkettenfunktion.
Da eine Funktion nur einen Wert liefert, kann sie in Ausdrücken wie eine Konstante oder Variable verwendet werden. Der Aufruf einer Funktion erfolgt durch Angabe des Namens, gefolgt von den in runden Klammern eingeschlossenen aktuellen Parametern (z.B.: INT(A)).
Dem Anwender stehen eine Reihe von fest eingebauten Funktionen (siehe

Abschn. 7.1.5.) zur Verfügung. Für spezielle Fälle können auch selbst Funktionen definiert werden. Neben den numerischen Funktionen sind vor allem die Zeichenkettenfunktionen hervorzuheben.

Operatoren führen eine mathematische, logische oder Zeichenkettenoperation auf einen oder zwei Operanden aus und liefern einen Wert.

Art	Notation	Ergebnis
arithmetische Operatoren	+ - * / ↑	numerischer Wert
Vergleichsoperatoren	= < > <= >= <> (ungleich)	0 für falsch oder 1 für wahr
logische Operatoren	& (UND) ! (ODER) ~ (KOMPLEMENT)	0 für falsch oder 1 für wahr
Zeichenkettenoperator	+ (Verkettung)	Zeichenkette

Die Reihenfolge der Ausführung der Operationen wird durch die Rangordnung der Operatoren bestimmt.

Hierarchie der Operatoren	
unäres +, unäres -, ~ ↑ *, / binäres +, binäres - Vergleichsoperatoren (=,<,>,<=,>=,<>) &, !	höchste ▽ niedrigste

Bei gleicher Rangordnung erfolgt die Berechnung von links nach rechts. Runde Klammern können verwendet werden, um die Berechnungsreihenfolge zu verändern.

7.1.3. Anweisungen

Der Programmalgorithmus wird durch Anweisungen realisiert. Ein Grundelement ist dabei die Wertzuweisung. Mit ihr wird einer Variablen ein Wert zugewiesen. Sie hat folgende Syntax:

```
Variable = Ausdruck          oder
LET Variable = Ausdruck
```

Das Wort LET kann bei der Eingabe weggelassen werden. In einer Anweisung (Zeile) können mehrere Zuweisungen auftreten, wenn sie durch Kommas getrennt werden. Das Gleichheitszeichen ist hierbei kein Vergleichsoperator, sondern ein Zuweisungsoperator, d.h. die Variable auf der linken Seite erhält den Wert des Ausdrucks auf der rechten Seite.

Die END/STOP-Anweisungen beenden die Programmabarbeitung. Sie bestehen nur aus den Wörtern:

```
END
STOP
```

Die END-Anweisung bewirkt ein Abschließen aller Dateien (close-request) und führt zur Ausgabe von "READY" und Anzeige der Dialogbereitschaft. Die STOP-Anweisung hält den Programmablauf an. Es erfolgt die Ausgabe von "STOP AT nnnn" (nnnn - Zeilennummer der STOP-Anweisung). Das Programm kann mit Hilfe eines Programmablaufsteuerkommandos an dieser Stelle fortgesetzt werden.

Ein wesentliches Element von Programmen sind Schleifenanweisungen, mit denen eine Gruppe von Anweisungen wiederholt ausgeführt werden kann. Im BASIC existiert hierfür die FOR/NEXT-Anweisung. Sie hat folgende Syntax:

```
FOR Variable_A = Ausdruck_1 TO Ausdruck_2  [STEP Ausdruck_3]
      .
      .
NEXT Variable_A
```

Die Variable_A ist dabei entweder eine reelle oder ganzzahlige Variable. Sie erhält zu Beginn den Wert vom Ausdruck_1. Die zwischen der FOR- und NEXT-Anweisung liegenden Anweisungen werden iterativ so lange ausgeführt, bis der Wert der Variablen_A den Wert des Ausdrucks_2 überschreitet. Falls kein STEP-Element (Schrittweite) angegeben wird, so wird der Wert der Variablen_A bei jedem Wiederholungschritt um 1, ansonsten um den Wert des Ausdrucks_3, erhöht. FOR/NEXT-Schleifen können auch ineinander verschachtelt werden.

Zur Programmverzweigung existiert eine Einfach- und eine Mehrfachverzweigungsanweisung. Sie haben folgendes Format:

```
GOTO Anweisungsnummer
ON Ausdruck GOTO Anw.nr.1, ..., Anw.nr.N
```

Die GOTO-Anweisung ist ein unbedingter Sprung zu der angegebenen Programmzeile. Falls unter der angegebenen Zeilennummer keine ausführbare Anweisung (z.B.: ein Kommentar) steht, so wird mit der nächstfolgenden ausführbaren Anweisung fortgesetzt. Mit der GOTO-Anweisung darf nicht in eine Funktionsdefinition gesprungen werden (auch nicht heraus).
Die Mehrfachverzweigung ON/GOTO verzweigt entsprechend des Wertes des Ausdrucks, der auf ON folgt. Falls der Wert 1 ist, so wird bei der ersten hinter GOTO angegebenen Anweisungsnummer fortgesetzt, bei 2 bei der zweiten usw. Wenn der Ausdruckswert kleiner 1 oder größer der Anzahl der angegebenen Anweisungsnummern ist, so wird die gesamte Anweisung ignoriert. Wenn der Wert des Ausdrucks nicht ganzzahlig ist, so wird er auf die nächste ganze Zahl gerundet.

Häufig sich wiederholende Programmteile können auch in BASIC in Form von Unterprogrammen realisiert werden. Dazu dienen die GOSUB- und die RETURN-Anweisung. Die Syntax der GOSUB-Anweisung ist analog zur GOTO-Anweisung:

```
GOSUB Anweisungsnummer
ON Ausdruck GOSUB Anw.nr.1, ..., Anw.nr.N
```

Die RETURN-Anweisung besteht nur aus dem Wort

```
RETURN
```

Sie bewirkt den Rücksprung zu der Anweisung, die auf die GOSUB-Anweisung folgt.

Eine Programmverzweigung wird mit Hilfe der IF/THEN-Anweisung realisiert. Sie hat folgende Syntax:

```
(a)  IF Ausdruck THEN Anweisungsnummer  [ELSE-Zweig]
(b)  IF Ausdruck THEN Anweisung  [ELSE-Zweig]
(c)  IF Ausdruck THEN DO
                      Anweisungen


     DOEND  [ELSE-Zweig]
```

Der optionale ELSE-Zweig kann wiederum eine der drei folgenden Formen haben:

```
(a)  ELSE Anweisungsnummer
(b)  ELSE Anweisung
(c)  ELSE DO
          Anweisungen
               .
               .
          DOEND
```

Falls der nach IF folgende Ausdruck wahr ist (d.h. der Wert ist ungleich Null), so werden die zum THEN-Zweig gehörenden Anweisungen ausgeführt. Falls der Wert des Ausdrucks gleich Null ist (d.h. falsch) und ein ELSE-Zweig vorhanden ist, so werden die dazu gehörenden Anweisungen ausgeführt. Falls der ELSE-Zweig fehlt, so wird mit der nach der IF-Anweisung folgenden Anweisung fortgesetzt. IF-Anweisungen können ebenfalls verschachtelt werden. Sie müssen dann innerhalb einer DO/DOEND-Gruppe erscheinen.

Da BASIC in seiner Grundform eine dialogorientierte Sprache ist, erfolgen Ein-/Ausgaben standardmäßig über Tastatur und Bildschirm. Zur Realisierung von Dialogeingaben dient die INPUT-Anweisung. Sie hat folgende Syntax:

```
INPUT Variablenliste                                    oder
INPUT Zeichenkettenkonstante, Variablenliste
```

Die Variablen innerhalb der Liste müssen durch Kommas getrennt werden. Bei der ersten Form zeigt ein Fragezeichen an, daß Eingabewerte erwartet werden.
Bei der zweiten Form wird das Fragezeichen durch die angegebene Zeichenkette ersetzt (z.B.: INPUT "Bitte Wert fuer S: ",S).

Falls mehrere Eingabewerte erwartet werden, so sind diese ebenfalls durch Kommas getrennt einzugeben. Während der Eingabe erfolgt eine Typprüfung.

Zur Ausgabe von Daten oder Zeichenketten dient die PRINT-Anweisung. Sie hat folgendes Format:

```
PRINT [Ausgabeliste]
```

Anstelle des Schlüsselwortes PRINT kann als Abkürzung auch der Doppelpunkt (z.B.: 100 A,B:) angegeben werden. Eine PRINT-Anweisung ohne Ausgabeliste bewirkt einen einfachen Zeilenvorschub. Die Ausgabeliste kann aus numerischen oder Zeichenkettenausdrücken bestehen. Die Elemente der Liste müssen durch Komma oder Semikolon getrennt werden. Ein Komma veranlaßt die Ausgabe zu Beginn der nächsten Tabulatorposition (implizit existieren 5 Tabulatorpositionen, alle 14 Zeichen), bei einem Semikolon erfolgt die Ausgabe dicht aufeinanderfolgend. Die Voreinstellung für die Länge einer Ausgabezeile (70) kann durch die SYSTEM-Anweisung "LINELEN" verändert werden.
Numerische Daten können auch formatiert ausgegeben werden. Dazu dient die PRINT/USING-Anweisung. Sie hat folgendes Format:

```
PRINT USING "Zeichenkettenkonstante", numerischer_Ausdruck
```

Die Formatbeschreibung erfolgt durch den nach USING folgenden Zeichenkettenausdruck (z.B.: PRINT "###.DD",A). Folgende Beispiele sollen die Kombinationsmöglichkeiten verdeutlichen:

Formatzeichenkette	Zahl	Ausgabe
###.##	1234.567	234.56
###.DD	1.2	1.20
+###	12.3	+ 12
##-	-78	78-
+##.DD	0	+ 0.00
###P##	123	123.
$##.DD	6.7	$ 6.70
***.DD	12.3	*12.30
##,###,###.DD	12345678	12,345,678.00
#.### ↑↑↑↑ (2stelliger Exponent)	234.5	2.345E+02
##.DD ↑↑↑↑↑ (3stelliger Exponent)	-34E+123	-34.00E+123

Daten können nicht nur durch Eingaben Variablen zugewiesen werden. Es besteht auch die Möglichkeit, innerhalb des Programms, Daten zu vereinbaren und diese dann Variablen zuzuweisen. Dazu dienen die READ/DATA/RESTORE-Anweisungen. Sie haben folgende Syntax:

```
READ Variablenliste
DATA Konstante_1, ..., Konstante_N
RESTORE [Zeilennummer]
ON Ausdruck RESTORE Zeilennr.1, ..., Zeilennr.N
```

Durch die DATA-Anweisung werden Konstanten innerhalb eines BASIC-Programms definiert. Über die READ-Anweisung können diese Konstanten Variablen zugewiesen werden. Dies erfolgt über einen internen Zeiger, der immer die nächste zuzuweisende Konstante adressiert (steht zu Beginn auf der ersten DATA-Anweisung, erste Konstante). Die Zuweisung erfolgt in der Reihenfolge der Konstantendefinitionen. Die RESTORE-Anweisung kann vor einer READ-Anweisung zur Veränderung dieses Zeigerwertes benutzt werden. Falls keine Zeilennummer angegeben ist, so wird der Zeiger wieder auf die erste DATA-Anweisung positioniert, ansonsten wird er auf die spezifizierte Zeilennummer gesetzt. Analog zur ON/GOTO-Anweisung kann dabei eine Mehrfachverzweigung für die RESTORE-Anweisung benutzt werden.

Gerade bei BASIC-Programmen ist eine Kommentierung der Aktionen innerhalb eines Programms besonders wichtig. Dazu dient die REM-Anweisung.

```
REM beliebiger_Text
```

Alle mit dem Schlüsselwort REM beginnenden Zeilen werden als Kommentarzeilen betrachtet und haben keinen Einfluß auf den Programmablauf. Kommentare können auch hinter jeder Anweisung stehen, sie müssen dann mit dem Zeichen "\" beginnen.
(z.B.: 110 PRINT A*3.4 \ Hier kann ein Kommentar stehen)

Im BASIC-Interpreter existiert die Funktion RND zur Erzeugung von Zufallszahlen. Die RANDOMIZE-Anweisung, mit dem Format:

```
RANDOMIZE
```

startet die Funktion mit einem neuen Anfangswert. Dadurch kann verhindert werden, daß in Programmabläufen stets die gleichen "Pseudozufallszahlen" generiert werden.

Zur Spezifizierung von bestimmten Systemfunktionen und Parametern dient die SYSTEM-Anweisung. Sie hat folgende Syntax:

```
SYSTEM Operation
```

Für Operation können dabei folgenden Zeichenketten mit möglichen Parametern stehen:

"WARNOFF"	Unterdrückung aller Warnmeldungen
"WARNON"	Ausgabe aller Warnmeldungen (ist impl. gesetzt)
"INDENTOFF"	stellt das automatische Einrücken beim Auslisten ab
"INDENTON"	automatisches Einrücken beim Auslisten (ist implizit gesetzt)

"ASINOFF" analog zu "INDENTOFF" für ASAVE-Dateien
(ist implizit gesetzt)
"ASINON" analog "INDENTON" für ASAVE-Dateien

"LINELEN" n
Setzen der Zeilenlänge für Ausgabezeile
(Wert zwischen 1 und 255; impl.: 70)

"AUTOCROFF" Ausschalten des automatischen Zeilenvorschubs
"AUTOCRON" Einschalten des automatischen Zeilenvorschubs

7.1.4. Zeichenketten und Felder

Eine Zeichenkette besteht aus einer in Anführungszeichen eingeschlossenen Folge von ASCII-Zeichen (maximal 255 Zeichen lang), außer den Zeichen " und < Diese Zeichen und die nicht druckbaren Zeichen können durch ihr numerisches Äquivalent eingeschlossen in spitze Klammern dargestellt werden. Die Notation <Dezimalzahl> steht dann innerhalb einer Zeichenkette für ein Zeichen (z.B.: <34> für "; <60> für < ; <9> für Tabulator und <13> für Zeilenvorschub; "NR<9>NAME<9>ABT<13>").
Zeichenkettenvariablen enthalten nach ihrer Vereinbarung durch eine DIM-Anweisung die leere Zeichenkette. Es besteht die Möglichkeit der Verarbeitung von Teilzeichenketten. Dazu wird folgende Notation verwendet:

Bei einfachen Zeichenkettenvariablen

Zeichenkettenname(erste_Zeichenposition)

entspricht der Teilzeichenkette ab dem spezifizierten Zeichen (Position)

Zeichenkettenname(erste_Zeichenpos.,letzte_Zeichenpos.)

entspricht der Teilzeichenkette, die aus den zwischen den angegebenen Positionen liegenden Zeichen gebildet wird.

(z.B.: wenn A$="VORNAME NAME ABT" ist, dann sind A$(9)="NAME ABT" und A$(9,12)="NAME")

Bei Zeichenkettenfeldern

Feldname(Elementnummer)

entspricht dem angegebenen Element des Zeichenkettenfeldes

Feldname(Elementnr.,erste_Zeichenposition)
Feldname(Elementnr.,erste_Zeichenpos.,letzte_Zeichenpos.)

entspricht der spezifizierten Teilzeichenkette des angegebenen Elements des Feldes

(z.B.: wenn B$ ein Feld von 10 Zeichenketten der Länge 25 ist, dann sind:
B$(6) die gesamte 6.Zeichenkette
B$(3,11) von der 3.Zeichenkette die letzten 15 Zeichen
B$(7,1,20) von der 7.Zeichenkette nur die ersten 20 Zeichen

Anstelle der runden Klammern können zur Kennzeichnung der Teilelemente auch eckige Klammern verwendet werden.

Felder von numerischen Größen werden über die DIM-Anweisung vereinbart. Dies ist notwendig, wenn sie mehr als 10 Elemente (oder 10 x 10 Elemente bei zweidimensionalen Feldern) enthalten. Die DIM-Anweisung hat folgende Syntax:

```
DIM Variable(ganze_Zahl), Variable(ganze_Zahl),
DIM Variable(ganze_Zahl,ganze_Zahl),
```

Die erste Form gilt für eindimensionale und die zweite Form für zweidimensionale Felder. Der Typ (ganzzahlig oder reell) wird durch den Variablennamen fixiert. Die Zählung der Indizes beginnt bei 1 (z.B.: besteht das Feld A(25) aus 25 reellen Zahlen und K%(5,10) ist ein Feld von ganzen Zahlen der Dimension 5 Spalten und 10 Zeilen). Nach der Feldvereinbarung besitzen alle Elemente den Wert Null.
Im Gegensatz zu numerischen Größen muß jede einfache Zeichenkettenvariable und jedes Zeichenkettenfeld (maximal eindimensional) über eine DIM-Anweisung vereinbart werden. Die erste Form der DIM-Anweisung vereinbart dann eine einfache Zeichenkettenvariable (der Parameter gibt die maximale Länge der Zeichenkette an) und die zweite Form ein eindimensionales Zeichenkettenfeld, wobei der erste Parameter die Anzahl der Zeichenketten und der zweite Parameter die maximale Länge der Zeichenketten definiert.
(z.B.: DIM A$(15) A$ ist eine Zeichenkette aus max. 15 Zeichen
DIM F$(5,10) F$ ist ein Feld aus 5 Zeichenketten der Länge 10)

Für Zeichenketten existiert ein Verkettungsoperator (+). Diese Operation kann in Verbindung mit der Teilzeichenkettenbildung und den zur Verfügung stehenden Zeichenkettenfunktionen zu vielfältigen Zeichenkettenmanipulationen angewendet werden. (z.B.: wenn N$ die Zeichenkette "PETER MEIER" darstellt, so druckt die Anweisung: PRINT "HALLO"+N$(1,5) die Zeichenkette HALLO PETER aus)

Daneben können Zeichenketten auch mit den Vergleichsoperatoren zur Bildung von logischen Ausdrücken verknüpft werden. Dabei sind zwei Zeichenketten nur dann gleich, wenn sie gleich lang sind und in jedem Zeichen übereinstimmen. Eine Zeichenkette ist kleiner als eine andere, wenn das erste verschiedene Zeichen einen kleineren numerischen Wert hat als das entsprechende Zeichen der anderen Zeichenkette.

Speziell zur Eingabe von Zeichenketten existiert noch die LINPUT-Anweisung. Sie hat folgende Form:

```
LINPUT Zeichenkettenvariable                                oder
LINPUT Zeichenkettenkonstante,Zeichenkettenvariable
```

Dabei werden alle eingegebenen Zeichen (einschließlich Anführungszeichen, Kommas und Leerzeichen) der Zeichenkettenvariable zugewiesen. Falls die Anzahl der eingegebenen Zeichen größer als die Dimension der Zeichenkettenvariable ist, so wird der Rest weggeschnitten.

7.1.5. Funktionen

Im BASIC-Interpreter sind eine Reihe von Funktionen fest integriert. Daneben kann der Anwender für seine Problemlösungen auch eigene Funktionen definieren.

Numerische Funktionen	
ABS(Ausdruck)	Liefert den Absolutbetrag des Ausdruckswertes
ATN(Ausdruck)	Liefert den Arcustangens des Ausdruckswertes (Ergebnis liegt im Bereich von -pi/2 bis +pi/2)
COS(Ausdruck)	Kosinusfunktion
EXP(Ausdruck)	Liefert den Wert e hoch Ausdruckswert (e=2,718281828)
INT(Ausdruck)	Liefert die größte ganze Zahl, die kleiner oder gleich dem Wert des Ausdrucks ist
LOG(Ausdruck)	Liefert den natürlichen Logarithmus des Ausdrucks (der Wert des Ausdrucks muß größer Null sein)
RND	Liefert eine Pseudozufallszahl zwischen 0 und 1
SGN(Ausdruck)	Liefert das Vorzeichen des Ausdruckswertes: 1, wenn Wert > 0 0, " " = 0 -1, " < 0
SIN(Ausdruck)	Sinusfunktion
SQR(Ausdruck)	Liefert die Quadratwurzel des Ausdruckswertes (Wert des Ausdrucks muß größer gleich Null sein)
TAN(Ausdruck)	Tangensfunktion

Neben den numerischen Funktionen existiert noch ein Satz von fest eingebauten Zeichenkettenfunktionen. Dabei kennzeichnet die Endung $ beim Funktionsnamen, daß ein Zeichenkettenwert geliefert wird.

Zeichenkettenfunktionen	
CHR$(ganzzahliger_Ausdruck)	Liefert das ASCII-Zeichen, das den Wert des Ausdrucks (0-255) besitzt (z.B.: ist CHR$(66) gleich dem Zeichen "B").
ASC(Zeichenkettenausdruck)	Liefert den numerischen Wert des ersten Zeichens des Ausdrucks (z.B.: ist ASC("A") gleich 65).
LEN(Zeichenkettenausdruck)	Liefert die Länge der als Parameter übergebenen Zeichenkette.
POS(Zeichenkette_A,Zeichenkette_B)	Liefert die Startposition der Teilzeichenkette_B in der Zeichenkette_A (z.B.: ist POS("ABCDEFG","EF") gleich 5). Falls die zweite Zeichenkette keine Teilkette der ersten ist, so wird der Wert Null geliefert.

VAL(Zeichenkettenausdruck)	Liefert den numerischen Wert, der durch die Zahlen in der Zeichenkette dargestellt wird (z.B.: ist VAL("123AB") gleich 123). Dabei kann auch die E-Notation benutzt werden (z.B.: wenn A$="ABC3EXY" ist, so liefert VAL(A$(4,5)+"2") den Wert 300).
STR$(numerischer_Ausdruck)	Bildet das Zeichenkettenäquivalent zum numerischen Wert, (z.B.: ist STR$(123) gleich der Zeichenkette "123").
LEFT$(Zeichenkettenausdruck,ganzzahliger_Ausdruck)	Liefert die linke Teilzeichenkette des ersten Parameters. Die Länge wird durch den Wert des zweiten Parameters spezifiziert. (z.B.: ist LEFT$("PETER MEIER",5) gleich der Zeichenkette "PETER")
RIGHT$(Zeichenkettenausdruck,ganzzahliger_Ausdruck)	Liefert die rechte Teilzeichenkette analog zur LEFT$-Funktion.
SEG$(Zeichenkettenausdr.,ganz._Ausdruck,ganz._Ausdruck)	Liefert eine Teilzeichenkette des ersten Parameters. Die Position des ersten Zeichens wird durch den Wert des zweiten Parameters und die des letzten Zeichens durch den Wert des dritten Parameters spezifiziert. (z.B.: ist SEG$("1020 BERLIN STR xyz",6,11) gleich der Zeichenkette "BERLIN")

Anwenderfunktionen werden innerhalb eines Programms durch eine DEF-Anweisung definiert und können wie eingebaute Funktionen verwendet werden. Die Funktionsnamen bestehen aus den Buchstaben "FN" gefolgt von einem normalen Variablennamen. Dabei kennzeichnet der Variablenname den Funktionstyp, d.h. die Funktion liefert einen ganzzahligen Wert, wenn der Name mit "%" endet, eine Zeichenkette, wenn er mit "$" endet, und sonst einen reellen Wert. Bei der Definition von Anwenderfunktionen wird zwischen ein- und mehrzeiligen Funktionen unterschieden.

- einzeilige Funktionsdefinitionen:

```
DEF Fkt.name (formale_Parameterliste) = Ausdruck          oder
DEF Fkt.name = Ausdruck
```

- mehrzeilige Funktionsdefinitionen:

```
DEF Fkt.name (formale_Parameterliste)                     oder
DEF Fkt.name
        Anweisungen
            .
            .
RETURN Ausdruck
FNEND
```

Der Typ des Ausdrucks muß mit dem Funktionstyp (durch Namen festgelegt) vereinbar sein. Die formalen Parameter sind lokale Größen innerhalb der Funktionsdefinition, d.h. sie stehen in keiner Beziehung zu Programm-

variablen gleichen Namens.
Bei einer mehrzeiligen Funktionsdefinition dürfen im Funktionskörper keine weiteren Funktionsdefinitionen auftreten. Aufrufe anderer Funktionen (auch rekursiv) sind dagegen erlaubt. Der Funktionskörper muß eine abgeschlossene Einheit bilden, d.h. Schleifen müssen vollständig im Körper liegen und es dürfen keine Sprünge aus dem Funktionskörper heraus auftreten.

7.1.6. Dateiarbeit

Neben der standardmäßigen Zuordnung der Ein-/Ausgabe zu Tastatur und Bildschirm können innerhalb von BASIC-Programmen die Ein-/Ausgabeströme auch anderen peripheren Geräten zugeordnet werden. Entsprechende Treiberprogramme müssen dazu im Betriebssystem vorhanden sein (z.B.: für die Druckerausgabe). Sie müssen vor Start des BASIC-Interpreters aktiviert werden, wenn sie in BASIC-Programmen verwendet werden sollen.

```
%ACTIVATE $PRINTER
%BASIC
```

Der entsprechende Ein-/Ausgabekanal wird im Programm durch eine FILE-Anweisung eröffnet und einer logischen Gerätenummer (zwischen 1 und 15) zugeordnet. Bei Ausgaben zum Beispiel wird einfach vor der Ausgabeliste die Gerätenummer gefolgt von einem Semikolon angegeben.

```
10  FILE #1;"$PRINTER"
      .
      .
100 PRINT #1;"DIESE AUSGABE GEHT UEBER DEN DRUCKER"
```

Ebenso können externe Daten als Inhalt von Dateien, die auf Disketten abgelegt sind, verarbeitet werden. Dazu werden die Möglichkeiten des Betriebssystems ausgenutzt, indem Standard-Ein-/Ausgabeanforderungen des DOS-Teils durch BASIC-Anweisungen ausführbar sind. Es können sowohl Dateien vom Typ ASCII (für Zeichenketten) als auch binäre Dateien (für Daten) manipuliert werden.
Die Fehlerbehandlung während der Dateiarbeit kann über zwei Wege erfolgen. Erstens: Es werden auftretende Fehler normal über den Bildschirm angezeigt. Sie können aber auch über eine TRAP-Anweisung (siehe folgenden Abschn. 7.1.7.) abgefangen werden.
Zweitens: Es werden keine Fehlermeldungen generiert, wenn innerhalb der Anweisungen die Fehlerkodes einer Variablen ("Returnkode") zugeordnet werden. Die Auswertung muß dann selbstständig durchgeführt werden.

Ein-/Ausgabeanforderungen zur Dateiarbeit:

Mit der FILE-Anweisung werden Dateien eröffnet und einer logischen Nummer zwischen 1 und 15 zugeordnet.

```
FILE #n;Namen_Optionenkette[,Returnvariable]
```

Die Namen_Optionenkette ist eine Zeichenkette, die den Dateinamen und mögliche durch Semikolon getrennte Optionen enthält.

Mögliche Optionen	
REC=Konstante	Legt Satzlänge der Datei fest (impl.: 128). Falls die Datei bereits mit einer anderen Satzlänge existiert, so wird diese auf den angegebenen Wert verändert.
RL=Konstante	Legt bei einer neu anzulegenden Datei die Satzlänge fest (impl.: 128).
ACC=Option	Legt den Typ der Eröffnungsanforderung fest. Die Option kann sein:
IN	zur Eingabe, d.h. Datei muß bereits existieren
OUT	zur Ausgabe, falls Datei bereits existiert, so wird der Inhalt gelöscht
NEW	zur Ausgabe, falls Datei bereits existiert, so wird Fehler 1 generiert
UPD	zur Aktualisierung einer bereits existierenden Datei (Dateizeiger wird auf den Anfang gesetzt)
Die Returnvariable kann folgende Werte annehmen:	
0	Operation fehlerfrei ausgeführt
1	Datei existiert bereits (bei ACC=NEW)
2	Datei existiert nicht (bei ACC=IN)
3	unzulässiger Dateiname
4	unerlaubte Operation
5	unzulässige Satzlänge
6	nicht genügend Pufferspeicher vorhanden

Beispiel zur Eröffnung einer neuen Datei mit einer Satzlänge von 512 Byte und dem Namen VERTRAEGE:

```
FILE #2;"VERTRAEGE;ACC=NEW;RL=512",R%
```

Zum Ende eines Programms werden alle Dateien automatisch abgeschlossen. Während der Programmausführung kann dies durch eine CLOSE-Anweisung auch direkt für die spezifizierten Dateien (über ihre logischen Nummern) erfolgen.

```
CLOSE #n,#m,...
```

Falls kein Parameter angegeben wird, so werden alle eröffneten Dateien abgeschlossen.

Für den Dateizugriff gibt es zwei Möglichkeiten. Einmal den sequentiellen Zugriff, dabei erfolgt die Adressierung der zu lesenden oder zu schreibenden Elemente über einen internen Dateizeiger, der nach jedem Zugriff automatisch auf das nächste Element zeigt. Dieser Zeiger kann über die SPACE-Anweisung vor und zurück positioniert werden.
Bei der zweiten Zugriffsart kann zusätzlich die Satznummer spezifiziert werden, ab der der Zugriff beginnt. Beide Zugriffsarten können auch gemischt auftreten.

Für die Dateieingabe werden die normalen Anweisungen verwendet, nur daß nach dem Schlüsselwort noch zusätzlich die logische Dateinummer angegeben

wird.

```
INPUT #n;Variablenliste
LINPUT #n;Zeichenkettenvariable
READ #n;Variablenliste
```

Für die Datenausgabe kommt noch die WRITE-Anweisung für die Ausgabe von numerischen Daten hinzu (eine reelle Zahl belegt dabei 8 Byte Speicherplatz und eine ganze Zahl 2 Byte).

```
PRINT #n [;Ausgabeliste]
WRITE #n;Ausgabeliste
```

Mit der SPACE-Anweisung kann der Dateizeiger auf die gewünschte Position innerhalb der Datei gesetzt werden. Dies kann durch Angabe einer relativen Entfernung (vor- und rückwärts) oder durch Angabe eines Suchzeichens erfolgen.

```
SPACE #n;Bewegungszähler
SPACE #n;Bewegungszähler,Suchzeichen ,Returnvariable
```

Die Returnvariable enthält als Wert die Anzahl der Bytes, um die der Dateizeiger tatsächlich bewegt wurde.

Ebenfalls zur Dateipositionierung kann die RESTORE-Anweisung verwendet werden.

```
RESTORE #n
RESTORE #n;Satznummer
```

In der ersten Form wird der Dateizeiger auf den Anfang der Datei gesetzt, in der zweiten auf den angegebenen Satz.

Im Zusammenhang mit der Positionierung des Dateizeigers wird die EOF-Funktion (end of file) verwendet.

```
EOF(n)
```

Sie zeigt mit dem Wert 1 an, daß das Ende der Datei mit der logischen Nummer n erreicht ist. Ansonsten liefert sie den Wert Null.

Für die zweite Dateizugriffsart wird bei den Ein-/Ausgabeanweisungen noch zusätzlich die Nummer des Satzes angegeben, ab dem der Zugriff beginnen soll. Bei der Eingabe in folgender Form:

```
INPUT #n;Satznummer;Variablenliste
LINPUT #n;Satznummer;Zeichenkettenvariable
READ #n;Satznummer;Variablenliste
```

und für die Ausgabe:

```
PRINT #n;Satznummer;Ausgabeliste
WRITE #n;Satznummer;Ausgabeliste
```

Zusätzlich können noch existierende Dateien verkürzt oder ganz gestrichen werden. Mit der TRUNCATE-Anweisung:

```
TRUNCATE #n
```

wird der hinter der augenblicklichen Dateizeigerposition befindliche Teil der Datei mit der logischen Nummer n gestrichen. Mit der ERASE-Anweisung:

```
ERASE "Dateiname" ,Returnvariable
```

wird die Datei mit dem angegebenen Namen gestrichen. Die Returnvariable kann dabei folgende Werte annehmen:

0	Operation fehlerfrei ausgeführt
1	Datei konnte nicht gestrichen werden
2	Benutzer darf diese Datei nicht streichen
3	angegebene Datei existiert nicht

7.1.7. Traps und Segmentierung

Die TRAP-Anweisung ist ein Mittel, um während der Abarbeitung eines BASIC-Programms auftretende Fehlerbedingungen abzufangen und um auf externe Interrupts reagieren zu können. Sie kann folgendes Format haben:

```
TRAP Bedingung TO Anweisungsnummer
TRAP Bedingung OFF
TRAP ESC
```

Die erste Form bewirkt beim Auftreten der angegebenen Bedingung eine Verzweigung zu der spezifizierten Zeilennummer. Mit der zweiten Form kann eine spezielle TRAP-Bedingung ausgeschaltet werden. Die dritte Form schaltet alle TRAP-Bedingungen aus. Für Bedingung können folgende Abkürzungen stehen:

ESC	Auslösung eines Traps bei Betätigung der ESC-Taste.
ERR	Auslösung eines Traps bei einem Laufzeitfehler.
EOF	Auslösung eines Traps bei Erreichen eines Dateiendes.
KEYS	Wenn während des normalen Programmablaufs Tasten betätigt werden (außer ESC), so werden die Werte in einem Puffer abgelegt. Auf diese Zeichen ist über die Funktion KEYS$ zugreifbar. Es wird ein Trap ausgelöst, wenn der Puffer voll ist.
EXT	Ein Trap mit dieser Bedingung wird generiert, wenn ein anderes Programm das externe Interruptflag von BASIC (Zeiger darauf steht auf Adresse 4008H) setzt. Danach wird dieses Flag zurückgesetzt und das BASIC-Programm kann eine entsprechende Aktion einleiten.

Neben der Funktion KEYS$ existieren noch folgende Funktionen zur genaueren Spezifizierung der eingetretenen Bedingung:

TRP — Liefert als Wert die Zeilennummer bei deren Abarbeitung die TRAP-Bedingung eingetreten ist.

ESC — Liefert den Wert 1, falls die ESC-Taste betätigt wurde, den Wert Null sonst.

ERR — Liefert die Fehlernummer des aufgetretenen Laufzeitfehlers.

Beispiel zur Anwendung der TRAP-Funktion:

```
10   TRAP ERR TO 1000
          .
          .
1000 PRINT "*** FEHLER IN ZEILE: ";TRP;" ***"
1010 PRINT "*** FEHLERNUMMER: ";ERR;" ***"
1020 STOP
```

Die Größe eines Anwenderprogramms ist durch den im BASIC verfügbaren Arbeitsspeicher (siehe SIZE-Kommando) begrenzt. Um auch größere Programme abzuarbeiten, wurde die Möglichkeit der Teilung des Programms in mehrere Segmente geschaffen. Alle Programmsegmente müssen auf der Diskette abgelegt sein und werden von dort in den Arbeitsspeicher geladen. Für den Programmweiterlauf zwischen den Segmenten wird die CHAIN-Anweisung verwendet. Sie hat folgendes Format:

```
CHAIN "Programmname"
```

Das aktuelle Programm wird beendet, und das angegebene BASIC-Programm wird geladen und gestartet. Es erfolgt keine automatische Rückkehr zu dem aufrufenden Programmsegment. Dies kann nur über eine weitere CHAIN-Anweisung erfolgen. Alle im aktuellen Programm eröffneten Dateien bleiben offen.
Während der CHAIN-Anweisung werden nur die Variablen gerettet, die in einer COM-Anweisung (common) vereinbart wurden. Alle restlichen Variablen verlieren ihren Wert. Durch die COM-Anweisung können also Variablen definiert werden, auf die von verschiedenen Programmsegmenten aus zugegriffen werden kann. Sie hat folgende Syntax:

```
COM Variablenliste
```

Die COM-Anweisung muß vor allen Anweisungen (Ausnahme: REM-Anweisungen) im Programm stehen. Wenn über die COM-Anweisung Felder vereinbart werden, so ist keine DIM-Anweisung mehr nötig. Die Variablenlisten der COM-Anweisungen in den verschiedenen Programmsegmenten brauchen nicht die gleiche Reihenfolge der Variablennamen aufzuweisen, sie müssen aber in den Namen und der Dimensionierung übereinstimmen.

7.2. Kopplung mit Assemblerprogrammen

Innerhalb eines BASIC-Programms können in Assemblersprache geschriebene Prozeduren über die CALL-Anweisung aufgerufen werden. Sie hat folgendes Format:

CALL "Prozedurname" [,Eingabeparameterliste] [;Ausgabeparameterliste]

Nach dem Prozedurnamen kann eine Liste von an die Prozedur zu übergebenden Parametern erscheinen. Eine mögliche Liste von Parametern, die von der Prozedur an BASIC zurückgegeben werden, wird durch ein Semikolon eingeleitet. Die Elemente der Parameterlisten können sowohl numerische Größen (reell oder ganzzahlig) als auch Zeichenketten sein. Sie werden durch Kommas voneinander getrennt.
Bei der Kopplung einer Assemblerprozedur mit einem BASIC-Programm sind zwei Aspekte wichtig, einmal die Bekanntmachung der externen Prozedurnamen an BASIC (dies erfolgt vor dem Start von BASIC) und das Parameterübergabeschema beim Aufruf der Prozedur. Zum ersten Punkt müssen innerhalb der Assemblerprozedur zwei Tabellen aufgebaut werden.

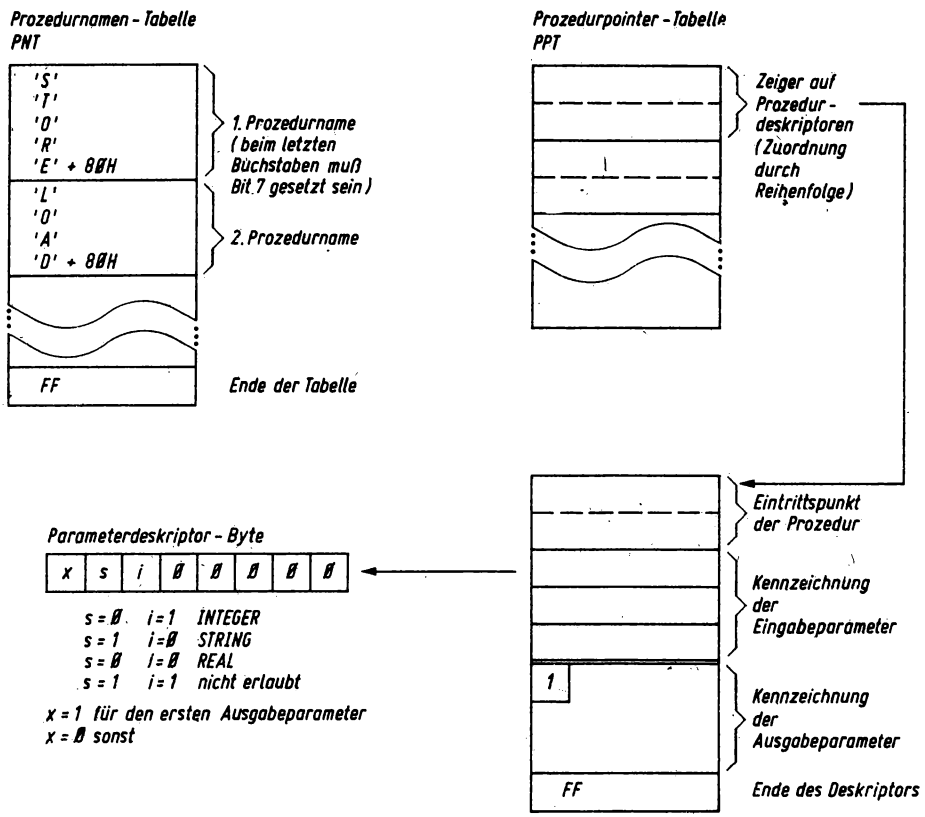


Bild 7.1. Aufbau der PNT- und PPT-Tabellen

Diese enthalten einmal alle Namen der von BASIC aus aufrufbaren Prozeduren (Prozedurnamen-Tabelle: PNT) und zum anderen die Adressen von Prozedurbeschreibungsblöcken (Prozedurpointer-Tabelle: PPT), in denen der Eintrittspunkt der Prozedur und die Anzahl und der Typ der Parameter fixiert sind.
Ganze Zahlen werden intern durch zwei Byte dargestellt. Reelle Zahlen werden intern durch eine 8 Byte große BCD-Gleitkommazahl repräsentiert. Die detaillierte Beschreibung der Zahlendarstellungen wird im Abschn. 9.2. (PASCAL-Datentypen) gegeben. Zeichenketten werden durch ihre ASCII-Werte (ein Zeichen ein Byte) dargestellt. Vor dem ersten Zeichen wird in 2 mal 2 Byte zweimal (!) die Länge der Zeichenkette angegeben.

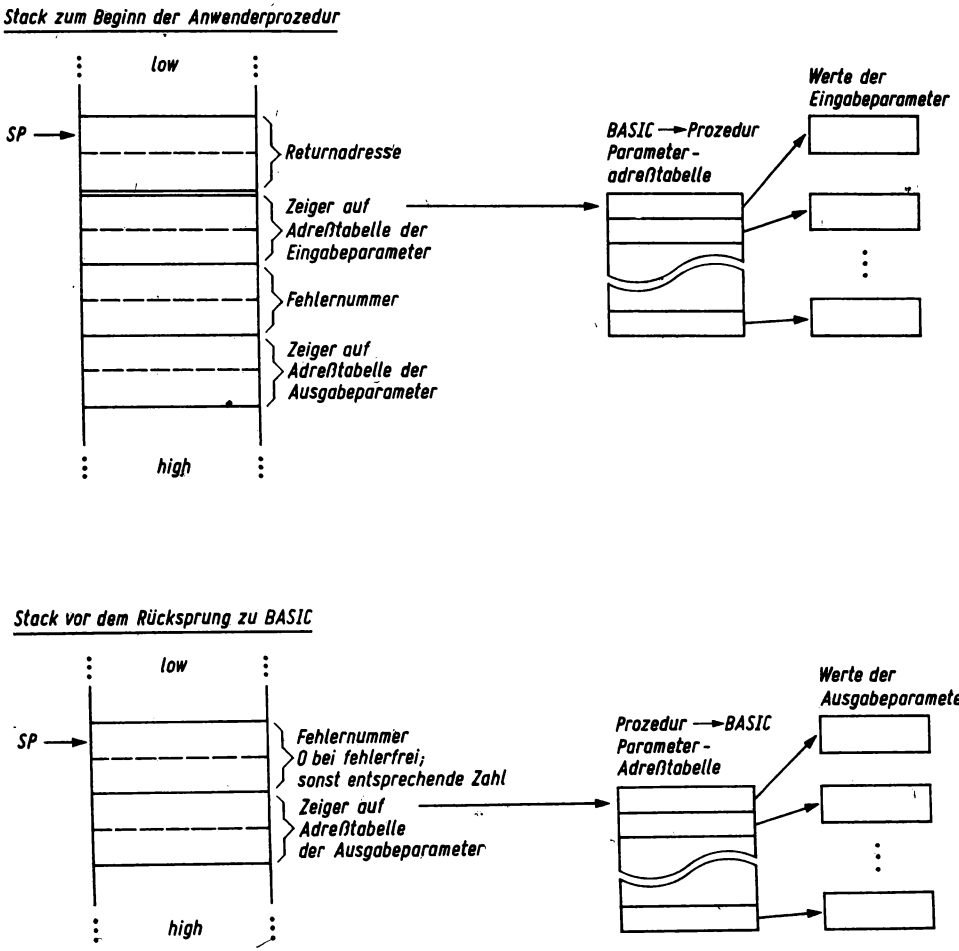


Bild 7.2. Parameterübergabeschema bei der Kopplung von BASIC mit externen Prozeduren

Die Parameterübergabe zwischen dem BASIC-Programm und der externen Prozedur erfolgt über den Stack. Bevor die Prozedur über CALL aufgerufen wird, werden drei 16-Bit-Werte in den Stack geschrieben (über Push-Befehle). Die ersten beiden Werte müssen von der Anwenderprozedur vor Rückkehr zu BASIC gesetzt werden. Der dritte Wert ist ein Zeiger auf eine Tabelle der Adressen der Eingabeparameter.

Der BASIC-Interpreter muß vor dem Start über die Existenz der beiden Tabellen (PNT und PPT) informiert werden. Dies geschieht über folgende Befehlsfolge zum Beginn der externen Prozedur:

```
LD HL,NAMETAB       ;Adresse von PNT
LD DE,POINTAB       ;Adresse von PPT
JP 4006H            ;BASIC-Startadresse
```

Die externe Prozedur sollte auf die höchstmögliche Adresse verbunden werden, um für den BASIC-Interpreter noch einen möglichst großen Arbeitsbereich zu erhalten.
Der Start von BASIC erfolgt dann über die externe Prozedur:

```
%USERPROZEDUR,      Laden der externen Prozedur, aber nicht starten
%BASIC,X C000       (Startadresse der Prozedur: C000)
                    Laden von BASIC und starten der Anwenderprozedur
```

Zu beachten ist noch, daß bei Aufruf der Prozedur der Stackpointer (SP) auf den internen BASIC-Stackbereich gesetzt ist. Die dort maximal garantierte Größe beträgt 30 Byte. Bei größerem Bedarf muß deshalb mit einem eigenen Stackbereich gearbeitet werden. Vor der Rückkehr zu BASIC ist der Stackpointer wieder auf die Adresse des BASIC-Stackbereichs zu setzen.

7.3. Programmbeispiele

Zur Demonstration werden die Programmlistings der im Abschn. 7.1.1. erwähnten Programme FLIST.BP und CAT.BP angegeben.

Das Programm FLIST erwartet die Eingabe eines Dateinamens, eröffnet diese Datei und gibt den Inhalt der Datei über den Bildschirm aus. Falls als erstes Zeichen des Dateinamens ein Doppelkreuz angegeben wird, so erfolgt zusätzlich noch eine Ausgabe der Zeilennummern. Das Einlesen des Inhalts der Datei (sie muß vom Typ ASCII sein) erfolgt zeilenweise.

```
 10 DIM A¤(200)
 20 INPUT "FILE: ",A¤
 30 IF A¤(1,1)="#" THEN  DO
 40 LET A¤=A¤(2)
 50 LET L%=1
 60 DOEND
 70 ELSE LET L%=0
 80 FILE #1;A¤+";ACC=IN",R%
 90 IF R%<>0 THEN  180
100 PRINT
110 LET I%=0
120 LET I%=I%+1
130 LINPUT #1;A¤
140 IF EOF(1) THEN  200
150 IF L% THEN PRINT I%;"<9>";A¤
160 ELSE PRINT A¤
170 GOTO  120
180 PRINT
190 PRINT "*** DATEIFEHLER: ";R%;" ***"
200 END
```

Bild 7.3. Programm FLIST.BP

Das Programm CAT ermittelt die Namen aller auf einer Diskette befindlichen Dateien. Es erwartet die Eingabe einer Laufwerksnummer. Danach kann noch eine Dateinamensendung (z.B.: .BP) spezifiziert werden. Es werden dann nur die Dateien, deren Namen mit der eingegebenen Zeichenkette endet aufgelistet.
Das Programm eröffnet das Inhaltsverzeichnis der angegebenen Diskette und liest den Inhalt satzweise (jeweils 128 Byte gleich ein Satz der Datei DIRECTORY) ein. Der Aufbau eines Satzes des Inhaltsverzeichnisses einer Diskette kann mit Hilfe des UDOS-Dienstprogramms FILE.DEBUG ermittelt werden. Dort sind die Dateinamen und der Verweis (Diskadresse) auf den Beschreibungssatz der Datei abgelegt (1 Byte Dateinamenslänge, Dateiname maximal 32 Byte, 2 Byte Zeiger).
Die Variable N erhält jeweils die Dateinamenslänge. In der Zeile 270 wird der Test auf Übereinstimmung mit der eingegebenen Endung durchgeführt, und in der Zeile 280 erfolgt die Ausgabe der Dateinamen.

```
 10 REM BASIC CAT-PROGRAMM
 20 REM 10/84
 30 TRAP ERR TO  330
 40 DIM A¤(128)
 50 DIM D¤(10)
 60 DIM E¤(20)
 70 LINPUT "LAUFWERK: ",D¤
 80 TRAP ERR TO  110
 90 LET D¤=":"+STR¤(VAL(D¤))
100 GOTO  130
110 LET D¤=""
120 TRAP ERR TO  330
130 PRINT
140 LINPUT "ENDUNG: ",E¤
150 PRINT
160 FILE #1;"¤NDOS"+D¤+"/DIRECTORY;ACC=IN"
170 READ #1;A¤
180 IF EOF(1) THEN  320
190 LET L=1 \ STARTPUNKT IN DER ZEICHENKETTE
200 LET N=ASC(A¤(L,L))
210 IF N=255 THEN  170
220 IF N>128 THEN  DO
230 LET N=N-128
240 GOTO  300
250 DOEND
260 IF N=0!N=127 THEN  170
270 IF A¤(L+N+1-LEN(E¤),L+N)<>E¤ THEN  300
280 PRINT A¤(L+1,L+N);
290 PRINT ,
300 LET L=L+1+N+2
310 GOTO  200
320 END
330 PRINT "CAT KANN NICHT AUSGEFUEHRT WERDEN, FEHLER: ";ERR
340 END
```

Bild 7.4. Programm CAT.BP

8. U 883 TINY-MPBASIC

Der Einchipmikrorechner U883 enthält in seinem 2 KByte großen internen ROM einen einfachen BASIC-Interpreter. Daneben existiert ein dazugehöriger Editor-/Debuggerteil, der alle Funktionen, die für die Programmentwicklung notwendig sind, enthält. Diese Komponenten liegen entweder im externen ROM, sie können dann nach vollendeter Entwicklungsarbeit entfallen, oder sie existieren auf einem Wirtsrechner.
Damit wird vielen potentiellen Anwendern des Einchipmikrorechners, die über keine Entwicklungstechnik verfügen, eine Möglichkeit gegeben Programme zu entwickeln (z.B.: für Steuerungs- oder Regelungsaufgaben im Rationalisierungsmittelbau). Der Anwender kann somit seine Programme im Zusammenspiel mit der von ihm erstellten Hardware austesten. Nach der Erprobung ist das fertige Programm in einen EPROM zu laden, und das Gerät kann eingesetzt werden.
Da die Problemlösung in Form von BASIC-Programmen erarbeitet wird, ist ein schnelles Erstellen und Modifizieren der Anwendersoftware möglich, ohne daß ein großer Aufwand für die sonst notwendige Entwicklungstechnik auftritt.

8.1. Sprachkonzept und Anwendung

Das in TINY-MPBASIC geschriebene Anwenderprogramm wird vom im internen ROM-Bereich des U883 befindlichen BASIC-Interpreters abgearbeitet.
Neben dem BASIC-Programm sind Initialisierungsteile und, falls notwendig, die Prozeduren GET_CHAR (Einzelzeicheneingabe) und PUT_CHAR (Einzelzeichenausgabe) zu erstellen. Diese sehr stark vom Einsatzfall abhängigen Teile sind in Assemblersprache zu realisieren.
Nach durchgeführter Initialisierung kann das BASIC-Anwenderprogramm aufgerufen werden. Dies erfolgt durch einen CALL-Befehl zur Adresse %7FD (Registerpointer auf %10: SRP #%10). Vorher sind in Register 6 der höherwertige Teil und in Register 7 der niederwertige Teil der Startadresse zu laden. Der Anwender kann für seine Problemlösung zusätzlich noch externe Prozeduren und Funktionen in Assemblersprache realisieren, die vom BASIC aus aufrufbar sind. Das Vorhandensein externer Prozeduren und Funktionen wird dem Interpreter dadurch bekannt gemacht, daß in den Registern 8 und 9 die Adresse einer Prozedurnamentabelle übergeben wird. Falls keine externen Prozeduren verwendet werden, so sind die Register 8 und 9 vor Aufruf des BASIC-Interpreters Null zu setzen. Das Setzen der Register 6 bis 9 erfolgt mit dem Registerpointerwert %00.
Das BASIC-Anwenderprogramm muß syntaktisch fehlerfrei sein. Es wird in verdichteter Form abgespeichert. Die Erstellung eines syntaktisch fehlerfreien Programms und der abarbeitbaren verdichteten Form erfolgt mit Hilfe des Editor/Debugger-Programmpakets.

Der TINY-MPBASIC-Interpreter verarbeitet intern 16 Bit breite Daten, die als Integergrößen (Zweierkomplementdarstellung: -32768 ... +32768), Wortgrößen oder Bytewerte (niederwertige 8 Bit) interpretiert werden können. Konstanten können in Dezimal- oder Hexadezimalschreibweise (durch das Prozentzeichen gekennzeichnet: %0 ... %FFFF) angegeben werden. Negative Dezimalzahlen müssen in Klammern gesetzt werden, damit das Vorzeichen nicht als Operator wirkt. Für Variablenbezeichnungen sind die Buchstaben A bis Z verwendbar. (Sie belegen im Registersatz die Adressen ab %20.)

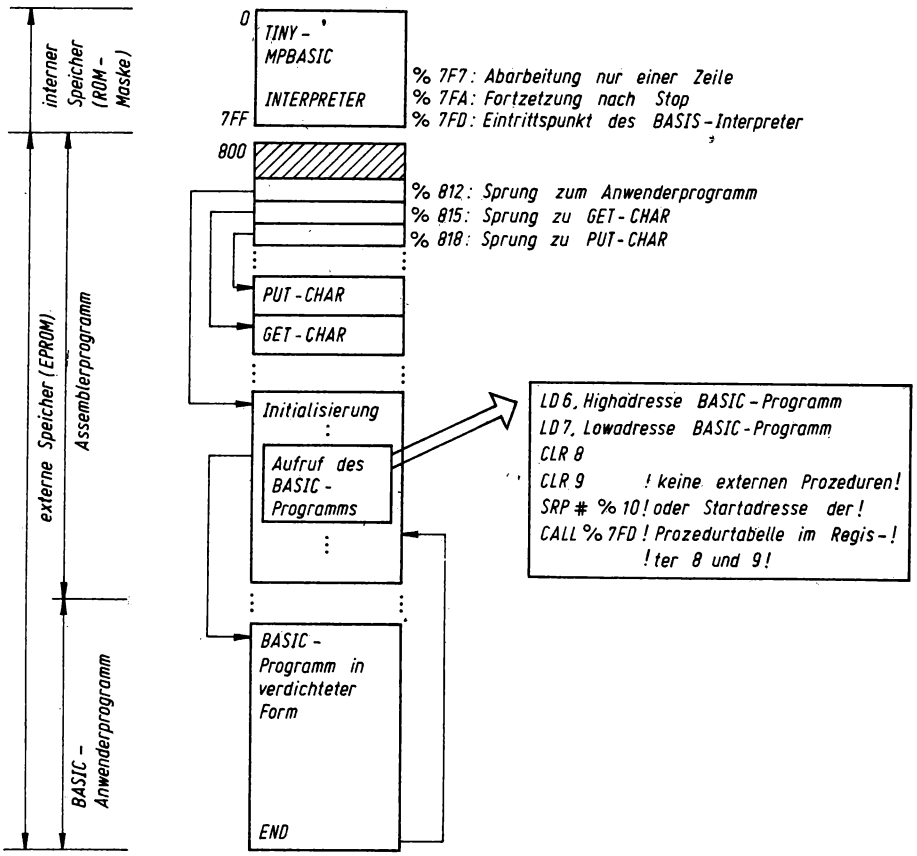


Bild 8.1. Zusammenspiel des U883-BASIC-Interpreters mit Anwenderteilen

Ausdrücke werden durch Verknüpfung von Konstanten, Variablen- oder Funktionswerten durch logische und arithmetische Operatoren gebildet.

Operatoren		Abkürzung	ASCII-Kode
+	Addition	+	2B
-	Subtraktion	-	2D
*	Multiplikation	*	2A
/	Division	/	2F
$MOD	Modulo	$M	24 4D
$AND	logisches UND (bitweise)	$A	24 41
$OR	logisches ODER "	$O	24 4F
$XOR	exklusives ODER "	$X	24 58

Ausdrücke werden von links nach rechts ausgewertet. Es besteht die Möglichkeit der Klammerung. Die Verschachtelungstiefe hängt dabei von der verfügbaren Stackgröße ab.

Eine Prozedur ist ein in Assemblersprache geschriebenes Programm, das einen Satz von BASIC übergebenen Eingabeparametern verarbeitet und Ausgabeparameter an den BASIC-Interpreter zurückgibt. Die Parameterübergabe erfolgt im Stackbereich. Der Aufruf erfolgt über den Prozedurnamen.

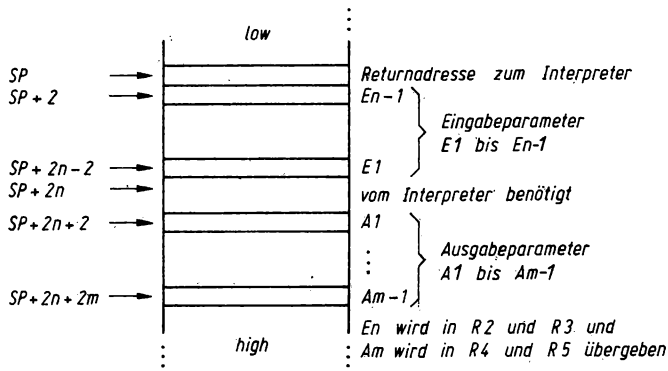


Bild 8.2. Parameterübergabeschema für externe Prozeduren

Funktionen sind Prozeduren, die genau einen Wert an den Interpreter übergeben. Sie können deshalb in Ausdrücken verwendet werden.
Neben einer Reihe von fest integrierten Standardprozeduren und -funktionen hat der Anwender die Möglichkeit eigene, seiner Hardwarekonfiguration angepaßte Prozeduren oder Funktionen, hinzuzufügen. Die Verbindung zu TINY-MPBASIC erfolgt über die schon erwähnte Prozedurnamentabelle.

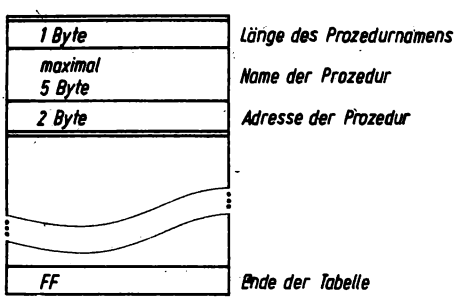


Bild 8.3. Aufbau der externen Prozedurnamentabelle

Das BASIC-Anwenderprogramm ist zeilenorientiert. Jede Zeile beginnt mit einer Nummer. Pro Zeile muß wenigstens eine Anweisung vorhanden sein. Mehrere Anweisungen auf einer Zeile sind möglich, wenn sie durch Semikolons getrennt sind. 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 Programmzeilen müssen in aufsteigender Folge markiert sein. Die Anweisungsnamen werden in abgekürzter Form abgelegt, um möglicht wenig Speicherplatz zu belegen, Leerzeichen werden außer in Kommentaren und

Texten entfernt. Die Sortierung und das Herstellen der verdichteten Form übernimmt normalerweise der Editorteil.

8.2. Anweisungen

Die Wertzuweisung für eine Variable ist die LET-Anweisung.

```
LET Variablenname = Ausdruck
```

Zur Programmverzweigung dient die GOTO-Anweisung.

```
GOTO Ausdruck
```

Unterprogramme können mit Hilfe der GOSUB-Anweisung aufgerufen werden. Das Ende eines Unterprogramms wird durch die RETURN-Anweisung markiert. Sie bewirkt die Programmfortsetzung bei der nach GOSUB folgenden Anweisung.

```
GOSUB Ausdruck
RETURN
```

Unterprogramme können weitere Unterprogramme aufrufen. Die Verschachtelungstiefe wurde zur Verhinderung eines Stacküberlaufs auf 15 beschränkt.

Programmverzweigungen werden mit der IF/THEN/ELSE-Anweisung realisiert. Sie hat folgende Form:

```
IF Ausdruck Vergleichsoperator Ausdruck THEN Anweisung
ELSE Anweisung
```

Vergleichsoperatoren	Abkürzung	ASCII-Kode
größer	>	3E
kleiner	<	3C
gleich	=	3D
größer/gleich	>=	3E 3D
kleiner/gleich	<=	3C 3D
ungleich	<>	3C 3E

Falls die nach IF folgende Bedingung wahr ist, so wird die nach THEN folgende Anweisung ausgeführt und der Rest übersprungen. Andernfalls wird die mit ELSE beginnende Zeile abgearbeitet.

Mit der PROC-Anweisung kann innerhalb des BASIC-Programms eine externe Prozedur aufgerufen werden.

```
PROC [Variablenliste] = Prozedurname [Parameterliste]
```

Die Variablen innerhalb der optionalen Variablenliste (Rückgabeparameter) und die Parameter innerhalb der ebenfalls optionalen Parameterliste (Eingabeparameter) sind durch Kommas zu trennen. Als Parameter sind Variablen und Konstanten zulässig.

Zur Ein-/Ausgabe dienen die Anweisungen PRINT, PRINTHEX und INPUT. Die Festlegung, über welche Geräte die Ein-/Ausgabeströme laufen, erfolgt durch die Programme GET_CHAR und PUT_CHAR. Dies kann je nach Anwendungsfall sehr unterschiedliche Hardware sein (z.B.: Terminal, Fernschreiber, LED- oder LCD-Anzeigen und verschiedene Tastaturen o.ä.).

```
PRINT ["beliebiger_Text"] [Ausdruck]
PRINTHEX ["beliebiger_Text"] [Ausdruck]
```

Die PRINT-Anweisungen geben einen Zahlenwert (bei PRINT: dezimal und bei PRINTHEX: hexadezimal) aus. Diesem Wert kann eine beliebige Textkette vorangestellt sein. Falls sowohl der Text als auch der Ausdruck fehlen, so wird lediglich ein Zeilenvorschub (%0D) ausgegeben. Wird die PRINT-Anweisung (bzw. PRINTHEX) mit einem Komma abgeschlossen, so unterbleibt die Ausgabe des Zeilenvorschubs (%0D).

```
INPUT ["beliebiger_Text"] Variablenname
```

Die INPUT-Anweisung gibt, falls vorhanden, erst den angegebenen Text aus und weist den eingegebenen Wert der spezifizierten Variable zu. Die Eingabe kann sowohl dezimal als auch hexadezimal (% vor dem Wert) erfolgen. Bei fehlerhafter Eingabe wird ein Fragezeichen ausgegeben und eine neue Eingabe erwartet. INPUT ist auch als Funktion verfügbar.

Zur Programmsteuerung dienen die STOP- und END-Anweisung.

```
STOP
END
```

Die Programmzeile, in der STOP auftritt, wird zu Ende abgearbeitet. Danach wird der Interpreter verlassen. Sie dient in Verbindung mit dem Debuggerteil zum Programmtest (Setzen von Unterbrechungspunkten). Durch Aufruf des Interpreters mit dem Eintrittspunkt %7FA kann der Programmablauf fortgesetzt werden. Die END-Anweisung kennzeichnet das Programmende. Sie bewirkt ebenfalls das Verlassen des BASIC-Interpreters.

Zur Kommentierung der Programme dient die REM-Anweisung.

```
REM [Kommentartext]
```

Bei der Anwendung dieser Anweisung sollte der zur Verfügung stehende Gesamtspeicherplatz für das Anwenderprogramm bedacht werden.

Zur Erzeugung von Warteschleifen kann die WAIT-Anweisung verwendet werden.

```
WAIT Ausdruck
```

Sie bewirkt das Durchlaufen einer Softwarewarteschleife. Der Ausdruckswert spezifiziert dabei die Anzahl der Durchläufe. Beim Wert 1 wird die Schleife einmal und beim Wert -1 (entspricht %FFFF) 65536-mal durchlaufen. Bei einer Taktfrequenz von 8 MHz dauert ein Durchlauf eine Millisekunde. Die Zeitangabe ist nicht ganz exakt, da für den Aufruf der Zeitschleife eine gewisse Zeit benötigt wird.

Mit Hilfe der CALL-Anweisung kann ein in Assemblersprache geschriebenes Programm aufgerufen werden, ohne daß Parameter übermittelt werden.

```
CALL Ausdruck
```

Der Ausdruckswert ist die Programmadresse. Das Maschinenprogramm muß mit einem RET-Befehl enden.

Mit Hilfe der TRAP-Anweisung kann laufend das Erfülltsein einer Bedingung getestet werden.

```
TRAP bedingung TO Ausdruck
```

Nach dieser Anweisung prüft der Interpreter vor der Abarbeitung jeder neuen Zeile ob die angegebene Bedingung erfüllt ist. Im positiven Fall wird zur TRAP-Routine verzweigt, deren Anfangsadresse durch Ausdruck bestimmt wird. Eine TRAP-Routine ist ist ein Unterprogramm, das mit Return endet. Sobald eine TRAP-Bedingung eingeschaltet ist, verlangsamt sich die weitere Programmabarbeitung. Durch die Anweisung

```
CLRTRP
```

wird eine aktive TRAP-Bedingung gelöscht.

Fest integriert im TINY-MPBASIC-Interpreter sind folgende Standard-Prozeduren und -Funktionen:

ABS[Parameter]	absoluter Betrag
NOT[Parameter]	logische Negation (bitweise)
GTC	Zeichen mittels GET_CHAR holen
INPUT	Zahl mittels GET_CHAR holen
PTC[Parameter]	Parameter mittels PUT_CHAR ausgeben
RL[Parameter]	links rotieren
RR[Parameter]	rechts rotieren
GETR[Register]	liefert den Inhalt des angegebenen Registers (die höherwertigen 8 Bit sind Null)
GETRR[Register]	liefert den Inhalt des spezifizierten Registers (höherwertige 8 Bit) und des nachfolgenden Registers (niederwertige 8 Bit)
GETEB[Adresse]	holt Bytewert aus externem Speicher
GETEW[Adresse]	holt Wortwert aus externem Speicher

Analog können auch Register und Bytes (bzw. Wörter) im externen Datenspeicher gesetzt werden:

SETR[Register,Wert]	Register setzen
SETRR[Register,Wert]	Doppelregister setzen
SETEB[Adresse,Wert]	externes Byte setzen
SETEW[Adresse,Wert]	externes Wort setzen

Innerhalb der verdichteten Form des BASIC-Programms werden für die Anweisungen folgende Abkürzungen verwendet:

Anweisung	Abkürzung	ASCII-Kode
LET	L	4C
GOTO	G	47
GOSUB	S	53
RETURN	R	52
IF...THEN	F....,	46...2C
ELSE...	... ;	...3E 3B
PROC	O	4F
INPUT	I	49
PRINT	P	50
PRINTHEX	H	48
STOP	T	54
END	E	45
REM	M	4D
WAIT	W	57
CALL	C	43
TRAP...TO	!....,	21...2C
CLRTRP	/	2F

Der restliche Programmtext wird ohne Leerzeichen in ASCII-Zeichen (die Zeilennummer als 2 Byte große Hexadezimalzahl) abgespeichert. Das Zeilenende wird durch %0D und das Programmende durch %00 gekennzeichnet.

8.3. Programmbeispiele

Die folgenden Demonstrationsbeispiele zur Anwendung von TINY-MPBASIC wurden aus /27/ und /46/ übernommen. Das Bild 8.4. zeigt das zur Initialisierung notwendige Assemblerprogramm. Die Bilder 8.5. und 8.6. zeigen BASIC-Demonstrationsprogramme in Quellform. Für weitergehende Informationen sei auf die oben angeführten Literaturstellen verwiesen.

```
                          1 U883BSP MODULE
                          2
                          3 ! BEISPIEL FUER START, PUT_CHAR, GET_CHAR UND HINIT !
                          4
                          5 CONSTANT
                          6 ! REGISTERVEREINBARUNGEN !
                          7     DSTH := R2
                          8     DSTL := R3
                          9     SRCH := R4
                         10     SRCL := R5
                         11
                         12 INTERNAL
P 0000                   13   START PROCEDURE
                         14   ENTRY
                         15     ABS %800
                         16   ! INTERRUPTSPRUNGE !
P 0800 30  54            17     JP @%54
P 0802 FF                18     NOP
P 0803 30  56            19     JP @%56
P 0805 FF                20     NOP
P 0806 30  58            21     JP @%58
P 0808 FF                22     NOP
P 0809 30  5A            23     JP @%5A
P 080B FF                24     NOP
```

Bild 8.4. Assemmblerprogramm für TINY-MPBASIC

```
P 080C 30  5C          25      JP @%5C
P 080E FF              26      NOP
P 080F 30  5E          27      JP @%5E
P 0811 FF              28      NOP
                       29    ! EINTRITTSPUNKTE !
P 0812 8D  0843        30      JP HINIT
P 0815 8D  081B        31      JP GET_CHAR
P 0818 8D  082F        32      JP PUT_CHAR
P 081B                 33    END START
                       34
                       35  GLOBAL
P 081B                 36    GET_CHAR PROCEDURE
                       37    ENTRY
P 081B 56  FA  F7      38      AND IRQ, #%F7       ! ALTEN REQUEST RUECKSETZEN !
                       39    GTC10:
P 081E 76  FA  08      40      TM IRQ, #8
P 0821 6B  FB          41      JR Z, GTC10         ! WARTEN AUF ZEICHEN !
P 0823 56  FA  F7      42      AND IRQ, #%F7       ! REQUEST RUECKSETZEN !
P 0826 38  F0          43      LD DSTL, SIO
P 0828 56  E3  7F      44      AND DSTL, #%7F      ! PARITAET RUECKSETZEN !
P 082B B0  E2          45      CLR DSTH
P 082D 58  E3          46      LD SRCL, DSTL       ! ECHO !
P 082F                 47    END GET_CHAR
                       48
P 082F                 49    PUT_CHAR PROCEDURE
                       50    ENTRY
P 082F 76  FA  10      51      TM IRQ, #%10        ! LETZTES ZEICHEN RAUS? !
P 0832 6B  FB          52      JR Z,PUT_CHAR
                       53    PTC10:
P 0834 56  FA  EF      54      AND IRQ, #%EF       ! REQUEST RUECKSETZEN !
P 0837 59  F0          55      LD SIO, SRCL
P 0839 A6  E5  0D      56      CP SRCL, #'%R'      ! VERGLEICH OB RETURN !
P 083C 6B  01          57      JR Z, PTC20
P 083E AF              58      RET
                       59    PTC20:
P 083F 5C  0A          60      LD SRCL, #'%L'      ! LINE-FEED !
P 0841 8B  EC          61      JR PUT_CHAR
P 0843                 62    END PUT_CHAR
                       63
                       64  INTERNAL
P 0843                 65    HINIT PROCEDURE
                       66    ENTRY
P 0843 31  F0          67      SRP #%F0            ! RP AUF KONTROLLREG. !
P 0845 8C  96          68      LD R8, #%96         ! NORMAL TIMING !
P 0847 EC  00          69      LD R14, #0
P 0849 FC  7F          70      LD R15, #%7F        ! SPH UND SPL !
P 084B 4C  02          71      LD R4, #2           ! 9600 BAUD !
P 084D 7C  49          72      LD R7, #%49
P 084F 5C  0D          73      LD R5, #%D
P 0851 1C  43          74      LD R1, #%43         ! TMR !
P 0853 BC  02          75      LD R11, #%2         ! IMR !
P 0855 9F              76      EI
P 0856 31  10          77      SRP #%10
P 0858 5C  0D          78      LD SRCL, #'%R'      ! CR AUSGEBEN !
P 085A D6  0834        79      CALL PTC10
P 085D                 80    END HINIT
                       81
                       82  ! NACHFOLGEND ANWENDERPROGRAMM !
                       83
                       84  END U883BSP
```

Bild 8.4. Assemblerprogramm für TINY-MPBASIC (Fortsetzung)

```
 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
```

Bild 8.5. TINY-MPBASIC-Programmbeispiel 1

```
  5 REM PROGRAMM LISTET TABELLEN ZUR UMRECHNUNG
  6 REM HEXADEZIMALZAHL IN DEZIMALZAHL UND
  7 REM ASCII-ZEICHEN AUF
 10 LET A=0, B=0, C=0
 20 LET A=A+1, B=B+%10, C=C+%100
 25 PRINT "HEXADEZIMAL/DEZIMAL-UMWANDLUNG"
 30 PRINTHEX A,
 40 PRINT A, " --> ",
 50 PRINTHEX B,
 60 PRINT B, " --> ",
 70 PRINTHEX C,
 80 PRINT C, " --> ",
 90 IF A >= %F THEN GOTO 110
100 GOTO 20
110 PRINT "ASCII-ZEICHENTABELLE"
120 LET A=%20
125 B=6
130 PRINTHEX A, "  ",
140 PROC PTC[A]
150 PRINT "     ",
160 LET B=B-1, A=A+%10
170 IF B > 0 THEN GOTO 130
180 PRINT
190 LET A=A-%5F
200 IF A = %30 THEN GOTO 220
210 GOTO 125
220 PRINT
230 END
```

Bild 8.6. TINY-MPBASIC-Programmbeispiel 2

9. U 880 PASCAL-Interpreter

Die Programmiersprache PASCAL hat sich in den letzten Jahren einen entscheidenden Platz bei der Aus- und Weiterbildung, der System- und Anwenderprogrammierung erorbert. Der Grund für diesen Erfolg liegt sowohl in der klaren, logischen Struktur von PASCAL als auch in der guten Übertragbarkeit auf verschiedene Rechnertypen (Portabilität). Das Spektrum der Implementationen reicht von modernen Großrechnern bis zu Mikrorechnern.
Unter dem Betriebssystem UDOS existiert für den U880 ein leistungsfähiges dialogorientiertes PASCAL-System, das für ein breites Anwendungsspektrum (von kommerziellen Projekten bis zu numerischen Berechnungen) genutzt werden kann.
In diesem Abschnitt werden die notwendigen Informationen zur Übersetzung und Abarbeitung von PASCAL-Quellprogrammen unter dem Betriebssystem UDOS dargelegt. Ebenso werden die Einschränkungen und Erweiterungen dieser PASCAL-Version erläutert und Hinweise zur Handhabung gegeben.
Für eine ausführliche PASCAL-Sprachbeschreibung sei auf die Standardliteratur /49/, /50/, /51/ u.a. verwiesen.
Das PASCAL-System benutzt einige Möglichkeiten des Betriebssystems UDOS, wie reihenweise Ein-/Ausgabe zur Dateiarbeit und die Speicherverwaltung (memory manager). Eine umfassende Beschreibung dieser Möglichkeiten ist der Anwenderdokumentation zum Betriebssystem /43/ zu entnehmen.

9.1. Arbeitsweise

Das PASCAL-System besteht aus drei Programmen: einem PASCAL-Compiler, einem Post-Prozessor und einem Interpreter. Der Compiler, selbst ein PASCAL-Programm, erzeugt P-Kode, einen symbolischen Zwischenkode für einen stack-orientierten Rechner. Der Post-Prozessor konvertiert den vom Compiler erzeugten P-Kode zur Abarbeitung auf kleineren Rechnern in einen verdichteten Objektkode, der dann unter Steuerung des Interpreters abgearbeitet wird.

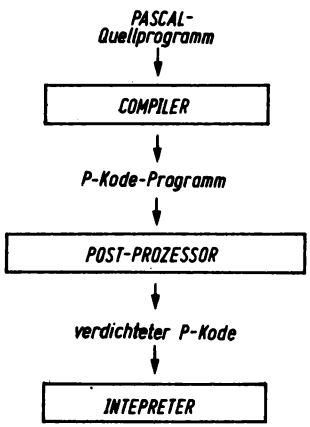


Bild 9.1. PASCAL-Ablaufübersicht

Um ein PASCAL-Programm abzuarbeiten, muß das Quellprogramm vom Compiler übersetzt werden. Der vom Compiler erzeugte P-Kode ist die Eingabe für den Post-Prozessor, der den verdichteten P-Kode erzeugt. Dieser verdichtete P-Kode wird dann interpretativ abgearbeitet. Das Bild 9.1. verdeutlicht diesen Prozeß.
Die mögliche Größe eines zu übersetzenden oder abzuarbeitenden PASCAL-Programms hängt von der Größe des verfügbaren zusammenhängenden Speicherbereichs ab. Deshalb sollten alle vom Anwender während der Arbeit mit PASCAL noch benötigten Programme (z.B.: Ein-/Ausgabetreiber) auf die höchstmöglichen Adressen geladen werden. Dadurch wird der größtmögliche zusammenhängende Speicherbereich für Anwenderprogramme zur Verfügung gestellt. Das Bild 9.2. verdeutlicht den Speicheraufbau bei der Übersetzung oder Abarbeitung von PASCAL-Programmen unter dem Betriebssystem UDOS.

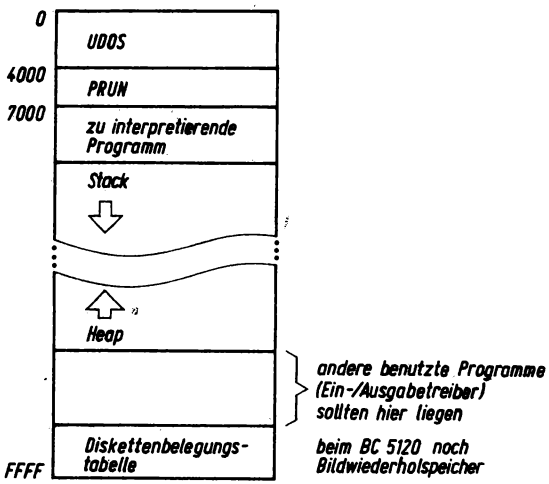


Bild 9.2. Speicherzuordnung

Das zu interpretierende Programm ist der verdichtete P-Kode entweder eines PASCAL-Anwenderprogramms, des Compilers oder des Post-Prozessors. Der von PRUN benötigte Speicherbereich besteht aus dem Interpreter, einem vom Interpreter benutzten Ein-/Ausgabepaket und einem kleinen Startmodul, der das zu interpretierende Programm vor dem Aufruf des Interpreters einliest. Der für den Stack- und Heap-Bereich benötigte Speicherbereich wird folgendermaßen dynamisch zugewiesen:

Bei Aufruf von PRUN veranlaßt der Startmodul eine Anforderung an den UDOS Memory Manager zur Ermittlung des größten verfügbaren Speichersegments.

Das zu interpretierende Programm wird an den Anfang dieses Bereichs eingelesen.

Der Rest dieses Bereichs wird für den Stack- und Heapaufbau benutzt. Der Stack wächst aufwärts zu den höheren Adressen und der Heap wächst rückwärts zu den niederen Adressen.

Je größer der verfügbare Stack-/Heap-Bereich, desto größer können die abzuarbeitenden Programme sein. PRUN wird auf die niedrigstmögliche Adresse im System (4000H) geladen. Wie aus dem Bild der Speicherzuordnung ersichtlich wird, bleibt kein Platz für Ein-/Ausgabepuffer, d.h., nach einmaligem Aufruf zur Speicherzuordnung übernimmt das PASCAL-System die Steuerung der Speicherzuweisung. Beim Eröffnen einer Datei liegt der Pufferbereich im PASCAL-Stackbereich. Beim Schließen einer Datei wird dieser Bereich nicht sofort freigegeben, erst bei Abschluß der entsprechenden Prozedur oder des Programms. Deshalb sollten Dateien in dem Niveau eröffnet werden, in dem sie deklariert werden.

Vor Aufruf des Compilers muß ein PASCAL-Quellprogramm mit dem UDOS-Editor erstellt werden. Dabei sind einige Einschränkungen und Erweiterungen des Compilers (siehe Abschn. 9.4.) zu beachten.
Der Compiler erzeugt wahlweise eine Listing- und eine Objektdatei. Die Namen dieser beiden Dateien werden in der Kommandozeile zum Aufruf des Compilers angegeben. Ein einfaches Beispiel für ein Listing wird im Bild 9.3. angegeben.
Die Listingdatei enthält die numerierten Quellzeilen und eine Anzeige des Verschachtelungsniveaus. Daneben existiert eine Spalte, in der kumulativ die Anzahl der erzeugten P-Anweisungen angegeben wird. Zum Abschluß eines jeden Listings werden die Anzahl der erkannten Fehler, die Anzahl der gelesenen Zeilen, die Anzahl der übersetzten Prozeduren und die Anzahl der erzeugten P-Anweisungen angegeben. Erkannte Fehler werden unter der entsprechenden Quellzeile angezeigt. Die Fehlernummern des Compilers und ihre Erläuterungen sind der Anwenderdokumentation /48/ zu entnehmen.

```
LINE  P_LC LVL    PASCAL   2.0

   1     4  1) program BSPXYZ(input, output);
   2     4  1)
   3     8  1) var  X : real;
   4    48  1)      Y : array[1..20] of integer;
   5    48  1)
   6     0  1) begin
   7    17  1)      readln(X,Y[10]);
   8    30  1)      writeln(X,Y[10]);
   9    48  1) end.


****    NO SYNTAX ERROR(S) DETECTED.
****     9 LINE(S) READ,    0 PROCEDURE(S) COMPILED,
****    31 P-INSTRUCTIONS GENERATED.
```

Bild 9.3. PASCAL-Compilerlisting

Erläuterung zur P-Kode-Zuweisungsspalte	
Zeilen-Nr.	Erläuterung
1	Vereinbarungsteil des Programms, 4 zeigt den Wert des P-Kode-Zuweisungszählers (LC) an. LC beginnt bei 0. Die Programmzeile zeigt an, daß zwei Ein-/Ausgabedateien benötigt werden. Entsprechend wurde der LC erhöht, um die notwendigen 2 Byte Stackbereich für jede Datei zu reservieren.
2	Der LC wurde um vier Byte erhöht, da vier Byte für die in der Zeile vereinbarte Variable X benötigt werden.
3	Der LC wurde um 40 erhöht, da die Größe des Feldes Y 20 ist (Erklärung siehe Abschn. 9.2. Datentypen).
4	Das Schlüsselwort BEGIN kennzeichnet das Ende des Vereinbarungsteils, d.h., der P-Befehlszähler wird für diese Prozedur, in diesem Fall das Hauptprogramm, neu initialisiert.
5	Nach Verarbeitung von READLN wurden 17 P-Anweisungen erzeugt.
6	Weitere 13 P-Anweisungen wurden nach Verarbeitung von WRITELN erzeugt.
7	Die letzte Zeile des Programms oder der Routine zeigt den kumulativen Wert des LC an.

Der Compiler erzeugt standardmäßig eine Listing- und Objektdatei. Falls diese Erzeugung nicht gewünscht wird, ist es notwendig, im Quellprogramm einen Kommentar der Form: (*$L-,C-*) unterzubringen (L für Listing- und C für P-Kode-Datei; - unterdrückt Erzeugung, + erlaubt Erzeugung).

Bei der Unterdrückung der Listingerzeugung wird noch eine kurze Ausgabe, die aus dem Listingkopf und den oben beschriebenen Abschlußmeldungen besteht, generiert. Zusätzlich wird für jede verarbeitete Quellzeile das Zeichen "+" ausgegeben. Nach jeweils 25 Zeilen wird zusätzlich die Anzahl der bis dahin verarbeiteten Quellzeilen ausgegeben.

Der Compileraufruf erfolgt durch folgende Kommandozeile:

```
%PRUN PASCAL,Dateiname,Dn1,Dn2,Dn3
```

wobei Dateiname die Quelldatei, Dn1 die Listingdatei, Dn2 die Table-Datei und Dn3 die P-Kode-Datei kennzeichnet. Dn2 und Dn3 sind Eingabedateien für den Post-Prozessor. Wenn die Ausgabedateien bereits existieren, dann werden sie überschrieben.
Durch Weglassen von Dn1 kann erreicht werden, daß keine Listingdatei angelegt wird. Die Ausgabe des Listings erfolgt dann über den Bildschirm.
(z.B.: übersetzt

```
%PRUN PASCAL,BSPXYZ,,TEMP.T,TEMP.P
```

das Programm BSPXYZ, gibt das Listing über den Bildschirm aus und erzeugt die Table-Datei TEMP.T und die P-Kode-Datei TEMP.P.)

Der Post-Prozessor erzeugt aus den vom Compiler erstellten Dateien einen verdichteten Objektkode (der dann interpretativ abgearbeitet wird) und eine Listingausgabe, die die Abarbeitung des Post-Prozessors dokumentiert. Diese Ausgabe besteht aus:

- Kopfzeile als Meldung des Post-Prozessors.
- Der Name einer jeden verarbeiteten Routine, gefolgt durch mehrere Pluszeichen, ein Zeichen jeweils für 20 P-Anweisungen innerhalb der Routine.
- Abschlußzeile, die die Anzahl der P-Anweisungen und die Anzahl der von diesen Anweisungen belegten Bytes enthält.

Der Aufruf des Post-Prozessors erfolgt durch Eingabe folgender Kommandozeile:

```
%PRUN PASM,Dn1,Dateiname,Dn2,Dn3
```

wobei Dn1 die zu verarbeitende P-Kode-Datei und Dn2 die Table-Datei kennzeichnet. Beide wurden vom Compiler erzeugt. Dateiname kennzeichnet die Listingausgabedatei des Post-Prozessors und Dn3 die Datei, die den verdichteten P-Kode enthält. Falls diese Dateien bereits existieren, so werden sie überschrieben. Durch Weglassen des Dateinamens ist es möglich, die Listingausgabe des Post-Prozessors direkt auf den Bildschirm zu legen.

(z.B: %PRUN PASM,TEMP.P,,TEMP.T,BSPXYZ.POBJ)

Das so übersetzte Programm kann nun mit Hilfe des Interpreters abgearbeitet werden. Dazu ist folgende Kommandozeile einzugeben:

```
%PRUN Dn1 [,Dateinamen]
```

wobei Dn1 das abzuarbeitende Programm (Datei mit verdichtetem P-Kode) kennzeichnet und nachfolgend noch Dateinamen folgen können, die Dateien spezifizieren, die während der Abarbeitung des Anwenderprogramms erzeugt oder benutzt werden (maximal 16 Stück).

(z.B.: arbeitet

```
%PRUN BSPXYZ.POBJ,EINGABEDATEINAME,AUSGABEDATEINAME
```

das Programm BSPXYZ.POBJ ab, dieses Programm korrespondiert mit zwei Dateien zur Ein-/Ausgabe von Daten.)
Weitere Informationen zur Arbeit mit Ein-/Ausgabedateien werden im Abschnitt 9.3. gegeben.

Um den Übersetzungsprozeß zu vereinfachen wird die Benutzung einer UDOS-Kommandodatei empfohlen. Beispiele für solche Dateien sind:

```
V;PRUN PASCAL,#1,,TEMP.T,TEMP.P;B
ECHO PASM?-> beliebige Taste oder QUIT-> ESC-Taste
PAUSE;V;PRUN PASM,TEMP.P,,TEMP.P,#1.POBJ;DELETE TEMP.P TEMP.T Q=N;B
```

```
V;PRUN PASCAL,#1,#1.L,TEMP.T,TEMP.P
PRUN PASM,TEMP.P,,TEMP.T,#1.POBJ
DELETE TEMP.P TEMP.T Q=N;B
```

```
V;PRUND PASCAL,#1,,TEMP.T,TEMP.P
PRUND PASM,TEMP.P,,TEMP.T,#1.POBJ
DELETE TEMP.P TEMP.T Q=N;B
```

Wichtig dabei ist, daß das DO-Kommando auf die höchstmögliche Adresse im System gelegt wird !

9.2. Datentypen

Innerhalb des UDOS-PASCAL-Systems sind folgende Datentypen definiert:

CHAR

Der Standardtyp CHAR ist definiert als:

```
TYPE CHAR = MINCHAR..MAXCHAR;
```

wobei ORD(MINCHAR)=0 und ORD(MAXCHAR)=127 ist. Variablen vom Typ CHAR belegen 16 Bit (High-Byte=0, Low-Byte=ASCII Zeichen).

INTEGER

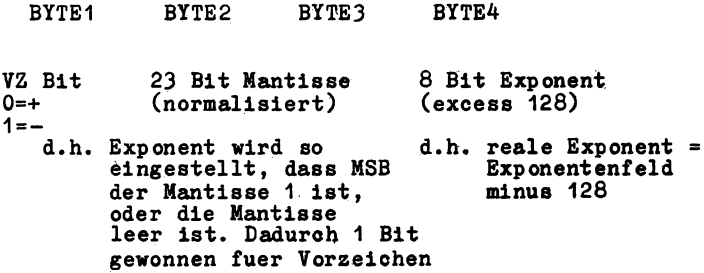
Der Standardtyp INTEGER ist definiert als:

```
TYPE INTEGER = -32768..32767;
```

Integergrößen sind in 16-Bit-Zweierkomplementarithmetik implementiert.

REAL

Das Programm PRUN enthält ein mathematisches Paket zur Verarbeitung von reellen Zahlen. Dieses Paket arbeitet mit einer 32-Bit-Gleitkommadarstellung; duales GK-Paket (4 Byte):



Beispiele:

```
7F FF FF FF = 16777215/16777216 * 2^127 = 1.7014117 * 10^38
00 00 00 81 = 1/2 * 2^1 = 1.0000000
00 00 00 01 = 1/2 * 2^(-127) = 2.9387359 * 10^(-39)
xx xx xx 00 = 0
FF FF FF FF = -16777215/16777216 * 2^127 = -1.7014117 * 10^38
```

Bild 9.4. Zahlendarstellung beim dualen Gleitkommapaket

Eine andere Version: PRUND arbeitet mit einer 64-Bit-Gleitkommadarstellung; BCD GK-Paket (8 Byte):

```
BYTE1    BYTE2    BYTE3    BYTE4          BYTE8

VZ Bit       13 BCD-Ziffern Mantisse        Exponent
0=+          (normalisiert)                 (excess 128)
1=-
      d.h. Exponent so eingestellt,         d.h. reale Exponent =
           dass das hoeherwertigste              Exponent - 128
           BCD-Zeichen ungleich 0 ist,
           oder ganze Mantisse leer ist.
```

Beispiele:

```
09 99 99 99 99 99 99 FF = .99999 99999 999 * 10^127
01 00 00 00 00 00 00 81 = 1.0
01 00 00 00 00 00 00 01 = .1 * 10^(-127)
xx xx xx xx xx xx xx 00 = 0
89 99 99 99 99 99 99 FF = -.99999 99999 999 * 10^127
```

Bild 9:5. Zahlendarstellung beim BCD-Gleitkommapaket

Falls die Anwenderprogramme viel Speicherbereich benötigen und keine reellen Zahlen benutzen, dann sollte die Version PRUNI verwendet werden (enthält dieselben Funktionen wie PRUN und PRUND, außer der Verarbeitung von reellen Zahlen).

Dabei ist zu beachten, daß die gleiche Version des Interpreters (PRUN, PRUND oder PRUNI) in allen drei Stufen der Übersetzung/Abarbeitung verwendet werden muß, ansonsten treten Fehler auf!

BOOLEAN

Ein logischer Wert wird intern in zwei Byte gespeichert:
High-Byte = 0
Low-Byte = 0 für FALSE, 1 für TRUE

ZEIGER

Eine Zeigervariable wird intern in 16 Bit gespeichert. Der Wert NIL wird durch 0 repräsentiert.

MENGEN

Mengen werden in einer variablen Anzahl von Bytes, von 4 (2 Wörter) bis 512 (256 Wörter) gespeichert. Die Anzahl ist für jede Menge fixiert und wird bei der Mengenvereinbarung ermittelt. Die Anzahl der Bytes ist immer gerade. Vor der aktuellen Menge stehen zwei Byte, die die Anzahl der Wörter der Menge enthalten.
Die größtmögliche Menge enthält maximal 4080 Elemente und wird durch 256 Wörter dargestellt (das erste Wort enthält die Länge 255. 16 Bit pro Wort x 255 = 4080 Elemente).

Somit müssen die Indexwerte der Menge und die Kardinalzahl des Basistyps zwischen 0 und 4079 liegen. Jedes Element in einer Menge wird durch ein Bit repräsentiert.

```
VAR s: SET OF 0..23;
     .
     .
BEGIN
      s := [7,8];

     .
```

s hat folgende Darstellung im Speicher:
(# ist Adresse von s)

```
#    : 0000 0000    High-Byte der Länge
#+1  : 0000 0010    Länge = 2 Wörter
#+2  : 0000 0000
#+3  : 0000 0000
#+4  : 0000 0001    Bit 8 gesetzt
#+5  : 1000 0000    Bit 7 gesetzt
```

Um Speicherplatz einzusparen, sollten Teilbereichstypen anstelle von Skalartypen bei der Mengenvereinbarung verwendet werden.

```
(z.B.:    VAR x   SET OF 0..79;      reserviert 12 Byte
          VAR x   SET OF INTEGER;    reserviert 512 Byte)
```

Skalar- und Teilbereichstypen

Wenn ein Datentyp aus einer aufgezählten Menge von Bezeichnern besteht, dann wird eine Variable dieses Typs im Speicher durch eine ganze Zahl dargestellt. Der Wert dieser Zahl ist die Ordinalzahl (Position) des Bezeichners, den sie repräsentiert.

Felder und Records

Felder und Records werden ab geraden Speicherplatzgrenzen (von Low- zu High-Adresse wachsend) abgelegt. Das Schlüsselwort PACKED wird ignoriert.

9.3. Dateiarbeit

In einem PASCAL-Anwenderprogramm können für Ein-/Ausgabeströme maximal 16 Dateien zur gleichen Zeit eröffnet sein. Es sind nur Character-Dateien (vom Typ ASCII) gestattet. Die vom PASCAL-Programm benutzten Dateien müssen in der ersten Zeile des Programms aufgeführt werden, ihre Namen sind abzukürzen. Zusätzlich müssen alle Dateien vom Typ TEXT sein und durch eine RESET- oder REWRITE-Anweisung eröffnet sein bevor sie benutzt werden.
Es gibt zwei Möglichkeiten, eine Verbindung zwischen UDOS-Dateien und internen PASCAL-Dateien herzustellen. Sie unterscheiden sich durch den Typ der verwendeten RESET/REWRITE-Anweisung.

9.3.1. Externe Dateinamensfestlegung

Hierbei erfolgt die Verbindung von PASCAL- und UDOS-Dateien über die Kommandozeile. Mit dieser Methode wird eine Verbindung entsprechend der angegebenen Reihenfolge zwischen den in der Kommandozeile aufgeführten Dateien und den internen PASCAL-Dateien hergestellt. Falls in der Kommandozeile anstelle eines Dateinamens zwei Kommas dicht stehen, dann wird der entsprechende Ein-/Ausgabestrom mit der Tastatur bzw. dem Bildschirm verbunden. Zu beachten ist, daß eine Standard RESET/REWRITE-Anweisung für Dateien die bereits eröffnet sind, ein Rücksetzen des Zeigers auf den Beginn der Datei bedeutet (die Datei wird nicht abgeschlossen). Bei Ausgabedateien geht dann zusätzlich der augenblickliche Inhalt verloren.

Folgende Beispiele zeigen diese Variante der Dateiarbeit:

```
program DATEI(0);                    (* BSP1: Erstellen einer Datei durch    *)
  var 0 : text;                      (*       ein PASCAL-Programm            *)
  begin                              (* externe Dateinamensfestlegung:       *)
    rewrite(0);                      (* Angabe des Dateinamens in der        *)
    writeln(0,'bel. Text');          (* Kommandozeile:                       *)
end.                                 (* %PRUN BSP1.POBJ,DATEI.OUT            *)


program COPY(I,0);                   (* BSP2:                                *)
  var I,0 : text;                    (* Die Dateinamen werden ausserhalb     *)
  begin                              (* in der Kommandozeile spezifiziert    *)
    reset(I); rewrite(0);            (* z.B.:                                *)
    while not eof(I)                 (* %PRUN BSP2.POBJ,DAT.IN,DAT.OUT       *)
    do begin 0^:=I^;
        put(0); get(I);
       end;
end.
```

Bild 9.6. Beispiele zur externen Dateinamensfestlegung

Anmerkung: Innerhalb von Quellprogrammen können Groß- und Kleinbuchstaben verwendet werden. In den Beispielen werden die Bezeichner von Basissymbolen klein geschrieben, um sie von den selbst gewählten Namen zu unterscheiden.

9.3.2. Interne Dateinamensfestlegung

Bei dieser Methode der Verbindung einer PASCAL-Datei mit einer UDOS-Datei wird der Dateiname innerhalb des Quellprogramms in der RESET- oder REWRITE-Anweisung angegeben.

RESET (F,ZEICHENKETTE) oder REWRITE (F,ZEICHENKETTE)

Dabei ist F die interne Dateibezeichnung, und die Zeichenkette legt den Dateinamen fest. Der Dateiname muß eine konstante Zeichenkette oder ein Feld von Zeichen sein. Wichtig ist, daß der Dateiname durch ein Leerzeichen begrenzt wird (siehe Beispiele im Bild 9.7.).
Mit dieser erweiterten RESET/REWRITE-Anweisung können Ein-/Ausgabeströme nicht mit der Tastatur bzw. dem Bildschirm verbunden werden. Dagegen ist es möglich, durch zusätzliche erweiterte RESET/REWRITE-Anweisungen Dateien zu schließen und neue Dateien zu eröffnen.

```
program DATEI1(0);                    (* BSP3:        analog BSP1,          *)
  var 0 : text;                       (* der Dateiname wird aber innerhalb  *)
  begin                               (* des PASCAL-Programms durch eine    *)
    rewrite(0,'DATEI.OUT ');          (* erweiterte REWRITE-Anweisung spez. *)
    writeln(0,'bel. Text');
end.


program COPY1(I,0);                   (* BSP4:        analog BSP2,          *)
  var I,0 : text;                     (* Dateinamen werden intern fixiert   *)
  begin
    reset(I,'DAT.IN '); rewrite(0,'DAT.OUT ');
    while not eof(I)
    do begin 0^:=I^; put(0); get(I);
       end;
end.


program DATEI2(I,0)                   (* BSP5: Dateiname wird intern durch  *)
  var                                 (*       die Variable STRING gebildet *)
    I,0    : text;
    K      : integer;
    STRING : array[1..32] of char;
  begin
    reset(I);
    K:=1;
    while not (eoln(I)) do
      begin read(I,STRING[K]);
            K:=K+1
      end;
    STRING[K]:=' ';                   (* Trennzeichen (Space) !!!           *)
    rewrite(0,STRING);
    writeln(0,'diese Datei wurde gerade eroeffnet');
end.
```

Bild 9.7. Beispiele zur internen Dateinamensfestlegung

Hinweise zur Eröffnung von Dateien:

Die Benutzung der erweiterten Form der RESET/REWRITE-Anweisung auf eine in einem höheren Niveau eröffnete Datei ist nicht gestattet.

Alle Dateien, die mit derselben logischen Einheit (d.h. mit demselben PASCAL-Dateinamen) verbunden sind, müssen die gleiche Satzlänge besitzen.

Wenn eine Datei mit der Tastatur bzw. dem Bildschirm verbunden wurde, dann kann sie nicht mehr mit der erweiterten Form der RESET/REWRITE-Anweisung eröffnet werden.

9.4. Einschränkungen und Erweiterungen

Die vorliegende Compilerversion enthält folgende Einschränkungen:

Es sind nur PASCAL-Dateien vom Typ ASCII gestattet.

Prozeduren oder Funktionen können nicht als Parameter anderen Prozeduren oder Funktionen übergeben werden.

Eine GOTO-Anweisung zu einer Adresse außerhalb einer Prozedur ist nicht gestattet.

Die Standardprozeduren PACK, UNPACK und PAGE sind nicht implementiert. Letztere kann durch WRITE(F,CHR(12)); ersetzt werden, wobei F eine Datei ist.

Bei der Ausgabe einer Zahl kann eine Feldlänge angegeben werden. Bei 32 Bit reellen Zahlen müssen jedoch mindestens 14 Zeichen; bei 64 Bit reellen Zahlen mindestens 21 Zeichen ausgegeben werden. Eine kleinere Feldlänge wird ignoriert. Bei der Ausgabe von reellen Zahlen wird der zweite Datenfeldlängenparameter ignoriert.

Prozeduren sollten nicht mehr als 400 bis 500 Quellzeilen besitzen (d.h. nicht mehr als 4 KByte umfassen).

Kommentare werden anstelle von "{" und "}" durch die Zeichenpaare "(*" und "*)" gekennzeichnet. (z.B: (* dies ist ein Kommentar *))

Die Prozedur-Verschachtelungstiefe ist auf 7 begrenzt.

Eine PASCAL-Quellzeile kann maximal 80 Zeichen enthalten.

Bereichs- und Indextests sind nicht implementiert.

Arithmetischer Überlauf wird nicht angezeigt.

Die implementierten Erweiterungen enthalten vier neue Standardprozeduren (TRAP, EXIT, MARK, RELEASE), sowie eine Reihe weiterer Möglichkeiten.

TRAP-Funktion

Die Funktion TRAP (I: integer; VAR R: bel. Typ) wurde in die Menge der vordefinierten Prozeduren mit aufgenommen, um den Aufruf eines externen Assemblerprogramms vom PASCAL-Programm aus zu ermöglichen. (Anwendung siehe Abschn. 9.4.1.)

EXIT

Die Funktion EXIT(I:integer) wurde in die Menge der vordefinierten Prozeduren mit aufgenommen. EXIT beendet das PASCAL-Programm und kann an beliebiger Stelle im Programm stehen. Das niederwertige Byte des Wertes der Integergröße wird vor Beendigung in den Akkumulator gebracht.

Andere Spracherweiterungen sind:

(1) neue Standardprozeduren MARK und RELEASE, die folgendes leisten:

MARK(P) Markiert den Heap-Bereich auf die angegebene Position (P ist Zeigertyp).
RELEASE(P) Gibt die Bereiche, die durch neue Anweisungen seit der entsprechenden MARK(P) Anweisung belegt wurden wieder frei.

Die Variable P von MARK(P) sollte bis nach Ausführung der entsprechenden RELEASE(P) Anweisung nicht geändert werden.

(2) Zusätzlich zu den Standardprozeduren RESET und REWRITE besteht die Möglichkeit, über die erweiterte Form dieser Prozeduren Dateien zu eröffnen, deren Namen innerhalb des PASCAL-Programms festgelegt werden.

(3) Beim Lesen eines Zeichens führt der Compiler, wenn es nicht notwendig ist, keinen Vorzugriff in die Puffervariable F aus (wobei F eine PASCAL-Datei ist). Deshalb braucht der Nutzer das nächste Zeichen nicht zu liefern, bevor es benötigt wird.

(4) Die CASE-Anweisung kann einen OTHERWISE-Zweig enthalten der ausgeführt wird, wenn keine Übereinstimmung mit den vorliegenden Werten des Selektors festgestellt wird. Die Marke OTHERWISE und die dazugehörigen Anweisungen müssen die letzte innerhalb der CASE-LABEL-Liste sein.

(5) Falls der Stack-Pointer größer als der Heap-Pointer wird, so erfolgt die Ausgabe der Nachricht "STACK/HEAP COLLISION" und die Ausführung wird beendet.

(6) Die Eingabe von ESCAPE bricht die Abarbeitung des Compilers und des Post-Prozessors ab.

(7) Tabulatorzeichen werden vom Compiler akzeptiert und wie Leerzeichen behandelt.

9.4.1. Kopplung mit Assemblerprogrammen

Zum Aufruf von externen Assemblerprogrammen vom PASCAL-Programm aus dient die TRAP-Funktion.

```
TRAP (I: integer; VAR R: beliebiger Typ)
```

Der erste Parameter wird vom Compiler als Funktionskode benutzt. Der zweite Parameter (bel. Typ einschließlich array oder record) kann durch das externe Assemblerprogramm vor Rückkehr ins PASCAL-Programm modifiziert werden.
Es können beliebig viele externe Assemblerroutinen aufgerufen werden. Der erste Parameter dient dabei zur Unterscheidung, welche Routine aufgerufen wird (I muß größer 0 sein !). Die Verbindung zu den externen Assemblerroutinen wird über ein kleines Startprogramm realisiert, das vor Start des PASCAL-Systems abgearbeitet werden muß.

```
PSTART:   EQU  4000H      ;PASCAL-System Startadr.
USER:     LD   HL,PTRTAB
          JP   PSTART
PTRTAB:   DEFW PROG1      ;Startadressen der Ass.-
          DEFW PROG2      ;programme
                 .
                 .
          DEFW PROGn
```

Der erste Parameter der TRAP-Anweisung entspricht der Position in der Startadressentabelle der Assemblerprogramme.
(z.B. übergibt

TRAP (2,X);

die Steuerung an das Assemblerprogramm PROG2 im oberen Beispiel.)

Beim Schreiben von Assemblerprogrammen sollten folgende Aspekte berücksichtigt werden:

- Beim ersten Aufruf des Assemblerprogramms enthält das Registerpaar HL die Adresse des High-Byte des Zeigers auf den zweiten Parameter ("X" im oberen Beispiel). Das bedeutet nach dem Befehl INC HL zeigt das Registerpaar HL auf das Low-Byte des Zeigers.

- Die Assemblerroutine muß mit einem RET-Befehl abgeschlossen werden. Vorher muß der Akkumulator gesetzt werden, um eine fehlerfreie oder fehlerhafte Aktion anzuzeigen.
  A=0 bedeutet fehlerfrei
  A=n zeigt Fehler an
  (entspricht EXIT(A), wobei A Inhalt des Akkumulators)

- Der für den Nutzer aktuelle Stack-Bereich ist der Stack-Bereich des Interpreters (200H Byte reserviert). Es gibt daher keine Garantie ob genügend Platz für den Anwender zur Verfügung steht. Da der Interpreter keine Tests auf Stack-Überlauf durchführt, wird dem Anwender empfohlen, bei Ausführung einer externen Assemblerroutine einen eigenen Stack-Bereich zu benutzen.

Nach Übersetzen der Assemblerprogramme sollten sie auf die höchstmöglichen Adressen gelegt werden (s. Abschn. 9.1.). Das dazugehörige PASCAL-Programm ist bis zum Post-Prozessor wie vorher beschrieben zu übersetzen. Zum Aufruf des Programms ist dann aber folgende Kommandozeile zu verwenden:

```
%PRUN,USER BSP.POBJ [,Dateinamen]
```

wobei USER der Name der Anwender-Assembler-Prozedurdatei und BSP.POBJ der Dateiname des verdichteten P-Kodes ist. Optional können noch weitere Dateinamen für Ein-/Ausgaben folgen.

Im folgenden Abschnitt wird ein Beispiel für die Kopplung eines PASCAL-Programms mit einem Assemblerprogramm angegeben.

9.5. Programmbeispiel

Das im Bild 9.8. gezeigte PASCAL-Programm ASSTEST benutzt zwei externe Assemblerroutinen. Die erste Routine berechnet den Wert W[2], indem es den ersten Integerwert des Feldes W mit 2 multipliziert. Die zweite Routine ermittelt die Werte W[2] und W[4] durch Multiplikation der Vorgängerwerte im Feld W mit 2. Das entsprechende Assemblerprogramm zeigt das Bild 9.9. Falls das übersetzte Assemblerprogramm als Datei PTEST und das PASCAL-Programm als Datei BSP.POBJ vorliegen, so erfolgt der Programmstart durch Eingabe von:

```
%PRUN,PTEST BSP.POBJ
```

Dabei ist darauf zu achten, daß hinter dem Wort PRUN kein Leerzeichen eingegeben wird.

```
(* dieses PASCAL-Programm benutzt zwei Assemblerroutinen *)
(* diese sind im Modul PTEST.S zu finden                 *)

program ASSTEST(input, output);

var
      W : array[1..4] of integer;
   I, J : integer;

begin

    for I:=1 to 3 do
      begin
        W[1]:=I;
        for J:=1 to 3 do
          begin
            W[3]:=J;
            trap(2,W);
            writeln(I,W[2],J,W[4]);
          end;
      end;

    for I:=1 to 3 do
      begin
        W[1]:=I;
        trap(1,W);
        writeln(I,W[2]);
      end;

end (* ASSTEST *).
```

Bild 9.8. PASCAL-Programm, das Assemblerprogramm aufruft

```
                                PTEST                                   PAGE   1
LOC   OBJ CODE M STMT SOURCE STATEMENT                                   ASM 5.9

                      1   GLOBAL ENTRY
                      2  PSTART: EQU 4000H
0000  210600    R     3  ENTRY:  LD HL,PTRTAB
0003  C30040          4          JP PSTART
                      5
0006  0A00      R     6  PTRTAB: DEFW RTN1
0008  1B00      R     7          DEFW RTN2
                      8
                      9  ; erste Assemblerroutine:
                     10  ; trap(1,W) mit W : array[1..4] of integer
                     11  ; berechnet W[2]:=2*W[1]
                     12
000A  56             13  RTN1:   LD D,(HL)
000B  23             14          INC HL
000C  5E             15          LD E,(HL)       ;DE zeigt auf das High-Byte von W[1]
000D  EB             16          EX DE,HL
000E  23             17          INC HL          ;HL zeigt auf das Low-Byte von W[1]
000F  7E             18          LD A,(HL)
0010  86             19          ADD A,(HL)
0011  23             20          INC HL
0012  23             21          INC HL
0013  77             22          LD (HL),A       ;Low-Byte von W[2]
0014  3002           23          JR NC,QUIT      ;Inkrement High-Byte ?
0016  2B             24          DEC HL
0017  34             25          INC (HL)
0018  3E00           26  QUIT:   LD A,0          ;fehlerfreie Aktion
001A  C9             27          RET
                     28
                     29  ; zweite Assemblerroutine:
                     30  ; trap(2,W) mit W : array[1..4] of integer
                     31  ; berechnet W[2]:=2*W[1] und W[4]:=2*W[3]
                     32
001B  56             33  RTN2:   LD D,(HL)
001C  23             34          INC HL
001D  5E             35          LD E,(HL)
001E  EB             36          EX DE,HL
001F  23             37          INC HL
0020  7E             38          LD A,(HL)
0021  86             39          ADD A,(HL)
0022  23             40          INC HL
0023  23             41          INC HL
0024  77             42          LD (HL),A
0025  3003           43          JR NC,NEXT
0027  2B             44          DEC HL
0028  34             45          INC (HL)
0029  23             46          INC HL
002A  23             47  NEXT:   INC HL
002B  23             48          INC HL          ;zeigt auf Low-Byte von W[3]
002C  7E             49          LD A,(HL)
002D  86             50          ADD A,(HL)
002E  23             51          INC HL
002F  23             52          INC HL
0030  77             53          LD (HL),A       ;Low-Byte von W[4]
0031  3002           54          JR NC,CONT
0033  2B             55          DEC HL
0034  34             56          INC (HL)
0035  3E00           57  CONT:   LD A,0          ;fehlerfreie Aktion
0037  C9             58          RET
```

Bild 9.9. Assemblerprogramm zur Kopplung mit PASCAL

Anhang

Befehlskodierung U8000 (oberes Befehlsbyte/Teil 1)

unteres Halbbyte

oberes Halbbyte

HEX	Ø	1	2	3	4	5	6	7
Ø	ADDB R←IR 2.1 R←IM	ADD R←IR 2.1 R←IM	SUBB R←IR 2.1 R←IM	SUB R←IR 2.1 R←IM	ORB R←IR 2.1 R←IM	OR R←IR 2.1 R←IM	ANDB R←IR 2.1 R←IM	AND R←IR 2.1 R←IM
1	CPL R←IR 2.1 R←IM	PUSHL IR←IR 2.1	SUBL R←IR 2.1 R←IM	PUSH IR←IR 2.2	LDL R←IR 2.1 R←IM	POPL IR←IR 2.1	ADDL R←IR 2.1 R←IM	POP IR←IR 2.1
2	LDB R←IR 2.1 R←IM	LD R←IR 2.1 R←IM	RESB IR,IM 1.2 R,R 6.3	RES IR,IM 1.2 R,R 6.3	SETB IR,IM 1.2 R,R 6.3	SET IR,IM 1.2 R,R 6.3	BITB IR,IM 1.2 R,R 6.3	BIT IR,IM 1.2 R,R 6.3
3	LDB R←BA 4.4 LDRB R←RA 4.4	LD R←BA 4.4 LDR R←RA 4.4	LDB BA←R 4.2 LDRB RA←R 4.2	LD BA←R 4.2 LDR RA←R 4.2	LDA R←BA 4.4 LDAR R←RA 4.4	LDL R←BA 4.4 LDRL R←RA 4.4		LDL BA←R 4.2 LDRL RA←R 4.2
4	ADDB R←X 2.1 R←DA	ADD R←X 2.1 R←DA	SUBB R←X 2.1 R←DA	SUB R←X 2.1 R←DA	ORB R←X 2.1 R←DA	OR R←X 2.1 R←DA	ANDB R←X 2.1 R←DA	AND R←X 2.1 R←DA
5	CPL R←X 2.1 R←DA	PUSHL IR←X 2.2 IR←DA	SUBL R←X 2.1 R←DA	PUSH IR←X 2.2 IR DA	LDL R←X 2.1 R←DA	POPL X←IR 2.1 DA←IR	ADDL R←X 2.1 R←DA	POP X←IR 2.1 DA←IR
6	LDB R←X 2.1 R←DA	LD R←X 2.1 R←DA	RESB X,IM 1.2 DA,IM	RES X,IM 1.2 DA,IM	SETB X,IM 1.2 DA,IM	SET X,IM 1.2 DA,IM	BITB X,IM 1.2 DA,IM	BIT X,IM 1.2 DA,IM
7	LDB R←BX 4.3	LD R←BX 4.3	LDB BX←R 4.1	LD BX←R 4.1	LDA R←BX 4.3	LDL R←R 2.1	LDA R←X 2.1 R←DA	LDL BX←R 4.1
8	ADDB R←R 2.1	ADD R←R 2.1	SUBB R←R 2.1	SUB R←R 2.1	ORB R←R 2.1	OR R←R 2.1	ANDB R←R 2.1	AND R←R 2.1
9	CPL R←R 2.1	PUSHL IR←R 2.1	SUBL R←R 2.1	PUSH IR←R 2.1	LDL R←R 2.1	POPL R←IR 2.1	ADDL R←R 2.1	POP R←IR 2.1
A	LDB R←R 2.1	LD R←R 2.1	RESB R,IM	RES R,IM 1.2	SETB R,IM 1.2	SET R,IM 1.2	BITB R,IM 1.2	BIT R,IM 1.2
B	DAB R 1.1	EXTS EXTSB EXTSL R 1.1	⑤	⑤	ADCB R←R 2.1	ADC R←R 2.1	SBCB R←R 2.1	SBC R←R 2.1
C	LDB R←IM 3.2	→						
D	CALR PC←RA 3.4	→						
E	JR PC←RA 3.3	→						
F	DJNZ DBJNZ 3.1	→						

Befehlskodierung U8000 (oberes Befehlsbyte/Teil 2)

unteres Halbbyte

8	9	A	B	C	D	E	F	HEX
XORB R←IR 2.1 R←IM	XOR R←IR 2.1 R←IM	CPB R,IR 2.1 R,IM	CP R,IR 2.1 R,IM	②	②	E	E	Ø
MULTL R←IR 2.1 R←IM	MULT R←IR 2.1 R←IM	DIVL R←IR 2.1 R←IM	DIV R←IR 2.1 R←IM	③	LDL IR←R 2.2	JP PC←IR 1.3	CALL PC←IR 1.1	1
INCB IR←IM 1.2	INC IR←IM 1.2	DECB IR←IM 1.2	DEC IR←IM 1.2	EXB R↔IR 2.1	EX R↔IR 2.1	LDB IR←R 2.2	LD IR←R 2.2	2
	LDPS PS←IR 2.3	④	④	INB R←IR 7.3	IN R←IR 7.3	OUTB IR←R 7.4	OUT IR←R 7.4	3
XORB R←X 2.1 R←DA	XOR R←X 2.1 R←DA	CPB R,X 2.1 R,DA	CP R,X 2.1 R,DA	②	②	E	E	4
MULTL R←X 2.1 R←DA	MULT R X 2.1 R DA	DIVL R←X 2.1 R←DA	DIV R←X 2.1 R←DA	③	LDL X←R 2.2 DA←R	JP PC←X 1.3 PC←DA	CALL PC←X 1.1 PC←DA	5
INCB X←IM 1.2 DA←IM	INC X←IM 1.2 DA←IM	DECB X←IM 1.2 DA←IM	DEC X←IM 1.2 DA←IM	EXB R↔X 2.1 R↔DA	EX R↔X 2.1 R↔DA	LDB X←R 2.2 DA←R	LD X←R 2.2 DA←R	6
	LDPS PS←X 2.3 PS←DA	HALT 9.3	⑧	EI DI VI,NVI 9.2	⑧		SC 9.5	7
XORB R←R 2.1	XOR R←R 2.1	CPB R,R 2.1	CP R,R 2.1	②	②	E	E	8
MULTL R←R 2.1	MULT R←R 2.1	DIVL R←R 2.1	DIV R←R 2.1	③		RET 9.4		9
INCB R←IM 1.2	INC R←IM 1.2	DECB R←IM 1.2	DEC R←IM 1.2	EXB R↔R 2.1	EX R↔R 2.1	TCCB R 1.4	TCC R 1.4	A
⑥		⑦	⑦	RRDB R 2.1	LDK R←IM 1.2	RLDB R 2.1		B
								C
								D
								E
								F

oberes Halbbyte

Ė steht für EPU-Extended-Befehle

Befehlskodierung U8000 (unteres Befehlsbyte/Teil 3)

unteres Halbbyte

oberes Halbbyte

②	Ø	1	2	3	4	5	6	7
ØC	COMB IR 1.1	CPB IR,IM 5.1	NEGB IR 1.1		TESTB IR 1.1	LDB IR←IM 5.1	TSETB IR 1.1	
ØD	COM IR 1.1	CP IR,IM 5.1	NEG IR 1.1		TEST IR 1.1	LD IR←IM 5.1	TSET IR 1.1	
4C	COMB X 1.1 DA	CPB X,IM 5.1 DA,IM	NEGB X 1.1 DA		TESTB X 1.1 DA	LDB X←IM 5.1 DA←IM	TSETB X 1.1 DA	
4D	COM X 1.1 DA	CP X,IM 5.1 DA,IM	NEG X 1.1 DA		TEST X 1.1 DA	LD X←IM 5.1 DA←IM	TSET X 1.1 DA	
8C	COMB R 1.1	LDCTLB R←FLGS 8.1	NEGB R 1.1		TESTB R 1.1		TSETB R 1.1	
8D	COM R 1.1	SETFLG FLGS←IM 9.1	NEG R 1.1	RESFLG FLGS←IM 9.1	TEST R 1.1	COMFLG FLGS←FLGS 9.1	TSET R 1.1	NOP

④	Ø	1	2	3	4	5	6	7
3A	INIB IR←IR 6.4 INIRB IR←IR 6.4	SINIB IR←IR 6.4 SINIRB IR←IR 6.4	OUTIB IR←IR 6.4 OUTIRB IR←IR 6.4	SOUTIB IR←IR 6.4 SOUTIRB IR←IR 6.4	INB R←DA 7.1	SINB R←DA 7.1	OUTB DA←R 7.2	SOUTB DA←R 7.2
3B	INI IR←IR 6.4 INIR IR←IR 6.4	SINI IR←IR 6.4 SINIR IR←IR 6.4	OUTI IR←IR 6.4 OUTIR IR←IR 6.4	SOUTI IR←IR 6.4 SOUTIR IR←IR 6.4	IN R←DA 7.1	SIN R←DA 7.1	OUT DA←R 7.2	SOUT DA←R 7.2

⑤	Ø	1	2	3	4	5	6	7
B2	RLB(1) R 1.1	SLLB SRLB R 5.1	RLB(2) R 1.1	SDLB R 6.6	RRB(1) R 1.1		RRB(2) R 1.1	
B3	RL(1) R 1.1	SLL SRL R 5.1	RL(2) R 1.1	SDL R 6.6	RR(1) R 1.1	SLLL SRLL R 5.1	RR(2) R 1.1	SDLL R 6.6

⑥	Ø	1	2	3	4	5	6	7
B8	TRIB IR 6.4		TRTIB IR 6.4		TRIRB IR 6.4		TRTIRB IR 6.4	

⑦	Ø	1	2	3	4	5	6	7
BA	CPIB IR 6.5	LDIB LDIRB IR←IR 6.4	CPSIB IR 6.5		CPIRB IR 6.5		CPSIRB IR 6.5	
BB	CPI IR 6.5	LDI LDIR IR←IR 6.4	CPSI IR 6.5		CPIR IR 6.5		CPSIR IR 6.5	

⑧	Ø	1	2	3	4	5	6	7
7B	IRET							
7D			LDCTL R ← FCW	LDCTL R ← RFRSH	LDCTL R ← PSAPSEG	LDCTL R ← PSAPOR	LDCTL R ← NSPSEG	LDCTL R ← NSPOFF

Formatangaben siehe Seite 69

Befehlskodierung U8000 (unteres Befehlsbyte/Teil 4)

unteres Halbbyte

8	9	
CLRB IR 1.1		ØC
CLR IR 1.1	PUSH IR←IM 5.1	ØD
CLRB X DA 1.1		4C
CLR X DA 1.1		4D
CLRB R 1.1	LDCTLB FLGS←R 8.2	8C
CLR R 1.1		8D

③	Ø	1
1C		LDM R←IR 6.1
5C		LDM R←X 6.1 R-DA
9C		

8	9
TESTL IR 1.1	LDM IR←R 6.2
TESTL X DA 1.1	X←R 6.2 DA←R
TESTL R 1.1	

8	9	A	B	
INDB IR←IR 6.4	SINDB IR←IR 6.4	OUTDB IR←IR 6.4	SOUTDB IR←IR 6.4	3A
INDRB IR←IR 6.4	SINDRB IR←IR 6.4	OUTDRB IR←IR 6.4	SOUTDBR IR←IR 6.4	
IND IR←IR 6.4	SIND IR←IR 6.4	OUTD IR←IR 6.4	SOUTD IR←IR 6.4	3B
INDR IR←IR 6.4	SINDR IR←IR 6.4	OUTDR IR←IR 6.4	SOUTDR IR←IR 6.4	

8	9	A	B	C	D	E	F	
RLCB(1) R 1.1	SLAB SRAB R 5.1	RLCB(2) R 1.1	SDAB R 6.6	RRCB(1) R 1.1		RRCB(2) R 1.1		B2
RLC(1) R 1.1	SLA SRA R 5.1	RLC(2) R 1.1	SDA R 6.6	RRC(1) R 1.1	SLAL SRAL R 5.1	RRC(2) R 1.1	SDAL R 6.6	B3
8	9	A	B	C	D	E	F	
TRDB IR 6.4		TRTDB IR 6.4		TRDRB IR 6.4		TRTDRB IR 6.4		B8
8	9	A	B	C	D	E	F	
CPDB IR 6.5	LDDB LDDRB IR←IR 6.4	CPSDB IR 6.5		CPDRB IR 6.5		CPSDRB IR 6.5		BA
CPD IR 6.5	LDD LDDR IR←IR 6.4	CPSD IR 6.5		CPDR IR 6.5		CPSDR IR 6.5		BB
8	9	A	B	C	D	E	F	
MSET	MRES	MBIT			MREQ			7B
		LDCTL FCW ←R	LDCTL RFRSH ←R	LDCTL PSAPSEG ←R	LDCTL PSAPOFF ←R	LDCTL NSPSEG ←R	LDCTL NSPOFF ←R	7D

oberes Halbbyte

Befehlskodierung U880 (Teil 1)

unteres Halbbyte \ oberes Halbbyte	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP	DJNZ e	JR NZ,e	JR NC,e	LD B,B	LD D,B	LD H,B	LD (HL),B	ADD A,B	SUB B	AND B	OR B	RET NZ	RET NC	RET PO	RET P
1	LD BC,nn	LD DE,nn	LD HL,nn	LD SP,nn	LD B,C	LD D,C	LD H,C	LD (HL),C	ADD A,C	SUB C	AND C	OR C	POP BC	POP DE	POP HL	POP AF
2	LD (BC),A	LD (DE),A	LD (nn),HL	LD (nn),A	LD B,D	LD D,D	LD H,D	LD (HL),D	ADD A,D	SUB D	AND D	OR D	JP NZ,nn	JP NC,nn	JP PO,nn	JP P,nn
3	INC BC	INC DE	INC HL	INC SP	LD B,E	LD D,E	LD H,E	LD (HL),E	ADD A,E	SUB E	AND E	OR E	JP nn	OUT (n),A	EX (SP),HL	DI
4	INC B	INC D	INC H	INC (HL)	LD B,H	LD D,H	LD H,H	LD (HL),H	ADD A,H	SUB H	AND H	OR H	CALL NZ,nn	CALL NC,nn	CALL PO,nn	CALL P,nn
5	DEC B	DEC D	DEC H	DEC (HL)	LD B,L	LD D,L	LD H,L	LD (HL),L	ADD A,L	SUB L	AND L	OR L	PUSH BC	PUSH DE	PUSH HL	PUSH AF
6	LD B,n	LD D,n	LD H,n	LD (HL),n	LD B,(HL)	LD D,(HL)	LD H,(HL)	HALT	ADD A,(HL)	SUB (HL)	AND (HL)	OR (HL)	ADD A,n	SUB n	AND n	OR n
7	RLCA	RLA	DAA	SFC	LD B,A	LD D,A	LD H,A	LD (HL),A	ADD A,A	SUB A	AND A	OR A	RST 0	RST 10H	RST 20H	RST 30H
8	EX AF,AF'	JR e	JR Z,e	JR C,e	LD C,B	LD E,B	LD L,B	LD A,B	ADC A,B	SBC A,B	XOR B	CP B	RET Z	RET C	RET PE	RET M
9	ADD HL,BC	ADD HL,DE	ADD HL,HL	ADD HL,SP	LD C,C	LD E,C	LD L,C	LD A,C	ADC A,C	SBC A,C	XOR C	CP C	RET	EXX	JP (HL)	LD SP,HL
A	LD A,(BC)	LD A,(DE)	LD HL,(nn)	LD A,(nn)	LD C,D	LD E,D	LD L,D	LD A,D	ADC A,D	SBC A,D	XOR D	CP D	JP Z,nn	JP C,nn	JP PE,nn	JP M,nn
B	DEC BC	DEC DE	DEC HL	DEC SP	LD C,E	LD E,E	LD L,E	LD A,E	ADC A,E	SBC A,E	XOR E	CP E	Tab. CB	IN A,(n)	EX DE,HL	EI
C	INC C	INC E	INC L	INC A	LD C,H	LD E,H	LD L,H	LD A,H	ADC A,H	SBC A,H	XOR H	CP H	CALL Z,nn	CALL C,nn	CALL PE,nn	CALL M,nn
D	DEC C	DEC E	DEC L	DEC A	LD C,L	LD E,L	LD L,L	LD A,L	ADC A,L	SBC A,L	XOR L	CP L	CALL nn	TAb. DD	TAb. ED	TAb. FD
E	LD C,n	LD E,n	LD L,n	LD A,n	LD C,(HL)	LD E,(HL)	LD L,(HL)	LD A,(HL)	ADC A,(HL)	SBC A,(HL)	XOR (HL)	CP (HL)	ADC A,n	SBC A,n	XOR n	CP n
F	RRCA	RRA	CPL	CCF	LD C,A	LD E,A	LD L,A	LD A,A	ADC A,A	SBC A	XOR A	CP A	RST 8	RST 18H	RST 28H	RST 38H

Befehlskodierung U880 (Teil 2/erstes Byte CB)

unteres Halbbyte / oberes Halbbyte	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	RLC B	RL B	SLA B		BIT 0,B	BIT 2,B	BIT 4,B	BIT 6,B	RES 0,B	RES 2,B	RES 4,B	RES 6,B	SET 0,B	SET 2,B	SET 4,B	SET 6,B
1	RLC C	RL C	SLA C		BIT 0,C	BIT 2,C	BIT 4,C	BIT 6,C	RES 0,C	RES 2,C	RES 4,C	RES 6,C	SET 0,C	SET 2,C	SET 4,C	SET 6,C
2	RLC D	RL D	SLA D		BIT 0,D	BIT 2,D	BIT 4,D	BIT 6,D	RES 0,D	RES 2,D	RES 4,D	RES 6,D	SET 0,D	SET 2,D	SET 4,D	SET 6,D
3	RLC E	RL E	SLA E		BIT 0,E	BIT 2,E	BIT 4,E	BIT 6,E	RES 0,E	RES 2,E	RES 4,E	RES 6,E	SET 0,E	SET 2,E	SET 4,E	SET 6,E
4	RLC H	RL H	SLA H		BIT 0,H	BIT 2,H	BIT 4,H	BIT 6,H	RES 0,H	RES 2,H	RES 4,H	RES 6,H	SET 0,H	SET 2,H	SET 4,H	SET 6,H
5	RLC L	RL L	SLA L		BIT 0,L	BIT 2,L	BIT 4,L	BIT 6,L	RES 0,L	RES 2,L	RES 4,L	RES 6,L	SET 0,L	SET 2,L	SET 4,L	SET 6,L
6	RLC (HL)	RL (HL)	SLA (HL)		BIT 0,(HL)	BIT 2,(HL)	BIT 4,(HL)	BIT 6,(HL)	RES 0,(HL)	RES 2,(HL)	RES 4,(HL)	RES 6,(HL)	SET 0,(HL)	SET 2,(HL)	SET 4,(HL)	SET 6,(HL)
7	RLC A	RL A	SLA A		BIT 0,A	BIT 2,A	BIT 4,A	BIT 6,A	RES 0,A	RES 2,A	RES 4,A	RES 6,A	SET 0,A	SET 2,A	SET 4,A	SET 6,A
8	RRC B	RR B	SRA B	SRL B	BIT 1,B	BIT 3,B	BIT 5,B	BIT 7,B	RES 1,B	RES 3,B	RES 5,B	RES 7,B	SET 1,B	SET 3,B	SET 5,B	SET 7,B
9	RRC C	RR C	SRA C	SRL C	BIT 1,C	BIT 3,C	BIT 5,C	BIT 7,C	RES 1,C	RES 3,C	RES 5,C	RES 7,C	SET 1,C	SET 3,C	SET 5,C	SET 7,C
A	RRC D	RR D	SRA D	SRL D	BIT 1,D	BIT 3,D	BIT 5,D	BIT 7,D	RES 1,D	RES 3,D	RES 5,D	RES 7,D	SET 1,D	SET 3,D	SET 5,D	SET 7,D
B	RRC E	RR E	SRA E	SRL E	BIT 1,E	BIT 3,E	BIT 5,E	BIT 7,E	RES 1,E	RES 3,E	RES 5,E	RES 7,E	SET 1,E	SET 3,E	SET 5,E	SET 7,E
C	RRC H	RR H	SRA H	SRL H	BIT 1,H	BIT 3,H	BIT 5,H	BIT 7,H	RES 1,H	RES 3,H	RES 5,H	RES 7,H	SET 1,H	SET 3,H	SET 5,H	SET 7,H
D	RRC L	RR L	SRA L	SRL L	BIT 1,L	BIT 3,L	BIT 5,L	BIT 7,L	RES 1,L	RES 3,L	RES 5,L	RES 7,L	SET 1,L	SET 3,L	SET 5,L	SET 7,L
E	RRC (HL)	RR (HL)	SRA (HL)	SRL (HL)	BIT 1,(HL)	BIT 3,(HL)	BIT 5,(HL)	BIT 7,(HL)	RES 1,(HL)	RES 3,(HL)	RES 5,(HL)	RES 7,(HL)	SET 1,(HL)	SET 3,(HL)	SET 5,(HL)	SET 7,(HL)
F	RRC A	RR A	SRA A	SRL A	BIT 1,A	BIT 3,A	BIT 5,A	BIT 7,A	RES 1,A	RES 3,A	RES 5,A	RES 7,A	SET 1,A	SET 3,A	SET 5,A	SET 7,A

Befehlskodierung U880 (Teil 3/erstes Byte ED)

unteres Halbbyte \ oberes Halbbyte	4	5	6	7	A	B
0	IN B,(C)	IN D,(C)	IN H,(C)		LDI	LDIR
1	OUT (C),B	OUT (C),D	OUT (C),H		CPI	CPIR
2	SBC HL,BC	SBC HL,DE	SBC HL,HL	SBC HL,SP	INI	INIR
3	LD (nn),BC	LD (nn),DE		LD (nn),SP	OUTI	OTIR
4	NEG					
5	RETN					
6	IM 0	IM 1				
7	LD I,A	LD A,I	RRD			
8	IN C,(C)	IN E,(C)	IN L,(C)	IN A,(C)	LDD	LDDR
9	OUT (C),C	OUT (C),E	OUT (C),L	OUT (C),A	CPD	CPDR
A	ADC HL,BC	ADC HL,DE	ADC HL,HL	ADC HL,SP	IND	INDR
B	LD BC,(nn)	LD DE,(nn)		LD SP,(nn)	OUTD	OTDR
C						
D	RETI					
E		IM 2				
F	LD R,A	LD A,R	RLD			

Befehlskodierung U880 (Teil 4)

Tabellen DD und FD

Erstes Byte DD für ii=IX, FD für ii=IY

09	ADD ii,BC	A6	AND (ii+d)
19	ADD ii,DE	AE	XOR (ii+d)
21	LD ii,nn	B6	OR (ii+d)
22	LD (nn),ii	BE	CP (ii+d)
23	INC ii	CB	Tab.DD/CB
29	ADD ii,ii	E1	POP ii
2A	LD ii,(nn)		
2B	DEC ii	E3	EX (SP),ii
34	INC (ii+d)	E5	PUSH ii
35	DEC (ii+d)	E9	JP (ii)
36	LD (ii+d),n	F9	LD SP,ii
39	ADD ii,SP		
46	LD B,(ii+d)		
4E	LD C,(ii+d)		
56	LD D,(ii+d)		
5E	LD E,(ii+d)		
66	LD H,(ii+d)		
6E	LD L,(ii+d)		
70	LD (ii+d),B		
71	LD (ii+d),C		
72	LD (ii+d),D		
73	LD (ii+d),E		
74	LD (ii+d),H		
75	LD (ii+d),L		
77	LD (ii+d),A		
7E	LD A,(ii+d)		
86	ADD A,(ii+d)		
8E	ADC A,(ii+d)		
96	SUB (ii+d)		
9E	SBC A,(ii+d)		

Tab. DD/CB+FD/CB

Erstes Byte DD(IX) oder FD(IY), zweites Byte CB

06	RLC (ii+d)
0E	RRC (ii+d)
16	RL (ii+d)
1E	RR (ii+d)
26	SLA (ii+d)
2E	SRA (ii+d)
3E	SRL (ii+d)
46	BIT 0,(ii+d)
4E	BIT 1,(ii+d)
56	BIT 2,(ii+d)
5E	BIT 3,(ii+d)
66	BIT 4,(ii+d)
6E	BIT 5,(ii+d)
76	BIT 6,(ii+d)
7E	BIT 7,(ii+d)
86	RES 0,(ii+d)
8E	RES 1,(ii+d)
96	RES 2,(ii+d)
9E	RES 3,(ii+d)
A6	RES 4,(ii+d)
AE	RES 5,(ii+d)
B6	RES 6,(ii+d)
BE	RES 7,(ii+d)
C6	SET 0,(ii+d)
CE	SET 1,(ii+d)
D6	SET 2,(ii+d)
DE	SET 3,(ii+d)
E6	SET 4,(ii+d)
EE	SET 5,(ii+d)
F6	SET 6,(ii+d)
FE	SET 7,(ii+d)

Befehlskodierung U881/U882-EMR

unteres Halbbyte

oberes Halbbyte HEX	0	1	2	3	4	5	6	7	F
0	DEC R_1	DEC IR_1	ADD r_1,r_2	ADD r_1,Ir_2	ADD R_2,R_1	ADD IR_2,R_1	ADD R_1,IM	ADD IR_1,IM	
1	RLC R_1	RLC IR_1	ADC r_1,r_2	ADC r_1,Ir_2	ADC R_2,R_1	ADC IR_2,R_1	ADC R_1,IM	ADC IR_1,IM	
2	INC R_1	INC IR_1	SUB r_1,r_2	SUB r_1,Ir_2	SUB R_2,R_1	SUB IR_2,R_1	SUB R_1,IM	SUB IR_1,IM	
3	JP IRR_1	SRP IM	SBC r_1,r_2	SBC r_1,Ir_2	SBC R_2,R_1	SBC IR_2,R_1	SBC R_1,IM	SBC IR_1,IM	
4	DA R_1	DA IR_1	OR r_1,r_2	OR r_1,Ir_2	OR R_2,R_1	OR IR_2,R_1	OR R_1,IM	OR IR_1,IM	
5	POP R_1	POP IR_1	AND r_1,r_2	AND r_1,Ir_2	AND R_2,R_1	AND IR_2,R_1	AND R_1,IM	AND IR_1,IM	
6	COM R_1	COM IR_1	TCM r_1,r_2	TCM r_1,Ir_2	TCM R_2,R_1	TCM IR_2,R_1	TCM R_1,IM	TCM IR_1,IM	
7	PUSH R_2	PUSH IR_2	TM r_1,r_2	TM r_1,Ir_2	TM R_2,R_1	TM IR_2,R_1	TM R_1,IM	TM IR_1,IM	
8	DECW RR_1	DECW IR_1	LDE r_1,Irr_2	LDEI Ir_1,Irr_2					DI
9	RL R_1	RL IR_1	LDE r_2,Irr_1	LDEI Ir_2,Irr_1					EI
A	INCW RR_1	INCW IR_1	CP r_1,r_2	CP r_1,Ir_2	CP R_2,R_1	CP IR_2,R_1	CP R_1,IM	CP IR_1,IM	RET
B	CLR R_1	CLR IR_1	XOR r_1,r_2	XOR r_1,Ir_2	XOR R_2,R_1	XOR IR_2,R_1	XOR R_1,IM	XOR IR_1,IM	IRET
C	RRC R_1	RRC IR_1	LDC r_1,Irr_2	LDCI Ir_1,Irr_2				LD r_1,x,R_2	RCF
D	SRA R_1	SRA IR_1	LDC r_2,Irr_1	LDCI Ir_2,Irr_1	CALL IRR_1		CALL DA	LD r_2,x,R_1	SCF
E	RR R_1	RR IR_1		LD r_1,Ir_2	LD R_2,R_1	LD IR_2,R_1	LD R_1,IM	LD IR_1,IM	CCF
F	SWAP R_1	SWAP IR_1		LD Ir_1,r_2		LD R_2,IR_1			NOP

2 3 1 Befehlsbyte

0	8	9	A	B	C	D	E
	LD r_1,R_2	LD r_2,R_1	DJNZ r_1,RA	JR cc,RA	LD r_1,IM	JP cc,DA	INC r_1

2 3 1 Befehlsbyte

Literaturverzeichnis

/1/ Claßen, L.: Programmierung des Mikroprozessorsystems U880 (K1520). Reihe Automatisierungstechnik, Bd. 192. 4.Auflage. Berlin: VEB Verlag Technik 1984

/2/ Oefler, U.; Claßen, L.: Mikroprozessor-Betriebssysteme, Reihe Automatisierungstechnik, Bd. 201. 3.Auflage. Berlin: VEB Verlag Technik 1984

/3/ Components Databook. Firmenschrift. Zilog Co. 1985

/4/ Paulin, G.: Kleines Lexikon der Mikrorechentechnik. Reihe Automatisierungstechnik, Bd. 206. 2. Auflage. Berlin: VEB Verlag Technik 1986

/5/ Werner, D.: Programmieren von Mikrorechnern. Berlin: VEB Verlag Technik 1983

/6/ Kieser, H.; Meder, M.: Mikroprozessortechnik. 3. Auflage. Berlin: VEB Verlag Technik 1984

/7/ Lampe, B.; Jorke, G.; Wengel, N.: Algorithmen der Mikrorechentechnik. 2. Auflage. Berlin: VEB Verlag Technik 1983

/8/ Zentrale Verarbeitungseinheit CPU U880D. Kundeninformation. Erfurt: VEB Mikroelektronik "Karl Marx" 1986

/9/ Z80-CPU Technical Manual. Firmenschrift. Zilog Co. 1977

/10/ Schaltkreis für parallele Ein- und Ausgabe PIO U855D Kundeninformation. Erfurt: VEB Mikroelektronik "Karl Marx" 1984

/11/ Z80-PIO Technical Manual. Firmenschrift. Zilog Co. 1977

/12/ Schaltkreis für serielle Ein- und Ausgabe SIO U856D. Kundeninformation. Erfurt: VEB Mikroelektronik "Karl Marx" 1984

/13/ Z80-SIO Technical Manual. Firmenschrift. Zilog Co. 1977

/14/ Schaltkreis für Zähl- und Zeitgeberfunktion CTC U857D. Kundeninformation. Erfurt: VEB Mikroelektronik "Karl Marx" 1984

/15/ Z80-CTC Technical Manual. Firmenschrift. Zilog Co. 1977

/16/ Schaltkreis für direkten Speicherzugriff (DMA) UA 858D, UB 858D. Kundeninformation. Erfurt: VEB Mikroelektronik "Karl Marx" 1984

/17/ Z80-DMA Technical Manual. Firmenschrift. Zilog Co. 1981

/18/ Münzer, B.-G.: Kopplung eines DMA-Schaltkreises an den Mikrorechner K1520. Nachrichtentechnik Elektronik 32 (1982), H. 12, S. 499-510

/19/ Zentrale Verarbeitungseinheit CPU U8000. Kundeninformation. Erfurt: VEB Mikroelektronik "Karl Marx" 1985

/20/ Walaschek, B.: U8000-Befehlsliste. Praktikantenarbeit. Technische Hochschule Ilmenau 1982

/21/ Mateosian, R.: Programming the Z8000. SYBEX Inc. 1980

/22/ Z8000-CPU Technical Manual. Firmenschrift. Zilog Co. 1981

/23/ Stuhlmüller, P.: 16-bit-Generation Z8000 Aufbau und Anwendung. München: te-wi Verlag 1980

/24/ Speicherverwaltungsbaustein MMU U8010D. Kundeninformation. Erfurt: VEB Mikroelektronik "Karl Marx" 1986

/25/ Z8010-MMU Technical Manual. Firmenschrift. Zilog Co. 1981

/26/ Einchip-Mikrorechner Übersicht UX881D, UX882D und UX883D. Kundeninformation. Erfurt: VEB Mikroelektronik "Karl Marx" 1984

/27/ Kieser, H.; Bankel, M.: Einchipmikrorechner. 1.Auflage. Berlin: VEB Verlag Technik 1986

/28/ Bennewitz, W.; Podszuweit, H.: Programmierung von Einchipmikrorechnern. Reihe Automatisierungstechnik, Bd. 215. 2. Auflage. Berlin: VEB Verlag Technik 1987

/29/ PLZ Version 1.0. User's Guide. Zilog Co. 1978

/30/ Report on the Programming Language PLZ. Zilog Co. 1978

/31/ Z8000 PLZ/ASM Assembly Language Programming Manual. Zilog Co. 1979

/32/ Z8 PLZ/ASM Assembly Language Programming Manual. Zilog Co. 1979

/33/ Handbuch UDOS-1526, Teil: Sprachbeschreibung/Manual Crosssoftware U881/882 ASM. VEB Robotron-Buchungsmaschinenwerk Karl-Marx-Stadt 1983

/34/ PLZ - Höhere Programmiersprachen für Mikrorechner. Technische Hochschule Karl-Marx-Stadt: Informationsschrift der Sektion IT. 1981

/35/ Höhere Programmiersprache für Echtzeitsteuerungen PLZRTC Vers. 1.0. Sprachbeschreibung. Technische Hochschule Karl-Marx-Stadt 1980

/36/ PLZRTC Compiler 1.0. Bedienhandbuch. Technische Hochschule Karl-Marx-Stadt 1980

/37/ Z80-Echtzeitbetriebssystemkern Kodegenerator RTCCG Vers. 1.0. Beschreibung/Bedienhandbuch. Technische Hochschule Karl-Marx-Stadt 1980

/38/ Z80-RIO Operating System User's Manual. Zilog Co. 1978

/39/ Z80-RIO Relocating Assembler and Linker User's Manual. Zilog Co. 1978

/40/ PLZ Linker User Guide. Zilog Co. 1980

/41/ Z80-RIO Text Editor User's Manual. Zilog Co. 1980

/42/ Z80-RIO File Debugger Reference Manual. Zilog Co. 1979

/43/ Betriebssystem UDOS 1526 Anwenderdokumentation. VEB Robotron-Buchungsmaschinenwerk Karl-Marx-Stadt 1983

/44/ Z80-BASIC Benutzerhandbuch. Kontron 1978

/45/ BASIC-Interpreter für UDOS 1526 Anwenderdokumentation. VEB Robotron-Buchungsmaschinenwerk Karl-Marx-Stadt 1983

/46/ Technische Beschreibung Einchipmikrorechner U883. Erfurt: VEB Mikroelektronik "Karl Marx" 1985

/47/ Z80-PASCAL User's Guide. Zilog Co. 1979

/48/ PASCAL für das Betriebssystem UDOS 1526 Anwenderdokumentation. VEB Robotron-Buchungsmaschinenwerk Karl-Marx-Stadt 1983

/49/ Jensen, K.; Wirth, N.: PASCAL User Manual and Report (second Edition). Springer-Verlag, Berlin/Heidelberg/New York 1978

/50/ Wirth, N.: Revidierter Bericht über die Programmiersprache PASCAL. Berlin: Akademie-Verlag 1976

/51/ Paulin, G.; Schiemangk, H.: Programmieren mit PASCAL. Berlin: Akademie-Verlag 1981

/52/ Schaltkreis für parallele Ein-/Ausgabe und für Zähl- und Zeitgeberfunktion U8036-CIO/U82536-CIO. Kundeninformation. Erfurt: VEB Mikroelektronik "Karl Marx" 1987

/53/ Z8036 Z-CIO / Z8536 CIO Technical Manual. Firmenschrift. Zilog Co. 1982

/54/ Initializing the CIO / Application Note. Firmenschrift. Zilog Co. 1982

/55/ Schaltkreis für serielle Ein-/Ausgaben U8030-SCC/U82536-SCC. Kundeninformation. Erfurt: VEB Mikroelektronik "Karl Marx" 1987

/56/ Z8030 Z-SCC / Z8530 SCC Technical Manual. Firmenschrift. Zilog Co. 1982

/57/ Z8530 and Z8030 SCC / Application Note. Firmenschrift. Zilog Co. 1982

/58/ 82530 Seriell Communications Controler (SCC). Firmenschrift. Intel Co. 1983

Sachwörterverzeichnis

Der Band vermittelt in umfassender Weise eine Systemübersicht über die in der DDR gefertigten Mikroprozessorsysteme U880 (8 Bit), U8000 (16 Bit) und U881/882 (Einchipmikrorechner). Dabei werden zwei grundlegende Einzelaspekte dargestellt, nämlich die Programmierung der Mikroprozessoren und Peripherieschaltkreise sowie die Softwareerarbeitung in PLZ/ASM, BASIC und PASCAL unter dem Betriebssystem UDOS.
Neu: Die universellen Ein-/Ausgabe-Peripherieschaltkreise U82530-/U8030-SCC und U82536-/U8036-CIO.

ISBN 3-341-00712-1