

Weiterbildungslehrgang – WB3

„Einchip-Mikrorechner“ (EMR)

Grundlehrgang – in der Zeit vom 16.01.1989 bis 20.01.1989

Ingenieurhochschule Berlin-Wartenberg

Zusammengestellt, bearbeitet und ergänzt im Jahre 2007
© Peter Salomon, Heinrich-Grüber-Str.159, 12621 Berlin

Die vorliegende Publikation ist urheberrechtlich geschützt. Alle Rechte, Irrtum und Änderungen vorbehalten.
Eine auch auszugsweise Vervielfältigung bedarf in jedem Fall der Genehmigung des Herausgebers.

Die hier wiedergegebenen Informationen, Dokumente, Schaltungen, Verfahren und Programmmaterialien wurden sorgfältig erarbeitet, sind jedoch ohne Rücksicht auf die Patentlage zu sehen, sowie mit keinerlei Verpflichtungen, noch juristischer Verantwortung oder Garantie in irgendeiner Art verbunden. Folglich ist jegliche Haftung ausgeschlossen, die in irgendeiner Art aus der Benutzung dieses Materials oder Teilen davon entstehen könnte. Für Mitteilung eventueller Fehler ist der Autor jederzeit dankbar.

Es wird darauf hingewiesen, daß die erwähnten Firmen- und Markennamen, sowie Produktbezeichnungen in der Regel gesetzlichem Schutz unterliegen.

Inhaltsverzeichnis

1.	Einführung	3
2.	Einsatzgebiete des EMR	4
3.	Baugruppen und Funktionsprinzip des EMR	10
4.	Erweiterung mit externen Schaltkreisen	17
5.	Programmierung	28
6.	Assemblernotation, Datenformate	38
7.	Befehlssatz, Beispiele	40
8.	Organisation des Stapelspeichers (Stack)	51
9.	Beschreibung der Steuerregister	52
10.	Verwendung und Programmierung der Ein-/Ausgabeschnittstellen	59
11.	Struktur, Programmierung und Verwendung der Zähler/Zeitgeber	64
12.	Funktion und Aufbau des seriellen Ein-/Ausgabe-Moduls	66
13.	Quellen, Organisation und Anwendung der Interruptverarbeitung	68
14.	Demonstration der interruptgesteuerten Schnittstellen-Verwaltung an einer konkreten EMR-Konfiguration	72
15.	Hinweise für den Einsatz des EMR als Prozessrechner	75
16.	Programmiertechnologie	81
Anhang 1: Befehlsliste		83
Anhang 2: Literatur u.a.		88

1. Einführung

Seit 1984/85 stehen in der DDR die Einchip-Mikrorechner der Serie U88xx aus dem Kombinat Mikroelektronik Erfurt (KME), in diesem Fall der Halbleiterbauelemente-Hersteller VEB Mikroelektronik „Karl Marx“ Erfurt (MME) zur Verfügung.

Diese Bauelemente sind Hard- und Software-kompatibel zu der international bekannten Z8-Typenreihe der US-amerikanischen Firma Zilog. Allerdings gibt es einen Unterschied: Die Gehäuseausführung der so genannten „Entwicklungsvariante“ U882x wird im „QUIL-Gehäuse“ (QUIL = Quad-InLine, d.h. im Gegensatz zu DIL = Dual-InLine = 2 Anschlußpin-Reihen hier 4 Anschlußpin-Reihen) gefertigt, d.h. jeweils eine Doppelreihe versetzt im 2,5mm Rastermaß. Dieses ist nicht identisch mit dem originalen 64-Pin-DIL-Gehäuse im 1,7mm Rastermaß des Vorbildtyps. Da ein 1,7mm-Leiterplatten-Raster bisher in der DDR nicht eingeführt ist und erhebliche Probleme bei der Leiterplattenherstellung zu erwarten sind, hat man sich auf diese Ausweichlösung verständigt, allerdings auch mit der Konsequenz, dass z.B. für dieses Bauelement kommerziell keine Steckfassung zur Verfügung stehen werden. Labormäßig lassen sich jedoch z.B. aus zweireihigen EFS-Buchsenleisten recht einfach entsprechende Steckfassungen bauen (siehe dazu Pkt. 3.6).

Die Serie besteht im wesentlichen aus den folgenden drei Typenreihen, die in den angegebenen TGL klassifiziert sind:

- U881x TGL 37360
- U882x TGL 42639
- U883x TGL 38607

Das „x“ bedeutet hier mögliche weitere unterscheidbare Typen.

Anmerkung: Später sollen noch weitere Typen dazukommen, so z.B. U84xx, U86xx usw. Diese werden dann in einem entsprechenden Aufbaulehrgang behandelt.

Da der Einsatz dieser neuartigen Bauelemente einerseits in vielen Bereichen der Volkswirtschaft sehr interessant sein wird, andererseits aber dabei doch noch erheblicher Informationsbedarf besteht, sollen hier anhand praktischer Beispiele eine Einführung in die Hardwaretechnik und die Programmierung dieser Einchip-Mikrorechnern gegeben werden.

2. Einsatzgebiete des EMR

Während das schon seit 1980 zur Verfügung stehende Mikrorechner-System U880 hauptsächlich für Zwecke der EDV eingesetzt wird (Bürocomputer, PC1715), ist das neue Einchipmikrorechner-System prädestiniert für den Einsatz in der Prozess-Steuer-, Meß- und Regeltechnik der industriellen Elektronik.

Gründe sind hierbei insbesondere die hohe Komplexität der EMR, so dass schon mit wenigen peripheren Bauelementen und demzufolge minimalem Leiterplattenaufwand die grundlegenden Eigenschaften:

- Ein- und Ausgabe von Daten,
- Datenbearbeitung und
- Datenspeicherung

einfacher Einchip-Mikrorechnersysteme realisiert werden können.

Hinzu kommt noch die wesentlich geringere Befehlsausführungszeit von ca. $2,5\mu\text{s}$ im Vergleich zu $4\mu\text{s}$ beim U880. Das hängt insbesondere mit der höheren Taktfrequenz von 8MHz (intern 4MHz) und dem vielfach einfacheren Befehlsaufbau (viele Einbyte-Befehle) zusammen. Hinzu kommt noch, dass durch spezielle Maßnahmen bei der Befehlsabarbeitung (sog. „Pipeline“) wertvolle Zeit eingespart wird.

Die EMR sind so konzipiert, dass sie sich universell für die verschiedensten Einsatzzwecke konfigurieren lassen.

2.1 32 Bit E/A, ohne externen Speicher

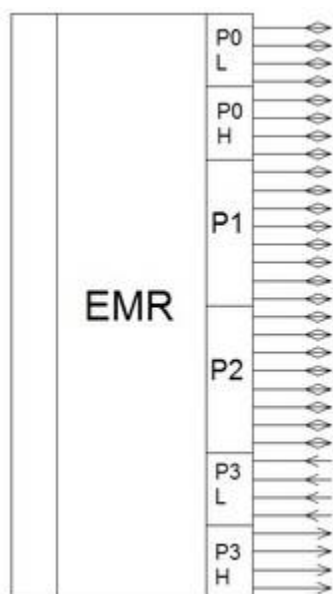


Bild 01: EMR-Konfiguration mit 32 E/A

2.2 24 Bit E/A, 256 Byte RAM/ROM

P1: Daten- und Adressbus

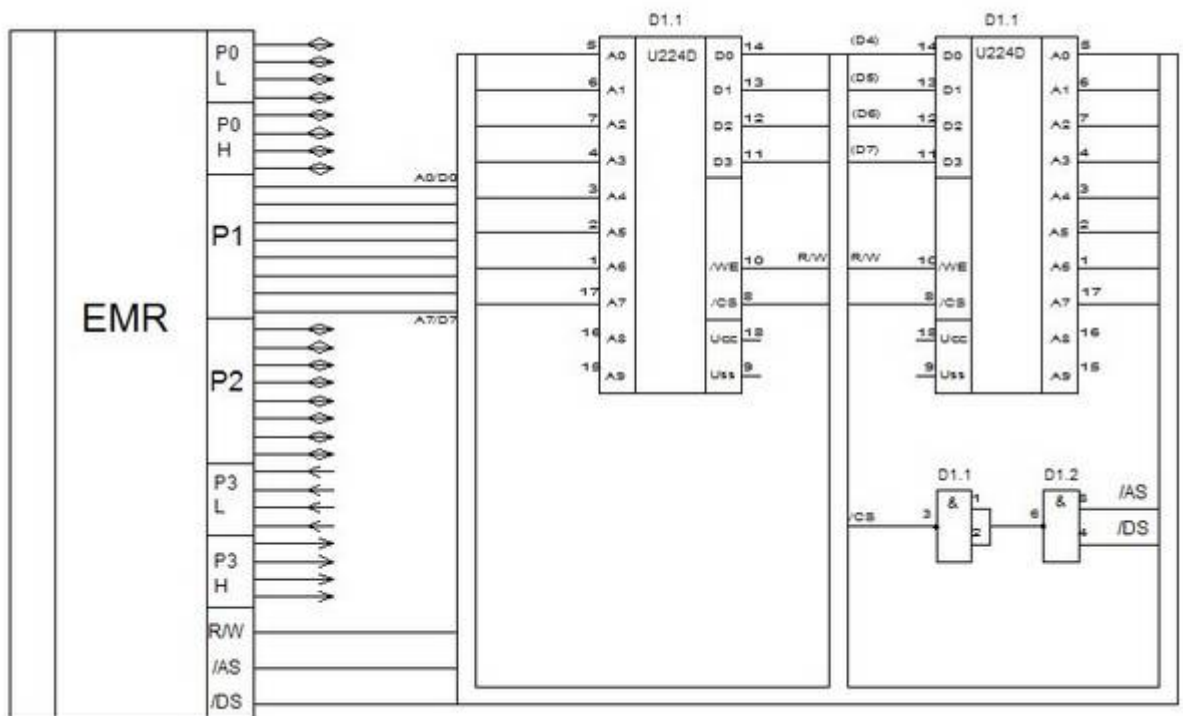


Bild 02: EMR-Konfiguration mit 24 E/A, 256 Byte RAM

Als Speicherinterface wird lediglich Port 1 benutzt, wobei Adressen und Daten gemultiplext anliegen.

Die einfache /CS-Bildung mit D1.1/2 ist eigentlich nicht ganz exakt, aber da bei internen Speicherzugriffen das /DS-Signal nicht gebildet wird, bleiben trotz Adressierung die Speicherausgänge hochohmig. Die Adresse wird im internen Latch des U224D mit der H/L-Flanke von /AS zwischengespeichert und die adressierten Daten stehen dann für die gesamte /DS-Zeit an den RAM-Ausgängen zur Verfügung.

Der Speicherbereich des 1k-SRAM U224D wird hier nur zum Teil ausgenutzt.

Für eine exakte Unterscheidung zwischen internen und externen Speicherzugriffen ist die Einbeziehung des /DM-Signals notwendig (siehe dazu Pkt. 2.3).

2.4 20 Bit E/A, 4k RAM/ROM

P1: Daten-/Adressbus, P0/0...3: Adr.



Da EPROMs leider keinen internen Adress-Speicher haben, muß die Adresse extern zwischengespeichert werden. Dazu wird das 8Bit-Latch DS8282D eingesetzt. Mit der H/L-Flanke des /AS-Signals wird die Adresse gespeichert und liegt an den Ausgängen solange an, bis eine neue L/H-Flanke eine neue Adresse an die Eingänge legt.

20 Bit E/A, 4k RAM/ROM

P1: Daten-/Adressbus, P0/0...3: Adr.

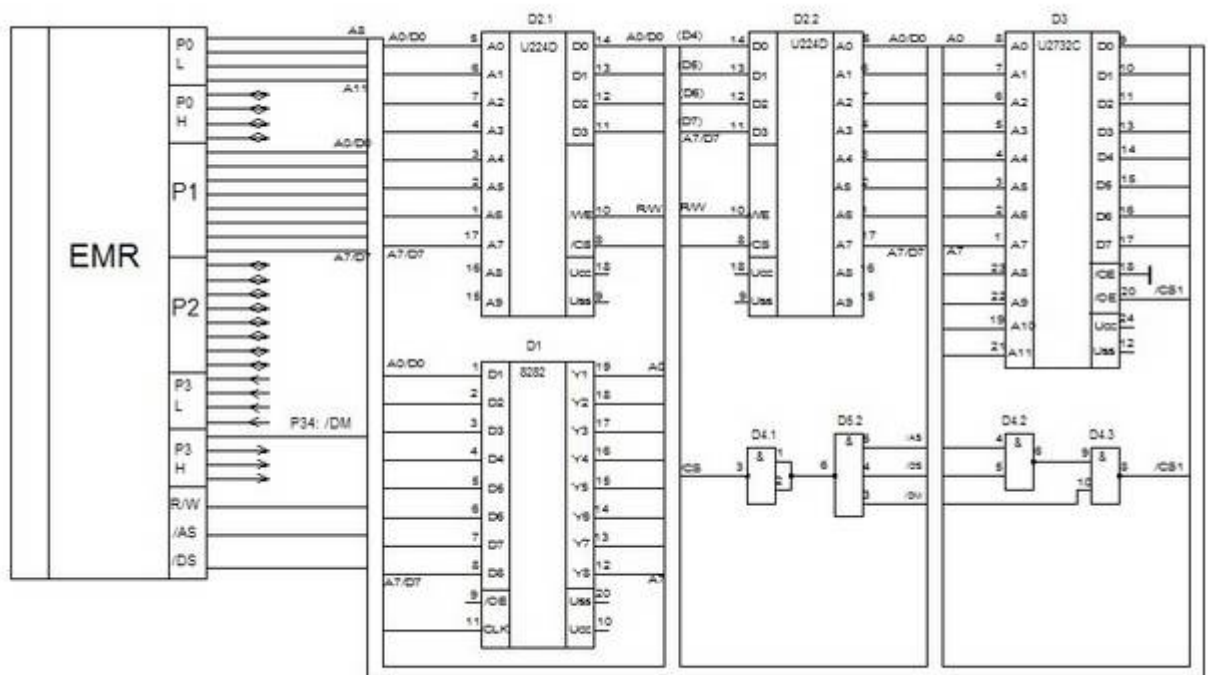


Bild 05: EMR-Konfiguration mit 19 E/A, 1kByte RAM, 4kByte EPROM

Auch hier wird wieder zusätzlich zum P1 noch ein halbes Port von P0, d.h. der L-Teil benötigt. Die Selektierung Programm-/Datenspeicher erfolgt ebenfalls mit dem /DM-Signal.

In diesem Beispiel sind nur 1kByte-Datenspeicher vorgesehen. Theoretisch könnten jedoch ebenfalls 4kByte angesprochen werden. Da z.Zt. noch keine größeren SRAMs zu Verfügung stehen ^{*)}, ist eine derartige Konfiguration natürlich mit hohem Selektionsaufwand verbunden, z.B. mit einem zusätzlichen Dekodier-IS DS8205D (siehe dazu auch Pkt 4.3).

**) Anmerkung: Die 2k-SRAM U6516D oder sogar die 8k-SRAM U6264 sind noch in Entwicklung und erste Musterstückzahlen sollen erst im Laufe des Jahres zur Verfügung stehen.*

2.6 15 Bit E/A, 62k RAM/ROM

P1: Daten-/Adressbus, P0: Adr.

P3/4: Data/Mem (/DM)

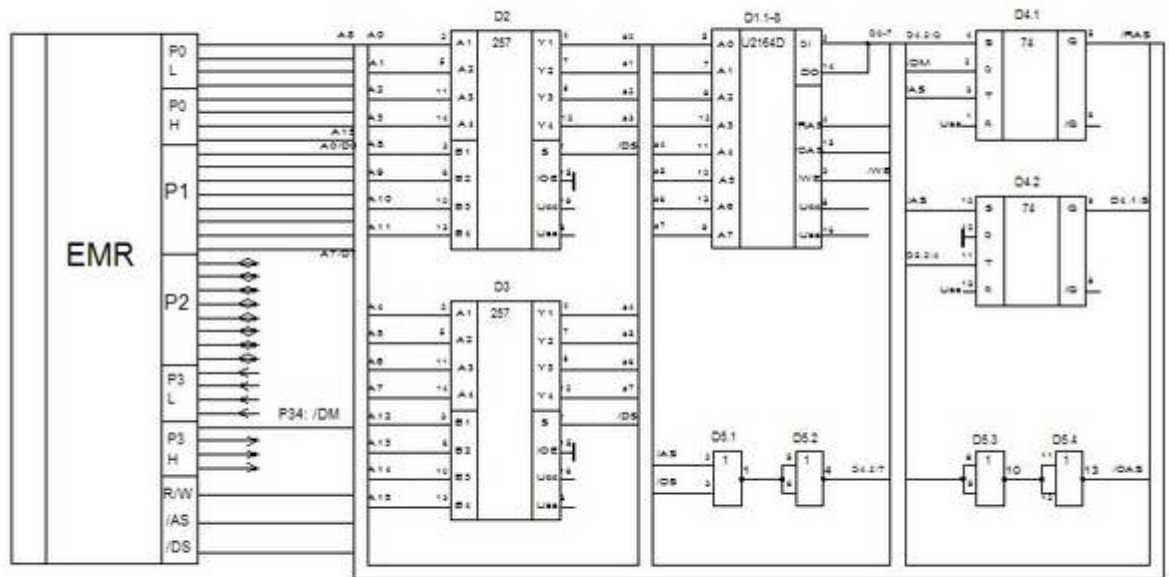


Bild 06: EMR-Konfiguration mit 15 E/A, 64kByte dRAM

Im Gegensatz zu den bisherigen Konfigurationen gestaltet sich die Anschaltung von dynamischen RAMs schon sehr aufwändig.

Für eine vollständige 16Bit-Adressierung wird außer dem Port 1 das ganze Port 0 benötigt. Dabei ist jedoch zu beachten, dass der EMR jedoch nur 62kByte adressieren kann, d.h. die ersten 2kByte sind im RAM nicht zu erreichen.

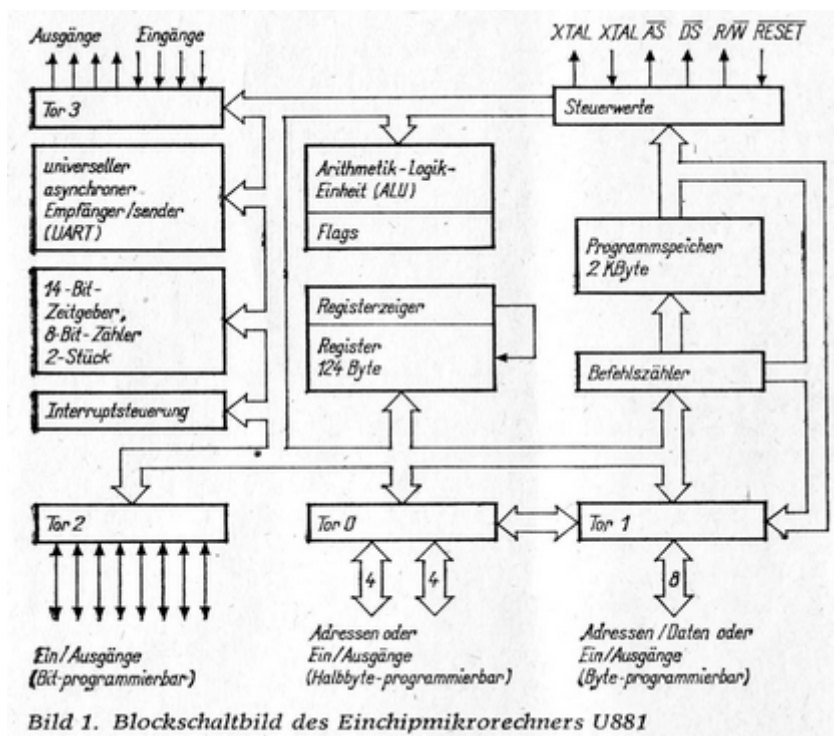
Die Adressen A0 bis A7 und A8 bis A15 werden mit den Multiplexern D2 und D3 an die Adresseingänge a0 bis a7 der RAM-Bank D1.1 bis D1.8 gelegt. Die /RAS-/CAS-Signale für die dRAMs werden wie üblich mit D-Flip-Flops (D4.1 bis D4.2) aus dem Steuersignalen /AS und /DS gebildet. Das Datenspeichersignal /DM wurde hier in die Selektierung mit einbezogen.

Dabei ist zu beachteten, dass durch die Speicher-Routinen des EMR der notwendige Refresh der dRAMs gewährleistet sein muß, d.h. dass innerhalb von 2ms auf alle 128 aufeinander folgenden Datenzellen zugegriffen wird. Anderenfalls muß eine spezielle Refresh-Routine implementiert werden, z.B. über einen Timer-gesteuerten Interrupt. Dabei bedeutet jedoch bereits eine 20%ige Auslastung des EMR.

3. Baugruppen und Funktionsprinzip des EMR

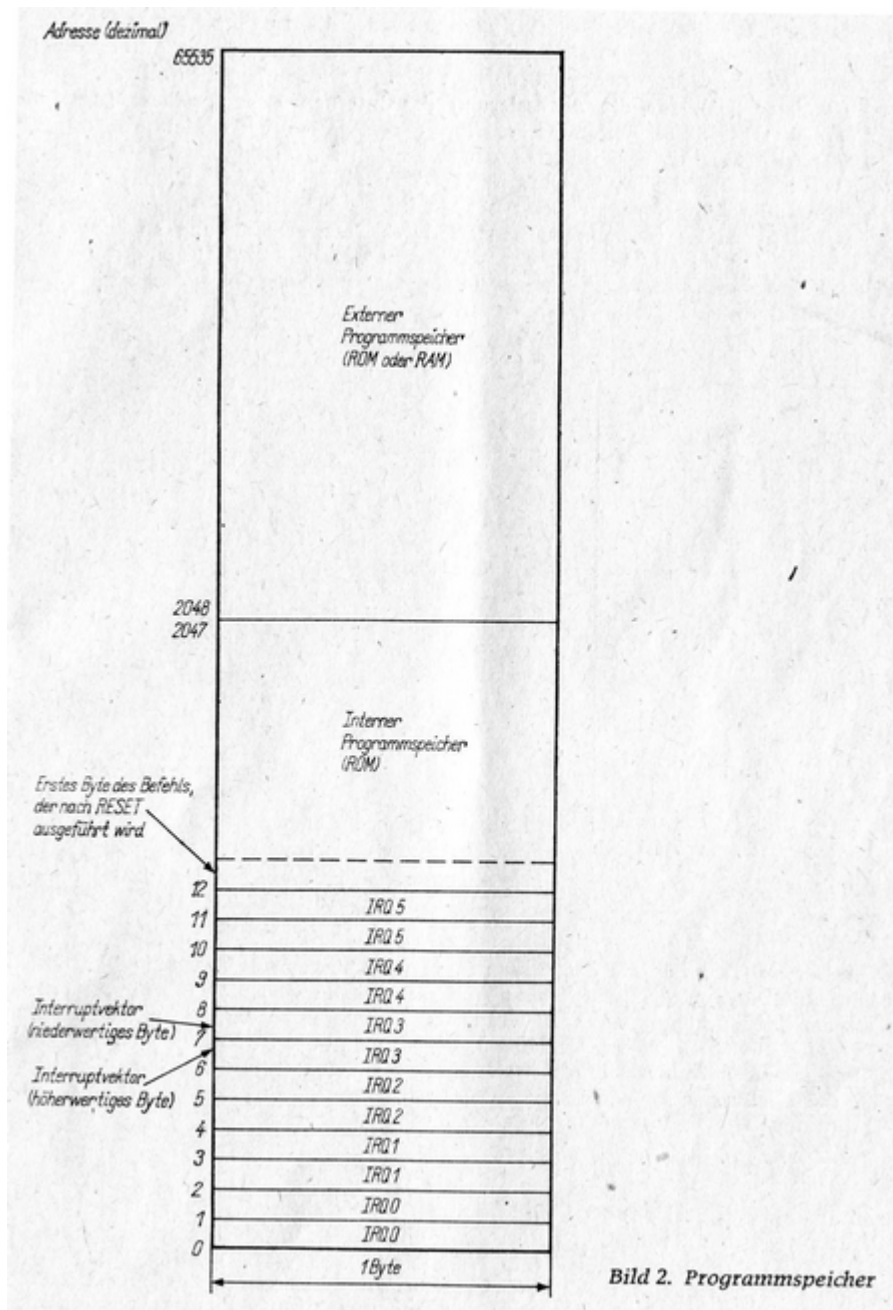
In dem EMR U88xx sind alle wesentlichen Funktionseinheiten moderner Mikrorechnerstrukturen auf einem Chip integriert:

- Taktgenerator,
- Befehlszähler,
- Steuerwerk,
- Registersatz, auch als Arithmetik-Logik-Einheit (ALU) und RAM-Bereich verwendbar,
- Programmspeicher (interner ROM),
- 2x Echtzeitzähler/Zeitgeberschaltungen,
- Interrupt/Prioritätssteuerlogik,
- 4x 8-Bit-Ports.



3.1 Programmspeicher

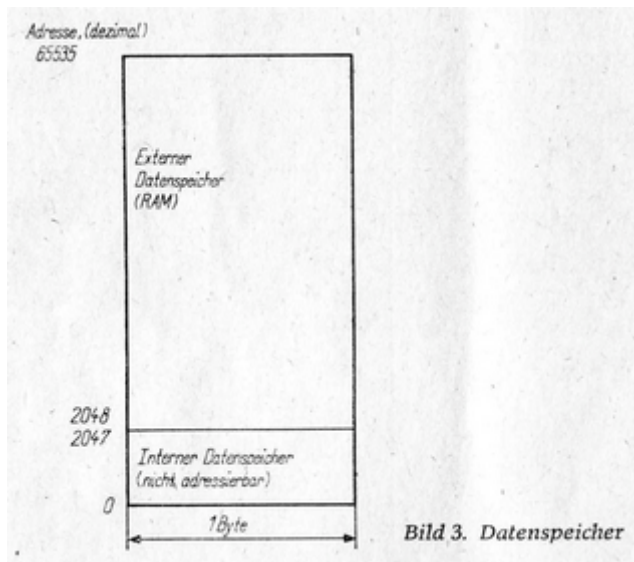
Der 16-Bit-Programmzähler kann einen $2^{16} = 64$ kByte großen Programmspeicher adressieren, welcher in einen internen und externen Teil unterschieden werden kann.



In den ersten 2 kByte (= interner Teil) liegen auch die 12 Byte der 6 Interrupt-Vektoren. Bei der Adressierung oberhalb Adresse 2047 erfolgt automatisch ein externer Speicherzugriff.

3.2 Datenspeicher

Der Datenspeicher kann maximal 62 kByte umfassen, wobei dieser entweder in den externen Programmspeicher integriert ist, oder auch separat ausgeführt werden kann. Dann wird mittels des Datenspeicher-Auswahlsignals (**/DM**) entschieden, welcher Speicher angesprochen werden soll (siehe dazu auch Bild 03).



Ausgenommen des Registersatzes ist der Datenspeichers erst ab Adresse 2048 (externer Datenspeicher) adressierbar.

3.3 Registersatz

Der 144 Byte große Registersatz untergliedert sich wie folgt:

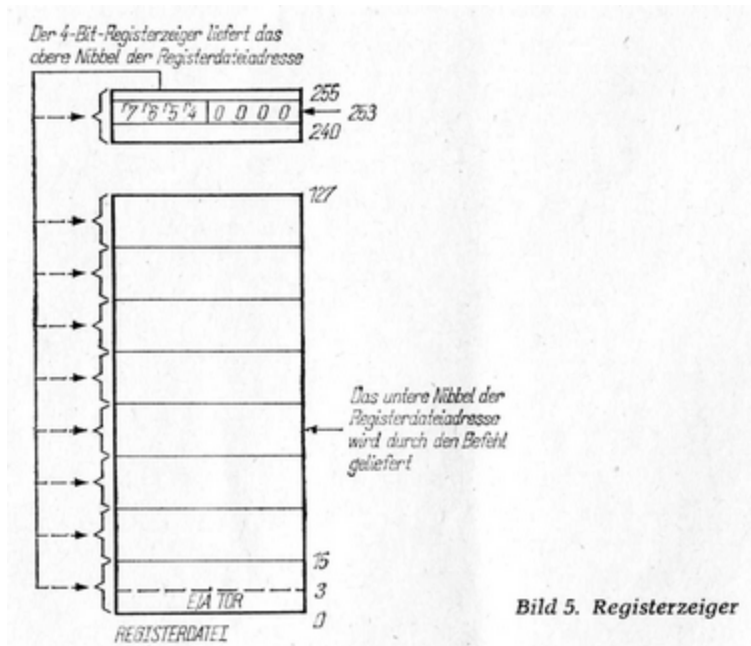
- 4x Ein-/Ausgabeport-Register (**R0 ... R3**),
- 124x Universal-Register (**R4 ... R127**),
- 16x Status- und Steuer-Register (**R240 ... R255**).

Adresse (dezimal)		Kurzzeichen
255	Stackpointer (Bit 7...0)	SPL
254	Stackpointer (Bit 15...8)	SPH
253	Registerpointer	RP
252	Flagregister	FLAGS
251	Interruptmaskenregister	IMR
250	Interruptanforderungsregister	IRQ
249	Interruptprioritätsregister	IPR
248	Betriebsart Tor0, Tor1	P01M
247	Betriebsart Tor3	P3M
246	Betriebsart Tor2	P2M
245	Vorteilerladeregister T0	PRE0
244	Zähler/Zeitgeber 0-Register	T0
243	Vorteilerladeregister T1	PRE1
242	Zähler/Zeitgeber 1-Register	T1
241	Zeitgeber-Mode-Register	TMR
240	Ein/Ausgaberegister seriell	SIO
	nicht vorhanden	
127		
	Mehrzweckregister	
4		
3	Tor3	T3
2	Tor2	T2
1	Tor1	T1
0	Tor0	T0
	1Byte	

Bild 4. Registerdatei

Bild 4. Registerdatei

Über eine 8-Bit-Adresse kann direkt oder indirekt (siehe Pkt. 5. und 8.) auf diese Register zugegriffen werden. Eine besonders effiziente Arbeitsweise bei der Registeradressierung ergibt sich durch die Aufteilung in 9 Arbeitsregistergruppen zu je 16 zusammenhängende Registern und der dann möglichen einer 4-Bit-Adressierung. Dazu wird ein als Registerzeiger ein Steuerregister verwendet (weiteres siehe auch unter Pkt. 5 bzw. 8.).



3.4 Stapelspeicher (Stack)

Der Stack kann sowohl im internen Registersatz, als auch im externen RAM definiert werden. Die Adressierung im internen Registersatz (Bereich 04H ... 0FFH) wird über das Stackzeiger-Register (**R255**) vorgenommen, hingegen bei Anwendung im externen RAM (Bereich zwischen 0800H ... 0FFFFH) das Doppelregisterpaar (**R255/R254**) notwendig ist.

3.5 Ein-/Ausgabeports

Maximal 32 Anschlüsse lassen sich in vielfältiger Weise konfigurieren, wobei je nach programmseitiger Konfiguration Ein- und Ausgänge für Adressen und Daten, digitale Ein- und Ausgänge für externe Peripherie, die sowohl bis zu 8-Bit parallel, als auch für serielle Datenströme genutzt werden können. Dazu kommen spezielle Takt-, Status- und Steuersignale, so z.B. für Handshake-Anwendungen. Alle Ein-/Ausgabeleitungen sind voll TTL-Signal-kompatibel.

3.6 Entwicklungsvariante U882x

Wie schon einleitend erwähnt, wird die Entwicklungsversion der EMR als U882x in einem 64-Pin-Quad-InLine-Gehäuse gefertigt. In der Bauelementebezeichnung ist deshalb oft ein „M“ anstelle des üblichen „D“ zu finden, z.B. U8820M.

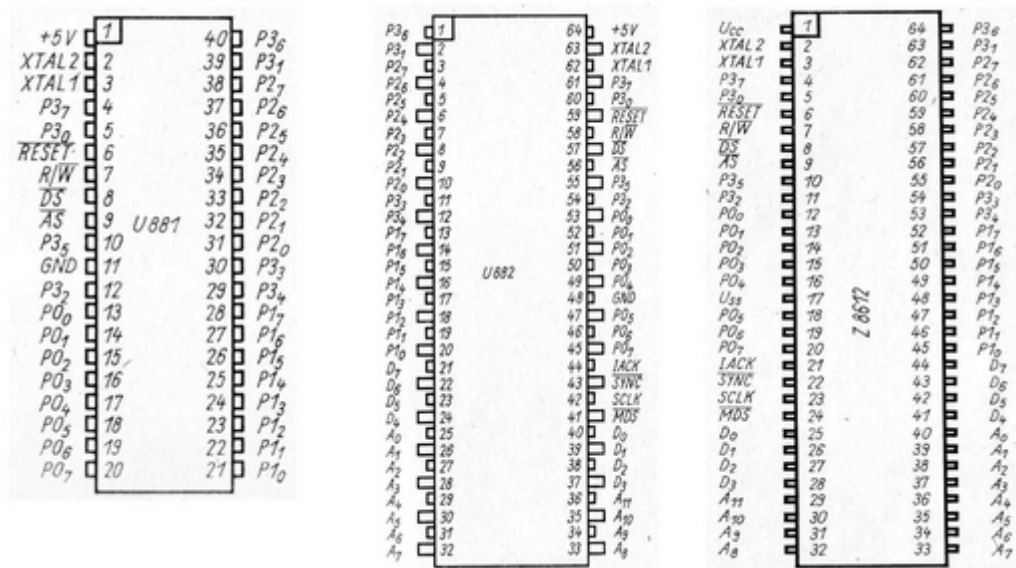


Bild 6: EMR-Anschlussbelegungen

Zu beachten ist, dass die unterschiedliche Anschlußbelegung der Entwicklungsversion U882 im Vergleich zum 40-poligen EMR U881 mit internem ROM.

Bei der Entwicklungsversion ist der interne ROM nicht vorhanden, aber die dazu notwendigen Adress-, Daten- und Steuerleitungen sind an besondere Anschluß-Pins herausgeführt. Mit einem entsprechend programmierten EPROM kann somit sehr einfach die ersten 2kByte Programmspeicher realisiert werden.

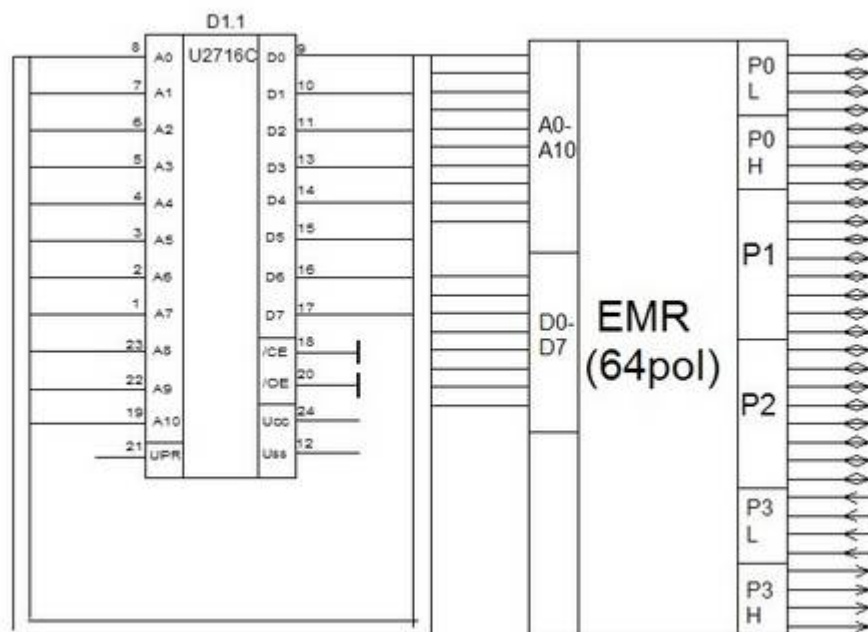
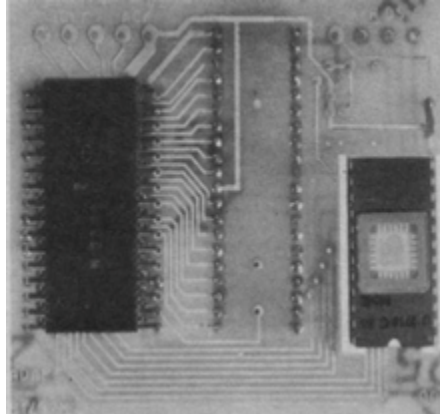
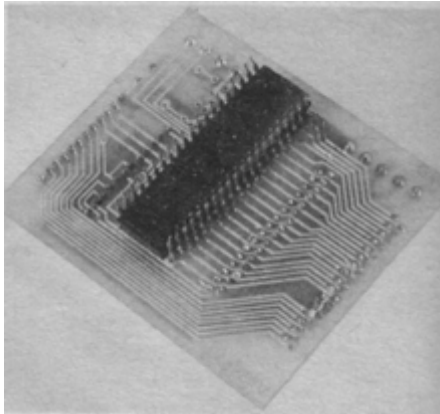


Bild 7: EPROM-Anschluß an der EMR-Entwicklerversion (64Pin-Variante)

Den labormäßigen Aufbau dieser Minimalkonfiguration zeigt Bild 8.

(hier Bild 8 einfügen)

Auch Testadapter mit EMR und zugehörigen externem EPROM wurden anstelle eines U881D verwendet.



4. Erweiterung mit externen Schaltkreisen

Prinzipiell wurden die Möglichkeiten zur externen Speichererweiterung bereits unter Pkt. 2 angegeben. Einige Besonderheiten bei den Typen U883x ... sollen hier jedoch nicht unerwähnt bleiben.

Jeder 40-polige U88xx-EMR enthält ein so genanntes Test-ROM, welches anstelle der ersten 64 Byte des internen ROM aktiviert werden kann. Dieser Test-ROM kann nun dazu benutzt werden, den internen ROM beim Start zu umgehen. Dazu ist ein spezielles „RESET“ nach Bild 9 anzulegen, bzw. mittels der Beispielschaltung nach Bild 10 zu bilden.

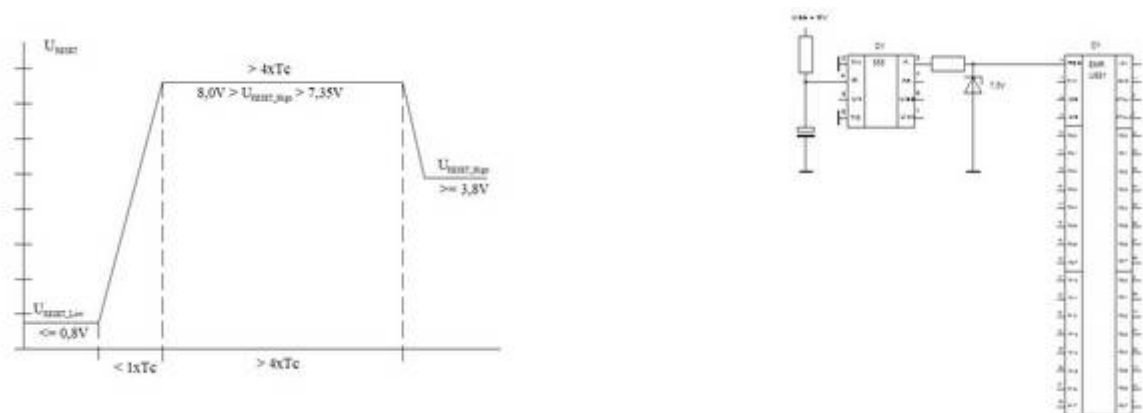


Bild 9 und Bild 10: Bootstrap-RESET

Folgendes Prozedere spielt sich dann ab:

- Initialisierung auf externen Speicher mit maximaler Größe,
 - Sprung auf die Adresse %0812, bzw. %1012 (4k-Variante U86xx)
 - Einrichten der Interruptvektoren für
- | | | | |
|------|--------|-------|-------|
| IV0: | %0800, | %1000 | - „ - |
| IV1: | %0803, | %1003 | - „ - |
| IV2: | %0806, | %1006 | - „ - |
| IV3: | %0809, | %1009 | - „ - |
| IV4: | %080C, | %100C | - „ - |
| IV5: | %080F, | %100F | - „ - |

Beim U883 mit integriertem TINY-BASIC-Interpreter ist das Verhalten etwas anders:

- IV wie beim Test-ROM,
- Test, ob eine Verbindung zwischen P3₂ und P3₅ besteht,
- wenn ja, dann zum Hardwaretest Ausgabe phasenverschobener Impulse an P0 und P1,

- wenn nicht, dann Test ob sich auf Adresse %812 eine RAM-Zelle befindet,
- wenn ja, dann Sprung auf %E000 (erster Befehl im ext. EPROM),
- wenn nicht, dann Sprung auf %812 (erster Befehl im ext. EPROM),
- es wird eine ISR für direkten Speicherzugriff initiiert, mit P3₂ als /BUSRQ und P3₅ als /BUSAk,
- im Register %7F Initialisierung für P01M.

Anmerkung: Die neue Bezeichnung für den EMR mit 4k internem ROM ist U8611DC. Davon gibt es eine „Power-Down“-Variante mit der Bezeichnung UL8611DC. Die dazugehörenden ROM-losen Varianten haben die Bezeichnung U8611DC/1 bzw. UL8611DC/1.

4.1 Anschluß eines EPROM bis 4kByte

Bild 11 zeigt noch einmal den Anschluß eines bis zu 4kByte-EPROM (siehe auch Pkt. 2.4) an einem 40-poligen EMR unter Ausnutzung des speziellen RESET, wie oben beschrieben.

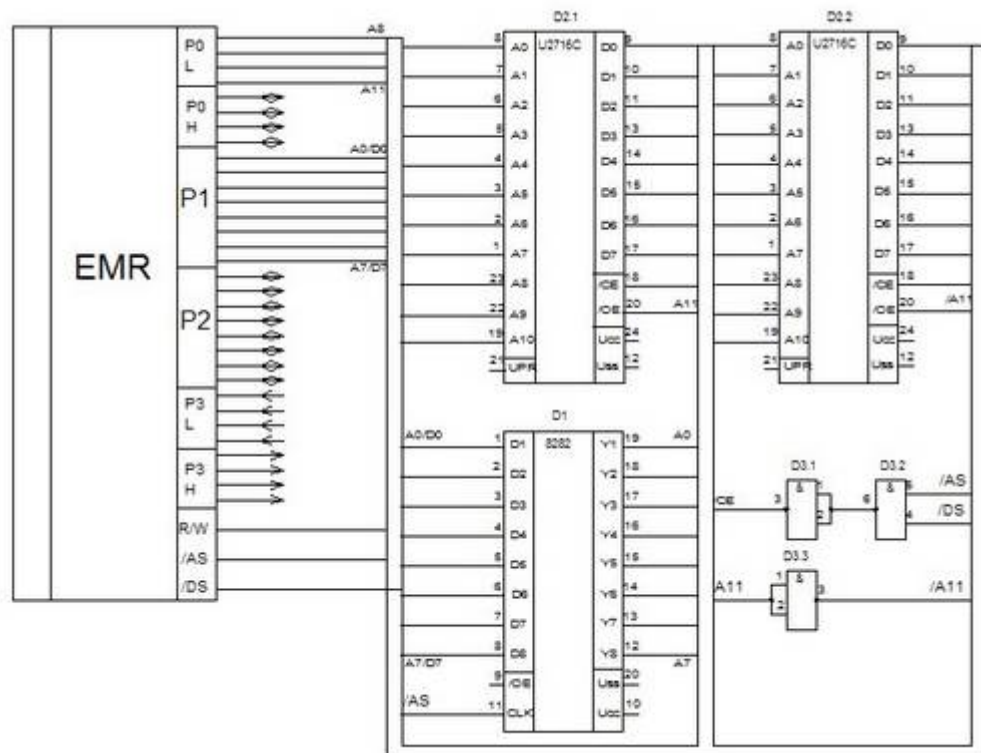


Bild 11: 4kByte-EPROM-Anschluß an EMR

Zu beachten ist der abweichende Adressbereich

EMR

0800H ... 0FFFH

1000H ... 1FFFH

EPROM

0800H ... 0FFFH

0000H ... 07FFH !!!

4.2 Anschluß eines 2kByte-RAM U6516D

Bild 12 zeigt den Anschluß eines 2kByte-RAM mit internem Adress-Latch.
(siehe auch Pkt. 2.2).

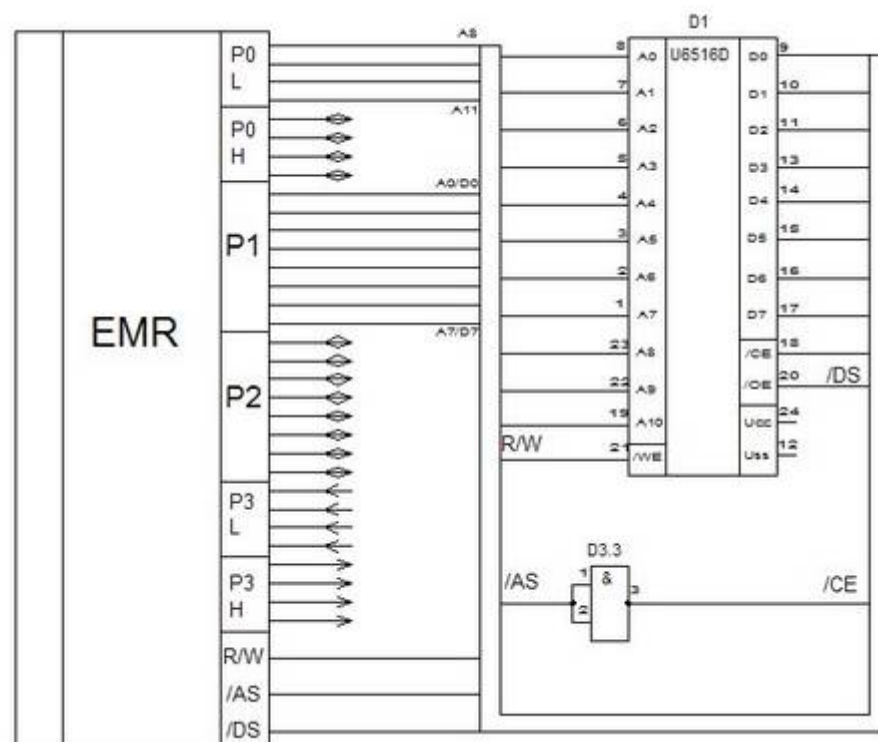


Bild 12: 2kByte-RAM mit internem Adress-Latch

4.3 Anschluß von RAMs ohne internem Adress-Latch

Sollen RAMs ohne internem Adress-Latch angeschlossen werden, wie z.B. der CMOS-RAM U214 (siehe dazu auch Pkt.2.2), so kann ganz allgemein eine Schaltung nach Bild 13 angegeben werden. Da die Adressen- und Datensignale am Port1 im Multiplex ausgegeben werden, ist eine Zwischenspeicherung mit einem 8Bit-Adress-Latch (z.B. DS8282D oder DL374D) notwendig.

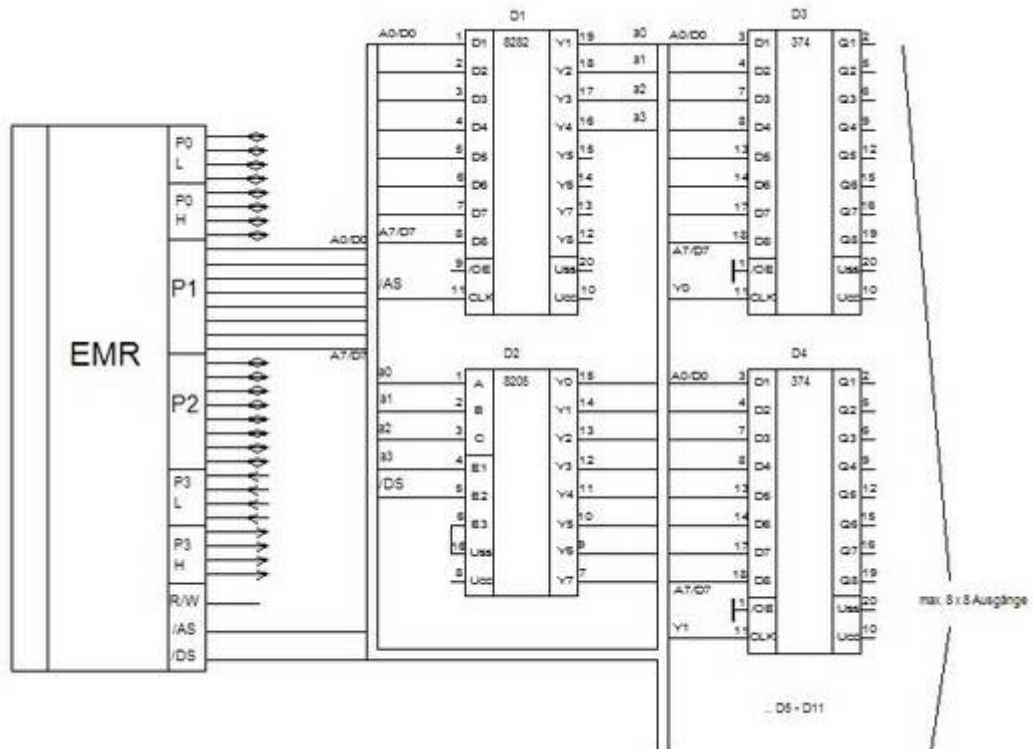


Bild 14: E/A-Erweiterung bis zu 8x8 Ausgänge

Andererseits lässt sich auch die Anzahl der möglichen Eingangsanschlüsse durch den Einsatz von Multiplexer-Schaltungen vervielfältigen. Die Abfrage der einzelnen Eingänge muß dann natürlich softwaremäßig unterstützt werden.

Die an den Eingangs-Latch DS8282D anliegenden Eingangs-Informationen werden je nach angewählter Adresse auf den gemultiplexten Adress-Datenbus gelegt und können somit über einen einfachen Ladebefehl in den EMR übernommen werden. Die Selektierung erfolgt in diesem Fall über das /OE-Signal. Da CLK ständig auf High-Potential liegt, werden die Eingangsinformationen direkt durchgeschoben.

Bild 15 zeigt ein entsprechendes Schaltbeispiel.

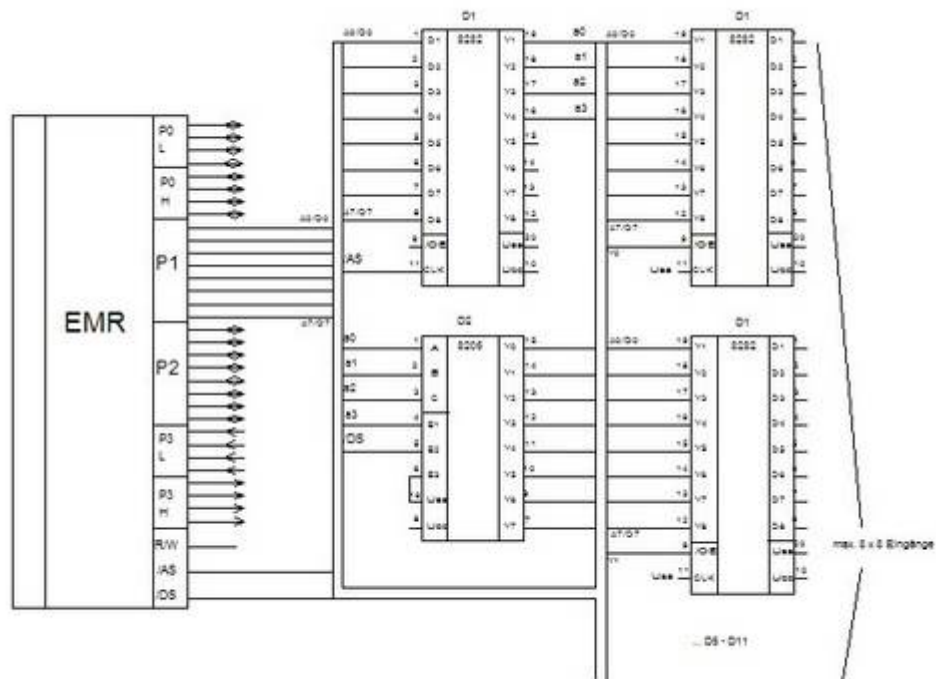


Bild 15: E/A-Erweiterung bis zu 8x8 Eingänge

4.6 Analogsignalverarbeitung

Sollen mit dem EMR U88xx analoge Signale verarbeitet werden, muß vorher eine Umwandlung in eine digitale Form erfolgen. Dazu stehen die verschiedensten A/D-Wandler als integrierte Bauelemente zur Verfügung. Als Kriterium für die Auswahl steht in erster Linie die gewünschte Auflösung, d.h. in wie viel diskrete Schritte der Eingangsspannungsbereich quantisiert werden soll (kann). Für die Umwandlung des analogen Eingangssignal in eine digitale Form wird ein bestimmter Zeitrahmen benötigt. Dieser Parameter ist ein weiteres Kriterium für die Auswahl eines geeigneten A/D-Wandlers. Vielfach ist das zu verarbeitende analoge Eingangssignal im Verarbeitungszeitraum nicht konstant, so dass durch weitere Schaltungsmaßnahmen für die erforderliche Konstanz gesorgt werden muß. Dazu werden so genannte „Sample-Hold“-Schaltungen verwendet. Auch diese gibt es als integrierte Bauelemente.

4.6.1 ADU-Interface mit C570D

Bild 16 zeigt ein entsprechendes Schaltbeispiel für einen 8-Bit-Analogsignaleingang am EMR. Der hier verwendete C570D ist ein schneller 8-Bit-A/D-Wandler mit 15...40µs Umsetzzeit, der aus den o.g. Gründen mit einem S/H-Schaltkreis, z.B. dem SU-Typ

KR1100SK2 oder dem CSSR-Typ MAC198, ergänzt worden ist.

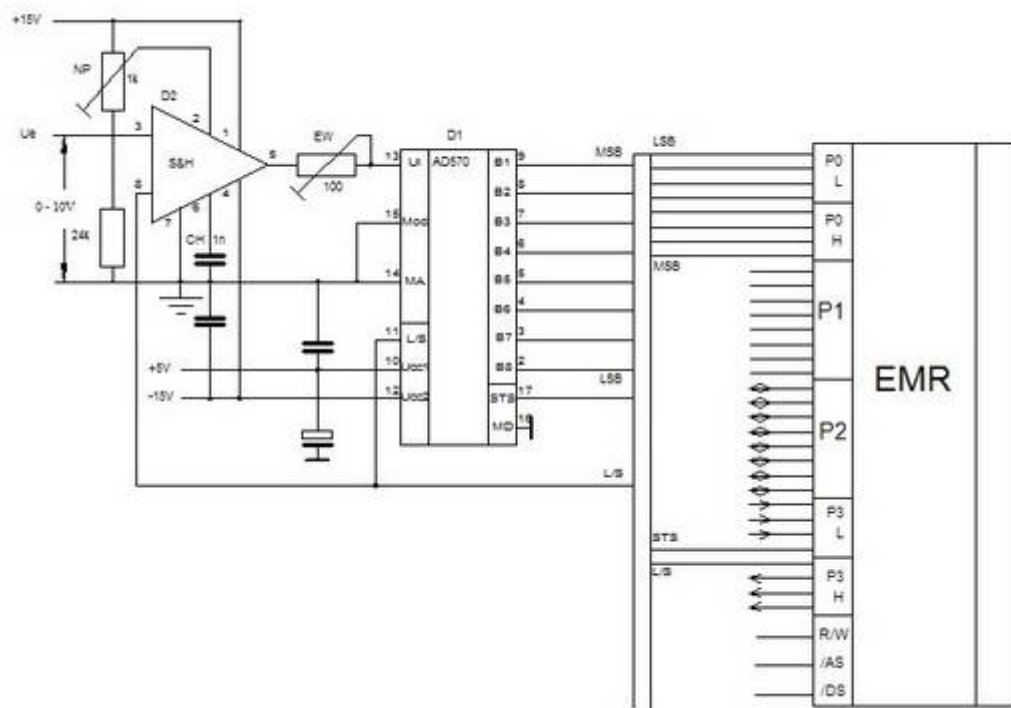


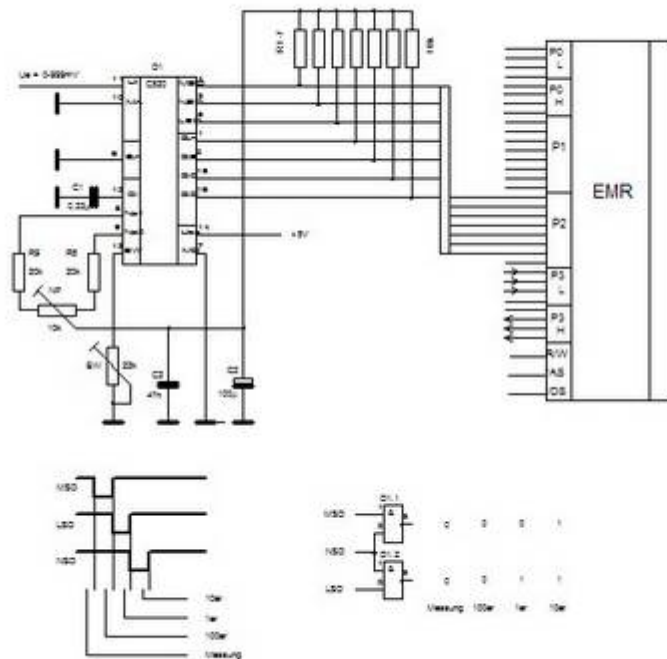
Bild 16: Anschluß AD-Wandler mit S&H-Schaltkreis

In die Steuerleitung zum A/D-Wandler, bzw. S&H-Schaltung kann auch noch ein Monoflop eingebaut werden, so dass dann die Steuerung auch mit dem Handshake-Signal des EMR möglich ist. Mit den beiden Einstellreglern „Gain“ (EW) und „Zero“ (NP) ist eine Kalibrierung der Schaltung möglich.

4.6.2 ADU-Interface mit C520D

Bei einer anderen Variante kommt der $3\frac{1}{2}$ -Digit-A/D-Wandler C520D zum Einsatz. Vorteilhaft bei diesem Schaltkreis ist insbesondere, dass nur eine +5V-Betriebsspannung benötigt wird. Allerdings sind die Interface-Bedingungen schon etwas komplizierter. Zu nächst benötigen alle Ausgänge des C520D wegen ihrer Open-Collektor-Eigenschaften entsprechende Pull-Up-Widerstände. Des weiteren sind die Ausgangssignale wegen der freilaufenden internen Taktung im C520D mit mehr oder wenig starkem „Jitter“ (Phasen-Schwankungen) versehen, so dass es ohne programmtechnisch realisierte „Entprellung“ (siehe dazu Pkt. 15) zu stark veränderlichen Ergebnissen kommen kann. Trotz des recht umfangreichen Programms wird der EMR auch bei ständigem Aufruf der

Bild 17 zeigt das Prinzipschaltbild mit den zugehörigen Taktschema, sowie einer schaltungstechnischen Variante zur Reduzierung der notwendigen Eingänge am EMR.



Ist die Anzahl der zur Ansteuerung des C520 notwendigen Eingänge am EMR beschränkt, kann, wie in Bild 17 angegeben, durch Einsatz zweier TTL-Gatter die notwendigen Eingänge für MSD/NSD/LSD auf zwei reduziert werden.

Als drittes Beispiel soll noch der prinzipielle Anschluß eines Präzisions-A/D-Wandlers C500D mit 14 Bit (C501D mit 11 Bit) Auflösung angegeben werden.

Die IS C500D benötigt eine Betriebsspannung von $\pm 15V$, wobei der Komparatorausgang mit einem Pull-Up-Widerstand auf $+U_B$ gezogen werden muß. Da nur drei Steuerleitungen notwendig sind, ergibt sich ein sehr einfaches Hardware-Interface, was allerdings dann eine umfangreichere Software bedeutet, d.h. die sonst im Digital-Prozessor C502D implementierte Funktionalität muß softwaremäßig mit dem EMR nachgebildet werden.

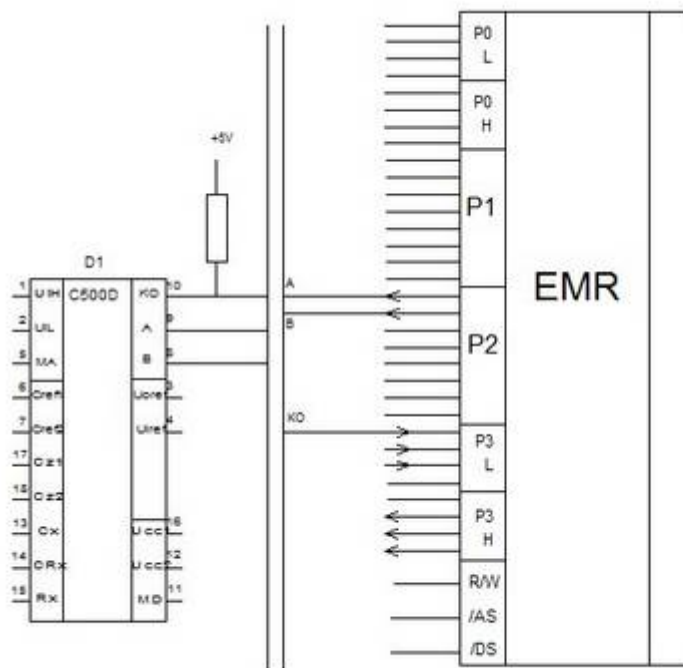


Bild 18: Interfaceprinzip des ADU C500D

Die Steuerung des C500D ergibt sich nach folgendem Schema:

	<u>auto-zero</u>	<u>Aufintegration</u>	<u>Abintegration</u>
	200ms	100ms	100ms
A:	0	1	1 0
B:	0	1	0 1
			---> 0(-)
Komp:			-----> 1(+)

Bei bis zu 14 Bit Auflösung werden immer zwei Messungen benötigt.

Die softwaremäßige Unterstützung erfolgt im Pollingbetrieb, kann aber auch als ISR implementiert werden. Weitere Details können den einschlägigen Applikationsschriften des HFO zu entnommen werden.

Allgemein seien hier noch ein paar Bemerkungen zu den Vorteilen periodendauer- oder frequenzproportionaler Signaldarstellung angegeben:

- die Übertragung erfolgt ausschließlich in digitalen Signalpegeln, d.h. störfest,
- die Digital-Wandlung ist mit systemeigenen mitteln möglich (Timer, Softwarezähler),
- nur ein Eingangs-Pin je Meßgröße erforderlich,

- flexibel, nicht an Wandler-Parameter gebunden.

Die Anwendungsgrenze ist wie folgt:

$$a * t_m = 1/f_{\max}$$

Häufig lohnt sich auch eine U/f-Wandlung.

4.7 Analogwert-Ausgabe

Bei hohen Ansprüchen an zeitliche Dynamik ist der Einsatz von integrierten D/A-Wandler notwendig, beispielsweise die Typen C560D oder DAC32.

In der Automatisierungs-Praxis werden aber auch vielfach die Methoden der Pulsdauer- bzw. Pulsanzahl-Modulation angewendet, wobei letztere für EMR besser geeignet ist.

Ein entsprechendes Schaltbeispiel zeigt Bild 19.

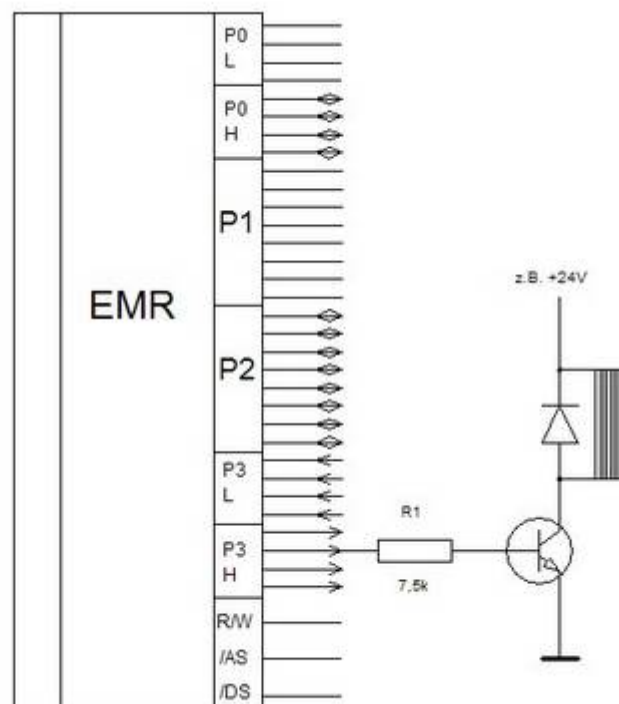


Bild 19: Einfache Analogwertausgabe

Hier wird ein Analogwert zur impulsmäßigen Ansteuerung eines Elektromagneten nachgebildet. Softwaremäßig wird das durch ständige Addition, z.B. in einer regelmäßig aufgerufenen ISR realisiert. Dabei hat sich in der Praxis die Verknüpfung mit einem Echtzeitgeber als günstig erwiesen.

Programmbeispiel der ISR:

in Register %9 ist der Ausgabewert, z.B. 1 0 1 0 0 0 0 0 = 62,5% „Ein“

Register %A ist das Restregister

ISR: ADD %A, %9 ; Rest + Ausgabewert

JRC, IS1 ; bei Übertrag, d.h. $|\%A| + |\%9| > 255$

AND 3, %%DF ; $P3_5 : = 0$

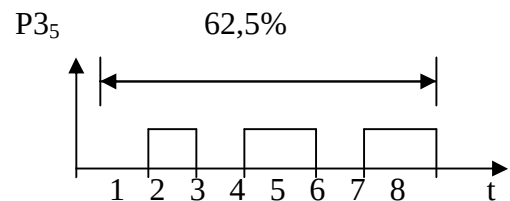
IRET

IS1: OR 3, %%20 ; $P3_5 : = 1$

IRET

(Diese ISR hat nur 16µs Bearbeitungszeit!)

Nr	CY	%A
		0 0 0 0 0 0 0 0
1	0	1 0 1 0 0 0 0 0
2	1	0 1 0 0 0 0 0 0
3	0	1 1 1 0 0 0 0 0
4	1	1 0 0 0 0 0 0 0
5	1	0 0 1 0 0 0 0 0
6	0	1 1 0 0 0 0 0 0
7	1	0 1 1 0 0 0 0 0
8	1	0 0 0 0 0 0 0 0
		
		



5. Programmierung

Der zur Programmierung des U88xx-EMR zur Verfügung stehende Befehlssatz mit 129 Befehlen ist stark an den des U880D angelehnt, verfügt er doch z.T. über ebensolche effizienten Blocktransfer-Befehle. Wegen der Unterschiede in der Organisation der Register ist der Syntax jedoch völlig anders.

Zur Inbetriebsetzung eines EMR sind:

- a) grundlegende Registereinstellungen zur Konfiguration des EMR vorzunehmen und
- b) das eigentliche Anwenderprogramm entsprechend der Aufgabenstellung im Programmspeicher bereitzustellen. Das Programm beginnt nach einem RESET (oder nach dem Einschalten der Betriebsspannung immer bei der Programm-Adresse *0CH*. Dort sollte als allererstes der Befehl **DI** zum Interrupt-Verbot stehen, damit die nachfolgende Initialisierung/Konfiguration des EMR ohne solche Einflüsse ablaufen kann. Danach - und wenn eine ISR vorgesehen ist – kann der Interrupt durch den Befehl **EI** wieder freigegeben werden.

5.1 Befehlsarten

Die zur Verfügung stehenden Befehle des U88xx können nach verschiedenen Kriterien entsprechend der definierten Befehlsformate unterschieden werden:

1-Byte-Befehle, darunter fallen auch die Befehle ohne Operanden:

OPC	CCF, DI, EI, IRET, NOP, RCF, RET, SCF
-----	--

aber auch Befehle, wo das Ziel (Destination) im Befehl enthalten ist:

dst	OPC	INCr
-----	-----	-------------

2-Byte-Befehle

OPC	dst	src	ADC, ADD, AND, CP, LD, LDE, LDEI, LDC, LDCI, OR, SBC, SUB, TMC, TM, XOR
-----	-----	-----	--

oder

OPC	n	SRP
-----	---	------------

oder

OPC	dst/src	CLR, COM, DA, DEC, DECW, INC,
-----	---------	--------------------------------------

INCW, POP, PUSH, RL, RLC, RR,
RRC, SRA, SWAP, JP IRR,
CALL IRR

oder

dst/scr	OPC
---------	-----

n/scr/dst

LD

oder

r, c	OPC
------	-----

e

DJNZ, JR

3-Byte-Befehle

OPC

scr	*
-----	---

dst	*
-----	---

**ADC, ADD, AND, CP, LD, OR,
SBC, SUB, TCM, TM, XOR**

oder

OPC

dst

n

**ADC, ADD, AND, CP, LD, OR,
SBC, SUB, TCM, TM, XOR**

oder

OPC

adr H

adr L

CALL, JP

oder

OPC

dst/scr	X
---------	---

adr

LD r,X ; LD X,r

*) bei Arbeitsregister-Adressierung gilt:

1110	dst/scr
------	---------

5.2 Adressierungsarten

Beim U88xx-EMR existieren umfangreiche Adressierungsmöglichkeiten, angefangen von der direkten Registeradressierung bis hin zur hochkomplexen Indexadressierung.

Folgende Operanden-Adressierung ist möglich:

R (Register)	Operand ist der Inhalt des durch 8-Bit adressierten Registers.
r (Arbeitsregister)	Operand ist der Inhalt des Registers, dessen Adresse sich aus Dem Registerpointers (höhere 4 Bit) und der im Befehl gegebenen 4 Bit zusammensetzt.
RR (Doppelregister)	Operand ist der Inhalt zweier aufeinanderfolgender Register.
IR (indirekt Register)	Operand ist der Inhalt des Registers, dessen Adresse im angegebenen Register steht.
Ir (indirekt Arbeitsregister)	Operand ist der Inhalt eines Registers, dessen Adresse im angegebenen Arbeitsregister steht
IRR (indirekt Doppelregister)	Operand ist der Inhalt einer 8-Bit-Speicherzelle, deren Adresse im angegeben Doppelregister steht.
Irr (indirekt Doppel-Arbeitsregister) wie IRR	
n (Direktooperand)	Operand ist die angegebene 8-Bit-Zahl
e (Sprungdistanz)	Gegeben ist die Adressdifferenz zwischen Sprungziel und Absprungstelle (8-Bit-Zweierkomplement)
adr (Zieladresse)	Gegeben ist die absolute 16-Bit-Zieladresse
X (indizierte Adressierung)	Operand ist der Inhalt des Registers, dessen Adresse die Summe eines Arbeitsregisterinhalts und einer 8-Bit-Konstanten ist.

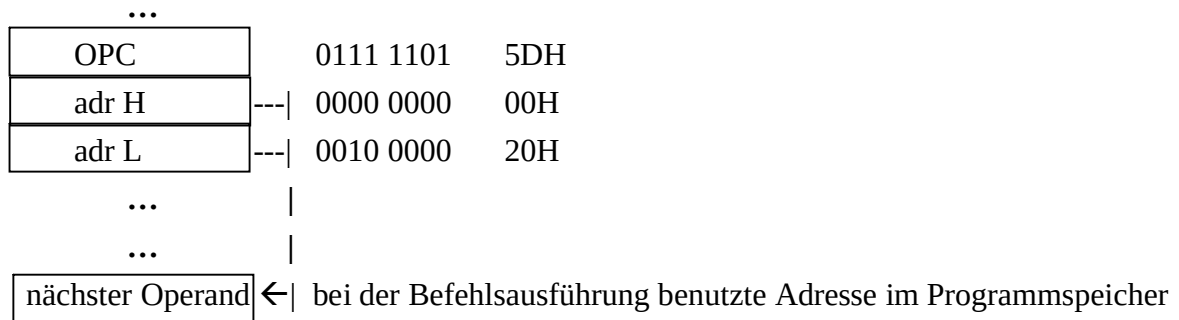
5.2.1 Direkte Adressierung

Bei der direkten Adressierung enthält der Operanden- und Adressteil die direkte Adresse des Operanden. Diese Adressierungsart wird nur bei Sprung- und Aufruf-Befehlen angewandt.

Beispiele:

JP C, %20	Sprung zur direkten 16-Bit-Adresse 0020H, wenn die Sprungbedingung – Carry-Bit – gesetzt war
Call Time	Der Programmablauf wird an der UP-Marke „Time“ fortgesetzt

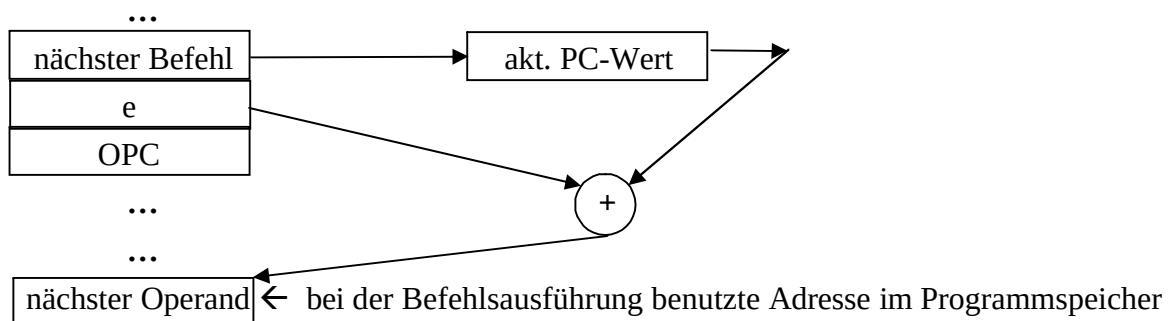
Programmspeicher



5.2.2 Relative Adressierung

Zur relativen Adressierung werden im Operanden- und Adressteil nur Adressdistanzen angegeben. Mit der entsprechenden Basisadresse ergibt sich dann durch diese verkürzte Art der Adressangaben die effektive Adresse. Die Distanzangabe ist vorzeichenbehaftet, so dass diese sowohl in Vorwärts-, als auch in Rückwärtsrichtung möglich ist. Durch Anwendung des Zweierkomplements bei der Distanzangabe sind dabei Distanzen von -128 bis +127 möglich.

Programmspeicher



Diese Adressierungsart wird nur bei folgenden zwei Befehlen angewandt:

JR cc, \$+e Addiere den konstanten Wert **e** zum aktuellen Inhalt des Programm-Zählers **\$** und springe zu dem berechneten neuen Programmzählerstand, wenn die Bedingung **cc** erfüllt ist.
In der Regel wird der konstante Wert **\$ + e** in Form einer Marke angegeben, welche dann bei der Assemblierung berechnet wird.

DJNZ r, \$+e Dekrementiere den Inhalt des Arbeitsregisters **r** solange, bis dessen Wert „0“ erreicht hat und springe dann zu dem berechneten neuen

Programmzählerstand.

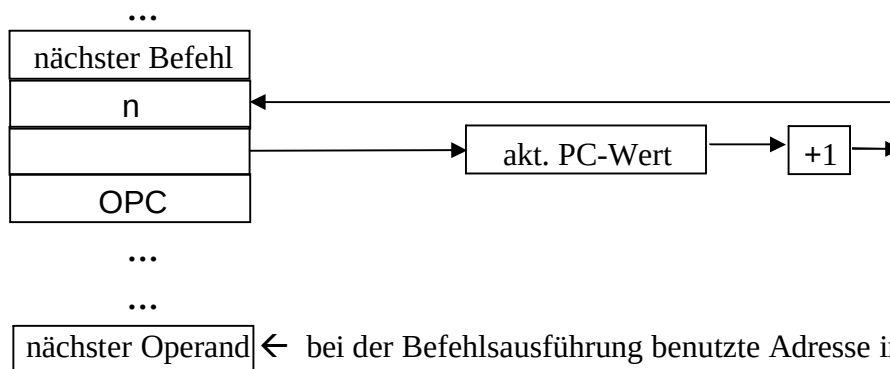
Auch hier wird in der Regel mit Marken gearbeitet.

5.2.3 Unmittelbare Adressierung

Die unmittelbare Adressierung wird häufig zum Laden von Registern mit Anfangswerten benutzt. Hier ist im Operanden direkt die Adresse des Befehls enthalten. Dadurch werden Programmumfang und Ausführungszeit reduziert.

In der Assembler-Notation (siehe auch Pkt. 6) steht hier vor dem Direktoperanden **n** das Symbol **#**. Dabei kann sowohl direkt ein 8-Bit-Wert (0 ... 255) eingesetzt werden, als auch ein Symbol, welches auf eine entsprechende Konstante weist. Es ist auch möglich, dafür eine Variable vorzusehen, nur muß dann sichergestellt sein, dass der Zahlenbereich von -128 bis +127 bzw. 0 bis 255 nicht überschritten wird.

Programmspeicher



Für eine effektive Behandlung von 2-Byte-Byteadressen sind hierbei spezielle High- (**HI**) und Low- (**LO**) Operatoren vorhanden. Dann wird der Direktwert-Operator **#** überflüssig.

Beispiele:

LD 90, #%12

Lade Register 90 mit dem Hex-Wert 12

LD COUNTER, #DATA

lade das mit „COUNTER“ bezeichnete Register mit der Adresse der Variablen „DATA“

LD R8, HI FELD

Lade das Arbeitsregister 8 des aktuellen Arbeitsregistersatzes mit dem höherwertigen Teil (Bit 8 bis 15) der durch „FELD“ bezeichnete 16-Bit-Konstante

LD R9, LO FELD

Lade das Arbeitsregister 9 des aktuellen Arbeitsregistersatzes mit dem niederwertigen Teil (Bit 0 bis 7) der durch „FELD“ bezeichnete 16-Bit-Konstante

5.2.4 Registeradressierung (8 Bit)

Diese Adressierungsart wird oft zum Umladen von Registern verwendet.

Beispiele:

LD 77, 20	Lade das Register 77 mit dem Inhalt des Registers 20
ADD 121, 25	Addiere den Inhalt des Registers 25 zum Inhalt des Registers 121 und lege das Ergebnis dort ab.
CLR 10	Lösche den Inhalt von Register 10, d.h. setze alle Bits auf „0“

5.2.5 Registerpaar-Adressierung

Zur Darstellung von 16-Bit-Werten - z.B. um eine Speicheradresse zu definieren – besteht die Möglichkeit zwei aufeinanderfolgende Register als Paare zu behandeln. Voraussetzung ist allerdings, dass das Registerpaar immer geradzahlig beginnt, also z.B. 0, 2, 4, usw, bzw. 240, 242, 244 usw. Natürlich kann das Registerpaar auch mit dem Variablennamen des geradzahligen Registers bezeichnet werden.

Die Registerpaar-Adressierung findet nur bei zwei Befehlen Anwendung:

DECW, INCW

Beispiele:

DECW 40	Dekrementiere den Inhalt des Registerpaares, das durch die Register 40 und 41 gebildet wird.
INCW 30	Inkrementiere den Inhalt des Registerpaares, das durch die Register 30 und 31 gebildet wird.

5.2.6 Arbeitsregister-Adressierung (4 Bit)

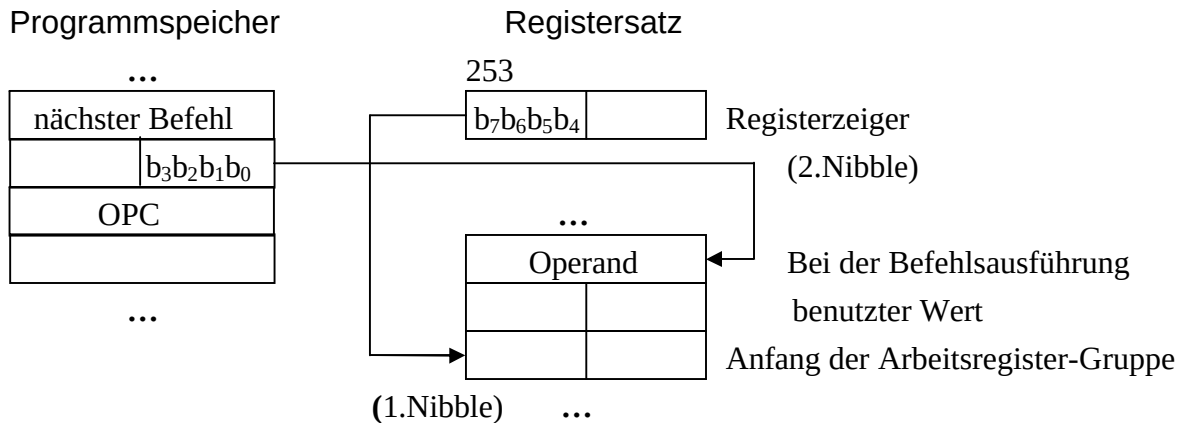
Eine besonders effiziente Adressierungsart beim U88xx-EMR ist die Adressierung unter Verwendung eines Registerzeigers. Dadurch reduziert sich die notwendige Adress-Größe auf 4 Bit, was wiederum eine Verkürzung der Befehlslänge und damit der Ausführungszeit zur Folge hat. Bei dieser Adressierungsart wird der gesamte Registersatz in 9 Arbeitsregister-Gruppen aufgeteilt, die je 16 aufeinanderfolgende Register enthalten. Der Registerzeiger im Steuerregister 253 adressiert dabei die Startadresse (= niedrigste Adresse) der aktiven Registergruppe und bildet damit den höherwertigen Teil (H-Nibble) der Adresse. Dafür ist im Befehlssatz der spezielle Befehl **SRP** vorgesehen.

Die vollständige Adresse wird dann durch Verkettung mit dem niederwertigen Teil (L-Nibble), d.h. der jeweiligen Arbeitsregister-Adresse gebildet.

Eine programmabhängige Änderung des Registerzeigers ermöglicht eine dynamische Variation des Arbeitsregistersatzes.

Die Anwendung der Arbeitsregister-Adressierung wird durch ein „**R**“ gekennzeichnet,

gefolgt von einer Zahl im Bereich von **0** bis **15**, entsprechend dem gewünschten Arbeitsregister.



Beispiele:

SRP #0010

Setze den Registerzeiger auf den Hex-Wert 10, d.h. lade das Steuerregister 253 mit 10H und damit mit der Anfangsadresse der 2. Arbeitsregister-Gruppe.

LD R1, 30

Lade den Inhalt des Registers 30 in das Arbeitsregister 1 der aktuellen Arbeitsregistergruppe 2.

ADD R5, R9

Addiere den Inhalt von Arbeitsregister 9 (Register 25) der 2. Arbeitsregistergruppe in das Arbeitsregister 5 (Register 21) dieser Gruppe und lege das Ergebnis dort ab.

5.2.7 Indirekte Registeradressierung

Die bei allen Lade-, Arithmetik-, logik-, Rotations- und Verschiebepfehlen mögliche Anwendung indirekter Adressierung erscheint auf den ersten Blick recht kompliziert. Im Hinblick auf die Möglichkeit, dass man damit im Programmablauf die konkrete, zu verwendende Adresse berechnen kann, eröffnet dem Programmierer ein weites Feld seiner Kreativität. Besonders bei der Bearbeitung von Tabellen ist diese Adressierungsart sehr effizient.

Der Operanden- und Adressteil besteht aus dem Symbol zur indirekten Adressierung @, gefolgt von einem Ausdruck in Form einer Register-, Arbeitsregister- oder Variablenbezeichnung.

Beispiele:

SRP #00

Setze den Registerzeiger auf die 1. Gruppe

CLR @ 10

setze den Inhalt des von Register 10 adressierten

LD R12, @ 10

Registers zurück

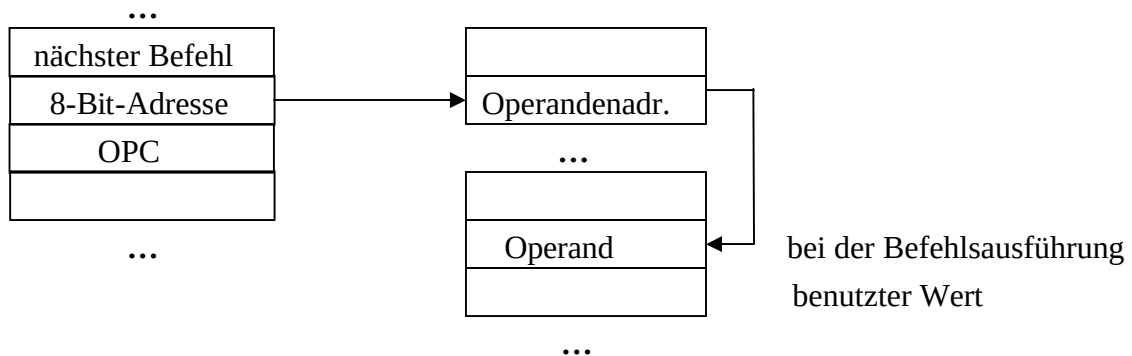
Lade den Inhalt des durch Register 10 adressierten Registers in das Register 12

LD @ RP, #%30

lade den Hex-Wert 30 an die Stelle, die durch RP (Registerpointer 253) adressiert wird

Programmspeicher

Registersatz



Die indirekte Adressierung wird auch zur Organisation des internen und externen Speicherverkehrs eingesetzt. Die vollständige 16-Bit-Speicheradresse ergibt sich dabei indirekt über das Register- bzw. Arbeitsregisterpaar mit der Adresse des ausgewählten Speicherplatzes. Dies gilt natürlich wiederum nur für geradzahlige Registeradresspaare, d.h. z.B. 0, 2, 4 ...usw. im Bereich des vorhandenen Registersatzes. Es ist auch möglich eine Variable einzusetzen, solange diese nur geradzahlige Werte im gültigen Bereich annehmen kann.

Beispiele:

JP @ 20

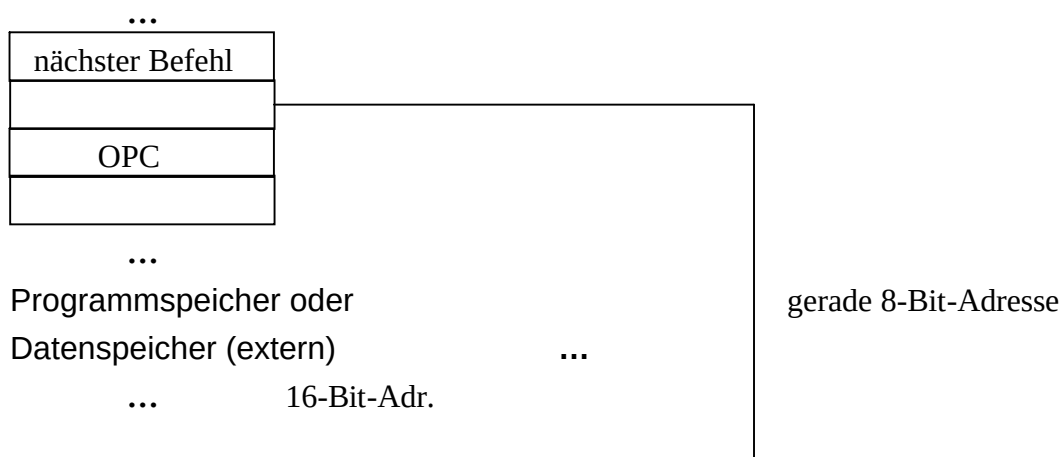
Springe zur Programmstelle, die durch die Register 20 und 21 adressiert wird

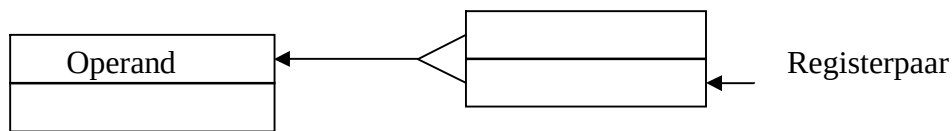
CALL @ 16

setze das Programm an der Stelle fort, die durch die Register 16 und 17 adressiert wird

Programmspeicher

Registersatz





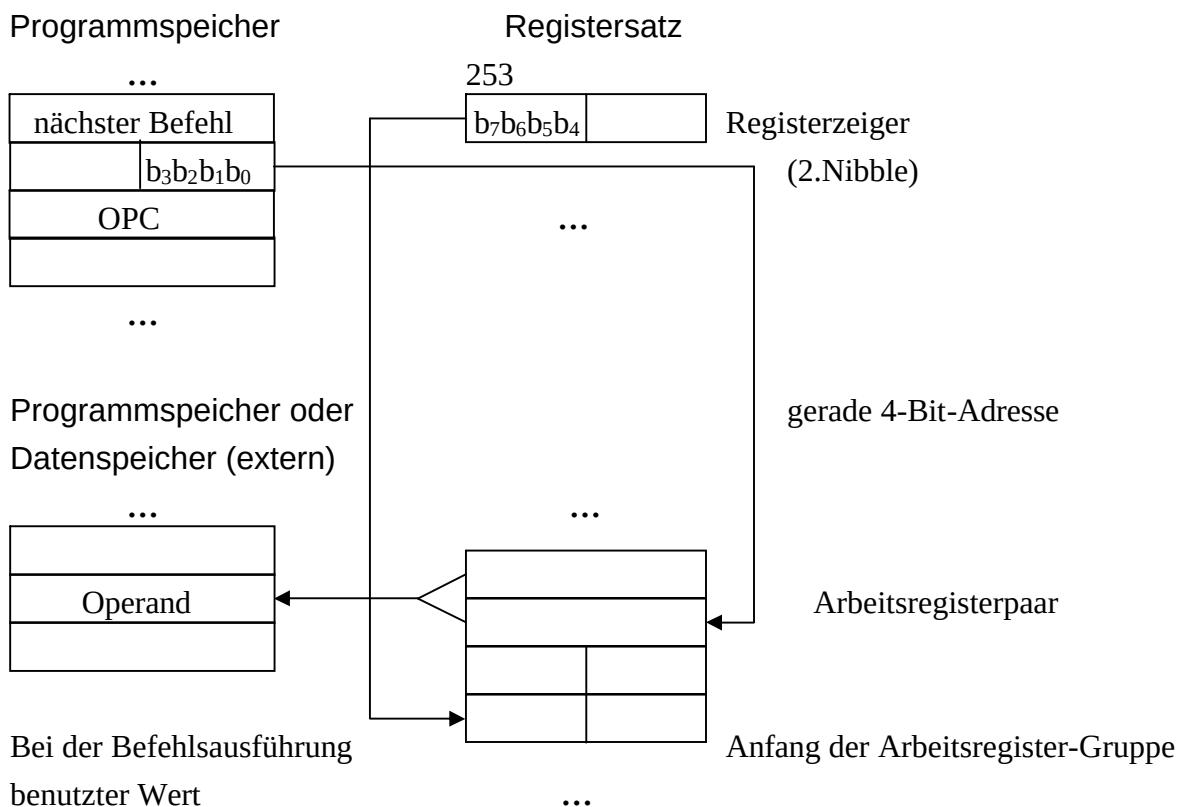
Es ist auch die Anwendung von Arbeitsregisterpaaren möglich. Dazu wird außer dem indirekten Adresspräfix @ die Buchstaben **RR** vor die zwei aufeinanderfolgenden Register innerhalb der aktuellen Arbeitsregistergruppe angegeben. Auch hier sind wiederum nur geradzahlige Registerzahlen zulässig.

Beispiele:

SRP #00

LDEI @ R3, @ RR4

Setze den Registerzeiger auf die 1. Registergruppe
Lade die Daten vom externen Speicher, adressiert durch den Inhalt der Arbeitsregister 4/5 der 1. Gruppe (Register 4/5) an die Stelle, die durch das Arbeitsregister 3 (Register 3) der gleichen Gruppe adressiert wird.



5.2.8 Indexadressierung

Die Index-Adressierung ist die komplizierteste Adressierungsart, wird aber beim U88xx-EMR auch nur von Ladebefehlen unterstützt.

Ein Arbeitsregister - auch Indexregister genannt - dessen Inhalt (Displacement) zu einer angegebenen 8-Bit-Registeradresse addiert wird weist auf eine Speicherstelle, deren Inhalt vom Befehl als Komplettierung zum Argument verwendet wird. Der das Displacement enthaltene Teil des Operanden (Index) wird bei der Assembler-Darstellung in Klammern gesetzt.

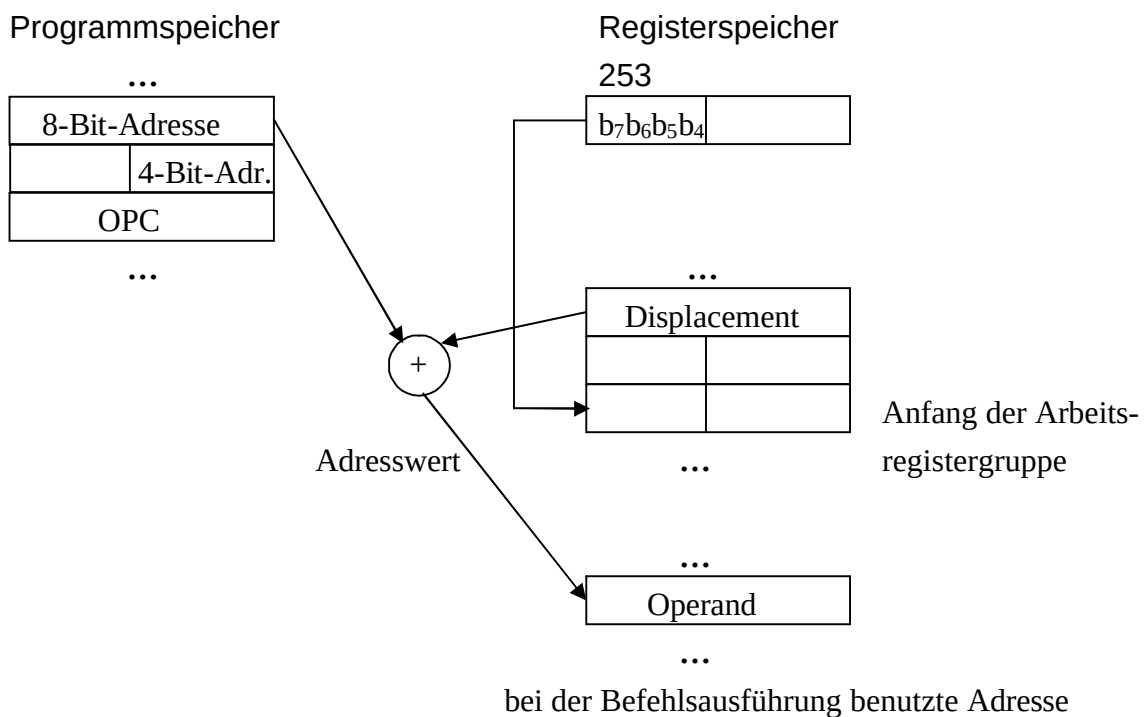
Beispiel:

SRP #00

LD 242(R2), R12

Registerzeiger mit der ersten Gruppe laden

Lade in das Register 242 + Inhalt des Arbeitsregisters 2 der 1.Arbeitsregistergruppe den Wert des Arbeitsregisters 12 der gleichen Gruppe



6. Assemblernotation, Datenformate

Grundsätzlich reicht der bei einem 8-Bit-EMR verwendbare binäre Zahlenbereich positiver Zahlen von 0H bis 0FFH. Um nun auch negative Zahlen darstellen zu können, wird vereinbart, dass dazu das Bit 7 als Kennzeichner verwendet wird:

$b_7 = 0 \rightarrow$ positive Zahl

$b_7 = 1 \rightarrow$ negatives Vorzeichen

Um eine solche Zahl mit negativem Vorzeichen bilden zu können, wird das so genannte „Zweier-Komplement“ gebildet, d.h. die Zahl wird bitweise negiert und +1 addiert.

Eine weiteres gebräuchliches Datenformat ist das BCD-Format (gepackt):

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	
80	40	20	10	8	4	2	1	Wertigkeit

Zum Umsetzen eines Assembler-Quelltextes in die vom EMR verständliche Maschinesprache sind bestimmte Bedingungen zu erfüllen. So bedarf es eines festgelegten Datenformates, damit der Assembler korrekt arbeiten kann. Im allgemeinen sieht dieses Datenformat etwa wie folgt aus:

[marke:] mnemonik [Zieloperand] [,Quelloperand] [!Kommentarfeld!]

Beispiel:

OTTO: ADD @%17 ,#%4F !Addiere ...!

- | | |
|---|--|
| - Registerkennzeichnung durch Adresse | - dezimal, z.B. 240
- hexadezimal, z.B. %F0 |
| - Arbeitsregisterkennzeichnung durch R | - dezimal, z.B. R10
- hexadezimal, z.B. R%A |
| - indirekte Registeradressierung durch @ | - dezimal, z.B. @127
- hexadezimal, z.B. @%7F |
| bei Verwendung von Arbeitsregistern | - dezimal, z.B. @R15
- hexadezimal, z.B. @R%F |
| bei Doppelregistern | - dezimal, z.B. @126
- hexadezimal, z.B. @%7E |
| bei Doppelarbeitsregistern | - dezimal, z.B. @RR14
- hexadezimal, z.B. @RR%E |
| - Direktoperanden durch # | - dezimal, z.B. #100 |

- hexadezimal, z.B. **##64**

Die in eckigen Klammern gesetzten Daten sind optional, d.h. nicht in jedem Fall zwingend anzugeben, sondern nur entsprechend dem jeweiligen Befehl.

Leider ist die Notation bei den verschiedenen Assembler-Programmen nicht ganz einheitlich. Zur Kennzeichnung des Kommentarfeldes wird als Trennzeichen häufig auch das „;“ vorgesehen.

Zusätzlich zur möglichen Angabe von dezimalen und hexadezimalen Zahlen gestatten manche Assembler auch die Eingabe binärer und oktalen Zahlen. Die Kennzeichnung erfolgt in diesem Fall beispielsweise durch:

%(2) → duale Zahl in der Form **b₇ b₆ b₅ b₄ b₃ b₂ b₁ b₀**
%(8) → oktale Zahl

Es sind aber auch andere Darstellweisen möglich, im Zweifelsfall ist die Dokumentation des Assembler-Programms konsultieren.

7. Befehlssatz, Beispiele

Der Befehlssatz des U88xx-EMR kann in folgende Befehlsgruppen eingeteilt werden:

- Transportbefehle (Lade- und Blocktransportbefehle)
- Verarbeitungsbefehle (Arithmetik-, Logik-, Bitmanipulations-, Rotations- und Schiebebefehle)
- Steuerbefehle (CPU- und Programmsteuerbefehle)

Für eine vollständige Übersicht zum vorhandenen Befehlssatz - siehe die Befehlsliste im Anhang.

7.1 Ladebefehle

Die Ladebefehle agieren alle nach dem gleichen Schema:

LD Ziel (destination), **Quelle** (source) Wirkung: Das Ziel wird mit dem Inhalt der Quelle geladen, die Quelle bleibt unverändert

Beispiel:

Der Inhalt von Register 7 sei 1 0 0 1 0 1 1 1

Register 8 sei 0 1 1 0 1 0 0 1

LD 7, 8 → **R 7** 0 1 1 0 1 0 0 1 Es erfolgt keine Veränderung des Flagregisters!

Weitere Ladebefehle:

LDC Ziel, Quelle für ext. Programmspeicher (/DM = H)

LDE Ziel, Quelle für ext. Datenspeicher (/DM = L)

CLR Ziel Löschen des Registerinhalts, d.h. alle Bits auf „0“ gesetzt

PUSH Quelle Kopieren des Registerinhaltes an die Speicherstelle, die durch den aktuellen Inhalt des Stackregisters adressiert wird.

Der Inhalt des Stackregisters wird um 1 vermindert (dekrementiert).

POP Ziel Kopieren des Inhalts der Speicherstelle, die durch den aktuellen Inhalt des Stackregisters adressiert wird, in das angegebene Register;

Die PUSH-/POP-Befehle agieren nur für 1 Byte!

Der Inhalt des Stackregisters wird um 1 erhöht (inkrementiert).

Weiters zum Stackregister siehe Pkt. 9.

7.2 Blocktransportbefehle

Wie beim U880 gibt es beim U88xx-EMR auch Blocktransportbefehle, jedoch nur in einfacher Form. Den beim U880 vorhandenen Befehl **LDIR** und den Blocksuchbefehl **CPIR** gibt es leider nicht. Trotzdem lassen sich damit recht einfach Datentransporte im Block realisieren.

Die Blocktransportbefehle sind ausschließlich für den blockweisen Datentransport zwischen Registerspeicher und externem Daten- oder Programmspeicher vorgesehen.

LDCI Ziel, Quelle wie bei **LD**, jedoch gilt dieser Befehl nur bei Zugriff im Programmspeicher (bei ext. Speicher wird /DM = H)

Beispiel:

Der Inhalt von	Register %40 sei	%80	(Doppelregisterpaar RR0
	Register %41 sei	%20	als Adresszeiger im ext. Speicher)
	Register %42 sei	%1F	(R2 als Register-Adresszeiger)
	Register %1F sei	%5A	

Für die Übertragung eines Datenblocks, z.B. für 5 Bytes, muß dann dieser Befehl in einer Schleife implementiert werden.

Loop: LDCI @RR0, @R2 D3 20

Anschließend sind die 8-Bit-Daten von Register-Adresse %1F bis %23 auf den externen Speicherbereich %8020 bis %8024 kopiert.

Neben effizienten Ladebefehlen bestimmen die Verarbeitungsbefehle die Leistungsfähigkeit eines EMR. Obwohl ein recht umfangreicher Satz vorhanden ist, fehlen – wie auch schon beim U880 - die so oft benötigten Multiplikationsbefehle.

Der einfachste Arithmetikbefehl ist die Addition des Inhalts zweier Speicherstellen. Diese können jedoch nur im Registersatz selbst vorliegen.

Beispiel:

XOR Ziel, Quelle wie bei AND, jedoch Verknüpfung mit logischem EXKLUSIV-
ODER

Es gelten folgende Wahrheitstabellen:

	AND		OR		XOR	
Ziel	0	1	0	1	0	1
Quelle						
0	0	0	0	1	0	1
1	0	1	1	1	1	0

Daraus kann man auch folgende Sätze ableiten:

- AND Ziel, Quelle** gilt auch, dass alle Bits gelöscht werden an den Stellen, wo in der Quelle die Bits auf „0“ gesetzt sind.
- OR Ziel, Quelle** gilt auch, dass alle Bits gesetzt werden an den Stellen, wo in der Quelle die Bits auf „1“ gesetzt sind.
- XOR Ziel, Quelle** gilt auch, dass alle Bits invertiert werden an den Stellen, wo in der Quelle die Bits auf „1“ gesetzt sind.

Eine logische Negation des gesamten Bytes im Ziel ist auch möglich:

COM Ziel Bildung des Komplements im Ziel

Ziel und **Quelle** können über Register-, indirekte und unmittelbare Adressierung bestimmt werden.

Im Gegensatz zum U880 wird das C-Flag bei den Logik-Befehlen des U88xx-EMR nicht beeinflusst.

7.2.3 Bitbefehle

Im Gegensatz zum umfangreichen Satz Bitmanipulationsbefehle des U880 ist hier beim U88xx-EMR die Ausstattung recht spartanisch. Es existieren nur zwei Befehle:

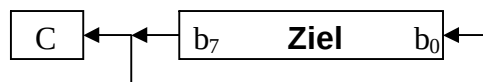
- TM Ziel, Quelle** testet durch Setzen entsprechender Bits in der Quelle die betreffenden Bits der Zieldaten auf „1“. Der Inhalt der Zieldaten wird nicht verändert.
Das Z-Flag ist dann „0“, wenn alle Bit-Tests „0“ ergeben, oder in der Wirksamkeit wie ein **NOR** mit **Quelle** und **Ziel**, jedoch ohne Veränderung der Zieldaten.
- TCM Ziel, Quelle** wie **TM**, jedoch testen auf „0“.
Das Z-Flag ist dann „1“, wenn alle Bit-Tests „1“ ergeben, oder in der Wirksamkeit wie ein **AND** mit **Quelle** und **Ziel**, jedoch ohne Veränderung der Zieldaten.

Auch hier gelten die gleichen Adressierungsregeln, wie bei **ADD** beschrieben.

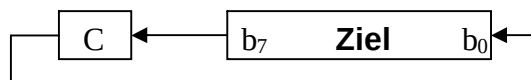
7.2.4 Rotations- und Verschiebepfehle

Da der U88xx-EMR keine speziellen Multiplikations- und Divisionsbefehle kennt, haben die Rotations- und Verschiebepfehle eine besondere Bedeutung. Rotations- und Verschiebepfehle bewirken eine Verlagerung der jeweiligen Bitstellen nach rechts oder links. Je nach dem, ob die rechts oder links herausgeschobene Bitstelle links oder rechts wieder eingefügt wird, spricht man von einer Bit-Rotation, sonst von einer Bit-Verschiebung. Häufig wird das Carry-Flag in die Verschiebe-Operation einbezogen. Die Rotations- und Verschiebepfehle sind nur auf die internen Register anwendbar.

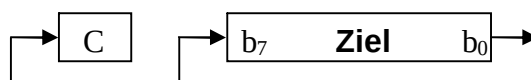
RL Ziel Der Inhalt von **Ziel** wird um eine Stelle nach links verschoben, wobei das links herausgeschobene b_7 sowohl in das Carry-Flag, als auch rechts als b_0 wieder eingetragen wird.



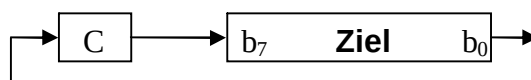
RLC Ziel Der Inhalt von **Ziel** wird zusammen mit dem Inhalt des Carry-Flags um eine Stelle nach links verschoben. Das Bit b_7 kommt in das Carry-Flag und der vorhergehende Inhalt des Carry-Flags wird als neuer Inhalt von Bitposition b_0 eingetragen.



RR Ziel Der Inhalt von **Ziel** wird um eine Stelle nach rechts verschoben, wobei das rechts herausgeschobene b_0 sowohl in das Carry-Flag, als auch in die Bitposition b_7 wieder eingetragen wird.



RRC Ziel Der Inhalt von **Ziel** wird um eine Stelle nach rechts verschoben, wobei das rechts herausgeschobene b_0 in das Carry-Flag verschoben wird und dessen vorheriger Inhalt als Bitposition b_7 eingetragen wird.



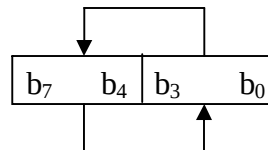
SRA Ziel

Der Inhalt von **Ziel** wird arithmetisch um eine Stelle nach rechts verschoben, in dem die Bitposition b_7 nach b_6 kopiert und der Inhalt von b_0 in das Carry-Flag verschoben wird.



SWAP Ziel

Durch diesen Befehl werden die beiden Tetraden (H-Nibble, L-Nibble) vertauscht.



Auch hier gelten die gleichen Adressierungsregeln, wie bei **ADD** beschrieben.

7.3 Programmsteuerbefehle

Zur Organisation des Programmablaufes werden Steuerbefehle benötigt, die entsprechend vorliegender Bedingungen den Programmablauf beeinflussen. Dazu gehören Befehle zum Aufruf von Unterprogrammen und die besonders wichtigen bedingten Sprungbefehle. Eine spezielle Form davon ist der unbedingte Sprung.

JP cc, Ziel

Sprung zur Adresse **Ziel**, wenn die Bedingung **cc** erfüllt ist.

Ziel sind entweder zwei aufeinander folgende Speicherstellen im Programmspeicher, wobei im Gegensatz zum U880 hier zuerst das High- und dann das Low-Byte anzugeben ist.

Eine andere Möglichkeit, die allerdings nur bei unbedingtem Sprung gilt, besteht in der indirekten Adressierung mit einem Doppelregisterpaar, in denen dann das eigentliche Sprungziel steht.

Wirkung: Der Programmzähler (ein sonst nicht erreichbares 16-Bit-Register) wird auf den Wert **Ziel** gesetzt.

JR cc, e

Sprung zu einer Adresse, die sich aus dem aktuellen Programmzählerstand addiert mit der Distanz **e** (Displacement) ergibt, wenn die Bedingung **cc** erfüllt ist. Vorausgesetzt ist hierbei, dass sich das Ansprungsziel -128 bis 127 Byte vom aktuellen Programmzählerstand befindet. Die Distanz **e** wird wie üblich bei Werten mit Vorzeichen im 2-Komplement angegeben.

DJNZ Quelle, e

Dekrementiere die **Quelle** (ein Arbeitsregister!) und springe zu der Adresse, die sich aus dem aktuellen Programmzählerstand addiert mit der Distanz **e** (Displacement) ergibt. Erreicht der Inhalt von **Quelle** den Wert „Null“, wird kein Sprung ausgeführt, sondern der nachfolgende Befehl.

Für **e** = 0 gilt ca. 700µs/Schleife.

CALL Ziel

Aufruf eines Unterprogramms, welches an der mit **Ziel** bezeichneten Adresse beginnt.

Ziel sind entweder zwei aufeinander folgende Speicherstellen im Programmspeicher, wobei im Gegensatz zum U880 hier zuerst das High- und dann das Low-Byte anzugeben ist.

Eine andere Möglichkeit besteht in der indirekten Adressierung mit einem Doppelregisterpaar, deren Inhalt die Startadresse des Unterprogramms ist.

Wirkung: Der Programmzähler (ein sonst nicht erreichbares 16-Bit-Register) wird auf den Wert **Ziel** gesetzt. Der Stackregisterzeiger (R254/255) wird um 1 vermindert und zunächst das L-Byte des Programmzählers im nun aktuellen Stackregister abgespeichert, dann noch einmal um 1 vermindert, um zum Schluß das H-Byte des Programmzählers abzuspeichern.

RET

Unterprogrammrücksprung

Wirkung: Vom aktuellen Stackregister-Inhalt wird zunächst das H-Byte in den Programmzähler geladen, dann wird der Stackregisterzeiger (R254/255) um 1 erhöht und danach das L-Byte in den Programmzähler geladen. Damit wird das Programm an der Stelle im Programm fortgesetzt, die dem zum **RET** zugehörigen **CALL**-Befehl folgt.

IRET

Dieser Befehl stellt das im Stack gerettete Flagregister und den Programmzähler wieder in den Zustand vor Aufruf der Interruptroutine. Außerdem wird im Interruptmaskenregister R251 automatisch das Master-Aktivierungsbit gesetzt, so dass der EMR wieder für Interruptanforderungen bereit ist (**EI** gesetzt).

Die beim U880 vorhandenen bedingten **CALLs** und **RETs** sind beim U88xx nicht existent.

Im Vergleich zum U880 sind jedoch die Sprungbedingungen beim U88xx-EMR viel komplexer. Nachfolgende Tabelle zeigt die vielfältigen Möglichkeiten.

Sprungbedingung	Codierung		Bedeutung	gesetzte Flags
Mnemonik cc	Binär	Hex		
	0 0 0 0	0	immer falsch (always false)	
LT	0 0 0 1	1	kleiner als (less than)	$(S \text{ XOR } V) = 1$
LE	0 0 1 0	2	kleiner als oder gleich (less than or equal)	$(Z \text{ OR } (S \text{ XOR } V)) = 1$
ULE	0 0 1 1	3	vorzeichenfrei kleiner als oder gleich (unsigned less than or equal)	$(C \text{ OR } Z) = 1$
OV	0 1 0 0	4	Überlauf (overflow)	$V = 1$
MI	0 1 0 1	5	Minus	$S = 1$
Z, EQ	0 1 1 0	6	Null (zero), gleich (equal)	$Z = 1$
C, ULT	0 1 1 1	7	Übertrag (carry), vorzeichenfrei kleiner als (unsigned less than)	$C = 1$
	1 0 0 0	8	immer richtig (always true)	
GE	1 0 0 1	9	größer als oder gleich (greater than or equal)	$(S \text{ XOR } V) = 0$
GT	1 0 1 0	A	größer als (greater than)	$(Z \text{ OR } (S \text{ XOR } V)) = 0$
UGT	1 0 1 1	B	vorzeichenfrei größer als (unsigned greater than)	$((C = 0) \text{ AND } (Z = 0)) = 1$
NOV	1 1 0 0	C	kein Überlauf (no overflow)	$V = 0$
PL	1 1 0 1	D	Plus	$S = 1$
NZ, NE	1 1 1 0	E	nicht Null (not zero), nicht gleich (not equal)	$Z = 0$
NC, UGT	1 1 1 1	F	kein Übertrag (no carry), vorzeichenfrei größer als oder gleich (unsigned greater than or equal)	$C = 0$

Für manche Bedingungscode existieren mehrere Mnemonics, bzw. Bedeutungen. In der Regel sollten in einem Programm immer die gleichen Mnemonics verwendet werden. Im

Ausnahmefall sind aber auch für eine bessere Lesbarkeit des Quelltextes beide Varianten zulässig.

7.3 CPU-Steuerbefehle

Zur Komplettierung des Befehlssatzes existieren eine Reihe besonders wichtiger Befehle.

NOP Befehl ohne Wirkung,
außer einer „Bearbeitungszeit“ von ca. 1,5µs

SRP Quelle lade den Registerpointer (R253) mit dem Inhalt der **Quelle**,
ist besonders wichtig für die Adressierung von Arbeitsregistern
Der hier als **Quelle** spezifizierte Wert wird als Bits b₄ bis b₇ in
R253 geladen. Die Bits b₀ bis b₃ sind dabei immer 0 gesetzt.
Zulässige Werte sind demnach:

(hex)	%00	(dezimal)	0
	%10		16
	%20		32
	%30		48
	%40		64
	%50		80
	%60		96
	%70		112
	%F0		240

Werte im Bereich %80 bis %E0 sind unzulässig!

DI Sperren der Interruptannahme,
Wirkung: Löschen von b₇ im R251, d.h. b₇ = 0

EI Aufheben der Interrupt-Sperre,
Wirkung: Setzen von b₇ im R251, d.h. b₇ = 1

Die spartanische Ausstattung mit Bitmanipulationsbefehlen wird beim U88xx-EMR bezüglich des Flagregisters R252 etwas erweitert. Da der Inhalt des Carry-Flags häufig als Bedingung für Programmsprünge verwendet wird, kann hiermit zweckmäßigerweise eine Voreinstellung vorgenommen werden. Für die anderen Flags existieren leider solche Möglichkeiten nicht, es sei denn über den Befehl **CLR 252**, welcher bewirkt, dass alle gesetzten Bits gelöscht, d.h. auf „0“ gesetzt werden. Eine gezielte Auswahl einzelner Bits, wie beim Carry-Flag ist bei den anderen nicht möglich.

RCF Rücksetzen des Carry-Flag, d.h. b₇ im R252 = 0

SCF Setzen des Carry-Flag, d.h. b₇ im R252 = 1

CCF Komplementiere das Carry-Flag, d.h. b₇ im R252 = /b₇

7.4 Beispielprogramme

In Anwendung des nunmehr erläuterten Befehlssatzes des U88xx-EMR sollen drei Beispielprogramm-Sequenzen angegeben werden, die in der Praxis immer wieder benötigt werden.

7.4.1 Addition für 16-Bit

Da keine 16-Bit-Additionsbefehle vorhanden sind, muß diese Aufgabe softwaremäßig gelöst werden.

Gegeben sei: 2x 16-bit-Zahlen (integer) in den Registern

	H	L
a:	%10	%11
b:	%12	%13

gesucht ist: $a := a + b$

Lösung:

Variante 1, mit vollständiger Register-Adressierung

ADD %11, %13	04 13 11	2,5µs
ADC %10, %12	14 12 10	<u>2,5µs</u>

5,0µs bei $f_T = 8 \text{ MHz}$

Variante 2, mit Arbeitsregisteradressierung, akt. Registerpointer = %10

ADD R1, R3	02 13	1,5µs
ADC R0, R2	12 02	<u>1,5µs</u>

3,0µs bei $f_T = 8 \text{ MHz}$

Man erkennt die wesentlich kleineren Bearbeitungszeiten bei Anwendung der Arbeitsregister-Adressierung.

7.4.2 Multiplikation für 8 Bit

Da keine Multiplikationsbefehle vorhanden sind, muß diese Aufgabe softwaremäßig gelöst werden.

Gegeben sei: 2x 8-bit-Zahlen (integer) in den Registern

a:	R0
b:	R1

gesucht ist: $a := a * b$ (im Doppelregister **RR2**)

Lösung:

MUL:	LD R4, #8	Zähler setzen
	CLR R2	Akku löschen
MU2:	RR R0	c := aktuelle Multiplikatorstelle ins Carry-Flag
	JR NC, MU1	wenn c = 0, sonst
	ADD R2, R1	aktualisieren der Teilsumme
MU1:	RRC R2	einbeziehen von Carry-Flag
	RRC R3	auffangen des niederwertigen Bits

DJNZ R4, MU2 das Ganze 8x

RET

Die Routine hat eine Länge von 17 Byte bei einer Ausführungszeit von 93µs
(bei $f_T = 8 \text{ MHz}$).

7.4.3 Division für 8 Bit

Da keine Divisionsbefehle vorhanden sind, muß diese Aufgabe softwaremäßig gelöst werden.

Gegeben sei: Dividend **c**, 16-bit-Zahl (integer) im Doppelregister **RR2**

 Divisor **b**, 8-Bit-Zahl in Register **R1**

gesucht ist: $a := c / b$ (im Register **R3**, Rest in **R2**)

Lösung:

DIV: LD R0, #8

DIV2: SUB R2, R1

JR NC, DIV1

ADD R2, R1

DIV1: RLC R3

RLC R2

DJNZ R0, DIV2

COM R3

RET

Die Routine hat eine Länge von 17 Byte bei einer Ausführungszeit von 98µs
(bei $f_T = 8 \text{ MHz}$).

8. Organisation des Stapel-Speichers (Stack)

Die universellen internen Register R4 bis R127 des U88xx-EMR werden für verschiedene Aufgaben benötigt:

- als Akkumulator für Rechenoperationen,
- als Adress-Register für die indirekte Register-Adressierung,
- als Adress-Register für die externe Speicher-Adressierung und
- als Stack-Register, wenn dieser intern festgelegt ist.

Während für die ersten drei Aufgaben bereits bei der Erläuterung des Befehlssatzes umfassende Informationen dargelegt wurden, sind bezüglich der Verwendung als Stack-Register noch einige Ergänzungen notwendig.

Je nach Hardware-/Softwareumfang des betreffenden EMR-Systems kann das Stack im internen Registersatz oder im externen RAM definiert werden. Die Festlegung der Betriebsart wird im Steuerregister R248 b_2 vorgenommen. Siehe dazu auch Pkt. (...). Wird das Bit gesetzt, liegt der Stack im internen Registersatz. Für den Stackpointer genügt dann das Register R255. Das Register R254 kann für allgemeine Zwecke verwendet werden.

Ist in dem b_2 von R248 eine „0“ eingetragen, wird der Stack im externen Datenspeicher definiert. Dann wird durch das Stackpointer-Registerpaar R254/255 der High- und Low-Teil der Adresse gebildet. Die Anfangsadresse (= Ladewert des Stackpointers) sollte dabei stets die höchste im System verwendbare RAM-Adresse sein, weil durch Rettungsaktionen für Registerinhalte (z.B. **PUSH**-Befehle) stets der Stackpointer dekrementiert wird. Bei *internem* Stack ist das allgemeine Register R127 in diesem Fall die Startadresse für den Registerpointer, zu laden mit dem Befehl

LD SPH, #80.

Beim *externen* Stack hängt diese Definition von den jeweiligen Hardwareverhältnissen ab, d.h. welche höchste RAM-Adresse im Datenspeicher vorhanden ist. Danach müssen beide Stackpointer-Register **SPL** und **SPH** geladen werden.

Grundsätzlich gilt, dass vom Programmierer immer überprüft werden sollte, ob durch den Stack nicht von „oben nach unten“ im Programm verwendete RAM-Zellen überschreiben könnten. Gleiches gilt auch für den Stackverlauf von „unten nach oben“, welcher im Fehlerfall (z.B. mehr **POPs** als **PUSHs**) den Stack „überlaufen“ lässt. Die Folge davon sind undefinierbare Programmabläufe → Programmabsturz.

9. Beschreibung der Steuerregister

Im U88xx-EMR sind im Bereich von R240 bis R255 des internen Registersatzes die Steuerregister angeordnet. Im Zusammenhang mit praktischen Anwendungen wird im Pkt. 14 auch noch einmal im Detail darauf eingegangen.

9.1 Serielles Ein-/Ausgaberegister R240

Das Register **R240** kann beschrieben und gelesen werden. Dort werden die seriell zu sendenden Daten abgelegt, bzw. werden die empfangenen Daten abgeholt.

Vorraussetzung ist, dass im Steuerregister **R247** $b_6 = 1$ gesetzt wurde, damit an P3₀ ein serieller Eingang und an P3₇ ein serieller Ausgang konfiguriert wird.

9.2 Zeitgeber-Betriebsartenregister R241

Durch Programmierung des Registers **241** werden die Funktionen der Timer T0 und T1 in den Betriebsarten Zähler/Zeitgebertakt festgelegt.

<u>b₇</u>	<u>b₆</u>	Steuerung der T _{out} -Funktion an P3 ₆
0	0	P3 ₆ wird durch b ₅ von R247 gesteuert (siehe Pkt. 9.8)
0	1	T0-Ausgang
1	0	T1-Ausgang
1	1	interner Taktausgang

<u>b₅</u>	<u>b₄</u>	Steuerung der T _{in} -Funktion an P3 ₁
0	0	externer Takteingang für T1, unter Umgehung der 1:4-Teiler direkte Ansteuerung der Vorteiler (siehe Bild ...), für Zähleranwendungen.
0	1	T1 wird durch den internen Takt (1:8 XTAL-Frequenz) über eine durch P3 ₁ gebildete Torsteuerung getaktet, durch einen H/L-Übergang wird außerdem ein Interrupt erzeugt, für Zeitmessungen geeignet.
1	0	T1 wird mit dem Inhalt von R242 geladen und mit dem internen Takt getaktet, wenn an P3 ₁ ein H/L-Übergang auftritt (Triggerfunktion). Weitere Ereignisse an beeinflussen die T1-Funktion dann nicht mehr.
1	1	wie vor, jedoch ist T1 retriggerbar, d.h. mit jedem H/L-Übergang an P3 ₁ beginnt die Zählfunktion von vorn

b ₃	Starten des T1-Zählers, wenn b ₃ = 1 gesetzt wird, in diesem Fall wird der Inhalt von R242 einschließlich Vorteiler R243 mit jedem Takt dekrementiert. Wird b ₃ zurückgesetzt, z.B. bei RESET, ist der Zählvorgang unterbrochen.
b ₂	Laden des Startwertes (Inhalt von R242 und R243) in T1. b ₂ und b ₃ können gleichzeitig geladen werden.
b ₁ , b ₀	wie vor, jedoch für T0

9.3 Zähler/Zeitgeberregister T1 (R242)

Das Register **242** kann gelesen und beschrieben werden.

In der Lesefunktion enthält es den aktuellen Zählerstand von T1.

Mit der Schreibfunktion wird der Anfangswert im Bereich von %1 bis %100 (1 bis 256 dezimal) für T1 festgelegt. Erreicht der Zählvorgang „0“, wird ein Interrupt ausgelöst.

9.4 Vorteilerlade-Register für T1 (R243)

Diese Register kann nur beschrieben werden.

In b₇ bis b₂ wird der Vorteiler-Anfangswert abgelegt.

b ₁	legt die Taktquelle fest, bei b ₁ = 0 wird der Takteingang Tin (P3 ₁) verwendet (vorrausgesetzt in R241 so festgelegt); bei b ₁ = 1 wird der interne Takt (1:4) verwendet.
b ₀	legt die Betriebsart des Vorteilers fest, bei b ₀ = 0 wird nach Laden des Anfangswertes der Inhalt von R242 einmalig bis auf „0“ heruntergezählt und dann ein Interrupt erzeugt; bei b ₁ = 1 wird der Anfangswert jedes Mal beim Erreichen von „0“ neu geladen, solange b ₃ in R242 gesetzt ist. Neue Anfangswerte beeinflussen den aktuellen Zählvorgang nicht, sie werden erst nach Erreichen von „0“ für die weiteren Zählzyklen berücksichtigt.

9.5 Zähler/Zeitgeberregister für T0 (R244)

Das Steuerregister **R244** realisiert die gleichen Funktionen wie **R242**.

9.6 Vorteilerlade-Register für T1 (R245)

Das Steuerregister **R245** realisiert die gleichen Funktionen wie **R243**, jedoch mit der Ausnahme, dass b₁ nicht wirksam ist.

9.7 Betriebsartenregister für Port 2 (R246)

Für Port 2 (**R2**) kann jedes Bit als Ein- oder Ausgang konfiguriert werden. Dazu werden im zugehörigen Betriebsartenregister **R246** die entsprechenden Bits gesetzt (= 1: Eingang) oder rückgesetzt (= 0: Ausgang). Nach einem RESET enthält **R246** den Wert %FF, d.h. alle 8 Anschlüsse sind Eingänge.

Eine weitere Konfiguration für Port 2 kann im b_0 von **R247** vorgenommen werden (siehe dort).

In **R246** kann nur geschrieben werden.

9.8 Betriebsartenregister für Port 3 (R247)

In diesem Steuerregister werden die Anschlüsse für das Port 3 (**R3**) konfiguriert. Es können Festlegungen für parallele Ein-/Ausgabe, Zeitgeber-Ein-/Ausgabe, Handshake-, oder Statusausgänge und Interruptanforderungen getroffen werden.

In **R247** kann nur geschrieben werden.

b_7	Wird durch Setzen von b_6 serielle Ein-/Ausgabe gewählt, kann mit $b_7 = 1$ ungerade Parität für Sende- und Empfangsdaten festgelegt werden, bei $b_7 = 0$ entfällt das.		
b_6	Mit $b_6 = 1$ wird $P3_0$ als serieller Eingang und $P3_7$ als serieller Ausgang definiert, bei $b_6 = 0$ ist $P3_0$ ein normaler Eingang und $P3_7$ ein normaler Ausgang.		
b_5	Mit $b_5 = 1$ werden die Anschlüsse $P3_1$ und $P3_6$ als Handshake-Steuerung für Port 2 konfiguriert, bei $b_5 = 0$ ist $P3_1$ ein normaler Eingang und $P3_6$ ein normaler Ausgang. Allerdings wirken dann die Konfigurationsbits b_1 im Steuerregister 243 und b_4 bis b_7 im Steuerregister 241 (siehe dort). Bei $b_5 = 1$ haben die Konfigurationsbits b_1 im Steuerregister 243 und b_4 bis b_7 im Steuerregister 241 keinen Einfluß.		
b_4, b_3	b_4	b_3	
	0	0	$P3_3 = \text{Eingang}, P3_4 = \text{Ausgang}$
	1	0	$P3_3 = \text{Eingang}, P3_4 = /DM$
	1	1	Handshake-Betrieb für Port 1
b_2	Mit $b_5 = 1$ werden die Anschlüsse $P3_2$ und $P3_5$ als Handshake-Steuerung für Port 0 konfiguriert, bei $b_2 = 0$ ist $P3_2$ ein normaler Eingang und $P3_5$ ein normaler Ausgang.		

b ₁	wird nicht benutzt
b ₀	Mit b ₀ = 0 werden alle Ausgänge von Port 2 als „Open-Drain-Ausgänge“ konfiguriert, so dass z.B. mit anderen Open-Drain-Ausgängen ein so genanntes „wired-OR“ (verdrahtetes ODER) realisiert werden kann.

9.9 Betriebsartenwahlregister für Port 0 und Port 1 (R248)

In diesem Steuerregister werden die Anschlüsse für Port 0 (**R0**) und Port 1 (**R1**) konfiguriert. Außerdem kann das Zeitverhalten externer Speicher berücksichtigt und die Lage des Stapelspeichers (Stack) festgelegt werden.

In das Register kann nur geschrieben werden.

b ₇ , b ₆	b ₇	b ₆	
	0	0	P0 ₄ bis P0 ₇ sind Ausgänge
	0	1	P0 ₄ bis P0 ₇ sind Eingänge
	1	x	P0 ₄ bis P0 ₇ sind Adressleitungen A12 bis A15
b ₅	Der Speicherzyklus wird um eine Taktperiode verlängert, wenn b ₅ = 1 gesetzt ist.		
b ₄ , b ₃	b ₄	b ₃	
	0	0	P1 ₀ bis P1 ₇ ist ein Byte-Ausgang
	0	1	P1 ₀ bis P1 ₇ ist ein Byte-Eingang
	1	0	P1 ₀ bis P1 ₇ sind Adressleitungen A0 bis A7
	1	1	P1 ₀ bis P1 ₇ sind hochohmig*
*) außer den Adressleitungen von Port 1 gehen auch /AS, /DS, /RW und für den Fall solcher Konfiguration auch P0 ₀ bis P0 ₃ bzw. P0 ₀ bis P0 ₇ in den „tri-state“-Zustand)			
b ₂	Mit b ₂ = 0 wird der Stack extern definiert, für b ₂ = 1 gilt interner Stack.		
b ₁ , b ₀	b ₁	b ₀	
	0	0	P0 ₀ bis P0 ₃ sind Ausgänge
	0	1	P0 ₀ bis P0 ₃ sind Eingänge
	1	x	P0 ₀ bis P0 ₃ sind Adressleitungen A8 bis A11

9.10 Interruptprioritätsregister R249

Die 6 möglichen vektorisierten Interrupts können durch dieses Register in 48 verschiedene Reihenfolgen priorisiert werden und dadurch lassen sich gleichzeitig die Interruptanforderungen entflechten.

Eine Zuordnung kann etwa wie folgt aussehen:

Prioritätsgruppe	Interruptebenen
A	IRQ 3 (SI/P3 ₀) und IRQ 5 (T1)
B	IRQ 0 (P3 ₂) und IRQ 2 (P3 ₁)
C	IRQ 1 (P3 ₃) und IRQ 4 (SO/T0)

Als Prioritätsvarianten lassen sich nun verschiedene Möglichkeiten konfigurieren:

b ₇ , b ₆	wird nicht verwendet			
b ₅	für Gruppe A gilt b ₅ = 0 : IRQ 5 > IRQ 3 b ₅ = 1 : IRQ 3 > IRQ 5			
b ₂	für Gruppe B gilt b ₂ = 0 : IRQ 2 > IRQ 0 b ₂ = 1 : IRQ 0 > IRQ 2			
b ₁	für Gruppe C gilt b ₁ = 0 : IRQ 1 > IRQ 4 b ₁ = 1 : IRQ 4 > IRQ 1			
b ₄ , b ₃ , b ₀	b ₄	b ₃	b ₀	
	0	0	0	nicht definiert
	0	0	1	C > A > B
	0	1	0	A > B > C
	0	1	1	A > C > B
	1	0	0	B > C > A
	1	0	1	C > B > A
	1	1	0	B > A > C
	1	1	1	nicht definiert

Wie ersichtlich bestimmen b₄, b₃, b₀ die Gruppenpriorität, während b₅, b₂, b₁ die Priorität innerhalb der Gruppe festlegt.

In das Register **R249** kann nur geschrieben werden.

9.11 Interruptanforderungsregister R250

Dieses Register kann gelesen, als auch beschrieben werden. Tritt ein Interrupt aus einer der 6 Quellen auf, wird an der betreffenden Stelle eine „1“ eingetragen. Bei gesperrter Interruptannahme (**R251**) kann auch durch zyklische Abfrage (Lesen) des **R250** ein „Polling“-Interrupt realisiert werden.

b ₇ , b ₆	wird nicht verwendet
b ₅	IRQ 5
b ₄	IRQ 4
b ₃	IRQ 3
b ₂	IRQ 2
b ₁	IRQ 1
b ₀	IRQ 0

9.12 Interruptmaskenregister R251

Mit diesem Steuerregister, welches beschreib- und lesbar ist, können die 6 Interrupt-Quellen einzeln oder insgesamt aktiviert oder deaktiviert werden.

b ₇	b ₇ = 0 : alle Interrupts deaktiviert; b ₇ = 1 : Interrupts freigegeben
b ₆	wird nicht verwendet
b ₅	b ₅ = 0 : IRQ 5 deaktiviert; b ₅ = 1 : IRQ 5 freigegeben
b ₄	b ₄ = 0 : IRQ 4 deaktiviert; b ₄ = 1 : IRQ 4 freigegeben
b ₃	b ₃ = 0 : IRQ 3 deaktiviert; b ₃ = 1 : IRQ 3 freigegeben
b ₂	b ₂ = 0 : IRQ 2 deaktiviert; b ₂ = 1 : IRQ 2 freigegeben
b ₁	b ₁ = 0 : IRQ 1 deaktiviert; b ₁ = 1 : IRQ 1 freigegeben
b ₀	b ₀ = 0 : IRQ 0 deaktiviert; b ₀ = 1 : IRQ 0 freigegeben

Bezüglich b₇ gibt es einige Besonderheiten, die beachtet werden sollten. Durch die CPU-Befehle **EI** und **DI** wird b₇ gesetzt bzw. zurückgesetzt. Wird b₇ durch einen Ladebefehl gesetzt, hat das keine interruptfreigebende Wirkung! Wird hingegen durch einen Ladebefehl b₇ zurückgesetzt, werden alle Interrupts gesperrt.

Vor Änderungen in **R249** und **R251** muß unbedingt durch den Befehl **DI** das b₇ im **R251** zurückgesetzt werden.

Während der Ausführung einer Interrupt-Routine wird b₇ im **R251** automatisch rückgesetzt und bei Rückkehr ins Hauptprogramm durch **IRET** wieder gesetzt. (weiteres siehe **Pkt ...**)

9.13 Flagregister R252

Diese Steuerregister enthält die Statusinformationen, welche nach Verarbeitungsbefehlen auftreten können. Es ist beschreib- und lesbar. Zwei Bits (F1, F2) sind allgemein verwendbar, allerdings nicht als Bedingung für Programmsprünge (siehe Pkt 7.3).

b ₇	Carry-Flag (C)
b ₆	Zero-Flag (Z)
b ₅	Sign-Flag (S); (Vorzeichen)
b ₄	Overflow-Flag (V); (Zahlenformat)
b ₃	Decimal-Adjust-Flag (D); (bei DA nach ADD , SUB)
b ₂	Half-Carry-Flag (H)
b ₁	F2
b ₀	F1

9.14 Arbeitsregister-Gruppenzeiger R253

Für die Adressierung mit Arbeitsregistern wird durch das Steuerregister **R253** eine jeweils aktuelle Gruppe von 16 universellen Registern (im Bereich von **R4** bis **R127**) ausgewählt. (siehe auch Pkt. 7.3)

b ₇ , b ₆ , b ₅ , b ₄	= das obere Nibble der Startadresse der aktuellen Arbeitsregistergruppe
b ₃ , b ₂ , b ₁ , b ₀	= sind beim Beschreiben beliebig, es wird jedoch immer „0“ ausgelesen

9.15 Stackpointer R254, R255 (SPL, SPH)

Der Stackpointer setzt sich aus zwei Registern **R254**, **R255** für den niederwertigen und den höherwertigen Teil zusammen. Sie werden auch als **SPL**, **SPH** bezeichnet und sind beschreib- und lesbar.

(weiteres siehe dazu Pkt. 8)

10. Verwendung und Programmierung der Ein-/Ausgabeschnittstellen

Beim U88xx-EMR lassen sich verschiedene Parallel- und auch serielle Schnittstellen konfigurieren. Insgesamt stehen dafür 32 Anschlüsse, d.h. je 8 Anschlüsse an 4 Ports zur Verfügung. Einzelnen Ports sind jedoch alternativ zur Anwendung als normale digitale Ein- und Ausgängen auch noch für bestimmte Anforderungen spezialisiert:

Port 0: Adressenausgabe für externe Speicher

Port 1: Daten-/Adressbus für externe Speicher

Port 3:

- Handshake-Signale für die Ports 0 bis 2,
- Ein-/Ausgang für Timer und SIO,
- der Bussteuerausgang /DM an P3₄

10.1 Parallelschnittstellen Port 0 bis Port 3

10.1.1 Struktur von Port 3

Grundsätzliche Festlegung: P3₀ bis P3₃ : 4x Eingänge
P3₄ bis P3₇ : 4x Ausgänge (gepuffert → Speichern bis auf
Widerruf!)

Konsequenzen: Lesen von P3 ergibt bei L-Nibble: Eingangssignale
H-Nibble: Ausgangssignale
Schreiben geht bei P3₄ bis P3₇ nur, wenn dort keine
Spezialfunktion prog. Ist

10.1.2 Struktur von Port 2

Grundsätzliche Festlegung: P2₀ bis P2₇ : jeder Anschluß ist bitweise als Ein- /oder
Ausgang programmierbar,
Ausgänge auch als Open-Drain progr.

10.1.3 Struktur von Port 1

Grundsätzliche Festlegung: P1₀ bis P1₇ : alle Anschlüsse sind nur byteweise als Ein-
/oder Ausgang programmierbar

10.1.4 Struktur von Port 0

Grundsätzliche Festlegung: P0₀ bis P0₃ : alle Anschlüsse sind nur halbbyteweise als
P0₄ bis P0₇ : Ein- /oder Ausgang programmierbar

Allgemein gilt, dass bei Schreiben in die Ports der Wert im Ausgangsregister abgelegt wird, aber nur dort an die Ausgangs-Anschlüsse gelangt, wenn der betreffende Anschluß als solcher programmiert ist.

Bei Lesen wird der Inhalt des betreffenden Eingabe-Registers gelesen, welches den aktuellen Eingabe-Logik-Pegel widerspiegelt, ohne z.B. das Handshake-Prozedere zu berücksichtigen.

Bei RESET werden alle Ports in Eingaberichtung programmiert!

10.2 Handshake

Für byteweise Kommunikation mit anderen Datenquellen (oder –Senken) wird oft ein so genanntes, hardwaregestütztes „Handshake“-Prozedere angewandt.

Der U88xx-EMR ist mit seinen Steuersignalen **/DAV**, **RDY** bestens dafür ausgerüstet.

Dabei gelten folgende Festlegungen:

/DAV = 0	Daten sind verfügbar (vom Sender)
RDY = 1	zur Übernahme bereit (vom Empfänger)

Diese Vereinbarungen werden beim U88xx-EMR wie folgt umgesetzt:

Bei einer H/L-Flanke von **/DAV** wird ein Interrupt angemeldet. Die darauf abzuarbeitende ISR meldet dem Sender mit der H/L-Flanke von **RDY**, dass die Daten übernommen werden, bzw. worden sind (je nach dem, wo der entsprechende Befehl angeordnet ist).

Nach dem Lesen des Eingabe-Registers und bei **/DAV** = 1 wird **RDY** wieder auf „1“ gesetzt.

Bei Datenausgabe gilt dieses Prozedere analog.

Für einen definierten Start ist eine entsprechende Initialisierung notwendig:

- vorerst Port 3 für normales E/A programmieren,
- Ausgabe von „1“ über Port 3,
- Port 3 für Handshake programmieren.

Eingangs- oder Ausgangs-Handshake wird von der Übertragungsrichtung des b_7 des betreffenden Ports bestimmt.

Beim Lesen von Port 3 wird auch die aktuelle Belegung der Handshake-Signale mit erfasst.

10.3 Programmierung der Ports

Grundsätzlich ist nach jedem RESET eine Initialisierung aller Ports notwendig. Dies erfolgt durch Laden der betreffenden Steuerregister:

- **P10M** Register %F8 (248), Port1 und 2 – Mode-Steuerregister

mit $b_3/b_4 = 11$ wird der BUS hochohmig, so dass z.B.
auch DMA möglich ist (betrifft für P0/P1 alle
Adressbits und /AS, /DS, /WR)

- **P3M** Register %F7 (247), Port3-Mode-Steuerregister
- **P2M** Register %F6 (246), Port2-Mode-Steuerregister

Beispiel 1:

Initialisierung für

- 2k EPROM (schnell)
- P0₄ bis P0₇ → Ausgänge
- P2₀ bis P2₅ → Eingänge
- P2₆, P2₇ → Ausgänge mit Open-Drain
- alle P3-Signale normale E/A
- interner SP

P10M (%F8) 0 0 0 1 0 1 1 x

P3M (%F7) x 0 0 0 0 0 x 0

P2M (%F6) 0 0 1 1 1 1 1 1

INIT: **SRP #F0** 31 F0

 ...

LD R8, #16 8C 16

LD R7, #0 7C 00

LD R6, #3F 6C 3F

 ...

Beispiel 2:

LED-Anzeige und Tastenabfrage

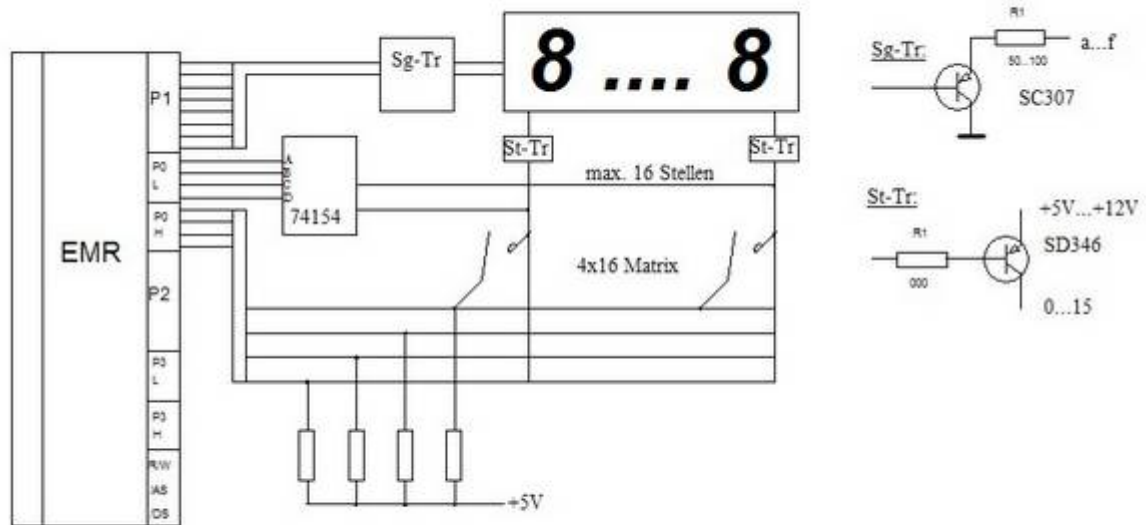


Bild 20: LED-Anzeige und Tastatur

Programmtechnik

- Anzeige durch ISR alle 0,5ms → entspricht ca. 125 Bilder/s
- Bildwiederholpeicher Reg. %16 bis Reg. %1F (7-Segment-Codes)
 Reg. %04 → Anzeigepointer
- Tastenabfrage mit UP Reg. %05 → Vergleichsregister zur Synchr. mit ISR
 Reg. %06 → Tastencode

ISR Anzeige:

INC 4	Pointer +1
AND 4, #%1F	Löschen 3 obere Bits (000xxxxx)
OR 4, #%10	Setzen von Bit 4 (0001xxxx)
LD 1, @4	Port1:= Code
LD 0, 4	Port0:= Auswahl
IRET	(10µs)

Ausführungszeit: 21,5µs → 4,3% Rechenkapazität

Dynamische Tastenabfrage (UP)

DYN:	LD 5, #%10	1. Vergleichswert
DY1:	DI	

	LD 6, 0	Port0-Belegung
	CP 4, 5	Vergleich
	EI	
	JR NZ, DY1	ggf. warten
	TCM 6, #F0	Test ob in Reg. 6 b ₄ bis b ₇ = 1
	JR NZ, DY2	wenn Taste gefunden
	INC 5	nächster Vergleichswert
	CP 5, #20	fertig?
	JR C, DY1	sonst nächste Spalte
DY2:	RET	
	Ausführungszeit: max 16ms	

Statische Tastenabfrage

STA:	CALL DYN	
	JR NZ, STA	Warten auf Ruhezustand
ST1:	CALL DYN	
	JR NZ, ST1	Warten auf Neubetätigung

11. Struktur, Programmierung und Verwendung der Zähler/Zeitgeber

Der U88xx-EMR beinhaltet zwei im wesentlichen völlig getrennte Zähler-Zeitgeberkanäle, die entsprechend der gewünschten Funktion separat programmiert werden können und unabhängig von den anderen Funktionsgruppen (z.B. allg. Registersatz) arbeiten.

Zur Abgrenzung kann angegeben werden, dass bei $f_T = 8 \text{ MHz}$ Eingangsfrequenzen bis zu 1MHz gezählt werden können. Beteiligt daran sind Vorteiler und der eigentliche Zähler **T0** oder **T1**. Das Ergebnis berechnet sich dann aus

$$V_G = V_V * V_Z$$

Vorteiler und Zähler arbeiten zusammen dekrementierend von einem Startwert aus (sog. Zeitkonstante), der vorher geladen werden muß. Erreicht der Zählerstand den Wert „0“, wird ein Interrupt erzeugt.

Wenn Zähler-Zeitgeberkanal **T0** die SIO bedient, wird kein Interrupt erzeugt.

Programmierung: (siehe auch Pkt. 9)

Das Frequenzteilerverhältnis wird durch Laden der Zeitkonstanten in die entsprechenden Steuerregister:

T0 (%F4) Zähler-Zeitgeber0-Steuerregister

T1 (%F2) Zähler-Zeitgeber1-Steuerregister

PRE0 (%F5) Vorteiler0-Laderegister

PRE1 (%F3) Vorteiler1-Laderegister

Das Betriebsarten-Mode-Register **TMR** (%F1) bestimmt die Verwendung der beiden Zähler-Zeitgeberkanäle **T0/T1**. Im Gegensatz zu den Vorteilerladeregistern ist dieses auch lesbar. Durch Lesen von **T0** oder **T1** kann der aktuelle Zählerstand ermittelt werden.

Ist in **P3M** (%F7) Handshake für Port2 programmiert, stehen $P3_1$ und $P3_6$ nicht als T_{in} bzw. T_{out} zur Verfügung (b_1 in **PRE1** ohne Wirkung!). In diesem Fall ist bei $b_1 = 1$ in **PRE1** die Belegung von b_4, b_5 in **TMR** egal.

Beispiel: **Zähler-Initialisierung**

- T0 alle 13µs Impuls an SIO

- T1 für Impulsbreitenmessung mit µs-Auflösung, Eingang $P3_1$

PRE0 0 0 0 0 0 1 x 1

T0 0 0 0 0 1 1 0 1

PRE1 0 0 0 0 0 1 0 1

T1 0 0 0 0 0 0 0 0

TMR 0 0 0 1 1 1 1 1

(nachfolgendes Programm/-beschreibung noch mal logisch prüfen!)

Mit der ersten L/H-Flanke an P3₁ (Impulsanfang) beginnt **T1** abwärts zu zählen.

Am Impulsende tritt IRQ 2 auf (H/L-Flanke an P3₁).

Die dazugehörige ISR ist dann:

IS2:	EI	für evtl. Übertrag → T1I
	LD %5F, %F2	Zählerstand
	COM %5F	Negation
	INC %5F	→ positive Zahl
	LD %5E, %5B	im Doppelregister %5E := Meßwert
	LD %F1, #%1E	nur T1 neu initialisieren!
	IRET	

Wird bei dem Zählvorgang „0“ erreicht, hat das einen Übertrag zur Folge, der wiederum den Interrupt IRQ5 erzeugt.

Die dazugehörige ISR ist dann:

T1I:	INC %5B
	IRET

12. Funktion und Aufbau des seriellen Ein-/Ausgabe-Moduls

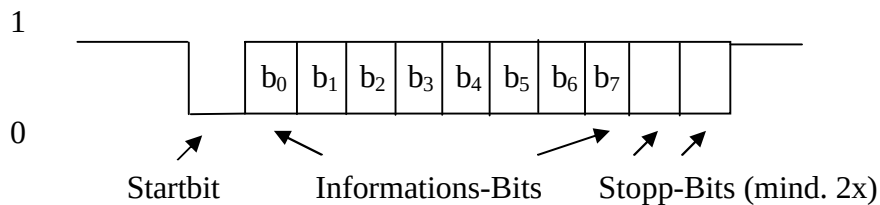
Das serielle Ein-/Ausgabeinterface (SIO) wird über Port3, Anschlüsse $P3_0 = S_{IN}$ und $P3_7 = S_{OUT}$, realisiert. Die dazu notwendigen Konfigurationseinstellungen wurden bereits unter Pkt. 9 erläutert.

Bei der Arbeit mit serieller Kommunikation wird nach folgenden Synchronisationsproblemen unterschieden:

- bitsynchron
- bytesynchron
- blocksynchron

In den beiden ersten Fällen wird auch von asynchronem Datenformat gesprochen.

Die Übertragung beginnt immer mit einer H/L-Flanke der sonst im Ruhezustand auf H-Potential liegenden Datenleitung.



Abgetastet wird immer auf der halben Taktbreite, allerdings unter der Voraussetzung, dass $f_{TS} = f_{TE}$ (bei einer Toleranz = 2%).

Senderseitig beginnt die Übertragung mit dem Einladen des zu sendenden Bytes in das Sende-(Schiebe)register Port3. Ist die Übertragung vollständig erfolgt, wird der Interrupt IRQ 4 generiert.

Empfängerseitig wird bei jedem vollständig eingelesenen Byte der Interrupt IRQ3 erzeugt.

Das empfangene Byte steht dann im Port3 zur Weiterverarbeitung zur Verfügung.

Die mögliche Übertragungsrate hängt u.a. von den Eigenschaften des Übertragungsmediums incl. eventueller Störeinflüsse ab und kann wie folgt berechnet werden:

$$T = T_C * V_{T0} * T_0 * 128,$$

d.h. maximal lassen sich unter optimalen Bedingungen alle $16\mu s$ ein Byte senden/empfangen, bzw. die daraus resultierende Übertragungsrate ist 62 kBit/s.

Unter Einhaltung der o.g. Toleranz lassen sich für eine Reihe standardisierter Übertragungsraten in nachfolgender Tabelle die notwendigen Teilerfaktoren beim Einsatz mit speziellem Quarztakt und Normaltakt 8 MHz angeben.

<u>Bit/s</u>	<u>7,3...MHz</u>	<u>8 MHz</u>
19200	1 : 3	
9600	1 : 6	
4800	1 : 12	1 : 13
2400	1 : 24	1 : 26
1200	1 : 48	1 : 52
600	1 : 96	1 : 104
300	1 : 192	1 : 208

Die Programmierung beschränkt sich auf das Laden des Steuerregisters **P3M**.

Ist dort $b_6 = 0$ gesetzt, ist die SIO nicht aktiv, d.h. **T0** verfügbar mit IRQ3, IRQ4.

P3₀, P3₇ sind normale Ein-/Ausgänge.

Wird in **P3M** $b_6 = 1$ gesetzt, ist die SIO aktiv; IRQ3, IRQ4 für T0 und P3₀, P3₇ als normale E/A sind nicht verfügbar!

Wenn Paritätsprüfung erforderlich ist, muß in **P3M** $b_7 = 1$ gesetzt werden. Dann erfolgt allerdings nur eine 7-Bit-Datenübertragung, da b_7 das Paritätsbit ist.

13. Quellen, Organisation und Anwendung der Interruptverarbeitung

Überall dort, wo vom Programm und dem zu steuernden Prozess eine Synchronisation erforderlich ist, d.h. ein gewisses Maß an Echtzeitverhalten verlangt wird, kommt Interrupt-Verarbeitung zum Einsatz. Hierbei sind weniger die beim PC bekannten Software-Interrupts (Traps) gemeint, sondern ausschließlich Ereignisse, die Hardware-bedingt sind.

13.1 Interrupt-Quellen und Programmauswirkung

Beim U88xx-EMR sind folgende Interruptquellen (IRQ0 bis IRQ5) definiert:

- 4 externe (Bei SIO-Betrieb nur 3!),
- 3 interne (SIO, To, T1)

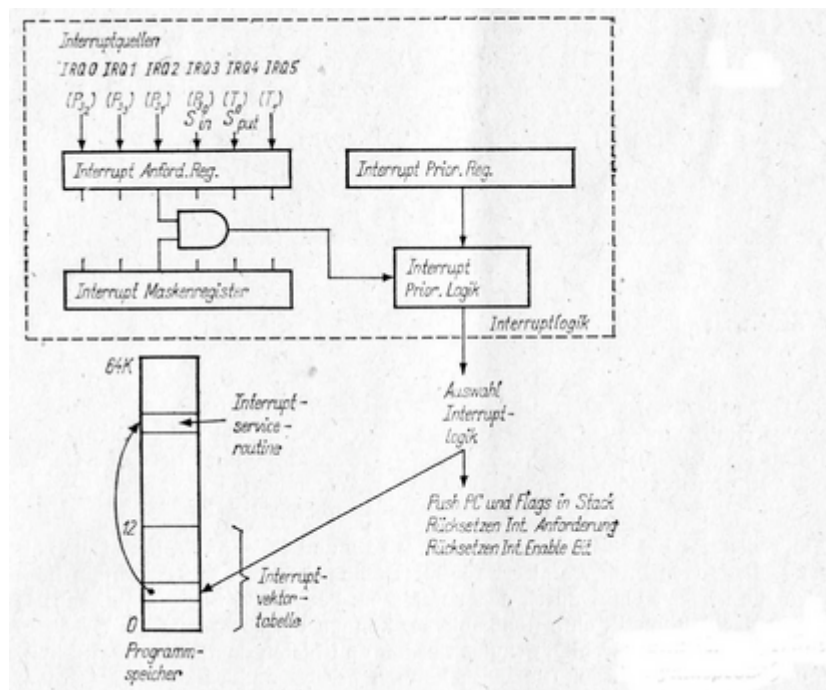


Bild 21: Interruptverarbeitung beim EMR

Die Interruptquellen-Anmeldung erfolgt durch Setzen des betreffenden Bits im Interruptanforderungs-Register **IRQ %FA** (250).

Erzeugt eine Interrupt-Quelle ein Interrupt-Signal, so wird anhand des zugeordneten Interrupt-Vektors am Anfang des Programmspeichers die betreffende Startadresse der so genannten Interrupt-Service-Routine (ISR) ausgelesen und dorthin gesprungen, d.h. der Programmzähler wird auf diese Adresse gestellt. Zuvor werden noch der aktuelle Programmzählerstand und der momentane Inhalt des Flag-Registers auf den Stack gerettet. Alle nachfolgenden Interrupts werden zunächst gesperrt. Es ist aber möglich durch Setzen

des Befehls **EI** gleich am Anfang der ISR wieder eine Freigabe zu programmieren. Damit sind auch „geschachtelte“ Interrupts möglich. Am Ende der ISR steht dann der Befehl IRET, der den Programmzählerstand und den Flagregister-Status vor Eintritt in die ISR durch Rückspeichern vom Stack wieder herstellt. Falls nicht schon innerhalb der ISR geschehen, werden ebenfalls die Interrupts wieder freigegeben.

13.2 Interrupt-Organisation

Der Inhalt des Interruptmaskenregister **IMR** %FB (251) entscheidet jedoch letztendlich über die Interrupt-Annahme (siehe dazu auch Pkt. 9).

Im **IMR** ist b_7 für die globale Interruptfreigabe zuständig. Gesteuert wird b_7 auch über die Befehle **EI** / **DI**.

Wenn mit Interrupts gearbeitet werden soll, ist in jedem Fall nach RESET die Interrupt-Freigabe durch **EI** zu programmieren. Sind hingegen Manipulationen am **IMR** oder am Interruptprioritätsregister **IPR** %F9 (249) vorgesehen, ist unbedingt vorher der Befehl **DI** zu setzen. Der Inhalt dieses Registers entscheidet über die Reihenfolge der Interrupt-Abarbeitung.

Das globale Interrupt-Sperren mit einem Ladebefehl bezüglich b_7 im **IMR** ist zwar möglich, es ist aber besser den Befehl **DI** zu verwenden.

Das globale Freigeben von Interrupts mit einem Ladebefehl bezüglich b_7 im **IMR** ist unsicher, besser ist immer den Befehl **EI** zu verwenden.

Als erster Befehl vor dem Laden der Steuerregister zur Initialisierung eines EMR-Systems sollte unbedingt ein **DI** und dann am Ende der Initialisierungssequenz, vor Beginn der Hauptprogramms ein **EI** gesetzt werden (siehe auch Pkt. 14)

13.3 Interruptannahme

Nach Ausführung jeden Befehls wird auf eine Interrupt-Anmeldung getestet. Bei erfüllten Bedingungen (s.o.) wird folgender Interrupt-Annahmezyklus eingeleitet:

- Rücksetzen des betreffenden Interrupt-Anmeldebits (IRQn)
- Dekrementieren des Stackpointers
- Laden des L-Teils des aktuellen Programmzählerstandes auf die vom Stackpointer adressierte Adresse (je nach Systemkonfiguration intern oder extern)
- Dekrementieren des Stackpointers
- Laden des H-Teils des aktuellen Programmzählerstandes auf die vom Stackpointer adressierte Adresse (je nach Systemkonfiguration intern oder extern)

- Dekrementieren des Stackpointers
- Laden des Inhalts des Flagregisters (%FC) auf die vom Stackpointer adressierte Adresse (je nach Systemkonfiguration intern oder extern)
- Laden des Programmzählers mit der Startadresse der ISR, die als Inhalt des betreffenden Interruptvektors (IRQn) in der Interrupt-Vektortabelle von 00 bis 12 des Programmspeichers steht (ersten 12 Byte des internen ROM, wenn vorhanden, erst H- dann L-Adresse!)

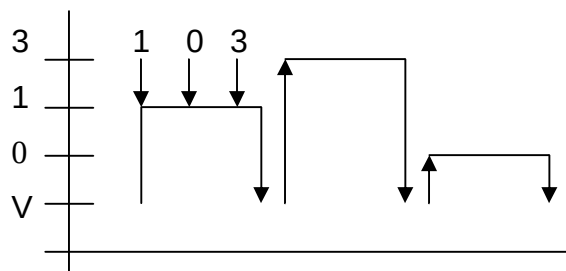
Als Ausführungszeit werden insgesamt ca. 10µs benötigt, wobei allein für **IRET** 4µs veranschlagt werden müssen.

13.4 Mehrebenen-Organisation

Können mehrere Interrupts quasi gleichzeitig auftreten, wird von einem Mehrebenen-Interrupt-System gesprochen. Hierzu gibt es verschiedene Varianten der Abarbeitungsstrategie in hierarchischen Interruptstrukturen.

1. Variante

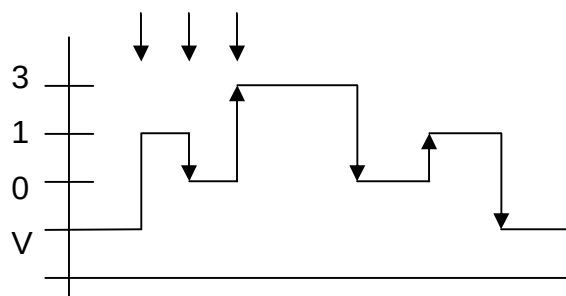
ISR: ...
...
IRET



Bei festgelegter Prioritätsreihenfolge $3 > 1 > 0$ ist diese Variante gut geeignet für Quellen mit höchster Priorität.

2. Variante

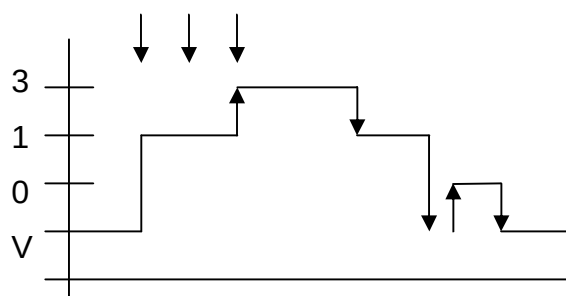
ISR: EI
...
IRET



Diese Variante ist nur geeignet für Quellen mit niedriger Priorität.

3. Variante

ISR: **PUSH %FB**
LD %FB, #...
...



...
DI
POP %FB
IRET

Mit dieser Variante werden die Interrupts in der Reihenfolge ihrer Priorität abgearbeitet.

Aus dem Variantenvergleich lässt sich folgende Struktur für minimalen Zeitverzug bei der Interrupt-Bearbeitung ableiten:

ISR: ...
 ... bis hierher sind zeitkritische Operationen vorzusehen
 EI
 ... ab hier stehen dann die zeitunkritischen Operationen
 IRET

13.5 Polling

Unter bestimmten Voraussetzungen ist es manchmal günstiger, statt einer Interrupt-Bearbeitung eine zyklische Abfrage vorzunehmen. Dies kann z.B. zeitgesteuert erfolgen, so in etwa, wie bei den Software-Interrupts im PC. Dabei ist zu beachten, dass im U88xx-EMR nicht alle Register abfragbar sind. Das betrifft:

%F9, %F8, %F7, %F6, %F5, %F3

Ein Lesevorgang würde als Ergebnis %FF haben!

Alle anderen Register sind lesbar.

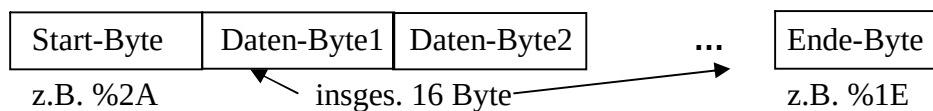
Wird z.B. anstelle einer ISR das Register **%FA** mit Bittestfolgen (**TCM**) abgefragt, so lässt sich mittel eines **CALL**-Befehls auch eine Programmverzweigung realisieren. Ggf. muß durch Rücksetzen des betreffenden IRQ-Bits die weitere „Interrupt-Annahme“ wieder freigegeben werden.

14. Demonstrationsbeispiel der interruptgesteuerten Schnittstellen-Verwaltung an einer konkreten EMR-Konfiguration

Gegeben sind zwei über eine SIO-Kommunikation gekoppelte EMR-Systeme, wovon das eine Messwerte erfasst und diese dem anderen via SIO überträgt. Zur Kontrolle der datenrichtigen Kommunikation sollen die empfangenden Daten vollständig zurückgesandt werden. In der Praxis ist dieses Verfahren allerdings nicht üblich, dort werden lediglich Prüfsummen zur Kontrolle zurückgesandt.

Als Übertragungsrate wird 4800 Bit/s festgelegt.

Für die Datenstruktur wird folgende Blockübertragung gewählt:



Gesucht ist:

- die ISR zum Ablegen der Daten ab Register %20
- die ISR zum Rücksenden
- die Initialisierung für dieses Beispiel

Zunächst sind einige Berechnungen für die Initialisierung notwendig.

Bei der festgelegten Datenübertragungsrate von 4800 Bit/s ergibt das einen Daten-Takt von 208,3µs. Mit dem intern auf 1 : 16 festgelegten SIO-Vorteiler für T0 ist demzufolge ein Teilerfaktor von 1 : 12 (bzw. 1 : 13 bei $f_T = 8$ MHz) anzuwenden.

INI: DI

SRP #%F0	Steuerregister sind aktueller Arbeitsregistersatz		
LD R15, #%80	Stack im internen RAM (Reg. 127)		
LD R11, #%38	IMR:	0	0 1 1 1 0 0 0
LD R9, #%08	IPR:	0	0 0 0 1 0 0 0
LD R8, #%44	P01M:	0	1 0 0 0 1 0 0
LD R7, #%40	P3M:	0	1 0 0 0 0 0 0
LD R6, #%3F	P2M:	0	0 1 1 1 1 1 1
LD R5, #%35	PRE0:	0	0 1 1 0 1 0 1
LD R4, #%01	T0:	0	0 0 0 0 0 0 1
LD R3, #%2B	PRE1:	0	0 1 0 1 0 1 1
LD R2, #%32	T1:	0	0 1 1 0 0 1 0

LD R1, #0F	TMR: 0 0 0 1 1 1 1
CLR 7	Empfangsspeicher löschen (: = 0)
LD 8, #31	Sendepointer := %31 → passiver Zustand
EI	
...	Beginn Hauptprogramm
...	

(die Angabe / Eingabe SIO-T0-Teilerfaktor ist noch mal zu prüfen)

Die Initialisierung für T1 ist für eine Anzeige-/Tastenabfrage (vgl. auch Beispiel 2 in Pkt. 10.3) für alle 0,5ms vorgesehen.

T1: 1 : 50

PRE1: 1 : 10

Als Pointer für die Ablage und das Zurücksenden der Daten wird Reg. 7 und Reg 8 in folgenden ISR **IR3** verwendet, die angesprungen wird, wenn IRQ3 aufgetreten ist, d.h. das übertragen Byte nach der seriellen Übertragung vollständig im SIO-Eingangsregister angekommen ist.

IR3:	CP 7, #20	Pointer < %20 ?
	JR C, I31	wenn Start-Byte
	CP 7, #30	Pointer > %2F ?
	JR NC, I32	wenn Ende-Byte
	LD @7, %F0	Abspeichern Datenbyte auf die Adresse, auf die der Pointer zeigt
	INC 7	Pointer +1
	IRET	
IR31:	CP %F0, #2A	Startbyte ?
	JR NZ, I33	wenn nicht
	LD 7, #20	den Pointer mit %20 laden
I33:	IRET	
I32:	CLR 7	Pointer auf „0“ setzen
	CP %F0, #1E	Ende-Byte ?
	JR NZ, I33	wenn nicht
	LD 8, #20	Sendepointer mit %20 laden

LD %F0, #%2A Senden Start-Byte
IRET

Ist ein Datenbyte vollständig aus dem SIO-Senderegister versandt worden, tritt IRQ4 auf und die dazugehörige ISR **IR4** wird wirksam.

IR4: **CP 8, #%30** Pointer < %30 ?
 JR NC, I41 wenn nicht
 LD %F0, @8 Senden Datenbyte
 INC 8 Pointer +1
 IRET
I41: **JR NZ, I42** wenn fertig, d.h. Pointer > %30
 LD %F0, #%1E Senden Ende-Byte
 INC 8 Pointer := %31
I42: **IRET**

15. Hinweise zum Einsatz des EMR als Prozessrechner

Obwohl der U88xx-EMR in MOS-Technologie gefertigt wird, ist davon auszugehen, dass mit max. 200mA für die Stromversorgung zu rechnen ist. Verlustärmere CMOS-Varianten sind erst für einen späteren Zeitraum geplant.

Da die interne Arbeitsweise mit dynamischen Speicherzellen erfolgt, ist unbedingt eine Taktung erforderlich. In der Regel sind hierzu Taktfrequenzen in Bereich von 1 ... 8 MHz vorzusehen. Versuche haben gezeigt, dass einzelne Exemplare auch noch bei 1 kHz richtig arbeiten und teilweise bis 15 MHz möglich sind. In jedem Fall gelten jedoch die Angaben in der entsprechenden TGL.

Bei externem Takt ist zu beachten, dass mit einer herkömmlichen TTL-Ansteuerung der vorgeschriebene H-Pegel von $\approx 3,8$ V nicht erreicht wird. Hier ist ein zusätzlicher Pull-Up-Widerstand gegen die Betriebsspannung +5 V vorzusehen.

Infolge der Erkenntnis, dass jede technische Einrichtung unabhängig vom Aufwand und Güte der Bauelemente vom Prinzip her störanfällig ist, müssen insbesondere bei Prozess- Rechnern Vorkehrungen getroffen werden, dass diese weder durch Fehlbedienung, noch durch Bauelemente-Ausfall oder sonstige betriebsmäßige Probleme, wie z.B. Ausfall der Betriebsspannung, gefährliche Fehlzustände einnehmen kann. Bei der Konzipierung einer elektronischen Steuerung sind deshalb folgende Forderungen zu berücksichtigen:

- Beim Einschalten der Anlage dürfen durch das, ggf. auch unterschiedliche Zuschalten der Stromversorgung für Elektronik und die Stellglieder keine gefährlichen Zustände in der Anlage hervorgerufen werden.
- Der Funktionsstart hat durch Einschalten (Schließer) des betreffenden Stromkreises, bzw. in elektronischen Schaltungen durch Setzen eines logischen „1“-Signals zu erfolgen.

Die Halt-Funktion hingegen ist durch Unterbrechen des Stromkreises (Öffner) oder durch Rücksetzen des logischen „1“-Signals zu realisieren. Ein Halt-Befehl muß stets Vorrang vor dem zugeordneten Startbefehl haben, d.h. ein Halt-Befehl muß den Start-Befehl außer Kraft setzen.

- Bei Schlüssen und/oder Unterbrechungen auf Signalleitungen oder solche zur Lastansteuerung, sowohl untereinander, wie auch gegen Bezugs- oder Erdpotential muß verhindert werden, dass sich dadurch gefährliche Auswirkungen ergeben.

Deshalb wird bei Steuerungsausgängen die geschaltete Last einseitig mit dem gemeinsamen niedrigen Potential der Stromversorgungsquelle für die Stellglieder, d.h. in der Regel mit Masse verbunden. Dadurch entstehen die so genannten „Plus-schaltenden“

Ausgänge. Für sicherheitsrelevante Signale ist das auch für Eingangskreise vorgeschrieben. Dadurch entstehen auch dort „Plus-schaltende Eingänge.“

- Sind Gefahren für Personen oder Schäden an der Maschine nicht auszuschließen, müssen in jedem Fall unabhängig von der eigentlichen Steuerung, also insbesondere von der Elektronik nicht beeinflussbar wirkende Not-Aus-Schalteinrichtungen vorgesehen werden.

15.1 Eingangsbeschaltung

In rauher Industrie-Umgebung sind Informationsleitungen mit teilweise hohen Störsignalen „verseucht“. Um die empfindlichen Eingänge des EMR zu schützen, sind entsprechende Eingangsschutzschaltungen, z.B. nach Bild 22 vorzusehen. Bei kontaktbetätigter Eingangsschaltung ist nach dem o.g. die „Plus-schaltende“ Variante zu bevorzugen.

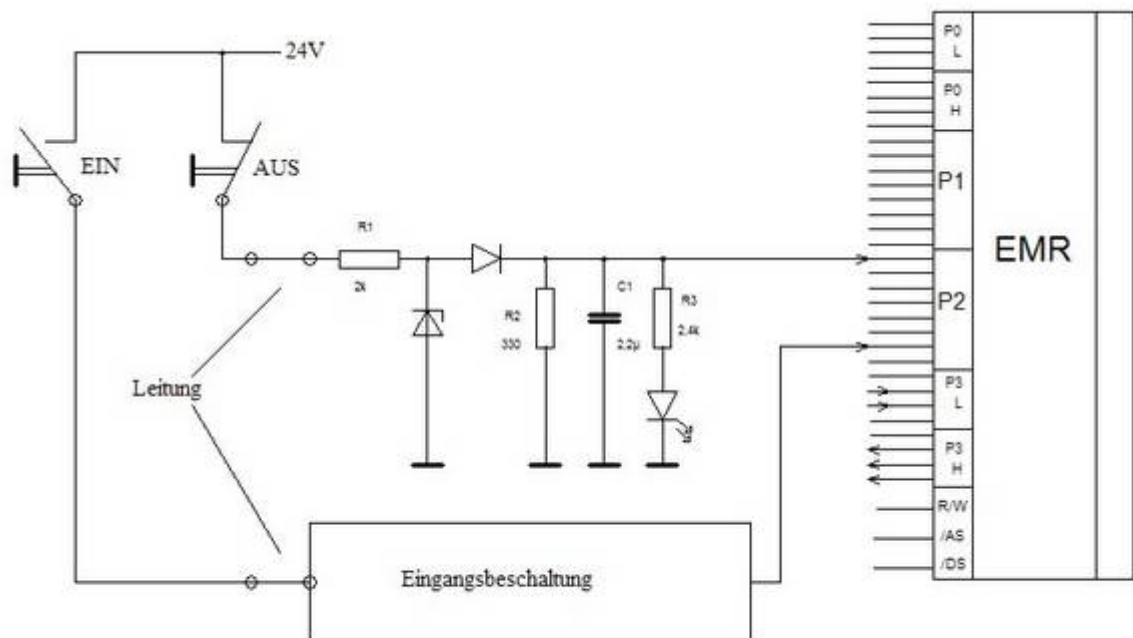


Bild 22: Eingangsbeschaltung für Tasten

Die Eingangsbeschaltung ist entsprechend der in der Industrie häufig vorkommenden 24V-Schaltspannung ausgeführt. Die Z-Diode am Eingang verhindert im Fehlerfall unzulässig hohe Eingangsspannung am EMR-Eingang. Möglicherweise sind noch ergänzende LC-Filterschaltungen (Tiefpass) notwendig. Die LED signalisiert den momentan logischen Zustand am Eingang: „1“ oder „0“.

Sollen Eingänge als Interrupt-Quellen dienen, ist unbedingt ein Schmitt-Trigger vorzuschalten (z.B. DL014D oder V4093D).

Werden mit den Eingängen die Zustände mechanischer Kontakte abgefragt, so z.B. Tasten, handbetätigte oder auch Endlagenschalter, so empfiehlt es sich, programmtechnisch eine „Entprellung“ vorzusehen.

MA1: LD R6, R2 ; der Inhalt von Port 2 wird in R6 geladen

NOP

LD R7, R2 ; das gleiche nochmals in R7

CP R6, R7

JRNZ, MA1 ; nur wenn beides mal der gleiche Inhalt in R2 enthalten
; war, ist davon auszugehen, das keine Prell-Jitter mehr
; vorhanden sind und das Programm weitergehen kann

Mitunter ist es notwendig, Eingangssignale aus dem 220V-Wechselspannungs-Niveau zu verarbeiten. Dann ist jedoch eine galvanischer Trennung unumgänglich. Eine entsprechende Schaltungsvariante zeigt Bild 23.

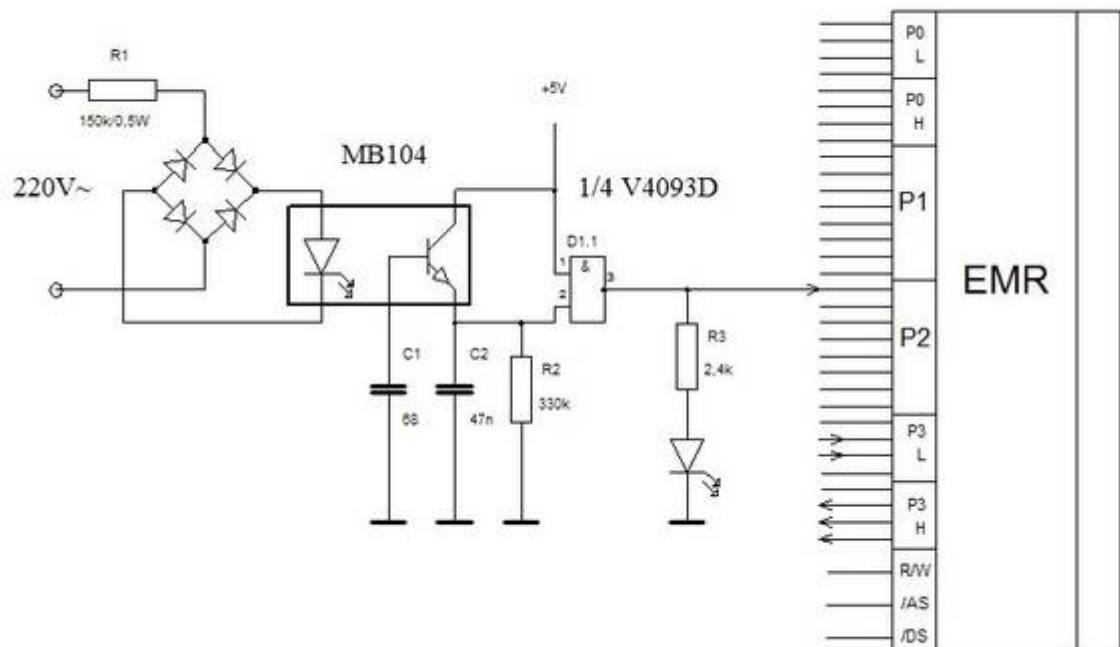


Bild 23: Eingangsbeschaltung für 220V~

Mit dem Optokoppler MB104 wird die galvanische Trennung realisiert.

Die LED signalisiert den momentan logischen Zustand am Eingang: „1“ oder „0“.

15.2 Ausgangsbeschaltung

Die EMR-Ausgänge haben einfaches TTL-Verhalten, d.h. bei

logisch 0 0 bis 0,4 V bei 2mA

logisch 1 2,4 bis 5V bei max. 0,25mA

In der Regel müssen deshalb Treiberschaltungen vorgesehen werden. Eine Schaltungsvariante nach dem oben gesagten zeigt Bild 24.

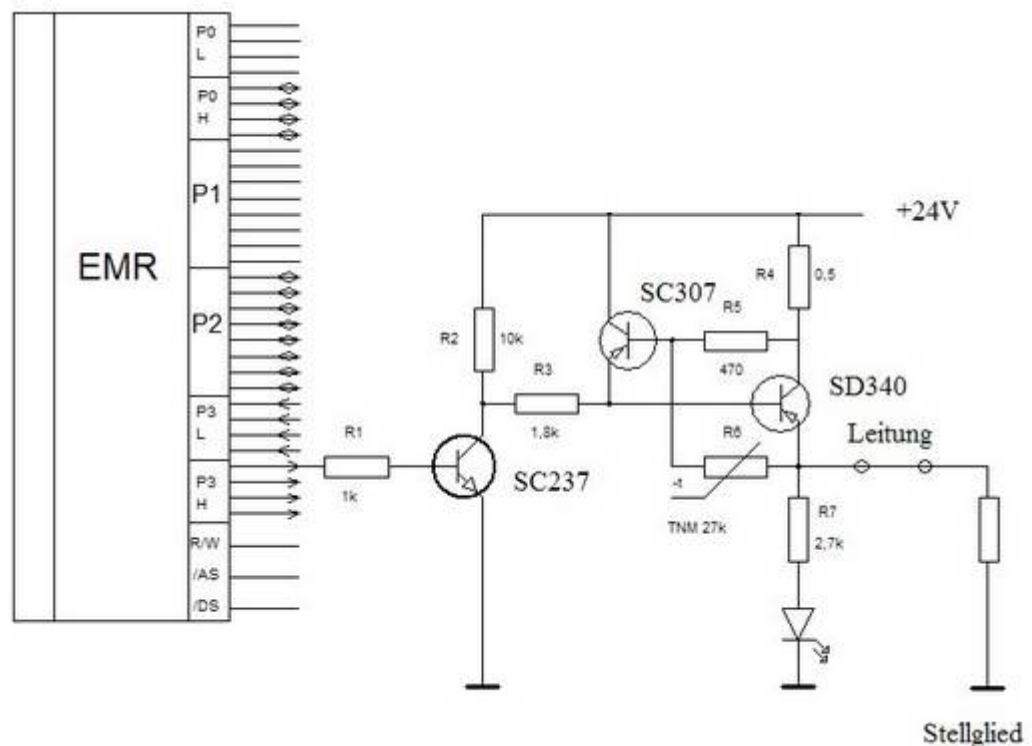


Bild 24 Ausgangsbeschaltung „Plus-schaltend“

Die 1A-Ausgangstreiberstufe ist kurzschlußsicher und gegen Übertemperatur geschützt.

Die LED signalisiert den momentan logischen Zustand am Ausgang: „1“ oder „0“.

Wird der logische Ausgangszustand wieder zurück auf einen EMR-Eingang geführt, kann man eine einfache Eigendiagnose-Schaltung entwickeln. Wird eine nicht vorherbestimmte Situation am betreffenden Ausgang erkannt, kann dadurch mittels einer ISR Alarm ausgelöst, bzw. anderweitige Maßnahmen ergriffen werden, die Mensch und Maschine ggf. vor weiteren Schäden schützt.

15.3 RESET-Schaltung

Um die negative Auswirkungen kurzzeitiger Schwankungen oder Unterbrechungen der

Stromversorgung zu vermeiden, werden mehr oder weniger intelligente RESET-Schaltungen angewendet. Eine sehr einfache Möglichkeit zeigt Bild 25.

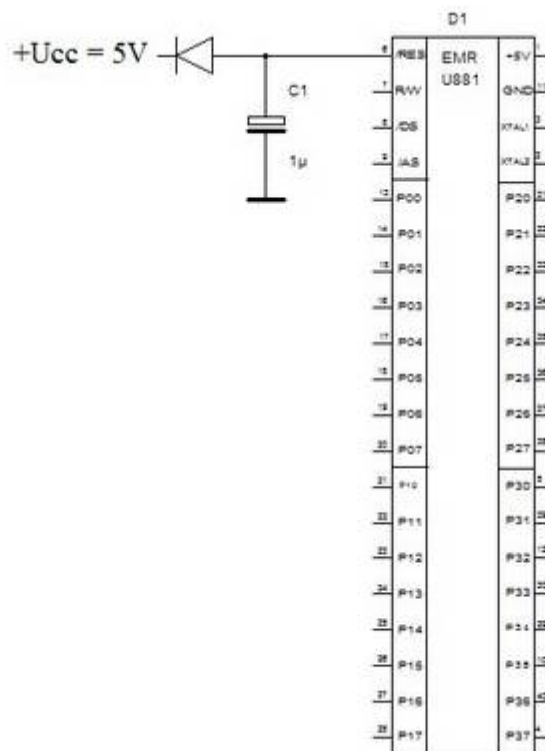


Bild 25: Einfache RESET-Schaltung

Durch die Diode eine schnelle Entladung des Kondensators bei Spannungsausfällen gewährleistet und damit gesichert, dass auch bei kurzzeitigen Versorgungsspannungsausfällen oder Spannungseinbrüchen auf der Betriebsspannungsschiene ein L-Impuls am Anschluß /RESET ausgelöst wird. Ein Pull-Up-Widerstand ist nicht notwendig, da dieser intern im EMR mit etwa 100kOhm vorhanden ist.

Zu beachten ist jedoch, dass sich nach jedem RESET (mind. 50ms, >18 Taktzyklen auf Low-Potential halten) folgender Anfangszustand des EMR einstellt:

- PC:= %0C,
- globale Interrupt-Sperre,
- Löschen aller IRQ,
- inaktivieren von Timern, SIO und Handshake,
- alle Anschlüsse sind auf Eingang programmiert.

Dürfen bei einem Ausfall der Betriebsspannung wichtige Daten nicht verloren gehen, so sind entsprechende Maßnahmen zu treffen. Der Spannungszusammenbruch muß rechtzeitig

erkannt und sofort eine entsprechende Rettungsroutine angesprungen werden, die die Daten z.B. in einen batteriegestützten CMOS-RAM zwischenspeichert. Dabei muß der EMR noch seine vollständige Funktion behalten, d.h. entsprechend große Stützkondensatoren für die Betriebsspannung sind vorzusehen. Für kurze Zeit kann auch die Registerbank zum Datenerhalt eingesetzt werden, wenn der EMR (Power-Down-Version) durch Absinken der Betriebsspannung in den „Schlafzustand“ versetzt wird.

Eine aufwendige, aber sehr umfassende Sicherheitsschaltung ist in [2] angegeben, wobei hier vorausschauend der korrekte Verlauf der Halbwellen der Netzspannung ausgewertet wird. Fehlen solche, wird ein entsprechender Interrupt zur Datenrettung ausgelöst, bevor der EMR in den „Schlafzustand“ durch das Abfallen der Betriebsspannung versetzt wird. Außerdem ist gleichzeitig noch eine so genannte „Watch-Dog“-Schaltung implementiert, die verhindert, dass der EMR in einer Schleife „festhängt“. Das wird durch regelmäßiges Abarbeiten einer bestimmten Routine erreicht, durch die - wenn sie nicht mehr abgearbeitet wird – zwangsweise einen RESET initiiert wird.

16. **Programmiertechnologie**

16.1 **Allgemeines zur EMR-Programmierung**

Evaluation-Boards oder ähnliche Entwicklungshilfen werden leider vom Hersteller (VEB MME) nicht bereitgestellt.

Zum effizienten Testen komplexer EMR-Systeme werden jedoch zunehmend so genannte „In-Circuit-Emulatoren“ eingesetzt. Vom IfAM Erfurt und auch von der TU-Ilmenau wird ein solches System (Hardware/Software) zur Nachnutzung angeboten.

Eine Alternative hierzu bietet auch ein so genannter EPROM-Simulator.

Hierbei - und wie sonst auch – muß das Programm zunächst auf einem Host-Rechner entwickelt werden. Als Host-Rechner kommen BC5120, PC1715, z.T. auch schon die 16-Bit-Rechentechnik und auch der MC80 in Frage. Sogar mit dem KC85/3 lassen sich EMR-Programme entwickeln.

(Anm.: siehe Homepage des Autors)

Folgende Schritte sind für eine erfolgreiche Programm-Entwicklung notwendig:

- Eingeben des Assembler-Quell-Programms mittels eines Editors
- Übersetzen der Quelle mittels eines Cross-Assemblers
- Adressberechnung mittels eines Linkers
- den erzeugten Programm-Code mittels eines EPROMMERS in den/die EPROMs brennen

Als Editor-Software wird beim BC/PC oft das Programm TP benutzt und als Cross-Assembler steht „AS8“ zur Verfügung.

Allen diesen Hilfsmittel haftet der prinzipielle Mangel an unzureichenden Testmöglichkeiten an, insbesondere wenn Echtzeitaufgaben mit Interrupt-Unterstützung zu lösen sind.

Aus diesem Grund wurde an der IHS Wartenberg eine autonome Entwicklungshilfe geschaffen, die auch auf der Basis eines Host-Rechners, aber mit spezieller Hardware-Erweiterung eine Kopplung mit dem zu testenden Prozessrechner ermöglicht.

Die Hardware-Erweiterung hat zusätzlich Monitor- und Tastaturanschluß.

Es wird ein Editor in Maschinensprache und symbolischer Adressierung verwendet.

Ein Linker bindet die einzelnen Programm-Moduln mit entsprechender Adressberechnung zusammen, so dass dann mit einer Testschaltung für Schrittbetrieb und Setzen von Breakpoints das Programm ausgiebig getestet werden kann.

16.2 Methoden der Programmentwicklung

Bei der Programmentwicklung umfangreicher Software (bzw. neudeutsch „Firmware“, weil das Programm fest in Halbleiterspeichern abgelegt ist), sollte folgende Aufgabenteilung vorgenommen werden:

- Bearbeitungsroutinen, die ständig im Vordergrund arbeiten,
- regelmäßige Interrupt-Service-Routinen, z.B. Timer usw.
- fremdgesteuerte Interrupt-Service-Routinen, z.B. Peripherieabfragen

Daraus lässt sich dann folgender Programmierablauf ableiten:

- (1) Entwicklung der einzelnen Programmteile, z.B. als Unterprogramme (UP),
- (2) Zusammenstellung des „Betriebssystems“,
d.h. Zusammenfügen der einzelnen UP zu einem durchgängigem Ablauf,
ggf. auch durch Überarbeitung des Gesamtkonzepts (PAP),
- (3) Analyse der Kapazitätsauslastung des EMR mit dem Ziel, Möglichkeiten zu suchen, die Hardware-gelöste Eigenschaften in den Bereich Software-Lösungen zu verschieben, da dies in der Regel preisgünstiger ist.

Anhang

Befehlsliste (aus Kieser/Bankel)

Befehlsliste - sortiert nach OP-Code

OP-Code-Kürzel:

R = Registeradresse 0...127, 240...255

H = 2x Halbbyte (H = 0...15, L = 0...15) Arbeitsregisteradresse

n = numerischer Wert 0...255

d = Displacement -127...+128 (Zweierkomplement)

OP-Code	Mnemonic
---------	----------

00 R	DEC R
01 R	DEC @R
02 H	ADD r, r
03 H	ADD r, @r
04 RR	ADD R, R
05 RR	ADD R, @R
06 Rn	ADD R, n
07 Rn	ADD @R, n
08 R	LD r0, R
09 R	LD R, r0
0A d	DJNZ r0, d

0C n	LD r0, n
0D nn	JP nn
0E	INC r0

10 R	RLC R
11 R	RLC @R
12 H	ADC r, r
13 H	ADC r, @r
14 RR	ADC R, R
15 RR	ADC R, @R
16 Rn	ADC R, n
17 Rn	ADC @R, n
18 R	LD r1, R
19 R	LD R, r1
1A d	DJNZ r1, d
1B d	JR LT, d
1C n	LD r1, n
1D nn	JP LT, nn
1E	INC r1

20 R	INC R
21 R	INC @R
22 H	SUB r, r
23 H	SUB r, @r
24 RR	SUB R, R
25 RR	SUB R, @R
26 Rn	SUB R, n
27 Rn	SUB @R, n
28 R	LD r2, R
29 R	LD R, r2
2A d	DJNZ r2, d
2B d	JR LE, d
2C n	LD r2, n
2D nn	JP LE, nn
2E	INC r2

30 R	JP @RR	
31 n	SRP n	
32 H	SBC r, r	
33 H	SBC r, @r	
34 RR	SBC R, R	
35 RR	SBC R, @R	
36 Rn	SBC R, n	
37 Rn	SBC @R, n	
38 R	LD r3, R	
39 R	LD R, r3	
3A d	DJNZ r3, d	
3B d	JR ULE, d	
3C n	LD r3, n	
3D nn	JP ULE, nn	
3E	INC r3	
40 R	DA R	
41 R	DA @R	
42 H	OR r, r	
43 H	OR r, @r	
44 RR	OR R, R	
45 RR	OR R, @R	
46 Rn	OR R, n	
47 Rn	OR @R, n	
48 R	LD r4, R	
49 R	LD R, r4	
4A d	DJNZ r4, d	
4B d	JR OV, d	
4C n	LD r4, n	
4D nn	JP OV, nn	
4E	INC r4	
50 R	POP R	
51 R	POP @R	
52 H	AND r, r	
53 H	AND r, @r	
54 RR	AND R, R	
55 RR	AND R, @R	
56 Rn	AND R, n	
57 Rn	AND @R, n	
58 R	LD r5, R	
59 R	LD R, r5	
5A d	DJNZ r5, d	
5B d	JR MI, d	
5C n	LD r5, n	
5D nn	JP MI, nn	
5E	INC r5	
60 R	COM R	
61 R	COM @R	
62 H	TCM r, r	
63 H	TCM r, @r	
64 RR	TCM R, R	
65 RR	TCM R, @R	
66 Rn	TCM R, n	
67 Rn	TCM @R, n	
68 R	LD r6, R	
69 R	LD R, r6	
6A d	DJNZ r6, d	
6B d	JR Z, d	(= JR EQ, d)
6C n	LD r6, n	

6D nn	JP z, nn	(= JP EQ, nn)
6E	INC r6	
70 R	PUSH R	
71 R	PUSH @R	
72 H	TM r, r	
73 H	TM r, @r	
74 RR	TM R, R	
75 RR	TM R, @R	
76 Rn	TM R, n	
77 Rn	TM @R, n	
78 R	LD r7, R	
79 R	LD R, r7	
7A d	DJNZ r7, d	
7B d	JR C, d	(= JR ULT, d)
7C n	LD r7, n	
7D nn	JP C, nn	(= JP ULT, nn)
7E	INC r7	
80 R	DECW RR	
81 R	DECW @RR	
82 H	LDE, r, @rr	
83 H	LDEI @r, @rr	
88 R	LD r8, R	
89 R	LD R, r8	
8A d	DJNZ r8, d	
8B d	JR d	
8C n	LD r8, n	
8D nn	JP nn	
8E	INC r8	
8F	DI	
90 R	RL R	
91 R	RL @R	
92 H	LDE @rr, r	
93 H	LDEI @rr, @r	
98 R	LD r9, R	
99 R	LD R, r9	
9A d	DJNZ r9, d	
9B d	JR GE, d	
9C n	LD r9, n	
9D nn	JP GE, nn	
9F	EI	
A0 R	INCW RR	
A1 R	INCW @RR	
A2 H	CP r, r	
A3 H	CP r, @r	
A4 RR	CP R, R	
A5 RR	CP R, @R	
A6 Rn	CP R, n	
A7 Rn	CP @R, n	
A8 R	LD rA, R	
A9 R	LD R, rA	
AA d	DJNZ rA, d	
AB d	JR GT, d	
AC n	LD rA, n	
AD nn	JP GT, nn	
AE	INC rA	
AF	RET	
B0 R	CLR R	
B1 R	CLR @R	

B2 H	XOR r, r	
B3 H	XOR r, @r	
B4 RR	XOR R, R	
B5 RR	XOR R, @R	
B6 Rn	XOR R, n	
B7 Rn	XOR @R, n	
B8 R	LD rB, R	
B9 R	LD R, rB	
BA d	DJNZ rB, d	
BB d	JR UGT, d	
BC n	LD rB, n	
BD nn	JP UGT, nn	
BE	INC rB	
BF	IRET	
C0 R	RRC R	
C1 R	RRC @R	
C2 H	LDC r, @rr	
C3 H	LDCI @rr, @r	
C7 HR	LD r, R+(r)	
C8 R	LD rC, R	
C9 R	LD R, rC	
CA d	DJNZ rC, d	
CB d	JR NOV, d	
CC n	LD rC, n	
CD nn	JP NOV, nn	
CE	INC rC	
CF	RCF	
D0 R	SRA R	
D1 R	SRA @R	
D2 H	LDC @rr, r	
D3 H	LDCI @rr, @r	
D4 R	CALL @RR	
D6 nn	CALL nn	
D7 HR	LD R+(r), r	
D8 R	LD rD, R	
D9 R	LD R, rD	
DA d	DJNZ rD, d	
DB d	JR PL, d	
DC n	LD rD, n	
DD nn	JP PL, nn	
DE	INC rD	
DF	SCF	
E0 R	RR R	
E1 R	RR @R	
E3 H	LD r, @r	
E4 R	LD R, R	
E5 R	LD R, @R	
E6 Rn	LD R, n	
E7 Rn	LD @R, n	
E8 R	LD rE, R	
E9 R	LD R, rE	
EA d	DJNZ rE, d	
EB d	JR NZ, d	(= JR NE, d)
EC n	LD rE, n	
ED nn	JP NZ, nn	(= JP NE, nn)
EE	INC rE	
EF	CCF	
F0 R	SWAP R	
F1 r	SWAP @R	

F3 H	LD @r, r	
F5 RR	LD @R, R	
F8 R	LD rF, R	
F9 R	LD R, rF	
FA d	DJNZ rF, d	
FB d	JR NC, d	(= JR UGE, d)
FC n	LD rF, n	
FD nn	JP NC, nn	(= JP UGT, nn)
FE	INC rF	
FF	NOP	

Literatur

(Anm. des Autors: Die Literatur-Zusammenstellung wurde erst 2007 erstellt)

- [1] Bennewitz, W., Podszuweit, H.: Programmierung von Einchipmikrorechnern, Reihe Automatisierungstechnik 215, VEB Verlag Technik, 1985, 68 Seiten
- [2] Heiko Kieser, Mario Bankel: Einchipmikrorechner, VEB Verlag Technik, 1986/1989
- [3] Dr. sc. tech. Klaus Franke, Dipl.-Ing. Andreas Leichsenring: Entwicklungsmodul für Einchip-Mikrorechner, rfe 1984, H1, S8-10
- [4] Dr.-Ing. Helmut Sterl, Dipl.-Ing. Dankwart Kurze: Bildschirmsteuerung mit Einchip-Mikrorechner, rfe 1984, H1, S10-13
- [5] Dr.-Ing. Steffen Burkhardt, Dipl.-Ing. Günter Uhlig: Einchip-Mikrorechner in Haushaltgeräten, rfe 1984, H1, S14-16
- [6] Dipl.-Ing. Uwe Hübner: Einchip-Mikrorechner im Telefon, rfe 1984, H1, S16-18
- [7] Dr.-Ing. Ulrich Neubert: Kfz-Zündsystem mit Einchip-Mikrorechner, rfe 1984, H1, S19-21
- [8] Dipl.-Ing. Joachim Müller, Dipl.-Ing. Matthias Thein: Entwicklungs- und Testmodul für Einchipmikrorechner-Programme, rfe 1983, H11, S721-723
- [9] Dipl.-Ing. Boytscho Nedelelev: Zum Einchip-Mikrorechner, rfe 1985, H1, S8-12
- [10] Dipl.-Ing. Mario Bankel: Einchip-Mikrorechner U881, U882, und U883, rfe 1985, H2, S81-84
- [11] Dipl.-Math. Siegmund Müller: Einchip-Mikrorechner U883D interpretiert Tiny-Basic, rfe 1985, H 3, S143-144,
- [12] Dr.-Ing. Wolfgang Graubner, Ing. Heinz Klinger, Dipl.-Ing. Dietmar Leuthold, Dipl.-Ing. Siegfried Riehle: Programmentwicklung für U882D, rfe 1985, H3, S145-146
- [13] Dr.-Ing. Wolfgang Bobe: Zeitzeichengesteuerte Uhr mit Einchip-Mikrorechner, rfe 1986, H4, S215-217
- [14] Lutz Schneegeß, Bernd Märtens: Digitales Abstimme- und Anzeigesystem mit Einchip-Mikrorechner, rfe 1986, H5, S311-313
- [15] Dipl.-Ing. Manfred Lohöffner: Tabellarischer Befehlssatz der Einchip-Mikrorechners U881D und U882D, rfe 1986, H7, S422
- [16] Dr.-Ing. Werner Kämpf: Einkartenrechner mit Einchip-Mikrorechner U884M, rfe 1985, H11, S690-692
- [17] Dr. rer. nat. Günter Neidhold: Didaktisch gestalteter Arbeitsplatz für Einchip-Mikrorechner, rfe 1986, H2, S110-113
- [18] Dipl.-Ing. Jürgen Delfs: Adapterplatine für U881 und U882, rfe 1986, H2, S13-114
- [19] Dr.-Ing. Peter Weber, Dr.-Ing. Martin Kunert, Dipl.-Ing. Wilhelm Tramert, Dr. sc. nat. Uwe Küster, Dr. rer. Nat. Christian Glier: Einchip-Mikrorechner U882 mit Display und Dezimaltastatur, rfe 1986, H11, S693-695

- [20] Dr.-Ing. Klaus Kabitzsch, Dipl.-Ing. Frank Weber, Dipl.-Ing. Günther Weimann: Programmierbare Zeitschaltanordnung mit U882D, rfe 1986, H11, S695-697
- [21] Dipl.-Ing. G. Thierfelder, Dipl.-Ing. S. Apel, Dr. W. Klein, Dipl.-Ing. H.-H. Körner, Dipl.-Ing. G. Krause, Dipl.-Ing. B. Uhlmann, Dipl.-Ing. T. Schäfer, Dipl.-Ing. H. Thieme, Dipl.-Ing. M. Feske, Math. M. Pröhl: 4-bit-Einchip-Mikrorechner U8047P mit RAM-Erweiterung U8246P, rfe 1987, H3, S144-149; H4, S258-262
- [22] Dipl.-Ing. Helmut Bresch, Dr.-Ing. Wolfgang Oberst: EPROM-Simulator, rfe 1987, H3, S188
- [23] Werner Schmidt, Angelika Schmidt: Experimentierschaltung mit U882, rfe 1987, H4, S255-256; H8, S502
- [24] Dr.-Ing. Martin Seitz: Interrupterweiterung bei Einchip-Mikrorechnern, rfe 1987, H5, S325-328
- [25] Dipl.-Ing. Hans-Jörg Schwarz: Familie der Einchip-Mikrorechner, rfe 1987, H9, S555-556
- [26] Prof. Dr. sc. techn. Jürgen Waldmann, Dipl.-Ing. Thomas Hoffmann, Ing. Bernhard Masche: Zeitsteuermodul mit UB8810D, rfe 1987, H9, S557-559
- [27] Frank Möckel: Testspeicher für UB8820M, rfe 1987, H9, S559-560
- [28] Dipl.-Ing. Herbert Hemke: Einchip-Mikrorechner U882M und U884M mit externem Kellerspeicher, rfe 1987, H9, S561-562
- [29] Lutz Schneegass, Bernd Märtens: Universelles digitales Abstim- und Anzeigesystem mit Einchip-Mikrorechner, rfe 1988, H1, S20-23
- [30] Dipl.-Ing. Axel Kropf: Dual-Port-RAM-Erweiterung für Einchip-Mikrorechner, rfe 1988, H1, S53-54
- [31] Dipl.-Ing. Karl-Heinz Kuschke, Dipl.-Ing. Lutz Landrock: U884-Entwicklungsmodul, rfe 1988, H4, S261-262
- [32] Dipl.-Ing. Detlef Eichelbaum, Dipl.-Ing. Matthias Rödiger: Aktive Tastatur mit Einchip-Mikrorechner U8820, rfe 1988, H6, S391-392
- [33] Dipl.-Ing. Ulrich Buhss: Multiplikationshardware für Einchip-Mikrorechner U882, rfe 1988, H6, S393-394
- [34] Dr. rer. nat. Ulrich Schmucker, Dipl.-Ing. Ingo Templin: Anlauf- und Wachsaltung für U886, rfe 1988, H11, S732
- [35] Jens Gollasch: Rhythmuscomputer mit Einchip-Mikrorechner, rfe 1989, H5, S313-318
- [36] Dipl.-Ing. Axel Kropf: Betriebsspannungskontrolle für Einchip-Mikrorechner, rfe 1989, H8, S498-499
- [37] Gerd Schachtschneider: Quarzgesteuerte Digitaluhr mit Einchip-Mikrorechner, rfe 1989, H9, S590-594
- [38] Dipl.-Ing. Olaf Skerl, Dipl.-Ing. Wolfram Schmidt: Dynamische Speicher an Einchip-Mikrorechnern U8820 und U8840, rfe 1989, H10, S635-636
- [39] Dipl.-Ing. Matthias Dobesch: Universelles Siebensegment-Multiplexinterface (für UB88xx), rfe 1989, H11, S735-736

- [40] Dipl.-Ing. Volkmar Rössler, Holgr Mädlr: Intelligenter Sensor mit U8820,
rfe 1990, H2, S119-121
- [41] Dipl.-Ing. Martin Orlick: Einfaches Hilfsmittel zur Programmtestung für U882 und U884,
rfe 1990, H3, S178-181
- [42] Dr.-Ing. Ulrich Müller: Funkuhr für Automatisierungssysteme (mit U882),
rfe 1990, H5, S296-298
- [43] Dr.-Ing. Olaf Hagenbruch, Dipl.-Ing. Lutz Voigt: EMR-Baugruppe mit UB8830D,
rfe 1990, H6, S392- 393
- [44] Dipl.-Ing. Frank Ryll: Einchipmikrorechner UB883 mit 4-Kbyte-Speicher,
rfe 1991, H5, S269-270
- [45] Dipl.-Ing. Klaus Lehmann: Modul zur Ansteuerung von Schrittmotoren (mit U8840),
rfe 1991, H5, S271-275
- [46] Dipl.-Phys. Thomas Bröcker, Dipl.-Ing. Matthias Sturm: Parallele Meßwerterfassung mit EMR und PC,
rfe 1991, H9, S528-530
- [47] Ing. P.-J. Theil: Die Multifunktions-Bildschirmkarte (mit EMR),
funkamateure 1990, H11, S540-543; H12, S589-590; H1, S17-18; H2, S75-76
- [48] Dokumentation Einkartenrechner EKR,
IfAM Erfurt, 23.10.1987, Dr. Kämpf
- [49] Dokumentation EPROM-Simulator, TH „Otto von Guericke“ Magdeburg,
Sektion Technische Kybernetik und Elektrotechnik, Dr.-Ing. Oberst, 05.01.1987

Sonstiges

Wichtige Namen und Tel.-Nr. an der IHS Wartenberg:

Einwahl (Berlin) 4815.0

Referent:	Dr.-Ing. Hoyer,	Wissenschaftl. Gerätebau,	273
	Dipl.-Ing. Dornheim,	Automatisierungstechnik,	355
	Herr Reuter,	BfSN (Nachnutzung),	342
	Frau Schönfeld,	Sekretariat,	353