Laborpraktikum Mikrorechentechnik

Mikrocontroller
SAB 80C517A/80C537

Inhalt:

Vorbereitungsaufgaben:

Diese Praktikumsanleitung ist zur Vorbereitung auf das Laborpraktikum durchzuarbeiten. Damit soll erreicht werden, dass der Praktizierende die angegebenen Beispielprogramme und das Rahmenprogramm erläutern kann, um sie im Praktikum als Grundlage für die zu erarbeitenden Programmmodule benutzen zu können.

Literatur:

/1/ Siemens AG:
SAB 80C517/80C537 8-Bit CMOS Single Chip Mikrocontroller
User´s Manual.
Siemens AG, Bereich Halbleiter, Marketing - Kommunikation, München, 1994

/2/ Keil Elektronik GmbH: C 51 Professional Developers Kit
- High Performance Development Tools for the 8051 Family.
Keil Elektronik GmbH, Grasbrunn b. München, 1994

/3/ Roth, A.: Das Mikrocontroller - Kochbuch.
IWT - Verlag, Vaterstetten b. München, 1992

/4/ Walter, J.: Mikrocomputertechnik mit der 8051 - Controller- Familie.
Springer Verlag, Berlin, Heidelberg, New York, 1996

/5/ Homepage des Labors Mikrorechentechnik:
http://www.htw-dresden.de/fakultaet-elektrotechnik/labore/mikrorechentechnik.html

Blockstruktur des Mikrocontrollers

Der Mikrocontroller SAB80C517 hat die im Bild1 dargestellte Blockstruktur. Daran ist ersichtlich, welche Prozessor-, E/A- und Überwachungseinheiten auf dem Chip integriert sind. Den Chip kann man im PLCC-84 - und P-MQFP-100 - Gehäuse in der Zielapplikation einsetzen.

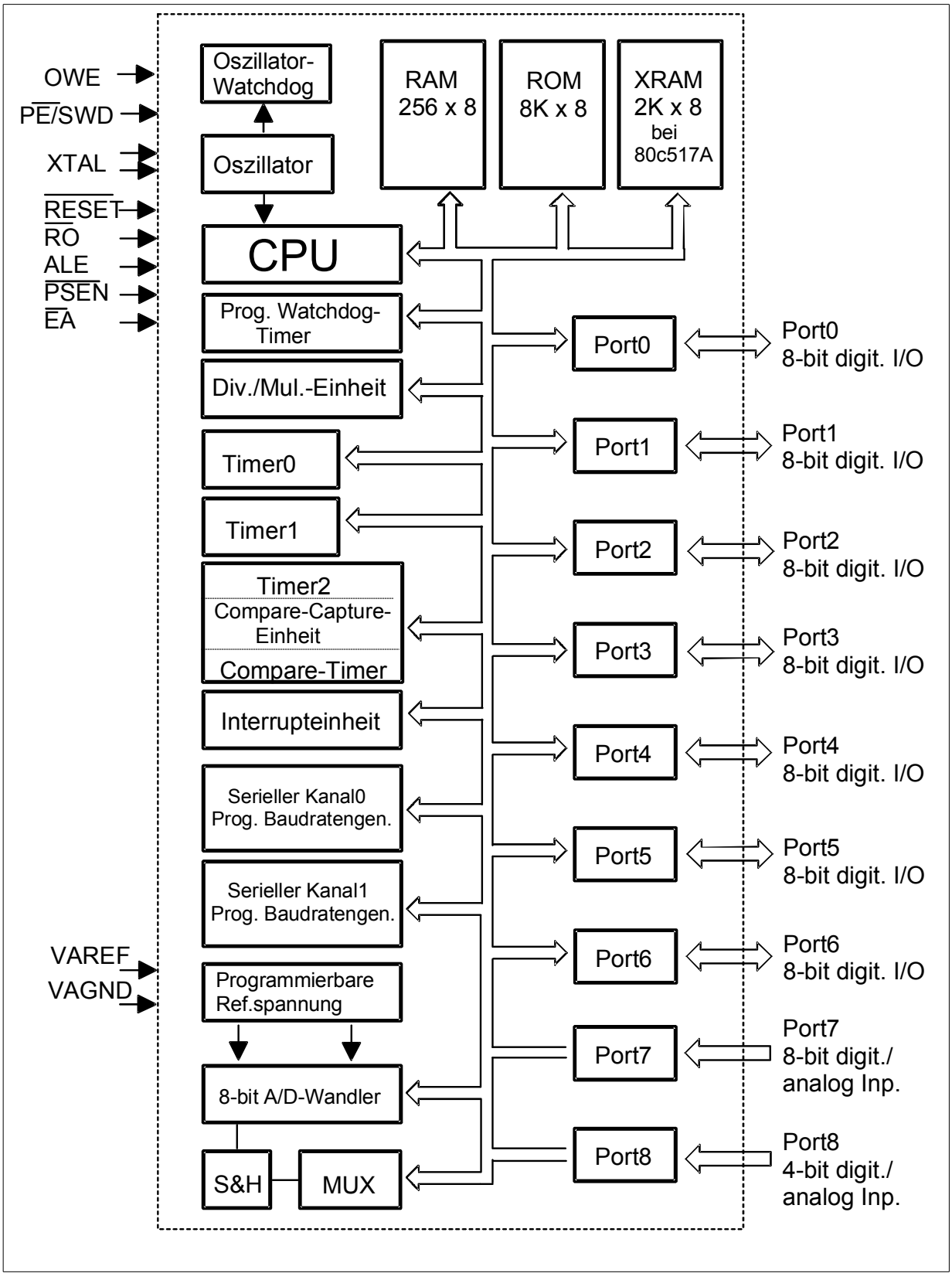

Bild1: Blockstruktur des µC SAB80C517

Speicherorganisation

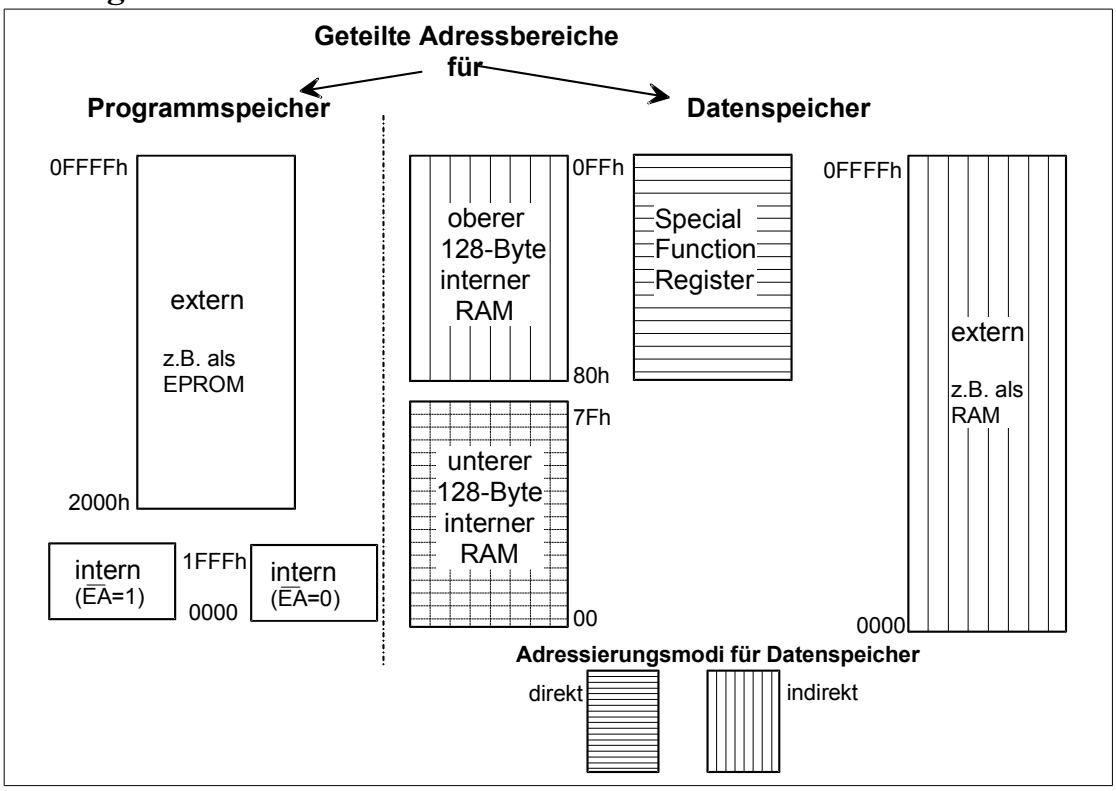

Bild2: Speicherorganisation des µC SAB80C517

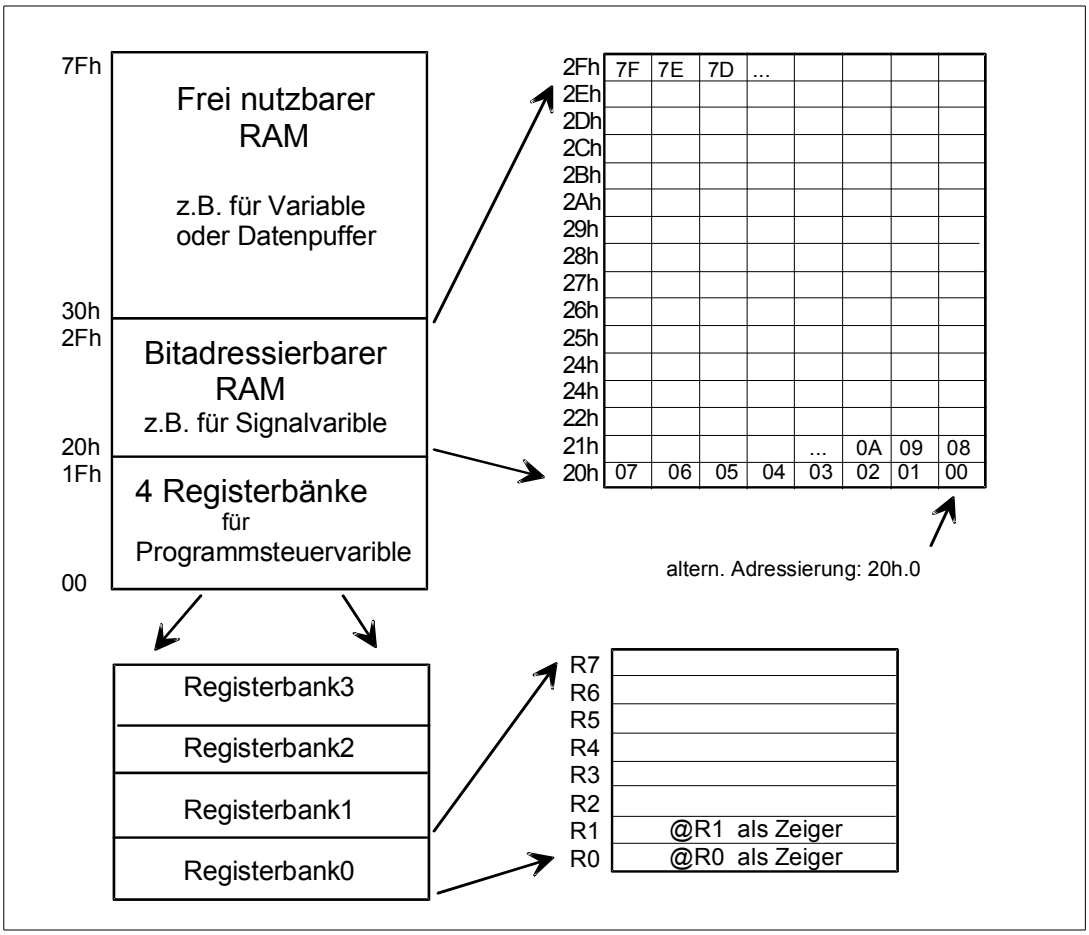

Bild3: Aufbau des unteren Teiles des internen Datenspeicherbereiches

Special Function Registers des SAB 80C517 (SFR)

Block	Symbol	Name	Adresse	Inhalt nach RESET
CPU	**ACC**	**Accumulator**	**$0E0_H$**[1]	**00_H**
	B	**B-Register**	**$0F0_H$**[1]	**00_H**
	DPH	Data Pointer, High Byte	83_H	00_H
	DPL	Data Pointer, Low Byte	82_H	00_H
	DPSEL	Data Pionter Select Register	92_H	$XXXX.X000_B$[3]
	PSW	**Program Status Word Register**	**$0D0_H$**[1]	**00_H**
	SP	Stack Pointer	81_H	07_H
Interrupt System	**IEN0**	**Interrupt Enable Register 0**	**$0A8_H$**[1]	**00_H**
	CTCON[2]	Comp.Timer Control Register	$0E1_H$	$0XXX.0000_B$[3]
	IEN1	**Interrupt Enable Register 1**	**$0B8_H$**[1]	**00_H**
	IEN2	Interrupt Enable Register 2	$9A_H$	$XXXX.00X0_B$[3]
	IP0	Interrupt Priority Register 0	$0A9_H$	00_H
	IP1	Interrupt Priority Register 1	$0B9_H$	$XX00.0000_B$[3]
	IRCON	**Interrupt Request Control Register**	**$0C0_H$**[1]	**00_H**
	TCON[2]	**Timer Control Register**	**88_H**[1]	**00_H**
	T2CON[2]	**Timer 2 Control Register**	**$0C8_H$**[1]	**00_H**
E/A-Ports	**P0**	**Port 0**	**80_H**[1]	**FF_H**
	P1	**Port 1**	**90_H**[1]	**FF_H**
	P2	**Port 2**	**$0A0_H$**[1]	**FF_H**
	P3	**Port 3**	**$0B0_H$**[1]	**FF_H**
	P4	**Port 4**	**$0E8_H$**[1]	**FF_H**
	P5	**Port 5**	**$0F8_H$**[1]	**FF_H**
	P6	Port 6	$0FA_H$	FF_H
	P7	Port 7, Analog/Digital Input	$0DB_H$	XX_H[3]
	P8	Port 8 Analog/Digital Input, 4 Bit	$0DD_H$	XX_H[3]
Timer0/ Timer 1	**TCON**	**Timer Control Register**	**88_H**[1]	**00_H**
	TH0	Timer 0, High Byte	$8C_H$	00_H
	TH1	Timer 1, High Byte	$8D_H$	00_H
	TL0	Timer 0, low Byte	$8A_H$	00_H
	TL1	Timer 1, low Byte	$8B_H$	00_H
	TMOD	Timer Mode Register	89_H	00_H
A/D-Converter	**ADCON0**	**A/D Converter Control Register 0**	**$0D8_H$**[1]	**00_H**
	ADCON1	A/D Converter Control Register 1	$0DC_H$	$XXXX.0000_B$[3]
	ADDAT	A/D Converter Data Register	$0D9_H$	00_H
	DAPR	D/A Converter Program Register	$0DA_H$	00_H
Pow.Save	PCON	Power Control Register	87_H	00_H
MUL/DIV Unit	ARCON	Arithmetic Control Register	$0EF_H$	$0XXX.XXXX_B$[3]
	MD0	Multiplication/Division Register 0	$0E9_H$	XX_H[3]
	MD1	Multiplication/Division Register 1	$0EA_H$	XX_H[3]
	MD2	Multiplication/Division Register 2	$0EB_H$	XX_H[3]
	MD3	Multiplication/Division Register 3	$0EC_H$	XX_H[3]
	MD4	Multiplication/Division Register 4	$0ED_H$	XX_H[3]
	MD5	Multiplication/Division Register 5	$0EE_H$	XX_H[3]

1) Bitadressierbares SFR.
2) Wiederholt aufgelistetes SFR, da Bits in verschiedenen Blöcken von Bedeutung.
3) X bedeutet undefinierten Zustand.

Block	Symbol	Name	Adresse	Inhalt n.RE-SET
Serial Channels	**ADCON0[2]**	**A/D Converter Control Register**	**$0D8_H$[1]**	**00_H**
	PCON[2]	Power Control Register	87_H	00_H
	S0BUF	Serial Channel 0, Buffer Register	99_H	XX_H[3]
	S0CON	**Serial Channel 0 Control Register**	**98_H[1]**	**00_H**
	S0RELL	Serial Channel 0, Reload Reg., low byte	$0AA_H$	$0D9_H$
	S0RELH	Serial Channel 0, Reload Reg., high byte	$0BA_H$	$XXXX.XX11_B$[3]
	S1BUF	Serial Channel 1, Buffer Register	$9C_H$	XX_H[3]
	S1CON	Serial Channel 1, Control Register	$9B_H$	$0X00.0000_B$[3]
	S1REL	Serial Channel 1, Reload Register	$9D_H$	00_H
	S1RELH	Serial Channel 1, Reload Reg., high byte	$0BB_H$	$XXXX.XX11_B$[3]
Watchdog	**IEN0[2]**	**Interrupt Enable Register 0**	**$0A8_H$[1]**	**00_H**
	IEN1[2]	**Interrupt Enable Register 1**	**$0B8_H$[1]**	**00_H**
	IP0[2]	Interrupt Priority Register 0	$0A9_H$	00_H
	IP1[2]	Interrupt Priority Register 1	$0B9_H$	$XX00.0000_B$[3]
	WDTREL	Watchdog Timer Reload Register	86_H	00_H
Compare/ Capture Unit (CCU)	CCEN	Compare/Capture Enable Register	$0C1_H$	00_H
	CC4EN	Compare/Capture 4 Enable Register	$0C9_H$	$X000.0000_B$[3]
	CCH1	Compare/Capture Register 1, High Byte	$0C3_H$	00_H
	CCH2	Compare/Capture Register 2, High Byte	$0C5_H$	00_H
	CCH3	Compare/Capture Register 3, High Byte	$0C7_H$	00_H
	CCH4	Compare/Capture Register 4, High Byte	$0CF_H$	00_H
	CCL1	Compare/Capture Register 1, Low Byte	$0C2_H$	00_H
	CCL2	Compare/Capture Register 2, Low Byte	$0C4_H$	00_H
	CCL3	Compare/Capture Register 3, Low Byte	$0C6_H$	00_H
	CCL4	Compare/Capture Register 4, Low Byte	$0CE_H$	00_H
	CMEN	Compare Enable Register	$0F6_H$	00_H
	CMH0	Compare Register 0, High Byte	$0D3_H$	00_H
	CMH1	Compare Register 1, High Byte	$0D5_H$	00_H
	CMH2	Compare Register 2, High Byte	$0D7_H$	00_H
	CMH3	Compare Register 3, High Byte	$0E3_H$	00_H
	CMH4	Compare Register 4, High Byte	$0E5_H$	00_H
	CMH5	Compare Register 5, High Byte	$0E7_H$	00_H
	CMH6	Compare Register 6, High Byte	$0F3_H$	00_H
	CMH7	Compare Register 7, High Byte	$0F5_H$	00_H
	CML0	Compare Register 0, Low Byte	$0D2_H$	00_H
	CML1	Compare Register 1, Low Byte	$0D4_H$	00_H
	CML2	Compare Register 2, Low Byte	$0D6_H$	00_H
	CML3	Compare Register 3, Low Byte	$0E2_H$	00_H
	CML4	Compare Register 4, Low Byte	$0E4_H$	00_H
	CML5	Compare Register 5, Low Byte	$0E6_H$	00_H
	CML6	Compare Register 6, Low Byte	$0F2_H$	00_H
	CML7	Compare Register 7, Low Byte	$0F4_H$	00_H
	CMSEL	Compare Input Select	$0F7_H$	00_H
	CRCH	Com./Rel./Capt. Register, High Byte	$0CB_H$	00_H
	CRCL	Com./Rel./Capt. Register, Low Byte	$0CA_H$	00_H
	CTCON	Com.Timer Control Register	$0E1_H$	$0XXX.0000_B$[3]
	CTRELH	Com. Timer Rel. Register, High Byte	$0DF_H$	00_H
	CTRELL	Com. Timer Rel. Register, Low Byte	$0DE_H$	00_H
	TH2	Timer 2, High Byte	$0CD_H$	00_H
	TL2	Timer 2, Low Byte	$0CC_H$	00_H
	T2CON	**Timer 2 Control Register**	**$0C8_H$[1]**	00_H

Programmstatuswort [PSW]

Das Programmstatuswort PSW enthält wichtige Informationen über den momentanen Programmzustand (Status). Es ist auf der direkt adressierbaren Speicheradresse 0D0H realisiert.
Beachte: Bei hexadezimaler Adressierung wird der Pseudotetrade eine Null vorangestellt.
Die Bitstellen des PSW sind mit den Bitverarbeitungsbefehlen einzeln ansprechbar.

0D7H	0D6H	0D5H	0D4H	0H3H	0D2H	0D1H	0D0H
CY	**AC**	**F0**	**RS1**	**RS0**	**OV**	**F1**	**P**

- **CY** **C**arr**Y** Bit

Ist die Verlängerung des Akkumulators um eine Stelle. Der Übertrag aus Bitstelle 2^7 des Akkumulators in die Bitstelle 2^8 wird im Carry-Bit gespeichert.
Bei den Mikrocontrollern der 8051-Familie spielt das Carry für die Bitverarbeitungsbefehle eine besondere Rolle, es wirkt als Bitakkumulator.
Alle arithmetischen Befehle beeinflussen das Carry.

- **AC** **A**uxiliary **C**arry oder Hilfs-Carry-Bit

Im Hilfs-Carry wird ein Übertrag aus dem niederwertigen in das höherwertige Halb-Byte des Akkumulators angezeigt. Der DA-Befehl (Decimal Adjust oder Dezimalkorrektur) zur Korrektur des Ergebnisses nach einer Addition zweier BCD-Zahlen wertet dieses Bit aus.

- **RS0** **R**egister Bank **S**elect Bit **0**
  RS1 **R**egister Bank **S**elect Bit **1**

Der Inhalt dieser beiden Bitstellen bestimmt die aktuelle Registerbank.

Registerbankauswahl:

RS1	RS0	Registerbank	Adressen der 8 Register
0	0	0	0 bis 7 (00H...07H)
0	1	1	8 bis 15 (08H...0FH)
1	0	2	16 bis 23 (10H...17H)
1	1	3	24 bis 31 (18H...1FH)

- **F0, F1** User **F**lag **0,1**

Die Bedeutung dieser Bits ist nicht festgelegt, der Anwender kann sie beliebig definieren. Diese beiden Flags unterscheiden sich von den ebenfalls nicht festgelegten Bits im internen Datenspeicher dadurch, daß sie bei Programmunterbrechungen durch Interrups in der Regel vom Anwender in den Stack gerettet werden. Sie können somit als lokales Flag genutzt und beim Rückspeichern des PSW wiederhergestellt werden.

- **OV** **OV**erflow Flag

Das Overflow-Flag ist zur Unterstützung der vorzeichenbehafteten Arithmetik vorgesehen. Es wird von den Arithmetik-Befehlen immer beeinflußt, ist aber nur von Bedeutung, wenn vorzeichenbehaftete Zahlen addiert bzw. subtrahiert werden.
Ist das OV gesetzt, zeigt es an, dass der Wertebereich (-128 bis + 127) unter- oder überschritten wurde.

- **P** **Pa**rity Bit

Das Paritätsbit ergänzt die Anzahl der im Akkumulator stehenden Einsen auf gerade Anzahl.
Befindet sich im Akkumulator eine gerade Anzahl von Einsen ist also das Paritätsbit "0",
sonst "1".
Das Paritätsbit richtet sich immer nach dem aktuellen Stand des Akkumulators. Alle Befehle, die den Akkumulator verändern, auch Transfer-Befehle, beeinflussen den Zustand des Paritätsbits.
Das Paritätsbit kann nur gelesen werden, ein Schreibvorgang in dieses Bit hätte keine Wirkung, da der Inhalt dieses Bits immer an den aktuellen Akkumulatorinhalt angeglichen wird.

Nach dem Rücksetzen des Controllers (nach RESET) sind PSW- und Akkumulator-Inhalt 00000000B.
Daraus ergibt sich folgender Zustand:
· Übertragsbits CY und AC sind gelöscht
· Überlaufbit für vorzeichenbehaftete Arithmetik OV ist gelöscht
· Registerbank 0 ist angewählt
· Paritätsbit P ist 0 (gerade Anzahl von Einsen im Akkumulator).

Timer 0 und 1

Der Mikrocontroller SAB 80C517/80C537 enthält neben dem Timer 2 und dem Compare Timer der Compare/Capture Unit [CCU] die Timer 0 und 1 für allgemeine Zähl- und Zeitgeberaufgaben, die im folgenden mit ihren Betriebsarten kurz beschrieben werden sollen.
Grundsätzlich wird zwischen Funktionen "Timer" [Zeitgeber] und "Counter" [Zähler] unterschieden.

- Timer Funktion: Das Zählregister wird mit jedem Maschinenzyklus um "Eins" erhöht.
  [Ein Maschinenzyklus besteht aus 12 Taktperioden]

- Counter Funktion: Das Zählregister wird mit jedem externen Ereignis
  [hier: fallende Flanke am Pin Port 3.4 (Alternativfunktion,
  Timer 0) bzw. am Pin Port 3.5 (Alternativfunktion,
  Timer 1) um "Eins" erhöht.
  Der Eingang wird während dem Zustand S5P2 jedes
  Maschinenzyklus abgetastet; die maximale Zählrate ist
  1/24 der Taktfrequenz.

Prinzipiell gibt es 4 Modi [Betriebsarten]:

- Mode 0 13 Bit Timer / Counter
- Mode 1 16 Bit Timer / Counter
- Mode 2 8 Bit Timer / Counter mit automatischem Nachladen (Auto Reload)
- Mode 3 Zwei 8 Bit Timer / Counter [nur für Timer 0; d.h. Timer 1 behält seinen Zählerstand als wäre TR1 = "0" gesetzt]

Die Funktionsweise ist den angegebenen Übersichten zu entnehmen.

Die Programmierung/Initialisierung erfolgt mit den beiden Special Function Registern TCON [Timer Control Register] und TMOD [Timer Mode Register].

TMOD BYTE - Timer 0 - Timer 1 - Modus Control Register

89H	GATE	C/T#	M1	M0	GATE	C/T#	M1	M0
	Timer 1				Timer 0			

Bit	Funktion
Gate	Gate-Steuerbit, Gate=1 bewirkt die Torfunktion des Zählers durch P3/2 für Timer0 und P3/3 für Timer1
C/T#	Zähler oder Zeitgeber Auswahlbit "1" Zähler Funktion (Eingang am Tx Eingangspin vom Controller) "0" Zeitgeber Funktion (Eingang vom internen Systemtakt)
M1/M0 0 0	Arbeitsmodi-Festlegung: 8-Bit-Zähler/Zeitgeber THx ist ein 8-Bit Zähler/Zeitgeber, TLx dient als 5-Bit Vorteiler
0 1	16-Bit-Zähler/Zeitgeber "THx" und TLx" ergeben zusammen einen 16-Bit-Zähler / Zeitgeber.
1 0	8-Bit automatisch ladender Zähler/Zeitgeber. THx enthält den Startwert für das Zählregister TLx. Dieser wird bei jedem Überlauf automatisch von THx nach TLx geladen.
1 1	Timer 0: Beide Register TH0 und TL0 sind eigenständige 8-Bit-Timer. TL0 ist ein 8-Bit-Zähler/Zeitgeber der von den Kontrollbits des Timer0 gesteuert wird. TH0 ist ein 8-Bit-Zeitgeber, der nur von den Zeitgeber1-Kontrollbits gesteuert wird.
1 1	Timer 1 stoppt in dieser Betriebsart

TCON BITS - Timer 0 - Timer 1 - Control Bits, bitadressierbar

	8FH	8EH	8DH	8CH	8BH	8AH	89H	88H
88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Die grau gekennzeichneten Bitpositionen sind hier nicht relevant.

Bit	Funktion
TR0	Timer 0 run control bit. Bit zum Starten und Stoppen des Zeitgebers T0 durch Software
TF0	Timer 0 overflow flag. Wird hardwaremäßig beim Zeitgeber/Zählerüberlauf gesetzt. Wird bei Aufruf der entsprechenden Interruptroutine automatisch zurückgesetzt.
TR1	Timer 1 run control bit. Bit zum Starten und Stoppen des Zeitgebers T1 durch Software
TF1	Timer 1 overflow flag. Wird hardwaremäßig beim Zeitgeber/Zählerüberlauf gesetzt. Wird bei Aufruf der entsprechenden Interruptroutine automatisch zurückgesetzt.

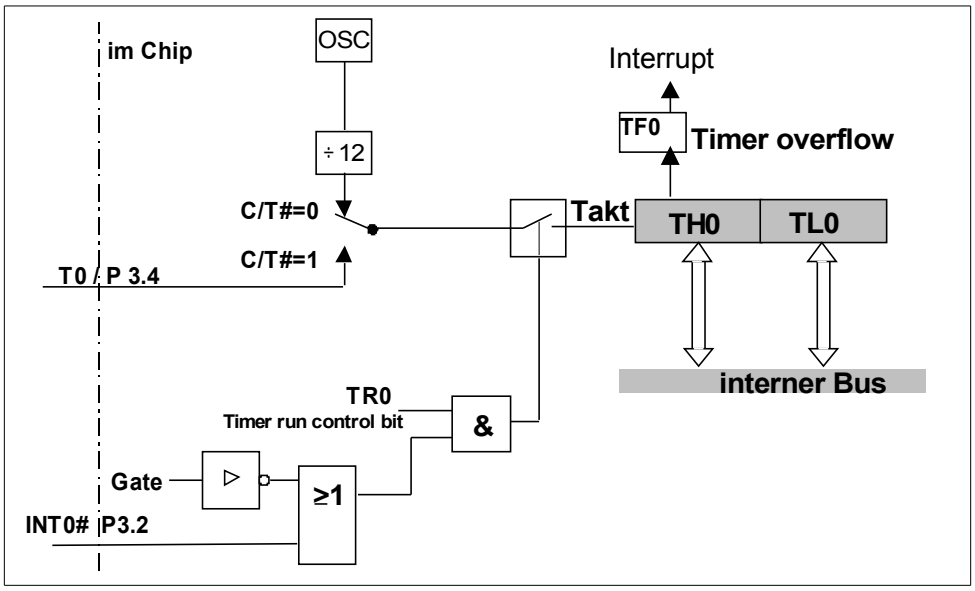

Bild4: Timer0/1 im Mode0 bzw. Mode1

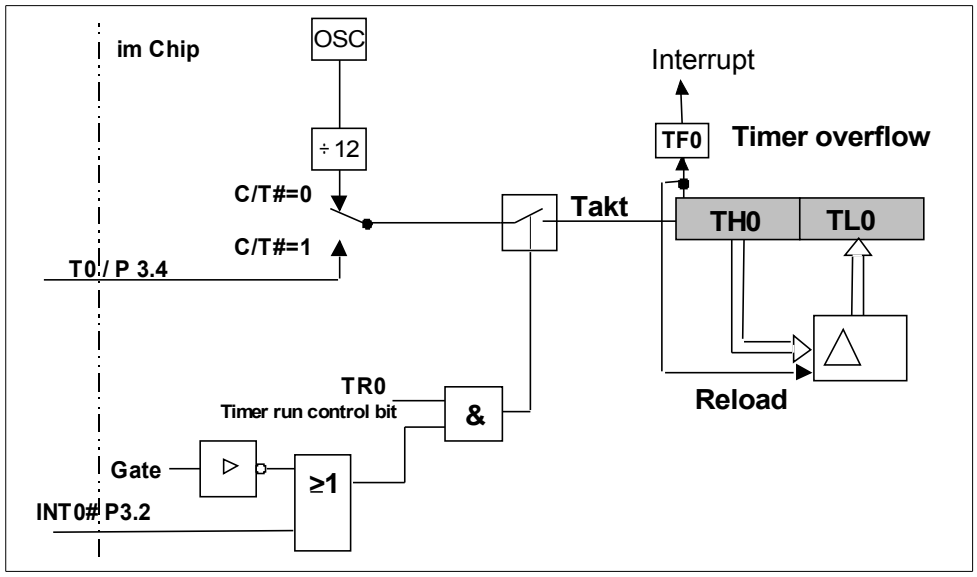

Bild5: Timer0/1 im Mode2 (8-Bit-Timer/Counter mit Autoreload)

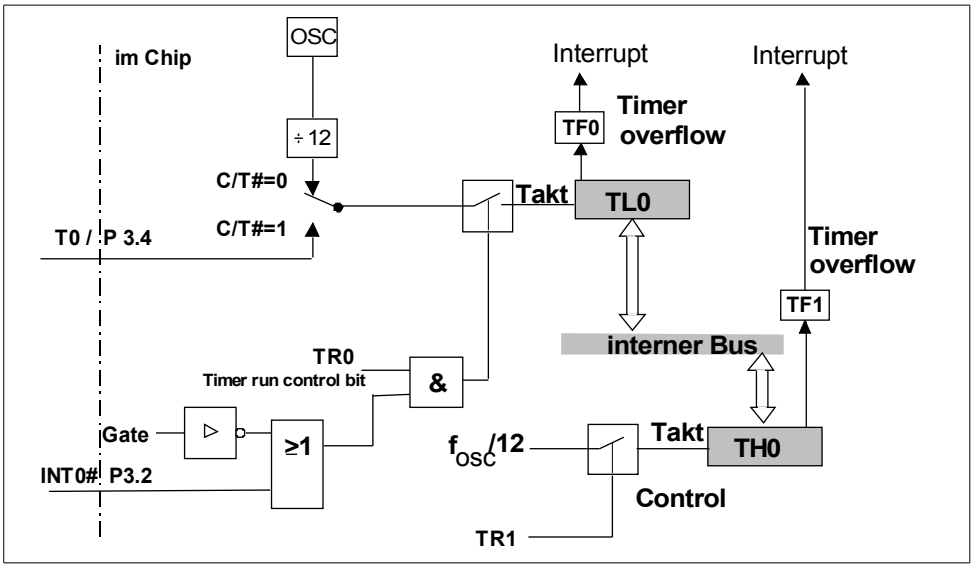

Bild6: Timer0/1 im Mode3 (Zwei 8-Bit-Timer/Counter)

Interruptsystem

Das Interruptsystem ermöglicht die unmittelbare Programmreaktion auf Ereignisse unterschiedlicher interner und externer Interrupt-Quellen durch Ausführung einer Interrupt-Serviceroutine (ISR). Die folgende Übersicht beinhaltet sämtliche Interrupt-Quellen mit ihren Vektoradressen. Auf diesen Programmspeicherplätzen müssen die Programmsequenzen der zugehörigen ISR beginnen.

Adresse	Interrupt-Name	Int.Requestflag	Erklärung
0003H	EXT0	IE0	External Interrupt 0
000BH	TIMER0	TF0	Timer 0 Overflow Interrupt
0013H	EXT1	IE1	External Interrupt 1
001BH	TIMER1	TF1	Timer 1 Overflow Interrupt
0023H	SINT0	RI0/TI0	Serial Channel 0 Interrupt
002BH	TIMER2	TF2/EXF2	Timer 2 Overflow/ External Reload Interrupt
0043H	ADCI	IADC	A/D-Converter Interrupt
004BH	EXT2	IEX2	External Interrupt 2 / Compare Event with CC4
0053H	EXT3	IEX3	External Interrupt 3 / Compare Event with CRC
005BH	EXT4	IEX4	External Interrupt 4 / Compare Event with CC1
0063H	EXT5	IEX5	External Interrupt 5 / Compare Event with CC2
006BH	EXT6	IEX6	External Interrupt 6 / Compare Event with CC3
0083H	SINT1	RI1/TI1	Serial Channel 1 Interrupt
0093H	CMINT	ICMPx	CMx Compare Register Interrupt (nur bei 80C517A)
009BH	COMPTIMER	CTF	Compare Timer Overflow
00A3H	SETINT	ICS	Compare Set Interrupt (nur bei 80C517A)
00ABH	CLRINT	ICR	Compare Clear Interrupt (nur bei 80C517A)

Interrupt-Prioritätslogik

Durch diesen Mechanismus ist die Unterbrechbarkeit einer niederprioren ISR durch eine hochpriore ISR möglich. Für alle Interrupt-Quellen existiert eine vierstufige Prioritätslogik, d.h. je nach Wertigkeit des Interrupt-Ereignisses kann vom Programmierer eine Prioritätsstufe für die ISR-Behandlung in den Bitpositionen der SFR IP0 und IP1 festgelegt werden. Dazu sind die Interruptquellen in sechs Prioritätsgruppen eingeteilt.

IP0	**-**	**-**	**IP0.5**	**IP0.4**	**IP0.3**	**IP0.2**	**IP0.1**	**IP0.0**
IP1	**-**	**-**	**IP1.5**	**IP1.4**	**IP1.3**	**IP1.2**	**IP1.1**	**IP1.0**

Prioritäten-Codierung		
Bit		Funktion
IP1.x	IP0.x	-
0	0	Priorität 0 (kleinste)
0	1	Priorität 1
1	0	Priorität 2
1	1	Priorität 3 (höchste)

Prioritätsgruppen	
Bit	Funktion
IP1.0 / IP0.0	IE0 / RI1+TI1 / IADC
IP1.1 / IP0.1	TF0 / IEX2
IP1.2 / IP0.2	IE1 / ICMPx / IEX3
IP1.3 / IP0.3	TF1 / CTF / IEX4
IP1.4 / IP0.4	RI0+TI0 / ICS / IEX5
IP1.5 / IP0.5	TF2 + EXF2 / ICR / IEX6

Beispiel: TIMER1 mit Prio3, SINT0 mit Prio2, ADCI mit Prio1
IP1.3/IP0.3=11 , IP1.4/IP0.4=10 , IP1.0/IP0.0=01 , alle anderen IP.x=0

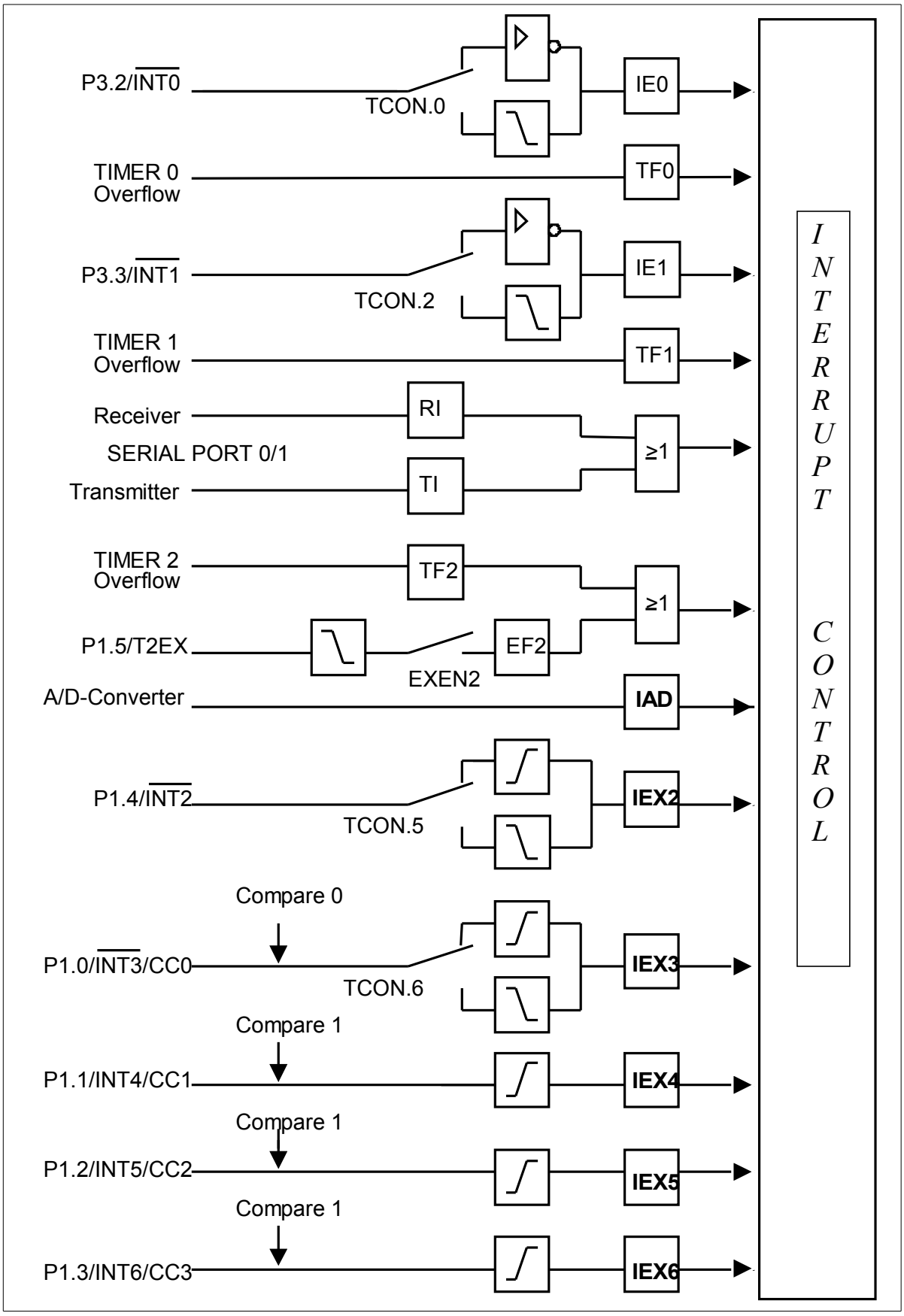

Bild7: Interrupt-Struktur des 80C517

Befehlsübersicht

Verwendete Symbole:

Rr	Inhalt der Register R0...R7
@Ri	Inhalt der über die Register R0 und R1 indirekt adressierten Datenspeicherzelle
dadr	Inhalt der direkt adressierten internen Datenspeicherzelle
konst	8-Bit-Konstante
konst16	16-Bit-Konstante
badr	Inhalt einer beliebigen Bitstelle; 128 Software-Flags, Ein-/Ausgabeleitungen, Steuer- und Statusbits
addr2K	Zieladresse im aktuellen 2-KByte-Bereich
addr64K	Zieladresse im gesamten 64-KByte-Bereich
rel	8-Bit-Offset im Bereich von -128< = rel < = + 127 relativ zur Adresse des ersten Bytes, das dem Sprungbefehl im Programmspeicher folgt

Transferbefehle

MOV	A,Rr	bringe Register in den Akku
MOV	A,dadr	bringe direkt adressierbare Datenspeicherzelle in den Akku
MOV	A,@Ri	bringe indirekt adressierbare Datenspeicherzelle in den Akku
MOV	A,#konst	bringe Konstante konst in den Akkumulator
MOV	Rr,A	bringe Akkumulator ins Register
MOV	Rr,dadr	bringe direkt adressierbare Datenspeicherzelle ins Register
MOV	Rr, #konst	bringe Konstante konst ins Register
MOV	dadr,A	bringe Akku in direkt adressierbare Datenspeicherzelle
MOV	dadr,Rr	bringe Register in direkt adressierbare Datenspeicherzelle
MOV	dadr,dadr	bringe direkt adressierbare Datenspeicherzelle in direkt adressierbare Datenspeicherzelle
MOV	dadr,@Ri	bringe indirekt adressierbare Datenspeicherzelle in direkt adressierbare Datenspeicherzelle
MOV	dadr, #konst	bringe Konstante konst in direkt adressierbare Datenspeicherzelle
MOV	@Ri,A	bringe Akku in indirekt adressierbare Datenspeicherzelle
MOV	@Ri,dadr	bringe direkt adressierbare Datenspeicherzelle in indirekt adressierbare Datenspeicherzelle
MOV	@Ri, #konst	bringe Konstante konst in indirekt adressierbare Datenspeicherzelle
MOV	DPTR, #konst16	bringe Konstante konst16 in den Datenzeiger (DPTR = DataPoinTeR)
MOVC	A,@A+PC	bringe relativ zum Programmzähler adressiertes Byte des Programmspeichers in den Akku
MOVC	A,@A+DPTR	bringe relativ zum Datenzeiger adressiertes Byte des Programmspeichers in den Akkumulator
MOVX	A,@Ri	bringe externe Datenspeicherzelle in den Akku (8-Bit Adresse)
MOVX	A,@DPTR	bringe externe Datenspeicherzelle in den Akku (16-Bit Adr.)
MOVX	@Ri,A	bringe Akku in externe Datenspeicherzelle (8-Bit Adresse)
MOVX	@DPTR,A	bringe Akku in externe Datenspeicherzelle (16-Bit Adresse)
PUSH	dadr	bringe direkt adressierbare Speicherzelle auf den Stack
POP	dadr	hole direkt adressierbare Speicherzelle vom Stack
XCH	A,Rr	vertausche Akku und Register
XCH	A,dadr	tausche Akku und direkt adressierbare Speicherzelle
XCH	A,@Ri	tausche Akku und indirekt adressierbare Speicherzelle
XCHD	A,@Ri	tausche die niederwertigen Halbbytes von Akku und indirekt adressierbare Speicherzelle
SWAP	A	tausche die Halbbytes des Akkumulator

Logik-Befehle

ANL	A,Rr	Akku UND Register
ANL	A,dadr	Akku UND direkt adressierbare Datenspeicherzelle
ANL	A,@Ri	Akku UND indirekt adressierbare Datenspeicherzelle
ANL	A,#konst	Akku UND Konstante konst
ANL	dadr,A	direkt adressierbare Datenspeicherzelle UND Akkumulator
ANL	dadr,#konst	direkt adressierbare Datenspeicherzelle UND Konstante konst
ORL	A,Rr	Akku ODER Register
ORL	A,dadr	Akku ODER direkt adressierbare Datenspeicherzelle
ORL	A,@Ri	Akku ODER indirekt adressierbare Datenspeicherzelle
ORL	A,#konst	Akku ODER Konstante konst
ORL	dadr,A	direkt adressierbare Datenspeicherzelle ODER Akku
ORL	dadr,#konst	direkt adressierbare Datenspeicherzelle ODER Konstante konst
XRL	A,Rr	Akku EXCLUSIV-ODER Register
XRL	A,dadr	Akku EXCLUSIV-ODER direkt adressierbare Datenspeicherzelle
XRL	A,@Ri	Akku EXCLUSIV-ODER indirekt adressierbare Datenspeicherzelle
XRL	A,#konst	Akku EXCLUSIV-ODER Konstante konst
XRL	dadr,A	direkt adressierbare Datenspeicherzelle EXCLUSIV-ODER Akku
XRL	dadr,#konst	direkt adressierbare Datenspeicherzelle EXCLUSIV-ODER konst
CLR	A	lösche Akkumulator
CPL	A	komplementiere Akkumulator

Bitverarbeitungsbefehle

CLR	C	lösche CARRY-Bit
CLR	badr	lösche beliebiges Bit
SETB	C	setze CARRY-Bit
SETB	badr	setze beliebiges Bit
CPL	C	komplementiere CARRY-Bit
CPL	badr	komplementiere beliebiges Bit
ANL	C,badr	CARRY-Bit UND beliebiges Bit
ANL	C,/badr	CARRY-Bit UND invertiertes beliebiges Bit
ORL	C,badr	CARRY-Bit ODER beliebiges Bit
ORL	C,/badr	CARRY-Bit ODER invertiertes beliebiges Bit
MOV	C,badr	bringe beliebiges Bit ins CARRY-Bit
MOV	badr,C	bringe CARRY-Bit in beliebiges Bit

Rotier-Befehle

RL	A	rotiere Akkumulator links
RLC	A	rotiere Akkumulator durchs CARRY-Bit links
RR	A	rotiere Akkumulator rechts
RRC	A	rotiere Akkumulator durchs CARRY-Bit rechts

Leerbefehl

NOP	Leerbefehl (No Operation = tue nichts)

Arithmetische Verknüpfungen

ADD	A,Rr	addiere Register zum Akkumulator
ADD	A,dadr	addiere direkt adressierbare Datenspeicherzelle zum Akku
ADD	A,@Ri	addiere indirekt adressierbare Datenspeicherzelle zum Akku
ADD	A,#konst	addiere Konstante konst zum Akku
ADDC	A,Rr	addiere Register und CARRY zum Akkumulator
ADDC	A,dadr	addiere direkt adressierbare Datenspeicherzelle und CARRY zum Akku
ADDC	A,@Ri	addiere indirekt adressierbare Datenspeicherzelle und CARRY zum Akku
ADDC	A,#konst	addiere Konstante konst und CARRY zum Akkumulator
SUBB	A,Rr	subtrahiere Register und CARRY vom Akkumulator
SUBB	A,dadr	subtrahiere direkt adressierbare Speicherzelle und CARRY vom Akku
SUBB	A,@Ri	subtrahiere indirekt adressierbare Speicherzelle und CARRY vom Akku
SUBB	A,#konst	subtrahiere Konstante konst und CARRY vom Akkumulator
INC	A	erhöhe Akkumulator um Eins
INC	Rr	erhöhe Register um Eins
INC	dadr	erhöhe direkt adressierbare Datenspeicherzelle um Eins
INC	@Ri	erhöhe indirekt adressierbare Datenspeicherzelle um Eins
INC	DPTR	erhöhe den Datenzeiger um Eins
DEC	A	erniedrige Akkumulator um Eins
DEC	Rr	erniedrige Register um Eins
DEC	dadr	erniedrige direkt adressierbare Datenspeicherzelle um Eins
DEC	@Ri	erniedrige indirekt adressierbare Datenspeicherzelle um Eins
MUL	AB	multipliziere Akku mit Register B
DIV	AB	dividiere Akku durch Register B
DA	A	korrigiere den Akku zu einer BCD-Zahl

Sprung- und Unterprogrammbefehle

ACALL	addr2K	rufe Unterprogramm im aktuellen 2K-Bereich auf
LCALL	addr64K	rufe Unterprogramm im gesamten Adreßbereich auf
RET		springe aus einem Unterprogramm zurück
RETI		springe aus einer Interrupt-Routine (ISR) zurück
AJMP	addr2K	springe im aktuellen 2K-Bereich
LJMP	addr64K	springe im gesamten Adreßbereich
SJMP	rel	springe relativ zum Programmzähler
JMP	@A+DPTR	springe indirekt relativ zum Datenzeiger
JZ	rel	springe, wenn der Akku null ist
JNZ	rel	springe, wenn der Akku nicht Null ist
JC	rel	springe, wenn das CARRY-Bit Eins ist
JNC	rel	springe, wenn das CARRY-Bit Null ist
JB	badr,rel	springe, wenn beliebiges Bit Eins ist
JNB	badr,rel	springe, wenn beliebiges Bit Null ist
JBC	badr,rel	springe, wenn beliebiges Bit Eins ist und lösche das Bit
CJNE	A,dadr,rel	vergleiche Akku und direkt adressierbare Datenspeicherzelle ...
CJNE	A,#konst,rel	vergleiche Akku und Konstante konst ...
CJNE	Rr,#konst,rel	vergleiche Register und Konstante konst ...
CJNE	@Ri,#konst,rel	vergleiche indirekt adressierbare Datenspeicherzelle und konst
	... und springe, wenn sie ungleich sind	
DJNZ	Rr,rel	erniedrige Register um Eins und springe, wenn ...
DJNZ	dadr,rel	erniedrige direkt adressierbare Datenspeicherzelle um Eins und
		und springe, wenn ... sie nicht Null geworden ist.

Direktiven und Steueranweisungen des Assemblers A51

Assembler-Direktiven

Symboldefinition:
- SEGMENT Deklaration von verschiebbaren (relokatiblen) Segmenten
- EQU Definition von Symbolen mit Ausdruck- oder Registersymbolzuweisung
- SET wiederholte Definition von Symbolen (ähnlich EQU)
- DATA, IDATA, BIT, XDATA, CODE
  Symboldefinition für Elemente des internen und externen Datenspeichers (RAM) und des Programmspeichers, Adresszuordnung

Initialisieren und Reservieren von Speicherbereichen:
- DS, DBIT Reservierung von zusammenhängenden Bytes bzw. Bits im Programm- oder Datenspeicher
- DB, DW Initialisierung von zusammenhängenden Bytes/Worten im Programmspeicher

Mehrmodul-Direktiven:
- PUBLIC Veröffentlichung von Namen für andere zu bindende Programmmodule
- EXTERN Bekanntgabe von Symbolen, die in anderen Modulen existieren
- NAME Namenszuweisung für Objektmodule

Assembler-Zustandsdirektiven und Segmentauswahl:
- ORG Adresspegelfestlegung für nachfolgende Maschinenbefehle
- END Beendet das Programmmodul in der letzten Zeile
- RSEG Ein definiertes Segment wird zum aktuellen Segment
- CSEG, DSEG, XSEG, ISEG, BSEG
  Erzeugung von absoluten Segmenten der Typen Code, Daten, ext. Daten, indirekt adressierbare Daten und bitadressierbare Daten
- USING Registerbankdefinition für nachfolgenden Code

Makro-Anweisungen und Makro-Aufrufe:
- MACRO, ENDM, LOCAL, REPT, IRP, IRPC, EXITM

Assembler-Steueranweisungen

Primäranweisungen: (dürfen nur einmal auftreten und werden mit $ eingeleitet)
- DATE
- DEBUG / NODEBUG
- ERRORPRINT / NOERRORPRINT
- OBJECT / NOOBJECT
- PAGELENGTH
- PAGEWIDTH
- PRINT / NOPRINT
- SYMBOLS / NOSYMBOLS
- XREF / NOEXREF
- TITLE
- MOD51 / NOMOD51
- COND / NOCOND
- MACRO / NOMACRO
- REGISTERBANK / NOREGISTERBANK

Sekundäranweisungen: (dürfen mehrmals auftreten und werden mit $ eingeleitet)
- EJECT
- INCLUDE
- LIST / NOLIST
- GEN / NOGEN
- SAVE / RESTORE
- SET / RESET
- IF
- ELSEIF
- ELSE
- ENDIF

Beispiel-Quellmodule

Beispiel_1: Datentransfer, Verknüpfung und Ausgabe

Aufgabe: Ein im Festwertspeicher befindliches Datenfeld ist in den internen RAM des µC zu kopieren, dort zu bearbeiten und anschließend taktgesteuert über ein Parallelport auszugeben.

Lösung:

1. Programmtechnische Realisierung der Generierung eines Datenfeldes im Programmspeicher:
   An geeigneter Stelle des Quellmoduls wird mittels der DB- oder DW-Assemblerdirektive ein Feld zusammenhängender Bytes erzeugt und mit den angegebenen Werten initialisiert.
   Feldposition (Label): *STEUERFELD*
   Feldgröße: 20Byte

```
STEUERFELD:
  db    20,13,98,255,1            ; dezimal
  db    62h,0ffh,0e1h,55h,73h     ; hexadezimal
  db    23o,153o,77o,16o,377o     ; oktal
  db    110b,11111111b,101b       ; binär
  db    11001100b,11b
```

2. Zur Bearbeitung muss dieses Feld in den internen RAM des µC kopiert werden:
   Man stellt die Registergruppe ein und initialisiert die Variablen für die Kopierschleife.
   Mit Hilfe der indirekten Adressierung greift man auf das Quelldatenfeld im Programmspeicher zu.
   Der Akku ist Zwischenspeicher, aus dem mit indirekter Adressierung das Datenbyte in den internen RAM transportiert wird.
   Am Schleifenende sind die Zeigervariablen um Eins zu erhöhen und die Laufvariable zu steuern.

```
PUFFER:  ds   20                ; 20Byte-Puffer im int. RAM

;Steuerfeld in int. RAM kopieren
         mov    PSW,#10h        ; Registerbank2 (RB2) einstellen
         mov    B,#20           ; Datenanzahl
         mov    DPTR,#STEUERFELD ; Zeiger auf Datenquelle
         mov    r0,#PUFFER      ; Zeiger auf Datensenke
   ;Kopierschleife
KOP:     clr    A               ; A=0, weil Akku im nächsten
                                ; Befehl als Index verwendet
         movc   A,@A+DPTR       ; ein Byte lesen
         mov    @r0,A           ; dieses Byte in Puffer schreiben
         inc    DPTR            ; Quellzeiger erhöhen
         inc    r0              ; Zielzeiger erhöhen
         djnz   B,KOP           ; Schleifendurchlaufsteuerung
```

3. Bearbeitung des Datenfeldes:
   Aufgabe: Einzelne Bitstellen sind zu löschen
   Schleifenvorbereitung wie unter Pkt.2
   Da einige Verknüpfungen, so auch der UND-Befehl, nur im Akku durchgeführt werden können, benutzt man wieder die indirekte Adressierung für den Datentransport zum und vom Akku.

```
   ;Bit_0 und Bit_5 bleiben unverändert, alle anderen Bitstellen
ANDMASKE  equ  00100001b  ; ... werden gelöscht
         mov    B,#20           ; Datenanzahl laden
         mov    r0,#PUFFER      ; Datenpufferzeiger laden
ANDSCHLEIFE:
         mov    A,@r0           ; Datenbyte in Akku laden
         anl    A,#ANDMASKE ; AND-Verknüpfung im Akku
         mov    @r0,A           ; Ergebnis zuzück in das Feld
         inc    r0              ; Datenpufferzeiger erhöhen
         djnz   B,ANDSCHLEIFE ; Schleifendurchlaufsteuerung
```

4. Von einem externen Takt gesteuerte byteweise Ausgabe des modifizierten Datenfeldes:
   Den Ausgabetakt liefert ein Timerprozess. (Interruptserviceroutine setzt TIM=1)
   Es ist dieses Taktflag auszutesten und zu löschen.

```
         mov    B,#20           ; Datenanzahl laden
         mov    r0,#PUFFER      ; Datenpufferzeiger laden
WARTE:   jnb    TIM,WARTE   ; Flag-Polling
         clr    TIM             ; Taktflag löschen
         mov    P4,@r0 ; Datenbyte auf Port4 ausgeben
         inc    r0              ; Datenpufferzeiger erhöhen
         djnz   B,WARTE         ; Schleifendurchlaufsteuerung
```

Beispiel_2: 16-Bit-Arithmetik

Aufgabe: Es ist die Summe von zwei im internen Datenspeicher befindlichen 16-Bit-Operanden zu bilden. Die 3-Byte-Summe ist in den Registern R0 (High_Teil) bis R2 (Low_Teil) der Registergruppe_3 zu verwalten.

Lösung:

Man stellt die Registergruppe ein und löscht die Summenregister.

Bei einem 8-Bit-Prozessor hat man die Low- und High-Teile der beiden Operanden unter Beachtung des Übertrags getrennt zu addieren.

```
SUM1:  ds    2        ; die beiden WORD-Summanden, getrennt nach
                      ; oberen und niederen 8-Bit-Teil, befinden
SUM2:  ds    2        ; sich im int. RAM, deshalb je zwei Byte reservieren.


       mov   PSW,#18h  ; PSW auf RB3 setzen
       mov   r0,#0     ; Summe_high löschen
       mov   r1,#0     ; Summe_middle löschen
       mov   r2,#0     ; Summe_low löschen
       mov   A,SUM1+1  ; Low_Teil vom 1. Summanden in Akku
       add   A,SUM2+1  ; Low_Teil des 2. Summanden addieren
       mov   r2,A      ; Ergebnis nach R2 transportieren
       mov   A,SUM1    ; High_Teil vom 1. Summanden in Akku
       addc  A,SUM2    ; High_Teil des 2. Summanden und
                       ; Übertrag der Low_Teil-Addition addieren
       mov   r1,A      ; Ergebnis nach R1 transportieren
       clr   A         ; neuen Übertrag nach r0 transportieren
       mov   ACC.0,C
       mov   r0,A
```

Beispiel_3: 16-Bit-Vergleichsoperation

Aufgabe: Es sind zwei im internen Datenpeicher befindliche 16-Bit-Operanden zu vergleichen. Als Ergebnis sind über das Port_5 drei LED-Elemente folgendermaßen zu aktivieren:

OP1 > OP2 → P5.0 = 1
OP1 = OP2 → P5.1 = 1
OP1 < OP2 → P5.2 = 1

Lösung:

Bei dem 8-Bit-Prozessor hat man die High- und Low-Teile der beiden Operanden getrennt zu vergleichen.

Zuerst vergleicht man die beiden High-Teile, bei Gleichheit noch die beiden Low-Teile.

Bei Ungleichheit der High-Teile zeigt im Ergebnis des CJNE-Befehls das C-Flag die Relation der beiden Operanden an.

```
OP1:   ds    2              ; die beiden Summanden, getrennt nach
                            ; oberen und niederen 8-Bit-Teil, befinden
OP2:   ds    2              ; sich im int. RAM, deshalb zwei Byte res.

LED_Loeschmaske  equ   11111000b

       anl   P5,#LED_Loeschmaske ; vor dem Vergleich alle LED's aus
       mov   A,OP1                ; erster High_Teil in Akku
       cjne  A,OP2,TESTGRH        ; und mit 2. High_Teil vergleichen
       ; beide High_Teile sind gleich, also noch Low_Teile vergleichen
       mov   A,OP1+1              ; erster Low_Teil in Akku
       cjne  A,OP2+1,TESTGRL      ; und mit 2. Low_Teil vergleichen
       ; beide Operanden sind gleich, also P5.1 = 1
       setb  P5.1
       jmp   ENDE
TESTGRL:  ; Welcher Low_Teil ist größer ?   (C-Flag-Auswertung)
       jnc   LED1_AN       ; springe, wenn C=0, also OP1_L > OP2_L
       jmp   LED3_AN       ; springe, wenn C=1, also OP2_L > OP1_L
TESTGRH:  ; Welcher High_Teil ist größer ?    (C-Flag-Auswertung)
       jnc   LED1_AN       ; springe, wenn C=0, also OP1_H > OP2_H
LED3_AN:                   ; OP2 ist größer als OP1
       setb  P5.2
       jmp   ENDE
LED1_AN:                   ; OP2 ist kleiner als OP1
       setb  P5.0
ENDE:
```

Assemblerlisting des Praktikum-Rahmenprogramms

```
DOS MACRO ASSEMBLER A51 V5.50
OBJECT MODULE PLACED IN PRAKT.OBJ
ASSEMBLER INVOKED BY: C:\C51P\BIN\A51.EXE PRAKT.A51 NOSB XR DB NOMO EP

LOC  OBJ            LINE     SOURCE

                       1     ;*******************************************************
                       2     ;  Hochschule für Technik und Wirtschaft Dresden(FH)    *
                       3     ;  FB ET,       Praktikum Mikrocontrollertechnik       *
                       4     ;-------------------------------------------------------*
                       5     ;Projekt:  Microcontrollersteuerung                    *
                       6     ;Modul:    prakt.a51                                   *
                       7     ;System:   SAB80C517A ( MCB517AC/KEIL )                *
                       8     ;Aufgabe:  Rahmenprogramm zu allen Praktikumsaufgaben  *
                       9     ;          In diesen 8051-Programmrumpf sind an den    *
                      10     ;          geeigneten Stellen die notwendigen Programm- *
                      11     ;          zeilen einzufügen.                          *
                      12     ;-------------------------------------------------------*
                      13     ;Autor: J. Huhle HTWD                                  *
                      14     ;       huhle@htw-dresden.de         Tel.: HA 2542     *
                      15     ;Datum: 08.05.2002                                     *
                      16     ;-------------------------------------------------------*
                      17     ;bearbeitet von:                                       *
                      18     ;*******************************************************
                      19
                      20     $include(reg517a.inc)     ;Register_Symboldefinitionen
                     219
                     220     ; Definitionen für verschiebare (relocatable)...
                     221     mainprog  segment  code   ; ... Programm-
                     222     const     segment  code   ; Konstanten-
                     223     main_data segment  data   ; Daten- und
                     224     main_bits segment  bit    ; Bitsegmente
                     225
                     226     ;***  glob. Nutzervereinbarungen und -definitionen  ***
                     227     ; Konstanten
  EC78               228       t0rel    equ  0-5000  ; T0_Reloadwert, @12MHz  T_INT = 5ms
  00EF               229       mask1    equ  0efh    ; Initialwert für MASKE
  0008               230       user_psw equ  8       ; Name für eine Registerbank_Adresse
                     231
                     232     ; Variable in der Reg_Bank 1  (Beispiele)
----                 233             dseg at 8     ; nutzt RG1 R0.. max. 8 Byte
0008                 234       MASKE:   ds   1     ; ein Byte für MASKE res. (entspricht R0)
0009                 235       POINTER: ds   1     ; ein Byte für POINTER res. (entspr.  R1)
                     236
                     237     ; Variable im frei nutzbarem Register_Bereich (ein Beispiel)
----                 238             rseg main_bits
0000                 239       Signal:  dbit 1       ; Anwender-Signalflag
----                 240             rseg main_data
0000                 241       PUFFER:  ds   4       ; vier Byte für Puffer reserviert
----                 242             iseg  at 0e0h
00E0                 243       usersp:  ds   20h     ; Reservierung für Stack ab E0H
                     244
                     245     ;------- Programmstart ab PC=0000 -----------------
----                 246             cseg    at 0
0000 020000   F      247             jmp     start   ; Der Sprung zur Startmarke wird
                     248                             ; notwendig, da ab PC=0003
                     249                             ; INT_Einsprünge liegen können
                     250
                     251     ;------- Interrupteinsprungadressen ---------------
                     252     ; Einsprung der T0-OV-ISR
----                 253             cseg at 0bh
000B 020000   F      254             jmp     t0_isr  ; Sprung zur eigentlichen ISR,
                     255                             ; da ab PC=0013h weitere ISR möglich
                     256
                     257     ;---- Initialisierungsteil mit UP-Aufruf von userinit ----
----                 258             rseg mainprog
0100                 259             org     100h
0100 C2AF            260     start:  clr     EAL             ; alle INT's sperren
0102 7581DF          261             mov     sp,#usersp-1    ; Stackpointer festlegen
                     262          ; Arbeitsregister 0...255 löschen
0105 75D000          263             mov     psw,#0
0108 E4              264             clr     a
0109 78FF            265             mov     r0,#255         ; Zähler r0 laden
010B F6              266     regcle: mov     @r0,a           ; Löschschleife
010C D8FD            267             djnz    r0,regcle
```

Fortsetzung des Assemblerlistings

```
                      268        ; USER_Variablen initialisieren
                      269           ;RG1 (PSW=8)   (Beispiel)
010E 75D008           270             mov     psw,#user_psw
0111 78EF             271             mov     r0,#mask1       ; MASKE initialisieren
0113 7900     F       272             mov     r1,#PUFFER      ; R1 wird als Zeiger genutzt
0115 750005   F       273             mov     PUFFER,#5       ; Puffer mit 5 initialisieren
                      274          ; On-Chip Peripherie initialisieren
0118 120000   F       275             call    OnChipInit
011B D2AF             276             setb    eal             ; INT freigeben
                      277
                      278     ;---------- Hauptprogramm -------------------------
011D                  279     main: ; Hauptprogrammschleife
                      280           ; muss mit Leben erfüllt werden
                      281         ;   hier werden alle nicht zeitkrtischen Funktionen gerufen
011D 80FE             282             jmp     main
                      283
                      284     ;-------- T0_ISR   Basiszeitgeber -----------------
011F 758CEC           285     t0_isr: mov     th0,#high t0rel ; T0_reload
0122 758A78           286             mov     tl0,#low  t0rel
0125 C0D0             287             push    psw
0127 C0E0             288             push    acc
0129 75D008           289             mov     psw,#user_psw
                      290           ;ISR_action   (Beispiel)
                      291           ;muss mit Leben erfüllt werden
012C 09               292             inc     P4              ; LED-Kette manipulieren
                      293           ; hier alle taktgesteuerten Funktionen realisieren
012D D0E0             294     t0isren:pop     acc
012F D0D0             295             pop     psw
0131 32               296             reti
                      297     ;------------ ENDE der T0_ISR ---------------------
                      298
                      299     ;*******************************************
                      300     ;    UP ONCHIPINIT                          *
                      301     ;    Initialisierung der benötigten On-chip-  *
                      302     ;    peripherie (Timer,INT-System...)         *
                      303     ;*******************************************
0132                  304     OnChipInit:
                      305          ;Timer0-Initialisierung
0132 758CEC           306             mov     th0,#high t0rel
0135 758A78           307             mov     tl0,#low  t0rel
0138 758901           308             mov     tmod,#1          ; 16-Bit_timer
013B D2A9             309             setb    et0              ; INT_Freigabe
013D D28C             310             setb    tr0              ; T0-start
                      311          ;u.U. ser.Schnittstelle initialisieren
                      312             ;siehe conea0.inc
                      313          ;INT-Prioritäten festlegen
013F 75A900           314             mov     ip0,#00000000b   ;
0142 75B902           315             mov     ip1,#00000010b   ; T0 auf Stufe2
0145 22               316             ret                      ; Ende des UP
                      317
                      318     ;hier notwendige Treiber einfügen
                      319     ;$include (conea0.inc)           ; z.B.
                      320
----                  321             rseg const               ; z.B. für LCD-Texte
                      322     ;*************************************************
0000                  323     LCD_Text1:
0000 4C43442D         324             db      'LCD-OK',0
0004 4F4B00
0007                  325     LCD_Text2:
0007 302C3030         326             db      '0,000 Volt',0
000B 3020566F
000F 6C7400
                      327     ;*************************************************
                      328
                      329             end                      ; Programmende

ASSEMBLY COMPLETE.  0 WARNING(S), 0 ERROR(S)
```

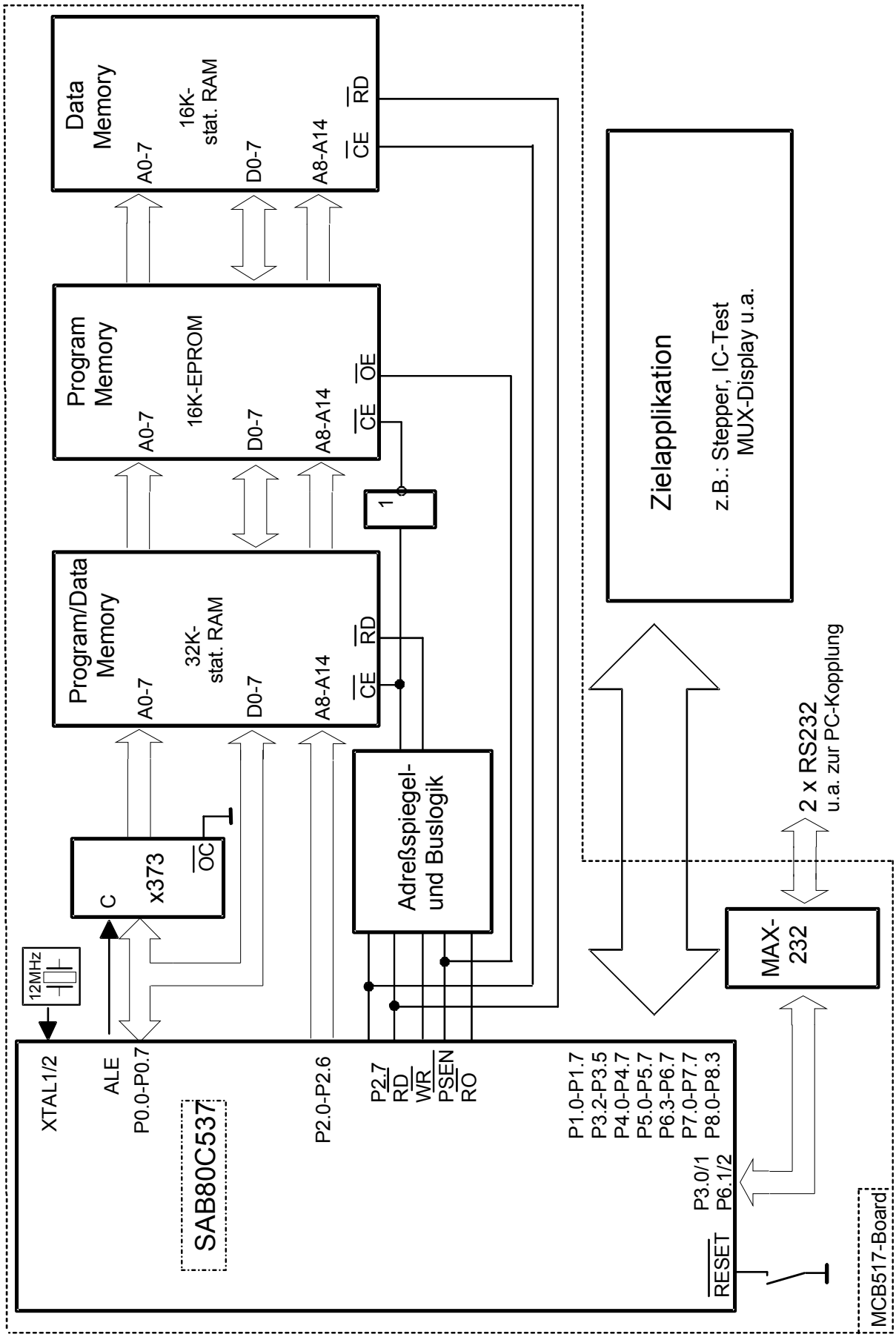

Bild8: Blockschaltbild des Mikrocontroller - Moduls