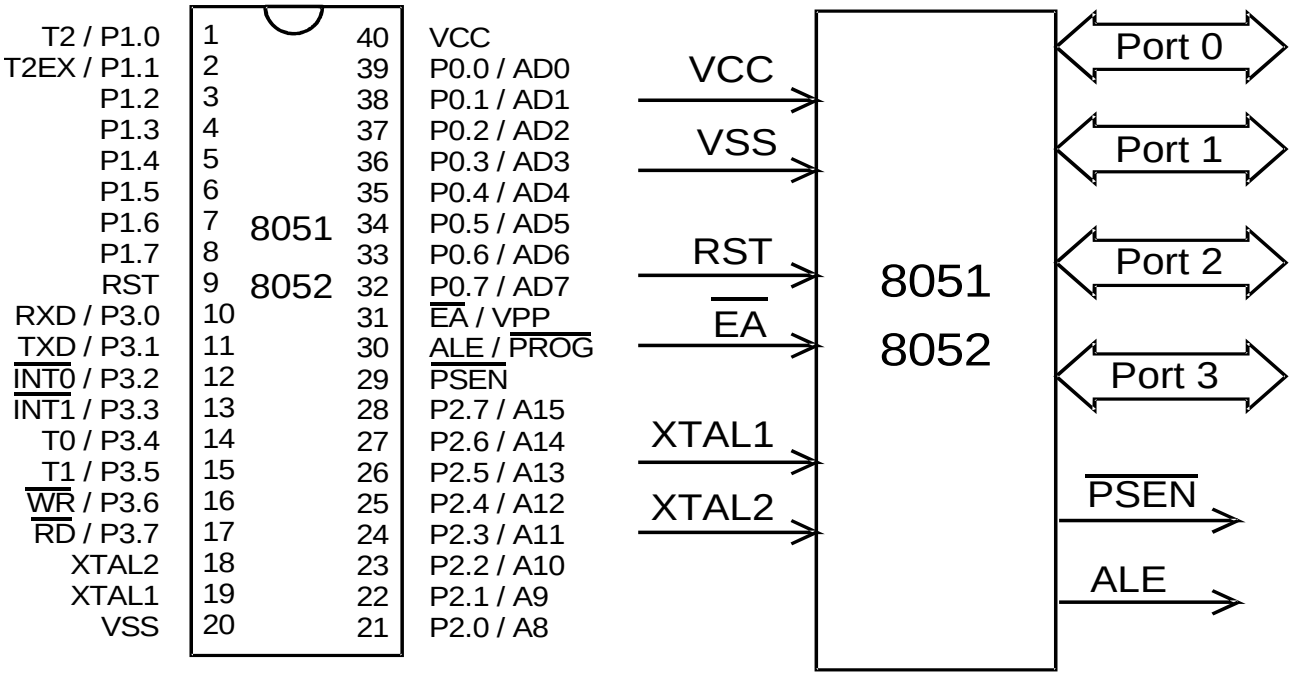
MCS51- Mikrocontroller

in der Steuerungstechnik



Lehrbuch

Inhaltsverzeichnis

1. Grundlagen

1.1 Zahlensysteme

In der Computertechnik sind drei Zahlensysteme als wichtig anzusehen. Das Dezimale, das Binäre und das Hexadezimale Zahlensystem. Das Dezimale als für den Menschen gebräuchliches. Das Binäre, da es nur die Ziffern 0 und 1 kennt und damit elektrisch verarbeitet werden kann. Das Hexadezimale als Kurzschreibweise für binäre Zahlenkolonnen.

Dezimal	-	Für Menschen gebräuchlich
Binär	-	Elektrisch verarbeitbar
Hexadezimal	-	Kurzschreibweise für Binärzahlen

Die drei Zahlensysteme lassen sich über vier gemeinsame Kennwerte beschreiben.

1. Basis	- Zahl auf die das Zahlensystem aufgebaut ist
2. Anzahl der Ziffern	- Entspricht direkt der Basis
3. Höchste Ziffer	- Entspricht Basis - 1
4. Potenzen	- (Stellenwertigkeiten) Entspricht direkt der Basis

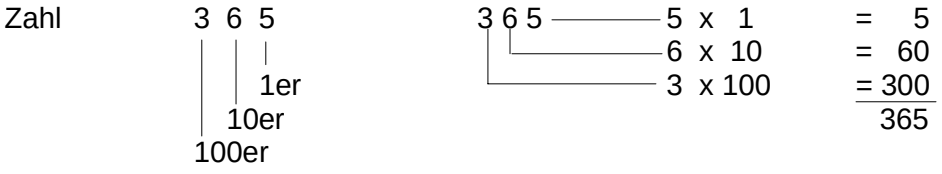
Zahlensystem	Dezimal	Binär	Hexadezimal
Basis	10	2	16
Anzahl Ziffern	10 (0-9)	2 (0/1)	16 (0-F)
Höchste Ziffer	9	1	F (15)
Potenzen	10er	2er	16er

Da es nur die Ziffern 0-9 gibt, werden im hexadezimalen Zahlensystem für die Wertigkeiten 10-15 die ersten sechs Buchstaben (A-F) des Alphabets benutzt.

A=10, B=11, C= 12, D=13, E=14, F=15

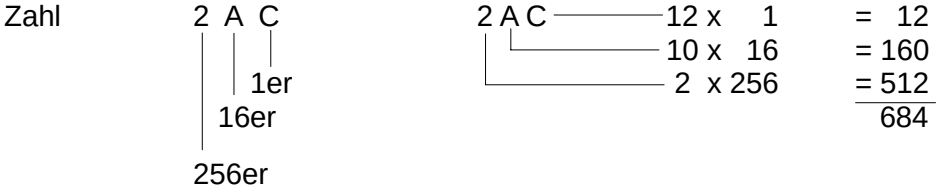
Zahlenbeispiele:

Dezimal - 10er Potenz bedeutet, die Stellenwertigkeit verzehnfacht sich Stelle für Stelle von rechts nach links.



Binär - 2er Potenz bedeutet, die Stellenwertigkeit verdoppelt sich Stelle für Stelle von rechts nach links.

Hexadezimal - 16er Potenz bedeutet, die Stellenwertigkeit versechzehnfacht sich Stelle für Stelle von rechts nach links.



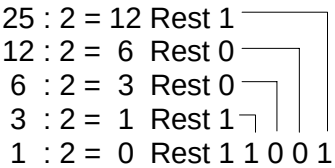
1.2 Umwandeln von Zahlen

Zur Umwandlung von Zahlen vom binären oder hexadezimalen Zahlensystem in Dezimalzahlen, kann die Aufschlüsselung nach Stellenwertigkeiten, wie vorab beschrieben, verwendet werden.

Zur Umwandlung von Dezimalzahlen in Zahlen des binären oder hexadezimalen Zahlensystems kann das Resteverfahren angewendet werden. Beim Resteverfahren wird die Dezimalzahl fortlaufend ganzzahlig (ohne Ziffern nach dem Komma) durch die Basis des gewünschten Zahlensystems geteilt. Der Quotient (Ergebnis der Teilung) wird jeweils als Dividend für die Folgedivision nach vorne gestellt, der Rest wird notiert. Entsteht der Quotient 0, ist die Umwandlung abgeschlossen. Die Reste, von unten nach oben gelesen sind das Ergebnis.

Beispiele:

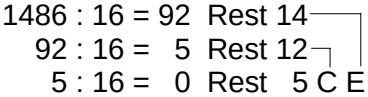
Binär



Kontrolle: 1 1 0 0 1

1 x 1 = 1
0 x 2 = 0
0 x 4 = 0
1 x 8 = 8
1 x 16 = 16
25

Hexadezimal



Kontrolle: 5 C E

14 x 1 = 14
12 x 16 = 192
5 x 256 = 1280
1486

Muß das Zahlensystem dem eine Zahl angehört mit angegeben werden, wie beispielsweise bei Assemblerprogrammierung, so wird an die Zahl ein Kennbuchstabe für das Zahlensystem angehängt.

Kein Buchstabe oder	d/D	= Dezimal	100 / 100D
	h/H	= Hexadezimal	100H
	b/B	= Binär	100B

1.3 Codierte Darstellungen

BCD - Binär Codierte Dezimal Ziffer/Zahl

Für jede Dezimalziffer wird eine vierstellige Binärzahl (Tetrade), welche die Wertigkeit der Dezimalziffer aufweist verwendet. Die Tetraden, welche eine Dezimalziffer repräsentieren, werden "echte Tetraden" genannt.

0 = 0000
1 = 0001
2 = 0010
3 = 0011
4 = 0100
5 = 0101
6 = 0110
7 = 0111
8 = 1000
9 = 1001

Beispiele:

3 6 5 = 0011 0110 0101

7 4 9 = ...

1 2 8 = ...

BCH - Binär Codierte Hexadezimal Ziffer/Zahl

Für jede Hexadezimalziffer wird eine vierstellige Binärzahl (Tetrade), welche die Wertigkeit der Hexziffer aufweist verwendet. Für die Ziffern 0 - 9 werden die gleichen Tetraden wie bei BCD verwendet (echte Tetraden). Für die Ziffern A - F die Verbleibenden von 1010 bis 1111 (Pseudotetraden).

A = 1010
B = 1011
C = 1100
D = 1101
E = 1110
F = 1111

Beispiele:

A 3 B = 1010 0011 1011

5 C E = ...

D F 7 = ...

ASCII - Code (7Bit Code)

Im ASCII-Code sind alle Ziffern, alle Buchstaben in Groß - und Kleinschreibung, eine Reihe Sonderzeichen und eine Reihe Steuerzeichen enthalten. Jedes Zeichen ist mit 7Bit codiert, wodurch sich eine Menge von 128 Zeichen ergibt (Codes 0 -127 dez. / 00 - 7F hex.). Der ASCII-Code ist genormt und wird zum Datenaustausch zwischen Datenverarbeitungsgeräten benutzt. Für Deutschland gibt es den Code DIN 66 003, welcher bis auf 8 Zeichen mit dem ASCII-Code identisch ist. Die Codes 0-31 (00-1FH) sind Steuerzeichen für angeschlossene Geräte. Die Codes ab 32 (20H) sind die darstellbaren Zeichen.

Abweichende Zeichen bei DIN 66 003:

HEX	DEZ	ASCII	DIN
40	64	@	§
5B	91	[	Ä
5C	92	\	Ö
5D	93	]	Ü
7B	123	{	ä
7C	124	|	ö
7D	125	}	ü
7E	126	~	ß

ASCII-Tabelle

HEX	DEZ	ASCII	HEX	DEZ	ASCII	HEX	DEZ	ASCII	HEX	DEZ	ASCII
00	0	NUL	20	32	SP	40	64	@	60	96	`
01	1	SOH	21	33	!	41	65	A	61	97	a
02	2	STX	22	34	"	42	66	B	62	98	b
03	3	ETX	23	35	#	43	67	C	63	99	c
04	4	EOT	24	36	$	44	68	D	64	100	d
05	5	ENQ	25	37	%	45	69	E	65	101	e
06	6	ACK	26	38	&	46	70	F	66	102	f
07	7	BEL	27	39	´	47	71	G	67	103	g
08	8	BS	28	40	(	48	72	H	68	104	h
09	9	HT	29	41	)	49	73	I	69	105	i
0A	10	LF	2A	42	*	4A	74	J	6A	106	j
0B	11	VT	2B	43	+	4B	75	K	6B	107	k
0C	12	FF	2C	44	,	4C	76	L	6C	108	l
0D	13	CR	2D	45	-	4D	77	M	6D	109	m
0E	14	SO	2E	46	.	4E	78	N	6E	110	n
0F	15	SI	2F	47	/	4F	79	O	6F	111	o
10	16	DLE	30	48	0	50	80	P	70	112	p
11	17	DC1	31	49	1	51	81	Q	71	113	q
12	18	DC2	32	50	2	52	82	R	72	114	r
13	19	DC3	33	51	3	53	83	S	73	115	s
14	20	DC4	34	52	4	54	84	T	74	116	t
15	21	NAK	35	53	5	55	85	U	75	117	u
16	22	SYN	36	54	6	56	86	V	76	118	v
17	23	ETB	37	55	7	57	87	W	77	119	w
18	24	CAN	38	56	8	58	88	X	78	120	x
19	25	EM	39	57	9	59	89	Y	79	121	y
1A	26	SUB	3A	58	:	5A	90	Z	7A	122	z
1B	27	ESC	3B	59	;	5B	91	[	7B	123	{
1C	28	FS	3C	60	<	5C	92	\	7C	124	|
1D	29	GS	3D	61	=	5D	93	]	7D	125	}
1E	30	RS	3E	62	>	5E	94	^	7E	126	~
1F	31	US	3F	63	?	5F	95	_	7F	127	DEL

SOH = Start of Heading
STX = Start of Text
ETX = End of Text
EOT = End of Transmission
ENQ = Enquiry
ACK = Acknowledge
BEL = Bell

BS = Backspace
HT = Horizontal Tab
LF = Line Feet
VT = Vertical Tab
FF = Form Feet
CR = Cariage Return
SO = Shift Out

SI = Shift In
DLE = Data Link Escape
DC1-4 = Device Control 1-4
NAK = Neg. ACK
ESC = Escape
SP = Space
DEL = Delete

Telexcode Fernschreibcode (5Bit Code)

Beim Telexcode gibt es durch die 5Bit Codierung nur 32 verschiedene Codes. Dies ist für die Codierung aller Ziffern, Buchstaben und Sonderzeichen zu wenig. Der Code wurde deshalb in zwei Bereiche mit jeweils gleichen Codes aber unterschiedlichen zugeordneten Zeichen aufgespalten. Es gibt eine Buchstabenreihe (Bu) und eine Zifferreihe (Zi). Erscheint im Code das Zeichen "Bu", entnimmt ein Empfänger alle folgenden Zeichen für den empfangenen Code aus der Buchstabenreihe, erhält er ein "Zi" entnimmt er alle folgenden Zeichen aus der Ziffernreihe. Es wird also jeweils zwischen Buchstaben- und Ziffernreihe umgeschaltet.

Der Telexcode wird dort eingesetzt, wo auf wenig verfügbaren Speicherplatz möglichst viele Zeichen untergebracht werden müssen (z.B: Scheckkarten). Durch die 5Bit Codierung werden pro Zeichen je 2Bit gegenüber ASCII eingespart. Um effizient zu arbeiten werden die Texte und Zahlen in Blöcken angeordnet, um möglichst selten zwischen den Codereihen umschalten zu müssen.

Bu.	**Zi.**	**Code**
A	-	11000
B	?	10011
C	:	01110
D	WD	10010
E	3	10000
F	so	10110
G	so	01011
H	so	00101
I	8	01100
J	KL	11010
K	(	11110
L	)	01001
M	.	00111
N	,	00110
O	9	00011
P	0	01101
Q	1	11101
R	4	01010
S	‘	10100
T	5	00001
U	7	11100
V	=	01111
W	2	11001
X	/	10111
Y	6	10101
Z	+	10001
	WR	00010
	ZL	01000
	Bu	11111
	Zi	11011
	Zwr	00100
	frei	00000

Beispiel: MEIER 0815

Bu	M	E	I	E	R	Zwr
11111,	00111,	10000,	01100,	10000,	01010,	00100

Zi	0	8	1	5
11011,	01101,	01100,	11101,	00001

KL	= Klingel
Bu	= Buchstabenreihe
Zi	= Ziffernreihe
WD	= Wer Da ?
WR	= Wagenrücklauf
ZL	= Zeilenvorschub
Zwr	= Zwischenraum
so	= Sonderzeichen
frei	= nicht benutzt

1.4 Standardisierte Binärgrößen

Bei Zahlen ist die Anzahl der Stellen maßgeblich am darstellbaren Zahlenbereich beteiligt. Im binären Zahlensystem haben sich einige Datenbreiten mit fester Stellenzahl mit eigenen Namen im Sprachgebrauch eingebürgert.

Breite	Englisch	Deutsch	Zahlenbereich
1Bit	Bit	Bit	0 - 1
4Bit	Nibble	Tetrade	0 - 15
8Bit	Byte	Byte	0 - 255
16Bit	Word	Wort	0 - 65 535
32Bit	Double Word	Doppel Wort	0 - 4 294 967 295
64Bit	Quad Word	Vierfach Wort	0 - $1{,}844674407^{19}$

Die Formel zum Ermitteln der Menge an Zahlen bei einer bestimmten Datenbreite im Binärsystem lautet:

2^n n = Anzahl der Binärstellen

Beispiel: 4Bit $= 2^4$ = 2x2x2x2 = 16 Zahlen (Zahlenbereich 0-15)

1.5 Binäre Arithmetik

Im binären und hexadezimalen Zahlensystem sind prinzipiell alle mathematischen Operationen möglich, welche im dezimalen Zahlensystem möglich sind. Die Mikrocontroller und Mikroprozessoren sind technisch aber nur für einge grundlegende Rechenoperationen ausgelegt (Addition, Subtraktion, Multiplikation und Division). Für komplexere Berechnungen müssen Programme diese durch die einfachen Grundoperationen nachbilden. Für Mikroprozessoren stehen mathematische Coprozessor Bausteine (z.B: 80387) zur Verfügung, welche auch komplexere Rechenaufgaben lösen können.

Binäre Addition:

```
  0 + 0 = 0
  0 + 1 = 1
  1 + 0 = 1
  1 + 1 = 0      +1Übertrag
1 + 1 + 1 = 1    +1Übertrag
```

```
4Bit Beispiel   1 0 1 0 = 10
              + 1 0 1 1 0 =  6
              -------------
                1 0 0 0 0 = 16
```

Binäre Subtraktion:

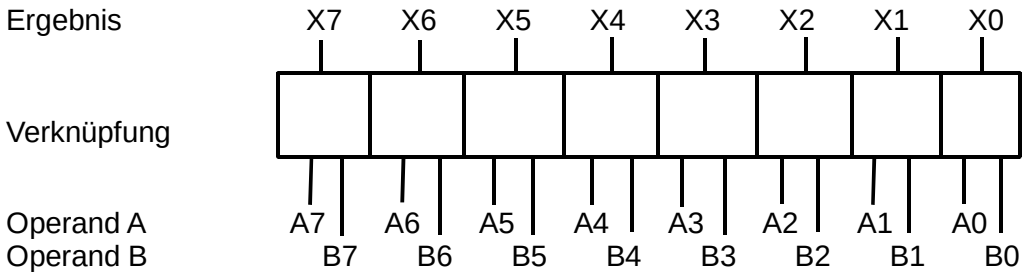
```
  0 - 0 = 0
  1 - 0 = 1
  1 - 1 = 0
  0 - 1 = 1      -1Übertrag
1 - 1 - 1 = 1    -1Übertrag
0 - 1 - 1 = 0    -1Übertrag
```

```
4Bit Beispiel   1 0 1 0 = 10
              -   0 1 1 0 =  6
              -------------
                0 1 0 0 =  4
```

1.6 Logische Verknüpfungen

Die CPU´s der Mikrocontroller und Mikroprozessoren stellen in ihren Befehlssätzen Operationen für die logische UND, ODER, EXCLUSIV-ODER und NICHT Verknüpfung zur Verfügung. Die logischen Verknüpfungen erfüllen in steuerungstechnischen Programmen, vor allem bei Mikrocontrollern, wichtige Aufgaben.

Die logischen Verknüpfungen werden bevorzugt auf Byte-Operanden angewandt. Dabei werden zwei 8Bit-Operanden miteinander zu einem 8Bit-Ergebnis verknüpft. Es werden immer nur die Bit gleicher Wertigkeit miteinander verknüpft.

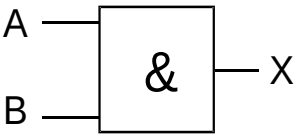


Durch die Verknüpfung eines Datenbytes mit einem Maskenbyte können einzelne oder mehrere Bit des Datenbytes gleichzeitig beeinflußt werden. Die Art der Beeinflussung wird durch die verwendete Verknüpfung vorgegeben.

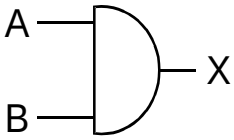
UND Verknüpfung	-	Bit rücksetzen	(Ausschalten)
ODER Verknüpfung	-	Bit setzen	(Einschalten)
EX-ODER Verknüpfung	-	Bit invertieren	(Umschalten)

Die UND-Verknüpfung (Konjunktion)

Logik Symbol:



DIN



Amerikanisch

Funktionsgleichung:
A ∧ B = X

Wahrheitstabelle:

B	A	X
0	0	0
0	1	0
1	0	0
1	1	1

Beschreibung:

Der Ausgang ist nur "1" wenn alle Eingänge "1" sind.

Anwendung: Zum Maskieren von Datenwerten

Um ein oder mehrere Bit eines beliebigen Datenwertes gezielt in den Zustand "0" zu bringen. Jedes Bit das im Maskenwert "0" ist, ist im Ergebnis "0". Jedes Bit das im Maskenwert 1 ist, ist im Ergebnis unverändert.

Steuerungstechnische Anwendung - **Ausschalten**

Beispiel:

Der 8Bit Datenwert wird bitweise mit dem 8Bit Maskenwert zu einem 8Bit Ergebnis verknüpft. Dabei wird Daten 0 mit Masken 0, Daten 1 mit Masken 1, ... Daten 7 mit Masken 7 verknüpft.

```
  0 1 0 1  0 1 0 1 - Datenwert
∧ 0 0 0 0  1 1 1 1 - Maskenwert
  0 0 0 0  0 1 0 1 - Ergebnis
```

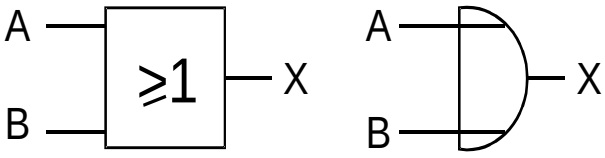
Übung:

```
  1 0 1 0  1 0 1 0
∧ 1 0 0 1  0 1 1 1
```

Die oberen 4 Bit werden mit Hilfe der logischen UND-Verknüpfung rückgesetzt. Die unteren 4 Bit bleiben unverändert.

Die ODER-Verknüpfung (Disjunktion)

Logik Symbol:



DIN Amerikanisch

Funktionsgleichung:

A ∨ B = X

Wahrheitstabelle:

B	A	X
0	0	0
0	1	1
1	0	1
1	1	1

Beschreibung:

Der Ausgang ist nur "0" wenn alle Eingänge "0" sind.

Anwendung: Zum Maskieren von Datenwerten

Um ein oder mehrere Bit eines beliebigen Datenwertes gezielt in den Zustand "1" zu bringen. Jedes Bit das im Maskenwert "1" ist, ist im Ergebnis "1". Jedes Bit das im Maskenwert 0 ist, ist im Ergebnis unverändert.

Steuerungstechnische Anwendung - **Einschalten**

Beispiel:

1 0 1 0 1 0 1 0 - Datenwert
∨ 0 0 0 0 1 1 1 1 - Maskenwert
1 0 1 0 1 1 1 1 - Ergebnis

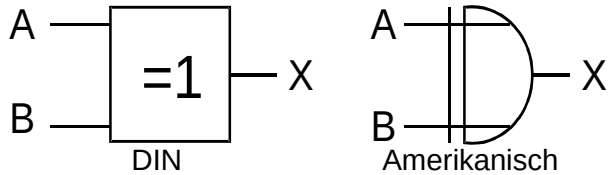
Übung:

0 1 0 1 0 1 0 1
∨ 1 0 0 1 0 1 1 0

Die unteren 4 Bit werden mit Hilfe der logischen ODER-Verknüpfung gesetzt. Die oberen 4 Bit bleiben unverändert.

Die EXCLUSIV-ODER Verknüpfung (Antivalenz)

Logik Symbol:



DIN Amerikanisch

Funktionsgleichung:

A ≢ B = X

Wahrheitstabelle:

B	A	X
0	0	0
0	1	1
1	0	1
1	1	0

Beschreibung:

Der Ausgang ist nur "1" wenn die Eingänge unterschiedliche Zustände haben.

Anwendung: Zum Maskieren von Datenwerten

Um ein oder mehrere Bit eines beliebigen Datenwertes gezielt zu invertieren. Jedes Bit das im Maskenwert "1" ist, ist im Ergebnis invertiert. Jedes Bit das im Maskenwert 0 ist, ist im Ergebnis unverändert.

Steuerungstechnische Anwendung - **Umschalten**

Beispiel:

1 0 1 0 1 0 1 0 - Datenwert
≠ 0 0 0 0 1 1 1 1 - Maskenwert
1 0 1 0 0 1 0 1 - Ergebnis

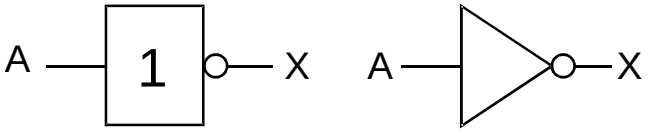
Übung:

0 1 0 1 0 1 0 1
≠ 1 0 0 1 0 1 1 0

Die unteren 4 Bit werden mit Hilfe der logischen EX-ODER-Verknüpfung invertiert. Die oberen 4 Bit bleiben unverändert.

Die NICHT-Verknüpfung (Negation)

Logik Symbol:



DIN Amerikanisch

Funktionsgleichung:
$\overline{A} = X$

Wahrheitstabelle:

A	X
0	1
1	0

Beschreibung:

Der Ausgang hat immer den invertierten (umgekehrten) Zustand des Eingangs.

Anwendung:

Um einen ganzen Operanden zu invertieren.

2 Mikrocontrollersysteme

Mikrocontroller werden heute in allen Bereichen der Elektronik eingesetzt. Sie verrichten ihre Arbeit in SPS-Anlagen (z.B: Kraftwerk, Rolltreppensteuerung), in Steuergeräten von Fahrzeugen (z.B: U-Bahn, Straba, Omnibus, KFZ) usw. ohne dabei nach aussen als Steuercomputer in Erscheinung zu treten. Sie besitzen meist nur wenige Bedien- und Anzeigeelemente (Schalter, Tasten, Leuchtdioden) oder arbeiten völlig selbständig ohne äußere Bedienbarkeit. Dem Benutzer ist meist nicht bewußt, daß er beim Einschalten des Gerätes oft mehrere Steuercomputer mit Mikrocontroller in Betrieb nimmt. Als Beispiel für moderne Controller-Steuerung ist hier ein Kraftfahrzeug gewählt. In einem KFZ können heute bis zu 50 Mikrocontroller gleichzeitig für die verschiedensten Steuer-, Regel- und Überwachungsaufgaben eingesetzt sein.

Steuergerät	Motronic (Digitale Motorelektronik - Bosch) (Digitale Diesel Elektronic - Bosch) (Motor Steuerung - Siemens)
Eingangsgrößen	Drehzahl, Kurbelwellensteuerung, Kühlmittel- und Ansaugluft-Temperatur, Luftmenge/-masse, Spannung der Lambdasonde, Abgastemperatur
Ausgangsgrößen	Einspritzsignal für Einspritzventile, Zündsignal für Zündkerzen
Systemfunktion	Regelung des Verbrennungsmotors

Steuergerät	Anti-Blockier-System ABS Automatische Stabilitätskontrolle ASC Motorschleppmomentregelung und Anti-Schlupfregelung ASR
Eingangsgrößen	Drehzahl der Räder von den Radsensoren, Betätigung von Gaspedal und Bremse
Ausgangsgrößen	Info an Einspritzelektronik, Motorregelung und Bremsen
Systemfunktion	Radschlupf soll in allen Fahrsituationen verhindert werden

Steuergerät	Karrosserieelektronik
Eingangsgrößen	Schalterstellung von Türschloß-, Sitzverstell-, Schiebedach-, Wisch/Wasch-Schalter Fahrgeschwindigkeitssignal
Ausgangsgrößen	Signal zur Ansteuerung der Motoren für Fensterheber-, Wisch/ Wasch-,Sitzverstellmotoren, usw.
Systemfunktion	- Türen verriegeln - Fensterheber - Sitzverstellung - Schiebedach - Scheiben wischen/waschen - Scheinwerfer reinigen - Anpreßdruck der Scheibenwischer - Stromüberwachung

Steuergerät	Instrumentenkombination
Eingangsgrößen	Analoge und digitale Signale von Sensoren: Temperatur, Druck, Geschwindugkeit, u.a.
Ausgangsgrößen	Analoge und digitale Anzeigen von Geschwindigkeit, Drehzahl, Öldruck, Öltemperatur, Momentanverbrauch und Statusan-zeigen
Systemfunktion	Information des Fahrers

Steuergerät	Heizungsregelung, Klimaanlage
Eingangsgrößen	Temperatur von Innenraum- und Klimakompressor Sollwerte
Ausgangsgrößen	Ansteuerung der Motoren für die Lüftungsklappen
Systemfunktion	Klimaregelung des Innenraums

Steuergerät	Tempomat
Eingangsgrößen	Ist-/Sollwert der Geschwindigkeit
Ausgangsgrößen	Signale an EML bzw. Motorsteuerung
Systemfunktion	Geschwindigkeitsanpassung

Steuergerät	Diebstahlwarnanlage
Eingangsgrößen	Tür-, Heckklappen-, Motorraumkontakte, Radsensoren, Glasbruchmelder, Ultraschallsensoren
Ausgangsgrößen	Signal zur Startblockierung an die Motorsteuerung Alarmauslösung
Systemfunktion	Diebstahlschutz

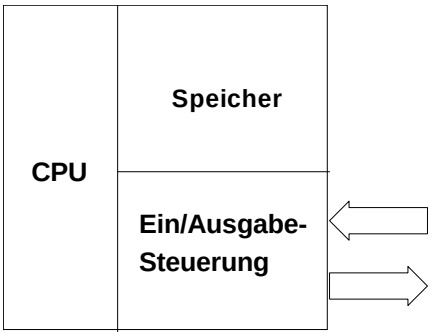
Steuergerät	Airbag
Eingangsgrößen	Crashsensor
Ausgangsgrößen	Signal zur Aktivierung der Zündkapseln
Systemfunktion	Schutz der Insassen

Steuergerät	Elektronische Dämpferverstellung
Eingangsgrößen	Radsensoren, Lenkradwinkelsensor, Geschwindigkeitssignal
Ausgangsgrößen	Elektrische Signale für die Hydraulikventile
Systemfunktion	Erhöhte Fahrsicherheit

Steuergerät	Bordcomputer, Multiinformationsdisplay
Eingangsgrößen	Tasten, Temperatur-Sensor, Signale von anderen Steuergeräten
Ausgangsgrößen	Display
Systemfunktion	Informationen über Wegstrecke, Durchschnittsgeschwindigkeit, Datum, Uhrzeit, Programmierung der Standheizung/-lüftung, u.a.

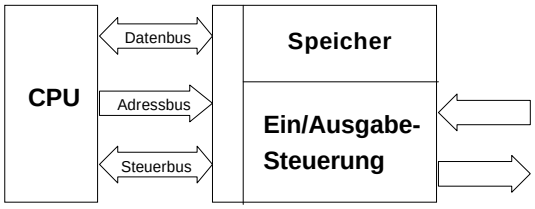
3 Struktur eines Mikrocontrollers

Ein Computersystem besteht immer aus drei Grundkomponenten. Der CPU (Central Processing Unit / Zentrale Recheneinheit), dem Speicher und einer Ein/Ausgabesteuerung. Aufgabe der CPU ist es die Befehle eines Programms aus dem Speicher zu lesen und auszuführen. Der Speicher beinhaltet einerseits die Befehle eines Programms und kann andererseits die Daten für die Programme aufnehmen. Über die Schnittstellen der Ein/Ausgabesteuerung tritt der Computer mit seiner Umgebung in Verbindung.



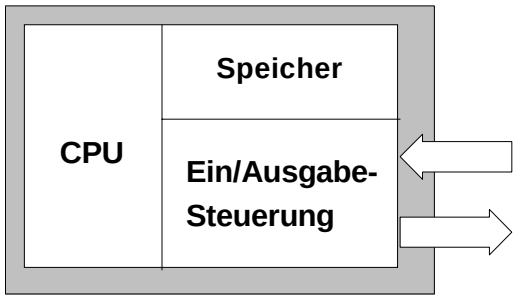
Blockschaltbild Computer

Bei den Mikrocomputern sind die drei Grundkomponenten des Systems durch getrennte, eigene Bausteine realisiert. Die Komponenten sind auf einer oder mehreren Platinen untergebracht. Sie sind über ein Bussystem, das aus Daten-, Adress- und einem Steuerbus besteht, miteinander verbunden. Durch das frei zugängliche Bussystem kann ein Mikrocomputer, in gewissen Grenzen, an viele Anwendungsfälle angepasst werden.



Blockschaltbild Mikrocomputer

Bei den Mikrocontrollern sind die drei Grundkomponenten in einem Baustein integriert. Das Bussystem ist nicht oder nur eingeschränkt zugänglich. Dadurch ist bedingt, daß ein Mikrocontrollersystem nicht oder nur schwierig erweiterbar ist. Reichen die Fähigkeiten eines Mikrocontrollers für eine bestimmte Aufgabe nicht aus, so wird meist nicht das System erweitert, sondern auf einen Mikrocontroller mit den entsprechenden Fähigkeiten zurückgegriffen. Daraus folgt, daß ein komplettes Mikrocontrollersystem oft nur aus einem einzigen Baustein besteht.



Blockschaltbild Mikrocontroller

Mikrocomputer
- leicht erweiterbar
- relativ hoher Platzbedarf
- relativ hohe Kosten

Mikrocontroller
- schwierig erweiterbar
- niedriger Platzbedarf
- Kostengünstig

3.1 Die Mikrocontrollerfamilie MCS51

Die Mikrocontrollerfamilie MCS51 basiert auf dem 8Bit Industrie Standard-Mikrocontroller 8051. Diese beinhaltet eine ganze Reihe unterschiedlicher Mikrocontroller von verschiedenen Herstellern. Die Familie wird laufend um neue Bausteine ergänzt. Alle diese unterschiedlichen Bausteine besitzen die gleiche CPU und den gleichen Grundaufbau. Sie unterscheiden sich hauptsächlich durch die mitintegrierte Hardware. Der Befehlssatz und die Programmierung sind bei allen Bausteinen der Familie gleich. Dadurch ist gewährleistet, daß die Programmerstellung mit immer den gleichen Entwicklungswerkzeugen (Assembler, Compiler, Debugger, usw.) durchgeführt werden kann. Die Programmierung und der Umgang mit den Entwicklungswerkzeugen muß nur einmal für den Grundtyp (8051) erlernt werden. Für die erweiterten Bausteine muß nur jeweils der Umgang mit der zusätzlichen Hardware dazugelernt werden. Wegen der Vielfalt der vorhandenen Bausteine werden in diesen Unterlagen nur die drei wichtigsten Typen 8051, 8052 und 80515 besprochen.

8051

4kByte Programmspeicher

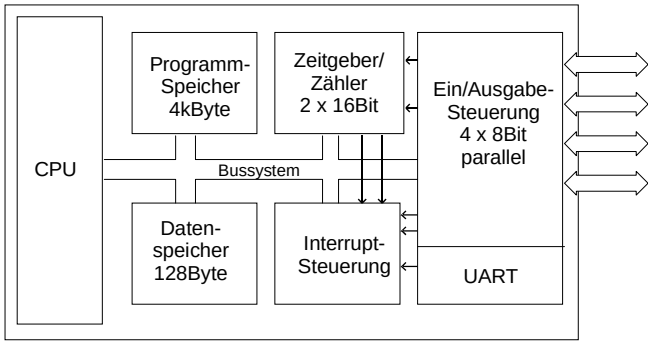
128Byte Datenspeicher

2 16Bit Zeitgeber/Zähler

2 externe und 3 interne Interrupts

32 Ein/Ausgabebit in 4 8Bit-Ports

1 UART synchron/asynchron vollduplex



8052

8kByte Programmspeicher

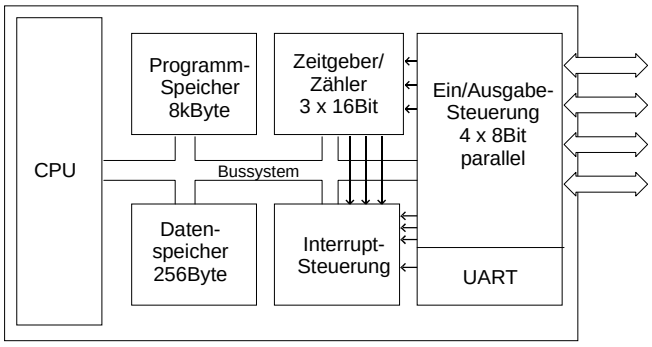
256Byte Datenspeicher

3 16Bit Zeitgeber/Zähler

3 externe und 4 interne Interrupts

32 Ein/Ausgabebit in 4 8Bit-Ports

1 UART synchron/asynchron vollduplex



80515

8kByte Programmspeicher

256Byte Datenspeicher

3 16Bit Zeitgeber/Zähler

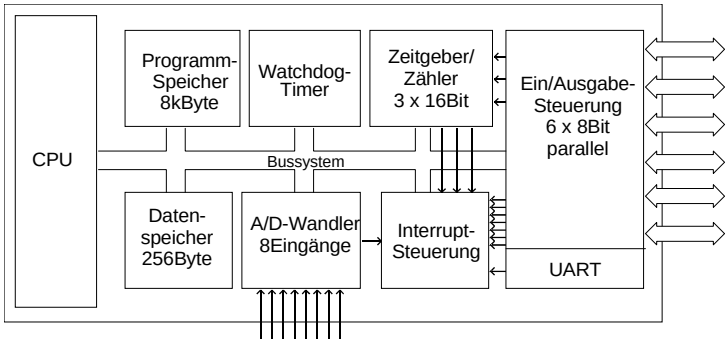
7 externe und 5 interne Interrupts

48 Ein/Ausgabebit in 6 8Bit-Ports

1 16Bit Watchdog-Timer

1 UART synchron/asynchron vollduplex

1 8Bit A/D-Wandler mit 8 gemultiplexten Eingängen



Zur Unterscheidung der einzelnen Bausteine wurde ein Bezeichnungsschema eingeführt, mit dessen Hilfe es einfach ist die Bausteine zu unterscheiden. Leider ist dieses Schema nicht bei allen Bausteinen und Herstellern durchgängig, weshalb im Zweifelsfall die Datenblätter zu Rate gezogen werden müssen.

805x Maskenprogrammiert

875x EPROM Version

803x Romlose Version

8xCxx CMOS Version

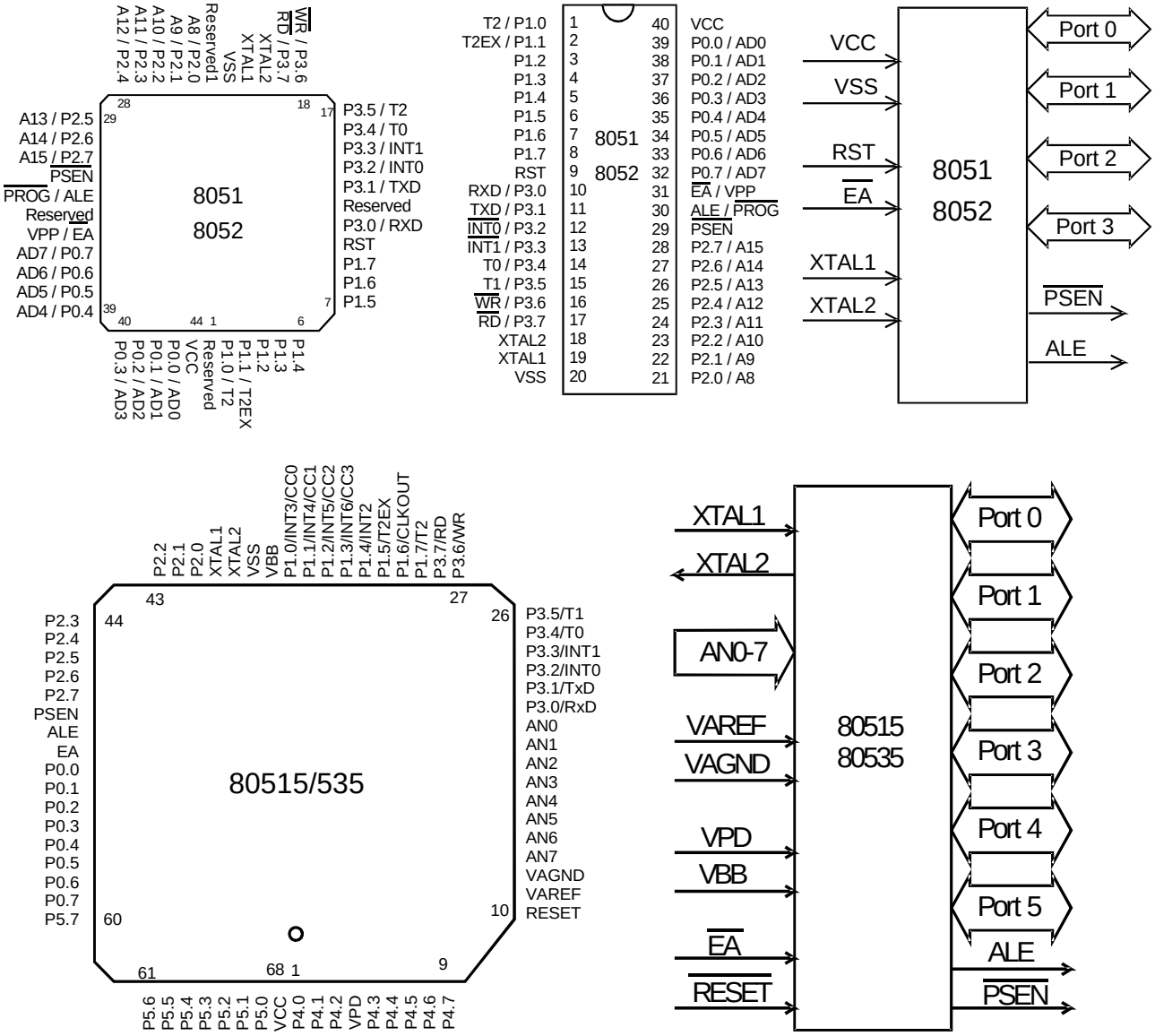
Maskenprogrammiert: Das Programm für den Mikrocontroller wird bereits bei der Herstellung fest im Baustein abgelegt. Nachträgliche Änderungen sind nicht mehr möglich. Diese Art der Programmierung ist für Großserien zu bevorzugen, da sie zu niedrigen Kosten für das einzelne Gerät führt.

Eprom: Ein Eprom ist ein programmierbarer/löschbarer Festwertspeicher (ROM). Der Baustein wird ohne Programm erworben. Er kann vom Anwender in einem speziellen Programmiergerät beschrieben und mit einem Löschgerät wieder gelöscht werden. Da die Eprom-Versionen der Mikrocontroller verhältnismäßig teuer sind, werden sie überwiegend in der Test- und Prototypenphase eingesetzt.

Romlos: Die romlosen Mikrocontroller sind Bausteine deren interner Programmspeicher nicht nutzbar ist. Sie müssen immer mit einem externen Programmspeicher betrieben werden.

CMOS: Die CMOS-Bausteine weisen gegenüber den NMOS-Bausteinen eine deutlich geringere Stromaufnahme auf. Unter den CMOS-Bausteinen befinden sich auch solche, deren Programmspeicher als Flash-Eprom ausgeführt ist (z.B: 80C1051, 80C2051). Bausteine mit einem Flash-Eprom können in speziellen Geräten oder in der Schaltung gelöscht und programmiert werden.

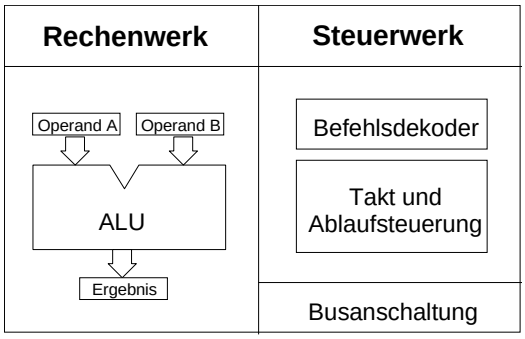
3.1.2 Sockelschaltbilder 8051, 8052 und 80515



4 Beschreibung der Hardware

4.1 CPU - Zentrale Rechen Einheit

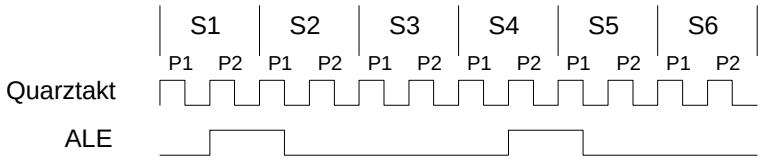
Die CPU der MCS-51 Mikrocontroller hat eine Verarbeitungsbreite von 8Bit. Daraus folgt, daß die Systemkomponenten (Speicher, Ein/Ausgabe) mit welchen die CPU zusammenarbeitet, auch für eine Datenbreite von 8Bit ausgelegt sind. Die CPU besteht im wesentlichen aus einem Steuerwerk und einem Rechenwerk.Im Steuerwerk befindet sich der Befehlsdekoder und die Takt- und Ablaufsteuerung. Der Befehlsdekoder hat die Aufgabe die Bitmuster der Befehle zu entschlüsseln und die Ablaufsteuerung dazu zu bewegen, alle nötigen Schritte durchzuführen um den Befehl auszuführen. Im Rechenwerk befindet sich die ALU (Arithmetic Logic Unit - Arithmetisch Logische Einheit). Sie kann zwei 8Bit breite Operanden zu einem 8Bit (9Bit mit Carry) breiten Ergebnis verarbeiten. Als Rechenoperationen stehen Addition, Subtraktion, Division und Multiplikation, als logische Operationen die UND, ODER, EXCLUSIV-ODER und NICHT Verknüpfung zur Verfügung. Zudem kann die ALU eine Reihe von Rotationsoperationen durchführen.



Blockschaltbild CPU

Die Arbeit der CPU ist fest an einen Takt gebunden. Ihre Arbeitsgeschwindigkeit ist maßgeblich von der Taktgeschwindigkeit abhängig. Die Taktgeschwindigkeit darf bei den meisten Bausteinen der MCS-51 zwischen 1,2MHz und 18MHz liegen. Es gibt aber auch neuere Bausteine deren Taktfrequenz bis zu 60MHz betragen darf. Der Quarz zur Takterzeugung wird direkt an den Mikrocontroller angeschlossen. Der Oszillator ist im Baustein integriert. Für Anwendungen in denen die Taktstabilität eine untergeordnete Rolle spielt, kann ein etwas preiswerterer Keramikschwinger eingesetzt werden.

Die CPU unterteilt ihren Takt in Maschinenzyklen (Zycles), Zustände (States) und Phasen (Phases). Eine Phase ist eine Taktperiode, ein State sind zwei Phasen, ein Zyklus sind sechs States (12 Taktperioden). Die kleinste Einheit zur Befehlsausführung ist ein Maschinenzyklus. Die meisten Befehle sind so ausgelegt, daß sie in einem oder zwei Zyklen ausgeführt werden können.



Taktschema

Durch die starre Kopplung der Befehlsausführung an Maschinenzyklen kann die Laufzeit jedes einzelnen Befehls exakt bestimmt werden.

Beispiel:	Taktfrequenz	= 12MHz
	Zyklusfrequenz	= Taktfrequenz / 12 = 12MHz / 12 = 1MHz
	Zykluszeit	= 1µs / Zyklusfrequenz(MHz) = 1µs / 1MHz = 1µs
	Befehl:	MOV dadr,#ko (2 Zyklen)
	Laufzeit eines Befehls	= Anzahl Zyklen * Zykluszeit = 2 * 1µs = 2µs

In jedem Zyklus wird zweimal auf den Speicher zugegriffen, wobei bei jedem Zugriff das Signal ALE (Adress Latch Enable) am entsprechenden Pin des Mikrocontrollers erscheint. Somit tritt ALE in jedem Zyklus zwei mal auf, wodurch es als Hilfstakt (Quarz / 6) für Peripherie-Bausteine verwendet werden kann.

Ausnahme: Bei Zugriffen auf den externen Datenspeicher erscheint ALE nur ein mal je Zyklus.

Da die meisten Systeme jedoch ohne externen Datenspeicher arbeiten, spielt diese Einschränkung nur eine untergeordnete Rolle.

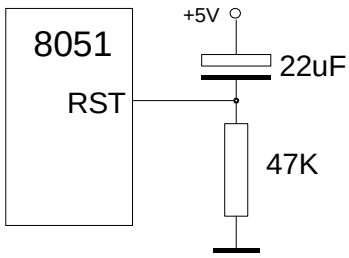
Hinweis für Praktiker:

Da das Signal ALE bei laufender CPU periodisch erscheint, kann daran der generelle Systemzustand abgelesen werden. Bei Messung mit dem Oszilloskop können zwei Fehlerzustände ermittelt werden.

ALE nicht vorhanden	- CPU arbeitet nicht (möglicherweise defekt)
Frequenz zu hoch	- Quarz schwingt auf einer Oberwelle
	(Dadurch ist der Arbeitstakt zu hoch)
	Sollfrequenz = Quarz / 6

4.1.2 Reset

Die CPU kann durch Reset in einen definierten Ausgangszustand versetzt werden. Nach Reset zeigt der Befehlszeiger (PC) auf die Adresse 0000h, Interrupts sind gesperrt und die Controllerhardware wird durch die SFR (Special Funktion Register) in einen definierten Grundzustand versetzt. beim Einschalten der Versorgungsspannung muß nach dem Erreichen von 4,75 Volt noch für 24 Taktzyklen der Reseteingang aktiv (High) gehalten werden, um ein sicheres Arbeiten des Controllers zu gewährleisten. Dies wird in der Praxis durch ein RC-Glied am Reseteingang erreicht.



Die Programmbearbeitung beginnt nach dem Einschalten oder Reset im Programmspeicher immer bei Adresse 0000h.

PC	Befehlszeiger	0000H	A	Akkumulator	00H
B	Hilfsakkumulator	00H	PSW	Programm Status Wort	0000 0000B
SP	Stackpointer	07H	DPL	Datenpointer Low	00H
DPH	Datenpointer High	00H	P0-P3	Port 0-3	1111 1111B
TMOD	Timer Modus	0000 0000B	TCON	Timer Control	0000 0000B
TL0	Timer 0 Zählregister Low	00H	TH0	Timer 0 Zählregister High	00H
TL1	Timer 1 Zählregister Low	00H	TH1	Timer 1 Zählregister High	00H
IE	Interrupt Enable	0XX0 0000B	IP	Interrupt Priority	XXX0 0000B
SCON	Serial Control	0000 0000B	PCON	Power Control NMOS	0XXX XXXXB
				CMOS	0XXX 0000B

Zustände der SFR nach Reset

Die CPU stellt dem Programmierer einen Befehlssatz bereit, mit dessen Befehlen die Programme aufgebaut werden können. Ein Programm besteht aus einer sinnvollen Aneinanderreihung von Befehlen, um eine bestimmte Aufgabe zu erfüllen. Die CPU der MCS-51 Familie ist immer die gleiche. Dies bedeuted daß Software, die für den 8051 entwickelt wurde, auch mit einem 8052 oder 80535 funktioniert. Programme für den 8051 können mit einem 8051 Assembler, einem Pascal- oder C-Compiler erstellt werden. Dabei werden die Befehle der Hochsprache durch die Compiler in Maschinenbefehle übersetzt. Jeder Maschinenbefehl hat zwei Formen, die Binäre (für die CPU) und die Mnemonische (für den Menschen). In der folgenden Tabelle sind nur die Mnemonischen Befehle aufgeführt. Die zugehörigen binären Befehle werden durch den Assembler automatisch erzeugt.

Allgemeine Transferbefehle

	A	Rr	dadr	#ko.	@Ri
MOV A,	-	*	*	*	*
MOV Rr,	*	-	*	*	-
MOV dadr,	*	*	*	*	*
MOV @Ri,	*	-	*	*	-
MOV DPTR,	-	-	-	*16	-

* = möglich

- = nicht

*16 = 16Bit Konstante

Spezielle Transferbefehle

Ext. Programmspeicher:	MOVC A,@A+PC	(A <- <<A+PC>>)
	MOVC A,@A+DPTR	(A <- <<A+DPTR>>)
Externer Datenspeicher:	MOVX A,@Ri	(A <- <<Ri>>)
	MOVX A,@DPTR	(A <- <<DPTR>>)
	MOVX @Ri,A	(<Ri> <- <A>)
	MOVX @DPTR,A	(<DPTR> <- <A>)

Stack:	PUSH dadr	(Stack <- <dadr>)
	POP dadr	(dadr <- <Stack>)
Tauschbefehle:	XCH A,Rr	(<A> <-> <Rr>)
	XCH A,dadr	(<A> <-> <dadr>)
	XCH A,@Ri	(<A> <-> <<Ri>>)
	XCHD A,@Ri	(<A2^0-2^3> <-> <<Ri 2^0-2^3>>)
	SWAP A	(<A2^0-2^3> <-> <A2^4-2^7>)

Logik-Befehle

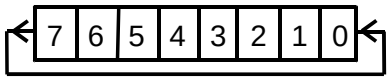
	A	Rr	dadr	#ko.	@Ri	Flags
ANL A,	-	*	*	*	*	P
ANL dadr,	*	-	-	*	-	-
ORL A,	-	*	*	*	*	P
ORL dadr,	*	-	-	*	-	-
XRL A,	-	*	*	*	*	P
XRL dadr,	*	-	-	*	-	-

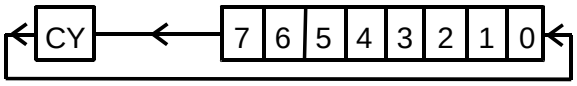
ANL Ziel,Quelle	(<Ziel> UND <Quelle> -> Ziel)
ORL Ziel,Quelle	(<Ziel> ODER <Quelle> -> Ziel)
XRL Ziel,Quelle	(<Ziel> EX-OR <Quelle> -> Ziel)

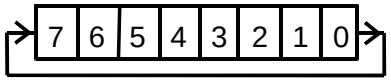
Spezielle Logik-Befehle

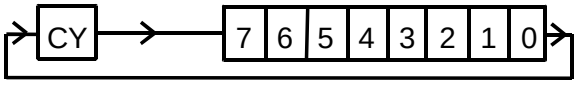
CLR A	00 -> A	Flags: P
CPL A	<A> NICHT -> A	Flags: -

Rotier-Befehle

RL A  AKKU Flags: -

RLC A  AKKU Flags: CY, P

RR A  AKKU Flags: -

RRC A  AKKU Flags: CY, P

Arithmetik-Befehle

	Rr	dadr	#ko.	@Ri	Funktion	
ADD A,	*	*	*	*	<A> + X -> A	Flags: OV,P
ADDC A,	*	*	*	*	<A> + X + CY -> A	Flags: OV,P
SUBB A,	*	*	*	*	<A> - X - CY -> A	Flags: OV,P

	A	Rr	dadr	@Ri	DPTR	Funktion	
INC	*	*	*	*	*	<X> + 1 -> X	Flags: -
DEC	*	*	*	*	-	<X> - 1 -> X	INC A / DEC A: P

MUL AB <A> x <B> -> A (LSB) und B (MSB) Flags: Cy,OV,P

DIV AB <A> / <B> -> A (Rest in B) Flags: Cy,OV,P

DA A Dezimalkorrektur des <A> nach Addition Flags: Cy,AC,P

Bitverarbeitungs-Befehle

CLR C 0 -> Cy Flags: P

CLR badr 0 -> Bit (falls Akkubit) Flags:P

SETB C 1 -> Cy Flags: Cy

SETB badr 1 -> Bit Flags: -

CPL C	<Cy> NICHT -> Cy	Flags: Cy
CPL badr	<Bit> NICHT -> Bit	Flags:P
ANL C,badr	<Cy> UND <badr> -> Cy	Flags: Cy
ANL C,/badr	<CY> UND <badr (inv.)> -> Cy	Flags: Cy
ORL C,badr	<Cy> ODER <badr> -> Cy	Flags: Cy
ORL C,/badr	<Cy> ODER <badr (inv.)> -> Cy	Flags: Cy
MOV C,badr	<badr> -> Cy	Flags: Cy
MOV badr,C	<Cy> -> badr	Flags: -

Unterprogramm-Befehle

ACALL adr11	<PC+2> -> Stack, adr11 -> PC 2^0-2^{10}	Flags: -
LCALL adr16	<PC+3> -> Stack, adr16 -> PC	Flags: -
RET	<Stack> -> PC	Flags: -
RETI	<Stack> -> PC, Interrupt freigeben	Flags: -

Sprungbefehle

AJMP adr11	adr11 -> PC 2^0-2^{10}	Flags: -
LJMP adr16	adr16 -> PC	Flags: -
SJMP rel	<PC+2> ± rel	Flags: -
JMP @A+DPTR	<A> + <DPTR> -> PC	Flags: -
JZ rel	Falls <A> = 00: <PC+2> ± rel -> PC Falls <A> <> 00: <PC+2> -> PC	Flags: -
JNZ rel	Falls <A> <> 00: <PC+2> ± rel -> PC Falls <A> = 00: <PC+2> -> PC	Flags: -
JC rel	Falls <Cy> = 1: <PC+2> ± rel -> PC Falls <CY> = 0: <PC+2> -> PC	Flags: -
JNC rel	Falls <Cy> = 0: <PC+2> ± rel -> PC Falls <Cy> = 1: <PC+2> -> PC	Flags: -
JB badr,rel	Falls <badr> = 1: <PC+2> ± rel -> PC Falls <badr> = 0: <PC+2> -> PC	Flags: -

JNB badr,rel — Falls <badr> = 0: <PC+2> ± rel -> PC
Falls <badr> = 1: <PC+2> -> PC
Flags: -

JBC badr,rel — Wie JB aber das Bit wird gelöscht

	dadr,	#ko.,	
CJNE A,	*	*	rel
CJNE Rr,	-	*	rel
CJNE @Ri,	-	*	rel

Falls ungleich: <PC+2> ± rel -> PC
Falls gleich: <PC+2> -> PC
Flags: Cy

Mögliche Formen:
CJNE A,dadr,rel
CJNE A,#ko.,rel
CJNE Rr,#ko.,rel
CJNE @Ri,#ko.,rel

DJNZ Rr,rel — Falls <Rr> - 1 <> 00: <PC+2> ± rel -> PC
Falls <Rr> - 1 = 00: <PC+2> -> PC
Flags: -

DJNZ dadr,rel — Falls <dadr> - 1 <> 00: <PC+2> ± rel -> PC
Falls <dadr> - 1 = 00: <PC+2> -> PC
Flags: P, falls dadr = A

Sonder-Befehl

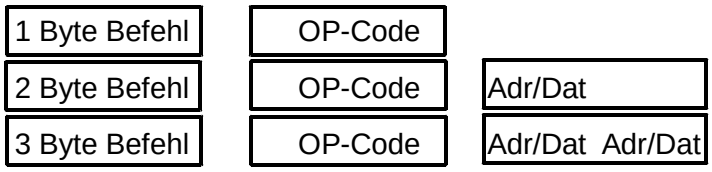
NOP — Leerbefehl (No OPeration)

Verwendete Kurzzeichen:

Rr	Register (R0..R7)
dadr	direkt adressierbare Speicherzelle
@Ri	indirekt adressierbare Speicherzelle (durch R0 oder R1)
#ko.	Konstante (8-Bit)
badr	Bitadresse
adr11	11-Bit Adresse (Bereich 2K)
adr16	16-Bit Adresse (Bereich 64K)
rel	relative 8-Bit Adresse (Bereich +127 bis -128)
< >	Verweist auf den Inhalt z.B: <A> = Inhalt des Akku

4.1.4 Befehlsausführung

Der Befehlssatz der CPU beinhaltet ein-, zwei- und dreibyte Befehle. Das erste oder, bei einbyte Befehlen Einzige, Byte ist immer der Operationscode (OP-Code). Der OP-Code ist das Steuerbitmuster, welches alle zur Ausführung des Befehls durch die CPU benötigten Informationen enthält. Die Informationen sind innerhalb des OP-Code binär codiert. Das oder die folgenden Byte sind zusätzliche Adress- oder Dateninformationen die zur Ausführung des Befehls benötigt werden.



Beim Neustart des Mikrocontrollers (Einschalten oder Reset) zeigt der Befehlszeiger auf die Programmspeicheradresse 0000h. Die Ablaufsteuerung der CPU generiert nun ein Speicher lesen mit der Adresse auf die der Befehlszeiger zeigt. Die Programmbearbeitung beginnt also bei Adresse 0000h im Programmspeicher. Das Byte, das sich in dieser Speicherzelle befindet wird über den Datenbus in den Befehlsdekoder der CPU gebracht. Dort wird das Bitmuster dekodiert und der entsprechende Befehl ausgeführt. Zusätzlich wird im gleichen Maschinenzyklus das Folgebyte aus dem Speicher gelesen und innerhalb der CPU gespeichert (zwei Speicherzugriffe je Zyklus). Handelt es sich bei dem gelesenen Bitmuster um einen einbyte Befehl, wird das zweite gelesene Byte verworfen. Handelt es sich um einen zweibyte Befehl, befindet sich das zweite Byte bereits in der CPU. Dies verkürzt die Ausführungszeit der zweibyte Befehle, sie können in einem Zyklus ausgeführt werden. Handelt es sich um einen dreibyte Befehl, muß zur Befehlsausführung ein zweiter Maschinenzyklus folgen. Im zweiten Zyklus werden wieder zwei Byte aus dem Speicher gelesen, wovon das zweite Byte grundsätzlich wieder verworfen wird. Durch das Lesen von jeweils zwei Byte je Zyklus wird die Arbeitsgeschwindigkeit bei Zweibytebefehlen erhöht, da das zweite Byte des Befehls nicht in einem eigenen Zyklus aus dem Speicher gelesen werden muß. Das Lesen und Verwerfen der nicht benötigten Byte wurde gewählt um die Ausführungszeiten der Befehle, unabhängig davon ob sich der OP-Code im Speicher oder bereits innerhalb der CPU befindet, immer gleich zu halten. Nur dadurch ist es auf einfache Art möglich die Laufzeiten der Befehle exakt zu bestimmen.

1. Befehl MOV A,#55h - 74 55 - Zweibyte Befehl

1. OP-Code (74) in den Befehlsdekoder lesen

2. Folgebyte (55) in einen Zwischenspeicher lesen

3. Befehl ausführen (Folgebyte nutzen) PC um zwei erhöhen.

Ausführungszeit: 1 Zyklus

2. Befehl MOV A,R0 - E8 - Einbyte Befehl

1. OP-Code (E8) in den Befehlsdekoder lesen

2. Folgebyte (90) in einen Zwischenspeicher lesen

3. Befehl ausführen (Folgebyte verwerfen) PC um eins erhöhen.

Ausführungszeit: 1 Zyklus

Adr.	Code	Mnemonic
0006	00	NOP
0005	34	
0004	12	MOV DPTR,#1234
0003	90	
0002	E8	MOV A,R0
0001	55	
0000	74	MOV A,#55

3. Befehl MOV DPTR,#1234 - 90 12 34 - Dreibyte Befehl

1. OP-Code (90) in den Befehlsdekoder lesen

2. Folgebyte (12) in einen Zwischenspeicher lesen

3. Zwei Folgebyte (34 00) in einen Zwischenspeicher lesen

4. Befehl ausführen (letztes Folgebyte (00) verwerfen) PC um drei erhöhen.

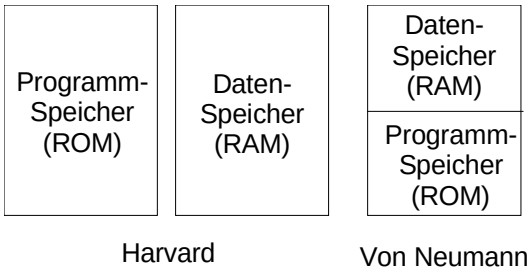
Ausführungszeit: 2 Zyklen

Der Befehlszeiger (PC) wird, abhängig von der Länge des gerade bearbeiteten Befehls, während der Befehlsausführung auf den OP-Code des Folgebefehls gesetzt. So kann nach Beendigung eines Befehls sofort mit der Bearbeitung des Nächsten begonnen werden. Bei Sprungbefehlen wird der Befehlszeiger während der Befehlsabarbeitung auf die Zieladresse gesetzt, wodurch als nächster Befehl der des Sprungziels bearbeitet wird. Ein Programm, welches immer aus mehreren Befehlen besteht, wird demnach Befehl für Befehl der Reihe nach mit maximaler Geschwindigkeit abgearbeitet.

4.2 Speicher

Der Speicher der MCS-51 Mikrocontroller ist nach Harvard-Architektur aufgebaut. Dies bedeutet, daß es eine strenge Trennung zwischen Programm- und Datenspeicher gibt. Der Programmspeicher besteht aus ROM (Read Only Memory) und kann nur gelesen, der Datenspeicher besteht aus RAM (Random Access Memory) und kann gelesen und beschrieben werden. Im Programmspeicher befinden sich die Befehle und die feststehenden Daten (Texte, Tabellen, ...) eines Programms. Im Datenspeicher befinden sich die veränderlichen Daten (Variablen, Messwerte, ...) eines Programms.

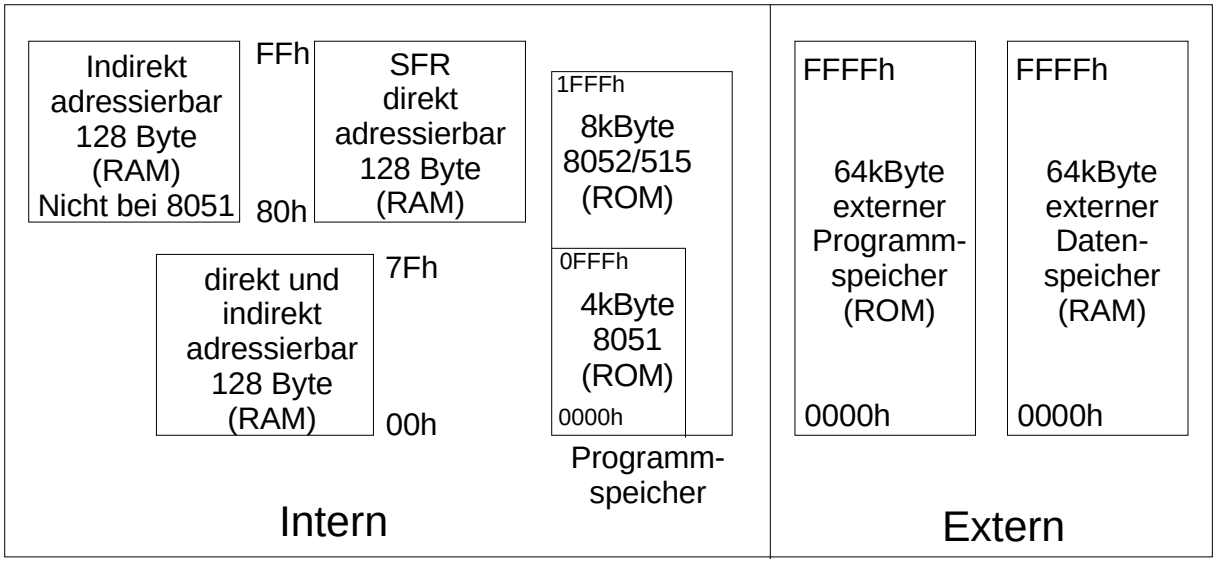
Im Gegensatz dazu wird bei den Mikrocomputern die Von Neumann-Architektur, welche einen gemischten Programm/Datenspeicher aufweist, benutzt. Durch den gemischten Speicher können in den Speicher Programme geladen und dort ausgeführt werden. Bei Harvard-Architektur ist es nicht möglich Programme in den Speicher zu laden.
Dafür verdoppelt sich jedoch die Menge an adressierbaren Speicher bei einer festgelegten Anzahl von Adressleitungen. Stehen wie bei MCS-51 sechzehn Adressleitungen bereit, so können 64kByte Daten- und 64kByte Programmspeicher (insgesamt 128kByte) angesprochen werden. Bei Von Neumann sind mit sechzehn Adressleitungen nur 64kByte Speicher verfügbar.



Die MCS-51 Mikrocontroller unterscheiden mehrere verschiedene Arten von Speicher. Sie kennen den internen und den externen Speicher. Der interne Speicher ist in den Mikrocontroller integriert. Der externe Speicher befindet sich in Bausteinen ausserhalb des Controllers. Im internen Speicher befindet sich der interne Programmspeicher, der interne Datenspeicher und die Sonder Funktions Register (SFR). Der externe Speicher kann entweder aus Programmspeicher oder aus Datenspeicher oder beiden bestehen. Der interne Speicher ist immer vorhanden, der externe ist nur bei Geräten zu finden die diesen benötigen.

Der interne Datenspeicher ist unterteilt in den direkt/indirekt adressierbaren Speicher (128Byte, Adr. 00-7Fh) und den nur indirekt adressierbaren Speicher (128Byte, Adr. 80-FFh). Der 8051/31 besitzt den nur indirekt adressierbaren Speicher nicht.

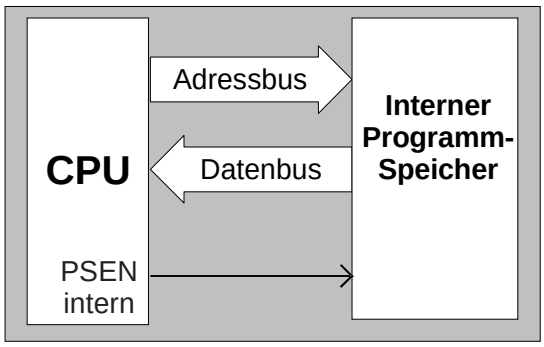
Der Bereich der SFR besteht aus beschreibbaren Speicherzellen (RAM), darf aber nicht als Datenspeicher genutzt werden. Die SFR sind Steuerregister zur Programmierung der Controllerinternen Hardware. Je nach verwendeten Mikrocontroller steht eine bestimmte Menge interner Programmspeicher zur Verfügung. Beim 8051 sind es 4kByte (Adr. 0000-0FFFh), beim 8052 und 80515 sind es 8kByte (Adr. 0000-1FFFh). Bei den romlosen Typen 8031/8032/80535 ist der interne Programmspeicher nicht nutzbar. Bei ihnen wird der Pin EA (External Access) aktiviert, wodurch sie ab Adresse 0000h mit externen Programmspeicher arbeiten. Der externe Speicher wird auch dort eingesetzt wo der interne für eine Anwendung nicht ausreicht. Durch den Einsatz von externen Speicher gehen, je nach Speichergröße und Art, eine bestimmte Anzahl Portbit, für den externen Adress/Datenbus und seine Steuersignale, als Ein/Ausgabeleitungen verloren.



4.2.1 Interner Programmspeicher

Der interne Programmspeicher befindet sich innerhalb des Mikrocontrollerbausteins. Je nach verwendeten Mikrocontroller hat er eine andere Größe. Der 8051 verfügt über 4kByte, der 8052 und 80515 über jeweils 8kByte. Er besteht immer aus Festwertspeicher (ROM), da während des laufenden Betriebes aus ihm nur gelesen werden muß. Bei den Controllertypen 8051, 8052 und 80515 besteht der Programmspeicher aus maskenprogrammiertem ROM. Das heißt, das Programm wird bereits während der Fertigung des Bausteins in diesem abgelegt. Die Typen 8031, 8032 und 80535 besitzen einen Programmspeicher mit undefiniertem Inhalt. Sie arbeiten immer mit externen Programmspeicher. Die Typen 8751 und 8752 besitzen als Programmspeicher ein EPROM. Dieses kann in einem speziellen Programmiergerät beschrieben und in einem speziellen Löschgerät wieder gelöscht werden. Nach der Programmierung des EPROM verhält sich ein 8751/52 wie ein 8051/52. Moderne Bausteine wie der 80C1051 verfügen als Programmspeicher über ein FLASH-EPROM, welches programmier- und wieder löschbar ist. Dazu kann der Baustein meist in der Schaltung verbleiben.

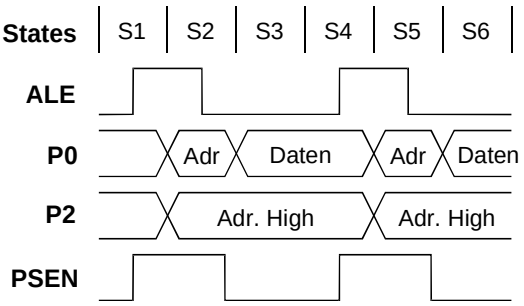
Der interne Programmspeicher ist über das interne Bussystem mit der CPU des Mikrocontrollers verbunden. Über den Adressbus wählt die CPU die Speicherzelle die gelesen werden soll an. Durch das interne PSEN-Signal (Program Store Enable, Programmspeicher Freigabe) wird der Speicher freigeschalten. Nach der Zugriffszeit des Speichers gelangt der Inhalt der Speicherzelle auf den internen Datenbus und kann in die CPU übernommen werden. Der Datenbus kennt bei Zugriffen auf den Programmspeicher nur eine Richtung, da der Programmspeicher nur gelesen werden kann. Wird auf eine Adresse oberhalb des internen Bereichs zugegriffen, wird automatisch das externe Bussystem des Controllers bedient. Dadurch werden die Ein/Ausgabebitmuster des Port 0 und Port 2 verändert. Durch dieses Verhalten kann es zu unvorhersehbaren Ein/Ausgaben auf diesen Ports kommen. Bei Geräten ohne externen Erweiterungsspeicher muß darauf geachtet werden, daß auf keine Adressen oberhalb des internen Programmspeicherbereichs zugegriffen wird.

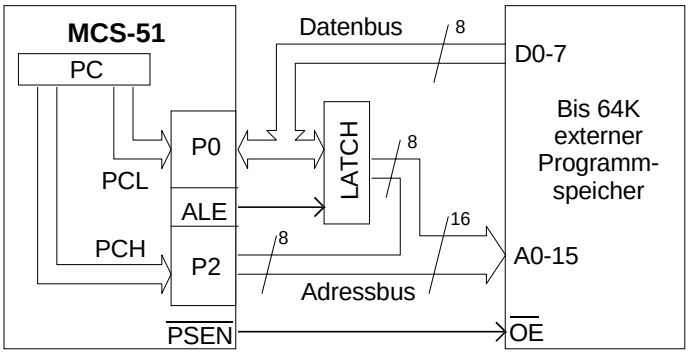


4.2.2 Externer Programmspeicher

Der externe Programmspeicher befindet sich in Bausteinen ausserhalb des Mikrocontrollers. Es wird ein externes Bussystem erforderlich, welches durch Port 0 und Port 2 des Mikrocontrollers gebildet wird. Diese beiden Ports gehen dann als Ein/Ausgabeleitungen verloren. Port 2 bildet die oberen 8Adressleitungen (Adr. 8-15) des 16Bit breiten Adressbus. Die Leitungen des Port 2 heißen jetzt A 8-15 (A = Adressleitung). Port 0 bildet die unteren 8 Adressleitungen (Adr. 0-7) und, gemultiplext, den 8Bit breiten Datenbus. Die Leitungen des Port 0 heißen jetzt AD 0-7 (AD = Adress/Datenleitung). Steht am Port 0 die Adressinformation an, erscheint am Mikrocontroller das Signal ALE (Adress Latch Enable, Adress Zwischenspeicher Freigabe). Durch die fallende Flanke von ALE kann die Adressinformation in ein Latch (D-Flip-Flop mit Pegelsteuerung) übernommen werden, wodurch der Port 0 anschließend als reiner Datenbus frei ist.

Das Signal PSEN (Program Store Enable, Programmspeicher Freigabe) des Mikrocontrollers wird zur Freigabe des Programmspeichers benutzt. Durch den sechzehn Bit breiten externen Adressbus ist es möglich, bis zu 64kByte externen Programmspeicher anzuschließen. Bei Verwendung von externen Programmspeicher bleiben nur noch Port 1 und Port 3 als Ein/Ausgabeports.





Merkmale bei externen Programmspeicher

Port 0 und Port 2 bilden das externe Bussystem

Port 1 und Port 3 stehen als Ein/Ausgabe zur Verfügung (16Bit)

Externe Speicher- und Latchbausteine erforderlich

Speicherausbau bis 64kByte möglich

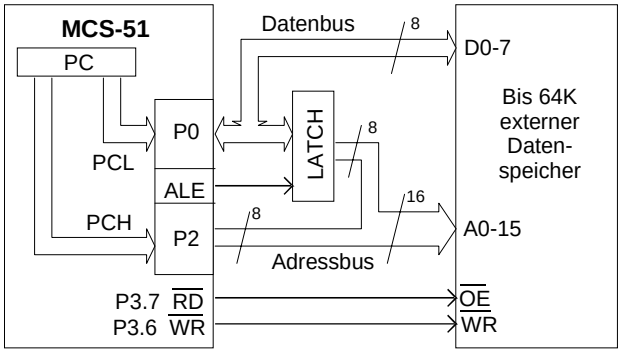
Befehle für den Programmspeicher:

Der Zugriff auf die Befehle eines Programms erfolg beim Fetch (Befehlsholzyklus) automatisch. Zum Lesen der Daten im Programmspeicher wird für den Datenwert immer der Akku, als Adresszeiger der Akku plus Datenpointer (DPTR) oder der Akku plus Befehlszeiger (PC) verwendet. Es werden immer 16Bit breite Adressen gebildet. Dabei steht eine Basisadresse im DPTR oder PC und ein positiver 8Bit breiter Adressoffset (Abstand zur Basisadresse) im Akku. Bei dieser Art der Adressierung wird der Zugriff auf Tabellen einfach, da die Basisadresse nur einmal geladen werden muß. Der Zugriff auf die Tabellenelemente erfolgt über den Offset.

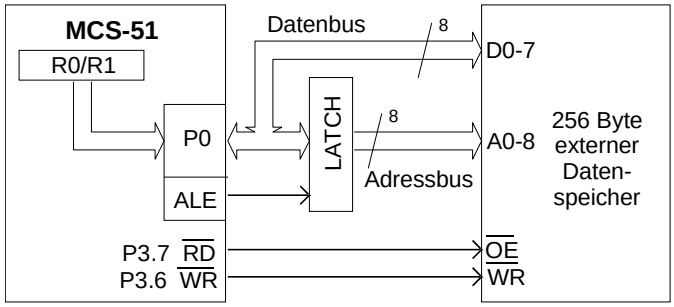
Befehle: MOVC A,@A+DPTR MOVC A,@A+PC

4.2.3 Externer Datenspeicher

Der externe Datenspeicher befindet sich, wie der externe Programmspeicher, in Bausteinen ausserhalb des Mikrocontrollers. Es wird das externe Bussystem mit Port 0 und Port 2 als Adress- und Datenbus benutzt. Zudem werden die zwei Bit P3.7 (RD Read, Lesen) und P3.6 (WR Write, Schreiben) als Steuerbus benötigt. READ ermöglicht Lesezugriffe aus dem, WRITE ermöglicht Schreibzugriffe in den externen Datenspeicher. Der externe Datenspeicher kann eine Größe bis 64kByte besitzen.



Für Anwendungsfälle in denen bis zu 256Byte externer Datenspeicher ausreichen, kann mit einer Adressbusbreite von 8Bit ein verkleinertes Bussystem zum Einsatz kommen. Dieses benützt nur den Port P0 als 8Bit Adress/Datenleitungen und die Signale READ und WRITE des Port P3. Port P2 arbeitet jetzt nicht als Adressbus und kann als Ein/Ausgabeport verwendet werden. Im Befehlssatz sind eigene Befehle (MOVX A,@Ri und MOVX @Ri,A) für das verkleinerte Bussystem vorgesehen. Als Zeiger (@Ri) auf die 8Bit breite Speicheradresse kann Register R0 oder R1 verwendet werden. Der Zugriff kann nur auf die erste Speicherseite (Page 0) mit den Adressen 0000h bis 00FFh erfolgen.

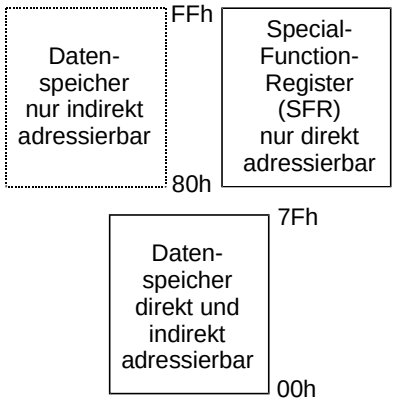


Befehle für den externen Datenspeicher:

Die Mnemonik für die Befehle lautet MOVX (eXternal datamemory). Bei Zugriffen auf den externen Datenspeicher wird für den Datenwert immer der Akku benutzt. Als Adresszeiger wird bei 16Bit breiten Adressen der Datenpointer (DPTR), bei 8Bit breiten Adressen Register R0 oder R1, verwendet

16Bit Adresse (0000h-FFFFh)	MOVX A,@DPTR	(Speicher lesen)
	MOVX @DPTR,A	(Speicher schreiben)
8Bit Adresse (0000h-00FFh)	MOVX A,@Ri	(Speicher lesen)
	MOVX @Ri,A	(Speicher schreiben)

4.2.4 Interner Datenspeicher



Je nach Typ des Mikrocontrollers besitzt der interne Datenspeicher eine andere Größe. Der 8051/31 enthält 128 Byte, der 8052/32 und 80515/535 enthalten 256 Byte. Die einzelnen Speicherzellen besitzen eine Breite von jeweils acht Bit. Die ersten 128 Byte (Adr. 00h-7Fh) sind bei allen Typen vorhanden. Dieser Speicherbereich ist direkt und indirekt adressierbar. Die oberen 128 Byte (Adr. 80h-FFh) liegen adresstechnisch parallel zu den Sonder-Funktions-Registern (SFR). In diesem Adressbereich wird durch die Adressierungsart festgelegt, ob mit einer Speicherzelle oder einem SFR gearbeitet wird. Auf die SFR kann nur mit direkter Adressierung (MNEMO ZIEL,dadr und MNEMO dadr,QUELLE), auf die Speicherzellen nur mit indirekter Adressierung (MNEMO ZIEL,@Ri und MNEMO @Ri,QUELLE) zugegriffen werden.

Direkt/indirekt adressierbarer Speicher

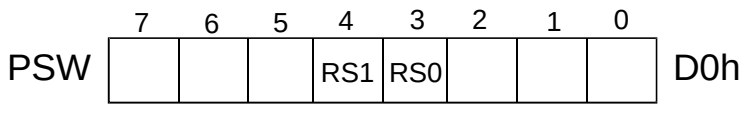
Dieser Speicherbereich ist 128 Byte groß (Adr. 00h-7Fh), hat eine Breite von 8Bit und besteht aus RAM. In ihm befinden sich die Register für die CPU (Adr. 00h-1Fh), ein Bereich von 16 Byte in dem jedes einzelne Bit eine eigene Bitadresse besitzt (Adr. 20h-2Fh) und ein funktionell nicht vorbelegter Bereich (Adr. 30h-7Fh) der als Stack und allgemeiner Datenspeicher genutzt wird.

Register

Register-Bank 3	Register 7	1Fh
	Register 6	1Eh
	Register 5	1Dh
	Register 4	1Ch
	Register 3	1Bh
	Register 2	1Ah
	Register 1	19h
	Register 0	18h
Register-Bank 2	Register 7	17h
	Register 6	16h
	Register 5	15h
	Register 4	14h
	Register 3	13h
	Register 2	12h
	Register 1	11h
	Register 0	10h
Register-Bank 1	Register 7	0Fh
	Register 6	0Eh
	Register 5	0Dh
	Register 4	0Ch
	Register 3	0Bh
	Register 2	0Ah
	Register 1	09h
	Register 0	08h
Register-Bank 0	Register 7	07h
	Register 6	06h
	Register 5	05h
	Register 4	04h
	Register 3	03h
	Register 2	02h
	Register 1	01h
	Register 0	00h

Im untersten Bereich des internen Datenspeichers befinden sich die Register des Mikrocontrollers. Sie sind unterteilt in vier Registerbänke mit jeweils acht Registern. Die Register einer Bank sind, beginnend mit 0, der Reihe nach durchnummeriert. Es gibt in jeder Bank ein Register R0, R1, R2, R3, R4, R5, R6 und R7. Zu einem Zeitpunkt kann immer nur eine der Registerbänke aktiv sein. Das Umschalten der Registerbank geschieht im Programm-Status-Wort (PSW) mit den beiden Bit RB0 und RB1 (Registerbank Select 0/1). Nach Reset ist die Registerbank 0 aktiv

RS1	RS0	Aktive Bank
0	0	Registerbank 0
0	1	Registerbank 1
1	0	Registerbank 2
1	1	Registerbank 3



Die Register der aktiven Bank werden mit den Registerbefehlen (MNEMO Rr,QUELLE und MNEMO ZIEL,Rr) angesprochen. Jedes Register, auch die der nicht aktiven Bänke, kann als direkt oder indirekt adressierbare Speicherzelle angesprochen werden. Nicht genutzte Registerbänke können als allgemeiner Datenspeicher verwendet werden..

Bitadressierbarer Bereich

Im Speicherbereich 20h bis 2Fh befinden sich 16 Speicherzellen innerhalb derer jedes Bit, mit einer eigenen Adresse, einzeln ansprechbar ist. In diesem Bereich kann direkt/indirekt adressiert auf die Speicherbyte oder mit den Bitbefehlen (SETB badr, CLR badr, ...) der CPU auf die einzelnen Bit zugegriffen werden.

Beispiele:

MOV 21h,#5Ch Byte 5Ch in Adr. 21h schreiben

SETB 1Dh Bit 5 in Adr. 23h setzen

Die Bitadresse läßt sich der Tabelle entnehmen oder berechnen.

Rechnung: Byteadresse (hex) - 20h * 8 + Bitoffset

Beispiel: Bit 5 in Speicherzelle 23h

23h - 20h = 03h 03h * 8 = 18h 18h + 5 = 1Dh

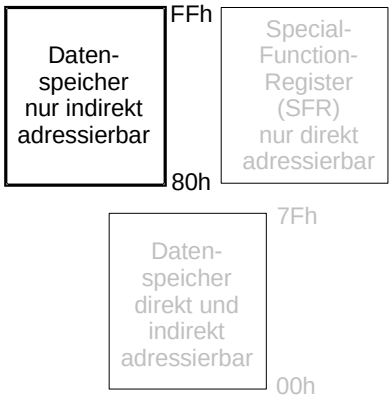
7F	7E	7D	7C	7B	7A	79	78	2Fh
77	75	75	74	73	72	71	70	2Eh
6F	6E	6D	6C	6B	6A	69	68	2Dh
67	66	65	64	63	62	61	60	2Ch
5F	5E	5D	5C	5B	5A	59	58	2Bh
57	56	55	54	53	52	51	50	2Ah
4F	4E	4D	4C	4B	4A	49	48	29h
47	46	45	44	43	42	41	40	28h
3F	3E	3D	3C	3B	3A	39	38	27h
37	36	35	34	33	32	31	30	26h
2F	2E	2D	2C	2B	2A	29	28	25h
27	26	25	24	23	22	21	20	24h
1F	1E	1D	1C	1B	1A	19	18	23h
17	16	15	14	13	12	11	10	22h
0F	0E	0D	0C	0B	0A	09	08	21h
07	06	05	04	03	02	01	00	20h

Datenspeicher und Stack

Der Speicherbereich von 30h bis 7Fh ist frei als allgemeiner Datenspeicher zu verwenden. In ihm kann auch der Stack für die CPU eingerichtet werden. Zu beachten ist, daß der Stackpointer (SP) nach Reset auf die Speicherzelle 07h (R0 der RB1) zeigt. Soll sich der Stack weiter oben befinden, ist der Stackpointer mit der entsprechenden Adresse zu laden.

Indirekt adressierbarer Datenspeicher

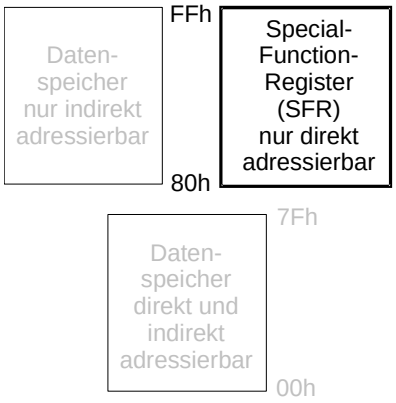
Über den internen nur indirekt adressierbaren Datenspeicher verfügen die Mikrocontroller 8052/32 und 80515/535. Der 8051/31 besitzt diesen Speicher nicht. Es handelt sich dabei um 128 Byte RAM mit einer Breite von 8Bit. Der Adressbereich deckt sich mit dem der SFR. Während auf die SFR nur direkt adressiert zugegriffen wird, erfolgt der Zugriff hier nur mit indirekter Adressierung. Bei Zugriffen im Adressbereich 80h bis FFh im internen Datenspeicher entscheidet also die Adressierungsart, ob ein SFR oder eine Speicherzelle angesprochen wird. Als Zeigerregister auf eine der Speicherzellen kann das Register R0 oder R1 der aktiven Registerbank zum Einsatz kommen. Die Register R2 bis R7 können nicht als Zeiger auf Speicherzellen arbeiten.



Befehle: MNEMO ZIEL,@Ri (Speicherzelle lesen)

MNEMO @Ri,QUELLE (Speicherzelle schreiben)

4.2.5 Special Function Register (SFR)



Bei den Special-Function-Registern (Sonder-Funktions-Register) handelt es sich um 8Bit Breite RAM-Zellen im Adressbereich 80h bis FFh im internen Speicher. Nicht alle Adressen in diesem Bereich sind belegt, die Anzahl der belegten SFR ist vom Typ des Mikrocontrollers abhängig. Nicht belegte SFR können nicht als Speicherzellen genutzt werden, da sie als RAM-Zellen nicht vorhanden sind. Die SFR dienen der Steuerung und Programmierung der Mikrocontroller-Hardware und der Ablaufsteuerung der CPU. Der Zugriff zu den SFR ist nur durch direkte Adressierung möglich.

Befehle: MNEMO ZIEL,dadr (SFR lesen)

MNEMO dadr,QUELLE (SFR schreiben)

Jedes einzelne SFR besitzt neben einer Adresse auch einen eindeutigen Namen. In SFR deren hexadezimale Byteadresse in der Low-Tetrade eine 0 oder eine 8 trägt (X0h oder X8h) besitzen die einzelnen Bit eine eigene Bitadresse. Die meisten dieser Einzelbit tragen zudem eigene Namen. Die Bitadressen ergeben sich aus den Byteadressen plus dem Bitoffset.

Rechnung: Byteadresse + Bitoffset = Bitadresse

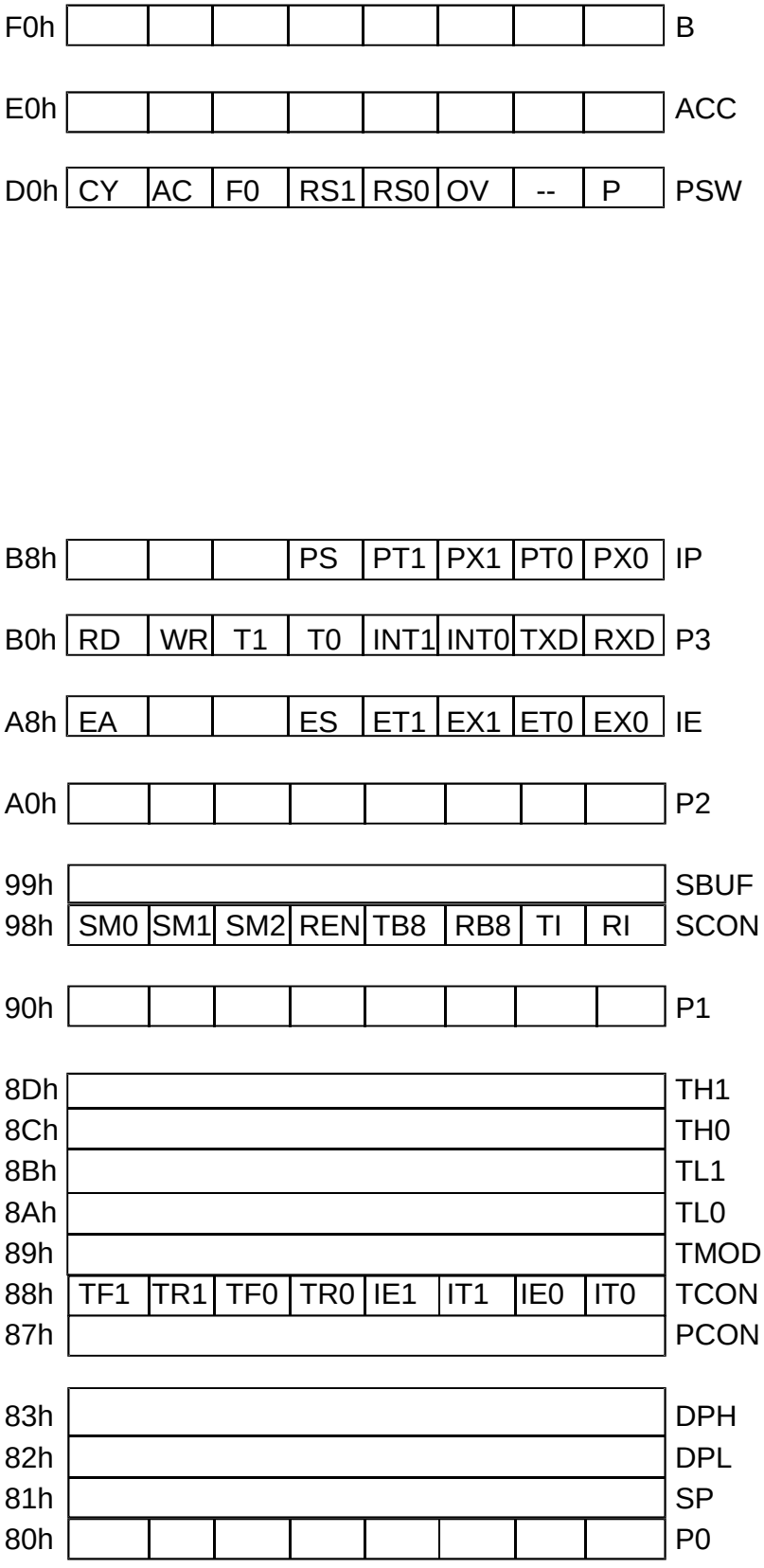
Beispiel: Bit RS0 = Bit 3 in SFR - PSW (D0h)

Byteadresse = D0h Bitoffset = 3

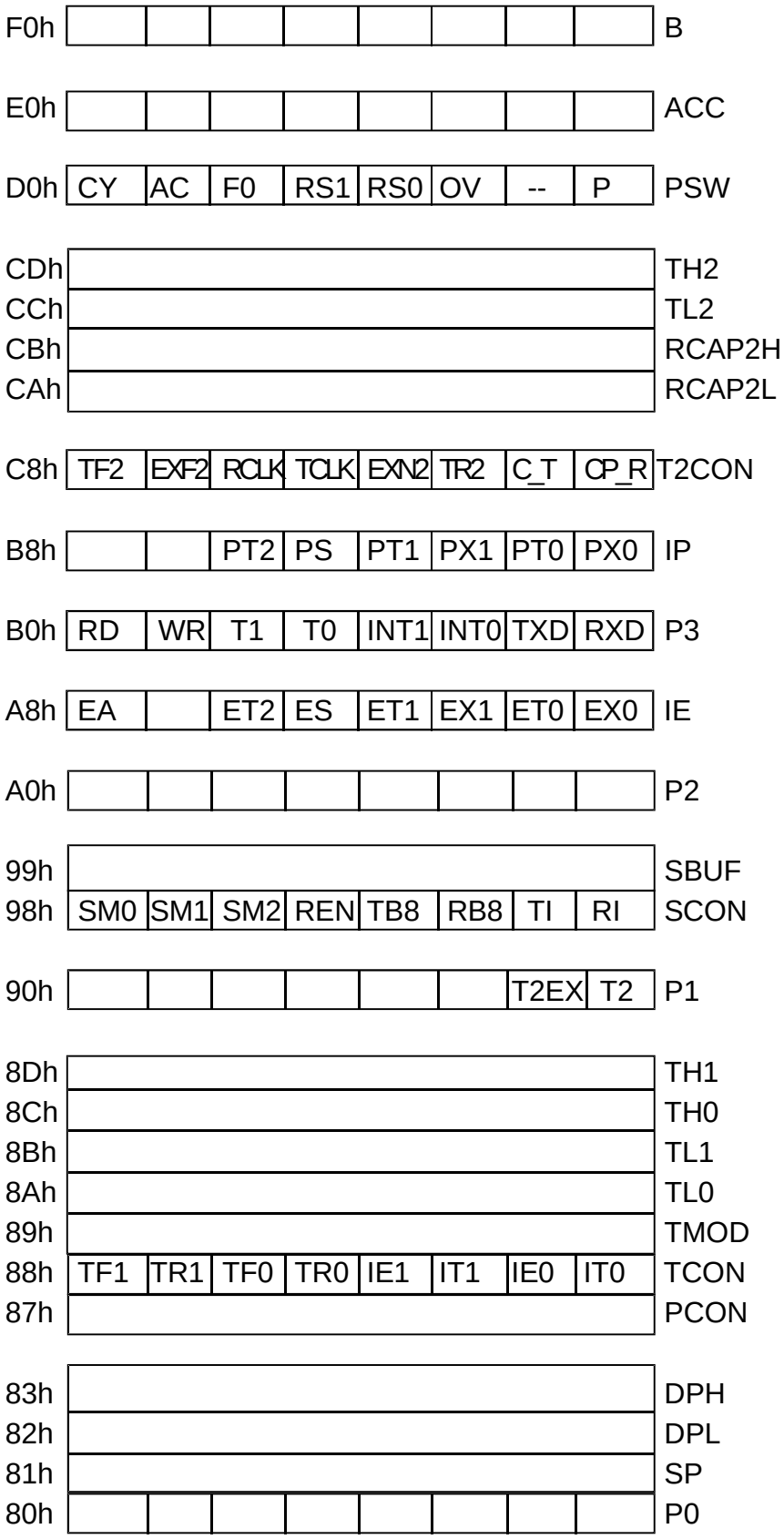
Bitadresse = D0h + 3 = D3h

Die SFR werden mit den Bytebefehlen in direkter Adressierung angesprochen. Die bitadressierbaren Einzelbit innerhalb der SFR werden mit den Bitbefehlen der CPU angesprochen.Die einzelnen Bit aller SFR können mit Hilfe der logischen Verknüpfungen geändert oder getestet werden.

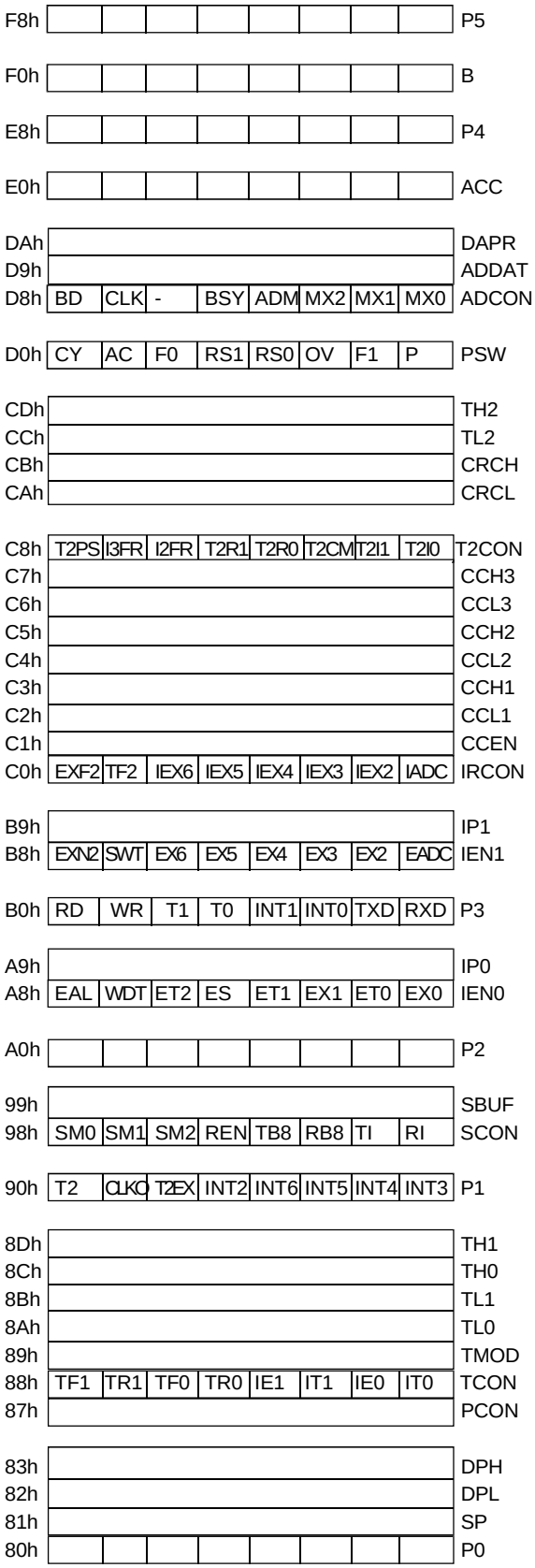
Die SFR des 8051



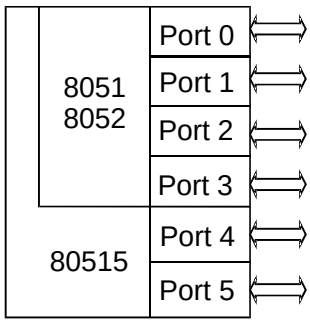
Die SFR des 8052



Die SFR des 80515

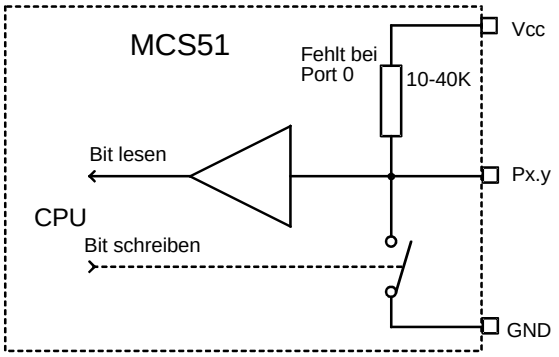


4.3 Ein/Ausgabeleitungen (Ports)



Da Mikrocontroller für Steuerungsaufgaben ausgelegt sind, kommt ihrer Ein/Ausgabefähigkeit eine besondere Bedeutung zu. Der Befehlssatz unterstützt eine einfache Handhabung der Ports. Die Mikrocontroller 8051/52 und ähnliche besitzen vier bidirektionale (Zweirichtungs) 8Bit breite Ports. Der 80515/535 besitzt sechs dieser Ports. Jedes einzelne Bit dieser Ports kann als eigenständiges Ein- oder Ausgabebit Anwendung finden. Einige der Ports besitzen neben der Ein/Ausgabefunktion noch Sonderfunktionen. Der Port P0 dient als Adress/Datenbus, der Port P2 als Adressbus und der Port P3 als Steuerbus für Speicher, Timer und Interrupt. Beim 80515/535 wird zusätzlich der Port P1 für Timer- und Interrupteingänge benutzt.

Der prinzipielle Aufbau ist bei allen Ports ähnlich, die Unterschiede im inneren Aufbau sind für das Verständnis ihrer Arbeitsweise unerheblich. Jedes Portbit besitzt einen Leseverstärker und einen Schalter (Transistorschalter), die am Portpin zusammengeschaltet und mit einem Pull-Up Widerstand gegen die positive Versorgungsspannung abgeschlossen sind. Beim Port P0 fehlt, wegen seiner Funktion als Datenbus, der Pull-Up Widerstand. Wird Port P0 als Ein/Ausgabeport benutzt, muß an jedem seiner Bit ein externer Arbeitswiderstand angeschaltet sein.



Ausgabe:

Gibt ein Befehl eine „0“ aus, so wird durch die integrierte Portlogik der Schalter geschlossen. Der Portpin ist über den Transistorschalter mit GND verbunden. Am Portpin erscheint der Pegel „0“.

Gibt ein Befehl eine „1“ aus, so wird durch die integrierte Portlogik der Transistorschalter geöffnet. Der Portpin ist über den Pull-Up Widerstand mit der positiven Versorgungsspannung verbunden. Am Portpin erscheint der Pegel „1“. Die „1“ wird über den Widerstand hochohmig erzeugt, weshalb diese nur wenig belastbar ist. Beim Port P0 muß der Arbeitswiderstand extern angeschaltet sein, da sonst keine logische „1“ ausgegeben werden kann.

Eingabe:

Bevor eine Eingabe über ein Portbit durchgeführt werden kann, ist sicherzustellen daß der Transistorschalter geöffnet ist (Ausgabe einer „1“), da der Portpin sonst gegen GND kurzgeschlossen ist.

Nach dem Einschalten der Versorgungsspannung oder bei Reset wird bei allen Ports FFh (1111 1111b) ausgegeben, wodurch alle Transistorschalter geöffnet und damit Eingaben möglich sind.

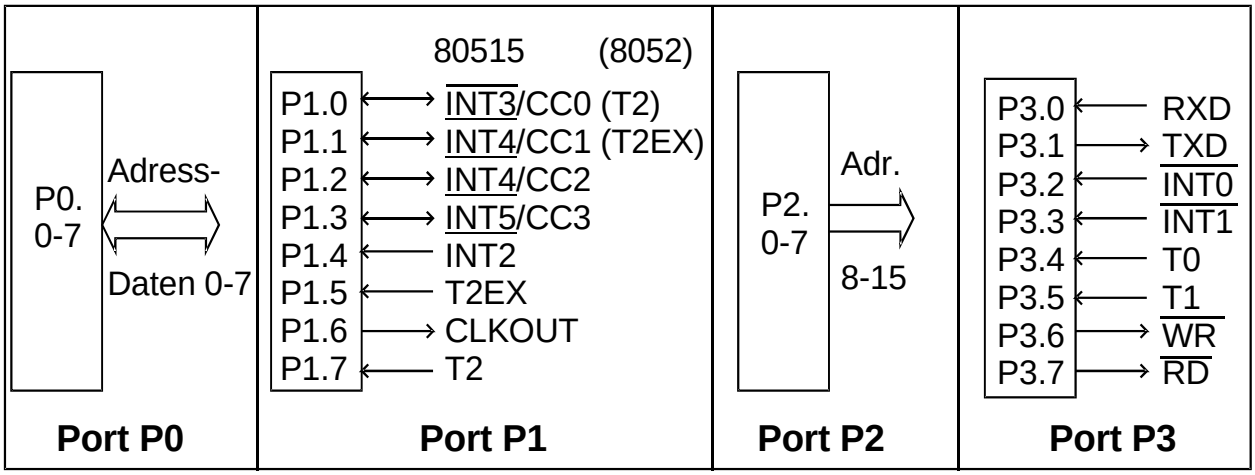
Die Hardware der Ports ist direkt mit den zugeordneten SFR - P0 bis P3/P5 verbunden. Ein/Ausgaben werden durch Befehle mit den SFR durchgeführt. Ein Lesen der SFR liefert den Zustand der Ein/Ausgabeleitungen. Ein Schreiben in das SFR verändert den Zustand der Ein/Ausgabeleitungen. Die Adressen der Port-SFR sind so gewählt, daß die Einzelbit jedes Port bitadressierbar sind.

Port P0 = 80h	Port P1 = 90h	
Port P2 = A0h	Port P3 = B0h	
Port P4 = E8h*	Port P5 = F8h*	* Nur 80515/535

Der Zugriff zu den Ports kann mit den Bytebefehlen auf alle acht Bit gleichzeitig, oder mit den Bitbefehlen auf einzelne Bit erfolgen. Die Ports werden als direkt adressierbare Speicherzellen angesprochen (Memory Mapped I/O). Dadurch ist gegeben, daß alle Operationen die mit direkt adressierbaren Speicherzellen durchführbar sind auch mit den Ports möglich sind.

Sonderfunktionen der Ports

Der Port P0 dient bei allen MCS51-Typen als Adress/Datenbus. Der Port P2 als Adressbus und der Port P3 für Sondersignale. RXD (Receive Data, P3.0) ist der serielle Eingang, TXD (Transmit Data, P3.1) der serielle Ausgang des UART. INT0 und INT1 (Interrupt 0 - P3.2, Interrupt 1 - P3.3) sind die Eingänge für externe Interrupts. T0 und T1 (Timer 0 - P3.4, Timer 1 - P3.5) sind die Eingänge der Zeitgeber/Zähler. WR (WRite, P3.6) und RD (ReaD, P3.7) sind die Signale des Steuerbus bei externem Datenspeicher.



Die Sonderfunktionen des Port P1 unterscheiden sich bei den einzelnen Mikrocontrollertypen.

8051/31 keine Sonderfunktionen.

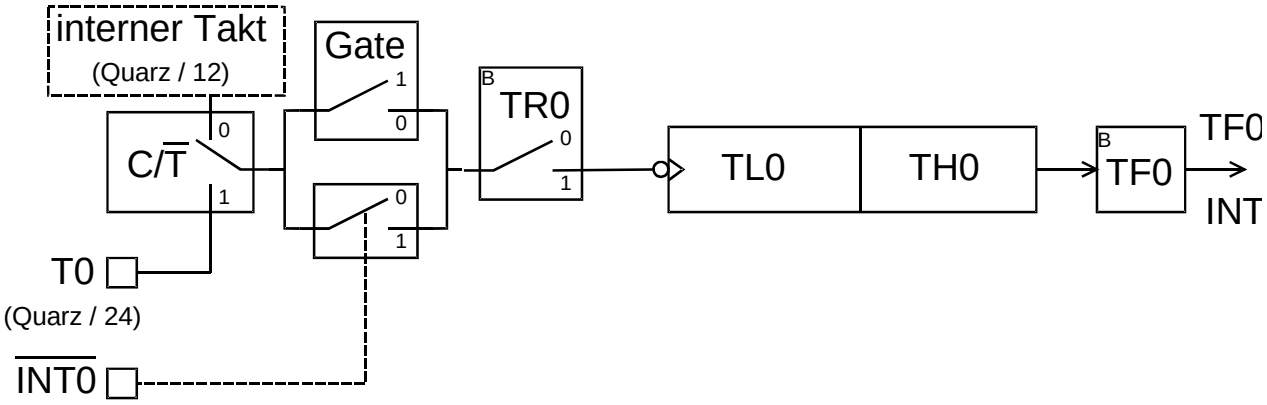
8052/32	P1.0 = T2	- Eingang für Timer 2
	P1.1 = T2EX	- Externer Interrupt 2
80515/535	P1.0 = INT3/CC0	- Externer Interrupt 3/Compare Capture 0
	P1.1 = INT4/CC1	- Externer Interrupt 4/Compare Capture 1
	P1.2 = INT5/CC2	- Externer Interrupt 5/Compare Capture 2
	P1.3 = INT6/CC3	- Externer Interrupt 6/Compare Capture 3
	P1.4 = INT2	- Externer Interrupt 2
	P1.5 = T2EX	- Steuereingang Interrupt 2
	P1.6 = CLKOUT	- Taktausgang
	P1.7 = T2	- Eingang für Timer 2

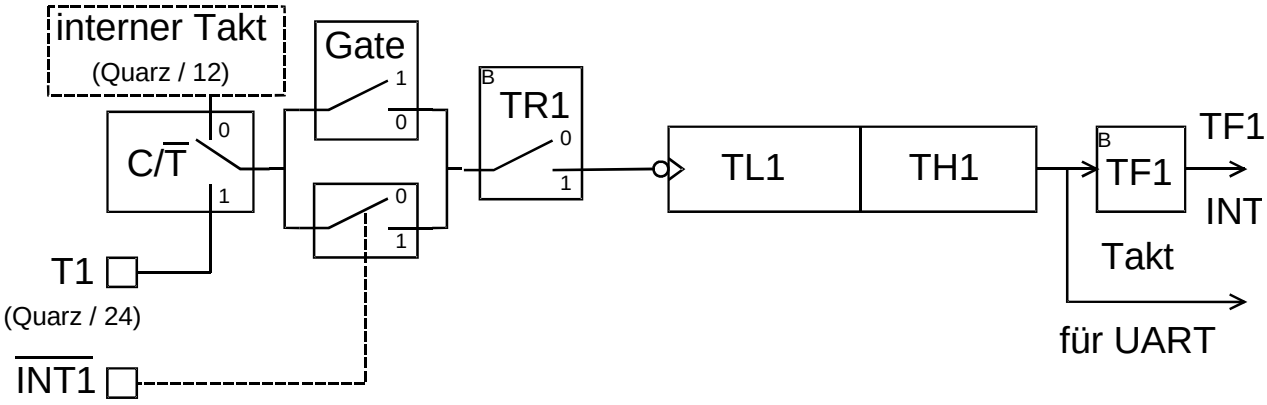
4.4 Zeitgeber/Zähler

Prinzipiell können Zeitbedingungen und Zählaufgaben allein mit Programmen durch die CPU bearbeitet werden, dabei wird diese aber so stark beschäftigt daß sie andere Aufgaben nur mehr eingeschränkt bewältigen kann. Aus diesem Grund verfügen die Mikrocontroller über eine eigens dafür integrierte Hardware, die Zeitgeber/Zähler. Die Zeitgeber/Zähler kennen zwei grundlegende Arbeitsweisen. Als Zeitgeber (Timer) zum Erzeugen/Erfassen von Zeiten. Als Zähler (Counter) zum Erfassen der Anzahl externer Ereignisse (z.B: Zählen einer Stückzahl). Die Anzahl und die genauen Betriebsarten der Timer sind vom Typ des Mikrocontrollers abhängig. Der 8051/31 besitzt zwei Timer (Timer T0 und T1). Der 8052/32 und der 80515/535 besitzen drei Timer (Timer T0, T1 und T2). Der Timer T2 ist bei 8052 und 80515 nicht identisch aufgebaut und kennt jeweils andere Betriebsarten und Funktionen.

4.4.1 Timer T0 und T1

Der Aufbau und die Funktion der Timer T0 und T1 sind bei allen Mitgliedern der MCS-51 Familie identisch. Das Zählregister ist 16Bit breit und zusammengesetzt aus zwei 8Bit breiten SFR. THx (Timer High 0/1) ist das High-Register mit Bit 8 bis Bit 15, TLx (Timer Low 0/1) ist das Low-Register mit Bit 0 bis Bit 7. Da sich die Zählregister im Bereich der SFR befinden, können sie als direkt adressierbare Speicherzellen angesprochen werden. Die Zählregister können gelesen und beschrieben werden.





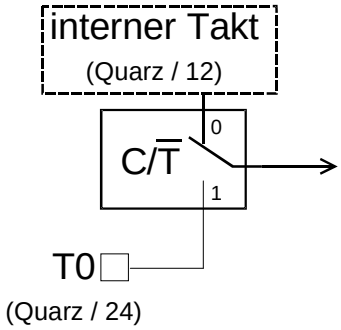
Mit jeder fallenden Flanke am Eingang des Zählregisters wird dessen Inhalt um eins erhöht. Hat das Zählregister seinen höchsten Zählerstand (z.B: FFFFh) erreicht und es wird noch eine Flanke gezählt, läuft es über. Dabei wird das Überlaufbit TFx (Timer Overflow 0/1) gesetzt und das Zählregister hat den Inhalt Null. Vom Überlauf des Timers T1 wird der Schiebetakt für die serielle Schnittstelle (UART) abgeleitet. Der Eingang des Zählregisters wird vom Bit TRx (Timer Run 0/1) gesteuert. Ist dieses Bit gesetzt („1“), ist der Signalweg geschlossen. Ist dieses Bit gelöscht („0“), ist der Signalweg unterbrochen. Durch das TRx-Bit ist der Timer Start- und Stopbar. Vor dem TRx-Bit ist der Signalweg zweigeteilt. Er ist einmal über das Gate und parallel dazu über einen vom INTx-Eingang gesteuerten Schalter geführt. Ist das Gate-Bit gelöscht („0“), ist der INTx-Schalter überbrückt. Dadurch spielt der Zustand des INTx-Eingangs keine Rolle auf die Arbeitsweise des Timers. Ist das Gate-Bit gesetzt („1“), ist der INTx-Schalter nicht überbrückt. Dadurch wird der Signalweg vom Zustand des INTx-Eingangs beeinflußt. Durch dieses Verhalten ist eine Torsteuerung der Timer durch den INTx-Eingang möglich. Bei Torsteuerung kann der Timer durch den Zustand am INTx-Eingang gestartet und gestopt werden. Ein Zählen des Zählregisters findet nur statt, solange der INTx-Eingang "1" ist.

Torsteuerung: Gate-Bit ist „1“

INTx-Eingang ist „1“ Zähler zählt

INTx-Eingang ist „0“ Zähler stopt

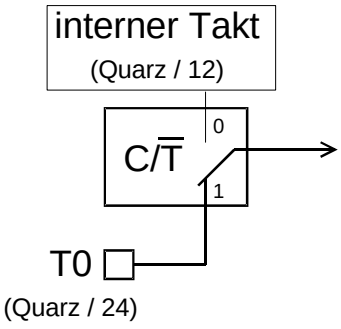
Je nach Arbeitsweise der Timer sind zwei unterschiedliche Signalquellen möglich. Arbeitet der Timer als Zeitgeber (Timer), ist die Signalquelle der interne Takt. Dieser entspricht der Quarzfrequenz geteilt durch 12 (Bei 12MHz Quarz 1MHz). Durch das Bit C/T = 0 (Counter/Timer) wird auf den internen Takt als Signalquelle umgeschaltet. Das Bit C/T befindet sich im SFR-TMOD. Es ist nicht Bitadressierbar.



Beispiel: C/T-Bit löschen

```
ANL TMOD,#0FBh    ;Für Timer T0
ANL TMOD,#0BFh    ;Für Timer T1
```

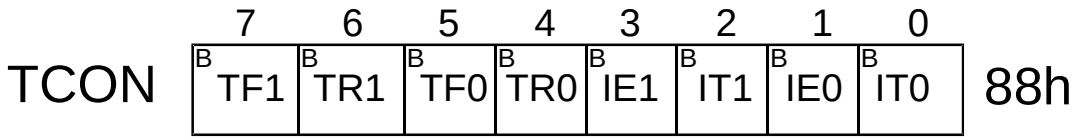
Arbeitet der Timer als Zähler (Counter), wird als Signalquelle der Tx-Eingang (T0/T1) verwendet. Die Taktfrequenz am Tx-Eingang darf die Quarzfrequenz geteilt durch 24 nicht überschreiten (Bei 12MHz Quarz 500kHz). Durch das Bit C/T = 1 wird auf den externen Takt als Signalquelle umgeschaltet. Das Bit C/T befindet sich im SFR-TMOD. Es ist nicht Bitadressierbar.



Beispiel: C/T-Bit setzen

```
ORL TMOD,#04h     ;Für Timer T0
ORL TMOD,#40h     ;Für Timer T1
```

Die Steuerung und Auswertung der Timer erfolgt in den SFR TMOD (Timer Mode) und TCON (Timer Control).

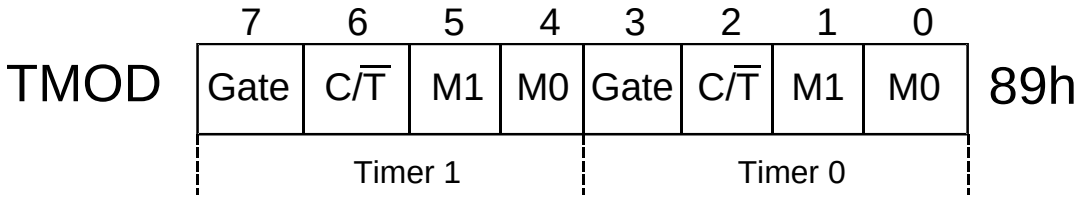


Tfx Timer overFlow 0/1 Überlaufbit für das Zählregister

Wird gesetzt wenn der Zähler seinen Überlauf erreicht. Dieses Bit kann durch Software abgefragt werden oder einen Interrupt auslösen. Es wird bei Annahme des Interrupt automatisch gelöscht, ohne Interrupt muß es durch Software gelöscht werden.

TRx Timer Run 0/1 Startet und stopt den Zeitgeber/Zähler

Das Bit wird durch Software gesetzt oder gelöscht.
"0" = Zeitgeber/Zähler angehalten
"1" = Zeitgeber/Zähler freigegeben



Gate Gate Control Torsteuerung des Zeitgeber/Zähler

Das Bit wird durch Software gesetzt oder gelöscht.
"0" = Torsteuerung durch INTx-Eingang gesperrt
"1" = Torsteuerung durch INTx-Eingang freigegeben

C/T Counter/Timer Zeitgeber/Zähler Umschaltung

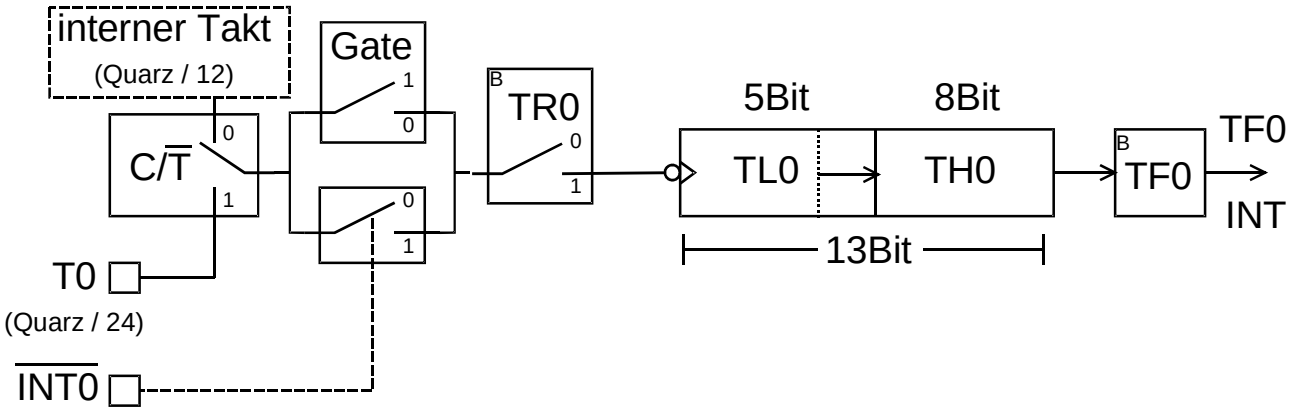
Das Bit wird durch Software gesetzt oder gelöscht.
"0" = Zeitgeber (Timer), interner Takt
"1" = Zähler (Counter), externer Takt

M1/M0 Mode 0/1 Wahl der Betriebsart

M1	M0	Betriebsart
0	0	13Bit Zählregister
0	1	16Bit Zählregister
1	0	8Bit Zählregister autoreload
1	1	Nur Timer 0 TL0 = 8Bit Zählregister TH0 = 8Bit Zählregister Timer 1 stopt in Betriebsart 3

Betriebsart 0

In Betriebsart 0 arbeiten Timer T0 und T1 als 13Bit Zeitgeber/Zähler. Das Low-Zählregister (TL0/1) wirkt als 5Bit Vorteiler für das High-Zählregister (TH0/1).Das Hochzählen des High-Registers erfolgt bereits beim Übertrag von Bit 4 nach Bit 5 des Low-Registers. Im Low-Register wird bis zum Zählerstand FFh weitergezählt. Dadurch sind die Zustände der Bit 5, 6 und 7 des Low-Register und die Bit 0, 1 und 2 des High-Register immer gleich.

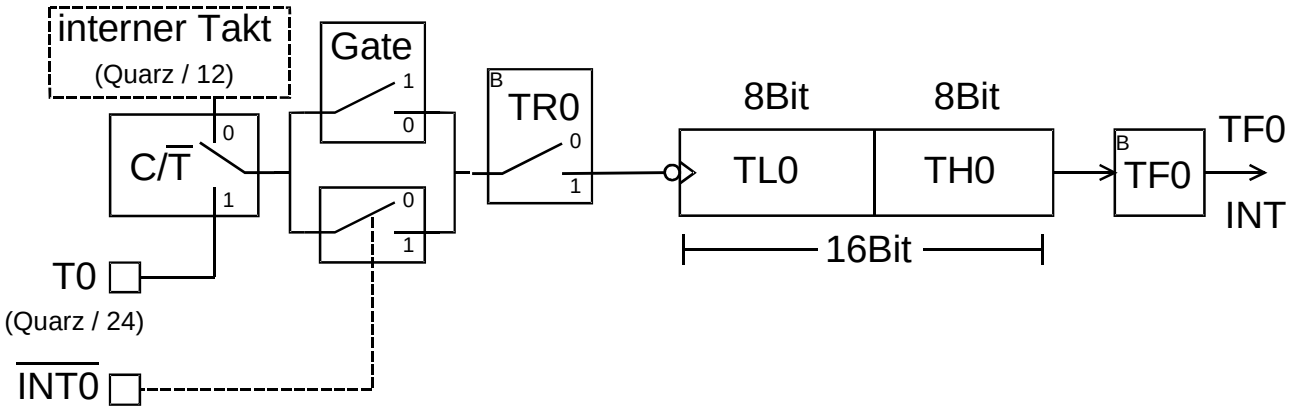


Zählbereich: 1 - 1FFFh = 1 - 8191d = 1 - 8191 Zyklen

Bei 12MHz Quarz: 1 - 8191 µs (8,191ms)

Betriebsart 1

In Betriebsart 1 wird für beide Timer die volle Breite von 16Bit des Zählregisters genutzt. Das Low-Register zählt bis FFh und erzeugt beim nächsten Zählschritt einen Übertrag, der das High-Register um eins erhöht.

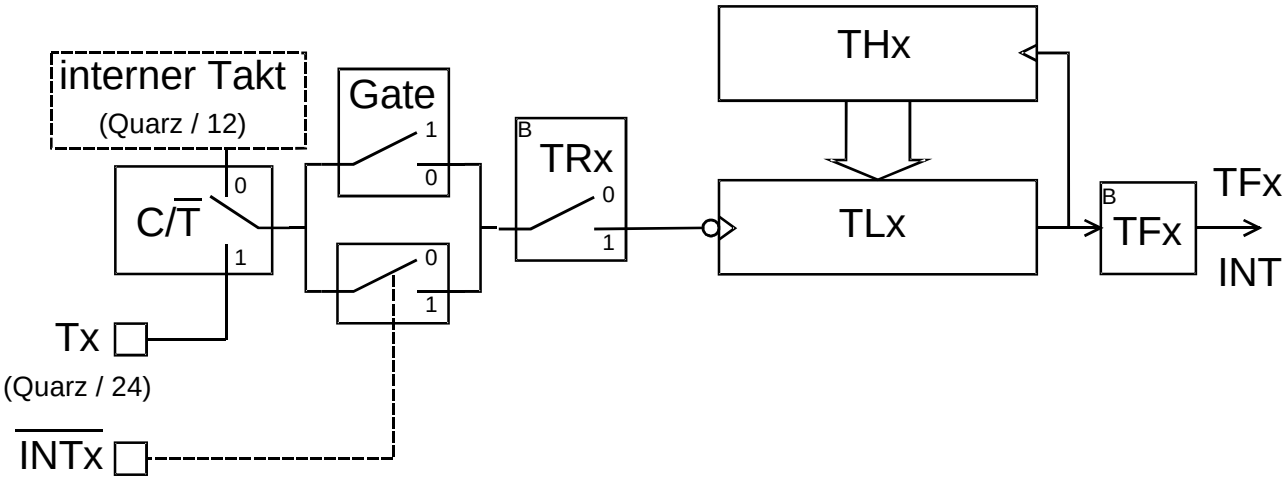


Zählbereich: 1 - FFFFh = 1 - 65535d = 1 - 65535 Zyklen

Bei 12MHz Quarz: 1 - 65535µs (65,535ms)

Betriebsart 2

In Betriebsart 2 arbeitet das Low-Zählregister als 8Bit Zeitgeber/Zähler. Erreicht das Zählregister seinen Überlauf (FFh+1), wird das Überlaufbit (TF0/1) gesetzt und zudem der Inhalt des High-Registers ins Low-Register übertragen. Der Zähler lädt sich automatisch bei Überlauf mit dem Ladewert aus dem High-Register nach (autoreload).

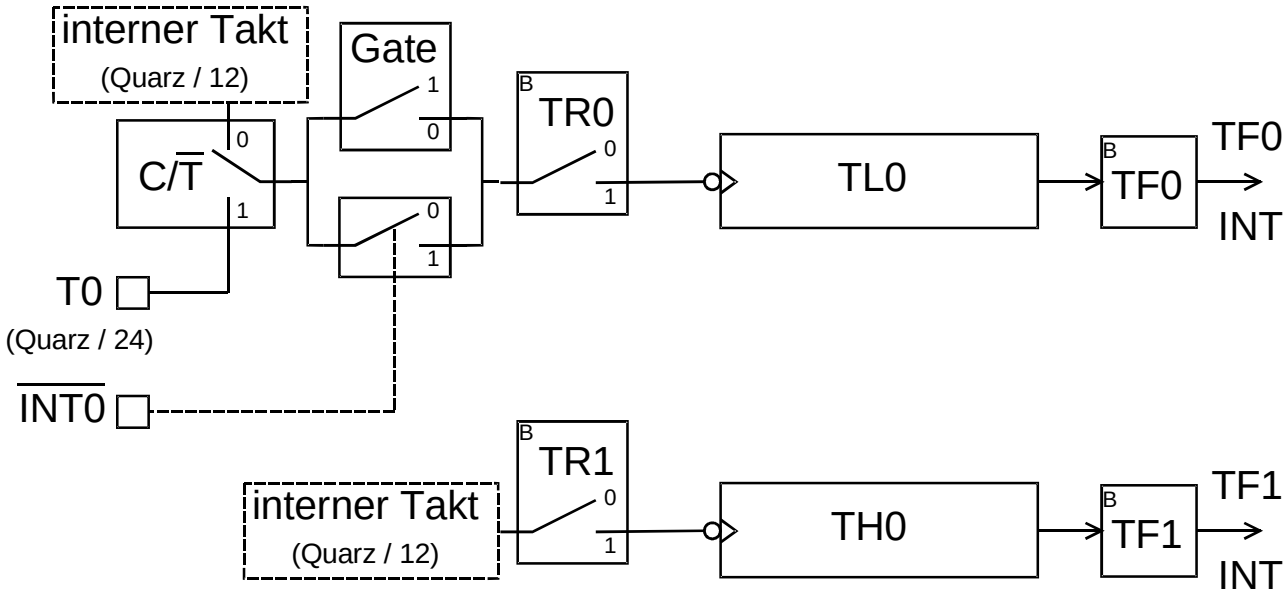


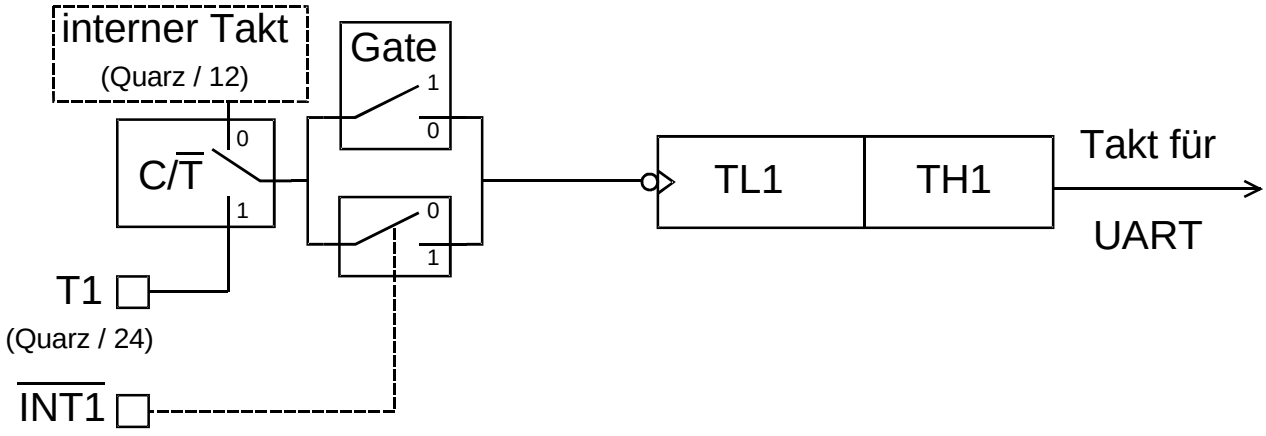
Zählbereich: 1 - FFh = 1 - 255d = 1 - 255 Zyklen

Bei 12MHz Quarz: 1 - 255µs

Betriebsart 3

Die Betriebsart 3 ist nur für Timer 0 zulässig, wird Timer 1 für diese Betriebsart programmiert, stopt er. In Betriebsart 3 arbeitet das Low-Register des Timer 0 (TL0) mit 8Bit Breite. Es werden die Steuerbit (C/T, T0, Gate, INT0, TR0 und TF0) des Timer 0 für den Zähler mit dem Low-Register benutzt. Das Timer 0 High-Register (TH0) arbeitet als 8Bit Zeigeber. Dieser wird nur mit dem internen Takt betrieben und benutzt zur Steuerung die Bit TR1 und TF1 des Timer 1. Der Timer 1 kann noch in Betriebsart 0, 1 oder 2 ohne TR1- und TF1-Bit arbeiten. Mit seiner Restfunktion kann er beispielsweise noch den Schiebetakt für die serielle Schnittstelle (UART) erzeugen.





Berechnen des Ladewerts

Wird ein Timer als Zeitgeber benutzt, ist in das Zählregister ein Zahlenwert zu laden, bei dem die Anzahl der Zählschritte bis zum Überlauf, der gewünschten Zeit entspricht. Dieser Wert wird Zeitwert genannt und ist die Anzahl der Zählschritte für die gewünschte Zeit. Er ist vom Zähltakt des Mikrocontrollers und der gewünschten Zeit abhängig. Paßt der Zeitwert in ein Register, kann er in Assemblerprogrammen als negative Dezimalzahl direkt in das Register geladen werden. Ist dies nicht möglich, muß ein Ladewert berechnet und auf die Zählregister verteilt werden. Der Ladewert ist die Zahl, die in das Zählregister geladen wird um die gewünschte Zeit zu erreichen.

Zählfrequenz = Frequenz am Timereingang = Quarzfrequenz (MHz) / 12

Taktzeit = Zeit für 1 Zähltakt = 1 / Zählfrequenz (MHz)

Zeitwert = Anzahl der Zählschritte (µs) = Gewünschte Zeit (µs) / Taktzeit (us)

Ladewert = Zahl für Zählregister = negativer Zeitwert = Überlaufwert - Zeitwert

Überlaufwerte:
8Bit Timer 256d (100h)
13Bit Timer 8192d (2000h)
16Bit Timer 65536d (10000h)

Beispiel: Quarzfrequenz = 12MHz Gewünschte Zeit = 10ms

Für 10ms (10000µs) muß ein 16Bit Timer programmiert werden.

Zählfrequenz = Quarz (MHz) / 12 = 12MHz / 12 = 1MHz

Taktzeit = 1 / Zählfrequenz (MHz) = 1 / 1MHz = 1µs

Zeitwert = Gewünschte Zeit (us) / Taktzeit (us) = 10000µs / 1µs = 10000 (2710h)

Ladewert = negativer Zeitwert = -10000d

= Überlaufwert - Zeitwert = 65536d - 10000d = 55536 (D8F0h)

In Assemblerprogrammen bestehen mehrere Möglichkeiten den Ladewert in das Zählregister zu bringen.

13/16Bit Zeitgeber: Zeitwert = 10000d (2710h) Ladewert = -10000 (D8F0h)

```
MOV TLx,#0F0h          ;Ladewert Low
MOV THx,#0D8h          ;Ladewert High
```

oder

```
MOV DPTR,#-10000       ;Zeitwert dezimal negativ
MOV TLx,DPL            ;Wert in das Zählregister -
MOV THx,DPH            ;umladen
```

8Bit Zeitgeber: Zeitwert = 200d (C8h) Ladewert = -200d (38h)

```
        MOV TLx,#-200       ;Zeitwert dezimal negativ
oder
        MOV TLx,#38h        ;Ladewert hexadezimal
```

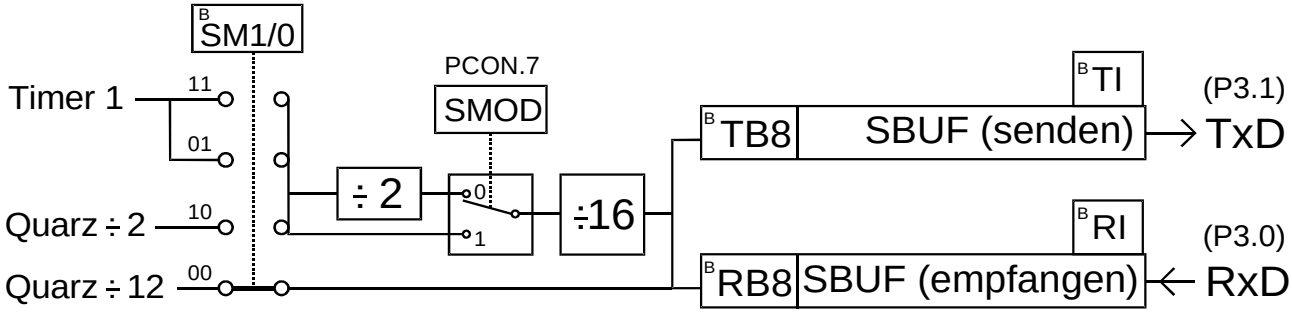
Programmbeispiel: Timer 0 als 16Bit Zeitgeber ohne Torsteuerung Zeit = 10ms

```
CLR TR0             ;Timer 0 anhalten
CLR TF0             ;evtl. vorhandenen Überlauf löschen
ANL TMOD,#0F0h      ;Alle Timer 0 Modus-Bit löschen
ORL TMOD,#01h       ;Timer 0 = 16Bit Timer ohne Gate
MOV TL0,#0F0h       ;Ladewert Low -
MOV TH0,#0D8h       ;Ladewert High für 10ms
SETB TR0            ;Timer 0 starten
JNB TF0,$           ;Auf Überlauf warten
CLR TF0             ;Überlaufbit löschen
```

Vom Start des Timers bis zum Setzen des Überlaufbit vergeht die gewünschte Zeit. Das Überlaufbit kann, wie hier, durch Software abgefragt werden oder einen Interrupt auslösen. Bei Interruptsteuerung wird das Überlaufbit TF0, bei Annahme des Interrupt, automatisch gelöscht. Bei Softwareabfrage muß das Überlaufbit durch Software gelöscht werden (CLR TF0).

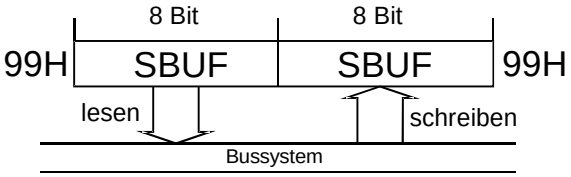
4.5 Serielle Schnittstelle (UART)

In die Mikrocontroller der MCS51-Familie ist ein UART mit verschiedenen Betriebsarten integriert. Das Wort UART steht für Universal Asynchron Receiver Transmitter. Dies bedeutet, daß es sich um eine programmierbare Einheit für serielle Schnittstellen im asynchron Betrieb, mit Sender und Empfänger handelt.



Der UART kann in vier verschiedenen Betriebsarten arbeiten. Es bestehen drei asynchrone Betriebsarten zum Datenaustausch mit anderen Geräten. Die vierte Betriebsart ist synchron und dient der seriellen Erweiterung der Ein/Ausgabeports. Die Übertragungsgeschwindigkeit (Baudrate) wird, je nach verwendeter Betriebsart, direkt aus der Quarzfrequenz oder dem Überlauf des Timer 1 abgeleitet. Wird die Baudrate vom Timer 1 erzeugt, ist sie Programmierbar. Wird sie aus der Quarzfrequenz abgeleitet, steht sie in einem festen Verhältnis zu dieser. In den asynchronen Betriebsarten kann mit einer Datenbreite von acht oder neun Bit gearbeitet werden. Als neuntes Sendebit wird TB8, als neuntes Empfangsbit RB8 verwendet. Beide Bit befinden sich im SFR-SCON und werden bei Programmierung als 9Bit-UART automatisch ins Datenformat eingefügt.

Das Übertragungsregister SBUF ist zweimal unter der gleichen Adresse vorhanden. Einmal für das Senden und einmal für das Empfangen von Daten. Das Senderegister SBUF kann nur beschrieben, das Empfangsregister SBUF nur gelesen werden. Bei Zugriffen auf SBUF legt also die Übertragungsrichtung fest, mit welchem der beiden SBUF-Register gearbeitet wird. Durch die zwei getrennten Übertragungsregister ist es möglich, gleichzeitig zu Senden und zu Empfangen (Voll-Duplex Betrieb).



Ist die Übertragung eines Zeichens beendet wird, für jede Richtung ein eigenes Kennzeichen-Bit gesetzt. Nach abgeschlossenem Senden wird das Bit-TI (Transmit Interrupt), nach abgeschlossenem Empfang das Bit-RI (Receive Interrupt) gesetzt. Diese Bit können durch Software abgefragt werden, oder einen Interrupt auslösen. Bei Interruptsteuerung lösen beide Bit den gleichen Interrupt (Serial Interrupt) aus, wodurch innerhalb der Interrupt-Service-Routine die Quelle des Interrupt geprüft werden muß. Aus diesem Grund werden die beiden Bit nicht durch die Hardware zurückgesetzt. Sie müssen immer durch Software gelöscht werden.

Die Steuerung des UART erfolgt im SFR-SCON.

98H	B SM0	B SM1	B SM2	B REN	B TB8	B RB8	B TI	B RI	SCON

SM0/SM1 Serial Mode 0/1 Wahl der Betriebsart des UART

SM0	SM1	Betriebsart
0	0	Synchron 8Bit Schieberegister
0	1	Asynchron 8Bit UART variable Baudrate
1	0	Asynchron 9Bit UART feste Baudrate
1	1	Asynchron 9Bit UART variable Baudrate

SM2 Serial Mode 2 Unterstützung des Netwerk-Betriebs

REN Receiver Enable Freigabe des Empfangs im UART

0 = Empfang gesperrt
1 = Empfang freigegeben

TB8 Transmit Bit 8 Neuntes Sendebit für Betriebsart 2 und 3

Wird durch die Hardware ins Datenformat eingefügt.

RB8 Receive Bit 8 Neuntes Empfangsbit für Betriebsart 2 und 3

Wird durch die Hardware ins Datenformat eingefügt.

TI Transmitter Interrupt Zeigt ein abgeschlossenes Senden an

0 = Sendevorgang läuft oder nach Reset
1 = Sendevorgang abgeschlossen
Wir durch Hardware gesetzt, muß immer durch Software gelöscht werden.

RI Receiver Interrupt Zeigt ein abgeschlossenes Empfangen an

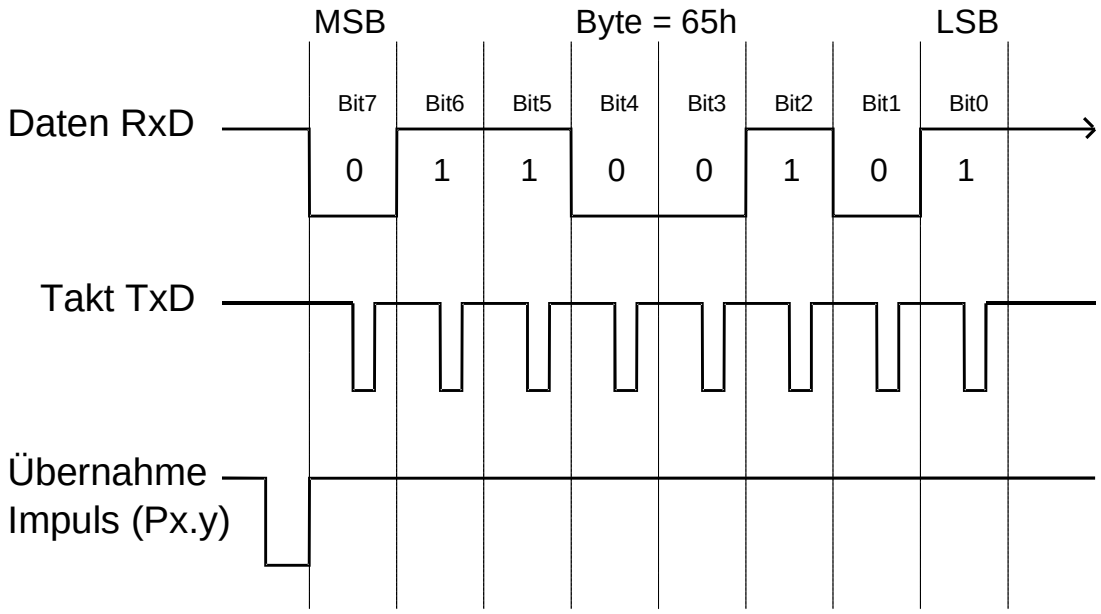
0 = Empfangsvorgang läuft oder nach Reset
1 = Empfangsvorgang abgeschlossen
Wir durch Hardware gesetzt, muß immer durch Software gelöscht werden.

4.5.1 Die Betriebsart 0 - synchrone Ein/Ausgabeerweiterung

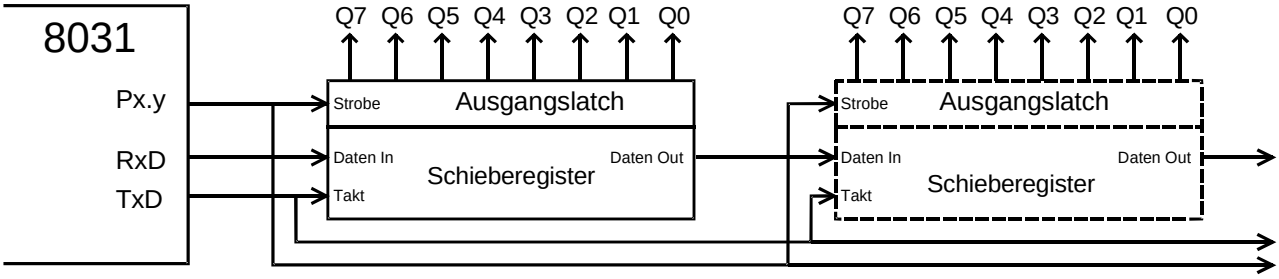
In Betriebsart 0 können die parallelen Ein/Ausgabeports, mit Hilfe von Schieberegistern, erweitert werden. Die Ein- oder Ausgabe eines Byte wird am Pin TxD durch jeweils acht Taktimpulse begleitet. Die Taktfreqeunz ist dabei festgelegt auf Quarzfrequenz/12. Bei einem 12MHz Quarz wird damit ein Schiebetakt von 1MHz (1µs) verwendet, ein Byte wird also in 8µs eingelesen oder ausgegeben. Es können theoretisch beliebig viele Schieberegister in Serie geschaltet werden, um die Anzahl der Ein/Ausgabeports beliebig zu erweitern. Praktisch ist die Anzahl der Schieberegister durch die Ausgangsbelastbarkeit der Mikrocontroller-Ausgänge eingeschränkt. Diese sind in der Lage einen Standard-TTL Eingang oder fünfzig CMOS Eingänge zu treiben.

Ausgabeerweiterung

Zur Ausgabeerweiterung muß der UART in Betriebsart 0 (SM0=0,SM1=0) programmiert sein. Die Ausgabe eines Byte wird durch Schreiben desselben in das SFR-SBUF gestartet. Der UART gibt die acht Bit des Zeichen am Pin RxD (P3.0) der Reihe nach aus. Die Übertragung beginnt mit dem niederwertigsten (LSB) und endet mit dem höchstwertigen (MSB) Bit des Byte. Begleitend zu den einzelnen Bit wird am Pin TxD (P3.1) der Takt mitgeführt. Beide Taktflanken befinden sich innerhalb der Ausgabezeit für das entsprechende Bit, wodurch Schieberegister mit Datenübernahme auf steigender oder fallender Flanke verwendet werden können. Ist das Byte vollständig ausgegeben, wird das Bit-TI durch die Hardware des UART gesetzt.



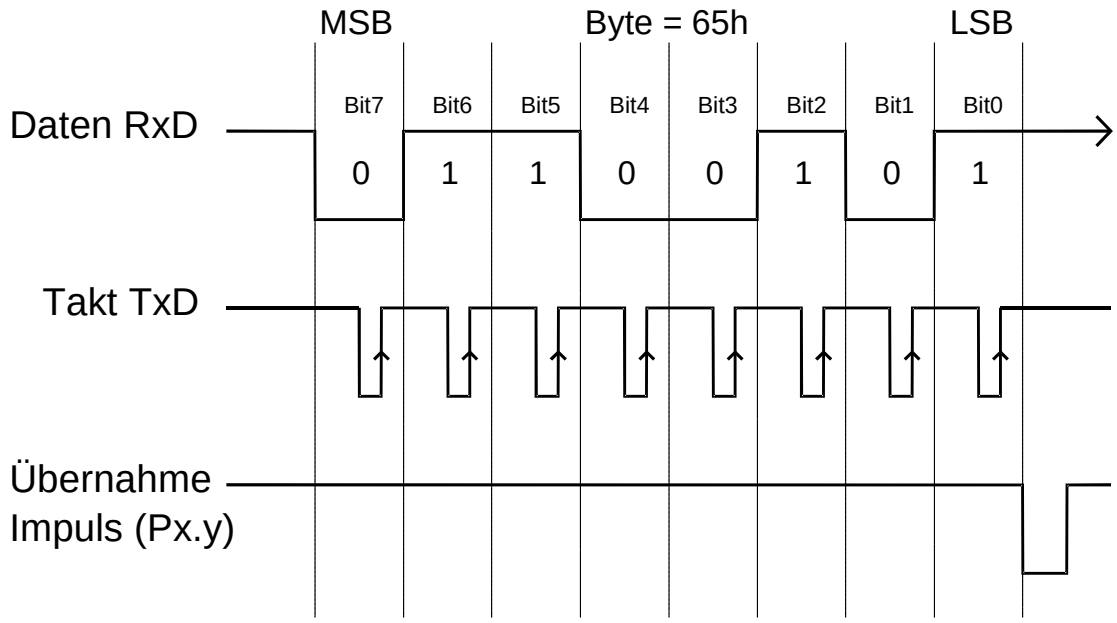
Die Erweiterung der Ausgabeports erfolgt durch Schieberegister mit seriellen Eingang, parallelen Ausgang und Ausgangslatch (z.B:74595). Das Ausgangslatch wird benötigt, um den Schiebevorgang von den Ausgabeleitungen ferzuhalten. Erst wenn das Byte vollständig im Schieberegister liegt, wird es durch einen Übernahmeimpuls in das Ausgangslatch und damit zu den Ausgängen übernommen. Der Übernahmeimpuls kann von einem beliebigen freien Portbit erzeugt werden.



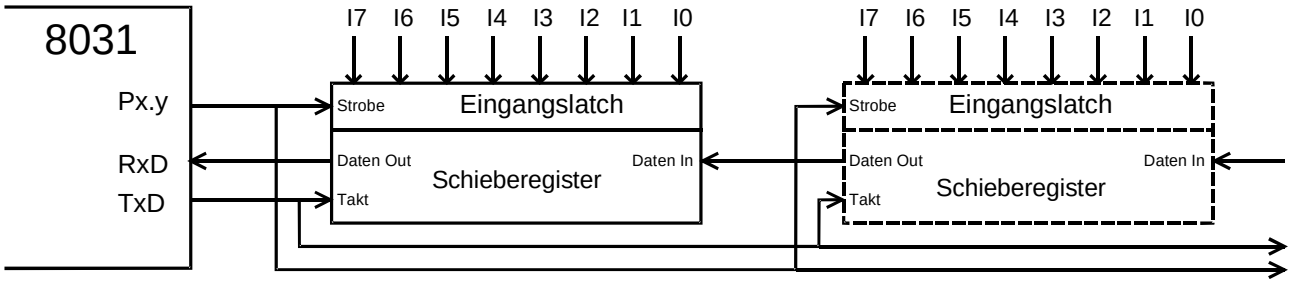
Die Erweiterung der Ausgabeports ist dabei nicht auf ein einziges Schieberegister beschränkt. Es können mehrere davon in Serie geschaltet werden. Dabei ist auf das FAN-OUT (Ausgangsbelastbarkeit) des Kontrollers zu achten. Die Ausgabebyte für die einzelnen Schieberegister werden der Reihe nach ausgegeben und erst am Ende wird ein gemeinsamer Übernahmeimpuls für alle Schieberegister erzeugt. Werden Ausgänge mit höherer Strombelastbarkeit benötigt, kann ein Schieberegister ohne Latch mit nachgeschaltetem Leistungslatch (z.B: 74374) verwendet werden.

Eingabeerweiterung

Zur Eingabeerweiterung muß der UART in Betriebsart 0 (SM0=0,SM1=0) programmiert sein. Das Einlesen wird gestartet, wenn das Bit-REN = "1" (Receive Enable) und das Bit-RI = "0" (Receiver Interrupt). Bei gelöschtem Bit-RI startet das Setzen von REN das Einlesen. Bei gesetztem Bit-REN startet ein Löschen von RI das Einlesen. Nach dem Start gibt der UART am Pin-TxD acht Taktimpulse aus und übernimmt bei jeder steigenden Flanke des Taktes das Bit von RxD. Nach den acht Taktimpulsen befindet sich das eingelesene Byte im SFR-SBUF und das Bit-RI wird gesetzt. Ein weiteres Einlesen kann nun durch Löschen von RI gestartet werden.

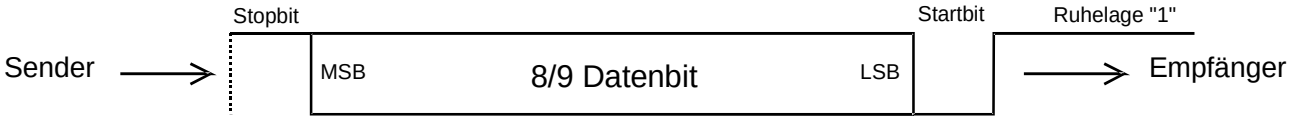


Die Erweiterung der Eingabeports erfolgt durch Schieberegister mit parallelen Eingang, seriellen Ausgang und Eingangslatch. Das Eingangslatch wird benötigt, um Änderungen an den Eingängen während des Einlesevorganges nicht zu Übernehmen. Vor dem eigentlichen Einlesen, werden durch einen Übernahmeimpuls die Zustände der Eingangsbit in das Latch übernommen. Der Übernahmeimpuls kann von einem beliebigen freien Portbit erzeugt werden.



Die Erweiterung der Eingabeports ist dabei nicht auf ein einziges Schieberegister beschränkt. Es können mehrere davon in Serie geschaltet werden. Dabei ist auf das FAN-OUT (Ausgangsbelastbarkeit) des Kontrollers zu achten. Zuerst wird ein gemeinsamer Übernahmeipuls für alle Schieberegister erzeugt. Darauf werden die Eingabebyte für die einzelnen Schieberegister der Reihe nach eingelesen.

Die asynchronen Betriebsarten dienen dem Datenaustausch mit anderen Datenverarbeitungs-Geräten. Da im asynchron-Betrieb kein Taktsignal zwischen den Teilnehmern existiert, wird die Synchronisation durch ein festgelegtes Datenformat erreicht. Das Datenformat ist ein Rahmen für jedes einzelne zu übertragende Zeichen. Findet kein Datentransfer statt, befindet sich die Datenleitung in einem definierten Ruhezustand. Dieser Ruhezustand ist der Logikpegel „1“ und damit am TxD-Ausgang des 8051-UART der Spannungspegel High. Die Übertragung eines Zeichens beginnt mit einem Startbit. Dieses Startbit erzeugt der Sender, indem er den Zustand der Datenleitung für die Dauer eines Bit in den Arbeitspegel (Logisch „0“) versetzt. Direkt nach dem Startbit folgt die Übertragung der Bit des Zeichens. Zuerst wird das niederwertigste (LSB) und zuletzt das höchstwertigste (MSB) Bit des Zeichens übertragen. Nach den Bit des Zeichens wird, automatisch durch den UART, ein Stopbit gesendet. Das Stopbit weist den Ruhepegel („1“) der Datenleitung auf.



Ist der Empfänger freigegeben (REN = 1), wartet er auf die fallende Flanke des Startbit. Ausgehend von dieser Flanke, wird die Zeit bis zur Startbitmitte abgewartet. Von diesem Zeitpunkt aus wird nach jeweils einer ganzen Bitzeit der Zustand der Datenleitung ausgewertet. Durch dieses synchronisieren auf die Mitte des Startbit ist bei gleicher Baudrate von Sender und

Empfänger sichergestellt, daß die folgenden Bit des Zeichens jeweils in ihrer Mitte abgetastet werden. Kleine Abweichungen der Baudraten von Sender und Empfänger sind Tolerierbar, solange die Auswertung des letzten Bit noch innerhalb dessen Bitzeit erfolgt. Bei jedem einzelnen Zeichen wird die Synchronisation durch das Startbit neu durchgeführt. Die Übertragungsrate wird als Baudrate bezeichnet, sie ist definiert als Bit pro Sekunde (Bit/s). Die Angabe 2400 Baud bedeutet, es werden 2400 Bit in einer Sekunde übertragen. Die Bitdauer ist damit 1/2400 Sekunden. In Betriebsart 1 und 3 wird die Baudrate aus den Überläufen des Timer 1 erzeugt. Für Senden und Empfangen wird die gleiche Baudrate verwendet. Besonders gut geeignet zum Erzeugen der Standard-Baudraten (150-19200 Baud) ist ein Quarz mit einer Frequenz von 11,0592 MHz. Bei der Berechnung des Ladewertes für Timer 1 ist der Zustand des Bit-SMOD (PCON.7) zu berücksichtigen. Mit dem Bit-SMOD ist es möglich die programmierte Baudrate zu verdoppeln bzw. zu halbieren. Der Timer 1 ist als 8Bit Zeitgeber autoreload zu Programmieren. Das Senden wird durch Schreiben eines Byte ins SFR-SBUF gestartet. Der Empfänger wird durch setzen des Bit-REN (SCON.4) freigegeben und wartet daraufhin auf die fallenden Flanken der Startbit. Der Empfänger bleibt aktiv, bis das Bit-REN wieder gelöscht wird.

$$\text{Ladewert Timer 1 (SMOD=0)} \quad 256 - \frac{\text{Quarzfrequenz (Hz)}}{(\text{Baudrate} * 12 / 0{,}03125)}$$

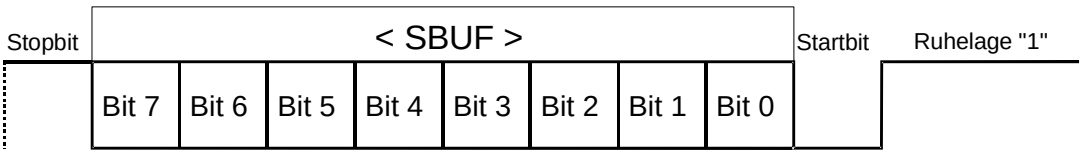
$$\text{Ladewert Timer 2 (SMOD=1)} \quad 256 - \frac{\text{Quarzfrequenz (Hz)}}{(\text{Baudrate} * 12 / 0{,}0625)}$$

Eine eventuell benötigte Paritätsprüfung wird durch den UART nicht unterstützt. Sie muß im höchsten Bit des Zeichens durch Software bedient werden.

Hinweis: In den Mikrocontrollern 80515/535 ist ein Baudratengenerator eingebaut. Wird dieser benutzt, wird die Baudrate nicht mehr von den Überläufen des Timer 1 abgeleitet. Der Baudratengenerator ist aktiv, wenn im SFR-ADCON das Bit-BD (ADCON.7) gesetzt ist.

4.5.3 Die Betriebsart 1 - 8Bit-UART variable Baudrate

In Betriebsart 1 arbeitet der UART asynchron mit programmierbarer Übertragungsrate und einer Datenbreite von acht Bit. Das Datenformat besteht aus einem Start-, 8Daten- und einem Stopbit. Wird eine Paritätsprüfung benötigt, stehen nur 7Daten- und ein Paritätsbit (Bit 7) zur Verfügung. Die Baudrate wird aus den Überläufen des Timer 1 erzeugt.

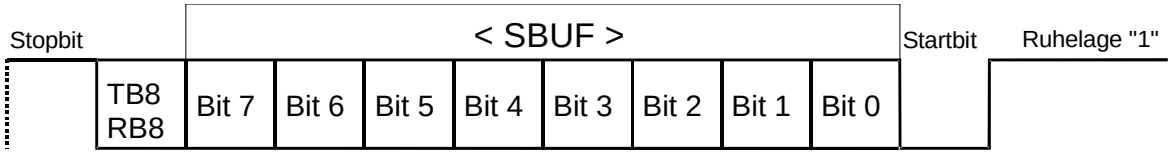


4.5.4 Die Betriebsart 2 - 9Bit-UART feste Baudrate

In Betriebsart 2 arbeitet der UART asynchron mit programmierbarer Übertragungsrate und einer Datenbreite von neun Bit. Das Datenformat besteht aus einem Start-, 9Daten- und einem Stopbit. Wird eine Paritätsprüfung benötigt, stehen nur 8Daten- und ein Paritätsbit zur Verfügung. Das neunte Sendebit ist das Bit-TB8 (SCON.3), das neunte Empfangsbit ist das Bit-RB8 (SCON.2). Diese beiden Bit werden durch den UART automatisch in das Datenformat eingefügt. Für das sinnvolle Beschreiben und Auswerten dieser beiden Bit ist die Anwendersoftware verantwortlich. Die Baudrate wird starr aus der Quarzfrequenz erzeugt.

Baudrate (SMOD=0) = 0,015625 * Quarzfrequenz

Baudrate (SMOD=1) = 0,03125 * Quarzfrequenz



4.5.5 Die Betriebsart 3 - 9Bit-UART variable Baudrate

In Betriebsart 3 arbeitet der UART asynchron mit programmierbarer Übertragungsrate und einer Datenbreite von neun Bit. Das Datenformat besteht aus einem Start-, 9Daten- und einem Stopbit. Wird eine Paritätsprüfung benötigt, stehen nur 8Daten- und ein Paritätsbit zur Verfügung. Das neunte Sendebit ist das Bit-TB8 (SCON.3), das neunte Empfangsbit ist das Bit-RB8 (SCON.2). Diese beiden Bit werden durch den UART automatisch in das Datenformat eingefügt. Für das sinnvolle Beschreiben und Auswerten dieser beiden Bit ist die Anwendersoftware verantwortlich. Das Datenformat entspricht der Betriebsart 2. Die Baudrate wird aus den Überläufen des Timer 1 erzeugt.

4.6 Interrupt

Es gibt in der Computertechnik zwei Möglichkeiten, Ereignisse der Hardware zu Erfassen. Zyklisches Abfragen (Polling) und Interrupt (Unterbrechung). Beim Polling wird, für das ständige Abfragen, Rechenzeit der CPU benötigt, wodurch die eigentliche Programmbearbeitung verlangsamt abläuft. Beim Interrupt wird ein laufendes Programm erst dann unterbrochen, wenn ein Bedienungswunsch eines Gerätes vorliegt. Ein Interrupt ist also eine Art „Türklingel“ mit der sich Geräte mit einem Bedienungswunsch bei der CPU anmelden können. Wird ein Interrupt angenommen, unterbricht die CPU die laufende Programmabarbeitung und führt die dem Interrupt zugeordnete Interrupt Service Routine (ISR) aus. Die ISR ist ein Unterprogramm, welches alle nötigen Befehle enthält um den Bedienungswunsch der Hardware zu erfüllen. Eine ISR endet immer mit dem RETI-Befehl (Return Interrupt). Nach Beendigung der ISR kehrt die Bearbeitung der CPU zur unterbrochenen Stelle im laufenden Programm zurück. Die Interruptquellen können interne und externe Ereignisse sein. Interne Interrupts werden durch die in den Mikrocontroller integrierte Hardware ausgelößt. Externe Interrupts werden durch Hardware ausserhalb des Mikrocontrollers ausgelößt. Sie gelangen durch je einen Pin am Mikrocontroller zur Interruptlogik. Jedem Interrupt ist eine eindeutige Adresse im Programmspeicher zugeordnet, die bei Annahme des Interrupt durch einen LCALL-Befehl angesprungen wird. An der Zieladresse kann die ISR selbst oder ein Sprung darauf stehen. Vom Auftreten des Interrupt bis zum Start der ISR vergeht eine bestimmte Zeit.

1 Befehlszyklus zum Speichern der Interrupts

1 Befehlszyklus zum Erkennen der Interrupts

2 Befehlszyklen zum Ausführen des LCALL-Befehls

Unter gewissen Umständen kann sich die Zeit allerdings verlängern.

Wenn gerade ein Interrupt gleicher oder höherer Priorität bearbeitet wird.

Wenn ein RETI-Befehl ausgeführt wird oder ein Zugriff auf die SFR IE oder IP stattfindet.

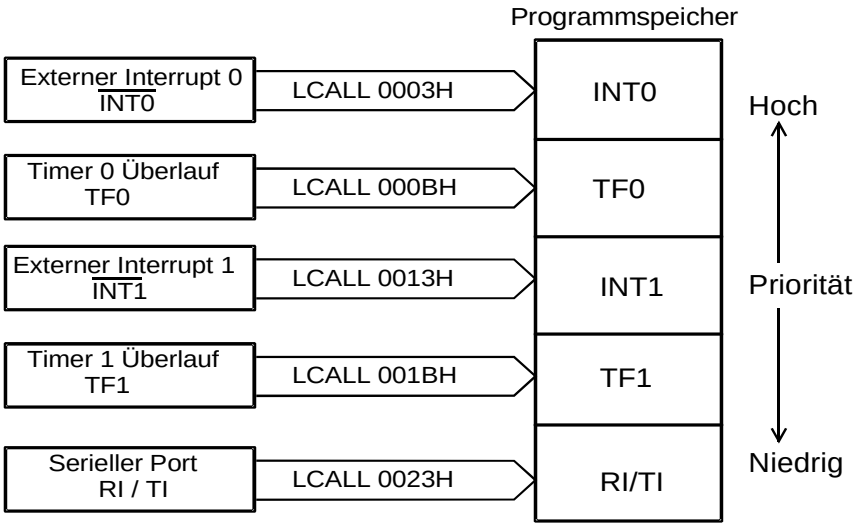
Wenn, bei Befehlen mit mehreren Befehlszyklen, der aktuelle Zyklus nicht der Letzte ist.

Interrupt des 8051

Der 8051/31 verfügt über drei interne und zwei externe Interrupts.

Intern:	Überlauf des Timer 0	- TF0
	Überlauf des Timer 1	- TF1
	Senden/Empfangen des UART	- RI/TI
Extern:	Externer Interrupt 0	- INT0
	Externer Interrupt 1	- INT1

Diesen Interrupts sind folgende Einsprungadressen zugeordnet.



Jeder Interrupt kann einer von zwei Prioritätsstufen zugeordnet werden. Die Programmierung der Prioritätsstufe findet im SFR-IP (Interrupt Priority) statt. Die ISR eines Interrupt niedriger Priorität kann durch einen Interrupt höherer Priorität unterbrochen werden. Durch einen Interrupt gleicher oder niedrigerer Priorität kann sie nicht unterbrochen werden. Sind mehrere Interrupts auf die gleiche Prioritätsstufe programmiert und treten gleichzeitig auf, entsprechen die Prioritäten der Reihenfolge der Abfrage durch die CPU.

Höchste < INT0, TF0, INT1, TF1, RI/TI > Niedrigste

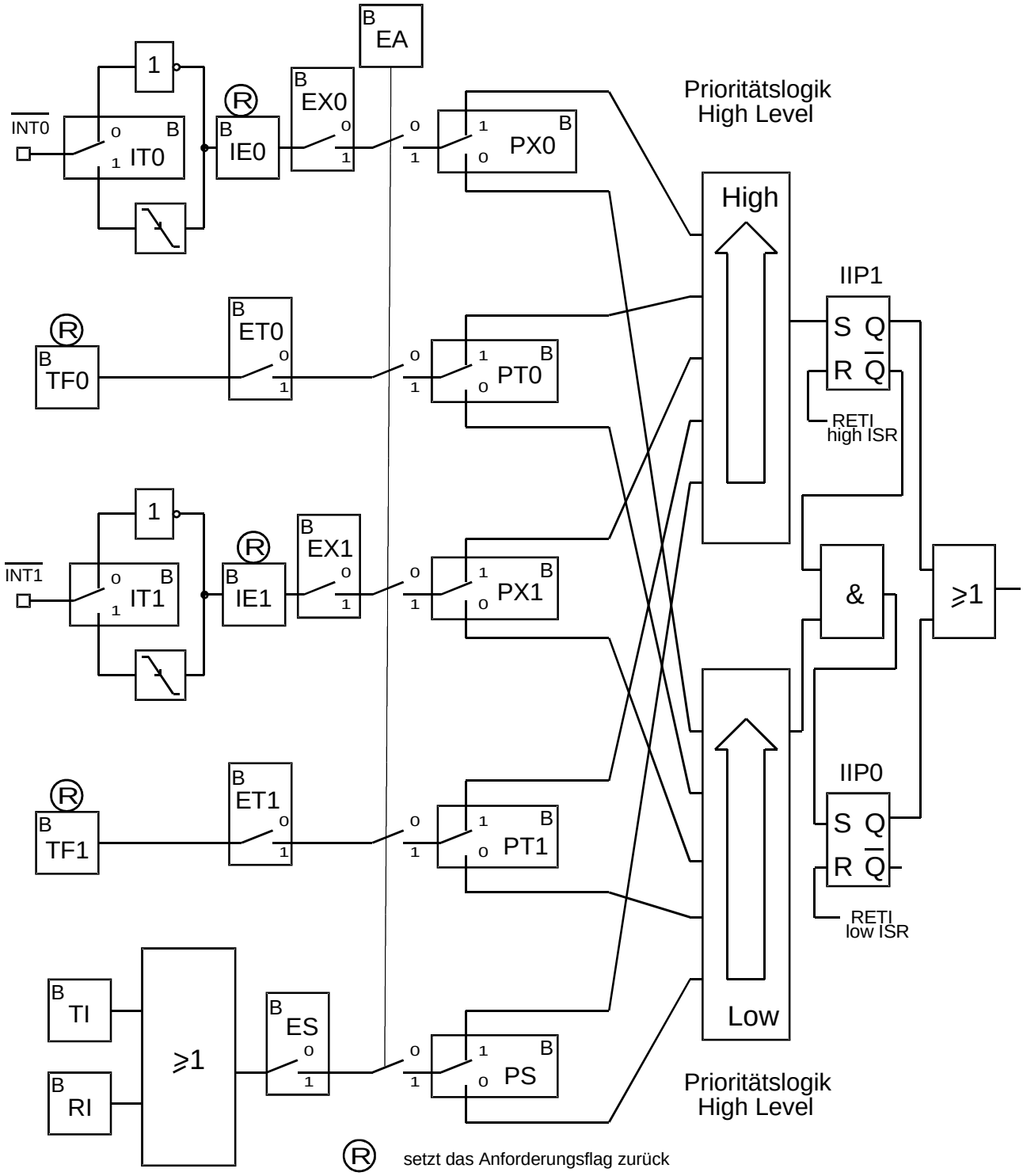
IP	B	B	B	B	B	B	B	B	B8h
	--	--	--	PS	PT1	PX1	PT0	PX0	

PX0	Priority eXternal interrupt 0	Priorität des externen Interrupt INT0
PT0	Priority Timer 0	Priorität des Timer 0 Überlauf TF0
PX1	Priority eXternal interrupt 1	Priorität des externen Interrupt INT1
PT1	Priority Timer 1	Priorität des Timer 1 Überlauf TF1
PS	Priority Serial port	Priorität des UART Interrupt RI/TI

Für alle Interrupts gilt: "0" = niedrige Priorität

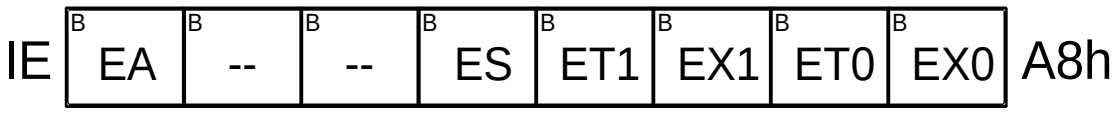
"1" = hohe Priorität

Nach Reset haben alle Interrupts niedrige Priorität: IP = xxx0 0000



Kommt ein Interrupt zur Interruptlogik durch, wird bei hoher Priorität das Flip-Flop IIP1 (Interrupt in Progress), bei niedriger Priorität das Flip-Flop IIP0 gesetzt. Durch das Setzen des Flip-Flop wird die Annahme eines weiteren Interrupt der gleichen Priorität verhindert. Wird das IIP1 Flip-Flop der hohen Prioritätsstufe gesetzt, verhindert dieses ein Setzen des Flip-Flop für niedrige Prioritäten. Das Interrupt Flip-Flop wird jeweils durch die Ausführung des RETI-Befehls zurückgesetzt, wodurch die Annahme neuer Interrupts möglich wird.

Die Interrupts können im SFR-IE (Interrupt Enable) gesperrt und freigegeben werden. Das Bit-EA beeinflußt alle Interrupts. Für jeden einzelnen Interrupt ist zudem ein eigenes Freigebebit vorhanden.



EA	Enable All	Sperren/Freigeben aller Interrupts
ES	Enable Serial	Sperren/Freigeben des RI/TI Interrupt
ET1	Enable Timer 1	Sperren/Freigeben des TF1 Interrupt
EX1	Enable eXternal interrupt 1	Sperren/Freigeben des externen Interrupt INT1
ET0	Enable Timer 0	Sperren/Freigeben des TF0 Interrupt
EX0	Enable eXternal interrupt 0	Sperren/Freigeben des externen Interrupt INT0

Für alle Interrupts gilt: "0" = Gesperrt

"1" = Freigegeben

Nach Reset sind alle Interrupts gesperrt: IE = 0xx0 0000

Die externen Interrupts INT0/1

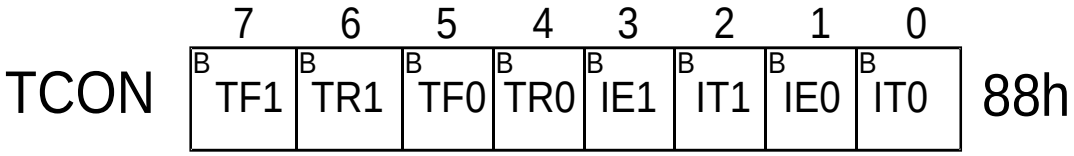
Die externen Interrupts gelangen über jeweils einen eigenen Port-Pin in den Mikrocontroller (INT0 = P3.2, INT1 = P3.3). Durch das Bit IT0/1 kann ihr Ansprechverhalten festgelegt werden. Ist IT0/1 "0", reagiert der Interrupt auf einen Low-Pegel. Ist IT0/1 "1", reagiert der Interrupt auf eine fallende Flanke. Tritt das Interrupt-Ereignis auf, setzt es das Anforderungsbit IE0/1 um den Interrupt zu speichern. Dieses Bit wird auch bei gesperrten Interrupt gesetzt. Bei gesetztem IE0/1-Bit wird der Interrupt, sofort nach seiner Freigabe, ausgeführt. Bei Annahme des Interrupt wird das Anforderungsbit automatisch gelöscht. Das Anforderungsbit befindet sich im bitadressierbaren SFR-TCON, wodurch es über Software gesetzt und gelöscht werden kann. Sollte der Interrupt nach seiner Freigabe nicht sofort durch ein eventuell gesetztes Anforderungsbit ausgeführt werden, ist das Bit vor der Freigabe zu löschen (CLR IE0/1). Das Anforderungsbit kann zum Testen einer ISR durch Software gesetzt werden, wodurch ein Interrupt vorgetäuscht wird.

Die Timerüberlauf Interrupts TF0/1

Beim Überlauf eines der Timer wird automatisch das entsprechende Überlaufbit (TF0/1) gesetzt. Bei freigegebenen Interrupt lößt das Überlaufbit diesen aus. Durch Annahme des Interrupt wird das Überlaufbit automatisch gelöscht.

Der Interrupt des UART

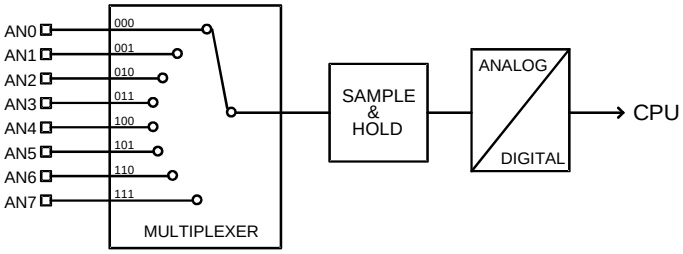
Wurde ein Zeichen vom UART vollständig gesendet, setzt er das Bit-TI (Transmitter Interrupt). Wurde ein Zeichen vom UART vollständig empfangen, setzt er das Bit-RI (Receiver Interrupt). Die beiden Bit lösen bei Freigabe den gleichen Interrupt aus. Die Interruptlogik kann nicht zwischen dem RI- und dem TI-Interrupt unterscheiden. Tritt der Interrupt der seriellen Schnittstelle auf, muß innerhalb der ISR die Quelle (RI oder TI) ermittelt werden. Die Anforderungsbit RI/TI werden durch die Interruptlogik nicht automatisch gelöscht. Sie müssen innerhalb der ISR durch Software gelöscht werden.



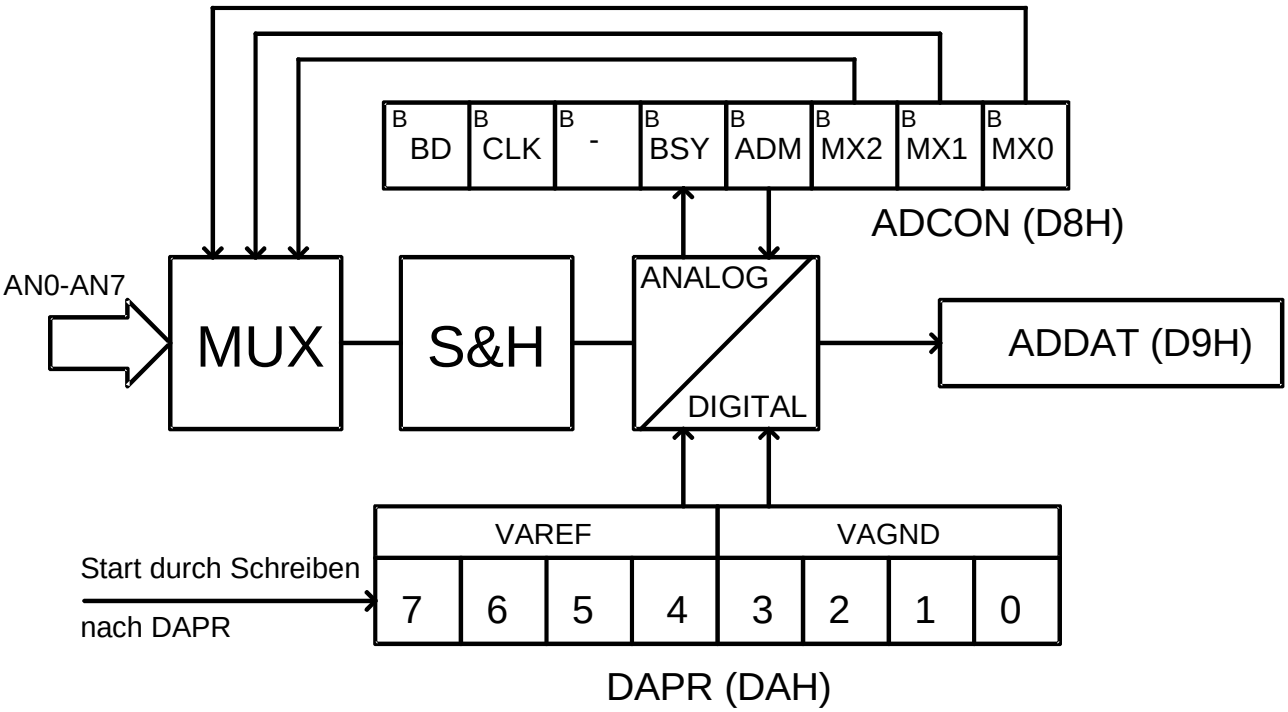
IT0	Interrupt Type 0	Ansprechverhalten externer Interrupt 0
	"0" = Pegelgesteuert, "1" = Flankengesteuert	
IE0	Interrupt Edge 0	Speicherbit externer Interrupt 0
	"1" = Interrupt steht an. Wird durch Annahme des Interrupt automatisch gelöscht.	
IT1	Interrupt Type 1	Ansprechverhalten externer Interrupt 1
	"0" = Pegelgesteuert, "1" = Flankengesteuert	
IE1	Interrupt Edge 1	Speicherbit externer Interrupt 1
	"1" = Interrupt steht an. Wird durch Annahme des Interrupt automatisch gelöscht.	

4.6 Der A/D-Wandler des 80515/535

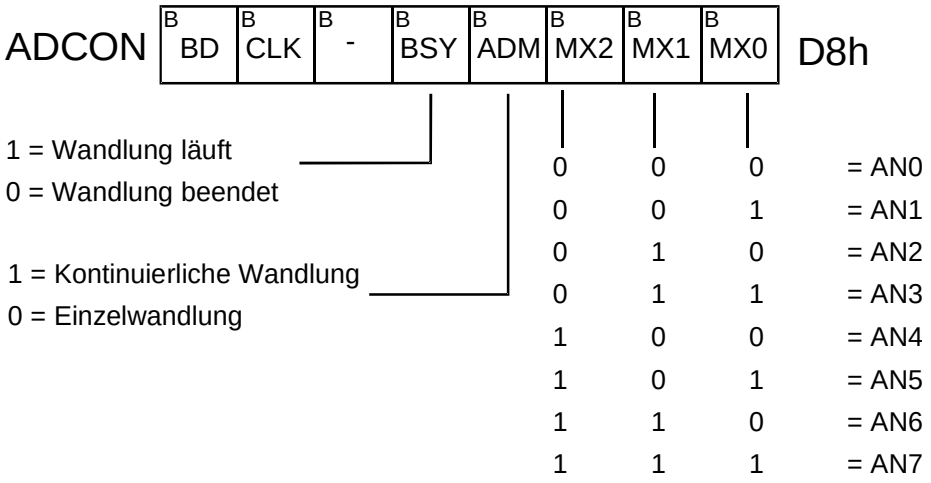
Der A/D-Wandler ist in den Mikrocontroller 80515/535 integriert. Es handelt sich dabei um einen 8Bit-Wandler mit vorgeschalteten Multiplexer, wodurch acht Analog-Kanäle (AN0-AN7) zur Verfügung stehen. Durch das Multiplexen kann immer nur einer dieser Kanäle zu einer Zeit mit dem A/D-Wandler verbunden sein.



Zur Wandlung wird das SAR-Verfahren benutzt, welches hier aber nicht mit R-2R-Netzwerk sondern mit einem kapazitiven Netzwerk arbeitet. Der Wandler stellt für die zu messende Spannung eine kapazitive Last dar. Der Innenwiderstand der zu messenden Spannungsquelle darf maximal 10 kW betragen. Die Referenzspannung des A/D-Wandlers ist programmierbar, wodurch bei einer speziellen Programmierung auch eine 10Bit-Auflösung erreicht wird. Der Vorteil der höheren Auflösung wird, gegenüber der 8Bit-Wandlung, mit einem höheren Softwareaufwand und einer damit verbundenen längeren Wandlungszeit erkauft. Die Wandlungszeit bei 8Bit-Auflösung beträgt 15 Maschinenzyklen mit einem 80515 (bei 12MHz Takt - 15us), und 13 Maschinenzyklen mit einem 80C515. Eine Wandlung wird durch einen Schreibvorgang ins Referenzspannungsregisters DAPR (DAH) gestartet, während der Wandlung ist das Bit BSY (BuSY, ADCON.4) gesetzt. Bei beendeter Wandlung wird das Bit BSY wieder rückgesetzt und das Bit IADC (IRCON.0) gesetzt, welches den A/D-Wandler Interrupt (Vektor 0043H) auslösen kann. Das Ergebnis der Wandlung steht am Ende im SFR-ADDAT (D9H) bereit.



Im SFR ADCON (A/D-CONtrol, D8H) wird die Programmierung des A/D-Wandlers vorgenommen. Die Bit MX0-MX2 wählen einen der acht Analog-Eingänge aus. Durch das Bit ADM (ADCON.3) wird einzel oder Dauerwandlung eingestellt. Bei Dauerwandlung wird direkt nach dem Ende einer Wandlung die nächste gestartet. In dieser Betriebsart sollte das Ende der Wandlung durch Interrupt erfaßt werden, da die Software unter Umständen die kurzen BSY-Signale nicht korrekt erkennt. Das Bit BSY zeigt durch eine „1“ eine laufende Wandlung an, eine „0“ zeigt das Ende einer Wandlung oder einen noch nicht gestarteten Wandler an.



Die Referenzspannung des A/D-Wandlers kann direkt aus der Versorgungsspannung VCC oder einer eigenen Spannungsquelle gewonnen werden. Es ist für eine gut stabilisierte Versorgungsspannung zu sorgen, da durch die von ihr abgeleitete Referenz die Messgenauigkeit abhängt. Um Störungen aus der Versorgungsspannung zu dämpfen, wird zwischen dieser und den Referenzeingängen ein Filter eingesetzt. Die Spannung VAREF muß im Bereich der positiven Versorgungsspannung VCC ± 5% liegen. Die Spannung VAGND muß im Bereich der Massespannung VSS bis VSS + 0,2V liegen, es sind keine negativen Spannungen erlaubt. Die intern verwendete Referenzspannung ist für beide Grenzwerte programmierbar. Der untere Grenzwert wird als VAGND, der Obere als VAREF bezeichnet. Bei der Programmierung der Referenzspannungen ist zu beachten, daß der obere Referenzwert VAREF um mindestens 1Volt positiver als VAGND bleiben muß. Die interne Referenzspannung wird bei ihrer Programmierung durch Teilung aus der externen Referenzspannung gewonnen. Die Schrittweite (Spannungswert je Binärschritt) des A/D-Wandlers ist maßgeblich von der programmierten Referenzspannung abhängig.

$$\text{Schrittweite} = \frac{\text{VAREF - VAGND}}{2^8} = \frac{\text{VAREF – VAGND}}{256}$$

Die Programmierung der Referenzspannungen wird im SFR DAPR (DAH) vorgenommen. Die untere Tetrade von DAPR dient zum Einstellen von VAGND, die obere Tetrade bestimmt den Wert von VAREF.

Wert	VAREF/GND	VAGND	VAREF
0	0000	0,0V	5,0V
1	0001	0,3125V	-
2	0010	0,625V	-
3	0011	0,9375V	-
4	0100	1,25V	1,25V
5	0101	1,5625V	1,5625V
6	0110	1,875V	1,875V
7	0111	2,1875V	2,1875V
8	1000	2,5V	2,5V
9	1001	2,8125V	2,8125V
A	1010	3,125V	3,125V
B	1011	3,4375V	3,4375V
C	1100	3,75V	3,75V
D	1101	-	4,0625V
E	1110	-	4,375V
F	1111	-	4,6875V

Bei den einzelnen Messungen wird vom ausgewählten Kanal jeweils der Spannungswert zwischen den beiden Referenzspannungen, mit einer Auflösung von 8Bit, erfaßt. Meßwerte die kleiner oder gleich VAGND sind, liefern das Ergebnis 00H, Werte größer oder gleich VAREF liefern das Ergebnis FFH. Da Analogspannungen unter VAGND nicht gemessen werden, wird zur Messung die Analogspannung abzüglich VAGND erfaßt.

$$\text{Meßwert} = \frac{\text{Analogspannung} - \text{VAGND}}{\text{Schrittweite}}$$

Beispiele:

VAGND = 0V, VAREF = 5V, Analogspannung = 3,5V, Auflösung = 8Bit Schrittweite = (VAREF - VAGND) / 256 = 5V/256 = 19,53mV Meßwert = (Analogspannung - VAGND) / Schrittweite = 3500mV / 19,53mV = 179 dezimal = B3 hexadezimal
VAGND = 2,5V, VAREF = 3,75V, Analogspannung = 3,5V, Auflösung = 8Bit Schrittweite = (VAREF - VAGND) / 256 = (3,75V - 2,5V) / 256 = 1250mV / 256 = 4,88mV Meßwert = (Analogspannung - VAGND) / Schrittweite = 1000mV / 4,88mV = 205 dezimal = CD hexadezimal

4.6.1 10Bit Wandlung

Die 10Bit-Auflösung wird durch zwei aufeinanderfolgende 8Bit-Wandlungen erreicht. Die erste Wandlung wird mit Referenzen von VAGND 0Volt und VAREF 5Volt durchgeführt. Sie soll den Bereich der Analogspannung grob ermitteln. Zur zweiten Messung werden die Referenzspannungen um den bei der ersten Messung festgestellten Wert herum eingestellt. Dabei ist zu beachten, daß VAREF mindestens 1Volt positiver als VAGND sein muß. Zum Ergebnis der zweiten Messung wird der Wert von VAGND hinzuaddiert und somit ein Ergebnis mit höherer Auflösung erreicht.

1. Messung: Feststellen des Bereichs der Analogspannung
2. Messung: Messen im festgestellten Bereich mit kleinerer Schrittweite.

   Schrittweite = (VAREF - VAGND)/256 = 1,25V/256 = 4,88mV

   Meßwert = (Analog - VAGND)/Schrittweite = 1V/4,88mV = 205

Ergebnis: Meßwert + (VAGND/Schrittweite) = 205 + 512 = 717d = 2CDh

4.6.2 10Bit Einzeldaten lesen (80515-Programm)

Im Hauptprogramm wird Analogkanal 1 vorgewählt und dann das Unterprogramm zur einfachen 10Bit Wandlung aufgerufen. Das Ergebnis wird anschließend über Standard-Port 0 (Low-Teil) und Standard-Port 1 (High-Teil) ausgegeben. Das Hauptprogramm wird solange wiederholt, bis das T0-Bit den Zustand „1“ annimmt. Dann wird die Autostartkennung im RAM gelöscht und das Betriebssystem bei Adresse 0000H neu gestartet.

Im Unterprogramm wird zunächst zum Ermitteln des Messbereichs eine 8Bit Messung mit einer 0-5V Referenz durchgeführt. Aus der High-Tetrade des Ergebnisses wird die Tabellenadresse des Referenzwertes für die zweite Messung gewonnen. Aus dem VAGND-Wert wird durch Multiplikation mit 64 der Additionswert für VAGND ermittelt. Das Ergebnis der zweiten Messung wird mit dem VAGND-Adittionswert zum Gesamtergebnis addiert. Der High-Teil des Ergebnis steht am Ende im Akku, das Low-Byte im Register R0.

Der Vorteil dieses Unterprogramms liegt darin, daß sich unabhängig vom Meßwert, jeweils gleiche Messzeiten ergeben (ca. 53us = ca. 19000 Messungen/s). Es ist damit für die verschiedensten Aufgaben einsetzbar.

```
0000                $p 80515              ;Ziel = Intel-Hex
0000
0000                ORG 4000H
4000 54 00          ANL A,#0              ;Autostart
4002 02 4020        LJMP START
4005
4005                ORG 4020H
4020                START:
4020 74 01          MOV A,#01             ;Analogkanal 1 wählen
4022 11 33          ACALL AD10BIT         ;10Bit Messung
4024 F5 90          MOV P1,A              ;Wert High zum S-Port1
4026 88 F8          MOV P5,R0             ;Wert Low zum S-Port 0
4028 30 B4 F5       JNB T0,START          ;Wiederholen bis T0=1
402B 90 4000        MOV DPTR,#4000h       ;Autostartkennung
402E E4             CLR A                 ;löschen
402F F0             MOVX @DPTR,A
4030 02 0000        LJMP 0                ;Ende
4033
```

```
4033                    AD10BIT:            ;10Bit Einzelmessung
4033 53 D8 C0           ANL ADCON,#0C0H     ;Einzelmessung
4036 42 D8              ORL ADCON,A         ;Analogkanal einstellen
4038 75 DA 00           MOV DAPR,#0         ;Ref.=0-5V, Wandlung starten
403B 90 405B            MOV DPTR,#REFTAB    ;Basisadresse Referenztabelle
403E 20 DC FD           JB BSY,$            ;Warten bis fertig
4041 E5 D9              MOV A,ADDAT         ;Messwert in Akku
4043 C4                 SWAP A              ;Offset für Tabelle
4044 54 0F              ANL A,#0FH          ;aus Messwert bilden
4046 93                 MOVC A,@A+DPTR      ;Referenzwert laden
4047 F5 DA              MOV DAPR,A          ;2.Wandlung starten
4049 54 0F              ANL A,#0Fh          ;Ref.-Low freistellen
404B 75 F0 40           MOV B,#64           ;Ref.-Low x 64 ist
404E A4                 MUL AB              ;Additionswert
404F 20 DC FD           JB BSY,$            ;Warten bis A/D fertig
4052 25 D9              ADD A,ADDAT         ;Refwert + Messwert
4054 F8                 MOV R0,A            ;ist Ergebnis
4055 E4                 CLR A
4056 E5 F0              MOV A,B             ;Rückgabe: High = Akku
4058 34 00              ADDC A,#0           ;Low = R0
405A 22                 RET                 ;Zurück zum Hauptprog.
405B
405B                    REFTAB:
405B 40 40 40 40 84 84 84 84
                                            db 40h,40h,40h,40h,84h,84h,84h,84h
4063 C8 C8 C8 C8 0C
     0C 0C 0C                               db 0C8h,0C8h,0C8h,0C8h,0Ch,0Ch,0Ch,0Ch
406B
```

5 Software

5.1 Der Befehlssatz

Der Befehlssatz der MCS51-CPU's unterscheidet mehrere Gruppen von Befehlen für die verschiedenen Aufgaben.

Transfer-Befehle:

Zum Datenaustausch zwischen den Komponenten des Mikrocontrollersystems.Die Transfer-Befehle verändern das P-Flag, wenn das Ziel der Akku ist.

Arithmetik-Befehle:

Zum Berechnen von Datenwerten und Adressen. Die arithmetischen Befehle werden in der ALU der CPU ausgeführt. Es stehen die Addition, Subtraktion, Inkrementieren, Dekrementieren, Multiplikation, Division und Dezimalkorrektur zur Verfügung. Die Arithmetischen-Befehle verändern die Flags des PSW.

Rotations-Befehle:

Zum Nachbilden von Schiebe-/Ringregistern und zum Multiplizieren mal zwei oder Dividieren durch zwei.

Logik-Befehle:

Zum logischen Verknüpfen von Datenwerten. Die logischen Verknüpfungen werden in der ALU der CPU ausgeführt. Es stehen die UND, ODER, EXCLUSIV-ODER und NICHT Verknüpfung zur Verfügung. Die Logik-Befehle verändern das P-Flag, wenn das Ziel der Akku ist.

Einzelbit-Befehle:

Zum Bearbeiten von Bitwerten. Es stehen Befehle zum löschen, setzen, invertieren, UND-Verknüpfen, ODER-Verknüpfen und zum Transfer zur Verfügung.

Sprung-Befehle:

Zum Verlassen des linearen Programmlaufs und für Programmentscheidungen. Es stehen bedingte und unbedingte Sprünge bereit. Die Bedingten Sprünge werden nur ausgeführt, wenn ihre Bedingung erfüllt ist. Sie werden für Programmentscheidungen eingesetzt. Die unbedingten Sprünge werden in jedem Fall ausgeführt. Es stehen Sprungbefehle für relative Adressen (+127/-128 Adressen), für Sprungziele innerhalb des aktuellen 2KB-Blocks und innerhalb des gesamten 64KB Programmspeichers zur Verfügung.

Unterprogramm-Befehle:

Zum Aufruf von Unterprogrammen und zur Rückkehr aus diesen. Es stehen CALL-Befehle für Unterprogramme innerhalb des gesamten 64KB Programmspeichers und innerhalb des aktuellen 2KB-Blocks zur Verfügung.

Sonder-Befehle:

Für spezielle Aufgaben.

5.1.1 Transfer-Befehle

MOV = MOVe = Bewegen

Mit Transfer-Befehlen werden Daten zwischen den Komponenten des Mikrocontrollersystems ausgetauscht. Es existieren Befehle zum Datenaustausch innerhalb des Mikrocontrollers und für den Datenaustausch mit dem externen Speicher. Die Daten werden immer von einer Quelle zu einem Ziel transferiert. Dabei wird nur mit einer Kopie der Quelle gearbeitet. Das heißt, die Quelle wird durch den Transfer nicht verändert. Nach abgeschlossenem Transfer besitzen Ziel und Quelle den gleichen Inhalt. In den Mnemonischen Bezeichnungen der Befehle wird das Ziel vor, die Quelle nach dem Komma angegeben.

Allgemeine Form: MOV ZIEL,QUELLE

Der Transfer innerhalb des Mikrocontrollers wird mit den MOV-Befehlen durchgeführt. Das Kürzel MOV ist dabei von dem englischen Wort „move“ (bewegen) abgeleitet. Durch MOV-Befehle wird das Parity-Flag verändert, sofern das Ziel der Akku ist. Die Flags CY, AC und OV werden durch Transfer-Befehle nicht beeinflußt.

	A	Rr	dadr	#ko.	@Ri
MOV A,	-	*	*	*	*
MOV Rr,	*	-	*	*	-
MOV dadr,	*	*	*	*	*
MOV @Ri,	*	-	*	*	-
MOV DPTR,	-	-	-	*16	-

* = möglich

- = nicht

*16 = 16Bit Konstante

Transfer vom Programmspeicher:

MOVC = MOVe Codememory = Transfer mit Programmspeicher

Da der Programmspeicher nur gelesen werden kann, existieren nur Befehle für die Leserichtung. Die Mnemonic der Befehle beginnt mit „MOVC“ (MOVe Codememory). Sie sind für indirekte Adressierung ausgelegt, wobei der Datenpointer (DPTR) oder der Befehlszähler (PC) als Zeigerregister eingesetzt werden. Zur Adresse im gewählten Zeigerregister wird jeweils noch der

Inhalt des Akku als Offset (Abstand) aufaddiert. Durch diese Art der Adressbildung kann auf einfache Weise auf die Elemente einer Tabelle im Programmspeicher zugegriffen werden. In das Zeigerregister wird die Basisadresse der Tabelle geladen, der Zugriff auf die einzelnen Elemente wird durch den Offset im Akku erreicht. So muß die Basisadresse nur ein einziges mal geladen werden. Das Ziel des Transfers ist immer der Akku. Dadurch wird der Offset im Akku durch den Datenwert überschrieben. Erfolgt der Zugriff auf eine Adresse innerhalb des internen Programmspeichers, wird das interne Bussystem benutzt. Es können alle Ports als Ein/Ausgabeport genutzt werden. Erfolgt der Zugriff auf eine Adresse oberhalb des internen Adressbereichs, wird automatisch das externe Bussystem mit Port P0 (AD0-7), Port P2 (A8-15) und PSEN aktiviert. Dadurch werden Ausgaben auf diesen Ports durch die Bussignale zerstört. In Sytemen ohne externen Speicher muß darauf geachtet werden, nicht versehentlich auf eine Adresse oberhalb des internen Bereichs zuzugreifen. Ist das Signal EA (External Access - externer Zugriff) aktiviert, ist der interne Programmspeicher abgeschaltet. Nun wird bei allen Adressen das externe Bussystem benutzt.

MOVC A,@A+PC — Bringt den Inhalt der Speicherzelle, deren Adresse sich aus dem momentanen Stand des Befehlszeigers plus dem momentanen Inhalt des Akku ergibt, in den Akkumulator.
Zugriff relativ zum Befehlszähler
Kurzbeschreibung: A <- <<A+PC>>

MOVC A,@A+DPTR — Bringt den Inhalt der Speicherzelle, deren Adresse sich aus dem Stand des Datenpointers plus dem momentanen Inhalt des Akku ergibt, in den Akkumulator.
Zugriff indirekt Adressiert über den Datenpointer
Kurzbeschreibung: A <- <<A+DPTR>>

Transfer mit dem externen Datenspeicher:

MOVX = MOVe eXternal datamemory = Transfer mit externem Datenspeicher

Der externe Datenspeicher kann gelesen und beschrieben werden, weshalb Befehle für beide Richtungen bereitstehen. Die Mnemonic der Befehle beginnt mit „MOVX“ (MOVe eXternal datamemory). die Befehle für den externen Datenspeicher arbeiten alle mit indirekter Adressierung. Das Ziel einer Leseoperation und die Quelle einer Schreiboperation ist immer der Akku. Beim externen Datenspeicher muß zwischen einer Erweiterung um 256 Byte (Page 0) oder um mehr als 256 Byte (bis 64KB) unterschieden werden.

Bei der kleinen Speichererweiterung werden für das externe Bussystem nur Port P0 und die Signale RD und WR (P3.7, P3.6) benötigt. Der Port P2 bleibt als Ein/Ausgabeport erhalten. Der kleine Speicherausbau wird über die 8Bit Zeigerregister R0 und R1 der aktiven Registerbank angesprochen.

MOVX A,@Ri Page 0 - Speicherzelle lesen
Kurzbeschreibung: A <- <<Ri>>

MOVX @Ri,A Page 0 - Speicherzelle beschreiben
Kurzbeschreibung: <Ri> <- <A>

Ri = R0 oder R1

Der große Speicherausbau benötigt das vollständige externe Bussystem, welches aus Port P0, P2 und den Bit RD und WR gebildet wird. Als Ein/Ausgabe bleiben nur der Port P3 und sechs Bit des Port P1 übrig. Als Zeigerregister wird der 16Bit breite Datenpointer benutzt.

MOVX A,@DPTR 64KB - Speicherzelle lesen
Kurzbeschreibung: A <- <<DPTR>>

MOVX @DPTR,A 64KB - Speicherzelle schreiben
Kurzbeschreibung: <DPTR> <- <A>

Transfer mit dem Stack:

PUSH = Schieben, POP = Auswerfen

Die Inhalte von direkt adressierbaren Speicherzellen können mit dem Stack Daten austauschen. Beim Transfer zum Stack (PUSH dadr) wird zunächst der Stackpointer um eins erhöht. Anschließend wird das Byte aus der Direktadresse in die Speicherzelle auf die der Stackpointer zeigt kopiert. Beim Transfer vom Stack (POP dadr) wird das Byte, auf das der Stackpointer zeigt, in die angegebene Direktadresse kopiert. Anschließend wird der Stackpointer um eins erniedrigt. Der Stackpointer zeigt immer auf den aktuellen Eintrag.

PUSH dadr Byte im Stack ablegen, Stackpointer + 1

POP dadr Byte aus dem Stack lesen, Stackpointer - 1

Der Datentransfer mit dem Stack wird zum kurzzeitigen Zwischenspeichern von Werten aus direkt adressierbaren Speicherzellen und Registern eingesetzt.

Tauschbefehle:

XCH = eXCHange = Tauschen

Bei den Tauschbefehlen werden die Datenwerte nicht von Quelle nach Ziel kopiert, sondern die Inhalte von Ziel und Quelle werden vertauscht. Nach Ausführung des Befehls befindet sich der Inhalt von Quelle in Ziel und der Inhalt von Ziel in Quelle. Die Befehle mit der Mnemonic „XCH“ (eXCHange - tauschen) vertauschen jeweils ein Byte. Der Befehl „XCHD A,@Ri“ vertauscht die Low-Tetrade des Akku mit der Low-Tetrade (Bit 2^0-2^3) einer indirekt adressierbaren Speicherzelle. Die High-Tetraden (Bit 2^4-2^7) bleiben unverändert. Der Befehl „SWAP A“ vertauscht die beiden Tetraden des Akku.

XCH A,Rr	Tauscht <Akku> mit <Register> als Byte Kurzbeschreibung: <A> <-> <Rr>
XCH A,dadr	Tauscht <Akku> mit <Direktadresse> als Byte Kurzbeschreibung: <A> <-> <dadr>
XCH A,@Ri	Tauscht <Akku> mit <indirektadresse> als Byte Kurzbeschreibung: <A> <-> <@Ri>
XCHD A,@Ri	Tauscht <Akku> Low-Tetrade mit <@Ri> Low-Tetrade Kurzbeschreibung: <A 2^0-2^3> <-> <@Ri 2^0-2^3>
SWAP A	Tauscht <Akku> Low-Tetrade mit <Akku> High-Tetrade Kurzbeschreibung: <A 2^0-2^3> <-> <A 2^4-2^7>

5.1.2 Arithmetische Befehle

Die arithmetischen Befehle werden in der ALU des Mikrocontrollers ausgeführt. Bei Operationen mit zwei Operanden ist immer der Akku beteiligt. Er stellt einen der Operanden und nimmt das Ergebnis auf. Die arithmetischen Operationen verändern die Flags der ALU und, sofern der Akku das Ziel ist, das Parity-Flag. Es stehen folgende Befehle bereit:

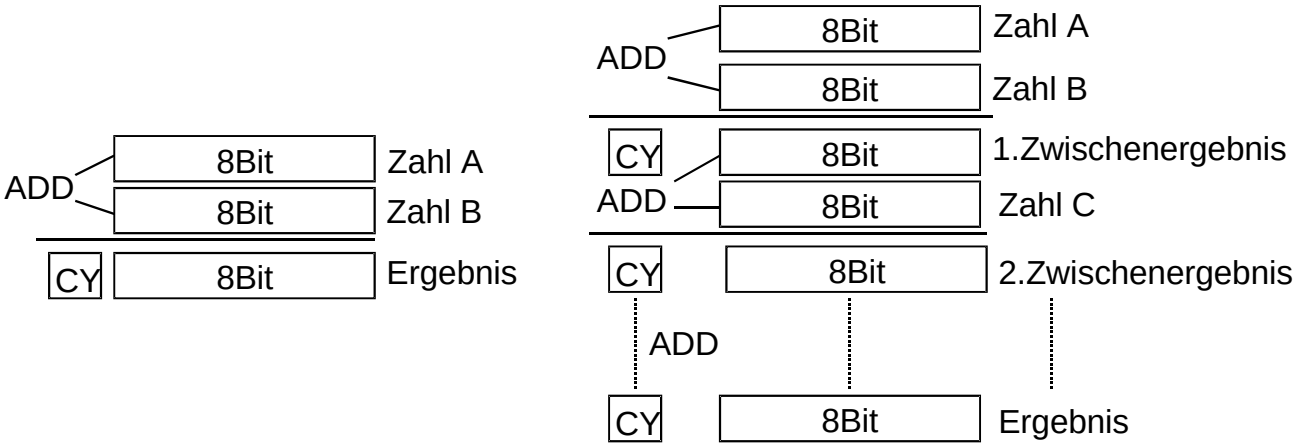
ADD	- Addition
ADDC	- Addition mit Carry
SUBB	- Subtraktion mit Carry
INC	- Inkrementieren
DEC	- Dekrementieren
MUL AB	- Multiplikation
DIV AB	- Division
DAA	- Dezmalkorrektur

Der Additionsbefehl ADD

ADD = ADDition = Addition

ADD A,	Rr	dadr	#ko.	@Ri	<A> + Quelle -> A

Der einfache Additionsbefehl ADD wird mit dem Akku ausgeführt. Im Befehl wird zum Inhalt des Akku ein 8Bit-Quelloperand binär aufaddiert. Entsteht dabei ein Übertrag (9.Bit mit Wertigkeit 256), ist dieser im CY-Flag zu finden. Der Operand, der sich vor Befehlsausführung im Akku befand geht verloren, er wird durch das Ergebnis überschrieben. Als Quelloperand kann der Inhalt eines Registers (Rr), einer direkt adressierbaren Speicherzelle (dadr), einer indirekt adressierbaren Speicherzelle (@Ri) oder eine 8Bit Konstante (#ko) eingesetzt werden. Der Befehl verändert das CY-, AC-, OV- und das P-Flag aufgrund des Ergebnis. Der Befehl ADD ist geeignet zwei 8Bit breite Operanden zu einem maximal 9Bit breiten Ergebnis zu addieren. Sollen mehrere 8Bit breite Operanden addiert werden, so müssen diese der Reihe nach verarbeitet werden. Dabei müssen die jeweils entstehenden Überträge durch Software berücksichtigt werden.

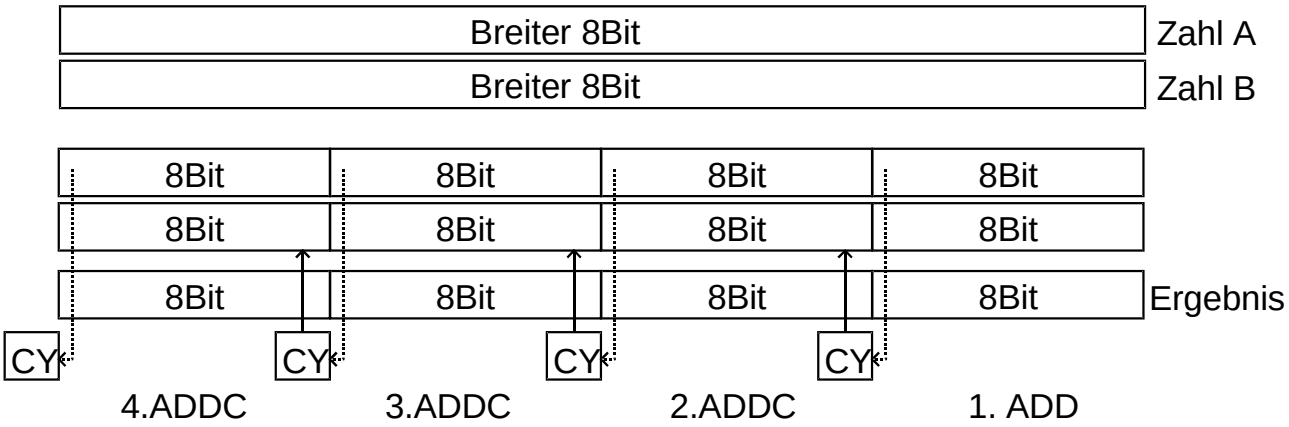


Der Additionsbefehl ADDC

ADDC = ADDition with Carry = Addition mit Übertrag

ADDC A,	Rr	dadr	#ko.	@Ri	<A> + Quelle + CY -> A

Der erweiterte Additionsbefehl ADDC wird mit dem Akku ausgeführt. Im Befehl wird zum Inhalt des Akku ein 8Bit-Quelloperand binär Aufaddiert, zusätzlich wird im niederwertigsten Bit der Zustand des CY-Flag dazuaddiert. Entsteht dabei ein Übertrag (9.Bit mit Wertigkeit 256), ist dieser im CY-Flag zu finden. Der Operand, der sich vor Befehlsausführung im Akku befand, geht verloren. Er wird durch das Ergebnis überschrieben. Als Quelloperand kann der Inhalt eines Registers (Rr), einer direkt adressierbaren Speicherzelle (dadr), einer indirekt adressierbaren Speicherzelle (@Ri) oder eine 8Bit Konstante (#ko) eingesetzt werden. Der Befehl verändert das CY-, AC-, OV- und das P-Flag aufgrund des Ergebnises. Der Befehl ADDC ist geeignet beliebig breite Operanden zu einem beliebig breiten Ergebnis zu addieren. Die Addition dieser breiten Zahlen wird in mehrere 8Bit-Additionen zerlegt, die der Reihe nach ausgeführt werden. Zur Addition des niederwertigsten Byte einer Zahl wird der Befehl ADD verwendet. Für alle Folgeadditionen findet der Befehl ADDC Verwendung, da dieser einen Übertrag der vorhergehenden Addition berücksichtigt.

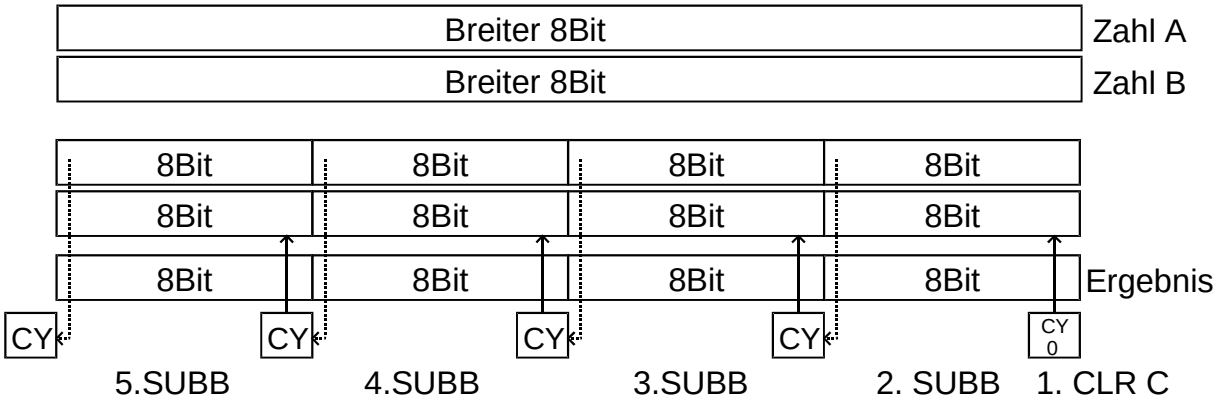


Der Subtraktionsbefehl SUBB

SUBB = SUBtract with Borrow = Subtrahieren mit Borgen

SUBB A,	Rr	dadr	#ko.	@Ri	<A> - Quelle - CY -> A

Der Subtraktionsbefehl wird immer mit dem Akku unter Berücksichtigung des CY-Flags ausgeführt. Im Befehl wird der 8Bit-Quelloperand vom Inhalt des Akku subtrahiert, zusätzlich wird im niederwertigsten Bit der Operanden der Zustand des CY-Flag abgezogen. Entsteht dabei im höchsten Bit ein Borgen (9.Bit mit Wertigkeit 256), wird das CY-Flag gesetzt. Der Operand, der sich vor Befehlsausführung im Akku befand geht verloren, er wird durch das Ergebnis überschrieben. Als Quelloperand kann der Inhalt eines Registers (Rr), einer direkt adressierbaren Speicherzelle (dadr), einer indirekt adressierbaren Speicherzelle (@Ri) oder eine 8Bit Konstante (#ko) eingesetzt werden. Der Befehl verändert das CY-, AC-, OV- und das P-Flag aufgrund des Ergebnis. Der Befehl SUBB ist geeignet beliebig breite Operanden zu einem beliebig breiten Ergebnis zu subtrahieren. Die Subtraktion dieser breiten Zahlen wird in mehrere 8Bit-Subtraktionen zerlegt, die der Reihe nach ausgeführt werden. Zur Subtraktion des niederwertigsten Byte einer Zahl ist vor dem Befehl SUBB das CY-Flag zu löschen (CLR C). Für alle Folgesubtraktionen darf das CY-Flag nicht gelöscht werden, da ein Borgen der vorhergehenden Subtraktion berücksichtigt werden muß. Sollen mehrere 8Bit breite Operanden Subtrahiert werden, ist vor jeder Subtraktion das CY-Flag zu löschen.



Der Inkrementbefehl INC

INC = INCrement = Inkrementieren

INC	A	Rr	dadr	@Ri	DPTR	<X> + 1 -> X

Der Inkrementbefehl benötigt zur Ausführung nur einen Operanden. Dieser Operand wird durch den Befehl um eins erhöht. Als Operand kann der Inhalt des Akku (A), eines Register (Rr), einer direkt adressierbaren Speicherzelle (dadr), einer indirekt adressierbaren Speicherzelle (@Ri) oder der Datenpointer (DPTR) eingesetzt werden. Ist der Operand der Datenpointer, wird das hochzählen über alle 16Bit durchgeführt. Bei allen anderen Operanden wird mit einer Breite von

8Bit hochgezählt. Ist der höchstmögliche Zahlenwert erreicht, so wird beim nächsten Inkrementieren der Operand zu Null (FFh + 1 = 00h, FFFFh + 1 = 0000h). Ist das Ziel der Operation der Akku, wird das Parity-Flag verändert. Sonst wird kein Flag geändert.

Der Dekrementbefehl DEC

DEC = DECrement = Dekrement

DEC	A	Rr	dadr	@Ri	<X> - 1 -> X

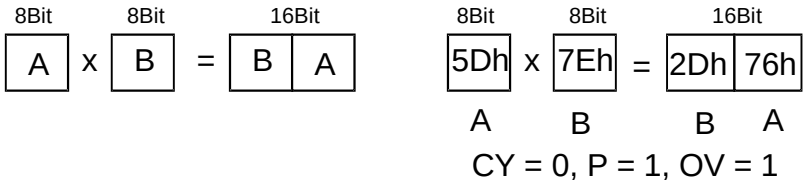
Der Dekrementbefehl benötigt zur Ausführung nur einen Operanden. Dieser Operand wird durch den Befehl um eins erniedrigt. Als Operand kann der Inhalt des Akku (A), eines Register (Rr), einer direkt adressierbaren Speicherzelle (dadr) oder einer indirekt adressierbaren Speicherzelle (@Ri) eingesetzt werden. Der Datenpointer (DPTR) ist als Operand **nicht** erlaubt. Bei allen anderen Operanden wird mit einer Breite von 8Bit abwärts gezählt. Ist der Zahlenwert Null erreicht, so erscheint beim nächsten Dekrementieren der höchstmögliche Zahlenwert (00h - 1 = FFh). Ist das Ziel der Operation der Akku, wird das Parity-Flag verändert. Sonst wird kein Flag geändert.

Der Multiplikationsbefehl MUL AB

MUL AB = MULtiply A & B = Multiplikation A und B

MUL AB	<A> x <B> -> A (LSB) und B (MSB)

Die vorzeichenlosen 8Bit-Inhalte des Akku (A) und des Hilfsakku (B) werden zu einem vorzeichenlosen 16Bit Ergebnis multipliziert. Das niederwertige Byte (LSB) des Ergebnis befindet sich nach MUL AB im Akku, das höherwertige (MSB) im Hilfsakku (B). Das CY-Flag wird gelöscht, das P-Flag durch den Akkuinhalt geändert. Entsteht ein Ergebnis größer 255 (<B> ungleich 00h), wird das OV-Flag gesetzt.

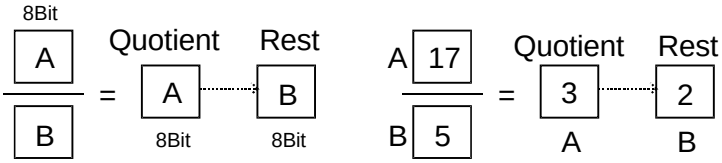


Der Divisionsbefehl DIV AB

DIV = Divide A by B = Division A durch B

DIV AB	<A> / <B> -> A (Rest in B)

Der vorzeichenlose 8Bit-Inhalt des Akku wird ganzzahlig (ohne Kommastellen) durch den vorzeichenlosen 8Bit-Inhalt des Hilfsakku (B) geteilt. Der Quotient (ganzzahliges Ergebnis) wird im Akku, der Divisionsrest in B abgelegt. Das CY-Flag wird gelöscht, das P-Flag durch den Akkuinhalt geändert. Bei einer Teilung durch Null wird das OV-Flag gesetzt, die Register A und B besitzen dann unbestimmte Inhalte



Der Dezimalkorrekturbefehl DA A

DA A = Decimal Adjust Accumulator = Dezimal Korrektur Akkumulator

DA A	Dezimalkorrektur des <A> nach Addition	Flags: Cy,AC,P

Der Befehl DA A dient der Korrektur des Ergebnis im Akku nach der Addition von BCD-Zahlen. Da eine Addition immer in binärer Form erfolgt, kann nach der Addition von Binärzahlen im Ergebnis der Bereich der Pseudotetraden (A-Fh) enthalten sein. Dies ist erkenntlich an einem Tetradenübertrag (AC, CY) oder einer Pseudotetrade im Ergebnis. Der DA A-Befehl ist in der Lage dies selbständig zu erkennen und das Ergebnis Tetradenweise zu korrigieren. Er beginnt mit der Low-Tetrade und prüft auf Tetradenübertrag (AC=1) oder Pseudotetrade (A-Fh). wird einer der Zustande erkannt, wird die Zahl 06h auf das Ergebnis addiert. Anschließend wird die High-Tetrade auf Tetradenübertrag (CY) oder eine Pseudotetrade geprüft. Wird einer der Zustände erkannt, wird die Zahl 60h auf das Ergebnis addiert. Nach dieser Korrektur entspricht das Ergebnis im Akku wieder einer BCD-Zahl. Entsteht bei der Korrektur ein Übertrag (CY = 1), entspricht dies der Stellenwertigkeit 100er. Der DA A-Befehl wird unmittelbar nach dem Additionsbefehl eingesetzt. Wird der DA A-Befehl nach einer Addition von Binär- oder Hex-Zahlen eingesetzt, verfälscht er das Ergebnis.

Beispiel:

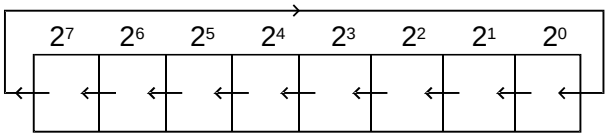
Dezimal	BCD
78 Zahl A	0 1 1 1 1 0 0 0 = 78 = BCD-Zahl A
+ 49 Zahl B	+ $0_1 1_1 0_1 0_1$ 1 0 0 1 = 49 = BCD-Zahl B
127 Ergebnis	1 1 0 0 0 0 0 1 = C1 = 1. Zwischenergebnis
	+ 0 0 0 0 0 1 1 0 = 06 = 1. Korrektur
	1 1 0 0 0 1 1 1 = C7 = 2. Zwischenergebnis
Programm:	+ $1_1 0_1$ 1 1 0 0 0 0 0 = 60 = 2. Korrektur
MOV A,#79h	1 0 0 1 0 0 1 1 1 = 127 = BCD-Ergebnis
MOV R0,#48h	
ADD A,R0	
DA A	

Die BCD-Zahlen 78 und 49 werden mit dem ADD-Befehl zusammenaddiert. Es entsteht zunächst das 1.Zwischenergebnis C1h. Der DA A-Befehl prüft nun die Low-Tetrade und erkennt einen Tetradenübertrag (AC gesetzt). Daraufhin wird zum Ergebnis die Zahl 06 addiert, womit die Low-Tetrade korrigiert ist. Nun wird die daraus entstandene High-Tetrade geprüft. Es wird eine Pseudotetrade erkannt (Cxh) und daraufhin die Zahl 60h zum Ergebnis addiert. Das Endergebnis im Akku, einschließlich Carry, ist die gewünschte BCD-Zahl.

Die Rotationsbefehle werden in der ALU des Mikrocontrollers ausgeführt. Der Operand steht dabei immer im Akku. Bei den Befehlen, die das CY-Flag in die Rotation einbeziehen wird dieses und das P-Flag beeinflußt. Bei den Befehlen ohne CY-Flag bleibt die Anzahl der "1" im Akku immer die gleiche, wodurch sich die Parität nicht ändert.

Der Rotationsbefehl RL A

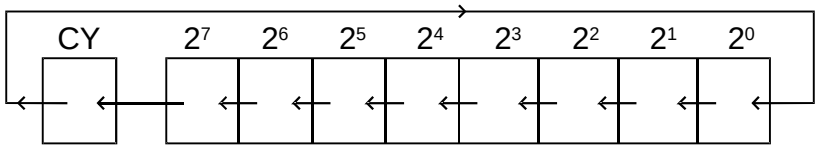
RL A = Rotate Left Accu = Rotiere links Akku



Jedes einzelne Bit des Akku wird um eine Bitposition nach links verschoben. Bit 2^0 nach Position 2^1, Bit 2^1 nach Position 2^2 usw. Das Bit 2^7 gelangt in Position 2^0. Es wird kein Flag verändert.

Der Rotationsbefehl RLC A

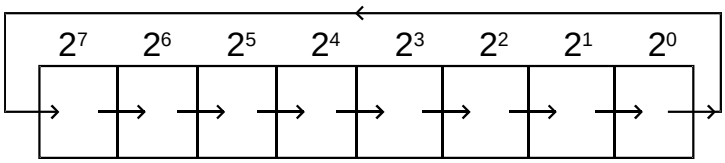
RLC A = Rotate Left throug Carry Accu = Rotiere links durch Carry Akku



Jedes einzelne Bit des Akku wird um eine Bitposition nach links verschoben, das CY-Flag ist in die Rotation einbezogen. Bit 2^0 nach Position 2^1, Bit 2^1 nach Position 2^2 usw. Das Bit 2^7 gelangt nach Carry. Das CY-Flag gelangt in Position 2^0. Es werden das CY- und das P-Flag verändert. Dieser Befehl ist zum Multiplizieren eines Operanden mal zwei geeignet, da jedes Bit in die doppelte Wertigkeit verschoben wird. Vor der Multiplikation ist das CY-Flag zu löschen, damit von unten eine "0" nachgeschoben wird. Das höchste Bit des Ergebnis der Multiplikation befindet sich im CY-Flag.

Der Rotationsbefehl RR A

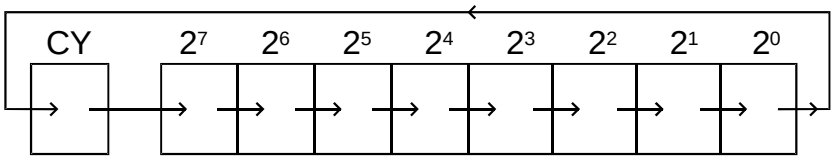
RR A = Rotate Right Accu = Rotiere rechts Akku



Jedes einzelne Bit des Akku wird um eine Bitposition nach rechts verschoben. Bit 2^7 nach Position 2^6, Bit 2^6 nach Position 2^5 usw. Das Bit 2^0 gelangt in Position 2^7. Es wird kein Flag verändert.

Der Rotationsbefehl RRC A

RLC A = Rotate Right throug Carry Accu = Rotiere rechts durch Carry Akku



Jedes einzelne Bit des Akku wird um eine Bitposition nach rechts verschoben, das CY-Flag ist in die Rotation einbezogen. Bit 2^7 nach Position 2^6, Bit 2^6 nach Position 2^5 usw. Das Bit 2^0 gelangt nach Carry. Das CY-Flag gelangt in Position 2^7. Es werden das CY- und das P-Flag verändert. Dieser Befehl ist zum Dividieren eines Operanden durch zwei geeignet, da jedes Bit in die halbe Wertigkeit verschoben wird. Vor der Division ist das CY-Flag zu löschen, damit von oben eine "0" nachgeschoben wird. Das niederwertigste Bit (2^{-1}, Wertigkeit 0,5) des Ergebnises der Division befindet sich im CY-Flag.

5.1.4 Logische Befehle

Die logischen Befehle werden in der ALU des Mikrocontrollers durchgeführt. Ziel der Operationen kann der Akku oder eine direkt adressierbare Speicherzelle sein. Ist der Akku das Ziel, wird das P-Flag beeinflußt. Die restlichen Flags werden durch die logischen Verknüpfungen nicht berührt. Es stehen Befehle für die UND-, die ODER-, die EXCLUSIV-ODER und die NICHT-Verknüpfung bereit:

ANL	- UND-Verknüpfung
ORL	- ODER-Verknüpfung
XRL	- EXCLUSIV-ODER-Verknüpfung
CPL A	- NICHT-Verknüpfung

Der Verknüpfungsbefehl ANL

ANL = ANd Logical = UND Logisch

ANL A,	Rr	dadr	#ko.	@Ri	<A> UND X -> A
ANL dadr,	A		#ko.		<dadr> UND X -> dadr

Der Zieloperand wird mit dem Quelloperanden bitweise UND-Verknüpft. Der Zieloperand geht verloren, er wird durch das Ergebnis überschrieben. Durch die UND-Verknüpfung können ein- oder mehrere Bit des Zieloperanden schnell in den Zustand "0" gebracht werden. Da der Befehl auch mit direkt adressierbaren Speicherzellen (dadr) als Ziel funktioniert, können damit die SFR und Portbit schnell bearbeitet werden. Ist das Ziel der Akku, wird das P-Flag beeinflußt. Sonst werden keine Flags geändert.

Der Verknüpfungsbefehl ORL

ORL = OR Logical = ODER Logisch

ORL A,	Rr	dadr	#ko.	@Ri	<A> ODER X -> A
ORL dadr,	A		#ko.		<dadr> ODER X -> dadr

Der Zieloperand wird mit dem Quelloperanden bitweise ODER-Verknüpft. Der Zieloperand geht verloren, er wird durch das Ergebnis überschrieben. Durch die ODER-Verknüpfung können ein- oder mehrere Bit des Zieloperanden schnell in den Zustand "1" gebracht werden. Da der Befehl auch mit direkt adressierbaren Speicherzellen (dadr) als Ziel funktioniert, können damit die SFR und Portbit schnell bearbeitet werden. Ist das Ziel der Akku, wird das P-Flag beeinflußt. Sonst werden keine Flags geändert.

Der Verknüpfungsbefehl XRL

XRL = eXclusiv oR Logical = EXCLUSIV-ODER Logisch

XRL A,	Rr	dadr	#ko.	@Ri	<A> EX-ODER X -> A
XRL dadr,	A		#ko.		<dadr> EX-ODER X -> dadr

Der Zieloperand wird mit dem Quelloperanden bitweise EXCLUSIV-ODER-Verknüpft. Der Zieloperand geht verloren, er wird durch das Ergebnis überschrieben. Durch die EXCLUSIV-ODER-Verknüpfung können ein- oder mehrere Bit des Zieloperanden schnell invertiert werden. Da der Befehl auch mit direkt adressierbaren Speicherzellen (dadr) als Ziel funktioniert, können damit die SFR und Portbit schnell bearbeitet werden. Ist das Ziel der Akku, wird das P-Flag beeinflußt. Sonst werden keine Flags geändert.

Der Verknüpfungsbefehl CPL A

CPL A = ComPlement Accu = Komplementiere Akku

CPL A	<A> NICHT -> A

Der Inhalt des Akku wird gesamt invertiert. Jede "1" wird zur "0" und umgekehrt. Da sich dadurch die Anzahl der "1" nicht ändert, bleibt das P-Flag unverändert.

5.1.5 Einzelbit Befehle

Die Befehle die auf ein einzelnes Bit wirken, werden mit dem CY-Flag oder einem der bitadressierbaren Bit durchgeführt. Bei Bitoperationen mit zwei Bit arbeitet das CY-Flag als Bitakku. Es stehen folgende Operationen bereit:

MOV	- Bit Transfer
CLR	- Bit löschen
SETB	- Bit setzen
CPL	- Bit invertieren
ANL	- UND-Verknüpfen
ORL	- ODER-Verknüpfen

Es ist kein Befehl zur EXCLUSIV-ODER-Verknüpfung verfügbar, da zum invertieren von Einzelbit der Befehl CPL geeignet ist.

Der Bitbefehl MOV

MOV = MOVe = Bewegen

MOV C,badr	<badr> -> CY
MOV badr,C	<CY> -> badr

Bei den Bittransferbefehlen wird der Zustand des Ouellbit ins Zielbit kopiert. Sofern ein Flag-Bit das Zielbit ist, wird sein Zustand beeinflußt. Bei allen anderen Transfers wird kein Flag verändert.

Der Bitbefehl CLR

CLR = CLeaR = Löschen

CLR C	0 -> CY
CLR badr	0 -> badr

Durch den CLR-Befehl wird das angegebene Bit in den Zustand "0" gebracht (gelöscht). Befindet sich das Bit im Akku, wird das P-Flag beeinflußt. Sofern nicht der Akku und kein Flag-Bit das Ziel ist, wird kein Flag geändert.

Der Bitbefehl SETB

SETB = SET Bit = Bit setzen

SETB C	0 -> CY
SETB badr	0 -> badr

Durch den SETB-Befehl wird das angegebene Bit in den Zustand "1" gebracht (gesetzt). Befindet sich das Bit im Akku, wird das P-Flag beeinflußt. Sofern nicht der Akku und kein Flag-Bit das Ziel ist, wird kein Flag geändert.

Der Bitbefehl CPL

CPL = ComPLement = Komplementieren

CPL C	0 -> CY
CPL badr	0 -> badr

Durch den CPL-Befehl wird das angegebene Bit invertiert. Befindet sich das Bit im Akku, wird das P-Flag beeinflußt. Sofern nicht der Akku und kein Flag-Bit das Ziel ist, wird kein Flag geändert.

Der Bitbefehl ANL

ANL = ANd Logical = UND Logisch

ANL C,badr	<CY> UND <badr> -> CY
ANL C,/badr	<CY> UND <badr> -> CY

Durch den Befehl ANL C,badr wird das angegebenen Bit mit dem CY-Flag UND-Verknüpft, das Ergebnis wird in CY abgelegt. Durch den Befehl ANL C,/badr wird der invertierte Zustand des angegebenen Bit mit dem CY-Flag UND-Verknüpft, das Ergebnis wird in CY abgelegt. Dadurch wird der Zustand des CY-Flag beeinflußt.

Der Bitbefehl ORL

ORL = OR Logical = UND Logisch

ORL C,badr	<CY> UND <badr> -> CY
ORL C,/badr	<CY> UND <badr> -> CY

Durch den Befehl ORL C,badr wird das angegebenen Bit mit dem CY-Flag ODER-Verknüpft, das Ergebnis wird in CY abgelegt. Durch den Befehl ORL C,/badr wird der invertierte Zustand des angegebenen Bit mit dem CY-Flag ODER-Verknüpft, das Ergebnis wird in CY abgelegt. Dadurch wird der Zustand des CY-Flag beeinflußt.

5.1.6 Sprungbefehle

Sprungbefehle verändern immer den Stand des Befehlszeigers (PC). Der Befehlszeiger zeigt immer auf den nächsten auszuführenden Befehl. Durch einen Sprungbefehl wird als nächster der Befehl ausgeführt, auf den der Befehlszeiger jetzt zeigt. Im Befehlssatz sind bedingte und unbedingte Sprungbefehle enthalten. Die unbedingten Sprünge werden in jedem Fall ausgeführt. Die bedingten Sprünge werden nur ausgeführt, wenn ihre Bedingung erfüllt ist. Andernfalls wird mit dem Folgebefehl weitergearbeitet. Die Sprungdistanzen und Befehlslängen sind unterschiedlich. Es gibt LONG-Jumps die als dreibyte Befehle jede Adresse innerhalb von 64kB anspringen können. Da sie aus drei Befehlsbyte bestehen benötigen sie zwei Maschinenzyklen zur Ausführung. Der AJMP-Befehl ist ein zweibyte Befehl. Er kann innerhalb eines 2kB großen Adressbereichs springen. Die Sprungbefehle mit einer relativen Sprungdistanz können bis zu 127 Adressen vorwärts und bis zu 128 Adressen rückwärts relativ zur Adresse des Folgebefehls springen. Sie benötigen, unabhängig von der Befehlslänge, zwei Maschinenzyklen. Die Sprungdistanz (+127 bis -128) ist im Befehlscode als vorzeichenbehaftete 8Bit-Zahl enthalten.

Unbedingte Sprünge

Der Sprungbefehl LJMP

LJMP = Long JuMP = Weiter Sprung

LJMP adr16	adr16 -> PC

Der Sprungbefehl LJMP ist ein unbedingter Sprung mit 16Bit Adressangabe. Das Sprungziel darf innerhalb des gesamten 64kByte großen Programmspeichers liegen. Der LJMP-Befehl ist drei Byte lang und benötigt 2 Maschinenzyklen.

Der Sprungbefehl SJMP

SJMP = Short JuMP = Kurzer Sprung

SJMP rel	PC ± rel. -> PC

Der Sprungbefehl SJMP ist ein unbedingter Sprung mit relativer Adressangabe. Das Sprungziel darf maximal 127 Adressen vorwärts und 128 Adressen rückwärts entfernt liegen. Die Bezugsadresse ist die des Folgebefehls. Der SJMP-Befehl ist zwei Byte lang und benötigt zwei Maschinenzyklen.

Der Sprungbefehl AJMP

AJMP = Absolute JuMP = Absoluter Sprung

AJMP adr11	adr11 -> PC 2^0-2^{10}

Der Sprungbefehl AJMP ist ein unbedingter Sprung mit absoluter Adressangabe. Das Sprungziel muß im aktuellen 2kByte Block liegen. Der AJMP-Befehl besteht aus zwei Byte. Im OP-Code sind die höchsten drei Bit für die Bit 2^{10} bis 2^8 (A8-A10) der Zieladresse reserviert. Das zweite Befehlsbyte beinhaltet die unteren 8Bit (A0-A7) der Zieladresse. Zur Befehlsausführung werden die unteren elf Bit des Befehlszeigers durch die 11Bit Adresse ersetzt. Die oberen fünf Bit des Befehlszeigers bleiben unverändert. An der oberen Grenze eines Blocks (ab xxxx x111 1111 1110, xxFEh) darf der Befehl nicht eingesetzt werden, da durch die Befehlsausführung, beim Einstellen der Folgeadresse, die oberen fünf Bit des Befehlszählers um eins erhöht werden. Dadurch führt der Sprung in den folgenden Block.

Adresse			OP-Code					
A10	A9	A8	0	0	0	0	1	1.Byte
A7	A6	A5	A4	A3	A2	A1	A0	2.Byte
Adresse								

Der Sprungbefehl JMP @A+DPTR

JMP = JuMP = Sprung

JMP @A+DPTR	<A> + <DPTR> -> PC

Der Sprungbefehl JMP @A+DPTR ist ein unbedingter Sprung mit 16Bit Adressangabe. Das Sprungziel darf innerhalb des gesamten 64kByte großen Programmspeichers liegen. Die Zieladresse ist der Inhalt des Datenpointer (DPTR), auf den der momentane Inhalt des Akku addiert wird. Mit diesem Befehl ist es möglich, das Sprungziel zu berechnen. Der AJMP-Befehl ist zwei Byte lang und benötigt zwei Maschinenzyklen.

Bedingte Sprünge

Der Sprungbefehl JZ

JZ = Jump if Zero = Sprung wenn Null

JZ rel	wenn <A> = 0, <PC+2> ± rel. -> PC wenn <A> <> 0, <PC+2> -> PC

Der Sprungbefehl JZ ist ein bedingter Sprung mit relativer Adressangabe. Das Sprungziel darf maximal 127 Adressen vorwärts und 128 Adressen rückwärts entfernt liegen. Die Bezugsadresse ist die des Folgebefehls. Der Sprung wird nur ausgeführt, wenn zur Ausführungszeit der Inhalt des Akku 00h ist. Ist der Inhalt des Akku zur Befehlsausführung ungleich 00h, wird mit dem Folgebefehl weitergearbeitet. Der JZ-Befehl ist zwei Byte lang und benötigt zwei Maschinenzyklen.

Der Sprungbefehl JNZ

JNZ = Jump if No Zero = Sprung wenn nicht Null

JNZ rel	wenn <A> <> 0, <PC+2> ± rel. -> PC wenn <A> = 0, <PC+2> -> PC

Der Sprungbefehl JNZ ist ein bedingter Sprung mit relativer Adressangabe. Das Sprungziel darf maximal 127 Adressen vorwärts und 128 Adressen rückwärts entfernt liegen. Die Bezugsadresse ist die des Folgebefehls. Der Sprung wird nur ausgeführt, wenn zur Ausführungszeit der Inhalt des Akku ungleich 00h ist. Ist der Inhalt des Akku zur Befehlsausführung gleich 00h, wird mit dem Folgebefehl weitergearbeitet. Der JNZ-Befehl ist zwei Byte lang und benötigt zwei Maschinenzyklen.

Der Sprungbefehl JC

JC = Jump if Carry = Sprung wenn Carry

JC rel	wenn CY = 1, <PC+2> ± rel. -> PC wenn CY = 0, <PC+2> -> PC

Der Sprungbefehl JC ist ein bedingter Sprung mit relativer Adressangabe. Das Sprungziel darf maximal 127 Adressen vorwärts und 128 Adressen rückwärts entfernt liegen. Die Bezugsadresse ist die des Folgebefehls. Der Sprung wird nur ausgeführt, wenn zur Ausführungszeit das CY-Flag gesetzt ist. Ist das CY-Flag zur Befehlsausführung nicht gesetzt, wird mit dem Folgebefehl weitergearbeitet. Der JC-Befehl ist zwei Byte lang und benötigt zwei Maschinenzyklen.

Der Sprungbefehl JNC

JNC = Jump if No Carry = Sprung wenn nicht Carry

JNC rel	wenn CY = 0, <PC+2> ± rel. -> PC wenn CY = 1, <PC+2> -> PC

Der Sprungbefehl JNC ist ein bedingter Sprung mit relativer Adressangabe. Das Sprungziel darf maximal 127 Adressen vorwärts und 128 Adressen rückwärts entfernt liegen. Die Bezugsadresse ist die des Folgebefehls. Der Sprung wird nur ausgeführt, wenn zur Ausführungszeit das CY-Flag nicht gesetzt ist. Ist das CY-Flag zur Befehlsausführung gesetzt, wird mit dem Folgebefehl weitergearbeitet. Der JNC-Befehl ist zwei Byte lang und benötigt zwei Maschinenzyklen.

Der Sprungbefehl JB

JB = Jump if Bit = Sprung wenn Bit

JB rel	wenn <badr> = 1, <PC+2> ± rel. -> PC wenn <badr> = 0, <PC+2> -> PC

Der Sprungbefehl JB ist ein bedingter Sprung mit relativer Adressangabe. Das Sprungziel darf maximal 127 Adressen vorwärts und 128 Adressen rückwärts entfernt liegen. Die Bezugsadresse ist die des Folgebefehls. Der Sprung wird nur ausgeführt, wenn zur Ausführungszeit das adressierte Bit gesetzt ist. Ist das Bit zur Befehlsausführung nicht gesetzt, wird mit dem Folgebefehl weitergearbeitet. Der JB-Befehl ist zwei Byte lang und benötigt zwei Maschinenzyklen.

Der Sprungbefehl JNB

JNB = Jump if No Bit = Sprung wenn nicht Bit

JNB rel	wenn <badr> = 0, <PC+2> ± rel. -> PC wenn <badr> = 1, <PC+2> -> PC

Der Sprungbefehl JNB ist ein bedingter Sprung mit relativer Adressangabe. Das Sprungziel darf maximal 127 Adressen vorwärts und 128 Adressen rückwärts entfernt liegen. Die Bezugsadresse ist die des Folgebefehls. Der Sprung wird nur ausgeführt, wenn zur Ausführungszeit das adressierte Bit nicht gesetzt ist. Ist das Bit zur Befehlsausführung gesetzt, wird mit dem Folgebefehl weitergearbeitet. Der JNB-Befehl ist zwei Byte lang und benötigt zwei Maschinenzyklen.

Der Sprungbefehl JBC

JBC = Jump if Bit and Clear = Sprung wenn Bit und lösche

JBC rel	wenn <badr> = 1, 0 -> badr, <PC+2> ± rel. -> PC wenn <badr> = 0, <PC+2> -> PC

Der Sprungbefehl JBC ist ein bedingter Sprung mit relativer Adressangabe. Das Sprungziel darf maximal 127 Adressen vorwärts und 128 Adressen rückwärts entfernt liegen. Die Bezugsadresse ist die des Folgebefehls. Der Sprung wird nur ausgeführt, wenn zur Ausführungszeit das adressierte Bit gesetzt ist. Das Bit wird durch die Befehlsausführung gelöscht. Ist das Bit zur Befehlsausführung nicht gesetzt, wird mit dem Folgebefehl weitergearbeitet. Der JBC-Befehl ist zwei Byte lang und benötigt zwei Maschinenzyklen.

Der Sprungbefehl CJNE

CJNE = Compare and Jump if Not Equal = Vergleiche und springe wenn ungleich

CJNE A,dadr,rel CJNE A,#ko,rel CJNE Rr,#ko,rel CJNE @Ri,#ko,rel	Falls ungleich: <PC+2> ± rel -> PC Falls gleich: <PC+2> -> PC

Der Sprungbefehl CJNE ist ein bedingter Sprung mit relativer Adressangabe. Das Sprungziel darf maximal 127 Adressen vorwärts und 128 Adressen rückwärts entfernt liegen. Die Bezugsadresse ist die des Folgebefehls. Der Sprung wird nur ausgeführt, wenn zur Ausführungszeit die Inhalte der beiden Operanden ungleich sind. Sind die Inhalte der Operanden zur Befehlsausführung gleich, wird mit dem Folgebefehl weitergearbeitet. Der Vergleich wird mit einer Subtraktion durchgeführt. Vom ersten Operanden wird der zweite Operand abgezogen, das Ergebnis wird verworfen. Die beiden Operanden besitzen nach dem Befehl den gleichen Inhalt wie vorher. Das CY-Flag wird gemäß der Subtraktion beeinflußt, wodurch es nach einem CJNE-Befehl möglich ist dieses auszuwerten. Der CJNE-Befehl ist drei Byte lang und benötigt zwei Maschinenzyklen.

Allgemeine Form: CJNE Op1,Op2,rel Subtraktion: Op1 - Op2 -> kein Ergebnis

Mit Hilfe eines Vergleichs kann das Verhältnis der Operanden zueinander festgestellt werden.

Op1 > Op2	Ergebnis nicht Null	CY = 0
Op1 = Op2	Ergebnis Null	CY = 0
Op1 < Op2	Ergebnis nicht Null	CY = 1

Ist das CY-Flag nach einem CJNE-Befehl gesetzt, ist Operand 1 kleiner als Operand 2. Ist das CY-Flag nicht gesetzt, ist Operand 1 größer/gleich Operand 2. Wird der Sprung ausgeführt sind die beiden Operanden ungleich. Wird der Sprung nicht ausgeführt sind die beiden Operanden gleich.

Der Sprungefehl DJNZ

DJNZ = Decrement and Jump if Not Zero = Dekrementiere und springe wenn nicht Null

DJNZ Rr,rel	<Rr>-1 <> 0, <PC+2> ± rel -> PC / <Rr>-1 = 0, <PC+2> -> PC
DJNZ dadr,rel	<dadr>-1 <> 0, <PC+2> ± rel -> PC / <dadr>-1 = 0, <PC+2> -> PC

Der Sprungbefehl DJNZ ist ein bedingter Sprung mit relativer Adressangabe. Das Sprungziel darf maximal 127 Adressen vorwärts und 128 Adressen rückwärts entfernt liegen. Die Bezugsadresse ist die des Folgebefehls. Der angegebene Operand wird um eins erniedrigt und falls er noch nicht 00h ist wird der Sprung ausgeführt. Der DJNZ-Befehl kann als Schleifenzähler Einsatz finden. Die Anzahl der Durchläufe entspricht dem Ladewert des Operanden.

5.1.7 Sonderbefehle

Der Sonderbefehl NOP

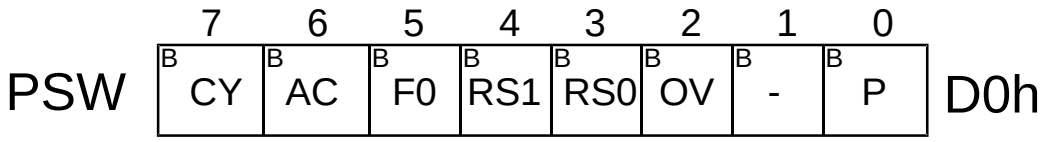
NOP = No OPeration = Nichts ausführen

NOP	<PC+1> -> PC

Der NOP-Befehl zeigt innerhalb eines Programms keine Wirkung auf die Komponenten des Mikrocontrollers. Er wird eingesetzt um an bestimmten Positionen im Programm Freiräume zu schaffen, an die später wirksamer Code eingesetzt werden kann. Jeder NOP-Befehl belegt ein Byte im Programmspeicher und dauert einen Maschinenzyklus.

5.2 Die Softwarebezogenen SFR

5.2.1 Programm-Status-Word (PSW)



Das PSW ist ein Software bezogenes SFR. Es befindet sich in der Adresse D0h im SFR-Bereich. Die einzelnen Bit innerhalb des PSW besitzen jeweils eine eigene Bitadresse.

CY = Carry = Übertrag

Übertrag von Ergebnisbit 2^7 nach 2^8.
Fand der Übertrag statt, ist CY gesetzt („1“).
Fand der Übertrag nicht statt, ist CY rückgesetzt („0“).
CY wird durch Operationen der ALU beeinflußt.

AC = Auxiliary Carry = Hilfsübertrag

Tetradenübertrag von Ergebnisbit 2^3 nach 2^4
Fand der Übertrag statt, ist AC gesetzt („1“).
Fand der Übertrag nicht statt, ist AC rückgesetzt („0“).
AC wird durch Operationen der ALU beeinflußt.

F0 = Flag 0 = Kennzeichenbit 0

Benutzerdefinierbares Einzelbit
Wird durch Software beeinflußt.

RS0/1 = Registerbank Select 0/1 = Registerbank Auswahl 0/1

Stellt die momentan aktive Registerbank ein.

RS1	RS0	Registerbank
0	0	Bank 0
0	1	Bank 1
1	0	Bank 2
1	1	Bank 3

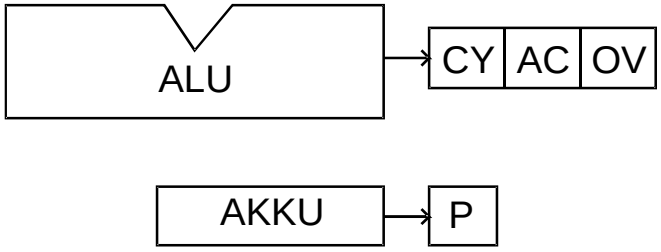
RS0/1 werden durch Software beeinflußt.

OV = Overflow = Überlauf

Zeigt bei vorzeichenbehafteter Arithmetik ein Über/Unterschreiten des Zahlenbereichs +127 / - 128 im Ergebnis an.
Bei Über/Unterschreiten ist OV gesetzt („1“)
Im erlaubten Bereich ist OV rückgesetzt („0“).
Bei Multiplikation mit Ergebnis > 255 gesetzt.
Bei Division durch Teilung mit Null gesetzt.
OV wird durch Operationen der ALU beeinflußt.

P = Parity = Parität

Stellt die Parität von Akku einschließlich P-Flag immer auf eine gerade (even) Anzahl „1“ ein. Befindet sich im Akku eine gerade Anzahl „1“ ist P-Flag „0“. Befindet sich im Akku eine ungerade Anzahl „1“ ist P-Flag „1“.
Das P-Flag wird automatisch durch den Akkuinhalt beeinflußt.



Nach einem Reset ist der Inhalt des PSW 0000 0000 binär (00h). Damit ist automatisch Registerbank 0 aktiviert.

5.2.2 Der Akkumulator

Der Akkumulator (ACC, Akku, Register A) ist das Register, das an den meisten Operationen der ALU beteiligt ist. Er stellt dabei meist einen der Operanden und nimmt das Ergebnis auf. Er ist ein bitadressierbares SFR (Adr. E0h), und kann damit auch als direkt adressierbare Speicherzelle behandelt werden. Nach einem Reset beinhaltet der Akku 00h.

5.2.3 Der Hilfsakkumulator B

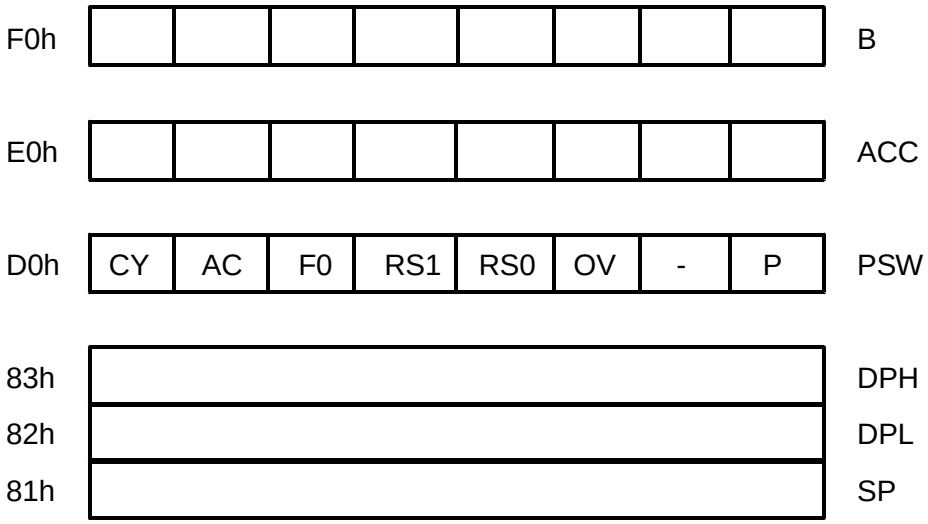
Der Hilfsakkumulator B ist das SFR mit der Adresse F0h. Er wird von der ALU für die Multiplikation und die Division benötigt. Sofern keine dieser Operationen durchgeführt wird, kann der Hilfsakku als direkt adressierbare Speicherzelle verwendet werden. Der Hilfsakku ist Bitadressierbar. Nach einem Reset beinhaltet der Hilfsakku 00h.

5.2.4 Der Stackpointer

Der Stack (Stapel) der MCS51-Mikrocontroller befindet sich im direkt/indirekt adressierbaren Speicher (Adr. 00h-7Fh). Der Stackpointer (Stapelzeiger, SP) ist ein 8Bit breites Zeigerregister auf den Stack. Er zeigt immer auf den aktuellen Stapeleintrag. Der Stackpointer ist das SFR mit der Adresse 81h. Er kann als direkt adressierbare Speicherzelle angesprochen werden. Er wird durch die Unterprogrammbefehle und die PUSH/POP-Befehle automatisch verwaltet. Bei einem PUSH-Befehl wird zunächst der SP um eins erhöht, und dann das Byte in den Stack eingetragen. Der SP bleibt auf den Eintrag stehen. Bei einem POP-Befehl wird zunächst das Byte aus dem Stack gelesen und dann der SP um eins erniedrigt. Bei den CALL- und RET-Befehlen werden jeweils zwei Byte (16Bit Rückkehradresse) im Stack verwaltet. Nach einem Reset beinhaltet der SP 07h, der erste Eintrag findet damit in Speicherzelle 08h statt. Soll der Stack an einer anderen Stelle im Speicher liegen, kann der SP durch den Befehl MOV SP,#ko mit der neuen Zieladresse geladen werden.

5.2.5 Der Datenpointer

Der Datenpointer (DPTR) ist ein 16Bit breites Zeigerregister. Er ist aus zwei 8Bit breiten SFR zusammengesetzt (DPL-DataPointer Low 82h, DPH-DataPointer High 83h). Er ist nicht bitadressierbar. Der Datenpointer wird als Zeiger für indirekte Adressierung im Programmspeicher und im externen Datenspeicher eingesetzt. Durch seine Breite von 16Bit kann er auf beliebige Adressen innerhalb eines 64kB großen Speicherbereichs zeigen. Es gibt im Befehlssatz Befehle um den Datenpointer mit einer 16Bit Konstante zu laden (MOV DPTR,#ko16), und um ihn zu Inkrementieren (INC DPTR). Er kann nicht Dekrementiert werden, weshalb Daten im Speicher immer so anzulegen sind, daß in aufsteigender Reihenfolge auf sie zugegriffen wird.



5.3 Adressierungsarten

Bei Mikrocontrollern der MCS51-Familie unterscheidet die CPU mehrere verschiedene Adressierungsarten.

5.3.1 Direkte Adressierung

Die Adresse der Speicherzelle wird im Befehl direkt als Zahlenwert oder mit ihren Namen angegeben. Der Akku und alle anderen SFR sind direkt adressierbare Speicherzellen. Sie werden immer in direkter Adressierung angesprochen.

Beispiel:

```
MOV A,21h   Adresse 21h als Zahl direkt angegeben
MOV A,P1    Adresse P1 (90h) mit Namen direkt angegeben
```

5.3.2 Indirekte Adressierung

Die Adresse der Speicherzelle wird vor dem eigentlichen Befehl in ein Zeigerregister geladen. Im Befehl wird das Zeigerregister in dem sich die Adresse befindet angegeben. Im internen Datenspeicher können Register R0 oder R1 als Zeigerregister verwendet werden. Im Programmspeicher (intern und extern) oder im externen Datenspeicher wird der Datenpointer (DPTR) als Zeigerregister eingesetzt. In den Mnemonischen Bezeichnungen der Befehle findet sich das @-Zeichen als Kennzeichen für indirekte Adressierung.

Beispiele:

```
MOV R0,#3Ch         Adresse in Zeigerregister laden
MOV A,@R0   Quelle (@R0 = 3Ch) ist indirekt adressiert

MOV DPTR,#14A0h     Adresse in Zeigerregister laden
MOVX A,@DPTR        Quelle (@DPTR = 14A0h) ist indirekt adressiert
```

5.3.3 Registeradressierung

Im Mnemonischen Befehl wird ein Registerkürzel (Rr) angegeben. Im Maschinencode befindet sich eine 3Bit-Kurzadresse für das entsprechende Register.

Register	Kurzadresse
R0	000
R1	001
R2	010
R3	011
R4	100
R5	101
R6	110
R7	111

Beispiel: MOV A,Rr
Befehlsrahmen = 1110 1rrr
rrr = Registerkurzadresse

MOV A,R5 = 1110 1101 = EDh

Die Adresse der Speicherzelle muß nicht gesondert angegeben werden, da sie die unmittelbar auf den OP-Code folgende ist.

Beispiel: MOV A,#12h Maschinencode 74 12

Der Datenwert 12h befindet sich in der Speicherzelle unmittelbar hinter dem OP-Code 74h.

5.3.4 Unmittelbare Adressierung

Die Adresse der Speicherzelle befindet sich in einem gewissen Abstand zum momentanen Befehlszählerstand. Wird bei Sprüngen über kurze Distanzen (SJMP rel, bedingte Sprünge) und beim Befehl MOVC A,@A+PC angewandt. Bei den Sprungbefehlen ist die relative Distanz ein vorzeichenbehafteter 8Bit-Wert. Dadurch ist die Sprungweite auf 127 Adressen vorwärts und 128 Adressen rückwärts eingeschränkt. Als Basisadresse zur Berechnung des Sprungziels wird die Adresse des Folgebefehls herangezogen.

Beispiele:

```
SJMP TEST   80 01      80 = OP-Code, 01 = Sprungdistanz (1 Adresse vorwärts)
NOP         00
TEST:
```

```
TEST:
NOP         00
SJMP TEST   80 FD      80 = OP-Code, FD = Sprungdistanz (3 Adressen rückwärts)
```

5.3.5 Relative Adressierung

Im Befehlssatz sind meist verschiedene Adressierungsarten in einem Befehl gleichzeitig vertreten.

Beispiele: MOV R5,#75h Ziel = R5 - Registeradressierung

Quelle = #75h - unmittelbare Adressierung

ANL P0,@R0 Ziel = P0 - direkte Adressierung

Quelle = @R0 - indirekte Adressierung