Mikrocontroller-Grundlagen Teil 7

Die einzelnen Befehle bzw. Befehlsgruppen mit jeweils einem kleinen Beispielprogramm beschreibt der vorliegende Artikel.

3. MCS-51-Befehlssatz

Der von INTEL festgelegte, gut strukturierte MCS-51-Befehlssatz läßt sich in Transport-, Logikverarbeitungs-, Arithmetik-, Bitmanipulations- und Programmsteuerbefehle unterteilen.

Den grundsätzlichen Aufbau der Beispielprogramme zeigt Abbildung 47. Ganz links sind die von dem Assembler erzeugte Adresse und der Maschinencode, der direkt vom Mikrocontroller abgearbeitet wird, zu sehen. Bei umfangreicheren Befehlen oder auch Parametern können hier auch mehrere Bytes nacheinander stehen. Der folgende Text der Zeile ist vom Programmierer zu erstellen.

```
1234 00    Lxx : NOP      ; Mache
1235 80FD      SJMP Lxx ; nichts
```

Bild 47: Grundsätzlicher Aufbau der Beispielprogramme

Das mit „Lxx:" gekennzeichnete Label dient als Sprungmarke für entsprechende Schleifenkonstruktionen, danach folgt der eigentliche Befehl. In unserem Beispiel wurde der Befehl „NOP" gewählt, der, wie die Abkürzung von „No Operation" schon sagt, keine Funktion besitzt, nur der Adreßzähler wird um 1 erhöht.

Nach dem folgenden „;" kann bis zum Zeilenende ein beliebiger Kommentartext stehen.

In der nächsten Zeile ist neben dem Maschinencode der Befehl „SJMP Lxx" zu sehen. Dieser Befehl springt relativ zum derzeitigen Programmzähler zu dem angegebenen Label („Lxx").

Diese Schleifenkonstruktion sorgt dafür, daß die Beispielbefehle vom Prozessor immer wieder durchlaufen werden, bis der Prozessor über den Reset-Taster zurückgesetzt oder die Spannungsversorgung abgeschaltet wird.

Die nachfolgend beschriebenen Beispielprogramme sind in dem EPROM vom Typ ELV9477 untergebracht, welches in die Mikrocontroller-Grundschaltung zu setzen ist. Hierzu müssen auf der Programm-Auswahlplatine, wie in Kapitel 2.17.3 („ELV-journal" 3/94, Seite 77) beschrieben, die Drehschalter S 1 und S 2 entsprechend eingestellt sein.

3.1 Transportbefehle

Einen großen Teil des Befehlsvorrates umfassen die Transportbefehle, die den Datentransfer zwischen den Registern, internem und externem RAM und den Port-Bausteinen übernehmen.

Abbildung 48 zeigt eine Übersicht über die Transportbefehle der MCS-51-Familie mit einer kurzen Beschreibung in englischer Sprache. Die Mnemoniks stellen die Abkürzung der in Originalsprache verfaßten Beschreibung dar. Die ersten 15 Befehle nehmen einen 8-Bit-Datentransfer über unterschiedliche Adressierungsarten vor. Tabelle 15 zeigt in übersichtlicher Form die Datentransfermöglichkeiten für 8-Bit-Daten über die unterschiedlichen Adressierungsarten, die im Kapitel 2.22 („ELV-journal" 4/94, Seite 80) ausführlich beschrieben wurden.

Tabelle 15: Datentransfermöglichkeiten für 8-Bit-Daten über die unterschiedlichen Adressierungsarten

von / nach	A	Rr	direct	@Ri	#data
A	-	x	x	x	x
Rr	x	-	x	-	x
direct	x	x	x	x	x
@Ri	x	-	x	-	x

Das Programm in Abbildung 49 zeigt ein Beispiel für die Verwendung des „MOV direct, direct"-Befehls. Hierzu wird in einer Endlosschleife der Inhalt des Ports P1 auf direktem Wege in den Port P3 kopiert. Praktisch bedeutet dies, daß die 8 Leuchtdioden auf der LED-Ausgabeplatine die Schalterstellungen der 8 Kippschalter auf der Schalterplatine widerspiegeln.

Für die direct-Adresse von Port 1 und Port 3 sind hier die Synonyme P1 und P3 eingesetzt. Der verwendete Assembler erkennt diese und übersetzt sie entsprechend.

Voraussetzung für den Betrieb der Beispielprogramme ist, daß die im Kapitel 2.17 beschriebene LED-Ausgabeschaltung, Schalterplatine und Programm-Auswahlplatine an die Mikrocontroller-Grund-

Mnemonik		Beschreibung
MOV	A,Rr	Move Register to Accumulator
MOV	A,direct	Move direct (byte) to Accumulator
MOV	A,@Ri	Move indirect RAM to Accumulator
MOV	A,#data	Move immediate data to Accumulator
MOV	Rr,A	Move Accumulator to Register
MOV	Rr,direct	Move direct (byte) to Register
MOV	Rr,#data	Move immediate data to Register
MOV	direct,A	Move Accumulator to direct byte
MOV	direct,Rr	Move Register to direct byte
MOV	direct,direct	Move direct (byte) to direct (byte)
MOV	direct,@Ri	Move indirect RAM to direct byte
MOV	direct,#data	Move immediate data to direct byte
MOV	@Ri,A	Move Accumulator to indirect RAM
MOV	@Ri,direct	Move direct (byte) to indirect RAM
MOV	@Ri,#data	Move immediate data to indirect RAM
MOV	DPTR,#data16	Load Data Pointer with a 16-bit constant
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to Accumulator
MOVC	A,@A+PC	Move Code byte relative to PC to Accumulator
MOVX	A,@Ri	Move external RAM (8-Bit addr) to Accumulator
MOVX	A,@DPTR	Move external RAM (16-Bit addr) to Accumulator
MOVX	@Ri,A	Move Accumulator to external RAM (8-Bit addr)
MOVX	@DPTR,A	Move Accumulator to external RAM (16-Bit addr)
PUSH	direct	Push direct (byte) onto stack
POP	direct	Pop direct (byte) from stack
XCH	A,Rr	Exchange Register with Accumulator
XCH	A,direct	Exchange direct (byte) with Accumulator
XCH	A,@Ri	Exchange indirect RAM with Accumulator
XCHD	A,@Ri	Exchange low-order Digit from ind.RAM w.A

Bild 48: Übersicht über die Transportbefehle der MCS-51-Familie

```
008E 8590B0  L1:   MOV   P3,P1    ; P1 nach P3 schreiben
0091 80FB          SJMP  L1       ; Schleife
```

Bild 49: Testprogramm 1

```
0093 E590    L2:   MOV   A,P1     ; P1 lesen
0095 F5B0          MOV   P3,A     ; Wert nach P3 schreiben
0097 80FA          SJMP  L2       ; Schleife
```

Bild 50: Testprogramm 2

schaltung angeschlossen sind.

Die gleiche Ausgabe wie das erste Testprogramm liefert das Programm 2 (Abbildung 50) mit dem Unterschied, daß in der ersten Programmzeile zunächst der Inhalt des Ports P1 in den Akku geladen und danach in der nächsten Zeile auf den Port P3 geschrieben wird.

Das Beispielprogramm 3 (Abbildung 51) zeigt die Zusammenhänge zwischen der direkten und indirekten Registeradressierung. Zunächst wird die Registerbank 1 selektiert und das Register R 0 mit dem konstanten Wert 0EH geladen. Der Befehl in der nächsten Zeile schreibt den Inhalt von Port P1 in die Speicherzelle, die durch

```
0099 75D008        MOV   PSW, #008H  ; Registerbank 1 wählen
009C 780E          MOV   R0,#0EH     ; Speicherplatz von R6
009E A690    L3:   MOV   @R0,P1      ; P1 ins RAM lesen
00A0 8EB0          MOV   P3,R6       ; Wert nach P3 schreiben
00A2 80FA          SJMP  L3          ; Schleife
```

Bild 51: Testprogramm 3

```
00A4 90E000        MOV   DPTR,#0E000H ; Zeiger laden
00A7 E0      L4:   MOVX  A, @DPTR    ; DIL-Schalter lesen
00A8 F5B0          MOV   P3,A        ; Wert nach P3 schreiben
00AA 80FB          SJMP  L4          ; Schleife
```

Bild 52: Testprogramm 4

```
00AC C090    L5:   PUSH  P1         ; P1 auf Stack ablegen
00AE D0B0          POP   P3         ; Wert nach P3 schreiben
00B0 80FA          SJMP  L5         ; Schleife
```

Bild 53: Testprogramm 5

```
00B2 9000C1        MOV   DPTR,#L6TAB  ; Zeiger laden
00B5 7820          MOV   R0,#020H     ; RAM-Zeiger
00B7 A690    L6:   MOV   @R0,P1       ; P1 ins RAM einlesen
00B9 7400          MOV   A, #000H     ; Akku mit 0 laden
00BB D6            XCHD  A, @R0       ; nur die unteren 4 Bit tauschen
00BC 93            MOVC  A, @A+DPTR   ; Tabelle lesen
00BD F5B0          MOV   P3,A         ; Wert nach P3 schreiben
00BF 80F6          SJMP  L6           ; Schleife
00C1 01      L6TAB: DB 00000001B       ; Schalterkombination 0
00C2 02             DB 00000010B      ; Schalterkombination 1
00C3 04             DB 00000100B      ; Schalterkombination 2
00C4 08             DB 00001000B      ; Schalterkombination 3
00C5 10             DB 00010000B      ; Schalterkombination 4
00C6 20             DB 00100000B      ; Schalterkombination 5
00C7 40             DB 01000000B      ; Schalterkombination 6
00C8 80             DB 10000000B      ; Schalterkombination 7
00C9 FE             DB 11111110B      ; Schalterkombination 8
00CA FD             DB 11111101B      ; Schalterkombination 9
00CB FB             DB 11111011B      ; Schalterkombination 10
00CC F7             DB 11110111B      ; Schalterkombination 11
00CD EF             DB 11101111B      ; Schalterkombination 12
00CE DF             DB 11011111B      ; Schalterkombination 13
00CF BF             DB 10111111B      ; Schalterkombination 14
00D0 7F             DB 01111111B      ; Schalterkombination 15
```

Bild 54: Testprogramm 6

das Register R0 (0EH) adressiert ist. Da die Registerbank 1 selektiert ist, schreibt der folgende Befehl den Inhalt von dem Register R 6 (Adresse 0EH) in den Port P3. Damit generiert dieses dritte Programm die gleichen Ausgaben wie die vorbeschriebenen, allerdings mit anderen Befehlen ausgeführt. Die zwischengespeicherten Werte lassen sich für weiterführende Aufgaben nutzen.

Eine ganz andere Funktion zeigt das Beispielprogramm 4 aus Abbildung 52. Zunächst wird der 16-Bit-Datenzeiger DPTR mit der Adresse 0E000H, über die die beiden 10fach-Drehschalter auf der Programm-Auswahlplatine auslesbar sind, geladen. In der Schleife wird anschließend die durch den 16-Bit-Zeiger DPTR adressierte externe Speicherzelle (0E000H) gelesen und mit dem folgenden Befehl auf den Port P3 ausgegeben. Nach dem Start des Programmes werden die an den beiden 10fach-Drehschaltern auf der Programm-Auswahlplatine anliegenden logischen Pegel zyklisch auf die 8 Leuchtdioden der Ausgabe-Schaltung gegeben.

Zu den Transportbefehlen gehört auch die Ablage von Speicherinhalten auf dem Stack. Abbildung 53 zeigt hier ein Programm, welches diese Funktion benutzt. In der ersten Zeile wird mit dem Befehl „PUSH" der Inhalt des Ports P1 auf dem Stack abgelegt, mit dem folgenden Befehl „POP" wieder zurückgeholt und in den Port P3 geschrieben. Das Ergebnis entspricht damit den Programmen 1 bis 3.

Das in Abbildung 54 abgedruckte Testprogramm 6 zeigt beispielhaft die Verwendung von ROM-Tabellen. Den wichtigsten Bestandteil dieses Programmes stellt die ab der Marke „L6TAB" beginnende ROM-Tabelle dar. Üblicherweise muß der Assembler die vom Programmierer erstellten Befehle in einen maschinenlesbaren Code umsetzen. Bei der Verwendung beispielsweise von ROM-Tabellen, muß der Assembler diese nicht mehr übersetzen, sondern direkt in den Code-Bereich übernehmen. Hierzu wurde der Pseudo-Befehl „DB" (Define Byte) eingeführt. Die nachfolgenden Einträge (bis zum Ende der Zeile) werden dann direkt vom Assembler in den Code-Speicher übernommen.

Das Testprogramm liest zyklisch die an Port P1 anstehenden Pegel aus. Die Stellungen der 4 niederwertigen Schalter S 0 bis S 3 ergeben in ihrer Binär-Kombination 16 verschiedene Zahlenwerte, die als Zeiger auf die Tabelle zeigen und den dort stehenden Eintrag auslesen und auf den Ausgabe-Port P3 ausgeben. Zur Verdeutlichung sind die Tabelleneinträge in binärer Schreibweise vorgenommen.

Sind beispielsweise alle Schalter S 0 bis S 3 auf 0 eingestellt (binär 0000), so wird auf den ersten Tabelleneintrag zugegrif-

Mnemonik		Beschreibung
ANL	A,Rr	AND Accumulator with Register to Accumulator
ANL	A,direct	AND Accumulator with direct (byte) to Accumulator
ANL	A,@Ri	AND Accumulator with indirect RAM to Accumulator
ANL	A,#data	AND Accumulator with immediate data to Accumulator
ANL	direct,A	AND direct with Accumulator to direct (byte)
ANL	direct,#data	AND direct with immediate data to direct (byte)
ORL	A,Rr	OR Accumulator with Register to Accumulator
ORL	A,direct	OR Accumulator with direct (byte) to Accumulator
ORL	A,@Ri	OR Accumulator with indirect RAM to Accumulator
ORL	A,#data	OR Accumulator with immediate data to Accumulator
ORL	direct,A	OR direct with Accumulator to direct (byte)
ORL	direct,#data	OR direct with immediate data to direct (byte)
XRL	A,Rr	Exclusive-OR Accumulator with Register to Accumulator
XRL	A,direct	Exclusive-OR Accumulator with direct (byte) to Accumulator
XRL	A,@Ri	Exclusive-OR Accumulator with indirect RAM to Accumulator
XRL	A,#data	Exclusive-OR Accumulator with immediate data to Accumulator
XRL	direct,A	Exclusive-OR direct with Accumulator to direct (byte)
XRL	direct,#data	Exclusive-OR direct with immediate data to direct (byte)
CLR	A	Clear Accumulator
CPL	A	Complement Accumulator
RL	A	Rotate Accumulator Left
RLC	A	Rotate Accumulator Left trough Carry flag
RR	A	Rotate Accumulator Right
RRC	A	Rotate Accumulator Right trough Carry flag
SWAP	A	Swap nibbles within the Accumulator

Bild 55: Übersicht über die Logikbefehle der MCS-51-Familie

fen, womit nach Ausgabe auf den Port P3 die Leuchtdiode D 0 aktiv ist und D 1 bis D 7 auf der 8-Bit-LED-Ausgabeschaltung erloschen sind.

Zunächst wird in dem Beispielprogramm der 16-Bit-Zeiger DPTR mit der Adresse des ersten Tabelleneintrages geladen. In der dritten Zeile wird der Inhalt des Ports P1 in die durch das Register R 0 adressierte Speicherzelle eingelesen. Nach dem Laden des Akkumulators mit 000H tauscht der folgende Befehl „XCHD A, @R0" den Inhalt der unteren 4 Bits des Akkumulators mit den unteren 4 Bits der durch das Register R 0 adressierten Speicherzelle aus. Da zuvor der Akkumulator mit dem Wert 0 geladen wurde und nur das untere Nibble ausgetauscht ist, ergibt sich daraus der maximale Zahlenumfang von 0 bis 15 im Akkumulator.

Mit dem folgenden Befehl „MOVC A,@A+DPTR" wird nun zunächst der Inhalt des Akkumulators zu dem Inhalt des 16-Bit-Datenzeigers DPTR addiert und anschließend der Inhalt der berechneten Speicherzelle aus dem internen ROM gelesen und in den Akkumulator zurückgespeichert.

Der nächste Befehl schreibt den Inhalt des Akkumulators direkt in den Port P3.

3.2 Logikverarbeitungs-Befehle

Einen wichtigen Teil des MCS-51-Befehlssatzes stellt die Logikverarbeitung dar, die fast ausschließlich in Zusammenarbeit mit dem Akkumulator stattfindet. Abbildung 55 zeigt eine Übersicht über die Logikbefehle der MCS-51-Familie mit einer Kurzbeschreibung.

Das Programm 7 in Abbildung 56 zeigt ein Beispiel für die bitweise UND-Verknüpfung der aus Port 1 gelesenen Eingangszustände mit der Konstanten 11111110B. Das im Akkumulator abgespeicherte Ergebnis wird anschließend auf den Port P3 geschrieben. Die bitweise UND-Verknüpfung bewirkt, daß das Bit gelöscht wird, wobei alle anderen Bits des Akkumulators unverändert bleiben.

Die UND-Verknüpfung wird vorwiegend dazu verwendet, um in einem Byte (8 Bit) ein oder mehrere Bits gleichzeitig auf 0 zu setzen (ausmaskieren).

Am Ausgangsport P3 erscheinen die logischen Zustände der Schalter S 1 bis S 7; D 0 bleibt unabhängig von der Stellung des Schalters S 0 auf Low-Pegel.

Das Beispielprogramm 8 (Abbildung 57) zeigt die Verwendung der bitweisen ODER-Verknüpfung. Der Inhalt des Ports 1 (Schalterstellung S 0 bis S 7) wird in den Akkumulator gelesen, dort mit der Konstanten 00000001B ODER-verknüpft und anschließend auf den Port P3 ausgegeben.

Die ODER-Verknüpfung bewirkt, daß die Schalterstellungen von S 1 bis S 7 unverändert auf die zugehörigen Ports bzw. Leuchtdioden D 1 bis D 7 ausgegeben werden. Durch die ODER-Verknüpfung von Bit 0 mit der Konstanten 1 liegt unabhängig von der Schalterstellung von S 0 an dem Port P3.0 ein High-Pegel an.

Die ODER-Verknüpfung wird meistens für das Setzen bestimmter Bits in einem Byte (8 Bit) verwendet.

Eine bitweise Invertierung läßt sich durch eine Exklusiv-ODER-Verknüpfung, wie in Abbildung 58 gezeigt, vornehmen. Durch die Verknüpfung des an Port 1 eingelesenen 8-Bit-Wertes mit der Konstanten 00000001B wird die Information des Bits 0 invertiert, wobei die Informationen der höherwertigen Bits 1 bis 7 unverändert bleiben. Praktisch wirkt sich diese Verknüpfung in der Art aus, daß die Schalterstellung des Schalters S 0 invertiert und die der Schalter S 1 bis S 7 unverändert auf die LED-Ausgabeplatine ausgegeben werden.

Für die gleichzeitige Invertierung aller 8 Bits im Akkumulator ist der Befehl „CPL

```
00D1 E590    L7:    MOV   A,P1              ; P1 lesen
00D3 54FE           ANL   A, #11111110B     ; Bit 0 ausblenden
00D5 F5B0           MOV   P3,A              ; Wert nach P3 schreiben
00D7 80F8           SJMP  L7                ; Schleife
```

Bild 56: Testprogramm 7

```
00D9 E590    L8:    MOV   A,P1              ; P1 lesen
00DB 4401           ORL   A, #00000001B     ; Bit 0 setzen
00DD F5B0           MOV   P3,A              ; Wert nach P3 schreiben
00DF 80F8           SJMP  L8                ; Schleife
```

Bild 57: Testprogramm 8

```
00E1 E590    L9:    MOV   A,P1              ; P1 lesen
00E3 6401           XRL   A, #00000001B     ; Bit 0 invertieren
00E5 F5B0           MOV   P3,A              ; Wert nach P3 schreiben
00E7 80F8           SJMP  L9                ; Schleife
```

Bild 58: Testprogramm 9

```
00E9 E590    L10:   MOV   A,P1              ; P1 lesen
00EB F4             CPL   A                 ; Akkuinhalt invertieren
00EC F5B0           MOV   P3,A              ; Wert nach P3 schreiben
00EE 80F9           SJMP  L10               ; Schleife
```

Bild 59: Testprogramm 10

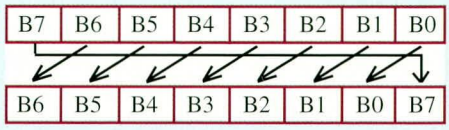

Bild 60: Funktion des Befehls RL A

A" vorhanden. Das Programm 10 in Abbildung 59 liest die an Port 1 befindlichen logischen Pegel der Schalter S 0 bis S 7 ein, invertiert alle 8 Bits und gibt das Ergebnis an den Port 3 mit der LED-Ausgabeschaltung aus. Die Funktion dieses Befehls entspricht der bitweisen Exklusiv-ODER-Verknüpfung mit dem Befehl „XRL A#0FFH", allerdings in kürzerer Ausführungszeit.

Für Bitmanipulationen werden neben den logischen Verknüpfungsbefehlen die Schiebe- bzw. Rotier-Befehle verwendet. Abbildung 60 zeigt die Funktion des Befehls „RL A" (Rotate Left Accumulator). Der Befehl schiebt den Inhalt des Akkumulators um 1 Bit nach links, d. h. Bit 0 → Bit 1, Bit 1 → Bit 2, ... Der Inhalt von Bit 7 wird in Bit 0 vom Akkumulator übertragen.

```
00F0 E590    L11:   MOV   A,P1      ; P1 lesen
00F2 23             RL    A        ; Linksschieben um 1
00F3 F5B0           MOV   P3,A     ; Wert nach P3 schreiben
00F5 80F9           SJMP  L11      ; Schleife
```

Bild 61: Testprogramm 11

Abbildung 61 zeigt das Beispielprogramm 11, in dem nach dem Lesen des Portinhaltes von P1 der Inhalt des Akkumulators um 1 nach links verschoben und anschließend auf Port 3 ausgegeben wird. Praktisch bedeutet dies, daß die Schalterstellungen von S 0 bis S 6 durch die Leuchtdioden D 1 bis D 7 abgebildet werden und D 0 die Schalterstellung von S 7 wiedergibt.

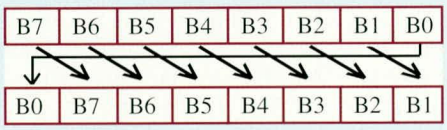

Bild 62: Funktion des Befehls RR A

Der Befehl „RR A", dessen Funktionsweise Abbildung 62 zeigt, läßt den Akkuinhalt um 1 nach rechts rücken, d. h. die Inhalte der Bits werden vom höherwertigen in das nächst niederwertige Bit geschoben. Der Inhalt von Bit 0 wird dabei nach Bit 7 übertragen.

Abbildung 63 zeigt ein Beispielprogramm, welches die Funktion dieses Befehls verdeutlicht. Dabei wird der Inhalt von Port 1 gelesen, um 1 nach rechts geschoben und auf den Port P3 ausgegeben. Die Leuchtdioden D 0 bis D 6 geben dabei die Schalterstellungen von S 1 bis S 7 wieder, während D 7 die Stellung von S 0 anzeigt.

Bild 63: Testprogramm 12

```
00F7 E590    L12:   MOV   A,P1      ; P1 lesen
00F9 03             RR    A        ; Rechtsschieben um 1
00FA F5B0           MOV   P3,A     ; Wert nach P3 schreiben
00FC 80F9           SJMP  L12      ; Schleife
```

Eine Erweiterung der Rotationsmöglichkeiten stellen die Befehle „RLC A" (Rotate Left with Carry) und „RRC A" (Rotate Right with Carry) dar, die vielfältige Anwendungsmöglichkeiten erlauben.

Abbildung 64 verdeutlicht die Funktion des Befehls „RLC A", der ähnlich wie der Befehl „RL A" die Inhalte der Bits 0 bis 6 um 1 nach links schiebt, wobei der Inhalt von Bit 7 in das Carry-Flag und dessen Inhalt in Bit 0 des Akkus übernommen wird. Die gleiche Funktion hat der in Abbildung 65 dargestellte Befehl „RRC A", mit dem Unterschied, daß der Inhalt nach rechts gerückt wird.

Abbildung 66 zeigt das Beispielprogramm 13, welches die Verwendungsmöglichkeiten der verschiedenen Rotierbefehle verdeutlicht. Zunächst wird der Inhalt von Port 1 eingelesen, über die folgenden Rotier-Befehle die Inhalte der Bits 0 und 1 im Akkumulator vertauscht und das Ergebnis auf den Port P3 ausgegeben. Praktisch hat dies zur Folge, daß die Schalterstellung von S 0 über D 1 und S 1 über D 0 wiedergegeben wird, während die übrigen Leuchtdioden die Inhalte der Schalter S2 bis S7 unverändert anzeigen. Zur Verdeutlichung der Funktionsweise dieses Programmes zeigt Abbildung 67 die Zustände

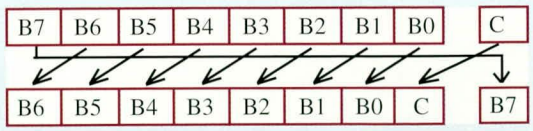

Bild 64: Funktionsweise des Befehls RLC A

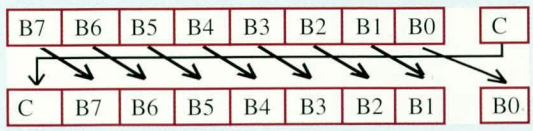

Bild 65: Funktionsweise des Befehls RRC A

des Akkumulators und des Carry-Bits nach der Ausführung der Rotierbefehle.

Besonders in der 4-Bit- und BCD-Arithmetik findet der „SWAP A"-Befehl (Abbildung 68) Anwendung, der das untere Nibble (4 Bit) mit dem oberen Nibble des Akkumulators vertauscht. Das gleiche Ergebnis würde auch die viermalige Verwendung des Befehls „RL A" oder „RR A" erzielen.

Abbildung 69 zeigt das Beispielprogramm 14, welches den Port P1 ausliest, das obere und untere Nibble vertauscht und das Ergebnis auf den Port P3 ausgibt. Praktisch bedeutet dieses, daß die Leuchtdioden D 4 bis D 7 die Schalterstellungen von S 0 bis S 3 wiedergeben, während D 0 bis D 3 die Schalterstellungen von S 4 bis S 7 anzeigen.

```
00FE E590    L13:   MOV   A,P1      ; P1 lesen
0100 13             RRC   A        ; Rechtsschieben mit Carry
0101 03             RR    A        ; Rechtsschieben
0102 33             RLC   A        ; Linksschieben mit Carry
0103 33             RLC   A        ; Linksschieben mit Carry
0104 F5B0           MOV   P3,A     ; Wert nach P3 schreiben
0106 80F6           SJMP  L13      ; Schleife
```

Bild 66: Testprogramm 13

Inhalt von A	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0	C
nach RRC A	C	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
nach RR A	P1.1	C	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.0
nach RLC A	C	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.0	P1.1
nach RLC A	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.0	P1.1	C

Bild 67 (links): Zustände des Akkumulators und Carry-Flags nach der Befehlsausführung

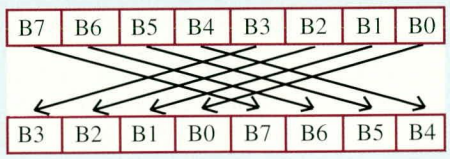

Bild 68: Funktion des Befehls SWAP A

Bild 69: Testprogramm 14

```
0108 E590    L14:   MOV   A,P1      ; P1 lesen
010A C4             SWAP  A        ; Nibble tauschen
010B F5B0           MOV   P3,A     ; Wert nach P3 schreiben
010D 80F9           SJMP  L14      ; Schleife
```

Im nächsten Teil dieser Artikelserie beschreiben wir die Arithmetik-, Bitmanipulations- und Programm-Steuerbefehle. ELV