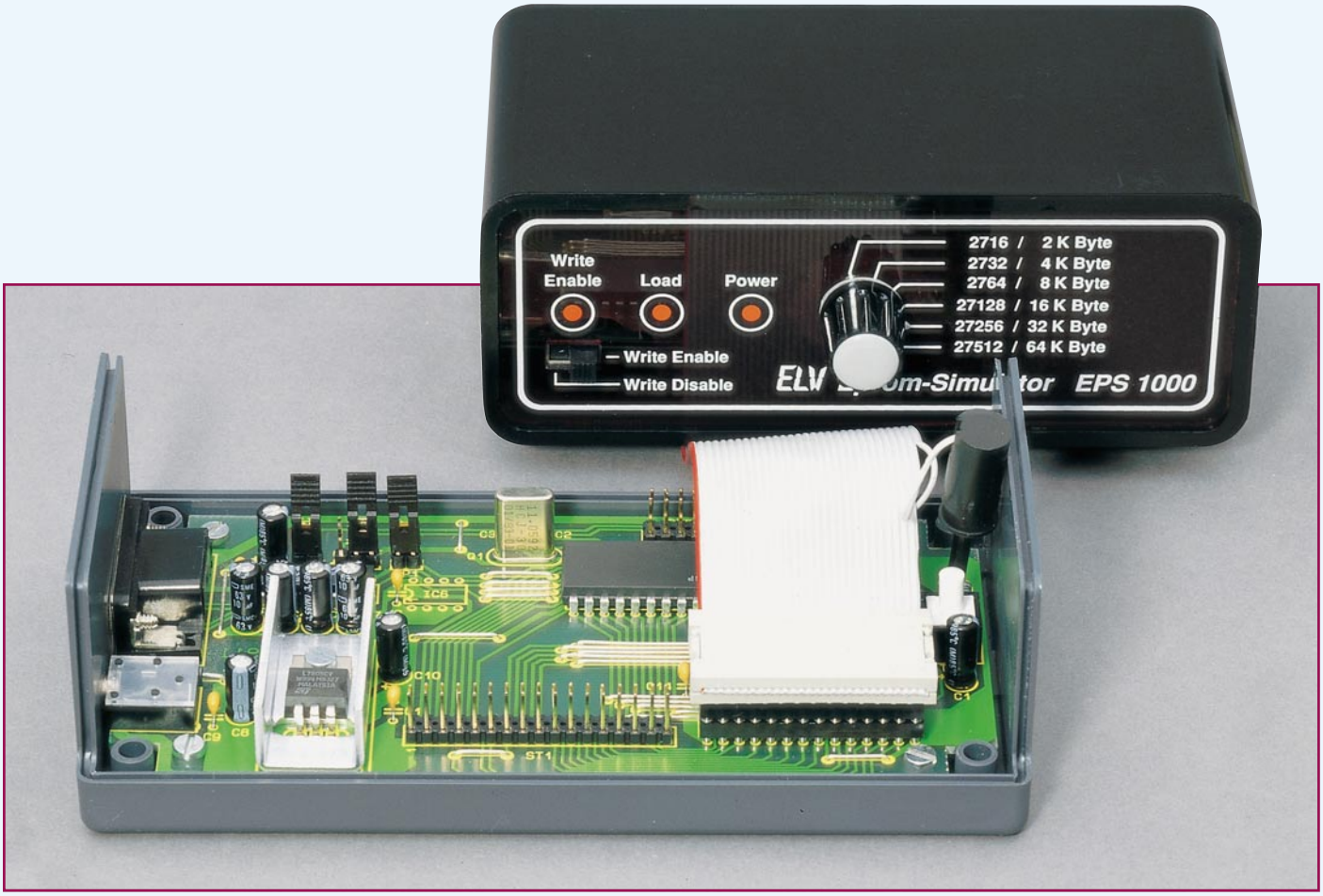

Mikrocontroller-Grundlagen

Teil 9

Im neunten Teil dieser Artikelserie werden die Programm-Steuerbefehle der MCS-51-Familie beschrieben.

3.5 Programm-Steuerbefehle

Um nicht nur eine lineare Abfolge von Befehlen abarbeiten zu können, sind die Programm-Steuerbefehle notwendig, womit sich unter anderem Programm- und Schleifenstrukturen sowie Bedingungsabfragen realisieren lassen.

Abbildung 86 zeigt eine Übersicht über die Programm-Steuerbefehle der MCS-51-Familie mit einer jeweiligen Kurzbeschreibung. Neben den unbedingten und bedingten Sprüngen beherrscht der Prozessor auch die Unterprogramm-Technik.

3.5.1 Unterprogramm-Befehle

Unterprogramm-Aufrufe verhalten sich wie unbedingte Sprungbefehle zu einer festen Adresse. Zusätzlich wird bei der Ausführung des Befehls ACALL oder LCALL die nach dem Befehl folgende Adresse (Rücksprungadresse) auf dem Stack, der sich im internen, direkt adressierbaren RAM befindet, abgelegt. Dazu erhöht der Prozessor den Stackpointer (SP) um 1 und legt zunächst das MSB (höherwertiges Byte) der Rücksprungadresse in der durch den Stackpointer adressierten Speicherzelle ab, erhöht den Stackpointer noch einmal und legt dann das niederwertige Byte der Rücksprungadresse ab.

Am Ende des Unterprogrammes steht der RET-Befehl. Bei der Ausführung wird zunächst die auf dem Stack abgelegte Rücksprungadresse gelesen, wobei jeweils der Stackpointer um 1 decrementiert wird. Der Prozessor lädt nun die Rücksprungadresse in den Programmcounter (PC) und fährt mit der Abarbeitung des nächsten Befehles fort.

Bei der Erstellung von Programmen ist zu beachten, daß die Stack-Statistik ausgeglichen sein muß. Nach dem Zurücksetzen des Prozessors ist der Stackpointer SP auf 07H initialisiert. Beim ersten Unterprogrammaufruf würde damit die Rücksprungadresse unter den Speicherzellen 08H und

09H abgelegt werden. Da sich im gleichen Bereich die Register der Registerbank 1 befinden, ist es unbedingt ratsam, gleich zu Anfang des Programmes den Stackpointer auf eine neue Anfangsadresse, die sich hinter dem Register- und Variablenbereich befinden sollte, zu initialisieren.

Der große Vorteil der Stack-Architektur liegt darin, daß sich eine automatisierte Unterprogramm- und Interrupt-Verarbeitungstechnik realisieren läßt. Damit lassen sich auch verschachtelte Unterprogrammstrukturen erstellen, ohne daß der Programmierer sich um die Verwaltung der Rücksprungadressen kümmern muß.

Vielfältige Verzweigungsmöglichkeiten erlauben einen effektiven Programmcode

Der Stack wächst mit jedem Unterprogrammaufruf oder auch Ablage von Daten in Richtung höherer Adressen des internen RAMs. Zu beachten ist deshalb, daß die Schachtelungstiefe der Unterprogramme sich dem zur Verfügung stehenden RAM-Speicherplatz anpassen muß, da anderenfalls ein unbeabsichtigtes Verhalten des Prozessors und damit ein Absturz die Folge wäre.

Weiterhin müssen unbedingt rekursive Programmaufrufe (Aufruf des eigenen Unterprogrammes) vermieden werden, da der MCS-51-Mikroprozessor einen begrenzten Speicherbereich besitzt.

Unterprogramme werden vielfach eingesetzt, um immer wiederkehrende Programmstücke von verschiedenen Stellen des Programmes ausführen zu lassen, da anderenfalls der Programmcode direkt in den Programmablauf einzufügen wäre. Abgeschlossene Teilaufgaben, die im ganzen Programm nur einmal vorkommen, werden ebenfalls vielfach zu Unterprogrammen zusammengefaßt, um die Übersichtlichkeit des Hauptprogrammes zu gewährleisten.

Unterschiedliche Programmstücke zur Lösung beispielsweise von Arithmetik-Aufgaben werden ebenfalls gerne als Unterprogramme zusammengefaßt, um diese dann in verschiedenen Programmen als abgeschlossene und getestete Programmteile verwenden zu können.

Beim Auftreten eines Interrupts (Unterbrechungsanforderung des Prozessors) wird ebenfalls wie beim Unterprogrammaufruf die Rücksprungadresse auf dem Stack abgelegt. Am Ende des Interrupt-Service-Programmteiles ist allerdings der RETI-Befehl zu verwenden, um das Interrupt-Flip-Flop des Prozessors zurückzusetzen.

Abbildung 87 zeigt das Testprogramm 25, welches die grundlegende Funktion des CALL-Befehls verdeutlicht. Der Befehl ACALL ruft das Unterprogramm mit der Bezeichnung L25A auf, das den Port-Inhalt von P 1 direkt zum Port P 3 kopiert. Durch den nachfolgenden Befehl RET erfolgt der Rücksprung zum Hauptprogramm, welches in der Endlosschleife fortwährend das Unterprogramm aufruft.

Abbildung 88 a zeigt den Zustand des Stacks vor Ausführung des CALL-Befehls. In unserem Beispiel zeigt der Stackpointer auf die RAM-Adresse 40 H. Nach Ausführung des CALL-Befehls ist die Rücksprungadresse gemäß Abbildung 88 b im RAM abgelegt und der Stackpointer entsprechend erhöht worden. Nach Abarbeitung des RET-Rücksprungbefehls wird der Inhalt des Stacks gelesen und der Stackpointer wieder um 2 decrementiert (Bild 88 c).

Der Stackpointer enthält wieder die Adresse 40 H. Im Stack selbst stehen noch die nicht mehr gebrauchten Rücksprungadressen, die automatisch bei weiteren Stack-Operationen überschrieben werden.

Die MCS-51-Familie unterscheidet 2 Befehle zum Aufrufen eines Unterprogrammes, deren Unterschied nur im Gültigkeitsbereich der Zieladresse besteht. Mit dem LCALL-Befehl (Long Call) läßt sich ein Unterprogramm an einer beliebigen Adresse im 64kByte-Adreßraum (16-Bit-Adresse) aufrufen. Der Befehl benötigt 3 Byte (1-Byte-Befehl + 2-Byteadresse) im Programmspeicher.

Für Unterprogrammaufrufe über kürzere Distanzen steht der Befehl ACALL (Absolute Call) zur Verfügung. Die Zieladresse muß sich dabei in dem gerade bearbeiteten 2k-Byte-Adreßbereich des Programmspeichers befinden. Befindet sich beispielsweise der momentane Programmstand bei Adresse 120 H, so darf sich die Sprungadresse des gewünschten Unterprogrammes im Bereich zwischen 0 und 7 FFH befinden.

Der nur 2 Byte lange Befehl enthält dazu im Befehlscode bereits die höherwertigen 3 Bit der Zieladresse, während die niederwertigen 8 Bit der Zieladresse sich im folgenden Byte befinden. Der Prozessor ergänzt nun diese 11-Bit-Adresse durch

ACALL	addr11	Absolute Subroutine Call
LCALL	addr16	Long Subroutine Call
RET		Return from Subroutine
RETI		Return from Interrupt
AJMP	addr11	Absolute Jump
LJMP	addr16	Long Jump
SJMP	rel	Short Jump (relative addr)
JMP	@A+DPTR	Jump indirect Accumulator relative to DPTR
JZ	rel	Jump if Accumulator is Zero
JNZ	rel	Jump if Accumulator is Not Zero
JC	rel	Jump if Carry flag is set
JNC	rel	Jump if Carry flag is Not set
JB	bit,rel	Jump if direct bit is set
JNB	bit,rel	Jump if direct bit is Not set
JBC	bit,rel	Jump if direct bit is set and Clear bit
CJNE	A,direct,rel	Compare direct to Accumulator and Jump if Not Equal
CJNE	A,#data,rel	Compare immediate data to A and Jump if Not Equal
CJNE	Rr,#data,rel	Compare immediate data to Reg. and Jump if Not Equal
CJNE	@Ri,#data,rel	Compare immediate data to ind.RAM and Jump if Not Equal
DJNZ	Rr,rel	Decrement register and Jump if Not Zero
DJNZ	direct,rel	Decrement direct and Jump if Not Zero
NOP		No Operation

Bild 86: Übersicht über die Programmablauf-Steuerungs-Befehle der MCS-51-Familie

```
01B5 31B9      L25:  ACALL   L25A   ; Unterprogrammaufruf (2kByte-Segment)
01B7 80FC            SJMP    L25    ; Schleife
01B9           L25A:                ; Unterprogramm-Einsprung
01B9 8590B0          MOV     P3, P1 ; Inhalt von P1 nach P3 kopieren
01BC 22              RET            ; zurück zum Hauptprogramm
```

Bild 87: Testprogramm 25

die höherwertigen 5 Bit des Programmcounters und erhält damit die 16-Bit-Ziel-Adresse.

3.5.2 Unbedingte Sprünge

AJMP-, LJMP- und SJMP sind Sprungbefehle, die den Prozessor anweisen, an der angegebenen Adresse mit dem Programmablauf fortzufahren. Die AJMP- und LJMP-Befehle unterscheiden sich, wie bei den CALL-Befehlen bereits beschrieben, nur durch die Länge des Adreßteils. Der 2-Byte-AJMP-Befehl (Absolute Jump) erlaubt einen Sprung in dem gerade aktiven 2kByte-Segment, während der 3-Byte-Befehl LJMP (Long Jump) einen Sprung zu einer beliebigen Adresse im 64-kByte-Befehlsraum des Mikroprozessors ermöglicht.

Zusätzlich kennt der Mikroprozessor noch den SJMP (Short Jump)-Befehl, der eine maximale Sprungweite von -128 bis +127 Byte, bezogen auf die Adresse des auf den Sprungbefehl unmittelbar folgenden Befehles, erlaubt. Bei dieser relativen Adressierungsart berechnet der Mikroprozessor zur Laufzeit des Programmes die Zieladresse.

Abbildung 89 zeigt ein Beispielprogramm, welches den Einsatz der Sprungbefehle verdeutlicht. Nach Abarbeitung des AJMP-Befehls wird das Programm an der mit L26A gekennzeichneten Adresse fortgesetzt, um nach Abarbeitung des dortigen LJMP-Befehles den Port-Inhalt von P 1 nach P 3 zu kopieren. Der anschließende SJMP-Befehl sorgt für die wiederkehrende Ausführung der Befehlsfolge.

Eine Besonderheit der absoluten Sprungbefehle stellt der JMP @A+DPTR-Befehl dar. Dieser berechnet zur Laufzeit des Programmes die Zieladresse aus der Summe des 8-Bit-Akkumulators und des 16-Bit-Datenzeigers DPTR, womit sich auf einfache Weise Sprungtabellen realisieren lassen.

Das in Abbildung 90 abgedruckte Testprogramm 27 zeigt beispielhaft die Verwendung des JMP @A+DPTR-Befehls. Das Testprogramm liest zyklisch den an Port P 1 anstehenden Inhalt aus. Die Stellungen der 3 niederwertigen Schalter S 0 bis S 2 (P1.0 bis P1.2) ergeben in ihrer Binär-Kombination 8 verschiedene Zahlenwerte. Der folgende ANL-Befehl maskiert die unteren 3 Bit aus, so daß sich zu diesem Zeitpunkt im Akkumulator ein Zahlenwert von 0 bis 7 befindet. Durch die beiden folgenden Befehle wird zunächst das Carry-Flag gelöscht und dessen Inhalt durch den nachfolgenden Schiebebefehl in den Akkumulator übertragen. Diese Rotation des Akkumulators bewirkt praktisch eine Multiplikation des Akkumulator-Inhaltes mit 2.

Durch den folgenden Sprungbefehl (JMP @A+DPTR) wird nun zum Inhalt des zu Anfang des Programmes geladenen Datenzeigers DPTR, der Inhalt des Akkumulators addiert und die Programmabarbeitung an der errechneten Adresse fortgesetzt. Beim Akku-Inhalt von 0 ergibt sich somit eine Programm-Fortsetzung mit dem Befehl AJMP L27T0, während beim Akkumulator-Inhalt 14 der Befehl AJMP L27T7 zur Abarbeitung gelangt. Die sich an den Sprungadressen befindenden Programmteile laden nun einen konstanten Wert in den Akku und geben bei der Programm-Adresse L27TGEM den Inhalt auf dem Port P 3 aus. Je nach Stellung der Schalter S 0 bis S 2 werden die verschiedenen Bit-Muster über die Leuchtdioden D 0 bis D 7 angezeigt.

3.5.3 Bedingte Sprünge

Eine weitere wichtige Befehlsgruppe stellen die bedingten Sprünge dar, deren Sprungziel von Bedingungen abhängig ist. Die Sprungweite und damit die Zieladresse liegt immer im Bereich von -128 bis +127 Byte, bezogen auf die Adresse des auf den Sprungbefehl unmittelbar folgenden Bytes.

Bedingte Sprung- bzw. Verzweigungsbefehle werden nur ausgeführt, wenn die

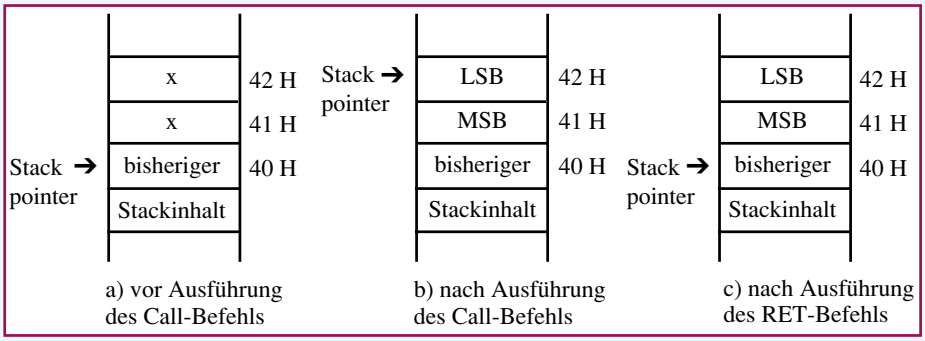

Bild 88 zeigt den Stackzustand

```
01BD 21C4      L26:   AJMP   L26A   ; Sprung im 2kByte-Segment
01BF 8590B0    L26B:  MOV    P3, P1 ; Inhalt von P1 nach P3 kopieren
01C2 80F9             SJMP   L26    ; Schleife
01C4 0201BF    L26A   LJMP   L26B   ; Sprung im 64kByte-Bereich
```

Bild 89: Testprogramm 26

```
01C7 9001D1  L27:     MOV    DPTR, #L27TAB  ; Zeiger auf Tabellenanfang setzen
01CA E590    L27A:    MOV    A, P1          ; P1 lesen
01CC 5407             ANL    A, #00000111B  ; nur die unteren 3 Bits (0..7)
01CE C3               CLR    C              ; Carry-Bit löschen
01CF 23               RL     A              ; Akkuinhalt mit 2 multiplizieren
01D0 73               JMP    @A+DPTR        ; weiter bei A+DPTR  (Tabelle)
01D1 21E1    L27TAB:  AJMP   L27T0          ; Sprungtabelle
01D3 21E5             AJMP   L27T1          ; Sprungtabelle
01D5 21E9             AJMP   L27T2          ; Sprungtabelle
01D7 21ED             AJMP   L27T3          ; Sprungtabelle
01D9 21F1             AJMP   L27T4          ; Sprungtabelle
01DB 21F5             AJMP   L27T5          ; Sprungtabelle
01DD 21F9             AJMP   L27T6          ; Sprungtabelle
01DF 21FD             AJMP   L27T7          ; Sprungtabelle
01E1 7401    L27T0:   MOV    A, #00000001B  ; Schalterkombination 0
01E3 801A             SJMP   L27TGEM        ; gemeinsame Weiterverarbeitung
01E5 7402    L27T1:   MOV    A, #00000010B  ; Schalterkombination 1
01E7 8016             SJMP   L27TGEM        ; gemeinsame Weiterverarbeitung
01E9 7404    L27T2:   MOV    A, #00000100B  ; Schalterkombination 2
01EB 8012             SJMP   L27TGEM        ; gemeinsame Weiterverarbeitung
01ED 7408    L27T3:   MOV    A, #00001000B  ; Schalterkombination 3
01EF 800E             SJMP   L27TGEM        ; gemeinsame Weiterverarbeitung
01F1 7410    L27T4:   MOV    A, #00010000B  ; Schalterkombination 4
01F3 800A             SJMP   L27TGEM        ; gemeinsame Weiterverarbeitung
01F5 7420    L27T5:   MOV    A, #00100000B  ; Schalterkombination 5
01F7 8006             SJMP   L27TGEM        ; gemeinsame Weiterverarbeitung
01F9 7440    L27T6:   MOV    A, #01000000B  ; Schalterkombination 6
01FB 8002             SJMP   L27TGEM        ; gemeinsame Weiterverarbeitung
01FD 7480    L27T7:   MOV    A, #10000000B  ; Schalterkombination 7
01FF F5B0    L27TGEM: MOV    P3, A          ; Ausgabe auf P3
0201 80C7             SJMP   L27A           ; Schleife
```

Bild 90: Testprogramm 27

Sprungbedingung erfüllt ist. Bei Nichterfüllung der Sprungbedingung setzt der Prozessor die Programmausführung bei der Adresse des auf den Sprungbefehl unmittelbar folgenden Bytes fort. Bei Erfüllung der Sprungbedingung wird entsprechend zur angegebenen Adresse verzweigt.

Je nach Befehl kann die Verzweigung bei gesetztem oder nicht gesetztem Bit erfolgen. Beim Befehl JZ (Jump if Zero) wird verzweigt, wenn der Inhalt des Akkumulators 0 ist, während beim Befehl JNZ (Jump if Not Zero) verzweigt wird, wenn der Inhalt des Akkumulators ungleich 0 ist.

Neben der 0-Prüfung des Akkumulators läßt sich die Programmausführung weiterhin durch den Inhalt des Carry Flags oder einer beliebigen bit-adressierbaren Variablen steuern.

Abbildung 91 zeigt das Testprogramm 28, welches die Funktionsweise des JNZ-Befehles demonstriert. Nur wenn der Akkumulator-Inhalt ungleich 0 ist, d.h. mindestens einer der Schalter S 0 bis S 7 muß auf High-Pegel liegen, wird zum Label L 28A verzweigt, wo der Wert 1 in den Akkumulator geladen wird und über den Port P 3.0 die Leuchtdiode D 0 aktiviert. Liegen alle logischen Pegel der Schalter S 0 bis S 7 auf „low", so findet keine Verzweigung statt. Da der Inhalt des Akkumulators an dieser Stelle 0 ist, werden durch den Sprung zum Label L 28GEM direkt alle LEDs desaktiviert.

Durch Nutzung des DJNZ-Befehls lassen sich mit wenigen Programmzeilen bereits komplette Funktionsabläufe relaisieren

Der kombinierte Vergleichs- und Sprungbefehl CJNE (Compare and Jump if Not Equal) ermöglicht den direkten Vergleich mit einem zweiten Operanten (direkt adressierbare Speicherzelle oder 8-Bit-Konstante). Sind beide Werte ungleich, so führt der Prozessor den Sprung zur angegebenen Adresse durch.

Bei ungleichem Inhalt kann anschließend durch das Carry-Flag geprüft werden, welcher der beiden Werte größer ist. Der Prozessor nimmt dazu eine Subtraktion der Konstanten vom Akkumulator- bzw. Registerinhalt vor. Ist das Ergebnis größer 0, d.h. die Konstante ist kleiner als der Akkumulator- bzw. Registerinhalt, ist das Carry-Flag gelöscht, anderenfalls ist es gesetzt.

Abbildung 92 zeigt das Testprogramm 29, welches die Funktionsweise des CJNE-Befehls demonstriert. Ist der Wert des Port-Inhalts P 1 = 00001111 B (S 0 bis S 3 = aktiv und S 4 bis S 7 = inaktiv), wird die Leuchtdiode D 0 aktiviert. Ist der binäre Wert der Schalter an P 1 größer als der Wert 0 FH, aktiviert das Testprogramm die Leuchtdiode an Port P3.1 (D 1), anderenfalls ist die Leuchtdiode D 2 (P3.2) aktiviert.

Besonders zur Realisierung von Zählschleifen eignet sich der Befehl DJNZ (Decrement and Jump if Not Zero), der das Register oder die direkt adressierbare Speicherzelle decrementiert und zu der angegebenen Adresse springt, wenn das Ergebnis ungleich 0 ist, ohne den Akku-Inhalt zu beeinflussen.

Abbildung 93 zeigt das Testprogramm 30, welches die Funktion des DJNZ-Befehls und die in Kapitel 3.51 angesprochene Unterprogrammtechnik demonstriert. Zunächst werden alle Leuchtdioden am Port P 3 desaktiviert und nach dem Aufrufen des Unterprogrammes L 30 DELAY, welches eine Verzögerung von bis zu 0,5 Sekunden bewirkt, wieder aktiviert und nochmals die Verzögerungsroutine aufgerufen. Durch die permanente Wiederholung dieser Befehlsfolge blinken die Leuchtdioden am Port P 3,

Das Unterprogramm L30DEL1 enthält 2 ineinander geschachtelte Verzögerungsschleifen. Die innere Schleife (L30 DEL1 bis DJNZ R1, L30DEL1) wird genau 256 mal durchlaufen und verbraucht somit 1792 Befehlszyklen.

Die äußere Schleife (abgeschlossen durch den DJNZ R0-Befehl) durchläuft je nach Inhalt von P1 bis zu 256mal die innere Schleife. Je nach Stellung der Schalter an P1 wird somit eine maximale Durchlaufzeit von 256 (P1 = 0) • 1792 = 458.752 Befehlszyklen erreicht. Bei einer Taktfrequenz von 11,0592 MHz beträgt dadurch die maximale Verzögerungszeit ca. 500 msek.

Im zehnten Teil dieser Artikelserie zeigen wir eine übersichtliche Darstellung des MCS-51-Befehlssatzes, gefolgt von der Beschreibung der Timerfunktionen des Mikroprozessors. ELV

```
0203 E590    L28:     MOV   A, P1              ; Inhalt von P1 lesen
0205 7002             JNZ   L28A               ; springe, wenn A <> 0 ist
0207 8002             SJMP  L28GEM             ; gemeinsame Weiterverarbeitung
0209 7401    L28A:    MOV   A, #00000001B      ; Konstante laden
020B F5B0    L28GEM:  MOV   P3, A              ; Ausgabe auf P3;
020D 80F4             SJMP  L28                ; Schleife
```

Bild 91: Testprogramm 28

```
020F E590    L29:     MOV   A, P1              ; Inhalt von P1 lesen
0211 B40F04           CJNE  A, #00001111B, L29A; springe, wenn ungleich 0FH
0214 7401             MOV   A, #00000001B      ; Konstante laden
0216 8008             SJMP  L29GEM             ; gemeinsame Weiterverarbeitung
0218 4004    L29A:    JC    L29B               ; springe, wenn P1 < 0FH ist
021A 7402             MOV   A, #00000010B      ; Konstante laden
021C 8002             SJMP  L29GEM             ; gemeinsame Weiterverarbeitung
021E 7404    L29B:    MOV   A, #00000100B      ; Konstante laden
0220 F5B0    L29GEM:  MOV   P3, A              ; Ausgabe auf P3;
0222 80EB             SJMP  L29                ; Schleife
```

Bild 92: Testprogramm 29

```
0228 75B000  L30:     MOV   P3, #00000000B     ; alle LEDs aus
022B 5134             ACALL L30DELAY           ; Verzögerung um ca. 500 msek.
022D 75B0FF           MOV   P3, #11111111B     ; alle LEDs an
0230 5134             ACALL L30DELAY           ; Verzögerung um ca. 500 msek.
0232 80F4             SJMP  L30                ; Schleife
0234         L30 DELAY:                        ; Unterprogramm Verzögerung
                                                 um ca. 500 msek.
0234 A890             MOV   R0, P1             ; Verz.-Zeit mit P1 bestimmen
0236 7900             MOV   R1, #00H           ; Schleifenwert laden
0238 00      L30DEL1: NOP                      ; 1 Befehl Verzögerung
0239 00               NOP                      ; 1 Befehl Verzögerung
023A 00               NOP                      ; 1 Befehl Verzögerung
023B 00               NOP                      ; 1 Befehl Verzögerung
023C 00               NOP                      ; 1 Befehl Verzögerung
023D D9F9             DJNZ  R1, L30DEL1        ; innere Schleife
023F D8F7             DJNZ  R0, L30DEL1        ; äußere Schleife
0241 22               RET                      ; zurück zum Hauptprogramm
```

Bild 93: Testprogramm 30