Zeitschrift: Bulletin des Schweizerischen Elektrotechnischen Vereins, des Verbandes Schweizerischer Elektrizitätsunternehmen = Bulletin de l'Association Suisse des Electriciens, de l'Association des Entreprises électriques suisses

Band: 68 (1977)

Heft: 6

Artikel: Der Aufbau von numerischer Steuerungen mittels Mikroprozessor und Programmbibliothek

Autor: Müller, A.

DOI: https://doi.org/10.5169/seals-915012

Der Aufbau von numerischen Steuerungen mittels Mikroprozessor und Programmbibliothek

Von A. Müller

681.513.2 : 519.685.7

Numerische Positioniersteuerungen können mit Hilfe handelsüblicher Mikroprozessoren sehr einfach realisiert werden, wenn eine Programmbibliothek zur Verfügung steht. Nach einer Übersicht über das Problem werden die Grundlagen abgeleitet, auf denen eine solche Bibliothek aufgebaut werden kann.

Les commandes numériques de positionnement sont réalisables très simplement au moyen de microprocesseurs courants, à condition de disposer d'une bibliothèque de programmes. Après un aperçu général de ce problème, l'auteur décrit les bases sur lesquelles une telle bibliothèque peut être établie.

1. Einführung

Eine Positioniersteuerung mit Mikroprozessor hat allgemein den in Fig. 1 dargestellten Aufbau. Dieser klare und übersichtliche Aufbau ergibt folgende Eigenschaften:

- Der Mikroprozessor umfasst hauptsächlich standardisierbare, auswechselbare Baugruppen. Nur die Interface-Baugruppen sind problembezogen aufgebaut.
- Eine Erweiterung der Steuerung oder Anpassung an neue Probleme ist im Rahmen der Platzreserve im Kartenrack ohne weiteres möglich.
- Nach Aufstellung des Pflichtenheftes kann gleichzeitig mit der Konstruktion der Hardware und dem Erstellen der Software (des Programmes) begonnen werden. Die Durchlaufzeit wird dadurch verringert, und die Steuerung ist schneller einsatzbereit.
- Die Verwendung von standardisierter Hardware und von Programmen aus einer Bibliothek (standardisierte Software) erleichtern Inbetriebnahme und Service. Auch das Ersatzteilproblem ist einfacher zu lösen.

Wie sieht die Software einer Steuerung aus, deren Programmierung mit Hilfe einer Programmbibliothek erfolgt? Fig. 2 zeigt das Flussdiagramm (Ablaufschema) einer einfachen numerischen Positioniersteuerung. Rechtecke stellen auszuführende Operationen dar; Rhomben markieren Entscheidungen, die auf Grund des Resultates der vorangegangenen Operationen vom Mikroprozessor getroffen werden.

Alle Operationen, zu deren Ausführung mehr als einige wenige Befehle benötigt werden, sollen der Bibliothek entnommen werden können. Die Rechtecke im Flussdiagramm stellen also, mit der erwähnten Einschränkung, Bibliotheksprogramme dar.

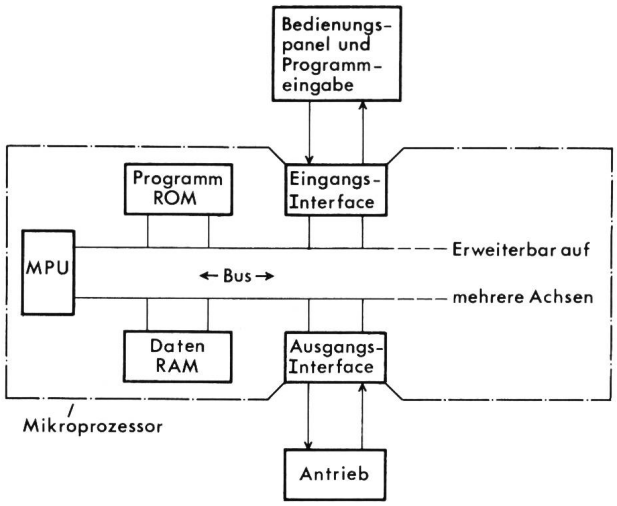

Fig. 1 Aufbau einer Positioniersteuerung mit Mikroprozessor
MPU Microprocessor Unit
RAM Random Access Memory
ROM Read Only Memory

Die Rhomben stehen für Entscheidungen. An diesen Stellen kann das Programm auf zwei verschiedene Arten weitergehen, je nachdem eine Bedingung erfüllt ist oder nicht. Es können dabei nur solche Bedingungen verwendet werden, deren Erfüllung durch ein Zeichen (Flag-Bit) im Zustandsregister (Condition Code Register) des Mikroprozessors signalisiert wird. Die Bibliotheksprogramme müssen also diese Flag-Bits setzen.

Das Hauptprogramm der Steuerung hat die Aufgabe, die richtigen Unterprogramme aufzurufen und die richtigen Entscheidungen zu bewirken. Es besteht nur aus Testoperationen und Aufrufen von Bibliotheksprogrammen. Diese werden durch den Befehl JSR (Jump to Subroutine) aufgerufen. Am Ende jedes Unterprogrammes bewirkt der Befehl RTS (Return from Subroutine) den Rücksprung ins Hauptprogramm.

Der hier dargelegte Aufbau des Programmes aus Unterprogrammen hat folgende wichtige Vorteile:

- Das Hauptprogramm wird sehr übersichtlich und leicht lesbar, da es genau dem Flussdiagramm entspricht.
- Programmänderungen sind auf einfache Art möglich.

2. Eigenschaften des Mikroprozessors

Die zurzeit erhältlichen Mikroprozessoren weisen unterschiedlichen Aufbau auf. Dies hat Rückwirkungen auf die Programmierung. Die folgende Beschreibung bezieht sich auf den Mikroprozessor Motorola M6800. Sie soll das Verständnis der anschliessend gegebenen Programmbeispiele erleichtern.

Das Mikroprozessorsystem hat den in Fig. 3 dargestellten einfachen und übersichtlichen Aufbau. Wichtig ist dabei, dass alle Bausteine an den gleichen Bus angeschlossen sind und dass die Schnittstellen mit der Umwelt (Peripherie-Schnittstellen) vom Prozessor genau gleich wie Speicherstellen behandelt werden. Durch den Bus ist gewährleistet, dass ein System ohne grossen Aufwand jederzeit ausgebaut werden kann. Die Behandlung der Schnittstellen als Speicherstellen ergibt eine einfache Programmierung, da keine speziellen Befehle für Ein- und Ausgabeoperationen nötig sind. Zudem ist auf diese Art eine fast beliebig grosse Anzahl von Schnittstellen möglich.

Der Mikroprozessor selbst weist ebenfalls einen sehr übersichtlichen Aufbau auf (Fig. 4). Die beiden 8-bit-Register A und B sind Rechenregister (Akkumulatoren) mit fast identischen Eigenschaften. In ihnen können alle logischen und alle arithmetischen Operationen ausgeführt werden. Viele Operationen können aber auch direkt in einer Speicherstelle ausgeführt werden. Dies ist beispielsweise für Ein- und Ausgabeoperationen (E/A-Operationen) oft sehr vorteilhaft. Da die Schnittstellen wie Speicherstellen behandelt werden, können viele Operationen direkt im Interface ausgeführt werden. Dies

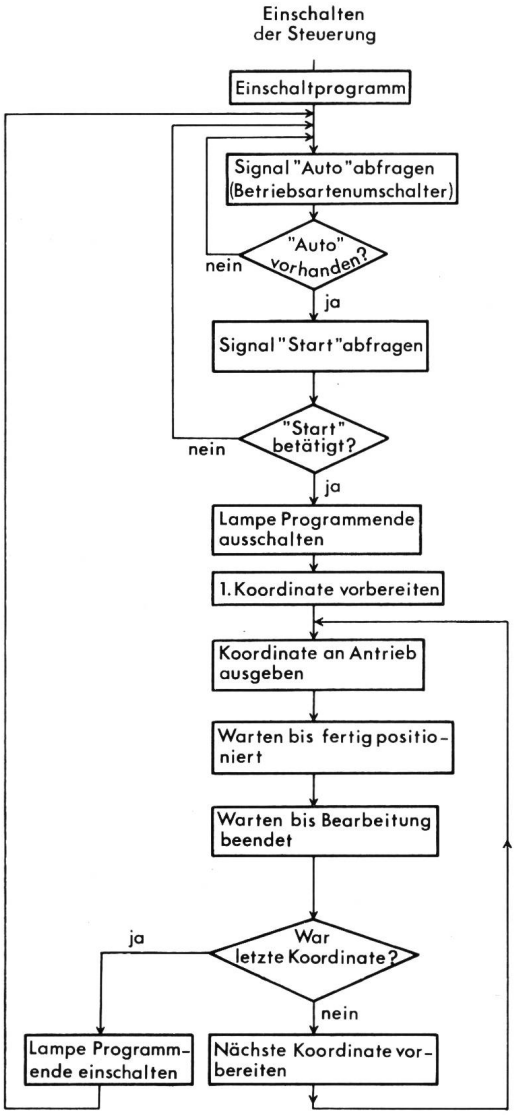

Fig. 2 Aufbau des Hauptprogrammes einer Steuerung

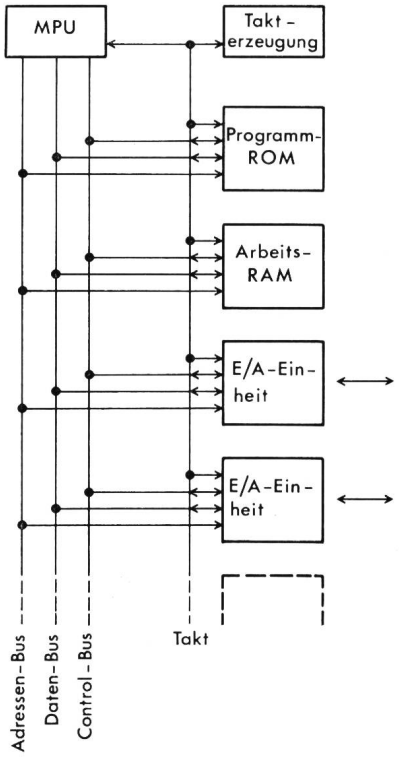

Fig. 3 Aufbau eines Mikroprozessor-Systems M 6800
E/A Eingabe/Ausgabe
MPU Microprocessor Unit
RAM Random Access Memory
ROM Read Only Memory

vereinfacht die Programmierung und verbessert die Lesbarkeit der Programme.

Das Indexregister X (16 bit) dient als Speicher für die Basisadresse bei indexierter Adressierung von Datenspeicherstellen. Diese Adressierungsart ist speziell geeignet, wenn das gleiche Programm mit verschiedenen Daten ablaufen soll. Die Daten werden dann blockweise gespeichert und vor dem Start des Programmes die Anfangsadresse des Datenblockes ins Indexregister gesetzt. Im Programm sind die Adressen der zu verarbeitenden Daten relativ zu dieser Anfangsadresse angegeben. Beispielsweise wird also der Wert in der Anfangsadresse verknüpft mit dem Wert in Anfangsadresse plus 1 und das Resultat abgespeichert in Anfangsadresse plus 2. Ist das Programm abgelaufen, so kann das Indexregister auf die Anfangsadresse des nächsten Datenblockes gesetzt werden, und das gleiche, unveränderte Programm verarbeitet den nächsten Datenblock.

Das Stackpointer-Register S ermöglicht die Speicherung von Zwischenresultaten in einem als «Stack» bezeichneten Teil des Datenspeichers, ohne dass im Programm feste Adressen für diese Speicherstellen vorgesehen werden müssen. Dazu wird im Einschaltprogramm das Stackpointer-Register auf die höchste Adresse des für den Stack reservierten Teiles des RAM gesetzt. Der Befehl Push (PSH) bewirkt dann, dass der Inhalt eines Akkumulators abgespeichert wird in der durch den Inhalt des Stackpointer-Registers gegebene Adresse und anschliessend der Registerinhalt um 1 verringert wird. Das S-Register enthält also stets die Adresse der obersten freien Speicherstelle des Stack. Durch den Befehl Pull (PUL) wird die gegenteilige Operation ausgelöst. Das S-Register wird zuerst um 1 erhöht und dann sein Inhalt als Adresse ausgegeben. Der Inhalt der adressierten Speicherstelle wird in einen der beiden Akkumulatoren eingeschrieben.

Zusätzlich dient der Stack noch als Speicher für die Rücksprungadressen bei Unterprogrammen und als Speicher für alle MPU-Register beim Signal «Interrupt».

Im Zustandsregister CCR (Condition Code Register) sind alle Flag-Bits vereinigt, die einen bestimmten Zustand des Rechners und bestimmte Eigenschaften des Resultates der

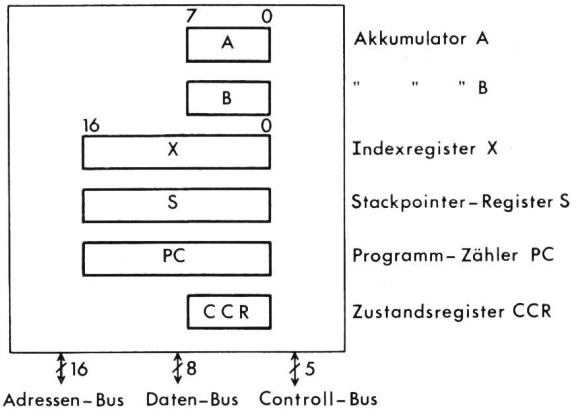

Fig. 4 Schema der Microprozessor Unit (MPU)

letzten ausgeführten Operation signalisieren. Folgende Zustände werden signalisiert:

C–Bit (Carry)	Die letzte Operation, betrachtet als Operation mit positiven Binärzahlen, hat einen Übertrag ergeben.
Z–Bit (Zero)	Das Resultat der letzten Operation war null
N–Bit (Negative)	Das Resultat der letzten Operation, betrachtet als Operation von Binärzahlen in Zweierkomplement-Darstellung, war negativ.
V–Bit (Overflow)	Die letzte Operation, betrachtet als Operation mit Binärzahlen in Zweierkomplementdarstellung, hat einen Übertrag ergeben.

3. Grundlegende Anforderungen an eine Programmbibliothek

Numerische Positioniersteuerungen arbeiten normalerweise mit mindestens 6dekadigen vorzeichenbehafteten Zahlen. Die Programme sollen also für 6dekadige Werte geschrieben sein, aber auf einfache Art auf höhere Dekadenzahlen erweitert werden können.

Die Bibliothek soll Programme enthalten für Dateneingabe, Datenausgabe, Datenverschiebungen, einfache arithmetische Operationen sowie Vergleichsoperationen. Durch die Programme müssen auch die Flag-Bits im Zustandsregister gesetzt werden, damit bedingte Programmverzweigungen im Hauptprogramm möglich sind.

Für komplexe numerische Verarbeitungen, die einfacher in Binärdarstellung erfolgen, müssen die nötigen Umrechnungsprogramme von BCD (Binary Coded Decimal) in Binär und umgekehrt vorhanden sein.

Eine einfache Bibliothek für Steuerungen mit Schrittmotor-Antrieben könnte beispielsweise folgende Programme umfassen:

Ausgabeprogramme:
- Ausgabe einer Schrittzahl zu einem Schrittmotor-Antrieb
- Ausgabe einer Zahl zu einer Kontroll-Anzeige
- Ausgabe von Steuersignalen an Ausgangsrelais und Kontrollampen

Eingabeprogramme:
- Eingabe eines Wertes von einem Dekadenschalter
- Eingabe von einem Lochstreifenleser
- Eingabe von Steuerbefehlen vom Bedienungspanel und vom Eingangsrelais

Verarbeitungsprogramme:
- Laden des Rechenwerkes aus dem Speicher
- Abspeichern des Rechenwerk-Inhaltes im Speicher
- Addition zweier Dezimalzahlen
- Vergleich zweier Dezimalzahlen
- Komplementieren, Inkrementieren, Dekrementieren, Normieren } einer Dezimalzahl
- Umrechnung BCD in Binär und umgekehrt
- Multiplikation und Division in Binär

4. Zahlendarstellung und einige Festlegungen

Für die Darstellung mehrdekadiger Dezimalzahlen in einem 8-bit-Rechner wird normalerweise eine der beiden folgenden Möglichkeiten benützt: Entweder belegt jede Ziffer einen Speicherplatz; in diesem Falle wird die Ziffer meistens durch ihren ASCII-Code (American Standard Code for Information Interchange) dargestellt. Oder aber ein Speicherplatz wird durch zwei Ziffern im BCD-Code belegt.

Die erste Möglichkeit erleichtert das Einlesen von Daten und ihre Anzeige auf alphanumerischen Anzeigen. Als Nachteil ist der gegenüber der zweiten Möglichkeit verdoppelte Bedarf an Speicherplatz zu erwähnen.

In numerischen Steuerungen sind häufig grössere Informationsmengen abzuspeichern. Es wird deshalb hier der zweiten Möglichkeit der Vorzug gegeben, obwohl für die Eingabe und die Ausgabe von Daten eventuell eine Umcodierung ASCII – BCD oder umgekehrt nötig werden kann.

Die Ablage der Zahlen im Speicher erfolgt vorteilhafterweise in einer Reihenfolge, die auf einfache Art eine Eingabe der Werte in die aufeinanderfolgenden Speicherstellen ermöglicht. Da normalerweise zuerst das Vorzeichen eingegeben wird, wird für dieses die Speicherstelle mit der niedrigsten Adresse gewählt. Es folgen die höchstwertigen Dekaden, und die niedrigste Dekade schliesslich steht in der Speicherstelle mit der höchsten Adresse. Fig. 5 zeigt diese Darstellungsform, wobei als Beispiel eine 6dekadige Zahl gewählt wurde. Zu beachten ist, dass im Speicher die führenden Nullen gespeichert werden müssen. Der Wert +35 ist also als +000035 abzuspeichern. Für die Speicherung 8dekadiger Zahlen wird eine Speicherstelle mehr benötigt, also 5 Speicherstellen pro Zahl.

Negative Zahlen werden entweder durch Betrag und Vorzeichen festgehalten oder in Zehnerkomplement-Darstellung (Fig. 6).

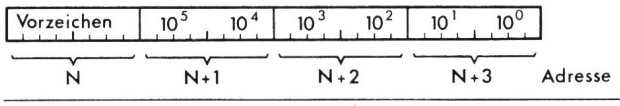

Fig. 5 Darstellung einer 6dekadigen Zahl mit Vorzeichen

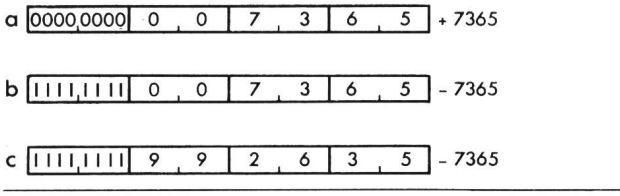

Fig. 6 Darstellung negativer Zahlen
a Positive Zahl
b Negative Zahl als Betrag und Vorzeichen dargestellt
c Negative Zahl als Zehner-Komplement dargestellt

Die arithmetischen Operationen mit den auf diese Art dargestellten 6dekadigen Zahlen müssen in 4 Teiloperationen (je Speicherplatz eine Operation) zerlegt werden. Zuerst werden in 3 Teiloperationen die Ziffern des Resultates berechnet und anschliessend als 4. Teiloperation das Vorzeichen. Diese Teiloperationen werden vorteilhafterweise im Akkumulator A durchgeführt, da in diesem der Befehl DAA (Decimal Adjust Accu A) für die BCD-Korrektur bei Additionen gilt.

Es muss nun noch festgelegt werden, wie die Operanden an die Unterprogramme übergeben werden. Dabei können entweder die Operanden selbst oder deren Adressen übergeben werden. Für die Übergabe der Adressen bietet sich das Indexregister an. Soll der Operand selbst übergeben werden, so müssen 4 Speicherstellen festgelegt werden, in denen die Übergabe erfolgt. Diese 4 Speicherstellen haben die Rolle eines Akkumulators; sie werden deshalb als Pseudo-Akkumulator PA bezeichnet.

Es bestehen also folgende Möglichkeiten zur Übergabe von Werten an die Unterprogramme:

a) Die Adresse des Wertes steht im Indexregister

b) Der Wert selbst steht im Pseudo-Akkumulator

c) Sind zwei Werte zu übergeben, so steht der eine im Pseudo-Akkumulator und die Adresse des anderen im Indexregister. Das Resultat der Verknüpfung der beiden Werte steht im Pseudo-Akkumulator.

Als Pseudo-Akkumulator können entweder 4 feste Adressen, beispielsweise 0000 bis 0003 bestimmt werden, oder es können 4 Stellen des Stack verwendet werden. Die Verwendung des Stack ergibt den Vorteil, dass die Unterprogramme automatisch re-entrant werden: Wird ein Unterprogramm durch einen Interrupt unterbrochen, so ist es im allgemeinen nicht möglich, das gleiche Unterprogramm auch im Interrupt-Serviceprogramm zu verwenden. Werden nämlich feste Speicherstellen für die Speicherung von Zwischenresultaten benützt, so werden diese dann im Interrupt-Serviceprogramm überschrieben und eine Rückkehr (Re-entrance) ins unterbrochene Programm ist nicht mehr möglich. Ein Programm, das re-entrant sein soll, darf zur Speicherung von Zwischenresultaten nur die Register des MPU und den Stack verwenden.

Allerdings ergibt die Verwendung des Stack als PA etwas unübersichtlichere Unterprogramme. Da zudem durch eine spezielle Auslegung des Interrupt-Serviceprogrammes (Abspeicherung des PA im Stack durch das Serviceprogramm) auch bei einem PA mit festen Adressen die gleichen Eigenschaften erzeugt werden können, wurden im vorliegenden Beispiel die Speicherstellen mit den Adressen 0000 bis 0003 als Pseudo-Akkumulator verwendet.

5. Beispiele für Bibliotheksprogramme

5.1 Allgemeiner Aufbau

Die Bibliotheksprogramme bestehen im allgemeinen aus 4 Teilen

1. Verarbeitung eines oder mehrerer Zahlenwerte
2. Verarbeitung des Vorzeichens
3. Setzen der Flag-Bits im Zustandsregister
4. Rücksprungbefehl

Teil 3 verlängert die Unterprogramme wesentlich. Es kann deshalb vorteilhaft sein, für die gleiche Operation zwei Unterprogramme zu besitzen, nämlich mit und ohne Teil 3. Ein Vergleich der beiden ersten Programmbeispiele zeigt, welcher Aufwand zum Setzen der Flag-Bits nötig ist.

5.2 Beispiel: Datentransfer vom Speicher zum PA, ohne Flag-Bits

Die Adresse des Vorzeichens der auszulesenden Speicherstellen steht im Indexregister.

LDAA 3, X	(Load Accu A)
STAA 03	Transfer der niedrigsten Dekaden (Store Accu A)
LDAA 2, X	
STAA 02	Transfer der mittleren Dekaden
LDAA 1, X	
STAA 01	Transfer der höchsten Dekaden
LDAA 0, X	
STAA 00	Transfer des Vorzeichens
RTS	Rücksprungbefehl

Dieses Programm belegt 17 Worte im Programmspeicher und hat bei 1 MHz Taktfrequenz eine Laufzeit von 41 µs.

5.3 Beispiel: Datentransfer vom Speicher zum PA, mit Flag-Bits

Die Basisadresse der auszulesenden Speicherstellen steht im Indexregister. Die Inhalte der Akkumulatoren A und B werden durch dieses Programm verändert.

CLRB	B nullen (Clear B)
LDAA 3, X	
STAA 03	Transfer der niedrigsten Dekaden
BEQ *+3	nächsten Befehl überspringen, falls Zahl 0 war (Branch if equal to zero)
INCB	B inkrementieren
LDAA 2, X	
STAA 02	Transfer der mittleren Dekaden
BEQ *+3	nächsten Befehl überspringen, falls Zahl 0 war
INCB	B inkrementieren
LDAA 1, X	
STAA 01	Transfer niedrigste Dekaden
BEQ *+3	nächsten Befehl überspringen, falls Zahl 0 war
INCB	B inkrementieren
LDAA 0, X	
STAA 00	Transfer Vorzeichen
BEQ *+4	nächsten Befehl überspringen, falls Vorzeichen +
ORAB #$ 80	Bit 7 in B setzen (Logisches «Oder» des Accu B mit hexadezimaler Zahl 0000 0000)
TSTB	setzt Flags N und Z im Zustandsregister (Test B)
RTS	Rücksprungbefehl

Der Akkumulator B dient hier um festzustellen, ob die transferierte Zahl 0 oder negativ ist. Am Anfang wird B null gesetzt. Nur wenn die transferierte Zahl 0 ist, werden alle 3 Befehle INCB übersprungen und B bleibt auf null. Ist das Vorzeichen der transferierten Zahl negativ, so wird das Bit 7 von B gesetzt Der Befehl TSTB prüft den Inhalt von B, ob er null oder negativ (Bit 7 = 1) ist und setzt das Zustandsregister entsprechend. Dieses Programm belegt 32 Worte im Programmspeicher und hat eine Laufzeit von 69 µs.

5.4 Beispiel für ein Eingabe-Unterprogramm: Einlesen eines Zeichens von einem Lochstreifenleser

Der Streifenleser steht im Ruhezustand auf dem noch nicht gelesenen Zeichen. Gleichzeitig mit dem Ablesen wird ein Impuls von ca. 2 ms Dauer ausgegeben, der den Leser um ein Zeichen weiterschaltet. 8 ms nach dem Ablesen eines Zeichens kann das nächste Zeichen abgelesen werden.

Es wird vorausgesetzt, dass die Einlesezeit nicht kritisch ist, so dass die Zeitintervalle mit einem Software-Timer erzeugt werden können. Das Unterprogramm WAIT2 erzeugt ein Intervall von 2 ms, WAIT6 ein solches von 6 ms.

Die Leserdaten werden über einen Peripheriebaustein mit der Adresse LEDATA eingelesen. Die Ausgabe des Weiterschaltimpulses erfolgt über Bit 3 eines Peripheriebausteines mit der Adresse LECTRL. Am Ende des Programmes steht das gelesene Zeichen in A und das Flag-Bit C ist 0 bei gerader Parität bzw. 1 bei ungerader Parität des Zeichens. Ungelochte Stellen und überlochte Stellen (alle 8 Spuren gelocht) werden überlesen. Die Inhalte von B und X werden durch das Programm verändert.

```
LEZCHV  LDAA LEDATA   Gelesenes Zeichen in A
        PSHA          Gelesenes Zeichen im Stack
                      speichern
        LDAB LECTRL
        ORAB #$ 08    Bit 3 = 1 setzen
        STAB LECTRL   Beginn Weiterschaltimpuls
        JSR WAIT2     Timer 2 ms (Jump to subroutine)
        LDAB LECTRL
        ANDB #$ F7    Bit 3 = 0 setzen
        STAB LECTRL   Ende Weiterschaltimpuls
        JSR WAIT6     Timer 6 ms
        PULA          Gelesenes Zeichen wieder in A
        CMPA #$ FF    Alle Spuren gelocht? (Compare)
        BEQ LEZCHV    falls ja, zurück zum Anfang, neues
                      Zeichen lesen
        CMPA #$ 00    ungelocht?
        BEQ LEZCHV    falls ja, zurück zum Anfang, neues
                      Zeichen lesen
        PSHA          Gelesenes Zeichen im Stack
                      speichern
        LDX #$ 0008   Loop counter für Paritäts-
                      bestimmung
        CLRB          B nullen
        RORA          Bit 0 von B geht in C
        ADCB #$ 00    C zu B addieren
        DEX           Loop counter abzählen
        BNE *-4       zurück zu RORA, solange X nicht 0
        RORB          Bit 0 von B geht in C
        PULA          gelesenes Zeichen wieder in A
        RTS           Rücksprungbefehl
```

Zur Bestimmung der Parität des Zeichens in A werden die 8 Bit dieses Zeichens nacheinander in C geschoben und zu B addiert. Bit 0 von B sagt, ob die Anzahl der 1 von A gerade oder ungerade ist. Dieses Bit 0 wird in C eingeschoben.

6. Das Hauptprogramm

Wie erwähnt, besteht das Hauptprogramm prinzipiell aus Aufrufen von Unterprogrammen und aus Entscheidungen. In Unterprogrammen sollen keine Entscheidungen vorkommen, die den Ablauf der Steuerung beeinflussen.

Auch das Hauptprogramm kann oft aus Teilprogrammen zusammengesetzt werden. Ähnlich wie bei Unterprogrammen ist jedem dieser Teilprogramme eine bestimmte Funktion der Steuerung zugeordnet. Ob eine Funktion durch ein Teilprogramm oder durch ein Unterprogramm realisiert wird, hängt hauptsächlich von zwei Faktoren ab, einerseits ob der Programmteil Entscheidungen enthält, die den Funktionsablauf beeinflussen, anderseits ob der gleiche Programmteil bei einem einmaligen Funktionsablauf mehrmals benötigt wird.
Mehrmaliges Auftreten des gleichen Programmteiles spricht für einen Ablauf als Unterprogramm, während das Auftreten von den Ablauf beeinflussenden Entscheidungen einen Aufbau als Teilprogramm vorteilhaft erscheinen lässt.

7. Zusammenfassung

Nach einem kurzen Überblick über den Aufbau von numerischen Steuerungen mit Mikroprozessoren wurden einige Überlegungen zum Aufbau einer Programmbibliothek für solche Steuerungen angestellt. Auf einfachen Nenner gebracht lautet das Resultat:

Das Programm aufteilen in möglichst viele Unterprogramme, von denen jedes eine bestimmte Funktion der Steuerung erzeugt. Diese Programme können mit Ihrer Funktion benannt werden. Dadurch wird die Programmdokumentation lesbar, da der Aufruf eines Unterprogrammes gleichbedeutend wird mit dem Auslösen einer bestimmten Funktion. Die Verwendung von Bibliotheksprogrammen reduziert die Programmierungskosten ganz wesentlich.

Adresse des Autors
Dr. sc. techn. *Arno Müller*, Omni Ray AG, Oberwiesenstrasse 4, 8304 Wallisellen.