ABC Einchip-Mikrorechner 1

Einchip-Mikrorechner sind mehr als nur Mikroprozessoren. Sie enthalten zusätzlich Speicher- und Peripherie-Baugruppen. Das gestattet den Aufbau von Mikrorechnern mit extrem geringem gerätetechnischen Aufwand, was besonders bei Prozeßrechnern eine große Rolle spielt. Daher findet man sie als Steuerung kommerzieller Technik (z. B. in Tastaturen), aber auch in Konsumgütern (z. B. Waschautomaten). Nicht zu vergessen ist der JU+TE-Selbstbau-Computer, der in diesen Kreis hineingehört.
Unser ABC soll mit dieser Technik (U 881 und dessen Varianten) bekanntmachen und nebenbei weitere Nutzungsmöglichkeiten des JU+TE-Computers zeigen. Dazu werden zunächst Funktionsweise und Aufbau erklärt, danach der Befehlssatz und einige Methoden dessen effektiver Anwendung. Den Abschluß bilden Beispiele zur Beschaltung von Einchip-Mikrorechnern.

1. Funktionsweise

Einchip-Mikrorechner sind Digitalrechner. Sie erfüllen alle Aufgaben durch die Ausführung von Programmen. Diese bestehen aus Befehlen, die recht elementare Operationen im Einchip-Mikrorechner bewirken. Sie werden Schritt für Schritt nacheinander ausgeführt. Der Einchip-Mikrorechner besitzt also eine sequentielle Arbeitsweise; alles, was er tut, braucht seine Zeit.
Der Programmierer muß für die Lösung jeder Aufgabe ein Programm entwickeln. Es besteht aus sinnvoll aneinander gereihten Befehlen, die den Lösungsalgorithmus widerspiegeln. Meist werden diese Befehle maschinennah formuliert, in Maschinen- oder Assemblersprache. Dieses Niveau gestattet am besten, die Leistungsfähigkeit des Einchiprechners auszunutzen. Aber auch höhere Programmiersprachen wie BASIC (vgl. JU+TE-Computer) und FORTH lassen sich nutzen, wenn die damit verbundene Reduzierung der effektiven Rechenleistung in Kauf genommen werden kann. Dafür spart man Zeit bei der Programmentwicklung, denn es sind problemnahe Formulierungen möglich. Deren Übersetzung in den Maschinencode erledigt ein universell nutzbares Programm (BASIC-Interpreter bzw. FORTH-Compiler), das zusätzlich zum Anwenderprogramm im Speicher stehen muß. Oft läßt sich ein Optimum finden, indem Programme teils maschinennah und teils in einer höheren Sprache entworfen werden.
Für das Aufbewahren von Programmen und Daten benötigt der Einchip-Mikrorechner wie jeder Digitalrechner Speicher. Das Ausführen der einzelnen Befehle erledigt die zentrale Verarbeitungseinheit (central processing unit, CPU), die auch das Zusammenspiel aller Baugruppen steuert. Für den Informationsaustausch mit der äußeren Umgebung benötigt der Einchiprechner Peripherieschaltungen. Besonders beim Einsatz als Prozeßrechner sind auch Zeitgeber (timer) zur Anpassung des Programmablaufs an die Prozeßdynamik und serielle Interfaces (SIO) zur Rechnerkopplung als periphere Baugruppen wichtig.
Die Kommunikation der Komponenten eines Rechners untereinander erfolgt über den Bus (Verbindungsleitungen). **Abb. 1** zeigt den grundsätzlichen Aufbau eines Mikrorechners. Zu einer CPU gehören immer mehrere Peripheriebaugruppen und viele Speicherzellen.
Der Einchip-Mikrorechner enthält bereits all diese Baugruppen in begrenzter Anzahl in einem Schaltkreis. Besonders die peripheren Komponenten sind sehr universell ausgelegt. Ihre spezielle Funktion bestimmt das Programm mit dem Laden einiger Steuerregister. Auch Erweiterungen mit externen Speicherschaltkreisen und zusätzlichen Peripheriebaugruppen können auf diese Weise vereinbart werden. Die Rechenleistung eines UB 8810 D läßt sich mit der des UA 880 D

(4-MHz-Version der CPU aller 8-Bit-Büro-, -Personal- und -Kleincomputer aus DDR-Produktion) vergleichen. Neben der CPU enthält ein Einchiprechner jedoch weitere Baugruppen, was insgesamt etwa 30 000 Transistorfunktionen in einem Schaltkreis erfordert. Das sind fast doppelt so viel, wie die 16-Bit-CPU U 8001 D enthält.

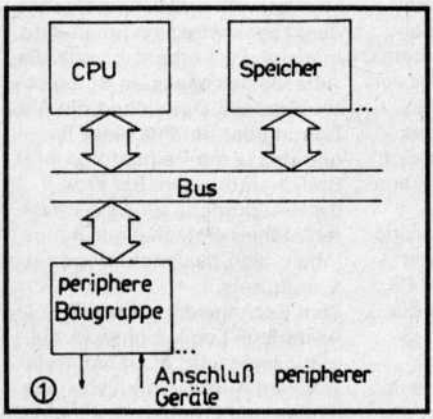
Äußerlich zeigt sich die Funktion des Einchiprechners an seinen Anschlüssen. Von den 40 Pins sind vier für Spannungsversorgung (GND, VCC) und Takterzeugung (XTAL1, XTAL2) reserviert. Die restlichen teilen sich in 32 Ein-Ausgabe- und vier Steuersignale. Teilweise werden beim Anschluß externer Speicherschaltkreise maximal 17 Ein-Ausgabe-Pins für den äußeren Bus verwendet. **Abb. 2** enthält das Impulsdiagramm für das Lesen von und das Schreiben in externe Speicherzellen.

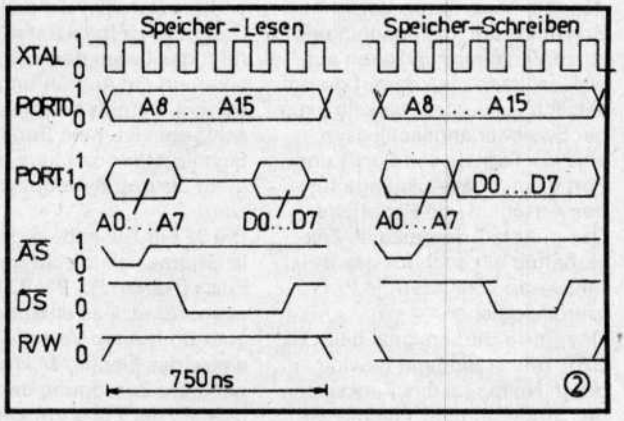
XTAL bezeichnet den vom internen Taktgenerator erzeugten oder von außen zugeführten Grundtakt, dessen Frequenz standardmäßig 8 MHz beträgt. Der interne Systemtakt (4 MHz) geht durch 1:2-Teilung daraus hervor. Er dient als Zeitbasis für alle Abläufe.

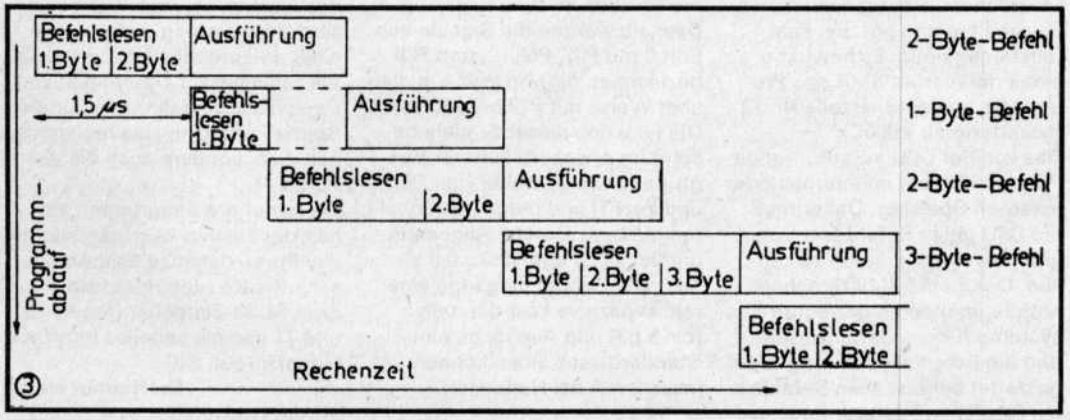
$\overline{\textbf{AS}}$ (address strobe) ist ein Steuersignal, das während der Adreßausgabe über Port 1 Nullpegel führt. Diese Informationen sind mit Sicherheit zu dem Zeitpunkt gültig, zu dem der 0-1-Übergang von $\overline{\text{AS}}$ erfolgt. Dadurch eignet sich dieses Steuersignal für das Takten von Auffang-Flipflops zum Aufbewahren der über Port 1 ausgegebenen Adreßbits während des gesamten Speicherzyklus. Die meisten Typen von Speicherschaltkreisen erfordern dieses Zwischenspeichern.

Port 1 und **Port 0** sind jeweils acht zusammengehörige Ein-Ausgabe-Signale. Sie bilden den Bus beim Anschluß externer Speicher. Port 0 gibt hierbei während des gesamten Zyklus die höheren acht Adreßbits (A8 bis A15) aus, während Port 1 zunächst der Adreßausgabe (A0 bis A7), danach jedoch der Daten-Ein- oder -Ausgabe (D0 bis D7) dient. Diese Mehrfachnutzung nennt man auch Multiplexen.

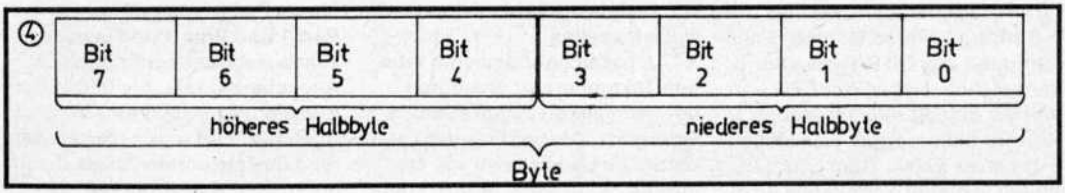
Den Zeitpunkt des Datenaustauschs mit dem Speicher kennzeichnet das Steuersignal $\overline{\textbf{DS}}$ (data strobe) mit Nullpegel. Beim Lesen wird dieses Signal früh aktiv, um dem angeschlossenen Speicherschaltkreis viel Reaktionszeit zum Bereitstellen des adressierten Datenbytes (Signale D0 bis D7 an Port 1) zu lassen. Beim Schreiben dagegen erfolgt







Zeichnungen: Haase, Grabowski



das Aktivieren erst, wenn die über Port 1 ausgegebenen Daten mit Sicherheit gültig sind.
Das Signal **R/W** (read/write) kennzeichnet Lesezyklen mit Einspegel, Schreibzyklen dagegen mit Nullpegel. Es wird statisch erzeugt und enthält dadurch keine Zeitsteuerfunktion wie etwa $\overline{DS}$. Ein Speicherzyklus dauert mindestens 750 ns und erlaubt den Speicherschaltkreisen eine Verzögerung (Adreßzugriffszeit) von etwa 400 ns. Notfalls kann programmtechnisch das Einfügen einer Wartezeit von 250 ns bei jedem Zugriff auf externe Speicher vereinbart werden. Bei höher integrierten Schaltkreisen, die ohnehin relativ kurze Verzögerungszeiten aufweisen, kann man darauf grundsätzlich verzichten. Ist kein externer Speicher angeschlossen, sind die Signale von Port 1 und Port 0 Ein- oder Ausgänge für den Anschluß von Peripherie.
Das in Abb. 2 dargestellte Zeitverhalten gilt auch für das Befehlslesen vom internen Programmspeicher.
Das vierte Steuersignal heißt $\overline{RESET}$. Dieser Eingang bewirkt durch Nullpegel das Rücksetzen der Steuerung des Einchip-Mikrorechners. Das ist nach jedem Einschalten und ggf. bei Fehlfunktionen nötig. Es bewirkt u. a. einen definierten Start des Programms ab Speicherzelle Nr. 12 (hexadezimal: %000C).
Das von der CPU auszuführende Programm steht im internen oder externen Speicher. Daher muß die CPU jeden Befehl erst lesen, um ihn dann ausführen zu können. Der Einchip-Mikrorechner enthält im Inneren getrennte Bussysteme für das Befehlslesen und die Befehlsausführung. Das gestattet bei fast allen Befehlen ein Pipelining (siehe **Abb. 3**), bei dem während der Befehlsausführung bereits der nächste Befehl gelesen wird. Dieses Verfahren gestattet trotz des relativ langsamen Speicherzugriffs (vgl. Abb. 2) eine hohe Rechengeschwindigkeit. Nur bei Ladebefehlen bezüglich des externen Speichers und bei Programmsprüngen blockiert die Ausführung das Lesen des nächsten Befehls.
Durch das Pipelining werden 1-Byte-Befehle nicht schneller ausgeführt als 2-Byte-Befehle. Um die Ausführungsfolge nicht zu stören, hat das Lesen des nächsten Bytes (1. Byte des folgenden Befehls) die doppelte Dauer. Bei 3-Byte-Befehlen erfolgt das Lesen des dritten Bytes, während die Ausführung des vorherigen Befehls bereits abgeschlossen ist. Kein Befehl des Einchip-Mikrorechners benötigt mehr als drei Byte Speicherkapazität.
Die 32 Ein-Ausgabe-Signale sind in Gruppen zu je acht den vier Ports (Häfen) P0, P1, P2 und P3 zugeordnet. Das entspricht der acht Bit breiten Verarbeitungsweise des Einchip-Mikrorechners. Die Zuordnung der Bits innerhalb der Ports entspricht deren Position im Byte (vgl. **Abb. 4**). Deshalb werden die Signale von Port 0 mit P07, P06, ... und P00 bezeichnet, die von Port 1 in gleicher Weise mit P17 bis P10 usw. Die Funktion dieser Signale besteht im Anschluß äußerer Baugruppen wie Speicher (nur Port 0 und Port 1) und periphere Schaltungen bzw. Geräte. Allgemein gilt der TTL-Pegel (0:0...0,8 V, 1:2...5 V), wobei Eingänge eine rein kapazitive Last darstellen (ca. 5 pF) und Ausgänge eine Standardlast treiben können (max. 2 mA bei Nullpegel).

2. Aufbau

Zur CPU des Einchip-Mikrorechners gehören das Steuer- und das Rechenwerk. Zum Steuerwerk zählt neben der allgemeinen Steuerung auch der Befehlszähler. Diese Baugruppen realisieren die sequentielle programmgesteuerte Arbeitsweise und koordinieren das Einbeziehen der anderen Baugruppen zur Befehlsausführung.
Der Befehlszähler (programm counter, PC) ist ein 16-Bit-Register, das stets die Adresse des als nächstes zu lesenden Befehls enthält. Er adressiert den Speicher beim Befehlslesen. Nach jedem Lesen wird sein Inhalt automatisch um 1 erhöht, so daß die Adresse des nächsten Befehlsbytes entsteht. Das sichert die Ausführung der Befehle eines Programms in der Reihenfolge ihrer Speicheradressen. Bei Programmsprüngen erhält der Befehlszähler einfach einen neuen Inhalt: die Speicheradresse des Sprungziels.
Zum Rechenwerk zählen die Arithmetik-Logik-Einheit (arithmetic logic unit, ALU) und Register zum Aufbewahren von Operanden und Ergebnissen. Die ALU des Einchip-Mikrorechners realisiert Verknüpfungsbefehle mit einer entsprechenden Binärlogik. Als Speicher für Operanden und Ergebnisse sind ihr nicht nur die Spezial- und Universalregister zugänglich, sondern auch die vier Ports.
Als periphere Baugruppen enthält der Einchip-Mikrorechner die vier Ports, denen je acht Anschlußstifte zugeordnet sind, zwei 14-Bit-Zeitgeber (timer) T0 und T1 und ein serielles Interface (serial in/out, SIO).

Dr. Helmut Hoyer

(wird fortgesetzt)

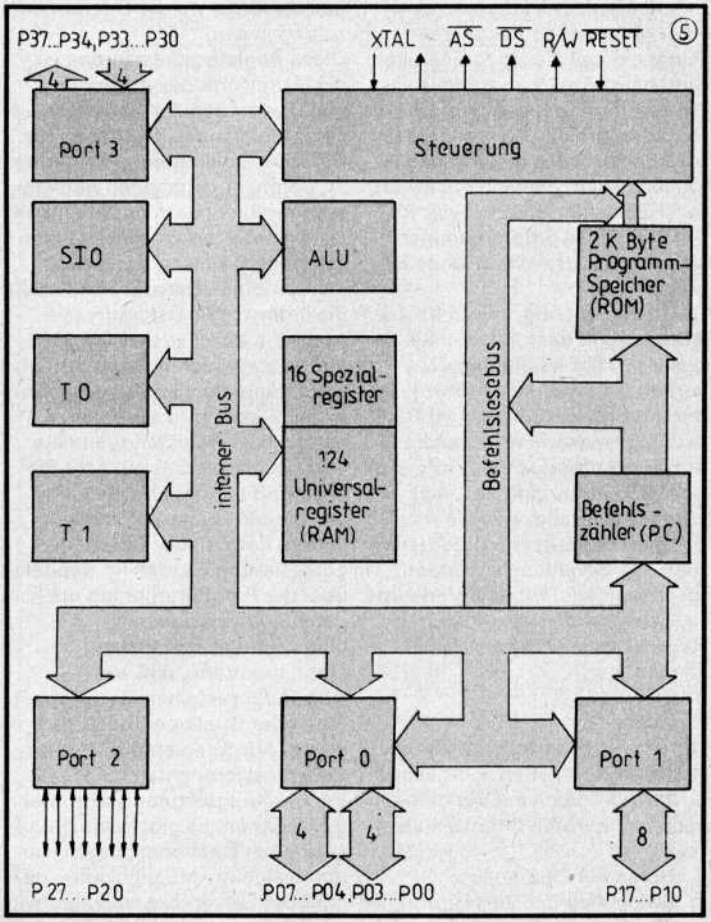
(Fortsetzung zu 2.)

Abb. 5 gibt eine Übersicht des umfangreichen Innenlebens. Den internen Programmspeicher enthalten nur 40polige Varianten des Einchip-Mikrorechners (z. B. UB 8810 D und UB 8830 D). Alternativ gibt es 64polige Gehäuse, bei denen unter den 24 zusätzlichen Signalen die Daten- und Adreßbits zum Anschluß eines externen EPROM anstatt des internen ROM herausgeführt sind. Diese Typen (z. B. UB 8820 M und UB 8840 M) eignen sich für den Aufbau von Programm-Entwicklungs- und -Test-Modulen (Emulatoren). Natürlich werden sie auch verbreitet als Prozeßrechner verwendet, wenn dieser Programmspeicherbereich von 2 bzw. 4 KByte bereits ausreicht.

2.1. Register

Ein Register des Einchip-Mikrorechners dient dem Aufbewahren eines Bytes (vgl. Abb. 4). Es enthält für jedes Bit ein Flipflop, das die Zustände 1 oder 0 einnehmen kann. Solch ein Flipflop kann vom Datenbus aus geladen werden und seinen Inhalt wieder auf den Datenbus geben. **Abb. 6** enthält solch eine Struktur als Logikschaltung. Die beiden Freigabesignale für das Schreiben eines Bits in das sowie das Lesen aus dem Flipflop werden von der internen Steuerung gebildet.
Für jede der acht Datenleitungen des internen Bus besitzt ein Register solch eine Struktur. Die Freigabesignale steuern die acht Flipflops parallel. Wenn das Schreibsignal 1-Pegel besitzt, übernimmt das Register die 8-Bit-Information vom Bus. Solange hier jedoch 0-Pegel vorliegt, bleibt diese Information im Register gespeichert. Auch das Lesen, bei dem das andere Freigabesignal 1-Pegel besitzen muß, verändert den Dateninhalt des Registers nicht. Solange das Freigabesignal Lesen passiv (0) bleibt, beeinflußt das Register den Datenbus nicht, es bleibt selbst passiv.

Da jeder Informationstransport bei der Befehlsausführung über die acht Datenleitungen des internen Bus stattfindet, sind dort natürlich alle Register und weitere Baugruppen (Abb. 5) parallel angeschlossen. Die Steuerung gewährleistet, daß stets nur ein Lese-Signal und ein anderes Schreib-Signal aktiv sind. Damit bestimmt sie einen Sender und



einen Empfänger, während alle anderen Register nichts tun, als ihren Dateninhalt zu bewahren.
Wie **Abb. 7** zeigt, gibt es bei den Registern im Einchip-Mikrorechner eine große Auswahl. Neben den 16 speziellen und den 124 universellen werden auch die vier Ports intern als Register behandelt. Hier gibt es natürlich schaltungstechnische Unterschiede, die später eine Rolle spielen werden. Bei der Ausführung eines Befehls benutzt der Einchip-Mikrorechner jeweils nur einzelne Register. Wie findet er sie aus dem großen Angebot heraus?
Zu diesem Zweck besitzt jedes Register eine Adresse. Aus Gründen der einfachen technischen Realisierbarkeit ist auch das eine Dualzahl. Sie besitzt acht Stellen und ließe daher 256 Register unterscheiden. Wegen des ungenutzten Bereichs kommen aber nur 144 Adressen zur Anwendung, ein Teil dieser Straße blieb unbebaut. **Abb. 7** enthält die Adreßzuordnung des internen Registerbereichs. Wie allgemein üblich wurde das hexadezimale Zahlensystem gewählt, da es mit einer Ziffer (0...9, A...F) vier Bit darstellt. Zwei Ziffern genügen bereits zur Darstellung eines Bytes.
Es ist sehr wichtig, zwischen der Adresse und dem Inhalt eines Registers zu unterscheiden. Das verhält sich wie die Anschrift und der Inhalt des Briefkastens. Die fest zugewiesene Adresse dient nur zum Auffinden einer Information. Der veränderliche Inhalt wird als Operand, also als Ausgangsinformation für die Ausführung von Befehlen verwendet.
Insgesamt sind für eine Verknüpfungsoperation eigentlich drei Register zu spezifizieren:
1. Operand,
2. Operand,
Ergebnis.
Um den erforderlichen Aufwand in Grenzen zu halten, wird ausgenutzt, daß solch eine Verknüpfung in mehreren Schritten abläuft:
1. Holen der Operanden,
2. Berechnen der Verknüpfung,
3. Abspeichern des Ergebnisses.

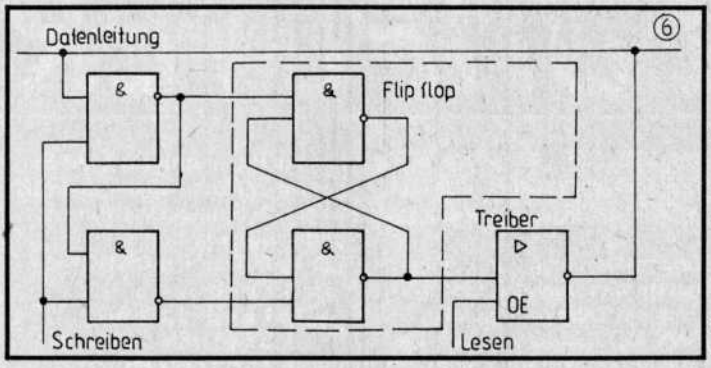
Das Abspeichern des Ergebnisses erfolgt beim Einchip-Mikrorechner einfach in das Register, dem der erste Operand entnommen wurde. Ihn nennt man daher den Zieloperanden (destination, dst.), den zweiten dagegen Quelloperand (source, src.). Es werden also Ziel- und Quelloperand verknüpft und das Ergebnis anschließend in das Zielregister geladen (geschrieben).
Damit muß ein Befehl nur noch zwei Registeradressen enthalten, der Einchip-Mikrorechner ist eine Zweiadreßmaschine. Ein Additionsbefehl besteht z. B. aus drei Byte. Das erste ist der Operationscode, der die Steuerung veranlaßt, eine Addition auszulösen. Ihm folgen zwei weitere Bytes, die Quelladresse und die Zieladresse. Diese beiden Zahlen sagen der CPU, wo die Operanden zu finden sind.
Diese Registeradressierung ist die Hauptform der Auffindung von Operanden. Es gibt z. B. auch Operationscodes, bei denen die CPU den Quelloperand nicht dem im Befehl angegebenen Register entnimmt, sondern dessen Inhalt als Adresse des Quelloperanden verwendet. Diesen Satz sollte man zweimal lesen. Er beschreibt die indirekte Registeradressierung. Die Adresse des Quelloperanden steht hier nicht im Befehl, sondern im dort adressierten Register. Dabei wird ausgenutzt, daß sowohl Registerinhalte als auch Registeradressen acht Bit breit sind und Inhalte als Adressen dienen können. Diese und andere Formen der Operandenadressierung werden im Kapitel über die Programmierung noch genauer beschrieben.
Eine wichtige Feststellung besteht aber darin, daß alle in **Abb. 7** dargestellten Register Ziel- oder Quelloperanden speichern, Adreßregister bei indirekter Adressierung darstellen und Verknüpfungsergebnisse aufnehmen können. Es gibt keine grundsätzlichen Einschränkungen, wodurch sich die Möglichkeiten gegenüber Einadreßmaschinen (wie UB 880 D) erheblich erweitern.
Auch für die Daten-Ein- und -Ausgabe über die vier Ports und die Behandlung der Spezialregister bedarf es keiner besonderer Befehle.
Man kann die Register zu vielen Zwecken gebrauchen, aber nicht zum Speichern von Programmen. Hierfür wird der interne ROM oder externe Speicher (zusätzliche Schaltkreise) benötigt. ROM und EPROM sind Festwertspeicher, deren Inhalt durch die CPU nicht beeinflußt werden kann.
Für ein Bit genügt bereits der in **Abb. 6** rechts dargestellte Treiber, dessen Dateneingang nicht mit einem Flipflop, sondern fest mit 0 oder 1 beschaltet ist. Das entscheidet bereits die Herstellung des Schaltkreises oder bei EPROM ein Programmiervorgang (Brennen) mittels eines speziellen Gerätes. Der viel geringere Aufwand je Bit gestattet eine entschieden größere Anzahl von Speicherzellen (so heißen hier die Register). Das ist auch nötig für das Aufbewahren des praktisch erforderlichen Umfangs an Programmen.
Die Spezialregister sind den verschiedenen Baugruppen des Einchip-Mikrorechners zugeordnet. Sie dienen u. a. dem Festlegen der speziellen Funktionsweise innerhalb der vielfältigen Möglichkeiten. Diese Spezialregister mit Steuerfunktion müssen nach jedem Rücksetzen ($\overline{\text{RESET}}=0$) durch ein initialisierendes Programm neu eingestellt werden. Ihre genaue Beschreibung erfolgt im ABC Einchip-Mikrorechner beim Behandeln der betreffenden Baugruppen.

2.2. Ports

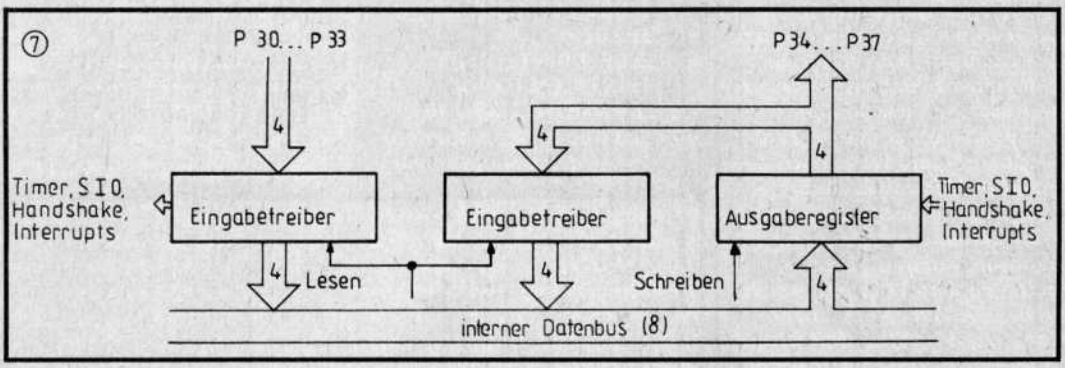
Die vier Ports sind die Häfen des Einchip-Mikrorechners. Die Handelsware besteht aus Informationen. Das sind aus der Sicht der inneren Baugruppen wie beschrieben Operanden bzw. Ergebnisse von Befehlsausführungen, aus der Sicht der Peripherie dagegen Ein- und Ausgabe-Daten. Port 3 besitzt vier Eingänge

(P30, P31, P32 und P33) sowie vier Ausgänge (P34, P35, P36 und P37). Bei den anderen Ports kann jedes der acht Pins Eingang oder Ausgang sein.

Zu Port 3 **(Abb. 8)** gehören vier Flipflops, die den Bits 4 bis 7 zugeordnet sind. Sie speichern die Ausgabeinformationen, bis sie neu beschrieben werden. Diese vier Bits liegen auch an den Anschlußstiften P34 bis P37 im TTL-Pegel vor, sofern die Port-3-Ausgänge nicht Ausgabefunktionen

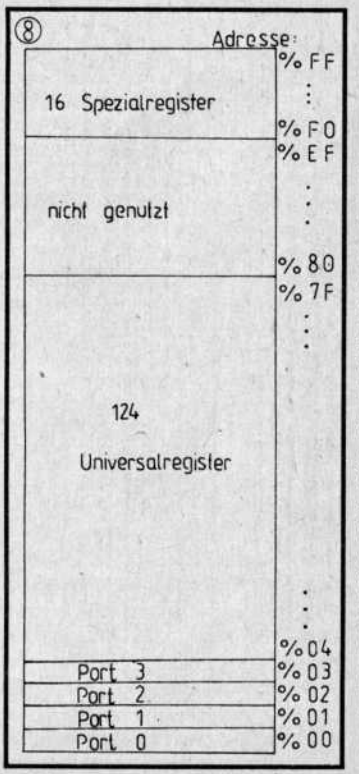


Zeichnungen: Grabowski



der Zeitgeber (Timer), des Serieninterface (SIO), der Handshake-Steuerung oder der Speicherverwaltung erfüllen. In diesen Fällen wird die ins Ausgaberegister eingetragene Information an den Pins nicht wirksam. Der Programmierer wählt mit dem Laden von Steuerregistern, ob P34 bis P37 als normale Ausgänge, die den Inhalt des Registers mit der Adresse 3 (%03) widerspiegeln, oder mit spezieller Funktion arbeiten.

Beim Lesen von Registeradresse 3 erhält man stets die aktuelle Belegung aller Port-3-Anschlüsse, im niederen Halbbyte die der Eingänge P30 bis P33, im höheren die der Ausgänge, gleichgültig, welche Funktionen erfüllt werden. Nebenbei läßt sich aber vereinbaren, daß Eingänge außerdem besondere Wirkungen haben. Sie können die Timer, das Serieninterface oder die Handshake-Steuerung bedienen sowie Interrupts auslösen. Das sehen wir später. Von einem Universalregister unterscheidet sich



das Port 3 also dadurch, daß es nur vier Flipflops besitzt und mit den Anschlußstiften P30 bis P37 sowie verschiedenen Baugruppen verbunden ist.

Diese besonderen Funktionen wählt der Programmierer mit dem Laden des Spezialregisters %F7. Es ist das Port-3-Steuerregister P3M (Port 3 Mode Register, **Abb. 9**). Die hier eingeschriebene Zahl bewirkt mit den einzelnen Bits folgendes:

D0 beeinflußt die Ausgänge von Port 2. Steht hier eine 1, hat Port 2 normale Ausgänge, die sowohl 0- als auch 1-Pegel erzeugen können. Eine 0 in diesem Bit sperrt dagegen die den 1-Pegel erzeugenden pull-up-Transistoren. Dadurch entstehen open-drain-Ausgänge (wie open collector), die mit der äußeren Beschaltung ein verdrahtetes UND bzw. ODER realisieren lassen. Das Bit D1 hat keine Wirkung. Die anderen sechs Bit steuern die paarweise einander zugeordneten Port-3-Signale.

Dr. Helmut Hoyer

Fortsetzung zu 2.2. Ports

D2 entscheidet über P32 und P35. Eine 0 bewirkt die Funktion als normale über Registeradresse 3 handhabbare Ein- und Ausgänge, eine 1 macht sie zu Handshakesignalen des Ports 0. D5 steuert in gleicher Weise P31 und P36, die für Port 2 das Handshake realisieren können. Sind diese Bits mit D5 = 0 als normale Ein- und Ausgänge definiert, können sie durch entsprechende Vereinbarungen in den Spezialregistern der Timer auch dort verwendet werden.
D4 und D3 steuern P33 und P34. Die Kombinationen 00 und 11 wirken wie bei den bereits beschriebenen Steuerbits. Die anderen beiden machen P33 zum normalen Eingang, während P34 das Signal $\overline{DM}$ (data memory) ausgibt. Es gestattet als zusätzliches Adreßbit den äußeren Speicher in zwei Bänke zu teilen: den externen RAM und den externen ROM. Diese Möglichkeit erlaubt, 124 KByte direkt zu adressieren, und zahlt sich bei speicherintensiven Anwendungen des Einchip-Mikrorechners aus.
D7 und D6 bestimmen die Verwendung des seriellen Interface (SIO). Ist D6 mit 0 belegt, bleibt die SIO ungenutzt, P30 und P37 sind normale Ein- und Ausgänge. Eine 1 in D6 aktiviert das serielle Interface mit P30 als Eingang RxD und P37 als Ausgang TxD. D7 entscheidet in diesem Fall, ob acht Datenbits ohne Paritätsbit oder sieben Datenbits plus Paritätsbit übertragen werden. Näheres dazu enthält der später folgende Abschnitt über das Serieninterface.
Die **Abb. 10** zeigt die Struktur der Ports 0, 1 und 2. Sie enthalten für jedes Bit ein Flipflop für die Eingabe und eins für die Ausgabe. Beim Schreiben auf die Adresse 0, 1 bzw. 2 wird das betreffende Ausgaberegister vom Datenbus geladen. Spezialregister entscheiden, welche Bits davon zu den Anschlüssen weitergeleitet werden und welche Ausgabetreiber passiv bleiben. So erfolgt das Auswählen von Pins für die Ein- oder Ausgabe. Im Grunde sind alle Portpins Eingänge, einige aber wegen aktiver Ausgabetreiber außerdem Ausgänge.
Die Eingaberegister dieser Ports werden von der Handshakelogik gesteuert. Enthält das Steuerregister P3M (%F7) eine Belegung, die kein Handshake für das betreffende Port vereinbart, folgt das Eingaberegister stets dem Logikpegel an den Anschlußstiften. Ist jedoch Handshake festgelegt, übernimmt das Eingaberegister, während RDY = 1 und $\overline{DAV}$ = 0 gilt. Ansonsten hält es die zuletzt eingetragene Information **(Abb. 11)**.
Unter Handshake versteht man die Taktsteuerung bei parallelem Datenaustausch. Der Einchip-Mikrorechner verarbeitet die beiden nötigen Steuersignale über Port-3-Pins (vgl. Abb. 9). RDY = 1 (ready: bereit) kennzeichnet die Übernahmebereitschaft des Datenempfängers. Mit $\overline{DAV}$ = 0 (data available: Daten verfügbar) erklärt der Sender seine Datenausgänge für gültig. Die Handshakelogik kann Sender oder Empfänger realisieren, je nach Übertragungsrichtung des betreffenden Ports. Bei Dateneingabe realisiert der zugeordnete Port-3-Eingang $\overline{DAV}$, der Port-3-Ausgang dagegen RDY. Bei Datenausgabe gilt die umgekehrte Zuordnung. Damit eignet sich dieses Signalpaar zum Steuern der parallelen Kopplung von Einchip-Mikrorechnern untereinander und zum Anschluß parallel verarbeitender Peripherie (z. B Lochbandgeräte). Wegen der Möglichkeit einer seriellen Übertragung kommt die Handshakelogik in der Praxis sehr selten zur Anwendung.
Die Eingabetreiber verbinden das jeweilige Eingaberegister mit dem internen Datenbus, wenn von der Registeradresse 0, 1 bzw. 2 gelesen wird. Sofern kein Handshake stattfindet, widerspiegelt das die aktuell an der Pins liegende Information.
Alle Port-2-Signale lassen sich unabhängig voneinander in Ein- oder Ausgaberichtung vereinbaren. Das erledigt der Programmierer mit dem Laden des Steuerregisters P2M (%F6, **Abb. 12).** Dessen Flipflops steuern die acht Ausgabetreiber von Port 2 einzeln. Eine 0 bewirkt hier die Freigabe des Treibers der entsprechenden Bitposition und damit auch die Vereinbarung der Ausgaberichtung. Ob dabei die pull-up-Transistoren aktiv werden, entscheidet wie beschrieben das Bit 0 des P3M (%F7). Eine 1 im P2M (%F6) erklärt das entsprechende Port 2 – Signal zum Eingang.
Port 0 und Port 1 können nicht nur als Ein- und Ausgabesignale, sondern auch als externes Bussystem (Daten- und Adreßbus) fungieren. Damit gibt es die drei alternativen Nutzungsmöglichkeiten Ausgabe, Eingabe und Bus, die für alle Bits von Port 1 gemeinsam, für Port 0 Halbbyteweise festzulegen sind. Dies geschieht zusammen mit zwei weiteren Entscheidungen bezüglich der Speichernutzung durch Laden des Spezialregisters P01M (%F8, **Abb. 13).**
D1 und D0 lassen für das niedere Halbbyte von Port 0 unter den

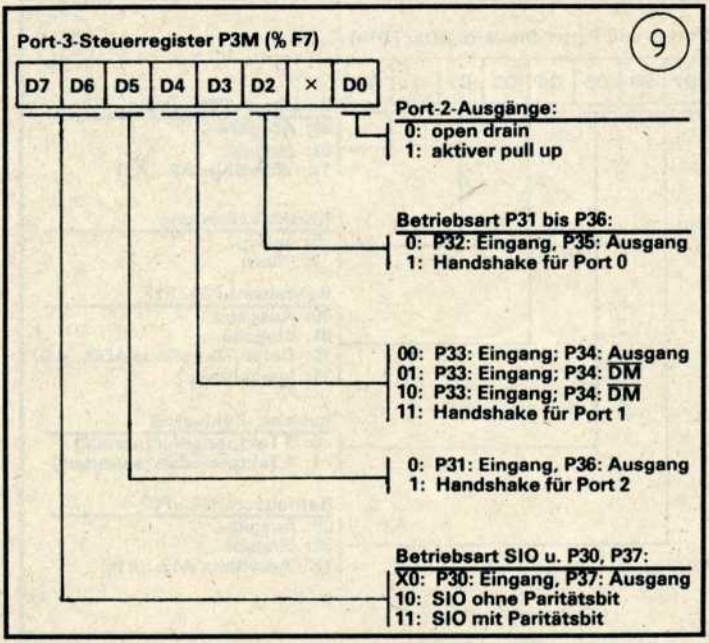
drei genannten Alternativen wählen, D7 und D6 entsprechend für das höhere Halbbyte. D2 entscheidet, ob der Stapelspeicher (stack) in den inneren Universalregistern oder in extern angeschlossenem RAM angelegt wird. Ihn braucht die CPU als Operativspeicher zum Ablegen von Rückkehradressen bei Unterprogrammsprüngen. Darauf kommen wir später noch zu sprechen. Mit D5 kann, sofern nötig, die Zykluszeit beim Zugriff zum externen Speicher von 750 ns auf 1 µs (8-MHz-Takt) erweitert werden.

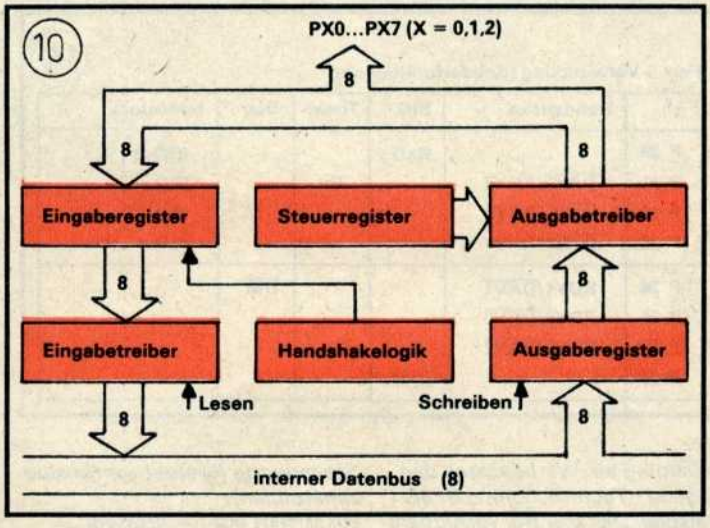
D4 und D3 bestimmen die Nutzung von Port 1. Neben den drei Alternativen gibt es hier die Möglichkeit, mit der Belegung 11 den gesamten äußeren Bus zu passivieren. Dabei werden die Steuersignale $\overline{AS}$, $\overline{DS}$, R/W sowie alle Port-1-Pins und die für Adreßausgabe vereinbarten Port-0-Signale in den hochohmigen Zustand gebracht. Diese Initialisierung gestattet die Busverwaltung durch einen anderen Prozessor (direkter Speicherzugriff, DMA). In der Praxis kommt das Nutzen dieser Möglichkeit höchst selten zur Anwendung.

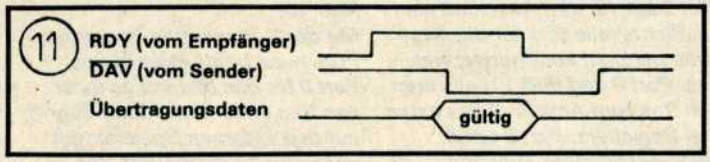
Das Steuerregister P01M (%F8) gestattet, den Einchip-Mikrorechner an verschiedene Kapazitäten externer Speicher anzupassen. Die Möglichkeiten reichen vom Arbeiten ohne externen Speicher (Port 0 und Port 1: Ein-/Ausgabe) über 4 KByte (Port 1 und P00 bis P03: Bus) bis 62 KByte (Port 0 und Port 1: Bus) externen Speichers. Durch Nutzen von P34 als $\overline{DM}$ (vgl. Abb. 9) verdoppeln sich diese Kapazitätsgrenzen.

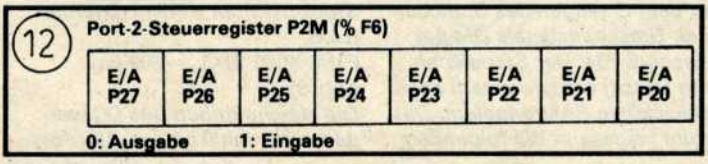
Die Portbits, die den externen Bus bilden, lassen sich weder durch Schreiben auf die Registeradresse beeinflussen noch lesen. Die **Abb. 14** gibt eine Übersicht der möglichen Port-3-Sonderfunktionen, die zum Teil später noch genauer erklärt werden. Zunächst wenden wir uns jedoch einem Beispiel zu:

Das Initialisieren der Portsteuerregister paßt den Einchip-Mikrorechner an dessen konkrete Be-

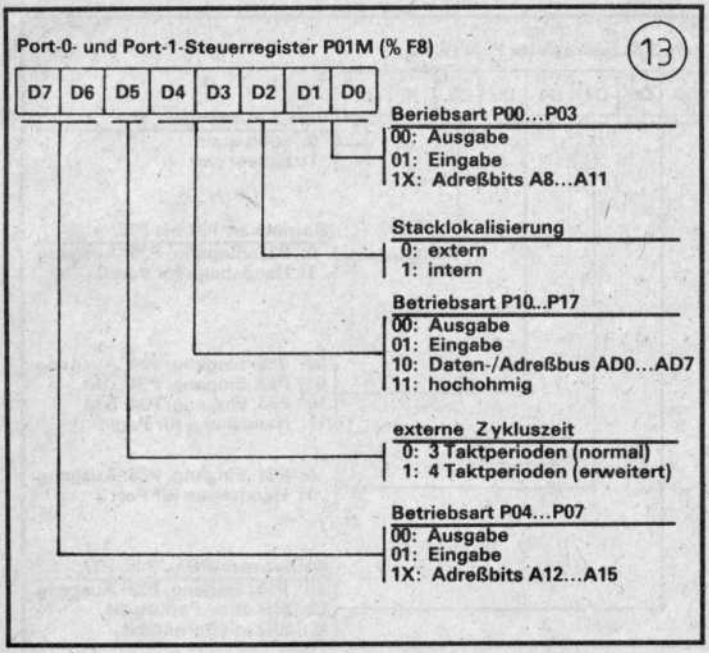








Zeichnungen: Haase



14

Port-3-Verwendung (Sonderfunktionen)

	Handshake	SIO	Timer	Bus	Interrupts
P 30		RxD			IRQ 3
P 31	$\overline{DAV2}$/RDY2		Tin		IRQ 2
P 32	$\overline{DAV0}$/RDY0				IRQ 0
P 33	$\overline{DAV1}$/RDY1				IRQ 1
P 34	RDY1/$\overline{DAV1}$			$\overline{DM}$	
P 35	RDY0/$\overline{DAV0}$				
P 36	RDY2/$\overline{DAV2}$		Tout		
P 37		TxD			

schaltung an. Wir benutzen den Jugend+Technik-Computer als Beispiel. Das hier frei verfügbare Port 2 soll für den Anschluß eines Lauflichts (alle Bits für die Ausgabe benutzt) konfiguriert werden. Port 0 und Port 1 realisieren den Bus zum Anschluß des externen Speichers. Port 3 erfüllt keine besonderen Funktionen außer dem Erzeugen des Speicherbank-Umschaltsignals $\overline{DM}$ am Anschluß P34. Der Stapelspeicher (stack) wird im extern angeschlossenen RAM angelegt. Das ergibt insgesamt die folgenden Steuerworte (% steht vor Hexadezimalzahlen):

P01M:1001 0010 = %92 (vgl. Abb. 13)

Mit der Kombination 10 werden Port 1 und beide Hälften von Port 0 für das Bilden des externen Bus initialisiert. Beim Zugriff auf den externen Speicher gilt normales Zeitverhalten, der Stack befindet sich im externen RAM.

P3M: X000 10X1 = %09 (vgl. Abb. 9)

Die gleichgültigen Bits (X) werden meist mit 0 belegt. Alle Port-3-Signalpaare sind normale Ein- bzw. Ausgänge, nur P34 dient als $\overline{DM}$. Die Bits 3 und 4 könnten auch mit 10 belegt werden. Für die Ausgänge von Port 2 bewirkt die Belegung des P3M aktive Pull-up-Transistoren.

P2M: 0000 0000 = %00 (vgl. Abb. 12)

Alle Port-2-Signale werden als Ausgänge konfiguriert. Das Initialisieren (Register %F6 bis %F8) und die Ein- und Ausgabe (Register 0 bis 3) erfolgen in Maschinen- bzw. Assemblersprache mit Ladebefehlen. Aber auch die höhere Sprache TINY-MP-BASIC des Einchip-Mikrorechners U 883 bietet mit der Prozedur SETR und der Funktion GETR die Möglichkeit, die Ports zu nutzen. Die Anweisungsfolge

```
PROC SETR [%F8,%92];
PROC SETR [%F7,9];
PROC SETR [%F6,Ø]
```

realisiert unser Beispiel. Das Anwenden dieser Prozedur auf die Registeradressen 0 bis 3 lädt die Ausgaberegister der Ports. Bei Port 0 und Port 1 bleiben solche Operationen hier jedoch wirkungslos, da sie den externen Bus bilden. Bei Port 3 setzen sich nur die Bits 5, 6 und 7 auf die Pins durch, da P30 bis P33 stets Eingänge sind und P34 das Signal $\overline{DM}$ ausgibt. Solange der JU+TE-Computer das Fernsehbild erzeugt, werden auch P36 und P37 alle 64 µs vom Betriebssystem beeinflußt, so daß auch hier Ausgaben praktisch wirkungslos bleiben.

Das Eingeben gelingt beim TINY-MP-BASIC mit der Funktion GETR. So erhält man mit GETR[2] die letzte Ausgabe von Port 2 zurück und mit GETR[3] die aktuelle Belegung aller acht Port-3-Anschlüsse. Die Funktion GETRR[2] nutzt die 16-Bit-Verarbeitungsbreite des BASIC aus und erzeugt im höheren Byte (Bits 8 bis 15) die Port-2-Belegung, im niederen (Bits 0 bis 7) dagegen die von Port 3. Eingaben von Port 0 und Port 1 ergeben beim JU+TE-Computer auf allen Bits 1-Belegung

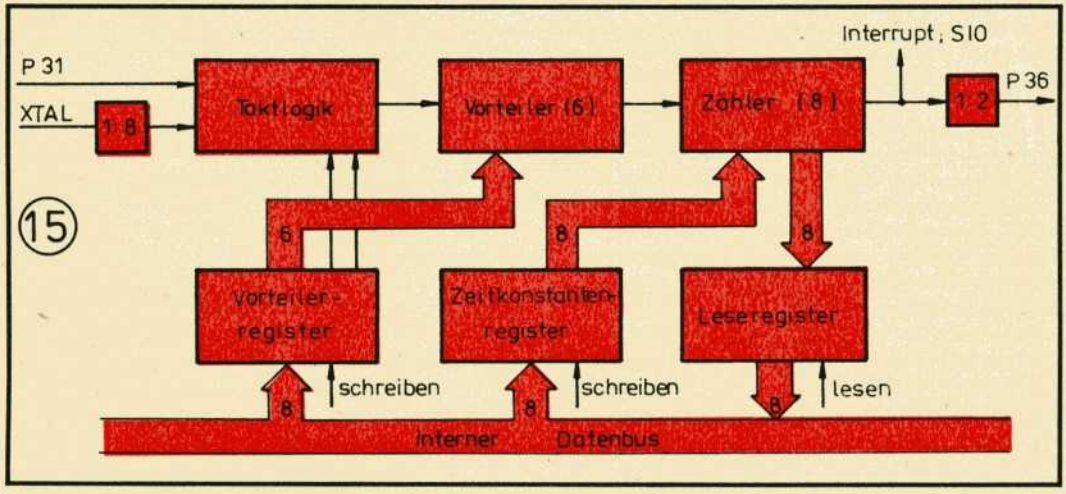
Dr. Helmut Hoyer

2.3. Zeitgeber

Es gibt Aufgaben, die ein Mikrorechner sehr regelmäßig ausführen muß. Dazu gehören die Ausgabe von Fernsehbildern, das Bearbeiten serieller Ein- und Ausgaben (z. B. Fernschreiber-Anschluß) und das Erzeugen von Taktsignalen für Peripheriegeräte (auch akustische Ausgabe). Die beiden Zeitgeber des Einchip-Mikrorechners können die Zeitbasis für solche Operationen bilden und für weitere Aufgaben wie Zählen externer Ereignisse und Impulsbreitenmessung verwendet werden.

Die beiden Zeitgeber T0 und T1 gewinnen ihre Ausgangssignale durch Frequenzteilung des Taktes XTAL. Abb. 15 zeigt die dabei wirksame Zählerkette und die drei zugehörigen Spezialregister. 14 Bit lassen sich programmtechnisch steuern. Das ergibt bei 8-MHz-Takt (XTAL) am Zählerausgang einstellbare Periodendauern von 1 µs bis 16,384 ms. Beim Nutzen eines Zeitgebers für die Ausgabe einer Frequenz über P36 wird eine weitere Zählstufe wirksam, so daß die doppelten Werte gelten (2 µs bis 32,768 ms) und ein 1:1-Tastverhältnis entsteht. Das Verwenden von P31 als Tin (Timer-Input von T1) und P36 als Tout (Timer-Output von T0 oder T1) läßt sich programmtechnisch vereinbaren, die Zeitgeber funktionieren aber auch ohne diese Portpins.

Sowohl Vorteiler als auch Zähler arbeiten als Rückwärtszähler. Beginnend vom Startwert (auch als Zeitkonstante bezeichnet) wird der Zählerstand bei jedem Eingangsimpuls um Eins verringert, solange der betreffende Zeitgeber aktiv ist. Beim Erreichen des Zählerstandes Null beginnen die Vorteiler automatisch wieder mit ihrem Startwert. Am Ausgang entsteht dabei jeweils ein Eingangsimpuls für den Zähler. Bei 8-MHz-Takt (XTAL) liegt die Periodendauer hier zwischen 1 µs und 64 µs. Für den Zähler läßt sich vereinbaren, ob er beim Erreichen von Null stehen bleibt (single pass = einmaliger Durchlauf) oder wie der Vorteiler von vorn beginnt (modulo n = automatischer Neustart). Das Erreichen des Zählerstandes Null löst eine Interruptanmeldung aus oder erzeugt einen Taktimpuls für das serielle Interface (SIO). Insgesamt fünf Spezialregister steuern die beiden Zeitgeber. Abb. 16 stellt das Vorteilerregister von T0 dar. D0 bestimmt hier die Betriebsart von T0. Ist dieses Bit mit 0 belegt, bleibt T0 stehen, sobald der Zählerstand Null erreicht ist. Andernfalls erfolgt automatischer Neustart, so daß sich der gesamte Zählablauf periodisch wiederholt. Das Register PRE0 (%F5) kann wie auch PRE1 (%F3) nicht gelesen werden. Die Adresse %F4 ist zwei Spezialregistern zugeordnet. Beim Schreiben erreicht man das Zeitkonstantenregister, das Lesen hat den Inhalt des Leseregisters, also den aktuellen Zählerstand zum Ergebnis (Abb. 17). Als kleinste Zahl läßt sich die Zeitkonstante 1 einstellen. Werden alle Bits des Registers %F4 mit 0 geladen, entsteht die Zeitkonstante 256. So können alle ganzzahligen Startwerte zwischen 1 und 64 (Vorteiler) bzw. 256 (Zähler) vereinbart werden. T0 arbei-



tet stets unabhängig von P31 mit dem externen Takt XTAL als Basis.
Der Zeitgeber T1 ist über die Adressen %F3 und %F2 in gleicher Weise wie T0 erreichbar, nur daß P31 in die Steuerung einbezogen werden kann (Abb. 18 und 19). Bei 1-Belegung von D1 im Vorteiler 1-Register PRE1 (%F3) spielt P31 für die Funktion des Zeitgebers T1 keine Rolle. In diesem Fall verwendet T1 wie T0 den externen Takt XTAL als Zeitbasis. Bei 0-Belegung des Bits D1 in PRE1 erhält P31 die Funktion Tin. Das ist allerdings nur dann möglich, wenn im P3M (%F7) kein Handshake für Port 2 vereinbart wurde. Worin die Funktion Tin besteht, entscheidet die Belegung der Bits D4 und D5 im Zeitgeber-Steuerregister TMR (Timer Mode Register).
Abb. 20 zeigt das Zeitgeber-Steuerregister, das die Benutzung von P31 und P36 spezifizieren und beide Zeitgeber steuern läßt. Die Freigabebits D1 und D3 sind mit Flipflops besetzt, die den Betriebszustand der Zeitgeber widerspiegeln (1: aktiv, 0: passiv). Das Schreiben einer 1 auf diese Positionen startet den jeweils zugeordneten Zeitgeber oder läßt einen bereits aktiven im Betriebszustand. Das Schreiben einer 0 stoppt dagegen oder erhält den Ruhezustand.
Die Bits D0 und D2 bewirken beim Schreiben einer 1 das Laden des betreffenden Zählers mit dem Inhalt des Zeitkonstantenregisters. Dieser Befehl kann beliebig mit dem Einstellen des Freigabebits kombiniert werden. Er wirkt aber nur kurzzeitig. Die Bits D0 und D2 sind nicht mit Flipflops besetzt. Der nächste Eingangsimpuls verringert bei freigegebenem Zeitgeber bereits wieder den Zählerstand. Sofern Handshake für Port 2 vereinbart wurde, ist die Belegung des höheren Halbbytes von TMR wirkungslos. Sonst stehen P31 und P36 als Ein- und Ausgang der Zeitgeber zur Verfügung. Die Vereinbarungen für P31 mit den Bits D4 und D5 bleiben jedoch gleichgültig, sofern durch D1 = 1 im PRE1 (vgl. Abb. 18) P31 nicht als Tin aktiviert wurde. Andernfalls kann P31 den Zeitgeber T1 auf vier verschiedene Weisen steuern. Die Belegung 00 der Bits D4 und D5 des TMR (Abb. 20) schaltet P31 auf den Takteingang des Vorteilers. T1 zählt in dieser Betriebsart die so zugeführten Impulse statt des geschalteten Systemtaktes, solange deren Frequenz höchstens XTAL/8 (1 MHz) beträgt.
In der Tor-Betriebsart (D5 = 0, D4 = 1) arbeitet der Zeitgeber T1 mit XTAL als Zeitbasis, solange P31 mit 1 belegt ist. Bei P31 = 0 bleibt die Zählerkette stehen. Damit eignet sich diese Betriebsart gut zum Messen von Impulsbreiten. Es wird hierbei einfach abgezählt, wieviel XTAL/8-Perioden während des 1-Pegels an P31 auftreten. Die Auflösung kann dabei je nach Vorteiler-Startwert zwischen 1 µs und 64 µs gewählt werden. Im Kapitel über die Interruptverarbeitung folgt ein Beispiel, wie die Interruptanmeldungen beim Abschalten von P31 (1-0-Flanke) und beim Überlauf des Zählers von T1 zur effektiven Programmierung solch eines Meßproblems ausgenutzt werden können.
Die beiden Trigger-Betriebsarten lassen den Start des Zeitgebers T1 zum Zeitpunkt der 1-0-Flanke an P31 vereinbaren. Retriggerbar heißt, daß dies auch während des Zählvorgangs als Neustart und nicht nur aus dem Ruhezustand erfolgen kann. Praktisch haben diese beiden Möglichkeiten selten einen Wert.
Die Belegung von D7 und D6 im TMR (Abb. 20) bestimmt die Nutzung von P36 als Tout. Die Belegung 00 läßt P36 von den Zeitgebern unbeeinflußt. 01 und 10 vereinbaren diesen Anschluß dagegen als Ausgang entsprechend der Darstellung in Abb. 15. Die Belegung 11 bewirkt die Ausgabe des internen Takts (XTAL/2) unabhängig von der Funktion der Zeitgeber. Zur Programmierung dieser beiden Peripherie-Baugruppen folgt nun ein Beispiel:

Der Zeitgeber T0 soll alle 64 µs einen Interrupt anmelden, um die Synchronimpulse für die Bildausgabe auszulösen. Solange an P31 1-Pegel anliegt, ist die Ausgabe des Kammertons a' über P36 mit dem anderen Zeitgeber (T1) zu realisieren. Meistens gibt es mehrere Möglichkeiten, Vorteiler und Zähler einzustellen, da das Produkt der einzelnen Teilungsverhältnisse das Gesamtergebnis bildet. Die 64 µs von T0 sollen mit einem Vorteiler von 1:64 und einem Zähler von 1:1 realisiert werden. Die Ausgabe des Kammertons erfordert eine etwas kompliziertere Überlegung:
Die geforderte Frequenz beträgt 440 Hz. Daraus folgt die Periodendauer:
T = 1/440 Hz = 2,2727 ms
Wie in Abb. 15 gut zu erkennen ist, wird P36 von einem 1:2-Teiler gesteuert. Ausgangs des Zählers muß daher die halbe Periodendauer anliegen:
2,2727/2 = 1,1363 ms (ca. 1136 µs)
Der Zeitgeber kann nicht genauer als 1 µs auflösen, da sein Eingangstakt 1 MHz beträgt. Die Zahl 1136 muß nun als Produkt von Vorteiler-Startwert und Zeitkonstante realisiert werden. Der Startwert des Vorteilers kann maximal 64, der des Zählers 256 betragen. So ist 8 × 142 = 1136 ein geeignetes Produkt. Mit diesen Zahlen ergeben sich die folgenden Steuerworte:
PRE0: 0000 00X1 = %01 (vgl. Abb. 16)
Als Startwert erhält dieser Vorteiler die Zahl 0, so daß er 1:64 teilt. Es wird automatischer Neustart von T0 vereinbart.
T0: 0000 0001 = %01 (vgl. Abb. 17)
Da der Vorteiler bereits den 64-µs-Takt erzeugt, muß der Zähler auf 1:1 eingestellt werden.
PRE1: 0001 0001 = %11 (vgl. Abb. 18)
Dieser Vorteiler realisiert ein Teilungsverhältnis von 1:8. Es werden automatischer Neustart von T1 und das Nutzen von P31 als Tin vereinbart.
T1: 1000 1110 = %8E (vgl.

Abb. 19, JU+TE 11/1988)
Der Berechnung entsprechend beträgt die Zeitkonstante von T1 (dezimal) 142.
TMR: 1001 1111 = %9F (vgl. Abb. 20)
P36 dient als Ausgang von T1. P31 steuert diesen Zeitgeber in der Tor-Betriebsart. Das heißt, daß T1 so lange aktiv ist, wie P31 mit 1-Pegel beschaltet ist. Beide Zeitgeber werden mit ihrer Zeitkonstanten geladen und gestartet.

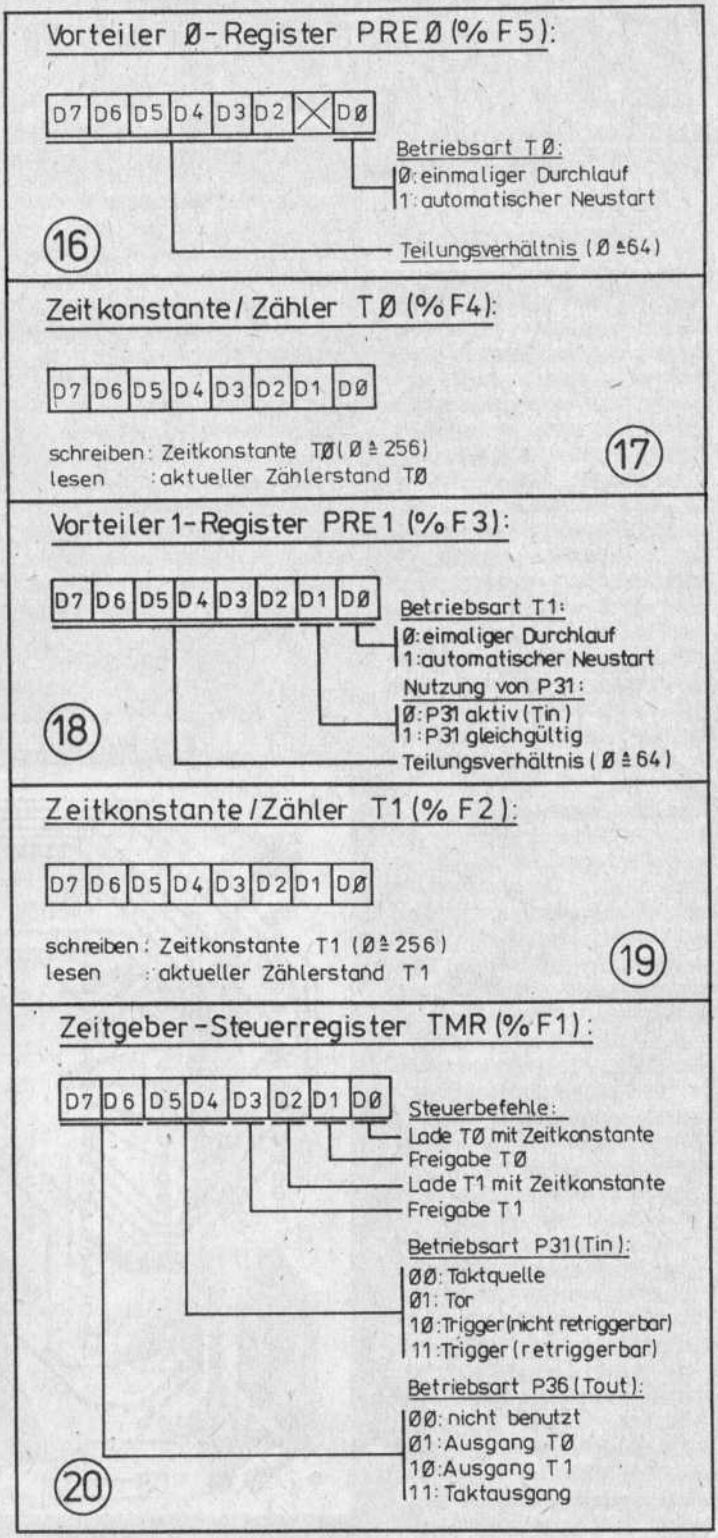
Die Programmiersprache TINY-MP-BASIC gestattet den Zugriff zu den Zeitgebern mit der Prozedur SETR sowie der Funktion GETR in gleicher Weise, wie es bezüglich der Port-Nutzung beschrieben wurde. Das Betriebssystem des JU+TE-Computers nutzt den T0-Interrupt zur Bilderzeugung, so daß dort nur T1 uneingeschränkt der freien Verfügung offen steht.

2.4. Serielles Interface

Das serielle Interface überträgt die acht Bits eines Bytes nicht parallel (wie die Ports) sondern nacheinander über eine Leitung. Das verringert den Aufwand für Verbindungskabel und läßt auf einfache Weise der Störunterdrückung dienende Potentialtrennung (z. B. mit Optokopplern) zu. Daher wird diese serielle Datenübertragung zunehmend nicht nur für die Rechnerkopplung, sondern auch für den Anschluß von peripheren Geräten (Drukker, Tastatur) an Büro- und Personalcomputer genutzt.

Die SIO-Baugruppe (serial input/output) des Einchip-Mikrorechners enthält einen Empfänger und einen Sender für das allgemein übliche asynchrone 8-Bit-Format. Hier wird der Bit-Takt (Schiebetakt) nicht übertragen, statt dessen zur Synchronisierung aber ein Startbit vorangestellt und nach den Datenbits eine Übertragungspause (Stoppbits) von mindestens zwei Bitzeiten angefügt. **Dr. Helmut Hoyer**

Zeichnungen: Liebig



(Fortsetzung zu 2.4. Serielles Interface)

Abb. 21 zeigt dieses Format als Zeitdiagramm. Die drei zusätzlichen Bits bewirken, daß insgesamt elf Bitzeiten T für die Übertragung eines Bytes aufgewendet werden müssen. Der SIO-Empfänger des Einchip-Mikrorechners benötigt nur ein Stoppbit, so daß beim Empfang von anderen Rechnern u. U. etwas Zeit gespart werden kann.
Die Struktur des seriellen Interface ist in Abb. 22 dargestellt. Der Empfänger benutzt das Port-3-Signal P30 als Eingang R×D. Das Schieberegister ordnet die nacheinander empfangenen Bits zum parallelen 8-Bit-Format. Sobald ein Byte komplett eingegangen ist, gelangt es in das Empfangs-Pufferregister. Gleichzeitig wird ein Interrupt angemeldet, damit ein Programm zum Verarbeiten der empfangenen Informationen rechtzeitig ablaufen kann. Bei der maximal möglichen Übertragungsgeschwindigkeit von 62 500 Bit/s bleiben dafür immerhin 176 µs. In dieser Zeit führt der Einchip-Mikrorechner etwa 80 Befehle aus, was für das Abholen des Bytes aus dem Pufferregister und dessen Verarbeitung völlig ausreicht. Bei geringerer Übertragungsrate bleibt entsprechend mehr Zeit.
Der Sender enthält ebenfalls ein Schieberegister als zentrale Baugruppe. Sobald hier mit einer Schreiboperation ein Byte parallel geladen wird, erzeugt es automatisch ein Startbit, dem die acht Informationsbits und mindestens zwei Stoppbits foigen. Der 1-Pegel des als Ausgang T×D verwendeten Port-3-Signals P37 bleibt danach bestehen, bis das nächste Byte in das Schieberegister gelangt. Um ein entsprechendes Programm zum richtigen Zeitpunkt starten zu können, erzeugt die SIO-Baugruppe nach dem Senden jedes Bytes eine Interruptanmeldung. Sender und Empfänger arbeiten stets mit gleicher Schiebetaktfrequenz, funktionieren aber sonst völlig unabhängig voneinander. Die SIO-Baugruppe kann daher gleichzeitig senden und empfangen, das nennt man Vollduplex. Paritätsgenerator und Paritätsprüfer sind nichts weiter als Zähl-Flipflops. Sie ermitteln, ob eine gerade oder ungerade Anzahl mit 1 belegter Informationsbits im Datenbyte enthalten ist. Damit lassen sich Übertragungsfehler erkennen. Der Programmierer kann wählen, ob er diese Möglichkeit nutzt oder nicht. Wenn er es tut, stehen nur noch die Bits DØ bis D6 für die Informationsübertragung zur Verfügung. Das Bit D7 ersetzt der Sender mit dem Ergebnis des Paritätsgenerators derart, daß insgesamt eine ungerade Anzahl mit 1 belegter Bits entsteht. Zwischen Start- und Stoppbits befinden sich damit ein, drei, fünf oder sieben 1-Bits. Der Empfänger trägt nach vollständigem Empfang eines Bytes das Ergebnis des Paritätsprüfers, der die sieben nutzbaren und das vom Sender zusätzlich erzeugte Bit auswertet, statt des Datenbits D7 in das Empfangs-Pufferregister ein. Eine 1 kennzeichnet dabei einen Übertragungsfehler. Damit hat der Programmierer die Möglichkeit, den falschen Empfang einzelner Bits und die Unterbrechung der Leitung zu erkennen. Die nutzbaren sieben Bits je Byte gestatten nicht mehr das direkte Übertragen der im Einchip-Mikrorechner verwendeten Zahlenformate, aber immerhin das Kodieren von alphanumerischen Zeichen (Buchstabe, Ziffer oder Sonderzeichen) z. B. nach dem ASCII-Standard. Deshalb wird die Möglichkeit der Paritätsprüfung meist dann genutzt, wenn Texte zu übertragen sind (Drucker-Anschluß).
Der Schiebetakt basiert auf dem Systemtakt XTAL. Mit dem Initialisieren von TØ läßt sich die Periodendauer T einstellen. Die 1:16-Teiler sind nötig, damit sich der Empfänger mit Hilfe des Startbits auf die Phasenlage des eingehenden Signals einstellen kann. Insgesamt gilt für die Berechnung der Schiebetakterzeugung:

$T = 16 * T_{T\emptyset}$

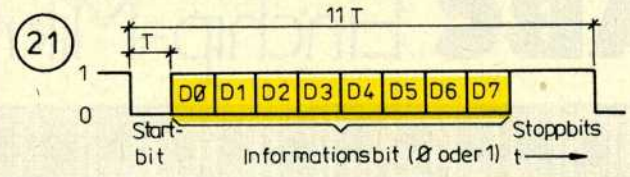
Damit lassen sich die Standard-Übertragungsraten bis 4800 Bit/s (8 MHz) bzw. 19 200 Bit/s (7,3728 MHz) realisieren. Abb. 23 gibt für beide Quarzfrequenzen die dabei nötigen Einstellwerte, die als Produkt der Frequenzteilungsverhältnisse (Zeitkonstanten) des Vorteilers PREØ und des Zählers TØ gebildet werden müssen.
Abb. 24 zeigt das Spezialregister SIO. Über die Adresse %F0 werden die seriell zu sendenden und empfangenen Daten transportiert. Wie bei den Ports (vgl. Abschnitt 2.2.) erfolgt der Zugriff beim Schreiben und beim Lesen auf zwei verschiedene Register. Hier sind es das Sende-Schieberegister und das Empfangs-Pufferregister. Das folgende Beispiel zeigt, wie unkompliziert sich das serielle Interface vorbereiten läßt.
Der JU+TE-Computer benutzt die SIO-Baugruppe für den Anschluß eines Magnetbandgerätes. Dabei wird eine Übertragungsrate von 600 Bit/s verwen-

det. Zum Modulieren der Bitfolge für die Magnetbandaufzeichnung muß der Anschluß P36 als Ausgang von TØ fungieren. Das nötige Frequenzteilungsverhältnis von 1:104 realisieren der Vorteiler PREØ (1:16) und der Zähler TØ (1:13) gemeinsam:
PREØ: ØØ1Ø ØØX1 = %21
(vgl. Abb. 16)
(TØ: ØØØØ 11Ø1 = %0D
(vgl. Abb. 17)
(Die Freigabe des Timers TØ durch das entsprechende Laden des Zeitgebersteuerregisters wird mit dem Vereinbaren von P36 als Ausgang von TØ verbunden:
TMR: Ø1XX XX11 = %43
(vgl. Abb. 20)
Es folgt die Freigabe der SIO-Baugruppe mit dem Laden des Port-3-Steuerregisters P3M für die Übertragung von acht Datenbits ohne Paritätsbit:
(P3M: Ø1XX XXXX = %40
(vgl. Abb. 9).
Als Abschluß des Kapitels über den Aufbau von Einchip-Mikrorechnern gibt Abb. 25 eine Übersicht der 16 Spezialregister. Neben der Kurzbezeichnung der Funktion sind die übliche Assemblernotation, die Adresse und die näher beschreibende Abbildung vermerkt. Die Erklärung der oberen sieben Register enthalten die folgenden Kapitel über die Programmierung und die Interruptverarbeitung.

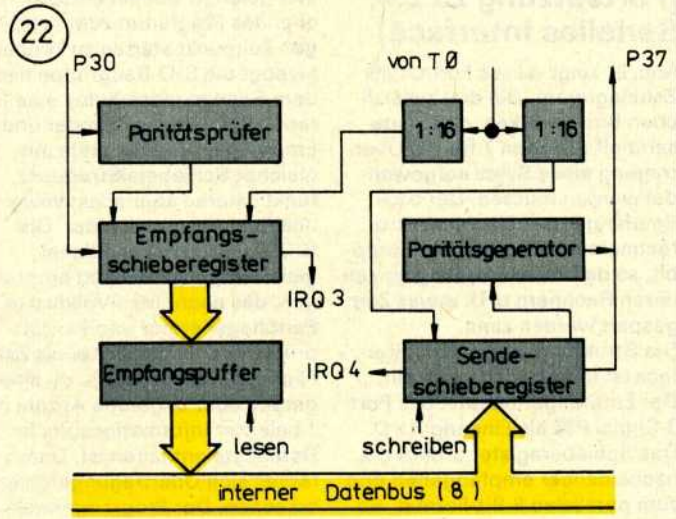
3. Programmierung

Die Entwicklung von Programmen ist das zentrale Problem beim Einsatz von Einchip-Mikrorechnern. In der Praxis übersteigen die Kosten der Softwareproduktion deutlich den Aufwand für die Hardware. Keine Anwendung kommt ohne Programm aus, denn der Einchip-Mikrorechner löst alle Aufgaben durch das Ausführen sinnvoll aneinandergereihter Befehle. Die Maschinen- bzw. Assemblersprache, die Gegenstand dieses Kapitels ist, steht dem Prozessor am nächsten und gestattet die beste Ausnutzung seiner Leistungsfähig-

21



22



23

Übertragungsrate (Bit/s)	Teilungsfaktor (TØ)	
	f_{XTAL} = 8 MHz	f_{XTAL} = 7,37 MHz
19 200	-	3
9 600	-	6
4 800	13	12
2 400	26	24
1 200	52	48
600	104	96
300	208	192

24

Serielles Ein-/Ausgaberegister SIO (%F0):

schreiben: Sende-Schieberegister (Sendedaten)
lesen: Empfangs-Pufferregister (Empfangsdaten)
bei Paritätsprüfung: D7 = 1 ≙ Fehler

keit. Dieses Sprachniveau bezieht sich direkt auf die Speicherbaugruppen, die sowohl die Programme aufnehmen als auch operative Daten enthalten.

3.1. Speichernutzung

Der Einchip-Mikrorechner benötigt Speicher für drei Zwecke:
- Programmspeicher
- Datenspeicher
- Stapelspeicher (stack).

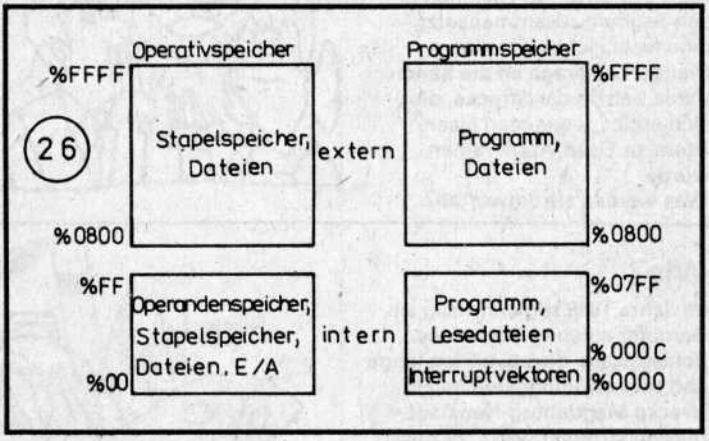
Abb. 26 zeigt, wo diese Ressourcen untergebracht werden können.

Der **Programmspeicher** enthält die vom Einchip-Mikrorechner auszuführenden Programme. Er wird mit dem Befehlszähler PC adressiert (vgl. Abb. 5). Auch Dateien wie Codetabellen und Zeichenketten können hier untergebracht werden. Der interne Programmspeicher (2-KByte-ROM) gestattet dem Prozessor nur Lesezugriffe. Im externen Bereich lassen sich auch RAM-Schaltkreise anschließen. Das macht variable Dateien möglich. Hier kann sich der Prozessor auch Programme selbst erzeugen. Die ersten zwölf Byte des internen Programmspeichers sind jedoch der Interruptverarbeitung zugeordnet. Hier stehen in jeweils einem Speicherplatzpaar die sechs 16-Bit-Startadressen der Interruptserviceroutinen. Das sind, wie noch erklärt wird, die Unterprogramme, die durch Interruptanmeldungen gestartet werden können. Auf der Adresse %000C muß der erste nach dem Einschalten (oder Rücksetzen) auszuführende Befehl stehen. Die weitere Nutzung des Programmspeichers steht im Ermessen des Programmierers. Er kann Programme und Dateien beliebig aneinanderreihen. Die Übersicht darf dabei aber nicht verlorengehen, denn der Prozessor deutet im Fehlerfall auch Datenbytes als Befehle, ohne es zu merken.

Als **Datenspeicher** eignet sich der interne Operativspeicher am besten. Er besteht aus den 16 Spezial- und den 124 Universalregistern sowie den vier Ports

(25)

Stackpointer$_L$	SPL	%FF	Abb. 29
Stackpointer$_H$	SPH	%FE	Abb. 29
Registerpointer	RP	%FD	Abb. 37
Flagbits	FLAGS	%FC	Abb. 30
Interruptfreigabe	IMR	%FB	Abb. 51
Interruptanmeldung	IRQ	%FA	Abb. 50
Interruptpriorität	IPR	%F9	Abb. 52
Port Ø- und 1-Modus	PØ1M	%F8	Abb. 13
Port 3-Modus	P3M	%F7	Abb. 9
Port 2-Modus	P2M	%F6	Abb. 12
Vorteiler TØ	PREØ	%F5	Abb. 16
Zähler TØ	TØ	%F4	Abb. 17
Vorteiler T1	PRE1	%F3	Abb. 18
Zähler T1	T1	%F2	Abb. 19
Zählersteuerung	TMR	%F1	Abb. 20
serielle Ein-/Ausgabe	SIO	%FØ	Abb. 24



(26)

(Abb. 7). Hier kann die Arithmetik-Logik-Einheit ALU direkt zugreifen. Reicht die Kapazität nicht aus, können Daten in den externen Operativ- oder Programmspeicher ausgelagert werden. Da dort nur Transportbefehle Zugriff haben, bringt man Daten, mit denen häufig operiert wird, vorzugsweise in den Universalregistern unter. Eine Besonderheit des Einchip-Mikrorechners besteht darin, daß auch die Ports und die Ein-/Ausgaberegister von SIO und Timern zum internen Operativspeicher gehören. Der direkte Zugriff der ALU gestattet besonders schnelle und effektive Ein-/Ausgabe-Programme.

Für das Speichern und Verarbeiten von Zahlen bietet der Einchip-Mikrorechner drei verschiedene Formate:
- vorzeichenloses 8-Bit-Format
- vorzeichenbehaftetes 8-Bit-Format
- BCD-Format.

Die ersten beiden basieren auf dem dualen Zahlensystem (Abb. 27). Bei der Verarbeitung vorzeichenbehafteter Zahlen bildet das Bit D7 die Vorzeichenstelle:
0 steht für plus,
1 steht für minus.
Die Darstellung negativer Zahlen entspricht dem Zweierkomplement.

Dr. Helmut Hoyer

Zeichnungen: Liebig

(Fortsetzung zu 3.1. Speichernutzung)

Im Bereich von 0 bis 127 stimmen die vorzeichenlosen und vorzeichenbehafteten Formate überein. Die einerseits für die Zahlen von 128 bis 255 verwendeten Kodes dienen andererseits der Darstellung der negativen Zahlen. Sie sind damit doppeldeutig. Der Programmierer muß sich daher stets im klaren sein, welches Format für die von der ALU berechneten Resultate gilt.

Die binäre Kodierung von Dezimalziffern (BCD) nutzt die Variationsmöglichkeiten nur unvollständig. Sie ordnet jeder Dezimalziffer direkt ein Halbbyte zu (Abb. 28). Dabei werden nur die Kombinationen 0000 bis 1001 (0 bis 9) verwendet. 1010 bis 1111 (A bis F) heißen Pseudotetraden und sind verboten. Wegen der größeren Umstände beim Entwickeln von Arithmetikprogrammen kommt dieses Format nur zum Einsatz, wenn es bei der Ein- und Ausgabe entscheidende Vorteile mit sich bringt.

Für die Verarbeitung von Zahlen größeren Betrages gestattet die ALU auch die Verwendung mehrerer Bytes. Mit 16 Bit (2 Bytes) gelten die folgenden Darstellungsgrenzen:

- vorzeichenlos: 0 bis 65535
- vorzeichenbehaftet: −32768 bis 32767
- BCD: 0 bis 9999

Den **Stapelspeicher** (stack) benötigt der Einchip-Mikrorechner als Ablage. Stack heißt eigentlich Keller, man stellt sich aber besser einen Stapel von Notizzetteln vor. Das Ablageprinzip besteht darin, daß die Notizen immer obenauf gelegt werden. Die Entnahme erfolgt ebenfalls von oben, so daß man stets die zuletzt abgelegte Notiz als erste zurück erhält. Auf englisch heißt dieses Prinzip last in first out (LIFO). Beim Einchip-Mikrorechner betehen diese Notizen meist aus Adressen, ab denen nach Verlassen eines Unterprogramms das Hauptprogramm fortzusetzen ist (vgl. GOSUB – RETURN). Es handelt sich allgemein um Dualzahlen. Jedes Byte belegt eine Speicherzelle oder ein Register.

Die Adressierung erfolgt mit dem Stapelzeiger (Stackpointer SP). Sein Inhalt ist die Adresse der Speicherzelle (oder des Registers), in der das zuletzt eingetragene und noch nicht wieder abgeholte Byte steht. Das entspricht der Lage des obersten Notizzettels. Beim Einspeichern eines Bytes in diese Ablage wird der Stackpointer automatisch um eins verringert (SP := SP − 1). Beim Abholen erfolgt entsprechend eine Erhöhung. Dadurch wächst der Stapelspeicher in Richtung der niederen Adressen. Das unterscheidet ihn zwar vom Notizzettelstapel, ändert aber das Prinzip nicht.

Abb. 29 zeigt den Stackpointer, der aus den Spezialregistern SPH (%FE) und SPL (%FF) besteht. Mit dem Laden des PortØ- und -1-Steuerregisters PØ1M (vgl. Abb. 13) läßt sich auswählen, ob der Stapelspeicher in den Universalregistern oder im externen Datenspeicher aufgebaut wird. Beim Initialisieren ist die Lokalisierung dieser Ablage aber noch genauer festzulegen. Dazu lädt man den Stackpointer mit der Adresse, unterhalb der sich der Stapelspeicher befinden soll. Für den externen Datenspeicher werden alle 16 Bit als Adresse benötigt. Bei internem Stapelspeicher bleibt der höhere Teil des Stackpointers unbenutzt, hier ist nur SPL voreinzustellen. In diesem Fall kann das Register SPH (%FE) als Universalregister genutzt werden.

Beim JU+TE-Computer befindet sich der Stapelspeicher im externen Bereich von %FD80 bis %FDFF. Das scheint sehr wenig zu sein, tatsächlich bleibt der größte Teil in der Praxis aber ungenutzt. Nach jedem Rücksetzen erhält der Stackpointer SP den Initialwert %FE00 (SPH := %FE, SPL := %00). Die erste „Notiz" gelangt so auf die Speicherzelle mit der um eins geringeren Adresse (%FDFF). Bei internem Stapelspeicher wird SPL meist mit %80 initialisiert.

Wenn man das Signal $\overline{DM}$ (vgl. Abb. 9) nicht für die Speicheradressierung benutzt, entsteht ein einheitlicher externer Bereich. Programme, Dateien und Stapelspeicher lassen sich dann wie beim U 880 D nur durch die 16-Bit-Adresse unterscheiden. Im Inneren bleibt es natürlich auch dann bei der Trennung zwischen Programm- und Datenspeicher.

3.2. Flagregister

In maschinennahen Programmen spielt das Flagregister häufig eine Rolle. Es enthält einige Flipflops, die das Ergebnis verschiedener Befehle bezüglich besonderer Eigenschaften kennzeichnen. Man kann sich diese Funktion gut anhand der in der Seefahrt üblichen Signalflaggen merken. Ausgewertet werden die Flagbits als Bedingung von Sprungbefehlen, sie gestatten datenabhängige Programmverzweigungen. Die BASIC-Anweisung IF bedingung THEN GOTO zeilennummer realisieren in Maschinensprache zwei Befehle:

einer, der die Flagbits der Bedingung entsprechend beeinflußt, und ein anderer, der einen durch die Belegung des Flagregisters bedingten Sprung realisiert. Einige Befehle verwenden das Flagbit C direkt als Operanden. Von den in Abb. 30 dargestellten acht Flagbits sind nur die links stehenden vier für die Programmierung von Bedeutung. Sie werden wie folgt gesetzt:
C = 1 bei Übertrag (carry)
(Überschreitung von [0, 255])
Z = 1 bei Null-Ergebnis (zero)
(alle Ergebnisbits = 0)
S = 1 bei Minus-Ergebnis (sign)
(S := BIT D7 bzw. D15)
V = 1 bei Überlauf (overflow)
(Überschreitung von [-128, 127])
Das **C**-Flag speichert den Übertrag von Byte zu Byte. Es gestattet einfache Verarbeitung von Zahlen, die mehr als ein Register in Anspruch nehmen (z. B. 16-Bit-Zahlen). Das gilt für alle drei Zahlenformate. Außerdem eignet es sich dazu, die Einhaltung des vorzeichenlosen Formats zu überwachen. Bei der Addition und Subtraktion mit Mehrbytezahlen wird mit dem niederwertigsten Byte begonnen und der Übertrag C von Byte zu Byte verrechnet. Entsteht bei der Behandlung des höchstwertigen Bytes ein Übertrag (C=1), dann läßt sich das Ergebnis nicht im vorzeichenlosen Format darstellen. Es ist entweder negativ oder zu groß für die gewählte Stellenzahl.
Das **V**-Flag überwacht die Einhaltung des vorzeichenbehafteten Formats. Es gilt stets nach der Verarbeitung des höchstwertigen Bytes. Zusätzlich läßt das **S**-Flag das Vorzeichen erkennen. Ist kein Überlauf eingetreten (V=0), gilt die bekannte Zuordnung S=0 für plus und S=1 für minus. Im Falle eines Überlaufs (V=1) wird das S-Flag jedoch entgegengesetzt belegt.
Das **Z**-Flag kennzeichnet mit Z=1 ein Nullergebnis in allen Ergebnisbits einer Befehlsausführung. Es ist daher in den meisten Fällen mit 0 belegt. Bei logischen Operationen und Bit-Tests dient es, wie wir noch sehen werden, selbst als Ergebnisbit.
Offensichtlich bieten die Flagbits viel mehr Auswertungsmöglichkeiten an, als der Programmierer im Einzelfall nutzen kann. Je nachdem, welches Zahlenformat gilt, wird eben C oder V zur Bereichsüberwachung genutzt.
Viele Befehle beeinflussen die Flagbits, ohne daß der Programmierer auch nur eines auswerten läßt. Die Verwendung als Sprungbedingung bietet insgesamt fünfzehn Möglichkeiten, bei denen die Flags z. T. einzeln, z. T. miteinander verknüpft eingehen. Abb. 31 enthält sie in der Reihenfolge, in der die entsprechenden Sprünge in der Befehlstabelle (Abb. 32) untereinander stehen. Neben der Grundbedeutung enthält Abb. 31 auch die Beschreibung als Ergebnis eines Vergleichs von Ziel- und Quelloperanden. Die Abkürzung vzl. steht dabei für vorzeichenlose Operandendarstellung (unsigned), cc bezeichnet den Bedingungscode (condition code) in der Assemblersprache. Er basiert auf der englischen Bezeichnung der Bedeutung (z. B. less than).
Für die Verarbeitung des BCD-Formats benutzt der Einchip-Mikrorechner intern die Flagbits **D** (D=1 nach Subtraktion, D=0 nach Addition) und **H** (Halbbyte-carry). Die Bits **F1** und **F2** beeinflußt die ALU nicht, so daß sie der Nutzung durch den Programmierer freistehen.

3.3. Assemblernotation

Die Assemblersprache ordnet jedem Maschinenbefehl, den der Einchip-Mikrorechner ausführen kann, ein Mnemonik zu. Das ist eine aus zwei, drei oder vier Buchstaben bestehende Kurzbezeichnung, die Assoziation zur Wirkung des Befehls zuläßt. Das Mnemonik **add** bezeichnet z. B. einen Additionsbefehl. Außerdem enthält die Assemblernotation Angaben zu den Operanden. **add 7,5** steht für den Maschinenbefehl, der die Inhalte der Register Nr. 7 und Nr. 5 addiert und das Ergebnis in das Register Nr. 7 lädt. Die Operandenangabe folgt dem Mnemonik nach einem Leerzeichen. Sie beginnt mit dem Zieloperanden, dem sich durch Komma getrennt die Quelle anschließt. Einige Befehle brauchen nur eine, andere gar keine Operandenangabe. Mehr als zwei sind nie erforderlich.
Mnemonik und Operandenspezifikation beschreiben einen Maschinenbefehl bereits vollständig. Befehlen, die Ziel von Programmsprüngen sind, wird eine zusätzliche Angabe vorangestellt. Sie bezeichnet die Adresse des Befehls mit einem Symbol, das Marke genannt wird. Ihm folgt gewöhnlich ein Doppelpunkt als Trennzeichen. Ein markierter Befehl sieht z. B. so aus:
MA1: add 7,5
Es darf auch noch ein Kommentar angefügt werden, um die Übersicht zu erleichtern. Beim Übersetzen in die Maschinensprache spielt er aber überhaupt keine Rolle. Das Mnemonik wird verbreitet auch mit Großbuchstaben notiert, wegen der üblichen Operandenbezeichnungen hat die Kleinschreibung beim Einchip-Mikrorechner jedoch den Vorteil, weniger zu verwirren. Bei den meisten Assemblern (Programme zum Übersetzen von Assembler- in Maschinensprache) ist vorgeschrieben, daß Marken mit einem Buchstaben beginnen müssen.
Beim Notieren eines Programms in Maschinensprache steht gewöhnlich ebenfalls eine Zeile für einen Befehl. Statt Marken sind die konkreten Speicheradressen der Operationscodes (erstes Byte jedes Befehls) üblich. Es folgen die Bytes des Befehls. All diese Angaben erfolgen hexadezimal:
E200 04 05 07
Hier gilt nicht grundsätzlich die Reihenfolge Operation – Ziel – Quelle. Unser Beispiel hat ein Befehlsformat, bei dem zuerst der Quelloperand und dann der Zieloperand angegeben werden muß.

Dr. Helmut Hoyer

Dualformat

Stellenwert: 128 64 32 16 8 4 2 1

D7	D6	D5	D4	D3	D2	D1	DØ

vorzeichenlos hexa.	vorzeichenlos dual	dez.	vorzeichenbehaftet dual	vorzeichenbehaftet hexa.
FF	1 1 1 1 1 1 1 1	255		
FE	1 1 1 1 1 1 1 0	254	mit 8 Bit nicht darstellbar	
⋮	⋮	⋮		
81	1 0 0 0 0 0 0 1	129		
80	1 0 0 0 0 0 0 0	128		
7F	0 1 1 1 1 1 1 1	127	0 1 1 1 1 1 1 1	7F
⋮	⋮	⋮	⋮	⋮
02	0 0 0 0 0 0 1 0	2	0 0 0 0 0 0 1 0	02
01	0 0 0 0 0 0 0 1	1	0 0 0 0 0 0 0 1	01
00	0 0 0 0 0 0 0 0	0	0 0 0 0 0 0 0 0	00
nicht darstellbar		-1	1 1 1 1 1 1 1 1	FF
		-2	1 1 1 1 1 1 1 0	FE
		⋮	⋮	⋮
		-127	1 0 0 0 0 0 0 1	81
		-128	1 0 0 0 0 0 0 0	80

(27)

Stackpointer SP (% FE und % FF)

SP 15	SP 14	SP 13	SP 12	SP 11	SP 10	SP 9	SP 8

SPH (% FE): nur bei externem Stapelspeicher benutzt

SP 7	SP 6	SP 5	SP 4	SP 3	SP 2	SP 1	SP 0

SPL (% FF)

(29)

Binär kodierte Dezimalzahlen (BCD)

Stellenwert: 80 40 20 10 8 4 2 1

D7	D6	D5	D4	D3	D2	D1	DØ

Zehner Einer

(28)

Flagregister FLAGS (% FC)

C	Z	S	V	D	H	F1	F2

(30)

(31)

Nr.	Flags	Bedeutung	cc
1	S∀V=1	kleiner	lt
2	Zv(S∀V)=1	kleiner oder gleich	le
3	CvZ=1	vzl. kleiner oder gleich	ule
4	V=1	Überlauf	ov
5	S=1	minus	mi
6	Z=1	Null, gleich	z, eq
7	C=1	Übertrag, vzl. kleiner	c, ult
8	–	unbedingt	
9	S∀V=0	größer oder gleich	ge
A	Zv(S∀V)=0	größer	gt
B	CvZ=0	vzl. größer	ugt
C	V=0	kein Überlauf	nov
D	S=0	plus	pl
E	Z=0	nicht Null, ungleich	nz, ne
F	C=0	kein Übertrag, vzl. größer oder gleich	nc, uge

Zeichnungen: Liebig

(Fortsetzung zu 3.3. Assemblernotation)

Die Formulierungsvorschriften (Syntax) enthalten einige Unregelmäßigkeiten, an die man sich aber schnell gewöhnen kann.
In der tabellarischen Zusammenstellung der Operationscodes (Abb. 32) sind sowohl Operationen als auch Operandenkombinationen übersichtlich angeordnet. Die Tabelle hat 16 × 16 Felder, jedes ist einem Operationscode zugeordnet. Einige Felder sind leer, die zugehörigen Codes bewirken keine sinnvolle Befehlsausführung. Die niederen vier Bit des Operationscodes bestimmen bis auf wenige Ausnahmen die Art der Operandenadressierung. Daher stehen die für die jeweilige Spalte typische Operandenkombinationen in der Kopfzeile der Abb. 32. Die mit den Symbolen **R, IR, IRR, r, Ir, Irr, n, e** und **adr** zugeordneten Adressierungsmechanismen stellen wir bei der Befehlsbeschreibung mit vor. **cc** steht auch hier für den Bedingungscode. Mit den höheren vier Bit des Operationscodes werden bei einigen Befehlen das verwendete Arbeitsregister **r** oder die Sprungbedingung **cc** festgelegt. Deswegen stehen diese beiden Symbole über der linken Spalte.
Im Zentrum jedes Feldes ist das Mnemonik notiert. Sofern die spaltentypische Operandenkombination nicht zutrifft, steht die für den jeweiligen Befehl gültige unter dem Mnemonik. Links oben ist der Operationscode hexadezimal notiert, der sich aus Zeilen- und Spaltennummer zusammensetzt.
Die rechte obere Ecke betrifft die Flags. Hat ein Feld dort keine Eintragung, verändert der betreffende Befehl die Flags nicht.
Auch die Sprungbefehle (Spalten B und D) lassen die Flags unverändert. Hier befindet sich jedoch die Assemblernotation der betreffenden Sprungbedingung rechts oben (vgl. Abb. 31). Befehle mit Flagbeeinflussung markiert eine Ziffer in der rechten oberen Ecke. Sie kodiert das Laden der interessierenden vier Bits C, Z, S und V entsprechend der Abb. 33. Ausschließlich der Einschränkungen bei **da** (decimal adjust) und **swap** (Halbbyte-Tausch) gelten die im Abschnitt 3.2. erklärten Bedeutungen der beeinflußten Flagbits.
Links unten ist die Ausführungszeit als Anzahl der inneren Taktperioden notiert. Bei 8-MHz-Quarz steht die 6 entsprechend für 1,5 µs, die 20 für 5 µs Ausführungszeit. Bei dieser Angabe ist das Pipelining (vgl. Abb. 3) berücksichtigt, so daß bei den meisten Befehlen nur die Zeit für das Befehlslesen durch diese Angabe widerspiegelt wird. Einige Felder enthalten zwei Angaben. Bei **push** (laden in den stack) unterscheiden sich die Ausführungszeiten bei innerem/äußerem Stapelspeicher. Die Sprungbefehle (Spalten A, B und D) dauern 3 µs, wenn sie ausgeführt, 2,5 µs, wenn sie ignoriert werden (Bedingung nicht erfüllt).
Die Zahl rechts unten kennzeichnet das Befehlsformat. Diese Angabe bezieht sich auf die Abb. 34, die angibt, welche Zusatzinformationen in dem Operationscode (OP) angefügten Bytes stehen müssen. Nur das Format 1 (Spalten E und F) braucht keine Zusätze. Zum Teil stehen in einem Byte zwei Informationen, die jeweils mit vier Bit kodiert sind.
Die Möglichkeiten zum Spezifizieren der Operanden faßt die Abb. 35 zusammen. Wie diese Adressierungsarten wirken, wird in den folgenden Abschnitten im Zusammenhang mit den verschiedenen Maschinenbefehlen beschrieben.

Das Beispiel vom Beginn dieses Abschnitts hat den Operationscode Ø4. Er steht in Abb. 32 entsprechend in der Zeile 0, Spalte 4. Rechts oben ist die Flagbeeinflussung notiert (vgl. Abb. 33): C, Z, S und V werden durch diesen Befehl neu belegt. Er benötigt zehn interne Taktperioden als effektive Ausführungszeit (links unten). Bei 8-MHz-Quarz bedeutet das 2,5 µs. Er hat das Format Nr. 7 (Abb. 34). Die drei Bytes enthalten Operationscode, Quelladresse und Zieladresse, konkret: Ø4 Ø5 Ø7.
Die Abb. 32 gibt eine umfassende Übersicht des Befehlssatzes. Sie ist daher wichtigstes Hilfsmittel bei der Programmentwicklung für Einchip-Mikrorechner.

3.4. Verarbeitungsbefehle

Verarbeitungsbefehle dienen der eigentlichen Datenverarbeitung. Hierzu zählen arithmetische und logische Operationen, die in der ALU ausgeführt werden. Verarbeitungsbefehle benötigen einen oder zwei Operanden. Dabei handelt es sich um Registerinhalte und Direktoperanden

Dr. Helmut Hoyer

(wird fortgesetzt)

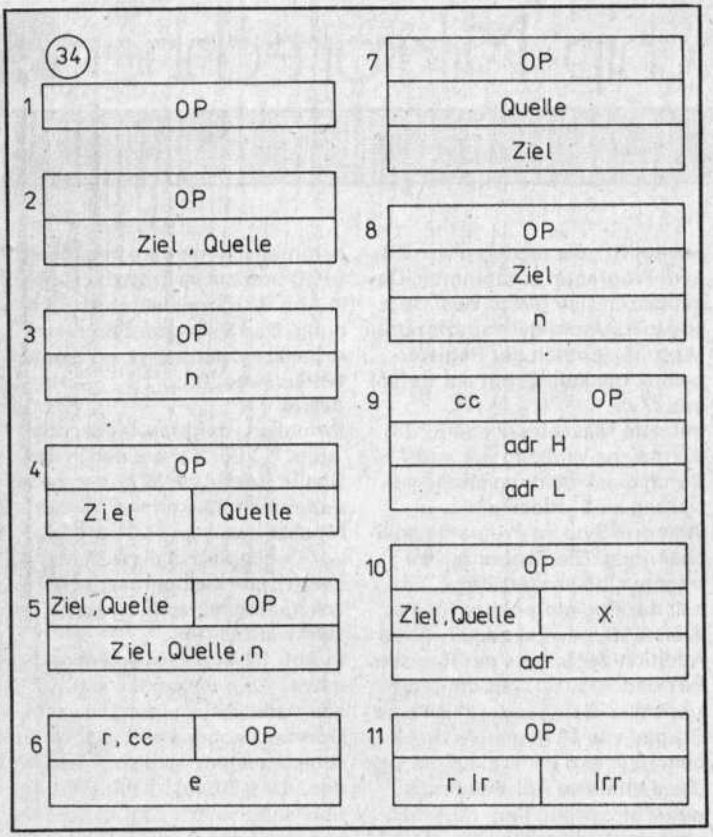
Abbildung 32

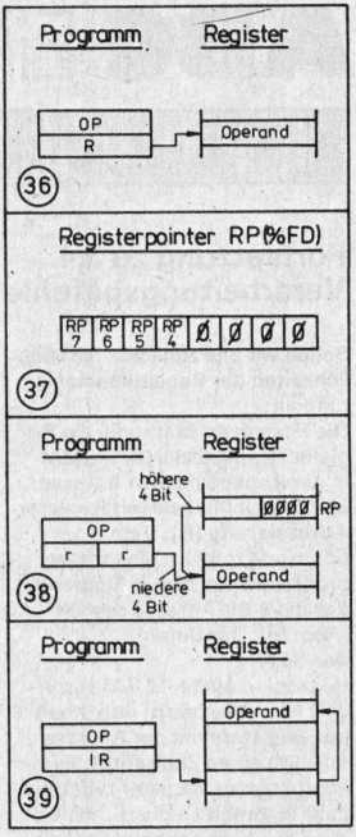
r / cc	0	1	2	3	4	5	6	7
	* R	* IR	r,r	r,IR	* * R,R	* * R,IR	* R,n	* IR,n
0	00 2 dec 6 2	01 2 dec 6 2	02 1 add 6 4	03 1 add 6 4	04 1 add 10 7	05 1 add 10 7	06 1 add 10 8	07 1 add 10 8
1	10 1 rlc 6 2	11 1 rlc 6 2	12 1 adc 6 4	13 1 adc 6 4	14 1 adc 10 7	15 1 adc 10 7	16 1 adc 10 8	17 1 adc 10 8
2	20 2 inc 6 2	21 2 inc 6 2	22 1 sub 6 4	23 1 sub 6 4	24 1 sub 10 7	25 1 sub 10 7	26 1 sub 10 8	27 1 sub 10 8
3	30 jp IRR 8 2	31 srp n 6 3	32 1 sbc 6 4	33 1 sbc 6 4	34 1 sbc 10 7	35 1 sbc 10 7	36 1 sbc 10 8	37 1 sbc 10 8
4	40 1 da 8 2	41 1 da 8 2	42 2 or 6 4	43 2 or 6 4	44 2 or 10 7	45 2 or 10 7	46 2 or 10 8	47 2 or 10 8
5	50 pop 10 2	51 pop 10 2	52 2 and 6 4	53 2 and 6 4	54 2 and 10 7	55 2 and 10 7	56 2 and 10 8	57 2 and 10 8
6	60 2 com 6 2	61 2 com 6 2	62 2 tcm 6 4	63 2 tcm 6 4	64 2 tcm 10 7	65 2 tcm 10 7	66 2 tcm 10 8	67 2 tcm 10 8
7	70 push 10/12 2	71 push 12/14 2	72 2 tm 6 4	73 2 tm 6 4	74 2 tm 10 7	75 2 tm 10 7	76 2 tm 10 8	77 2 tm 10 8
8	80 2 decw 10 2	81 2 decw 10 2	82 ldе r,lrr 12 11	83 ldei lr,lrr 18 11				
9	90 1 rl 6 2	91 1 rl 6 2	92 ldе lrr,r 12 11	93 ldei lrr,lr 18 11				
A	A0 2 incw 10 2	A1 2 incw 10 2	A2 1 cp 6 4	A3 1 cp 6 4	A4 1 cp 10 7	A5 1 cp 10 7	A6 1 cp 10 8	A7 1 cp 10 8
B	B0 clr 6 2	B1 clr 6 2	B2 2 xor 6 4	B3 2 xor 6 4	B4 2 xor 10 7	B5 2 xor 10 7	B6 2 xor 10 8	B7 2 xor 10 8
C	C0 1 rrc 6 2	C1 1 rrc 6 2	C2 ldc r,lrr 12 11	C3 ldci lr,lrr 18 11				C7 ld r,X 10 10
D	D0 1 sra 6 2	D1 1 sra 6 2	D2 ldc lrr,r 12 11	D3 ldci lrr,lr 18 11	D4 call IRR 20 2		D6 call adr 20 9	D7 ld X,r 10 10
E	E0 1 rr 6 2	E1 1 rr 6 2		E3 ld 6 4	E4 ld 10 7	E5 ld 10 7	E6 ld 10 8	E7 ld 10 8
F	F0 1 swap 8 2	F1 1 swap 8 2		F3 ld lr,r 6 4		F5 ld IR,R 10 7		

r / cc	8	9	A	B	C	D	E	F
	* r,R	R,r	r,e	cc,e	r,n	cc,adr	r	
0	08 ld 6 5	09 ld 6 5	0A djnz 12/10 6		0C ld 6 5		0E inc 2 6 1	
1	18 ld 6 5	19 ld 6 5	1A djnz 12/10 6	1B jr lt 12/10 6	1C ld 6 5	1D jp lt 12/10 9	1E inc 2 6 1	
2	28 ld 6 5	29 ld 6 5	2A djnz 12/10 6	2B jr le 12/10 6	2C ld 6 5	2D jp le 12/10 9	2E inc 2 6 1	
3	38 ld 6 5	39 ld 6 5	3A djnz 12/10 6	3B jr ule 12/10 6	3C ld 6 5	3D jp ule 12/10 9	3E inc 2 6 1	
4	48 ld 6 5	49 ld 6 5	4A djnz 12/10 6	4B jr ov 12/10 6	4C ld 6 5	4D jp ov 12/10 9	4E inc 2 6 1	
5	58 ld 6 5	59 ld 6 5	5A djnz 12/10 6	5B jr mi 12/10 6	5C ld 6 5	5D jp mi 12/10 9	5E inc 2 6 1	
6	68 ld 6 5	69 ld 6 5	6A djnz 12/10 6	6B jr z 12/10 6	6C ld 6 5	6D jp z 12/10 9	6E inc 2 6 1	
7	78 ld 6 5	79 ld 6 5	7A djnz 12/10 6	7B jr c 12/10 6	7C ld 6 5	7D jp c 12/10 9	7E inc 2 6 1	
8	88 ld 6 5	89 ld 6 5	8A djnz 12/10 6	8B jr 12 6	8C ld 6 5	8D jp 12 9	8E inc 2 6 1	8F di 6 1
9	98 ld 6 5	99 ld 6 5	9A djnz 12/10 6	9B jr ge 12/10 6	9C ld 6 5	9D jp ge 12/10 9	9E inc 2 6 1	9F ei 6 1
A	A8 ld 6 5	A9 ld 6 5	AA djnz 12/10 6	AB jr gt 12/10 6	AC ld 6 5	AD jp gt 12/10 9	AE inc 2 6 1	AF ret 14 1
B	B8 ld 6 5	B9 ld 6 5	BA djnz 12/10 6	BB jr ugt 12/10 6	BC ld 6 5	BD jp ugt 12/10 9	BE inc 2 6 1	BF iret 1 16 1
C	C8 ld 6 5	C9 ld 6 5	CA djnz 12/10 6	CB jr nov 12/10 6	CC ld 6 5	CD jp nov 12/10 9	CE inc 2 6 1	CF rcf 3 6 1
D	D8 ld 6 5	D9 ld 6 5	DA djnz 12/10 6	DB jr pl 12/10 6	DC ld 6 5	DD jp pl 12/10 9	DE inc 2 6 1	DF scf 3 6 1
E	E8 ld 6 5	E9 ld 6 5	EA djnz 12/10 6	EB jr nz 12/10 6	EC ld 6 5	ED jp nz 12/10 9	EE inc 2 6 1	EF ccf 3 6 1
F	F8 ld 6 5	F9 ld 6 5	FA djnz 12/10 6	FB jr nc 12/10 6	FC ld 6 5	FD jp nc 12/10 9	FE inc 2 6 1	FF nop 6 1

Fortsetzung zu 3.4. Verarbeitungsbefehle

Sehen wir uns zunächst die Möglichkeiten der Registeradressierung an:
Die einfachste Methode, ein Register zu spezifizieren, besteht in der Angabe dessen Adresse. Man bezeichnet sie mit **Register-Adressierung (R).** Dem Operationscode folgt im Maschinenprogramm ein Byte je Operand, das die 8-Bit-Adresse enthält (Abb. 36). Der Befehl
dec %7A
(Maschinencode ØØ 7A) verringert (dekrementiert) den Inhalt des Registers mit der Adresse %7A um einen Zählschritt. Speicherte dieses Register z. B. die Zahl 15, erhält es durch Ausführung des angegebenen Befehls den neuen Inhalt 14. Im Beispiel des vorherigen Abschnitts (add 7,5) sind bei dieser Adressierungsweise insgesamt drei Byte für den Maschinencode erforderlich, da zwei Operanden mit Registeradressierung zu spezifizieren sind. Das erlaubt zwar viele Kombinationsmöglichkeiten, benötigt aber entsprechend große Programmspeicher-Kapazität und eine verhältnismäßig lange Rechenzeit.
Meist ist es günstiger, das Spezialregister **Registerpointer (%FD)** zum Festlegen der höheren vier Adreßbits zu benutzen. Wie in Abb. 37 dargestellt, enthält es nur Flipflops auf den Positionen D4 bis D7. Im unteren Halbbyte findet man stets Ø-Bits, gleichgültig, was hierhin geladen wurde. Bei der **Arbeitsregister-Adressierung (r)** setzt sich die Registeradresse aus zwei Komponenten zusammen: Die höheren vier Bit werden dem Registerpointer RP, die niederen vier Bit dem Programm entnommen. Damit lassen sich gleich zwei Operanden in einem Byte spezifizieren (Abb. 38). Enthält der Registerpointer die Zahl Ø, hat der Befehl
add r7,r5
mit dem Maschinencode Ø2 75 die gleiche Wirkung wie add 7,5. Durch die Arbeitsregisteradressierung sind jedoch nur zwei statt drei Byte im Programmspeicher nötig. Die Rechenzeit beträgt nur 1,5 µs statt 2,5 µs. Enthält der Registerpointer RP den Wert %1Ø, bewirkt add r7, r5 die Addition der Inhalte der Register %17 und %15. So wird mit jeder sinnvollen Belegung von RP eine Gruppe von 16 Registern zu Arbeitsregistern erklärt, auf die sich diese effektive Adressierungsweise anwenden läßt. Natürlich kann der Registerpointer im Laufe eines Programms auch neu geladen werden, um andere Register als bislang zu Arbeitsregistern zu machen. Sinnvolle Einstellwerte von RP sind %ØØ, %1Ø, %2Ø, %3Ø, %4Ø, %5Ø, %6Ø, %7Ø und %FØ. Auch die Spezialregister können so als Arbeitsregister deklariert werden (vgl. Abb. 7). Ihrer Vorzüge wegen ist die Adressierungsweise r in der Praxis die häufigste. Die meisten Befehle lassen Arbeitsregister-Adressierung auch dort zu, wo acht Bit für die Spezialisierung eines Operanden im Befehlsformat zur Verfügung stehen. Die betreffenden Befehle sind mit einem * in der Kopfzeile der Abb. 32 (vgl. JU+TE 1/1989) markiert. Hier kann durch die Dualkombination 1110 (%E) im höheren Halbbyte der Registerpointer einbezogen werden. Der Einchip-Mikrorechner ersetzt dieses %E (das eine Adresse des nicht genutzten Bereichs zur Folge hätte) automatisch mit dem Inhalt von RP. Damit entsteht letztlich die in Abb. 38 dargestellte Adreßbildung. Das Dekrementieren von Arbeitsregister 8 wird in Assemblersprache
dec r8
formuliert, der Maschinencode lautet ØØ E8. Nur bei den in der Spalte 9 der Abb. 32 notierten Ladebefehlen funktioniert dieser Mechanismus nicht. Hier muß der Quelloperand als Arbeitsregister (r), der Zieloperand jedoch vollständig mit acht Bit adressiert werden (R).
In Abb. 32 ist gut zu erkennen, daß es für die Addition wie für alle Verknüpfungsbefehle sechs Operationscodes gibt. Der Unterschied besteht nur in der Operandenadressierung, die ALU-Operation ist bei allen sechs gleich. Neben den bekannten Symbolen R und r finden wir auch **IR** und **Ir.** Damit wird die indirekte Adressierung gekennzeichnet. Hier enthält der Befehl nicht die Adresse des Operanden, sondern die Adresse eines Registers, dessen Inhalt die Operandenadresse ist. Als Postanschrift entspricht das etwa: Den Empfänger dieses Briefes gibt Ihnen Herr Meier, Straße soundso ... an. Darauf läßt sich die Post natürlich nicht ein, beim Programmieren ist solch ein Register, in dem die Operandenadresse auch variiert werden kann, jedoch oft hilfreich. Man braucht die indirekte Adressierung z. B. beim Arbeiten mit Codetabellen.
In Abb. 39 ist die **indirekte Register-Adressierung (IR)** dargestellt. Zum Vermitteln der Operandenadresse kann jedes Register verwendet werden. Es gibt hier keine Einschränkungen.
Abb. 40 zeigt die **indirekte Arbeitsregister-Adressierung (Ir).**





Hier enthält ein Arbeitsregister die Adresse des Operanden, der selbst nicht in einem Arbeitsregister stehen muß. So kann wegen des frei wählbaren Arbeitsregister-Inhalts mit Ir ein Operand an beliebiger Stelle im Registerbereich trotz nur vier Adreßbits im Maschinencode spezifiziert werden. Natürlich muß die Anwendung dieses Mechanismus mit Voreinstellen von RP und des vermittelnden Arbeitsregisters vorbereitet werden.

Für das Inkrementieren (+1) und das Dekrementieren (−1) enthält der Befehlssatz des Einchip-Mikrorechners auch Befehle, die 16-Bit-Zahlen verarbeiten. Bei diesen Wort-bezogenen Operationen (**incw** und **decw**) werden zwei Register als ein Operand verwendet. Das eine muß eine gerade Adresse haben (niederwertiges Adreßbit = Ø). Es enthält die acht höheren Bits des Operanden. Die acht niederen speichert das Register mit der nächsthöheren Adresse (niederwertigstes Adreßbit = 1). Wegen dieser festen Zuordnung muß im Befehl nur eine Adresse stehen: die des Registers mit dem höheren Byte des Operanden. In Abb. 41 ist diese **Doppelregister-Adressierung (RR)** dargestellt.

incw %7A

(Maschinencode AØ 7A) erhöht den Inhalt von Register %7A und %7B als 16-Bit-Zahl um einen Zählschritt. Auch die **Doppelarbeitsregister-Adressierung (rr)** ist bei diesen Befehlen möglich.

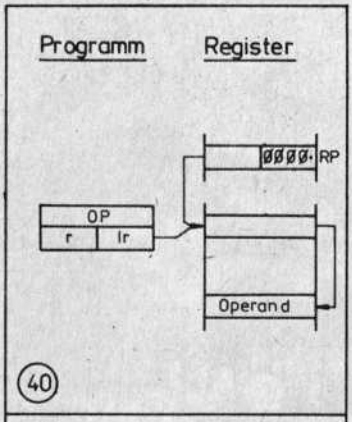
incw rr8

(Maschinencode AØ E8) erhöht das aus r8 und r9 bestehende Doppelregister. Die höheren vier Adreßbits des Operanden-Registerpaares entnimmt der Einchip-Mikrorechner wie beschrieben dem Registerpointer RP. Auch hier läßt sich außerdem indirekt adressieren:

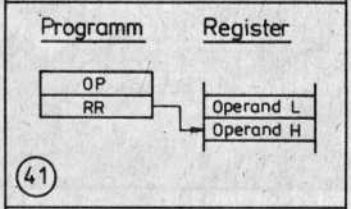
incw @r8

(Maschinencode A1 E8) erhöht den Inhalt des Registerpaares, dessen Adresse im Arbeitsregister r8 steht. Bemerkenswert ist, daß grundsätzlich weder der Arbeitsregister-Mechanismus noch die indirekte Adressierung zusätzliche Rechenzeit beanspruchen. Das Erhöhen und Verringern von 16-Bit-Worten dauert dagegen eine Mikrosekunde länger als beim 8-Bit-Format, denn hier muß die ALU zweimal aktiviert werden.

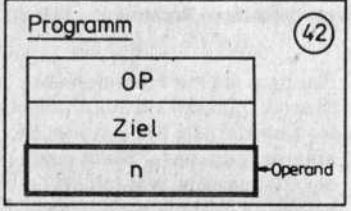
Die letzte Adressierungsart, die wir bei Verarbeitungsbefehlen finden, ist die einfachste: Abb. 42 stellt den **Direktoperanden (n)** dar. Operand ist das als Teil des Befehls gegebene Byte. Es läßt sich durch die Befehlsausführung nicht verändern. Daher können Direktoperanden nur als Quelle, nicht als Ziel verwendet



(40)



(41)



(42)

(33) Beeinflussung der Flags C, Z, S und V:

1	alle	undefiniert: bei da: : V bei swap: C , V
2	Z , S und V	C bleibt unverändert
3	nur C	Z , S und V unverändert

Zeichnungen: Liebig

(35)

Kurzzeichen (Abb. 32)	Assembler-notation		Bedeutung
R		Zahl	Register
IR	@	Zahl	indirekt Register
r	r	Zahl	Arbeitsregister
Ir	@r	Zahl	indirekt Arbeitsregister
n	#	Zahl	Direktoperand
RR		Zahl	Doppelregister
IRR	@	Zahl	indirekt Doppelregister
rr	rr	Zahl	Doppelarbeitsregister
Irr	@rr	Zahl	indirekt Doppelarbeitsregister
X	Zahl (r	Zahl)	indiziert
adr	Marke		absolute Adresse
e	Marke		relative Adresse

werden. Der Befehl
add 7,#3
(Maschinencode Ø6 Ø7 Ø3) erhöht den Inhalt von Register 7 um drei Zählschritte.
Wie schon zu sehen war, müssen in der Assemblersprache alle von der Adressierungsweise R oder RR abweichenden Operandenarten mit einem vorangestellten Zeichen gekennzeichnet werden (Abb. 35). Auch hexadezimale Zahlenangaben benötigen zum Unterscheiden vom Dezimalen solch einen Präfix. Hier nun eine Zusammenstellung der verwendeten Zeichen:
@ indirekt
r Arbeitsregister
Direktoperand
% hexadezimal.
Bei Kombinationen mehrerer Präfixe wird zuerst das hier weiter oben stehende Zeichen notiert.

3.4.1. Arithmetik-Befehle

Die ALU des Einchip-Mikrorechners kann fünf verschiedene arithmetische Verknüpfungen zweier Operanden ausführen. Die Operationscodes Ø2 und Ø7 bewirken die Addition:
add ziel,quelle
Ziel:=Ziel+Quelle.
Zuerst werden Ziel und Quelle addiert, dann das Ergebnis in das Zielregister gespeichert. Da diese Operation alle Flagbits neu belegt, lassen sich Bereichsüberschreitungen (s. Abschnitt 3.2.), Vorzeichen und Nullergebnis mit anschließenden Sprungbefehlen bei Bedarf gesondert behandeln. Für die Verarbeitung von Mehrbytezahlen gibt es eine weitere Additionsverknüpfung, die Überträge aus niederen Bytes einbezieht (Operationscodes 12 bis 17):
adc ziel,quelle
Ziel:=Ziel+Quelle+C.
Addiert werden Ziel, Quelle und die vorliegende (alte) Belegung des C-Flags. Danach erhalten das Zielregister das Ergebnis und die Flags einschließlich C dessen Merkmale als neuen Inhalt. Für die Subtraktion enthält der Befehlssatz entsprechendes. Ohne Einfluß ist die Belegung des C-Flags bei den Operationscodes 22 bis 27:
sub ziel,quelle
Ziel:=Ziel−Quelle.
Für das Berücksichtigen des Übertrags aus dem Subtrahieren niederer Bytes eignen sich die Operationscodes 32 bis 37:
sbc ziel,quelle
Ziel:=Ziel−Quelle−C.

Dr. Helmut Hoyer

(wird fortgesetzt)

(Fortsetzung zu 3.4.1 Arithmetik-Befehle)

Die fünfte arithmetische Verknüpfung basiert auch auf der Subtraktion. Sie läßt jedoch nicht nur die Quelle, sondern auch das Ziel unverändert. Das Ergebnis dieser Operation besteht einzig in der Flagbeeinflussung. Deren Belegung widerspiegelt den Vergleich (compare) von Ziel und Quelle entsprechend der Abb. 33 (Operationscodes A2 bis A6):

cp ziel,quelle
Ziel-Quelle

Das Ergebnis einer arithmetischen Verknüpfung (außer cp) kann mit dem Befehl **da** nachträglich dezimal korrigiert werden. Voraussetzung dafür ist, daß beide Operanden im gepackten BCD-Format vorlagen und die Flags seit der betreffenden Verknüpfung nicht verändert wurden. Dieser Einoperanden-Befehl (Operationscodes 4Ø und 41) erzeugt im Zielregister das gepackte BCD-Format und aktualisiert die Flags. Nur die Belegung von V läßt sich nicht sinnvoll auswerten.

da ziel
Dezimalkorrektur von Ziel

Vier weitere Einoperandenbefehle bewirken die Ausführung eines Zählschritts. Sie verändern nicht das C-Flag. Das Erhöhen eines Einbyte-Operanden gelingt mit den Operationscodes 2Ø, 21 und mit den Befehlen der gesamten Spalte E. Hier ist die Arbeitsregisteradresse Teil des Operationscodes:

inc ziel
Ziel:=Ziel+1.

Für die umgekehrte Zählrichtung gibt es nur die Befehle mit den Operationscodes ØØ und Ø1:

dec ziel
Ziel:=Ziel−1.

Das Zählen läßt sich mit den einzigen 16-Bit-Befehlen des Einchip-Mikrorechners auch auf Doppelregister anwenden. Die Operationscodes AØ und A1 betreffen das Erhöhen von 16-Bit-Operanden:

incw ziel
Ziel:=Ziel+1.

Für das Verringern solcher Wort-Operanden stehen die Operationscodes 8Ø und 81:

decw ziel
Ziel:=Ziel−1.

Wie bei 8-Bit-Prozessoren allgemein üblich, enthält der Einchip-Mikrorechner keine Multiplikations- und Divisionsbefehle. Die Punktrechenarten erfordern Unterprogramme, die mit Additions-, Subtraktions- und Rotationsbefehlen arbeiten.

Der Einchip-Mikrorechner-Schaltkreis U 883 enthält im inneren ROM Unterprogramme für die vier Grundrechenarten. Sie verarbeiten vorzeichenbehaftetes 16-Bit-Format. Mit call %ØØ7E wird die Addition gerufen (rr2:=rr2+rr4):

```
add r3,r5
adc r2,r4
jr ov, ERA
ret
```

Im Fehlerfall (Überschreitung des Zahlenbereichs von −32768 bis 32767) wird mit dem Sprung jr ov,ERA zu einem Programmteil verzweigt, das den Fehler für den Interpreter kenntlich macht. Sonst bewirkt ret die Rückkehr ins rufende Hauptprogramm (vgl. RETURN-Anweisung). Die Subtraktion (rr2:=rr2−rr4) beginnt auf der Speicheradresse %ØØ85:

```
sub r3,r5
sbc r2,r4
jr ov,ERA
ret
```

Die Multiplikation (rr2:=rr2∗rr4) ab %ØØBA und die Division (rr2:=rr2/rr4) ab %ØØEØ realisieren kompliziertere Programme, die auch die Arbeitsregister r6 bis r11 benutzen.

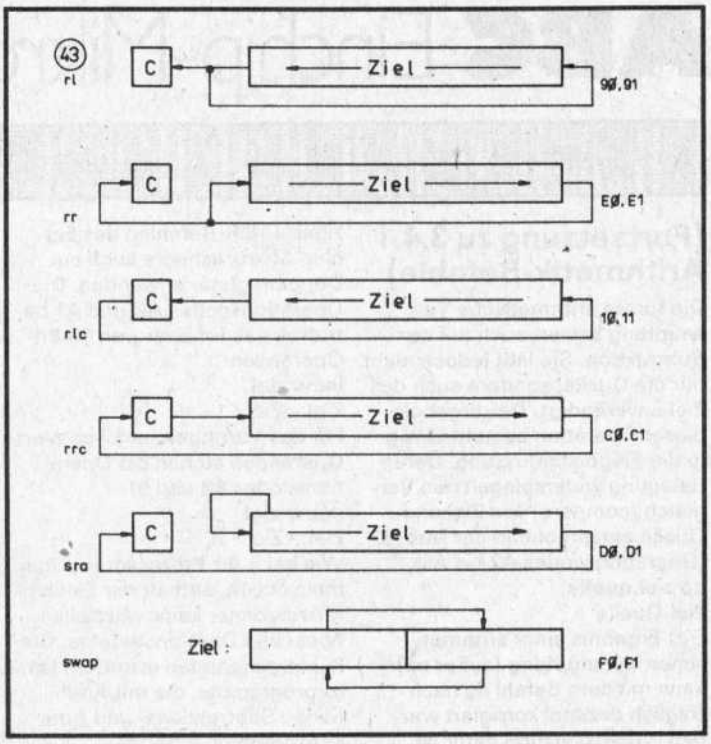
3.4.2. Rotations- und Schiebebefehle

Diese Befehle behandeln einen 8-Bit-Operanden, der als Register, Arbeitsregister, indirekt Register oder indirekt Arbeitsregister adressiert werden kann, als Schieberegister. Die Operationscodes befinden sich in den Spalten Ø und 1 der Abb. 32. Der Inhalt des Zielregisters wird durch diese Operationen um eine Bitposition nach links oder rechts verschoben. Dabei gelangt jede Information auf die Nachbarstelle, deren Stellenwert das Doppelte bzw. die Hälfte beträgt. Deshalb lassen sich Linksverschiebungen als Multiplikation mit 2 und Rechtsverschiebungen als Division durch 2 deuten. Abb. 43 gibt eine Übersicht der Rotations- und Schiebebefehle. Links stehen die Assemblerbezeichnungen, rechts die Operationscodes. Bei **rl** (rotate left) werden alle Bits nach links verschoben, das von Position D7 gelangt auf die Position DØ und in das Flagbit C. Bei **rr** (rotate right) kommt der Übertrag aus der Rechtsrotation von DØ nach C und D7. **rlc** (rotate left through carry) und **rrc** beziehen das C-Flag in die Rotation ein. Damit lassen sich Überträge aus vorherigen Operationen auf DØ bzw. D7 transportieren, während der neue Übertrag in C landet. Mit diesen Befehlen können also auch Verschiebungen über mehrere Bytes ausgeführt werden.

Zum Halbieren vorzeichenbehafteter Zahlen eignet sich **sra** (shift right arithmetical). Hier bleibt D7 (Vorzeichenbit) erhalten, während alles nach rechts rückt und

C wie gewohnt den Übertrag, hier aus DØ, erhält. Mit **swap** werden die beiden Tetraden des Ziels vertauscht, also D7 mit D3, D6 mit D2, D5 mit D1 und D4 mit DØ (vgl. Abb. 4). Während die übrigen Rotations- und Schiebebefehle alle vier interessierenden Flagbits normal aktualisieren, haben C und V nach swap keine sinnvolle Belegung.

Wie beim Addieren und Subtrahieren kann mit rrc und rlc der Übertrag aus vorherigen Operationen berücksichtigt werden, um Mehrbytezahlen zu verarbeiten. Beim Rechtsverschieben wandert der Übertrag jedoch nicht vom niederen zu höheren Positionen, sondern umgekehrt. Zum Halbieren der vorzeichenbehafteten 16-Bit-Zahl in rr2 eignet sich daher die Befehlsfolge:
sra r2
rrc r3
mit den Maschinencodes DØ E2 und CØ E3.



3.4.3. Logikbefehle

Bei logischen Verknüpfungen gibt es keine Überträge von einer Bitposition zur anderen. Hier bildet die ALU aus den beiden Bits der jeweils gleichen Position in Ziel und Quelle ein Ergebnisbit im Zielregister. So laufen bei jeder Befehlsausführung acht voneinander unabhängige Berechnungen ab.

Die Disjunktion (oder) ist erfüllt, wenn mindestens eines der beiden Operandenbits mit 1 belegt ist (Operationscodes 42 bis 47):

or ziel,quelle

Ziel:=Ziel ∨ Quelle.

Die Operation or erzeugt daher nur auf den Positionen Ø-Belegung, auf denen beide Operanden mit Ø belegt sind. Die Konjunktion (und) wird bei 1-Belegung beider Operandenbits erfüllt (Operationscodes 52 bis 57):

and ziel,quelle

Ziel:=Ziel ∧ Quelle.

Bei and entstehen nur auf den Positionen 1-Belegungen, auf denen beide Operanden mit 1 belegt sind. Die dritte logische Verknüpfung berechnet die Antivalenz (Ungleichheit, exklusive or). Sie ist erfüllt, wenn die beiden Operandenbits ungleich belegt sind (Operationscodes B2 bis B7):

xor ziel,quelle

Ziel:=Ziel ∨ Quelle.

Ø-Belegungen entstehen hier auf den Positionen, auf denen beide Operandenbits mit Ø oder beide mit 1 belegt sind. Neben diesen Verknüpfungen kann die ALU auch eine logische Einoperanden-Operation ausführen: die Inversion (Operationscodes 6Ø und 61):

com ziel

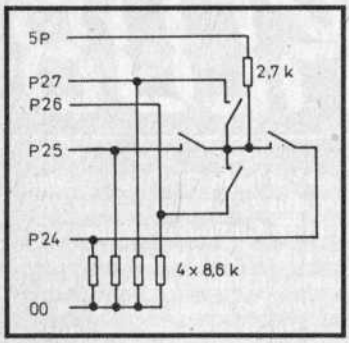
Ziel:=$\overline{\text{Ziel}}$.

Hier enthalten alle acht Bit des Zielregisters die entgegengesetzte Belegung des ursprünglichen Inhalts. Aus Ø wird 1 und aus 1 wird Ø. Wie die drei logischen Verknüpfungen beeinflußt com die Flags Z, S und V, während C unverändert bleibt. Da sich die zu verknüpfenden Operanden auf gleichen Positionen in Ziel und Quelle befinden müssen, kommen and, or und xor zur Berechnung binärer Logik selten zur Anwendung. Mit den Möglichkeiten der Bitmanipulation bietet der Einchip-Mikrorechner effektivere Methoden.

Aus arithmetischer Sicht berechnet com das Einerkomplement. Da negative Zahlen aber im Zweierkomplement dargestellt werden, entspricht das nicht der Negation. Sie erfordert zwei Befehle:
com r2
inc r2
negiert den Inhalt von Arbeitsregister 2. Für das 16-Bit-Format (rr2 := −rr2) eignet sich die Befehlsfolge:
com r2
com r3
incw rr2

3.4.4. Bitmanipulation

Bei Binärsteuerungen werden aus mit Ø oder 1 belegten Eingangsvariablen Ergebnisse berechnet, die ebenfalls solche Ja/Nein-Entscheidungen darstellen.



Zeichnungen: Liebig

Dazu müssen die Bits in den Registern des Einchip-Mikrorechners einzeln angesprochen werden. Die bereits vorgestellten logischen Verknüpfungsbefehle eignen sich zum Setzen, Löschen und Invertieren einzelner Bits. Gewöhnlich wird dabei ein Direktoperand als Quelle verwendet. Seine Belegung entscheidet, welche Bitpositionen von der Manipulation betroffen sind und welche unverändert bleiben.

and ziel,quelle löscht alle Zielbits, deren Positionen in der Quelle mit Ø belegt sind.

or ziel,quelle setzt alle Zielbits, deren Positionen in der Quelle mit 1 belegt sind.

xor ziel,quelle invertiert alle Zielbits, deren Positionen in der Quelle mit 1 belegt sind.

So lassen sich mit entsprechenden Quelloperanden einzelne oder mehrere Bits im Zielregister beeinflussen, um das Ergebnis von Steuerungsprogrammen zu speichern. Auch die Ports (Register Ø bis 3) können als Ziel adressiert werden, wenn sie als normale Ein- und Ausgänge initialisiert sind. Deshalb eignen sich die genannten Befehle auch zur Ausgabe auf einzelnen Anschlußstiften. Das setzt natürlich die Vereinbarung der betreffenden Bits als Ausgang voraus.

Zur Behandlung binärer Eingabedaten stehen zwei Testbefehle zur Verfügung, die das konjunktive (AND) und disjunktive (OR) Verknüpfen der Bits eines Registers gestatten. Sie benutzen die AND-Verknüpfung zweier 8-Bit-Operanden, wobei jedoch wie beim Vergleichsbefehl weder Ziel noch Quelle verändert werden. Das Ergebnis besteht einzig in der Neubelegung des Z-Flags.

tm ziel,quelle

Ziel ∧ Quelle

(Operationscodes 72 bis 77)

Das Z-Flag wird mit 1 belegt, wenn die AND-Verknüpfung von Ziel und Quelle auf allen Positionen ein Nullergebnis hat. Die Quelle, meist als Direktoperand n formuliert, legt in der Praxis fest, welche Bits im Zielregister bei diesem Test eine Rolle spielen.

Allgemein gilt für **tm:**

Z := NOR aller Zielbits, deren Positionen in der Quelle mit 1 belegt sind.

Man kann sich das anhand von Beispielen mit Bleistift und Papier klarmachen. Die AND-Verknüpfung dient nur der Auswahl der aktiven Zielbits. Mit der Logik zur Berechnung des neuen Inhalts vom Z-Flag erledigt die ALU die NOR-Verknüpfung.

Der zweite Testbefehl verwendet den negierten Inhalt des Zielregisters. Die Inversion und die AND-Verknüpfung finden nur ALU-intern statt, Ziel und Quelle bleiben unverändert.

tcm ziel,quelle

$\overline{\text{Ziel}}$ ∧ Quelle

(Operationscodes 62 bis 67)

Allgemein gilt für **tcm:**

Z := AND aller Zielbits, deren Positionen in der Quelle mit 1 belegt sind.

Zum Testen von Einzelbits muß ein Operand gewählt werden, der nur auf einer Position mit 1 belegt ist. Das Z-Flag erhält mit:

tcm 3,#2 (66 Ø3 Ø2)

den Inhalt von P31. Der tm-Befehl mit einem Quelloperanden, der nun ein mit 1 belegtes Bit enthält, wirkt wie der BIT-Befehl des U 880. So erhält das Z-Flag mit:

tm %1D,#%8Ø (76 1D 8Ø)

den negierten Inhalt des Bit D7 im Register %1D. Zur Realisierung von Binärsteuerungen mit Einchip-Mikrorechner folgt einer tm- oder tcm-Verknüpfung stets ein von Z abhängiger Sprung. Das sind die Befehle mit den Operationscodes 6B, 6D, EB und ED. Damit lassen sich die beiden möglichen Ergebnisse (Verknüpfung wahr bzw. falsch) mit verschiedenen Programmteilen behandeln.

Das Betriebsystem des JU+TE-Computers verwendet den Befehl:

or 3,#%8Ø (46 Ø3 8Ø)

zum Einschalten und den Befehl:

and 3,#%7F (56 Ø3 7F)

zum Ausschalten des über P37 ausgegebenen Synchronsignals für die Bilderzeugung. Den Tastenpiepton an P36 erzeugt der alle 64 µs ausgeführte Befehl:

xor 3,#%4Ø (B6 Ø3 4Ø)

Diese Ausgabeoperationen beeinflussen jeweils nur P37 bzw. P36 und lassen alle weiteren Port-3-Signale unverändert.

Beim Anschluß eines Steuerhebels (joystick) an Port 2 (vgl. Abb. Seite 229) können Ø, 1 oder 2 der vier Bits gleichzeitig mit 1 belegt sein. Sie lassen sich mit tcm leicht unabhängig von den übrigen Port-2-Signalen, die z. B. von einem zweiten Steuerhebel belegt sind, auswerten.

tcm 2,#%9Ø (66 Ø2 9Ø)

hat nur dann Z=1 zum Ergebnis, wenn gleichzeitig der obere und der rechte Kontakt geschlossen werden.

tcm 2,#%1Ø (66 Ø2 1Ø)

testet den rechten Kontakt unabhängig von allen anderen.

In TINY-MP-BASIC-Programmen können solche Tests mit der AND-Verknüpfung ($A) in IF-Anweisungen ausgeführt werden. Die Anweisungen:

```
15Ø IF GETR [2]$A%2Ø<>Ø
    THEN LET C=C-1
16Ø IF GETR [2]$A%1Ø<Ø
    THEN LET C=C+1
```

realisieren das Lenken im Programm „Autocross" (JU+TE 8/88, S. 630) mit dem Steuerhebel. Die Zeile 140 wird hierbei nicht benötigt.

Die angegebene Schaltung vermeidet übrigens gefährliche Lastströme auch bei Fehlinitialisierung von Port 2. Als Auslöseknopf kann ein fünfter Kontakt je Steuerhebel in gleicher Weise z. B. an die Pins P32 und P33 angeschlossen werden.

Dr. Helmut Hoyer

3.5. Transportbefehle

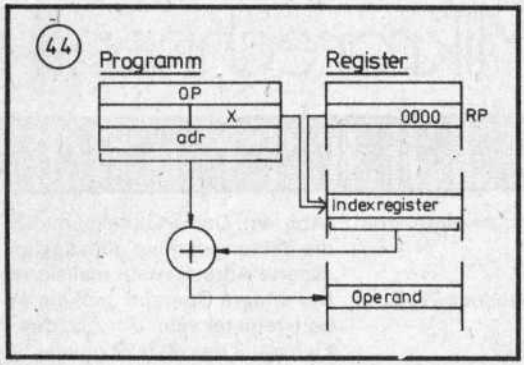
Transportbefehle dienen dem Informationsaustausch innerhalb des Registersatzes und zwischen den verschiedenen Speicherressourcen des Einchip-Mikrorechners. Sie arbeiten mit zwei Operanden: Ziel und Quelle. Die Funktion entspricht dem Kopieren. Das Ziel erhält den Inhalt der Quelle, die Quelle bleibt erhalten. Auf diese Weise werden die Informationen durch Transportbefehle vervielfältigt, was der umgangssprachlichen Bedeutung dieses Begriffes widerspricht. Die Flagbits bleiben grundsätzlich unverändert.

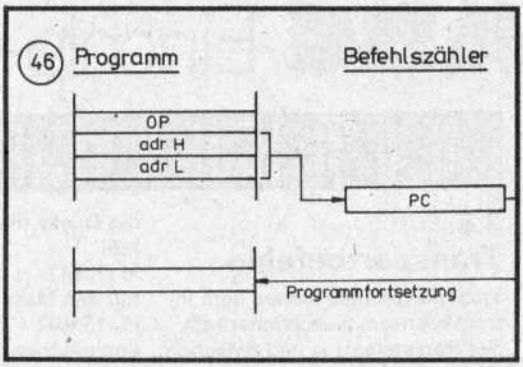
3.5.1. Ladebefehle

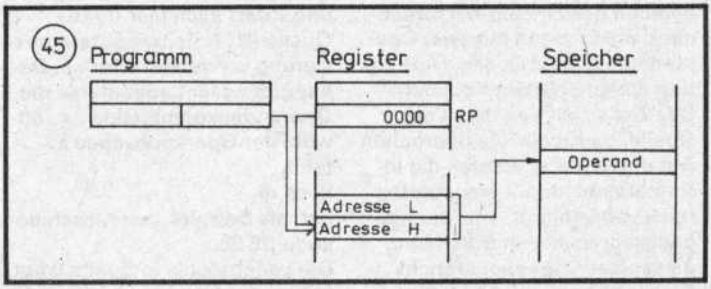
Wie die Verarbeitungsbefehle benutzen Ladebefehle Registerinhalte und Direktoperanden als Ziel und Quelle:
ld ziel,quelle
Ziel := Quelle
Die Operationscodes E3 bis E7 benutzen die bekannten Operandenkombinationen. Hinzu kommen F3 und F5, die das Ziel indirekt adressieren. Während für die Spalte 3 allgemein die Kombination r,Ir (Ziel: Arbeitsregister, Quelle: indirekt Arbeitsregister, **bitte in der Kopfzeile der Abb. 32 korrigieren!**) gilt, benutzt der Ladebefehl mit dem Operationscode F3 die Kombination Ir,r.
In der Spalte 8 der Abb. 32 (JU+TE 1/89, S. 69) sind Ladebefehle notiert, die als Ziel ein Arbeitsregister haben. Dessen Adresse ist Teil des Operationscodes, die Zeilennummer der Abb. 32 gleicht daher der Position des Zielregisters im Arbeitsregisterbereich. Das zweite Byte des Maschinencodes adressiert die Quelle (R). Der Assemblerbefehl
ld r7,%42
hat den Maschinencode 78 42,
ld r13,%42
entsprechend D8 42.
Wie in den Spalten 0, 1, 4, 5, 6 und 7 darf auch hier für die Quelle (R) Arbeitsregisteradressierung verwendet werden. Daher gibt es für Ladebefehle die Operandenkombination r,r , obwohl der Operationscode E2 fehlt.
ld rØ,r5
hat, als Beispiel, den Maschinencode Ø8 E5.
Die Ladebefehle in Spalte 9 enthalten die Arbeitsregister-Adresse der Quelle im Operationscode, während das Ziel mit dem zweiten Byte des Befehls als Register (R) festgelegt wird. Für
ld %15,r6
gilt der Maschinencode 69 15.
Hier kann das Ziel nicht als Arbeitsregister adressiert werden. Für das Laden von Arbeitsregistern mit Direktoperanden gibt es in der Spalte C ebenfalls Zweibyte-Befehle:
ld r9,#%4F
hat den Maschinencode 9C 4F.
Im Gegensatz zum Operationscode E6 werden nur 1,5 µs statt 2,5 µs Rechenzeit benötigt. So können bei allen bisher genannten Operandenkombinationen Befehle mit minimaler Ausführungszeit zum Einsatz kommen, wenn wenigstens einer der Operanden ein Arbeitsregister ist.
Die Operationscodes C7 und D7 verbinden für Quelle bzw. Ziel die Register- mit der indirekten Arbeitsregister-Adressierung. Beide Anteile werden in der ALU addiert. So ist die Adresse des betreffenden Operanden gleich dem Inhalt des Arbeitsregisters plus der im Befehl als drittes Byte enthaltenen 8-Bit-Zahl adr (Abb. 44). Damit läßt sich die für die Vektorrechnung günstige indizierte Adressierung realisieren. Der andere Operand muß ein Arbeitsregister sein. Um z. B. das Element 3 des ab %2Ø gespeicherten Vektors in das Arbeitsregister 7 zu laden, eignet sich, sofern im Arbeitsregister 4 die Zahl %2Ø steht, der Befehl
ld r7,3(r4)
mit dem Maschinencode
D7 74 Ø3. Da diese Adressierungsweise nur bei Ladebefehlen funktioniert, kommt sie in der Praxis sehr selten zur Anwendung.
Zum Löschen von Registerinhalten gibt es einen speziellen Ladebefehl:
clr ziel
Ziel := Ø
Da nur das Ziel zu adressieren ist, gehört er zu den Ein-Operand-Befehlen (Codes BØ und B1). Auch für das Laden des Registerpointers RP mit einem Direktoperanden existiert ein spezieller Operationscode (31):
srp quelle
RP := Quelle
Er gestattet, mit nur 1,5 µs Rechenzeit den Arbeitsregistersatz zu wechseln.

3.5.2. Stackbezogene Transporte

Auch der Stapelspeicher kann Ziel oder Quelle von Ladeoperationen sein. Dabei wird der Stackpointer SP zur Adressierung benutzt.
push quelle
SP := SP-1
@SP := Quelle
(Operationscodes 7Ø und 71).
Nach Verringern des Stackpointers erhält die mit dessen neuen Inhalt adressierte Speicherzelle im externen RAM bzw. im inter-







nen Registersatz (je nach Initialisierung des PØ1M) den Inhalt des Quellenregisters. In umgekehrter Richtung transportiert der pop-Befehl:

pop ziel
Ziel := @SP
SP := SP+1

(Operationscodes 5Ø und 51). Mit push und pop lassen sich Registerinhalte zeitweilig speichern, um zwischenzeitlich mit anderen Inhalten arbeiten zu können. Das eignet sich gut für das Retten des Registerpointers RP, um in Unterprogrammen Arbeitsregister-Adressierung anwenden zu können, auch wenn andere Bereiche als im Hauptprogramm genutzt werden. Das Vermeiden von Stapelfehlern erfordert das Ausgleichen jedes push-Befehls mit einem pop-Befehl vor dem nächsten Rücksprung (ret bzw. iret).

3.5.3. Datenaustausch mit externen Speichern

Für den Zugriff auf Dateien außerhalb des Registersatzes gibt es besondere Transportbefehle. Dabei wird die 16-Bit-Adresse, die für das Auffinden von Speicherzellen nötig ist, einem Doppelarbeitsregister entnommen. Die Adressierungsweise heißt daher Irr (Abb. 45). Für den Datenaustausch mit externem Daten- und Programmspeicher unterscheiden sich die Operationscodes.

Ide ziel,quelle
Ziel := Quelle

bezieht sich auf den Operativspeicher (RAM). P34 wird, sofern als $\overline{DM}$ benutzt, während der Transportoperation mit Ø-Pegel belegt. Beim Operationscode 82 ist ein Arbeitsregister Ziel, eine Speicherzelle Quelle. In umgekehrter Richtung überträgt der Befehl mit dem Operationscode 92. Im Gegensatz zu einigen Veröffentlichungen erwartet der Einchip-Mikrorechner hier als Operandenangabe die Adresse des Arbeitsregisters (Quelle) als höhere und die Adresse des Doppelarbeitsregisters (Adresse des Ziels) als niedere Tetrade des zweiten Bytes (vgl. Abb. 33, JU+TE 2/89). Alle acht Transportbefehle, die Speicherzellen als Ziel oder Quelle verwenden, benutzen das niedere Halbbyte zu deren indirekter Adressierung.

Idc ziel,quelle
Ziel := Quelle

bezieht sich auf den Programmspeicher. Die Operationscodes C2 und D2 wirken wie 82 und 92, nur das $\overline{DM}$ nicht aktiviert wird. Es läßt sich auch der innere ROM so adressieren.

Die Operationscodes 83, 93, C3 und D3 verwenden auch für den Operanden im Registersatz indirekte Adressierung (Ir). Dadurch ist es möglich, im Anschluß an den Transport die Adressen von Ziel und Quelle zu erhöhen. Das geschieht durch Inkrementieren des im zweiten Befehlsbyte angegebenen Arbeitsregisters und Doppelarbeitsregisters.

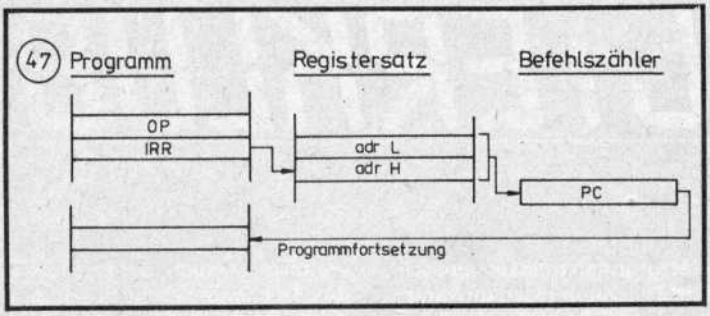
Idei ziel,quelle
Ziel := Quelle
Zieladresse := Zieladresse +1
Quelladresse := Quelladresse +1

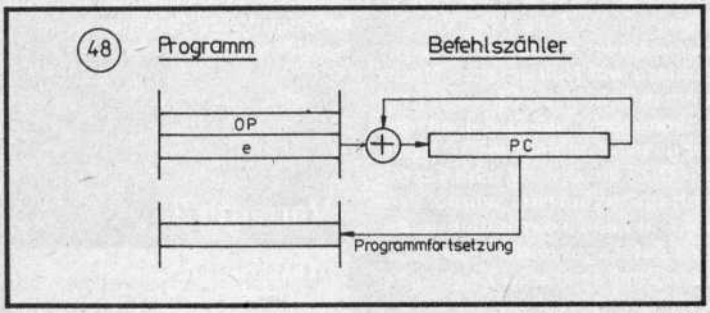
bezieht sich wie Ide auf den externen Datenspeicher,

Idci ziel,quelle
Ziel := Quelle
Zieladresse := Zieladresse +1
Quelladresse := Quelladresse +1

dagegen auf den Programmspeicher. In Verbindung mit dem djnz-Befehl gestatten sie einen schnellen Transport von Datenblöcken (7,5 µs je Byte) zwischen internen und externen Speicherbereichen.

Zur Ausgabe des Kammertones a über P36 (vgl. Abschnitt 2.3., JU+TE 10/88) eignet sich die Befehlsfolge
ld %F3,#%11





Zeichnungen: Liebig

ld %F2,#%8E
ld %F1,#%8A
mit 7,5 µs Rechenzeit. Bei Arbeitsregisteradressierung mit
srp #%FØ
ld r3,#%11
ld r2,#%8E
ld r1,#%8A
werden nur 6 µs benötigt.

3.6. Steuerbefehle

Für die Funktion des Einchip-Mikrorechners ist das Ausführen von Verarbeitungs- und Transportbefehlen in ganz bestimmten Folgen wichtig. Sie entsprechen nicht immer der Reihenfolge der Speicheradressen. Steuerbefehle gestatten durch Verändern des Befehlszählers PC, Programmsprünge auszuführen. Außerdem lassen sie zu, den Prozessor-Status zu beeinflussen, um z. B. die Annahme von Interrupts zu gestatten oder zu verbieten.

3.6.1. Programmsteuerbefehle

Der Programmablauf wird durch den Befehlszähler gesteuert. Das Eintragen einer Adresse in dieses Register des Steuerwerks (vgl. Abb. 5, JU+TE 8/88, S. 627) bewirkt die Programmfortsetzung ab dieser Adresse:

jp ziel
PC := Ziel

Als Ziel kann eine im Programm gegebene Konstante **adr** (Operationscode 8D, Abb. 46) oder der Inhalt eines Doppelregisters **IRR** (Operationscode 3Ø, Abb. 47) festgelegt werden. Die Wirkung entspricht den BASIC-Anweisungen GOTO 1Ø bzw. GOTO A. Von großer Bedeutung ist die Datenabhängige Programmverzweigung. Der Einchip-Mikrorechner nutzt dazu die Flagbits (vgl. Abb. 30 u. 31, JU+TE 12/88, S. 945). Die daraus abgeleiteten Bedingungen cc sind Teil des Operationscodes (1D bis FD, Abb. 32):

jp cc,ziel
PC := Ziel, wenn cc erfüllt

Für den Fall, daß die betreffende Bedingung nicht erfüllt ist, bleibt der Inhalt von PC unverändert. Dadurch kommt der nach dem Sprung im Programm stehende Befehl als nächster zur Ausführung.

Für Sprünge über eine kurze Distanz ($-128 \leq e \leq 127$) eignet sich die relative Adressierung (Abb. 48). Hier folgt dem Operationscode die Distanz **e**, die die Anzahl der zu überspringenden Bytes angibt. Es gelten sonst die gleichen Bedingungen, wie bei den Absolutsprüngen.

jr cc,e
PC := PC+e, wenn cc erfüllt

Der Vorteil der Relativsprünge (Codes 1D bis FB) liegt in dem um ein Byte geringeren Platzbedarf. Der unbedingte Sprung (8B) ist auch hier ein Spezialfall des bedingten:

jr e
PC := PC+e

Bei Rücksprüngen ist e negativ und muß im Zweierkomplement angegeben werden. Bei der Ausführung geht der Befehlszählerstand, der nach dem Lesen des Sprungbefehls entsteht, in Rechnung. Bei e=0 entsteht also die Adresse des nach dem Sprung im Programm notierten Befehls (ein praktisch wertloser Fall). Der Programmierer muß die Zahlen adr und e meist nicht selbst ermitteln. Symbolische Angaben (Marken) erlauben es dem Assembler, diese Zahlen zu berechnen. Das nennt man Binden (link).

Für Programmschleifen enthält der Befehlssatz des Einchip-Mikrorechners einen besonderen Befehl. Er dekrementiert ein Arbeitsregister und springt um die angegebene Distanz e, wenn dabei nicht der Arbeitsregisterinhalt Ø entsteht. Die Flags bleiben unverändert und dienen auch nicht als Bedingung.

Dr. Helmut Hoyer

Fortsetzung zu 3.6. Steuerbefehle

Die Arbeitsregisteradresse r ist Teil des Operationscodes (ØA bis FA):

djnz r,e

r := r−1;
PC := PC+e, wenn r ≠ Ø

Programmteile, die mehrfach genutzt werden können, lassen sich vorteilhaft als Unterprogramme formulieren. An den Punkten im Hauptprogramm, an denen ihre Dienste benötigt werden, notiert der Programmierer einen Rufbefehl. Bei der Ausführung wirkt das so, als stünde das ganze Unterprogramm an Stelle des Rufbefehls. Dadurch spart die Unterprogrammtechnik Programmspeicherkapazität.

call ziel

SP := SP−1
@SP := PC_L
SP := SP−1
@SP := PC_H
PC := Ziel

Der Rufbefehl lädt den Befehlszählerstand in den Stack und trägt dann die Zieladresse ein. Sie kann direkt im Programm stehen (adr, Operationscode D6) oder einem Registerpaar entnommen werden (IRR, D4). Die weitere Ausführung erfolgt wie nach jp ab der Zieladresse. Unterprogramme führen jedoch als letzten den Return-Befehl aus, der den Befehlszähler aus dem Stack lädt (Operationscode AF). Damit kehrt der Prozessor in das Hauptprogramm zurück:

ret

PC_H := @SP
SP := SP+1
PC_L := @SP
SP := SP+1

Das sichert die Programmfortsetzung mit dem Befehl, der nach dem call im rufenden Hauptprogramm steht. Der Einchip-Mikrorechner besitzt eine Interruptsteuerung, die das Aufrufen von Unterprogrammen (ISR) durch die Gerätetechnik gestattet. Näheres enthält das folgende Kapitel. Solche Unterprogramme benötigen den speziell angepaßten Rückkehrbefehl iret mit dem Operationscode BF:

iret

FLAGS := @SP
SP := SP+1
PC_H := @SP
SP := SP+1
PC_L := @SP
SP := SP+1

globale Interruptfreigabe.

Damit läßt sich alles wiederherstellen, was die Interrupt-Hardware zum Starten der ISR verändern mußte.

Ein Beispiel soll verdeutlichen, wie man mit dem djnz-Befehl Programmschleifen organisieren kann. Es transportiert 16 Bytes aus den Registern %4Ø bis %4F in den RAM-Bereich ab Adresse %F3EØ. Die Arbeitsregister rØ und r1 adressieren den externen RAM, r2 die internen Register. Zum Zählen der Schleifendurchläufe wird das Arbeitsregister r3 benutzt:

```
ld rØ,#%F3
ld r1,#%EØ
ld r2,#%4Ø
ld r3,#%1Ø
MA1: ldei @rØ,@r2
djnz r3,MA1
```

Da r3 mit dem Inhalt %1Ø (dez. 16) beginnt, kommt der ldei-Befehl 16 mal zur Ausführung. Dann läßt der djnz-Befehl nicht mehr auf Marke MA1 zurückspringen, da r3 dann auf Ø zurückgezählt wurde. In Maschinensprache sieht unser Teilprogramm wie folgt aus:

```
ØC F3
1C EØ
2C 4Ø
3C 1Ø
93 2Ø
3A FC
```

3.6.2. CPU-Steuerbefehle

Zu dieser Kategorie zählen Befehle, die den Zustand der zentralen Verarbeitungseinheit CPU beeinflussen. Beim Einchip-Mikrorechner handelt es sich im wesentlichen um die Festlegung, ob die Annahme von Interrupts gestattet oder verboten wird.

ei

globale Interruptfreigabe

Dieser Befehl mit dem Operationscode 9F initialisiert die Interrupt-Hardware und gestattet Interrupts bis auf Widerruf. Rücksetzen läßt sich dieser Zustand mit dem Operationscode 8F:

di

globale Interruptsperre

Außerdem werden Befehle zum Beeinflussen des Flagbits C zu den Steuerbefehlen gezählt.

rcf

C := Ø

hat den Operationscode CF und bewirkt nichts als eine Ø in die Position 7 des Flagregisters einzutragen.

scf

C := 1

(Operationscode DF) setzt das C-Bit auf 1. Mit dem Befehlscode EF erhält es das Entgegengesetzte des vorherigen Inhalts:

ccf

C := $\overline{C}$

Diese drei Befehle eignen sich gut zum Vorbereiten von Schiebeoperationen wie rrc und rlc (vgl. Abb. 43). Den Abschluß unserer Befehlsbeschreibung bildet der Befehl mit dem Code FF, der nichts bewirkt:

nop keine Operation

4. Interrupts

Interrupt heißt Unterbrechung. In der Rechentechnik bezeichnet man damit das zeitweilige Unterbrechen des „normalen" Programmablaufs zu dem Zweck, die Ausführung eines besonderen Programms einzufügen. Man nennt es Interruptserviceroutine (ISR). Nach deren Ende wird die Ausführung des unterbrochenen Programms fortgesetzt.

4.1. Anwendung

Man kann den Einchip-Mikrorechner gut ohne Verwendung der Interruptsteuerung einsetzen. Ihre Nutzung aber eröffnet viele Anwendungsmöglichkeiten, besonders beim Steuern zeitkritischer Prozesse. Der Vorteil von Interrupts liegt offenbar darin, den Zeitpunkt der Ausführung einer ISR unabhängig vom Programmablauf festlegen zu können. Das gibt die Möglichkeit, mit dem Einchip-Mikrorechner relativ schnell auf veränderte Situationen zu reagieren. Die Interruptverarbeitung wird daher vorwiegend dort angewendet, wo Ein-Ausgabe-Geräte zu bedienen sind. Dabei gibt es zwei Methoden:
Die erste eignet sich zum Realisieren eines Datentransports mit Handshake. Zu dem Zeitpunkt, zu dem Peripherie Bereitschaft meldet, wird per Interruptservice das nächste Datenbyte übergeben bzw. empfangen. Da Peripheriegeräte gewöhnlich viel langsamer arbeiten als der Einchip-Mikrorechner, bleibt bis zum nächsten Interrupt viel mehr Zeit, als die ISR für ein Byte benötigt. So kann der Prozessor andere Aufgaben ohne merkliche Kapazitätseinschränkung bearbeiten, während er quasi nebenbei eine oder mehrere Ein- oder Ausgaben tätigt. So entsteht das parallele Arbeiten verschiedener Geräte, was besonders bei mechanischer Peripherie (Lochbandleser, Lochbandstanzer, Drucker) beeindrucken kann.
Die zweite Methode beruht auf regelmäßig ausgelösten Interrupts. Sie eignet sich zum Realisieren von im festen Rhythmus auszulösenden Aktionen mit ISR. Beim JU+TE-Computer wird die Bilderzeugung mit einem Interruptservice alle 64 µs ausgeführt. In diesem Raster sind die Synchronimpulse zu erzeugen. Das erledigt der Prozessor scheinbar nebenbei, während er z. B. ein BASIC-Programm interpretiert. In Wirklichkeit kostet ihn die Bilderzeugung etwa 85 % seiner Rechenleistung. Deshalb muß man für die WAIT-Anweisung sieben statt der Millisekunde je Zählschritt kalkulieren. Die regelmäßigen Interrupts werden häufig für Zeit- und Frequenzmessungen, gelegentlich auch zur programmtechnischen Realisierung serieller Datenformate (vgl. Druckersteuerung, JU+TE-Computerklub 5/89 S. 376 ff.) angewendet.

4.2. Interruptanmeldung

Die Interruptlogik des Einchip-Mikrorechners leitet aus acht verschiedenen Quellen Interruptanmeldungen ab. Nur sechs davon können gleichzeitig wirken, je nachdem, ob das serielle Interface benutzt wird oder nicht. (Abb. 49). Die vier Anmeldungen (request) IRQØ bis IRQ3 sind äußeren Quellen zugeordnet. Sie folgen auf 1-Ø-Flanken an den vier Port-3-Eingängen. So können auch Port-Handshake-Signale mit ISR bedient werden. Für regelmäßige Interrupts eigenen sich die internen Timer. Beim Erreichen des Zählerstandes Null bewirken sie die Anmeldung von IRQ4 (TØ) und IRQ5 (T1).
Weitere innere Quellen sind die Fertigmeldungen des seriellen Interface (SIO). Wird es benutzt, dienen der Timer TØ als Zeitbasis und P3Ø als serieller Eingang. Ihre Anmeldungen hätten wenig Sinn. Daher erzeugen bei aktiver SIO weder eine 1-Ø-Flanke an P3Ø noch ein Zählerüberlauf bei TØ eine Interruptanmeldung. Statt dessen sind in diesem Fall IRQ3 dem SIO-Empfänger und IRQ4 dem seriellen Sender zugeordnet. Ein vollständig empfangenes und vom Pufferregister (Adresse %FØ) abzuholendes Zeichen setzt die Anmeldung IRQ3, ein vollständig gesendetes dagegen IRQ4. Ob SIO oder P3Ø und TØ Interuptanmeldungen erzeugen, entscheidet das Bit D6 im Port-3-Mode-Register P3M (Abb. 9).
Für jede Quelle existiert im Einchip-Mikrorechner ein Anmelde-Flipflop. Bei Erfüllung der jeweili-

(49) Interruptquellen:

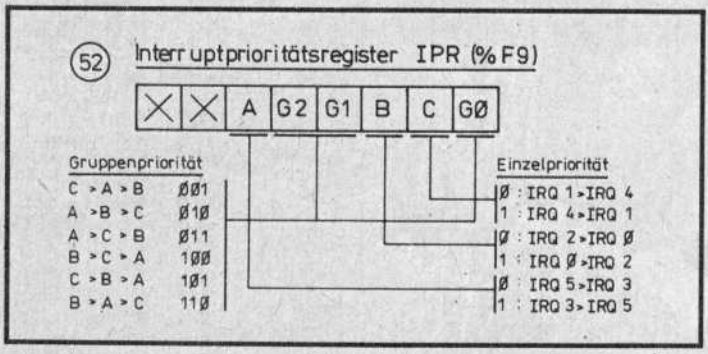
	Quelle	Verwendung
Ø	P32 1-Ø-Flanke	Handshake , universell
1	P33 1-Ø-Flanke	Handshake , universell
2	P31 1-Ø-Flanke	Handshake , universell
3	P30 1-Ø-Flanke	universell
	SIO - Eingabe	Synchronisation mit SIO
4	TØ - Zählende	Echtzeitgeber, Zählererweiterung
	SIO - Ausgabe	Synchronisation mit SIO
5	T1 - Zählende	Echtzeitgeber, Zählererweiterung

(50) Interruptanmelderegister IRQ(%FA)

		IRQ 5	IRQ 4	IRQ 3	IRQ 2	IRQ 1	IRQ Ø

(51) Interruptmaskenregister IMR(%FB)

EI		EI 5	EI 4	EI 3	EI 2	EI 1	EI Ø



gen Anmeldebedingung wird es gesetzt. Im Register %FA (IRQ, Abb. 50) sind alle sechs zusammengefaßt. Sie lassen sich auch programmtechnisch beeinflussen (setzen und löschen) und auswerten (testen). Die Positionen 6 und 7 sind nicht besetzt.
Für die Interrupt-Hardware speichert IRQ Interruptanmeldungen, bis die zugeordnete ISR gestartet wird. Damit fällt eine Anmeldung nicht unter den Tisch, nur weil die Interruptannahme vorübergehend gesperrt war. Allerdings besitzt hier jede Quelle nur ein Flipflop, so daß weitere Anmeldungen der gleichen Quelle, die vor dem Start der ISR eingehen, unberücksichtigt bleiben.

4.3. Interruptannahme

Das beschriebene Setzen der Bits DØ bis D5 im Register %FA (IRQ) erfolgt gerätetechnisch unabhängig vom Programmablauf und vom Zustand des Prozessors. Damit eine Interruptanmeldung auch zum Starten der ISR führt, müssen zwei weitere Bedingungen erfüllt sein. Das sind die globale und die individuelle Interruptfreigabe. Die betreffenden Flipflops enthält das Interruptmaskenregister %FB (IMR, Abb. 51). Zu jedem Anmeldeflipflop des IRQ gibt es hier ein Freigabeflipflop. Hinzu kommt auf Position 7 das globale Interruptfreigabeflipflop, das von den Befehlen ei und di beeinflußt wird. Das Flipflop D6 hat keine Wirkung.
Damit die dem Eingang P31 zugeordneten ISR gestartet wird, müssen also folgende drei Bedingungen erfüllt sein:
1. 1-Ø-Flanke am Anschluß P31, d. h. D2 im IRQ ist gesetzt
2. D2 im IRM steht auf 1
3. D7 im IRM steht auf 1

Wenn mehrere Interruptanforderungen gleichzeitig ihre jeweils drei Bedingungen erfüllen, tritt eine Rangfolge (Priorität) in Kraft, die die Reihenfolge der natürlich nicht gleichzeitig ausführbaren ISR festlegt. Sie wird vom Inhalt des Interruptprioritätsregister %F9 (IPR, Abb. 52) bestimmt. Jeweils zwei Quellen bilden die Gruppen A, B und C. Die Reihenfolge innerhalb der Gruppen regelt je ein Bit. Die Priorität der Gruppen untereinander wird von drei weiteren Bits (G2, G1 und GØ) festgelegt. Das gestattet sehr viele mögliche Rangfolgen, von denen der Programmierer eine auswählen muß. Aber nur bei sehr zeitkritischen Anwendungen des Einchip-Mikrorechners spielt es eine Rolle, welche er wählt. Die Belegung des IPR mit %Ø8 bewirkt z. B. die Rangfolge 5, 3, 2, Ø, 1, 4. Im Gegensatz zu IRQ und IMR kann das Prioritätsregister nur beschrieben, nicht aber gelesen werden.
Am Ende jedes Befehlslesens werden Anmeldungen, Freigabe und Prioritäten geprüft. Ist ein Interrupt angemeldet, individuell und global freigegeben und hat er von den angemeldeten und freigegebenen auch noch die höchste Priorität, führt der Prozessor den Interrupt-Annahmezyklus (IACK) aus. Zuerst wird der gelesene Befehl ausgeführt, dann der aktuelle Befehlszählerstand (PC, 2 Byte) und die Belegung des Flagregisters (%FC, 1 Byte) im Stapelspeicher abgelegt. Durch das Laden des Befehlszählers mit dem zugeordneten Interruptvektor (IVØ,...IV5, je nach Quellen) aus den untersten Adreßbereich des inneren Programmspeichers (Abb. 53) kommt die ISR als nächstes zur Ausführung. Mit dem vorherigen Löschen des betreffenden Anmeldebits im IRQ quittiert der Prozessor automatisch die Interruptannahme. Indem er außerdem das globale Freigabeflipflop im IRM löscht, sperrt er für die Dauer der ISR weitere Interruptannahmen.
Der ganze Annahmezyklus kostet nicht mehr als 6 µs Rechenzeit. Kommt schließlich am Ende der ISR der iret-Befehl zur Ausführung, wird fast alles wieder rückgängig gemacht. Nur das Anmeldeflipflop im IRQ bleibt gelöscht. So sichert die Hardware das Fortsetzen des unterbrochenen Programms, wenn nicht gerade die nächste Interruptanmeldung zu bedienen ist. In diesem Fall würde die Fortsetzung erneut ausgesetzt.

Dr. Helmut Hoyer

4.4. Initialisierung

Der Mechanismus der Interruptbehandlung ist recht kompliziert. Er gewährleistet eine hohe Flexibilität, erfordert aber entsprechend viele Festlegungen. Unterlassungssünden, die gerade bei Anfängern nicht selten sind, blockieren das Aufrufen der ISR oder lassen den Rechner einfach abstürzen. Dabei geht zum Glück nichts kaputt.
Das RESET-Signal löscht alle Anmeldeflipflops (IRQ) und die globale Freigabe (Bit D7 im IMR). Nach jedem Rücksetzen des Prozessors muß daher die Interruptannahme z. B. im Rahmen eines initialisierenden Programms vorbereitet werden. Veränderungen in Registern, die die Interruptverarbeitung beeinflussen, dürfen nur bei gesperrter globaler Freigabe (z. B. nach dem Befehl di) erfolgen. Zum Initialisieren gehören das Setzen der individuellen Freigabebits im IRM, das Festlegen einer Prioritätsfolge und als letztes die globale Interruptfreigabe. Sie muß das erste Mal nach jedem RESET mit dem Befehl ei erfolgen. Später darf sie mit der individuellen Freigabe in Form eines Ladebefehls bezüglich der Adresse %FB kombiniert werden.
Als Interruptvektoren IVØ bis IV5 auf den ersten zwölf Adressen des inneren ROM müssen die Startadressen der maximal sechs ISR stehen. Bei 40poligen Einchip-Mikrorechnern ist die Weiterverzweigung mit einem Sprungbefehl im Adreßbereich %Ø8ØØ bis %Ø811 üblich. Der JU+TE-Computer verwendet hier Register-indirekte Sprünge. Entsprechend sind vor der globalen Freigabe die betreffenden Register (z. B. %7E und %7F für T1-Interrupts) mit der Startadresse der ISR zu laden (vgl. Computerklub in JU+TE 3/89). Natürlich muß sich die betreffende Interruptserviceroutine auch tatsächlich ab dieser Adresse im Speicher befinden. Um das Fortsetzen des unterbrochenen Programms zu gewährleisten, darf sie nur mit Registerinhalten arbeiten, die dort nicht gebraucht werden. Grundsätzlich ist zu bedenken, daß die ISR aus der Sicht des vordergründig ausgeführten Programms an nicht vorhersehbaren Stellen unterbricht.

5. Beschaltung

Der Einchip-Mikrorechner enthält bereits viele Baugruppen, die ein Computer benötigt. Mit anwendungsspezifischer Software im inneren ROM müssen nur noch die Ein-/Ausgabe-Schaltungen angeschlossen werden, die der spezielle Prozeß (z. B. Waschautomat) erfordert. Neben der Stromversorgung braucht der Schaltkreis lediglich einen Rücksetzkondensator und eine Taktresonanz (Schwingquarz), schon ist der Rechnerkern arbeitsfähig.

5.1. Versorgung

Alle Signale beziehen sich auf den Masseanschluß GND (Pin 11 bei 40poligem, Pin 48 bei 64poligem Gehäuse). Die Versorgungsspannung Vcc (Pin 1 bzw. Pin 64) muß zwischen 4,75 und 5,25 V liegen und wird vom Einchip-Mikrorechner mit höchstens 200 mA belastet. Das Taktsignal XTAL dient u. a. dem Erzeugen einer zusätzlichen internen Betriebsspannung und darf daher nie fehlen. Die Stromaufnahme würde sonst auf ca. 300 mA ansteigen und eine gefährliche Erwärmung auslösen.
Der Einchip-Mikrorechner besitzt einen internen Taktgenerator, dessen Eingang XTAL1 und Ausgang XTAL2 bei der Standardvariante an den Pins zugänglich sind. Ein Schwingquarz (Frequenz: 1 MHz bis 8 MHz) zwischen diesen Anschlüssen ergänzt den Taktgenerator mit der fehlenden Taktresonanz. Zwei kleine Kondensatoren (ca. 20 pF) zwischen XTAL1 bzw. XTAL2 und Masse verbessern dessen Eigenschaften.
Schaltkreise der Power-down-Variante (vgl. Abschnitt 5.5.) besitzen keinen Taktausgang XTAL2. Hier wird ein extern aufgebauter Generator an XTAL1 angeschlossen. Abweichend vom TTL-Standard muß der 1-Pegel des von außen zugeführten Taktes mindestens 3,8 V betragen. Auch die Standardvariante des Einchip-Mikrorechners erlaubt einen externen Taktgenerator, das Signal XTAL2 bleibt dann unbeschaltet.

5.2. Anschlußbedingungen

Für alle Anschlüsse des Einchip-Mikrorechners außer XTAL, die in Abb. 5 (JU+TE 8/88, S. 627) dargestellt sind, gelten die TTL-Spannungspegel. Eingänge stellen eine rein kapazitive Last (max. 10 pF) dar. Bei der Schaltungsentwicklung ist darauf zu achten, daß nie Spannungen unterhalb des Massepotentials oder oberhalb der Versorgungsspannung Vcc an den Eingängen des Einchip-Mikrorechners auftreten können. Eingangsspannungen bis 0,8 V bewertet der Einchip-Mikrorechner mit Ø, ab 2,0 V mit 1.
Die Ausgänge realisieren den TTL-Pegel (Ø: bis 0,4 V, 1: ab 2,4 V) bei Laststrømen gegen Masse bis 0,25 mA und gegen die

Versorgungsspannung bis 2 mA. Das reicht zum Treiben einer TTL-Standardlast oder von bis zu sieben Low-Power-Eingängen. Stärkere Ströme können die Ausgangsstufen zerstören. Die Lastkapazität darf bis zu 100 pF betragen. **Abb. 54** zeigt Transistorschaltungen als Ausgabeverstärker mit Basisvorwiderständen. Die zusätzlichen Ausgänge der 64poligen Schaltkreise zum Anschluß eines EPROM statt des inneren ROM gestatten nur Lastströme bis 0,1 mA bei 1-Pegel und 1 mA bei Ø-Pegel. Das ist für diesen Zweck völlig ausreichend.

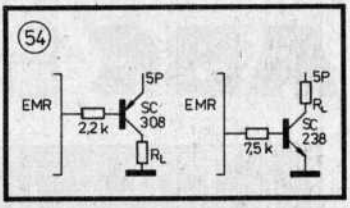
5.3. Rücksetzen

Zum definierten Start der Programmausführung nach dem Einschalten und als Neustart besitzt der Einchip-Mikrorechner den $\overline{\text{RESET}}$-Eingang (vgl. Kapitel 1). Hat er nach eingeschwungener Spannungs- und Taktversorgung für mindestens 18 Taktperioden Nullpegel, erzeugt er im Inneren des Einchip-Mikrorechners einige festgelegte Zustände. Dazu gehört das Ausschalten der Timer (TØ, T1), des seriellen Interface (SIO), der Interruptanmeldungen (IRQ) und der globalen Interruptfreigabe im IMR, das Vereinbaren aller Port-Ø-, Port-1- und Port-2-Signale als Eingang sowie das Laden des Befehlszählers mit %ØØØC. Erreicht der $\overline{\text{RESET}}$-Eingang 1-Pegel, beginnt die Programmausführung ab Adresse %ØØØC (im inneren ROM) mit passiver Peripherie- und Interruptsteuerung.

Im Schaltkreis verbindet ein 100-k-Widerstand das $\overline{\text{RESET}}$-Signal mit der Versorgungsspannung Vcc. Für das Rücksetzen beim Einschalten reicht ein Kondensator (1µF) zwischen Masse und $\overline{\text{RESET}}$, der mit diesem Widerstand allmählich 1-Pegel erzeugt. Eine Rücksetztaste kann dazu parallelgeschaltet werden.

5.4. Test-ROM

Alle 40poligen Einchip-Mikrorechner aus DDR-Produktion enthalten für beim Hersteller nötige Testzwecke einen kleinen ROM-Bereich, der statt der ersten 64 Byte des inneren Programmspeichers aktiviert werden kann. Hier weisen die Interruptvektoren der Reihenfolge nach auf die Adresse %Ø8ØØ, %Ø8Ø3, ... , %Ø8ØF (U 8611 DC: %1ØØØ, ...), den Beginn des externen Programmspeichers. Ein kurzes Programm ab %ØØØC im Test-ROM konfiguriert Port Ø, Port 1 und P34 für den Anschluß des externen Speichers und springt auf Adresse %Ø812 (U 8611 DC: %1Ø12). Das gestattet, alle 40poligen Einchip-Mikrorechner mit einem im äußeren Bereich (EPROM) untergebrachten Anwenderprogramm uneingeschränkt und unabhängig vom inneren ROM zu nutzen.
Zum Aktivieren dieses alternativen Programmspeichers muß das $\overline{\text{RESET}}$-Signal vom Ø-Pegel innerhalb einer Taktperiode auf einen Spannungswert zwischen 7,3 V und 8 V gebracht werden. Typen, deren innerer ROM nicht nutzbar



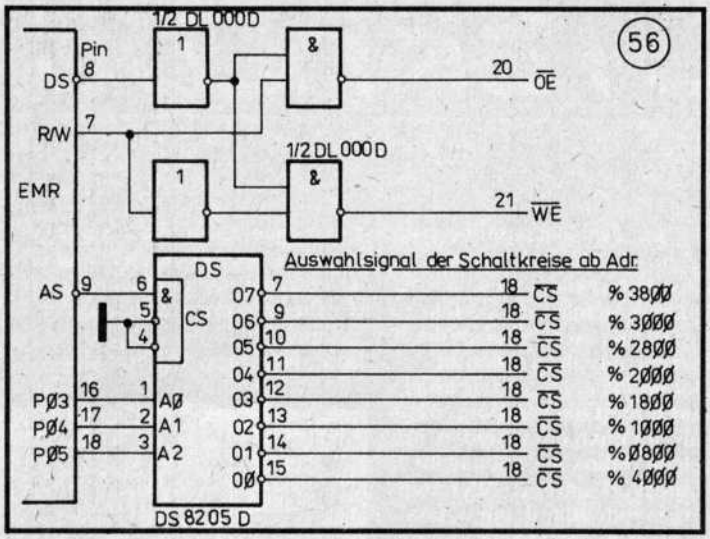
ist (siehe Abb. 55), lassen sich ausschließlich auf diese Weise einsetzen. Der UB 8830 D (UB 8831 D) besitzt auch bei „normalem" Rücksetzen die genannten Interruptvektoren und springt auf die Adresse %Ø812, wenn dort kein RAM bestückt ist und P32 keine Verbindung zu P35 hat. 64polige Typen lassen bereits mit dem EPROM-Anschluß statt des inneren ROM beliebige Anwenderprogramme zu und benötigen daher keinen Test-ROM.

5.5. Power-down-Variante

Beim Verbinden (Bonden) des Siliziumkristalls (chip) mit den Anschlußstiften durch feine Drähtchen bestimmt der Hersteller, ob der Ausgang des internen Taktgenerators oder statt dessen die Versorgungsspannung des Universalregistersatzes mit dem Pin XTAL2/VMM verbunden wird. Jeder konkrete Schaltkreis bietet daher entweder die Möglichkeit, den Takt intern zu erzeugen oder getrennte Stromversorgung der Universalregister. Die Bezeichnung power down (geringe Leistung) weist auf die Hauptanwendung der zweiten Bondvariante. Durch Batteriestützung kann der Inhalt der Universalregister bei ausgeschaltetem Rechner bewahrt werden.
Voraussetzung für den Datenerhalt ist das Nullsetzen des $\overline{\text{RESET}}$-Einganges vor Unterschreiten der unteren Betriebsspannungsgrenze von Vcc = 4,75 V. Während der Rechner ausgeschaltet ist, muß $\overline{\text{RESET}}$ ständig auf Ø liegen und an XTAL2/VMM eine Spannungsversorgung von mindestens 3 V gesichert sein. Diese „Schlafspannung" belastet der Einchip-Mikrorechner mit ca.

55

ROM-Ausstattung	mit internem Taktgenerator	Power-down-Variante
40-polig:		
2K-Anwender-ROM	UB 8810 D	UB 8811 D
2K-BASIC-Interpreter	UB 8830 D	UB 8831 D
2K-ROM, nicht nutzbar	UB 8860 D	UB 8861 D
4K-Anwender-ROM	UB 8611 DC	UL 8611 DC
4K-ROM, nicht nutzbar	U 8611 DC/1	UL 8611 DC/1
64-polig:		
2K-Entwicklungsversion	UB 8820 M	UB 8821 M
4K-Entwicklungsversion	UB 8840 M	UB 8841 M



Zeichnungen: Liebig

10 mA. Das erfordert eine recht komplizierte Beschaltung. Eine Batteriestützung für CMOS-RAM als externe Speicher ist weniger aufwendig und daher vorzuziehen. Beim Betrieb ohne Datenerhalt im ausgeschalteten Zustand wird das Pin XTAL2/VMM von Schaltkreisen der Power-down-Variante mit Vcc verbunden.

5.6. Typenübersicht

Abb. 55 gibt eine Übersicht der Einchip-Mikrorechner-Schaltkreise. Sie unterscheiden sich hauptsächlich durch den inneren ROM. Einen universell nutzbaren Inhalt hat der innere ROM des UB 883Ø D (UB 8831 D). Neben dem TINY-MP-BASIC-Interpreter enthält er ein Hardware-Testprogramm und eine Interruptserviceroutine, die den direkten Zugriff auf extern angeschlossene Speicher und Ein-/Ausgabe-Baugruppen anderen Busverwaltern gestattet (DMA).
Bei Schaltkreisen mit 2 Kbyte ROM beginnt der äußere Bereich ab der Adresse %ØØ8ØØ, bei den mit 4 Kbyte ab %1ØØØ. Die Entwicklungsversionen fertigt das VEB Kombinat Mikroelektronik „Karl Marx" in Erfurt in einem 64poligen Quadro-in-line-Plastgehäuse. Neben den 40 standardmäßigen (vgl. Abb. 5) sind das zwölf Adreßausgänge (AØ bis A11, A11 nur bei 4K-Variante), acht Dateneingänge (DØ bis D7) und vier Steuersignale (MDS, SYNC, SCLK, IACK), die beim Einsatz als Prozeßrechner unbeschaltet bleiben. Das Programm und ggf. die Interruptvektoren enthält ein EPROM, dessen Adreß- und Datenleitungen an die gleichnamigen des Einchip-Mikrorechners angeschlossen werden. CS und OE erhalten Massepotential. Das sichert die Nutzbarkeit von ldc- und ldci-Befehlen sowie der Interruptverarbeitung. Vpp des U 2716 C wird mit Vcc verbunden.
Das B als zweiter Buchstabe der Typenbezeichnung steht für den Standard-Taktfrequenzbereich von 1 MHz bis 8 MHz. Ein C kennzeichnet Schaltkreise mit einer maximalen Taktfrequenz von 5 MHz, ein D von 3,6 MHz.

5.7. Externe Speicher

Zum Realisieren von externen Speicherresourcen (vgl. Abb. 26, JU+TE 11/88, S. 875) müssen die Ports Ø und 1 sowie die Signale AS, DS und R/W als externer Bus genutzt werden. Die entsprechende Verwaltung durch den Einchip-Mikrorechner bewirkt das Laden des Registers PØ1M (%F8, Abb. 13, JU+TE 9/88, S. 712) mit einer geeigneten Konstanten. Wie bei 16-Bit-Rechnern dienen die Datenleitungen zu Beginn jedes Speicherzugriffs der Ausgabe von Adreßbits. Um die Adresse für den gesamten Zyklus bereitzustellen, ist ein Auffangregister nötig. Dafür eignen sich die Schaltkreise DS 8212 D, DS 8282 D und DL 374 D. Das Signal AS sichert als Takt die Übernahme der Adreßbits zum richtigen Zeitpunkt, die Ausgänge bleiben ständig aktiv.
Mit den Steuersignalen DS und R/W des Einchip-Mikrorechners lassen sich die standardisierten Freigabesignale OE und WE leicht erzeugen (Abb. 56). Sie werden wie die Adreß- und Datenleitungen allen Speicherschaltkreisen parallel angeboten. Das Unterscheiden zwischen den einzelnen erfolgt allein mit dem CS-Eingang. Das gestattet das optimale Nutzen der Zugriffszeiten. Die Abb. 56 realisiert mit dem Dekoder DS 8205 D das Erzeugen dieser Auswahlsignale für acht Schaltkreise mit je 2 Kbyte Kapazität. Es können wahlweise EPROM U 2716 C oder statische RAM U 6516 D bzw. pinkompatible Importe bestückt werden. Das Einbeziehen von AS bei der CS-Erzeugung sichert die für Typen mit Adreßpuffer nötige Taktsteuerung. Beim Zugriff auf den internen Bereich wird DS nicht aktiv, so daß die zugeordneten Ausgänge des Dekoders für den Adreßbereich ab %4ØØØ taugen. Bei Einchip-Mikrorechnern mit 4 Kbyte innerem ROM betrifft das auch Pin 14 des Dekoders, der dann der Adresse %48ØØ zugeordnet werden kann.
Der Anschluß dynamischer RAM-Schaltkreise erfordert eine aufwendigere Beschaltung und eine Interruptserviceroutine. Deshalb kommen sie nur selten zur Anwendung.

(Schluß) **Dr. Helmut Hoyer**

Weiterführende Literatur

Kieser, Bankel: Einchipmikrorechner, VEB Verlag Technik, Berlin 1986