Peter Heiß

Z80 Maschinen-sprachekurs

für den Schneider CPC 464/664/6128

Peter Heiß

Z80 Maschinensprachekurs für den Schneider CPC 464/664/6128

Mit 25 Abbildungen,
13 Tabellen,
27 Beispielprogrammen,
43 Übungsaufgaben mit Lösungen
und einem Direkt-Assembler

Verlag Heinz Heise GmbH

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Heiß, Peter:
Z80 Maschinensprachekurs (Z-achtzig Maschinensprachekurs)
für den Schneider CPC 464, 664, 6128 / Peter Heiß. – Hannover: Heise, 1986.

ISBN 3-922705-28-6

Printed in W. Germany

Aufl.: 5 4 3 2 1
Jahr: 1990 89 88 87 86

Umschlaggestaltung: Wolfgang Ulber, Hannover
Druck: HAHN-Druckerei, Hannover

ISBN 3-922705-28-6

Inhalt

Vorwort

Ein Buch über Maschinensprache kann viele Ziele haben: Es können die Befehle eines Mikroprozessors in allen Einzelheiten untersucht werden, die Programmstruktur von Maschinenprogrammen kann im Mittelpunkt stehen, oder die Hardware und das Betriebssystem eines speziellen Computers können unter die Lupe genommen werden. Ein einführendes Buch wird keines dieser Ziele vollständig erreichen, andererseits muß es von jedem dieser Aspekte etwas beinhalten.
Dem Leser dieses Buches soll eine vollständige Übersicht über den Befehlsvorrat des Z80-Mikroprozessors gegeben werden. Gleichzeitig soll er einen ersten Eindruck von der Methodik der Programmierung in Maschinensprache gewinnen. Dazu ist es wichtig, in jedem Kapitel die Demonstrationsprogramme durchzuarbeiten, die Wirkung und Anwendung der neu zu erlernenden Maschinenbefehle zeigen sollen. Daß Durcharbeiten mehr heißt als das bloße Eintippen, ist klar. Daher steht in jedem Kapitel die detaillierte Beschreibung des jeweiligen Demonstrationsprogramms im Mittelpunkt. Eine Zusammenfassung soll als schnelle Wiederholungsmöglichkeit früherer Kapitel dienen. Der Inhalt der „Zusatzinformationen“ kann beim erstmaligen Durcharbeiten überschlagen werden.
Da der CPC in diesem Buch die Rolle eines Übungsgeräts spielt, sind einige Kenntnisse seiner Hardware und Firmware notwendig. Im Zusammenhang mit der Ein/Ausgabe- und Interruptprogrammierung nimmt die Beschreibung der verschiedenen Bausteine des CPC (Gate-Array, Video-Controller, Tongenerator) einen relativ breiten Raum ein. Firmware-Routinen werden in vielen Demonstrationsprogrammen genutzt, und der Besprechung der Sprungtabellen für die ROM-Routinen ist ein eigenes Kapitel gewidmet. Trotzdem hat die Beschreibung der allgemeinen Eigenschaften des Z80-Prozessors insgesamt Vorrang vor einer zu intensiven Besprechung von speziellen Tips und Tricks für den CPC.
Als das Buch begonnen wurde, existierte erst der CPC464. Während der Entstehungszeit kamen zunächst der CPC664 und schließlich der CPC6128 dazu. Da alle diese Computer den Z80A-Prozessor besitzen, wurde versucht, das Buch so zu gestalten, daß es von den Besitzern aller drei CPC-Typen gleichermaßen verwendet werden kann. Das war überraschend einfach. Zwar gibt es Unterschiede

beim Betriebssystem und beim BASIC-Interpreter. Die wichtigsten Adressen (insbesondere die Firmware-Ansprungtabellen) und Ports wurden aber gegenüber dem CPC464 beim CPC664 und beim CPC6128 nicht verändert. Die meisten Demonstrationsprogramme des Buchs laufen daher ohne jede Anpassung auf allen drei Typen. In den anderen Programmen muß bei der Verwendung des CPC664 bzw. CPC6128 meist nur eine Adresse geändert werden. Diese Änderungen sind an den entsprechenden Stellen im Text und in den Programmen in Klammern angegeben. Die Angaben ohne Klammern gelten, wenn nichts anderes gesagt wird, für den CPC464.

Peter Heiß

Korschenbroich, im Januar 1986

Lohnt sich Maschinen-sprache überhaupt?

1

Als Computerbesitzer haben Sie sich sicher schon einmal über eine Zeitungsüberschrift wie „Computer irrte sich ...“ amüsiert. Natürlich wußten Sie, daß sich der Computer bestimmt nicht geirrt hatte; er war entweder defekt, oder der Programmierer hatte sich geirrt. Irren kann sich ein denkendes Wesen; ein Computer ist lediglich eine komplizierte Ansammlung von Schaltern, deren Stellungen sich gegenseitig beeinflussen können. Allerdings sind diese Beeinflussungsmöglichkeiten derart vielfältig, daß ein Computer bei geeigneter Programmierung manche Tätigkeiten nachvollziehen kann, die man eigentlich nur denkenden Wesen zutraut: Rechnen, Schachspielen oder das Steuern einer Maschine. Daher ist es in vielen Fällen einfach bequemer, in einem Computer ein denkendes Wesen zu sehen. Auch wenn man die Gesamtheit der Beeinflussungsmöglichkeiten der Schalter eines Computers mit dem Wort Maschinen„sprache“ bezeichnet, stellt man sich unbewußt auf diesen Standpunkt. In diesem Sinne soll auch der folgende Vergleich verstanden werden:

Jeder, der anfängt, sich mit einem Computer zu beschäftigen, ist in einer ähnlichen Situation wie ein Reisender, der ein fremdes Land besuchen will. Um sich mit den Einwohnern dieses Landes unterhalten zu können, gibt es im wesentlichen zwei Möglichkeiten: Entweder man lernt ihre Sprache, oder man verläßt sich auf Leute, die die eigene in die fremde Sprache übersetzen können. Beide Wege der Verständigung haben Vor- und Nachteile: Das Erlernen der fremden Sprache dauert längere Zeit, außerdem ist es oft genauso wichtig, die Denkgewohnheiten und Gebräuche der Fremden kennenzulernen. Die damit verbundene Mühe wird sich nur lohnen, wenn bei der Reise eine lange und intensive Beschäftigung mit den Fremden geplant ist. Für eine kurze Urlaubsreise ist meist der andere Weg angebracht. Ein Dolmetscher nimmt dem Reisenden viel Arbeit ab. Allerdings kostet dann jede Unterhaltung mit den Fremden mehr Zeit, und der Kontakt bleibt gezwungenermaßen oberflächlich. Außerdem sind Dolmetscher nicht immer zuverlässig.

Natürlich haben Sie die Parallele sofort bemerkt. Der Computerdolmetscher ist der BASIC-Interpreter (oder der Interpreter bzw. Compiler für eine andere „Hochsprache“). Denn BASIC ist unserer natürlichen Sprache sehr ähnlich. (Da es in Amerika entwickelt wurde, kommt es wegen seines Wortschatzes man-

chem allerdings etwas fremd vor.) Und der BASIC-Interpreter übersetzt unsere Sprache in die eigentliche Computersprache. Für gelegentliche Ausflüge ins Computerland reicht das aus. Aber das Übersetzen kostet Zeit. In extremen Fällen dauert ein BASIC-Programm mehrere hundert Mal so lang wie ein entsprechendes Maschinenprogramm. Außerdem ist der BASIC-Interpreter nicht immer ganz zuverlässig. Manche Dinge übersetzt er dem Computer gar nicht oder nicht richtig. Schauen wir uns ein Beispiel an.

Drei POKEs legen den Computer lahm

Sie wissen, daß Sie beim CPC ein BASIC-Programm mit der ESC-Taste unterbrechen können. Darüber hinaus versetzt die CTRL/SHIFT/ESC-Tastenkombination den Computer in seinen Einschaltzustand. Welche Tasten gerade gedrückt werden, untersucht der Computer während regelmäßiger Programmunterbrechungen, die auch als „Interrupts" bezeichnet werden. In gewissen Situationen, z. B. wenn der CPC464 eine Kassette liest oder beschreibt, finden keine Unterbrechungen statt. Das Drücken einer Taste wird vom Computer dann nicht registriert, und auch die CTRL/SHIFT/ESC-Kombination hat keine Wirkung. (Das gilt nur für den eigentlichen Lesevorgang und nicht für die ganze Zeit, in der sich das Band bewegt.) Nun gibt es in BASIC einen Befehl mit der Bezeichnung DI. Das heißt: „Disable Interrupt" („mache einen Interrupt unmöglich"). Wenn wir das Programm

```
10 DI
20 GOTO 20
```

starten, sollte der Computer unrettbar (d. h. bis zum Ausschalten) in einer Dauerschleife gefangen sein. Trotzdem läßt sich das Programm mit ESC oder SHIFT/CTRL/ESC ohne weiteres anhalten: Der Interpreter hat unser Kommando DI gar nicht so ernst genommen und es somit nicht richtig an den Computer weitergegeben. Jetzt geben Sie einmal folgende Zeilen ein:

```
10 POKE 1000,&F3:POKE 1001,&18:POKE 1002,&FE
20 CALL 1000
```

Wenn Sie jetzt RUN eingeben, haben Sie den Computer wirklich in einer Dauerschleife gefangen. Kein Tastendruck hält das Programm an. Die drei POKEs in Zeile 10 bedeuten dasselbe wie das obenstehende BASIC-Programm. Nur haben Sie dem Computer die Anweisung in Maschinensprache gegeben, ohne durch den Interpreter bevormundet zu werden. Die eben verwendete Methode, Maschinenprogramme einzugeben, werden wir später noch ausführlich behandeln.
Ganz nebenbei sind Sie gerade einer der wichtigsten Eigenschaften der Maschinenprogrammierung begegnet. Als zukünftiger Maschinenprogrammierer

arbeiten Sie ohne Netz und doppelten Boden. Nach Testläufen Ihrer Programme wird der Griff zum Ausschalter genauso geläufig werden wie das völlig unerwartete Auftauchen der Schneider-Begrüßungsmeldung auf dem Monitor und das damit verbundene Verschwinden Ihres Programms aus dem Speicher. Programmierfehler führen bei Maschinenprogrammen nicht zu Fehlermeldungen, sondern meist zu sogenannten Systemabstürzen: Der Computer führt entweder sinnlose Befehlssequenzen durch, die nicht mehr unter Kontrolle des Betriebssystems stehen und daher auch nicht mehr unterbrochen werden können, oder die Befehlskette führt schließlich zum Startprogramm und damit zum Löschen des kompletten Programmspeichers. Häufiges Speichern der in der Entwicklung befindlichen Programme ist das einzige Gegenmittel, das hier hilft.
Die Vorteile, die dieses riskante Programmieren mit sich bringt, sind im wesentlichen genannt: Maschinenprogramme sind extrem schnell, benötigen relativ wenig Speicherplatz und sind im Prinzip in der Lage, alle Möglichkeiten des Computers auszuschöpfen, auch Möglichkeiten, die der BASIC-Interpreter bewußt oder unbewußt außer acht läßt.

Was ist eine Z80A-CPU?

Das „Herz“ oder „Gehirn“ des CPC-Computers ist eine Z80A-CPU. CPU heißt „Central Processing Unit“ oder kurz „Zentraleinheit“. Z80A ist eine Typenbezeichnung: Z steht für den Firmennamen Zilog, die 8 in 80 gibt an, daß es sich um einen 8-Bit-Prozessor handelt (darüber gleich mehr), und der Buchstabe A bedeutet, daß man es mit einer schnellen Version dieses Prozessors zu tun hat. Diese CPU ist weit verbreitet: Die kleinen Sinclair-Computer ZX-81 und ZX-Spectrum sind genauso damit ausgerüstet wie sämtliche Computer, die dem neuen MSX-Standard entsprechen.
Der Grund für die weite Verbreitung des Z80A ist der umfangreiche Befehlsvorrat. Über 600 Befehle enthält seine Maschinensprache. Erschrecken Sie nicht über diese Zahl. Viele dieser Befehle lassen sich in Gruppen zusammenfassen. Kennt man einen Befehl dieser Gruppe, so kann man auch mit den anderen umgehen. Andere Befehle haben sehr spezielle Aufgaben, man kommt zunächst auch ohne sie aus. Es ist wie bei einer richtigen Fremdsprache: Um einen Engländer nach der Zeit zu fragen, braucht man kein Shakespeare-Experte zu sein. Sie werden sehen: Das Erlernen der wichtigsten Vokabeln der Maschinensprache ist eigentlich das geringste Problem. Schwieriger ist es, sich an die Struktur von Maschinenprogrammen zu gewöhnen. Jedes Problem muß in kleinste Schrittchen zerlegt werden, um der Arbeitsweise des Computers Rechnung zu tragen. Ein komplexes Programm aus winzigen Einzelschritten aufzubauen ist das eigentliche Problem der Maschinenprogrammierung. Dieses Buch will Ihnen anhand vieler Beispiele helfen, den Einstieg in diese Denkweise zu finden. Wir wollen Ihnen aber nichts vormachen: Wenn Sie das Buch durchgearbeitet

haben, fängt die eigentliche Arbeit für Sie erst richtig an. Erst die Umsetzung eigener Probleme führt zu einer wachsenden Beherrschung der neuen Programmiertechnik. Für das Programmieren in Maschinensprache gilt das Motto „Übung macht den Meister" in besonderer Weise. Wenn Sie die Z80-Sprache einmal beherrschen, haben Sie nicht nur Zugang zum Innenleben des CPC, sondern genauso zu dem anderer Z80-Rechner, und davon gibt es, wie schon gesagt, eine ganze Menge. Darüber hinaus wird es Ihnen nicht schwerfallen, sich in die Maschinensprache eines anderen Prozessors, z. B. des 6502, einzuarbeiten. Denn obwohl sich das Vokabular der Maschinensprachen von Z80, 6502 und anderer CPUs voneinander unterscheidet, die Arbeitsweisen dieser Prozessoren sind doch recht ähnlich.

Die „Gehirnzellen“ des Computers

2

Wenn Sie bereits alles über duale und hexadezimale Zahlensysteme wissen, dann werden Sie in diesem Kapitel schnell vorwärtskommen. Überschlagen sollten Sie es auf keinen Fall, da hier wichtige Einzelheiten über die interne Struktur des Z80 erklärt werden. Und ohne diese Kenntnisse können Sie kein Maschinenprogramm verstehen, geschweige denn eines schreiben.

Speicher, Register, Flaggen

Im ersten Kapitel wurde bereits gesagt, daß ein Computer nichts anderes als eine Ansammlung sich gegenseitig beeinflussender Schalter ist. Natürlich sind das elektronische und keine mechanischen Schalter, obwohl es solche Computer früher einmal gegeben hat. Diese Schalter heißen „Bits“ und sind beim CPC meist in 8er-Blöcken und in wenigen Fällen in 16er-Blöcken zusammengefaßt. Einen 8er-Block nennt man „Byte“. Die meisten Bytes (fast 100000 beim CPC464 und über 180000 beim CPC6128) befinden sich im Speicher des Computers. Diese Speicherstellen sind durch Leitungen, die man Adreß-, Daten- und Steuerleitungen nennt, mit der CPU verknüpft. In der Z80-CPU befinden sich einige 8er- und 16er-Bitblöcke, die man „Register“ nennt. Außerdem enthält die CPU noch den eigentlichen Verarbeitungsteil, die „ALU“ (Arithmetic and Logical Unit). In der ALU findet die Verknüpfung der in die Register eingegebenen Werte nach Regeln statt, die durch die verschiedenen Befehle der Maschinensprache aufgerufen werden können. Ein stark vereinfachtes, aber für unsere Zwecke ausreichendes Blockschema des Z80 zeigt Bild 1.
Die mit A, B, C, D, E, H, L bezeichneten Kästchen sind die bereits angesprochenen 8-Bit-Register. Alle Daten, die verarbeitet werden sollen, müssen zuerst einmal in einem dieser Register stehen. Spezielle Bedeutung hat das A-Register, das auch „Akkumulator“ genannt wird. Wir werden später sehen, daß es vielseitiger ansprechbar ist als die anderen Register. Eine Spezialität der Z80-CPU ist es, daß je

A	F	A'	F'
B	C	B'	C'
D	E	D'	E'
H	L	H'	L'

IX
IY
SP
PC

Bild 1 *Registersatz der Z80-CPU.*

zwei 8-Bit-Register paarweise zu 16-Bit-Registern zusammengefaßt werden können. Daraus entstehen die BC-, DE- und HL-Registerpaare, die zu den besonderen Stärken des Z80-Prozessors gehören. Warum diese Paare so wichtig sind, werden wir gleich sehen. Außerdem gibt es noch das F-Register; es wird auch „Flaggen-" oder „Prozessor-Status-Register" genannt. Dieses Register kann im Gegensatz zu den eben besprochenen Registern nicht ohne weiteres von außen mit Daten beschickt werden. Es ist vielmehr dazu da, den momentanen Zustand des Prozessors zu signalisieren. Daher kommt auch die Bezeichnung „Flaggen" für die Bits des F-Registers, mit denen wir später noch viel zu tun haben werden. (In der deutschsprachigen Fachliteratur wird sehr häufig auch das englische Wort „flag" verwendet.)
Alle diese Register sind doppelt vorhanden. Allerdings werden wir uns um den Zweitregistersatz A', B', C', D', E', H', L', F' zunächst nicht kümmern; er wird vom Betriebssystem des CPC benutzt, und seine Verwendung ist eine etwas heikle Sache, die besondere Vorsichtsmaßnahmen erfordert. Eine unbedachtsame Benutzung der Zweitregister würde zum sofortigen Systemzusammenbruch führen.
Neben den 8-Bit-Registern gibt es noch vier 16-Bit-Register: das IX-, IY-, SP- und PC-Register. Die IX- und IY-Register, die auch Indexregister heißen, werden hauptsächlich dazu verwendet, um sich in dem Speicherbereich außerhalb der CPU zurechtfinden zu können. SP bedeutet „Stack Pointer" (seinen Zweck werden wir an gegebener Stelle kennenlernen), und PC heißt „Program-Counter". Das PC-Register führt sozusagen Buch darüber, welche Instruktion des Programms gerade verarbeitet wird. Neben diesen Registern gibt es noch die nicht im Blockschema enthaltenen I- und R-Register, die sehr spezielle Aufgaben haben und daher für uns weniger interessant sind, sowie eine Reihe interner Register, die für das Funktionieren der CPU zwar wichtig sind, auf die der Programmierer aber keinen Zugriff hat.
Wenn Sie jetzt verwirrt sind, ist das kein Wunder. Das Zusammenspiel all dieser Register untereinander und mit dem Speicher sowie ihre optimale Verwendung wollen wir ja gerade in diesem Buch kennenlernen. Aber fürs erste sollten Sie sich merken:

Zusammenfassung

In der Z80-CPU gibt es 8-Bit- und 16-Bit-Register. Das vielseitigste 8-Bit-Register ist das A-Register (Akkumulator). Die Register B, C, D, E, H, L können paarweise zu den 16-Bit-Registern BC, DE, HL zusammengefaßt werden. Zudem gibt es noch die 16-Bit-Register IX und IY (sowie das SP- und PC-Register).

Bytes als Schalter und Zähler

Der momentane Inhalt eines Registers ist durch die Stellungen seiner Schalter bzw. Bits gegeben. Üblicherweise werden dabei die Bits von 0 bis 7 bzw. 0 bis 15 durchnumeriert. Häufig haben diese Schalterstellungen nichts miteinander zu tun. So ist das z. B. im F-Register, wo jede der einzelnen Flaggen unabhängig von den anderen einen bestimmten Zustand des Prozessors anzeigt. Das ist dann so ähnlich wie in einer Küche, in der der Spülmaschinenschalter den Schalter des Herds weder beeinflußt noch von ihm beeinflußt wird. Auch die Speicherstellen, die den Tongenerator und den Videoteil steuern, können teilweise als unabhängige Schalter angesehen werden, die einfach aus organisatorischen Gründen in Achtergruppen zusammengefaßt sind. Ein weiteres Beispiel für diese Verwendung von Bytes ist der Bildschirmspeicherbereich: Hier entspricht jedes Bit einem Punkt auf dem Bildschirm. Ist das Bit gesetzt, so ist der Bildschirmpunkt hell, sonst ist er dunkel. (Auf die Farbmöglichkeiten soll hier nicht eingegangen werden.) Ein Byte kontrolliert also gerade acht nebeneinanderliegende Punkte, und die Stellungen der Bits haben miteinander nichts zu tun. Ganz anders ist es, wenn man den Inhalt eines Bytes als Zahl auffassen will. Das Byte wirkt dann als Zähler, wie Sie ihn vom Tachometer eines Autos kennen. Der Unterschied ist nur der, daß beim Kilometerzähler jede Stelle den Wert von 0 bis 9 annehmen kann; ein Bit dagegen hat nur zwei Möglichkeiten: „Aus“ bedeutet 0, „an“ bedeutet 1. Das Umschalten einer Zählerstelle kann hier auch das Umschalten anderer Zählerstellen bewirken: Die Schalterstellungen sind also voneinander abhängig, und die acht Bits eines Bytes müssen hier immer in ihrer Gesamtheit gesehen werden.

Zahlen im Dualsystem

Der Ziffernvorrat 0 und 1 der Bits bedingt eine etwas ungewohnte Zählweise. Sollten Sie schon alles über Dual- und Hexadezimalzahlen wissen, so können Sie die nächsten Abschnitte überblättern. (Die folgende Erklärung des Dualsystems

stammt übrigens aus einem Rechenbuch für Grundschüler.) Um eine bestimmte Anzahl von Gegenständen abzuzählen, werden sie (links beginnend) in Päckchen zu je 2 Stück gesammelt. Diese Zweierpäckchen erster Stufe werden wieder paarweise zu Zweierpäckchen zweiter Stufe gepackt usw. Das sieht bei 13 Gegenständen aus, wie in Bild 2 gezeigt:

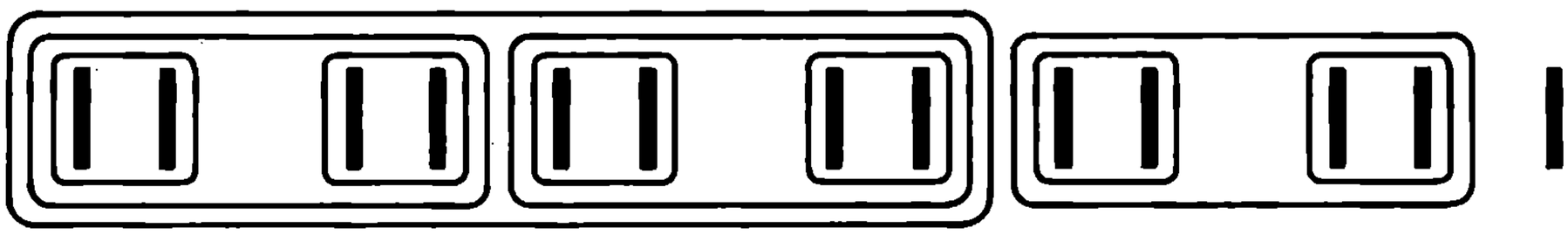


Bild 2 *Dualzahldarstellung der Zahl 13 nach der „Päckchenmethode"*

Nun macht man für jedes zum Schluß allein bleibende Päckchen in einer Stufe eine 1. Bleibt in dieser Stufe kein Päckchen übrig, schreibt man eine 0 hin. Die nullte Stufe (also das Einzelstück) entspricht dabei der Stelle am rechten Ende. Die erste Stufe (Zweierpäckchen) entspricht der nächsten Stelle und so weiter. Das Päckchenmuster der Zahl 13 sieht also folgendermaßen aus: 1 1 0 1. Das heißt: 1 Päckchen 3. Stufe + 1 Päckchen 2. Stufe + 0 Päckchen 1. Stufe + 1 Einzelstück. Diese Kombination von Nullen und Einsen ist genau die Dualzahldarstellung. Unser Vorgehen zur Bestimmung der Dualzahldarstellung zeigt auch sofort den konventionellen Weg zur Berechnung von Dualzahlen. Man dividiert eine vorgegebene Zahl durch 2, den ganzzahligen Anteil des Ergebnisses dividiert man wieder durch 2 und so weiter. Die sich bei diesen Divisionen ergebenden Reste, die entweder 0 oder 1 betragen können, schreibt man von rechts nach links auf. Für die Zahl 13 sieht die konkrete Berechnung ihrer dualen Darstellung so aus, wie in Tabelle 1 gezeigt.

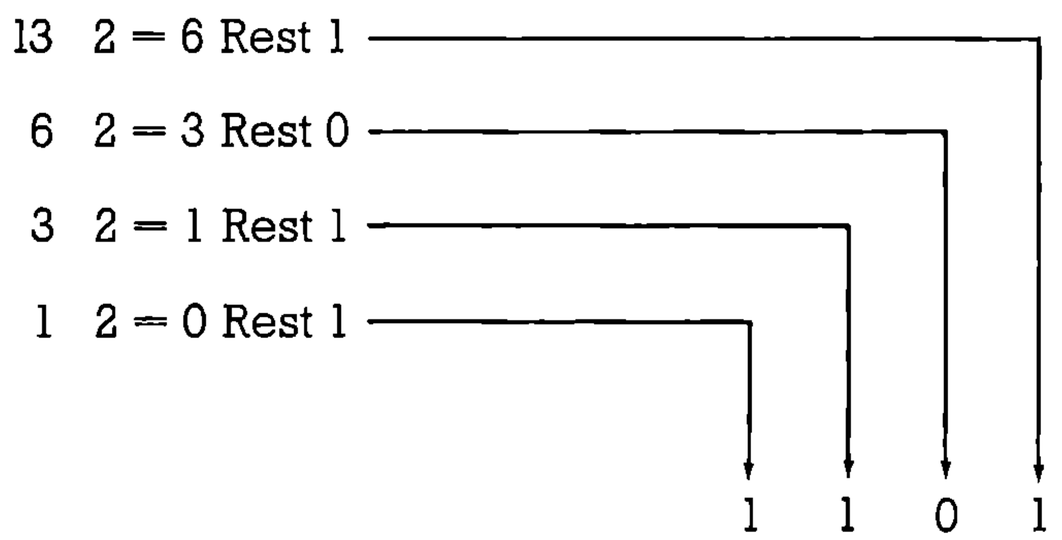


Tabelle 1 *Umwandlung der Zahl 13 in die duale Darstellung*

Dieses Divisionsverfahren entspricht genau unserer Päckchenmethode. Probieren Sie, falls Ihnen die Methode nicht ganz klar ist, das Verfahren doch an einigen weiteren Beispielen aus und vergleichen Sie Ihre Ergebnisse mit der Tabelle im Anhang F.
Sicher haben Sie bemerkt, daß sich in jeder Stufe die Zahl der Gegenstände in einem Päckchen verdoppelt. Päckchen enthalten in der ersten Stufe 2, in der zweiten Stufe 2x2=4, in der dritten Stufe 2x2x2=8 Gegenstände und so fort. Die Zahl

der Gegenstände in einem Päckchen ist also eine Zweierpotenz, wobei die Hochzahl gerade der Stufennummer entspricht. So kann man die Dualzahl 1101 wieder in die gewohnte Dezimalschreibweise zurückverwandeln:

1x8+1x4+0x2+1=13

Auf diese Weise kann man jedem Bitmuster eines Bytes eine Zahl zwischen 0 (entspricht 00000000) und 255 (entspricht 11111111) zuordnen. Dabei ergibt sich 255 aus 1x128+1x64+1x32+1x16+1x8+1x4+1x2+1x1. Um größere Zahlen darzustellen, benötigt man mehr als acht Bits. Verwendet man zwei Bytes, so hat man sechzehn Bits zur Verfügung. Falls Sie es noch nicht wissen, können Sie sich ausrechnen, daß damit die 65536 (Dezimal)zahlen zwischen 0 und 65535 dargestellt werden können. Und das entspricht auch der Anzahl der Speicherstellen, auf die die Z80-CPU ohne zusätzliche Tricks zugreifen kann. Denn jede Speicherstelle muß eine eigene „Adresse" haben, und es stehen dem Prozessor gerade sechzehn Bits zur Verfügung, um diese Adresse darzustellen. Daher sind das IX- und das IY-Register und die vorhin erwähnten Doppelregister BC, DE und HL so wichtig und so praktisch; bei CPU-Typen, die nur über 8-Bit-Register verfügen, sind Berechnungen im Adreßbereich bis 65535 viel mühsamer durchzuführen. (Weiter unten werden wir lernen, wie es im CPC gelungen ist, auf 65536 Adressen 98304 bzw. 180224 Speicherstellen unterzubringen.)

Maschinensprache und Assembler

3

In diesem Kapitel wird das Hexadezimalsystem vorgestellt. Danach werden Sie erfahren, was der Unterschied zwischen der Assemblersprache und einem Assembler ist, und schließlich lernen Sie eine merkwürdige Regel kennen, die bei der Angabe von Adressen in der Maschinensprache beachtet werden muß.

Das Hexadezimalsystem

Aus dem bisher Gesagten ergibt sich, daß jede Mitteilung an den Computer aus Schalterstellungen, d. h. aus Bit-Kombinationen, besteht, die in die Register der CPU gebracht werden. Es ist sehr schwierig, sich diese Bit-Kombinationen und ihre Auswirkungen zu merken. Eine gewisse Vereinfachung kann man dadurch erreichen, daß man bei der Angabe des Inhalts eines Bytes jeweils vier Bits für sich betrachtet. (Diese Vierergruppen werden manchmal als „Tetraden“ oder „Nibbles“ bezeichnet.)
Mit einer Vierergruppe kann man Zahlen zwischen 0 und 1x8 + 1x4 + 1x2 + 1x1 = 15, also insgesamt 16 verschiedene Zahlen darstellen. Und nun gibt man diese Kombinationen einfach durch eine einzige Ziffer an, denn eine einzelne Ziffer kann man sich besser merken als eine Kombination aus vier Einsen und Nullen. Da zur Darstellung von sechzehn verschiedenen Ziffern die Symbole des dezimalen Ziffernsystems nicht mehr ausreichen, macht man nach der 9 einfach mit Buchstaben weiter. „A“ entspricht der 10, „B“ der 11 usw., bis zum „F“, das der Zahl 15 entspricht. Im einzelnen zeigt das die nebenstehende Tabelle.
Die Ziffern 0 bis F sind dabei Bestandteile des 16er-Zahlensystems. Mit einem (aus dem Griechischen und Lateinischen kommenden) Fremdwort wird dieses System auch als „Hexadezimalsystem“ bezeichnet. Es ist am Anfang schon etwas ungewohnt, die Ziffernkombination „10“ einmal als „zwei“ (im Dualsystem), dann als „zehn“ (im Dezimalsystem) und schließlich als „sechzehn“ (im Hexadezimalsystem) interpretieren zu müssen. Aber schließlich kann man sich D3 bes-

dual	hexadezimal	dezimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

ser merken als die Bit-Kombination 11010011. Eine Hexadezimalzahl mit mehreren Stellen kann man dann im Prinzip genauso ins Dezimalsystem umrechnen wie eine Dualzahl. So bedeutet z. B. „BC0E“: 11x(16x16x16) + 12x(16x16) + 0x16 + 14 = 48142, und „FFFF“ bedeutet, daß in zwei Bytes alle Bits gesetzt sind, und das entspricht wieder der Dezimalzahl 65535. Um Verwechslungen auszuschließen, wollen wir im folgenden Hexadezimalzahlen durch ein vorangestelltes „&“ und Dualzahlen durch ein „X“ kennzeichnen.

Assembler-Sprache

Natürlich kann man sich auch die Auswirkungen einer Hexadezimalzahl auf die CPU schlecht merken. Daher haben sich schon die ersten Computerpioniere den Trick einfallen lassen, einen Maschinenbefehl durch ein Merkwort zu bezeichnen, das etwas mit der Auswirkung dieses Maschinenbefehls zu tun hat. Solche Merkwörter heißen „mnemonische Abkürzungen“ und beziehen sich auf englischsprachige Begriffe. So ist beim Z80 dem Maschinenbefehl &7C das Merkwort LD A,H zugeordnet. Diese Anweisung an die CPU bedeutet: Lade (load) das A-Register mit dem Inhalt des H-Registers. Jeder kann sich jetzt vorstellen, daß auch die Befehle LD B,E; LD L,A und so fort existieren. Der erste lädt das B-Register mit dem Inhalt des E-Registers, der zweite das L-Register mit dem Inhalt des A-Registers. Diese Merkwörter bilden den Bestandteil der „Assemblersprache“. Sie merken jetzt sicher, warum man vor den 600 Befehlen des Z80 keine

so große Angst zu haben braucht. Die sieben 8-Bit-Register A, B, C, D, E, H, L benötigen für den Datenaustausch untereinander schon 42 Befehle. Die entsprechenden Assemblerbefehle sind nach dem eben Gesagten eigentlich klar; man braucht sie sich gar nicht im einzelnen zu merken.

Was ist ein „Assembler"?

Das Erstellen eines Maschinenprogramms geht im allgemeinen folgendermaßen vor sich: Zuerst schreibt man das Programm mit Hilfe der mnemonischen Assemblerbefehle. Diese müssen dann in den eigentlichen „Maschinencode" übersetzt und in den Arbeitsspeicher geladen werden. Den Übersetzungs- und Ladevorgang kann ein spezielles Programm durchführen, das „Assembler" genannt wird. (Üblicherweise werden, etwas mißverständlich, sowohl die mnemonische Sprache als auch das Übersetzungsprogramm mit dem Begriff „Assembler" bezeichnet.)
Im Anhang B/C finden Sie einen „Direktassembler", der, wenn Sie ihn richtig eintippen, diese Übersetzungsarbeit für Sie durchführt. (Ein „Direkt"assembler übersetzt einen mnemonischen Maschinenbefehl nach der Eingabe direkt in Maschinencode.) In einfachen Fällen kann die Übersetzung auch mit Hilfe einer Tabelle, wie sie im Anhang D angegeben ist, von Hand ausgeführt und der entsprechende Maschinencode mit Hilfe eines BASIC-Programms in den Speicher gePOKEt werden. Für diese „Handassemblierung" muß ein wichtiger Punkt beachtet werden: Wir haben vorhin gesehen, daß zur Angabe einer Adresse im Bereich bis 65535 zwei Bytes benötigt werden. Bei der Adresse &AB80 beispielsweise enthält das sogenannte High-Byte (das sind die beiden linken Ziffern der hexadezimalen Adresse) den Teil &AB, das Low-Byte (das sind die beiden rechten Ziffern) enthält den Wert &80. In allen Maschinenbefehlen, die eine Adreßangabe benötigen, muß zuerst das Low-Byte und dann das High-Byte angegeben werden. Ein Beispiel soll das klarmachen:
&3A 84 65 ist ein Maschinenbefehl, in dem das zweite und dritte Byte eine hexadezimale Adreßangabe bedeuten. Der Befehl besagt, daß der Akkumulator mit dem Inhalt der Speicherstelle geladen werden soll, die die Adresse &6584 (nicht die Adresse &8465) hat. Die mnemonische Abkürzung lautet: LD A,(&6584) und nicht etwa LD A,(&8465).
Merken Sie sich diese Vereinbarung. Eine Nichtbeachtung ist sonst eine ständige Quelle von Irrtümern:
Bei einer Adreßangabe in der Maschinensprache muß zuerst das Low-Byte und dann das High-Byte angegeben werden. In der mnemonischen Assemblersprache wird eine Adresse in der „normalen" Reihenfolge angegeben: Zuerst das High-Byte, dann das Low-Byte.
Vielleicht fragen Sie sich jetzt, was der wesentliche Unterschied zwischen der Assemblersprache und einer anderen Programmiersprache (wie BASIC oder

Pascal) ist. Kurz gesagt: Jedem Assemblerbefehl entspricht genau ein Maschinenbefehl. Bei keiner anderen Programmiersprache ist das der Fall.

Low-Bytes, High-Bytes und Dezimalzahlen

Bei der Umrechnung von Hexadezimaladressen in Dezimaladressen kann man auch direkt von den Werten der beiden Bytes ausgehen. Dabei muß man dann den Inhalt des High-Bytes mit 16x16 = 256 multiplizieren und dazu den Inhalt des Low-Bytes addieren. &BC0E = &BCx256 + &0E. Mit &BC = 188 und &0E = 14 ergibt sich daraus

$$\&BC0E = 188x256 + 14 = 48142$$

Will man umgekehrt eine Dezimaladresse in die hexadezimale Darstellung umwandeln, so erhält man das High Byte als ganzzahligen Teil bei der Division der Adresse durch 256. Das Low Byte ist der Rest dieser Division. Wieder entsprechen diese Umrechnungsvorschriften den Verfahren, die wir bereits beim Dualsystem kennengelernt haben.

Die Speichereinteilung des CPC

4

Zunächst erhalten Sie eine kurze Übersicht über die Speicherorganisation des CPC. Sie erfahren, daß unterschiedliche Speicherstellen dieselbe Adresse haben können. Anschließend lernen Sie eine einfache Methode zur Eingabe kleiner Maschinenprogramme kennen.
Die beiden neueren Typen CPC664 und CPC6128 haben eine andere Speichereinteilung als der CPC464, da sie mehr ROM- bzw. RAM-Speicherplätze besitzen. Da diese Unterschiede im folgenden keine Rolle spielen, wird auf die Speichereinteilung dieser Typen nicht näher eingegangen. Alle drei CPC-Computer haben jedoch gemeinsam, daß zusätzliche Speicherplätze auf verschiedenen „Etagen“ mit derselben Adresse untergebracht sind.

ROM, RAM und „Speicheretagen“

Während die vorherigen Kapitel sich auf jeden Computer mit einer Z80-CPU beziehen können, müssen wir, bevor es mit den Maschinenprogrammen endlich losgeht, noch etwas über die Speichereinteilung des CPC sagen. Bei Maschinenprogrammen sorgt das Betriebssystem nicht automatisch dafür, daß sich verschiedene Programmabläufe im Speicher nicht stören.

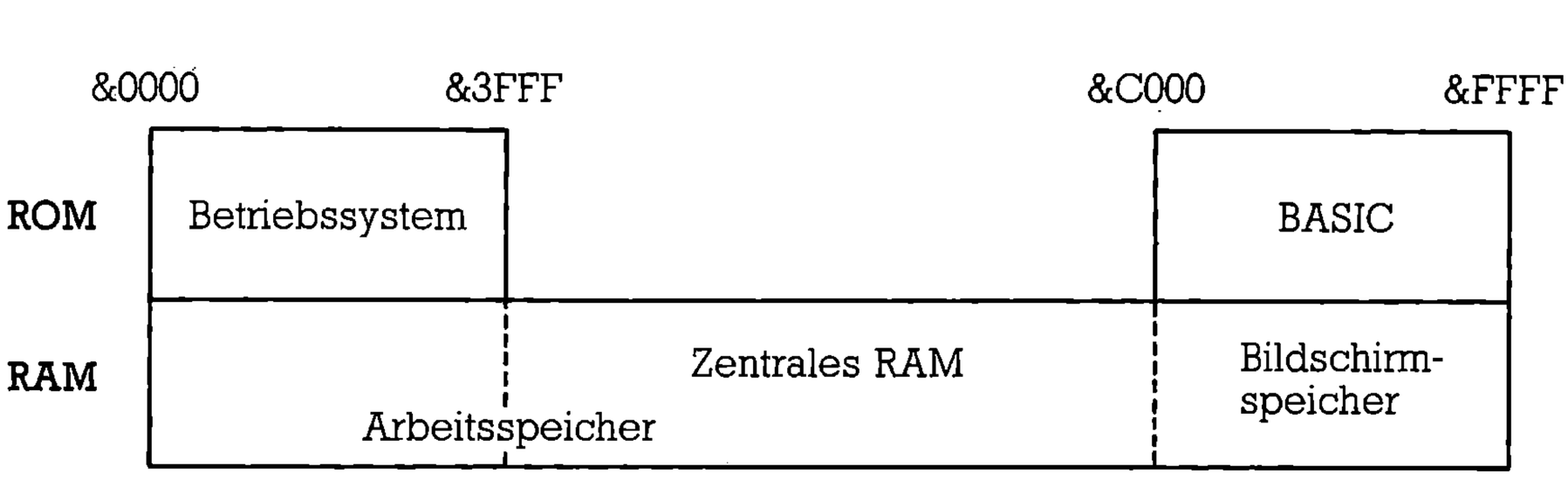


Bild 3 *Speichereinteilung des CPC464*

Neben 64 KByte RAM besitzt der CPC464 noch 32 KByte ROM (1 KByte entspricht dabei 1024 Bytes). Da die CPU nur 64 KByte adressieren kann, belegt der ROM-Bereich Adressen, die auch vom RAM-Bereich eingenommen werden. Über eine zusätzliche Leitung kann dann die CPU die Stockwerke des Adreßbereichs anwählen: Im Erdgeschoß sitzt das RAM, im ersten Stock das ROM. Im nichtlöschbaren ROM sind die Programme enthalten, die auch nach dem Abschalten des Computers nicht verlorengehen dürfen: Der BASIC-Interpreter belegt den Adreßbereich von &C000 bis &FFFF, das Betriebssystem den Bereich von &0000 bis &3FFF. Wir werden später sehen, daß die Programme des ROM-Bereichs auch von Maschinenspracheprogrammen aus benutzt werden können, was natürlich viel eigene Programmierarbeit spart. Der Schreib-Lese-Speicher RAM steht unseren Maschinenprogrammen allerdings nicht vollständig zur Verfügung.
Ein Viertel des gesamten RAM-Speichers wird ständig zur Speicherung des Bildschirminhalts benötigt. Normalerweise reicht dieser Bereich von &C000 bis &FFFF. Dieser riesige Bildschirmspeicher ist einerseits der Preis für die hohe Auflösung der CPC-Graphik, aber andererseits auch der Preis für die leichte Mischbarkeit von Text und Graphik. Bei anderen Computern, wie dem Commodore 64, ist das schwieriger; dafür benötigt der Bildschirmspeicher beim C64 im Schriftmodus auch nur 1000 Bytes. Man kann eben nicht alles haben. Vom restlichen RAM-Speicher belegt das Betriebssystem noch die Bereiche von &0000 bis &0170 und von &AB80 bis &BFFF. Für den Benutzer bleiben dann rund 42 KByte RAM zum Programmieren übrig; für den Anfang sollte das wohl reichen.

BASIC-Lader für Maschinenprogramme

Um unser erstes Maschinenprogramm in den Rechner zu bringen, verwenden wir die „Fußgängermethode“ des BASIC-Laders. Dieser sieht folgendermaßen aus:

```
10 REM BASICLADER
20 MODE 2
30 A=&4000
40 READ C
50 IF C=-1 THEN 70
60 POKE A,C:A=A+1:GOTO 40
70 CALL &4000
80 GOTO 80
100 DATA &3E,&FF,&32,&00,&C0,&C9,-1
```

In diesem Programm schalten wir zunächst auf den Graphikmodus 2 um, in dem wir bevorzugt arbeiten werden. Es ist wichtig, daß der Mode-2-Befehl unmittelbar vor dem Start der folgenden Maschinenprogramme erfolgt. &4000 ist die erste Adresse des Maschinenprogramms. Dieses liegt in den Werten der DATA-Zeilen vor. Der Wert −1, der als Maschinencode nicht in Frage kommt, schließt

die Eingabe ab. Danach wird das Maschinenprogramm mit dem CALL-Befehl gestartet und schließlich bleibt das Programm in Zeile 80 hängen, bis wir es mit der ESC-Taste erlösen. Dadurch verhindern wir, daß der BASIC-Cursor eventuell Ergebnisse unseres Maschinenprogramms überschreibt. Wie schon vorhin gesagt, müssen wir jetzt auf viele Dinge achten, die bei BASIC-Programmen vom Betriebssystem geregelt werden.

Daten werden transportiert

5

In diesem Kapitel lernen Sie die ersten Maschinenbefehle kennen. Der in den vorausgegangenen Kapiteln besprochene Umgang mit Assemblerbefehlen, Maschinencode und dem BASIC-Lader wird anhand eines kleinen Programms konkret durchgeführt.

Das erste Maschinenprogramm

Mit unserem ersten Maschinenprogramm wollen wir eine Speicherstelle des RAM mit einem Zahlenwert laden. Diese Aufgabe ließe sich auch durch einen POKE-Befehl von BASIC aus erledigen. Wir werden aber bald sehen, welchen ungeheuren Geschwindigkeitsvorteil die Programmierung in Maschinensprache bringt. Um den Erfolg unserer Bemühungen kontrollieren zu können, wählen wir eine Stelle aus dem Bildschirmspeicher, z. B. die Anfangsadresse &C000. Ein Programm, das diese Speicherstelle mit 255 = X11111111 lädt, hat folgendes Aussehen:

```
4000                    1           ORG   #4000
4000  3EFF              2           LD    A,#FF          ;LADE AKKU MIT #FF.
4002  3200C0            3           LD    (#C000),A      ;BRINGE DEN INHALT VON A
                        4                                ;IN DEN SPEICHER #C000.
4005  C9                5           RET                  ;FERTIG.
                        6 ;P1
```

Die kurze Programmliste, deren Struktur wir auch im folgenden beibehalten wollen, zeigt links die Adressen für die Befehle, dann kommt der Maschinencode und schließlich die mnemonischen Befehle, aus denen Sie den Maschinencode mit Hilfe der Befehlstabelle oder des Assemblers im Anhang erzeugen können. Dabei ist zu beachten, daß in den Programmlisten hexadezimale Zahlen durch ein vorangestelltes „#“ und nicht durch ein „&“ gekennzeichnet werden. (Weitere Punkte, die beim Lesen der abgedruckten Programmlisten beachtet werden

müssen, werden im Anhang A beschrieben. Bitte lesen Sie diesen Abschnitt, bevor Sie versuchen, die Programme abzuschreiben.) Die letzte Zeile der Programmliste gibt die Programmnummer an.
Der erste Befehl LD A,&FF lädt den Akkumulator mit der Zahl &FF. Der zweite Befehl LD (&C000),A lädt die Speicherstelle &C000 mit dem Akkumulatorinhalt. Das scheint umständlich zu sein. Aber die Z80-CPU hat keinen Befehl, der einen Wert unmittelbar in eine Stelle des RAM-Speichers bringen kann. Hier muß der Akkumulator als Datendrehscheibe herhalten. In dieser Form hätte auch keines der anderen 8-Bit-Register diese Aufgabe übernehmen können; einen Befehl LD (&C000),B beispielsweise gibt es nicht.
Die beiden LD-Befehle unseres Programms lassen eine wichtige Regel beim Gebrauch der Datentransportbefehle erkennen: Nach der mnemonischen Abkürzung LD wird zuerst der Empfänger und dann der Absender der Daten angegeben.
Es ist ferner wichtig zu wissen, daß bei den LD-Befehlen der Inhalt des Absenders nicht verändert wird. Man könnte daher auch von einem „Datenkopierbefehl" sprechen. Das ist genauso wie bei der BASIC-Anweisung A=B, bei dem der Wert des Speichers B ja auch erhalten bleibt.
Der dritte Befehl des Maschinenprogramms heißt RET, was für „Return" steht. Er hat die Aufgabe, nach Ende des Programms die Kontrolle an das Betriebssystem bzw. das BASIC-Programm zurückzugeben. Ohne den RET-Befehl würde der Prozessor die hinter &4005 folgenden Speicherstellen als Programmbestandteil ansehen und bei der Ausführung dieses „Programms" abstürzen.

Zusammenfassung

Datentransportbefehle haben den mnemonischen Assemblercode LD (load). Danach kommt zuerst der Empfänger, dann der Absender der Daten (wie im zweiten Befehl) oder der Inhalt der Sendung (wie im ersten Befehl).
Dabei gilt die folgende Vereinbarung: Zahlen in Klammern bedeuten Adreßangaben, Zahlen ohne Klammern bedeuten Daten.
Jedes Maschinenprogramm muß mit einem RET-Befehl abgeschlossen werden.

Low-High-Regel bei Operanden

Der Maschinencode für die Assemblerbefehle besteht in diesem ersten Programmbeispiel aus einem, zwei oder drei Bytes. (Später werden wir auch noch 4-Byte-Befehle kennenlernen.) Dabei ist links die Wirkungsweise des Befehls codiert. Dieser Befehlsteil wird oft „Operator" genannt. Danach kommen die veränderlichen Anteile des Befehls, die auch „Operanden" genannt werden. Das sind entweder Daten (wie im ersten Befehl) oder Adressen (wie im zweiten

Befehl). Beachten Sie auch hier die Low-Byte/High-Byte-Reihenfolge bei der Adreßschreibweise (&00 und &C0 statt C000). Manche Befehle, wie z. B. der RET-Befehl, kommen völlig ohne Operanden aus.
Wenn Sie jetzt den Maschinencode unseres Programms mit dem BASIC-Lader eingegeben und gestartet haben, dann erscheint in der linken oberen Ecke ein kurzer waagerechter Strich, der aus acht gesetzten Punkten des Bildschirms besteht.
Die DATA-Zeile hat bei der Eingabe unseres ersten Maschinenprogramms die Form

```
DATA &3E,&FF,&32,&00,&C0,&C9,—1
```

(Falls etwas nicht geklappt hat, prüfen Sie nicht nur den Maschinencode in der DATA-Zeile, sondern auch den BASIC-Lader; insbesondere der Mode-2-Befehl muß vorhanden sein. Das gilt auch für die folgenden Programme.)

Übung 1 Ändern Sie im Programm P1 die Adresse &C000 in &C001 usw. um. Welcher Zusammenhang besteht zwischen dieser Adresse und der Position des Strichs auf dem Bildschirm?
Versuchen Sie, den Strich in der Mitte der ersten Zeile auszugeben.

Übung 2 Ändern Sie das Programm P1 so ab, daß ein Strich mit der halben Zeichenbreite (4 Punkte) ausgegeben wird.

Zusatzinformationen

Transportbefehle LD gibt es für die Datenübertragung zwischen den 8-Bit-Registern A, B, C, D, E, H, L. Dabei sind 7x7 = 49 Kombinationen möglich. LD C,L z. B. bringt den Inhalt von L nach C. Der Aufbau des Prozessors erlaubt sogar die sinnlosen Befehle LD A,A; LD B,B und so weiter. Transportbefehle zwischen den Doppelregistern wie LD BC,HL gibt es nicht. Zu diesem Zweck können die beiden Befehle LD B,H und LD C,L verwendet werden.
Alle Register können direkt mit Zahlen geladen werden: LD L,&10 lädt das L-Register mit &10; LD BC,&1234 lädt das BC-Register mit &1234. Dabei steht das High-Byte &12 in B und das Low-Byte &34 in C.
Ein für die Entwicklung Ihrer eigenen Programme wichtiger Befehl hat die mnemonische Abkürzung NOP und den Befehlscode &00. NOP steht für die englischen Worte „no operation" (keine Operation). Er hat keinerlei Wirkung und eignet sich daher als Platzhalter für später evtl. einzufügende Befehle. Einige NOPs an der richtigen Stelle können verhindern, daß Sie bei der Korrektur eines Fehlers das ganze Programm neu eingeben müssen.

Adreßangaben beim Datentransport: Direkt oder um die Ecke

6

In diesem Kapitel werden die Adressierungsmöglichkeiten für Sender und Empfänger bei Datentransportbefehlen erklärt. Anfängern bereitet manchmal das Verständnis der sogenannten indirekten Adressierung Schwierigkeiten. Die meisten BASIC-Programmierer haben bei der Verwendung von PEEK und POKE die indirekte Adressierung bereits kennengelernt.

Unmittelbare und absolute Adressierung

Bevor wir das nächste Problem in Angriff nehmen, müssen wir uns noch etwas systematischer mit den unterschiedlichen Adressierungsmöglichkeiten bei den Datentransportbefehlen LD beschäftigen. Zwei Möglichkeiten haben wir schon kennengelernt:
Eine Zahl, die nicht in Klammern steht, bedeutet einfach einen konstanten Wert (die Zahl muß dabei sinnvollerweise rechts vom Komma, also an der Absenderstelle stehen): LD B,&08 lädt das B-Register mit der Zahl &8; LD DE,&AB80 lädt das DE-Register mit der Zahl &AB80. Das High-Byte steht dabei immer im ersten Teil des Doppelregisters. Hier wird also D mit &AB und E mit &80 geladen. Häufig wird diese Art der Adressierung als „unmittelbar" (englisch: immediate) bezeichnet.
Eine Zahl, die in Klammern steht (entweder links oder rechts vom Komma), bezeichnet keinen Wert, sondern eine Speicheradresse. Diese Art der Adressierung heißt „absolut". LD A,(&1234) bedeutet, daß der Inhalt der Speicherstelle &1234 in den Akkumulator gebracht werden soll. LD (&1234),A bringt dagegen den Inhalt von A in die Speicherstelle &1234. Kein anderes 8-Bit-Register außer A hat diese Möglichkeit.
Bei den Doppel- bzw. 16-Bit-Registern werden durch die absolute Adressierung zwei Speicherstellen angesprochen: diejenige, deren Adresse in der Klammer steht, und die darauf folgende. LD HL,(&1234) lädt die Inhalte der Speicherstellen

&1235 ins H- und &1234 ins L-Register. Das heißt, die Inhalte der Speicherstellen &1234 und &1235 werden als ein 2-Byte-Wert aufgefaßt; das Low-Byte ist in der im LD-Befehl angegebenen Speicherstelle enthalten. Entsprechend lädt LD (&C000),DE den Inhalt des Doppelregisters DE in die Speicherstellen &C000 und &C001; dabei kommt das High-Byte aus D in den Speicher &C001.

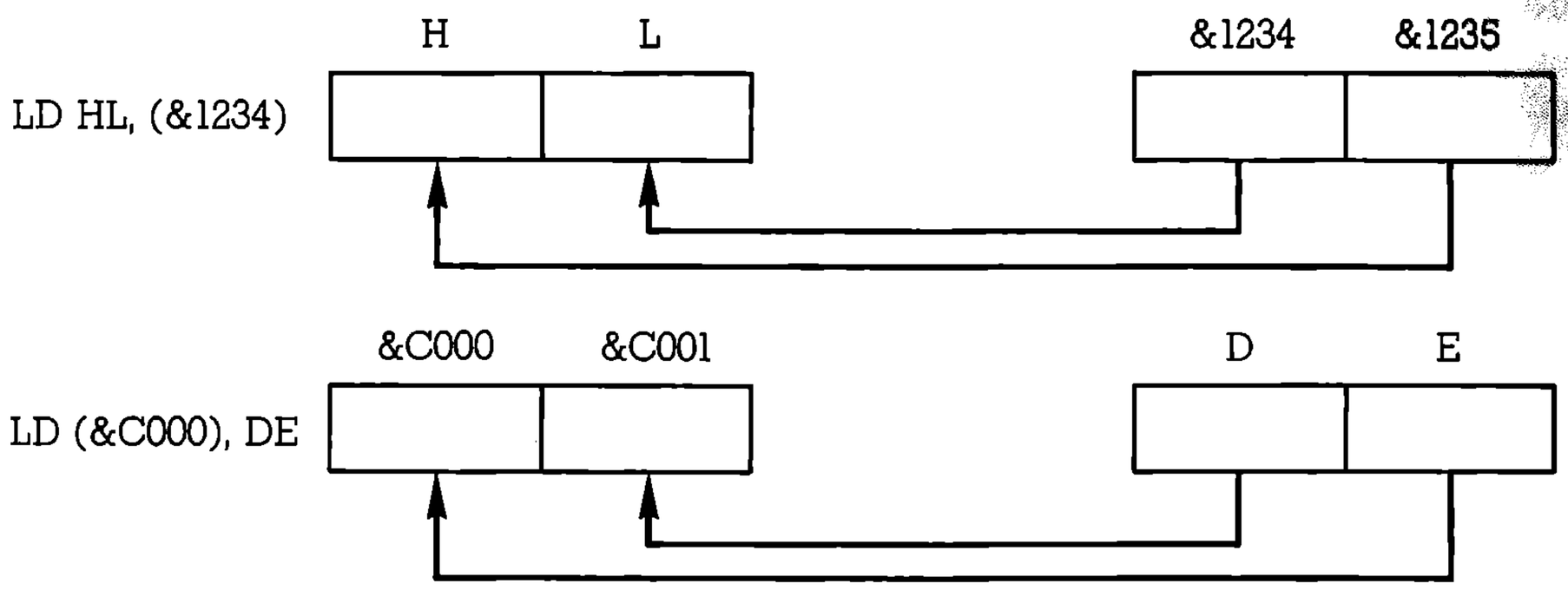


Bild 4 *Absolute Adressierung mit Doppelregistern*

Indirekte Adressierung

Besonders vielseitig ist die „indirekte Adressierung". Wenn Sie in BASIC schon mit PEEK und POKE gearbeitet haben, dann kennen Sie diese Methode bereits. Die beiden Anweisungen A=1000:PRINT PEEK(A) liefern nicht den Wert von A, nämlich 1000, sondern den Inhalt der Speicherstelle 1000. Ganz ähnlich liegt die Situation bei A=1000:POKE A,30. Dieser Befehl bringt die Zahl 30 nicht in den Speicher A, sondern in die Speicherstelle, deren Adresse (1000) in A steht.
Die indirekte Adressierung ist ein sehr wichtiges Verfahren für die Maschinenprogrammierung. In den LD-Befehlen des Z80 kann die indirekte Adressierung über die HL-, IX-, IY-Register und teilweise auch über die BC- und DE-Register erfolgen. In diesem Fall stehen die Doppelregister in Klammern: LD (DE),A gibt an, daß der Inhalt des Akkumulators in die Speicherstelle gebracht wird, deren Adresse im DE-Register steht. Entsprechend bringt LD A,(BC) den Inhalt der Speicherstelle, deren Adresse in BC steht, in den Akkumulator. Weiter bedeutet LD (HL),&FF, daß die Zahl &FF in die Speicherstelle gebracht wird, deren Adresse im Doppelregister HL steht.
Wir schreiben unser erstes Maschinenprogramm jetzt etwas um:

```
4000                    1           ORG   #4000
4000  2100C0            2           LD    HL,#C000      ;LADE HL MIT #C000.
4003  36FF              3           LD    (HL),#FF      ;LADE DIE ADRESSE IN HL
                        4                               ;MIT DEM WERT #FF.
4005  C9                5           RET                 ;FERTIG.
                        6 ;P2
```

Laden und starten Sie das Programm. Es hat dieselbe Wirkung wie unser erstes Programm. Trotzdem sind wir jetzt ganz anders vorgegangen. Der erste Befehl lädt die Zahl &C000 ins Doppelregister HL. (Das L-Byte ins L-Register, das H-Byte ins H-Register.) Der zweite Befehl lädt die Zahl &FF nicht nach HL (beachten Sie die Klammer), sondern in die Speicherstelle, deren Adresse sich in HL befindet, also in &C000. RET schließt das Programm wieder ab. Im obigen Programm kann übrigens das HL-Register nicht durch das BC- oder DE-Register ersetzt werden. Finden Sie anhand der Befehlsliste im Anhang D den Grund heraus!

Zusammenfassung

Für LD-Befehle gilt: Eine Zahl ohne Klammern bedeutet einen konstanten Wert, eine Zahl in Klammern bedeutet die Adresse des Absenders bzw. des Empfängers. Wenn ein 16-Bit-Register in Klammern gesetzt wird, so ist dieses Register nicht Absender oder Empfänger, sondern es enthält die Adresse des Absenders bzw. des Empfängers.

Übung 3 Welche der folgenden Befehle gehören zur Assemblersprache der Z80-CPU?

a) LD C,&10
b) LD (&C000),BC
c) LD B,(&2000)
d) LD DE,&1234
e) LD (BC),A
f) LD DE,IX

Übung 4 Kann im Programm P2 das Registerpaar HL durch die Registerpaare BC bzw. DE ersetzt werden?

Zusatzinformation

Die beschriebene Systematik mit den Klammern bei der absoluten Adressierung ist in der Z80-Assemblersprache nicht durchgehend gültig. Zum Beispiel bedeutet JP &1234 einen Sprung an die Adresse &1234. Formal betrachtet müßte es eigentlich JP (&1234) heißen.

7 Wir programmieren eine Schleife

In diesem Kapitel werden neue Befehle eingeführt: Mit INC und DEC können Registerinhalte heraufgesetzt bzw. vermindert werden. Außerdem wird ein „bedingter" Sprungbefehl vorgestellt, der der IF-Anweisung in BASIC entspricht. Die neuen Befehle ermöglichen die Programmierung von Schleifen.

Register als Zeiger und Zähler

Der Strich, den wir mit dem vorigen Programm ausgegeben haben, soll jetzt die Länge einer ganzen Zeile haben. Dazu verwenden wir das folgende Programm:

```
4000                    1          ORG   #4000
4000  0650               2          LD    B,#50          ;LADE B MIT #50.
4002  2100C0             3          LD    HL,#C000       ;LADE HL MIT #C000.
4005  36FF               4 L1:      LD    (HL),#FF       ;LADE DIE IN HL ANGEGE-
                         5                               ;BENE ADRESSE MIT #FF.
4007  23                 6          INC   HL             ;VERGROESSERE HL UM 1.
4008  05                 7          DEC   B              ;VERMINDERE B UM 1.
4009  C20540             8          JP    NZ,L1          ;SPRINGE NACH L1 WENN
                         9                               ;B<>0.
400C  C9                10          RET                  ;FERTIG.
                        11 ;P3
```

Die ersten drei Befehle sind ja schon bekannt: Wir laden das B-Register, um es im weiteren Programmverlauf als Schleifenzähler verwenden zu können; &50 (80) ist ja in Mode 2 gerade die Zeichenzahl pro Zeile. Die nächsten beiden Befehle sind dieselben wie im vorausgegangenen Programm. Das HL-Register wird in diesem Zusammenhang auch „Zeiger", „Vektor" oder „Pointer" genannt (es zeigt hier auf den ersten Speicherplatz des Bildschirmspeichers). Das entscheidende Kennzeichen dieses Verfahrens ist es, daß der Zeiger HL leicht verändert werden kann. Ohne die indirekte Adressierung im dritten Befehl wäre eine derartige Schleife nur umständlich zu programmieren.
Der Befehl INC HL erhöht den Inhalt des Registers HL um den Wert 1 (INC kommt

vom englischen „increment“, Zunahme). Dabei wird die Adresse in HL von &C000 auf &C001 erhöht. Danach verringert der Befehl DEC B den Inhalt des B-Registers um 1 (DEC kommt von „decrement“, Verminderung), also von &50 auf &4F. Der sechste Befehl

JP NZ,L1

heißt übersetzt: Springe („jump“), wenn der Inhalt des B-Registers nicht Null ist (NZ, „not zero“) zur Adresse L1. Die „Sprungmarke“ L1 muß in der Maschinensprache (und im Direktassembler des Anhangs) durch die Adresse des Befehls LD (HL),&FF, also &4005 ersetzt werden. In leistungsfähigeren Assemblern können solche Marken oder „Labels“ benutzt werden. Das Programm wird von der Sprungmarke an so oft durchlaufen, bis der Inhalt von B den Wert Null erreicht hat. Das ist nach 80 Durchläufen der Fall.
Da in jedem Durchlauf der Befehl INC HL den Adressenzeiger in HL um eins weiterrückt, ergeben sich schließlich 80 kurze Striche, die genau die erste Zeile füllen. Der JP NZ-Befehl wirkt nach dem 80sten Strich nicht mehr, da ja die NZ-Bedingung (Nicht-Null-Bedingung) für das B-Register nicht mehr erfüllt ist. Wenn kein Sprung mehr ausgeführt wird, kommt einfach der nächste Befehl RET an die Reihe, der das Programm beendet.
DEC- und INC-Befehle gibt es für alle Register und Doppelregister. Die Register verhalten sich dabei wie Kilometerzähler: Nach dem höchsten Wert &FF bzw. &FFFF kommt bei INC-Befehlen wieder der Wert &00. Bei DEC kommt nach dem Wert &00 wieder der höchste Wert. Probieren Sie das im Programm aus, indem Sie den ersten Befehl

LD B,&50 durch LD B,&B0

und

DEC B durch INC B

ersetzen. Nach &50maligem Inkrementieren steht im B-Register der Wert &00, da

&B0+&50=&100

und da die führende 1 von &100 in dem 8-Bit-Register B nicht mehr dargestellt werden kann.

Zusammenfassung

Mit den INC- und DEC-Befehlen kann man die Registerinhalte um 1 erhöhen bzw. verringern.

Übung 5 Ändern Sie das Programm P3 so ab, daß zwei Zeilen von je 40 gleichlangen Strichen und Lücken entstehen.
Hinweis: Ein einziger zusätzlicher Befehl genügt.

Die INC- und DEC-Befehle wirken auf alle Allzweckregister und Doppelregister: INC A, INC B,..., INC L, INC BC, INC DE, INC HL, INC IX, INC IY. Speicherstellen lassen sich bei diesen Befehlen über die Register HL, IX, IY indirekt adressieren. Beispiel: Die Befehlsfolge

```
LD HL,&C100
INC (HL)
```

erhöht den Inhalt der Speicherstelle &C100 um den Wert 1. Entsprechendes gilt für DEC.

8 Die erste Flagge

Wenn Sie das B-Register mit einer größeren Zahl als &50 laden, können Sie den Strich, den unser Programm produziert, in die nächste Zeile hinein verlängern. Allerdings ist die maximal erreichbare Länge dadurch festgelegt, daß in B keine größere Zahl als 255 stehen kann. Das ergibt einen Strich, der sich über knapp 3 1/4 Zeilen erstreckt. Wenn wir weiter zeichnen wollen, müssen wir unseren Schleifenzähler größer machen. Dazu bieten sich natürlich die Doppelregister an. Man ist versucht, den Befehl LD B,&50 z. B. durch LD BC,&4000 und den Befehl DEC B durch DEC BC zu ersetzen. Wenn Sie das so modifizierte Programm laufen lassen, erleben Sie entweder einen Systemzusammenbruch, oder Sie erhalten einen einzelnen Strich in der linken oberen Ecke.

Schleifen mit Doppelregistern

Der Grund dafür ist, daß der JP NZ-Befehl merkwürdigerweise nicht reagiert, wenn der Inhalt des BC-Registers den Wert Null annimmt. Auf diese Weise überschreiben Sie u. U. den kompletten RAM-Speicher des CPC mit dem Wert &FF, und das System verabschiedet sich. Das richtig abgeänderte Programm sieht wie folgt aus:

```
4000                    1          ORG   #4000
4000  010040            2          LD    BC,#4000      ;SCHLEIFENZAEHLER LADEN.
4003  2100C0            3          LD    HL,#C000      ;STARTADRESSE DES
                        4 ;                            ;BILDSCHIRMSPEICHERS.
4006  36FF              5 L1:      LD    (HL),#FF      ;#FF IN DEN
                        6 ;                            ;BILDSCHIRMSPEICHER.
4008  23                7          INC   HL            ;NAECHSTER BILDSCHIRM-
                        8                              ;SPEICHERPLATZ.
4009  0B                9          DEC   BC            ;ZAEHLER VERMINDERN.
400A  78                10         LD    A,B           ;SCHLEIFENZAEHLER
400B  B1                11         OR    C             ;AUF NULL ABFRAGEN.
400C  C20640            12         JP    NZ,L1         ;NACH L1 WENN BC<>0.
400F  C9                13         RET                 ;SONST FERTIG.
                        14 ;P4
```

Um den Grund für die beiden zusätzlichen Befehle

LD A,B und OR C

zu verstehen, müssen Sie zunächst einmal wissen, was eine Flagge ist. Im Blockschema der Z80-CPU haben Sie das F-Register, das auch Flaggenregister oder Prozessorstatus-Register genannt wird, gesehen. Das F-Register besteht aus acht einzelnen Bits, die nichts miteinander zu tun haben. Zwei dieser Bits sind unbenutzt. Die anderen Bits werden vom Prozessor beim Eintreten bestimmter Bedingungen gesetzt (Inhalt = 1) oder gelöscht (Inhalt = 0). Das sechste Bit insbesondere wird gesetzt, wenn sich als Ergebnis bestimmter Befehle der Wert Null ergibt. Es heißt daher „Zero-" oder „Z-Flagge". Entsprechend wird diese Flagge gelöscht, wenn das Ergebnis dieser Operation ungleich Null ist.

Die Z-Flagge reagiert nicht immer

Bei weitem nicht alle Befehle, die Null ergeben können, beeinflussen die Z-Flagge. Bei LD A,&00 bleibt sie genauso unbeeinflußt wie bei DEC BC. Dagegen reagiert sie bei den INC- und DEC-Befehlen der Einzelregister. Um sich darüber Klarheit zu verschaffen, muß man in der entsprechenden Tabelle des Anhangs D nachschlagen. Der JP NZ-Befehl reagiert nur, wenn die Z-Flagge gesetzt ist. Das heißt, die Null muß durch einen passenden Befehl erzeugt worden sein.
Da der DEC BC-Befehl die Z-Flagge nicht beeinflußt, benötigt man einen Trick: Nach dem Befehl DEC BC wird zuerst mit LD A,B der Inhalt des B-Registers in den Akkumulator geladen (dadurch wird die Z-Flagge noch nicht beeinflußt), dann wird mit OR C das C-Register mit dem Akkumulatorinhalt durch ein logisches Oder verknüpft. Nur wenn sowohl B als auch C eine Null enthalten (d. h., wenn der Inhalt des Doppelregisters den Wert &0000 hat), ist das Ergebnis dieser Operation wieder Null. Und da der OR-Befehl die Z-Flagge beeinflußt, können wir endlich den JP NZ-Befehl richtig einsetzen.
Wenn Sie das noch nicht genau verstanden haben, ist es nicht weiter schlimm. Den logischen Verknüpfungen, zu denen OR gehört, ist noch ein eigenes Kapitel gewidmet. Merken Sie sich aber den verwendeten Trick zur Abfrage der Doppelregister. Wir werden ihn noch häufiger verwenden.
Wenn Sie das Programm jetzt ablaufen lassen, erhalten Sie einen Eindruck von der mit Maschinenprogrammen erzielbaren Geschwindigkeit. Schlagartig wird der ganze Bildschirm gefüllt. Die 25 Zeilen mit je 80 Zeichen enthalten genau 2000 Zeichen. Jedes Zeichen wird durch acht Bytes aufgebaut (darüber gleich mehr). Damit ergeben sich für den gesamten Bildschirm 8x2000 = 16000 Bytes. Das heißt, unser „Strich", der jetzt &4000 Zeichen lang ist, füllt nun den gesamten Bildschirm. Da &4000 größer als 16000 ist, werden sogar noch 384 Speicherstellen geladen, die nicht zum Bildschirm gehören.
Neben dem Befehl JP NZ,nn gibt es auch einen JP Z,nn-Befehl. (nn steht für eine

hexadezimale Adresse, also z. B. JP NZ,&1234.) Bei diesem findet der Sprung zur Adresse nn nur statt, wenn die Z-Flagge gesetzt ist. Weiter gibt es den Befehl JP nn. Bei diesem wird in jedem Fall zur durch nn angegebenen Adresse gesprungen. Entsprechend gibt es neben dem uns bereits bekannten RET-Befehl, der ein Programm auf jeden Fall beendet, noch die Befehle RET Z und RET NZ. Diese Befehle beenden das Programm nur, wenn die Z-Flagge gesetzt bzw. gelöscht ist.

Zusammenfassung

Flaggen sind 1-Bit-Speicher in der CPU. Der Z80-Prozessor besitzt eine Zero-Flagge, die gesetzt oder gelöscht wird, je nachdem, ob das Ergebnis bei bestimmten Befehlen Null oder ungleich Null ist. Beim Befehl JP Z,nn wird der Sprung nach nn nur ausgeführt, wenn die Z-Flagge gesetzt ist. Entsprechendes gilt für die Befehle JP NZ,nn sowie RET Z und RET NZ.

Übung 6 Schreiben Sie ein BASIC-Programm, das dieselbe Wirkung wie das Maschinenprogramm P4 hat. Vergleichen Sie die Ausführungsgeschwindigkeiten.

Übung 7 Ersetzen Sie die beiden letzten Befehle des Programms P4 (JP NZ,L1 und RET) durch eine andere Kombination bzw. Form von bedingten RET- und JP-Befehlen, wie sie im letzten Abschnitt dieses Kapitels beschrieben wurden. Durch diese Maßnahme soll sich am Ablauf des Programms nichts ändern.

9 Ein Graphikzeichen erscheint auf dem Bildschirm

In diesem Kapitel wird zuerst der Aufbau des Bildschirmspeichers, der ab der Adresse &C000 angelegt ist, beschrieben. Mit diesen Kenntnissen ist es möglich, ein Graphikzeichen auf dem Bildschirm auszugeben. In dem dazugehörigen Programm wird zum ersten Mal das IX-Register benutzt.

Der Bildschirmaufbau

Mit dem Programm des vorherigen Kapitels können Sie den Aufbau des Bildschirmspeichers untersuchen. Da der Schirm in 25 Grobzeilen eingeteilt ist, die ihrerseits aus je 8 Feinzeilen bestehen, besitzt der Schirm insgesamt 200 Zeilen. Sie haben jedoch bemerkt, daß diese nicht gleichmäßig von oben nach unten vollgeschrieben werden. Statt dessen werden zuerst die obersten Zeilen aller Grobzeilen aufgefüllt, dann die zweitobersten Zeilen der Grobzeilen und so fort. Wenn Sie am Beginn des Programms P4 das BC-Register mit dem Wert &801 laden, sehen Sie 25 über die ganze Bildschirmbreite verlaufende Striche, die gerade die oberste Feinzeile jeder Grobzeile füllen. Darüber hinaus erscheint in der linken oberen Bildschirmecke genau unter dem obersten Strich der Anfang eines 26. Strichs, der die zweite Feinzeile der obersten Grobzeile markiert. Diese Bildschirmorganisation muß man beachten, wenn man einen Buchstaben oder ein Graphikzeichen auf den Bildschirm bringen will. Die Bitmuster aller Zeichen sind fest im ROM ab der Adresse &3800 gespeichert. Dabei liegen die acht Bytes, die zu einem Zeichen gehören, genau hintereinander. Wenn die Zeichen auf dem Bildschirm erscheinen, müssen die den Bytes entsprechenden Punktmuster natürlich genau untereinanderliegen. Wir haben gesehen, daß das im Bildschirmspeicher einem Abstand von genau &800 Adreßplätzen entspricht.
Wie wir auf den ROM-Speicher (Sie erinnern sich an das erste „Stockwerk" in Bild 3) zugreifen können, werden wir erst später lernen. Die Bitmuster der letzten 16 Zeichen werden jedoch beim Einschalten aus dem ROM ab Adresse &AB80

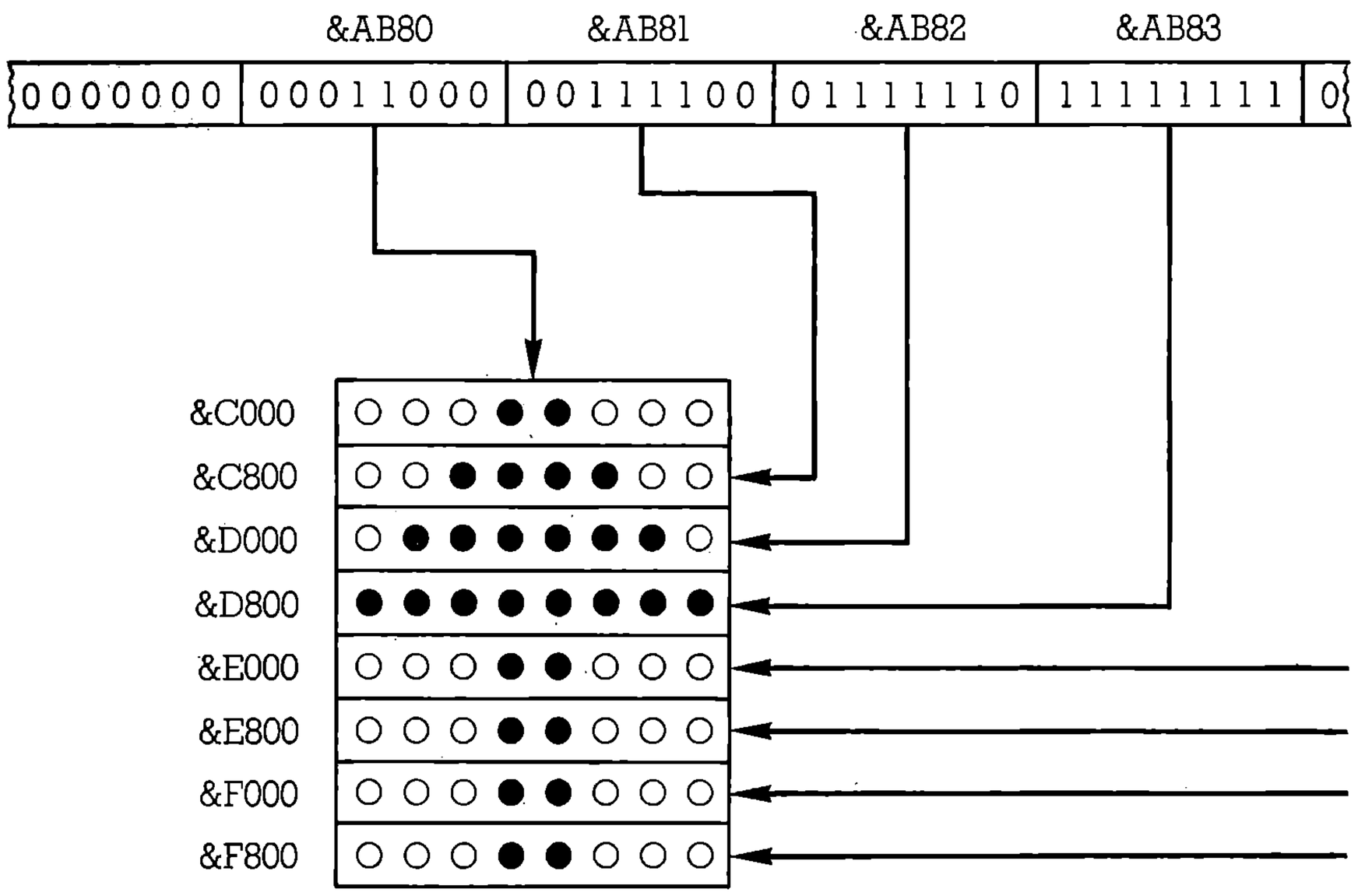


Bild 5 *Zuordnung eines im Zeichengenerator gespeicherten Bitmusters zum Bildschirmspeicher. Beim CPC664/6128 müssen die Adressen &AB80,..., &AB83 durch &A67C,..., &A67F ersetzt werden.*

(&A67C beim CPC664/6128) ins RAM kopiert. Das folgende Programm bringt uns das erste Zeichen auf den Bildschirm. (Hier ist es besonders wichtig, daß unmittelbar vor dem Start des Maschinenprogramms der MODE-2-Befehl gegeben wird.)

```
4000                    1          ORG   #4000
4000  1608              2          LD    D,#08          ;SCHLEIFENZAEHLER LADEN.
4002  DD2180AB          3          LD    IX,#AB80       ;STARTADR. BITMUSTER.
4006  2130C0            4          LD    HL,#C030       ;BILDS.SPEICHERPLATZ.
4009  010008            5          LD    BC,#0800       ;FEINZEILENABSTAND.
400C  DD7E00            6 L1:      LD    A,(IX+#00)     ;BITMUSTER IN DEN AKKU.
400F  77                7          LD    (HL),A         ;BITMUSTER IN DEN
                        8                                ;BILDSCHIRMSPEICHER.
4010  DD23               9          INC   IX             ;NAECHSTES BYTE DES BIT-
                       10                                ;MUSTERS.
4012  09                11          ADD   HL,BC          ;NAECHSTE FEINZEILE.
4013  15                12          DEC   D              ;ZAEHLER VERMINDERN.
4014  C20C40            13          JP    NZ,L1          ;NACH L1 WENN D<>0.
4017  C9                14          RET                  ;SONST FERTIG.
                       15 ;P5
```

Änderung für den CPC664/6128: In Zeile 3 muß die Adresse &AB80 durch &A67C ersetzt werden.

Erster Kontakt mit dem IX-Register

Wir verwenden hier zum ersten Mal das IX-Register. Im wesentlichen kann man damit umgehen wie mit dem HL-Register. Sogar die Befehlscodes sind recht ähnlich. Bei vielen Befehlen erhält man den Code für IX aus dem für HL einfach durch Davorsetzen des Bytes &DD. So ist &23 der Code für INC HL, der Code für INC IX ist &DD23. Entsprechendes gilt auch für das IY-Register. Nur muß hier das erste Byte &DD durch &FD ersetzt werden: Der Code für den Befehl INC IY ist damit &FD23. Zwei wichtige Unterschiede zum HL-Register gibt es aber: Die beiden Registerhälften von IX und IY können (jedenfalls mit den „offiziellen" Z80-Befehlen) nicht getrennt angesprochen werden, während die beiden Bytes von HL auch als Einzelregister H und L verwendet werden können. Außerdem steht in vielen Befehlen neben IX und IY noch eine „Verschiebeadresse" d, z. B. LD (IX+d),A. Wir wollen uns zunächst noch nicht darum kümmern. Wenn wir wie im fünften Befehl des Programms P5, dieser Verschiebung den Wert Null geben, können wir das IX-Register tatsächlich genau wie das HL-Register benutzen. Der Code von LD A,(HL) ist &7E, der von LD A,(IX+&00) ist &DD7E00. Das erste Byte weist wieder auf das IX-Register hin, das letzte gibt die Verschiebung d an, der wir den Wert &00 gegeben haben.
Das Programm macht wieder Gebrauch von der indirekten Adressierung. Das D-Register dient als Schleifenzähler und erhält den Anfangswert 8, da jedes Zeichen aus acht Bytes besteht, die auf den Bildschirm übertragen werden müssen. Das IX-Register dient als Zeiger auf das erste Byte des Bitmusters, das HL-Register wird mit der Adresse der &30sten Stelle des Bildschirmspeichers geladen. Schließlich erhält BC den Wert &0800, der den Adressenabstand zwischen zwei übereinanderliegenden Feinzeilen einer Grobzeile im Bildschirmspeicher angibt.
Dann wird das erste Byte des Bitmusters in den Bildschirmspeicher gebracht. Absender und Empfänger werden über die indirekte Adressierung mittels des IX- bzw. HL-Registers ermittelt. Da ein Befehl der Art LD (HL),(IX) nicht existiert, dient der Akkumulator wieder als Datendrehscheibe. Der Zeiger auf das Bitmuster wird dann mit INC IX um eins vorgerückt. Der Zeiger HL im Bildschirmspeicher muß auf die nächste Feinzeile zeigen und daher um &800 erhöht werden. Das bewirkt der Befehl

```
ADD HL,BC
```

der den Inhalt von BC (nämlich &0800) zum Inhalt von HL addiert und die Summe in HL abspeichert. Insgesamt wird die Schleife achtmal durchlaufen, bis das Zeichen auf dem Bildschirm steht.
Die Bitmuster der nächsten 15 Zeichen fangen dann jeweils um 8 Stellen versetzt an, also bei &AB88, &AB90, &AB98, &ABA0 und so weiter. (&A684, &A68C, &A694, &A69C usw. beim CPC664/6128.)

Zusammenfassung

Mit dem ADD-Befehl kann man 16-Bit-Zahlen addieren (also im Zahlenbereich 0 bis 65535 rechnen).
Die IX- und IY-Register kann man weitgehend wie das HL-Register benutzen. In fast allen Befehlen mit indirekter Adressierung muß dabei (HL) durch (IX+&00) bzw. (IY+&00) ersetzt werden.

Übung 8 **Ändern Sie das Programm so, daß das fünfte Zeichen (ein auf dem Kopf stehendes Dreieck) an der zehnten Stelle der ersten Zeile ausgegeben wird.**

Übung 9 **Verwenden Sie im Programm P5 statt des HL-Registers das IY-Register.**

Übung 10 **Bringen Sie das auszugebende Zeichen so auf den Bildschirm, daß jede zweite Feinzeile freibleibt (Zebramuster).**

Zusatzinformationen

ADD-Befehle gibt es für die Addition von 8-Bit- oder 16-Bit-Zahlen. Die 8-Bit-Additionsbefehle werden wir später kennenlernen. ADD-Befehle für 16-Bit-Zahlen gibt es in folgenden Versionen:

ADD HL,BC	ADD IX,BC	ADD IY,BC
ADD HL,DE	ADD IX,DE	ADD IY,DE
ADD HL,HL	ADD IX,IX	ADD IY,IY
ADD HL,SP	ADD IX,SP	ADD IY,SP

Bei diesen Befehlen wird der Inhalt des rechts stehenden Registers zu dem des links stehenden Registers (HL oder IX oder IY) addiert. Das Ergebnis wird im links stehenden Register abgespeichert.

Wir lernen die Carry-Flagge kennen und kürzen das Zeichensetzprogramm

10

Die Carry-Flagge, die jetzt vorgestellt wird, ist in gewisser Weise die wichtigste Prozessorflagge. Sie kann zu Rechen- und Kontrollzwecken herangezogen werden und ermöglicht darüber hinaus noch die Untersuchung einzelner Bits eines Bytes. Für die Abfrage der Carry-Flagge gibt es ähnliche Verzweigungsbefehle wie für die Z-Flagge.

Die Carry-Flagge

Das Zeichensetzprogramm des letzten Kapitels werden wir im folgenden häufig benutzen. Es wäre daher günstig, wenn wir es noch kürzer machen könnten. Das ist tatsächlich möglich, wenn wir eine weitere Flagge des F-Registers berücksichtigen. Es handelt sich dabei um die „Carry-Flagge", die häufig mit C abgekürzt wird, aber nicht mit dem C-Register verwechselt werden darf. „Carry" kommt aus dem Englischen und bezeichnet den Übertrag beim Addieren bzw. Subtrahieren. Da die Register nur 8- bzw. 16-Bit-Zahlen speichern können und daher die größten darstellbaren Zahlen 255 bzw. 65535 sind, erhält man, wenn bei einer Rechenoperation diese Maximalwerte überschritten werden, falsche Ergebnisse. Dasselbe erlebt man auch, wenn der Tageskilometerzähler des Autos auf 995 km steht und man fährt noch 30 km. Anschließend zeigt der Kilometerzähler 25 km an, da Tageskilometerzähler nur dreistellig sind und die Tausender einfach verschluckt werden.
Da ein Computer auch zum Rechnen eingesetzt werden soll, ist es natürlich nicht tragbar, wenn Rechenergebnisse wie 255+1 = 0 oder 253+7 = 4 vorkommen. Um das auszuschließen, besitzt der Prozessor eine besondere Flagge, die gesetzt wird, wenn ein Ergebnis das Fassungsvermögen des Registers überschreitet. Und das ist eben die Carry-Flagge. Man kann sie in diesem Moment als zusätzliches (d. h. neuntes bzw. siebzehntes) Bit des Registers betrachten. Allerdings müssen sich alle Register dieses Bit teilen. Wie mit dieser Flagge dann die

scheinbar falschen Rechenergebnisse korrigiert werden, sehen wir uns in einem späteren Kapitel an.
Wegen der Lage des Bildschirmspeichers am Ende des Speicherbereichs tritt die Carry-Flagge auch in unserem Zeichenprogramm in Aktion. Um von einer Feinzeile zur nächsten zu kommen, addieren wir zur Anfangsadresse des Zeichens im Bildschirmspeicher (in unserem Beispiel &C030) jeweils &800 (2048) dazu. Zwei Additionen erhöhen die Adresse gerade um &1000 (4096). Nach sieben Additionen, also bei der letzten Feinzeile, hat sich die Adresse von &C030 auf &F830 erhöht. Eine weitere Addition von &800 würde &10030 ergeben, also über die höchste 16-Bit-Zahl &FFFF hinausgehen: In diesem Moment zeigt die Carry-Flagge den Überlauf an. Oder anders ausgedrückt: Wenn die Carry-Flagge gesetzt wird, ist unser Zeichen fertig auf dem Bildschirm ausgegeben.

Bedingte Sprünge mit Carry

Genauso wie die Z-Flagge kann auch der Zustand der C-Flagge in den Sprungbefehlen abgefragt werden. Statt NZ und Z schreibt man für die Bedingung, ob die Carry-Flagge gelöscht oder gesetzt ist, NC und C. Damit können wir uns den Schleifenzähler D sparen und unser neues Programm heißt:

```
4000                    1          ORG   #4000
4000   DD2180AB         2          LD    IX,#AB80          ;STARTADRESSE BITMUSTER.
4004   2130C0           3          LD    HL,#C030          ;BILDS.SPEICHERPLATZ.
4007   010008           4          LD    BC,#0800          ;FEINZEILENABSTAND.
400A   DD7E00           5 L1:      LD    A,(IX+#00)        ;BITMUSTER IN DEN AKKU.
400D   77               6          LD    (HL),A            ;BITMUSTER IN DEN
                        7                                  ;BILDSCHIRMSPEICHER.
400E   DD23             8          INC   IX                ;NAECHSTES BYTE DES
                        9                                  ;BITMUSTERS.
4010   09              10          ADD   HL,BC             ;NAECHSTE FEINZEILE.
4011   D20A40          11          JP    NC,L1             ;NACH L1 WENN CARRY
                        12                                 ;NICHT GESETZT.
4014   C9              13          RET                     ;SONST FERTIG.
                        14 ;P6

L1       400A
```

Änderung für den CPC664/6128: In Zeile 2 muß die Adresse &AB80 durch &A67C ersetzt werden.

Die Befehle LD D,&08 und DEC D des Programms P5 fehlen jetzt. Statt des IX- könnte jetzt auch das DE-Register benutzt werden. Das Programm wäre dann noch etwas kürzer (nämlich um vier Bytes) und schneller. Andererseits wollen wir uns an den Gebrauch dieser Register gewöhnen.
Die Carry-Flagge reagiert aber nicht nur auf einen Überlauf. Auch bei einer Addition oder Subtraktion ohne Überlauf wird sie beeinflußt: Sie hat dann auf jeden Fall den Wert 0, selbst wenn sie vorher gesetzt war. Beachten Sie aber: Auf Über-

und Unterläufe infolge von INC- und DEC-Befehlen reagiert die Carry-Flagge nicht. (Ein Unterlauf bedeutet die Unterschreitung des Werts 0.)
Eine weitere Aufgabe der Carry-Flagge besteht darin, einzelne Bits aus einem Byte herauszupicken. Die dazu benötigten „Rotations-" und „Schiebebefehle" lernen wir in den folgenden Kapiteln kennen.

Zusammenfassung

Die Carry-Flagge zeigt mit dem Wert 1 den Überlauf bei einer Addition oder den Unterlauf bei einer Subtraktion an. Durch INC- und DEC-Befehle wird die Carry-Flagge nicht beeinflußt. Beim Befehl

```
JP NC,nn
```

wird der Sprung zur Adresse nn nur ausgeführt, wenn C nicht gesetzt ist, d. h. den Wert 0 hat. Entsprechendes gilt für die Befehle

```
JP C,nn
RET NC
RET C
```

Übung 11 Welchen Inhalt haben die Register IX und HL nach Beendigung des Programms P6?

Übung 12 Geben Sie für die folgenden Werte der HL- und BC-Register den Inhalt dieser Register und den Wert der Carry-Flagge nach der Ausführung des Befehls ADD HL,BC an.
a) HL = &F800 , BC = &0A00
b) HL = &1000 , BC = &2C00
c) HL = &8000 , BC = &8000

Zusatzinformationen

Zur Beeinflussung der Carry-Flagge gibt es eigene Befehle: SCF (Set Carry Flag) mit dem Code &37 setzt die Carry-Flagge; CCF (Complement Carry Flag) mit dem Code &3F bringt sie vom momentanen in den entgegengesetzten Zustand. Zum Löschen der Carry-Flagge verwendet man häufig den merkwürdigen Befehl AND A (Code &A7).

Beim CPC könnte der Bildschirmspeicher in einen anderen Bereich verlegt werden (z. B. &4000 bis &7FFF). In diesem Fall würde am Ende des Zeichenprogramms kein Überlauf erfolgen, und man müßte wieder einen Schleifenzähler einführen.

Wir rufen Unterprogramme auf und lernen den Stapelspeicher kennen

11

Der Stapelspeicher ist ein wichtiges Hilfsmittel für den Maschinenprogrammierer. Vom Prozessor wird der Stapelspeicher benutzt, um die Rücksprungadressen bei der Verwendung von Unterprogrammen abzuspeichern. Spezielle Datentransportbefehle mit den mnemonischen Codes PUSH und POP erlauben es dem Programmierer, Daten auf dem Stapel kurzfristig zwischenzuspeichern und so einen schnellen Datenaustausch zwischen den Registern der CPU durchzuführen.

Ein „Bildschirmfüll"programm

Nun soll der ganze Bildschirm mit Zeichen vollgeschrieben werden. Dazu verwenden wir das Programm des vorigen Kapitels in einer etwas abgewandelten Form. Zunächst muß für jedes Zeichen ein neuer Startpunkt im Bildschirmspeicher berechnet werden. Hierbei muß man dafür sorgen, daß die beschränkte Zahl von Registern sowohl für das Zeichenprogramm als auch für die Berechnung der Startadressen im Bildschirmspeicher ausreicht. Praktisch arbeiten wir mit zwei Programmen; eines berechnet die Startpunkte im Bildschirmspeicher, und das andere entspricht dem Programm des letzten Kapitels. Dieses zweite Programm wird vom ersten aufgerufen. Der entsprechende Assemblerbefehl heißt CALL (mit dem Code &CD). Der CALL-Befehl muß als Operand natürlich die Startadresse des Unterprogramms enthalten. Die Wirkung ist ähnlich wie bei der GOSUB-Anweisung in BASIC. Allerdings muß die Adresse nach CALL nicht zum eigenen Assembler-Programm gehören; es ist z. B. auch möglich, irgendein Programm des Betriebssystems aufzurufen.
Genauso wie in BASIC die letzte Anweisung eines Unterprogramms RETURN sein muß, wird auch ein Maschinenunterprogramm mit dem RET-Befehl abgeschlossen. Unser Programm zum Füllen des Bildschirms sieht somit folgendermaßen aus:

```
4000                    1          ORG   #4000
4000  3E02              2          LD    A,#02
4002  CD0EBC            3          CALL  #BC0E          ;MODE 2 EINSCHALT.
4005  2100C0            4          LD    HL,#C000       ;BILDS.SPEICHER AN
4008  DD2180AB          5          LD    IX,#AB80       ;ANFANG BITMUSTER.
400C  01D007            6          LD    BC,#07D0       ;ZAHL DER ZEICHEN.
400F  CD1A40            7 L1:      CALL  ZP             ;ZEICHENPROG. AUFRU
4012  23                8          INC   HL             ;NAECHSTE POSITION.
4013  0B                9          DEC   BC             ;ZAEHLER VERMINDERN.
4014  78               10          LD    A,B            ;SCHLEIFENZAEHLER
4015  B1               11          OR    C              ;AUF NULL PRUEFEN.
4016  C20F40           12          JP    NZ,L1          ;NACH L1 WENN BC<>0.
4019  C9               13          RET                  ;SONST FERTIG.
                       14 ;
401A  E5               15 ZP:      PUSH  HL             ;HL AUF DEN STAPEL.
401B  DDE5             16          PUSH  IX             ;IX AUF DEN STAPEL.
401D  110008           17          LD    DE,#0800       ;FEINZEILENABSTAND.
4020  DD7E00           18 L2:      LD    A,(IX+#00)     ;BITMUSTER IN DEN AKKU.
4023  77               19          LD    (HL),A         ;BITMUSTER AUF BILDS.
4024  DD23             20          INC   IX             ;NAECHSTES BYTE DES
                       21                               ;BITMUSTERS.
4026  19               22          ADD   HL,DE          ;NAECHSTE FEINZEILE.
4027  D22040           23          JP    NC,L2          ;NACH L2 SOLANGE DIE
                       24                               ;CARRYFLAGGE GELOESCHT
402A  DDE1             25          POP   IX             ;IX VOM STAPEL.
402C  E1               26          POP   HL             ;HL VOM STAPEL.
402D  C9               27          RET                  ;ZURUECK ZUM HAUPTPROG.
                       28 ;P7

L1      400F   L2      4020   ZP      401A
```

Änderung für den CPC664/6128: In Zeile 5 muß die Adresse &AB80 durch &A67C ersetzt werden.

Die ersten beiden Befehle bewirken in Maschinensprache dasselbe wie der BASIC-Befehl MODE 2. Dazu wird mit dem CALL-Befehl ein Programm des Betriebssystems aufgerufen, dessen Startadresse &BC0E ist. Im Akkumulator wird der Wert 2 ans Unterprogramm übergeben. Wir werden noch weitere Betriebssystemsroutinen kennenlernen. Die Benutzung dieser Unterprogramme erspart in vielen Fällen erhebliche Programmierarbeit. (Zur Übung werden wir natürlich auch weiterhin Programme schreiben, die schon im Betriebssystem vorhanden sind.)

Der Rest des ersten Programmteils besteht aus einer Schleife, welche die &7D0 (2000) Startplätze der einzelnen Zeichen im Bildschirmspeicher berechnet und diese zusammen mit dem Zeiger IX, der auf das Bitmuster des auszugebenden Zeichens zeigt, an das eigentliche Zeichenprogramm ZP weitergibt. Dieses Programm wird &7D0mal mit CALL ZP aufgerufen. (Sowohl im Direktassembler als auch im Maschinencode wird ZP durch die Startadresse &401A ersetzt.) Das Unterprogramm zum Setzen der Zeichen unterscheidet sich vom Programm des vorherigen Kapitels in zwei wichtigen Details: Die Ladebefehle für IX und HL fehlen, da diese Register im Hauptprogramm geladen werden. Außerdem enthält das Programm zwei neue Befehle mit den mnemonischen Codes PUSH und POP.

Der Stapelspeicher

Um diese Befehle zu verstehen, müssen wir etwas ausholen. Normalerweise wird ein Programm Befehl für Befehl, so wie sie hintereinander im Speicher stehen, abgearbeitet. Die Startadresse des jeweils nächsten zu bearbeitenden Befehls steht dabei im Programmzähler. Dieser ist ein 16-Bit-Register der CPU, das normalerweise mit „PC“ (Program Counter) abgekürzt wird. Bei Sprungbefehlen, wie z. B. JP nn, wird die im Befehl angegebene neue Adresse nn in den PC geladen. Der Prozessor unterbricht dann die normale Reihenfolge und setzt das Programm bei der im Sprungbefehl angegebenen Adresse fort.
Ähnlich ist das bei Unterprogrammaufrufen mit CALL; auch hier wird das PC-Register mit der Adresse im CALL-Befehl geladen. Allerdings besteht ein wichtiger Unterschied zum JP-Befehl. Das Hauptprogramm soll ja nach Abarbeitung des Unterprogramms mit dem nach CALL folgenden Befehl fortgesetzt werden. Und dazu muß sich der Prozessor die Rücksprungadresse merken. Daher bewirkt der CALL-Befehl, daß die Adresse des nächsten Befehls in einen speziellen Speicherbereich geladen wird, der „Stack“ oder „Stapel“ heißt. Der RET-Befehl holt am Ende des Unterprogramms die Rücksprungadresse vom Stapel, lädt sie ins PC-Register, und der Prozessor macht dann an dieser Stelle weiter. Das klingt beim ersten Durchlesen sicher recht kompliziert; glücklicherweise aber braucht sich der Programmierer um diese Vorgänge nicht zu kümmern, da sie automatisch ablaufen, d. h., sie werden durch CALL und RET gesteuert. Trotzdem ist es nützlich, die Verhältnisse zu kennen, da man den Stapelspeicher auch für andere Zwischenspeicherungen nutzen kann.
Um auch mit Unterprogrammen, die ihrerseits weitere Unterprogramme aufrufen, arbeiten zu können, muß der Stack eine spezielle Struktur haben: Die Adressen, die als letzte eingespeichert werden, müssen als erste wieder greifbar sein. (Vielleicht haben Sie schon den Begriff LIFO-Struktur gehört; das kommt von „Last In – First Out“: Das Unterprogramm, das als letztes aufgerufen wurde, wird auch als erstes wieder verlassen. Das ist wie bei einem Stapel Karten, bei dem nur oben aufgelegt und von oben wieder weggenommen wird.)

Stapeln mit PUSH und POP

Der Stapelspeicher ist aber nicht nur für die interne Speicherverwaltung bei Unterprogrammaufrufen da, sondern er steht auch dem Programmierer zur Verfügung.
Die 16-Bit-Register BC, DE, HL, IX, IY können mit den Befehlen PUSH BC, PUSH DE

usw. auf den Stapel kopiert werden. Das heißt, nach Ausführung des PUSH BC-Befehls ist der Inhalt von BC unverändert, gleichzeitig liegt er aber auch auf dem Stapel. Mit dem Befehl POP BC wird das oberste Element vom Stapel heruntergenommen und ins BC-Register gespeichert. Für die anderen 16-Bit-Register gilt das entsprechend. Weiterhin gibt es die Befehle PUSH AF und POP AF, die den Akkumulator und das Flaggenregister gleichzeitig auf den Stapel kopieren bzw. vom Stapel herunternehmen.
Die PUSH- und POP-Befehle sind für verschiedene Zwecke sehr praktisch: Erstens läßt sich mit ihrer Hilfe der Inhalt eines Doppelregisters sehr schnell in ein anderes Doppelregister kopieren:

```
PUSH IX
POP DE
```

kopiert den Inhalt von IX nach DE. Das kann man natürlich auch mit den LD-Befehlen

```
LD (&4000),IX
LD DE,(&4000)
```

bewirken, wobei die Speicherstellen &4000 und &4001 als Zwischenspeicher benutzt werden. Der dazugehörige Maschinencode ist aber acht Bytes lang gegenüber drei Bytes der eben verwendeten PUSH-POP-Kombination, und die Ausführungszeit ist fast doppelt so lang.
Zweitens kann man den Stapel zur vorübergehenden Sicherung der Registerinhalte verwenden. In unserem Programm benutzen Haupt- und Unterprogramm dieselben Register. Beispielsweise wird IX im Hauptprogramm als Zeiger auf das Bitmuster verwendet und muß dort immer denselben Wert &AB80 behalten. Im Unterprogramm ändert es dagegen dauernd seinen Wert, da es nacheinander alle Bytes des Bitmusters adressiert. Um keine Fehler durch die Mehrfachbenutzung der Register entstehen zu lassen, muß man dafür sorgen, daß die aktuellen Werte der Register, die im Unterprogramm verändert werden, am Beginn des Unterprogramms mit den PUSH-Befehlen abgespeichert und am Ende mit den POP-Befehlen wieder in die richtigen Register zurückgeladen werden. Dabei ist wichtig, daß wegen der LIFO-Struktur des Stapels die Reihenfolge der Register beim PUSHen umgekehrt ist wie die beim POPen: Also

```
PUSH HL
PUSH IX
....
POP IX
PUSH HL
```

und nicht

```
PUSH HL
PUSH IX
....
POP HL
POP IX
```

Schauen Sie sich das einmal im Unterprogramm an, das ab Adresse ZP im Programm P7 dieses Kapitels steht.
Das Speichern und Wiederladen kann auch unmittelbar vor und hinter dem CALL-Befehl erfolgen. Das ist immer dann notwendig, wenn ein Betriebssystemprogramm aufgerufen wird, das ein gerade benutztes Register selber verwendet. Es treten dann unerklärliche Fehler auf, die sich durch Zwischenspeichern der Register abstellen lassen. Man muß jedoch darauf achten, daß zueinandergehörende PUSH- und POP-Befehle nicht teils im Haupt- und teils im Unterprogramm vorkommen.

1.

```
PUSH DE
CALL nn
POP DE
```

2.

```
PUSH DE
CALL nn
   :
nn :
   :
POP DE
RET
```

} Unterprogramm

Tabelle 2 *Zwischenspeicherung von Registern auf dem Stapel*

Die erste Befehlsfolge in Tabelle 2 ist eine korrekte Möglichkeit zur Zwischenspeicherung, die zweite führt höchstwahrscheinlich zum Zusammenbruch des Systems, da hier DE-Inhalt und Fortsetzungsadresse vertauscht werden. Denn CALL entspricht im wesentlichen einem PUSH PC und RET einem POP PC, so daß die falsche Reihenfolge

```
PUSH DE
PUSH PC
...
POP DE
POP PC
```

entsteht.

Zusammenfassung

Mit dem Befehl CALL nn wird ein Unterprogramm aufgerufen, das bei der Adresse nn beginnt. Jedes Unterprogramm muß mit RET enden, wenn die Kontrolle wieder zum aufrufenden Programm zurückgehen soll.
Mit PUSH- und POP-Befehlen kann der Inhalt der 16-Bit-Register sowie der Kombination von Akkumulator und Flaggenregister auf dem Stapelspeicher zwischengespeichert und wieder abgehoben werden. Beim Zwischenspeichern mehrerer Register muß die Reihenfolge der Ablage (PUSH) umgekehrt wie die Reihenfolge der Wegnahme (POP) sein. Die Zahl der PUSH-Befehle muß immer gleich der Zahl der POP-Befehle sein.

Übung 13 Schreiben Sie ein Programm, das mit Hilfe der Systemau... routine &BB5A den Bildschirm mit der Ziffer „0" vollschrei... ASCII-Code des auszugebenden Zeichens muß der Syste... tine im Akkumulator übergeben werden.
Hinweis: Der ASCII-Code von „0" ist &30. Das Programm &BB... verändert keine Register.

Übung 14 Das Register HL soll den Inhalt &1000, das Register BC den Inhalt &EEEE haben. Geben Sie die Inhalte der Register an nach Ausführung der Befehlsfolgen

a)
```
PUSH HL
POP BC
```

b)
```
PUSH HL
PUSH BC
POP HL
POP BC
```

Zusatzinformationen

Der Stapelspeicher fängt beim CPC bei der Speicherstelle &BFFF an und wird zu niedrigeren Adressen hin aufgebaut (oberhalb &BFFF liegt ja der Bildschirmspeicher). Die höchsten Adressen liegen also am Boden, die niedrigeren an der Oberfläche des Stapels. Da alle 16-Bit-Daten zwei Speicherstellen benötigen, wird auch hier zuerst das H-Byte an der höheren Adresse und dann das L-Byte an der niedrigeren Adresse (also an der Stapeloberfläche) abgespeichert. Um sich im Stapel zurechtzufinden, benutzt der Prozessor das 16-Bit-Register SP (Stackpointer = Stapelzeiger). Dieser Stapelzeiger enthält normalerweise die Adresse des L-Bytes des letzten Eintrags in den Stapel (also der Stapeloberfläche). Bei je-

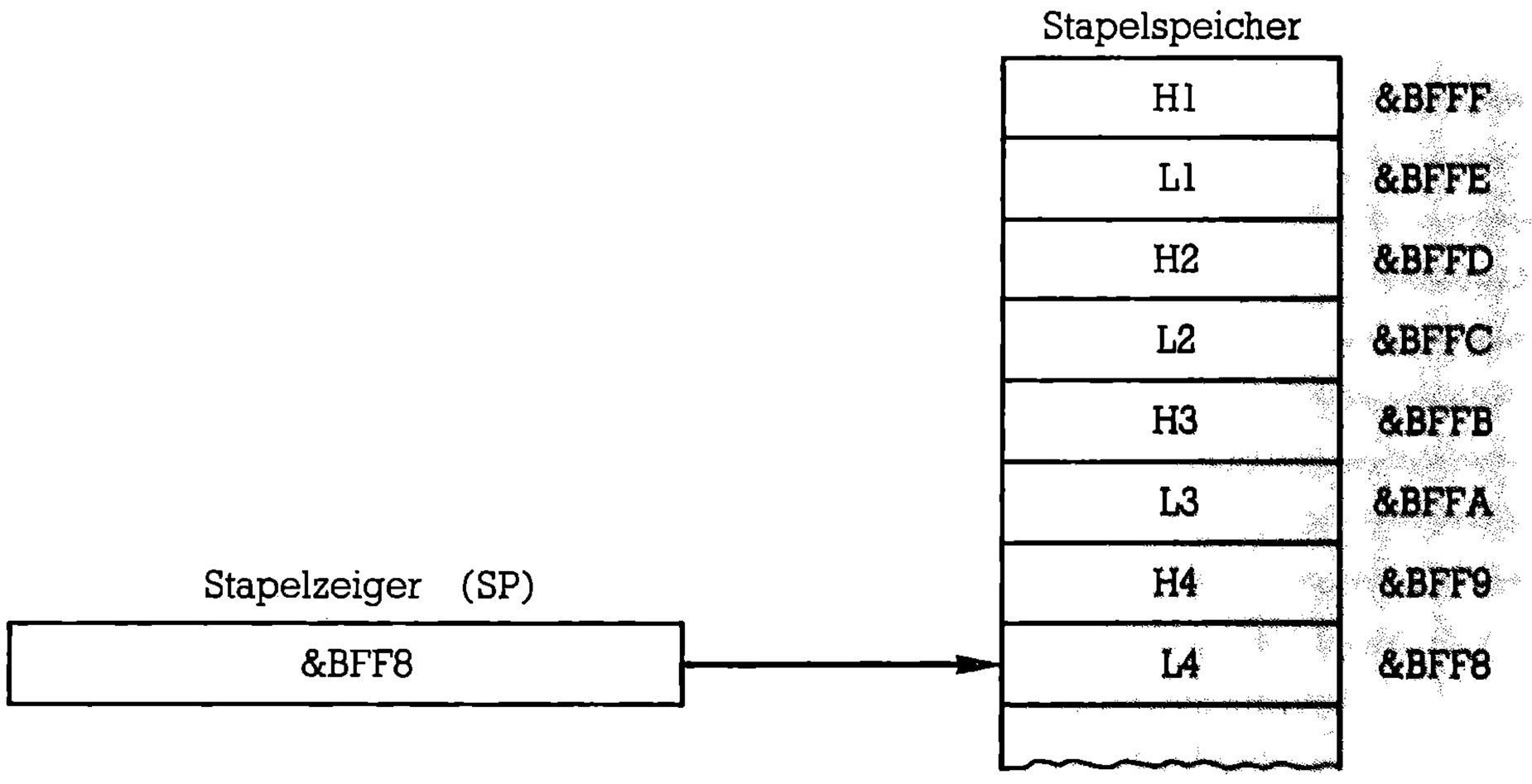


Bild 6 *Stapelspeicher (Stack) und Stapelzeiger (SP) nach vier Einträgen.*

dem PUSH- und CALL-Befehl wächst der Stapel um zwei Stellen (H-Byte und L-Byte). Das SP-Register wird also um den Wert 2 dekrementiert (Wachstum nach unten!), entsprechend erhöht sich der SP-Inhalt bei jedem POP und RET um 2.
Jeder RET-Befehl kann durch

```
POP HL
JP (HL)
```

ersetzt werden. Durch POP wird die Rücksprungadresse vom Stapel geholt und SP um 2 erhöht; JP (HL) versorgt dann den Programmzähler PC mit der Rücksprungadresse, und das Programm wird dort fortgesetzt.
Entsprechend könnte der Befehl CALL nn durch die Kombination

```
LD HL,mm
PUSH HL
JP nn
```

ersetzt werden, wobei mm die Adresse des ersten Befehls nach JP nn ist. Es wird natürlich nicht empfohlen, CALL und RET so zu ersetzen, aber die Wirkungsweise und das Arbeiten mit dem Stapel kann man sich so recht gut klarmachen.

Manchmal kann es notwendig werden, im Stapel „herumzuwühlen". Zum Beispiel kann man sich vorstellen, daß es beim Auftreten bestimmter Bedingungen praktisch sein kann, aus einem Unterprogramm direkt ins Betriebssystem zurückkehren zu können (oder bei komplizierteren Verschachtelungen aufgerufene Unterprogramme zu überspringen). Dazu muß vor dem RET-Befehl das SP-Register um so viele Stellen inkrementiert werden, daß es auf die Rücksprungadresse des Hauptprogramms zeigt. Zur Beeinflussung des SP-Registers gibt es die Befehle

```
INC SP
DEC SP
```

sowie Ladebefehle, die SP als Empfänger aufweisen. Natürlich ist es klar, daß der kleinste Fehler hier zu einem Systemabsturz führt.
Das folgende kleine Programm demonstriert diesen Umgang mit dem Stapel:

```
4000                    1       ORG   #4000
4000  3E41              2       LD    A,#41          ;ASCII-CODE VON "A".
4002  CD5ABB            3       CALL  #BB5A          ;"A" AUSGEBEN.
4005  CD0E40            4       CALL  UP             ;AUSGABE VON "B".
4008  3E43              5       LD    A,#43          ;ASCII-CODE VON "C".
400A  CD5ABB            6       CALL  #BB5A          ;"C" AUSGEBEN.
400D  C9                7       RET                  ;FERTIG.
                        8 ;
400E  3E42              9 UP:   LD    A,#42          ;ASCII-CODE VON "B".
4010  CD5ABB           10       CALL  #BB5A          ;"B" AUSGEBEN.
4013  33              11       INC   SP             ;STAPELZEIGER
4014  33              12       INC   SP             ;ZURUECKSETZEN.
4015  C9              13       RET                  ;ZUM HAUPTPROG.?
                       14 ;P8

UP      400E
```

Hier wird dreimal hintereinander das Betriebssystems-Unterprogramm zur Ausgabe von Zeichen aufgerufen. Es werden nacheinander die ASCII-Codes A (&41), B (&42) und C (&43) im Akkumulator übergeben. Der zweite Aufruf (der den Buchstaben B) erfolgtr über das Unterprogramm UP bei der Adresse &4[illegible]. Wenn die beiden Befehle INC SP fehlen, erscheinen auf dem Bildschirm nach dem Aufruf des Programms die Buchstaben ABC. Mit den Befehlen INC SP erscheint nur AB. Wegen der vorhin besprochenen Manipulation des Stapelzeigers kehrt das Programm aus UP ohne den Umweg über den dritten Ausdruckbefehl direkt ins Betriebssystem zurück.
Den vom Betriebssytem eingerichteten Stapel ab &C000 muß man nicht unbedingt benutzen. Der Befehl

LD SP,nn

ermöglicht es dem Programmierer, einen eigenen Stapel einzurichten. Alle weiteren PUSH-, POP-, CALL- und RET-Befehle beziehen sich dann auf den neu gesetzten Stapelzeiger. Daß bei der Einrichtung eines „privaten" Stapels größte Umsicht nötig ist, braucht kaum erwähnt zu werden.
Sollten Sie einmal an einem Z80-Rechner Assemblerprogramme schreiben, bei dem Sie nicht wissen, ob er Ihnen einen Stapel-Bereich einrichtet (oder wie groß dieser ist), dann legen Sie als erstes einen eigenen Stapel an. Nicht nur für Assembler-Anfänger sind Programmfehler, die mit dem Stapel zusammenhängen, sehr schwer zu finden.
Beim Programmieren mit dem weitverbreiteten Betriebssystem CP/M hat sich das Anlegen eines eigenen Stapels regelrecht eingebürgert, denn für fast alle Anwendungen reicht der von CP/M reservierte Stapel von der Größe her nicht aus.

Wir zerlegen Bytes und setzen große Zeichen

12

In diesem Kapitel werden die Rotations- und Schiebebefehle vorgestellt, für die es in BASIC kein Gegenstück gibt. Mit Hilfe dieser Befehle kann das Bitmuster eines Bytes verändert und die Werte einzelner Bits abgefragt werden.

Rotations- und Schiebebefehle

In den letzten Kapiteln haben wir Zeichen mit Hilfe von Bitmustern aufgebaut. Jedes der 64 Bits des Musters bestimmt im Bildschirmspeicher den Zustand eines Punktes auf dem Bildschirm. Wir wollen jetzt das Zeichen so vergrößern, daß jedes Bit des Bitmusters ein ganzes Zeichen setzt. Wenn wir diese Zeichen wieder in acht Reihen mit je acht Kästchen anordnen, dann erhalten wir die ursprüngliche Form achtmal in jede Richtung vergrößert.

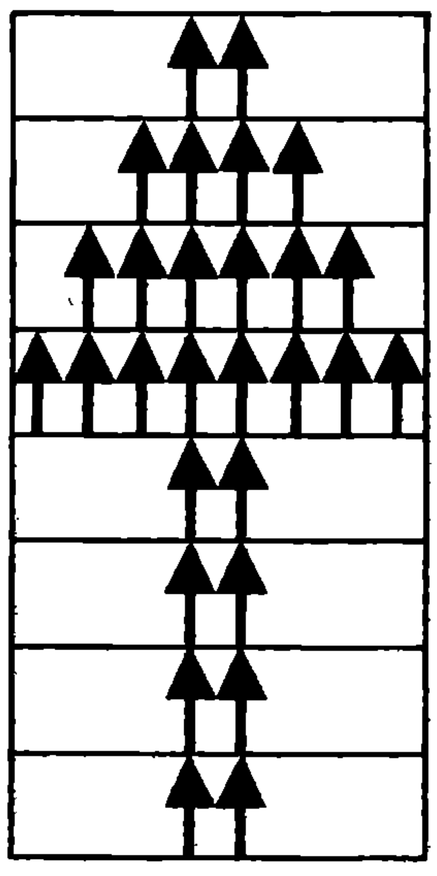


Bild 7 *Ausgabe eines großen Zeichens.*

Um die Information über einzelne Bits zu erhalten, stellt der Z80 eine ganze Reihe von Befehlen zur Verfügung. Das sind die sog. Rotations- und Schiebebefehle (englisch „rotate“ und „shift“). Wir betrachten zunächst die Befehle

RL (rotate left)
RLC (rotate left circular)
SLA (shift left arithmetic)

die die 8-Bit-Register direkt sowie die Speicherstellen des RAM indirekt über HL, IX und IY adressieren können. Das heißt, es gibt die Befehle

RL A
RL B
...
RL (HL)
RL (IX+d)
RL (IY+d)

und entsprechende Befehle für RLC und SLA.
Die Wirkungen dieser drei Befehle bzw. Befehlsgruppen werden in den Bildern 8, 9 und 10 gezeigt.

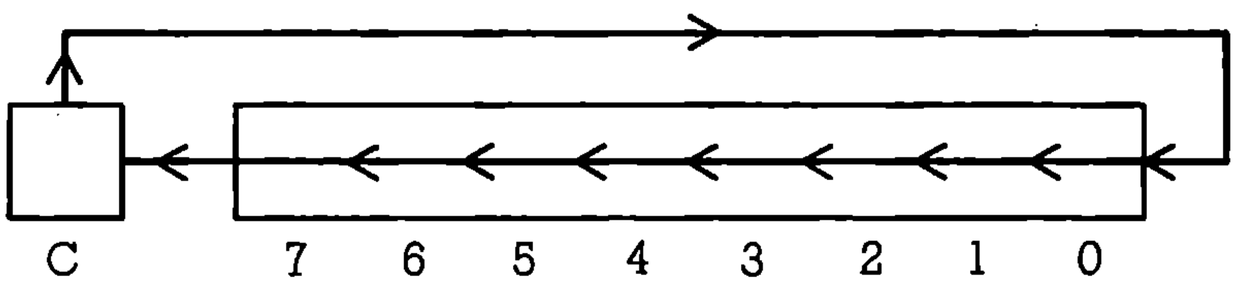


Bild 8 *RL: Byte links durch das Carry rotieren (Rotate left). Bits 0 bis 6 werden jeweils um eine Stelle nach links geschoben. Bit 7 wird in die Carry-Flagge geschoben, und deren Inhalt gelangt in Bit 0.*

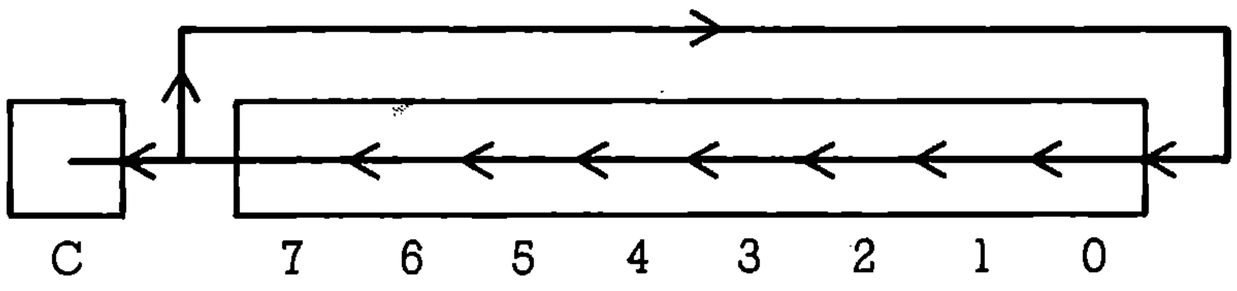


Bild 9 *RLC: Byte links im Kreis rotieren (Rotate Left Circular). Bits 0 bis 6 werden jeweils um eine Stelle nach links geschoben. Bit 7 wird sowohl ins Bit 0 als auch in die Carry-Flagge geschoben.*

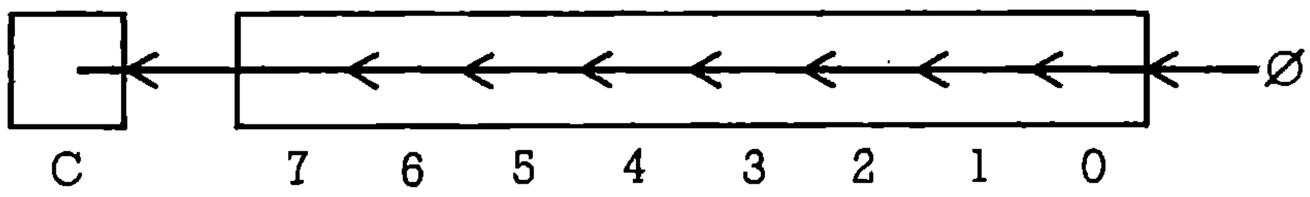


Bild 10 *SLA: Byte nach links schieben (Shift Left Arithmetic). Bits 0 bis 6 werden jeweils um eine Stelle nach links geschoben. Bit 7 gelangt in die Carry-Flagge. Ins Bit 0 wird der Wert 0 nachgeschoben.*

Um einen ersten Eindruck von der Wirkung der Rotations- und Schiebebefehle zu erhalten, ändern wir das Zeichenausgabeprogramm P6 aus Kapitel 10 etwas ab. Vor den Befehl INC IX fügen wir den Befehl RLC (IX+&00) ein. Wird das Programm ausgeführt, so rotiert aufgrund des neuen Befehls jedes Byte des Bitmusters um eine Bitposition nach links. Das entspricht auf dem Bildschirm gerade einer Punktbreite, und wenn das Programm mehrfach hintereinander aufgerufen wird, so können wir beobachten, wie der dargestellte Pfeil sich am linken Rand der Zeichenposition herausschiebt und wie die verschwundenen Teile des Pfeils am rechten Rand der Zeichenposition wieder hereinkommen. Nach acht Programmdurchläufen hat der Pfeil seine ursprüngliche Form wieder angenommen, da jedes Byte des Bitmusters nach acht Ein-Bit-Rotationen seinen Ausgangszustand erreicht hat.
Ersetzt man den Befehl RLC (IX+&00) entweder durch SLA (IX+&00) oder durch RL (IX+&00), so schiebt sich bei jeder Programmausführung der Pfeil um eine Punktbreite am linken Rand aus der Zeichenposition heraus. Nach acht Programmdurchläufen ist die Bildschirmposition leer.
Bei dem Befehl SLA ist dieses Verhalten leicht verständlich, da für jedes links aus einem Byte herausgeschobene Bit rechts der Wert 0 „nachgefüllt" wird. Bei dem Befehl RL wird die Null auf der rechten Seite aus der Carry-Flagge übernommen, die (wie im Kapitel 10 besprochen) hier durch den ADD-Befehl gelöscht wird. Lediglich beim ersten Durchlaufen der bei der Marke L1 beginnenden Schleife wird der am Programmanfang vorhandene Wert der Carry-Flagge in das erste Byte des Bitmusters übergeben.
Soll das durch diese Programme zerstörte Bitmuster des Pfeils wiederhergestellt werden, so ist das durch die BASIC-Anweisung „SYMBOL AFTER 240" möglich.
Auf den ersten Blick sehen die neuen Befehle vielleicht etwas kurios aus. Man kann jedoch sehr viel damit anfangen. Der SLA-Befehl z. B. eignet sich dazu, eine Zahl mit 2, 4, usw. zu multiplizieren, was wir später noch besprechen und auch ausnutzen werden. Andererseits sind wir jetzt in der Lage, einzelne Bits eines Bytes zu überprüfen. Nach spätestens acht Rotations- oder Schiebebefehlen haben wir jedes Bit in die Carry-Flagge gebracht, und dort können wir es mit der C- oder NC-Bedingung eines entsprechenden Befehls abfragen.

Ein Programm für große Zeichen

Das durch die Kapitelüberschrift angekündigte Programm zur Ausgabe eines großen Zeichens hat folgendes Aussehen:
Zuerst wird Mode 2 eingeschaltet. Dann werden das IX- und IY-Register mit der Startadresse des Bitmusters von „Pfeil nach oben" geladen, und schließlich zeigt das HL-Register auf eine Adresse in der oberen Mitte des Bildschirmspeichers. Das ist der linke obere Eckpunkt des 8x8-Zeichenfeldes, in dem der große Pfeil

```
4000                    1           ORG   #4000
4000  3E02              2           LD    A,#02          ;MODE 2 SETZEN.
4002  CD0EBC            3           CALL  #BC0E          ;ZEIGER AUF BIT-
4005  DD2180AB          4           LD    IX,#AB80       ;MUSTER UND
4009  FD2180AB          5           LD    IY,#AB80       ;BILDSCHIRMADRESSE.
400D  2170C0            6           LD    HL,#C070       ;SCHLEIFENZAEHLER AUSSEN
4010  1608              7           LD    D,#08          ;SCHLEIFENZAEHLER INNEN.
4012  1E08              8 L1:       LD    E,#08          ;BITMUSTER NACH B.
4014  FD4600            9           LD    B,(IY+#00)     ;BITWEISE INS CARRY.
4017  CB20             10 L2:       SLA   B              ;DE AUF DEN STAPEL.
4019  D5               11           PUSH  DE             ;ZEICHENAUSGABE WENN
401A  DC2E40           12           CALL  C,ZP           ;CARRY GESETZT.
                       13                                ;DE ZURUECKHOLEN.
401D  D1               14           POP   DE             ;NAECHSTE ZEICHENPOS..
401E  23               15           INC   HL             ;ENDE INNERE SCHLEIFE.
401F  1D               16           DEC   E              ;
4020  C21740           17           JP    NZ,L2          ;NACHSTES BYTE DES BITM.
4023  FD23             18           INC   IY             ;ADRESSE FUR ANFANG
4025  014800           19           LD    BC,#0048       ;DER NAECHSTEN ZEILE.
4028  09               20           ADD   HL,BC          ;ENDE DER
4029  15               21           DEC   D              ;AEUSSEREN SCHLEIFE.
402A  C21240           22           JP    NZ,L1          ;FERTIG.
402D  C9               23           RET
                       24 ;
402E  DDE5             25 ZP:       PUSH  IX             ;IX AUF STAPEL.
4030  E5               26           PUSH  HL             ;HL AUF STAPEL.
4031  110008           27           LD    DE,#0800       ;ZEICHENSETZPROG.
4034  DD7E00           28 L3:       LD    A,(IX+#00)     ;WIE GEHABT.
4037  77               29           LD    (HL),A
4038  DD23             30           INC   IX
403A  19               31           ADD   HL,DE
403B  D23440           32           JP    NC,L3
403E  E1               33           POP   HL             ;HL ZURUEKHOLEN.
403F  DDE1             34           POP   IX             ;IX ZURUEKHOLEN.
4041  C9               35           RET                  ;ZUM HAUPTPROGRAMM.
                       36 ;P9

L1        4012   L2        4017   L3        4034
ZP        402E
```

Änderung für den CPC664/6128: In den Zeilen 4 und 5 muß jeweils die Adresse &AB80 durch &A67C ersetzt werden.

erscheinen wird. D- und E-Register werden als Schleifenzähler für zwei geschachtelte Schleifen verwendet. Die äußere (E)-Schleife läuft über die acht Bytes des Bitmusters, die innere Schleife (D) sorgt dafür, daß alle acht Bits jedes Bytes in die Carry-Flagge geschoben werden, wo man ihren Wert feststellen kann. Anfangsadresse der äußeren Schleife ist L1 = &4012. Zunächst wird mit Hilfe der indirekten Adressierung über das IY-Register das erste Byte des Musters ins B-Register gespeichert. Dann beginnt die innere Schleife: Die im B-Register befindlichen Bits werden nacheinander in die Carry-Flagge geschoben. Wenn sie den Wert 1 haben, wird unser Zeichensetzprogramm, dessen Startadresse ZP ist, durch den bedingten Aufruf CALL C, ZP aktiviert.
Die beiden Befehle PUSH DE und POP DE vor und nach dem Unterprogrammaufruf sorgen für eine Zwischenspeicherung der Schleifenzähler D und E, da das DE-Register im ZP-Programm benutzt wird. Das HL-Register, in dem die Startadresse der nächsten zu besetzenden Zeichenposition an das ZP-Programm übergeben wird, wird anschließend um 1 erhöht.

Nach der Ausgabe einer Zeichenreihe wird die Schleife verlassen und mit INC IY der Zeiger IY auf das nächste Byte des Bitmusters gesetzt. Außerdem muß der Bildschirmzeiger HL auf den Beginn der nächsten Achterreihe von Zeichen gesetzt werden. (Eine Grobzeile nach unten, acht Zeichen zurück.) Dazu muß zur Adresse des Reihenendes die Zahl 80—8=72(&48) addiert werden.
Das Unterprogramm ZP zur Zeichenausgabe entspricht dem des Programms P7. Nur die Startadresse hat sich von &401A auf &402E verändert.
Das Programm läßt eine ganze Reihe von Alternativen zu. Zunächst kann der SLA-Befehl natürlich durch RL oder RLC ersetzt werden, ohne daß sich irgend etwas ändert. Dann könnte man versuchen, die Zwischenspeicherung im B-Register zu umgehen und die Verschiebung mit SLA (IY+&00) im Bitmuster selbst durchzuführen. Das funktioniert aber nicht, und am Ende des Programms ist das Bitmuster zerstört und muß vor dem nächsten Programmstart mit „SYMBOL AFTER 240“ wiederhergestellt werden. Verwenden wir statt SLA (IY+&00) den Befehl RLC (IY+&00), dann ist das Bitmuster zum Schluß wiederhergestellt, aber das ZP-Programm gibt verstümmelte Zeichen aus.

Zusammenfassung

Die Befehle RL, RLC und SLA können die Bitwerte eines Bytes rotieren bzw. verschieben. Außerdem werden diese Werte in die Carry-Flagge gebracht, wo sie abgefragt werden können.
CALL C,nn ruft das Unterprogramm bei nn nur auf, wenn die Carry-Flagge gesetzt ist. Entsprechend ruft CALL NC,nn das Unterprogramm auf, wenn die Carry-Flagge gelöscht ist.

Übung 15 Der Akkumulator soll den Wert &F0 enthalten, und die Carry-Flagge soll gelöscht sein. Geben Sie den Inhalt des Akkumulators (duale und hexadezimale Darstellung) und den Zustand der Carry-Flagge nach der Ausführung folgender Befehle an:

a) RL A
RL A

b) RLC A
RLC A

c) SLA A
SLA A

Übung 16 Ändern Sie das Programm so ab, daß das große Zeichen invers ausgegeben wird, d. h., ein gesetztes Bit des Bitmusters soll ein Leerzeichen, ein gelöschtes Bit ein Zeichen setzen.
Hinweis: Es genügt die Änderung eines einzigen Befehls.

Übung 17 Geben Sie das große Zeichen so aus, daß es auf dem Kopf steht.

Neben den Befehlen RL, RLC, SLA gibt es noch Rotations- und Schiebebefehle nach rechts:

RR (rotate right),
RRC (rotate right circular)
SRL (shift right logical)
SRA (shift right arithmetic)

Die Wirkung dieser Befehle ist in Bild 11 beschrieben.

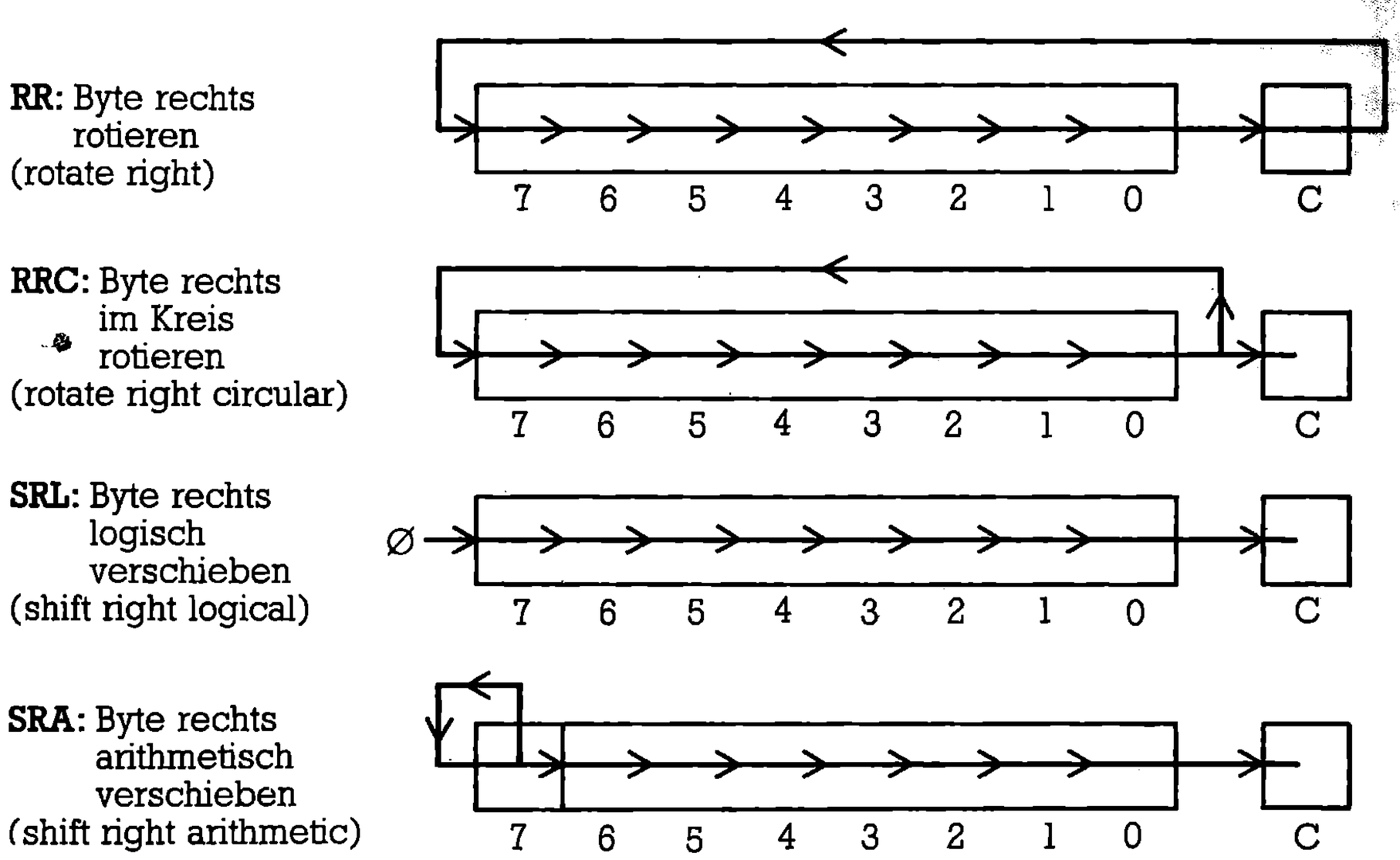


Bild 11 *Rotations- und Schiebebefehle nach rechts.*

Der Unterschied zwischen SRL (logische Rechtsverschiebung) und SRA (arithmetische Rechtsverschiebung) besteht darin, daß bei SRL immer der Wert Null in das Bit 7 nachgeschoben wird, während bei SRA der in Bit 7 vorhandene Wert stets erhalten bleibt. Das hat etwas damit zu tun, daß bei Rechenoperationen das 7. Bit häufig als Vorzeichen interpretiert wird und daher manchmal erhalten bleiben soll.

Sämtliche Rotationen und Verschiebungen lassen sich auf die 8-Bit-Register A, B, C, D, E, H, L anwenden und erlauben eine indirekte Adressierung von Speicherplätzen durch (HL) bzw. (IX+&00), bzw. (IY+&00).

Die vier Rotationsbefehle gibt es auch in Spezialvarianten für die Rotation des Akkumulators. Neben RR A (Code &CB1F) gibt es noch den Befehl RRA (Code &1F). Beide Befehle beeinflussen den Akkumulator in gleicher Weise, wirken aber unterschiedlich auf die Flaggen. RR A beeinflußt z. B. die Z-Flagge, RRA läßt sie unverändert. Dasselbe gilt auch für RRCA, RLA und RLCA.

Zusätzlich zu den besprochenen Befehlen gibt es noch zwei Spezialbefehle, die

ganze Nibbles (Halbbytes) zwischen dem Akkumulator und einer durch (HL) indirekt adressierten Speicherstelle verschieben:
RLD (Rotiere die Ziffern („digits") nach links, Code &ED6F) ist in Bild 12 beschrieben.

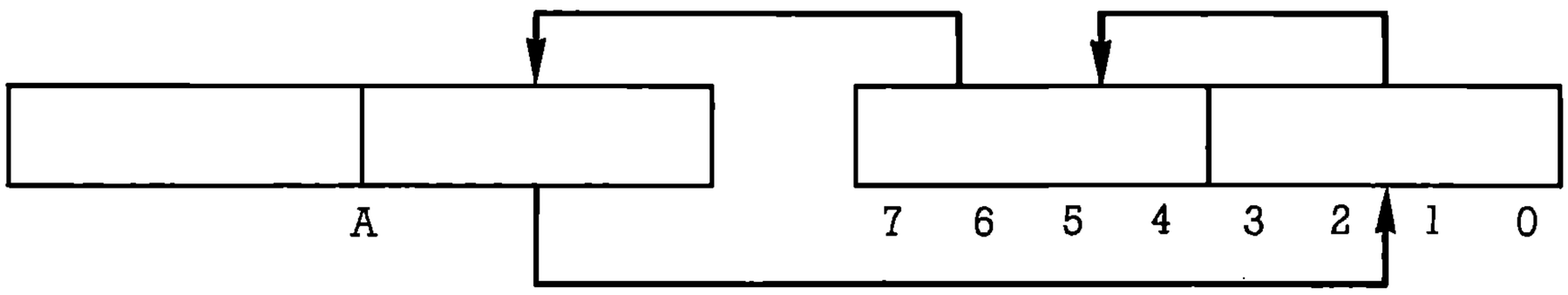


Bild 12 *RLD: Rotiere Ziffern nach links (Rotate Left Digits).*

RRD (Rotiere die Ziffern nach rechts, Code &ED67) ist in Bild 13 beschrieben.

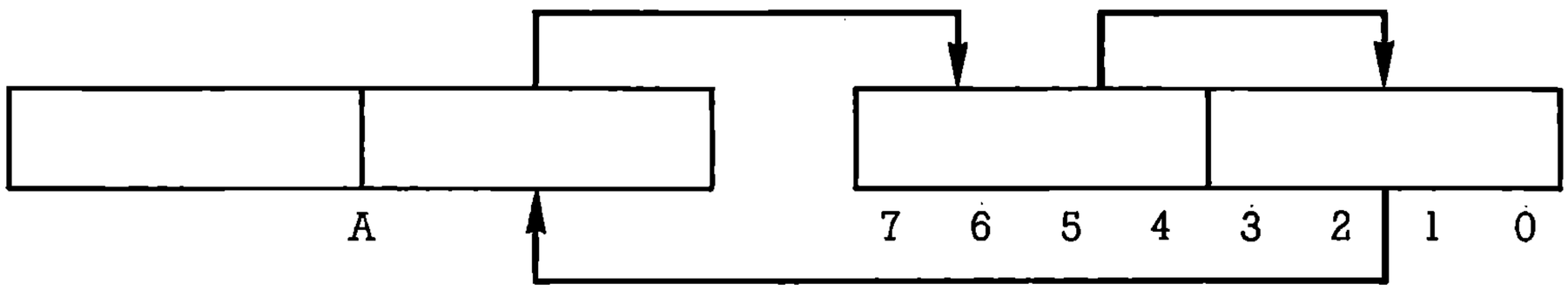


Bild 13 *RRD: Rotiere Ziffern nach rechts (Rotate Right Digits).*

Diese Befehle sind speziell zur Ziffern- bzw. Zeichenausgabe geeignet, und wir werden den Umgang damit in einem der nächsten Kapitel kennenlernen.

13 Das sanfte Gleiten

Im letzten Kapitel haben wir die Rotations- und Schiebebefehle zur Überprüfung einzelner Bits verwendet. Jetzt lernen wir, wie man mit Hilfe dieser Befehle ein Zeichen punktweise auf dem Bildschirm verschiebt.
Im Programm dieses Kapitels werden ein neuer Befehl und eine neue Adressierungsart vorgestellt: Mit dem CP-Befehl kann man den Akkumulatorinhalt mit dem Inhalt anderer Speicherstellen vergleichen, und die „indizierte" Adressierung erweist sich als eine Erweiterung der „indirekten" Adressierung.

„Soft-scrolling"

Sicher haben Sie bei einem Computerspiel schon gesehen, wie sich Fahrzeuge oder Figuren ruckfrei auf dem Bildschirm bewegen. Auch diese gleitenden Bewegungen können mit Rotations- und Schiebebefehlen realisiert werden.
Wenn wir ein Zeichen von einer Bildschirmposition zur nächsten bewegen, bedeutet das eine Verschiebung um acht Bildschirmpunkte. Um einen Ruck zu verhindern, dürfen wir das Bitmuster nicht in Achtergruppen in die nächste Bildschirmadresse übertragen, sondern müssen es bitweise verschieben.
Betrachten wir zwei nebeneinanderliegende Achtergruppen von Bildschirmpunkten, dann sieht eine Rechtsverschiebung so aus, wie in Bild 14 gezeigt wird.

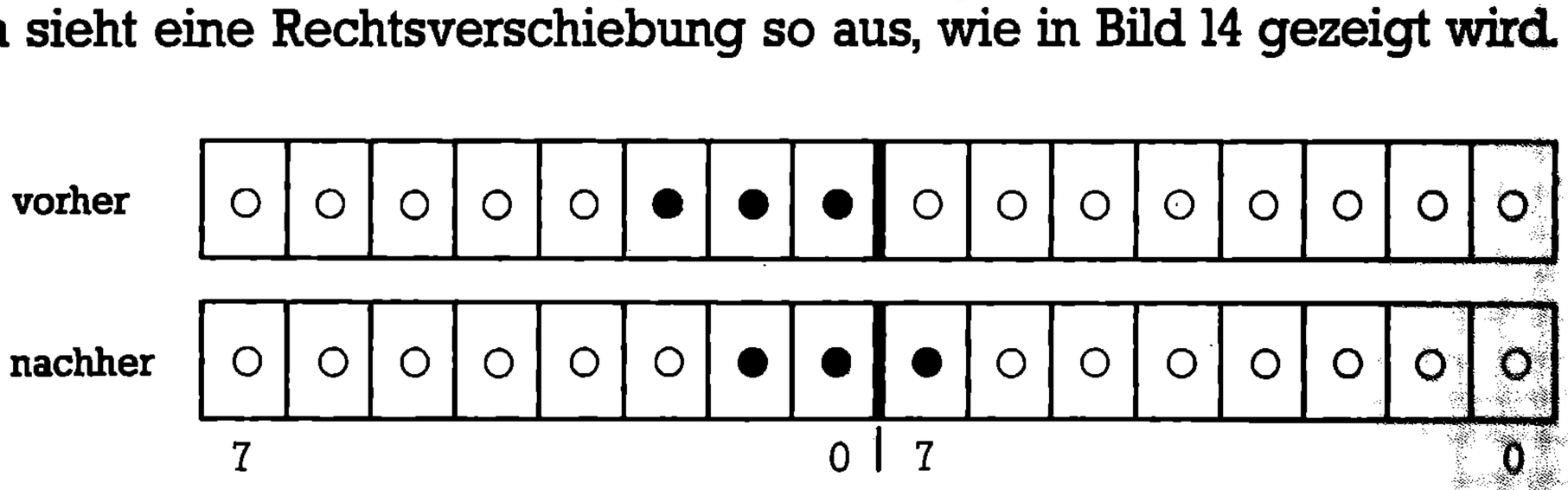


Bild 14 *Sanftes Gleiten: Verschiebung des Bitmusters um einen Punkt.*

Die entsprechenden Bytes im Bildschirmspeicher können mit einer Kombination von Schiebe- und Rotationsbefehlen genau in der erforderlichen Weise beeinflußt werden. Das folgende Programm verschiebt ein kleines Männchen aus dem Zeichensatz (CHR$(250)) nach rechts:

```
4000                    1           ORG   #4000
4000  3E02              2           LD    A,#02
4002  CD0EBC            3           CALL  #BC0E                ;MODE 2 SETZEN.
4005  3EFA              4           LD    A,#FA                ;MAENNCHEN
4007  CD5ABB            5           CALL  #BB5A                ;AUSGEBEN.
400A  010008            6           LD    BC,#0800             ;FEINZEILENABSTAND.
400D  2100C0            7           LD    HL,#C000             ;ZEIGER AUF BILDS.ANF..
4010  1E08              8 L1:       LD    E,#08                ;SCHLEIFENZAEHLER.
4012  CD06BB            9 L2:       CALL  #BB06                ;TASTATURABFRAGE.
4015  FE51             10           CP    #51                  ;TASTE "Q" ?
4017  C8               11           RET   Z                    ;WENN JA,DANN ENDE.
4018  FE52             12           CP    #52                  ;TASTE "R" ?
401A  C21240           13           JP    NZ,L2                ;WENN NEIN,WEITERFRAGEN.
401D  CD2840           14           CALL  VER                  ;SONST ZEICHEN
                       15                                      ;VERSCHIEBEN.
4020  1D               16           DEC   E                    ;NACH ACHT
4021  C21240           17           JP    NZ,L2                ;VERSCHIEBUNGEN ZUR
4024  23               18           INC   HL                   ;NAECHSTEN BILDSCHIRM-
                       19                                      ;POSITION.
4025  C31040           20           JP    L1                   ;ZUM ANFANG ZURUECK.
                       21 ;
4028  E5               22 VER:      PUSH  HL                   ;BILDSCHIRMPOSITION
4029  DDE1             23           POP   IX                   ;INS IX-REGISTER.
402B  DDCB003E         24 L3:       SRL   (IX+#00)             ;BYTE UM EINEN PUNKT
402F  DDCB011E         25           RR    (IX+#01)             ;WETERSCHIEBEN.
4033  DD09             26           ADD   IX,BC                ;NAECHSTE FEINZEILE.
4035  D22B40           27           JP    NC,L3                ;WENN FERTIG,
4038  C9               28           RET                        ;ZURUECK.
                       29 ;P10

L1        4010   L2        4012   L3        402B
VER       4028
```

Dieses Programm enthält einige neue Techniken, die hier zunächst erklärt werden sollen. Wir verwenden hier zum ersten Mal ein Programm des Betriebssystems: Die Systemroutine &BB06 unterbricht unser Programm solange, bis eine Taste gedrückt wird. Der ASCII-Code des Tastenzeichens steht danach im Akkumulator.

Der CP-Befehl

Um den Tasteninhalt abzufragen, benutzen wir den CP-Befehl. CP kommt von „compare", vergleichen. Mit Hilfe dieses Befehls wird der Inhalt des Akkumulators mit der im CP-Befehl als Operand angegebenen Speicherstelle oder Zahl verglichen. Zum Vergleich können die Register A, B, C, D, E, H, L, Zahlen zwischen &00 und &FF sowie die durch (HL), (IX+&00) und (IY+&00) indirekt adressierten Speicherstellen herangezogen werden. So vergleicht

CP B

den Inhalt von A mit dem des B-Registers;

CP (HL)

vergleicht den Inhalt von A mit der Speicherstelle, deren Adresse in HL steht, und

CP &10

vergleicht den Inhalt von A mit der Zahl &10.
Bei allen Vergleichen bleibt der Inhalt von A erhalten, und das Ergebnis kann an der Stellung der Flaggen abgelesen werden. Insbesondere gilt: Wenn der Vergleich ein „Gleich" ergibt, wird die Z-Flagge gesetzt; bei „Ungleich" wird sie zurückgesetzt. Mit JP Z,nn oder JP NZ,nn kann nach dem Vergleich entsprechend verzweigt werden. Die C-Flagge wird gesetzt, wenn der Wert in A kleiner als der Vergleichswert ist, sonst wird sie zurückgesetzt. Während also die Abfrage der Z-Flagge die Entscheidung zwischen „Gleich" und „Ungleich" erlaubt, kann man mit der C-Flagge zwischen „kleiner" (<) und „größer oder gleich" (≥) unterscheiden. (Tabelle in der Zusammenfassung.)

Das Hauptprogramm

Am Anfang des Hauptprogramms wird der Mode 2 eingeschaltet, und ein kleines Männchen (Code 250(&FA)) mit der Systemroutine &BB5A in der linken oberen Ecke ausgegeben. Dann wird HL wieder als Zeiger auf den Bildschirmbeginn gesetzt. Jetzt beginnt eine Schleife, die mitzählt, wie viele Verschiebungen bereits durchgeführt wurden. Nach jeweils acht Verschiebungen ist das Männchen eine Zeichenbreite vorgerückt, und der Verschiebevorgang kann an dieser Stelle von vorn beginnen. Das wird einfach dadurch erreicht, daß der Bildschirmzeiger (bei &4021) um den Wert 1 erhöht wird und danach zum Beginn der Schleife zurückgesprungen wird.
Um die Verschiebung nicht zu schnell durchzuführen und das Programm jederzeit abbrechen zu können, wird vor jeder Verschiebung die Tastatur mit der bereits erklärten Routine &BB06 abgefragt. Drückt man die Q-Taste (Großschreibung, ASCII-Code &51), dann ist wegen des Vergleichs CP &51 bei &4014 die Z-Flagge gesetzt, und das Programm wird beendet. Sonst wird mit der R-Taste (Code &52) verglichen. Ist eine andere Taste gedrückt worden, dann ist die Z-Flagge gelöscht und der JP NZ-Befehl verzweigt zur Tastaturabfrage zurück. Falls aber „R" gedrückt ist, rückt das Männchen mit Hilfe des Unterprogramms VER (bei &4025) eine Punktbreite weiter. Dann wird der Schleifenzähler E heruntergesetzt, um feststellen zu können, ob das Männchen wieder komplett in einen Bildschirmzeichenbereich vorgerückt ist oder noch auf zwei verschiedene Positionen verteilt ist.

Die Verschieberoutine

Das Kernstück des Verschiebeprogramms ist die Routine VER ab Adresse &4025. Dort wird zunächst die Position des Zeichens (genauer gesagt, die seiner obersten Zeile) aus HL in IX übergeben. Dann wird wieder der Feinzeilenabstand ins BC-Register geladen, und danach beginnt der eigentliche Verschiebevorgang. Welche Wirkung er auf eine Zeile des Bitmusters haben muß, ist in Bild 14 gezeigt. Um das zu erreichen, werden zunächst die Bits des obersten Bytes mit SRL (IX+&00) um eine Position nach rechts verschoben. Das Bit 0 wird dabei ins Carry übertragen. Der nächste Befehl RR (IX+&01) nimmt den Wert aus dem Carry auf und bringt ihn ins siebente Bit des nächsthöheren Speicherplatzes. Die folgenden Zeilen des Bitmusters werden dann genauso angesprochen wie in unserem Zeichenausgabeprogramm, und am Ende ist das ganze Zeichen um eine Punktposition auf dem Bildschirm nach rechts gerückt. Das ganze Bitmuster ist jetzt auf zwei benachbarte Zeichenbereiche und damit auf sechzehn Bytes verteilt. Erst nach acht Verschiebungen ist das Bitmuster wieder in einer Zeichenposition zusammen, und die Verschiebung kann bei der nächsten Bildschirmadresse weitergehen.

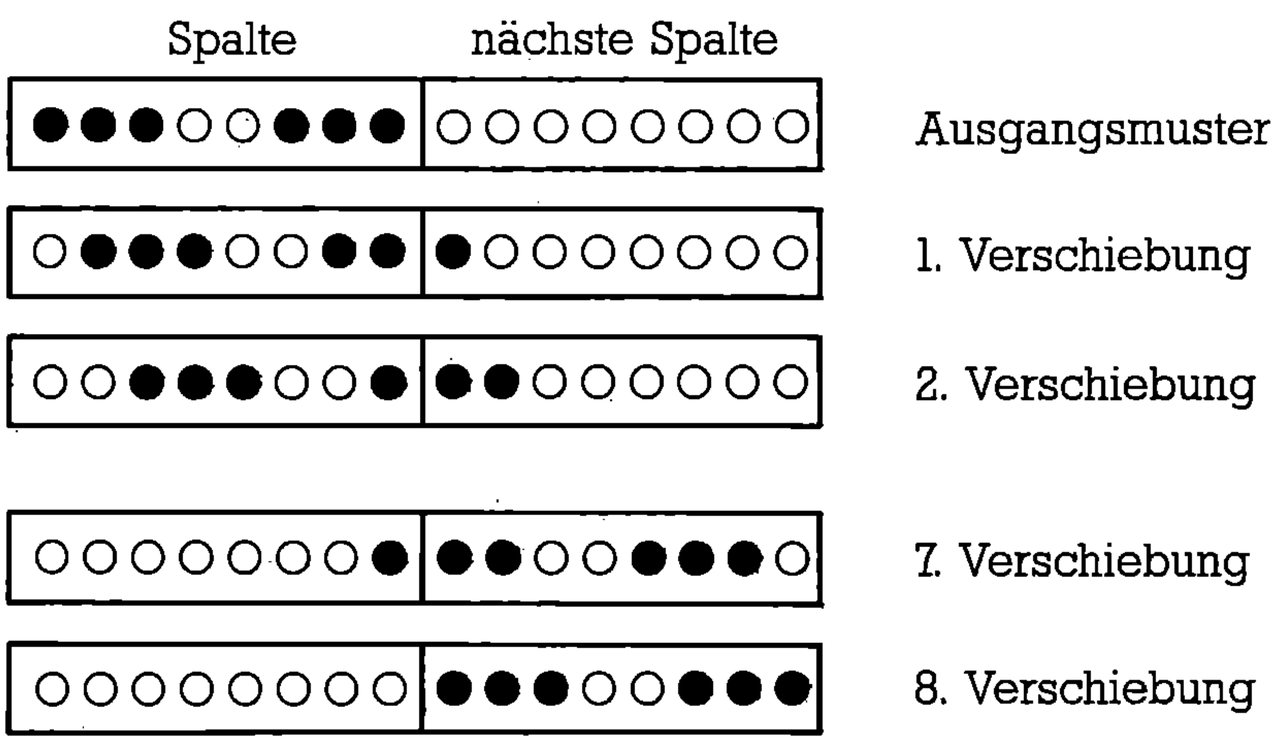


Bild 15 *Ein Bitmuster wird durch punktweise Verschiebungen auf zwei Zeichenpositionen verteilt. Erst nach acht Verschiebungen gehört das Bitmuster vollständig zur nächsten Zeichenposition.*

Die indizierte Adressierung

Sie haben sicher schon gemerkt, daß wir beim Befehl RR (IX+&01) eine neue Adressierungsart benutzt haben, die man „indizierte Adressierung" nennt. Eigentlich ist es eine Erweiterung der bereits bekannten indirekten Adressierung. Nur ergibt sich jetzt die Adresse der angesprochenen Speicherstelle nicht allein durch den in IX stehenden Wert, sondern es muß noch der Indexwert d in IX+d berücksichtigt werden. Wenn z. B. in IX der Wert &C000 steht, dann wird mit RR (IX+&01) die Speicherstelle &C001 angesprochen. (IX+&02) würde &C002 ansprechen, und so weiter. Das geht bis (IX+&7F). (Näheres erklären die Zusatzinformationen.)

Zusammenfassung

Der CP-Befehl vergleicht den Inhalt des Akkumulators mit dem hinter CP angegebenen Operanden. Das kann eine Zahl, ein 8-Bit-Register oder eine durch HL, IX, IY adressierte Speicherstelle sein. Dabei gilt:

Wenn A = Vergleichswert, dann Z-Flagge = 1
Wenn A ≠ Vergleichswert, dann Z-Flagge = 0
Wenn A < Vergleichswert, dann C-Flagge = 1
Wenn A ≥ Vergleichswert, dann C-Flagge = 0

Die „indizierte Adressierung" durch (IX+d) bzw. (IY+d) ist eine Erweiterung der indirekten Adressierung. Die Adresse der angesprochenen Speicherstelle ergibt sich aus dem Inhalt des IX- (bzw. IY-) Registers, zu dem der Wert von d addiert wird. d darf dabei nicht größer als 127 = &7F sein.

Übung 18 Was würde sich am Programmablauf von P10 ändern, wenn Sie den SRL-Befehl bei L3 durch den SRA-Befehl ersetzen würden? (Erst überlegen, dann ausprobieren!)

Übung 19 Was würde sich am Programmablauf ändern, wenn Sie den RR-Befehl in Zeile 25 von P10 durch den RRC-Befehl ersetzen würden?

Übung 20 Ändern Sie das Programm P10 so, daß das Männchen am Ende der ersten Zeile erscheint und dann mit der Taste L nach links bewegt werden kann. (Für die Positionierung des Cursors am Beginn des Programms verwendet man das BASIC-Kommando LOCATE 80,1 unmittelbar vor dem Aufruf des Maschinenprogramms durch CALL &4000.) Zur richtigen Änderung der indizierten Befehle müssen Sie die Zusatzinformationen gelesen haben.

Neben der Systemroutine &BB06 gibt es zur Abfrage der Tastatur noch die Routine &BB09. Dabei wird nicht auf den Tastendruck gewartet. Falls keine Taste gedrückt wurde, enthält der Akkumulator den Wert 0, sonst den ASCII-Wert der gedrückten Taste.
Bei der indizierten Adressierung kann der Wert von d in (IX+d) bzw. (IY+d) zwischen &00 und &FF = (255) betragen. Falls d größer ist als &7F (127), also ab &80 (128), wird jedoch d nicht mehr zu IX bzw. IY addiert, sondern der Wert (256-d) wird vom Wert des Indexregisters subtrahiert. So bedeutet (IX+&FF) in Wirklichkeit (IX—1) und (IX+&80) bedeutet (IX-&80). Ob addiert oder subtrahiert wird, entscheidet sich am Wert des führenden (siebten) Bits von d. Wenn dieser Wert 1 ist, wird in der angegebenen Weise subtrahiert. Diese spezielle Art der Arithmetik werden wir noch ausführlich besprechen.
Beim Maschinencode der indizierten Befehle ist zu beachten, daß der Indexwert d immer das dritte Byte ist. Wie schon vorhin erwähnt, kann man den Maschinencode der indiziert adressierten Befehle einfach aus dem Code der entsprechenden (durch HL) indirekt adressierten Befehle herleiten. Für IX-Befehle wird als erstes Byte ein &DD davorgesetzt, für IY-Befehle ein &FD. Dann kommt das erste Byte des HL-Codes, dann der Wert des Indexbytes, dann das eventuelle zweite Byte des HL-Codes.

Beispiel: SLR (HL) hat den Code &CB3E
SLR (IX+&02) hat den Code &DDCB023E

Datentransport „en gros“ und kurze Sprünge

14

In diesem Kapitel wollen wir zunächst die „Blockladebefehle“ kennenlernen. Das sind spezielle Datentransportbefehle. Im Gegensatz zu den LD-Befehlen können durch die Blockladebefehle aber nicht nur ein oder zwei Bytes transportiert werden, sondern ganze Speicherbereiche. Diese dürfen im Extremfall sogar eine Größe von 64 KBytes haben. Blockladebefehle sind in Wirklichkeit richtige Unterprogramme, die durch einen einzigen Maschinenbefehl aufgerufen werden.
Nach den Blockladebefehlen beschäftigen wir uns mit Sprungbefehlen, die eine relative Adressierung besitzen. Im Unterschied zu den JP-Befehlen, bei denen die Adresse des Sprungziels angegeben ist, wird bei den neuen Sprungbefehlen die Sprungweite angegeben. Der Vorteil dieser Methode besteht darin, daß bei der Verschiebung von Programmen in andere Speicherbereiche die Adressierung dieser Sprungbefehle nicht geändert werden muß.

Verschiebung von Speicherblöcken

Manchmal ist es wünschenswert, Daten oder Programme innerhalb des Speichers als ganze Blöcke verschieben zu können. Wenn z. B. dasselbe Unterprogramm in verschiedenen Programmen benutzt werden soll, dann ist es günstig, dieses im Anschluß an das jeweilige Hauptprogramm zu laden, um es dann mit diesem abspeichern zu können. So könnte man das Zeichenausgabeprogramm aus Kapitel 10 hinter dem Hauptprogramm P7 in den Bereich ab &401A laden und es dann für das Programm P9 im nächsten Kapitel in den Bereich ab &402E verschieben. Für diese Verschiebung ganzer Speicherblöcke hat die Z80-CPU äußerst wirksame Befehle: Zunächst schauen wir uns den Befehl LDDR (Code &EDB8) an, in dessen mnemonischer Bezeichnung die Begriffe Load, Decrement und Repeat (wiederholen) enthalten sind. Dieser Befehl lädt den Inhalt der durch (HL) indirekt adressierten Speicherstelle in die durch (DE) indirekt adressierte

Speicherstelle, dekrementiert DE, HL und BC und wiederholt das ganze, bis BC den Inhalt Null hat. Natürlich ist Ihnen jetzt alles klar; trotzdem zur Vorsicht noch ein Beispiel:

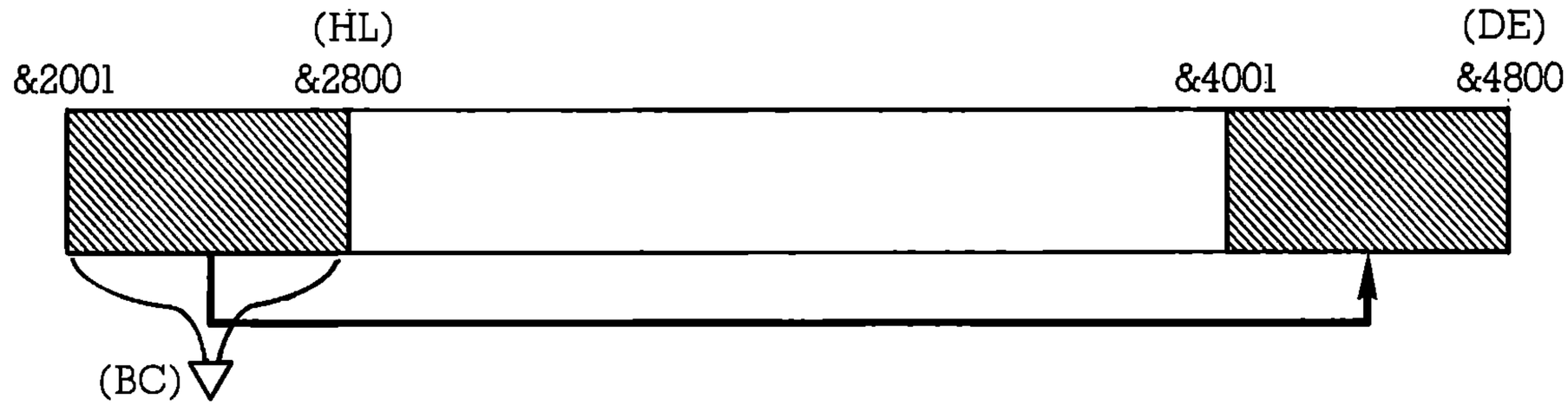


Bild 16 *Die Wirkung des Blockladebefehls LDDR.*

Wenn Sie HL mit &2800, DE mit &4800 und BC mit &800 laden und den Befehl LDDR geben

```
LD HL,&2800
LD DE,&4800
LD BC,&0800
LDDR
```

dann ist nach Ablauf des Befehls der Inhalt des Bereichs &2001 bis &2800 in den Bereich &4001 bis &4800 kopiert. Am besten kann man das im Bildschirmspeicher verfolgen.

Geben Sie dazu folgendes Programm ein:

```
4000                    1          ORG   #4000
4000  3E02               2          LD    A,#02
4002  CD0EBC             3          CALL  #BC0E          ;MODE 2 SETZEN.
4005  01D007             4          LD    BC,#07D0       ;ZAHL DER ZEICHEN.
4008  3EF0               5 L1:      LD    A,#F0          ;ASCII-CODE DES PFEILS.
400A  CD5ABB             6          CALL  #BB5A          ;PFEIL AUSGEBEN.
400D  0B                 7          DEC   BC             ;SCHLEIFENZAEHLER
                         8                               ;HERABZAEHLEN,
400E  78                 9          LD    A,B            ;UND AUF NULL
400F  B1                10          OR    C              ;PRUEFEN.
4010  C20840            11          JP    NZ,L1          ;NACH L1 WENN BC<>0.
4013  21FFDF            12          LD    HL,#DFFF       ;LETZTE ADRESSE DES
                        13                               ;KOPIERTEN BLOCKS.
4016  11FFFF            14          LD    DE,#FFFF       ;LETZTE ADRESSE DES
                        15                               ;ZIELBLOCKS.
4019  010020            16          LD    BC,#2000       ;ZAHL DER KOPIERTEN
                        17                               ;SPEICHERSTELLEN.
401C  EDB8              18          LDDR                 ;BLOCKTRANSFER.
401E  C9                19          RET                  ;FERTIG.
                        20 ;P11

L1       4008
```

Mit den ersten neun Befehlen füllen wir den Bildschirm mit nach oben zeigenden Pfeilen (Code &F0 (240)). Der Bequemlichkeit halber verwenden wir dazu die Systemroutine &BB5A. Ab Speicherstelle &4013 wird die untere Hälfte des Bildschirmspeichers (höchste Adresse &DFFF, Länge &2000) mit Hilfe des LDDR-Be-

fehls in die obere Hälfte des Bildschirmspeichers (höchste Adresse &FFFF) übertragen. Da der untere Teil des Speicherbereichs für die oberen vier Feinzeilen jeder Grobzeile verantwortlich ist, verschwinden jetzt die Pfeilschäfte, und der ganze Bildschirm ist mit Pfeilspitzen gespickt.
Eine noch interessantere Anwendung des LDDR-Befehls besteht im horizontalen Scrollen: Hierbei wird der gesamte Bildschirm um eine Zeichenbreite nach rechts oder links verschoben. Das folgende kurze Programm löst diese Aufgabe bis auf kleine Schönheitsfehler für die Rechtsverschiebung:

```
4000                    1           ORG   #4000
4000  01FF3F            2           LD    BC,#3FFF          ;BLOCKLAENGE.
4003  21FEFF            3           LD    HL,#FFFE          ;ENDE ORIGINALBLOCK.
4006  11FFFF            4           LD    DE,#FFFF          ;ENDE KOPIERTER BLOCK.
4009  EDB8              5           LDDR                    ;KOPIERBEFEHL.
400B  C9                6           RET                     ;FERTIG.
                        7 ;P12
```

Sie sehen, daß Anfangs- und Endbereich fast identisch sind. Die vorletzte Speicherstelle wird in die letzte übertragen. Unmittelbar darauf wird der ursprüngliche Inhalt der vorletzten Speicherstelle von dem der vorvorletzten überschrieben, und so weiter. Nur der Inhalt der allerersten Speicherstelle &C000 bleibt erhalten, was man bei mehrfachem Anwenden des Programms deutlich sieht. Man könnte natürlich die Speicherstelle &BFFF mit dem Wert 0 laden und noch mitverschieben. Die links herausfallenden Zeichen schieben sich von rechts eine Zeile tiefer wieder in den Bildschirm, und bei wiederholtem Programmaufruf windet sich der ganze Bildschirminhalt sozusagen schraubenförmig nach unten aus dem Bildschirm hinaus ins Nichts.
In der Abkürzung LDDR steht wie bereits erwähnt, neben „Load" (LD) für den Vorgang des Datentransports noch D für „Decrement" (also vermindern) und R für „Repeat" (also wiederholen). Sie können nun schon beinahe erraten, daß es daneben noch einen LDIR-Befehl (Code &EDB0) gibt. Bei diesem werden Absenderadresse (HL) und Empfängeradresse (DE) hochgezählt (inkrementiert). Der Schleifenzähler BC wird allerdings auch hier bis zum Wert 0 herabgezählt. Beide Befehle sind gleichwertig, solange man einen Speicherblock in einen anderen kopiert, der mit diesem keine gemeinsamen Adressen besitzt.
Bei einer Überschneidung des neuen und des alten Blockbereichs benötigt man den LDDR-Befehl, falls die Anfangsadresse des Empfängerblocks größer als die des Absenderblocks ist. Andernfalls benötigt man den LDIR-Befehl.

Zusammenfassung

Die Blocktransportbefehle LDDR und LDIR kopieren bzw. verschieben Speicherblöcke mit einer vorher im BC-Register angegebenen Länge. Im HL-Register muß die Endadresse (Anfangsadresse bei LDIR) des Originalblocks stehen, im DE-Register muß die Endadresse (Anfangsadresse bei LDIR) des kopierten Blocks stehen.

Relative Sprünge

Wir wollen jetzt wieder auf den ursprünglichen Zweck des Verschiebens zurückkommen, der darin besteht, ein Programm aus einem Speicherplatzbereich in einen anderen zu bringen. Bei fast allen von uns bisher geschriebenen Programmen würde sich in diesem Fall das Problem ergeben, daß die Adressen in den Sprungbefehlen JP auf die alten Speicherstellen zeigen würden. Das Programm wäre dann erst nach dem Umrechnen der Sprungadressen auf den neuen Speicherbereich lauffähig. Damit der Programmierer solchen Problemen von vornherein aus dem Weg gehen kann, bietet die Z80-CPU neben den absoluten Sprungbefehlen JP noch relative Sprünge an, die mit der mnemonischen Abkürzung JR („jump relative") bezeichnet werden. Bei den relativen Sprüngen gibt man nicht an, wohin das Programm springen soll, sondern um wie viele Speicherstellen das Programm vor- oder zurückspringen soll. Intern läuft das so ab: Beim JP-Befehl wird der Programmzähler mit der bei JP angegebenen Adresse geladen. Dagegen wird beim JR-Befehl die bei JR angegebene Zahl zum Inhalt des Programmzählers hinzuaddiert bzw. von ihm subtrahiert.
Die Sprungweiten sind allerdings eingeschränkt: Maximal 127 Speicherstellen vorwärts und 128 rückwärts können im JR-Befehl angegeben werden. Ausgangspunkt ist dabei immer die Adresse des Befehls, der unmittelbar hinter dem Sprungbefehl steht. (Wir erinnern uns, daß bei der Ausführung eines Befehls der Programmzähler PC auf der ersten Adresse des nächsten Befehls steht.)
Den relativen Sprung gibt es in fünf Varianten: Einen Sprung ohne Bedingung und vier Sprünge mit den Bedingungen Z, NZ, C, NC (die Codes sind hinter den Befehlen in Klammern angegeben):

```
JR e       (&18n)
JR Z,e     (&28n)
JR NZ, e   (&20n)
JR C,e     (&38n)
JR NC,e    (&30n)
```

Dabei sollte die hexadezimal ausgedrückte Sprungweite in Bytes sein. Im Maschinencode bedeutet n bei Sprüngen nach vorn (e›0) einfach die Sprungweite e (n=e); bei Rückwärtssprüngen (e‹0) erhält man n, indem man die Rücksprungweite (also den Betrag von e) von der Zahl 256 subtrahiert (n=&100+e).
Wenn Sie inzwischen mit dem Direktassembler aus dem Anhang oder einem anderen Assembler arbeiten, brauchen Sie für die JR-Befehle die Sprungweiten nicht zu berechnen. Aus der Angabe der Sprungadresse, die wie bei den JP-Befehlen erfolgt, berechnet der Assembler für Sie die notwendigen Werte. Wenn Sie aber weiterhin die Assemblierung von Hand durchführen, können Sie das Vorgehen anhand der folgenden Beispiele lernen, das insbesondere für kurze Sprünge gut verwendbar ist.

Das Beispiel in Tabelle 3 für einen Rückwärtssprung ist ein kurzer Programmabschnitt, der acht Nullen auf den Bildschirm ausgibt.

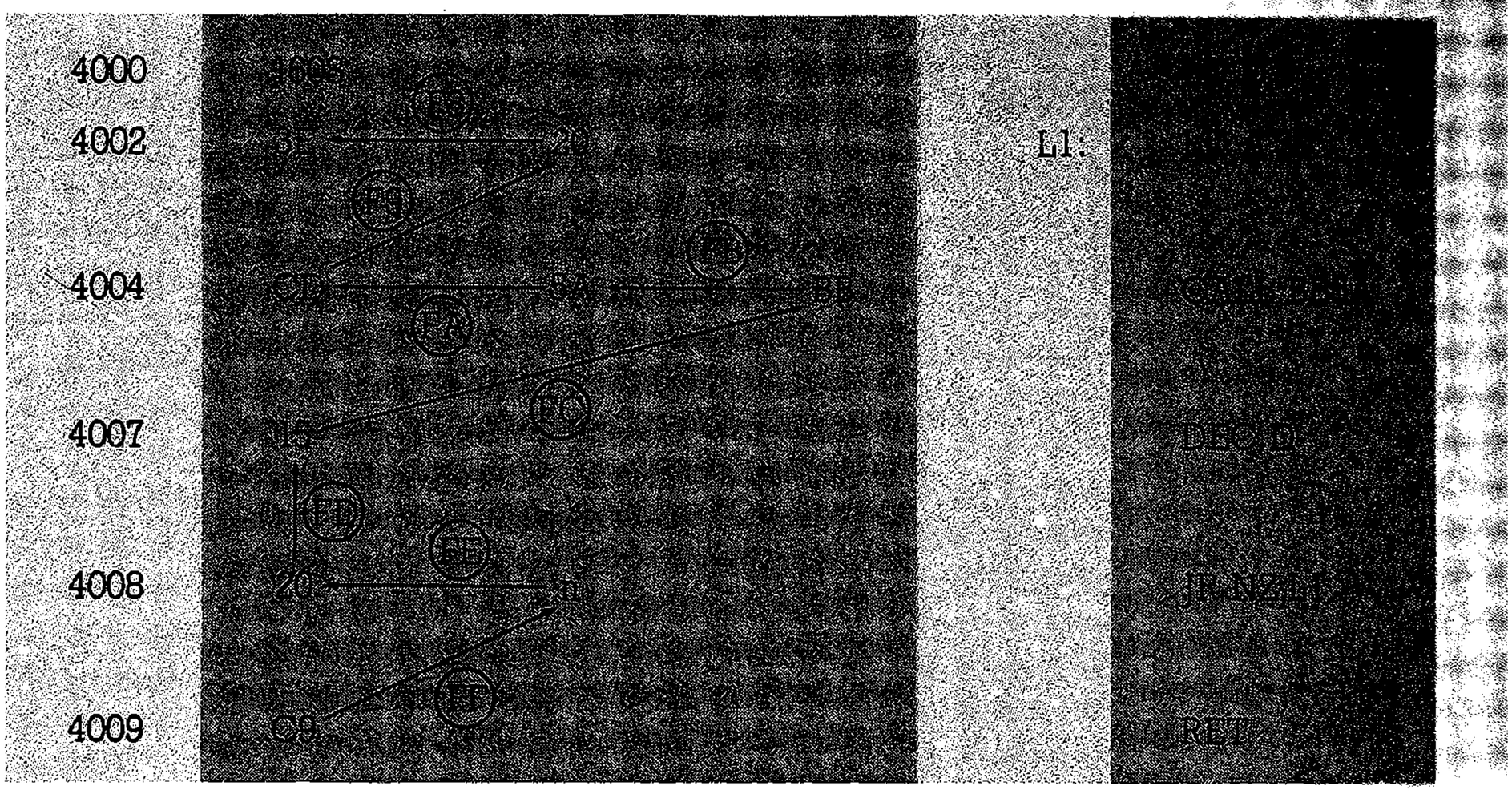


Tabelle 3 *Relative Sprungweite für einen Rückwärtssprung.*

Um den Wert n des zweiten Bytes im Operationscode von JR zu erhalten, zählt man vom ersten Byte des Befehls nach dem Sprungbefehl (&C9) bis zum ersten Byte des anzusprechenden Befehls (&3E) von 255 (&FF) aus rückwärts. Die dadurch ermittelte Zahl muß dann für n eingesetzt werden. In unserem Beispiel gilt also n=&F8. Dieselbe Zahl ergibt sich auch, wenn wir die Zahl der rückwärtszuspringenden Bytes (nämlich 8) von 256 subtrahieren: 256-8=248=&F8.
Anschließend in Tabelle 4 ein kurzes Programmbeispiel für einen Vorwärtssprung: Beim Druck auf eine beliebige Taste soll eine 0 ausgegeben werden; nur wenn die Q-Taste (Code &51) gedrückt wird, soll die Ausgabe unterbleiben.

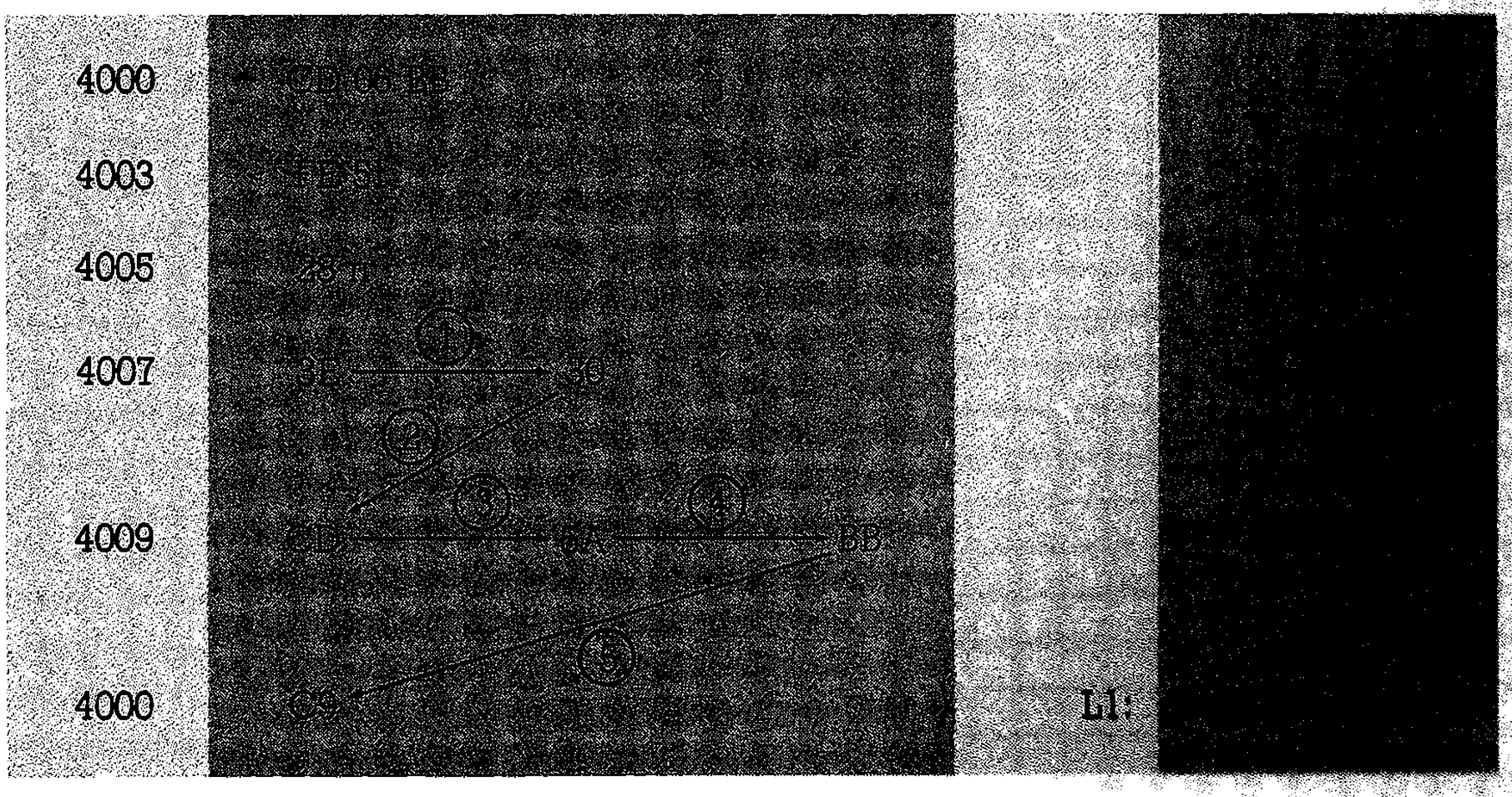


Tabelle 4 *Relative Sprungweite für einen Vorwärtssprung.*

Vom ersten Byte des Befehls nach dem Sprungbefehl (&3E) bis zum ersten Byte des anzuspringenden Befehls (&C9) sind es fünf Bytes. Für n muß also &05 eingesetzt werden.

Zusammenfassung

Bei relativen Sprüngen JR wird keine Adresse, sondern eine Sprungweite angegeben. Diese kann 127 Adressen in Vorwärtsrichtung und 128 Adressen in Rückwärtsrichtung betragen. Die Berechnung dieser Sprungweite übernimmt normalerweise ein Assembler.

Übung 21 Schreiben Sie ein Programm, das den Bildschirm nach links „scrollt".

Übung 22 Geben Sie die Werte der Sprungweiten e für folgende Fälle an:
a) &4005 JR Z,L1 mit L1 = &400F
b) &600A JR L1 mit L1 = &6000
c) &6000 JR L1 mit L1 = &6100

Zusatzinformationen

Zusätzlich zu den genannten relativen Sprungbefehlen gibt es noch den Befehl

```
DJNZ e
```

der die beiden Befehle

```
DEC B
JR NZ,e
```

ersetzt. DJNZ ist die mnemonische Abkürzung von „Decrement and Jump if Not Zero".
Neben den Befehlen LDDR und LDIR gibt es noch die Befehle LDD und LDI. Diese Befehle haben eine ähnliche Wirkung wie LDDR und LDIR, nur wird der Ladevorgang nicht wiederholt (daher fehlt auch das R(Repeat) beim mnemonischen Ausdruck). Auch hier wird doppelt indirekt adressiert, und LDD lädt die durch DE adressierte Speicherstelle mit dem Inhalt der durch HL adressierten Speicherstelle und dekrementiert dann DE, HL und BC, hört dann im Gegensatz zu LDDR aber auf. Entsprechendes gilt für LDI.
Zusätzlich zu den Blockkopierbefehlen gibt es noch die „Blockvergleichsbefehle":

```
CPD   (Compare and Decrement)
CPDR  (Compare, Decrement and Repeat)
CPI   (Compare and Increment)
CPIR  (Compare, Increment and Repeat)
```

Die Wirkung kann man fast erraten: So vergleicht CPDR den Inhalt des durch

(HL) indirekt adressierten Speicherplatzes mit dem Akkumulatorinhalt und dekrementiert dann HL und BC. Die Ausführung des Befehls wird beendet, wenn entweder der Inhalt von BC gleich Null ist oder wenn einer der angesprochenen Speicherstellen denselben Wert wie der Akkumulator enthält. Diese Vergleichsbefehle beeinflussen die Z- und nicht die C-Flagge. Bei allen Blockbefehlen (Lade- und Vergleichsbefehlen) wird der Stand des Zählregisters BC durch die bisher noch nicht besprochene P/V-Flagge (siehe Kapitel 19) angezeigt. Diese wird zurückgesetzt, wenn der Inhalt von BC gleich Null ist. Bei den CPDR- und CPIR-Befehlen kann man so durch Abfragen dieser Flagge herausfinden, ob die Suche aufgrund des Zählerstands abgebrochen wurde oder weil das gesuchte Byte gefunden wurde.

15 Masken für die Bytes

Wir haben in Kapitel 12 gesehen, wie man mit Hilfe von Schiebe- und Rotationsbefehlen den Inhalt von Speicherstellen und Registern bitweise abfragen kann. Wenn es aber darum geht, gezielt ein ganz bestimmtes Bit einzeln zu prüfen oder zu verändern, wäre die Anwendung von Rotations- oder Schiebebefehlen sehr umständlich. Zum Ansprechen einzelner Bits oder einer Gruppe von Bits innerhalb eines Bytes stellt die Z80-CPU eine Reihe von Befehlen zur Verfügung, von denen wir jetzt die sogenannten logischen Verknüpfungen AND, OR und XOR kennenlernen wollen.

AND - OR - XOR

Die AND(Und)-Verknüpfung zweier Bits wirkt so wie das Zusammenspiel der beiden Schalter am Monitor und am Hauptgerät Ihres CPC. Der Computer ist nur dann betriebsbereit, wenn beide Schalter an sind. Entsprechend ist das Ergebnis der AND-Verknüpfung zweier Bits nur dann eine 1, wenn beide Ausgangsbits eine 1 enthalten. In allen anderen Fällen hat das Ergebnisbit den Wert 0.
Die OR(Oder)-Verknüpfung ist in gewisser Hinsicht das Gegenstück zur AND-Verknüpfung: Das Ergebnisbit hat nur dann den Wert 0, wenn beide Ausgangsbits den Wert 0 haben. In allen anderen Fällen ist das Ergebnis eine 1.
Die XOR(Exclusive Oder)-Verknüpfung entspricht dem „entweder-oder" der Umgangssprache. Ein Kind, welches das Angebot erhält, entweder eine Mark oder eine Tafel Schokolade zu bekommen, würde nach streng logischen Regeln leer ausgehen, wenn es beides fordert oder beides ablehnt. (Natürlich können hier pädagogische Gesichtspunkte Vorrang vor logischen Regeln gewinnen.)
Die XOR-Verknüpfung zweier Bits ergibt immer dann eine 0, wenn beide Ausgangsbits denselben Inhalt besitzen.
Tabelle 5 soll die Wirkung der logischen Verknüpfungen noch einmal zusammenfassen.

1 AND 1 = 1	1 OR 1 = 1	1 XOR 1 = 0
1 AND 0 = 0	1 OR 0 = 1	1 XOR 0 = 1
0 AND 1 = 0	0 OR 1 = 1	0 XOR 1 = 1
0 AND 0 = 0	0 OR 0 = 0	0 XOR 0 = 0

Tabelle 5 *Logische Verknüpfung einzelner Bits.*

Die logischen Befehle der Z80-CPU verknüpfen nicht nur zwei Bits, sondern immer ganze Bytes miteinander. Das geht so vor sich, daß jeweils die beiden Bits mit derselben Nummer verknüpft werden. Eines der Ausgangsbytes steht dabei im Akkumulator, das zweite Byte ist im Operanden des Befehls angegeben. Der Operand kann ein Zahlenwert zwischen 0 und 255, ein 8-Bit-Register oder eine durch (HL), (IX+d), (IY+d) adressierte Speicherstelle sein. Das Ergebnisbyte steht nach der Operation im Akkumulator. Als Beispiel sollen in Tabelle 6 die Verknüpfungen der beiden Zahlen &E3 und &B9 gezeigt werden:

	AND	OR	XOR
&E3	11100011	11100011	11100011
&B9	10111001	10111001	10111001
	10100001 (&A1)	11111011 (&FB)	01011010 (&5A)

Tabelle 6 *Logische Verknüpfung von Bytes.*

Als Resultat von Maschinenbefehlen könnten diese Werte auftreten, wenn im Akkumulator zunächst &E3 steht, während das B-Register &B9 enthält. Nach der Ausführung von

AND B

steht im Akkumulator das Ergebnis &A1. Nach

OR B bzw. XOR B

steht im Akkumulator

&FB bzw. &5A

Programmbeispiel: Unterdrückung der Kleinschreibung

Eine wichtige Anwendung der logischen Befehle zeigt das nächste Programm:

```
4000                    1          ORG   #4000
4000  CD06BB            2 START:   CALL  #BB06      ;TASTATURABFRAGE.
4003  FE51              3          CP    #51       ;"Q" GEDRUECKT ?
4005  C8                4          RET   Z         ;WENN JA,FERTIG.
4006  E6DF              5          AND   #DF       ;SONST MASKIERE CODE,
4008  CD5ABB            6          CALL  #BB5A     ;UND GIB AUS.
400B  18F3              7          JR    START     ;ZURUECK ZUM ANFANG.
                        8 ;P13

START  4000
```

Im Programm wird zunächst die Tastatur abgefragt und, falls ein Q gedrückt wurde, zum Betriebssystem zurückgesprungen. Falls die Q-Taste nicht gedrückt wurde, wird der ASCII-Code des eingegebenen Zeichens mit der Zahl &DF=X11011111 durch ein logisches AND verknüpft. Dadurch wird das fünfte Bit des ASCII-Codes auf Null gesetzt, der Rest des Bitmusters bleibt aber erhalten. Im fünften Bit steckt aber der Unterschied zwischen Groß- und Kleinbuchstaben. Bei Großbuchstaben hat das fünfte Bit den Wert 0, bei Kleinbuchstaben den Wert 1. (Siehe die Tabelle im CPC-Bedienungshandbuch.) Nach der AND-Verknüpfung gibt die uns bereits bekannte Betriebssystems-Routine &BB5A den Buchstaben immer groß aus, unabhängig davon, wie er eingegeben wurde. (Vorsicht, nur Buchstaben eingeben!) Dann verzweigt das Programm zur Eingabe zurück.

Bytes werden maskiert

Die Verarbeitung von Bytes, die wir im letzten Programmbeispiel angewandt haben, nennt man häufig „Maskieren". Um diese Bezeichnung zu verstehen, schauen wir uns nochmals die Tabelle 5 an. Wir sehen, daß der ursprüngliche Wert eines Bits bei einer AND-Verknüpfung mit 1 unverändert ins Ergebnis übernommen wird. Wird ein Bit dagegen mit dem Wert 0 durch AND verknüpft, so hat das Resultat den Wert 0. Im Programm P13 war unsere „Maske" das Bitmuster 11011111. Diese „Maske" zeigt für alle Bits des mit ihr verknüpften Bytes den ursprünglichen Wert. Nur das fünfte Bit ist durch eine Null „verdeckt". Man kann das verallgemeinern: Bei einer AND-Verknüpfung eines Bytes mit einer aus Nullen und Einsen bestehenden Maske enthält das Resultat die Bitwerte des Ausgangsbytes an den Stellen, an denen die Maske eine 1 hat. An den Stellen, an denen die Maske eine 0 hat, haben auch die Bits des Resultats den Wert Null.

Anders ist das bei einer OR-Verknüpfung. Für diesen Fall zeigt unsere Tabelle, daß der ursprüngliche Wert eines Bits bei einer Verknüpfung mit 0 unverändert ins Resultat übernommen wird. Wird ein Bit dagegen mit dem Wert 1 durch OR verknüpft, so hat auch das Resultat den Wert 1. Daraus ergibt sich, daß eine OR-Verknüpfung dazu verwendet werden kann, einzelne Bits eines Bytes zu setzen. Wenn wir das letzte Programm so abändern wollen, daß bei der Ausgabe immer Kleinbuchstaben erscheinen, müssen wir den Befehl AND &DF (&DF=X11011111) durch OR &20 (&20=X00100000) ersetzen. Diese Maskierung setzt das fünfte Bit des ASCII-Codes und läßt alle anderen Bits unverändert. Beim Drücken einer Buchstabentaste wird daher jeweils der Code eines Kleinbuchstabens erzeugt. Durch kleine Änderungen an unserem Zeichensetzprogramm P6 können die Wirkungen der verschiedenen Maskierungen sehr anschaulich demonstriert werden. Fügen Sie zwischen die Befehle

```
LD A,(IX+&00)
```

und

```
LD (HL),A
```

zunächst den Befehl

```
AND &0F
```

ein.
&0F entspricht der Dualzahl X00001111, und eine AND-Verknüpfung mit dieser Maske löscht die erste Hälfte jedes zum Bitmuster des auszugebenden Zeichens gehörenden Bytes. Auf dem Bildschirm wird daher nur die rechte Hälfte des Zeichens ausgegeben.
Ersetzt man den Befehl AND &0F durch OR &0F, so wird die zweite Hälfte jedes Bytes des Bitmusters mit den Bitwerten 1 aufgefüllt. Auf dem Bildschirm ist daher die rechte Hälfte der Zeichenposition völlig ausgefüllt.
Setzen Sie als nächstes statt des Befehls OR &0F den Befehl XOR &0F ein. Die XOR-Verknüpfung mit dem Wert 0 ändert den Wert der Bits des Bitmusters nicht. Dagegen werden durch die XOR-Verknüpfung mit dem Wert 1 die Bits gerade umgedreht. Aus dem Bitwert 1 wird der Bitwert 0 und umgekehrt. Die XOR-Verknüpfung mit der Maske X00001111 läßt also die linke Hälfte des Zeichens unverändert, während die rechte Hälfte invertiert wird: Das Zeichen nimmt die Hintergrundfarbe an und der Hintergrund die Zeichenfarbe. Eine völlige Invertierung des Zeichens ergibt sich schließlich mit der Verknüpfung XOR &FF.
Weitere Anwendungen der logischen Verknüpfungen, insbesondere im Zusammenhang mit Rechenoperationen, lernen wir in den folgenden Kapiteln kennen.

Zusammenfassung

Die Befehle AND, OR, XOR führen logische Verknüpfungen zwischen dem Inhalt des Akkumulators und dem im Operanden des Befehls angegebenen Byte durch. Im Operanden kann eine 8-Bit-Zahl (unmittelbare Adressierung), jedes 8-Bit-Allzweckregister (A, B, C, D, E, H, L) und eine durch (HL), (IX+d) sowie (IY+d) adressierte Speicherstelle angegeben werden: Beispiele

```
AND &3F
OR C
XOR (HL)
```

Übung 23 **Setzen Sie im Programm P13 die Maske so, daß beim Drücken der Tasten A, B,..., I die Ziffern 1, 2,..., 9 ausgegeben werden. (Verwenden Sie die ASCII-Tabelle des CPC-Bedienungshandbuchs.)**

Zusatzinformationen

Die AND-, OR- und XOR-Befehle löschen die Carry-Flagge und beeinflussen die Z-Flagge entsprechend dem Resultat der Verknüpfung. Die eigentlich sinnlosen Befehle

```
AND A   und   OR A
```

die den Akkumulatorinhalt mit sich selbst verknüpfen und nicht verändern, können daher benutzt werden, um die Carry-Flagge zu löschen, da ein eigener Befehl dazu nicht existiert. Der Befehl

```
XOR A
```

löscht den Akkumulatorinhalt genau wie

```
LD A,&00
```

ist aber nur ein Byte lang. Trotzdem ist die Wirkung dieser Befehle nicht genau gleich: Während LD A,&00 die Carry-Flagge nicht beeinflußt, wird sie durch XOR A gelöscht.

Zugriff auf einzelne Bits. Eine umfangreiche Gruppe von Befehlen zur Beeinflussung bzw. Kontrolle von Einzelbits wird vom Z80-Prozessor mit den BIT-, RES- und SET-Befehlen bereitgestellt. Diese Befehle können Einzelbits abfragen, löschen und setzen. Ihre Wirkung kann man am besten anhand von Beispielen erklären:

```
SET 3,A
```

setzt das dritte Bit im Akkumulator auf den Wert 1.

```
RES 3,A
```

setzt dasselbe Bit auf den Wert 0 („reset" = zurücksetzen).

BIT 3,A

setzt die Z-Flagge, wenn das dritte Bit des Akkumulators den Wert 0 besitzt. Besitzt dieses Bit den Wert 1, dann wird die Z-Flagge gelöscht. Statt des Bits mit der Nummer 3 können natürlich alle anderen Bits mit den Nummern von 0 bis 7 gesetzt, zurückgesetzt und getestet werden. Statt des Akkumulators kann jedes andere 8-Bit-Allzweckregister B,...,L und jede durch (HL), (IX+d) und (IY+d) adressierte Speicherstelle angesprochen werden. So setzt

RES 0,D

das nullte Bit des D-Registers zurück, und so weiter. Im dem Großbuchstabenprogramm dieses Kapitels kann der Befehl

AND &DF

der das fünfte Bit mit einer Null maskiert, auch durch

RES 5,A

ersetzt werden. Genauso wie SET beeinflußt RES im Gegensatz zu AND keine Flaggen, was in diesem Programm aber keine Rolle spielt. Eine Übersicht über die BIT-, RES- und SET-Befehle finden Sie in der Befehlstabelle im Anhang D.

Wir setzen einen Punkt 16

Der wichtigste Bestandteil jedes Graphikprogramms ist ein Unterprogramm, das an einer ausgewählten Stelle des Bildschirms einen Punkt setzen kann. Ein derartiges Programm wird häufig als „Plot-Programm" bezeichnet. Die Ortsangabe für den auszugebenden Punkt erfolgt mit Hilfe von Zeilen- und Spaltennummern, die aber nicht mit den Zeilen- und Spaltenangaben der Zeichenpositionen verwechselt werden dürfen, sondern eine viel feinere Einteilung des Bildschirms zulassen. Beim CPC ist der Bildschirm in 200 Zeilen mit je 640 Spalten unterteilt. Die wesentliche Aufgabe des Plotprogramms besteht darin, aus der Spaltennummer X und der Zeilennummer Y diejenige Adresse A im Bildschirmspeicher zu berechnen, die für den auszugebenden Punkt zuständig ist. Wir werden sehen, daß die Berechnungsformel für diese Adresse A lautet:

A = &C000+&50*INT(Y/8)+INT(X/8)+&800*MOD(Y/8)

Dabei bezeichnet INT den ganzzahligen Anteil und MOD den Rest der in Klammern angegebenen Division.
Da der Z80-Prozessor keine Befehle zum Multiplizieren und Dividieren beliebiger 8-Bit-Zahlen besitzt, müssen wir die in der Formel auftretenden Rechnungen anders bewältigen. Die in den letzten Kapiteln besprochenen Rotations- und Schiebebefehle sowie die logischen Befehle spielen hier eine entscheidende Rolle.

Die Bildschirmorganisation

Der Bildschirm des CPC ist horizontal in 640 Punkte und vertikal in 200 Punkte unterteilt. Das ergibt sich sofort aus den 25 Grobzeilen zu je 8 Feinzeilen (25x8 = 200) und den 80 Zeichen, die 8 Punkte breit sind (80x8 = 640). Um eine Punktposition ansprechen zu können, geben wir die jeweilige horizontale Position (X-Koordinate) und die dazugehörende vertikale Position (Y-Koordinate) an. Die Koordi-

naten der linken oberen Ecke sind beide Null; die X-Koordinate kann demnach Werte zwischen 0 und 639, die Y-Koordinate Werte zwischen 0 und 199 annehmen.

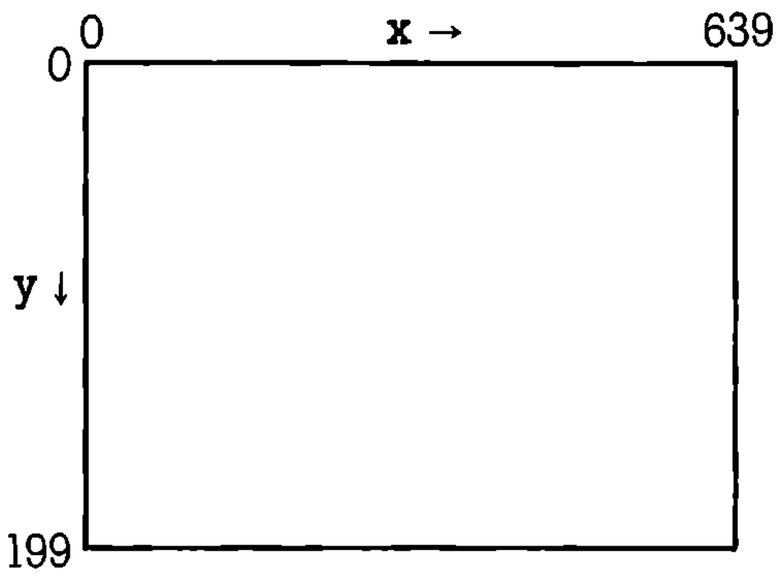


Bild 17 *Die Feineinteilung des Bildschirms.*

Wenn wir einen Punkt mit gegebener X- und Y-Koordinate (X,Y) setzen wollen, dann müssen wir zuerst die Adresse im Bildschirmspeicher ermitteln, die für diesen Punkt zuständig ist. Da aber jede Bildschirmspeicherstelle gleichzeitig 8 Punkte „steuert", muß dann noch innerhalb dieses Bytes das für den zu setzenden Punkt zuständige Bit bestimmt werden. Dazu gehen wir folgendermaßen vor: Zunächst bestimmen wir die Nummer der Grobzeile, in der unser Punkt liegt, wobei die Numerierung der Grobzeilen mit Null beginnen soll. Da nach je 8 Y-Positionen eine neue Grobzeile beginnt, muß zu diesem Zweck Y durch 8 dividiert und vom Ergebnis der ganzzahlige Anteil genommen werden. In BASIC würde man das mit INT(Y/8) berechnen. Als Beispiel betrachten wir einen Punkt mit Y=70 und X=90. Die Grobzeilennummer ist INT (70/8)=8.
Als nächstes bestimmen wir die Nummer der Zeichenposition. In Gedanken laufen wir dazu von der linken oberen Ecke des Bildschirms die Grobzeilen entlang bis zu unserem Punkt. Jede Grobzeile enthält 80 Zeichenpositionen. Bis zum Beginn der Zeile, in der sich unser Punkt befindet, haben wir 80*INT(Y/8) Positionen durchlaufen. Die nullte Zeile ist dabei natürlich mit berücksichtigt worden. Da jede Zeichenposition 8 Punkte breit ist, kommen in der letzten Zeile noch INT (X/8) Zeichenpositionen zu.
In unserem Beispiel liegt der Punkt in der Zeile mit der Nummer 8. Einschließlich der nullten Zeile liegen also acht komplette Zeilen davor, und das entspricht 640 Zeichenpositionen. Dazu kommen in der achten Zeile nochmals INT(90/8) = 11 Positionen, so daß die Nummer der gesuchten Zeichenposition 651 ist.
Jetzt bestimmen wir innerhalb der gefundenen Zeichenposition die Nummer der Feinzeile, auf der unser Punkt liegt. Da in Y-Richtung nach je acht Feinzeilen eine neue Zeichenposition anfängt, ist die Nummer der Feinzeile innerhalb eines Zeichens der Rest durch 8 bei der Division von Y. In unserem Beispiel ergibt sich: Rest von (70/8) = 6. Und schließlich ergibt der Achterrest der X-Koordinate das zu setzende Bit, wobei entsprechend dem Aufbau des Bildschirmspeichers das linkssitzende Bit gerade das höchstwertige ist. In unserem Fall muß gerade das fünfte Bit gesetzt werden. Die zu setzenden Bitnummer ist also gerade 7 weniger dem Rest von (X/8) (s. Bild 18).

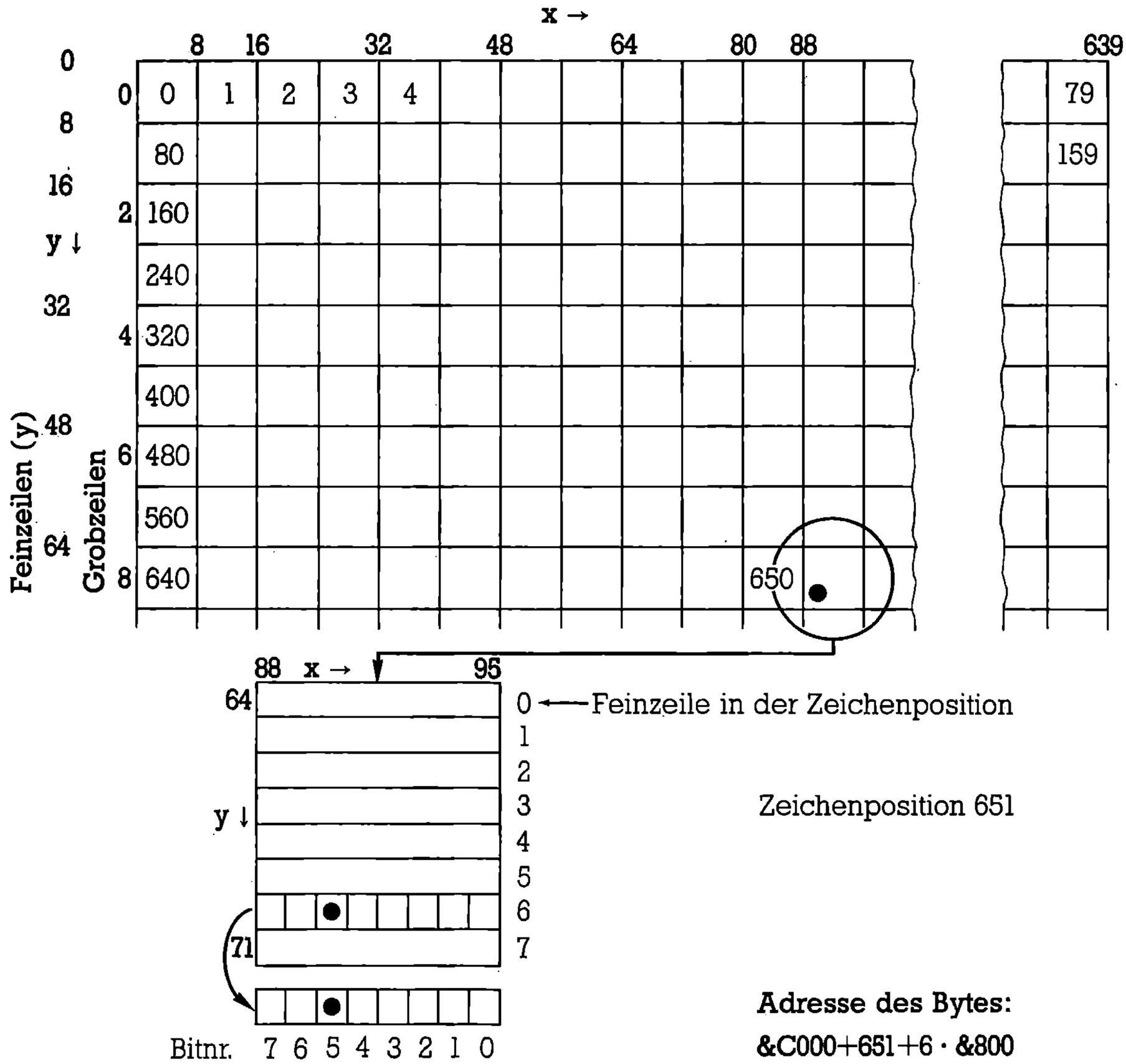


Bild 18 *Bildschirmausschnitt mit Grob- und Feineinteilung. Die Zeichenpositionen sind zur Vereinfachung quadratisch gezeichnet, obwohl das in Mode 2 nicht der Fall ist. Der Punkt mit der x-Koordinate 90 und der y-Koordinate 70 befindet sich in der Feinzeile 6 der Zeichenposition mit der Nummer 651. Bit 5 der Bildschirmspeicheradresse &C000+651+6*&800 muß gesetzt werden.*

Um einen Punkt mit den Koordinaten X und Y zu setzen, muß also folgende Vorschrift programmiert werden:

1. **Man bestimmt die Zeichenposition, in der sich der Punkt befindet: 80*INT(Y/8)+INT(X/8), addiert den Bildschirmspeicheranfang &C000 und hat damit die Adresse der obersten Feinzeile der Zeichenposition.**
2. **Man bestimmt die Nummer der Feinzeile innerhalb der Zeichenposition durch die Berechnung des Rests von (Y/8). Da der Feinzeilenabstand jeweils &800 beträgt, erhält man für die Adresse der gesuchten Speicherstelle:**

   &C000+80*INT(Y/8)+INT(X/8)+&800*MOD(Y/8).

3. **In dieser Speicherstelle muß dann das Bit mit der Nummer 7-MOD(X/8) gesetzt werden.**

Das Plot-Programm

X- und Y-Koordinaten müssen dem Programm übergeben werden. Da die X-Koordinate Werte bis 639 annehmen kann, brauchen wir dazu zwei Bytes; für die Y-Koordinate reicht dagegen ein Byte. Im folgenden Programm wählen wir zu diesem Zweck die Speicherstellen &100 (für das L-Byte von X), &101 (für das H-Byte von X) und &102 (für Y).

```
4000                    1           ORG   #4000
4000  3A0201            2           LD    A,(#0102)          ;Y-KOORDINATE NACH A.
4003  E6F8              3           AND   #F8                ;BILDE 8*INT(Y/8).
4005  6F                4           LD    L,A                ;DIESER WERT
4006  2600              5           LD    H,0                ;NACH HL
4008  4F                6           LD    C,A                ;UND
4009  0600              7           LD    B,0                ;NACH BC.
400B  29                8           ADD   HL,HL              ;DIESE 4 BEFEHLE
400C  29                9           ADD   HL,HL              ;MULTIPLIZIEREN
400D  09               10           ADD   HL,BC              ;DEN INHALT VON HL
400E  29               11           ADD   HL,HL              ;MIT 10.
400F  3A0201           12           LD    A,(#0102)          ;Y-KOORDINATE NACH A.
4012  E607             13           AND   #07                ;REST VON Y/8.
4014  CB27             14           SLA   A                  ;DIESE 3 BEFEHLE
4016  CB27             15           SLA   A                  ;MULTIPLIZIEREN
4018  CB27             16           SLA   A                  ;A MIT 8.
401A  84                17           ADD   A,H                ;A WIRD ZUM HIGH-BYTE
401B  67                18           LD    H,A                ;ADDIERT UND DADURCH MIT
                        19                                    ;#100 MULTIPLIZIERT.
401C  ED4B0001          20           LD    BC,(#0100)         ;X-KOORDINATE NACH BC.
4020  CB38             21           SRL   B                  ;DIESE 6 BEFEHLE
4022  CB19             22           RR    C                  ;DIVIDIEREN
4024  CB38             23           SRL   B                  ;DEN INHALT
4026  CB19             24           RR    C                  ;VON BC
4028  CB38             25           SRL   B                  ;OHNE REST
402A  CB19             26           RR    C                  ;DURCH 8.
402C  09               27           ADD   HL,BC              ;BILDET INT(X/8)+HL.
402D  0100C0           28           LD    BC,#C000           ;BILDSCHIRMSPEICHERANF.
4030  09               29           ADD   HL,BC              ;WIRD DAZUADDIERT.
4031  3A0001           30           LD    A,(#0100)          ;X-LOW NACH A.
4034  E607             31           AND   #07                ;REST VON X/8.
4036  DD214340         32           LD    IX,ZP              ;ZEIGER AUF TABELLE.
403A  323F40           33           LD    (XR+2),A           ;INDEXBYTE D =
                       34 ;                                  ;REST VON X/8.
403D  DD7E00           35 XR:       LD    A,(IX+#00)         ;BIT WIRD GESETZT,
4040  B6               36           OR    (HL)               ;UND AN DIE BERECHNETE
4041  77               37           LD    (HL),A             ;SPEICHERSTELLE GEBRACHT
4042  C9               38           RET                      ;FERTIG.
4043  80402010         39 ZP:       DEFB #80,#40,#20,#10     ;TABELLE
4047  08040201         40           DEFB #08,#04,#02,#01     ;DER ZWEIERPOTENZEN.
                       41 ;P14

XR      403D   ZP       4043
```

Rechnen mit Tricks

Wir wollen jetzt versuchen, die im Programm durchgeführten Rechenoperationen zu verstehen. Wie schon oben erwähnt wurde, besitzt der Z80-Prozessor weder Multiplikations- noch Divisionsbefehle. Wir sind daher gezwungen, Ersatzlösungen mit Hilfe der vorhandenen Befehle zu suchen. Entsprechende Methoden für das Zehnersystem lernt jedes Kind bereits in der Grundschule kennen. Dazu machen wir uns klar, daß für Dualzahlen die Multiplikation mit 2 (bzw. die Division durch 2) dieselben Auswirkungen hat wie für eine Zahl im Zehnersystem die Multiplikation mit 10 (bzw. die Division durch 10): Bei der Multiplikation mit 10 wird eine Null an die Zahl angehängt. Das kann man auch dadurch ausdrükken, daß jede Ziffer der Zahl um eine Stelle nach links verschoben wird und die rechts freiwerdende Stelle mit einer Null aufgefüllt wird. Bei den 8-Bit-Dualzahlen wird diese Operation gerade durch den SLA-Befehl ausgeführt.
Eine Division durch 10 entspricht bei Dezimalzahlen einer Ziffernverschiebung um eine Stelle nach rechts. Entsprechend wird die Division von Dualzahlen durch 2 mit dem SRL-Befehl bewirkt. Allerdings geht dabei die rechts herausgeschobene Ziffer verloren, wodurch nur der ganzzahlige Anteil des Ergebnisses übrigbleibt. Ein Sonderfall ergibt sich, wenn der ganzzahlige Anteil der Division wieder mit einer Potenz von 10 bzw. 2 multipliziert wird. Als Beispiel betrachten wir im Zehnersystem 100*INT(4251/100) = 100x42 = 4200. Bei dieser Operation werden einfach die letzten beiden Ziffern durch Nullen ersetzt. Bei Dualzahlen wird eine entsprechende Rechnung einfach durch eine Maskierung des Bytes mit Hilfe eines AND-Befehls durchgeführt.

Der Programmablauf

Im Programm wird zunächst die Y-Koordinate in den Akkumulator geholt. Durch den Befehl AND &F8 (&F8 = X11111000) werden dann bei diesem Wert die letzten drei Ziffern durch Nullen ersetzt. Nach dem eben besprochenen Verfahren entspricht das der Rechenoperation 8*INT(Y/8). Das Ergebnis wird dann sowohl ins HL- als auch ins BC-Doppelregister geladen. Da es sich um eine 8-Bit-Zahl handelt, hat das High-Byte den Wert Null.
Die anschließenden vier Additionsbefehle bewirken insgesamt eine Multiplikation mit 10. Das geht nach der folgenden Methode: Verdopple eine Zahl (ADD HL,HL), verdopple das Ergebnis, addiere dazu die ursprüngliche Zahl (ADD HL,BC) und verdopple das Ergebnis mit ADD HL,HL nochmals. Probieren Sie das z. B. mit der 3:

```
 3+ 3= 6
 6+ 6=12
12+ 3=15
15+15=30
```

Zum Schluß dieser Operationen steht im HL-Register der Wert 10*8*INT(Y/8) – 80*INT(Y/8).
Der nächste Befehl holt die Y-Koordinate nochmals in den Akkumulator. Dann wird mit AND &07 der Rest bei der Division durch 8 bestimmt (MOD (Y/8)). Auch beim Verständnis dieses Befehls hilft eine Erinnerung ans Zehnersystem. Der Rest von 8721:1000 ist 721; das sind die letzten drei Ziffern von 8721. Im Dualsystem ist 8=X1000. Daher besteht der Rest der Division durch 8 aus den letzten drei Bits der Zahl, und die erhält man mit dem Befehl AND &07 (7=X111). Die drei SLA A-Befehle multiplizieren diesen Rest mit 8. Das Ergebnis wird anschließend zum High-Byte des HL-Registers addiert. Da das High-Byte die hexadezimalen &100er-Stellen enthält, bedeutet diese Operation, daß wir &100*8*MOD(Y/8) = &800*MOD(Y/8) zum Anfangswert des HL-Registers addiert haben. (Man kann auch sagen, daß jeder Beitrag zum High-Byte &100mal soviel wert ist wie ein entsprechender Beitrag zum Low-Byte.)
Der Befehl

```
LD BC,(&0100)
```

holt Low- und High-Byte der X-Koordinate ins BC-Register. Dann wird dieser Wert dreimal hintereinander durch 2 dividiert, was insgesamt den ganzzahligen Anteil der Division von X durch 8 ergibt.
Bei der Division einer 16-Bit-Zahl durch 2 geht man in zwei Schritten vor: Zuerst wird das High-Byte mit SRL um eine duale Stelle nach rechts geschoben. Dann verschiebt man das Low-Byte mit dem RR-Befehl, wobei das aus dem High-Byte ins Carry geschobene Bit automatisch an die höchste Stelle des Low-Bytes übertragen wird. Zusammen mit der Anfangsadresse &C000 des Bildschirmspeichers wird das Ergebnis der eben durchgeführten Operation zum Inhalt von HL addiert. Dort steht jetzt die Adresse der Speicherstelle, die für den zu setzenden Punkt zuständig ist.
Schließlich muß noch das in dieser Speicherstelle zu setzende Bit ermittelt werden. Dazu wird der Achterrest der X-Koordinate bestimmt. Da Vielfache von 256 immer durch 8 teilbar sind, muß bei dieser Bestimmung das High-Byte von X nicht berücksichtigt werden.
Da das Setzen einzelner Bits dem Abspeichern von Zweierpotenzen entspricht, sind diese Potenzen in einer „Tabelle“ am Programmende abgespeichert. Sie von dort zu holen geht viel schneller, als sie jedesmal neu zu berechnen. Ihre Reihenfolge in der Tabelle entspricht der Nummer des zu setzenden Bits.
Um die richtige Zweierpotenz zu laden, wird die Adressierung mit Hilfe des IX-Registers durchgeführt. Zunächst wird IX als Zeiger auf den Tabellenanfang (also auf &80) gerichtet. Der Tabellenwert wird dann mit dem Befehl

```
LD A,(IX+&00)
```

der an der Adresse XR=&403D steht, in den Akkumulator geladen. Das dritte Byte dieses Befehls ist das Abstandsbyte von IX und hat zunächst den Wert Null. Vor der Ausführung dieses Befehls wird jedoch in dem Abstandsbyte der Achterrest der X-Koordinate mit Hilfe des Befehls

LD (XR+2),A

abgespeichert (XR+2 bedeutet die Adresse &403F des Abstandsbytes). Das bedeutet, daß (IX+d) jedesmal auf den richtigen Tabellenwert zeigt. Man sagt, daß sich das Programm dabei selbst modifiziert. Dieser Vorgang wird in Bild 19 veranschaulicht.

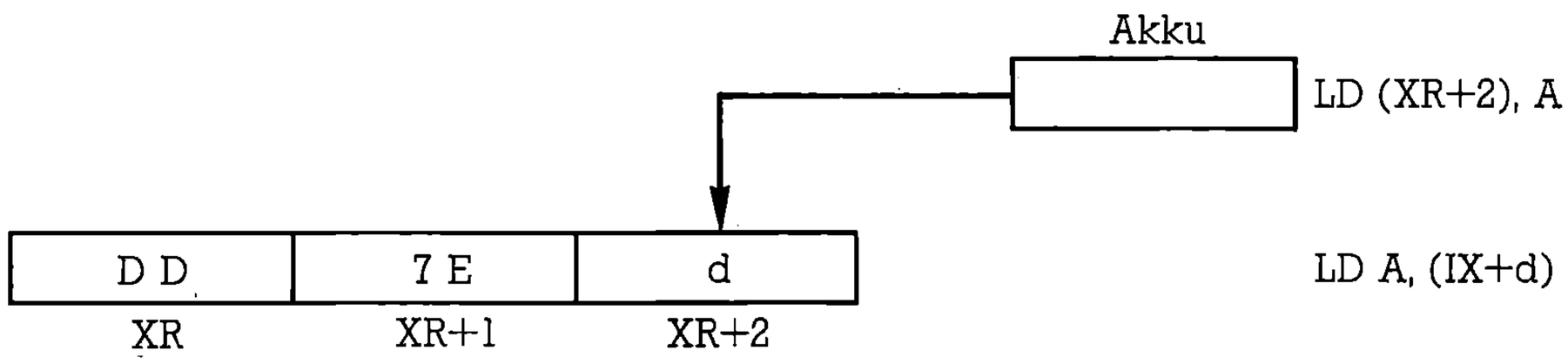


Bild 19 *Schematische Darstellung der Selbstmodifikation des Programms.*

Nach der Berechnung der Adresse und des zu setzenden Bits muß man zum Schluß nochmals aufpassen: Um die anderen Bits des angesprochenen Bytes nicht zu ändern und dadurch bereits vorher gesetzte Nachbarpunkte nicht unbeabsichtigt zu löschen, wird der bisherige Wert der Speicherstelle mit der gefundenen Zweierpotenz durch ein logisches Oder verknüpft, und erst dieser Wert wird abgespeichert.
Um das Punktsetzprogramm ansprechen zu können, schreiben wir ein kleines BASIC-Programm, das einen Strich von der linken oberen Ecke schräg zum unteren Rand ziehen soll. X- und Y-Koordinate sollen der Einfachheit halber bei jedem Punkt denselben Wert haben, daher endet der Strich nicht in der rechten unteren Ecke (X=640; Y=200).

```
10 REM BASICVORSPANN FUER P14
20 MODE 2:POKE &101,0
30 FOR I=0 TO 199
40 POKE &100,I:POKE &102,I:CALL &4000
50 NEXT
```

Zusammenfassung

Der Befehl SRL dividiert eine 8-Bit-Zahl durch 2. Der Divisionsrest geht dabei verloren. Der Befehl SLA multipliziert eine 8-Bit-Zahl mit 2. Das Ergebnis darf nicht größer als 254 sein. Wird der Inhalt des Akkumulators durch 2 bzw. 4 bzw. 8 usw. dividiert, so erhält man den Rest der Division mit AND 1 bzw. AND 3 bzw. AND 7 usw. Entsprechend geht man bei 16-Bit-Zahlen vor: Der Inhalt des Doppelregisters BC wird mit der Befehlsfolge

```
SRL B
RR C
```

durch 2 dividiert. Der Rest der Division geht verloren. Der Inhalt des Doppelregisters BC kann mit der Befehlsfolge

```
SLA C
RL B
```

mit 2 multipliziert werden. Das Ergebnis darf nicht größer als 65534 sein. Entsprechendes gilt für die anderen Doppelregister.

Übung 24

a) Durch welchen Befehl erhält man den Rest der Division des Akkumulatorinhalts durch 16?
b) Welche Assembler-Befehle ergeben 2*INT(B/2), wobei B der Inhalt des B-Registers sein soll?
c) Welche Assembler-Befehle verdoppeln den Inhalt des BC-Registers?

Übung 25

Schreiben Sie ein Maschinenprogramm zum Aufruf unserer Punktsetzroutine, das dieselbe Linie ergibt wie das oben angegebene BASIC-Programm.

Zusatzinformationen

Selbstmodifikation. Die im Punktsetzprogramm dieses Kapitels verwendete Methode zur Beeinflussung des Indexbytes an der Adresse XR+2 (&403F) soll hier noch etwas näher erläutert werden. Die indizierte Adressierung mit (IX+d) und (IY+d) beim Z80-Prozessor hat die Schwäche, daß für das Verschiebebyte d kein Register in der CPU vorhanden ist. So kann während des Programmablaufs zwar der Inhalt der IX- und IY-Register verändert werden, eine Beeinflussung von d ist aber nur möglich, wenn das im Programmspeicher stehende Byte, das den Wert von d angibt, abgeändert wird wie hier mit dem Befehl

```
LD (XR+2),A
```

Dabei ist XR+2 nur eine mnemonische Bezeichnung für die Adresse, die zwei Bytes hinter der mit XR bezeichneten Adresse steht.
Eine derartige Änderung des Programms durch sich selbst nennt man Selbstmodifikation. Obwohl manche Leute dieser Methode aus guten Gründen skeptisch gegenüberstehen, wird sie von professionellen Programmierern häufig verwendet. Allerdings ist eine Selbstmodifikation eines im ROM stehenden Programms nicht möglich. Daher müssen die Teile des Betriebssystems, bei denen das Verfahren angewandt wird, ins RAM ausgelagert werden.
Assemblertabellen. Vielleicht haben Sie sich über die ab Adresse ZP stehende Tabelle der Zweierpotenzen etwas gewundert. Natürlich wäre es auch möglich gewesen, die zum Setzen des Punktes benötigte Zweierpotenz jedesmal zu berechnen. Aber die dazu erforderlichen Befehle hätten wahrscheinlich mehr Platz

verbraucht als unsere Tabelle, und vor allem ist der Zugriff auf eine Tabelle schneller. Der Zugriff auf eine Tabelle hat gegenüber der direkten Berechnung der Werte immer dann Vorteile, wenn nur relativ wenige Werte abgespeichert werden müssen und wenn eine entsprechende Berechnung zeitaufwendig wäre.

Speicherbereiche lesen und schreiben

17

Bisher haben wir uns hauptsächlich damit beschäftigt, die einzelnen Punkte des Bildschirms anzusprechen; auch die Ausgabe von Zeichen erfolgte ja punktweise. Letztlich haben wir also den Speicherinhalt zumindest bei der Ausgabe immer als Bitmuster benutzt. Nun aber wollen wir den Speicherinhalt als Zahlen interpretieren und zunächst versuchen, diese Zahlen auf dem Bildschirm auszugeben. In BASIC erledigt ein PRINT PEEK(.) diese Aufgabe. In jedem Maschinensprache-Monitor gibt es ein sog. Dump-Programm, das nicht nur den Inhalt einer Speicherstelle, sondern den eines ganzen Speicherbereichs ausgibt.

Ein DUMP-Programm zeigt den Speicherinhalt

Ein solches Programm wollen wir jetzt schreiben. Dazu müssen wir uns folgendes überlegen: Jeder Byte-Wert kann durch zwei Hexadezimalziffern angegeben werden. Für eine Ausgabe dieser Ziffern müssen wir den Zahlenwerten ihre ASCII-Codes zuordnen. Die ASCII-Codes der Ziffern 0 bis 9 sind &30 bis &39. Zur Ausgabe holen wir den Zahlenwert in den Akkumulator, setzen durch einen OR-Befehl eine 3 davor und rufen dann die Ausgaberoutine &BB5A auf.
Bei den Hex-Ziffern, die größer als 9 sind, ist die Ausgabe etwas komplizierter. Schauen wir uns das am Beispiel der Dezimalzahl 10 an. Ihre Dualdarstellung ist X1010; das Zeichen A, das wir dieser Zahl zuordnen wollen, hat den ASCII-Code &41=X01000001. Entsprechend ist 11=X1011, und der Code von B ist &42=X01000010, und so fort. Um von der Zahl zum Code des zugeordneten Zeichens zu kommen, müssen wir also das dritte Bit der Zahl löschen, das sechste Bit setzen und schließlich das Ergebnis um den Wert 1 vermindern. Das klingt natürlich etwas verwirrend; eine einfachere Möglichkeit wäre es, mit Hilfe der später noch ausführlicher zu besprechenden Additionsbefehle zu den Zahlenwerten 10, 11,... die Zahl 55=&37 zu addieren, um die entsprechenden ASCII-Codes von A, B,..., F zu erhalten.

Um unser Speicherausgabeprogramm einerseits möglichst flexibel zu gestalten, es aber andererseits nicht zu umfangreich werden zu lassen, benutzen wir einen kurzen BASIC-Vorspann. Darin geben wir die Anfangsadresse und die Länge des auszugebenden Bereichs ein, zerlegen diese Größen jeweils in Low- und High-Byte und speichern sie in den Adressen &100, &101 (Anfangsadresse) und &102, &103 (Länge) ab, wo sie vom Maschinenprogramm übernommen werden. Natürlich wäre es möglich, die BASIC-Zeilen am Beginn des Programms auch durch Maschinencode zu ersetzen. Im zweiten Teil dieses Kapitels wird sich aber zeigen, daß die Eingabe von Zahlen noch etwas mehr Aufwand erfordert als die Ausgabe von Zahlen. Das bedeutet, daß ein nur in Maschinencode geschriebenes Programm etwa doppelt so lang wäre, wie das jetzt folgende:

```
10 REM BASICVORSPANN FUER P15 UND P16
20 REM ANFANGSADRESSE
30 INPUT A:AH=INT(A/256):AL=A-256*AH:IF AH<0 THEN AH=AH+256
40 POKE &100,AL:POKE &101,AH
50 REM LAENGE < &8000
60 INPUT L:LH=INT(L/256):LL=L-256*LH
70 POKE &102,LL:POKE &103,LH
80 CALL &4000
```

```
4000                    1          ORG   #4000
4000  2A0001            2          LD    HL,(#0100)      ;BEREICHSANFANG.
4003  ED4B0201          3          LD    BC,(#0102)      ;BEREICHSLAENGE.
4007  AF                4 LO:      XOR   A               ;AKKU LOESCHEN.
4008  CD1740            5          CALL  AUSG            ;AUSGABE 1.ZIFFER.
400B  CD1740            6          CALL  AUSG            ;AUSGABE 2.ZIFFER.
400E  ED6F              7          RLD                   ;BYTE IN AUSGANGSZUSTAND
                        8                                ;BRINGEN.
4010  23                9          INC   HL              ;NAECHSTE ADRESSE.
4011  0B                10         DEC   BC              ;ALLE BYTES
4012  78                11         LD    A,B             ;BEREITS
4013  B1                12         OR    C               ;AUSGEGEBEN ?
4014  20F1              13         JR    NZ,LO           ;WENN JA,NACH LO.
4016  C9                14         RET                   ;SONST FERTIG.
                        15 ;
4017  ED6F              16 AUSG:   RLD                   ;ZIFFER IN DEN AKKU.
4019  F5                17         PUSH  AF              ;AKKU ZWISCHENSPEICHERN.
401A  FE0A              18         CP    #0A             ;ZIFFER >= 10 ?
401C  3004              19         JR    NC,L1           ;WENN NEIN,
401E  F630              20         OR    #30             ;ASCII-CODE ERZEUGEN,
4020  1803              21         JR    L2              ;UND AUSGEBEN.
4022  3D                22 L1:     DEC   A               ;SONST ASCII-CODE
4023  EE48              23         XOR   #48             ;FUER BUCHSTABEN
                        24 ;                             ;ERZEUGEN.
4025  CD5ABB            25 L2:     CALL  #BB5A           ;AUSGABE.
4028  F1                26         POP   AF              ;URSPRUENGL. AKKU-INHALT
4029  C9                27         RET                   ;HOLEN UND ZURUECK.
                        28 ;P15

AUSG    4017   LO      4007   L1      4022
L2      4025
```

Der RLD-Befehl hilft beim DUMPen

Der wichtigste Befehl in diesem Programm ist RLD, dessen Funktionsweise bereits in Kapitel 12 beschrieben wurde. RLD zerlegt den Inhalt der durch HL indirekt adressierten Speicherstelle in zwei 4-Bit-Blöcke (Nibbles oder Halbbytes) und schiebt den vorderen Block in die niederwertigen vier Bits des Akkumulators (siehe Bild 20). Da jeder 4-Bit-Block gerade eine Hexadezimalziffer enthält, kann diese im Akkumulator in den entsprechenden ASCII-Code umgewandelt und dann mit der Systemroutine &BB5A ausgegeben werden. Genau das wird im Unterprogramm AUSG durchgeführt. Nachdem mit RLD die Ziffer in den Akkumulator gebracht wurde, wird dieser vor der weiteren Verarbeitung mit PUSH AF auf den Stapel kopiert, und dann wird die Ziffer mit dem Wert &A=10 verglichen. Das Ergebnis des Vergleichs kann an der Carry-Flagge abgelesen werden; entsprechend kann mit JR NC verzweigt werden.

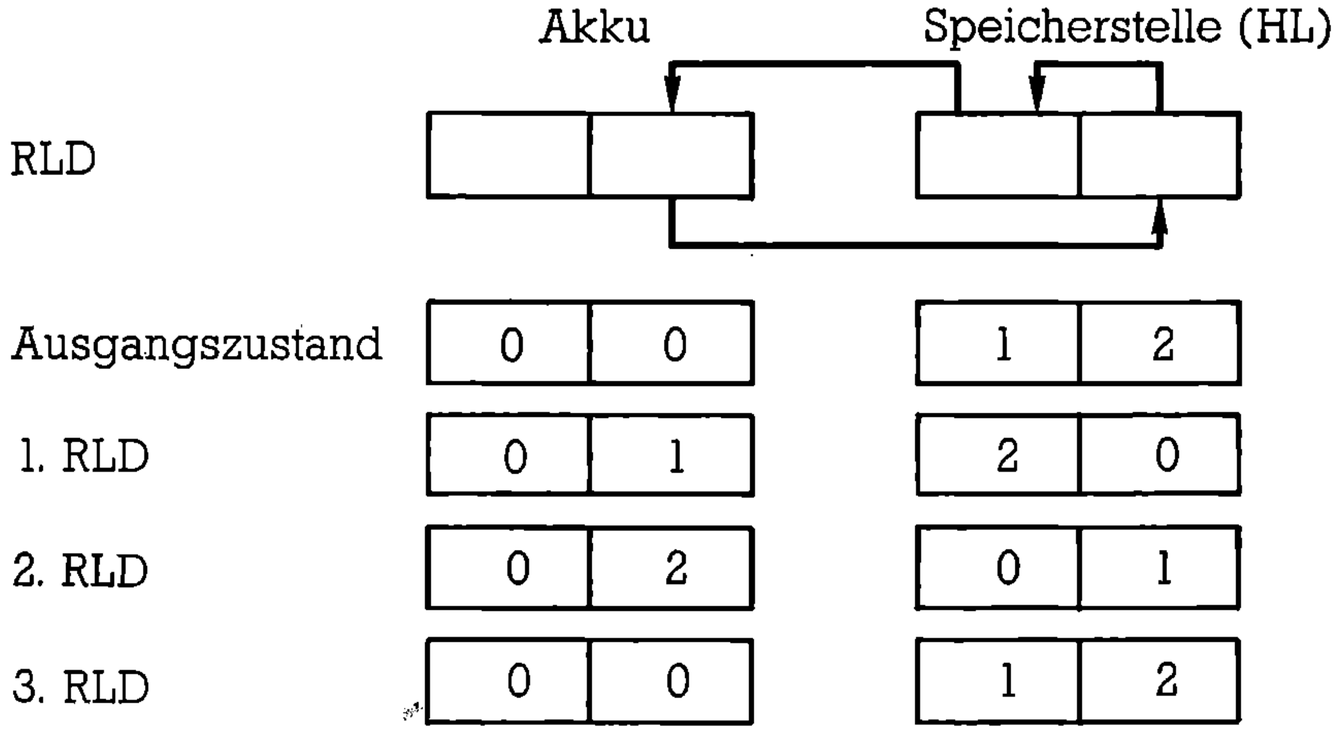


Bild 20 *Nach dreimaliger Anwendung des Befehls RLD befindet sich das durch (HL) adressierte Byte wieder im Ausgangszustand.*

Ist die Ziffer kleiner als 10, dann wird mit OR &30 ihr ASCII-Code erzeugt und dann zum Ausgabeprogramm &BB5A gesprungen. Sonst wird die Ziffer um den Wert 1 vermindert, und mit XOR &48 wird das dritte Bit gelöscht und das sechste gesetzt. Wie oben besprochen wurde, entsteht dadurch der ASCII-Code des entsprechenden Buchstabens A,..., F. Nach der Ausgabe wird der ursprüngliche Inhalt des Akkumulators vom Stapel geholt, und dann erfolgt der Rücksprung ins Hauptprogramm.
Im Hauptprogramm wird die Anfangsadresse des auszugebenden Bereichs in HL und die Zahl der auszugebenden Bytes in BC geladen. Dann beginnt die Ausgabeschleife. Zuerst wird mit XOR A der Akkumulator gelöscht. Dann erfolgt durch zweimaligen Aufruf des Unterprogramms AUSG die Ausgabe der beiden

Ziffern des adressierten Bytes. Um den Inhalt des gerade untersuchten Bytes wiederherzustellen, muß danach nochmals der RLD-Befehl ausgeführt werden. (Auch in AUSG diente die Zwischenspeicherung des Akkumulators auf dem Stapel dem Ziel, den Inhalt des auszugebenden Bytes nicht zu zerstören.) Der Rest des Programms muß wohl nicht weiter erläutert werden.

Hex-Zahlen: von der Taste in den Speicher

Nach dem DUMP-Programm ist es nicht allzu schwierig, ein entsprechendes „POKE"-Programm zu schreiben. Wir wollen dabei, beginnend mit einer bestimmten Speicherstelle, eine gewisse Anzahl von Bytes in den Speicher eingeben. Im Vergleich mit dem DUMP-Programm ist hier allerdings eine kompliziertere Abfrage notwendig, da nur die Zifferntasten und die Buchstabentasten für A bis F angenommen werden sollen. Die erlaubten ASCII-Codes liegen zwischen &30 und &39 für die Ziffern und zwischen &41 und &46 für die Buchstaben. Um diese Bereiche einzugrenzen, sind insgesamt vier Abfragen notwendig.
Da es sich hier um „Größer-oder-gleich"- oder „Kleiner"-Entscheidungen handelt, wird für die Verzweigungen nach dem CP-Befehl jeweils der Zustand der Carry-Flagge abgefragt. Die Umwandlung des ASCII-Codes in die Dualdarstellung der Hex-Ziffern erfolgt entsprechend den oben durchgeführten Überlegungen: Für die Ziffern 0 bis 9 muß lediglich mit AND &0F die führende 3 des ASCII-Codes weggeblendet werden. Bei den Buchstabencodes wird zunächst eine 1 addiert, und danach werden mit XOR &48 das dritte Bit gesetzt und das sechste Bit gelöscht. Beispiel: Der ASCII-Code von A ist &41. Die Addition von 1 ergibt &42=X01000010. Das Setzen des dritten Bits und Löschen des sechsten Bits ergibt X00001010=&0A=10.
Das Eingabe-Programm hat denselben BASIC-Vorspann wie das Dump-Programm. Der Rest hat folgendes Aussehen:

```
4000                    1          ORG   #4000
4000  2A0001            2          LD    HL,(#0100)        ;BEREICHSANFANG.
4003  ED4B0201          3          LD    BC,(#0102)        ;BEREICHSLAENGE.
4007  CD1440            4 LO:      CALL  EING              ;EINGABE 1. ZIFFER.
400A  CD1440            5          CALL  EING              ;EINGABE 2. ZIFFER.
400D  23                6          INC   HL                ;NAECHSTE ADRESSE.
400E  0B                7          DEC   BC                ;PRUEFEN OB SCHON
400F  78                8          LD    A,B               ;ALLE BYTES EIN-
4010  B1                9          OR    C                 ;GEGEBEN SIND.
4011  20F4             10          JR    NZ,LO             ;WENN NEIN,NACH LO.
4013  C9               11          RET                     ;SONST FERTIG.
                       12 ;
4014  CD06BB           13 EING:    CALL  #BB06             ;TASTATURABFRAGE.
4017  FE30             14          CP    #30               ;VERGLEICH MIT ASC(0).
4019  381B             15          JR    C,END             ;WENN KLEINER,
                       16                                  ;NICHT ANNEHMEN.
401B  FE3A             17          CP    #3A               ;VERGLEICH MIT ASC(9).
401D  3005             18          JR    NC,L1             ;WEITER WENN GROESSER.
401F  F5               19          PUSH  AF                ;EINGABE FUER KONTROLLE
```

```
                    20
4020  E60F          21            AND   #0F
4022  180C          22            JR    ROT
4024  FE41          23 L1:        CP    #41
4026  380E          24            JR    C,END
                    25
4028  FE47          26            CP    #47
402A  300A          27            JR    NC,END
                    28
402C  F5            29            PUSH  AF
                    30
402D  3C            31            INC   A
402E  EE48          32            XOR   #48
4030  ED6F          33 ROT:       RLD
4032  F1            34            POP   AF
4033  CD5ABB        35            CALL  #BB5A
4036  C9            36 END:       RET
                    37 ;P16

EING    4014   END     4036   LO      4007
L1      4024   ROT     4030

;SPEICHERN.
;ZIFFERNWERT ABTRENNEN.
;ZUM EINSPEICHERN.
;VERGLEICH MIT ASC(A).
;WENN KLEINER,
;NICHT ANNEHMEN.
;VERGLEICH MIT ASC(F).
;WENN GROESSER
;NICHT ANNEHMEN.
;EINGABE FUER KONTROLLE
;SPEICHERN.
;ZIFFERNWERT
;ABTRENNEN.
;ZIFFER EINSPEICHERN.
;AKKU WIEDERHOLEN
;UND AUSGEBEN.
;ZUM HAUPTPROG.
```

Übung 26 Ändern Sie das Dump-Programm P15 so ab, daß zwei Bytes jeweils durch ein Leerzeichen voneinander getrennt ausgegeben werden.

Übung 27 Ändern Sie das Dump-Programm P15 so ab, daß für die Ziffern A bis F Kleinbuchstaben ausgegeben werden.

Übung 28 Streichen Sie im Unterprogramm AUSG des Dump-Programms P15 die Befehle PUSH AF und POP AF, und überlegen Sie sich die Folgen. Führen Sie dann das geänderte Programm aus.

18 Der Z80 als Rechner

In diesem und den nächsten Kapiteln werden wir uns um die arithmetischen Befehle der Z80-CPU kümmern. Es wird sich herausstellen, daß der Prozessor eigentlich nur im Zahlbereich von 0 bis 65535 addieren und subtrahieren kann. Wegen dieser bescheidenen Rechenfähigkeiten sind einige Programmierkenntnisse erforderlich, um kompliziertere Berechnungen durchführen zu können. Eine entscheidende Rolle bei der Erweiterung der arithmetischen Fähigkeiten der Z80-CPU spielen verschiedene Flaggen.

Rechnen: mangelhaft

Das Hauptanwendungsgebiet der ersten Computer war die Durchführung komplizierter Rechnungen. Das besagt schon der Name „Computer", der das englische Wort für „Rechner" ist. Auch BASIC und natürlich andere höhere Programmiersprachen statten den Computer in dieser Beziehung mit erheblichen Fähigkeiten aus. Neben den Grundrechenarten können eine ganze Reihe komplizierter Funktionen berechnet werden, und ein Benutzer mit entsprechenden Kenntnissen kann schon mit Hilfe von Homecomputern verwickelte mathematische Probleme lösen.
Es ist daher auf den ersten Blick überraschend, daß die gängigsten Mikroprozessoren auf der Ebene der Maschinensprache nur bescheidene mathematische Fähigkeiten aufweisen. Während z. B. die CPU 6502 (6510), die in Commodore- und Apple-Computern verwendet wird, nur Additionen und Subtraktionen im Bereich zwischen −128 und 127 bzw. zwischen 0 und 255 ausführen kann, beherrscht die Z80 CPU diese Rechenarten immerhin im Bereich zwischen −32768 und 32767 bzw. zwischen 0 und 65535.
Alle komplizierteren Rechenoperationen müssen aus einfachsten Grundoperationen aufgebaut werden. Das bedeutet, daß die oben angesprochenen mathematischen Fähigkeiten der Programmiersprache BASIC auf umfangreichen Pro-

grammen beruhen, die im Betriebssystem bzw. dem BASIC-Interpreter des Rechners enthalten sind. Natürlich können diese mathematischen Systemunterprogramme auch von Maschinenspracheprogrammen aufgerufen werden. In diesem Kapitel werden wir uns jedoch um die unmittelbaren mathematischen Fähigkeiten des Prozessors kümmern und nicht um die Leistungsfähigkeit des Betriebssystems.
Der Z80-Prozessor besitzt zwei Arten von arithmetischen Befehlsgruppen. Zum einen gibt es Befehle, die mit einem Byte (also 8 Bit) rechnen, zum anderen gibt es 16-Bit-Rechenbefehle. Von diesen Befehlen haben wir die INC- und DEC-Befehle sowie den Additionsbefehl ADD bereits kennengelernt und auch verwendet.

Rechnen mit 8 Bits

Wir wollen uns jetzt zunächst um die bisher nicht verwendeten 8-Bit-Additionsbefehle ADD (Addition) und ADC (Addition mit Carry) sowie die Subtraktionsbefehle SUB (Subtraktion) und SBC (Subtraktion mit Carry) kümmern. Diese Befehle haben die Struktur

```
ADD A,r
ADC A,r
SUB r
SBC A,r
```

wobei r für eines der Register A, B, C, D, E, H, L, eine 8-Bit-Zahl n oder eine durch (HL), (IX+d) bzw. (IY+d) adressierte Speicherstelle steht. Beim ADD-Befehl wird r zum Inhalt von A addiert. ADC addiert r und den Wert der Carry-Flagge zu A. Entsprechend subtrahiert SUB den Inhalt von r vom Inhalt von A, und SBC subtrahiert r und den Wert der Carry-Flagge von A. Bei allen vier Befehlen wird das Ergebnis in A gespeichert.
Es scheint auf den ersten Blick etwas inkonsequent zu sein, daß beim SUB-Befehl die Angabe des Operanden A fehlt. Andererseits gibt es von diesem Befehl im Gegensatz zu ADD, ADC, SBC keine 16-Bit-Version, so daß eine Angabe von A zur Vermeidung einer Verwechslung nicht notwendig ist. Den Sinn des ADC- und SBC-Befehls werden wir gleich kennenlernen. Bei ihrer Anwendung muß man darauf achten, durch ein eventuelles Löschen der C-Flagge (mit OR A oder AND A) das Ergebnis nicht zu verfälschen.
Die Anwendung dieser Befehle ist unproblematisch, solange Ausgangswerte und Ergebnisse nichtnegative Zahlen unterhalb von 256 sind. Wir wollen das einmal ausprobieren. Um den Programmieraufwand möglichst gering zu halten, verwenden wir zur Ein- und Ausgabe von Zahlen die BASIC-Befehle POKE und PEEK. Als Übergabespeicher für das Maschinenprogramm verwenden wir die Adressen &100 und &101.

```
10 REM BASICVORSPANN FUER P17
20 INPUT A:POKE &100,A:REM 1.SUMMAND
30 INPUT B:POKE &101,B:REM 2.SUMMAND
40 CALL &4000
50 PRINT PEEK(&100)
```

```
4000                    1           ORG   #4000
4000  210101            2           LD    HL,#0101          ;ZEIGER AUF 1.SUMMANDEN.
4003  3A0001            3           LD    A,(#0100)         ;2.SUMMAND IN DEN AKKU.
4006  86                4           ADD   A,(HL)            ;ADDIEREN.
4007  320001            5           LD    (#0100),A         ;SUMME NACH #100.
400A  C9                6           RET                     ;ZURUECK NACH BASIC.
                        7 ;P17
```

Solange das Ergebnis in den Akkumulator A hineinpaßt, also nicht größer als 255 ist, erhalten wir mit dem obigen Programm korrekte Ergebnisse. Ersetzen wir

ADD A,(HL)

durch

ADC A,(HL)

dann ist das Ergebnis, je nach dem Zustand der C-Flagge zu Programmbeginn, evtl. um 1 zu groß. Um sicherzugehen, löscht man in diesem Fall vor dem ADC-Befehl die C-Flagge mit OR A. Entsprechend können wir mit SUB (HL) und SBC A,(HL) auch subtrahieren. Das Ergebnis muß aber größer oder gleich Null sein, d. h., daß der Inhalt der Speicherstelle (&101) nicht größer als der der Speicherstelle (&100) sein darf. Die Verarbeitung negativer Zahlen wird erst weiter unten besprochen.

Rechnen mit doppelter Genauigkeit

In vielen Fällen reicht natürlich der Zahlenbereich bis 255 nicht aus. Schon bei der Addition von 128+128 liefert unser obiges Programm das falsche Ergebnis 0. Das Ergebnisbyte hat die bereits bekannte „Kilometerzählerreaktion" gezeigt und bei Überschreitung der Kapazität die führende Stelle scheinbar verschluckt. Aber wie wir bereits von früheren Anwendungen her wissen, ist diese führende Stelle nicht verloren, sondern wird in der Carry-Flagge aufbewahrt.
Um mit unserer 8-Bit-Arithmetik in einem größeren Zahlenbereich rechnen zu können, benötigen wir daher mehrere Bytes und die Möglichkeit, den Inhalt der Carry-Flagge beim Übergang von einem Byte zum nächsten berücksichtigen zu können. Diese Möglichkeit geben uns gerade die ADC- und SBC-Befehle. Schauen wir uns das bei einer Rechnung im 16-Bit-Bereich zwischen 0 und 65535 an. Zunächst müssen wir unsere Zahlen in ein High-Byte (Vielfache von 256) und ein Low-Byte (Rest) zerlegen. (Wir haben das ja bereits in den früheren Kapiteln kennengelernt.) Diese Zerlegung führen wir wieder in BASIC durch. Die erste zu

verarbeitende Zahl soll dabei in den Adressen &100 und &101 und die zweite in &102 und &103 abgespeichert werden. Die Addition und Subtraktion übernimmt das Maschinenprogramm, und die Umrechnung des Ergebnisses aus L- und H-Byte wird wieder in BASIC durchgeführt.

```
10 REM BASICVORSPANN FUER P18 UND P19
20 INPUT A:AH=INT(A/256):AL=A-256*AH:REM 1. SUMMAND
30 POKE &100,AL:POKE &101,AH
40 INPUT B:BH=INT(B/256):BL=B-256*BH:REM 2. SUMMAND
50 POKE &102,BL:POKE &103,BH
60 CALL &4000
70 PRINT 256*PEEK(&101)+PEEK(&100)
```

```
4000                    1           ORG   #4000
4000   210201           2           LD    HL,#0102        ;ZEIGER AUF LOW-BYTE
                        3                                   ;DES 1.SUMMANDEN.
4003   3A0001           4           LD    A,(#0100)       ;LOW-BYTE 2.SUMMAND.
4006   86               5           ADD   A,(HL)          ;ADDITION DER LOW-BYTES.
4007   320001           6           LD    (#0100),A       ;LOW-BYTE DER SUMME.
400A   23               7           INC   HL              ;ZEIGER AUF HIGH-BYTE
                        8                                   ;DES 1.SUMMANDEN.
400B   3A0101           9           LD    A,(#0101)       ;HIGH-BYTE 2.SUMMAND.
400E   8E               10          ADC   A,(HL)          ;ADDITION DER HIGH-BYTES
                        11                                  ;UND DES UEBERTRAGS.
400F   320101           12          LD    (#0101),A       ;HIGH-BYTE DER SUMME.
4012   C9               13          RET                   ;ZURUECK NACH BASIC.
                        14 ;P18
```

Zum Programm ist nur zu sagen, daß, wie beim normalen Addieren bzw. Subtrahieren von Hand, mit der Verarbeitung der niederwertigen Stellen (also der L-Bytes) begonnen werden muß, um einen eventuellen Übertrag an die höherwertigen Stellen mit Hilfe des ADC-Befehls weitergeben zu können. Die Verwendung des ADD-Befehls statt des ADC-Befehls für die Addition der L-Bytes erspart uns das Löschen der C-Flagge; ein Übertrag ins L-Byte hinein ist ja ausgeschlossen. Bild 21 zeigt noch einmal den Rechengang.

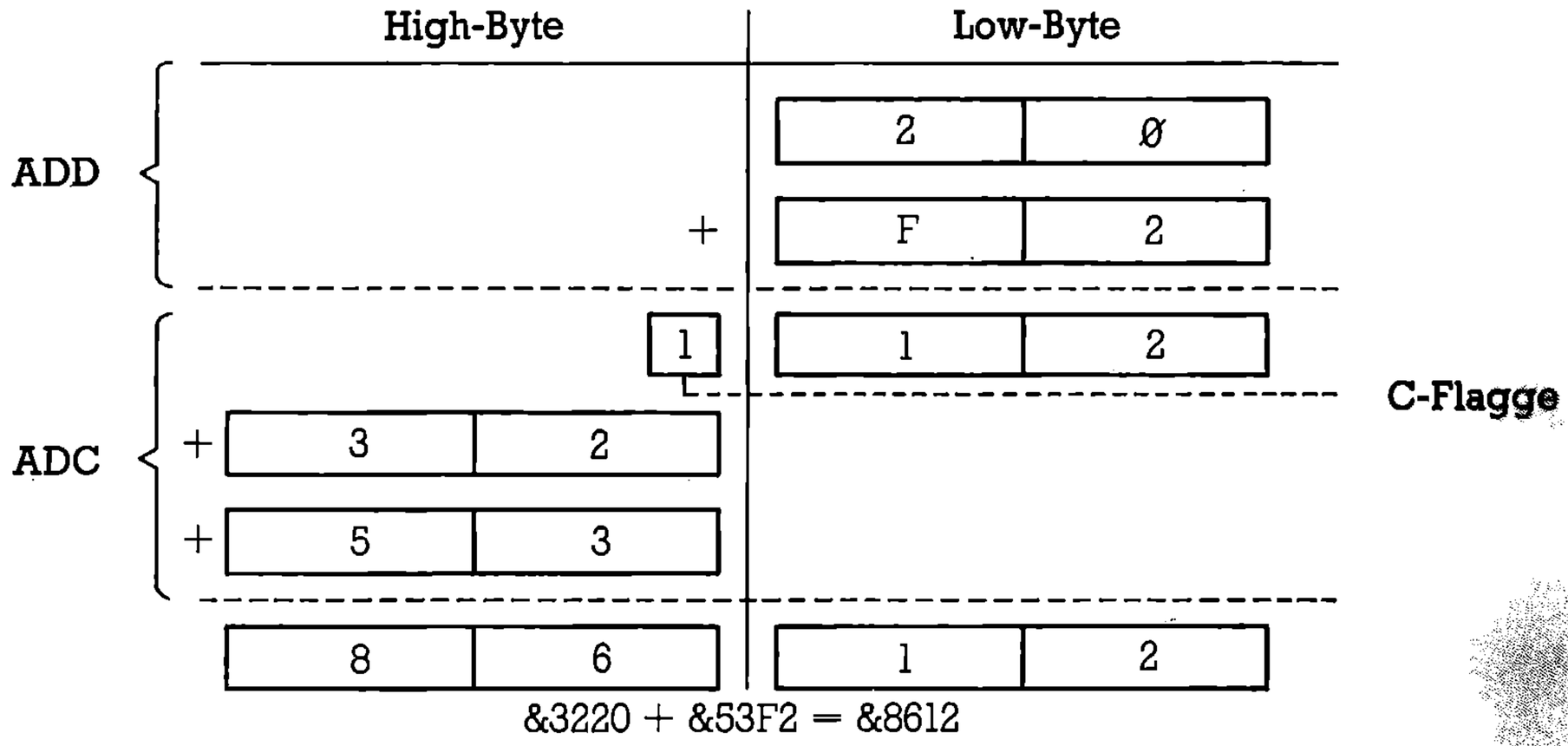


Bild 21 *Der Rechenweg im Programm P18 zur Berechnung von &3220+&53F2=&8612.*

Natürlich hätte das Ziel des obigen Programms auch mit geringerem Aufwand, nämlich durch die Benutzung von 16-Bit-Befehlen erreicht werden können. Es zeigt uns aber den Weg, wie wir mit Hilfe von mehreren Bytes für eine Zahl und die Aneinanderkettung der entsprechenden Rechenbefehle Berechnungen mit beliebig vielen Ziffern ausführen können.
Ersetzen wir die Additionsbefehle

```
ADD A,(HL)
```

und

```
ADC A,(HL)
```

durch die Subtraktionsbefehle

```
SUB (HL)
```

und

```
SBC A,(HL)
```

so können wir im Zahlenbereich zwischen Null und 65535 subtrahieren, wobei das Ergebnis nicht negativ werden darf. Falls das L-Byte der zweiten Zahl größer als das L-Byte der ersten Zahl ist, ergibt sich nach dem ersten Subtraktionsbefehl ein negativer Übertrag, der wie bei der Addition in der Carry-Flagge gespeichert und durch den SBC-Befehl bei der Subtraktion der H-Bytes berücksichtigt wird.

Die Stärke des Z80: 16-Bit-Befehle

Doch schauen wir uns nun die 16-Bit-Arithmetikbefehle an: Neben dem bereits bekannten 16-Bit-Additionsbefehl ADD gibt es noch die Befehle ADC und SBC in einer 16-Bit-Version. Hierbei übernimmt das HL-Register die Rolle, die der Akkumulator bei den 8-Bit-Befehlen spielt, d. h., einer der Ausgangswerte und das Ergebnis stehen in HL. Die andere Zahl kann in BC, DE, HL oder SP stehen. (Eine Ausnahme bildet, wie in Kapitel 9 besprochen, der ADD-Befehl, bei dem HL auch durch IX und IY ersetzt werden kann.) Das 16-Bit-Additionsprogramm hat mit diesen Befehlen folgendes Aussehen:

```
4000                    1          ORG   #4000
4000  2A0001            2          LD    HL,(#0100)      ;1.SUMMAND NACH HL.
4003  ED4B0201          3          LD    BC,(#0102)      ;2.SUMMAND NACH BC.
4007  09                4          ADD   HL,BC           ;2-BYTE ADDITION.
4008  220001            5          LD    (#0100),HL      ;SUMME NACH #100,#101.
400B  C9                6          RET                   ;ZURUECK NACH BASIC.
                        7    P19
```

Würde man anstelle des ADD- den ADC-Befehl verwenden, so müßte zunächst die Carry-Flagge gelöscht werden. Dasselbe gilt immer bei einer 16-Bit-Subtraktion, da hier nur der SBC-Befehl in Frage kommt. Bei einer zweistufigen Anwendung dieser Befehle wäre es bereits möglich, mit 32-Bit-Zahlen zu rechnen. Das

ist der Bereich zwischen 0 und $2^{32}-1 = 4294967296$. Da die Integerarithmetik von BASIC nicht mehr in der Lage ist, derart große Zahlen zu verarbeiten, können wir die Zerlegung der Zahlen in einzelne Bytes auch nicht mehr ohne weiteres mit Hilfe von BASIC-Befehlen durchführen. Daher wollen wir hier auf die Durchführung einer derartigen Rechnung verzichten.

Zusammenfassung

Die Z80-CPU besitzt die 8-Bit-Additions- und Subtraktionsbefehle

```
ADD A,r
ADC A,r
SBC A,r
SUB r
```

r steht für eines der 8-Bit-Register A, B, C, D, E, H, L, eine 8-Bit-Zahl oder eine durch (HL), (IX+d) bzw. (IY+d) adressierte Speicherstelle.
ADD addiert r zu A und speichert das Ergebnis in A.
ADC addiert r und den Wert der C-Flagge zu A und speichert das Ergebnis in A.
SUB subtrahiert r von A und speichert den Inhalt in A.
SBC subtrahiert r und die C-Flagge von A und speichert das Ergebnis in A.
Neben den 8-Bit-Befehlen gibt es noch die 16-Bit-Rechenbefehle

```
ADD HL,rr
ADC HL,rr
SBC HL,rr
```

rr steht hier für eines der 16-Bit-Register BC, DE, HL, SP. Die Befehle haben entsprechende Wirkungen wie die 8-Bit-Befehle. Die Rolle, die dort der Akkumulator spielt, übernimmt bei den 16-Bit-Befehlen das HL-Doppelregister.
Die Befehle ADC und SBC können bei Rechnungen verwendet werden, bei denen eine Zahl durch mehrere Bytes dargestellt wird. Durch diese Befehle wird der in der C-Flagge gespeicherte Übertrag aus dem niederwertigen Byte (bzw. Doppelbyte) ins höherwertige Byte (bzw. Doppelbyte) berücksichtigt.

Übung 29 Schreiben Sie die Dezimalzahlen 12, 15, 127, 128, 200 in die Dualzahldarstellung um. Addieren Sie in der Dualdarstellung

12+15
127+12
128+200

Übung 30 Schreiben Sie ein Additionsprogramm, bei dem 24-Bit-Zahlen (Bereich 0 bis 16777215) verarbeitet werden können.

Übung 31 Ersetzen Sie im Programm P18 den Befehl ADC A,(HL) durch ADD A,(HL), und beobachten Sie an einigen Beispielen, wie sich die Vernachlässigung des Übertrags aus dem L-Byte auswirkt.

Negative Zahlen sind die „Größten“

19

In vielen Fällen reichen die bisher besprochenen arithmetischen Fähigkeiten unserer Z80-CPU aus. Manchmal ist es jedoch nicht zu vermeiden, auch mit negativen Zahlen zu rechnen. Genaugenommen haben wir das sogar bisher gemacht, ohne es zu merken: Beim Subtrahieren addiert der Rechner nämlich negative Zahlen.
Wir wollen uns zunächst anschauen, wie der Rechner negative Zahlen darstellt. Wenn wir das verstanden haben, können wir auch negative Zahlen als Ergebnis einer arithmetischen Operation zulassen. Dabei wird sich glücklicherweise herausstellen, daß die CPU bei der vorzeichenbehafteten Arithmetik gar nicht anders rechnet als bisher; nur die Zahlen werden dabei anders interpretiert. Und dafür stellt die CPU noch eine Interpretationshilfe zur Verfügung: die Overflow- oder V-Flagge.

Das Vorzeichenbit

Zunächst beschränken wir uns wieder auf Ein-Byte-Zahlen. Von den acht Bits verwendet man jetzt eines zur Darstellung des Vorzeichens, so daß für den Betrag der Zahlen nur noch sieben Bits zur Verfügung stehen. Bei der Z80-CPU wird das linksstehende (siebte) Bit als Vorzeichenbit verwendet. Hat dieses Bit den Wert 1, so wird die Zahl als negativ angesehen. Daraus ergibt sich, daß die Zahlen zwischen 0 und 127=X01111111 ihre bisherige Bedeutung behalten. Die Bitmuster zwischen X10000000 und X11111111, die bisher als die Zahlen von 128 bis 255 interpretiert wurden, werden jetzt als negative Zahlen angesehen. Dieses Verfahren nennt man die 8-Bit-„Zweierkomplement-Darstellung“. Will man eine negative Zahl darstellen, so subtrahiert man den Betrag dieser Zahl von 256. Dabei sind nur Werte von −1 bis −128 erlaubt. Konkret heißt das: 255=256−1 wird als −1 interpretiert, 254=256−2 wird als −2 interpretiert..., 128=256−128 wird als −128 interpretiert. Vereinbart man, daß ein eventuell auftretender Übertrag ins Carry einfach

ignoriert wird, so liefert die Addition dieser Zahlen korrekte Ergebnisse, solange diese und das Ergebnis im erlaubten Bereich zwischen −128 und +127 liegen.

Zahl	Zweierkomplement	Bitmuster
−128	256 − 128 = 128	1 0 0 0 0 0 0 0
−127	256 − 127 = 129	1 0 0 0 0 0 0 1
−2	256 − 2 = 254	1 1 1 1 1 1 1 0
−1	256 − 1 = 255	1 1 1 1 1 1 1 1
0	0	0 0 0 0 0 0 0 0
1	1	0 0 0 0 0 0 0 1
127	127	0 1 1 1 1 1 1 1

Tabelle 7 *Bildung des 8-Bit-Zweierkomplements.*

Für das Folgende ist es weiterhin wichtig zu wissen, daß das Zweierkomplement einer bereits im Zweierkomplement dargestellten negativen Zahl den Betrag dieser Zahl ergibt. Beispiel: Die Zahl hat den Wert −10; das erste Zweierkomplement ergibt

$$256-10 = 246$$

Nochmalige Zweierkomplementbildung ergibt

$$256-246 = 10$$

Hier einige Beispiele für Rechnungen mit dem Zweierkomplement:

1. 10+(−7) wird gerechnet:

$$10+256-7=256+3$$

Der Summand 256 bedeutet im Endergebnis nur, daß die Carry-Flagge gesetzt wird. Das Ergebnisbyte hat den Wert 3. Da wir vereinbarungsgemäß die Carry-Flagge ignorieren, erhalten wir das korrekte Ergebnis:

$$10+(-7)=3$$

2. 10+(−15) wird gerechnet:

$$10+256-15=256-5$$

Das Ergebnis liegt im negativen Bereich (siebtes Bit gesetzt) und wird vereinbarungsgemäß nicht als 251, sondern als −5 interpretiert. Also:

$$10+(-15)=-5.$$

3. −15+(−5) wird gerechnet:

256−15+256−5=256+256−20

Der erste Summand 256 erscheint wieder in der Carry-Flagge und wird ignoriert. Übrig bleibt 256−20, was korrekt als −20 interpretiert wird.

Rechnen mit positiven und negativen Zahlen

Ein kleines Beispielprogramm soll das Funktionieren dieser Methode zeigen. Die Vorbereitung der Rechnung und die Ausgabe des Ergebnisses soll wieder mit Hilfe einiger BASIC-Anweisungen durchgeführt werden.

```
10 REM BASICVORSPANN FUER P20
20 INPUT A:IF A<0 THEN A=A+256:REM 1. SUMMAND (ZWEIERKOMPLEMENT)
30 POKE &100,A
40 INPUT B:IF B<0 THEN B=B+256:REM 2. SUMMAND (ZWEIERKOMPLEMENT)
50 POKE &101,B
60 CALL &4000
70 PRINT CHR$(PEEK(&100));:PRINT PEEK(&101)
```

```
4000                    1           ORG   #4000
4000  210101            2           LD    HL,#0101          ;ZEIGER AUF 1.SUMMAND.
4003  3A0001            3           LD    A,(#0100)         ;2.SUMMAND IN DEN AKKU.
4006  86                4           ADD   A,(HL)            ;ADDITION.
4007  FA1140            5           JP    M,L1              ;NACH L1 FALLS
                        6                                   ;7.BIT GESETZT (MINUS).
400A  320101            7           LD    (#0101),A         ;SONST SUMME NACH #101.
400D  3E2B              8           LD    A,#2B             ;ASCII-CODE VON "+".
400F  1807              9           JR    L2                ;ZUR AUSGABE.
4011  ED44             10 L1:       NEG                     ;ZWEIERKOMPLEMENT
                        11 ;                                ;ERGIBT HIER BETRAG.
4013  320101           12           LD    (#101),A          ;ERGEBNIS NACH #101.
4016  3E2D             13           LD    A,#2D             ;ASCII-CODE VON "-".
4018  320001           14 L2:       LD    (#0100),A         ;VORZEICHEN NACH #100.
401B  C9                15           RET                     ;NACH BASIC.
                        16 ;P20

L1      4011  L2      4018
```

Im BASIC-Teil des Programms werden die eingegebenen Werte zunächst in ihr Zweierkomplement umgewandelt, falls sie negativ sind. Im Maschinenprogramm werden dann zwei neue Befehle benutzt. Nach dem Additionsbefehl wird mit

JP M,L1

nach L1 verzweigt, falls das Ergebnis negativ ist. Dieser Befehl erinnert an die bereits bekannten Verzweigungsbefehle JP C,nn bzw. JP Z,nn. Tatsächlich wird auch hier eine Flagge abgefragt. Das ist die sog. Vorzeichenflagge (S-Flagge vom englischen „sign", Vorzeichen), die einfach den Wert des siebten Bits beim Er-

gebnis einer Rechenoperation anzeigt. Die Bedingung M (von „minus") ist erfüllt, wenn das siebte Bit und damit die S-Flagge den Wert 1 hat. Falls der Wert der S-Flagge Null ist, kann diese Bedingung mit JP P,nn abgefragt werden (P von „positiv"). Falls im obigen Programm die M-Bedingung nicht erfüllt ist, also bei positivem Ergebnis oder Null, wird das Ergebnis in Adresse &101 gespeichert. Anschließend wird der ASCII-Wert des Zeichens „+" in die Speicherstelle &100 gebracht. Bei negativem Resultat wird der Sprung nach L1 ausgeführt. Der Befehl

NEG

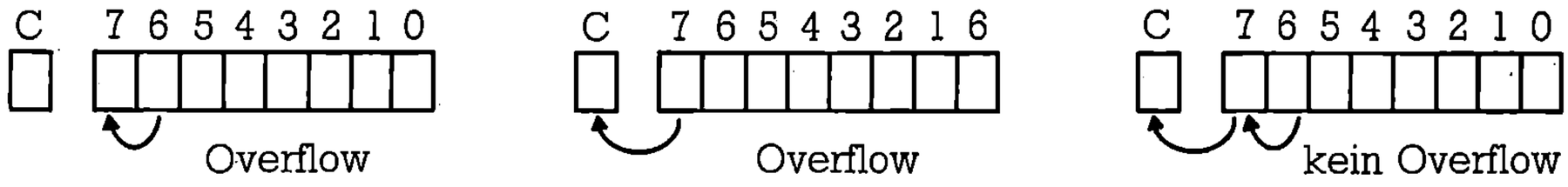
den wir hier zum ersten Mal benutzen, bewirkt, daß das Zweierkomplement des Rechenergebnisses, das im Akkumulator steht, gebildet wird. Dadurch wird das negative Resultat in seinen Betrag verwandelt, der dann zusammen mit dem Vorzeichen „—" ausgegeben wird.

Die Overflow-Flagge

Unser Rechenprogramm P20 funktioniert, solange die Ergebnisse im Bereich zwischen —128 und +127 liegen. Schwierigkeiten tauchen aber auf, wenn diese Grenzen überschritten werden. Probieren Sie einmal, mit dem Programm 100+100 zu berechnen. Statt +200 erhalten Sie das Ergebnis —56. Entsprechend ergibt z. B. —128—120 nicht den Wert —248, sondern +8. Der Grund für diese Fehler ist offensichtlich: Die Summe 100+100 liegt oberhalb von 127. Das siebte Bit ist also gesetzt, und daher wird der Wert als negative Zahl interpretiert. 200 ist gerade das Zweierkomplement von —56: 256—56=200, und das wird auf dem Bildschirm ausgegeben. Entsprechend wird bei —120—128 mit dem Zweierkomplementen gerechnet:

256—120+256—128=256—120+128=256+8

Da der Summand 256 als Carryübertrag vernachlässigt wird, bleibt der Wert 8 übrig, der als +8 ausgegeben wird. Offensichtlich rechnet also der Computer korrekt, und wir sind an dieser Stelle nur nicht in der Lage, die Ergebnisse zu interpretieren.
Hier hält nun der Computer eine weitere Flagge zu unserer Hilfe bereit. Das ist die „Overflow-" oder „V-Flagge" (vom englischen „overflow", Überlauf). Sie wird jedesmal gesetzt, wenn ein Ergebnis im falschen Vorzeichenbereich landet. Das kann einmal dadurch passieren, daß ein Übertrag aus dem sechsten Bit ins siebte Bit des Akkumulators stattfindet, ohne daß dieser das Vorzeichen verändernde Übertrag durch einen entsprechenden Übertrag aus dem siebten Bit ins Carry wieder wettgemacht wird (wie bei 100+100). Oder der Overflow kann durch einen Übertrag aus dem siebten Bit ins Carry ausgelöst werden, der nicht durch einen Übertrag aus dem sechsten ins siebte Bit ausgeglichen wird (wie bei —120—128). (Siehe auch Bild 22.)



100 + 100 = X01100100 + X01100100

```
 01100100
 01100100
 11001000
 ↷
```

Übertrag ins 7. Bit
kein Übertrag ins Carry
V-Flagge wird gesetzt

−120 + (−128) = X1000100 + X10000000

```
  10001000
  10000000
C 1 00001000
  ↷
```

kein Übertrag ins 7. Bit
Übertrag ins Carry
V-Flagge wird gesetzt

Bild 22 *Die Verwendung der Overflow-Flagge.*

Wie unsere Beispiele vermuten lassen, kann ein Overflow nur bei der Addition zweier Zahlen mit gleichem Vorzeichen (oder der Subtraktion zweier Zahlen mit unterschiedlichen Vorzeichen) eintreten.

16-Bit-Zahlen mit Vorzeichen

Durch Abfrage der V-Flagge kann eine Fehlinterpretation des Rechenergebnisses vermieden werden. Das folgende Programm zeigt, wie man mit den arithmetischen 16-Bit-Befehlen im Ergebnisbereich zwischen −65535 und +65535 rechnen kann, wobei die Summanden allerdings nur zwischen −32768 und +32767 liegen dürfen.

```
10 REM BASICVORSPANN FUER P21
20 INPUT A:IF A<0 THEN A=A+65536:REM 1. SUMMAND (ZWEIERKOMPLEMENT)
30 AH=INT(A/256):AL=A-256*AH
40 POKE &100,AL:POKE &101,AH
50 INPUT B:IF B<0 THEN B=B+65536:REM 2. SUMMAND (ZWEIERKOMPLEMENT)
60 BH=INT(B/256):BL=B-256*BH
70 POKE &102,BL:POKE &103,BH
80 CALL &4000
90 PRINT CHR$(PEEK(&100));:PRINT 256*PEEK(&103)+PEEK(&102)
```

```
4000                    1       ORG   #4000
4000   2A0201           2       LD    HL,(#0102)      ;1.SUMMAND NACH HL.
4003   ED4B0001         3       LD    BC,(#0100)      ;2.SUMMAND NACH BC.
4007   B7               4       OR    A               ;CARRY LOESCHEN.
4008   ED4A             5       ADC   HL,BC           ;ADDITION.
```

```
400A  FA1440        6          JP   M,L1          ;VORZEICHEN ABFRAGEN.
400D  EA1740        7          JP   PE,L2         ;V-FLAGGE ABFRAGEN.
4010  3E2B          8 L3:      LD   A,#2B         ;ASCII-CODE VON "+".
4012  180C          9          JR   L4            ;ZUR AUSGABE
4014  EA1040       10 L1:      JP   PE,L3         ;V-FLAGGE GESETZT ?
4017  7C           11 L2:      LD   A,H           ;WENN NEIN,DAS
4018  2F           12          CPL                ;EINERKOMPLEMENT
4019  67           13          LD   H,A           ;FUER JEDES BYTE
401A  7D           14          LD   A,L           ;DER SUMME BILDEN,
401B  2F           15          CPL                ;
401C  6F           16          LD   L,A           ;
401D  23           17          INC  HL            ;UND 1 ADDIEREN.DAS GIBT
                   18                             ;DAS ZWEIERKOMPLEMENT.
401E  3E2D         19          LD   A,#2D         ;ASCII-CODE VON
4020  320001       20 L4:      LD   (#0100),A     ;VORZEICHEN NACH #100.
4023  220201       21          LD   (#0102),HL    ;ERGEBNIS NACH #102,#103
4026  C9           22          RET                ;UND NACH BASIC.
                   23 ;P21

L1      4014  L2      4017  L3      4010
L4      4020
```

Im BASIC-Programm wird zunächst das 16-Bit-Zweierkomplement für negative Zahlen gebildet. Dazu wird der Betrag dieser Zahlen von der ersten Zahl subtrahiert, die nicht mehr in 16 Bit darstellbar ist. Das ist 2^{16}=65536. (Statt den Betrag zu subtrahieren, wird hier die negative Zahl addiert.) Das anschließende Zerlegen der Summanden in Low- und High-Byte ist ja aus dem vorhergehenden Kapitel bereits bekannt. Im Maschinenprogramm werden für das Laden der Register und für die Addition genau vier Befehle benötigt. Der ADC-Befehl muß hier verwendet werden, da ADD die V-Flagge nicht beeinflußt. Nach der Addition werden nacheinander die Vorzeichen- und Overflowflagge abgefragt. Die Abfrage der Vorzeichenflagge mit JP M,nn ist bereits bekannt. Die Befehle zur Abfrage der V-Flagge heißen

JP PE,nn und JP PO,nn

PE bedeutet, daß die V-Flagge gesetzt ist, PO bedeutet, daß sie gelöscht ist. Diese Bezeichnungen gehen darauf zurück, daß die V-Flagge bei Verschiebebefehlen als sog. Paritätsflagge verwendet wird. Genaueres darüber ist in den Zusatzinformationen dieses Kapitels zu erfahren.
Die Kombination der Verzweigungen wird folgendermaßen programmiert: Wenn die V-Flagge und die S-Flagge gleichzeitig gesetzt oder gelöscht sind (Plus und kein Overflow oder Minus und Overflow), ist die Zahl positiv. Das heißt, wenn die V-Flagge gesetzt ist, muß das durch die S-Flagge angezeigte Vorzeichen verändert werden.

S-Flagge \ V-Flagge	PE	PO
M	+	−
P	−	+

Tabelle 8 *Vorzeichenkorrektur bei Overflow.*

Bei einem positiven Resultat wird bei der Marke L3 des Programms einfach das „+"-Zeichen nach Adresse &100 gebracht; bei einem negativen Resultat muß hinter L2 zuerst das Zweierkomplement des Rechenergebnisses gebildet werden, um den Betrag der negativen Zahl zu erhalten. Dazu kann nicht der NEG-Befehl verwendet werden, weil dieser auf 8-Bit-Zahlen zugeschnitten ist. Eine Addition des Resultats zu 65536 (wie in BASIC) ist auch nicht ohne weiteres möglich, da 65536 keine 16-Bit-Zahl mehr ist. Daher wird ein anderer Weg beschritten: Das Zweierkomplement einer Dualzahl ergibt sich auch, wenn alle Nullen der Zahl durch Einsen, alle Einsen durch Nullen ersetzt werden und wenn zu dieser neuen Zahl anschließend 1 addiert wird. Das wird durch die beiden CPL-Befehle (vom englischen „complement") und das anschließende INC HL bewirkt.
Der für uns neue CPL-Befehl bildet dabei das (Einer-)Komplement des Akkumulators: Er dreht alle Bits um. Bis auf die Flaggenbeeinflussung hat er dieselbe Wirkung wie XOR &FF.

Zusammenfassung

Vorzeichen
Das höchstwertige Bit einer Zahl kann als Vorzeichen interpretiert werden. Dabei bedeutet eine Eins in diesem Bit ein Minuszeichen, entsprechend bedeutet Null ein Pluszeichen.

Zweierkomplement einer 8-Bit-Zahl
Bei der Bildung des Zweierkomplements bleibt eine positive Zahl unverändert. Das Zweierkomplement einer negativen 8-Bit-Zahl erhält man, indem man den Betrag der Zahl von 256 subtrahiert. Eine zweimalige Bildung des Zweierkomplements ergibt den Betrag des Ausgangswerts. Der Befehl NEG bildet das Zweierkomplement des im Akkumulator stehenden Betrags einer negativen 8-Bit-Zahl.

Zweierkomplement einer 16-Bit-Zahl
Beim Zweierkomplement einer negativen 16-Bit-Zahl muß der Betrag der Zahl von 65536 subtrahiert werden. Derselbe Effekt wird erreicht, wenn man das Low-Byte und das High-Byte in den Akkumulator bringt, dort mit dem Befehl CPL jeweils das Einerkomplement bildet und anschließend zum Gesamtergebnis die Zahl 1 addiert. Beim Einerkomplement wird der Wert jedes Bits in sein Gegenteil (Komplement) verkehrt.

Die S-Flagge
Das Vorzeichen einer Rechenoperation, d. h. der Wert des höchstwertigen Bits, wird durch die S-Flagge angezeigt. Entsprechend dem Wert der S-Flagge kann mit JP M,nn (bei gesetzter Flagge) bzw. mit JP P,nn (bei gelöschter Flagge) verzweigt werden.

Die V-Flagge
Eine Änderung des höchstwertigen Bits, die nicht als Vorzeichenänderung interpretiert werden darf, heißt Overflow (Überlauf) und wird durch das Setzen der V-Flagge angezeigt. Ein Overflow kann nur bei der Addition zweier Zahlen mit

gleichem Vorzeichen oder bei der Subtraktion zweier Zahlen mit entgegengesetztem Vorzeichen vorkommen. Entsprechend dem Wert der V-Flagge kann mit JP PE,nn (bei gesetzter Flagge) bzw. mit JP PO,nn (bei gelöschter Flagge) verzweigt werden.

Übung 32 Berechnen Sie die Bit-Darstellung des 8-Bit-Zweierkomplements von −120 und −10

a) durch Subtraktion des Betrags von 256 und anschließende Umwandlung in die Dualzahldarstellung,

b) durch Umwandlung des Betrags in die Dualzahldarstellung, Bildung des Einerkomplements und Addition von Eins.

Übung 33 Bei welchen der folgenden 8-Bit-Rechnungen wird die V-Flagge gesetzt?

22+15
60+100
100+36
90−100
−20−60
−40−110
−120−10

Übung 34 Können für die Abfragen im Programm P21 statt der Befehle JP M,nn und JP PE,nn auch die Verzweigungsbefehle JP P,nn und JP PO,nn verwendet werden?

Zusatzinformationen

Beeinflussung und Abfrage der S- und V-Flaggen. Die S- und V-Flaggen werden von allen 8-Bit-Arithmetik-Befehlen (einschließlich INC und DEC) und von den Logikbefehlen OR, AND, XOR beeinflußt. Von den 16-Bit-Rechenbefehlen beeinflussen dagegen nur ADC und SBC diese Flaggen.
Zur Abfrage der S- und der V-Flagge existieren keine relativen Sprungbefehle JR. Dagegen gibt es neben den JP-Verzweigungen noch bedingte RET- und CALL-Befehle, die den Zustand dieser Flaggen abfragen. So wird bei CALL PO,nn das Unterprogramm bei nn nur aufgerufen, falls die V-Flagge gelöscht ist, sonst wird der CALL-Befehl ignoriert. Eine entsprechende Wirkung haben die anderen bedingten Anweisungen, die in Tabelle 9 gezeigt werden.

RET M	RET P	RET PE	RET PO
CALL M,nn	CALL P,nn	CALL PE,nn	CALL PO,nn
JP M,nn	JP P,nn	JP PE,nn	JP PO,nn

Tabelle 9 *Verzweigungen mit der S- und der P/V-Flagge*

Die Paritätsflagge. Auch die meisten Schiebe- und Rotationsbefehle beeinflussen die S- und die V-Flagge. Während die S-Flagge den Wert des siebten Bits nach Ausführung des Befehls anzeigt, hat die V-Flagge hier eine ganz andere Bedeutung als bei den arithmetischen Befehlen. Sie wird gesetzt, wenn nach der Befehlsausführung eine gerade Zahl von Bits den Wert 1 besitzt (natürlich bedeutet das bei insgesamt acht Bits dann auch eine gerade Zahl von Nullen). Bei einer ungeraden Zahl von Einsen bzw. Nullen wird die Flagge zurückgesetzt. Man spricht im ersten Fall auch von einer geraden Parität („parity even") des Bytes, im zweiten Fall von einer ungeraden Parität („parity odd"). Die Flagge heißt in diesem Zusammenhang auch nicht mehr Overflow, sondern Paritätsflagge, so daß man insgesamt von der P/V-Flagge spricht. Eine Ausnahme machen hier die Befehle RRCA, RLCA, RRA, RLA, die weder die S- noch die P/V-Flagge beeinflussen. Das bedeutet z. B., daß die Befehle RRCA (Code &0F) und RRC A (Code &CB0F) zwar dieselbe Wirkung auf den Akkumulator und die C-Flagge haben (siehe Kapitel 12), aber die S- und P/V-Flagge unterschiedlich beeinflussen.

Die P/V-Flagge bei den Blockbefehlen. Noch eine andere Verwendung besitzt die P/V-Flagge bei den Blocktransfer- und Vergleichsbefehlen (siehe Kapitel 14). Solange der Bytezähler BC bei diesen Befehlen noch nicht den Wert Null erreicht hat, bleibt die P/V-Flagge gesetzt. Beim Erreichen des Werts Null wird sie zurückgesetzt.

Die letzten Flaggen und das Rechnen im Dezimalsystem 2

Bei allen bisherigen Rechnungen verwendete der Z80 das Dual- bzw. Hexadezimalsystem. Der Inhalt einer Speicherstelle oder eines 8-Bit-Registers kann bei dieser Arbeitsweise als zweistellige Hexadezimalzahl angesehen werden, wobei eine Ziffer in den Bits 0 bis 3 und die andere in den Bits 4 bis 7 gespeichert ist. Das entscheidende Kennzeichen der hexadezimalen Arithmetik ist, daß ein Übertrag von einer niederwertigen zu einer höherwertigen Ziffer erst beim Überschreiten des Werts &F=15 und nicht, wie im Dezimalsystem, schon beim Überschreiten des Werts 9 stattfindet.

Dezimalarithmetik mit DAA

Von der hexadezimalen Arithmetik haben wir in den vorhergehenden Kapiteln nur deshalb nichts bemerkt, weil bei der Ein- und Ausgabe mit Hilfe der Anweisungen POKE und PEEK der BASIC-Interpreter automatisch die Umrechnungen vom Dezimal- ins Hexadezimalsystem und umgekehrt durchgeführt hat. Die Betriebssystemsprogramme, die bei diesen Umrechnungen aufgerufen werden, sind kompliziert und recht umfangreich. Falls man direkt im Dezimalsystem rechnen und ohne diese Umrechnungsprogramme auskommen will, kann man auf einen bisher nicht besprochenen Befehl des Z80-Prozessors zurückgreifen, der es erlaubt, ihn auch als Dezimalrechner zu benutzen. Dieser Befehl heißt

DAA

was die Abkürzung der englischen Wörter „Decimal Adjust Accumulator" ist. Übersetzt heißt das soviel wie: „Stelle den Akkumulator auf das Dezimalsystem um." Die Wirkung des Befehls besteht darin, daß jetzt der Übertrag von der niederwertigen Ziffer des Akkumulators, die sich in Bit 0 bis Bit 3 befindet, zur höherwertigen Ziffer in Bit 4 bis Bit 7 schon beim Überschreiten des Werts 9 stattfindet. Man macht sich das am besten wieder mit Hilfe eines Beispielprogramms klar

(Programm P22). In diesem Programm soll gezählt werden, wie oft eine Zifferntaste auf der Tastatur gedrückt wurde. Der aktuelle Zählerstand soll jeweils auf dem Bildschirm angezeigt werden. Der Drücken der „Q"-Taste soll das Programm beenden.

```
4000                    1           ORG   #4000
4000  AF                2           XOR   A                ;AKKU UND ZAEHL-
4001  320001            3           LD    (#0100),A        ;SPEICHER LOESCHEN.
4004  CD06BB            4 L0:       CALL  #BB06            ;TASTATURABFRAGE.
4007  FE3A              5           CP    #3A              ;ASCII-CODE > &39 ?
4009  3023              6           JR    NC,L1            ;WENN JA,NACH L1.
400B  FE30              7           CP    #30              ;ASCII-CODE < #30 ?
400D  381F              8           JR    C,L1             ;WENN JA,NACH L1.
                        9                                  ;SONST ZIFFENTASTE
                        10                                 ;GEDRUECKT.
400F  3A0001            11          LD    A,(#0100)        ;ZAEHLSPEICHER NACH A.
4012  3C                12          INC   A                ;HOCHZAEHLEN.
4013  27                13          DAA                    ;DEZIMALABGLEICH.
4014  320001            14          LD    (#0100),A        ;NEUER STAND IN DEN
                        15                                 ;ZAEHLSPEICHER.
4017  CD6CBB            16          CALL  #BB6C            ;BILDSCHIRM LOESCHEN.
401A  AF                17          XOR   A                ;AKKU LOESCHEN.
401B  210001            18          LD    HL,#0100         ;ZEIGER AUF ZAEHL-
                        19                                 ;SPEICHER.
401E  ED6F              20          RLD                    ;1.ZIFFER IN DEN AKKU.
4020  F630              21          OR    #30              ;ASCII-CODE ERZEUGEN
4022  CD5ABB            22          CALL  #BB5A            ;UND AUSGEBEN.
4025  ED6F              23          RLD                    ;2.ZIFFER
4027  CD5ABB            24          CALL  #BB5A            ;AUSGEBEN.
402A  ED6F              25          RLD                    ;AUSGANGSBYTE WIEDER-
                        26                                 ;HERSTELLEN.
402C  18D6              27          JR    L0               ;NAECHSTE ABFRAGE.
402E  FE51              28 L1:      CP    #51              ;"Q"-TASTE ?
4030  20D2              29          JR    NZ,L0            ;WENN NEIN,WEITER.
4032  C9                30          RET                    ;SONST FERTIG.
                        31 ;P22

L0       4004   L1       402E
```

Als Speicher für den Tastendruckzähler wird im Programm die Speicherstelle &100 benutzt. Nachdem dieser Zähler am Programmanfang gelöscht wurde, wird mit dem Systemprogramm &BB06 die Tastatur abgefragt. Liegt der Code der gedrückten Taste nicht im Bereich der Zifferntasten, also zwischen &30 und &39, so wird noch die Q-Taste abgefragt, und das Programm ist entweder beendet oder es springt zum Abfragebefehl zurück. Falls jedoch eine Zifferntaste gedrückt wurde, wird der Zählspeicher um 1 erhöht. Danach sorgt der DAA-Befehl dafür, daß der Inhalt der Speicherstelle &100 als Dezimalzahl interpretiert werden kann. Das bedeutet insbesondere, daß Überträge von einer Stelle auf die nächsthöhere schon nach Überschreiten der Zahl 9 vorgenommen werden.
Der Inhalt der Speicherstelle &100 wird dann mit Hilfe des RLD-Befehls und der Systemroutine &BB5A ziffernweise ausgegeben. Wie schon in Kapitel 17 besprochen, muß dabei das HL-Registerpaar die Adresse der Speicherstelle enthalten, deren Inhalt mit dem RLD-Befehl in den Akkumulator gebracht werden soll. Vor jeder Ausgabe wird mit der Systemroutine &BB6C der Bildschirm gelöscht und der Cursor in die linke obere Ecke gebracht.
Um die Wirkung von DAA kennenzulernen, sollte man diesen Befehl einmal weglassen (oder durch den Leerbefehl NOP ersetzen): Zu Beginn des Zählvor-

gangs ist noch nichts zu merken; beim Überschreiten des Zählerstands „0[illegible] die Ausgabe jedoch unverständlich. Anstatt „10“ erscheint auf dem Bild[illegible] die Zeichenkombination „0:“. Danach kommen „0;“, dann „0‹“, und so weit[illegible] Grund ist einfach zu verstehen: Da nach „09“ kein Übertrag erzeugt wird, [illegible] dem die Ziffernkombination „0A“ folgt, werden bei der Ausgabe durch den [illegible] fehl OR &30 die ASCII-Codes &30 und &3A erzeugt, und diese entsprechen [illegible] Zeichen „0“ und „:“. Beim Weiterzählen ergeben sich dann die Codes von „;“ (&3B), „‹“ (&3C) usw., bis schließlich beim sechzehnten Tastendruck „10“ angezeigt wird.

Genaueres über DAA

Die Arbeitsweise des DAA-Befehls ist im einzelnen recht verwickelt. Der Befehl muß dafür sorgen, daß die sechs für den Dezimalbereich ungültigen Ziffern A, B, C, D, E und F übersprungen werden. Das wird bei der Addition dadurch erreicht, daß unter bestimmten Umständen zur niederwertigen Ziffer oder zur höherwertigen Ziffer oder zu beiden Ziffern eine 6 addiert wird. Wir wollen uns das an einigen Beispielen anschauen:

1. &08+&04=&0C (Rechnung ohne DAA)
   Der DAA-Befehl addiert zusätzlich &06: &0C+&06=&12 und erreicht dadurch die korrekte Ziffernfolge für die dezimale Rechnung:

   08+04=12 (Rechnung mit DAA)

2. &67+&75=&DC (Rechnung ohne DAA)
   Der DAA-Befehl addiert zusätzlich &66: &DC+&66=&142. Damit ergibt sich wieder die richtige dezimale Ziffernfolge:

   67+75=142 (Rechnung mit DAA)

   wobei die Hunderterstelle in der Carry-Flagge steht.

3. &61+&72=&D3 (Rechnung ohne DAA)
   Der DAA-Befehl addiert zusätzlich &60: &D3+&60=&133, und damit ergibt sich rein optisch die richtige Ziffernfolge:

   61+72=133 (Rechnung mit DAA)

4. &21+&32=&53 (Rechnung ohne DAA)
   Das Ergebnis ist auch im Dezimalsystem korrekt; der DAA-Befehl läßt das Ergebnis unverändert.

Bei Subtraktionen werden in den entsprechenden Fällen als Korrektur die Zweierkomplemente von &06, &60 oder &66 addiert.
Man sieht also zunächst, daß alle Rechnungen nach wie vor im Hexadezimalsy-

stem durchgeführt werden, daß aber durch die von DAA eingebrachten zusätzlichen Summanden (&06, &60, &66 oder deren Zweierkomplemente) Ziffernkombinationen erreicht werden, die sich bei einer Rechnung im Dezimalsystem ergeben würden.
Die Frage ist nun, woher der Prozessor die Information darüber erhält, welche der verschiedenen Korrekturen in einem konkreten Fall vom DAA-Befehl angebracht werden muß. Zum Zeitpunkt der Ausführung dieses Befehls stehen ja unter Umständen nicht mehr die Ausgangswerte, sondern nur noch das Endergebnis zur Verfügung.

Flaggen steuern die Dezimalkorrektur

Wenn wir uns an die Vorzeichenkorrektur erinnern, die wir im vorherigen Kapitel durchgeführt haben, können wir uns schon denken, daß auch für die Arbeit des DAA-Befehls Flaggen eine wesentliche Rolle spielen. Alle Korrekturen werden durch insgesamt drei Flaggen gesteuert. Da ist zunächst die uns bereits bekannte Carry-Flagge. Dazu kommen jetzt noch zwei bisher nicht erwähnte Flaggen: Das sind die

Halbcarry-Flagge (H-Flagge)

und die

Subtraktions-Flagge (N-Flagge)

Die H-Flagge reagiert auf einen Übertrag aus dem dritten Bit genauso wie die C-Flagge auf einen Übertrag aus dem siebten Bit. Bei 16-Bit-Rechenbefehlen zeigt die H-Flagge entsprechend einen Übertrag aus dem 11. Bit an. Die N-Flagge wird bei Subtraktionsbefehlen gesetzt und bei Additionsbefehlen zurückgesetzt. Beide Flaggen dienen im wesentlichen zur Überwachung der DAA-Korrekturen, haben keine eigenen Verzweigungsbefehle und sind daher von geringerer Bedeutung als die vier uns bereits bekannten Carry-, Zero-, S- und P/V-Flaggen. Wir werden am Ende dieses Kapitels sehen, daß es auf Umwegen doch möglich ist, die N- und die H-Flagge abzufragen. Auch damit kann dann in einem Programm entsprechend verzweigt werden. Für praktische Zwecke dürfte diese Möglichkeit allerdings kaum eine wesentliche Bedeutung haben.
Da die Korrekturbedingungen für den DAA-Befehl durch Flaggen gesteuert werden, muß man darauf achten, daß zwischen Rechenbefehlen und dem DAA-Befehl diese Flaggen nicht durch andere Befehle verändert werden. Genauso wichtig ist es, vor der Verwendung des DAA-Befehls im Zusammenhang mit INC- und DEC-Befehlen für den richtigen Wert der Carry-Flagge zu sorgen, da diese von INC und DEC nicht beeinflußt wird. Ein Beispiel dafür ist weiter unten im Programm P27 zu finden.

Zusammenfassung

DAA-Befehl, H-Flagge, N-Flagge
Der DAA-Befehl ermöglicht es, mit den arithmetischen Befehlen des Z80 im
malsystem zu rechnen. Vier Bits eines Bytes enthalten dann jeweils eine
des Zehnersystems, so daß ein Byte eine zweistellige Dezimalzahl spei
kann. Die vom DAA-Befehl vorgenommenen Korrekturen werden durch
C-Flagge, die H-Flagge (Halbcarry-Flagge) und die N-Flagge (Subtrakti
Flagge) gesteuert. Der Programmierer muß darauf achten, daß diese Flagg
vor der Korrektur nicht durch andere Befehle beeinflußt werden.

Übung 35 Vertauschen Sie im Programm P22 die Reihenfolge der Tasten
codeabfragen (CP &30 und CP &34) mitsamt den entsprechenden
Verzweigungsbefehlen. Überlegen Sie sich den Grund für die
Auswirkung dieses Tausches.

Übung 36 Warum kann im Programm P22 bei der Ausgabe der zweiten Ziffer der Befehl OR &30 zur Umwandlung in den ASCII-Code weggelassen werden?

Übung 37 Ändern Sie das Programm P22 so, daß das Drücken von Tasten gerader Zahlen den Zähler um Eins erhöht, während das Drücken von Tasten ungerader Zahlen den Zähler um Eins vermindert.

Zusatzinformationen

Speicherverbrauch bei Dezimalarithmetik. Die Bequemlichkeit, die die Benutzung des Dezimalsystems bietet, muß theoretisch mit einem gewissen Verlust an Speicherkapazität erkauft werden. Ein Byte kann im Dezimalsystem nur den Zahlenbereich von 0 bis 99 statt von 0 bis 255 darstellen. Um eine zehnziffrige Dezimalzahl im Bereich bis 2^{32} darzustellen, benötigt man im Dezimalmodus fünf Bytes, während in der üblichen hexadezimalen Darstellungsweise vier Bytes ausreichen.
Die Flaggen. Die Flaggen der Z80-CPU sind im F-Register in der in Bild 23 gezeigten Reihenfolge angeordnet.

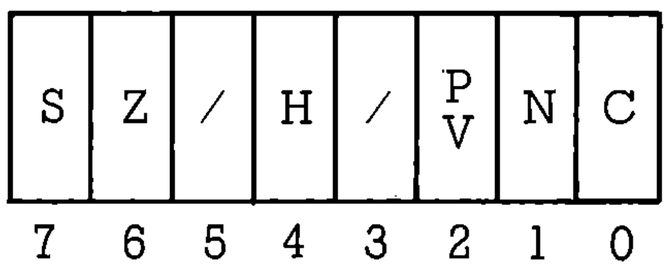


Bild 23 *Das F-Register.*

Die Bits 3 und 5 des F-Registers sind unbenutzt.

Zur Übersicht sollen nochmals alle Flaggen und die dazugehörigen Verzweigungsbefehle in Tabelle 10 zusammengestellt werden.

Flaggenbezeichnung	Abkürzung	Verzweigungsbedingungen	Verzweigungsbefehle
Vorzeichen-Flagge	S	M, P	JP, RET, CALL
Zero-Flagge	Z	Z, NZ	JP, JR, RET, CALL
Halbcarry-Flagge	H	—	—
Paritäts-Overflow-Flagge	P/V	PE, PO	JP, RET, CALL
Subtraktion-Flagge	N	—	—
Carry-Flagge	C	C, NC	JP, JR, RET, CALL

Tabelle 10 *Sämtliche Flaggen mit dazugehörigen Verzweigungsbefehlen. Bei der Zero-Flagge ist noch DJNZ als Spezialform des JR-Befehls zu beachten.*

Da ihre Positionen im F-Register bekannt sind, können die H- und die N-Flagge durch einen PUSH AF-Befehl mit einem anschließenden POP-Befehl isoliert und dann (z. B. mit einem BIT-Befehl) abgefragt werden.
Beispiel: Abfrage der N-Flagge mit

```
PUSH AF
POP BC
BIT 1,C
JR NZ,nn
```

Die Befehlssequenz führt zu einem Sprung nach nn, falls die N-Flagge gesetzt ist.

Im CPC steckt Musik

21

Bis zu dieser Stelle haben wir nur Befehle benutzt, die Daten aus dem Speicher des Computers geholt, sie bearbeitet und sie dann wieder in den Speicher zurückgeschrieben haben. Es ist dabei nicht ganz klar geworden, wie Daten in den Computer hinein- oder aus ihm herausgelangen. Für die Eingabe haben wir die Systemroutinen &BB06 bzw. &BB09 benutzt. Was passiert aber beim Aufruf dieser Routinen? Für die Ausgabe haben wir bisher nur den Bildschirm verwendet. Dabei wurden Informationen im Speicherbereich &C000 bis &FFFF abgelegt, von da aus gelangen sie dann mit einer speziellen elektronischen Schaltung, dem Video-Controller, auf den Bildschirm des Monitors. Diese Art der Datenausgabe, d. h. ihre Übergabe in einem bestimmten Speicherbereich an einen anderen Baustein, ist aber nur eine Ausgabemöglichkeit des Z80-Prozessors. Er besitzt darüber hinaus noch spezielle Ein- und Ausgabebefehle, deren Verwendung wir uns im Zusammenhang mit der Programmierung des Tongenerators jetzt klarmachen wollen.
Hier sollen keine Mißverständnisse entstehen: Es geht in diesem Kapitel nicht darum, die guten Programmiermöglichkeiten des Tongenerators, die in BASIC- und Betriebssystemsroutinen bereits vorhanden sind, zu verbessern. Der Tongenerator dient hier nur als Beispiel für die Anwendung der Ein/Ausgabebefehle des Z80.

Datenaustausch mit Peripheriegeräten

Um die 16-Bit-Adressen des Speichers verwalten zu können, besitzt der Z80-Prozessor 16 Adreßleitungen, die im allgemeinen mit A0 bis A15 bezeichnet werden. Zusätzlich zu diesen Adreßleitungen gibt es noch Leitungen, die die Namen MREQ und IORQ haben. Mit Hilfe dieser beiden Leitungen kann der Prozessor seine Ausgabe dirigieren: Ein Signal auf der MREQ (Memory Request)-Leitung zeigt an, daß die Adresse den Speicher (memory) betrifft. Das ist der bisher von

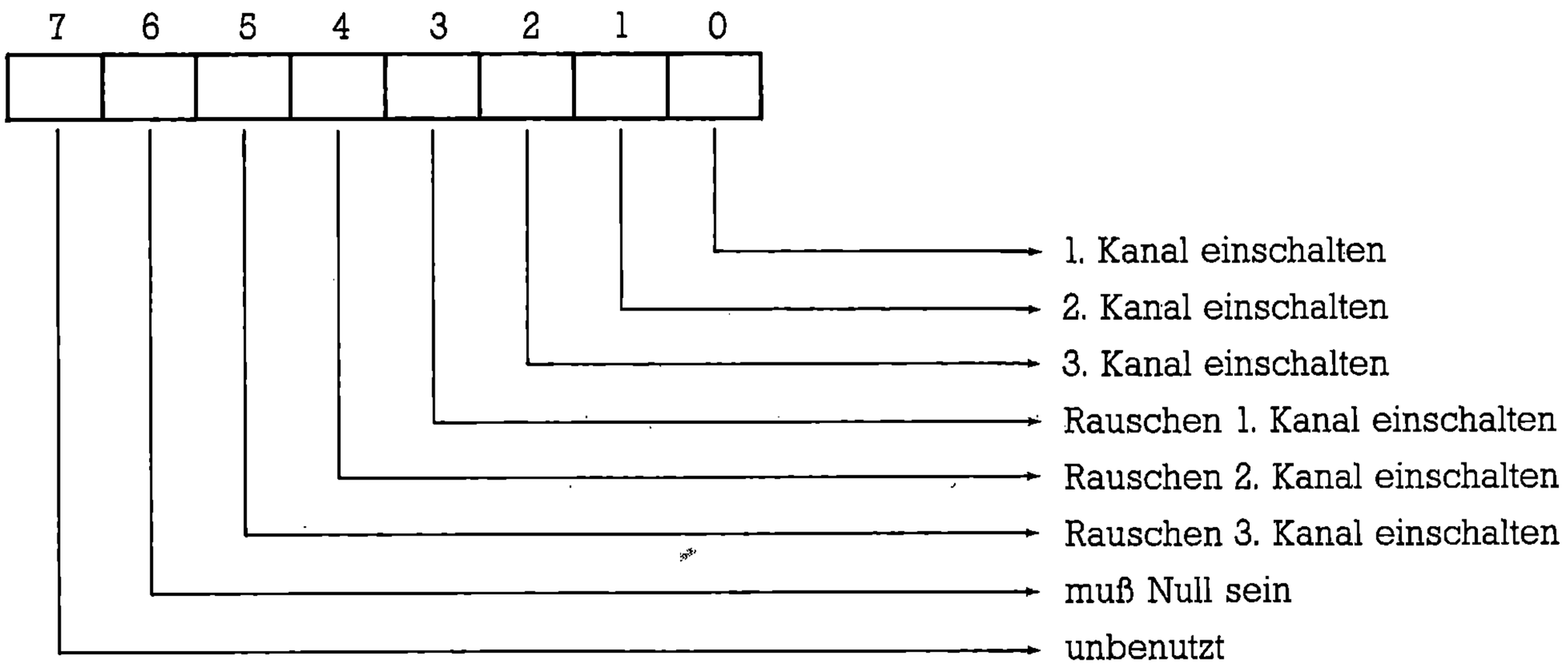
uns betrachtete Fall. Ein Signal auf der IORQ (Input/Output Request)-Leitung zeigt an, daß die Adresse ein Ein/Ausgabegerät betrifft. Das kann nun alles mögliche sein: die Tastatur, ein Drucker, der Rekorder und eben auch der Tongenerator. Peripheriegeräte werden also genauso adressiert wie der Speicher. Die Ein/Ausgabeadressen werden „Ports" genannt. Die Umschaltung von der MREQ- auf die IORQ-Leitung erfolgt durch Ein- und Ausgabebefehle, die IN und OUT heißen und im folgenden näher beschrieben werden sollen.
Der Tongenerator mit der Typenbezeichnung AY-3-8912 ist nicht direkt an den Prozessor angeschlossen. Vermittler ist ein Ein/Ausgabebaustein mit der Typenbezeichnung 8255. Die Daten für den Tongenerator müssen zuerst an den 8255 übergeben werden, und der gibt sie an den Tongenerator weiter. Der Vorteil dieser etwas umständlichen Methode besteht darin, daß so nicht für jedes der 13 Register des Tongenerators ein eigener Port zur Verfügung gestellt werden muß. Vielmehr verläuft die gesamte Kommunikation über zwei Ports des 8255, die Port A und Port C genannt werden. Über den Port A werden Daten an den Tongenerator übergeben, der Port C überträgt eine Kennung, aus der zu ersehen ist, ob die Daten des Ports A als Nummer eines Tonregisters oder als Inhalt für die Register interpretiert werden müssen. Ein wenig kompliziert wird die ganze Angelegenheit noch dadurch, daß die Ports des 8255 gleichzeitig noch für andere Aufgaben, wie die Überwachung der Tastatur und des Kassettengeräts, herangezogen werden.

Die Register des Tongenerators

Um den Tongenerator programmieren zu können, muß man natürlich zuerst die Funktionen seiner 13 Register kennen. Bekanntlich ist es beim CPC möglich, drei unabhängige Stimmen oder „Kanäle" zu programmieren. Die entsprechenden Tonhöhen erhält man, indem man eine im CPC erzeugte Grundfrequenz von 62500 Hz durch eine Zahl im Bereich zwischen 1 und 4095 teilt. Dieser Teiler wird für die erste Stimme in der üblichen L-Byte-, H-Byte-Reihenfolge in den Registern 0 und 1 des Tongenerators angegeben. Die Register 2 und 3 bzw. 4 und 5 enthalten dieselbe Tonhöheninformation für die zweite und dritte Stimme. Da man zur Darstellung der Zahl 4095 nur 12 Bits benötigt, sind die oberen vier Bits der jeweiligen H-Bytes bedeutungslos.
Die Lautstärke der drei Stimmen wird durch die Register 8, 9 und 10 bestimmt. Dabei entspricht der Registerinhalt 15 der maximalen Lautstärke. Ist jedoch das vierte Bit des Lautstärkeregisters gesetzt (z. B. Inhalt = 16), dann wird die Lautstärke über die in Register 13 anzugebende „Hüllkurvennummer" geregelt. Die Hüllkurven sind teils sog. Sägezahnschwingungen, teils sog. Dreiecksschwingungen, teils erzeugen sie ein einmaliges An- und Abschwellen mit anschließendem Halten des Tons. Die Hüllkurvennummern gehen dabei von 1 bis 15. Die Wirkungen probiert man am besten selbst aus. Die Hüllkurvennummer 14 er-

Nr.	Funktion	
0	Tonperiode des 1. Kanals (L-Byte)	
1	Tonperiode des 1. Kanals (H-Byte, nur Bits 0 bis 3)	
2, 3	Wie Register 0, 1, für den 2. Kanal.	
4, 5	Wie Register 0, 1, für den 3. Kanal.	
6	Rauschperiode (nur Bits 0 bis 5).	
7	Schaltregister (s. unten).	
8	Lautstärke und Hüllkurvenfreischaltung für den 1. Kanal	} (s. unten)
9	Wie Register 8, für den 2. Kanal.	
10	Wie Register 8, für den 3. Kanal.	
11	Hüllkurven-Periode (L-Byte)	
12	Hüllkurven-Periode (H-Byte)	
13	Hüllkurvenform (Bits 0 bis 3).	

Schaltregister 7: Bitwert „0" bedeutet Schalter „ein"



Register 8, 9, 10

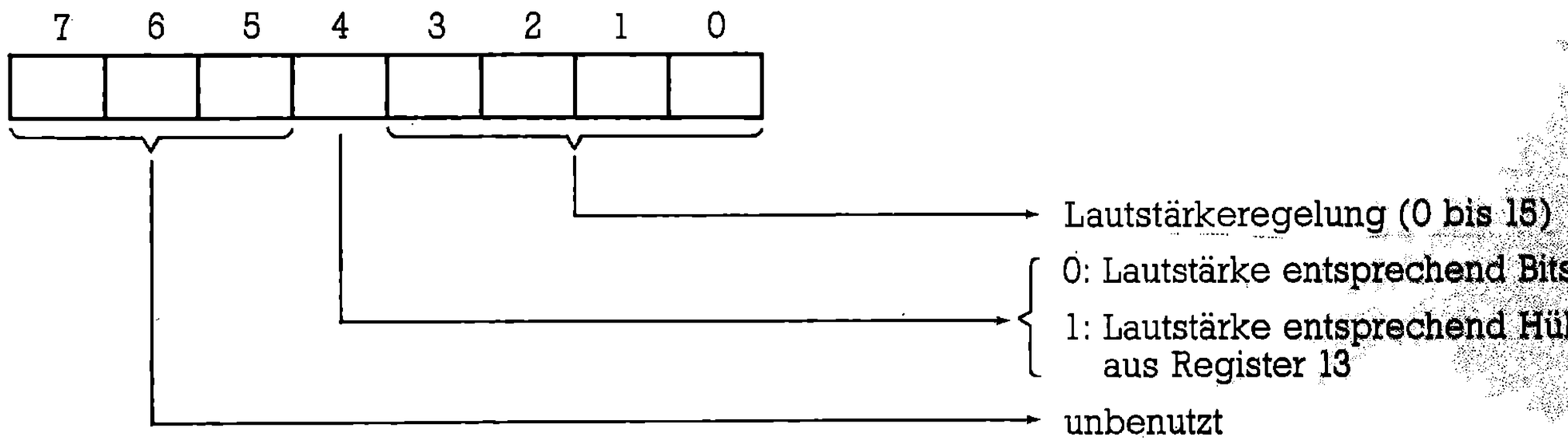


Tabelle 11 *Register des Tongenerators.*

zeugt z. B. ein deutlich hörbares, gleichmäßiges An- und Abschwellen des Tons, das sich dauernd wiederholt (Dreiecksschwingung). Die Hüllkurvennummer 13 erzeugt ein einmaliges Anschwellen mit anschließendem Halten des Tons, und die Hüllkurvennummer 12 erzeugt ein sich dauernd wiederholendes Anschwellen mit plötzlichem Abbrechen des Tons (Sägezahnschwingung). Die Periodendauer der Hüllkurve wird mit den Registern 11 (L-Byte) und 12 (H-Byte) geregelt. Auch hier ergibt eine große Zahl eine kleine Frequenz der Hüllkurve. Die hier beschriebenen Hüllkurven können für die drei Stimmen nicht getrennt eingestellt werden. Daher verfügt der CPC zusätzlich über programmierte Hüllkurven, die über den BASIC-Interpreter angesprochen werden können.
Im Register 7 des Tongenerators haben die einzelnen Bits Schalterfunktionen. Dabei bedeutet der Bit-Wert Null immer die Schalterstellung „Ein". Die Bits 0, 1, 2 schalten die drei Stimmen in der entsprechenden Reihenfolge ein. Die Bits 3, 4, 5 schalten den Rauschgenerator zu den drei Stimmen hinzu. Die Bits 6 und 7 sollten bei der Musikerzeugung den Wert 0 haben.
Schließlich bestimmt das Register 6 in der bereits bekannten Weise die „Ton"höhe des Rauschgenerators. Die Tonhöhenwerte werden durch die Bits 0 bis 5 angegeben. Tabelle 11 zeigt eine Übersicht aller Register des Tongenerators.
Um alle diese Register mit zwei Portadressen bedienen zu können, besitzt der Tongenerator noch eine Art 2-Bit-Register (eigentlich sind es zwei Anschlüsse des Tongenerators 8912). Diese beiden Leitungen sind mit den Bits 6 und 7 von Port C (Adresse &F600) des 8255-Bausteins verbunden. Sind beide Bits gesetzt (z. B. Registerwert &C0=X11000000), so bedeutet das, daß der Wert am Dateneingang des Tongenerators als Registernummer zu interpretieren ist und daß dieses Register jetzt sende- bzw. empfangsbereit geschaltet ist. Ist das siebte Bit ge-

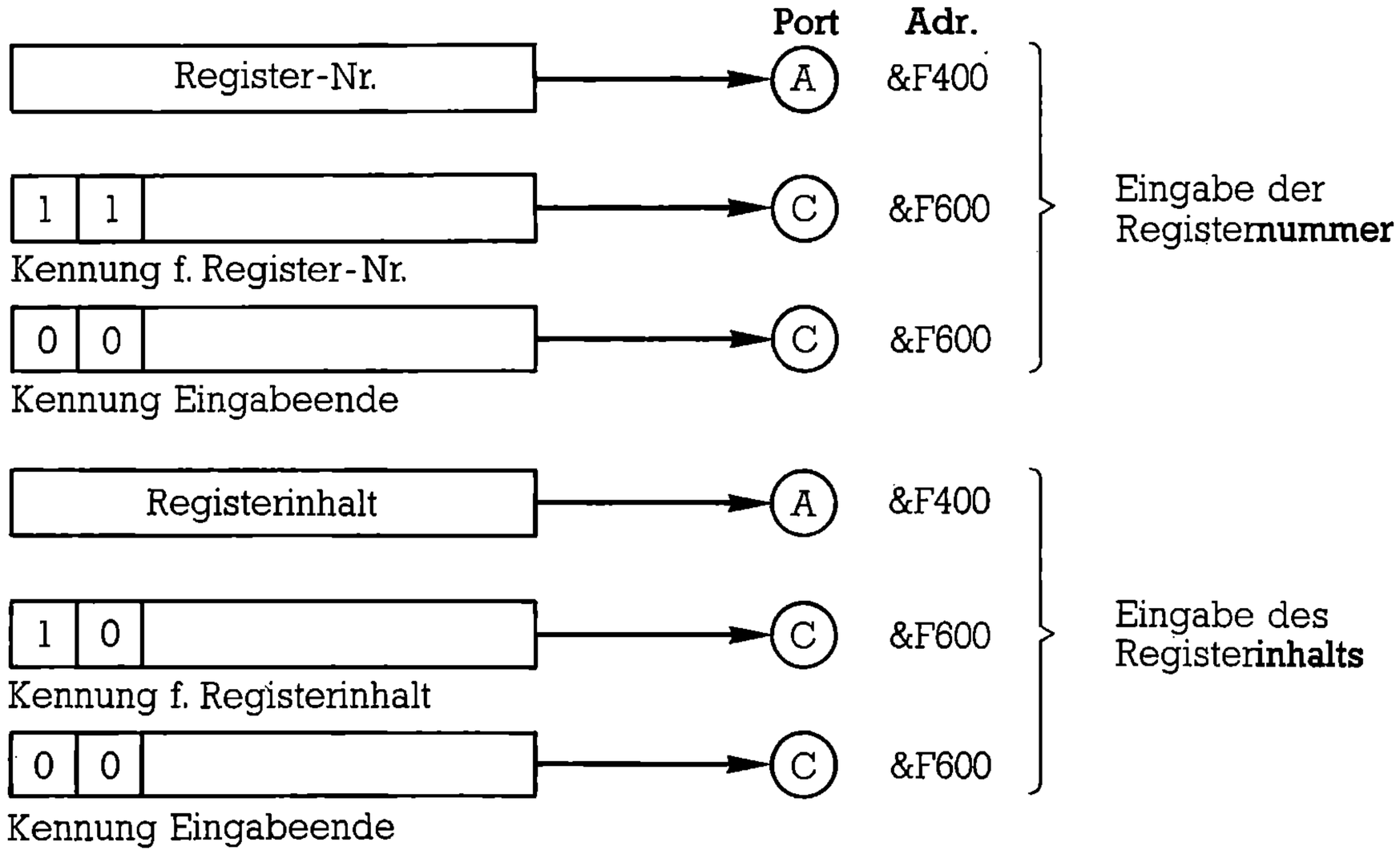


Bild 24 *Sechs Ausgabebefehle der Z80-CPU an die Ports des 8255-Bausteins sind notwendig, um ein Register des Tongenerators mit einem Wert zu versorgen.*

setzt und das sechste gelöscht (z. B. Registerwert X10000000), wird der Wert am Dateneingang in das angewählte Register geschrieben. Ist das siebte Bit gelöscht und das sechste gesetzt (z. B. Registerwert X01000000), so kann aus dem angewählten Register gelesen werden, und wenn beide Bits den Wert Null haben, dann ist der Ein- oder Ausgabevorgang abgeschlossen. Der eben erwähnte Dateneingang des Tongenerators ist mit dem Port A des 8255 verbunden, der die Adresse &F400 hat. Die Ausgabe eines Werts an ein Register des Tongenerators ist in Bild 24 schematisch dargestellt.

Programmierung des Tongenerators

Um den Tongenerator zu programmieren, müssen wir jetzt nur noch die entsprechenden Maschinenbefehle kennenlernen. Der wichtigste Ausgabebefehl heißt

OUT (C),r

r ist dabei eines der 8-Bit-Allzweckregister A,..., L und (C) ist eine Abkürzung für (BC) und deutet auf einen durch BC indirekt adressierten Port hin. OUT (C),r hat fast dieselbe Wirkung wie LD (BC),r. In beiden Befehlen wird der Inhalt von r zur Adresse, die im BC angegeben ist, geschickt. Nur ist beim OUT-Befehl nicht die MREQ-, sondern die IORQ-Leitung aktiviert. Und daher ist der Empfänger nicht die Speicherstelle, sondern der Port mit der in BC angegebenen Adresse. (Um Verwirrungen vorzubeugen: Der Befehl LD (BC),r existiert nur für r=A, aber der Vergleich erleichtert hier vielleicht das Verständnis des OUT-Befehls.)
Das Gegenstück zum OUT-Befehl ist der Befehl

IN r,(C)

mit einer Wirkung, die derjenigen des Befehls LD r,(BC) entspricht. Auch hier ist wieder die IORQ-Leitung aktiviert, so daß das Register r nicht mit dem Inhalt einer Speicherstelle, sondern mit dem Inhalt des durch BC adressierten Ports geladen wird.
Bei der Programmierung des Tongenerators sind noch folgende Punkte zusätzlich zu beachten: Für die Adressierung der 8255-Ports wird das Low-Byte der Adresse überhaupt nicht verwendet. Es reicht also aus, vor den OUT-Befehlen jeweils nur das High-Byte der Adresse ins B-Register zu laden. Außerdem werden (wie schon im Abschnitt über die Register des Tongenerators besprochen) in Port C nur die Bits 6 und 7 zur Steuerung des Tongenerators benutzt, die anderen Bits dieses Ports überwachen die Tastatur und das Kassettengerät. Man muß daher mit entsprechenden logischen Verknüpfungen dafür sorgen, daß diese anderen Bits nicht geändert werden. Da das Laden der Tonregister immer nach demselben Schema abläuft (siehe Bild 24), ist es natürlich sinnvoll, den Ladevorgang als Unterprogramm ablaufen zu lassen und die aktuellen Werte (das sind Registernummer und -inhalt) aus dem Hauptprogramm zu übernehmen.

Ein Tonleiterprogramm

Ein Programm zum Abspielen der Tonleiter könnte dann folgendermaßen aussehen:

```
4000                    1           ORG  #4000
4000  CDA7BC            2           CALL #BCA7              ;TONGENERATOR IN AUS-
                        3                                   ;GANGSZUSTAND BRINGEN.
4003  DD216140          4           LD   IX,MUDA            ;ZEIGER AUF TONDATEN.
4007  3E08              5           LD   A,#08              ;LAUTSTAERKEREGISTER
4009  0E0F              6           LD   C,#0F              ;AUF MAXIMUM.
400B  CD4040            7           CALL TGL                ;REGISTEREINGABE.
400E  3E07              8           LD   A,#07              ;ERSTE STIMME
4010  0E3E              9           LD   C,#3E              ;EINSCHALTEN.
4012  CD4040           10           CALL TGL
4015  3E00             11 L1:       LD   A,#00              ;LOW-TONHOEHENREG.
4017  DD4E00           12           LD   C,(IX+#00)         ;LADEN.
401A  CD4040           13           CALL TGL
401D  DD4E01           14           LD   C,(IX+#01)         ;DATEN FUER TONHOEHE
4020  79               15           LD   A,C                ;MIT ENDMARKE
4021  FEFF             16           CP   #FF                ;VERGLEICHEN,
4023  280E             17           JR   Z,L2               ;BEI ENDMARKE FERTIG.
4025  3E01             18           LD   A,#01              ;SONST HIGH-TONHOEHEN-
4027  CD4040           19           CALL TGL                ;REGISTER LADEN.
402A  DD23             20           INC  IX                 ;ZEIGER AUF
402C  DD23             21           INC  IX                 ;NAECHSTEN TON.
402E  CD3740           22           CALL WARTE              ;WARTESCHLEIFE.
4031  18E2             23           JR   L1                 ;NAECHSTE TONEINGABE.
4033  CDA7BC           24 L2:       CALL #BCA7              ;ZUM SCHLUSS TON-
4036  C9               25           RET                     ;GENERATOR ABSCHALTEN.
                       26 ;
4037  010080           27 WARTE:    LD   BC,#8000           ;WARTESCHLEIFE.
403A  0B               28 L3:       DEC  BC
403B  78               29           LD   A,B
403C  B1               30           OR   C
403D  20FB             31           JR   NZ,L3
403F  C9               32           RET
                       33 ;
4040  F3               34 TGL:      DI                      ;BEIM LADEN DES TONGEN.
                       35                                   ;SYSTEMINTERRUPT AUS.
4041  06F4             36           LD   B,#F4              ;DATENPORT MIT
4043  ED79             37           OUT  (C),A              ;REGISTERNUMMER LADEN.
4045  06F6             38           LD   B,#F6              ;STEUERPORT
4047  ED78             39           IN   A,(C)              ;LESEN.
4049  F6C0             40           OR   #C0                ;KENNUNG FUER REG.NUMMER
404B  ED79             41           OUT  (C),A              ;AN STEUERPORT.
404D  E63F             42           AND  #3F                ;KENNUNG FUER EINGABE-
404F  ED79             43           OUT  (C),A              ;ENDE AN STEUERPORT.
4051  06F4             44           LD   B,#F4              ;DATENPORT LADEN MIT
4053  ED49             45           OUT  (C),C              ;REGISTERINHALT.
4055  06F6             46           LD   B,#F6              ;STEUERPORT MIT
4057  F680             47           OR   #80                ;KENNUNG FUER REG.INHALT
4059  ED79             48           OUT  (C),A              ;LADEN.
405B  E63F             49           AND  #3F                ;KENNUNG FUER ENDE
405D  ED79             50           OUT  (C),A              ;AN STEUERPORT.
405F  FB               51           EI                      ;SYSTEMINTERRUPT EIN.
4060  C9               52           RET                     ;ZUM HAUPTPROGRAMM.
4061  8E007F00         53 MUDA:     DEFB #8E,#00,#7F,#00    ;TONDATENTABELLE.
4065  71006A00         54           DEFB #71,#00,#6A,#00
```

```
4069   5F005400      55            DEFB #5F,#00,#54,#00
406D   4B004700      56            DEFB #4B,#00,#47,#00
4071   00FF          57            DEFB #00,#FF
                     58 ;P23

L1       4015   LZ       4033   L3       403A
MUDA     4061   TGL      4040   WARTE    4037
```

Da das Laden der Tonregister eine recht umständliche Sache ist und für jeden Ton vier Register geladen werden müssen, ist das Programm ziemlich lang geworden. Der eigentliche Ladevorgang wird im Unterprogramm TGL durchgeführt und entspricht den Angaben in Bild 24.
Die Befehle DI und EI werden wir im übernächsten Kapitel besprechen. Sie können auch weggelassen werden; man riskiert dann aber, daß ab und zu ein Ladevorgang nicht korrekt durchgeführt wird, weil er durch den Systeminterrupt (übernächstes Kapitel) unterbrochen wird. Zunächst einmal wird das High-Byte &F4 der Adresse des Ports A ins B-Register geladen; das Low-Byte der Adresse wird nicht benötigt. Dann wird mit dem ersten OUT-Befehl die gewünschte Registernummer des Tongenerators aus dem Akkumulator an den Port A übergeben. Um dem Tongenerator mitzuteilen, daß die anliegenden Daten eine Registernummer bedeuten, müssen Bit 6 und 7 in Port C (Adresse &F6xx) gesetzt werden. Damit die Werte der anderen Bits dieses Ports, die unter anderem für die Tastatur und den Kassettenrekorder zuständig sind, nicht geändert werden, wird zuerst mit dem IN-Befehl der Inhalt von Port C in den Akkumulator gebracht, dort werden mit OR &C0 die erforderlichen Bits 7 und 6 gesetzt, und dann geht's zurück in den Port C. Der Ladevorgang muß unbedingt damit abgeschlossen werden, daß anschließend die Bits 7 und 6 in Port C gelöscht werden. Da der Portinhalt noch im Akkumulator steht, erledigen die logische Verknüpfung mit AND &3F und der anschließende OUT-Befehl diese Aufgabe. Damit ist die erste Hälfte des Ladevorgangs abgeschlossen; der Tongenerator weiß jetzt, in welches Register die im Port A zu übergebenden Daten geladen werden sollen.
Die eigentliche Dateneingabe erfolgt dann fast genauso wie die Eingabe der Registernummer. Nur der IN-Befehl fehlt jetzt, da der Akkumulator die zu rettenden Bits des Steuerports C ja schon enthält; außerdem muß als Kennung dafür, daß jetzt ein Registerinhalt übertragen wird, das siebte Bit in Port C gesetzt werden, während das sechste Bit gelöscht sein muß. Die entsprechende logische Verknüpfung wird mit OR &80 durchgeführt.
Im Hauptprogramm wird zu Beginn und am Ende die Systemroutine &BCA7 aufgerufen, die den Ausgangszustand des Tongenerators herstellt. Das IX-Register wird als Zeiger auf die Tonhöhendatentabelle MUDA verwendet, deren Zustandekommen am Ende des Kapitels kurz erklärt wird. Im übrigen werden abwechselnd der Akkumulator mit der Registernummer des Tongenerators und das C-Register mit dem zu übertragenden Inhalt geladen, und dann wird das Ladeprogramm TGL aufgerufen. Im Warteprogramm wird lediglich Zeit verbraucht. Die Dauer des Tons kann über den Anfangswert von BC beeinflußt werden.
Das Ende der Musikdaten wird durch den Wert &FF im High-Byte der Tonhöhe signalisiert. Da beim High-Byte nur die niederwertigen vier Bits benutzt werden, ist keine Verwechslung mit echten Tönen möglich.

Zusammenfassung

IN- und OUT-Befehle
Die Befehle

IN r,(C)
OUT (C),r

sind Datentransportbefehle, wobei (C) eigentlich eine Abkürzung für (BC) darstellt. Anders als bei den LD-Befehlen werden die Daten aber nicht zwischen Speicherstellen und den Registern der CPU transportiert, sondern zwischen einem durch (BC) indirekt adressierten Ein/Ausgabeport und einem durch r angegebenen 8-Bit-Allzweckregister A, B, C, D, E, H, L. Mit Hilfe der IN- und OUT-Befehle kann die CPU Daten mit externen Geräten (Drucker, Tongenerator, Kassettenrekorder usw.) austauschen.

Übung 38 Mit welchen Befehlen wird
a) der Port &7F00 ins E-Register ausgelesen?
b) der Port &EF00 mit dem ASCII-Code von „1" geladen?

Übung 39 Schreiben Sie das Programm P23 so um, daß die Töne der Tonleiter beim Drücken der Tasten „0" bis „7" ausgegeben werden.

Zusatzinformationen

Weitere Ein/Ausgabebefehle. Es wurde oben schon erwähnt, daß für die Adressierung der Ports beim CPC nur das High-Byte verwendet wird. Bei den meisten anderen Z80-Systemen werden die Ports nur über das Low-Byte adressiert, wie es auch der Spezifikation der Fa. Zilog für den Z-80-Prozessor entspricht. Bei Schneider weicht man hier sehr stark vom „Üblichen" ab. Die folgenden Ein/Ausgabebefehle sind speziell für diese Adressierungsweise konzipiert und können auf dem CPC nicht sinnvoll eingesetzt werden.
Zusätzlich zu den besprochenen Ein/Ausgabebefehlen gibt es noch die Befehle

OUT (n),A

und

IN A,(n)

Hier entspricht der Inhalt von A sowohl dem Datenbyte als auch dem High-Byte der Adresse, während n das Adreß-Low-Byte angibt.
Die Blockausgabe-Befehle

OUTI

und

OUTD

haben die Struktur OUT (C),(HL) mit anschließendem INC HL (bzw. DEC HL) und DEC B. Bei den Befehlen

OTIR

und

OTDR

wird der Befehl so lange wiederholt, bis B den Inhalt Null hat.
Entsprechende Eingabebefehle sind

INI
INIR
IND
INDR

Alle diese Befehle sind nur dann gut zu verwenden, wenn die Ports mit dem Adreß-Low-Byte adressiert werden. Sie sollten beim CPC nicht verwendet werden.

Portadressen. Einige weitere Portadressen für andere Bausteine des CPC sind:

&F5xx	Port B des 8255
&EFxx	Centronics-Schnittstelle
&BCxx	Videocontroller-Adressenregister

(Hier kann eines der Register 0 bis 17 des Videokontrollers angewählt werden.)

&BDxx	Eingabe-Port für das angewählte Videoregister (Schreiben)
&BFxx	Ausgabe-Port für das angewählte Videoregister (Lesen)
&7Fxx	Port für das Gate Array (siehe nächstes Kapitel)

Das mit xx angegebene Low-Byte der Portadressen hat keine Bedeutung und kann beliebige Werte haben.

Tonhöhendaten. Die Tonhöhendaten wurden folgendermaßen berechnet: Die Grundfrequenz sollte 440 Hz sein. Um 440 Hz zu erhalten, muß die Ausgangsfrequenz 62500 Hz durch 142 dividiert werden (Low-Byte=142, High-Byte=0). Die Frequenz eines Tons der chromatischen Tonleiter liegt um den Faktor 1,05946 höher als die Frequenz des vorausgegangenen Tons. Bei einem Unterschied von einem ganzen Ton ist das Frequenzverhältnis dann $1{,}05946^2 = 1{,}12246$. Wenn die Frequenz größer wird, muß die Teilerzahl um denselben Faktor kleiner werden. Demnach sind die Daten des zweiten Tons der Tonleiter 142/1,12246=127 (gerundet) für das Low-Byte und 0 für das High-Byte. Die weiteren Daten werden entsprechend berechnet.

Zugriff aufs ROM

22

In allen bisherigen Kapiteln haben wir uns nur mit dem RAM des CPC beschäftigt. Das ist der Speicher, der vom Benutzer zum Abspeichern seiner Programme und Daten verwendet werden kann. Das vom Hersteller des CPC mitgelieferte Betriebssystem und der BASIC-Interpreter sind dagegen im ROM abgespeichert. ROM heißt „Read Only Memory". Das bedeutet, daß der Inhalt dieses Speichers nur gelesen, aber nicht verändert werden kann. Dies hat den großen Vorteil, daß der Inhalt des ROM auch beim Abschalten des Computers nicht verlorengeht, und daß daher nach dem Einschalten das Betriebssystem und der BASIC-Interpreter sofort zur Verfügung stehen und nicht erst geladen werden müssen.

Das Gate-Array organisiert den CPC

Das RAM des CPC464/664 hat einen Umfang von 65536 Bytes (131072 Bytes beim CPC6128), das ROM besitzt nochmals 32768 Bytes (49152 Bytes beim CPC664/6128), und damit ergibt sich das Problem, diese Bytes im Adreßbereich des Z80-Prozessors von 0 bis 65535 unterzubringen. Die Lösung haben Sie bereits im Speicherschema des Kapitels 4 gesehen: ROM und RAM belegen denselben Adreßbereich, nur sind sie in verschiedenen Etagen untergebracht: Man kann sagen, daß das Betriebssystem die erste Etage des Bereichs von &0000 bis &3FFF belegt und daß der BASIC-Interpreter in der ersten Etage des Bereichs von &C000 bis &FFFF untergebracht ist. Im Erdgeschoß befindet sich jeweils das RAM. Es muß also eine Möglichkeit geben, der CPU mitzuteilen, welche Etage der angegebenen Adresse gerade gemeint ist.
Betrachten wir ein konkretes Beispiel: Der Zeichengenerator (also die Bitmuster aller im CPC verfügbaren Zeichen) ist im ROM von Adresse &3800 bis &3FFF abgespeichert. Wie ist es möglich, diese Zeichen aus dem ROM auszulesen und auf den Bildschirm zu bringen? Falls wir versuchen, mit dem Zeichensetzprogramm

aus Kapitel 9 den Bereich ab &3800 auszulesen, erhalten wir einen leeren Bildschirm. Die CPU greift, wenn keine besonderen Vorkehrungen getroffen werden, auf das Erdgeschoß des Speicherbereichs (also das RAM) zu.
Der „Portier", der den Zugang zur ROM-Etage freigibt, ist das sog. Gate Array (das im folgenden mit GA abgekürzt werden soll). Das ist ein speziell für den CPC entwickelter Baustein, der eine ganze Reihe von Verwaltungsaufgaben erledigt. Unter anderem regelt er auch den Zugang zu den Speicheretagen. Die CPU spricht das GA über die Portadresse &7Fxx an. (Wie bei den anderen Ports des CPC wird auch hier nur das H-Byte zur Adressierung benutzt). Um dem GA mitzuteilen, daß ihm eine Nachricht über die Wahl der Speicheretagen übermittelt werden soll, müssen Bit 7 des Ports den Wert 1 und Bit 6 den Wert 0 erhalten. Bit 5 hat in diesem Zusammenhang keine Bedeutung, Bit 4 beeinflußt den Systeminterrupt. Den Zugang zum ROM regeln die Bits 3 und 2. Hat Bit 3 den Wert 0, so ist das obere ROM (&C000 bis &FFFF) eingeschaltet. Hat Bit 2 den Wert 0, so ist das untere ROM (&0000 bis &3FFF) eingeschaltet. Mit diesen beiden Bits können also die oberen Etagen des Speicherbereichs freigegeben werden. Die Bits 1 und 0 schließlich bestimmen den Bildschirm-Mode. Für Mode 2 muß Bit 1 gesetzt und Bit 0 gelöscht sein. Es ist jedoch nicht ohne weiteres möglich, den Graphikmodus durch Setzen dieser beiden Bits zu ändern, da das Betriebssystem diese Werte kontrolliert und eventuell auf den aktuellen Stand zurücksetzt.

Bit	Funktion
7 6	1 0 } Kennung für eine Mitteilung über die Speicherorganisation
5	0 (reserviert)
4	0 (löscht mit „1" einen Zähler für Interruptzwecke)
3	Oberes ROM bei „1" gesperrt, bei „0" freigeschaltet.
2	Unteres ROM bei „1" gesperrt, bei „0" freigeschaltet.
1 0	1 0 } Mode 2 — 0 1 } Mode 1 — 0 0 } Mode 0

Tabelle 12 *Der Port des Gate-Array (&7Fxx).*

Das Zeichen-ROM wird ausgelesen

Die Funktionen der einzelnen Bits des Ports &7Fxx müssen genau beachtet werden, wenn ein Zugriff auf den Zeichengenerator erfolgen soll. Die Mitteilung an das GA, das untere ROM zum Auslesen des Zeichensatzes im Graphikmodus 2 freizugeben, entspricht laut unserer Tabelle 12 folgendem Bitmuster des Ports: X10001010 = &8A. Das Ausleseprogramm hat dann das folgende Aussehen:

```
4000                    1           ORG   #4000
4000  3E02              2           LD    A,#02
4002  CD0EBC            3           CALL  #BC0E                ;MODE 2 EINSCHALTEN.
4005  F3                4           DI                         ;SYSTEMINTERRUPT AUS.
4006  DD210038          5           LD    IX,#3800             ;ZEIGER AUF CHAR.ROM..
400A  2100C0            6           LD    HL,#C000             ;BILDSCHIRMANFANG.
400D  110008            7           LD    DE,#0800             ;FEINZEILENABSTAND.
4010  018A7F            8           LD    BC,#7F8A             ;MIT DEM GA-PORT UNTERES
4013  ED49              9           OUT   (C),C                ;ROM FREISCHALTEN.
4015  0600             10           LD    B,#00                ;SCHLEIFENZAEHLER.
                       11 ;                                    ;FUER ZEICHENAUSGABE
4017  E5               12 L1:       PUSH  HL                   ;BILDSCHIRMPOSITION
                       13 ;                                    ;ZWISCHENSPEICHERN.
4018  DD7E00           14 L2:       LD    A,(IX+#00)           ;BITMUSTER IN DEN
401B  77               15           LD    (HL),A               ;BILDSCHIRMSPEICHER.
401C  DD23             16           INC   IX                   ;NAECHSTES BYTE
                       17                                      ;DES BITMUSTERS.
401E  19               18           ADD   HL,DE                ;NAECHSTE FEINZEILE.
401F  30F7             19           JR    NC,L2                ;ZEICHEN FERTIG ?
4021  E1               20           POP   HL                   ;WENN JA,NAECHSTE
4022  23               21           INC   HL                   ;ZEICHENPOSITION.
4023  10F2             22           DJNZ  L1                   ;ALLE ZEICHEN
                       23                                      ;AUSGEGEBEN ?
4025  018E7F           24           LD    BC,#7F8E             ;WENN JA,UNTERES
4028  ED49             25           OUT   (C),C                ;ROM SPERREN.
402A  FB               26           EI                         ;SYSTEMINTERRUPT EIN.
402B  C9               27           RET                        ;FERTIG.
                       28 ;P24

L1      4017  L2      4018
```

Das Programm entspricht weitgehend den Zeichensetzprogrammen früherer Kapitel und soll daher auch nicht in allen Einzelheiten erklärt werden. Einige Besonderheiten sind aber zu beachten. Zunächst darf das Programm nicht im Bereich der ersten 16 KByte (&0000 bis &3FFF) abgespeichert werden, da dort ja auf die ROM-Etage umgeschaltet wird. Der Prozessor würde das Programm, das ja im RAM steht, gar nicht finden. Dann muß unbedingt mit dem Befehl DI der Systeminterrupt abgeschaltet werden, da der Prozessor sonst nach 1/300 Sekunde die Standardspeicherkonfiguration wiederherstellt. Das würde bedeuten, daß das Zeichen-ROM nur teilweise ausgelesen würde.
Vor dem ersten OUT-Befehl wird das BC-Doppelregister mit &7F8A geladen. Das bedeutet nicht, daß die Portadresse &7F8A ist. Wie schon im letzten Kapitel erwähnt wurde, wird nur das High-Byte &7F für die Portadressierung verwendet. Das Low-Byte, das ja im C-Register steht, enthält die „Botschaft" &8A und wird mit dem Befehl OUT (C),C in das GA-Register geschrieben. Nach Beendigung des Schreibvorgangs wird mit dem zweiten OUT-Befehl das ROM wieder abgeschaltet. Dazu werden die Bits 2 und 3 im GA-Port auf 1 gesetzt, was bedeutet, daß das C-Register mit &8E geladen werden muß (&E=X1110). Eigentlich wäre der zweite OUT-Befehl nicht notwendig. Im ersten Interrupt nach der Freigabe durch den Befehl EI würde das Betriebssystem die Standardspeicherkonfiguration auch ohne unser Zutun wiederherstellen.

Die RST-Befehle

Nachdem wir jetzt in der Lage sind, auf das ROM zuzugreifen, können wir auch die dort stehenden Routinen nutzen. Das im letzten Kapitel verwendete Unterprogramm TGL zum Laden der Register des Tongenerators ist die Kopie eines ab &0826 (&0853 beim CPC664; &0863 beim CPC6128) im ROM stehenden Maschinenprogramms. Zur Übung können Sie versuchen, in unserem Tonleiterprogramm diese Routine statt des TGL-Programms zu nutzen (siehe Übungen).
Vom Betriebssystem ist eine noch einfachere Möglichkeit zur Freischaltung des ROMs vorgesehen, als die bisher beschriebene. Dazu dienen die sog. Restart-Befehle. Das sind Maschinenbefehle mit den Abkürzungen

RST n

wobei n für eine der acht Zahlen &00, &08, &10, &18, &20, &28, &30 und &38 steht. Der Befehl RST n bewirkt dasselbe wie der Befehl CALL n. Allerdings benötigt der Befehlscode von RST n nur ein Byte, während es bei CALL n drei Bytes sind. Außerdem wird ein RST-Befehl fast doppelt so schnell ausgeführt wie der entsprechende CALL-Befehl. Aus diesem Grund liegen bei den Adressen &00, &08, &10, &18, &20, &28, &30 und &38 die Startpunkte wichtiger und oft benutzter Systemroutinen.
Für uns ist hier der Befehl RST &28 (Code &EF) interessant. Mit diesem Befehl kann zu jeder Routine des Betriebssystems gesprungen werden. Die Startadresse der gewünschten Routine muß unmittelbar hinter dem Befehl RST &28 in der üblichen L-Byte-, H-Byte-Reihenfolge folgen. Bei der Verwendung des Befehls RST &28 muß man berücksichtigen, daß dieser Befehl wie ein JP zu der Adresse wirkt, die in den beiden Bytes hinter RST &28 angegeben ist. Will man RST &28 wie einen CALL-Befehl verwenden, d. h. soll nach der Ausführung der ROM-Routine mit der Verarbeitung der Befehle fortgefahren werden, die hinter RST &28 (und der dazugehörigen Adressenangabe) folgen, so müssen besondere Maßnahmen ergriffen werden.
Wie man einen JP- zu einem CALL-Befehl erweitert, haben wir schon in Kapitel 11 angedeutet. Man muß dazu vor der Ausführung des JP-Befehls die Rückkehradresse auf dem Stapel abspeichern. Will man z. B. die ab Adresse &0826 (&0853 beim CPC664; &0863 beim CPC6128) im ROM stehende Routine aufrufen, so könnte das folgendermaßen aussehen:

```
4100   LD HL,&4107     ;Rückkehradresse
4103   PUSH HL         ;auf den Stapel.
4104   RST &28         ;Sprung ins ROM.
4105   &26             ;Adresse im ROM (L-Byte).
4106   &08             ;Adresse im ROM (H-Byte).
4107                   ;Erste Adresse nach dem
                       ;Unterprogrammaufruf.
```

Ein schnelleres, kürzeres und übersichtlicheres Verfahren ist es jedoch, den Befehl RST &28 hinter dem Hauptprogramm abzuspeichern und ihn vom Hauptprogramm aus mit einem CALL-Befehl aufzurufen:

```
4100  CALL &4200    ;Durch CALL wird die Rückkehr-
4103                ;adresse &4103 auf den Stapel gebracht.
....  ....
41FF  RET           ;Ende des Hauptprogramms.
4200  RST &28       ;Sprung ins ROM
4201  &26           ;nach
4202  &08           ;&0826.
```

Die nach RST &28 angegebene Adresse muß nicht unbedingt im Bereich zwischen &0000 und &3FFF liegen. Es wäre also möglich, die OUT-Befehle des Programms P24 aus diesem Kapitel mitsamt den dazugehörigen BC-Ladebefehlen wegzulassen, zu Beginn des Programms mit RST &28 und der anschließenden Adresse &4000 die Speicherkonfiguration umzustellen und das Programm zu starten. Darüber hinaus könnten in diesem Fall die DI- und EI-Befehle weggelassen werden, da bei dieser Art der Speicherumstellung die Rückkehr zur normalen Konfiguration erst nach Beendigung des aufgerufenen Programms erfolgt (siehe Übungsaufgaben).

Zusammenfassung

Über das Gate Array können die ROM-Bereiche des Speichers freigeschaltet werden. Die Portadresse des GA ist &7Fxx. Das Betriebssystem-ROM wird durch die Ausgabe von &BA ins GA-Register freigeschaltet. Das BASIC-ROM wird durch &86, beide ROMs gemeinsam werden durch &82 freigeschaltet.
Einfacher erfolgt die Freischaltung des Betriebssystems mit dem Befehl RST &28, gefolgt von der Startadresse des gewünschten Programms. Die RST-Befehle sind eine Spezialform der CALL-Befehle. So bewirkt RST &28 dasselbe wie CALL &0028.
Auch bei freigeschalteten ROMs wird bei Schreibbefehlen immer auf das RAM zurückgegriffen.

Übung 40 Ersetzen Sie im Programm P24 die Befehle zur Steuerung des Gate Array durch den Befehl RST &28.

Übung 41 Ersetzen Sie im Tonleiterprogramm P23 des vorigen Kapitels das Unterprogramm TGL durch die Betriebssystemsroutine ab &826 (&853 beim CPC664; &863 beim CPC6128). Verwenden Sie dazu den Befehl RST &28.

Übung 42 Überlegen Sie, welchen Grund es hat, daß in Programm P24 dem Schleifenzähler B für die Zeichenausgabe der Anfangswert Null gegeben wurde.

RST-Befehle. RST &00 führt einen sogenannten „Reset" des Computers durch, wie er auch durch das Drücken der Tastenkombination CTRL/SHIFT/ESC ausgelöst wird.
RST &08 ermöglicht den Zugriff auf das untere ROM (&0000 bis &3FFF). Die beiden dem Befehl RST &08 folgenden Bytes enthalten in Bit 0 bis Bit 13 die Adresse der auszuführenden Routine. Die Bits 14 und 15 definieren den Schaltzustand der ROMs. Hat Bit 14 den Wert Null, so ist das untere ROM freigeschaltet. Hat Bit 15 den Wert Null, so ist das obere ROM freigeschaltet. Auch RST &08 nn entspricht einem JP- und keinem CALL-Befehl.
RST &18 ermöglicht den Zugriff auf den ganzen Speicherbereich (RAM und ROM). Die zwei Bytes, die auf den Befehl folgen, enthalten die Adresse eines 3-Byte-Blocks. Die ersten beiden Bytes dieses Blocks geben die Adresse der aufzurufenden Routine an. Die für uns interessanten Werte des dritten Bytes sind:

252 (oberes und unteres ROM sind freigeschaltet)
253 (oberes ROM frei, unteres ROM gesperrt)
254 (oberes ROM gesperrt, unteres ROM frei)
255 (oberes und unteres ROM gesperrt)

Genau wie bei RST &28 wird bei RST &08 und RST &18 nach Beendigung der Routine die Ausgangspeichersituation wiederhergestellt. RST &28 entspricht jedoch einem CALL-Befehl.
RST &30 wird nicht vom Betriebssystem verwendet.
RST &38 wird für den Systeminterrupt verwendet (siehe nächstes Kapitel).

Was ist ein Interrupt?

23

Sicher haben Sie schon in einer Zeitschrift, einer Programmanleitung oder einem Computerbuch das Wort „Interrupt“ gelesen. Für den Uneingeweihten hört sich dieser Begriff geheimnisvoll und kompliziert an. Vielleicht ist deshalb das englische Wort „Interrupt“ in den Schneider-Handbüchern durch das deutsche Wort „Unterbrechung“ ersetzt worden.

Die Anwendung von Unterbrechungen

Um zu verstehen, was dieser Begriff bedeutet, schauen wir uns zunächst einmal zwei typische Anwendungen der Interruptprogrammierung an: Bei vielen Computerspielen läuft gleichzeitig mit der Spielhandlung eine vom Computer erzeugte Begleitmusik ab. Der Computer muß dabei zwei unabhängige Programme parallel bearbeiten: das Spiel- und das Musikprogramm.
Weniger spektakulär, aber viel alltäglicher ist ein Vorgang der abläuft, während Sie ein Programm in Ihren CPC eingeben. Vielleicht sind Sie der Meinung, daß die Arbeit des Computers erst mit dem Start des eingegebenen Programms beginnt. In Wirklichkeit ist Ihr CPC schon vorher in voller Aktion. Er muß Ihre Eingaben nicht nur in interne Codes umwandeln und abspeichern, sondern unabhängig davon ständig überprüfen, welche Tasten gerade gedrückt werden.
Die Betonung liegt hier auf dem Wort „unabhängig“. Mit Hilfe des Interrupts kann der Computer (fast) gleichzeitig mehrere Programme bearbeiten, die vom Aufbau her gar nicht miteinander verknüpft sind.
Um zu verdeutlichen, wie das vor sich geht, wird häufig folgender Vergleich verwendet: Stellen Sie sich vor, daß Sie einen wichtigen Besuch erwarten. Bis zu seiner Ankunft müssen Sie noch einige dringende Arbeiten an Ihrem Schreibtisch erledigen. Ausgerechnet an diesem Tag ist Ihre Haustürklingel defekt. Um Ihren Gast nicht unnötig warten zu lassen oder ihn eventuell sogar zur Umkehr zu veranlassen, stehen Sie jede Minute vom Schreibtisch auf und schauen vor der

Haustür nach, ob schon jemand eingetroffen ist. Daß Sie dabei mit der zu erledigenden Arbeit nicht vorankommen, versteht sich. Wäre die Klingel aber in Ordnung, dann könnten Sie sich in aller Ruhe auf Ihre Arbeit konzentrieren, bis der Besuch sich mit Hilfe dieser Klingel meldet.
Ein derartiges „Klingelsystem" ist auch bei Ihrem Computer installiert. Das Klingelzeichen heißt IRQ-Signal. Es wird vom Gate Array 300mal in der Sekunde erzeugt und erreicht die Z80-CPU über den sog. IRQ-Anschluß. IRQ ist die Abkürzung für den englischen Begriff „Interrupt Request", was soviel wie Unterbrechungs-Aufforderung heißt. Beim Empfang dieses Signals wird der momentan bearbeitete Befehl noch zu Ende geführt. Anschließend unterbricht der Prozessor jedoch das Programm, das gerade ausgeführt wird. Er legt lediglich die Adresse des nächsten zu bearbeitenden Befehls auf den Stapel und macht dann mit der Bearbeitung des an der Adresse &0038 stehenden Befehls weiter. Man kann auch sagen, daß das IRQ-Signal die Ausführung des Befehls CALL &0038 bzw. RST &38 erzwingt. Der Unterschied zu einem normalen Unterprogrammaufruf ist sicher klar: Ein normaler CALL-Befehl wird erst bearbeitet, wenn alle vor ihm stehenden Befehle ausgeführt sind. Das IRQ-Signal kann im Unterschied dazu mitten in eine Befehlsfolge „hineinplatzen".
Das Interrupt-Bearbeitungsprogramm, das an der Speicherstelle &0038 beginnt, darf natürlich nicht allzu lang sein. Das nächste IRQ-Signal folgt ja schon $^1/_{300}$ Sekunden später, und in der Zwischenzeit muß nicht nur das IRQ-Programm ausgeführt werden, sondern es soll ja noch genügend Zeit bleiben, um nach Beendigung des IRQ-Programms das unterbrochene Programm weiter bearbeiten zu können. Glücklicherweise kann die Z80-CPU während $^1/_{300}$ Sekunde ein- bis zweitausend Maschinenbefehle ausführen, so daß die Bearbeitung von ein- oder zweihundert Befehlen eines IRQ-Programms dem Benutzer nicht weiter auffällt.
Um das unterbrochene Programm nach Beendigung der „Störung" weiter ausführen zu können, müssen die Inhalte aller verwendeten Register der CPU zum Zeitpunkt der Unterbrechung zwischengespeichert und nach Beendigung der Unterbrechung wieder in die Register geladen werden. Das geschieht nicht automatisch, sondern ist Aufgabe des Programmierers, der ein Unterbrechungsprogramm schreibt. Die Programmierer des CPC-Betriebssystems verwenden für diese Zwischenspeicherung den Zweitregistersatz der Z80-CPU. Das ist auch der Grund, warum wir diesen Zweitregistersatz nie benutzt haben. Da in den Zweitregistern wichtige Informationen abgespeichert sind, die während des IRQ-Programmablaufs verwendet werden, würde eine Änderung dieser Registerinhalte durch den Benutzer mit Sicherheit zum Systemzusammenbruch führen.

Unterbrechungen verhindern

Bevor wir uns anschauen, was das IRQ-Programm eigentlich tut und wie wir es für unsere Zwecke abändern können, muß noch ein wichtiger Gesichtspunkt des IRQ-Betriebs besprochen werden. Bei manchen Tätigkeiten des Computers wäre ein Interrupt ausgesprochen störend, wenn nicht gar gefährlich. Ein Beispiel dafür sind Programme, bei denen der Prozessor aufs ROM zugreift. Bei einer Unterbrechung schaltet das Interruptprogramm eventuell wieder auf das parallel liegende RAM um, und nach Beendigung der Unterbrechung findet der Computer statt des erwarteten ROM-Speichers plötzlich den RAM-Speicher vor, was natürlich fatale Folgen hätte.
Für solche Fälle hat der Programmierer die Möglichkeit, mit dem Befehl

DI

die „Haustürklingel" abzustellen. DI ist die mnemonische Abkürzung für die englischen Wörter „Disable Interrupt", was soviel wie „Verhindere eine Unterbrechung" heißt. Nach diesem Befehl werden alle IRQ-Signale ignoriert, bis der Interrupt durch den Befehl

EI

(„Enable Interrupt", deutsch: „Ermögliche eine Unterbrechung") wieder freigegeben wird.

Eine interruptgesteuerte Digitaluhr

Doch nun zum eigentlichen Interruptprogramm, das an der Adresse &38 beginnt. Dieses Programm fragt die Tastatur auf gedrückte Tasten ab und verwaltet im übrigen sog. Warteschlangen, in die ein Benutzer „Ereignisse" einreihen kann. „Ereignisse" sind z. B. Ton- oder Graphikprogramme, die mit der eben besprochenen Methode unabhängig von anderen Programmen ablaufen sollen. Da wir aber in erster Linie die Arbeitsweise der Z80-CPU kennenlernen wollen, verzichten wir zunächst auf die vom Betriebssystem angebotene Möglichkeit und verwenden den Interrupt, indem wir das IRQ-Programm des Betriebssystems entsprechend abändern.
Als Beispiel für die Anwendung der Interrupttechnik wollen wir eine Digitaluhr programmieren, die in der linken oberen Ecke des Bildschirms (im Mode 2) immer mitläuft. Das Programm wird recht umfangreich werden, da in ihm nochmals die in den vorhergehenden Kapiteln entwickelten Verfahren verwendet werden

sollen. Damit wiederholen wir die Inhalte dieser Kapitel und vermeiden gleichzeitig die Systemroutinen, die das Interruptprogramm zwar kürzer, aber gleichzeitig auch langsamer machen würden.
Um das Uhrenprogramm in das Betriebssystem einfügen zu können, schauen wir uns die Befehlsfolge des Interruptprogramms ab der Einsprungstelle &0038 an. An dieser Stelle steht nur der Befehl JP &B939 (JP &B941 beim CPC664/6128), der zum eigentlichen Interruptprogramm des Betriebssystems weiterleitet. Man könnte nun auf die Idee kommen, die Sprungadresse &B939 (&B941 beim CPC664/6128) durch die Anfangsadresse unseres Uhrenprogramms zu ersetzen und den Sprung nach &B939 (&B941 beim CPC664/6128) am Schluß des Uhrenprogramms auszuführen. Das würde aber nicht funktionieren, da die Adresse &0038 sowohl im RAM als auch im ROM vorhanden ist. Wenn das Betriebssystem das ROM freigeschaltet hat, benutzt die CPU die dort liegende Speicherstelle &0038 als Einsprungspunkt für das IRQ-Programm, und da wir den ROM-Inhalt nicht verändern können, würde unser Uhrenprogramm nur bei einem Teil der Interruptereignisse durchlaufen werden.
Damit ist aber auch ein Weg klar, den wir gehen können, um das Uhrenprogramm in das IRQ-Systemprogramm einzufügen (Bild 25).

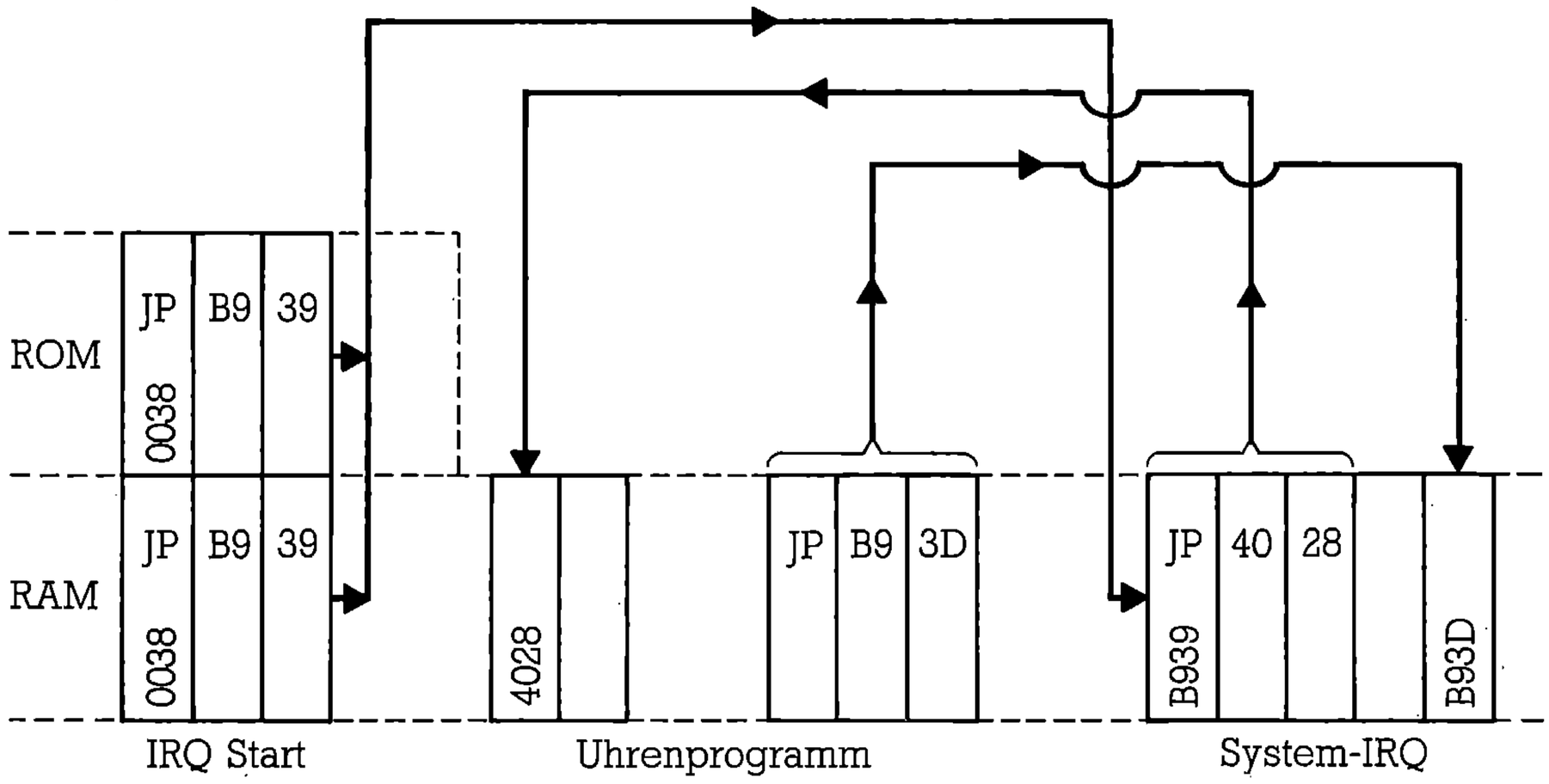


Bild 25 *Einfügung des Uhrenprogramms in das Systeminterruptprogramm. Beim CPC664/6128 müssen die Adressen &B939,..., &B93D durch &B941,..., &B945 ersetzt werden.*

Da das IRQ-Programm auf jeden Fall über die RAM-Speicherstelle &B939 (&B941 beim CPC664/6128) verläuft, schreiben wir an diese Stelle den Sprungbefehl JP &4028 zum Beginn unseres Uhrenprogramms. Da dieser Befehl eine Länge von drei Bytes hat, überschreiben wir damit die Speicherstellen &B939, &B93A und &B93B (&B941, &B942 und &B943 beim CPC664/6128), die die Befehle

```
DI
EX AF,AF
JR C,&B970 (JR C,&B978 beim CPC 664/6128)
```

enthalten. Die überschriebenen Befehle hängen wir dann einfach an das Uhren-

programm an und geben danach einen Sprungbefehl an die Speicherstelle &B93D (&B945 beim CPC664/6128) zum ersten intaktgebliebenen Befehl des System-IRQ-Programms. Auf diese Weise durchläuft die CPU nach dem Uhrenprogramm die normale Befehlssequenz, und das Betriebssystem merkt von unserem Eingriff nichts.
Das Programm, das über 100 Befehle lang ist, sieht folgendermaßen aus:

```
4000             1          ORG   #4000
4000  21F240       2          LD    HL,#40F2        ;ZEIGER AUF SPEICHER
                   3                                 ;FUER ZEITANGABEN.
4003  0603         4          LD    B,#03           ;3 ZAHLEN MIT JE
4005  CD06BB       5 LO:      CALL  #BB06           ;2 ZIFFERN
4008  ED6F         6          RLD                   ;EINGEBEN (HH MM SS).
400A  CD06BB       7          CALL  #BB06
400D  ED6F         8          RLD
400F  2B           9          DEC   HL
4010  10F3        10          DJNZ  LO
                  11                                ;AENDERN DES IRQ-PROG.
4012  F3          12          DI                    ;SYSTEMINTERRUPT AUS.
4013  212540      13          LD    HL,NPR          ;ZEIGER AUF NEUEN
                  14                                ;SPEICHERINHALT.
4016  1139B9      15          LD    DE,#B939        ;ZEIGER AUF IRQ-PROG..
4019  010300      16          LD    BC,#0003        ;3 SPEICHERSTELLEN
401C  EDB0        17          LDIR                  ;WERDEN UMKOPIERT.
401E  03          18          INC   BC              ;TAKTZAEHLER
401F  ED43F440    19          LD    (#40F4),BC      ;AUF 1 SETZEN.
4023  FB          20          EI                    ;SYSTEMINTERRUPT EIN.
4024  C9          21          RET                   ;FERTIG...
4025  C32840      22 NPR:     DEFB #C3,#28,#40      ;JP #4028
                  23 ;
                  24 ;                              ;IRQ-EINSPRUNG.
4028  F3          25          DI                    ;SYSTEMINTERRUPT AUS.
4029  F5          26          PUSH  AF              ;REGISTER RETTEN.
402A  E5          27          PUSH  HL
402B  D5          28          PUSH  DE
402C  C5          29          PUSH  BC
402D  DDE5        30          PUSH  IX
402F  FDE5        31          PUSH  IY
4031  ED4BF440    32          LD    BC,(#40F4)      ;TAKTZAEHLER LADEN,
4035  0B          33          DEC   BC              ;HERUNTERZAEHLEN
4036  ED43F440    34          LD    (#40F4),BC      ;UND ABSPEICHERN.
403A  78          35          LD    A,B             ;TAKTZAEHLER
403B  B1          36          OR    C               ;AUF 0 ?
403C  C29540      37          JP    NZ,IRQE         ;WENN NEIN,FERTIG.
403F  012C01      38          LD    BC,#012C        ;SONST WIEDER AUF 300
4042  ED43F440    39          LD    (#40F4),BC      ;HOCHSETZEN.
4046  21F040      40          LD    HL,#40F0        ;SEKUNDENZAEHLER.
4049  1E60        41          LD    E,#60           ;HOECHSTER WERT.
404B  CDA440      42          CALL  TICK            ;HOCHZAEHLEN UND MIT
                  43 ;                              ;60 VERGLEICHEN.
404E  380C        44          JR    C,UAU           ;ZUR ZEITAUSGABE WENN
                  45 ;                              ;60 NICHT ERREICHT.
4050  23          46          INC   HL              ;SONST MIN.ZAEHLER
4051  CDA440      47          CALL  TICK            ;HOCHZAEHLEN UND
4054  3806        48          JR    C,UAU           ;ZUR ZEITAUSGABE WENN
                  49                                ;60 NICHT ERREICHT.
4056  23          50          INC   HL              ;SONST STD.ZAEHLER
4057  1E24        51          LD    E,#24           ;HOCHZAEHLEN UND MIT
4059  CDA440      52          CALL  TICK            ;24 VERGLEICHEN.
                  53 ;                              ;ZEITAUSGABE.
405C  2AC9B1      54 UAU:     LD    HL,(#B1C9)      ;BILDSCHIRMOFFSET.
405F  114700      55          LD    DE,#47          ;ANFANGSPOSITION DER
4062  19          56          ADD   HL,DE           ;ZIFFERNAUSGABE
4063  7C          57          LD    A,H             ;AM ENDE DER 1.
```

```
4064  E607          58            AND   #07              ;ZEILE
4066  F6C0          59            OR    #C0              ;BERECHNEN.
4068  67            60            LD    H,A
4069  E5            61            PUSH  HL              ;BERECHNETE ADDRESSE
406A  FDE1          62            POP   IY              ;INS IY-REGISTER.
406C  DD218039      63            LD    IX,#3980        ;BITMUSTER DER ZIFFERN.
4070  21F240        64            LD    HL,#40F2        ;STUNDENZAEHLER.
4073  110008        65            LD    DE,#0800        ;FEINZEILENABSTAND.
4076  AF            66            XOR   A               ;AKKU LOESCHEN.
4077  0603          67            LD    B,#03           ;3 ZAEHLER
4079  0E02          68 L11:       LD    C,#02           ;MIT JE 2 ZIFFERN.
407B  ED6F          69 L1:        RLD                   ;ZIFFER IN DEN AKKU.
407D  F5            70            PUSH  AF              ;A ZWISCHENSPEICHERN.
407E  87            71            ADD   A,A             ;DIESE 3 BEFEHLE
407F  87            72            ADD   A,A             ;MULTIPLIZIEREN DIE
4080  87            73            ADD   A,A             ;ZIFFER MIT 8 UM DAS
                    74                                  ;RICHTIGE BITMUSTER ZU
                    75                                  ;ERHALTEN.
4081  32B940        76            LD    (LX+2),A        ;ABSPEICHERN NACH DD
                    77                                  ;(ZEILE 112).
4084  CDAD40        78            CALL  ZAUS            ;ZIFFERNAUSGABE.
4087  F1            79            POP   AF              ;AKKU HOLEN.
4088  FD23          80            INC   IY              ;NAECHSTE BILDS.POSITION
408A  0D            81            DEC   C               ;BEIDE ZIFFERN
408B  C27B40        82            JP    NZ,L1           ;SCHON AUSGEGEBEN ?
408E  ED6F          83            RLD                   ;BYTE WIEDERHERSTELLEN.
4090  FD23          84            INC   IY              ;LEERSTELLE LASSEN.
4092  2B            85            DEC   HL              ;NAECHSTER ZAEHLER.
4093  10E4          86            DJNZ  L11             ;ZEITAUSGABE FERTIG ?
4095  FDE1          87 IRQE:      POP   IY              ;WENN JA,REGISTER
4097  DDE1          88            POP   IX              ;VOM STAPEL HOLEN.
4099  C1            89            POP   BC
409A  D1            90            POP   DE
409B  E1            91            POP   HL
409C  F1            92            POP   AF
409D  08            93            EX    AF,AF           ;UEBERSCHRIEBENE
409E  DA70B9        94            JP    C,#B970         ;BEFEHLE NACHHOLEN,
40A1  C33DB9        95            JP    #B93D           ;UND ZUM SYSTEMIRQ.
                    96 ;
40A4  7E            97 TICK:      LD    A,(HL)          ;ZAEHLERSTAND HOLEN.
40A5  3C            98            INC   A               ;HOCHZAEHLEN.
40A6  27            99            DAA                   ;DEZIMALABGLEICH.
40A7  77           100            LD    (HL),A          ;ABSPEICHERN.
40A8  BB           101            CP    E               ;HOECHSTSTAND ?
40A9  D8           102            RET   C               ;WENN NEIN,FERTIG.
40AA  AF           103            XOR   A               ;SONST AUF NULL
40AB  77           104            LD    (HL),A          ;ZURUECKSETZEN.
40AC  C9           105            RET                   ;FERTIG.
                   106 ;
40AD  FDE5         107 ZAUS:      PUSH  IY              ;BENUTZTE
40AF  DDE5         108            PUSH  IX              ;REGISTER
40B1  C5           109            PUSH  BC              ;ZWISCHENSPEICHERN.
40B2  018A7F       110            LD    BC,#7F8A        ;UNTERES ROM
40B5  ED49         111            OUT   (C),C           ;FREISCHALTEN.
40B7  DD7E00       112 LX:        LD    A,(IX+#00)      ;BITMUSTER MIT MODI-
                   113 ;                                ;FIZIERTEM DD HOLEN.
40BA  FD7700       114            LD    (IY+#00),A      ;IN DEN BILDS.SPEICHER.
40BD  DD23         115            INC   IX              ;NAECHSTES BYTE.
40BF  FD19         116            ADD   IY,DE           ;NAECHSTE FEINZEILE.
40C1  D2B740       117            JP    NC,LX           ;ZEICHEN FERTIG ?
40C4  018E7F       118            LD    BC,#7F8E        ;UNTERES ROM
40C7  ED49         119            OUT   (C),C           ;SPERREN.
40C9  C1           120            POP   BC              ;ALTE REGISTER-
40CA  DDE1         121            POP   IX              ;INHALTE
40CC  FDE1         122            POP   IY              ;HOLEN.
40CE  C9           123            RET                   ;ZUM HAUPTPROG.
                   124 ;P25
```

```
IRQE    4095   LO      4005   L1      407B
L11     4079   LX      40B7   NPR     4025
TICK    40A4   UAU     405C   ZAUS    40AD
```

Änderungen für den CPC664/6128:
Zeile 15: &B939 ersetzen durch &B941
Zeile 54: &B1C9 ersetzen durch &B7C4
Zeile 94: &B970 ersetzen durch &B978
Zeile 95: &B93D ersetzen durch &B945

Wenn Sie das Programm eingegeben und mit CALL &4000 gestartet haben, muß zunächst die Zeit in der Form HH MM SS angegeben werden. Sowohl die Stundenangabe HH als auch die Minuten- und Sekundenangaben MM und SS müssen zweistellig, also eventuell mit führender Null erfolgen. Eine Überprüfung der Eingabe nach unsinnigen Werten erfolgt nicht! Für die Eingabe wird als einzige Systemroutine des Programms die Tastaturabfrageroutine &BB06 verwendet. Die Speicherstellen für die Stunden-, Minuten- und Sekundenanzeige haben die Adressen &40F2, &40F1 und &40F0. Der erste Programmteil entspricht weitgehend dem Speichereingabeprogramm des Kapitels 17.
Der nächste Programmabschnitt ab Zeile 12 ändert, wie beschrieben, mit Hilfe des Blocktransferbefehls LDIR die IRQ-Routine ab &B939. Natürlich muß während dieser Änderung der Systeminterrupt mit dem Befehl DI abgeschaltet werden. Nach der Änderung des Systeminterrupts wird in der Speicherstelle &40F4 eine Eins abgespeichert (Zeilen 18, 19). Die Speicherstellen &40F4 (Low-Byte) und &40F5 (High-Byte) dienen als Zähler für die Interruptimpulse. Nach dem Setzen des Interruptzählers ist das Initialisierungsprogramm in Zeile 21 schon beendet, und die Kontrolle geht an BASIC bzw. an das Betriebssystem zurück. Ab jetzt ruft der Computer jede 1/300 Sekunde vor Erreichen der eigentlichen Interruptroutine unser Uhrenprogramm ab &4028 auf.

Wie die Uhr tickt

Das Uhrenprogramm selbst arbeitet mit vier Zählern: dem Interruptzähler, der 1/300 Sekunden zählt, sowie dem Sekunden-, Minuten- und Stundenzähler. Bei jedem Interrupt wird in Zeile 33 der Inhalt des Interruptzählers um Eins verringert. Solange der Inhalt noch nicht Null ist, wird das Uhrenprogramm über die Marke IRQE in Zeile 87 wieder verlassen, ohne daß eine Aktion erfolgt; ist jedoch der Wert Null erreicht, und das geschieht einmal pro Sekunde, dann wird zunächst der Interruptzähler wieder auf den Ausgangswert 300 gesetzt. Anschließend wird in Zeile 42 der Sekundenzähler um Eins erhöht und mit dem Höchstwert 60 verglichen. Ist dieser Wert erreicht, dann wird der Sekundenzähler auf 0 zurückgesetzt und der Minutenzähler um Eins erhöht. Entsprechend wird der Stundenzähler um Eins erhöht, wenn der Minutenzähler den Wert 60 erreicht. Um den In-

halt dieser Zähler direkt ausgeben zu können, wird das Zählen im Dezimalsystem durchgeführt. Da der Zähl- und Vergleichsvorgang für Sekunden, Minuten und Stunden dieselben Programmschritte erfordert, wird zu diesem Zweck ein Unterprogramm benutzt, das den Namen TICK besitzt.
Der Ausgabeteil des Programms beginnt bei der Marke UAU in Zeile 54. Zunächst wird aus den Speicherstellen &B1C9 und &B1CA (&B7C4 und &B7C5 beim CPC664/6128) der Wert geladen, den man zu &C000 addieren muß, um den Bildschirmspeicherplatz der linken oberen Ecke des Bildschirms zu erhalten. Obwohl nach dem Einschalten oder einem Mode-Befehl die linke obere Bildschirmecke der Speicheradresse &C000 entspricht, verschiebt sich dieser Wert ständig durch das Scrollen des Bildschirms, und über diese Verschiebung führt das System in den genannten Speicherstellen Buch. Mit Hilfe dieser Speicherstellen wird dann die Bildschirmadresse am rechten Rand der obersten Zeile berechnet, an der die Ausgabe der Uhrzeit erfolgen soll. Diese Berechnungen sind in Zeile 60 abgeschlossen.
Die Ausgabe der einzelnen Ziffern erfolgt mit dem Unterprogramm ZAUS, das ab Zeile 107 steht und weitgehend den Zeichenausgabeprogrammen früherer Kapitel entspricht. Zwei wichtige Unterschiede zu den früheren Programmen sind jedoch vorhanden. Erstens muß zum Zeichen-ROM zugegriffen werden. Dieses wird daher vor dem Zugriff mit Hilfe des Gate-Array-Ports freigeschaltet und nach dem Zugriff wieder gesperrt (siehe vorausgegangenes Kapitel). Zweitens muß eine Steuerung vorhanden sein, die den Zugriff zum Bitmuster der Ziffer ermöglicht, die gerade ausgegeben werden soll.
Diese Steuerung wird folgendermaßen erreicht: Zunächst erhält das IX-Register den Wert &3980, denn dort beginnen im Zeichen-ROM die Bitmuster der Ziffern. Dann wird mit dem RLD-Befehl an der Marke L1 (Zeile 69) die auszugebende Ziffer in den Akkumulator gebracht. Durch die dreimalige Anwendung des Befehls ADD A,A ergibt sich das Achtfache des Ziffernwerts, und da jedes Bitmuster gerade acht Bytes lang ist, erhält man die Anfangsadresse des richtigen Bitmusters durch Addition dieses Werts zum Inhalt des IX-Registers. Diese Addition wird mit Hilfe der Selbstmodifikation des Programms erreicht, durch die das Abstandsbyte d in dem indiziert adressierten Ladebefehl LD A,(IX+d), der an der Adresse LX (Zeile 112) im Zeichenausgabeprogramm steht, entsprechend verändert wird. (Zur Selbstmodifikation von Programmen siehe Kapitel 16.)
Im Rest des Programms (ab der Marke IRQE) wird der Einsprung in die Systeminterrupt-Routine vorbereitet. Zunächst werden die gespeicherten Registerwerte vom Stapel zurückgeholt, dann folgen zwei Befehle, die am Beginn des Systemprogramms ab &B939 (&B941 beim CPC664/6128) überschrieben wurden, und schließlich erfolgt der Sprung zur Adresse &B93D (&B945 beim CPC664/6128). Von dieser Stelle ab wird der normale Systeminterrupt abgearbeitet.

Zusammenfassung

Interrupt (Unterbrechung)
Eine Unterbrechungsanforderung (IRQ) wird durch ein Signal auf der IRQ-Leitung an den Prozessor übermittelt. Im CPC wird durch das Gate Array alle 1/300 Sekunden ein IRQ-Signal erzeugt. Der Prozessor unterbricht dann die Bearbeitung des laufenden Programms und führt den Befehl RST &38 aus. An der Adresse &0038 beginnt das sog. Interrupt-Bearbeitungsprogramm. Nach Beendigung des Interruptprogramms wird die Bearbeitung des unterbrochenen Programms fortgesetzt. Für den Benutzer scheint die Bearbeitung beider Programme parallel zu erfolgen.
Eine Unterbrechung durch ein IRQ-Signal kann mit dem Befehl DI verhindert werden. Der Befehl EI läßt die Unterbrechung wieder zu.

Zusatzinformationen

Die Anfangsadresse des Bildschirmspeichers. Von unserem Vorsatz, bei der Programmierung des Uhrenprogramms P25 möglichst ohne das Betriebssystem auszukommen, sind wir an einer Stelle abgewichen: Um ständig die richtige Adresse für die linke obere Bildschirmecke zu erhalten, haben wir in Zeile 54 aus den Speicherstellen &B1C9 und &B1CA (&B7C4 und &B7C5 beim CPC664/6128) den sogenannten Offset (Verschiebung) des Bildschirmspeichers ausgelesen. Nach dem Einschalten oder nach der Ausführung eines Mode-Befehls entspricht die linke obere Bildschirmecke (also der Bildschirmanfang) der Anfangsadresse des Bildschirmspeichers &C000. Diese Zuordnung ändert sich jedoch beim „Scrollen". Es würde zu lange dauern, dabei den ganzen, 16 K umfassenden Bildschirmspeicher zu verschieben. Stattdessen wird ausgenutzt, daß der Video-Controller in der Lage ist, dem Bildschirmanfang auch andere Adressen des Bildschirmspeichers zuzuordnen. Aus technischen Gründen müssen diese Adressen allerdings geradzahlig sein. Beim Scrollen wird der Bildschirmanfang daher einfach auf den Beginn der nächsten Bildschirmzeile verschoben, und es muß dann nur die unten am Bildschirm neu dazukommende Zeile neu beschrieben werden. Als Folge dieses Verfahrens entspricht dann die erste Adresse des Bildschirmspeichers &C000 irgendeiner, bei jedem Scrollvorgang wechselnden Bildschirmposition. Das ist auch der Grund dafür, warum wir am Beginn vieler unserer Übungsprogramme mit dem Befehl Mode 2 die Standardzuordnung des Bildschirmspeichers zu den Positionen des Bildschirms herstellen mußten.
Die Differenz zwischen der aktuellen Bildschirmanfangsadresse und der Anfangsadresse des Bildschirmspeichers &C000 ist der erwähnte Offset.
Das Bildschirmstartregister im Video-Controller. Natürlich muß im Betriebssystem ständig darüber Buch geführt werden, an welcher Adresse sich der Bildschirmanfang gerade befindet. Diese Buchführung kann vom Benutzer in den Speicherstellen &B1C9 und &B1CA (&B7C4 und &B7C5 beim CPC664/6128) ein-

gesehen werden. Aber es ist auch möglich, die Information über den Bildschirmanfang direkt vom Video-Controller zu beziehen. Sie ist in den Registern 12 und 13 seiner insgesamt 19 Register abgelegt. Dabei enthält Register 13 das Low-Byte und Register 12 das High-Byte der Verschiebung. Da diese maximal 2000 Bytes betragen kann (entsprechend 25 Zeilen mit je 80 Zeichen) und da, wie schon erwähnt, der Bildschirmanfang nur an geradzahligen Adressen liegen kann, reichen zur Darstellung der Verschiebung zehn Bits aus. Das sind die acht Bits von Register 13 und die zwei niederwertigen Bits von Register 12. Die sich daraus ergebende 10-Bit-Zahl muß man allerdings noch mit 2 multiplizieren, um den Wert des Offsets zu erhalten.

Der Video-Controller wird ausgelesen. Der Video-Controller ist mit der CPU durch drei Ports verbunden: Dem Adreßport, dem Eingabeport und dem Ausgabeport (Leseport).

Um ein Register des Video-Controllers auszulesen, muß zuerst die Registernummer dem Adreßport des Video-Controllers mitgeteilt werden. Seine Portadresse ist &BCxx (auch hier wird nur das H-Byte zur Adressierung benutzt). Dann kann der Lesezugriff über die Portadresse &BFxx erfolgen. Das sieht beim Auslesen der Register 13 und 12 konkret folgendermaßen aus:

```
LD A,&0D      ;&0D = 13.
LD B,&BC      ;Adreßport.
OUT (C),A     ;Register 13 anwählen.
LD B,&BF      ;Leseport.
IN L,(C)      ;Inhalt von R13 nach L.
LD A,&0C      ;&0C = 12.
LD B,&BC      ;Auslesen von
OUT (C),A     ;Register 12
LD B,&BF      ;wie oben.
IN A,(C)      ;Inhalt von R12 nach A.
AND &03       ;Bits 0 und 1 abtrennen.
LD H,A        ;A nach H bringen.
ADD HL,HL     ;HL mit 2
RET           ;multiplizieren.
```

Nach Ausführung dieses Programms steht der Offset im HL-Register. Das ist natürlich ein umständlicheres Verfahren, als vom Angebot des Betriebssystems Gebrauch zu machen, dieselbe Zahl aus den Speicherstellen &B1C9 und &B1CA (&B7C4 und &B7C5 beim CPC664/6128) auszulesen. Um die Anfangsadresse des Bildschirms zu erhalten, muß man den Offset noch zur Anfangsadresse des Bildschirmspeichers &C000 addieren.

Weitere Informationen über Interrupts

24

Die bisher besprochene Verwendung des Interrupts, verschiedene Programme quasi parallel verarbeiten zu können, ist nicht die einzige Einsatzmöglichkeit dieser Technik. Eine andere Anwendung ist die Überwachung von Peripheriegeräten. Dabei signalisiert ein Peripheriegerät durch einen IRQ-Impuls, daß es mit dem Prozessor in Verbindung treten will. Dadurch wird es dem Prozessor erspart, den Zustand des Peripheriegeräts kontinuierlich zu überprüfen. (Denken Sie an das Beispiel mit der Haustürklingel am Beginn des vorigen Kapitels.) Ohne entsprechende Zusatzeinrichtungen kann der CPC diese Methode jedoch nicht verwenden.

Interruptmodi des Z80

Der Z80-Prozessor besitzt insgesamt drei Möglichkeiten, auf ein IRQ-Signal zu reagieren. Zwei dieser Möglichkeiten sind speziell für die eben erwähnten Überwachungsaufgaben geeignet. Zum Einschalten dieser verschiedenen Interruptmodi gibt es eigens die Befehle IM 0, IM 1 und IM 2. (Verwechseln Sie die Interruptmodi nicht mit den Graphikmodi!) Im Modus 0 erwartet der Prozessor auf der Datenleitung Anweisungen des Peripheriegeräts; der Modus 0 kann daher nur mit entsprechenden Zusatzgeräten genutzt werden. Der Modus 1 ist die vom CPC verwendete Art: Jedes IRQ-Signal erzwingt den Befehl RST &38. Im Modus 2 kann die Startadresse des Interruptbearbeitungsprogramms frei gewählt werden. Das Low-Byte dieser Adresse wird vom Peripheriegerät über die Datenleitung geliefert. Das High-Byte steht in einem Spezialregister des Prozessors, das I-Register heißt („I“ steht natürlich für Interrupt). Das I-Register kann mit dem Befehl LD I,A über den Akkumulator geladen werden.

NMI und BUSRQ

Neben den bisher besprochenen Unterbrechungsmöglichkeiten gibt es noch zwei weitere, für die sogar eigene Leitungen zum Prozessor vorgesehen sind. Das ist einmal die NMI-Leitung, die einen Interrupt signalisiert, der nicht mit dem Befehl DI abgeschaltet werden kann. Daher kommt auch die Bezeichnung „nicht maskierbarer Interrupt" (englisch „non-maskable interrupt"). Zudem gibt es noch die BUSRQ(Bus Request)-Leitung. Ein Signal auf dieser Leitung erzeugt eine Unterbrechung, bei der noch nicht einmal das Ende des gerade bearbeiteten Befehls abgewartet wird. Auch diese Leitungen können beim CPC nur durch spezielle Zusatzelektronik genutzt werden.
Wenn bei einem Z80-System mehrere Unterbrechungsmöglichkeiten vorgesehen sind, existiert eine ganz bestimmte Prioritätenregelung. Das bedeutet, daß es Unterbrechungen verschiedener Wichtigkeit gibt. Für die Freigabe evtl. zurückgestellter Unterbrechungsaufforderungen sind spezielle Befehle zur Programmbeendigung vorgesehen, die RETI („Return from Interrupt") bzw. RETN („Return from Non-Maskable Interrupt) heißen. Falls die eben besprochenen Unterbrechungsmöglichkeiten nicht vorhanden sind, haben sie dieselbe Wirkung wie der Befehl RET.
Zur Gruppe der speziellen Befehle zur Interruptbearbeitung gehört auch noch der HALT-Befehl. Er hält den Prozessor bis zum nächsten Interruptsignal an.

Die EX-Befehle

Zur Rettung der Register während des Interrupt-Bearbeitungsprogramms dienen die Befehle

EX AF,AF'

und

EXX

Der mnemonische Code EX kommt vom englischen „exchange", d. h. „austauschen". Durch diese Befehle werden die Registerinhalte mit denen des Zweitregistersatzes vertauscht. EX AF,AF' tauscht die Inhalte von A und F mit denen von A' und F' aus. EXX tauscht die Inhalte von BC, DE und HL mit denen von B'C', D'E' und H'L' aus.
Diese Befehle gehören mit zur Gruppe der Datentransportbefehle. Im Gegensatz zu den LD-Befehlen bleibt aber bei den EX-Befehlen der Inhalt des Absenderre-

gisters nicht erhalten; beide angegebenen Register (bzw. Registersätze) sind sowohl Sender als auch Empfänger.
Zu den EX-Befehlen gehören auch der relativ häufig verwendete Befehl

EX DE,HL

der den Inhalt der Registerpaare DE und HL austauscht, sowie drei Befehle der Form

EX (SP),rr

Für rr kann dabei eines der Register HL, IX oder IY stehen. Diese Befehle vertauschen den Inhalt des obersten Stapelelements mit dem Inhalt des angegebenen Registers rr. Dabei ist wieder zu beachten, daß ein Stapeleintrag aus zwei Bytes besteht.

Die Benutzung der ROM-Routinen

25

Das Ziel dieses Buches war es, Sie mit den Eigenschaften des Z80-Prozessors vertraut zu machen. Der CPC diente dabei nur als Schulungsgerät. Das hervorragende Betriebssystem dieses Computers erspart es dem Benutzer im allgemeinen, in allzu engen Kontakt mit dem Prozessor und seinen Eigenschaften zu kommen, was natürlich dem Ziel unseres Kurses widerspricht. Aus diesem Grund wurde es in den vorangegangenen Kapiteln auch vermieden, ausgiebigen Gebrauch von den Betriebssystemsroutinen zu machen. Wir haben dabei zur Übung häufig „das Rad“ zum zweiten Mal erfunden.
Wenn man den CPC nicht als Schulungsgerät ansieht, sondern ihn in erster Linie zur Lösung anstehender Probleme verwenden will, ergibt sich natürlich eine andere Perspektive. Die Nutzung des Betriebssystems spart dann viel eigene Arbeit. Die Betriebssystemsroutinen sind natürlich im ROM abgespeichert, können aber über das RAM aufgerufen werden. Die Ansprungstellen beginnen bei der Adresse &B900. Ein Verzeichnis all dieser Routinen sowie ihrer Einsprung- und Aussprungbedingungen finden Sie im Firmware-Handbuch der Fa. Schneider oder in entsprechenden Büchern.

Die Betriebssystemsroutinen

In diesem Kapitel soll nun an einigen Beispielen die Verwendung der Firmware gezeigt und einige Routinen beschrieben werden. Zunächst folgt eine Zusammenstellung einiger dieser Ansprungstellen (Tabelle 13). Fast alle davon haben wir schon benutzt bzw. wir werden sie in diesem Kapitel noch benutzen. Die Namen in Spalte 2, die für die Programmierung keine Bedeutung haben, und die übrigen Angaben sind dem Firmware-Handbuch entnommen. Der Inhalt der in der Spalte „Aussprung“ in Klammern angegebenen Register wird durch die Routine überschrieben und muß gegebenenfalls vor dem Aufruf zwischengespeichert werden.

Adresse	Name	Wirkung	Einsprung	Aussprung
&BB06	KM WAIT CHAR	Auf Tastendruck warten	–	A enthält den ASCII-Code der gedrückten Taste (F)
&BB09	KM READ CHAR	Zeichen einlesen	–	Bei Tastendruck wie &BB06. Wenn kein Tastendruck, wird Carry gelöscht (AF)
&BB5A	TXT OUTPUT	Zeichen ausgeben	A enthält ASCII-Code	–
&BB6C	TXT CLEAR WINDOW	Textfenster löschen (Wie CLS)	–	(AF, BC, DE, HL)
&BB75	TXT SET CURSOR	Wie LOCATE	H gibt die Spalte an	(AF, H, L)
			L gibt die Zeile an	
&BB81	TXT CUR ON	Cursor zulassen	–	–
&BB84	TXT CUR OFF	Cursor abschalten	–	–
&BBC0	GRA MOVE ABSOLUTE	Grafikcursor setzen	DE enthält die X-Koordinate	(AF, BC, DE, HL)
			HL enthält die Y-Koordinate	
&BBEA	GRA PLOT ABSOLUTE	Punkt plotten	DE enthält die X-Koordinate	(AF, BC, DE, HL)
			HL enthält die Y-Koordinate	
&BBF6	GRA LINE ABSOLUTE	Linie zwischen Cursor und angegebener Position ziehen	DE enthält die X-Koordinate	(AF, BC, DE, HL)
			HL enthält die Y-Koordinate	
&BBFC	GRA WR CHAR	Zeichen an der Grafikposition ausgeben	A enthält den ASCII-Code	(AF, BC, DE, HL)
&BC0E	SCR SET MODE	Grafikmode setzen	A enthält die Modenummer	(AF, BC, DE, HL)
&BCE9	KL ADD TICKER	Block an Taktliste übergeben	HL enthält die Taktblockadresse	(AF, BC, DE)
			DE enthält den Taktzähleranfangswert	
			BC enthält den Taktzählerwiederanlaufswert	
&BCEC	KL DEL TICKER	Block aus Taktliste entfernen	HL enthält die Taktblockadresse	(AF, HL)
&BCEF	KL INIT EVENT	Ereignisblock einrichten	HL enthält die Ereignisblockadresse	
			DE enthält die Adresse des Ereignisprogramms	
			B enthält die Priorität	
			C enthält die ROM-Auswahl	

Tabelle 13 *Firmware-Routinen*

Um die Anwendung der Systemroutinen zu demonstrieren, soll zunächst mit Hilfe des Linien-Programms &BBF6 von der linken unteren Ecke aus fächerförmig eine Reihe von Linien gezogen werden. Das ergibt ein hübsches geometrisches Muster. Das Programm hat folgendes Aussehen:

```
4000                    1          ORG   #4000
4000  3E02               2          LD    A,#02
4002  CD0EBC             3          CALL  #BC0E              ;MODE 2 EINSCHALTEN.
4005  210000             4          LD    HL,#0000           ;Y-KOORDINATE DES
                         5                                   ;ENDPUNKTS DER 1.LINIE.
4008  0632               6          LD    B,#32              ;ZAHL DER LINIEN (50)
400A  C5                 7 L1:      PUSH  BC                 ;B UND
400B  E5                 8          PUSH  HL                 ;HL ZWISCHENSPEICHERN.
400C  110000             9          LD    DE,#0000           ;GRAPHIK-CURSOR IN
400F  210000            10          LD    HL,#0000           ;DIE LINKE UNTERE
4012  CDC0BB            11          CALL  #BBC0              ;ECKE SETZEN.
4015  117F02            12          LD    DE,#027F           ;X-KOORDINATE DER
                        13                                   ;ENDPUNKTE (639).
4018  E1                14          POP   HL                 ;Y-KOORDINATE HOLEN
4019  E5                15          PUSH  HL                 ;UND WIEDER SPEICHERN.
401A  CDF6BB            16          CALL  #BBF6              ;LINIE ZIEHEN.
401D  E1                17          POP   HL                 ;Y-KOORDINATE DES
401E  110800            18          LD    DE,#0008           ;ENDPUNKTS UM 8
4021  19                19          ADD   HL,DE              ;VERGROESSERN.
4022  C1                20          POP   BC                 ;SCHLEIFENZAEHLER B
4023  10E5              21          DJNZ  L1                 ;HOLEN UND PRUEFEN.
4025  C9                22          RET                      ;FERTIG.
                        23 ;P26

L1       400A
```

Der Koordinatenursprung liegt in der linken unteren Ecke. Die Endpunkte aller Linien haben die X-Koordinate 639. Die Y-Koordinaten werden immer um 8 vergrößert, so daß nach 50 Linien der Endpunkt der letzten Linie in der rechten oberen Ecke liegt. (Da der Punktabstand in vertikaler Richtung doppelt so groß ist wie der Punktabstand in horizontaler Richtung, werden von den Graphikroutinen Y-Koordinaten benutzt, die doppelt so groß sind wie die entsprechende Feinzeilennummer.) Vor dem Ziehen einer Linie wird der Graphikcursor mit der Routine &BBC0 an den Ausgangspunkt zurückgesetzt. Die Koordinaten werden in DE und HL übergeben. Wie aus der Tabelle zu ersehen ist, überschreiben die verwendeten Routinen alle Register außer IX und IY. Die benutzten Register müssen daher entweder zwischengespeichert oder neu gesetzt werden.

Nochmals: Die Uhr

Als zweites und abschließendes Beispiel für die Benutzung des Betriebssystems wollen wir uns die vom Hersteller vorgesehene Methode der Erstellung einer Interruptroutine anschauen. Als Programmbeispiel soll wieder unsere Softwareuhr dienen. Nur wollen wir jetzt, im Gegensatz zum ersten Programm, ausgiebig Gebrauch von den Betriebssystemsroutinen machen. Das fängt schon mit der

Initialisierung des Uhrenprogramms an. Statt das Systeminterruptprogramm abzuändern, übergeben wir dem Betriebssystem einige Daten über unser Uhrenprogramm. Am einfachsten ist das, wenn wir nicht den schnellen Takt benutzen, der alle 1/300 Sekunden erfolgt (Herstellerbezeichnung FAST TICKER), sondern einen untersetzten Interrupttakt (TICKER), der alle 1/50 Sekunden erzeugt wird. Die vom Betriebssystem benötigten Daten sind in zwei sechs Byte langen Blökken zusammengefaßt, die „Ereignis-" und „Taktblock" genannt werden. Der Ereignisblock muß direkt hinter dem Taktblock liegen. Wir müssen für diese Blökke Platz im zentralen RAM, also dem Bereich zwischen &4000 und &BFFF vorsehen. Den Ereignisblock kann der Benutzer über die Systemroutine KL INIT EVENT (&BCEF) mit Werten versorgen. Vor dem Aufruf dieser Routine müssen folgende Angaben gemacht werden:

1. HL enthält die Anfangsadresse des Ereignisblocks.
2. B enthält eine Prioritätsangabe.
3. C enthält die sog. ROM-Auswahladresse.
4. DE enthält die Adresse der Benutzerroutine (hier des Uhrenprogramms).

Während die Punkte 1. und 4. klar sein dürften, sind zu den Punkten 2. und 3. Erklärungen notwendig. Für die ROM-Auswahladresse können wir den Wert 0 setzen, wenn unsere Routine oberhalb &3FFF, also im zentralen RAM liegt.
In der Prioritätsangabe müssen wir Bit 0 und Bit 7 setzen, was den Wert &81 ergibt. Bit 0 enthält die Kennung dafür, daß sich unser Uhrenprogramm im zentralen RAM befindet, und das Setzen von Bit 7 garantiert, daß unser Programm nicht in eine Warteschlange eingereiht, sondern direkt ausgeführt wird. Das bedeutet aber nicht, daß das Uhrenprogramm bei jedem TICKER, also alle 1/50 Sekunden, aufgerufen wird.
Angaben über die Häufigkeit des Aufrufs müssen vielmehr in dem oben erwähnten Taktblock gemacht werden. Das geschieht mit Hilfe der Systemroutine &BCE9 (KL ADD TICKER). Vor dem Aufruf dieser Routine muß

1. HL mit der Anfangsadresse des Taktblocks,
2. DE mit dem Ausgangswert des Taktzählers,
3. BC mit dem Wiederanlaufwert des Taktzählers geladen werden.

Der Taktzähler wird bei jedem TICKER um den Wert 1 verringert. Erst wenn der Wert 0 erreicht ist, wird das Benutzerprogramm ausgeführt. Anschließend wird der Taktzähler auf den im BC angegebenen Wiederanlaufwert gesetzt.
Da Takt- und Ereignisblock vom Betriebssystem benutzt werden, sollten sie nur über die eben beschriebenen Routinen mit Daten versorgt werden. Ein direktes Beschreiben der Blöcke kann ohne genaue Kenntnisse aller Zusammenhänge zu Systemzusammenbrüchen führen.
Unsere neue Version des Uhrenprogramms hat folgendes Aussehen:

```
4000                    1          ORG   #4000
4000  21F240            2          LD    HL,#40F2
4003  0603              3          LD    B,#03
4005  CD06BB            4 LO:      CALL  #BB06
4008  ED6F              5          RLD
400A  CD06BB            6          CALL  #BB06
400D  ED6F              7          RLD
400F  2B                8          DEC   HL
4010  10F3              9          DJNZ  LO
```

```
4012  210641      10          LD    HL,#4106          ;ADR. EREIGNISBLOCK.
4015  0681        11          LD    B,#81             ;PRIORITAET.
4017  0E00        12          LD    C,#00             ;ROM-AUSWAHL.
4019  112C40      13          LD    DE,#402C          ;STARTADRESSE DES
                  14                                  ;EIGENEN IRQ-PROGRAMMS.
401C  CDEFBC      15          CALL  #BCEF             ;EREIGNISBLOCK
                  16                                  ;EINRICHTEN.
401F  210041      17          LD    HL,#4100          ;ADR. TAKTBLOCK.
4022  110100      18          LD    DE,#0001          ;TAKTZAEHLER ANF.WERT.
4025  013200      19          LD    BC,#0032          ;TAKTZAEHLER LADEWERT 50
4028  CDE9BC      20          CALL  #BCE9             ;UEBERGABE DES IRQ-PROG.
                  21                                  ;ANS BETRIEBSSYSTEM.
402B  C9          22          RET
                  23
402C  F3          24          DI                    ;START IRQ-PROGRAMM.
402D  B7          25          OR    A               ;CARRY AUS FUER DAA.
402E  21F040      26          LD    HL,#40F0
4031  1E60        27          LD    E,#60
4033  CD7B40      28          CALL  TICK
4036  380C        29          JR    C,UAU
4038  23          30          INC   HL
4039  CD7B40      31          CALL  TICK
403C  3806        32          JR    C,UAU
403E  23          33          INC   HL
403F  1E24        34          LD    E,#24
4041  CD7B40      35          CALL  TICK
4044  2A85B2      36 UAU:     LD    HL,(#B285)        ;ALTE CURSORPOSITION
4047  22F840      37          LD    (#40F8),HL        ;ZWISCHENSPEICHERN.
404A  2647        38          LD    H,#47             ;CURSORSPALTE (71).
404C  2E01        39          LD    L,#01             ;CURSORZEILE.
404E  CD75BB      40          CALL  #BB75             ;CURSOR SETZEN.
4051  CD84BB      41          CALL  #BB84             ;CURSOR UNSICHTBAR.
4054  21F240      42          LD    HL,#40F2          ;ADR. ZEITZAEHLER.
4057  0603        43          LD    B,#03             ;AUSGABE DER DREI
4059  AF          44          XOR   A                 ;ZEITZAEHLER (HH MM SS)
405A  ED6F        45 LA1:     RLD                     ;WIE IM DUMP-PROGRAMM.
405C  F630        46          OR    #30
405E  CD5ABB      47          CALL  #BB5A
4061  ED6F        48          RLD
4063  CD5ABB      49          CALL  #BB5A
4066  ED6F        50          RLD
4068  3E20        51          LD    A,#20
406A  CD5ABB      52          CALL  #BB5A
406D  2B          53          DEC   HL
406E  10EA        54          DJNZ  LA1
4070  2AF840      55          LD    HL,(#40F8)        ;ALTE CURSORPOSITION
4073  2285B2      56          LD    (#B285),HL        ;ZURUECKHOLEN.
4076  CD81BB      57          CALL  #BB81             ;CURSOR SICHTBAR.
4079  FB          58          EI
407A  C9          59          RET
                  60 ;
407B  7E          61 TICK:    LD    A,(HL)
407C  3C          62          INC   A
407D  27          63          DAA
407E  77          64          LD    (HL),A
407F  BB          65          CP    E
4080  D8          66          RET   C
4081  3600        67          LD    (HL),#00
4083  C9          68          RET
                  69 ;P27

LO      4005  LA1     405A  TICK    407B
UAU     4044
```

Änderungen für den CPC664/6128: In den Zeilen 36 und 56 muß jeweils die Adresse &B285 durch &B726 ersetzt werden.

So tickt die neue Uhr

Zunächst werden die Zeit-Daten wie in der ersten Version eingegeben. Dann werden ab den Zeilen 10 bzw. 17 (wie oben beschrieben) der Ereignis- und der Taktblock eingerichtet. Als Anfangsadressen dieser Blöcke wurden hier &4100 (für den Taktblock) und &4106 (für den Ereignisblock) gewählt. Die Anfangsadresse des Uhrenprogramms ist &402C. Der Wiederanlaufwert des Taktzählers wird auf 50 (&32) gesetzt, damit genau einmal pro Sekunde das Uhrenprogramm ausgeführt wird; ein eigener Taktzähler kann jetzt natürlich entfallen.
Im eigentlichen Uhrenprogramm ab Zeile 24 kann die Zwischenspeicherung der Register entfallen, dafür sorgt jetzt das Betriebssystem. Wichtig ist der Befehl OR A in Zeile 25, der die Carry-Flagge löscht, um eine fehlerfreie Dezimalkorrektur durch den DAA-Befehl sicherzustellen. Die Sekunden-, Minuten- und Stundenzählung funktioniert wie in der ersten Version. Dagegen wird der Ausgabeteil ab Zeile 36 mit Hilfe von Betriebssystemsroutinen stark gekürzt. Um die Uhrzeit ans Ende der ersten Zeile schreiben zu können, wird die momentane Cursorposition, die in der Speicherstelle &B285 steht (&B726 beim CPC664/6128), im Speicher &40F8 zwischengespeichert. Dann werden die neuen Werte für die Cursorspalte (71 (&47)) und die Cursorzeile (1) in die Register H und L gebracht, und schließlich wird in Zeile 40 der Cursor mit der Routine &BB75 auf diese Werte gesetzt. Um das Cursorviereck nicht in der Zeitanzeige erscheinen zu lassen, wird die Cursordarstellung mit der Routine &BB84 abgeschaltet und erst nach Beendigung der Zeitausgabe in Zeile 57 mit &BB81 wieder angeschaltet. Das macht sich durch ein kurzes Verlöschen des Cursors bei jeder Zeitausgabe bemerkbar. Die Ausgabe der Zeit erfolgt mit Hilfe der Systemroutine &BB5A und entspricht der im DUMP-Programm (P15) verwendeten Methode.
Wenn Sie das Uhrenprogramm aus dem Betriebssystem entfernen wollen, ohne das Gerät abzuschalten, benötigen Sie dazu die Löschroutine &BCEC (KL DEL TICKER). Laden Sie HL mit der Anfangsadresse (&4100) des Taktblocks unseres Uhrenprogramms und rufen Sie diese Routine auf, so ist die Uhr wieder abgestellt.
Für eine eingehende Beschreibung aller Möglichkeiten, die das Betriebssystem bietet, ist hier natürlich kein Platz. Die entsprechenden Informationen enthält das Firmware-Handbuch für den CPC.

Übung 43 Schreiben Sie ein Interruptprogramm, bei dem abwechselnd die Männchen mit CHR$(248) und CHR$(249) in der rechten oberen Ecke des Bildschirms ausgegeben werden. (Das ergibt eine Figur, die immer wieder dieselbe Bewegung vollführt.)
Hinweis: Nützen Sie das Uhrenprogramm P27 so gut wie möglich aus.

Anhang 26

Erläuterungen zu den Programmlisten A

Die in diesem Buch aufgelisteten Assemblerprogramme wurden mit dem Schneider-„HISOFT"-Assembler erstellt. Die Listen sind alle nach demselben Schema aufgebaut. Eine typische Zeile hat folgendes Aussehen:

```
4015 3E00    11 L1: LD A,#00    ;Low-Tonhöhenregister
```

Ganz links ist die Adresse angegeben, bei der der Befehl abgelegt ist. Dann folgt der Maschinencode des Befehls. Diese beiden Angaben sind in hexadezimaler Zahlendarstellung gemacht. Als nächstes kommt (dezimal) die Zeilennummer und eventuell ein „Label". Labels sind Bezeichnungen von Adressen von Sprungmarken, Unterprogrammen oder Tabellen. Rechts des Labels folgt der mnemonische Befehlscode und hinter einem Semikolon ein Kommentar.
Bei der Eingabe der abgedruckten Programme ist folgendes zu beachten: Bei Verwendung des BASIC-Laders wird einfach Byte für Byte des Maschinencodes (also der Spalte 2) in den DATA-Zeilen abgelegt. Dabei können natürlich mehrere Programmzeilen in einer DATA-Zeile zusammengefaßt werden.
Wenn der im Anhang abgedruckte Direktassembler verwendet wird, dürfen keine Zeilennummern, Labels und Kommentare, sondern nur die mnemonischen Assemblercodes eingegeben werden. Die Spalten 1 und 2 mit den Adressen und dem Maschinencode werden vom Assembler erzeugt. Bei der Eingabe der mnemonischen Codes müssen folgende Punkte beachtet werden:
Der erste Befehl ORG ... wird weggelassen; ORG enthält die Startadresse des Programms, die beim Direktassembler anders eingegeben wird (siehe Beschreibung des Direktassemblers, Anhang C). Der HISOFT-Assembler verwendet zur Kennzeichnung von Hexadezimalzahlen das „#"-Zeichen. Im Direktassembler muß das „#"-Zeichen durch das im Schneider-BASIC übliche „&" ersetzt werden. Beispiel: LD A,#08 muß durch LD A,&08 ersetzt werden.

Die in Sprung-, CALL- und LD-Befehlen verwendeten Labels müssen beim Direktassembler durch die entsprechenden Adressen ersetzt werden. Die einem Label zugeordnete Adresse kann entweder direkt aus dem Programm oder aus der im Anschluß an das Programm ausgedruckten Liste entnommen werden.
Beispiele:
Im Programm P3 muß in Zeile 8 statt JP NZ,L1 der Befehl JP NZ,&4005 geschrieben werden (L1=&4005).
Im Programm P7 muß in Zeile 7 statt CALL ZP der Befehl CALL &401A verwendet werden (ZP=&401A).
Im Programm P14 muß in Zeile 33 statt LD (XR+2),A der Befehl LD (&403F),A geschrieben werden (XR=&403D und XR+2=&403F).
Im Programm P25 muß in Zeile 13 statt LD HL,NPR der Befehl LD HL,&4025 geschrieben werden (NPR=&4025).
Für Zahlenangaben in Tabellen verwendet der HISOFT-Assembler den „Pseudobefehl" DEFB. Diese vom englischen „Define Byte" kommende Anweisung teilt dem Assembler mit, daß die Eingaben in der anschließenden Zeile direkt als (hexadezimale) Konstanten abgespeichert werden sollen. Im Direktassembler entfällt die DEFB-Anweisung, und die Konstanten werden unmittelbar eingegeben.
Beispiel:
Im Programm P14 heißt die Zeile 39

```
DEFB #80,#40,#20,#10
```

Für den Direktassembler werden statt dessen entweder die beiden Zeilen

```
&8040
&2010
```

oder die vier Zeilen

```
&80
&40
&20
&10
```

eingegeben.

Programmliste des Direktassemblers B

```
10 ON ERROR GOTO 420
20 DIM a1(14),a2(24),a3(13),a4(5),a6(6),a7(8),a8(5)
30 FOR i=0 TO 13:READ w:a1(i)=w:NEXT
40 FOR i=0 TO 23:READ w:a2(i)=w:NEXT
50 FOR i=0 TO 12:READ w:a3(i)=w:NEXT
60 FOR i=0 TO 4:READ w:a4(i)=w:NEXT
70 FOR i=0 TO 5:READ w:a6(i)=w:NEXT
80 FOR i=0 TO 7:READ w:a7(i)=w:NEXT
90 FOR i=0 TO 4:READ w:a8(i)=w:NEXT
100 b1$="CCF CPL DAA DI  EI  EXX HALTNOP RET RLA RLCARRA RRCASCF"
110 b2$="CPD* CPDR*CPI* CPIR*IMO* IM1* IM2* IND* INDR*INI* INIR*LDD* LDDR*LDI* L
DIR*NEG* OTDR*OTIR*OUTD*OUTI*RETI*RETN*RLD* RRD*"
120 b3$="RL RLCRR RRCSLASRASRLDECINC]]]]]]POPPUSH"
130 AR1$="A*B*C*D*E*H*L*(HL)*"
140 AR2$="BC*DE*HL*SP*AF*IX*IY*"
150 b4$="ANDCP OR SUBXOR"
160 b5$="BITRESSET"
170 b6$="ADC*ADD*SBC*LD*"
180 i$="(IX(IY"
190 AR3$="M* P* PE*PO*C* NC*Z* NZ*"
200 b7$="CALLJP  JR  DJNZRET RST"
210 AR4$="A,(BC)A,(DE)(BC),A(DE),AI,A   A,I   R,A   A,R   SP,HL SP,IX SP,IY"
220 AR5$="(SP),HLAF,AF  DE,HL  (SP),IX(SP),IY"
230 DATA &3f,&2f,&27,&f3,&fb,&d9,&76,&00,&c9,&17,&07,&1f,&0f,&37
240 DATA &A9,&B9,&A1,&B1,&46,&56,&5e,&AA,&BA,&A2,&B2,&A8,&B8,&A0,&B0,&44,&8b,&b3
,&ab,&a3,&4d,&45,&6f,&67
250 DATA &10,0,&18,8,&20,&28,&38,5,4,11,3,&C1,&C5
260 DATA &A0,&B8,&B0,&90,&AB
270 DATA &88,&80,&98,&4a,9,&42
280 DATA &A,&1A,2,&12,&47,&57,&4F,&5F
290 DATA &E3,8,&EB,&DDE3,&FDE3
300 DATA 6,1,&3E,0,&CD,&4D,&BC,&C9
310 FOR I=100 TO 107:READ W:POKE I,W:NEXT
320 MODE 2:nf=0
330 INPUT "STARTADRESSE  ",bz:bza=bz
340 LOCATE 25,3:GOTO 430
350 PRINT HEX$(bz);"    ";
360 m=mc1:GOSUB 2670:GOTO 380
370 PRINT HEX$(bz);"    ";:GOTO 390
380 nf=1
390 m=mc:GOSUB 2670
400 IF VPOS(#0)>22 THEN CALL 100:LOCATE 25,23:GOTO 430
410 LOCATE 25,VPOS(#0)+1:GOTO 430
420 LOCATE 25,VPOS(#0)
430 LINE INPUT "",ac$:LOCATE 2,VPOS(#0)-1
440 IF ac$="END" THEN END
450 IF ac$="G" THEN CALL bza:END
460 IF ac$="S" THEN SAVE"OBJ",B,bza,bz-bza
470 IF LEFT$(ac$,1)="&" THEN mc=VAL(ac$):GOTO 370
480 IF LEFT$(ac$,2)<>"A&" GOTO 500
490 bz=VAL(RIGHT$(ac$,5)):GOTO 400
500 kof=INSTR(ac$,",")
510 blf=INSTR(ac$," ")-1
520 IF kof>0 GOTO 1150
530 IF blf>0 GOTO 630
540 REM ************ CODES OHNE OPERAND ******************
550 i=(INSTR(b1$,ac$)-1)/4
560 IF i<0 THEN 580
```

```
570 mc=a1(i):GOTO 370
580 ac1$=ac$+"*"
590 i=(INSTR(b2$,ac1$)-1)/5
400 IF i<0 THEN 420
610 mc=&ED*256+a2(i):GOTO 370
620 REM ************ CODES 1 OP 1. GR    ******************
630 ac1$=LEFT$(ac$,b1f)
640 j=LEN(ac$)-LEN(ac1$)-1
650 op$=RIGHT$(ac$,j)+"*"
660 GOSUB 2460
670 i=(INSTR(b3$,ac1$)-1)/3
680 IF i<0 GOTO 860
690 ON ARF GOTO 720,420,760,700
700 IF i>6 THEN jf=4:aa=a3(i)+42:GOTO 2340
710 jf=3:bb=&CB:aa=a3(i):GOTO 2340
720 IF i<7 GOTO 750
730 IF i>10 GOTO 420
740 mc=a3(i)+8*zu:GOTO 370
750 mc=&CB*256+a3(i)+zu:GOTO 370
760 IF i<7 GOTO 420
770 i2=i+2:IF i>10 THEN i2=i
780 IF zu>2 GOTO 800
790 mc=a3(i2)+&10*zu:GOTO 370
800 IF zu>4 GOTO 840
810 IF zu=3 AND i<11 GOTO 790
820 IF zu=4 AND i>10 THEN zu=3:GOTO 790
830 GOTO 420
840 mc=(&DD+&20*(zu-5))*256+a3(i2)+&20:GOTO 370
850 REM ************ CODES 1 OP 2. GR    ******************
860 i=(INSTR(b4$,ac1$)-1)/3
870 IF i<0 GOTO 930
880 ON ARF GOTO 910,900,420,890
890 jf=4:aa=a4(i):GOTO 2340
900 mc=(a4(i)+&46)*256+VAL(op$):GOTO 370
910 mc=a4(i)+zu:GOTO 370
920 REM ************ CODES 1 OP 3. GR    ******************
930 i=(INSTR(b7$,ac1$)-1)/4+1
940 ON i GOTO 960,960,980,980,1070,1100
950 GOTO 420
960 IF ARF<>2 GOTO 1020
970 mc1=&D7-10*i:mc=VAL(oq$):GOTO 350
980 IF ARF<>2 GOTO 420
990 GOSUB 2630
1000 IF e<0 OR e>255 THEN 420
1010 mc=(&30-8*i)*256+e:GOTO 370
1020 IF i<>2 GOTO 420
1030 IF op$="(HL)*" THEN mc=&E9:GOTO 370
1040 IF op$="(IX)*" THEN mc=&DD*256+&E9:GOTO 370
1050 IF op$="(IY)*" THEN mc=&FD*256+&E9:GOTO 370
1060 GOTO 420
1070 IF ARF<>5 THEN GOSUB 2560
1080 IF ARF<>5 GOTO 420
1090 mc=&C0+8*zu:GOTO 370
1100 IF ARF<>2 GOTO 420
1110 zu=VAL(op$)
1120 IF zu>&38 OR zu<>8*INT(zu/8) THEN 420
1130 mc=&C7+zu:GOTO 370
1140 REM ************ CODES 2 OP 1. GR    ******************
1150 ac1$=LEFT$(ac$,3)
1160 i=(INSTR(b5$,ac1$)-1)/3+1
1170 IF i<1 GOTO 1260
1180 j$=MID$(ac$,5,1):j=VAL(j$)
1190 ko=LEN(ac$)-kof
1200 op$=RIGHT$(ac$,ko)+"*"
1210 GOSUB 2460
1220 ON ARF GOTO 1240,420,420,1230
1230 jf=3:bb=&CB:aa=&40*i+8*j:GOTO 2340
```

```
1240 mc=&CB*256+&40*i+8*j+zu:GOTO 370
1250 REM ********** CODES 2 OP 2. GR    ********************
1260 ac1$=LEFT$(ac$,3)+"*"
1270 i=(INSTR(b6$,ac1$)-1)/4
1280 zz=kof-blf-2
1290 ol$=MID$(ac$,blf+2,zz)
1300 ko=LEN(ac$)-kof
1310 op$=RIGHT$(ac$,ko)+"*"
1320 GOSUB 2460
1330 IF i<0 GOTO 1540
1340 IF ol$<>"A" GOTO 1400
1350 ON ARF GOTO 1370,1380,420,1360
1360 jf=4:aa=a6(i):GOTO 2340
1370 mc=a6(i)+zu:GOTO 370
1380 mc=(a6(i)+&46)*256+VAL(op$):GOTO 370
1390 REM ********** ADC HL,ZZ usw. ************************
1400 IF ARF<>3 GOTO 420
1410 IF ol$<>"HL" GOTO 1460
1420 IF zu>3 GOTO 420
1430 IF i=1 GOTO 1450
1440 mc=256*&ED+a6(i+3)+zu*&10:GOTO 370
1450 mc=a6(4)+zu*&10:GOTO 370
1460 IF ol$="IX" AND i=1 THEN zv=5:GOTO 1490
1470 IF ol$="IY" AND i=1 THEN zv=6:GOTO 1490
1480 GOTO 420
1490 IF zu=2 THEN 420
1500 IF zu=zv THEN zu=2
1510 IF zu>3 THEN 420
1520 mc=256*(&DD+(zv-5)*&20)+a6(4)+zu*&10:GOTO 370
1530 REM ********** CODES 2 OP 3. GR    ********************
1540 ac1$=LEFT$(ac$,blf)
1550 i=(INSTR(b7$,ac1$)-1)/4
1560 IF i<0 OR i>2 GOTO 1670
1570 IF ARF<>2 GOTO 420
1580 oz$=op$:op$=ol$:GOSUB 2560
1590 IF ARF<>5 GOTO 420
1600 IF i=2 AND zu>3 GOTO 420
1610 IF i=2 GOTO 1630
1620 mc1=&C4-2*i+8*zu:mc=VAL(oq$):GOTO 350
1630 op$=oz$:GOSUB 2630
1640 IF e<0 OR e>255 GOTO 420
1650 mc=(&20+8*zu)*&100+e:GOTO 370
1660 REM **********     LD-CODES          ********************
1670 IF LEFT$(ac$,2)<>"LD" GOTO 2120
1680 ARR=ARF:zr=zu:os$=op$:op$=ol$+"*":GOSUB 2460
1690 IF ARF<>1 GOTO 1780
1700 ON ARR GOTO 1720,1740,420,1760
1710 GOTO 1830
1720 IF zu=6 AND zr=6 THEN 420
1730 mc=64+8*zu+zr:GOTO 370
1740 IF VAL(os$)>255 THEN 420
1750 mc=(6+8*zu)*&100+VAL(os$):GOTO 370
1760 IF zu=6 GOTO 420
1770 jf=4:aa=64+8*zu:op$=os$:GOTO 2340
1780 IF ARF<>4 OR ARR<>1 GOTO 1810
1790 IF zr=6 GOTO 420
1800 jf=4:aa=&6A+zr:GOTO 2340
1810 IF ARF<>4 OR ARR<>2 GOTO 1830
1820 jf=3:bb=&36:aa=VAL(os$)-6:GOTO 2340
1830 j=LEN(ac$)-3:oo$=RIGHT$(ac$,j)
1840 i=(INSTR(AR4$,oo$)-1)/6
1850 IF i<0 GOTO 1910
1860 IF i>7 GOTO 1890
1870 mc=a7(i):IF i>3 THEN mc=&ED*256+mc
1880 GOTO 370
1890 mc=&F9:IF i>8 THEN mc=256*(&DD+&20*(i-9))+mc
1900 GOTO 370
```

```
1910 IF ARF<>3 OR ARR<>2 GOTO 1950
1920 IF zu>4 GOTO 1940
1930 mc1=1+&10*zu:mc=VAL(oq$):GOTO 350
1940 mc1=256*(&DD+(zu-5)*&20)+&21:mc=VAL(oq$):GOTO 350
1950 IF ARF+ARR<>20 GOTO 2060
1960 mc=VAL(oq$)
1970 zs=&10*zr
1980 IF ARF<ARR THEN zs=&10*zu+8
1990 IF zs<>&20 AND zs<>&28 THEN 2010
2000 mc1=zs+2:GOTO 350
2010 IF zs=&40 OR zs=&48 GOTO 420
2020 IF zs>&48 GOTO 2040
2030 mc1=256*&ED+&43+zs:GOTO 350
2040 zh=INT((zs-80)/16)
2050 mc1=256*(&DD+&20*zh)+&22+zs-&50-16*zh:GOTO 350
2060 mc=VAL(oq$)
2070 IF os$<>"A*" OR ARF<>17 GOTO 2090
2080 mc1=&32:GOTO 350
2090 IF op$<>"A*" OR ARR<>17 GOTO 420
2100 mc1=&3A:GOTO 350
2110 REM ************ CODES FUER EX      ******************
2120 IF LEFT$(ac$,2)<>"EX" GOTO 2180
2130 j=LEN(ac$)-3:oo$=RIGHT$(ac$,j)
2140 i=(INSTR(AR5$,oo$)-1)/7
2150 IF i<0 GOTO 420
2160 mc=a8(i):GOTO 370
2170 REM ************ CODES FUER OUT      ******************
2180 IF LEFT$(ac$,7)<>"OUT (C)" GOTO 2220
2190 aa=&41
2200 IF ARF<>1 OR zu=6 GOTO 420
2210 mc=256*&ED+aa+8*zu:GOTO 370
2220 IF LEFT$(ac$,6)<>"OUT (&" GOTO 2270
2230 IF RIGHT$(ac$,1)<>"A" GOTO 420
2240 op$=MID$(ac$,6,3)
2250 mc=&D3*256+VAL(op$):GOTO 370
2260 REM ************ CODES FUER IN      ******************
2270 IF LEFT$(ac$,2)<>"IN" GOTO 420
2280 IF RIGHT$(ac$,3)<>"(C)"  GOTO 2300
2290 op$=MID$(ac$,4,1)+"*":GOSUB 2460:aa=&40:GOTO 2200
2300 IF LEFT$(ac$,4)<>"IN A" GOTO 420
2310 op$=RIGHT$(ac$,4)
2320 mc=256*&DB+VAL(op$):GOTO 370
2330 REM ****************** INDEXREGISTER ****************
2340 oq$=LEFT$(op$,3)
2350 ii=(INSTR(i$,oq$)-1)/3
2360 IF ii<0 THEN 420
2370 iz=VAL(RIGHT$(op$,5))
2380 ON jf GOTO 420,420,2390,2420
2390 mc1=256*(&DD+&20*ii)+bb
2400 mc=256*iz+aa+6
2410 GOTO 350
2420 mc1=&DD+&20*ii
2430 mc=256*(aa+6)+iz
2440 GOTO 350
2450 REM ******************** PRUEFUNG DES OP. ****************
2460 zu=INSTR(AR1$,op$)
2470 IF zu=0 THEN 2500
2480 zu=(zu-1)/2-1:IF zu=-1 THEN zu=7
2490 ARF=1:RETURN
2500 IF LEFT$(op$,1)<>"&" THEN 2540
2510 ARF=2
2520 IF LEN(op$)<5 THEN RETURN
2530 oq$="&"+MID$(op$,4,2)+MID$(op$,2,2):RETURN
2540 zu=(INSTR(AR2$,op$)-1)/3
2550 IF zu>=0 THEN ARF=3:RETURN
2560 zu=7-(INSTR(AR3$,op$)-1)/3
2570 IF zu<=7 THEN ARF=5:RETURN
```

```
2580 oi$=LEFT$(op$,4)
2590 IF oi$="(IX+" OR oi$="(IY+" THEN ARF=4:RETURN
2600 IF LEFT$(op$,2)<>"(&" THEN ARF=0:RETURN
2610 ARF=17:op$=MID$(op$,2,5):GOTO 2530
2620 REM ********** REL. SPRUNGWEITE **********
2630 e=VAL(op$)-bz-2
2640 IF e<0 THEN e=256+e
2650 RETURN
2660 REM **********   LADETEIL   **********
2670 IF ABS(m)<256 AND nf=0 THEN mm=m:GOTO 2740
2680 IF m<0 THEN m=m+65536
2690 mm=INT(m/256)
2700 m1=INT(mm/16):PRINT HEX$(m1);
2710 m2=mm-16*m1:PRINT HEX$(m2);
2720 POKE bz,mm:bz=bz+1
2730 mm=m-256*mm
2740 m1=INT(mm/16):PRINT HEX$(m1);
2750 m2=mm-16*m1:PRINT HEX$(m2);
2760 POKE bz,mm:bz=bz+1
2770 nf=0
2780 RETURN

10000 CLS:INPUT "Zeilenzahl  ";zeiza:adr=&170:zg=0
10001 ZONE 15
10002 CLS
10003 PRINT "Zeilennummern                    Kontrollzahl":PRINT:PRINT
10005 zeina=PEEK(adr+2)+256*PEEK(adr+3)
10010 FOR kk=1 TO zeiza
10020 zz=0
10030 adre=adr+PEEK(adr)+256*PEEK(adr+1)-1
10040 IF PEEK(adr+4)=&C5 THEN 10090
10050 FOR i=adr+2 TO adre
10060 IF PEEK(I)=&20 THEN 10080
10070 zz=zz+PEEK(i)
10080 NEXT:zg=zg+zz
10090 zeinu=PEEK(adr+2)+256*PEEK(adr+3)
10095 IF zef=1 THEN zeina=zeinu:zef=0
10100 IF 10*INT(kk/10)=kk THEN PRINT zeina;"-";zeinu,,zg:zg=0:zef=1
10105 adr=adre+1
10110 NEXT kk
10120 PRINT zeina;"-";zeinu,,zg
```

Eingabe und Bedienung des Direktassemblers C

Da eine fehlerfreie Eingabe der fast 280 Programmzeilen des Direktassemblers sehr schwierig ist, wurde zur Kontrolle der Fehlerfreiheit ein Prüfsummenprogramm beigefügt. Geben Sie dieses Programm mit den Zeilennummern ab 10000 zuerst ein. Bei der Eingabe des Assemblerprogramms können Sie nach je zehn eingegebenen Programmzeilen die Prüfsumme berechnen und mit den unten angegebenen Werten vergleichen. Dazu starten Sie das Programm mit RUN 10000 und geben die Anzahl der bisher eingetippten Zeilen an. Es werden die Zeilennummern, bis zu denen die Prüfsumme gebildet wurde, und die Prüfsumme ausgegeben. REM-Zeilen müssen eingegeben werden, ihr Inhalt bleibt jedoch bei der Prüfsummenbildung genauso wie „Blanks" in allen übrigen Zeilen unberücksichtigt.

Zeilennummern			Kontrollzahl
10	-	100	26924
110		200	25884
210	-	300	24469
310	-	400	15535
410	-	500	21447
510		600	14208
610	-	700	18662
710	-	800	19942
810	-	900	19572
910		1000	16704
1010	-	1100	19331
1110	-	1200	19297
1210	-	1300	21139
1310	-	1400	15112
1410		1500	20622
1510		1600	18543
1610	-	1700	21345
1710	-	1800	21502
1810		1900	22059
1910		2000	23337
2010		2100	21298
2110		2200	15418
2210	-	2300	20931
2310	-	2400	17419
2410		2500	15461
2510	-	2600	22301
2610		2700	16813
2710		2780	16438

Nach dem Laden und Starten mit RUN meldet sich der Direktassembler mit dem Wort „STARTADRESSE". Diese Adresse muß als vierstellige Hexzahl eingegeben werden, die größer als &3000 sein soll. Danach können die mnemonischen Assemblerbefehle eingegeben werden, die der Assembler direkt nach der Eingabe in Maschinencode übersetzt und an die jeweils angezeigte Adresse lädt. Bei der Eingabe ist zu beachten:

1. Großschreibung der mnemonischen Codes
Es müssen Großbuchstaben eingegeben werden. (Arbeiten Sie mit CAPS-LOCK.)

2. Zahlenangaben
Alle Zahlen müssen als Hexadezimalzahlen mit vorgestelltem „&" eingegeben werden. Vierstellige Zahlen (z. B. Adreßangaben) müssen vierstellig, zweistellige Zahlen (z. B. Indexverschiebungen) müssen zweistellig eingegeben werden. Also: &00C0 statt &C0 bzw. &08 statt &8.

3. Relative Sprünge
Bei relativen Sprüngen JR muß die Zieladresse angegeben werden (genauso wie bei absoluten Sprüngen JP). Die relative Sprungweite wird vom Assembler berechnet.

4. Hexcode-Eingabe
Statt mnemonischer Befehle können auch zweistellige Hexadezimalzahlen eingegeben werden. Das ist für die Eingabe von Tabellen wichtig. Diese Zahlen werden an der angegebenen Adresse direkt abgespeichert.

5. Neue Adresse
Die Eingaben werden normalerweise an fortlaufenden Adressen abgespeichert. Soll die Eingabe an einer anderen Adresse abgespeichert werden, z. B. zur Korrektur einer falschen Eingabe, so kann statt eines Befehls eine neue Adresse als vierstellige Hexadezimalzahl mit vorangestelltem „A" eingegeben werden. Also: A&5000 setzt den Adreßzähler auf den Wert &5000.

6. Abweichungen vom Standard
Die Befehle IM 0, IM 1 und IM 2 müssen als IM0, IM1 und IM2 eingegeben werden.

7. Starten des Maschinenprogramms
Mit dem Kommando „G" wird die Eingabe beendet und das eingespeicherte Maschinenprogramm von der Startadresse ab ausgeführt. Das Kommando „END" beendet die Eingabe, ohne das Programm auszuführen. Das Programm kann dann natürlich mit dem CALL-Befehl gestartet werden.

8. Abspeichern
Mit dem Kommando „S" wird das Maschinenprogramm unter dem Namen OBJ auf Kassette abgespeichert. Der Assemblertext kann nicht abgespeichert werden.

9. Fehler bei der Eingabe
Wenn bei der Eingabe ein Syntaxfehler erkannt wird, geht der Cursor auf den Anfang des Befehls zurück. Der Befehl muß dann völlig neu geschrieben werden. Da das Programm einige Syntaxfehler nicht erkennt, muß man bei der Eingabe sehr sorgfältig sein.

10. NOP schafft Platz

Anders als bei BASIC-Programmen können in Maschinenprogramme nur neue Befehle zur Korrektur oder Erweiterung eingefügt werden, wenn vorher Speicherplatz für diese Befehle vorgesehen ist. Es empfiehlt sich daher beim Schreiben eines neuen Programms, in regelmäßigen Abständen einige NOP-Befehle einzufügen. Da die NOP-Befehle keinen Einfluß auf den Programmablauf haben, können sie bei Bedarf durch andere Befehle ersetzt werden. Auch zum Überschreiben überflüssiger Befehle kann NOP verwendet werden.

Ein Direktassembler kann nur als Notbehelf für den ersten Kontakt mit der Maschinensprache angesehen werden. Für eine intensivere Beschäftigung mit der Materie und für die Erstellung umfangreicherer Programme ist jedoch die Anschaffung eines Assemblers, der symbolische Adressen, Variablennamen und Labels verarbeiten kann, dringend zu empfehlen.

Z80-Befehlsliste

Veränderliche Teile der Operanden:
nn: 8-Bit-Konstante
ee: 8-Bit-Sprungweite, relativ
dd: 8-Bit-Verschiebung bei Indizierung
hhll: 16-Bit-Wert mit dem High-Byte hh und Low-Byte ll

Operations-Code (hex.) Maschinen-Code	mnemonischer Code Assembler-Befehl	Befehlsbeschreibung
8E	ADC A,(HL)	**Addiert den Inhalt des zweiten Operanden und der Carry-Flagge zum Inhalt des Akkumulators.**
DD8Edd	ADC A,(IX+dd)	
FD8Edd	ADC A,(IY+dd)	
8F	ADC A,A	
88	ADC A,B	
89	ADC A,C	
8A	ADC A,D	
8B	ADC A,E	
8C	ADC A,H	
8D	ADC A,L	
CEnn	ADC A,nn	
ED4A	ADC HL,BC	**Addiert den Inhalt des zweiten Operanden und der Carry-Flagge zum Inhalt des HL-Registers.**
ED5A	ADC, HL,DE	
ED6A	ADC, HL,HL	
ED7A	ADC, HL,SP	
86	ADD A,(HL)	**Addiert den Inhalt des Operanden zu dem des Akkumulators.**
DD86dd	ADD A,(IX+dd)	
FD86dd	ADD A,(IY+dd)	
87	ADD A,A	
80	ADD A,B	
81	ADD A,C	
82	ADD A,D	
83	ADD A,E	
84	ADD A,H	
85	ADD A,L	
C6nn	ADD A,nn	
09	ADD HL,BC	**Addiert den Inhalt des zweiten Operanden zu dem des ersten Operanden (HL, IX, IY).**
19	ADD HL,DE	
29	ADD HL,HL	
39	ADD HL,SP	
DD09	ADD IX,BC	
DD19	ADD IX,DE	
DD29	ADD IX,IX	
DD39	ADD IX,SP	

Operations-Code (hex.) Maschinen-Code	mnemonischer Code Assembler-Befehl		Befehlsbeschreibung
FD09	ADD	IY,BC	
FD19	ADD	IY,DE	
FD29	ADD	IY,IY	
FD39	ADD	IY,SP	
A6	AND	(HL)	Logisches UND (AND) zwischen Akkumulator und Operanden.
DDA6dd	AND	(IX+dd)	
FDA6dd	AND	(IY+dd)	
A7	AND	A	
A0	AND	B	
A1	AND	C	
A2	AND	D	
A3	AND	E	
A4	AND	H	
A5	AND	L	
E6nn	AND	nn	
CB46	BIT	0,(HL)	Fragt das angegebene Bit des Operanden ab.
DDCBdd46	BIT	0,(IX+dd)	
FDCBdd46	BIT	0,(IY+dd)	
CB47	BIT	0,A	
CB40	BIT	0,B	
CB41	BIT	0,C	
CB42	BIT	0,D	
CB43	BIT	0,E	
CB44	BIT	0,H	
CB45	BIT	0,L	
CB4E	BIT	1,(HL)	
DDCBdd4E	BIT	1,(IX+dd)	
FDCBdd4E	BIT	1,(IY+dd)	
CB4F	BIT	1,A	
CB48	BIT	1,B	
CB49	BIT	1,C	
CB4A	BIT	1,D	
CB4B	BIT	1,E	
CB4C	BIT	1,H	
CB4D	BIT	1,L	
CB56	BIT	2,(HL)	
DDCBdd56	BIT	2,(IX+dd)	
FDCBdd56	BIT	2,(IY+dd)	
CB57	BIT	2,A	
CB50	BIT	2,B	
CB51	BIT	2,C	
CB52	BIT	2,D	
CB53	BIT	2,E	
CB54	BIT	2,H	
CB55	BIT	2,L	
CB5E	BIT	3,(HL)	
DDCBdd5E	BIT	3,(IX+dd)	
FDCBdd5E	BIT	3,(IY+dd)	
CB5F	BIT	3,A	
CB58	BIT	3,B	

Operations-Code (hex.) Maschinen-Code	mnemonischer Code Assembler-Befehl		Befehlsbeschreibung
CB59	BIT	3,C	
CB5A	BIT	3,D	
CB5B	BIT	3,E	
CB5C	BIT	3,H	
CB5D	BIT	3,L	
CB66	BIT	4,(HL)	
DDCBdd66	BIT	4,(IX+dd)	
FDCBdd66	BIT	4,(IY+dd)	
CB67	BIT	4,A	
CB60	BIT	4,B	
CB61	BIT	4,C	
CB62	BIT	4,D	
CB63	BIT	4,E	
CB64	BIT	4,H	
CB65	BIT	4,L	
CB6E	BIT	5,(HL)	
DDCBdd6E	BIT	5,(IX+dd)	
FDCBdd6E	BIT	5,(IY+dd)	
CB6F	BIT	5,A	
CB68	BIT	5,B	
CB69	BIT	5,C	
CB6A	BIT	5,D	
CB6B	BIT	5,E	
CB6C	BIT	5,H	
CB6D	BIT	5,L	
CB76	BIT	6,(HL)	
DDCBdd76	BIT	6,(IX+dd)	
FDCBdd76	BIT	6,(IY+dd)	
CB77	BIT	6,A	
CB70	BIT	6,B	
CB71	BIT	6,C	
CB72	BIT	6,D	
CB73	BIT	6,E	
CB74	BIT	6,H	
CB75	BIT	6,L	
CB7E	BIT	7,(HL)	
DDCBdd7E	BIT	7,(IX+dd)	
FDCBdd7E	BIT	7,(IY+dd)	
CB7F	BIT	7,A	
CB78	BIT	7,B	
CB79	BIT	7,C	
CB7A	BIT	7,D	
CB7B	BIT	7,E	
CB7C	BIT	7,H	
CB7D	BIT	7,L	
DCllhh	CALL	C,hhll	**Ruft das Unterprogramm bei hhll, falls angegebene Bedingung wahr ist.**
FCllhh	CALL	M,hhll	
D4llhh	CALL	NC,hhll	
C4llhh	CALL	NZ,hhll	
F4llhh	CALL	P,hhll	
ECllhh	CALL	PE,hhll	

Operations-Code (hex.) Maschinen-Code	mnemonischer Code Assembler-Befehl	Befehlsbeschreibung
E4llhh	CALL PO,hhll	
CCllhh	CALL Z,hhll	
CDllhh	CALL hhll	Ruft das Unterprogramm bei hhll.
3F	CCF	Setzt Carry-Flagge auf entgegengesetzten Wert (Complement).
BE	CP (HL)	Vergleich von Akkumulator und Operanden (Compare).
DDBEdd	CP (IX+dd)	
FDBEdd	CP (IY+dd)	
BF	CP A	
B8	CP B	
B9	CP C	
BA	CP D	
BB	CP E	
BC	CP H	
BD	CP L	
FEnn	CP nn	
EDA9	CPD	Vergleicht Akku und Speicherstelle (HL) (Compare.) Vermindert HL und BC um 1 (Decrement).
EDB9	CPDR	Wie CPD, aber wiederholt, bis BC = 0 (Repeat).
EDA1	CPI	Vergrößert HL und vermindert BC um 1 (Increment).
EDB1	CPIR	Wie CPI, aber wiederholt bis BC = 0 (Repeat).
2F	CPL	Einerkomplement des Akkumulators (Complement Accu).
27	DAA	Dezimalkorrektur des Akkumulators (Decimal Adjust Accu).
35	DEC (HL)	Vermindert den Inhalt des Operanden um 1 (Decrement Operand).
DD35dd	DEC (IX+dd)	
FD35dd	DEC (IY+dd)	
3D	DEC A	
05	DEC B	
0B	DEC BC	
0D	DEC C	
15	DEC D	
1B	DEC DE	
1D	DEC E	
25	DEC H	
2B	DEC HL	
DD2B	DEC IX	
FD2B	DEC IY	
2D	DEC L	
3B	DEC SP	

Operations-Code (hex.) Maschinen-Code	mnemonischer Code Assembler-Befehl		Befehlsbeschreibung	
F3	DI		Verhindert Interrupts (Disable Interrupts).	
10ee	DJNZ	ee	Vermindert B und springt relativ, wenn B≠0 (Decrement and Jump if not Zero).	
FB	EI		Ermöglicht Interrupts (Enable Interrupts).	
E3	EX	(SP),HL	Tauscht den Inhalt ... (Exchange)	... der Stapelspeicher-stelle (SP) mit dem des zweiten Operanden
DDE3	EX	(SP),IX		
FDE3	EX	(SP),IY		
08	EX	AF,AF'		... von AF mit dem von A'F'
EB	EX	DE,HL		... von DE mit dem von HL
D9	EXX			... von BC,DE,HL mit B'C', D'E', H'L
76	HALT		Wartet bis zum nächsten Interrupt.	
ED46	IM	0	Setzt den Interrupt-Modus.	
ED56	IM	1		
ED5E	IM	2		
ED78	IN	A,(C)	Lädt den ersten Operanden mit dem Inhalt von Port (BC). (Input)	
ED40	IN	B,(C)		
ED48	IN	C,(C)		
ED50	IN	D,(C)		
ED58	IN	E,(C)		
ED60	IN	H,(C)		
ED68	IN	L,(C)		
34	INC	(HL)	Vergrößert den Operanden um den Wert 1 (Increment).	
DD34dd	INC	(IX+dd)		
FD34dd	INC	(IY+dd)		
3C	INC	A		
04	INC	B		
03	INC	BC		
0C	INC	C		
14	INC	D		
13	INC	DE		
1C	INC	E		
24	INC	H		
23	INC	HL		
DD23	INC	IX		
FD23	INC	IY		
2C	INC	L		
33	INC	SP		
DBnn	IN	A,(nn)	Lädt den Akkumulator mit dem Inhalt von Port (nn) (Input).	

Operations-Code (hex.) Maschinen-Code	mnemonischer Code Assembler-Befehl		Befehlsbeschreibung
EDAA	IND		Lädt Speicherstelle (HL) mit dem Inhalt des Ports (BC) (Input). Vermindert HL und B um 1.
EDBA	INDR		Wie IND, aber wiederholt (Repeat) bis B = 0.
EDA2	INI		Vergrößert HL und vermindert B jeweils um 1.
EDB2	INIR		Wie INI,aber wiederholt bis B = 0.
C3llhh	JP	hhll	Sprung zur angegebenen Speicherstelle (Jump).
E9	JP	(HL)	
DDE9	JP	(IX)	
FDE9	JP	(IY)	
DAllhh	JP	C,hhll	Sprung zur angegebenen Speicherstelle, falls die angegebene Bedingung wahr ist (Jump).
FAllhh	JP	M,hhll	
D2llhh	JP	NC,hhll	
C2llhh	JP	NZ,hhll	
F2llhh	JP	P,hhll	
EAllhh	JP	PE,hhll	
E2llhh	JP	PO,hhll	
CAllhh	JP	Z,hhll	
38ee	JR	C,ee	Relativer Sprung um ee Speicherplätze, falls die angegebene Bedingung wahr ist (Jump Relative).
30ee	JR	NC,ee	
20ee	JR	NZ,ee	
28ee	JR	Z,ee	
18ee	JR	ee	Unbedingter relativer Sprung um ee Speicherplätze.
02	LD	(BC),A	Lädt den ersten Operanden mit dem Inhalt des zweiten Operanden (Load).
12	LD	(DE),A	
77	LD	(HL),A	
70	LD	(HL),B	
71	LD	(HL),C	
72	LD	(HL),D	
73	LD	(HL),E	
74	LD	(HL),H	
75	LD	(HL),L	
36nn	LD	(HL),nn	
DD77dd	LD	(IX+dd),A	
DD70dd	LD	(IX+dd),B	
DD71dd	LD	(IX+dd),C	
DD72dd	LD	(IX+dd),D	
DD73dd	LD	(IX+dd),E	
DD74dd	LD	(IX+dd),H	
DD75dd	LD	(IX+dd),L	
DD36ddnn	LD	(IX+dd),nn	
FD77dd	LD	(IY+dd),A	
FD70dd	LD	(IY+dd),B	
FD71dd	LD	(IY+dd),C	
FD72dd	LD	(IY+dd),D	
FD73dd	LD	(IY+dd),E	

Operations-Code (hex.) Maschinen-Code	mnemonischer Code Assembler-Befehl		Befehlsbeschreibung
FD74dd	LD	(IY+dd),H	
FD75dd	LD	(IY+dd),L	
FD36ddnn	LD	(IY+dd),nn	
32llhh	LD	(hhll),A	
ED43llhh	LD	(hhll),BC	
ED53llhh	LD	(hhll),DE	
22llhh	LD	(hhll),HL	
DD22llhh	LD	(hhll),IX	
FD22llhh	LD	(hhll),IY	
ED73llhh	LD	(hhll),SP	
0A	LD	A,(BC)	
1A	LD	A,(DE)	
7E	LD	A,(HL)	
DD7Edd	LD	A,(IX+dd)	
FD7Edd	LD	A,(IY+dd)	
3Allhh	LD	A,(hhll)	
7F	LD	A,A	
78	LD	A,B	
79	LD	A,C	
7A	LD	A,D	
7B	LD	A,E	
7C	LD	A,H	
ED57	LD	A,I	
7D	LD	A,L	
3Enn	LD	A,nn	
ED5F	LD	A,R	
46	LD	B,(HL)	
DD46dd	LD	B,(IX+dd)	
FD46dd	LD	B,(IY+dd)	
47	LD	B,A	
40	LD	B,B	
41	LD	B,C	
42	LD	B,D	
43	LD	B,E	
44	LD	B,H	
45	LD	B,L	
06nn	LD	B,nn	
ED4Bllhh	LD	BC,(hhll)	
01llhh	LD	BC,hhll	
4E	LD	C,(HL)	
DD4Edd	LD	C,(IX+dd)	
FD4Edd	LD	C,(IY+dd)	
4F	LD	C,A	
48	LD	C,B	
49	LD	C,C	
4A	LD	C,D	
4B	LD	C,E	
4C	LD	C,H	
4D	LD	C,L	
0Enn	LD	C,nn	
56	LD	D,(HL)	

Operations-Code (hex.) Maschinen-Code	mnemonischer Code Assembler-Befehl		Befehlsbeschreibung
DD56dd	LD	D,(IX+dd)	
FD56dd	LD	D,(IY+dd)	
57	LD	D,A	
50	LD	D,B	
51	LD	D,C	
52	LD	D,D	
53	LD	D,E	
54	LD	D,H	
55	LD	D,L	
16nn	LD	D,nn	
ED5Bllhh	LD	DE,(hhll)	
11llhh	LD	DE,hhll	
5E	LD	E,(HL)	
DD5Edd	LD	E,(IX+dd)	
FD5Edd	LD	E,(IY+dd)	
5F	LD	E,A	
58	LD	E,B	
59	LD	E,C	
5A	LD	E,D	
5B	LD	E,E	
5C	LD	E,H	
5D	LD	E,L	
1Enn	LD	E,nn	
66	LD	H,(HL)	
DD66dd	LD	H,(IX+dd)	
FD66dd	LD	H,(IY+dd)	
67	LD	H,A	
60	LD	H,B	
61	LD	H,C	
62	LD	H,D	
63	LD	H,E	
64	LD	H,H	
65	LD	H,L	
26nn	LD	H,nn	
2Allhh	LD	HL,(hhll)	
21llhh	LD	HL,hhll	
ED47	LD	I,A	
DD2Allhh	LD	IX,(hhll)	
DD21llhh	LD	IX,hhll	
FD2Allhh	LD	IY,(hhll)	
FD21llhh	LD	IY,hhll	
6E	LD	L,(HL)	
DD6Edd	LD	L,(IX+dd)	
FD6Edd	LD	L,(IY+dd)	
6F	LD	L,A	
68	LD	L,B	
69	LD	L,C	
6A	LD	L,D	
6B	LD	L,E	
6C	LD	L,H	
6D	LD	L,L	

Operations-Code (hex.) Maschinen-Code	mnemonischer Code Assembler-Befehl		Befehlsbeschreibung	
2Enn	LD	L,nn		
ED4F	LD	R,A		
ED7Bllhh	LD	SP,(hhll)		
F9	LD	SP,HL		
DDF9	LD	SP,IX		
FDF9	LD	SP,IY		
31llhh	LD	SP,hhll		
EDA8	LDD		Lädt den Inhalt der Speicherstelle (DE) mit dem Inhalt der Speicherstelle (HL).	Vermindert DE, HL, BC um 1 (Decrement).
EDB8	LDDR			Wie LDD,aber wiederholt (Repeat), bis BC = 0.
EDA0	LDI			Vergrößert (Increment) DE, HL um 1, vermindert BC um 1.
EDB0	LDIR			Wie LDI,aber wiederholt (Repeat) bis BC = 0.
ED44	NEG		Zweierkomplement des Akkumulators (Negate).	
00	NOP		Keine Wirkung (No Operation).	
B6	OR	(HL)	Logisches ODER (OR) von Operand und Akkumulator.	
DDB6dd	OR	(IX+dd)		
FDB6dd	OR	(IY+dd)		
B7	OR	A		
B0	OR	B		
B1	OR	C		
B2	OR	D		
B3	OR	E		
B4	OR	H		
B5	OR	L		
F6nn	OR	nn		
ED8B	OTDR		Lädt den Port (BC) mit dem Inhalt der Speicherstelle (HL), vermindert B und HL um 1 und wiederholt bis B = 0.	
EDB3	OTIR		Wie OTDR, aber HL wird um 1 vergrößert (inkrementiert).	
ED79	OUT	(C),A	Lädt den Port (BC) mit dem Inhalt des zweiten Operanden (Output).	
ED41	OUT	(C),B		
ED49	OUT	(C),C		
ED51	OUT	(C),D		
ED59	OUT	(C),E		
ED61	OUT	(C),H		
ED69	OUT	(C),L		
D3nn	OUT	(nn),A	Lädt den Port (nn) mit dem Akkumulator (Output).	

Operations-Code (hex.) Maschinen-Code	mnemonischer Code Assembler-Befehl	Befehlsbeschreibung
EDAB	OUTD	Lädt den Port (BC) mit dem Inhalt der Speicherstelle (HL), vermindert B und HL um 1.
EDA3	OUTI	Wie OUTD, aber HL wird um 1 vergrößert (inkrementiert).
F1	POP AF	Lädt den Operanden mit dem Wert an der Oberfläche des Stapelspeichers.
C1	POP BC	
D1	POP DE	
E1	POP HL	
DDE1	POP IX	
FDE1	POP IY	
F5	PUSH AF	Kopiert den Operanden auf den Stapelspeicher.
C5	PUSH BC	
D5	PUSH DE	
E5	PUSH HL	
DDE5	PUSH IX	
FDE5	PUSH IY	
CB86	RES 0,(HL)	Setzt das angegebene Bit des Operanden auf den Wert 0 (Reset).
DDCBdd86	RES 0,(IX+dd)	
FDCBdd86	RES 0,(IY+dd)	
CB87	RES 0,A	
CB80	RES 0,B	
CB81	RES 0,C	
CB82	RES 0,D	
CB83	RES 0,E	
CB84	RES 0,H	
CB85	RES 0,L	
CB8E	RES 1,(HL)	
DDCBdd8E	RES 1,(IX+dd)	
FDCBdd8E	RES 1,(IY+dd)	
CB8F	RES 1,A	
CB88	RES 1,B	
CB89	RES 1,C	
CB8A	RES 1,D	
CB8B	RES 1,E	
CB8C	RES 1,H	
CB8D	RES 1,L	
CB96	RES 2,(HL)	
DDCBdd96	RES 2,(IX+dd)	
FDCBdd96	RES 2,(IY+dd)	
CB97	RES 2,A	
CB90	RES 2,B	
CB91	RES 2,C	
CB92	RES 2,D	
CB93	RES 2,E	
CB94	RES 2,H	
CB95	RES 2,L	
CB9E	RES 3,(HL)	
DDCBdd9E	RES 3,(IX+dd)	
FDCBdd9E	RES 3,(IY+dd)	
CB9F	RES 3,A	

Operations-Code (hex.) Maschinen-Code	mnemonischer Code Assembler-Befehl	Befehlsbeschreibung
CB98	RES 3,B	
CB99	RES 3,C	
CB9A	RES 3,D	
CB9B	RES 3,E	
CB9C	RES 3,H	
CB9D	RES 3,L	
CBA6	RES 4,(HL)	
DDCBddA6	RES 4,(IX+dd)	
FDCBddA6	RES 4,(IY+dd)	
CBA7	RES 4,A	
CBA0	RES 4,B	
CBA1	RES 4,C	
CBA2	RES 4,D	
CBA3	RES 4,E	
CBA4	RES 4,H	
CBA5	RES 4,L	
CBAE	RES 5,(HL)	
DDCBddAE	RES 5,(IX+dd)	
FDCBddAE	RES 5,(IY+dd)	
CBAF	RES 5,A	
CBA8	RES 5,B	
CBA9	RES 5,C	
CBAA	RES 5,D	
CBAB	RES 5,E	
CBAC	RES 5,H	
CBAD	RES 5,L	
CBB6	RES 6,(HL)	
DDCBddB6	RES 6,(IX+dd)	
FDCBddB6	RES 6,(IY+dd)	
CBB7	RES 6,A	
CBB0	RES 6,B	
CBB1	RES 6,C	
CBB2	RES 6,D	
CBB3	RES 6,E	
CBB4	RES 6,H	
CBB5	RES 6,L	
CBBE	RES 7,(HL)	
DDCBddBE	RES 7,(IX+dd)	
FDCBddBE	RES 7,(IY+dd)	
CBBF	RES 7,A	
CBB8	RES 7,B	
CBB9	RES 7,C	
CBBA	RES 7,D	
CBBB	RES 7,E	
CBBC	RES 7,H	
CBBD	RES 7,L	
C9	RET	**Rücksprung (Return) vom Unterprogramm.**
D8	RET C	**Rücksprung (Return) vom Unterprogramm, falls die angegebene Bedingung wahr ist.**
F8	RET M	
D0	RET NC	

Operations-Code (hex.) Maschinen-Code	mnemonischer Code Assembler-Befehl	Befehlsbeschreibung
C0	RET NZ	
F0	RET P	
E8	RET PE	
E0	RET PO	
C8	RET Z	
ED4D	RETI	Rücksprung (Return) vom Interrupt.
ED45	RETN	Rücksprung (Return) vom nichtmaskierbaren Interrupt.
CB16	RL (HL)	Rotiert den Operanden nach links durchs Carry (Rotate Left).
DDCBdd16	RL (IX+dd)	
FDCBdd16	RL (IY+dd)	
CB17	RL A	
CB10	RL B	
CB11	RL C	
CB12	RL D	
CB13	RL E	
CB14	RL H	
CB15	RL L	
17	RLA	Rotiert den Akkumulator nach links durchs Carry (Rotate Left Accu.)
CB06	RLC (HL)	Rotiert den Operanden kreisförmig nach links (Rotate Left Circular).
DDCBdd06	RLC (IX+dd)	
FDCBdd06	RLC (IY+dd)	
CB07	RLC A	
CB00	RLC B	
CB01	RLC C	
CB02	RLC D	
CB03	RLC E	
CB04	RLC H	
CB05	RLC L	
07	RLCA	Rotiert den Akkumulator kreisförmig nach links (Rotate Left Circular Accu.).
ED6F	RLD	Rotiert Ziffern (Digits) nach links zwischen Akkumulator und Speicherstelle (HL).
CB1E	RR (HL)	Rotiert Operanden nach rechts durchs Carry (Rotate Right).
DDCBdd1E	RR (IX+dd)	
FDCBdd1E	RR (IY+dd)	
CB1F	RR A	
CB18	RR B	
CB19	RR C	
CB1A	RR D	
CB1B	RR E	
CB1C	RR H	
CB1D	RR L	
1F	RRA	Rotiert den Akkumulator nach rechts durchs Carry (Rotate Right Accu.).

Operations-Code (hex.) Maschinen-Code	mnemonischer Code Assembler-Befehl	Befehlsbeschreibung
CB0E	RRC (HL)	Rotiert den Operanden kreisförmig nach rechts (Rotate Right Circular).
DDCBdd0E	RRC (IX+dd)	
FDCBdd0E	RRC (IY+dd)	
CB0F	RRC A	
CB08	RRC B	
CB09	RRC C	
CB0A	RRC D	
DB0B	RRC E	
CB0C	RRC H	
CB0D	RRC L	
0F	RRCA	Rotiert den Akkumulator kreisförmig nach rechts (Rotate Right Circular Accu.).
ED67	RRD	Rotiert Ziffern (Digits) nach rechts zwischen Akkumulator und Speicherstelle (HL).
C7	RST &00	Ruft das Unterprogramm bei der angegebenen Speicherstelle (Restart).
CF	RST &08	
D7	RST &10	
DF	RST &18	
E7	RST &20	
EF	RST &28	
F7	RST &30	
FF	RST &38	
DEnn	SBC A,nn	Subtrahiert den zweiten Operanden und das Carry vom ersten Operanden. Das Ergebnis steht im ersten Operanden (Substract with Carry).
9E	SBC A,(HL)	
DD9Edd	SBC A,(IX+dd)	
FD9Edd	SBC A,(IY+dd)	
9F	SBC A,A	
98	SBC A,B	
99	SBC A,C	
9A	SBC A,D	
9B	SBC A,E	
9C	SBC A,H	
9D	SBC A,L	
ED42	SBC HL,BC	
ED52	SBC HL,DE	
ED62	SBC HL,HL	
ED72	SBC HL,SP	
37	SCF	Setzt Carry-Flagge.
CBC6	SET 0,(HL)	Setzt das angegebene Bit des Operanden.
DDCBddC6	SET 0,(IX+dd)	
FDCBddC6	SET 0,(IY+dd)	
CBC7	SET 0,A	
CBC0	SET 0,B	
CBC1	SET 0,C	
CBC2	SET 0,D	
CBC3	SET 0,E	
CBC4	SET 0,H	
CBC5	SET 0,L	
CBCE	SET 1,(HL)	

Operations-Code (hex.) Maschinen-Code	mnemonischer Code Assembler-Befehl	Befehlsbeschreibung
DDCBddCE	SET 1,(IX+dd)	
FDCBddCE	SET 1,(IY+dd)	
CBCF	SET 1,A	
CBC8	SET 1,B	
CBC9	SET 1,C	
CBCA	SET 1,D	
CBCB	SET 1,E	
CBCC	SET 1,H	
CBCD	SET 1,L	
CBD6	SET 2,(HL)	
DDCBddD6	SET 2,(IX+dd)	
FDCBddD6	SET 2,(IY+dd)	
CBD7	SET 2,A	
CBD0	SET 2,B	
CBD1	SET 2,C	
CBD2	SET 2,D	
CBD3	SET 2,E	
CBD4	SET 2,H	
CBD5	SET 2,L	
CBDE	SET 3,(HL)	
DDCBddDE	SET 3,(IX+dd)	
FDCBddDE	SET 3,(IY+dd)	
CBDF	SET 3,A	
CBD8	SET 3,B	
CBD9	SET 3,C	
CBDA	SET 3,D	
CBDB	SET 3,E	
CBDC	SET 3,H	
CBDD	SET 3,L	
CBE6	SET 4,(HL)	
DDCBddE6	SET 4,(IX+dd)	
FDCBddE6	SET 4,(IY+dd)	
CBE7	SET 4,A	
CBE0	SET 4,B	
CBE1	SET 4,C	
CBE2	SET 4,D	
CBE3	SET 4,E	
CBE4	SET 4,H	
CBE5	SET 4,L	
CBEE	SET 5,(HL)	
DDCBddEE	SET 5,(IX+dd)	
FDCBddEE	SET 5,(IY+dd)	
CBEF	SET 5,A	
CBE8	SET 5,B	
CBE9	SET 5,C	
CBEA	SET 5,D	
CBEB	SET 5,E	
CBEC	SET 5,H	
CBED	SET 5,L	
CBF6	SET 6,(HL)	
DDCBddF6	SET 6,(IX+dd)	

Operations-Code (hex.) Maschinen-Code	mnemonischer Code Assembler-Befehl		Befehlsbeschreibung
FDCBddF6	SET	6,(IY+dd)	
CBF7	SET	6,A	
CBF0	SET	6,B	
CBF1	SET	6,C	
CBF2	SET	6,D	
CBF3	SET	6,E	
CBF4	SET	6,H	
CBF5	SET	6,L	
CBFE	SET	7,(HL)	
DDCBddFE	SET	7,(IX+dd)	
FDCBddFE	SET	7,(IY+dd)	
CBFF	SET	7,A	
CBF8	SET	7,B	
CBF9	SET	7,C	
CBFA	SET	7,D	
CBFB	SET	7,E	
CBFC	SET	7,H	
CBFD	SET	7,L	
CB26	SLA	(HL)	**Verschiebt die Bits des Operanden arithmetisch nach links (Shift Left Arithmetic).**
DDCBdd26	SLA	(IX+dd)	
FDCBdd26	SLA	(IY+dd)	
CB27	SLA	A	
CB20	SLA	B	
CB21	SLA	C	
CB22	SLA	D	
CB23	SLA	E	
CB24	SLA	H	
CB25	SLA	L	
CB2E	SRA	(HL)	**Verschiebt die Bits des Operanden arithmetisch nach rechts (Shift Right Arithmetic).**
DDCBdd2E	SRA	(IX+dd)	
FDCBdd2E	SRA	(IY+dd)	
CB2F	SRA	A	
CB28	SRA	B	
CB29	SRA	C	
CB2A	SRA	D	
CB2B	SRA	E	
CB2C	SRA	H	
CB2D	SRA	L	
CB3E	SRL	(HL)	**Verschiebt die Bits des Operanden logisch nach rechts (Shift Right Logical).**
DDCBdd3E	SRL	(IX+dd)	
FDCBdd3E	SRL	(IY+dd)	
CB3F	SRL	A	
CB38	SRL	B	
CB39	SRL	C	
CB3A	SRL	D	
CB3B	SRL	E	
CB3C	SRL	H	
CB3D	SRL	L	

Operations-Code (hex.) Maschinen-Code	mnemonischer Code Assembler-Befehl	Befehlsbeschreibung
96	SUB (HL)	Subtrahiert den Operanden vom Akkumulator.
DD96dd	SUB (IX+dd)	
FD96dd	SUB (IY+dd)	
97	SUB A	
90	SUB B	
91	SUB C	
92	SUB D	
93	SUB E	
94	SUB H	
95	SUB L	
D6nn	SUB nn	
AE	XOR (HL)	Exklusives ODER zwischen dem Operanden und dem Akkumulator.
DDAEdd	XOR (IX+dd)	
FDAEdd	XOR (IY+dd)	
AF	XOR A	
A8	XOR B	
A9	XOR C	
AA	XOR D	
AB	XOR E	
AC	XOR H	
AD	XOR L	
EEnn	XOR nn	

Flaggenbeeinflussung der Befehle

Bedeutung der Abkürzungen
Für Operanden bedeutet **r** je nach Befehl ein 8-Bit-Register, (HL), (IX+d), (IY+d) oder eine 8-Bit-Zahl. **rr** bedeutet ein 16-Bit-Register.

Flaggenbeeinflussung
+ Flagge wird dem Resultat der Operation entsprechend beeinflußt.
— Flagge wird nicht beeinflußt.
1 Flagge wird gesetzt.
0 Flagge wird gelöscht.
? Flagge wird willkürlich beeinflußt.
a) H enthält den bisherigen Inhalt der C-Flagge.
b) Z wird gesetzt, wenn B nach der Ausführung den Inhalt 0 hat, sonst wird Z zurückgesetzt.
c) P/V wird zurückgesetzt, wenn BC nach der Ausführung den Inhalt 0 hat, sonst wird P/V gesetzt.
d) Z wird gesetzt, wenn A=(HL) ist.
e) P/V enthält den Inhalt von IFF2. (IFF2 ist eine Kontrollflagge für den Interrupt und gehört nicht zum F-Register.)

	S	Z	H	P/V	N	C
ADD A,r / ADC A,r	+	+	+	+	0	+
SUB r / SBC A,r / CP r / NEG	+	+	+	+	1	+
AND r	+	+	1	+	0	0
OR r / XOR r	+	+	0	+	0	0
INC r	+	+	+	+	0	—
DEC r	+	+	+	+	1	—
BIT n,r	?	+	1	?	0	—
ADD HL,rr / ADD IX,rr / ADD IY,rr	—	—	+	—	0	+
ADC HL,rr	+	+	+	+	0	+
SBC HL,rr	+	+	+	+	1	+
RL r / RLC r / SLA r / RR r / RRC r / SRA r / SRL r	+	+	0	+	0	+
RLA / RLCA / RRA / RRCA	—	—	0	—	0	+
RLD / RRD	+	+	0	+	0	—
DAA	+	+	+	+	—	+
CPL	—	—	1	—	1	—
CCF	—	—	a)	—	0	+
SCF	—	—	0	—	0	1
IN r,(C)	+	+	0	+	0	—
INI / IND / OUTI / OUTD	?	b)	?	?	1	—
INIR / INDR / OTIR / OTDR	?	1	?	?	1	—
LDI / LDD	—	—	0	c)	0	—
LDDR / LDIR	—	—	0	0	0	—
CPI / CPIR / CPD / CPDR	+	d)	+	c)	1	—
LD A,I / LD A,R	+	+	0	e)	0	—

Dez-Hex-Dual-Tabelle

Dez	Hex	Dual	Dez	Hex	Dual	Dez	Hex	Dual
0	0	0	56	38	111000	112	70	1110000
1	1	1	57	39	111001	113	71	1110001
2	2	10	58	3A	111010	114	72	1110010
3	3	11	59	3B	111011	115	73	1110011
4	4	100	60	3C	111100	116	74	1110100
5	5	101	61	3D	111101	117	75	1110101
6	6	110	62	3E	111110	118	76	1110110
7	7	111	63	3F	111111	119	77	1110111
8	8	1000	64	40	1000000	120	78	1111000
9	9	1001	65	41	1000001	121	79	1111001
10	A	1010	66	42	1000010	122	7A	1111010
11	B	1011	67	43	1000011	123	7B	1111011
12	C	1100	68	44	1000100	124	7C	1111100
13	D	1101	69	45	1000101	125	7D	1111101
14	E	1110	70	46	1000110	126	7E	1111110
15	F	1111	71	47	1000111	127	7F	1111111
16	10	10000	72	48	1001000	128	80	10000000
17	11	10001	73	49	1001001	129	81	10000001
18	12	10010	74	4A	1001010	130	82	10000010
19	13	10011	75	4B	1001011	131	83	10000011
20	14	10100	76	4C	1001100	132	84	10000100
21	15	10101	77	4D	1001101	133	85	10000101
22	16	10110	78	4E	1001110	134	86	10000110
23	17	10111	79	4F	1001111	135	87	10000111
24	18	11000	80	50	1010000	136	88	10001000
25	19	11001	81	51	1010001	137	89	10001001
26	1A	11010	82	52	1010010	138	8A	10001010
27	1B	11011	83	53	1010011	139	8B	10001011
28	1C	11100	84	54	1010100	140	8C	10001100
29	1D	11101	85	55	1010101	141	8D	10001101
30	1E	11110	86	56	1010110	142	8E	10001110
31	1F	11111	87	57	1010111	143	8F	10001111
32	20	100000	88	58	1011000	144	90	10010000
33	21	100001	89	59	1011001	145	91	10010001
34	22	100010	90	5A	1011010	146	92	10010010
35	23	100011	91	5B	1011011	147	93	10010011
36	24	100100	92	5C	1011100	148	94	10010100
37	25	100101	93	5D	1011101	149	95	10010101
38	26	100110	94	5E	1011110	150	96	10010110
39	27	100111	95	5F	1011111	151	97	10010111
40	28	101000	96	60	1100000	152	98	10011000
41	29	101001	97	61	1100001	153	99	10011001
42	2A	101010	98	62	1100010	154	9A	10011010
43	2B	101011	99	63	1100011	155	9B	10011011
44	2C	101100	100	64	1100100	156	9C	10011100
45	2D	101101	101	65	1100101	157	9D	10011101
46	2E	101110	102	66	1100110	158	9E	10011110
47	2F	101111	103	67	1100111	159	9F	10011111
48	30	110000	104	68	1101000	160	A0	10100000
49	31	110001	105	69	1101001	161	A1	10100001
50	32	110010	106	6A	1101010	162	A2	10100010
51	33	110011	107	6B	1101011	163	A3	10100011
52	34	110100	108	6C	1101100	164	A4	10100100
53	35	110101	109	6D	1101101	165	A5	10100101
54	36	110110	110	6E	1101110	166	A6	10100110
55	37	110111	111	6F	1101111	167	A7	10100111

Dez	Hex	Dual
168	A8	10101000
169	A9	10101001
170	AA	10101010
171	AB	10101011
172	AC	10101100
173	AD	10101101
174	AE	10101110
175	AF	10101111
176	B0	10110000
177	B1	10110001
178	B2	10110010
179	B3	10110011
180	B4	10110100
181	B5	10110101
182	B6	10110110
183	B7	10110111
184	B8	10111000
185	B9	10111001
186	BA	10111010
187	BB	10111011
188	BC	10111100
189	BD	10111101
190	BE	10111110
191	BF	10111111
192	C0	11000000
193	C1	11000001
194	C2	11000010
195	C3	11000011
196	C4	11000100
197	C5	11000101
198	C6	11000110
199	C7	11000111
200	C8	11001000
201	C9	11001001
202	CA	11001010
203	CB	11001011
204	CC	11001100
205	CD	11001101
206	CE	11001110
207	CF	11001111
208	D0	11010000
209	D1	11010001
210	D2	11010010
211	D3	11010011
212	D4	11010100
213	D5	11010101
214	D6	11010110
215	D7	11010111
216	D8	11011000
217	D9	11011001
218	DA	11011010
219	DB	11011011
220	DC	11011100
221	DD	11011101
222	DE	11011110
223	DF	11011111
224	E0	11100000
225	E1	11100001
226	E2	11100010
227	E3	11100011
228	E4	11100100
229	E5	11100101
230	E6	11100110
231	E7	11100111
232	E8	11101000
233	E9	11101001
234	EA	11101010
235	EB	11101011
236	EC	11101100
237	ED	11101101
238	EE	11101110
239	EF	11101111
240	F0	11110000
241	F1	11110001
242	F2	11110010
243	F3	11110011
244	F4	11110100
245	F5	11110101
246	F6	11110110
247	F7	11110111
248	F8	11111000
249	F9	11111001
250	FA	11111010
251	FB	11111011
252	FC	11111100
253	FD	11111101
254	FE	11111110
255	FF	11111111

1.
LD (&C000),A muß durch LD (&C028),A ersetzt werden.

2.
LD A,&FF muß durch LD A,&0F bzw. LD A,&F0 ersetzt werden.

3.
Die Befehle a, b, d und e existieren.

4.
Nein. Die Befehle LD (BC),n bzw. LD (DE),n mit einem festen Wert n gibt es nicht.

5.
Zwischen INC HL und DEC B wird noch ein Befehl INC HL eingefügt.

6.
```
10 MODE 2
20 FOR I=&C000 TO &FFFF: POKE I,&FF:NEXT
```
Zeit BASIC 39 s, Maschinenprogramm ca. 0,2 s.

7.
Statt
```
   JP NZ,L1
   RET
```
kann es auch heißen
```
   RET Z
   JP L1
```

8.
Die Adressen &AB80 (&A67C beim CPC664/6128) und &C030 müssen durch die Adressen &ABA8 (&A6A4 beim CPC664/6128) und &C00A ersetzt werden.

9.
In den Zeilen 4, 7 und 11 müssen die Befehle
```
   LD IY,&C030
   LD (IY+&00),A
   ADD IY,BC
```
eingesetzt werden.

10.
Ersetzen Sie LD D,&08 durch LD D,&04 und LD BC,&0800 durch LD BC,&1000 (2*&800=&1000). Fügen Sie außerdem nach INC IX nochmals den Befehl INC IX ein.

11.
IX=&AB88 und HL=&0030
(IX=&A684 beim CPC664/6128)

12.
Der jeweilige Inhalt von BC bleibt unverändert.
a) HL=&0200 Carry=1
b) HL=&3C00 Carry=0
c) HL=&0000 Carry=1

13.
```
    LD BC,&07D0
L1: LD A,&30
    CALL &BB5A
    DEC BC
    LD A,B
    OR C
    JP NZ,L1
    RET
```
L1=&4003

BASIC-Vorspann:
```
10 MODE 2: CALL &4000
20 GOTO 20
```

14.
a) HL=&1000; BC=&1000
b) HL=&EEEE; BC=&1000

15.
a) A=&X11000001=&C1 Carry=1
b) A=&X11000011=&C3 Carry=1
c) A=&X11000000=&C0 Carry=1

16.
Der Befehl CALL C,ZP muß in P9 durch CALL NC,ZP ersetzt werden.

17.
Der Zeiger IY muß in P9 auf das Ende des Bitmusters zeigen: LD IY,&AB87 (LD IY,&A683 beim CPC664/6128) in Zeile 5. Weiter muß INC IY in Zeile 18 durch DEC IY ersetzt werden.

18.
Das siebte Bit eines Bytes bleibt bei dem Befehl SRA immer erhalten. Da für die gesenkte Hand des Männchens das siebte Bit gesetzt ist, zieht diese Hand beim Verschieben einen Strich hinter sich her.

19.
Die Bitwerte der vorherigen Position werden jetzt nicht mehr aus der Carry-Flagge in die Speicherstelle der nächsten Position übertragen. Nach acht Verschiebungen ist daher das Männchen verschwunden.

20.
Mode 2 darf jetzt nicht mehr im Maschinenprogramm gesetzt werden, da sonst die Cursorstellung des LOCATE-Befehls wieder geändert wird. Weitere Änderungen:

Zeile 7:	LD HL,&C000 wird ersetzt durch LD HL,&C04F (Zeiger auf letzte Position der ersten Zeile).
Zeile 12:	CP &52 wird ersetzt durch CP &4C (Abfrage auf L).
Zeile 18:	INC HL wird ersetzt durch DEC HL.
Zeile 24:	SRL (IX+&00) wird ersetzt durch SLA (IX+&00).
Zeile 25:	RR (IX+&01) wird ersetzt durch RL (IX+&FF). Laut Zusatzinformation entspricht der Wert &FF des Distanzbytes d gerade der Verschiebung „−1“.

21.
```
LD HL,&C001
LD DE, &C000
LD BC, &3FFF
LDIR
RET
```

22.
a) n=&08
b) n=&F4
c) Die Sprungweite ist zu groß.

23.
Der Befehl
AND &DF
wird ersetzt durch den Befehl
XOR &70
oder durch die beiden Befehle
AND &0F
OR &30

24.

```
a) AND &0F        ;(&0F=X00001111)
b) LD A,B
   AND &FE        ;(&FE=X11111110)
c) SLA C
   RL B
```

25.

```
     LD A,&02
     CALL &BC0E
     XOR A
     LD (&0101),A
     LD A,&C7
L1:  LD (&0100),A
     LD (&0102),A
     PUSH AF
     CALL &4000
     POP AF
     DEC A
     JR NZ,L1
     RET
```

26.

Hinter Zeile 9 (INC HL) sind die Befehle LD A,&20 und CALL &BB5A einzufügen. (&20 ist der ASCII-Code des Leerzeichens.)

Achtung: Das Programm kann sich selbst nicht „dumpen", da durch die RLD-Befehle eine Selbstmodifikation eintritt.

27.

Der Befehl XOR &48 muß durch XOR &68 ersetzt werden.

28.

Der Inhalt von A wird durch die ASCII-Umwandlung verändert. Ohne eine Zwischenspeicherung würde die zweite Ziffer des Bytes falsch ausgegeben werden. Auch der Inhalt des ausgegebenen Speicherbereichs würde verändert werden.

29.

12=X1100; 15=X1111; 127=X1111111; 128=X10000000; 200=X11001000

```
  1100      1111111      10000000
  1111          1100     11001000
 -----     ---------    ---------
 11011     10001011     10001000
```

Die Stellen, aus denen ein Übertrag erfolgte, sind unterstrichen.

30.
Der BASIC-Vorspann lautet:

```
10 INPUT A: A0=INT(A/65536):A1=INT((A-65536*A0)/256):
   A2=A-A0*65536-A1*256
20 POKE &100,A2: POKE &101,A1: POKE &102,A0
30 (Wie Zeile 10)
40 POKE &103,A2:POKE &104,A1:POKE &105,A0
50 CALL &4000
60 PRINT 65536*PEEK(&102)+256*PEEK(&101)+PEEK(100)
```

Maschinenprogramm:

```
    OR A
    LD HL,&0100
    LD DE,&0103
    LD B,&03
L1: LD A,(DE)
    ADC A,(HL)
    LD (HL),A
    INC DE
    INC HL
    DJNZ L1
    RET
```

31.
Entfällt.

32.
a) Zweierkomplement von −120: 256−120=136=X10001000
Zweierkomplement von −10: 256−10=246=X11110110
b) 120=X01111000
(Einerkomplement von 120)+1=X10000111+1=X10001000
10=X00001010
(Einerkomplement von 10)+1=X11110101+1=X11110110

33.
Die V-Flagge wird gesetzt bei:

```
  60+100=160
 100+36=136
 -40-100=-140
-120-10=-130
```

Die V-Flagge wird gelöscht bei:
22+15=37
90−100=−10

34.
Die Ersetzung der Abfragen ist möglich: Man muß gleichzeitig den Befehl JP M,L1 durch JP P,L1 und die Befehle JP PE,L2 sowie JP PE,L3 durch JP PO,L2 sowie JP PO,L3 ersetzen. So wird wieder für (P,PE) und (M,PO) nach L2 und für (P,PO) und (M,PE) nach L3 verzweigt.

35.
Auch bei Vertauschung der Abfragen werden die Zifferntasten richtig aussortiert. Da jetzt aber der zweite Verzweigungsbefehl auf die NC-Bedingung abfragt, ist bei der ersten Ausführung des DAA-Befehls die C-Flagge gesetzt, und der Dezimalabgleich erfolgt falsch.

36.
Die höherwertigen vier Bits des Akkumulators werden durch den RLD-Befehl und auch durch die Systemroutine &BB5A nicht abgeändert. Daher bleiben die durch den ersten Befehl OR &30 gesetzten Bits erhalten.

37.
Nach der Tastaturabfrage sind die nächsten drei Befehle

```
LD A,(&0100)
INC A
DAA
```

durch folgende Befehle zu ersetzen:

```
      AND &01
      LD A,(&0100)
      JR Z,L11
      DEC A
      JR L12
L11: INC A
L12: DAA
L11=&4019
L12=&401A
```

Dabei dient der Befehl AND &01 (auch BIT 0,A wäre möglich) zur Unterscheidung zwischen geraden und ungeraden Zahlen.

38.
a)
```
LD BC,&7F00
IN E,(C)
```
b)
```
LD BC,&EF00
LD A,&31
OUT (C),A
```

39.
Die Zeilen 4 bis 25 von P23 werden folgendermaßen abgeändert: Zeile 4 erhält die

Sprungmarke L1:. Die Sprungmarke in Zeile 11 entfällt. Zwischen die Befehle **L1: LD IX,MUDA** und LD A,&08 in Zeile 5 müssen folgende Befehle eingeschoben **werden:**

```
CALL &BB06
CP &51
RET Z
AND &0F
ADD A,A
LD D,&00
LD E,A
ADD IX,DE
```

Mit diesen Befehlen wird die Tastatur abgefragt und der doppelte Ziffernwert zu IX addiert, um auf die richtigen Tondaten zu kommen. „Q" beendet das Programm.
Im weiteren Programm entfallen die Zeilen 15, 16, 17, 20, 21 und 25. Die Reihenfolge der Befehle in den Zeilen 23 und 24 wird vertauscht, wobei die Marke L2: entfällt.

Neue Werte für Marken sind:

```
L1=&4003
L3=&403E
MUDA=&4065
TGL=&4044
WARTE=&403B
```

40.
Es enfallen die Befehle der Zeilen 4, 8, 9, 24, 25 und 26. (DI, ROM freischalten, ROM sperren, EI.) Dabei ändern sich natürlich auch die Adressen der Sprungmarken L1 und L2. An irgendeiner Stelle außerhalb des Programms muß der Befehl RST &28 stehen. An den folgenden Speicherstellen steht die Adresse &4000 in der Reihenfolge Low-Byte, High-Byte. Also z. B.

```
3000   RST &28
3001   DEFB &00,&40
```

Das Programm wird jetzt mit CALL &3000 aufgerufen. (Da das untere ROM freigeschaltet ist, muß ein Benutzerprogramm, das von RST &28 aufgerufen wird, oberhalb &3FFF liegen.)

41.
Das Programm TGL wird ersetzt durch

```
TGL: RST &28
     DEFB &26,&08
```

(DEFB &53,&08 beim CPC664; DEFB &63,&08 beim CPC6128)

Ein RET-Befehl ist hier nicht notwendig.

42.
Da der DJNZ-Befehl das B-Register vor der Abfrage auf Null dekrementiert, ist nur so eine Ausgabe aller 256 Zeichen zu erreichen. Würde man B mit &FF laden, so würde das letzte Zeichen nicht ausgegeben werden.

43.
Im Programm P27 entfallen die Zeilen 2 bis 9. Die entsprechenden Befehle werden ersetzt durch

```
LD A,&F8;(&F8=248)
LD (&40F0),A
```

In Zeile 13 wird die Startadresse &402C durch &401F ersetzt.
Beim eigentlichen Interruptprogramm entfallen die Zeilen 25 bis 35, 42 bis 54 und 60 bis 69. Zwischen die übrigbleibenden Befehle CALL &BB84 (Zeile 41) und LD HL,(&40F8) (Zeile 55) werden folgende Befehle eingefügt:

```
LD A,(&40F0)      ;ASCII-Code holen.
XOR 1             ;ASCII-Code ändern.
LD (&40F0),A      ;ASCII-Code wieder abspeichern.
CALL &BB5A        ;Ausgabe
```

Außerdem kann noch in Zeile 38 der Setzbefehl für die Cursorspalte von LD H,&47 in LD H,&50 abgeändert werden.

Literatur H

Brückmann, Rolf, Englisch, Lothar und Klaus Gerits
CPC464 Intern
Data Becker GmbH, Düsseldorf 1985.

CPC464 Firmware Handbook
Schneider Computer Division, Türkheim 1984.

Huslik, Winfried
CPC464 Inside Out
S. Huslik Verlag, Augsburg 1985.

Nichols, Elisabeth A., Nichols, Joseph C. und Peter E. Rony
Z-80 Einführung und Programmierung
Elektor Verlag, Gangelt 1979.

Zaks, Rodney
Programmierung des Z-80
Sybex Verlag, Düsseldorf 1982.

Sachwortregister

Notizen

Notizen