SIEMENS

Einführung in die Assembler Programmierung der Mikroprozessoren SAB 8080/85

Von Hansjochen Benda und Franz Hund

SIEMENS AKTIENGESELLSCHAFT

Herausgegeben von
Siemens Aktiengesellschaft
Bereich Bauelemente
D-8000 München 80, Balanstraße 73

Vorwort

Dieses Buch ist bestimmt für technisch interessierte Leser ohne Vorkenntnisse in der Datenverarbeitung. Sein Schwerpunkt ist Assembler-Software. Über Hardware wird nur soviel gebracht, wie zum Verständnis des Programmierens in maschinenorientierter Sprache notwendig ist.
Nach der Lektüre sollte der Leser in der Lage sein, Nachrichten wenigstens in der populären Fachliteratur mit Verständnis zu lesen.

Inhaltsverzeichnis

Bilderverzeichnis 6

Einleitung 8

1. Arbeitsweise und Aufbau eines Computers 9

1.1. Zentraleinheit, Befehle, Programm 9
1.2. Digitaltechnik, Bit, Datenwort 9
1.3. Speicher und Schnittstellen 10

2. Grobstruktur des Mikrocomputers 12

2.1. Bausteine 12
2.2. Leitungen 15
2.3. Größe der Speicherplätze 15
2.4. Anzahl der Speicherplätze 16
2.5. Adressierung der E/A-Kanäle 17

3. Erstes Programmbeispiel 18

4. Befehlsvorrat 23

4.1. Verschiedene Arten von Befehlen 23
4.2. Transferbefehle für 8 Bit 26
4.3. Adressierungsarten 32
4.4. Transferbefehle für 16 Bit 33
4.5. Arithmetische Operationen mit 8 Bit-Zahlen 36
4.5.1. Das Übertragsbit 37
4.5.2. Befehle für arithmetische Operationen mit 8 Bit-Zahlen 37
4.6. Arithmetische Operationen mit 16 Bit-Zahlen 48
4.7. Logische Operationen 53
4.7.1. Logische Verknüpfungen 53
4.7.2. Weitere logische Operationen 57
4.8. Weitere Befehle 59

5. Befehlszyklus 61

5.1. Definition der maßgebenden Zeitspannen 61
5.2. Einzelheiten eines Befehlszyklus 62
5.3. Steuersignale 65

6. Sprünge, Unterprogramme und Unterbrechungen 66

6.1. Programmsprünge 66
6.2. Unterprogramme 75
6.3. Unterbrechungen 80

7. Weitere Programmbeispiele 84

8. Herstellung von Programmen 90
8.1. Entwicklungssystem 90
8.2. Ein Beispiel: Geschwindigkeitsmessung 93

9. Zahlensysteme, binäre Addition und Subtraktion . 103
9.1. DEZIMAL-System 103
9.2. OKTAL-System 105
9.3. BINÄR-System (auch Dualsystem) 106
9.4. HEXADEZIMAL-System (auch Sedezimalsystem) 107
9.5. Abkürzungen von BINÄR-Zahlen durch OKTAL- oder HEXA-DEZIMAL-Zahlen 108
9.6. Addition und Subtraktion binärer Zahlen 108

Sachverzeichnis 114
Tabellen 4.1 bis 4.4 115
Anschriften unserer Geschäftsstellen 139

Bilderverzeichnis

Nr.	Titel	Seite
1.1	Beispiel für eine physikalische Realisierung der Information „1010" . .	9
2.1	Die wichtigsten Teile eines Mikrocomputers	12
2.2	Die Register und die Speicherplätze	13
3.1	Zum Programmbeispiel der Tabelle 3-1; Speicher- und Registerinhalte zum Zeitpunkt 0	20
3.2	Zum Programmbeispiel der Tabelle 3-1; Speicher- und Registerinhalte zum Zeitpunkt 4	22
4.1	Geschwindigkeitsmessung	23
4.2	Assembler-Listing am Beispiel der Programmstelle von Tabelle 3-1 . .	25
4.3	Der Befehl MVI D, D8 mit D8 = Ø1	26
4.4	Der Befehl MVI M, D8 mit D8 = Ø1 bei dem angegebenen Inhalt von Reg.-Paar HL	27
4.5	Der Befehl MOV E, B	28
4.6	Der Befehl MOV B, M bei dem angegebenen Inhalt von Reg.-Paar HL .	29
4.7	Der Befehl LDA Adr mit Adr = D67E	30
4.8	Der Befehl STAX B bei dem angegebenen Inhalt von Reg.-Paar BC . .	31

Nr.	Titel	Seite
4.9	Der Befehl LXI B, D16 mit D16 = 1234	33
4.10	Der Befehl LHLD Adr mit Adr = 25DA	34
4.11	Der Befehl XCHG	35
4.12	Addition zweier 24 Bit-Zahlen	38
4.13	Addition zweier 16 Bit-Zahlen	39
4.13a)	Niederwertiges Byte der 1. Zahl in den Akku	39
4.13b)	Obere Hälfte der Adresse 1123 nach Reg. H.	40
4.13c)	Niedere Hälfte der Adresse 1123 nach Reg. L.	41
4.13d)	Addition der niederwertigen Hälften der Zahlen	42
4.13e)	Summe der niederwertigen Hälften in Reg. C.	43
4.13f)	Adresse in Reg. Paar HL wird inkrementiert	44
4.13g)	Höherwertiges Byte der 1. Zahl in den Akku	45
4.13h)	Addition der höherwertigen Hälften der Zahlen	46
4.13i)	Das Resultat steht im Reg. Paar BC	47
4.14	Addition zweier 16 Bit-Zahlen mit Befehlen für Registerpaare	49
4.14a)	Erste Zahl in Reg.-Paar HL	49
4.14b)	Erste Zahl in Reg.-Paar DE	50
4.14c)	Zweite Zahl in Reg.-Paar HL	51
4.14d)	Addition	52
4.15	Beispiel für Eingabe-Maskierung	56
4.16	Der Befehl RAL	59
4.17	Der Befehl RAR	59
4.18	Der Befehl RLC	60
4.19	Der Befehl RRC	60
5.1	Die maßgebenden Zeitspannen	61
5.2	Einholen eines Operationscodes I	63
5.3	Einholen eines Operationscodes II	63
5.4	Anschlußbelegung des Mikroprozessors	64
6.1	Bedingter Sprung im Beispiel von Bild 4.15	67
6.2	Bedingter und unbedingter Sprung	68
6.3	Programmablaufplan zu Bild 6.2	69
6.4	Programmablaufplan für Zeitschleife	72
6.5	Auswertung des Vergleiches zweier Zahlen	73
6.6	Adressenverwaltung während eines Sprunges	74
6.7	Mehrfacher Aufruf eines Unterprogrammes (hier mit unbedingten Aufruf- und Rückkehrbefehlen)	75
6.8	Momentaufnahme von Stapelzeiger, Befehlszähler und Kellerspeicher während des 1. Unterprogramm-Ablaufes von Bild 6.7	76
6.9	Verschachtelung zweier Unterprogramme Unten: Momentaufnahmen des Kellerspeichers mit Stapelzeiger	77
6.10	Inhalt des Kellerspeichers zum Textbeispiel	79
6.11	Einfache Unterbrechungsschaltung	82
7.1	Zum ersten Programmbeispiel	84
7.2	Zum zweiten Programmbeispiel	86
8.1	Ein- und Ausgabe für die Geschwindigkeitsmessung	94
8.2/1	Programmablaufplan für die Geschwindigkeitsmessung	96
8.2/2	Programmablaufplan für die Geschwindigkeitsmessung	97
8.3	Das Quellenprogramm für die Geschwindigkeitsmessung	98
8.4/1	Das Assembler-Listing für die Geschwindigkeitsmessung/1. Teil	99
8.4/2	Das Assembler-Listing für die Geschwindigkeitsmessung/2. Teil	100
8.4/3	Das Assembler-Listing für die Geschwindigkeitsmessung/3. Teil	101
9.1	Kilometerzähler	103

Einleitung

Mikrocomputer unterscheiden sich von anderen Computern dadurch, daß sie kleiner und billiger sind. Die Abgrenzung zu den nächstgrößeren Minicomputern ist nicht immer sehr scharf. Auf jeden Fall ist richtig, daß Mikrocomputer in ihren sämtlichen Teilen aus hochintegrierten Halbleiterchips bestehen.
Ein Chip mit seinem Gehäuse wird als Baustein bezeichnet.
Eine Familie von unterschiedlichen Bausteinen steht jeweils zum Zusammenstellen individueller Mikrocomputer zur Verfügung.

Der wichtigste Baustein in jeder Familie ist der Mikroprozessor, im vorliegenden Fall der SAB 8080 oder SAB 8085.
Die Buchstabengruppe SAB besagt normgemäß, daß es sich um einen Digitalchip für Temperaturen von 0 bis 70°C handelt.
Die Bausteine dieser Familie haben mindestens 16, höchstens 40 Kontakte; es sind „Dual-In-line-Packages" (DIPs). Sie sind teils in n-Kanal-MOS-Technik aufgebaut, teils in Schottky-Bipolar-Technik. Als Versorgungsspannung braucht man immer +5V, gelegentlich auch +12V und −5V.

Der SAB 8085 stellt hardwaremäßig eine deutliche Verbesserung des SAB 8080 dar. Softwaremäßig ist er um die Befehle RIM und SIM erweitert worden. Seine übrigen Befehle sind identisch mit denen des SAB 8080. Deshalb kann diese Einführung auch zum Kennenlernen des SAB 8085 benutzt werden.
Unabhängig vom benutzten Mikrocomputer eignet sich diese Einführung zum Kennenlernen der grundlegenden Begriffe.

Diese und alle anderen Angaben betreffen den heutigen Stand der Technik. Die Integration wird laufend vorangetrieben; das führt dazu, daß einzelne Bausteine immer komplexere Aufgaben übernehmen.

Die vorliegende Schrift wendet sich an Techniker, die bisher noch keinen Kontakt mit der Datenverarbeitung hatten. Das Ziel ist, einen Überblick über Arbeitsweise und Anwendung zu geben. Der Leser sollte danach in der Lage sein, Nachrichten wenigstens in der populären Fachliteratur mit einigem Verständnis zu lesen. Der leichteste Einstieg in das fremde Gebiet besteht wohl im Studium von Software auf Maschinenniveau. Die Hardware wird dabei nur auf dem Niveau von Blockschaltbildern zur Kenntnis genommen. Dem entsprechend wurde in dieser Schrift angestrebt, so früh wie möglich einen einfachen Programmausschnitt zu diskutieren. Unmittelbar danach wird schon der Befehlsvorrat so vollständig wie zu diesem Zeitpunkt möglich diskutiert. Dies hat den Vorteil, daß spätere abstrakte Definitionen mit Leben erfüllt werden. In Kauf genommen wurde der Nachteil, daß zur Erklärung mancher Begriffe auf spätere Abschnitte verwiesen werden mußte.

Kapitel 1 geht stichwortartig auf die Arbeitsweise und den Aufbau von Computern ein. Einzelheiten des Aufbaues speziell beim Mikrocomputer (und dann auch gleich beim SAB 8080) bringt Kapitel 2; wer die hier vorausgesetzte Kenntnis der binären Zahlen noch nicht besitzt, kann dies mit Hilfe des Kapitels 9 nachholen. Dort findet man auch Mitteilungen über Hexadezimalzahlen; man braucht sie im 3. Kapitel, das sich mit dem angekündigten Programmbeispiel beschäftigt.
Kapitel 4 diskutiert den Befehlsvorrat so vollständig, wie in diesem Stadium möglich.
Kapitel 5 befaßt sich noch einmal mit Hardwarefragen, und zwar mit einigen Details des Befehlszyklus.
Im 6. Kapitel werden Verzweigungen und Sprünge im Programm eingeführt; daraufhin können schon etwas anspruchsvollere Programmbeispiele gebracht werden, was in Kapitel 7 geschieht. Die Entstehung von Programmen wird in Kapitel 8 beschrieben und an einem Beispiel vorgeführt.
Kapitel 9 bringt einige Zahlensysteme und die binäre Subtraktion.

1. Arbeitsweise und Aufbau eines Computers

1.1. Zentraleinheit, Befehle, Programm

Der wichtigste Bestandteil eines Computers ist die **„Zentraleinheit"**; ihre Aufgabe ist es, die Tätigkeit des Computers zu überwachen und zu steuern. Genauer gesagt: Die Zentraleinheit holt **Befehle** aus dem Speicher (Abschnitt 1.3) und führt sie aus. Die Gesamtheit der nacheinander eingeholten und ausgeführten Befehle zum Erfüllen einer bestimmten Aufgabe heißt **Programm.** Die Zeitspanne, in der ein Befehl erst eingeholt und dann ausgeführt wird, heißt **Befehlszyklus.** Damit setzt sich der zeitliche Ablauf eines Programmes aus einer längeren oder kürzeren Folge von Befehlszyklen zusammen.
Ein Computerprogramm zur Lösung einer bestimmten Aufgabe stellt man auf, indem man die entsprechende Folge von Befehlen schreibt.

Man bezeichnet die „harten", physikalischen Teile des Computers und der angeschlossenen Geräte als **Hardware** (sprich „Hardwähr"; hard (engl.) = hart). Die Programme und zugehörige Notizen und Daten dagegen heißen **Software** (sprich „Softwähr"; soft (engl.) = weich).

Ein Computer kann beliebig viele verschiedene Programme ausführen und damit beliebig viele verschiedene Aufgaben lösen. Diese heute selbstverständlich erscheinende Eigenschaft beruht im wesentlichen auf Überlegungen des ungarisch-amerikanischen Mathematikers John von Neumann (1902–1957).

1.2. Digitaltechnik, Bit, Datenwort

Die in Computern vorkommenden Schaltungen benutzen die Methoden der **Digitaltechnik**; das tun auch viele Schaltungen in anderen Anlagen. Digitaltechnik ist das Gegenstück zur Analogtechnik; dies sind zwei unterschiedliche Methoden der Nachrichtentechnik. Bei beiden Methoden wird eine Information durch eine elektrische Größe dargestellt, beispielsweise durch eine elektrische Spannung. In der Analogtechnik hängt der genaue Augenblickswert einer solchen Spannung von der darzustellenden Information ab, dagegen nicht in der Digitaltechnik: Hier besteht die (kleinste Einheit der) Information einfach darin, ob das elektrische Potential an einem Punkt von einem Bezugspotential abweicht oder nicht, ob also Spannung besteht oder nicht.

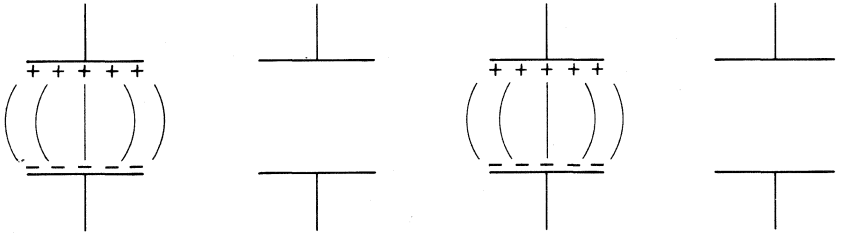


Bild 1.1
Beispiel für eine physikalische Realisierung der Information „1010"

Entsprechend diesem Prinzip „Spannung oder keine Spannung" beschreibt man die im Computer enthaltene Information zweckmäßig mit Hilfe der Zeichen „1" oder „0".
Das physikalische Vokabular des Computers besteht aus Kombinationen der Zustände „Spannung" und „keine Spannung"; diese Kombinationen können auf dem Papier durch entsprechende Kombinationen der Zeichen „1" und „0" dargestellt werden. Der Informationsgehalt ist in beiden Fällen derselbe.
So zeigt z. B. Bild 1.1 eine Gruppe von 4 Kondensatoren in der Kombination „Spannung", „keine Spannung", „Spannung", „keine Spannung". Dies kann auf dem Papier durch die Kombination „1010" dargestellt werden.

Man bezeichnet eine solche Kombination (unterschiedlicher Länge) als **Datenwort** und jeden ihrer (hier 4) Bestandteile als **Bit.** Genauer: Die Zeichen „1" oder „0" drücken den jeweiligen Wert des betreffenden Bits aus. In einer weiter oben benutzten Ausdrucksweise ist das Bit die kleinste Einheit der Information. Ein Datenwort sieht aus wie eine Zahl des binären Zahlensystems, in dem es nur die Ziffern 1 und 0 gibt. Tatsächlich kann ein Datenwort auch eine Zahl bedeuten. Im allgemeinen besteht aber ein begrifflicher Unterschied zwischen Datenwort und Zahl.

1.3. Speicher und Schnittstellen

In Abschnitt 1.1 wurde die Frage zurückgestellt, woher die Zentraleinheit eines Computers die Befehle holt. Sie holt die Befehle aus einem **Speicher.** Ein Speicher besteht aus einer großen Anzahl von Anordnungen wie z. B. den Kondensatoren in Bild 1.1. Die Anzahl von Bits, die man in einem Speicher unterbringen kann, heißt die **Kapazität** des Speichers. So haben z. B. 256 Anordnungen wie in Bild 1.1 eine Kapazität von 1024 Bit. Dadurch ist aber ein Speicher noch nicht vollständig beschrieben; man muß u. a. auch seine „Organisation" angeben, d. h. die Art seiner Aufteilung.

Zum Speicher gehören auch Vorrichtungen, um den Inhalt festzustellen und ggf. zu ändern; diese Tätigkeiten werden als Lesen und Schreiben des Speicherinhaltes bezeichnet.
Als **Speicherplatz** bezeichnet man eine größere oder kleinere Gruppe von physikalisch realisierten Bits, denen man gleichzeitig Information einschreiben oder auslesen kann. Dazu muß man jeden Speicherplatz als ganzes „adressieren", d. h. erreichen können. Gibt man die Größe der Speicherplätze an, so beantwortet man die Frage nach der Aufteilung des Speichers: z. B. 128 × 8 Bit oder 256 × 4 Bit.

Die **Adresse** eines Speicherplatzes ist eine Zahl, mit deren Hilfe man ihn finden kann. Man kann die Speicherplatzadressen mit den Hausnummern in einer Straße vergleichen. Der Vergleich ist dann korrekt, wenn zu jeder Hausnummer eine immer gleiche Anzahl von Menschen gehört, die den Bits entsprechen. Ebenso wie man zwischen der Hausnummer und den Bewohnern unterscheiden muß, hat man auch zu unterscheiden zwischen der Adresse und dem Informationsinhalt eines Speicherplatzes.

Außer durch Laden von Kondensatoren kann man die Information in Speichern noch durch viele andere physikalische Methoden aufbewahren.

Speicher werden nicht nur benötigt, um Befehle aufzubewahren. Sie können auch Daten enthalten, die zur Durchführung des Programmes benötigt werden.
Hier soll nicht untersucht werden, ob man Befehle mit zu den Daten zählen sollte oder nicht. In dieser Schrift wird immer, wenn von Befehlen die Rede ist, auch das Wort „Befehle" gebraucht werden.

Zur Durchführung von Programmen benötigt man zwei Arten von Daten.
Zur ersten Art gehören solche, die bei jeder Bearbeitung der gleichen Aufgabe diesel-

ben sind, z. B. die Zahl pi = 3,1416 beim Berechnen des Kreisumfanges aus dem Durchmesser. Solche Daten sind gewissermaßen Bestandteile des Programmes, auch wenn sie nicht direkt in Befehlen vorkommen. Sie heißen deshalb **Konstanten** und werden, wenn sie nicht in den Befehlen selbst vorkommen, am Schluß des Programmes gespeichert.

Zur zweiten Art von Daten gehören solche, die bei jeder speziellen Durchführung einer gleichbleibenden Aufgabe andere Werte haben. Im soeben benutzten Beispiel ist das etwa der Kreisumfang; er wird ja erst aus einem zuvor nicht bekannten Durchmesser berechnet. Solche Daten werden erst im Verlauf der Programmdurchführung in einen Speicher eingeschrieben. Sie heißen nicht Konstante, sondern **Variable** (= Veränderliche).

Andere Variable werden von außen in den Computer eingegeben, so im Beispiel der Durchmesser für einen speziellen Fall. Die **Eingabe** kann z. B. bestehen in einem Signal von einem Instrument, von einer Magnetplatte oder von einer Lochkarte; sie kann auch dadurch erfolgen, daß ein Mensch eine Tastatur betätigt. Es gibt also, je nach Art der Aufgabe, unterschiedliche **Eingabe-Geräte.**
Ebenso muß es **Ausgabe-Geräte** geben; denn niemand hat etwas von der Tätigkeit eines Computers, wenn nicht deren Resultate in irgendeiner Form an die Außenwelt gegeben werden. Die **Ausgabe** kann erfolgen z. B. durch Drucker oder Datensichtgerät (für Menschen), durch Lochstreifen zur Weiterverarbeitung oder auch durch Beeinflussung technischer Einrichtungen wie Ventile oder Widerstände. Diese Aufzählung ist keineswegs vollständig.

Die Ein- und Ausgabegeräte sind nicht Bestandteil des Computers. Zwischen Computer und Geräten sind meist Vorrichtungen zur elektrischen Anpassung erforderlich. Eine gedachte Grenzfläche zwischen Computer und Geräten heißt **Schnittstelle, Nahtstelle** oder **Interface** (= Zwischenschicht). Wenn die Anpassungsvorrichtungen serienmäßig zum Computer gehören, denkt man sich die Schnittstelle außerhalb dieser Vorrichtungen.

Die Gesamtheit der Eingabe- und Ausgabe-Geräte wird abgekürzt als „**E/A**" bezeichnet.
Auch die Bezeichnung „Peripheriegeräte" ist üblich (Peripherie = Außenrand).

2. Grobstruktur des Mikrocomputers

2.1. Bausteine

Im 1. Kapitel wurden die Begriffe Zentraleinheit, Speicher und E/A eingeführt. Hier soll auf den Fall des Mikrocomputers spezialisiert werden, und dann gleich auf die Familie SAB 8080.

Bild 2.1 zeigt die wichtigsten Bestandteile eines Mikrocomputers. Zu den Bausteinen, die in der Einleitung erwähnt wurden, gehört die Zentraleinheit; sie heißt im Mikrocomputer der **Mikroprozessor.** Zum Einholen und Ausführen der Befehle enthält der Mikroprozessor eine große Anzahl von speziellen Schaltungen. Als Beispiele: Der Einholung der Befehle dient u.a. eine Dekodier- (= Entschlüsselungs-)Vorrichtung für Befehle; der Ausführung der Befehle dient u.a. eine arithmetisch-logische Einheit.
Für die Besprechung von Softwarefragen genügt aber die Kenntnis der **Register;** das sind interne Speicherplätze des Mikroprozessors. Es gibt Register für spezielle Zwecke und Register zur universellen Verwendung. Hier werden nur die universellen Register besprochen; die anderen werden z.T. dann erwähnt, wenn mit ihnen feste Vorstellungen verknüpft werden können.

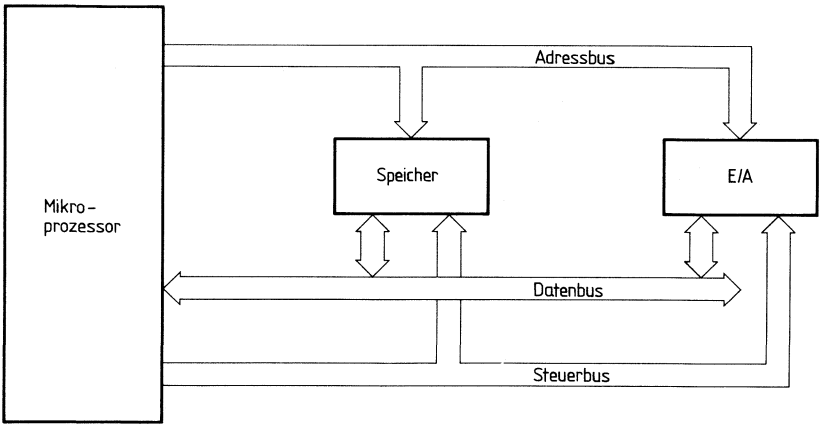


Bild 2.1
Die wichtigsten Teile eines Mikrocomputers

Weil die universellen Register sich im Mikroprozessor befinden, kann in ihnen schneller gelesen und geschrieben werden als in den normalen Speicherplätzen. Weil es nur wenige von ihnen gibt, kann ihre Adresse kurz sein. Im Folgenden wird der Deutlichkeit halber anstatt „Adresse" der universellen Register das Wort „Name" gebraucht. Die Namen der universellen Register sind im SAB 8080:

A (der „Akumulator"),
B und C (auch als „Registerpaar B" oder „BC" zusammenfaßbar),
D und E (auch als „Registerpaar D" oder „DE" zusammenfaßbar) und
H und L (auch als „Registerpaar H" oder „HL" zusammenfaßbar).

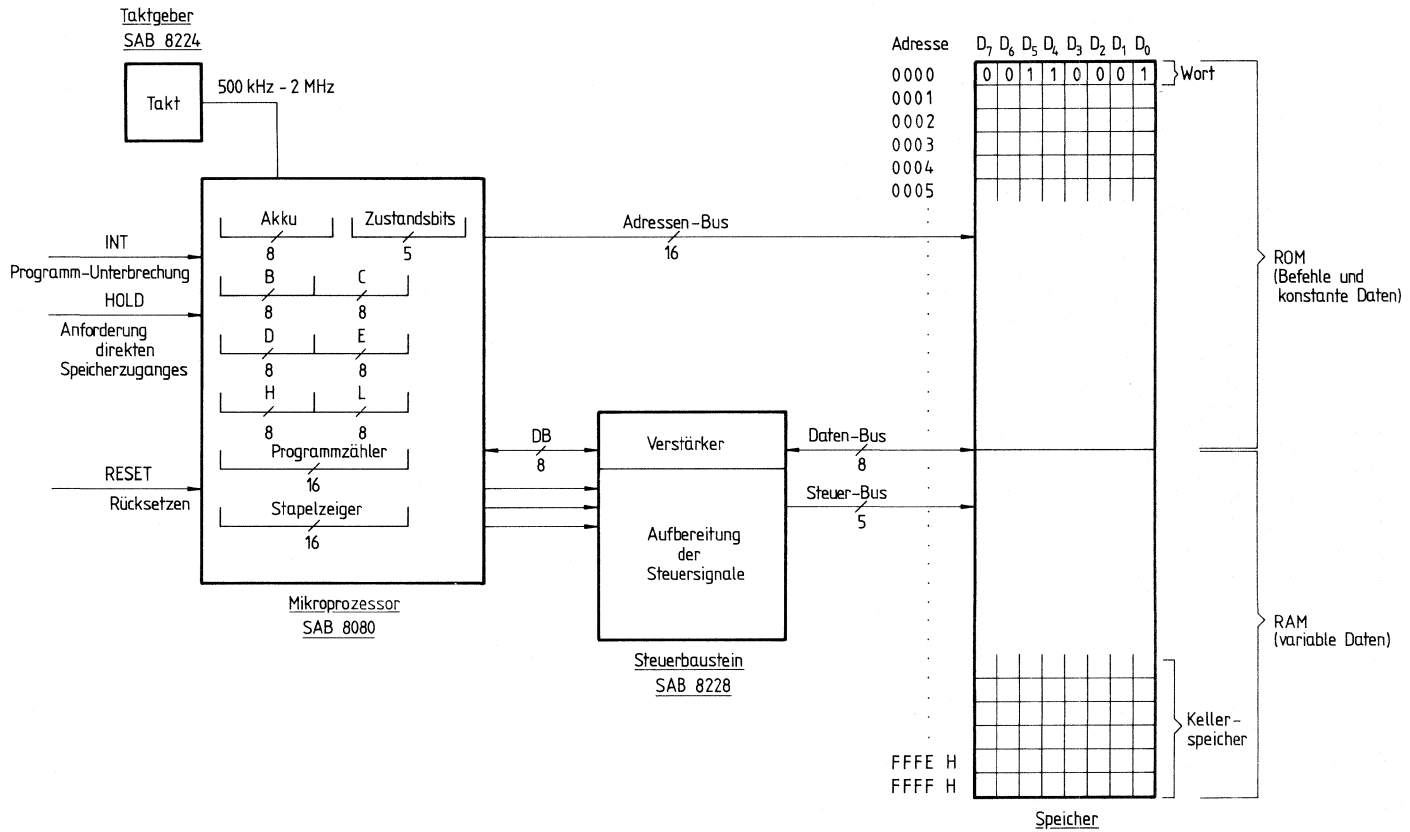


Bild 2.2
Die Register und die Speicherplätze

Jedes Einzelregister (im Gegensatz zum Registerpaar) umfaßt 8 Bit, ebenso wie die Speicherplätze. Bild 2.2 zeigt diese Register und dazu die Speicherplätze in RAM und ROM (s. unten). Einige weitere Einzelheiten im Bild 2.2 werden erst später erläutert.

Der Speicher wird aus **Speicherbausteinen** zusammengesetzt. Speicherbausteine können nach unterschiedlichen physikalischen Prinzipien arbeiten.
Wichtiger ist hier ein anderer Unterschied zwischen zwei Arten von Speicherbausteinen:
Zum Speichern von Befehlen und von konstanten Daten benutzt man meist Speicher, deren Inhalt sich danach nicht mehr verändern läßt. Die Daten dieser Speicher werden durch Abschalten der Versorgungsspannung nicht zerstört. Sie können auch nicht irrtümlich geändert werden.
Man kann also aus diesen Speichern nur lesen, aber nicht in sie schreiben. Deshalb heißt ein solcher Speicher **ROM** (Read-Only-Memory = nur-lese-Gedächtnis; Gedächtnis = Speicher).
Als Arbeitsspeicher für variable Daten braucht man dagegen solche Speicherbausteine, in die man auch etwas Neues hineinschreiben kann. Solche Speicher heißen **RAM** (Random-Access-Memory = Speicher mit wahlfreiem Zugriff). Mit diesem Namen wird eine Eigenschaft betont, die der ROM an sich auch hat: Alle Speicherplätze können gleichschnell erreicht werden.
Im Prinzip können die Befehle für den SAB 8080 auch im RAM gespeichert werden. Der Einfachheit halber wird hier trotzdem der Befehlsspeicher immer auch als ROM bezeichnet, was in der Praxis meistens zutrifft.
Im übrigen ist der Unterschied zwischen RAM und ROM zeitbedingt; die technische Weiterentwicklung wird ihn früher oder später zumindest mildern.
Nach dem üblichen Sprachgebrauch sagt man: Der Speicher besteht aus dem ROM und dem RAM; diese wiederum bestehen aus ROM-Bausteinen und RAM-Bausteinen.
Die Größe des ROM ergibt sich aus der Länge des Programmes. Die Größe des RAM ergibt sich aus der Höchstmenge variabler Daten, die gleichzeitig gespeichert werden müssen.

Der Kasten mit der Bezeichnung „Speicher" in Bild 2.1 steht für die Gesamtheit der Speicherbausteine; ebenso steht der Kasten „E/A" für die Gesamtheit der Anschlüsse an Ein-/Ausgabe-Geräten. E/A-Geräte können auf sehr unterschiedliche Weise an den Mikrocomputer angeschlossen werden. Es kommt vor allem darauf an, ob man immer das gleiche Gerät anschließen will oder ob man möglichst anpassungsfähige Anschlüsse haben möchte. Der letztere Fall liegt dann vor, wenn man komplett bestückte Mikrocomputer ohne festgelegte Verwendung aufbauen will. Dann verwendet man universelle **E/A-Bausteine** aus der Familie SAB 8080. Wird dagegen keine Vielseitigkeit gebraucht, so können wesentlich billigere Vorrichtungen ausreichend sein, z. B. normale TTL-Bausteine.

Den Adressen der Speicherplätze und den Namen der universellen Register entsprechen die **Kanal**nummern der E/A-Anschlüsse. Ein E/A-Baustein kann einen oder mehrere Kanäle versorgen. Wieviel Kanäle für ein bestimmtes Gerät gebraucht werden, hängt vom Einzelfall ab.
Für „Kanal" ist auch die Bezeichnung **„Port"** üblich. Zwischen E/A-Baustein und Mikroprozessor können pro Kanal 8 Bit parallel übertragen werden. Zwischen E/A-Baustein und -Gerät werden (je nach Art des Gerätes) Daten entweder parallel oder seriell übertragen. Für beide Fälle gibt es universell verwendbare Bausteine zur Anpassung der E/A-Geräte an die inneren Leitungen des Mikrocomputers.

2.2. Leitungen

Alle Bausteine sind durch Leitungen miteinander verbunden. Läßt man die Stromversorgung und das Taktsignal außer acht, so gibt es Leitungen für 3 Zwecke.
Erstens müssen Daten zwischen Mikroprozessor einerseits, RAM und E/A andererseits übertragen werden, und es müssen auch Befehle einseitig vom ROM zum Mikroprozessor übertragen werden. Zweitens müssen irgendwie die Adressen und die E/A-Kanalnummern übermittelt werden, unter denen der Mikroprozessor Daten oder Befehle findet oder zu denen er Daten befördert.
Drittens sind auch noch Steuersignale erforderlich; sie gehen oft, aber nicht immer vom Mikroprozessor aus.

Die Steuersignale werden u.a. dadurch erforderlich, daß im Mikrocomputer das Bus-Prinzip angewendet ist. Dieses Prinzip ist heute auch sonst in der Meß- und Regeltechnik dann üblich, wenn eine größere Anzahl von Einzelgeräten miteinander in Wechselwirkung treten soll.
Der **Bus** ist eine Mehrfachleitung mit mehreren parallelen Einzelleitungen. An diese Einzelleitungen werden alle in Frage kommenden Geräte gleichberechtigt angeschlossen, was die Leitungsführung sehr vereinfacht; im Mikrocomputer sind die „Geräte" die Bausteine. Dann muß aber auch dafür gesorgt werden, daß jeweils der richtige Baustein als Sender und die richtigen Bausteine als Empfänger aktiviert werden. Dies geschieht durch Steuersignale, die vom Mikroprozessor ausgehen.

Im System SAB 8080 gibt es einen **Datenbus** und einen besonderen **Adreßbus.** Sie sind im Bild 2.1 eingezeichnet. Die (meist zahlreichen) Bausteine, die durch Datenbus und Adreßbus mit dem Mikroprozessor verbunden werden, sind in diesem Bild ganz pauschal durch „Speicher" und „E/A" dargestellt. Durch Pfeile ist angedeutet, daß der Mikroprozessor Adressen nur aussenden, dagegen Daten auch empfangen kann (Befehle sogar nur empfangen kann).
Als dritter Bus ist der „Steuerbus" eingezeichnet, für den im wesentlichen der Mikroprozessor als Sender dient. Die Bezeichnung „Bus" ist auch in diesem Falle üblich, aber nicht ganz korrekt. Es gibt nämlich keinen Steuerbus, an den alle Bausteine gleichberechtigt angeschlossen wären und über den alle Steuersignale liefen. Vielmehr sind unterschiedliche Bausteine an unterschiedliche Steuer-Einzelleitungen angeschlossen; manche Steuersignale werden auf ihrem Wege noch umgewandelt, und schließlich gehen Steuersignale gelegentlich auch über den Datenbus.
Für eine Softwarenbetrachtung wird aber durch den gedachten Steuerbus das Bild 2.1. übersichtlicher.
Der im genaueren Bild 2.2 eingezeichnete „Steuerbus" ist dagegen wirklich physikalisch vorhanden.

2.3. Größe der Speicherplätze

Als **Wortlänge** eines Computers bezeichnet man diejenige Anzahl von Bit, die die Zentraleinheit in jedem ihrer Register speichern kann und die sie in ihrem Rechenwerk verarbeiten kann. Sinnvollerweise gibt man dann auch den Speicherplätzen die Kapazität einer Wortlänge und dem Datenbus die entsprechende Anzahl von Einzelleitungen. Dann können jeweils sämtliche Bits eines Speicherplatzes parallel übertragen werden (insbesondere zum und vom Mikroprozessor). Die gleiche Anzahl von Bits kann auch maximal von einem Eingabegerät oder zu einem Ausgabegerät gleichzeitig übertragen werden.

Mikrocomputer haben Wortlängen von 4 bis 16 Bit, am häufigsten z.Zt. 8 Bit. Auch der Mikroprozessor SAB 8080 hat eine Wortlänge von 8 Bit.

Eine Anordnung von 8 Bit wird unabhängig vom benutzten Mikrocomputer immer als ein **Byte** (sprich: Bait) bezeichnet. Beim SAB 8080 ist also die Wortlänge gleich einem Byte.

Die Bits in einem Byte werden von rechts nach links mit den Nummern 0, 1, . . ., 6, 7 bezeichnet. Bit 0 entspricht also dem niederwertigsten Bit (LSB), Bit 7 dem höchstwertigsten (MSB).

Somit hat ein Speicherplatz, ob RAM oder ROM, die Kapazität 1 Byte und der Datenbus 8 Einzelleitungen. Speziell die Länge der Befehle beträgt in einem Programm für den SAB 8080 1, 2 oder 3 Byte. Als erstes, ggf. einziges Byte steht in jedem Fall ein als **Operationscode** bezeichnetes Byte (eigentlich müßte es eher als codierte Information bezeichnet werden).
Der Operationscode liefert den allgemeinen Charakter eines Befehles. Ein weiteres Byte kann notwendig sein, um einen Operanden anzugeben (das ist ein Datenwort, auf das der Befehl angewendet wird), oder auch eine E/A-Kanalnummer. Sogar zwei weitere Byte braucht man, um Speicheradressen anzugeben oder um 2-Byte-Operanden zu nennen.

Da 1 Byte 8 Bit enthält und jedes Bit den Wert 0 oder 1 haben kann, gibt es $2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 = 2^8$ („zwei hoch acht") = 256 verschiedene Bytes.
Da jeder Operationscode die Länge 1 Byte hat, gibt es also auch 256 verschiedene Möglichkeiten für Operationscodes. Diese Anzahl wird nahezu ausgenutzt, denn der SAB 8080 und der SAB 8085 kennen 244 verschiedene Befehle (außer dem RIM und SIM des SAB 8085).

Je nach Zusammenhang können die Bytes auch ganz andere Dinge bedeuten als Operationscodes. Wenn z. B. ein Byte eine binäre Zahl bedeutet, so liegt diese zwischen Null = 00000000 B und Zweihundertfünfundfünfzig = 11111111 B.
Hierbei ist ein nachgestelltes „B" zur Kennzeichnung einer binären Zahl benutzt worden; ebenso bezeichnet man eine Hexadezimalzahl durch ein nachgestelltes „H" und eine Dezimalzahl, wenn nötig, durch ein nachgestelltes „D".

2.4. Anzahl der Speicherplätze

Die Länge der Speicherplatzadressen bestimmt die Maximal-Anzahl möglicher Speicherplätze. Haben die Adressen eine Länge von nur 1 Bit, so gibt es maximal 2 Plätze; sie haben die Adressen 0 und 1. Mit 2 Bit kann man schon $2 \times 2 = 2^2 = 4$ Plätze adressieren, ihre Adressen sind 00,01, 10 B und 11 B.
Mit einer Adresse der Länge 1 Byte gibt es $2^8 = 256$ Möglichkeiten. Das ist noch nicht sehr viel.
Beim SAB 8080 hat der Adreßbus 16 Einzelleitungen, und jede Adresse ist dem entsprechend 2 Byte lang. Es gibt daher $2^{16} = 65536$ verschiedene Adressen und mögliche Speicherplätze.
Die möglichen Adressen laufen von Null = 00000000 00000000 B bis 65535 D = 11111111 11111111 B.
Will man aus irgendwelchen Gründen eine Adresse speichern, so benötigt man dafür 2 Speicherplätze.

Der Programmierer legt den Bedarf an RAM- und ROM-Plätzen fest. Die Verteilung der Adressen zwischen RAM und ROM hängt von der Hardware ab. Häufig gibt man dem ROM die niedrigsten Adressen. Der Grund hierfür ist: Nach dem Einschalten oder auch nach einem Steuersignal RESET (= setze zurück, d. h. fange neu an) holt der SAB 8080/85 immer zuerst den Inhalt der untersten Adresse „Null" ein. Am einfachsten ist es, wenn dort dann auch das Programm beginnt.

Der z.Zt. kleinste ROM-Baustein enthält 256 Speicherplätze. Besteht der ganze ROM aus einem solchen Baustein, so laufen die ROM-Adressen z.B. von Null bis 00000000 11111111 B = 255 D; RAM-Adressen sind dann möglich von 00000001 00000000 B = 256 D aufwärts.

Zum leichteren Aussprechen großer Potenzen von 2 benutzt man die Abkürzung $K = 2^{10} = 1024$. In dieser Ausdrucksweise kann der SAB 8080 maximal 64 K Speicherplätze adressieren (sprich „64 ka"). Die maximale Speicherkapazität beträgt 64 K Byte (sprich „64 ka Byte", nicht etwa „64 Kilobyte"), oder auch 512 K Bit, das sind 524288 Bit.
Z.Zt. sind gängig Speicherbausteine von 4 K Bit . . . 16 K Bit . . . 64 K Bit, und die Entwicklung wird weitergehen. Magnetplatten (Floppy Disk) für Mikrocomputer haben z.B. eine Kapazität von 250 K Byte = 2000 K Bit.
Derartige Kapazitäten werden für die achtziger Jahre auch bei einzelnen Bausteinen erwartet.

2.5. Adressierung der E/A-Kanäle

Beim Addressieren unterscheidet man im Normalfall durch Steuersignale zwischen Speicher und E/A. Der Mikroprozessor erhält etwa einen Befehl, der Datenaustausch mit E/A anordnet; er sendet dann Steuersignale aus, die sich nicht an den Speicher richten, sondern an die Eingabe- oder Ausgabe-Stellen.

Die Zahl, die der Mikroprozessor dann auf den Adreßbus gibt, wird nicht als Speicheradresse aufgefaßt, sondern als Kanalnummer. Diese Zahl ist (im Gegensatz zu Speicheradressen) nur 1 Byte lang; es gibt nämlich maximal 256 Eingabe-Kanäle und 256 Ausgabe-Kanäle. Wieviele Kanäle für ein Gerät benötigt werden, hängt vom Einzelfall ab; man ist völlig frei in der Zuordnung.

Ausnahmsweise kann man die E/As auch wie Speicherplätze behandeln, indem man die entsprechenden Leitungsverbindungen herstellt. Braucht man z.B. nicht mehr als die Hälfte der möglichen 65536 Speicherplätze, so kann man die andere Hälfte (etwa alle mit „1" beginnenden Adressen) den E/As zuweisen. Voraussetzung ist aber, daß die Geräte mit ihren Anschlußschaltungen sich auch physikalisch wie Speicherplätze benehmen. Das ist z.B. der Fall bei Lumineszenzdioden-Anzeigen; sie benehmen sich wie Speicherplätze, in die man hineinschreibt. Es ist nicht der Fall bei langsameren Geräten; diese müssen dem Mikroprozessor eine Datenübermittlung zuvor anzeigen oder hinterher in irgendeiner Weise quittieren, benehmen sich also nicht wie Speicherplätze.

3. Erstes Programmbeispiel

Aus den ersten Kapiteln ist zu rekapitulieren: Befehle werden (i.a.) im ROM gespeichert, Daten dagegen im RAM. In beiden Teilen des Speichers faßt ein Speicherplatz jeweils 1 Byte. Im ganzen Speicher ist jeder Speicherplatz eindeutig gekennzeichnet durch eine Adresse der Länge 2 Byte. Ein Byte umfaßt 8 Bit. Ein Befehl belegt 1 bis 3 Speicherplätze; auf dem ersten (ggf. einzigen) Platz steht der Operationscode. Der Mikroprozessor besitzt interne Register von ebenfalls der Kapazität 1 Byte. Sie können auch paarweise zusammengefaßt werden.

Die aus Einsen und Nullen bestehenden Bytes werden direkt in digitale Signale umgewandelt; deshalb heißt die aus ihnen bestehende Sprache **Maschinensprache.** Um den Umgang mit der Maschinensprache zu erleichtern, wird hier in üblicher Weise die Stenografie mit Hilfe von Hexadezimalziffern eingeführt, siehe Kapitel 9.

Die Daten und Adressen in den Beispielen werden weitgehend willkürlich ausgewählt werden.
Bei den vorkommenden Befehlen wird die Befehlsliste des SAB 8080 benutzt. Es ist üblich, hierzu eine andere Art der Abkürzung speziell für Befehle einzuführen, die aus der Maschinensprache herausführt. Das sind die **mnemonischen Abkürzungen.** Der Name deutet an, daß die Abkürzungen an den Inhalt des betreffenden Befehles erinnern sollen.
Man kann beliebig viele Systeme von mnemonischen Abkürzungen ausdenken. Beim SAB 8080 werden (wie in vielen anderen Fällen) aus dem Englischen abgeleitete Abkürzungen verwendet.
Die mnemonischen Abkürzungen erleichtern die Programmierarbeit erheblich, und darin besteht ihr Zweck.

Nun zum Programmbeispiel. Es sollen zwei im RAM befindliche Zahlen zwischen 0 und 255 addiert werden. Diese Zahlen können also durch je ein Byte ausgedrückt werden; jede von ihnen belegt einen Speicherplatz. Nach der Addition soll das Resultat auf einem dieser beiden Speicherplätze abgelegt werden.

> Durch das Wort „ablegen" oder „abspeichern" oder „laden" soll ausgedrückt werden: Die irgendwo im Computer vorhandene Information wird einem bestimmten Speicherplatz oder Register einkopiert. Konkret im vorliegenden Fall:
>
> Das Resultat der Addition steht nach der Addition zuerst im Akkumulator, und es soll nun in einen der beiden genannten Speicherplätze geschrieben werden. Dieser enthält dann also nicht mehr einen der beiden Summanden, sondern die Summe. Im Akkumulator steht das Resultat danach immer noch, und zwar so lange, bis es durch ein anderes Datenwort überschrieben wird oder bis der Computer ausgeschaltet wird.
>
> Die Einzelheiten werden an Hand der Bilder 3.1 und 3.2 deutlich.

Tabelle 3-1 zeigt den Programmausschnitt. In der linken Spalte stehen die betreffenden ROM-Adressen, hexadezimal abgekürzt; so steht z.B. die erste Adresse 0021 H für 00000000 00100001 B, oder auch 33 D.
In der zweiten Spalte steht der Inhalt eines jeden ROM-Platzes, ebenfalls hexadezimal abgekürzt. Da der Inhalt eines Speicherplatzes 8 Bit umfaßt, sind zur Abkürzung nur 2 Hexadezimalziffern erforderlich; für die Adressen mit ihren 16 Bit braucht man dagegen 4 Hexadezimalziffern.
Die Zeichen in der zweiten Spalte sind (im Gegensatz zur ersten Spalte) i.a. keine Zahlen. Das Zeichen 3A im ersten Speicherplatz ist z.B. ein Operationscode. Das nächste Zeichen 2D ist allerdings (wie sich zeigen wird) Teil einer Adresse und damit einer Zahl.

In der dritten Spalte steht ebenfalls der Inhalt der ROM-Plätze, hier jedoch jeweils für einen ganzen Befehl durch mnemonische Abkürzung zusammengefaßt.

Tabelle 3-1

ROM-Adresse hexadezimal	ROM-Inhalt hexadezimal	ROM-Inhalt, Op. Code mnemonisch, Operanden usw. hexadezimal	Zeitpunkt im Programm dezimal	Inhalt des Reg. Paares HL hexadezimal	Inhalt des Akku hexad.	Inhalt des RAM-Platzes A12D H hexadezimal	Inhalt des RAM-Platzes ØE01 H hexadezimal
			Ø	?	?	B6	1A
Ø021	3A	LDA A12DH					
Ø022	2D						
Ø023	A1						
			1	?	B6	B6	1A
Ø024	21	LXI H,ØE01H					
Ø025	Ø1						
Ø026	ØE						
			2	ØE01	B6	B6	1A
Ø027	86	ADD M					
			3	ØE01	DØ	B6	1A
Ø028	77	MOV M,A					
			4	ØE01	DØ	B6	DØ

Ein Programm läuft normalerweise in der Reihenfolge der Befehlsadressen ab (Ausnahmen in Kapitel 6). Daher kann man die Zeilenfolge von oben nach unten in Tabelle 3-1 auch als Zeitskala auffassen. Dies ermöglicht die Angabe von Zeitpunkten in der vierten Spalte. Für diese Zeitpunkte sind schließlich die jeweiligen Inhalte einiger Speicherplätze oder Register in den letzten Spalten eingetragen.

Zur Zeit Null befindet sich im RAM-Platz mit der Adresse A12DH = 10100001 00101101 B das Datenwort B6H; es soll hier die Zahl 182D = 10110110B bedeuten.
Im Platz 0E01 H befindet sich die Zahl 1AH = 26D = 00011010B. Diese beiden Zahlen sollen addiert werden. Die Verhältnisse zum Zeitpunkt 0 zeigt Bild 3.1; der ROM-Inhalt ist natürlich zu allen Zeiten derselbe.

Es folgt nun der erste eingeholte Befehl. Alle Befehle werden an Hand der Tabelle 4-1 ausführlich kommentiert werden; deshalb genügt es hier, kurz die Bedeutung der vorkommenden Befehle anzugeben.
Der hexadezimal geschriebene Operationscode 3A bedeutet: „Lade in das Register A (den Akkumulator) dasjenige Byte (denjenigen Speicherplatzinhalt), dessen Speicherplatzadresse in den folgenden 2 Bytes dieses Befehles gegeben wird."
Beim Speichern von Adressen und von anderen 2-Byte-Zahlen kommt immer die niederwertige Hälfte der Zahl in den Speicherplatz mit der niederen Adresse. Im vorliegenden Fall steht deshalb die niederwertige Hälfte 2D der gespeicherten Adresse A12D auf dem ROM-Platz mit der niederen Adresse 0022, und die höherwertige Hälfte A1 der gespeicherten Adresse auf dem ROM-Platz mit der höheren Adresse 0023.
Das ist in sich konsequent, bringt aber auf dem Papier eine Abweichung von der gewohnten Reihenfolge. In der mnemonischen Abkürzung ist dagegen für den menschlichen Leser die gewohnte Reihenfolge eingehalten.

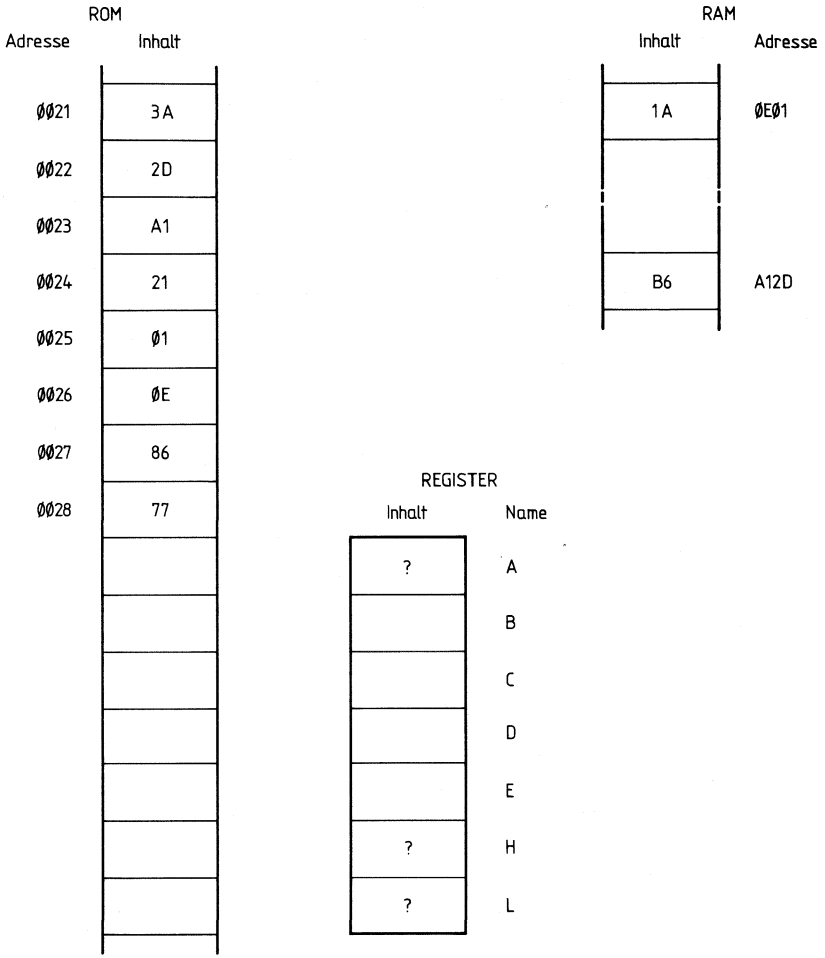


Bild 3.1
Zum Programmbeispiel der Tabelle 3-1;
Speicher- und Registerinhalte zum Zeitpunkt 0.
Adressen und Daten hexadezimal wie bei allen folgenden Bildern dieser Art.
Achtung: Die niederen Adressen sind oben im Bild

Die mnemonische Abkürzung lautet LDA Adr; dabei steht LDA für „load direct accumulator" (= lade den Akkumulator direkt) und Adr als allgemeine Abkürzung einer Adresse.
Zum Zeitpunkt 1 ist der Befehl ausgeführt.

Der nächste hexadezimale Operationscode 21 bedeutet: „Lade in das Registerpaar HL das angegebene 2-Byte-Wort." Auch hier folgt zuerst die niederwertige Hälfte des Wortes. Das Datenwort bedeutet hier zufällig eine Adresse; das müßte (im Gegensatz zum ersten Befehl) nicht so sein. Im vorliegenden zweiten Befehl tritt die Adresse aber nicht als solche auf, sondern einfach als Operand; dies wird sich im dritten Befehl ändern.
Die mnemonische Abkürzung lautet allgemein LXI rp, D16. In „LXI" steht L für „load"; der Buchstabe X wird gelegentlich (aber nicht konsequent) verwendet, um die Beschäftigung mit einem Registerpaar anzudeuten. Der Buchstabe I schließlich steht für „immediate" (= unmittelbar); er besagt, daß der Operand unmittelbar aus dem ROM entnommen werden kann und daher eine Konstante im Sinne von Abschnitt 1.3 ist.
„rp" steht allgemein für Angabe eines Registerpaares (z. B. BC, DE oder HL). „D16" steht für Angabe einer 16-Bit-Konstanten. Im vorliegenden Fall heißt das erste „H" in LXI H, ØEØ1H: „Registerpaar HL". Das „H" am Schluß heißt „hexadezimal"; anstatt ØEØ1H könnte auch dastehen ØØØØ111ØØØØØØØ1B oder 3585D oder einfach 3585.
Zum Zeitpunkt 2 ist der Befehl ausgeführt.

Der dritte Befehl 86 (hexadezimal) ohne weitere Angaben besagt: „Addiere zum Inhalt des Akkumulators diejenige Zahl, deren Adresse im Registerpaar HL steht." Hier ist der Operand des zweiten Befehls zur Operandenadresse des dritten Befehls geworden.
Die mnemonische Abkürzung lautet ADD M für „add memory" (= addiere Speicherplatz).
Zum Zeitpunkt 3 steht die Summe 182D + 26D = 208D = DØH im Akkumulator.

Der letzte Befehl 77 (hexadezimal) besagt „speichere den Inhalt des Akkumulators auf denjenigen Platz, dessen Adresse im Registerpaar HL steht."
Die mnemonische Abkürzung ist MOV M, r; dabei steht MOV für „move" (= befördere). „M" heißt wieder „memory", und „r" steht für die Angabe eines Registers. Hier ist zuerst der Bestimmungsort der zu befördernden Daten angegeben, danach der Ursprungsort.
Die Verhältnisse zum Zeitpunkt 4 zeigt Bild 3.2.

Es kann vorkommen, daß die Addition zweier 8-Bit-Zahlen ein Ergebnis bringt, das nicht durch 8 Bit ausgedrückt werden kann. Beispiel:

$$\begin{array}{rr} & 11111111\ \text{B}\ (=\ 255\ \text{D}) \\ + & \underline{00000001\ \text{B}\ (=\ \ 1\ \text{D})} \\ = & 100000000\ \text{B}\ (=\ 256\ \text{D}) \end{array}$$

Die Frage, was bei einem solchen „Überlauf" zu geschehen hat, wird in Abschnitt 4.5 und in Kapitel 9 behandelt.

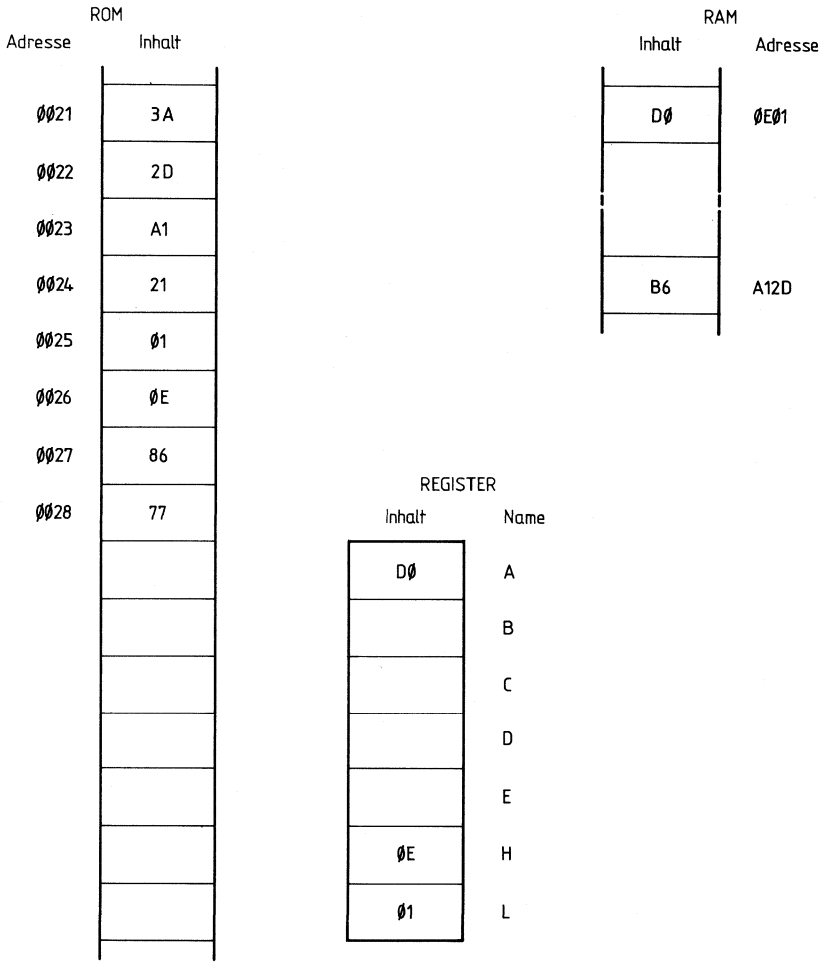


Bild 3.2
Zum Programmbeispiel der Tabelle 3-1;
Speicher- und Registerinhalte zum Zeitpunkt 4

4. Befehlsvorrat

4.1. Verschiedene Arten von Befehlen

Bild 4.1 zeigt eine (sehr einfache) Anwendung für einen Mikrocomputer. In einer Einbahnstraße sind zum Zweck der Geschwindigkeitsmessung zwei Lichtschranken quer zur Fahrbahn gelegt. Jedem Lichtstrahl ist ein Bit im Eingabekanal Nr. 20 zugeordnet. Durch Fotodioden und zugehörige Schaltungen wird dafür gesorgt, daß bei ungestörtem Lichtstrahl das Bit den Wert 1 hat, sonst 0. Einzelheiten siehe Abschnitt 8.2.
Das im ROM gespeicherte Programm veranlaßt die folgende Tätigkeit des Mikroprozessors:
Das Signal vom Eingabekanal wird in den Akkumulator hereingeholt. Der Mikroprozessor prüft, ob das Bit Nr. 0 schon den Wert 0 bekommen hat, ob also von links ein Fahrzeug hereinfährt. Die Prüfung wird solange wiederholt, bis dies der Fall ist. Dann setzt der Mikroprozessor einen Zeit-Zählvorgang in Tätigkeit. Er prüft darauf, ob auch der rechte Lichtstrahl schon gestört wird. Ist dies der Fall, wird der Zählvorgang beendet. Aus dem Stand des Zählers wird durch eine vorgegebene Formel die Geschwindigkeit des Fahrzeuges berechnet und im Ausgabekanal 21 zur Anzeige ausgegeben.

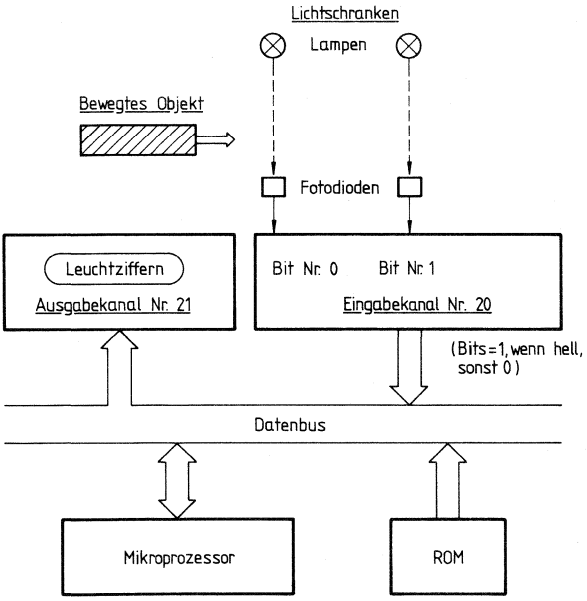


Bild 4.1
Geschwindigkeitsmessung

Das Hereinholen des Signales vom Eingabekanal Nr. 20 in den Akkumulator wird als Transfer von Daten bezeichnet, ebenso das Herausgeben der errechneten Geschwindigkeit zum Ausgabekanal Nr. 21. Transfer ist eine Beförderung von einer bestimmten

Stelle zu einer bestimmten anderen Stelle. Die „beförderten" Daten sind danach allerdings nicht von ihrem Ursprungsort verschwunden; hier ist aber allein wichtig, daß die Daten jedenfalls am Bestimmungsort eingeschrieben werden.
Alle Befehle, die einen Transfer von Daten zwischen Registern, Speicherplätzen und E/As anordnen, heißen Transferbefehle.

Eine andere Art von Befehlen ordnet die Durchführung arithmetischer Operationen an. Unter Arithmetik versteht man die Mathematik der Grundrechenarten. Anstatt des Fachausdruckes „arithmetische Operation durchführen" kann man also auch sagen „rechnen". Gerechnet wird im Beispiel von Bild 4.1 dann, wenn aus der gemessenen Zeit die Geschwindigkeit ermittelt werden soll.

Im Beispiel wird ferner mehrfach geprüft, ob die Bits Nr. 0 und 1 vom Eingabekanal 20 noch = 1 oder schon = 0 sind. Bei dieser Prüfung benutzt man oft eine andere Art von Operationen als arithmetische, nämlich die sogenannten logischen Operationen. Auch für diese gibt es besondere Befehle.

Wenn im Beispiel der Mikrocomputer auf ein Fahrzeug wartet, durchläuft er einen gewissen Programmteil immer wieder, bis der linke Lichtstrahl gestört wird. Vom Ende dieses Programmteiles zu seinem Anfang führt ein besonderer „Sprungbefehl". Die Sprungbefehle und einige Transferbefehle sind die einzigen, die noch nicht im vorliegenden Kapitel 4 besprochen werden, sondern erst im Kapitel 6.

Neben den genannten 4 großen Gruppen von Befehlen gibt es noch einige Befehle für spezielle Zwecke.

Die im Folgenden benutzten Tabellen 4 - 1 bis 4 - 4 finden Sie am Schluß des Buches.

In Tabelle 4-1 ist die Einteilung in die oben aufgezählten Befehlsarten berücksichtigt. In jeder Spalte steht links der Operationscode in Maschinensprache, hexadezimal abgekürzt; in der Mitte die mnemonische Abkürzung des Operationscodes und, soweit nötig, rechts eine mnemonisch abgekürzte Angabe über Operanden, Adressen und dergleichen. Wird eine solche Angabe gemacht, so muß der Befehl deshalb noch nicht aus mehr als einem Byte bestehen. Ein zusätzliches Byte ist dann nötig, wenn rechts steht „D8" (= 8-Bit-Daten) oder „Nr" (= E/A-Kanalnummer); zwei zusätzliche Bytes sind nötig, wenn dort steht „D16" oder „Adr" (= Speicheradresse).

Tabelle 4-1 ist zweckmäßig zum Auffinden des hexadezimalen Operationscodes für einen bestimmten (hier mnemonisch abgekürzten) Befehl. Tabelle 4 - 4 bringt umgekehrt die Befehle in der Reihenfolge der hexadezimalen Operationscodes. Die mnemonischen Abkürzungen sind in Tabelle 4 - 2 erklärt. Dort wird auch der Speicherplatzbedarf eines jeden Befehles angegeben, ferner der Zeitverbrauch durch Angabe der benötigten „Takte" (siehe Abschnitt 5.1). Aus Tabelle 4 - 2 entnimmt man auch, daß von den 256 Möglichkeiten für verschiedene Operationscodes nur 10 nicht ausgenutzt werden. Tabelle 4-3 ist inhaltlich identisch mit Tabelle 4-2, nur die mnemonischen Abkürzungen sind in alphabetischer Reihenfolge aufgelistet.

Die Beispiele für die Diskussion der Befehlsliste und erst recht die späteren, anspruchsvolleren Beispiele sollen praxisnah in Form des **„Assembler-Listings"** vorgeführt werden. Das Assembler-Listing ist ein Protokoll des Assemblierungsvorganges. Bei diesem Vorgang wird ein Programm, das der Programmierer in bequemer Symbolik („Assemblersprache") geschrieben hat, durch einen Computer in die Maschinensprache übersetzt; dies wird im 8. Abschnitt näher ausgeführt. Die Assemblersprache besteht aus den mnemonischen Abkürzungen, die zu Beginn des Kapitels 3 eingeführt wurden.

ØØ21 ØØ24 ØØ27 ØØ28	3A2DA1 21Ø1ØE 86 77	HOL1: HOL2: ZURUE:	LDA LXI ADD MOV	ØA12DH H,ØE Ø1H M M,A	;ERSTE ZAHL IN AKKU. ;ADDEND IN RPH. ;ADDITION. ;RESULTAT IM SPEICHER.
1. ROM- Adr. hex.	2. ROM- Inhalt hex.	3. Symbol. Adresse	4. Op.- Code mnem.	5. Restl. Befehls- teile mnem.	6.
Objekt- programm		Quellenprogramm			Kommentar

Die symbolischen Adressen (3. Spalte) sind nichts weiter als Markierungen dieser Zeile und weitgehend dem Ermessen des Programmierers überlassen. Sie sind dann notwendig, wenn an anderer Stelle des Programmes auf diese Zeile Bezug genommen werden soll, ohne daß der Programmierer schon die endgültigen Adressen im ROM kennt. Näheres im Abschnitt 6.1.

Bild 4.2
Assembler-Listing am Beispiel der Programmstelle von Tabelle 3-1.
RPH heißt Registerpaar HL

Im Protokoll stehen 6 Spalten (siehe Bild 4.2). In der 3. bis 5. Spalte steht das „Quellenprogramm", das der Programmierer geschrieben hatte, in der 6. Spalte sein Kommentar dazu. In der 1. und 2. Spalte steht das „Objektprogramm", das der assemblierende Computer daraus gemacht hat. Im einzelnen:

4. Spalte: Operationscode, mnemonisch abgekürzt.
3. Spalte: Symbolische Adresse dieses Operationscodes (ggf.).
   Zweck der symbolischen Adresse: Markierung dieses Speicherplatzes, falls er in anderen Befehlen genannt wird; dort muß dann (in der 5. Spalte) dieselbe symbolische Adresse genannt werden. Beim Assemblieren teilt der Computer jeder symbolischen Adresse eine absolute Adresse zu.
5. Spalte: Operanden oder Adressen wie jeweils rechts in Tabelle 4-1. Adressen meist symbolisch.
1. Spalte: Adresse des Operationscodes absolut, als **hexadezimale Zahl.**
2. Spalte: Befehl **hexadezimal abgekürzt.**

Genau genommen, wird durch den Operationscode der Maschinensprache nicht immer nur der Inhalt der 4. Spalte ausgedrückt, sondern oft auch ganz oder teilweise der Inhalt der 5. Spalte. Die Assemblersprache zieht also die Grenze zwischen „Operationscode" und „Operand" etwas anders als die Maschinensprache.

Weitere Einzelheiten z. B. über formale Vorschriften sind hier nicht erforderlich. Als Beispiel zeigt Bild 4.2 ein Assembler-Listing der Programmpassage nach Tabelle 3-1. In diesem Fall können die Kommentare rechts nur primitiv sein (sie sagen hier nichts anderes, als was die Befehle sowieso besagen); denn es gibt hier keine andere Programmstelle, mit der man im Kommentar Zusammenhänge herstellen könnte. In einem realen Fall sollte jedoch der Kommentar den Programmierer an das erinnern, was nicht unmittelbar aus den hingeschriebenen Befehlen hervorgeht.

4.2. Transferbefehle für 8 Bit

Ganz oben in der Befehlsliste Tabelle 4-1 finden sich die Transferbefehle. Sie sind hier weiterhin unterteilt in 8 Bit-Transfer und 16 Bit-Transfer. Da der Datenbus 8 Einzelleitungen hat, kann ein 8 Bit-Transfer vom und zum Mikroprozessor in einem Schritt stattfinden. Dasselbe gilt für Transfer innerhalb des Mikroprozessors von einem Register zum anderen; der Mikroprozessor hat einen „inneren Datenbus" mit ebenfalls 8 Einzelleitungen. 16 Bit-Transfer (sofern er notwendig ist) findet zweckmäßig mit eigens dazu bestimmten Befehlen statt. Bei der Ausführung solcher Befehle sind zwar im Mikroprozessor kompliziertere Vorgänge erforderlich; aber im ganzen wird durch solche Befehle Arbeitszeit des Computers gespart (Arbeitszeit des Programmierers sowieso, und auch ROM-Plätze).

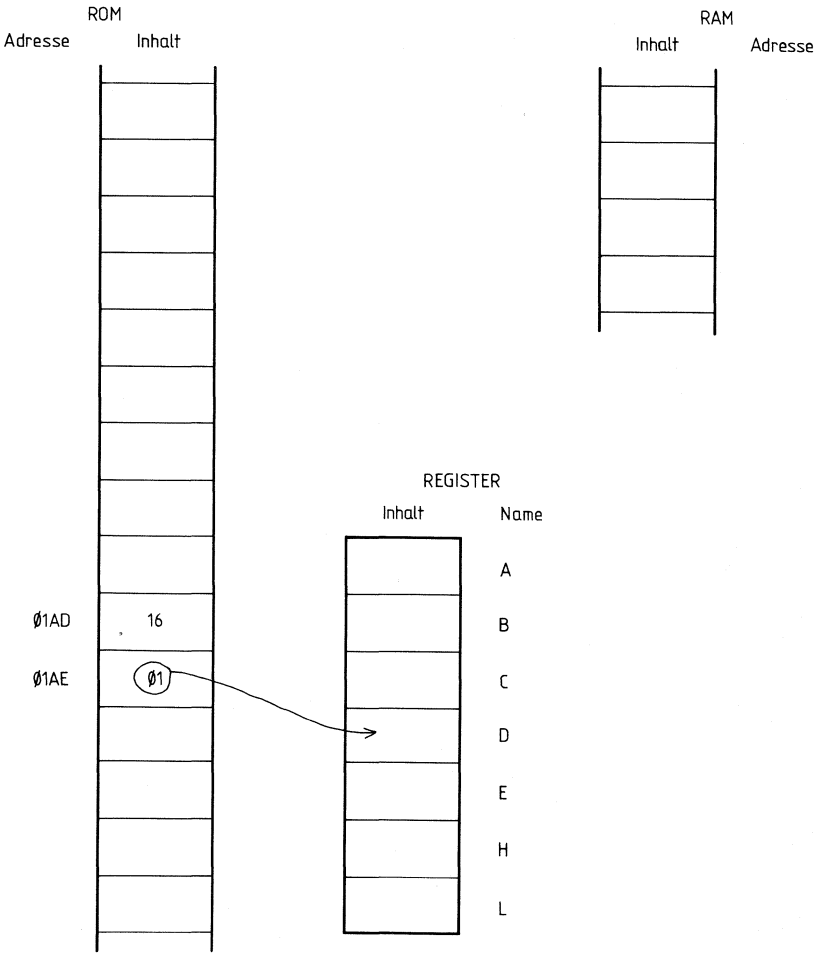


Bild 4.3
Der Befehl MVI D, D8 mit D8 = Ø1

Zur Vermeidung von Mißverständnissen sei betont: Die Unterteilung in 8 Bit-Transfer und 16 Bit-Transfer ist eine andere als die Unterteilung in 1-Byte-Befehle und mehr-Byte-Befehle! Die erste Unterteilung bezieht sich auf die Länge von Operanden, die zweite auf die Länge der Befehle selbst.

Zur Bedeutung der nun zu besprechenden Befehle wird auch auf Tabelle 4-3 verwiesen.

Die Transferbefehle in Tabelle 4-1 beginnen mit den Ausgabe- und Eingabebefehlen OUT und IN; als zweites Byte tritt im Befehl die Nummer des E/A-Kanals auf.
Ob das betreffende Gerät den Aus- oder Eingabebefehl im Arbeitstempo des Mikroprozessors ausführen kann, bedarf jeweils einer besonderen Überlegung. Notfalls muß durch Programm- oder durch Hardware-Maßnahmen auf die Eigenschaften des Gerätes Rücksicht genommen werden.

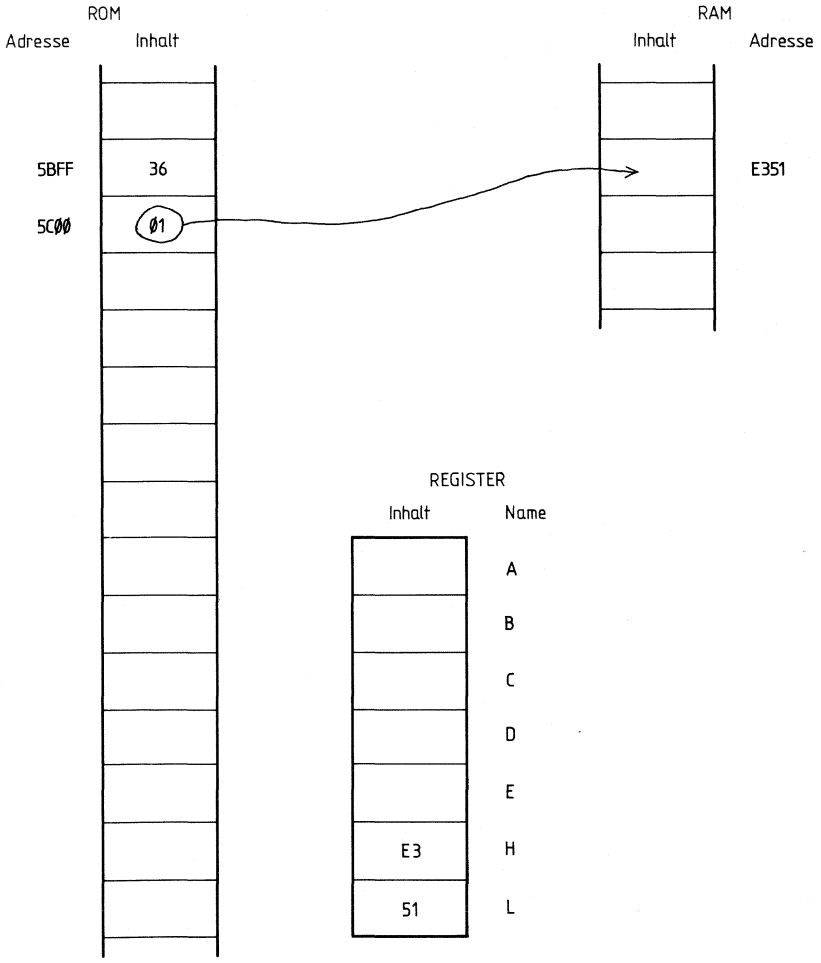


Bild 4.4
Der Befehl MVI M, D8 mit D8 = Ø1 bei dem angegebenen Inhalt von Reg.-Paar HL

Alle übrigen Transferbefehle ordnen Datentransfer innerhalb des Mikrocomputers an. Die Gruppe MOVE IMMEDIATE (= bringe unmittelbar) bringt jeweils eine 8 Bit-Konstante aus dem Programm zur angegebenen Stelle, also z. B. aus dem ROM. Die Konstante ist das 2. Byte des Befehles. Dagegen wird der Bestimmungsort durch den Operationscode ausgedrückt.
Als Bestimmungsort kommen in Frage die 7 Register A, B, C, D, E, H und L. Ferner kann man einen Speicherplatz M (memory = Gedächtnis) angeben; das ist immer „der Speicherplatz, dessen Adresse im Speicherpaar HL steht". Bild 4.3 und 4.4 zeigen die Vorgänge.

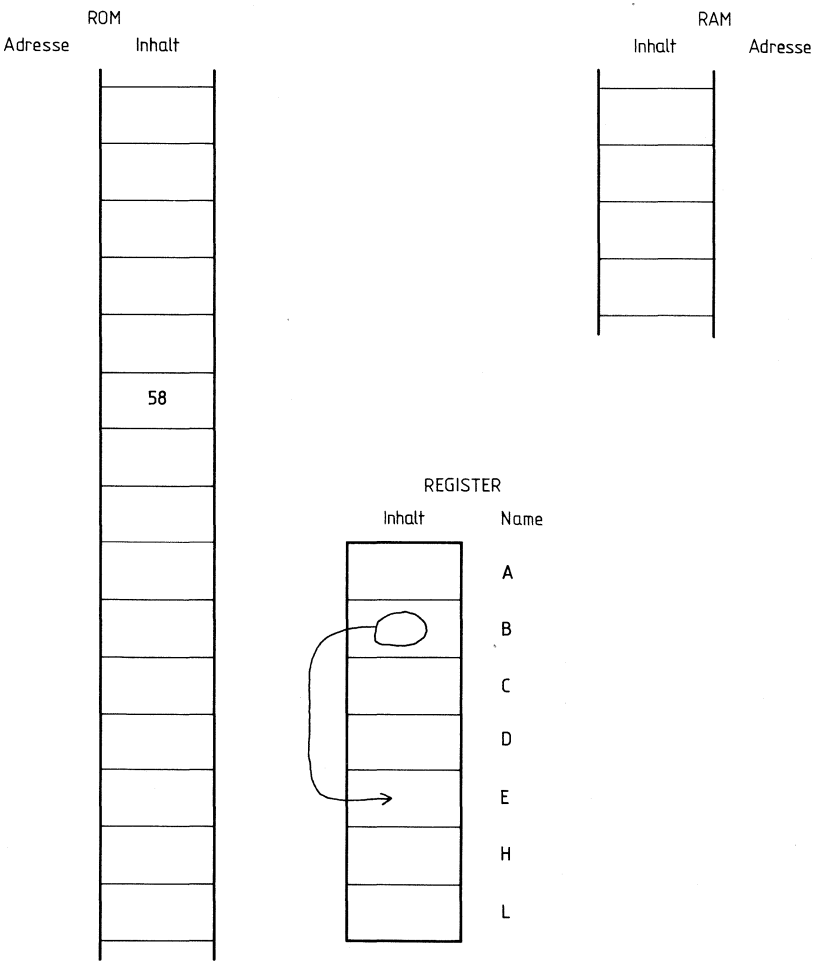


Bild 4.5
Der Befehl MOV E, B

Als Beispiel folgt das Löschen (mit Null laden) des Registers C; im Assembler-Listing sieht das so aus:

Ø2C5	ØEØØ	LOESC:	MVI	C,Ø	;RC GELOESCHT
(z. B.)		(z. B.)			(z. B.)

Hierbei steht „RC" für Register C.

Die Befehle vom Typ MOV erlauben es, den Inhalt eines jeden der 7 Register und des Speicherplatzes M zu einem anderen Partner dieser Gruppe zu transferieren. Die Registerinhalte kann man sogar „zu sich selbst" transferieren; dies bewirkt nichts als Zeitverbrauch.
Die MOV-Befehle sind 1-Byte-Befehle. Bild 4.5 und 4.6 geben Beispiele.

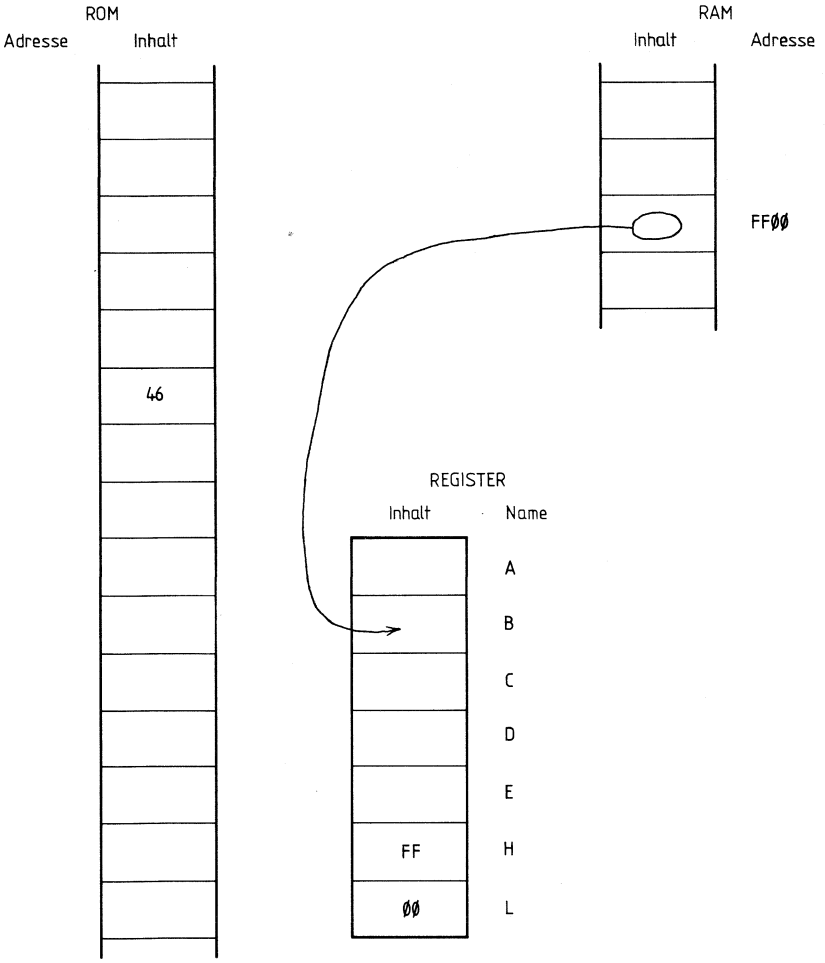


Bild 4.6
Der Befehl MOV B, M bei dem angegebenen Inhalt von Reg.-Paar HL

Weiteres Beispiel: Der Inhalt von Register D wird auf Ausgabekanal Nr. 255 D gegeben durch

3A8C	7A	MOV	A, D
3A8D	D3FF	OUT	255D

Der Programmierer kann, wie hier geschehen, die Kanalnummer auch dezimal schreiben. In der 2. Spalte steht nach dem Assemblieren trotzdem FF (H).

Das wichtigste Register ist der Akkumulator (Register A). Er ist der Partner für Datenein- und Ausgabe; er spielt auch eine besondere Rolle bei arithmetischen und logischen Operationen. Deshalb gibt es einige Transferbefehle speziell für den Akku. Sie stehen in Tabelle 4-1 unter LOAD/STORE (= laden/speichern). Durch LDA Adr, „load direct accumulator" (= lade Akku direkt) kann man den Inhalt der

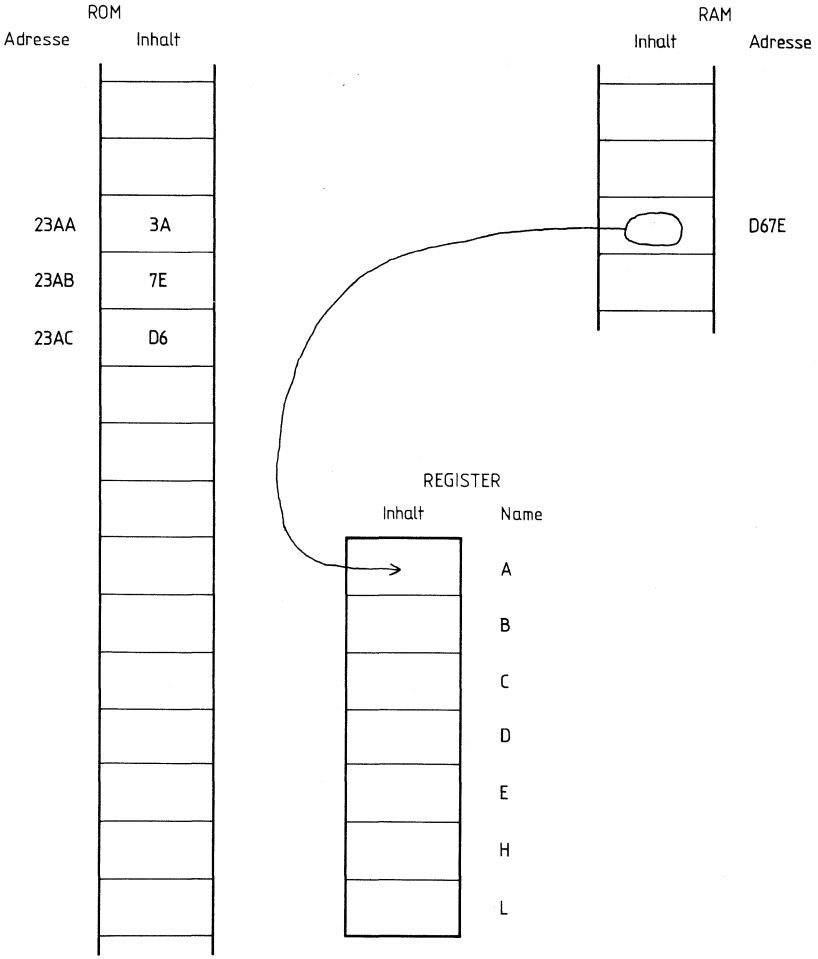


Bild 4.7
Der Befehl LDA Adr mit Adr = D67E

betreffenden Adresse in den Akku bringen; entsprechend bringt man durch STA Adr, „store accumulator" (= speichere Akku) den Akkuinhalt zu dieser Adresse. Dies sind 3-Byte-Befehle, siehe Bild 4.7. Hier steht wieder die untere Hälfte der angegebenen Adresse auf der unteren ROM-Adresse, die obere Hälfte auf der oberen ROM-Adresse.

Eine weitere Bevorzugung des Akkus besteht in den Befehlen LDAX und STAX. Die anderen Register können (wie auch der Akku) in Kontakt treten mit dem Speicherplatz M, „dessen Adresse im Registerpaar HL steht". Für den Akku gibt es entsprechende Transfers auch zu und von solchen Speicherplätzen, deren Adresse im Registerpaar BC oder DE steht:
Durch LDAX B wird z. B. der Inhalt desjenigen Speicherplatzes in den Akku gebracht, dessen Adresse im Registerpaar BC steht; dies ist eine Parallele zu MOV A, M.
Umgekehrt entspricht STAX B dem Befehl MOV M, A, siehe Bild 4.8.

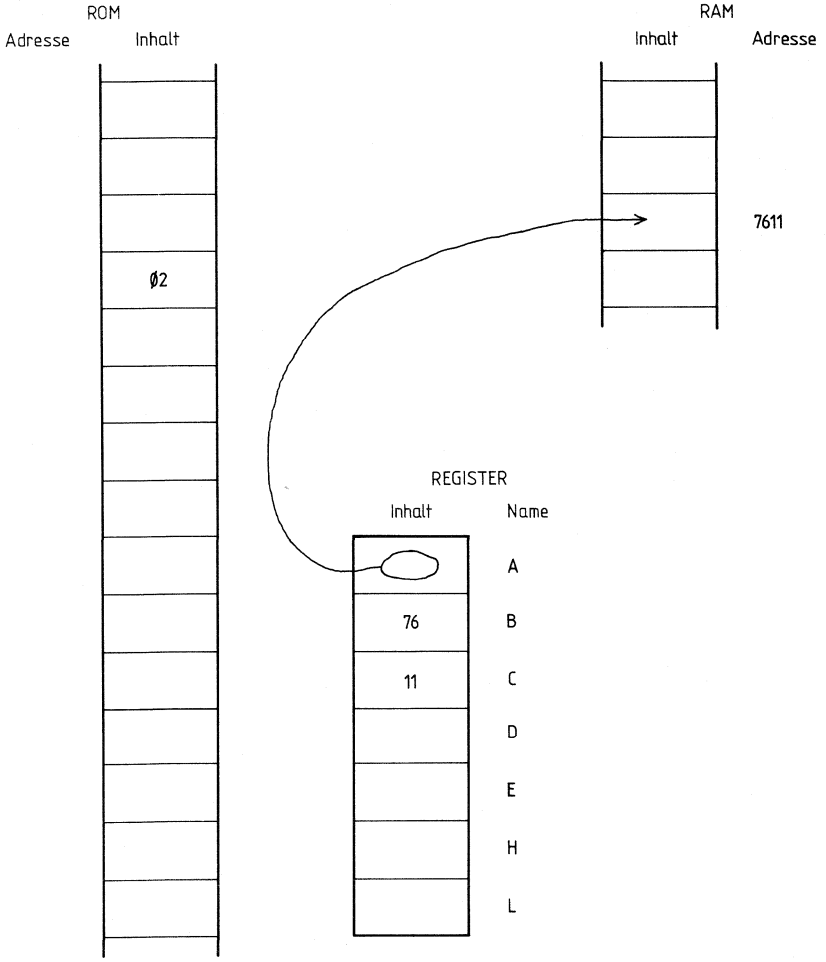


Bild 4.8
Der Befehl STAX B bei dem angegebenen Inhalt von Reg.-Paar BC

Beispiele: Der Inhalt des Speicherplatzes 564D H kommt in den Akku durch

```
105F    3A4D56          LDA     564DH
```

Dies ist ein 3-Byte-Befehl. Wenn das Registerpaar BC schon die Adresse 564D H enthält, dann genügt 1 Byte:

```
105F    0A              LDAX    B
```

Stünde dagegen die Adresse im Registerpaar HL, so hieße es

```
105F    7E              MOV     A,M
```

Der Name des Registerpaares HL enthält H für „high" (= hoch) und L für „low" (= niedrig); damit ist angedeutet, daß eine Adresse in HL mit der höherwertigen Hälfte im Register H gespeichert wird. Entsprechend gehört im Registerpaar BC die höherwertige Hälfte nach B, im Registerpaar DE nach D.

Zum Schluß eine Bemerkung über die Ein-/Ausgabestellen. Am Ende des 2. Kapitels wurde bemerkt, daß man E/As auch ausnahmsweise als Speicherplätze behandeln kann. Dem kann hier hinzugefügt werden: In einem solchen Fall werden die Daten nicht durch den Befehl IN hereingeholt, sondern z. B. durch LDA Adr oder MOV r, M oder LDAX B oder LDAX D.
Ausgegeben werden sie nicht durch OUT, sondern z. B. durch STA Adr oder MOV M, r oder STAX B oder STAX D.
Ein Vorteil dieses Verfahrens besteht darin, daß man nicht nur den Akku, sondern jedes Register als Ursprungs- oder Bestimmungsort der Daten benutzen kann. Ein weiterer Vorteil besteht darin, daß man mit den Befehlen LHLD und SHLD (Abschnitt 4.4) auch 16 Bit-Daten eingeben oder ausgeben kann.
Ein Nachteil kann andererseits dadurch auftreten, daß man zur Angabe einer Adresse 1 Byte mehr benötigt als zur Angabe einer Kanalnummer.

4.3. Adressierungsarten

In Abschnitt 4.2 sind bereits alle Adressierungsarten aufgetreten, die dem Programmierer zur Verfügung stehen. Als Adressierung bezeichnet man das Auffinden eines Operanden, d. i. eines Datenwortes, auf das der Befehl angewendet werden soll. Bei Transferbefehlen ist Operand das Datenwort, das transferiert wird. Ebenfalls kennt man den Begriff des Operanden bei arithmetischen und logischen Operationen; z. B. ist das eine Zahl, die (zum Akkuinhalt) addiert werden soll.

Der Operand kann durch unterschiedliche Methoden aufgefunden werden. Es gibt

- Die unmittelbare Adressierung; z. B. MVI D, D8, siehe Bild 4.3.
- Die direkte Speicheradressierung; z. B. LDA Adr, Bild 4.7.
- Die direkte Registeradressierung; z. B. MOV E, B; Bild 4.5.
- Die indirekte Registeradressierung; z. B. MOV B, M; Bild 4.6 (eigentlich „per" Register).

Die unmittelbare Adressierung wird verwendet bei Konstanten, die im ROM stehen können (siehe Abschnitt 1.3).
Die beiden Arten der direkten Adressierung sind flexibler: Es wird nur die RAM-Adresse oder das Register in den ROM geschrieben, wo die (variablen) Operanden zu finden sind. Bei Abläufen eines bestimmten Programmes mit unterschiedlichen Eingabedaten können sich auf diesen Plätzen ebenfalls unterschiedliche Operanden befinden.

Bei der indirekten Adressierung wird sogar offengehalten, unter welcher Adresse man zu suchen hat. Wozu ist dies nützlich? Es kann z. B. vorkommen, daß man nacheinander viele Adressen aufzusuchen hat, die aufeinander folgen oder wenigstens gleichen Abstand voneinander haben. Dann braucht man nur die Anfangsadresse in ein Registerpaar zu laden und danach von Fall zu Fall zu vergrößern oder verkleinern. Der volle Nutzen dieses Verfahrens kann erst nach Einführung der Programmschleife in Kapitel 6 erkannt werden, siehe das 3. Beispiel in Kapitel 7.

4.4. Transferbefehle für 16 Bit

Zurück zur Tabelle 4-1. Es kommt oft vor, daß man 16-Bit-Wörter als Operanden hat. Das ist z. B. der Fall beim Speichern einer Adresse. Es kann aber auch bei Berechnungen eine Zahl zu groß sein, um durch 8 Bit ausgedrückt zu werden.

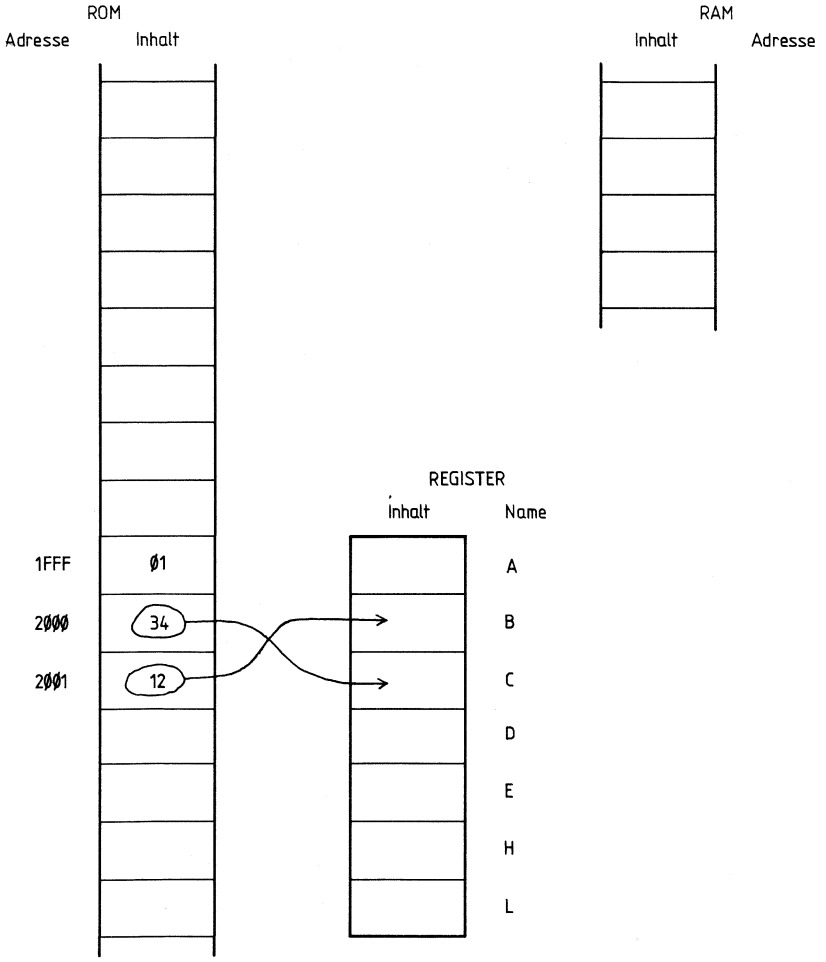


Bild 4.9
Der Befehl LXI B, D16 mit D16 = 1234

In solchen Fällen kann man im Prinzip die Behandlung der Daten trotzdem Byteweise vornehmen. Für häufig auftretende Fälle gibt es aber einige Befehle für 2-Byte-Operanden, die das Programmieren sehr vereinfachen, und außerdem Speicherplatz (ROM) sparen.

So entspricht dem 1-Byte-Transfer MVI r, D8 der 2-Byte-Transfer LXI rp, D16 mit „rp" für Registerpaar. „LI" heißt „load immediate" (= lade unmittelbar); der Buchstabe X steht manchmal (aber nicht konsequent) für „Registerpaar".
Das gemeinte Registerpaar wird durch den Operationscode ausgedrückt; die Konstante D16 wird durch 2 weitere Bytes genannt. Als Registerpaar kommt außer BC, DE und HL auch in Frage „SP", das heißt „stack pointer" (= Stapelzeiger), was erst im 6. Kapitel erklärt werden kann. Bild 4.9 zeigt das Wesentliche.

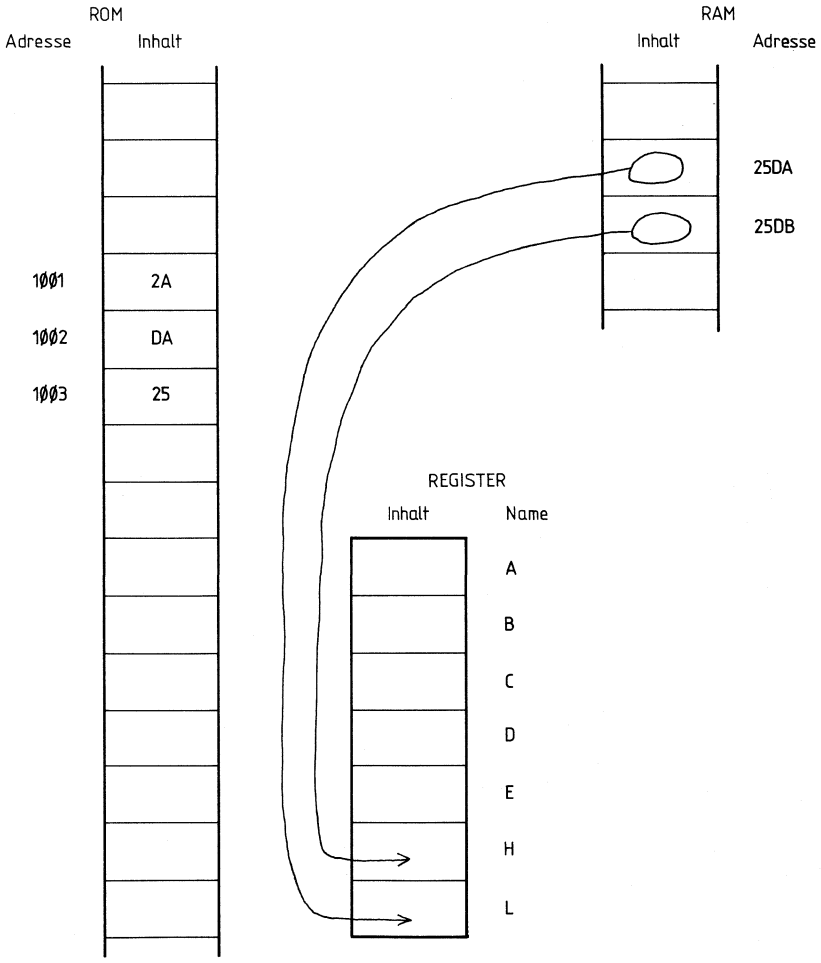


Bild 4.10
Der Befehl LHLD Adr mit Adr = 25DA

Eine Anwendung: Man könnte den Inhalt des Speicherplatzes 9ØEA H in den Akku holen durch

```
Ø5FF    3AEA9Ø          LDA     9ØEAH
```

Wenn aber diese Aktion sehr oft verlangt wird und das Registerpaar BC zur Verfügung steht, ist es besser, wie folgt zu verfahren:

```
Ø5FF    Ø1EA9Ø          LXI     B,9ØEAH
Ø6Ø2    ØA              LDAX    B
```

mit wiederholter Anwendung des letzten Befehles.

Eine besondere Rolle des Registerpaares HL ergibt sich schon aus der Existenz des Speicherplatzes M. Eine weitere Auszeichnung folgt aus dem Befehl LHLD Adr, „load HL direct" (= lade HL direkt), siehe Bild 4.10. Das Registerpaar HL wird mit dem Inhalt der Speicherplätze Adr und Adr + 1 geladen; dies ist bei keinem anderen Registerpaar möglich.

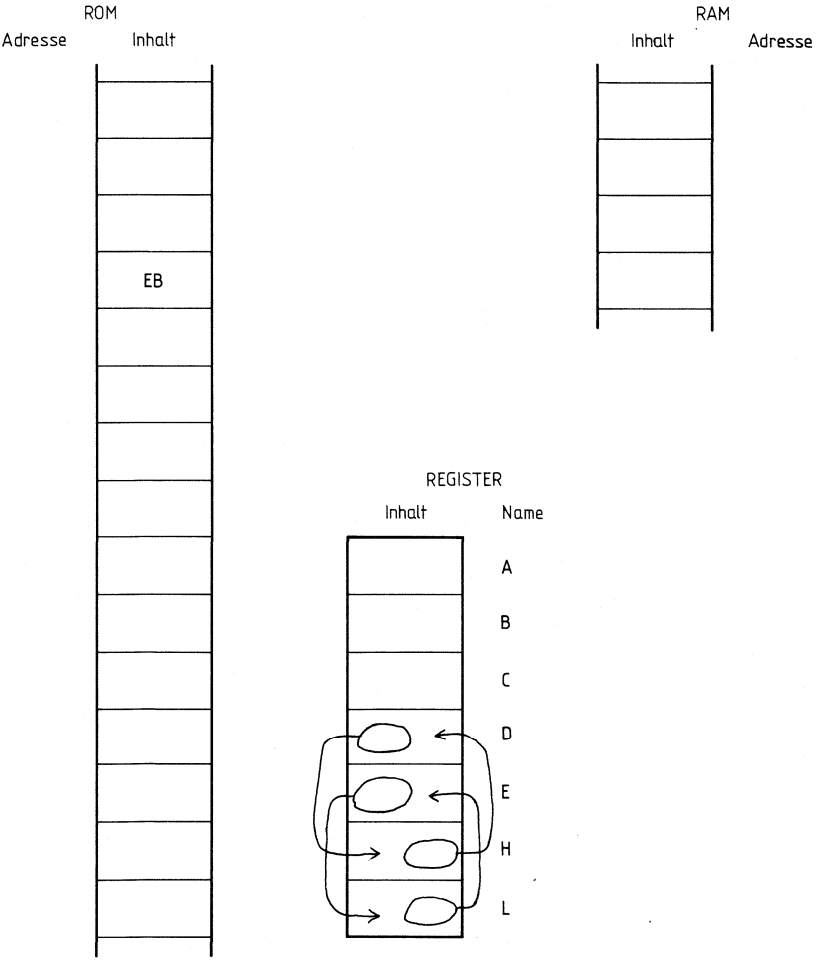


Bild 4.11
Der Befehl XCHG

Der umgekehrte Vorgang wird durch SHLD Adr veranlaßt, „store HL direct".
Der Befehl XCHG, siehe Bild 4.11, ermöglicht es aber auch dem Registerpaar DE, auf dem Umweg über HL mit dem Speicher in Kontakt zu treten: Die Inhalte der Registerpaare HL und DE werden vertauscht. Die mnemonische Abkürzung steht für „exchange" (= tausche aus).

Beispiel: Der Inhalt der Speicherplätze mit den Adressen 4AAAH und 4AAB H soll in das Registerpaar DE gebracht werden. Eine Möglichkeit mit Hilfe von 1-Byte-Befehlen wäre

```
Ø61Ø    3AAA4A          LDA     4AAAH
Ø613    5F              MOV     E,A
Ø614    3AAB4A          LDA     4AABH
Ø617    57              MOV     D,A
```

Wenn aber das Registerpaar HL frei ist, geht es besser so:

```
Ø61Ø    2AAA4A          LHLD    4AAAH
Ø613    EB              XCHG
```

Man spart 4 Byte im ROM und auch entsprechende Arbeitszeit des Programmes (Takte).

Die restlichen 2-Byte-Transferbefehle werden erst in Kapitel 6 besprochen.

4.5. Arithmetische Operationen mit 8 Bit-Zahlen

Bei den Transferbefehlen spielt es keine Rolle, welche Bedeutung die transferierten Daten haben. Bei arithmetischen Operationen werden die Operanden dagegen stets als binäre Zahlen aufgefaßt. Es hätte keinen Sinn, etwa Operationscodes zusammenzuzählen.
Einzelheiten über die vorkommenden Zahlensysteme und über die binäre Addition und Subtraktion bringt Kapitel 9; hier wird Kenntnis dieses Kapitels vorausgesetzt.

Das Rechenwerk des SAB 8080 kann Binärzahlen addieren und subtrahieren. Andere Rechenarten müssen vom Programm zusammengesetzt werden; es gibt dafür auch Unterprogramme in Bibliotheken. Ein Beispiel für Multiplikation wird in Kapitel 7 vorgeführt.

Bei Additionen und Subtraktionen von 1-Byte-Zahlen steht immer ein Operand im Akku. Der andere Operand wird gefunden durch unmittelbare Adressierung, durch direkte Registeradressierung oder durch indirekte Registeradressierung. Eine direkte Speicheradressierung kommt nicht vor. Nach der Berechnung steht das Resultat im Akku.

Spezielle Additionen oder Subtraktionen sind das Inkrementieren oder Dekrementieren, d.h. Addieren oder Subtrahieren einer Eins. Hierzu gibt es besondere Befehle. Bei ihrer Ausführung wird i.a. der Akku nicht benutzt; die zu inkrementierende oder dekrementierende Zahl wird aus dem Speicherplatz M oder einem Register geholt und nach der Operation dorthin zurückgebracht.

Vor der Besprechung der Befehle muß noch der Begriff des Übertragsbits eingeführt werden; dies geschieht im folgenden Abschnitt 4.5.1.

4.5.1. Das Übertragsbit

Bei der Addition von 2 Dezimalzahlen kann das Resultat mehr Ziffern haben als die beiden addierten Zahlen. Z. B. ergibt sich

```
  823
+ 201
= 1024
```

Dasselbe ist möglich bei der Addition binärer Zahlen, z. B.

```
  11000101 B (= 197 D)
+ 01011001 B (=  89 D)
= 100011110 B (= 286 D)
```

Beim Rechnen auf dem Papier schreibt man einfach eine Ziffer mehr hin. Wenn dagegen der Mikrocomputer 1-Byte-Zahlen addiert, ist die Anzahl der Binärziffern auf 8 begrenzt. Im obigen Beispiel würde das Ergebnis 00011110 B = 30 D erscheinen.

Der Mikroprozessor sollte dann die Tatsache festhalten, daß ein „Überlauf" entstanden ist und eine „Bereichsüberschreitung" stattgefunden hat. Hierzu dient das Übertragsbit.
Das Übertragsbit ist der Inhalt eines 1-Bit-Registers im Mikroprozessor. Sein Wert kann 1 oder 0 betragen. Er wird auf 1 gesetzt, wenn bei einer Addition ein Übertrag entsteht; er wird auf 0 gesetzt, wenn kein Übertrag entsteht. Durch Inkrementierungs- oder Dekrementierungsbefehle wird (beim SAB 8080) das Übertragsbit nicht beeinflußt.

Durch das Übertragsbit zeigt der SAB 8080 auch den etwaigen negativen Übertrag bei einer Subtraktion an. Er befolgt dabei die Regel:

> Das Übertragsbit wird gesetzt (erhält oder behält den Wert 1), wenn eine Addition (nicht Inkrementierung) einen Übertrag ergab oder wenn eine Subtraktion (nicht Dekrementierung) einen negativen Übertrag ergab; sonst wird es zurückgesetzt (erhält oder behält den Wert 0).

In der Tabelle 4-1 werden „Zustandsbits" erwähnt. Das sind fünf 1-Bit-Register, zu denen das Übertragsbit gehört. Die anderen werden erst in Kapitel 6 bzw. Abschnitt 4.7.2 benötigt.

4.5.2. Befehle für arithmetische Operationen mit 8 Bit-Zahlen

In Tabelle 4-1 erscheinen 2 Arten von 8-Bit-Additionsbefehlen und auch 2 Arten von 8-Bit-Subtraktionsbefehlen. Der Unterschied ist, daß bei der einen Addition (bzw. Subtraktion) der Wert des Übertragsbits mit eingeht, bei der anderen nicht.
Angenommen, es sollen zwei 24-Bit-Zahlen addiert werden:

```
  10000000 00010100 11000101 B (=  8393925 D)
+ 11001101 11100111 01011001 B (= 13494105 D)
= 1 01001101 11111100 00011110 B (= 21888030 D)
```

Der SAB 8080 kann keine 24 Bit-Zahlen direkt addieren. Also werden dreimal 8-Bit-Zahlen addiert (z. B.). Bei der Addition der niederwertigsten 8 Bits darf kein Übertrag aus früheren Operationen berücksichtigt werden; diese haben mit der aktuellen Addition nichts zu tun. Daher benutzt man hier einen Additionsbefehl o h n e Berücksichtigung des Übertragsbits. Für das Addieren der mittleren 8 Bits spielt es aber sehr

wohl eine Rolle, ob bei den niederwertigsten 8 Bits ein Übertrag auftrat; hierzu benutzt man einen Additionsbefehl mit Berücksichtigung des Übertragsbits. Dessen Wert (1 oder 0) wird in die unterste Stelle der mittleren 8 Bits zusätzlich hineinaddiert. Bild 4.12 zeigt den Vorgang; er wiederholt sich beim höchstwertigen Byte.

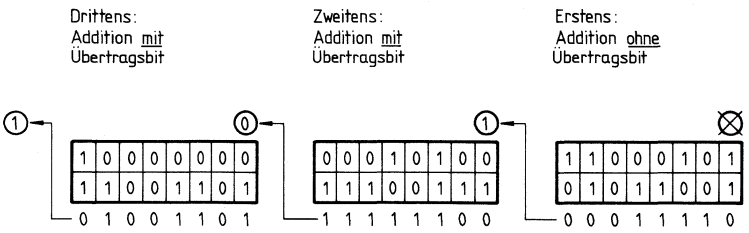


Bild 4.12
Addition zweier 24-Bit-Zahlen

Konkret bedeutet ADD r und ADD M, daß der Inhalt des betreffenden Registers oder Speicherplatzes zum Akkuinhalt addiert werden soll ohne Berücksichtigung des Übertragsbits. Speziell ADD A hat zur Folge, daß der Akkuinhalt verdoppelt wird. Dagegen ordnet ADC r und ADC M die entsprechende Operation mit Berücksichtigung des Übertragsbit an. Das „C" steht für „carry" (= Übertrag). Zum Addieren einer 8 Bit-Konstanten dient ADI D8 („add immediate") ohne Übertragsbit, ACI D8 mit Übertragsbit.

In Tabelle 4-1 stehen neben den Additionsbefehlen die ganz entsprechenden Subtraktionsbefehle; hier kennzeichnet der Buchstabe „B" für „borrow" (= Geborgtes) den Fall mit Berücksichtigung des (hier negativen) Übertrages.

Die restlichen Befehle sind INR r und INR M zum Inkrementieren sowie DCR r und DCR M zum Dekrementieren von 8-Bit-Zahlen.

Es folgen einige Beispiele.
Zum Inhalt des Speicherplatzes 1ØØØ H wird 25 D addiert:

```
8123    3AØØ1Ø          LDA    1ØØØH
8126    C619             ADI    19H (z.B.)
```

Es folgt eine Addition von zwei 16-Bit-Zahlen. Dazu werden hier bewußt nicht die Befehle verwendet, die für 16-Bit-Zahlen existieren (Abschnitt 4.6). Der Vergleich wird zeigen, wie wertvoll eine reichhaltige Befehlsliste ist.
Die Zahlen seien 1234 H und ABCD H. Sie sollen im RAM stehen, wie in Bild 4.13a gezeigt.

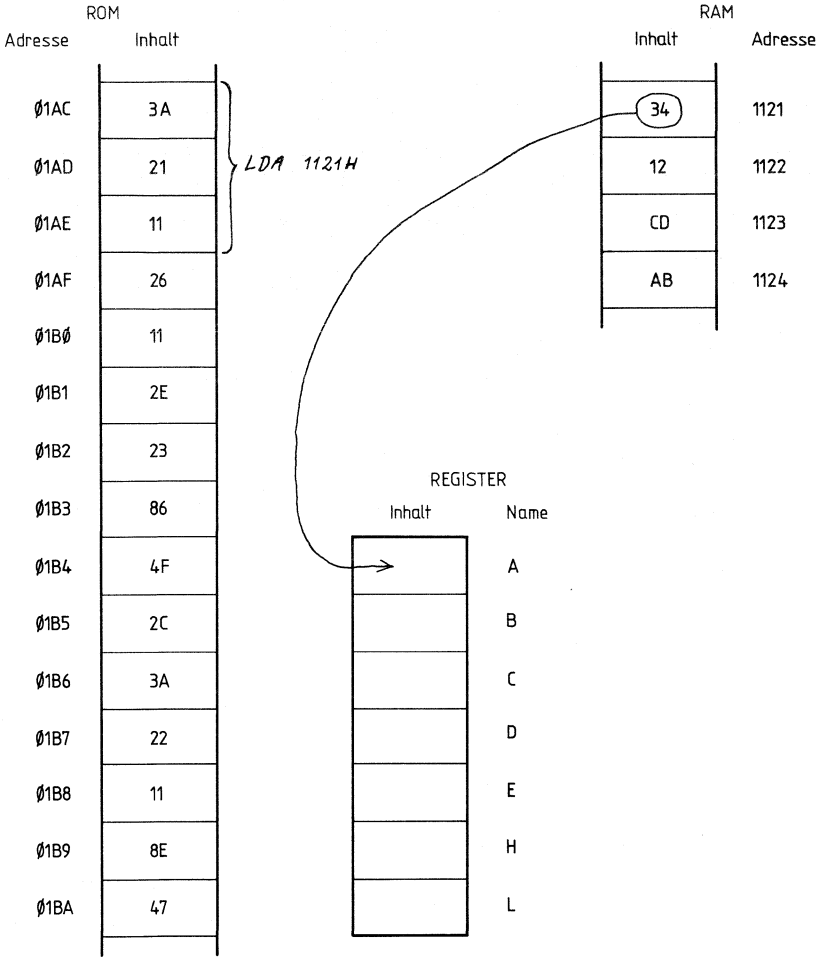


Bild 4.13
Addition zweier 16 Bit-Zahlen
a) Niederwertiges Byte der 1. Zahl in den Akku

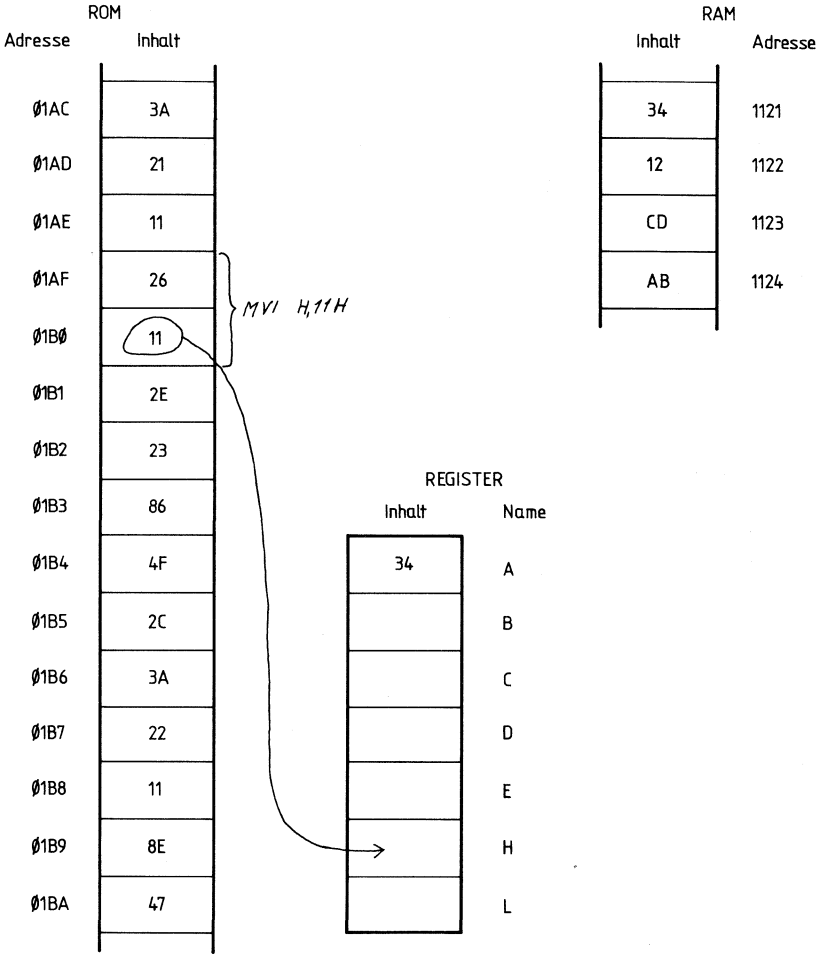


Bild 4.13
Addition zweier 16 Bit-Zahlen
b) Obere Hälfte der Adresse 1123 nach Reg. H

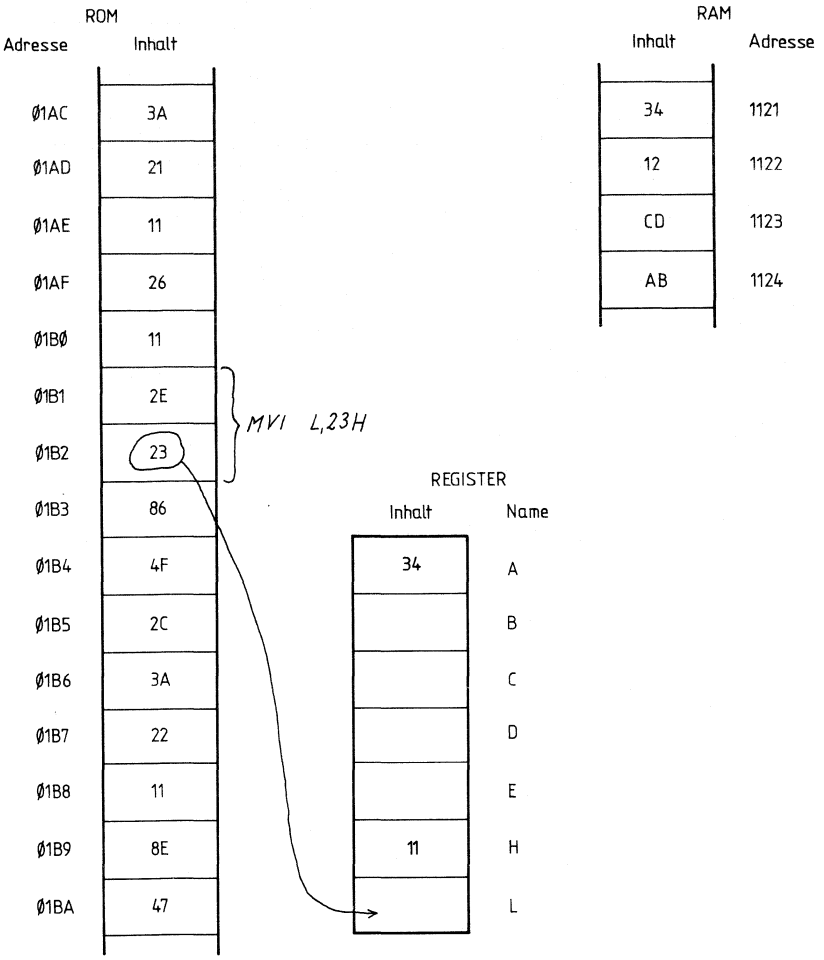


Bild 4.13
Addition zweier 16 Bit-Zahlen
c) Niedere Hälfte der Adresse 1123 nach Reg. L

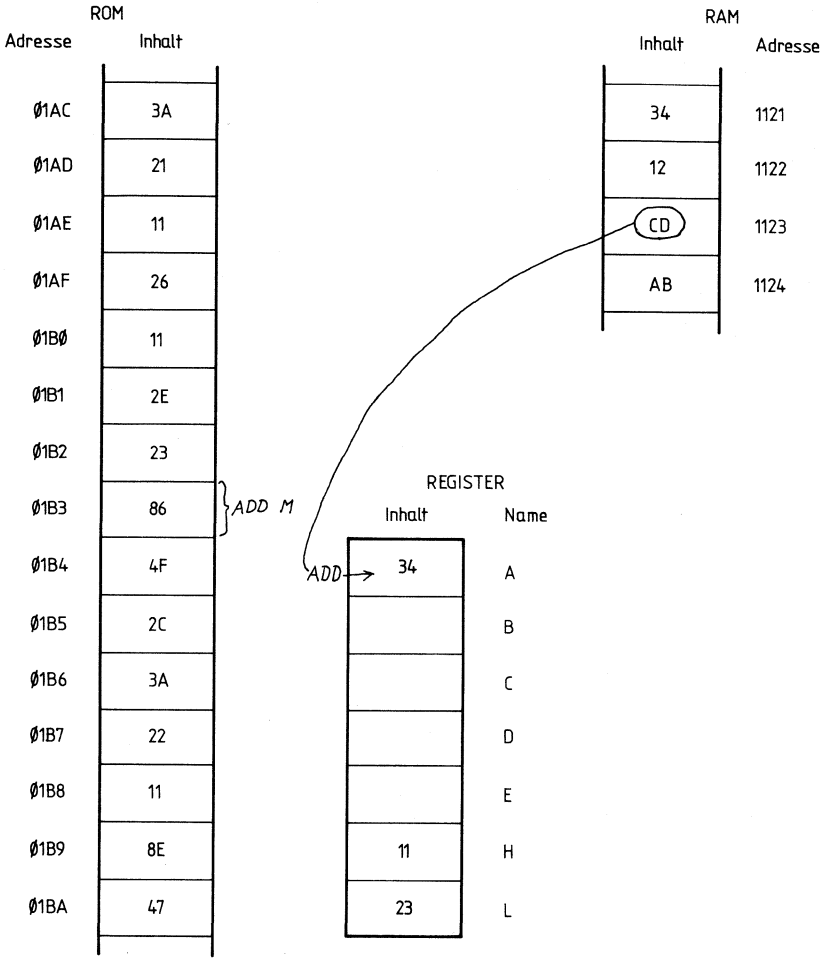


Bild 4.13
Addition zweier 16 Bit-Zahlen
d) Addition der niederwertigen Hälften der Zahlen

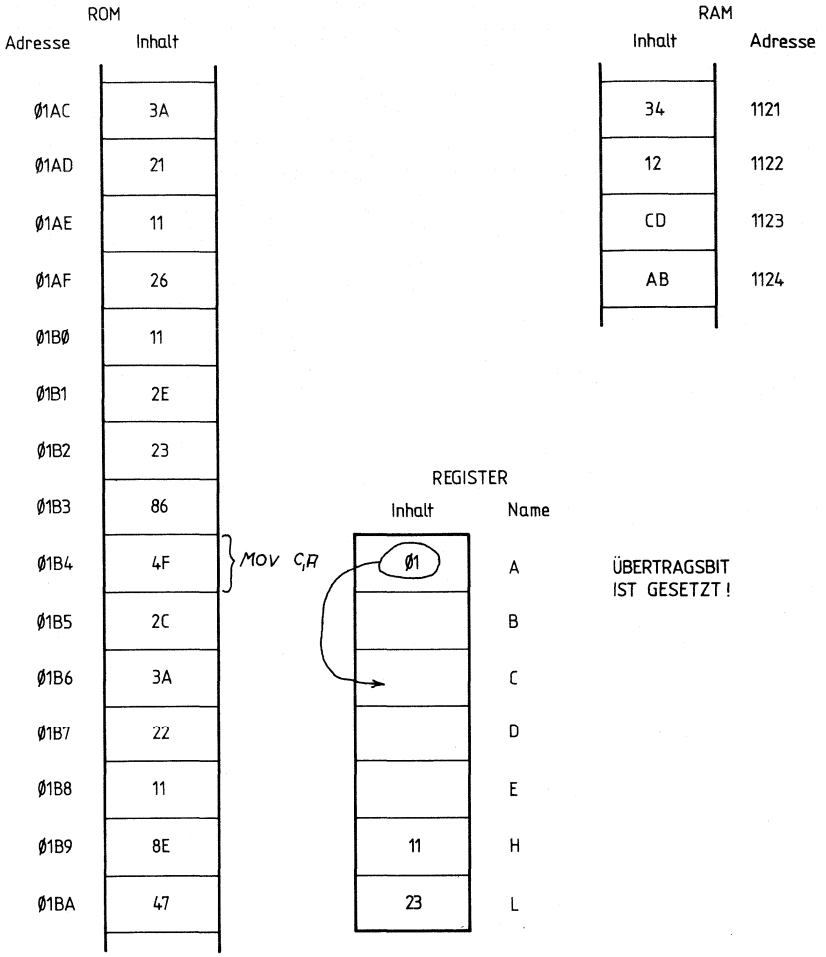


Bild 4.13
Addition zweier 16 Bit-Zahlen
e) Summe der niederwertigen Hälften in Reg. C

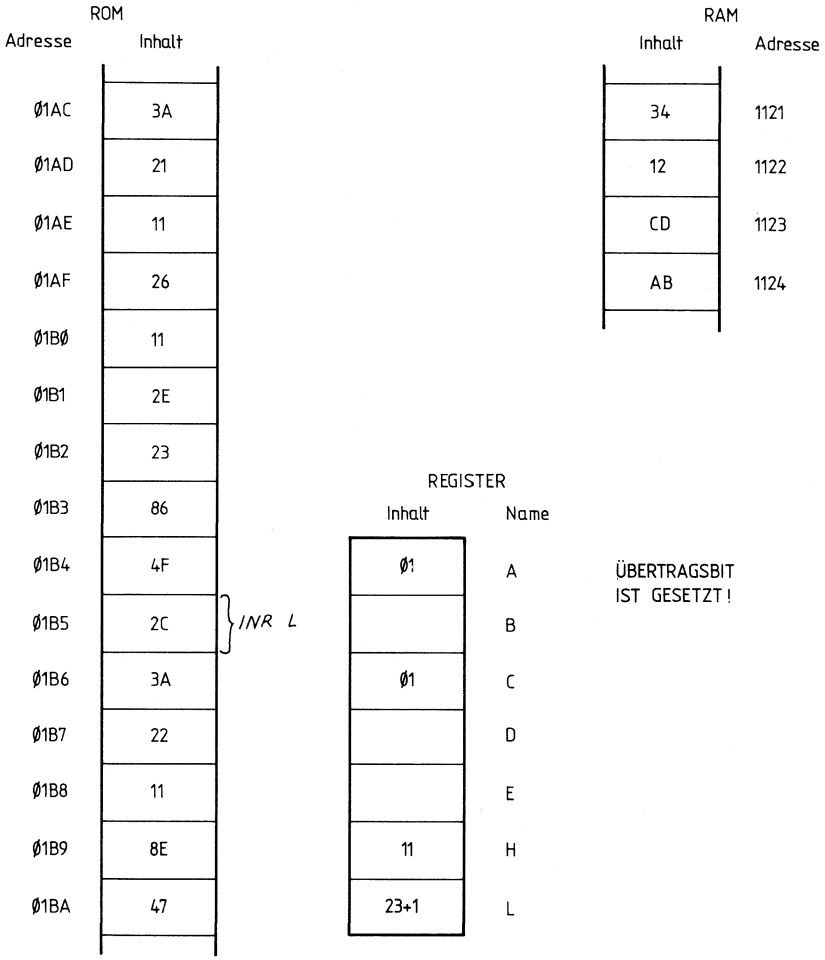


Bild 4.13
Addition zweier 16 Bit-Zahlen
f) Adresse in Reg.-Paar HL wird inkrementiert

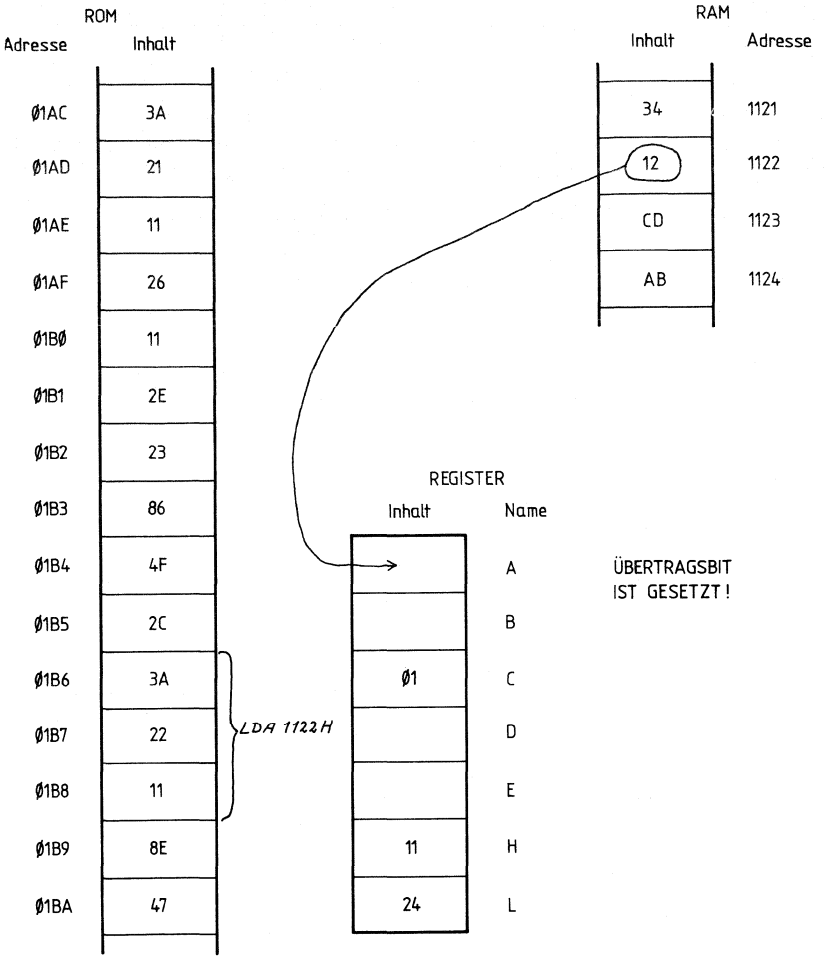


Bild 4.13
Addition zweier 16 Bit-Zahlen
g) Höherwertiges Byte der 1. Zahl in den Akku

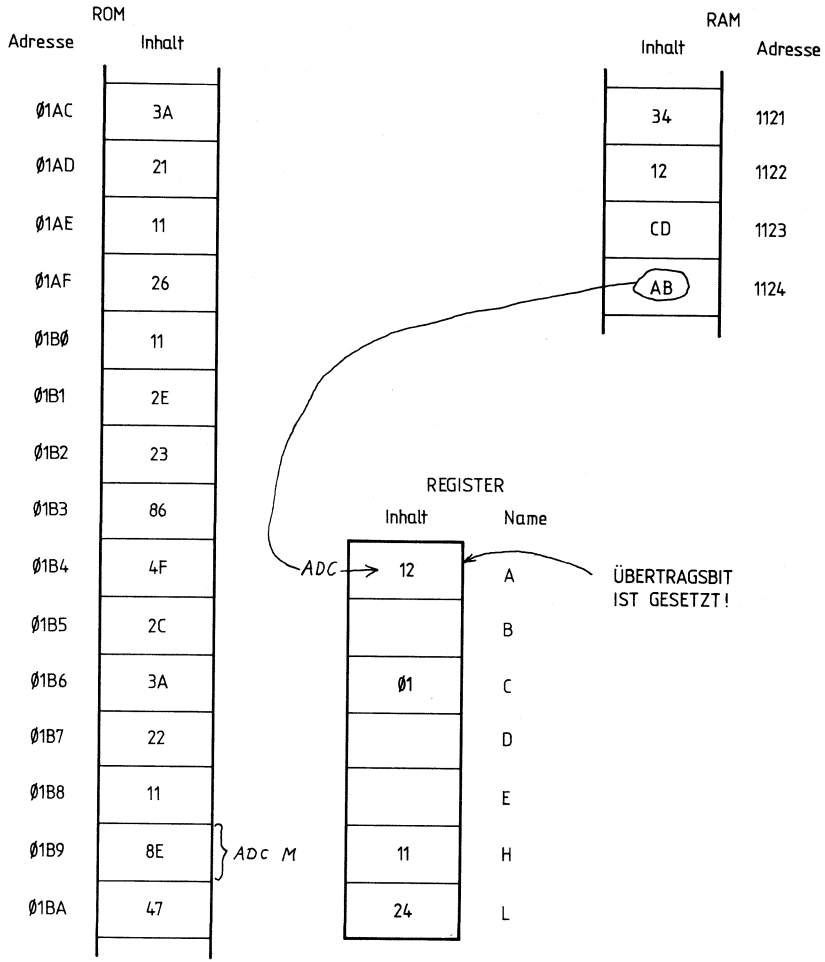


Bild 4.13
Addition zweier 16 Bit-Zahlen
h) Addition der höherwertigen Hälften der Zahlen

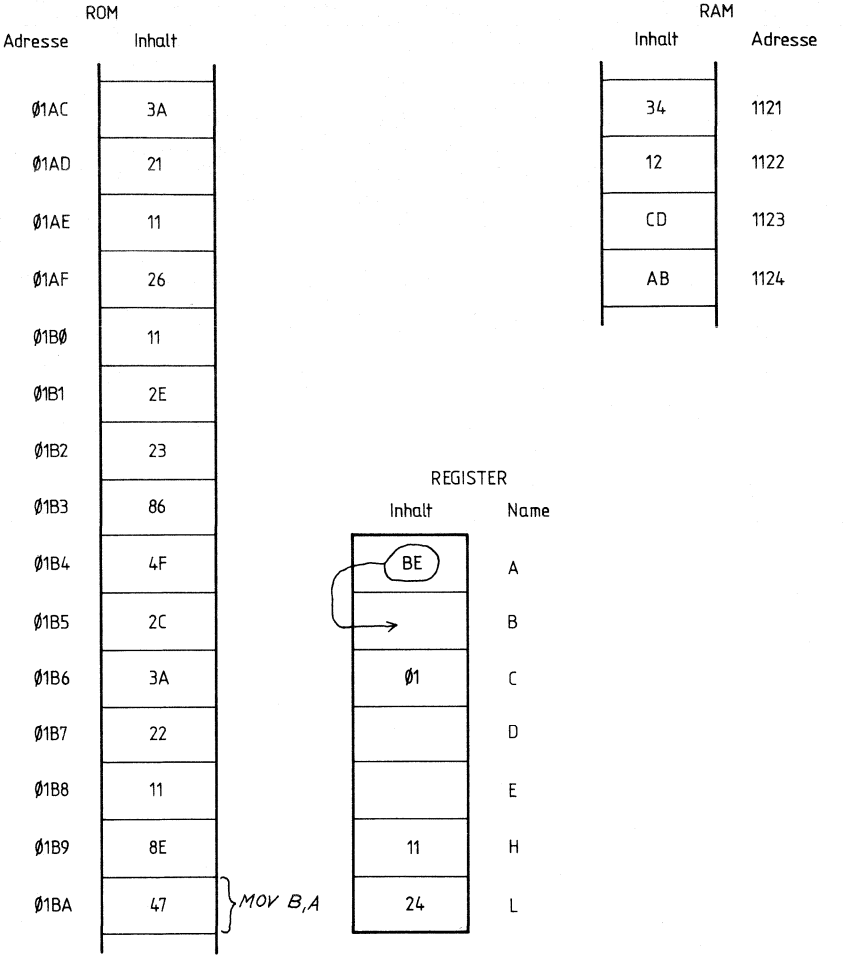


Bild 4.13
Addition zweier 16 Bit-Zahlen
i) Das Resultat steht im Reg.-Paar BC

Die Einzelheiten der Prozedur sind aus dem folgenden Assembler-Listing und aus der Bilderfolge 4.13a bis 4.13i ersichtlich.

```
Ø1AC    3A2111      LDA     1121H
Ø1AF    2611        MVI     H,11H
Ø1B1    2E23        MVI     L,23H
Ø1B3    86          ADD     M       ;OHNE UEBERTRAGSBIT.
Ø1B4    4F          MOV     C,A
Ø1B5    2C          INR     L
Ø1B6    3A2211      LDA     1122H
Ø1B9    8E          ADC     M       ;MIT UEBERTRAGSBIT.
Ø1BA    47          MOV     B,A     ;RESULTAT IN RPB.
```

Hier ist konsequenterweise auch der 2-Byte-Transfer LXI rp nicht benutzt worden.

Das nächste Beispiel zeigt die Subtraktion zweier 16-Bit-Zahlen. Die Zahlen sollen in den Registerpaaren BC und DE stehen. Der Inhalt von DE soll vom Inhalt von BC abgezogen werden.

```
6412    79          MOV     A,C     ;MINUEND NIED. BYTE.
6413    93          SUB     E       ;SUBTRAKTION NIED. BYTE.
6414    4F          MOV     C,A
6415    78          MOV     A,B     ;MINUEND HOEH. BYTE.
6416    9A          SBB     D       ;SUBTRAKTION HOEH. BYTE.
6417    47          MOV     B,A     ;RESULTAT IN RPB.
```

Hier wurden nur kurze Befehle benötigt, weil die Operanden bereits in Registern vorlagen. „RPB" steht in den beiden letzten Beispielen für „Registerpaar BC".
Beim Befehl SBB wird ein etwaiger negativer Übertrag dadurch berücksichtigt, daß der Wert „1" des Übertragsbits zusätzlich vom niederwertigsten Bit dieser Subtraktion abgezogen wird; das ist also das Bit Nr. 8 der 16-Bit-Zahlen.

4.6. Arithmetische Operationen mit 16 Bit-Zahlen

Der Befehl INX rp bewirkt die Inkrementierung einer 16-Bit-Zahl. Das ist nicht einfach eine Inkrementierung der niederwertigen Hälfte dieser Zahl. So ergibt sich bei Inkrementierung von 00000000 11111111 beispielsweise 00000001 000000000. Hier liefert das niedere Byte beim Inkrementieren einen Übertrag; diese Möglichkeit muß man berücksichtigen, wenn man den einfachen INR r- oder DCR r-Befehl benutzt. Es ist deshalb eine Erleichterung der Arbeit, wenn man durch INX rp und DCX rp den Inhalt von Registerpaaren inkrementieren und dekrementieren kann. Das vierte in Frage kommende Registerpaar ist dasselbe wie bei LXI rp, nämlich SP.

Man kann außerdem Addition (nicht Subtraktion) von 16-Bit-Zahlen direkt durchführen. Der Befehl DAD rp veranlaßt, daß der Inhalt des Registerpaares rp zum Inhalt des Registerpaares HL addiert und dann das Resultat im Registerpaar HL gespeichert wird. Speziell DAD H bedeutet Verdoppelung des Inhaltes von HL. Hier hat das Registerpaar HL eine ähnlich bevorzugte Rolle wie bei 8-Bit-Zahlen der Akkumulator.

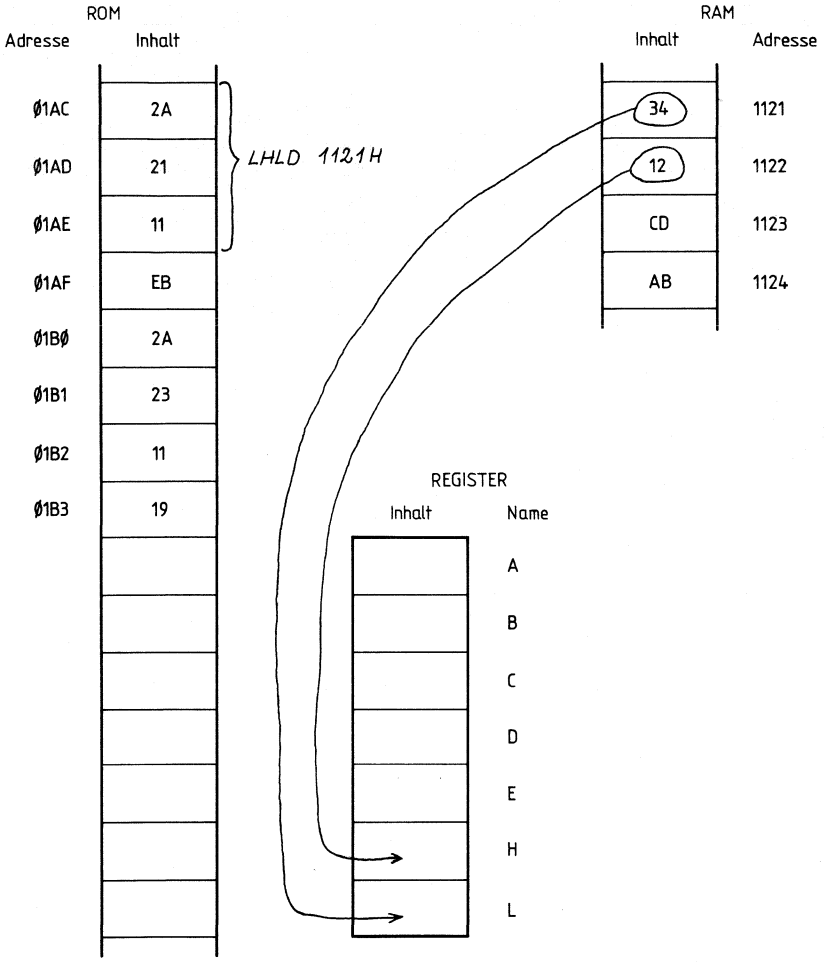


Bild 4.14
Addition zweier 16 Bit-Zahlen mit Befehlen für Registerpaare
a) Erste Zahl in Reg.-Paar HL

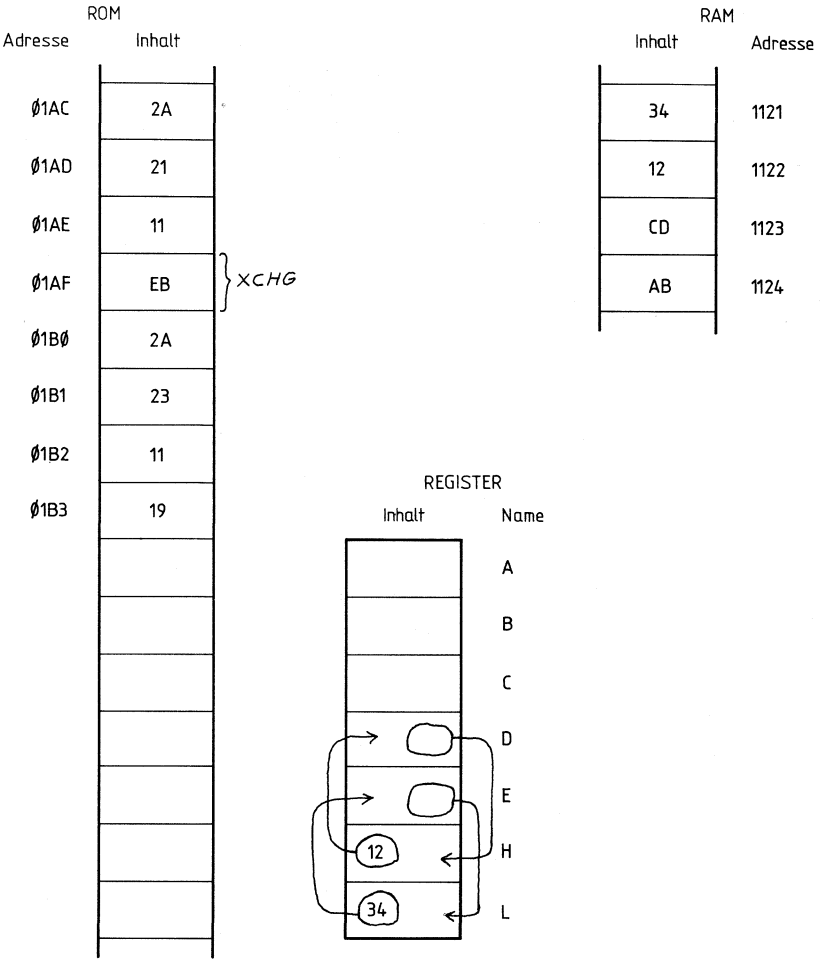


Bild 4.14
Addition zweier 16 Bit-Zahlen mit Befehlen für Registerpaare
b) Erste Zahl in Reg.-Paar DE

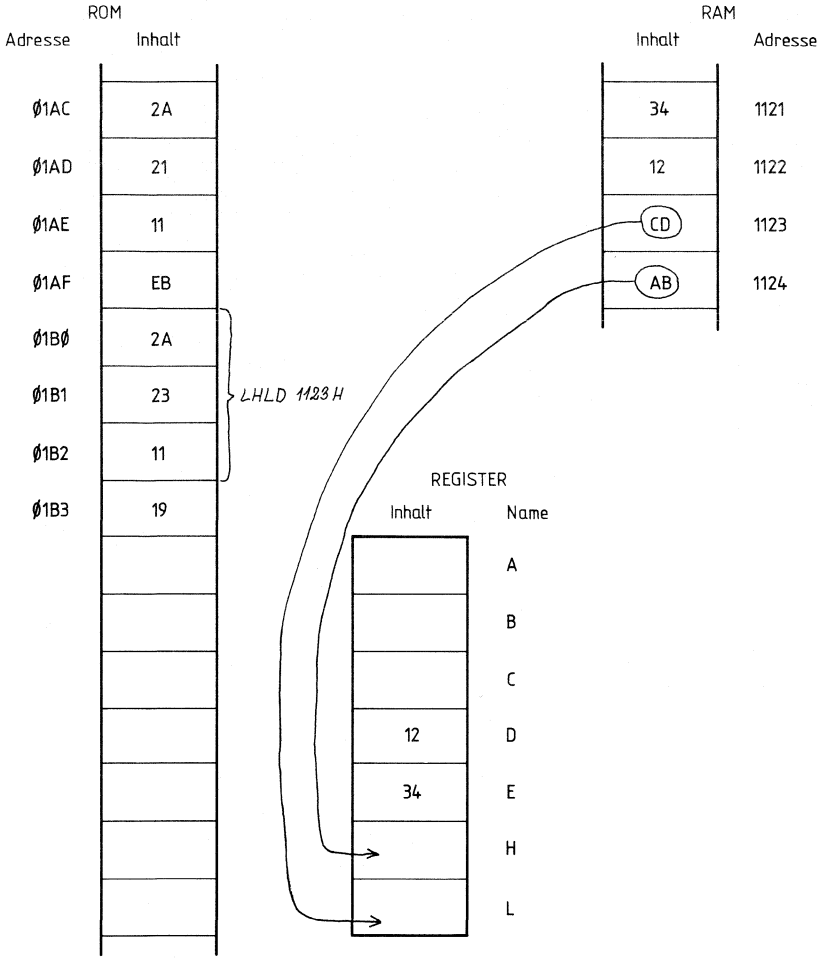


Bild 4.14
Addition zweier 16 Bit-Zahlen mit Befehlen für Registerpaare
c) Zweite Zahl in Reg.-Paar HL

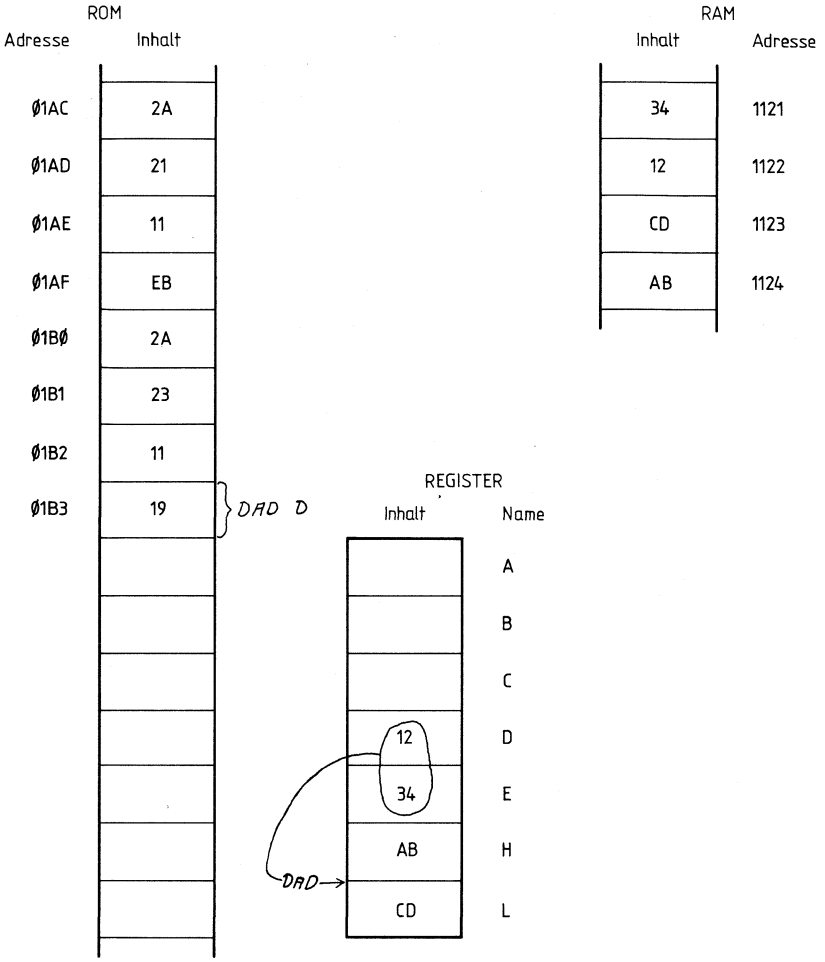


Bild 4.14
Addition zweier 16 Bit-Zahlen mit Befehl für Registerpaare
d) Addition

Das vorletzte Beispiel in Abschnitt 4.5 (Addition zweier 16-Bit-Zahlen) wird nun mit Registerpaarbefehlen verbessert; siehe Bild 4.14.

```
Ø1AC    2A2111          LHLD    1121H
Ø1AF    EB              XCHG                    ;1. SUMMAND IN RPD.
Ø1BØ    2A2311          LHLD    1123H           ;2. SUMMAND IN RPH.
Ø1B3    19              DAD     D               ;RESULTAT IN RPH.
```

Hier sind nur 8 ROM-Plätze verbraucht worden, gegenüber 15 im Fall der 1-Byte-Additionsbefehle. Mit Hilfe von Tabelle 4-3 findet man außerdem: Hier werden zur Ausführung nur 46 Takte verbraucht, gegenüber 75 im Fall der 1-Byte-Additionsbefehle. Hier ist allerdings angenommen, daß es keine Rolle spielt, in welchem Registerpaar das Ergebnis schließlich steht.

Der Inhalt von Registerpaar BC wird zum Inhalt von Registerpaar DE addiert durch

```
1B45    EB              XCHG
1B46    Ø9              DAD     B
1B47    EB              XCHG
```

4.7. Logische Operationen

4.7.1. Logische Verknüpfungen

Diese Überschriften sind geeignet, den Uneingeweihten mit Sorge zu erfüllen. Dazu besteht aber kein Anlaß. Die Durchführung logischer Operationen ist eine wesentlich einfachere Aufgabe, als gut Skat zu spielen.

Mit einer einfachen Fragestellung wird begonnen, nämlich der Frage, ob in einem bestimmten Büroraum geraucht werden darf. Es soll die folgende Regelung bestehen: Wenn jede der beiden anwesenden Personen einverstanden ist, darf geraucht werden; wenn auch nur eine Person dagegen ist, darf nicht geraucht werden.
Eine logische Variable (= veränderliche Größe) mit Namen x soll den Wert 1 haben, wenn der Angestellte Xaver nichts gegen das Rauchen hat; sonst soll x den Wert 0 haben. Eine weitere logische Variable y soll den Wert 1 haben, wenn die Angestellte Yvonne nichts gegen das Rauchen hat, sonst 0.
Der Wert einer dritten Variablen z steht auf einem Zettel an der Wand; er ist 1, wenn geraucht werden darf, sonst 0. Xaver und Yvonne sind frei in ihrer Meinungsbildung; daher heißen x und y unabhängige Variable. Was auf dem Zettel steht, unterliegt aber den Spielregeln und den Meinungen von Xaver und Yvonne; daher heißt z eine abhängige Variable.

Die Begriffe „unabhängige Variable" und „abhängige Variable" sind auch bei mathematischen (z.B. arithmetischen) Operationen geläufig. Sie werden hier benutzt, um dem Leser etwas ggf. Neues präziser erklären zu können.

Der Zusammenhang zwischen x, y und z ist der folgende.
Wenn x = 1 und y = 1, dann ist z = 1.
Wenn x = 1 und y = 0, dann ist z = 0.
Wenn x = 0 und y = 1, dann ist z = 0.
Wenn x = 0 und y = 0, dann ist z = 0.

Dieser Zusammenhang wird kürzer als „Wahrheitstafel" dargestellt:

x	y	z
1	1	1
1	0	0
0	1	0
0	0	0

UND-Operation

z wird ermittelt durch Anwendung einer logischen Operation auf x und y. Diese Operation besteht darin, in der Wahrheitstafel nachzusehen, welcher Wert von z zu den beiden vorhandenen Werten x und y gehört. Das ist alles!
Eine Operation mit dieser Wahrheitstafel heißt die UND-Operation (engl.: and), was nichts mit Addition zu tun hat.

Das Beispiel wirkt sehr simpel. Ebenso simpel sind aber auch die vielen Einzelschritte, aus denen sich eine komplizierte mathematische Berechnung im Computer zusammensetzt.

Die Situation im Büro kann auch auf andere Weise beschrieben werden. Wenn mindestens einer der Anwesenden gegen das Rauchen protestiert, ist es verboten.
Jetzt soll x oder y den Wert 1 dann bekommen, wenn die betreffende Person protestiert; z bekommt den Wert 1, wenn das Verbot besteht. Die Operation, die jetzt den Wert z aus den Werten x und y ermittelt, heißt ODER-Operation (engl.: or).

x	y	z
1	1	1
1	0	1
0	1	1
0	0	0

ODER-Operation

Schließlich ist noch folgende Situation denkbar. Die Angestellten im Büro sind zerstritten, wenn ihre Meinung über das Rauchen unterschiedlich ist. Im Fall von Streit sei z = 1.

Für den z-Wert ist es belanglos, wann x und y den Wert 1 haben; nehmen wir an, x = 1 bzw. y = 1 heiße „mit Rauchen einverstanden". Hier gilt die Wahrheitstafel der Exklusiv-ODER-Operation (X-or).

x	y	z
1	1	0
1	0	1
0	1	1
0	0	0

Exklusiv-ODER-Operation

Die logischen Operationen UND, ODER und Exklusiv-ODER werden auch als logische Verknüpfungen bezeichnet.
Sie werden jeweils auf 2 Byte angewandt; wie bei Addition und Subtraktion steht eines im Akku und eines an anderer Stelle, das Resultat steht danach im Akku. Die Operation wird Bit für Bit vorgenommen: Erst das Bit Nr. 0 vom 1. Byte mit dem Bit Nr. 0 vom 2. Byte, dies ergibt das Bit Nr. 0 des Resultats; danach paarweise weiter bis zu den Bits Nr. 7. Zur Numerierung der Bits siehe Abschnitt 2.3.

Im Unterschied zu arithmetischen Operationen hat der Begriff des Übertrages hier keinen Sinn.

Beispiel:

```
          01011100   (= x)
UND       11000111   (= y)
ergibt    01000100   (= z).
```

Solche Aufgaben wurden bisher weitgehend durch festverdrahtete Logikschaltungen gelöst. Jede logische Operation an zwei 1-Bit-Veränderlichen erfordert ein „Gatter", das aus mehreren Transistoren besteht. Ein Befehl des Programmes wie im obigen Beispiel ersetzt also die einmalige Tätigkeit von 8 Gattern.
Eine Anwendung kann z. B. wie folgt aussehen. Ein Prozeßrechner speichert zu einem Zeitpunkt 0 den Zustand von 8 Schaltern und danach noch einmal zu einem Zeitpunkt 1. Jedes Bit sei dann = 1, wenn der zugehörige Schalter geschlossen ist.
Braucht nun das Programm die Angabe, welche Schalter in der Zwischenzeit ihren Zustand geändert haben, so kann es die beiden Bytes exklusiv-ODERieren:

```
          01011100   (= x)
exklusiv-ODER 11000111   (= y)
ergibt    10011011   (= z)
```

Die Schalter 0, 1, 3, 4 und 7 haben ihren Zustand geändert.

Für die Befehle zur Durchführung logischer Verknüpfungen gibt es dieselben Adressierungsarten wie bei den arithmetischen Operationen.
ANI D8 UNDiert das Datenwort D8 unmittelbar mit dem Akkuinhalt. ANA r und ANA M UNDieren das betreffende Byte mit dem Akkuinhalt. ANA A läßt demnach den Akkuinhalt unverändert, weil 1 UND 1 wieder 1 gibt und 0 UND 0 wieder 0 gibt.

Beispiel: Im Speicherplatz 3 sollen die Bits Nr. 4 bis 7 gelöscht werden:

```
43BA    3AØ3ØØ      LDA    3
43BD    E6ØF        ANI    ØFH
43BF    32Ø3ØØ      STA    3
```

Denn das UNDieren mit 0 löscht ein Bit, das UNDieren mit 1 läßt es unverändert.

Ganz entsprechend gibt es für ODER die Befehle ORI D8, ORA r und ORA M.
ORA A läßt den Akkuinhalt unverändert.

Beispiel: Es sollen im Register E das Bit Nr. 3 gesetzt und die anderen Bits unverändert gelassen werden:

```
2360    7B          MOV    A,E
2361    F608        ORI    8
2363    5F          MOV    E,A
```

Es ist nämlich 8 D = 00001000 B, also nur das Bit Nr. 3 von Null verschieden. Durch das ODERieren wird an den übrigen Bits im Akku nichts geändert, aber Nr. 3 wird gesetzt.
Dasselbe erreicht man mit

```
2360    3E08        MVI    A,8
2362    B3          ORA    E
2363    5F          MOV    E,A
```

Für Exklusiv-ODER gibt es XRI D8, XRA r und XRA M. Durch XRA wird der Akkuinhalt gelöscht, weil zwei gleiche Bits immer 0 ergeben.

Wenn auch der Begriff des Übertrages bei den logischen Verknüpfungen keinen Sinn hat, so läßt man doch aus Gründen der Zweckmäßigkeit das Übertragsbit beeinflussen: Es wird bei Durchführung einer jeden logischen Verknüpfung zurückgesetzt.

Deshalb kann man z. B. den Befehl ANA A (der ja am Akkuinhalt nichts ändert) dazu benutzen, das Übertragsbit zurückzusetzen; dasselbe bewirkt ORA A.
Eine wichtige Anwendung der logischen Verknüpfungen besteht in der „Maskierung", d. h. dem Herausfiltern gewisser Teile einer Information.
Sei z. B. die Aufgabe gestellt, den Wert von Bit Nr. 4 des Akkuinhalts zu ermitteln:

3223 E61Ø ANI ØØØ1ØØØØ B (oder 16D oder 1ØH)

Hierdurch werden alle Bits außer dem gesuchten gelöscht, dieses aber unverändert gelassen. Danach kann man prüfen, ob jetzt das ganze Datenwort Null geworden ist oder nicht. Die Prüfung besteht in der Anordnung eines von dieser Bedingung abhängigen Programmsprunges (Kapitel 6).

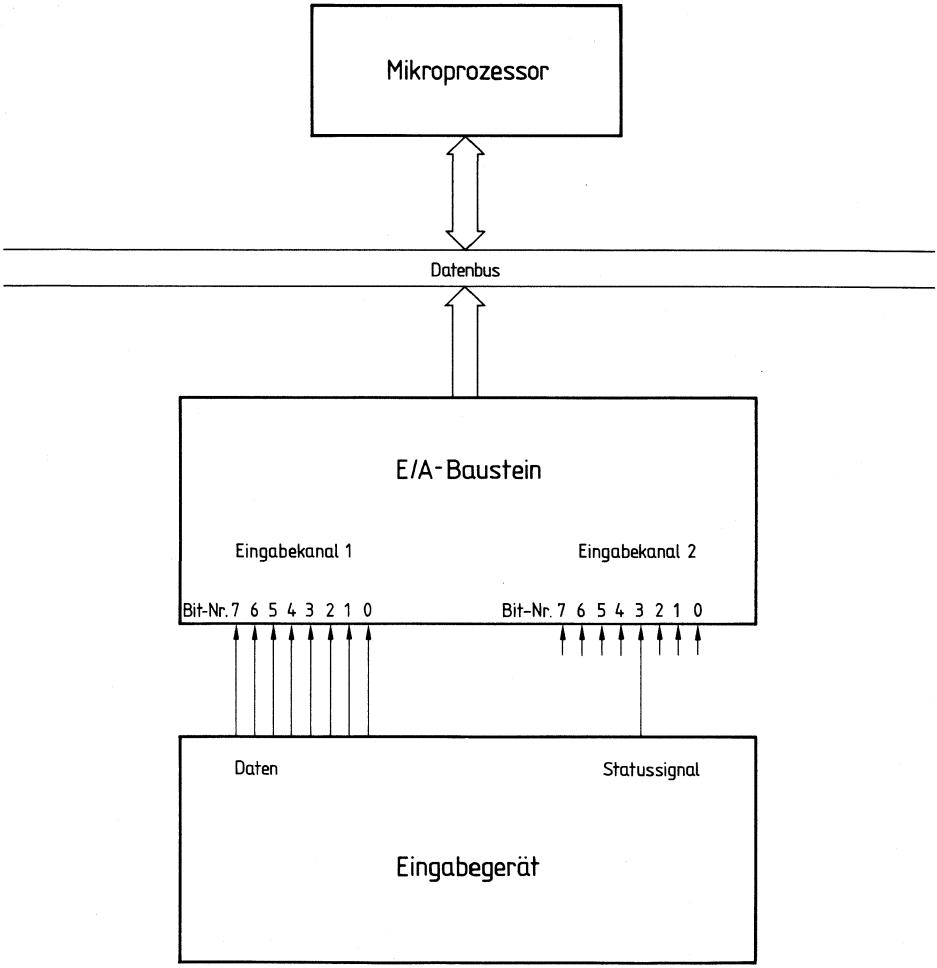


Bild 4.15
Beispiel für Eingabe-Maskierung

Außer durch logische Verknüpfung kann man auch durch andere Befehle die Voraussetzung für eine solche Prüfung schaffen; siehe hierzu die Rotationsbefehle in Abschnitt 4.8.
Maskierung kommt häufig bei Ein-/Ausgabe-Aktionen vor. Bild 4.15 zeigt eine solche Situation.
Ein Eingabegerät liefert Daten zu unvorhersehbaren Zeitpunkten an den Eingabekanal Nr. 1. Der Mikroprozessor prüft regelmäßig, ob Daten da sind; dies wird als Abfrageverfahren bezeichnet. Hierzu kann der Mikroprozessor im Prinzip prüfen, was am Eingabe-Kanal Nr. 1 anliegt. Dann müßte man aber dafür sorgen, daß definiert ist, woran man „neue Daten" erkennen kann. Oft ist es einfacher, ein Steuersignal („Statussignal"; Status = Zustand) vom Eingabegerät in einen anderen Eingabekanal zu geben. Das Steuersignal wird immer dann geschickt, wenn neue Daten eingegeben werden. Das Signal soll im Beispiel darin bestehen, daß das Bit Nr. 3 im Eingabekanal Nr. 2 den Wert 1 bekommt. Die anderen Bits dieses Kanals können anderen Zwecken dienen.
Die Befehle zum Abfragen lauten

```
12A4    DBØ2        IN      2
12A6    E6Ø8        ANI     8
```

Danach wird durch bedingten Sprung (Kapitel 6) geprüft, ob jetzt der Akkuinhalt 0 ist; wenn ja, ist kein Signal des Eingabekanals Nr. 1 angekündigt.
Das gesamte Datenwort im Eingabekanal Nr. 2 kann aus verschiedenen Statussignalen bestehen und heißt dann Statuswort oder Zustandswort.

Ein Maskierungsbeispiel für eine Ausgabe-Aktion:
Es sei die Aufgabe gestellt, 4 von 8 Lämpchen zu löschen und die anderen unverändert zu lassen. Der Zustand der Lämpchen werde gesteuert durch ein vom Mikroprozessor zum Ausgabekanal 3 ausgegebenes Datenwort; jedem Bit soll ein Lämpchen entsprechen, und Bitwert 1 heißt „eingeschaltet". Ferner kann der Mikroprozessor ein Zustandswort vom Eingabekanal 4 einholen, das den gegenwärtigen Zustand der Lämpchen anzeigt.
Wenn die Lämpchen 1, 3, 5 und 7 gelöscht werden sollen, dann lauten die Befehle

```
12BØ    DBØ4        IN      4     ;ZUSTANDSWORT GEHOLT
12B2    E655        ANI     55H   ;ODER Ø1Ø1Ø1Ø1B
12B4    D3Ø3        OUT     3     ;DATENWORT AUSGEGEBEN
```

4.7.2. Weitere logische Operationen

Bei den **Vergleichsoperationen** wird ein durch den Befehl bezeichnetes Byte als Zahl aufgefaßt und mit dem Akkuinhalt verglichen. Dies geschieht durch Subtraktion des Byte vom Akkuinhalt; nur wird das Resultat nicht im Akku gespeichert. Wenn aber das subtrahierte Byte größer war als der Akkuinhalt, wird das Übertragsbit gesetzt; dies wird wie üblich danach durch eine Prüfung festgestellt (Kapitel 6). Auch andere Zustandsbits werden beeinflußt und können z.T. zu ähnlichen Prüfungen verwendet werden.

Mit denselben Adressierungsarten wie bei den logischen Verknüpfungen gibt es die Befehle CPI D8, CMP r und CMP M. „CP" oder „CMP" steht für „compare" (= vergleiche).

Soll z.B. geprüft werden, ob die Bits Nr. 0, 1, 6 und 7 eines Byte gleichzeitig den Wert 1 haben, so bringt man das Byte in den Akku und ordnet dann an

```
2453    E6C3        ANI     ØC3H    ;BITS 2 BIS 5 GELOESCHT
2455    FEC3        CPI     ØC3H
```

Danach ist zu prüfen, ob das Resultat = 0 war (Kapitel 6).

Eine weitere logische Operation ist das **Komplementieren.** Hierbei wird im Akkuinhalt jede Null durch Eins ersetzt und umgekehrt. Aus 11001001 wird so 00110110. Der Befehl dazu lautet CMA, „complement accumulator". Ist das komplementierte Datenwort eine Zahl, so ergibt die Komplementierung das sogenannte Einerkomplement dieser Zahl. Addiert man 1 zum Einerkomplement, so ergibt sich das Zweierkomplement; seine Bedeutung für die binäre Subtraktion wird in Kapitel 9 erläutert.

Manchmal müssen Signale (unabhängig davon, ob sie Zahlen sind) ganz allgemein aus Hardwaregründen umgepolt werden, nachdem sie eingegeben wurden oder bevor sie ausgegeben werden. Auch hierzu kann der Komplementierungsbefehl dienen. Man spricht dann von Valenzanpassung.
Wenn man aber beim Herstellen des Programmes noch nicht die Peripheriegeräte kennt, weiß man auch noch nicht, ob das Umpolen nötig sein wird. Dann kann man sich wie folgt absichern. Man setzt hinter jedes IN Nr und vor jedes OUT Nr den Befehl XRI Ø. Hierdurch wird der Akkuinhalt nicht geändert; dies überlegt man sich mit Hilfe der Wahrheitstafel der Exklusiv-ODER-Operation. Das Übertragsbit und die anderen Zustandsbits werden allerdings beeinflußt.
Stellt sich dann heraus, daß irgendein Bit in einem E/A-Kanal umgepolt, d.h. komplementiert werden muß, so wird im Operanden des XRI-Befehles dieses Bit von „0" auf „1" geändert. Diese Änderung im Befehlsspeicher kann in der Praxis häufig durch Ändern eines PROM-Speichers (d.h. eines vom Anwender programmierbaren ROM) „vor Ort" erfolgen.

Eine spezielle Operation wird bewirkt durch DAA, „decimal adjust accumulator" (= Akku **dezimal-korrigieren**). Man braucht sie in folgendem Zusammenhang.

Dezimalzahlen treten in der Digitaltechnik oft in BCD-verschlüsselter Form auf, d.h. als „Binär Codierte Dezimalzahlen". So bürgert sich beispielsweise bei dezimal anzeigenden Digitalvoltmetern immer mehr ein BCD-Ausgang ein.
Die Verschlüsselung besteht darin, daß jede Dezimalziffer einzeln ersetzt wird durch eine 4 Bit-Binärzahl des gleichen Wertes, z.B. 3 durch 0011, 9 durch 1001.
Werden solche Zahlen in den Computer gegeben, so kann man sie mit Hilfe eines Unterprogrammes in Binärzahlen verwandeln; man muß es aber nicht unbedingt, und dann braucht man nach arithmetischen Operationen ggf. den DAA-Befehl.

Die BCD-Zahlen sind keine Binärzahlen; denn auf 00001001 folgt binär 00001010, aber in BCD 00010000. Es gibt, anders ausgedrückt, 4-Bit-Gruppen, denen keine Dezimalziffer entspricht.
Ohne Begründung: Das liegt daran, daß man die Zehn nicht aus lauter Faktoren Zwei zusammensetzen kann wie z.B. die Sechzehn.
Wenn man im Computer zwei BCD-Zahlen addiert, kann der Computer nicht anders rechnen als sonst, nämlich binär. Dabei kann es vorkommen, daß im Resultat auch solche 4 Bit-Gruppen erscheinen, denen keine Dezimalziffer entspricht. Wenn man das Resultat wieder korrekt in Dezimalziffern zurückübersetzt ausgeben will, muß man die DAA-Prozedur anwenden. Sie besteht darin, daß man zu jeder 4 Bit-Gruppe, die größer als 9 ist, noch 6 addiert und dabei die verschiedenen Überträge beachtet. Dies ist der einzige Befehl, dessen Ausführung vom Zustand des „Hilfsübertragsbits" abhängt; das ist ein Zustandsbit, das den Übertrag vom Bit Nr. 3 zum Bit Nr. 4 feststellt.

Der Befehl DAA wird unter den Befehlen für logische Operationen aufgeführt, weil die Entscheidung „größer als 9?" vorkommt.

Solange nur einfache Rechnungen vorkommen, kann dieses Verfahren bequem sein. Bei verwickelteren Rechnungen kann es einfacher sein, immer zuerst die eingegebenen BCD-Zahlen in Binärzahlen zu übersetzen. Auf jeden Fall in Binärzahlen übersetzen

muß man dann, wenn die Zahlen eine absolute Bedeutung für das Arbeiten des Computers haben, z. B. zum Zählen der Schritte eines Schrittmotors.

4.8. Weitere Befehle

Die große Gruppe der Sprungbefehle wird in Kapitel 6 besprochen. Hier folgen noch einige spezielle Befehle.

Rotationsbefehle: Durch RAL, „rotate accumulator left" (= rotiere Akku linksherum), Bild 4.16, wird der Akkuinhalt mit Einschluß des Übertragsbits linksherum rotiert.
Dasselbe entsprechend durch RAR (R = right = rechts), Bild 4.17.

Die Befehle RLC und RRC (Bild 4.18 und 4.19) enthalten den Buchstaben C für „content" (= Inhalt). Es wird dasjenige Bit ins Übertragsbit geschrieben, das bei der Akku-Rotation „außenherum" befördert wird.

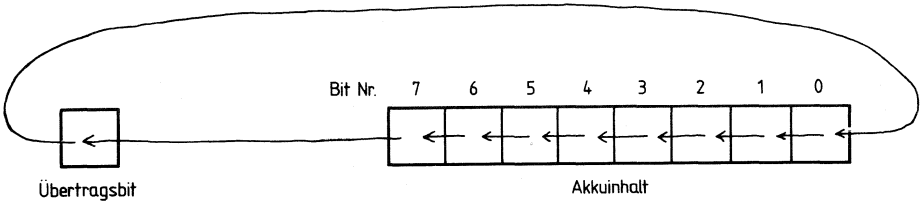


Bild 4.16
Der Befehl RAL

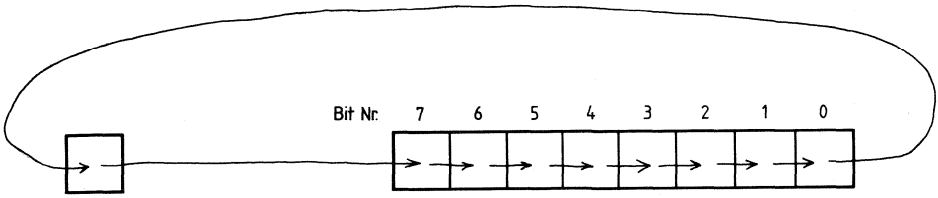


Bild 4.17
Der Befehl RAR

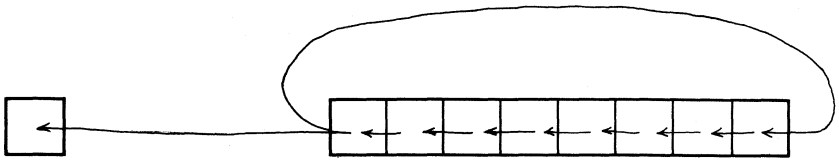


Bild 4.18
Der Befehl RLC

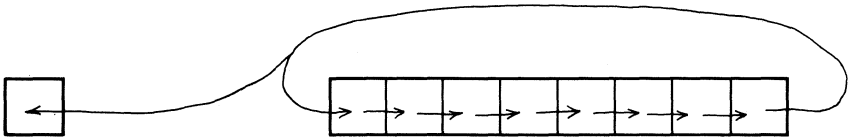


Bild 4.19
Der Befehl RRC

Derartige Befehle gestatten die Prüfung einzelner Bits; eine solche Prüfung wurde in Abschnitt 4.7.1 bereits durch logische Verknüpfungen ermöglicht.
Man erfährt den Wert des Bits Nr. 0, indem man RAR anordnet und prüft, ob jetzt das Übertragsbit gesetzt ist (Kapitel 6). Für das Bit Nr. 1 braucht man schon zweimal den Befehl RAR und für Bit Nr. 2 dreimal. Im letzteren Fall ist der Befehl ANI 4 mit 2 Bytes bereits ökonomischer; nach diesem Befehl wird nicht das Übertragsbit, sondern ein anderes Zustandsbit geprüft (das Nullbit, s. Kapitel 6).

Aus den 4 Rotationsbefehlen lassen sich auch andere Arten von Verschiebung des Akkuinhaltes ableiten. Beispiel:

```
Ø311    A7          ANA    A
Ø312    1F          RAR
```

Durch ANA A wird das Übertragsbit zurückgesetzt. Beide Befehle zusammen sind ein Einzelschritt einer „Rechtsverschiebung mit nachlaufenden Nullen."

Das besonders wichtige **Übertragsbit** kann direkt beeinflußt werden durch STC, „set carry" (= setze Übertrag) und CMC, „complement carry" zur Komplementierung. Ein besonderer Befehl zum Zurücksetzen ist wegen ANA A oder ORA A oder SUB A überflüssig.

Der Befehl NOP, **„no operation"** (= keine Tätigkeit) veranlaßt, daß nichts geschieht. Es verstreicht nur etwas Zeit; dies kann erwünscht sein. Der Operationscode und damit ganze Befehl lautet ØØH.
Durch NOPs kann man die Plätze etwa gestrichener Befehle füllen, falls man wünscht, daß sich die Adressen der übrigen Befehle nicht ändern. Umgekehrt kann man durch NOPs Plätze freihalten.

Der Befehl HLT, **„halt"** hält den Mikroprozessor an. Die Register und Zustandsbits behalten ihre Information. Weiterarbeit kann nur durch eine Unterbrechung (Abschnitt 6.3) veranlaßt werden.

5. Befehlszyklus

5.1. Definition der maßgebenden Zeitspannen

Kapitel 1 und 2 haben sich, wenn auch grob, mit den Hardware-Eigenschaften von Computern und speziell von Mikrocomputern befaßt. Kapitel 3 mit seinem Programmbeispiel und Kapitel 4 mit dem Befehlsvorrat behandelten reine Softwarefragen. Hier folgt eine Verfeinerung der Hardware-Betrachtung, nämlich der zeitliche Ablauf des Einholens und des Ausführens eines Befehles durch den Mikroprozessor.

Die Arbeit eines Computers wird zeitlich geordnet durch ein Taktgebersignal. Dieses Signal zerlegt die Arbeitszeit in **Takte**. Eine Taktlänge kann beim SAB 8080/85 zwischen 0,2 und 2 µsec betragen; die obere Grenze wird gesetzt durch die Notwendigkeit rechtzeitiger Auffrischung der Information, weil seine Register nach dem Kondensatorprinzip arbeiten (Bild 1.1).
Bei bekannter Länge des Taktes kann man mit Hilfe von Tabelle 4-3 berechnen, wie lange das Einholen und Ausführen eines bestimmten Befehles dauert; z. B. beim 0,5 µsec-Takt zwischen 2 µsec und 9 µsec je nach Befehl.

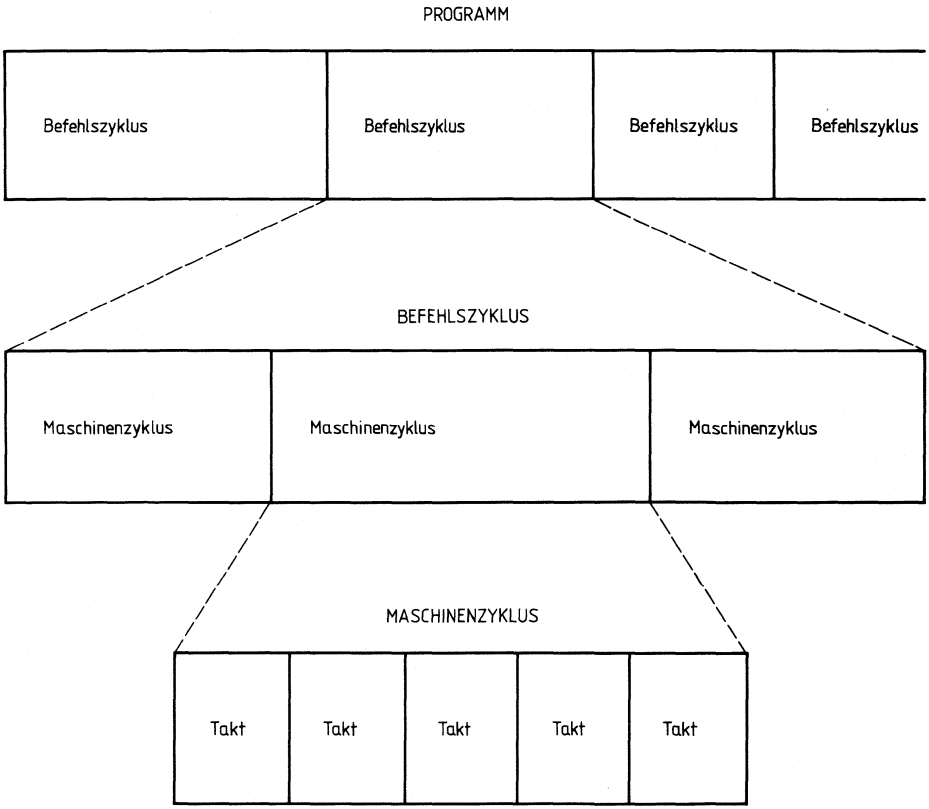


Bild 5.1
Die maßgebenden Zeitspannen

Die gesamte Zeitspanne des Einholens und Ausführens eines Befehles wird als **Befehlszyklus** bezeichnet. Beim SAB 8080/85 definiert man zudem als **Maschinenzyklus** eine Zeitspanne der Länge 3 bis 5 Takte; die Definition folgt sogleich. Jeder Befehlszyklus besteht aus mindestens einem Maschinenzyklus. Bild 5.1 zeigt die Zusammenhänge.
Die Definition besagt: Ein neuer Maschinenzyklus beginnt immer dann, wenn der Mikroprozessor eine besondere Art von Steuersignalen aussendet. Diese Signale teilen den anderen Bausteinen mit, welche Aktivität nun zu beginnen hat. Es gibt die Möglichkeiten „Speicher lesen", „Speicher schreiben", „Eingabe", „Ausgabe" sowie einige besondere Fälle, z. B. Quittieren des Befehles HLT. Soll bei „Speicher lesen" speziell ein Befehl eingeholt werden, so wird vor dem Einholen des Operationscode noch ein spezielles Signal hinzugefügt.
Für normale Befehlszyklen ohne die erwähnten besonderen Ereignisse kann man die Definition des Maschinenzyklus wie folgt vereinfachen: Ein neuer Maschinenzyklus beginnt immer dann, wenn der Mikroprozessor in einen neuen Datenaustausch mit Speicher oder E/A treten muß. Eine Ausnahme hiervon bildet der Befehl DAD; dies soll hier nicht weiter vertieft werden.

Auch für das Steuersignal zu Beginn eines Maschinenzyklus wird die Bezeichnung „Statuswort", „Zustandswort" oder „Statussignal" gebraucht (Schluß des Abschnitts 4.7.1); hier handelt es sich aber nicht um einen vorhandenen, sondern um einen kommenden Zustand.

Die Anzahl von Maschinenzyklen innerhalb eines Befehlszyklus liefert eine nicht so genaue Angabe über den Zeitverbrauch wie die Anzahl von Takten; denn ein Maschinenzyklus kann aus 3, 4 oder 5 Takten bestehen. Trotzdem wird in Abschnitt 5.2 jeder Befehlszyklus nur in seine Maschinenzyklen aufgelöst, weil dies eine anschauliche Vorstellung vom Arbeiten des Mikrocomputers liefert.

5.2. Einzelheiten eines Befehlszyklus

Als Beispiel wird der Befehl MOV C,A von Bild 4.13e benutzt; Bild 5.2 und 5.3 zeigen, wie er eingeholt wird.
Als erste Maßnahme (Bild 5.2) wird über den „Steuerbus" den anderen Bausteinen mitgeteilt „Speicher lesen, und zwar Befehl einholen". Hier tritt der Fall ein, der am Schluß von Abschnitt 2.2 erwähnt wurde: Dieses Signal geht in Wirklichkeit über den Datenbus hinaus; dieser wird in diesem Stadium noch nicht zur Datenübertragung benötigt.

Der Hinweis auf Einholen eines Befehles bedeutet nicht, daß jetzt nur der ROM aktiviert würde. Beim SAB 8080/85 können Befehle durchaus auch im RAM gespeichert sein. Woher der Befehl geholt wird, das bestimmt einzig und allein die nun vom Mikroprozessor anzugebende Adresse.
Diese Adresse findet der Mikroprozessor in einem internen „Befehslzähler" oder „Programmzähler". Das ist ein Registerpaar nur für diesen speziellen Zweck. Es wird vom Mikroprozessor selbst auf dem laufenden gehalten: Immer wenn ein Byte eines Befehles eingeholt wird, inkrementiert der Mikroprozessor den Befehlszähler.
Der Befehlszähler kann durch Sprungbefehle (Kapitel 6) auch vom Programm her beeinflußt werden. Hier wird aber vorläufig nur vom Normalfall gesprochen, in dem eine solche Beeinflussung unterbleibt.

Bild 5.2 illustriert, wie der Mikroprozessor den Inhalt des Befehlszählers auf den Adreßbus sendet. Dies geschieht (ebenso wie das Aussenden des Steuersignales über den „Steuerbus") im 1. Takt des Befehlszyklus, mit dem auch der 1. Maschinenzyklus dieses Befehlszyklus beginnt.

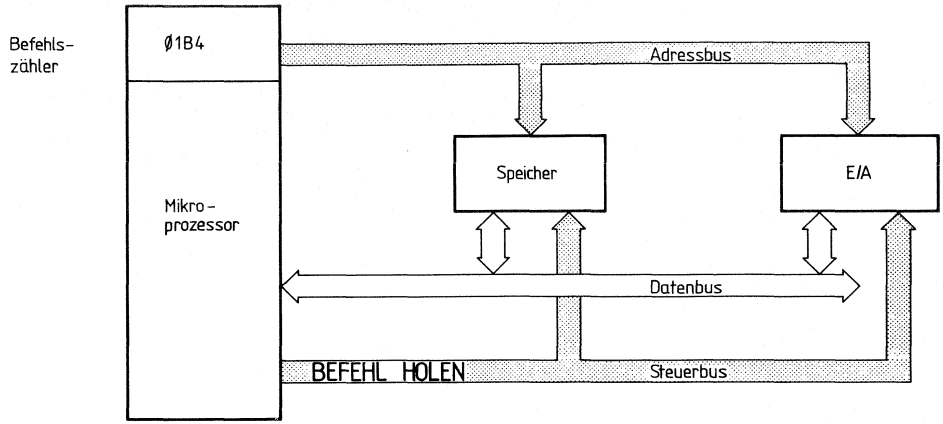


Bild 5.2
Einholen eines Operationscodes I

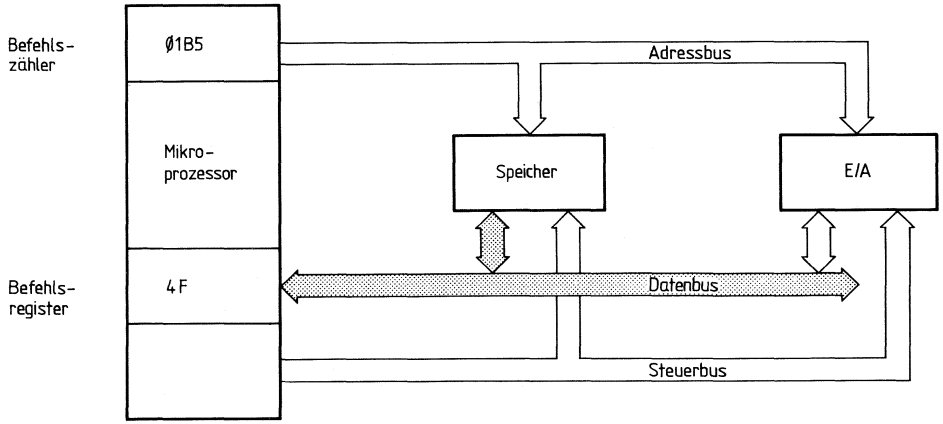


Bild 5.3
Einholen eines Operationscodes II

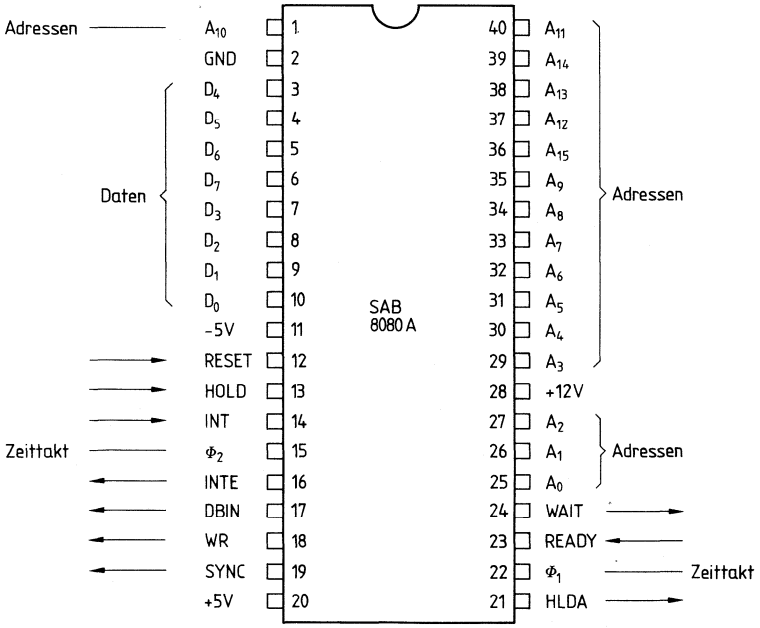
Bild 5.3 zeigt, wie daraufhin der ROM den Inhalt des angegebenen Speicherplatzes auf den Datenbus gibt; dies geschieht zu Beginn des 3. Taktes. Der Mikroprozessor „weiß", daß er dieses Byte als Operationscode zu interpretieren hat. Deshalb legt er

das Byte in seinem Befehlsregister ab, einem 8-Bit-Register; es dient (wie der 16-Bit-Befehlszähler) nur einem speziellen Zweck. Im 2. Takt hat der Mikroprozessor den Inhalt des Befehlszählers inkrementiert und verfügt damit über die Adresse des nächsten Operationscodes (oder bei Mehr-Byte-Befehlen des 2. Bytes).

Die Ausführung des Befehles „4F", d.h. MOV C,A erfordert keinen besonderen Maschinenzyklus mehr. Denn der Befehl verlangt nur eine Datenverschiebung im Inneren des Mikroprozessors. Diese findet in einem 4. und 5. Takt des ersten und einzigen Maschinenzyklus statt.

Anders ist es im Fall des Bildes 4.13a beim Befehl LDA Adr. Hier muß der Mikroprozessor den Inhalt eines Speicherplatzes einholen. Das Einholen allein des Befehles erfordert 3 Maschinenzyklen, weil die Adresse des betreffenden Speicherplatzes angegeben werden muß. Das 2. und 3. Byte wird in Hilfsregistern (W und Z) abgelegt. Ein 4. Maschinenzyklus dient der Ausführung des Befehles.

Schließlich noch der Fall des Befehles INR M. Hier ist der ganze Befehl nach dem 1. Maschinenzyklus eingeholt; er umfaßt ja nur das Byte „34". Aber die Ausführung braucht 2 weitere Maschinenzyklen; denn der Inhalt des Speicherplatzes M muß erst zur Inkrementierung geholt, und das inkrementierte Byte muß dann wieder in den Speicher gebracht werden.



Stromversorgung:
Stifte 2(GND), 11(-5V), 20(+5V), 28(+12V)

Bild 5.4
Anschlußbelegung des Mikroprozessors SAB 8080A

5.3. Steuersignale

Wie am Ende von Abschnitt 2.2 erwähnt, laufen Steuersignale auf unterschiedlichen Wegen. Der Grund dafür ist u.a. die begrenzte Anzahl von 40 Kontakten am Gehäuse des Mikroprozessors, siehe Bild 5.4.
16 Kontakte werden an den Adreßbus angeschlossen, 8 an den Datenbus, 4 an die Stromversorgung und 2 an den Zeittakt. Im Bild 5.4 ist durch Pfeile markiert, welche 10 Kontakte für reine Steuerzwecke übrigbleiben. Die Richtung der Pfeile zeigt, ob es sich um herein- oder hinausgehende Signale handelt.
Das „Statuswort", das vor einem neuen Maschinenzyklus über den Datenbus geht, läuft normalerweise zu einem Kontrollbaustein (Bild 2.2); er leitet daraus Signale ab, die dann tatsächlich über reine Steuerleitungen zu den anderen Bausteinen gehen. Der Engpaß sind eben die Kontakte des Mikroprozessors.

Es folgen einige Stichworte zum Zweck der Steuerkontakte am Mikroprozessor.
Zu Beginn eines neuen Maschinenzyklus gibt der Mikroprozessor am Kontakt SYNC ein Synchronisationssignal hinaus.
Am Kontakt READY erhält der Mikroprozessor im 2. Takt ggf. ein Signal vom Speicher oder von E/A, wenn mehr Zeit bis zum Datenaustausch benötigt wird. Dies wird dann durch ein Signal aus dem Kontakt WAIT quittiert. „Ready" heißt „bereit", und „wait" heißt „warten". In einem solchen Warte-Fall dauert also die Ausführung eines Befehls länger, als Tabelle 4-3 angibt.
Ein Signal des Mikroprozessors über DBIN, „data bus in" (= Datenbus herein) teilt mit, daß jetzt Daten vom Datenbus zum Mikroprozessor hereingeholt werden.
Über den Kontakt WR, „write" (= schreiben) wird beim Speicher-Schreiben oder beim Ausgeben von Daten ein zusätzliches Synchronisationssignal geschickt.
Die Kontakte INTE, INT, HOLD und HLDA dienen der Abwicklung von Anforderungen der Peripheriegeräte. Es handelt sich erstens um die Unterbrechung (= „interrupt"; siehe Abschnitt 6.3), zweitens um direkten Speicherzugang eines Peripheriegerätes. Während des letzteren befindet sich der Mikroprozessor im „Hold"-Zustand (= halten); dies darf keinesfalls verwechselt werden mit der Wirkung des Programm-Befehles HLT.

Das Steuersignal RESET schließlich setzt den Programmzähler auf die Adresse Null (siehe Abschnitt 2.4).

Zu den Steuersignalen im weitesten Sinne kann man auch das Steuerwort für programmierbare E/A-Bausteine rechnen. Es gibt sehr flexible E/A-Bausteine, denen je nach Anwendung unterschiedliche Arbeitsweisen verordnet werden können. Ein Beispiel: Man kann bestimmen, welche Kontakte dieser Bausteine zur Daten-Eingabe und welche zur Daten-Ausgabe dienen sollen.
Solche Festlegungen brauchen nicht hardwaremäßig zu erfolgen, sondern können durch das Programm ausgesprochen werden. Hierzu geht ein Steuerwort an die Steuerelektronik des E/A-Bausteinees. Ein solcher Baustein belegt also eine Kanal-Nr. für die Steuerinformation.
Das in Bild 8.1 behandelte Beispiel benutzt diese Methode.

6. Sprünge, Unterprogramme und Unterbrechungen

6.1. Programmsprünge

Ein Programm besteht aus einer Aneinanderreihung von Befehlen, die i.a. in einem ROM gespeichert sind. Normalerweise werden die Befehle in derjenigen Reihenfolge eingeholt und ausgeführt, die der Reihenfolge ihrer ROM-Adressen entspricht. Dies wird vom Mikroprozessor mit Hilfe des Befehlszählers erreicht (Abschnitt 5.2). Bisher kamen im Text nur solche Programmausschnitte vor, bei denen der jeweils nächste Befehl immer vom jeweils nächsten ROM-Platz eingeholt werden mußte.

Wenn man diese Übereinstimmung von Adressen- und Befehlsreihenfolge von Anfang bis Ende eines Programmes durchhält, bekommt man entweder ein primitives oder ein umständliches Programm. Flexibler werden Programme durch die Möglichkeit, je nach Lage an manchen Stellen von der Adressen-Reihenfolge abzuweichen. Bei Erfüllung einer geeigneten Bedingung sucht dann der Mikroprozessor den nächsten Befehl nicht auf der folgenden Adresse, sondern auf derjenigen, die ihm angegeben wird. Man sagt dann, daß das Programm einen **bedingten Sprung** ausführt. Gelegentlich ist auch ein nicht an Bedingungen geknüpfter, **unbedingter Sprung** erforderlich.
Ein Sprung kann vorwärts oder rückwärts führen.

Bild 6.1 zeigt als erstes Beispiel einen bedingten Sprung; vorläufig werden die Sprungbefehle noch durch ausführlichen Text dargestellt und die Operationscodes durch mnemonische Abkürzung. Es wird hier angeknüpft an das Beispiel von Bild 4.15. Die Daten vom Eingabekanal 1 können abgerufen werden, wenn das Bit Nr. 3 im Eingabekanal 2 den Wert 1 hat. Um dies zu prüfen, werden durch ANI 8 alle Bits außer dem Bit Nr. 3 gelöscht. Die eigentliche Prüfung erfolgt durch den bedingten Sprungbefehl. Der Akkuinhalt ist jetzt dann und nur dann = 0, wenn das Bit Nr. 3 = 0 ist. In diesem Fall sind keine Daten vom Kanal 1 zu holen; der bedingte Sprung führt zu einer anderen Tätigkeit. Wenn aber Daten vorliegen, ist das Bit Nr. 3 und damit der Akkuinhalt nicht = 0. Dann wird der bedingte Sprungbefehl ignoriert, es folgt das Einholen und Verarbeiten der Daten von Kanal 1.

An die Verarbeitung der Daten von Kanal 1 schließt die Adresse 161C H an, wo das Programm bei Nichtvorliegen von Daten fortzusetzen war. Der bei 161C beginnende Programmteil muß nicht notwendigerweise die richtige Fortsetzung nach Verarbeiten von Daten des Kanals 1 sein. Wenn nicht, dann kann ein unbedingter Sprungbefehl zur richtigen Fortsetzung führen: Bild 6.2.

Eine Programmstelle, an der ein Befehl zu bedingtem Sprung steht, heißt **Verzweigung**. Dieser Name wird verständlich beim Anblick eines **Programmablaufplanes.** Das ist eine grafische Darstellung der Computertätigkeit; sie wird vor dem Schreiben eines Programmes angefertigt, um einen besseren Überblick zu geben. Bild 6.3 zeigt einen Programmablaufplan für Bild 6.2.
Hier ist die mehr oder weniger zufällige Reihenfolge der Speicherplätze unbeachtet geblieben; schon aus diesem Grunde erhält man ein übersichtlicheres Bild.

Befehle für bedingte Sprünge werden formuliert mit Hilfe der **Zustandsbits.** Das sind 1-Bit-Register im Mikroprozessor; von ihnen wurde in Abschnitt 4.5.1 bereits das **Übertragsbit** eingeführt. In Abschnitt 4.7.2 wurde das Hilfsübertragsbit erwähnt. Es spielt jedoch bei bedingten Sprüngen keine Rolle und wird daher hier nicht beachtet. Die restlichen 3 Zustandsbits werden von einfacheren Regeln beherrscht als das Übertragsbit. Sie werden gesetzt oder zurückgesetzt je nach den Eigenschaften eines Resultats arithmetischer oder logischer Operationen. Dabei spielt es keine Rolle, ob das Resultat im Akku oder anderswo erscheint oder nicht.

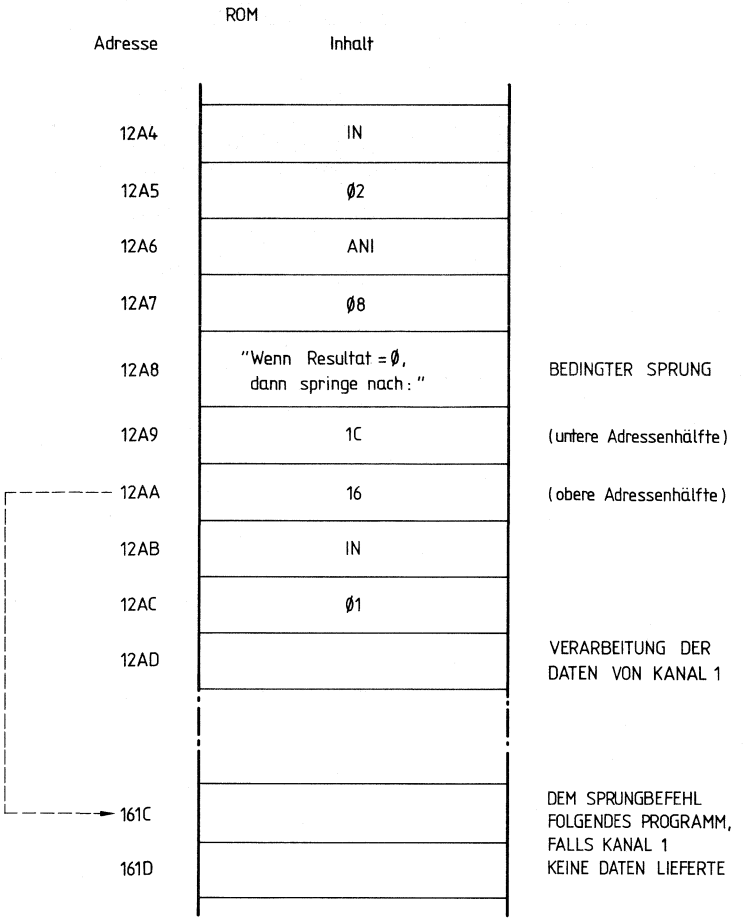


Bild 6.1
Bedingter Sprung im Beispiel von Bild 4.15

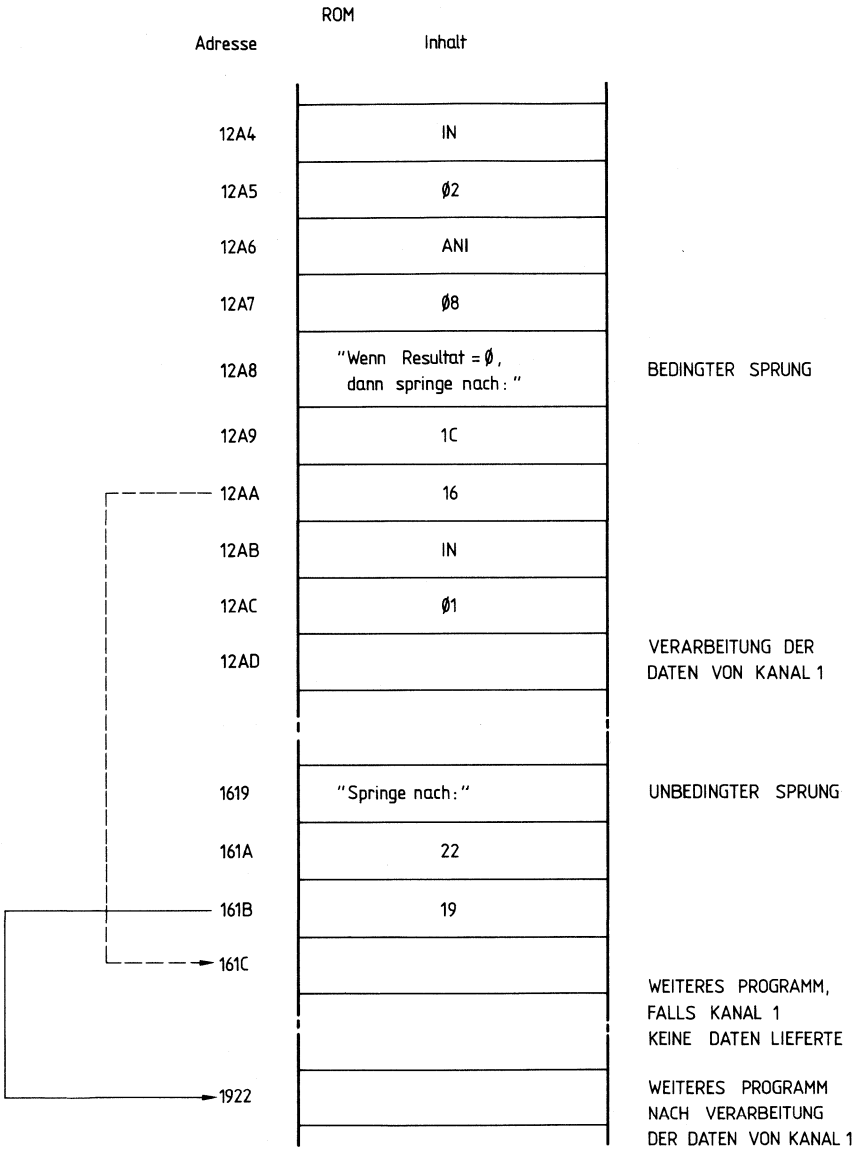


Bild 6.2
Bedingter und unbedingter Sprung

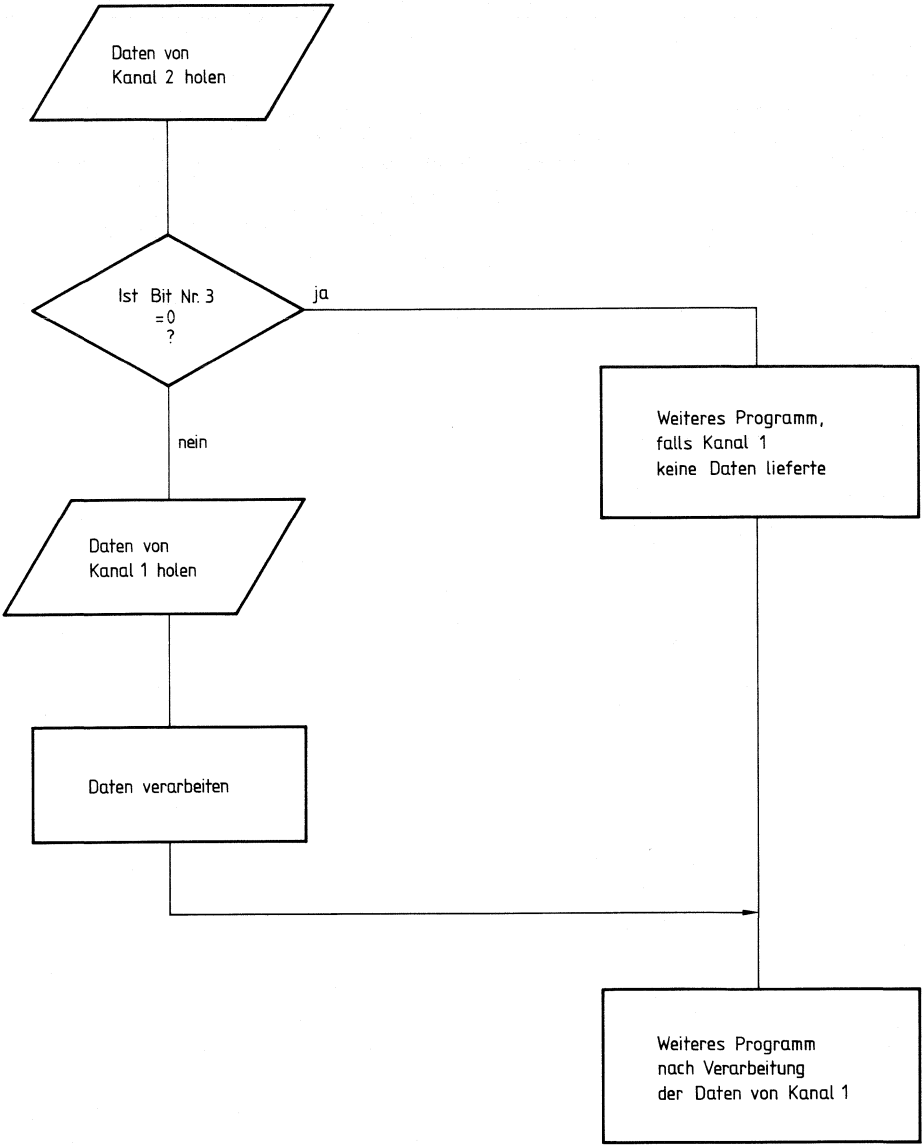


Bild 6.3
Programmablaufplan zu Bild 6.2

Die Tabelle 6-1 nach der folgenden Aufzählung zeigt, welche Befehle die Zustandsbits in welcher Weise beeinflussen. Außer dem Übertragsbit (und Hilfsübertragsbit) gibt es:

- **Nullbit:** Wird gesetzt, wenn ein Resultat den Wert 0 hat (ohne Berücksichtigung des Übertragsbits), sonst zurückgesetzt.
- **Vorzeichenbit:** Wird gesetzt, wenn Bit Nr. 7 eines Resultates = 1 ist, sonst zurückgesetzt. Der Name „Vorzeichenbit" bezieht sich auf eine spezielle, gelegentliche Bedeutung des Bits Nr. 7 (siehe Abschnitt 9.6).
- **Paritätsbit:** wird gesetzt, wenn Quersumme eines Resultates gerade ist (wenn das Resultat eine „gerade Parität" hat, nicht etwa, wenn es eine gerade Zahl ist!), sonst zurückgesetzt.

Es folgt die Tabelle über die Beeinflussungsmöglichkeiten der Zustandsbits ohne Hilfsübertragsbit.

Tabelle 6-1
Beeinflussung der Zustandsbits

Durch Eigenschaften eines Resultates werden beeinflußt:
Alle Zustandsbits bei ADD, ADI, ADC, ACI, SUB, SUI, SBB, SBI, CMP, CPI, DAA.
Das **Übertragsbit** bei DAD.
Alle Zustandsbits außer dem Übertragsbit bei INR, DCR (nicht bei INX, DCX!).

Trivialerweise werden beeinflußt:
Alle Zustandsbits bei POP PSW.
Das **Übertragsbit** bei RLC, RRC, RAL, RAR, STC, CMC.

Willkürlich zurückgesetzt wird:
Das **Übertragsbit** bei ANA, ANI, XOR, XRI, OR, ORI.

Mit Hilfe der Tabelle 6-1 versteht man die Beeinflussung der Zustandsbits im folgenden Beispiel (dessen Befehlsreihenfolge keinen Sinn ergibt). Die Zustandsbits sind durch ihre Anfangs-Buchstaben bezeichnet. In Klammern stehen ihre Werte dann, wenn sie mangels Beeinflussung unverändert blieben.

	Befehl	**Akkuinhalt**	**Ü**	**N**	**V**	**P**
		10010111	(x)	(x)	(x)	(x)
(C-Inhalt: 11001111)	ADD C	01100110	1	0	0	1
	ORA A	01100110	0	0	0	1
	CPI 64D	01100110	0	0	0	0
	RAL	11001100	0	(0)	(0)	(0)
	RAL	10011000	1	(0)	(0)	(0)
	ORA A	10011000	0	0	1	0
(B-Inhalt: 11000011)	SUB B	11010101	1	0	1	0
	SBI 1	11010100	0	0	1	1
	SUB A	00000000	0	1	0	1
	STC	00000000	1	(1)	(0)	(1)
	RRC	00000000	0	(1)	(0)	(1)
	DCR A	11111111	(0)	0	1	1

Die Werte der Zustandsbits werden gelegentlich mit dem Inhalt des Akkus zu einem 2-Byte-Datenwort zusammengefaßt, dem Inhalt des „Registerpaares PSW", „pro-

cessor status word" (= Statuswort des Prozessors). Dabei nimmt der Akku die höherwertige Hälfte ein, die Zustandsbits nehmen 5 Plätze der niederwertigen Hälfte ein. Das Registerpaar PSW wird betroffen von den Befehlen PUSH PSW und POP PSW (Abschnitt 6.2).

Der jeweilige Wert der Zustandsbits (außer dem Hilfsübertragsbit) wird beim **bedingten Sprung** als Entscheidungsmerkmal benutzt. Mit Abkürzungen für carry = Übertrag, zero = Null, even = gerade, odd = ungerade lauten die Befehle (siehe auch Tabelle 4-1)

JC	= jump on carry	= springe, wenn Übertragsbit gesetzt
JNC	= jump on no carry	= springe, wenn es zurückgesetzt,
JZ	= jump on zero	= springe, wenn Nullbit gesetzt,
JNZ	= jump on no zero	= springe, wenn es zurückgesetzt,
JM	= jump on minus	= springe, wenn Vorzeichenbit gesetzt,
JP	= jump on positive	= springe, wenn es zurückgesetzt,
JPE	= jump on parity even	= springe, wenn Paritätsbit gesetzt,
JPO	= jump on parity odd	= springe, wenn es zurückgesetzt.

Hinter jedem Operationscode für einen solchen Befehl folgt die Adresse, von der der nächste Befehl geholt werden soll.
Mit den genannten mnemonischen Abkürzungen lautet der bedingte Sprung im Bild 6.1 in Assembler-Listing:

```
12A8    CA1C16          JZ      161CH
```

Ein Befehl für **unbedingten Sprung** ist JMP Adr, „jump" (= springe) mit darauffolgender Adresse. Ein weniger direkter Befehl für unbedingten Sprung ist PCHL, „to program counter from HL" (= zum Befehlszähler von HL); hier erfolgt der Sprung zu derjenigen Adresse, die im Registerpaar HL steht.
So lautet im Assembler-Listing der unbedingte Sprung von Bild 6.2:

```
1619    C32219 .        JMP     1922H
```

Weiteres Beispiel: Das vom Programmverlauf abhängige Sprungziel stehe in den Speicherplätzen 564C H und 564D H. Der unbedingte Sprung erfolgt durch

```
Ø53Ø    2A4C56          LHLD    564CH
Ø533    E9              PCHL
```

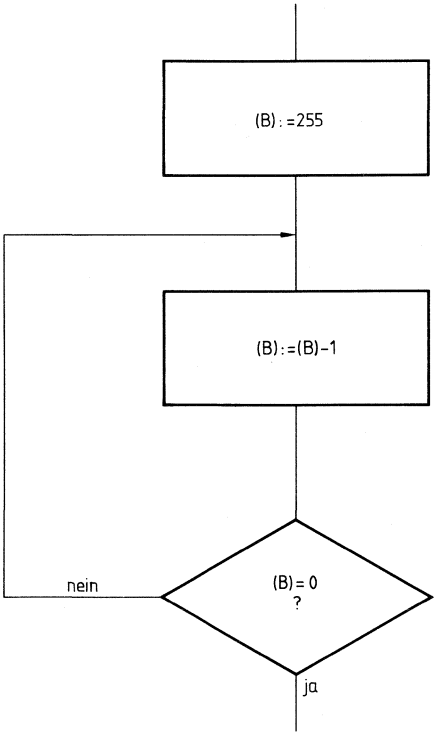
Eine andere Möglichkeit für ein datenabhängiges Ziel eines unbedingten Sprunges wäre Eingabe der Adresse durch ein Peripheriegerät:

```
Ø53Ø    DBØ4            IN      4
Ø532    67              MOV     H,A
Ø533    DBØ5            IN      5
Ø535    6F              MOV     L,A
Ø536    E9              PCHL
```

Es folgt ein weiteres Beispiel für bedingte Sprünge. Hier soll das Verstreichen einer bestimmten Zeitspanne erreicht werden. Dazu lädt der Programmierer ein Register (hier B) mit einer geeigneten Zahl (hier FF H = 255 D) und dekrementiert den Inhalt des Registers schrittweise. Nach jeder Dekrementierung wird geprüft, ob der Inhalt des Registers schon zu Null geworden ist. Bild 6.4 enthält den Programmablaufplan für das folgende Beispiel:

```
21AD    Ø6FF            MVI     B,ØFFH  ;ZAEHLER SETZEN.
21AF    Ø5      ZAEHL:  DCR     B       ;ZAEHL-
21BØ    C2AF21          JNZ     ZAEHL   ;/SCHLEIFE.
```

Danach folgt der nächste Befehl, der bei Nichtausführung des Sprunges (weil in B jetzt Null steht) auszuführen ist.



sprich: "Inhalt von B ersetzt durch 255"

sprich: "Inhalt von B ersetzt durch bisherigen Inhalt von B, der um 1 vermindert ist."

Bild 6.4
Programmablaufplan für Zeitschleife

Die Adresse ZAEHL ist eine willkürliche Markierung des Sprungzieles im Sinne von Bild 4.2.
Tabelle 4-1 zeigt, daß der Befehl DCR r das Nullbit beeinflußt; dies ermöglicht den bedingten Sprung JNZ Adr. Solange der Inhalt des Registers B noch nicht auf Null heruntergezählt ist, erfolgt jedesmal ein Rücksprung zu erneuter Dekrementierung.
Tabelle 4-3 gibt für MVI r,D8 eine Dauer von 7 Takten an, für DCR r von 5 Takten und für JNZ Adr von 10 Takten. Die letzteren beiden Befehle werden je 255mal angewendet. Die Programmpassage verbraucht also insgesamt 3832 Takte. Wenn z. B. ein Takt 0,5 µsec dauert, entspricht dies einer Zeitspanne von 1,916 msec.

Geringfügige Verlängerungen einer abzuwartenden Zeitspanne erreicht man durch Einfügen von Leerbefehlen wie NOP oder MOV A,A. Größenordnungsmäßige Verlängerungen erreicht man durch Schachteln von „Schleifen" wie folgt:

```
Ø1Ø2    ØEFF            MVI   C,255D
Ø1Ø4    Ø6FF    SCHL2:  MVI   B,255D;   Beginn Außenschleife
Ø1Ø6    Ø5      SCHL1:  DCR   B      ;   Beginn Innenschleife
Ø1Ø7    C2Ø6Ø1          JNZ   SCHL1 ;   Ende Innenschleife
Ø1ØA    ØD              DCR   C
Ø1ØB    C2Ø4Ø1          JNZ   SCHL2 ;   Ende Außenschleife
```

Das wiederholte Durchlaufen eines Programmstückes wird als Durchlaufen einer **Schleife** bezeichnet. In den beiden letzten Beispielen bestand der Zweck der Schleifen

darin, eine bestimmte Zeitspanne verstreichen zu lassen. Das kommt vor in Programmen, in denen die Zeit eine absolute Rolle spielt, die sich in „real time" (= in wirklicher Zeit) abspielen. Dazu gehört Prozeßsteuerung im weitesten Sinne. Beispielsweise muß ein Computer einem Schrittmotor Zeit lassen für das Erreichen eines neuen Ruhezustandes, bevor er ihm die nächste Anordnung ausgibt.

Bei anderen Programmen wie z. B. Lösung mathematischer Aufgaben ohne Bezug auf eine gerade laufende Prozeßsteuerung hat die Zeit keine absolute Bedeutung (wenn es auch durchaus interessiert, wieviel ggf. teure Rechenzeit verbraucht wird). Dann kann eine Programmschleife z. B. den Zweck haben, eine schrittweise Annäherung an ein Rechenresultat zu bewirken.

Bild 6.5 zeigt eine Kombination zweier bedingter Sprünge.
Der Programmablauf sei davon abhängig, ob zwei 8-Bit-Zahlen gleichgroß sind oder welche von ihnen größer ist. Man bringt die Zahlen in die Register A und B und ordnet zur Einleitung der Prüfung an CMP B.
Dann erfolgt die Prüfung auf Gleichheit durch JZ Adr oder JNZ Adr. Wenn keine Gleichheit besteht, muß geprüft werden, ob bei der Vergleichssubtraktion (A)−(B) ein negativer Übertrag auftrat; dies geschieht durch JC oder JNC.

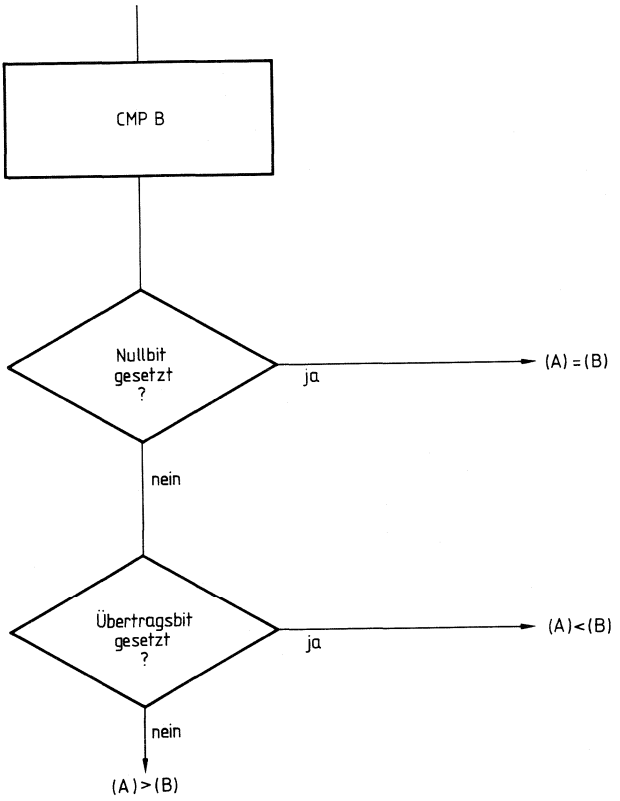
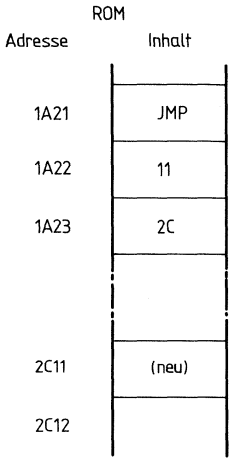


Bild 6.5
Auswertung des Vergleiches zweier Zahlen

Wenn ein Sprungbefehl in das Befehlsregister und in die Hilfsregister W und Z aufgenommen und wenn ggf. die Bedingung für den Sprung erfüllt ist, dann gibt der Mikroprozessor ausnahmsweise nicht den Inhalt des Befehlszählers auf den Adreßbus, sondern die soeben erhaltene Adresse aus den Hilfsspeichern.
Nach Ausführung des nach dem Sprung vorgefundenen Befehles erhält der Befehlszähler als neuen Inhalt die d a n n folgende Adresse. Damit ist nach einem Sprungbefehl die Erinnerung an die Stelle verlorengegangen, von der aus der Sprung erfolgte. Dieser Gesichtspunkt spielt eine Rolle bei den anschließend zu besprechenden Unterprogrammen.
Bild 6.6 zeigt den Inhalt der zuständigen Register und des Adreßbus in den verschiedenen Phasen eines Sprunges.



Zeitpunkt Nr.	Adresse auf Bus (16 Bit)	Inhalt Befehls-zähler (16 Bit)	Inhalt Befehls-register (8 Bit)	Inhalt Register W (8 Bit)	Inhalt Register Z (8 Bit)
1	1A21	1A21			
2	—	1A22	JMP		
3	1A22	1A22	JMP		
4	—	1A23	JMP		11
5	1A23	1A23	JMP		11
6	—	1A24	JMP	2C	11
7	2C11 !	1A24	JMP	2C	11
8	—	2C12	(neu)	2C	11
9	2C12	2C12	(neu)	2C	11

Bild 6.6
Adressenverwaltung während eines Sprunges

6.2. Unterprogramme

Eine spezielle Art von Sprüngen ist der Aufruf von Unterprogrammen. Sie dienen der Ausführung von Befehlsfolgen, die im Programm oft vorkommen, aber zwecks Ersparnis von ROM-Plätzen nur einmal gespeichert werden.
Der Aufruf ist ein Sprungbefehl zur Anfangsadresse eines Unterprogrammes. Er kann wie die bisher behandelten Sprungbefehle bedingt oder unbedingt sein. Bei bedingtem Aufruf werden wieder die Zustandsbits als Entscheidungsmerkmale verwendet.
Wozu braucht man dann überhaupt unterschiedliche Befehle für Sprünge im engeren Sinne einerseits, Aufrufe andererseits? Der Unterschied besteht darin, daß ein Unterprogramm mit der Rückkehr in das Hauptprogramm endet, wie es im Bild 6.7 gezeigt ist. Das ist jedesmal derjenige Befehl im Hauptprogramm, der dem Aufruf folgt; es ist also nach jeder Benutzung eines Unterprogrammes eine andere Adresse anzusteuern.

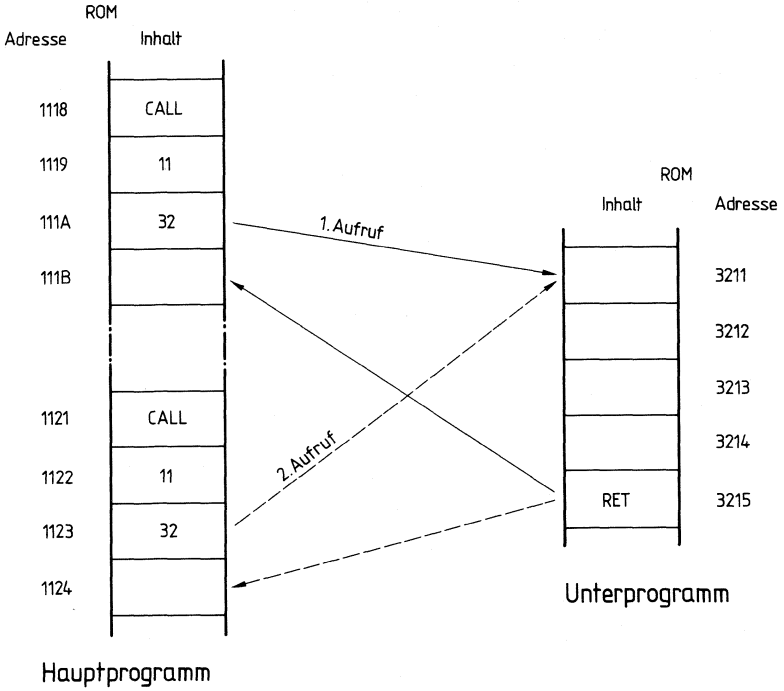


Bild 6.7
Mehrfacher Aufruf eines Unterprogrammes
(hier mit unbedingten Aufruf- und Rückkehrbefehlen)

Nun wurde bereits am Ende von Abschnitt 6.1 festgestellt, daß im Befehlszähler die Erinnerung an den Startpunkt eines Sprunges verlorengeht. Die Unterprogramm-Aufrufe unterscheiden sich von den einfachen Sprüngen durch zusätzliche Maßnahmen zur Rettung der Rückkehradresse. Bei einem bedingten Aufruf, der wegen Nichterfüllung der Bedingung nicht ausgeführt wird, unterbleiben diese Maßnahmen. Deshalb gibt Tabelle 4-3 auch jeweils zwei unterschiedliche Anzahlen von Takten bei bedingten Aufrufen an.

Also benötigen Unterprogramme ihre eigenen Befehle. Außerdem benötigen sie zusätzlich Rückkehrbefehle. Auch diese können bedingt oder unbedingt sein und sind, falls bedingt, wieder den Zustandsbits unterworfen. Siehe hierzu Tabelle 4-1. Der unbedingte Aufruf wird durch CALL (= rufe) bewirkt, der unbedingte Rückkehrbefehl ist RET, „return" (= kehre zurück). Die entsprechenden bedingten Befehle sind zusammengesetzt aus dem Anfangsbuchstaben C bzw. R und denselben Abkürzungen für die Sprungbedingung wie bei den Sprungbefehlen im engeren Sinn.

Nun zu den zusätzlichen Maßnahmen, mit denen die Rückkehradresse gerettet wird. Hierzu besitzt der Mikrocomputer einen **Kellerspeicher** (engl. stack = Stapel). Das ist ein Teil des RAM, der für diesen Zweck reserviert wird. Gewöhnlich benutzt man dazu die höchsten RAM-Adressen. Siehe auch Bild 2.2.
Den Zustand des Kellerspeichers verfolgt der **Stapelzeiger** (stack pointer). Das ist ein 16-Bit-Register im Mikroprozessor. Ihm wird zu Beginn eines Programmes eine Anfangsadresse eingeschrieben, üblicherweise die höchste RAM-Adresse. Immer wenn im Kellerspeicher etwas abgelegt wird (nicht immer eine Adresse, aber immer ein 16 Bit-Wort), zählt der Stapelzeiger mit. Dies geht wie folgt vor sich. Zuerst wird der Stapelzeiger dekrementiert und dann auf den Adreßbus geschaltet zwecks Adressierung des Kellerspeichers. Auf dieser Adresse wird die höherwertige Hälfte des zu speichernden 16 Bit-Wortes abgelegt. Dann erfolgt der gleiche Vorgang für die niederwertige Hälfte. Der Stapelzeiger zeigt damit auf das zuletzt gespeicherte Byte.

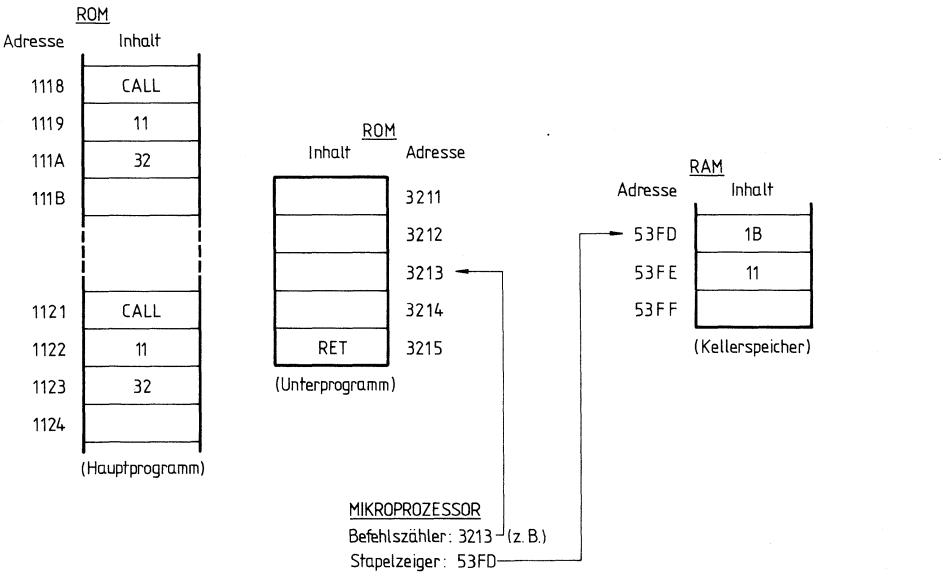


Bild 6.8
Momentaufnahme von Stapelzeiger, Befehlszähler und Kellerspeicher während des 1. Unterprogramm-Ablaufes von Bild 6.7

Dieser Vorgang erfolgt automatisch beim Aufruf eines Unterprogrammes (bei bedingtem Aufruf nur, wenn die Bedingung erfüllt ist). Dies kann z. B. dem Bild 6.8 für das Beispiel des Bildes 6.7 entnommen werden. Angenommen, der Kellerspeicher sei vor dem ersten Befehl CALL 3211H noch leergewesen; dann zeigte der Stapelzeiger noch auf die höchste RAM-Adresse 53FF H. Nach dem 1. Aufruf kommt

die Rückkehradresse vom Befehlszähler zum Kellerspeicher. Nun besteht die im Bild 6.8 gezeigte Situation von Kellerspeicher und Stapelzeiger. Der Befehlszähler zeigt nacheinander die Adressen des Unterprogrammes, hier gerade 3213H.

Bei Rückkehr in das Hauptprogramm wird die gerettete Rückkehradresse vom Kellerspeicher in den Befehlszähler zurückgebracht. Wird die Rückkehr bedingt angeordnet, so geschieht auch dies nur bei Erfüllung der Bedingung.
Es läuft alles umgekehrt ab wie bei Rettung der Adresse. Zuerst geht die niederwertige Hälfte in den Befehlszähler, und der Stapelzeiger rückt einen RAM-Platz aufwärts. Dann folgt die höherwertige Hälfte, und der Stapelzeiger wird wieder inkrementiert.

Zwischen zwei Befehlen zeigt der Stapelzeiger also immer auf die niederwertige Hälfte des 16-Bit-Wortes, das zuletzt in den Kellerspeicher kam und das zuerst wieder herausgeholt werden kann. Bevor dies geschieht, kann aber zuvor noch etwas anderes im Kellerspeicher abgelegt werden (und dann zuerst wieder herausgeholt werden).
Es ist wie beim Ablegen von Gegenständen in einen offengehaltenen Sack. Man kann fast unbegrenzt weitere Gegenstände hineinlegen, und ggf. wird das zuerst entnommen, was zuletzt hineinkam.

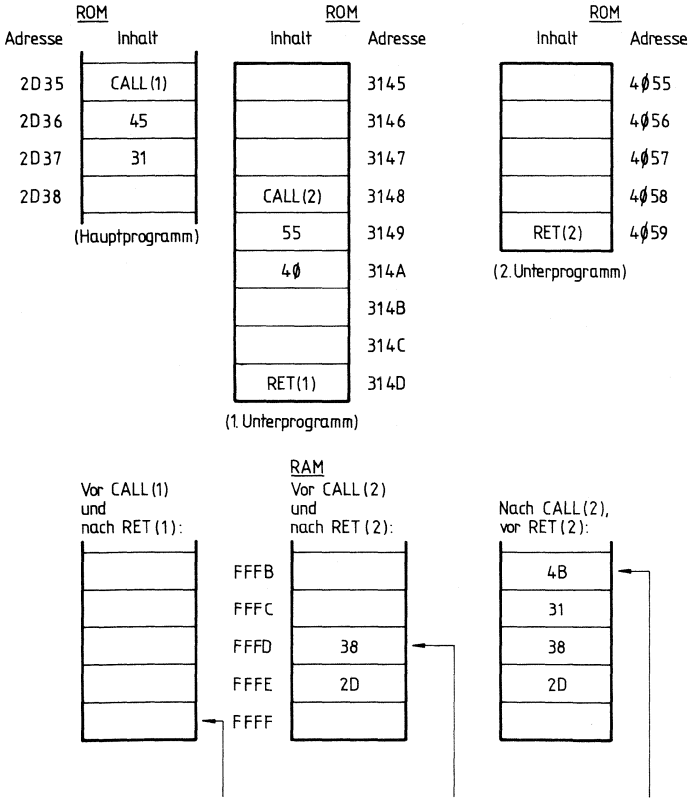


Bild 6.9
Verschachtelung zweier Unterprogramme
Unten: Momentaufnahmen des Kellerspeichers mit Stapelzeiger

Diese Eigenschaft des Kellerspeichers wird benutzt beim Auftreten verschachtelter Unterprogramme. Damit meint man den Vorgang, daß innerhalb eines Unterprogrammes wieder ein Unterprogramm aufgerufen werden kann. So zeigt Bild 6.9 die Verschachtelung von zwei Unterprogrammen (im Prinzip ist eine beliebig weitgehende Verschachtelung möglich).
Im unteren Teil des Bildes ist der jeweilige Zustand des Kellerspeichers gezeigt. Während des Ablaufes des zweiten Unterprogrammes sind zwei Rückkehradressen gespeichert.
Die beschriebene Tätigkeit von Kellerspeicher und Stapelzeiger erfolgt automatisch bei Aufruf von Unterprogrammen und bei Rückkehr aus Unterprogrammen, in beiden Fällen ggf. nur bei erfüllter Sprungbedingung. Man kann aber auch vom Programm her den Kellerspeicher und den Stapelzeiger beeinflussen. Genauer: Man kann erstens auch vom Programm her anordnen, daß 16 Bit-Daten in den Kellerspeicher gebracht oder von dort geholt werden; in diesem Fall geht der Stapelzeiger mit dem Ladezustand des Kellerspeichers mit, genau wie in den bisherigen Fällen. Außerdem kann man aber sogar zweitens den Stapelzeiger ändern, ohne daß am Kellerspeicher etwas geändert wird.
Im Folgenden wird zuerst die Beeinflussung des Kellerspeichers mit automatischem Mitgehen des Stapelzeigers besprochen.

Der Programmierer kann den Inhalt von Registerpaaren in den Kellerspeicher retten durch Transferbefehle (Tabelle 4-1) PUSH rp (= stoße hinein) und POP rp (= lasse heraussspringen). In Frage kommen die Registerpaare BC, DE, HL und PSW (siehe Abschnitt 6.1). Der Vorteil dieses Verfahrens besteht darin, daß man sich später nicht um die Adressen der zurückzuholenden Registerpaarinhalte kümmern muß.

Es kann z. B. vorkommen, daß alle Register belegt sind, jedoch für eine Berechnung gebraucht werden. Man ordnet an:

```
2465    C5                  PUSH    B
2466    D5                  PUSH    D
2467    E5                  PUSH    H
2468    F5                  PUSH    PSW
. . . . . (Berechnung) . . . . . .
2491    F1                  POP     PSW
2492    E1                  POP     H
2493    D1                  POP     D
2494    C1                  POP     B
```

An diesem Beispiel sieht man, daß der Programmierer auf die richtige Reihenfolge seiner POP-Befehle achten muß. Wenn im Beispiel die Befehle D1 und C1 miteinander vertauscht werden, dann ist im Endeffekt der Inhalt von BC mit dem Inhalt von DE vertauscht worden. Dies kann in manchen Fällen allerdings beabsichtigt sein.

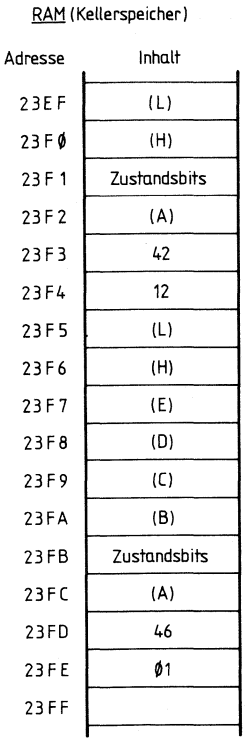
Will man die Zustandsbits ändern (sie stehen in der niederwertigen Hälfte von PSW), so kommt in Frage

```
1321    F5          PUSH    PSW
1322    C1          POP     B
1323    79          MOV     A,C
. . . . (Änderung der Zustandsbits) . . . .
1331    4F          MOV     C,A
1332    C5          PUSH    B
1333    F1          POP     PSW
```

Noch ein Beispiel: Die im Registerpaar HL gespeicherte Adresse soll um 160 D erhöht werden; die Registerpaare BC und DE seien belegt:

```
2E11    C5          PUSH    B
2E12    Ø1AØØØ      LXI     B,ØØAØH
2E15    Ø9          DAD     B
2E16    C1          POP     B
```

Die Benutzung des Kellerspeichers kann abwechselnd durch den Aufruf von Unterprogrammen und durch PUSH und POP erfolgen. Ein Beispiel (siehe Bild 6.10):




```
Ø143    CD3312          CALL    MULTI
Ø146    ....
....
1233    F5      MULTI:  PUSH    PSW
1234    C5              PUSH    B
1235    D5              PUSH    D
1236    E5              PUSH    H
....
123F    CD4712          CALL    ADD
1242    E1              POP     H
1243    D1              POP     D
1244    C1              POP     B
1245    F1              POP     PSW
1246    C9              RET
1247    F5      ADD:    PUSH    PSW
1248    E5              PUSH    H
1249    ....    BEIS:   ....
....
1256    E1              POP     H
1257    F1              POP     PSW
1258    C9              RET
```

Bild 6.10
Inhalt des Kellerspeichers zum Textbeispiel

Zuerst erfolgt unbedingter Aufruf, also wird die Rückkehradresse Ø146 H gerettet. Im Unterprogramm ab MULTI kommen zuerst vier PUSH-Befehle; die zuständigen Registerpaarinhalte füllen den Kellerspeicher weiter auf. Ein weiterer Unterprogrammaufruf bringt die zweite Rückkehradresse 1242 H in den Kellerspeicher. Im Verlauf dieses Unterprogrammes wird durch erneuten PUSH-Befehl der jetzige Inhalt von PSW und HL in den Kellerspeicher gebracht. Bei Erreichen der Adresse BEIS hat der Kellerspeicher den im Bild 6.10 gezeigten Inhalt.

Nach POP H und POP PSW erfolgt Rückkehr nach 1242 H, dann nach weiteren vier POP Rückkehr nach Ø146 H.

Außer durch PUSH rp und POP rp kann man den Kellerspeicher beeinflussen durch XTHL, „exchange top of stack with HL" (= tausche oberstes Datenwort des Kellerspeichers mit HL aus). Das „oberste" 16-Bit-Wort ist natürlich das zuerst zu entnehmende, in den gezeigten Bildern oben, aber mit der untersten Adresse, kurz: die letzte Eintragung.

Das Programm kann auch den Stapelzeiger direkt beeinflussen, o h n e am Kellerspeicherinhalt etwas zu ändern. Beim Transferbefehl LXI rp und bei den arithmetischen Operationen INX rp und DCX rp kommt der Stapelzeiger als viertes Registerpaar in Frage. Dagegen wird er durch DAD SP nicht beeinflußt. Ferner gibt es den Befehl SPHL, „to stack pointer from HL" (= zum Stapelzeiger von HL); hierdurch wird der Stapelzeiger auf die Adresse gerichtet, die in HL steht.
Zu Beginn eines Programmes kann man die Lage des Kellerspeichers festlegen durch LXI SP, D16 oder SPHL.
Diese Anfangsfestlegung ist notwendig, weil der Stapelzeiger sonst eine sinnlose Zufallsadresse enthält.

Ob sich ein Unterprogramm lohnt, muß von Fall zu Fall geprüft werden. Für jeden Aufruf braucht man 3 ROM-Plätze (dagegen für die Rückkehr nur einmal einen ROM-Platz im Unterprogramm selbst); ein Unterprogramm sollte also jedenfalls mehr als 3 Plätze belegen. Je nach seiner Länge gibt es eine Mindestanzahl von Aufrufen, von der ab das Unterprogramm sich lohnt.
Es gibt allerdings nicht nur den Gesichtpunkt der Speicherplatzökonomie, sondern auch den Gesichtspunkt der Rechenzeitökonomie. Bei zeitkritischen Programmteilen (wenn also ein Computer gegenüber real ablaufenden Ereignissen nicht in Verzug geraten darf) wird man gelegentlich auf Unterprogramme verzichten, um die Aufrufe und Rückkehrbefehle einzusparen. Dann läuft das Programm schneller ab; der Preis, den man dafür zahlt, besteht in größerem Aufwand an Speicherplätzen.
Auf Unterprogramme verzichten wird man auch dann, wenn das Programm sonst gar keinen RAM benötigen würde und wenn man einen RAM nur wegen des Kellerspeichers dazunehmen müßte.

Eine gewisse Ähnlichkeit mit einem Unterprogramm hat ein „Makro". Dieser Begriff wird am Schluß von Abschnitt 8.1 erklärt.

6.3. Unterbrechungen

Eine letzte Art von Sprungbefehlen sind die 8 Befehle RST D3, „restart" (= starte wieder), mit einer Binärzahl D3 zwischen 0 und 7 im Operationscode.
Dies ist ein 1-Byte-Sprungbefehl zu der Adresse 8 × D3. Die möglichen Zieladressen sind also 0, 8, 16, . . ., 56 D; das ist hexadezimal ØØØØ, ØØØ8, ØØ1Ø, . . ., ØØ38.
Der Befehl RST D3 veranlaßt ebenso wie ein Unterprogrammaufruf, daß die im Befehlszähler enthaltene Adresse in den Kellerspeicher gerettet wird. Die durch RST eingeleitete Befehlsfolge kann daher wie ein Unterprogramm durch einen Rückkehrbefehl beendet werden.
Tatsächlich besteht der einzige Unterschied zwischen RST D3 und den anderen Unterprogrammaufrufen darin, daß RST D3 nur 1 Byte benötigt gegenüber sonst 3; der Preis dafür sind die festgelegten Zieladressen.

Der Programmierer kann die RST-Befehle verwenden, wie er will. Ihr Hauptzweck besteht aber in der hardwarenmäßigen Bedienung von Unterbrechungen, die von E/A-Geräten angefordert werden. Der Grund für solche Unterbrechungen ist meistens, daß die Geräte viel langsamer arbeiten als der Mikroprozessor. Sie können dann nur zu unvorhersehbaren Zeitpunkten Daten liefern oder entgegennehmen.
Dann gibt es 2 Möglichkeiten. Entweder fragt der Mikroprozessor die Geräte ab (siehe z. B. Bild 4.15), oder es wird eben die Unterbrechungsmöglichkeit geschaffen.

Dabei gibt das Gerät hardwaremäßig erstens ein Signal auf den Kontakt INT des Mikroprozessors (Abschnitt 5.3) und sorgt zweitens dafür, daß ein RST-Befehl auf den Datenbus kommt. Der RST-Befehl kommt also in diesem Fall nicht aus dem Speicher.
Die Unterbrechungsmöglichkeit kostet also zusätzliche Hardware; dafür entfällt der zusätzliche Zeitaufwand, den das Abfragen verursachen würde.

Man kann hardwaremäßig dafür sorgen, daß die Unterbrechung ein für allemal möglich oder ausgeschlossen ist. Man kann aber auch durch die Programmbefehle DI und EI (Schluß von Tabelle 4-1) softwaremäßig von Fall zu Fall die Unterbrechungsmöglichkeit geben.
„DI" steht für „disable interrupt" (= schließe Unterbrechung aus), „EI" für „enable interrupt" (= ermögliche Unterbrechung). Durch EI oder DI wird ein 1 Bit-Register im Mikroprozessor gesetzt oder zurückgesetzt; sein Zustand kann am Kontakt INTE des Mikroprozessors geprüft werden (Abschnitt 5.3).
Man kann in zeitkritischen Programmpassagen vorübergehend die Unterbrechungsmöglichkeit ausschließen (durch DI) und danach wieder zulassen (durch EI).

Läßt man hardwaremäßig Unterbrechungen zu, so muß man dafür sorgen, daß auf den betreffenden Zieladressen der RST-Befehle geeignete Unterprogramme beginnen. Speziell für RST 7 kann das Unterprogramm beliebig lang sein. Beim Start auf RST 0 bis RST 6 ist die Anzahl der Speicherplätze u.U. begrenzt und muß ggf. für Sprungbefehle zu noch anderen ROM-Plätzen benutzt werden.

Die Adresse ØØØØ H ist durch das RESET-Signal ausgezeichnet (siehe Abschnitt 2.4 und 5.3); durch dieses Steuersignal erhält der Programmzähler den Inhalt ØØØØ. Daher muß, wenn die Befehle RST1 usw. benutzt werden, wenn also das Programm bei der Adresse ØØØØ noch nicht beginnen kann, auf ØØØØ und den folgenden Plätzen ein unbedingter Sprungbefehl zum eigentlichen Programmbeginn stehen. Infolgedessen kann (ohne weitere Maßnahmen) nicht zwischen der Wirkung von RESET und RSTØ unterschieden werden.

Durch ggf. Kombination von Hardware- und Softwaremaßnahmen können Prioritäten gesetzt werden für unterschiedliche Unterbrechungsanforderungen. Einzelheiten werden hier nicht besprochen. Die Art der Prioritäten entscheidet mit über die Frage, ob eine Unterbrechungsanforderung angenommen wird.

Einige weitere Einzelheiten zum Ablauf der Unterbrechung:
Die Anforderung erfolgt durch ein Signal auf den Kontakt INT des Mikroprozessors. Wird die Unterbrechung angenommen, so setzt der Mikroprozessor das erwähnte 1 Bit-Register zurück und verhindert damit zunächst eine Unterbrechung der Unterbrechung. Er gibt im nächsten Maschinenzyklus ein spezielles Steuersignal INTA, „interrupt acknowledge" (= Anerkennung der Unterbrechung) auf den Datenbus (der anfangs noch als „Steuerbus" tätig ist, siehe Abschnitt 5.1) und erwartet einen Befehl. Das Gerät muß dafür sorgen, daß ein RST-Befehl auf den Datenbus geht. Im RST-Unterprogramm kann der Programmierer durch EI dafür sorgen, daß ggf. doch eine Unterbrechung höherer Priorität die erste Unterbrechung ihrerseits unterbrechen kann.

Das RST-Unterprogramm beginnt gewöhnlich mit einigen PUSH-Befehlen zur Rettung der Registerinhalte. Dann folgt die vom Gerät benötigte Aktivität, z. B. Entgegennahme und Prüfung von Daten. Zum Schluß vor dem Rückkehrbefehl werden die Registerinhalte durch POP-Befehle wieder dem Kellerspeicher entnommen; ferner wird (sofern nicht schon früher geschehen) wieder EI angeordnet.

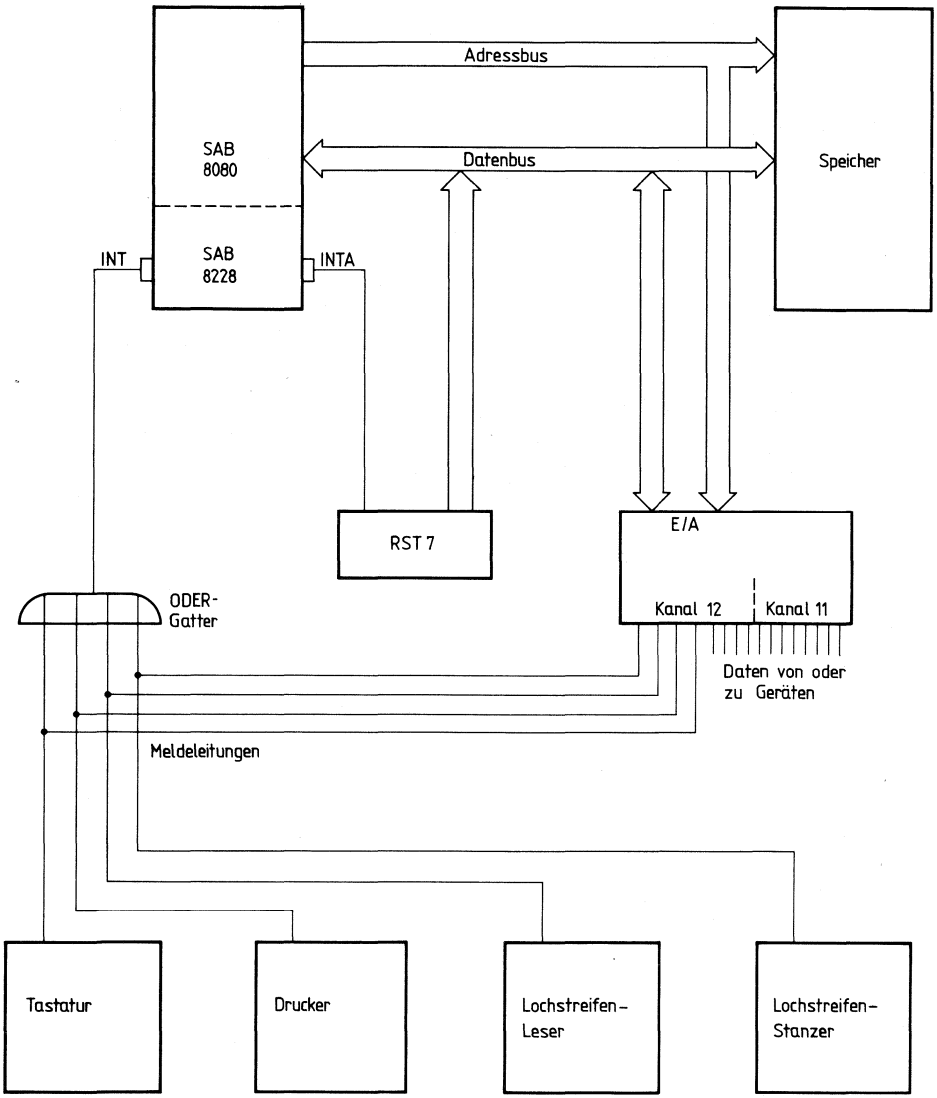


Bild 6.11
Einfache Unterbrechungsschaltung

Ein relativ einfaches Beispiel zeigt Bild 6.11.
Hier sind 4 Eingabe- und Ausgabegeräte dargestellt. Ihre Datenausgänge oder -eingänge sind nicht gezeichnet. Dafür sind Meldeleitungen gezeichnet, mit denen jedes Gerät eine Unterbrechnung anfordern kann. Über ein ODER-Gatter geht die Anforderung an den Mikroprozessor, wenn mindestens ein Gerät sich meldet. Das Signal INTA, „interrupt acknowledge" (= Unterbrechungs-Anerkennung) ist genau genommen das Steuersignal, das über den Datenbus ausgegeben wird. Zur besseren Übersicht wird hier dafür eine gedachte Steuer-Einzelleitung benutzt. Sie führt zu einem speziellen Baustein RST 7, in dem das Bitmuster dieses Befehles gespeichert ist; auf INTA hin gibt er das Bitmuster auf den Datenbus.
Die Meldeleitungen führen außerdem zum Eingabekanal 12. Die Nummern der anderen Eingabe- und Ausgabekanäle, über die Daten herein- und hinausgehen können, treten, wie gesagt, in diesem Beispiel nicht direkt auf.

Im folgenden Programmbeispiel (für das Unterprogramm ab Adresse 56 = 8 × 7) treten dafür symbolische Adressen für einige Unter-Unterprogramme zwecks Benutzung der verschiedenen Geräte auf.

```
0038  F5       PUSH  PSW    ;AKKU UND ZUST.BITS RETTEN.
0039  DB0C     IN    12     ;MELDELEITUNGEN ABHOLEN.
003B  07       RLC          ;HOECHSTES BIT PRUEFEN.
003C  DC1123   CC    LOSTA  ;WENN SIGNAL, U.PROGRAMM
                            ;/FUER LOCHSTREIFENSTANZER.
003F  07       RLC          ;NAECHSTES BIT.
0040  DC4223   CC    TASTA  ;TASTATUR.
0043  07       RLC          ;NAECHSTES BIT.
0044  DC5823   CC    LOLES  ;LOCHSTREIFENLESER.
0047  07       RLC          ;LETZTES BIT.
0048  DC7323   CC    DRUCK  ;DRUCKER.
004B  F1       POP   PSW    -PSW REGENERIEREN.
004C  FB       EI           ;NEUE UNTERBR. ERLAUBT.
004D  C9       RET          ;FORTSETZUNG DES UNTER-
                            ;/BROCHENEN PROGRAMMS.
```

Die beschriebene Methode hat den Vorteil, daß der Programmierer durch die Reihenfolge in der Abfrage der Meldeleitungen (nicht notwendigerweise mit RLC-Befehlen) in jedem Programmstadium die relative Wichtigkeit der Meldungen festlegen kann (Prioritätsvergabe per Software).

7. Weitere Programmbeispiele

Erstes Beispiel: Der Inhalt von 100 aufeinanderfolgenden RAM-Plätzen soll gelöscht werden. Der erste dieser Plätze hat die Adresse A200 H. Den Programmablaufplan zeigt Bild 7.1.

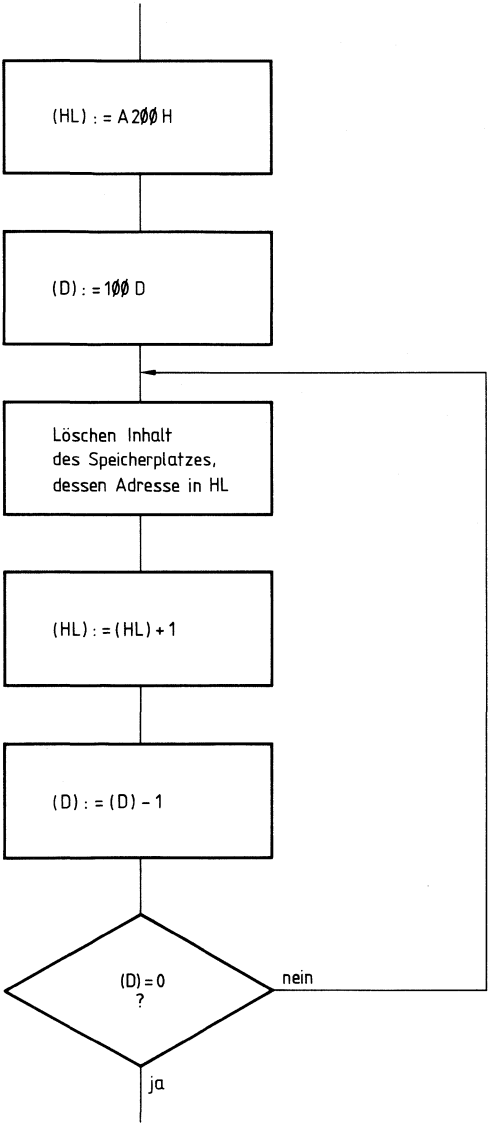


Bild 7.1
Zum ersten Programmbeispiel

```
3AAE    2100A2          LXI     H,0A200H
3AB1    1663            MVI     D,100D
3AB3    3600    LOESC:  MVI     M,0
3AB5    23              INX     H
3AB6    15              DCR     D
3AB7    C2B33A          JNZ     LOESC
```

Im Registerpaar HL steht jeweils die Adresse, deren Inhalt gelöscht wird. Dies ist ein Paradebeispiel für den Nutzen der indirekten Registeradressierung (Abschnitt 4.3). Das Register D dient als Zählwerk. Der Befehl DCR r beeinflußt das Nullbit; hierdurch wird die Prüfung mit JNZ ermöglicht. Wenn (D) = 0 geworden ist, wird der Befehl JNZ ignoriert, und es geht weiter im Programm.

Zweites Beispiel: Auf dem RAM-Platz C512 H steht ein Steuerwort, in dem nur ein Bit = 1 ist. Je nachdem, welches Bit dies ist, soll in unterschiedliche Programmteile gesprungen werden. Die Anfangsadressen dieser Programmteile stehen auf den ROM-Plätzen 0F00 H und 0F01 H (für Bit Nr. 0), 0F02 H und 0F03 H (für Bit Nr. 1), . . ., 0F0E H und 0F0F H (für Bit Nr. 7). Die Aufgabe besteht darin, die richtige (hier nicht genannte) Startadresse in den Befehlszähler zu bringen. Den Programmablaufplan zeigt Bild 7.2.

```
1001    3A12C5          LDA     0C512H
1004    21000F          LXI     H,0F00H
1007    1F      ROTA:   RAR
1008    DA1010          JC      ZIEL
100B    23              INX     H
100C    23              INX     H
100D    C30710          JMP     ROTA
1010    5E      ZIEL:   MOV     E,M
1011    23              INX     H
1012    56              MOV     D,M
1013    EB              XCHG
1014    E9              PCHL
```

Drittes Beispiel: Suche nach dem obersten RAM-Platz (um den Stapelzeiger zu setzen, wenn der ROM auf den höchsten Adressen liegt).
Man beginnt bei der höchstmöglichen Adresse und versucht, ein Datenwort (hier 55 H) einzuschreiben. Anschließend wird der Inhalt herausgelesen. Wenn es ein RAM-Platz ist, muß man dabei dasselbe erhalten, was eingeschrieben wurde. Bei einem ROM-Platz kann man nicht einschreiben. Damit nicht zufällig der Inhalt eines ROM-Platzes getroffen wurde, wird anschließend etwas anderes (hier AA H) eingeschrieben. Wird auch dann dasselbe ausgelesen, so hat man den obersten RAM-Platz gefunden. Außer RAM- und ROM-Platz gibt es noch die Möglichkeit, daß gar kein Speicherplatz mit der betreffenden Adresse vorhanden ist. Das Einschreiben hat dann keine Wirkung. Beim Versuch, aus der nicht vorhandenen Adresse zu lesen, erhält man immer „FF H", also etwas anderes als eingeschrieben. Ein fehlender Platz wirkt also wie ein ROM-Platz.

Die Adressen werden immer um 0100 H = 256 D dekrementiert, weil kein Speicherbaustein für den SAB 8080 weniger Plätze umfaßt. Vorausgesetzt muß werden, daß auch wirklich ein RAM vorhanden ist; sonst würde das Programm „hängenbleiben": Nach dem Platz mit der Adresse 00FF H würde immer wieder der Platz FFFF H untersucht.

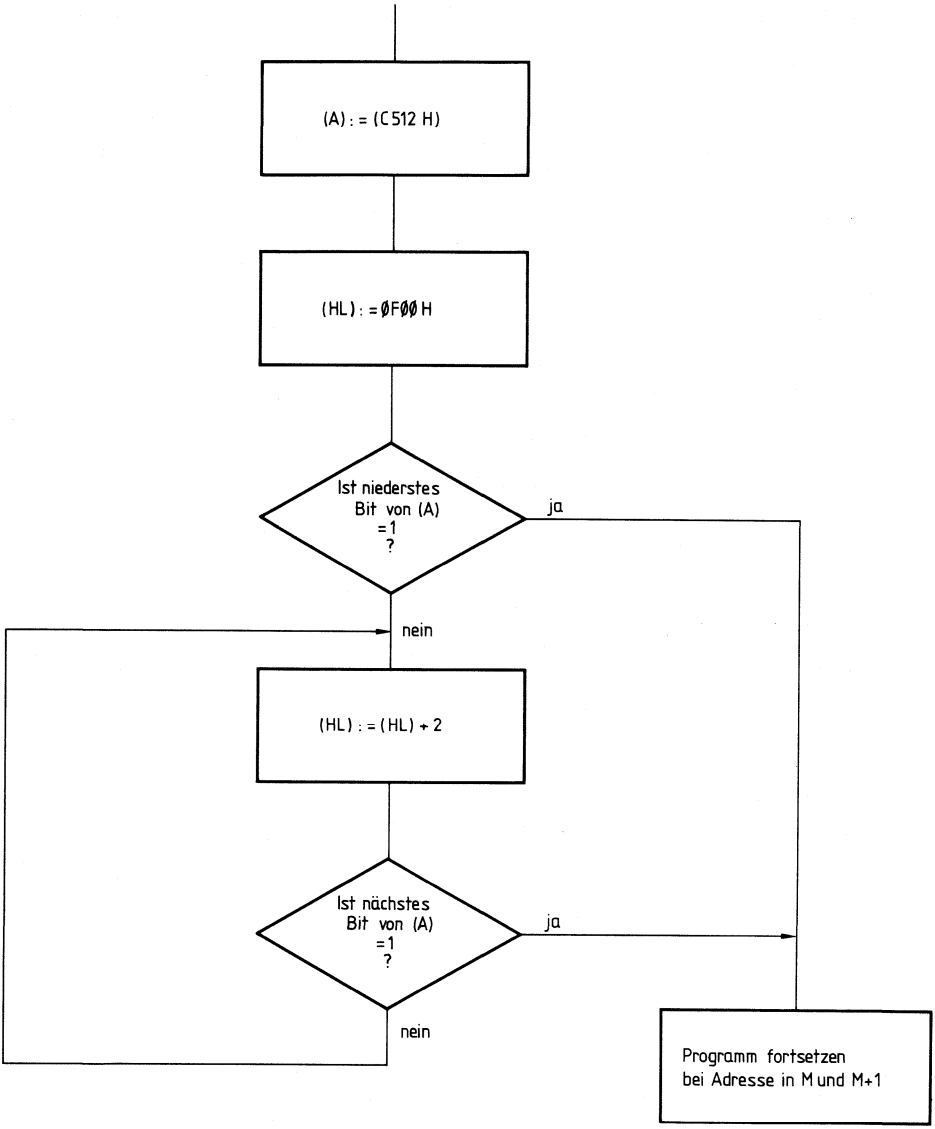


Bild 7.2
Zum zweiten Programmbeispiel

Auch dies ist ein Beispiel für indirekte Adressierung.

```
134D    21FFFF          LXI     H,ØFFFFH
135Ø    Ø655            MVI     B,55H
1352    7Ø      CONTI:  MOV     M,B
1353    7E              MOV     A,M
1354    B8              CMP     B
1355    CA5C13          JZ      EQUAL
1358    25      DECRE:  DCR     H
1359    C35213          JMP     CONTI
135C    2F      EQUAL:  CMA
135D    77              MOV     M,A
135E    4E              MOV     C,M
135F    B9              CMP     C
136Ø    C25813          JNZ     DECRE
1363    F9              SPHL
```

Viertes Beispiel: Multiplikation zweier 8-Bit-Zahlen. Die gewohnte handschriftliche Multiplikation von Dezimalzahlen besteht aus folgenden Einzelschritten: Nacheinander wird die eine Zahl (der Multiplikand) mit den Ziffern der anderen Zahl (des Multiplikators) multipliziert; die Resultate werden zwecks Addition jeweils um eine Ziffer versetzt untereinandergeschrieben. Dieses Verfahren ist bei Binärzahlen ebenfalls anwendbar und sogar noch einfacher, weil die Ziffern des Multiplikators nur 0 oder 1 sein können. Nach diesem Schema werden im Folgenden zwei 8 Bit-Zahlen multipliziert.
Der Multiplikator sei auf dem RAM-Platz FØØ1 H und der Multiplikand auf dem RAM-Platz FØØ2 H gespeichert.

```
1ØAA    3AØ2FØ          LDA     ØF002H
1ØAD    5F              MOV     E,A
1ØAE    16ØØ            MVI     D,Ø     ;MULTIPLIKAND IN RPD.
1ØBØ    3AØ1FØ          LDA     ØF001   ;MULTIPLIKATOR IN RA.
1ØB3    21ØØØØ          LXI     H,Ø     ;RPH FUER RESULTAT.
1ØB6    Ø6Ø8            MVI     B,8     ;RB ALS SCHRITTZAEHLER.
1ØB8    29      SHIFT:  DAD     H       ;ZEILENVERSCHIEBUNG.
1ØB9    17              RAL             ;MULTIPL.ATOR-BITPRUEFG.
1ØBA    D2BE1Ø          JNC     WEITR   ;  WENN BIT = 1,
1ØBD    19              DAD     D       ;  WIRD MULT.ND ADDIERT.
1ØBE    Ø5      WEITR:  DCR     B       ;ZUR NAECHSTEN ZEILE
1ØBF    C2B81Ø          JNZ     SHIFT
1ØC2    225643          SHLD    4356H   ;RESULTAT GESPEICHERT.
```

Dies bedarf einer ausführlichen Erläuterung. Zuvor ist noch eine Bemerkung angebracht.
Beim Rechnen auf dem Papier berechnet man nacheinander die Teilprodukte „Multikand mal eine Ziffer des Multiplikators"; die Teilprodukte werden immer eine Ziffer versetzt untereinandergeschrieben und erst zum Schluß addiert. Für den Computer ist es bequemer, jedes Teilprodukt (entsprechend versetzt) sofort hinzuzuaddieren. Diese Zwischensumme steht hier jeweils im Registerpaar HL; nach dem letzten Schritt ist es das Endresultat.
Bei den Zwischenadditionen muß immer wieder der Multiplikand zum Inhalt von HL addiert werden; deshalb wird der Multiplikand in das Registerpaar DE gelegt; genauer: Er wird in das Register E gelegt und Register D gelöscht. Die Versetzung um eine Ziffer wird im Registerpaar HL vorgenommen. Nun zur eigentlichen Erläuterung der Multiplikation.

Die ersten 3 Befehle holen den Multiplikanden in das Registerpaar DE. Danach wird der Multiplikator in den Akku geholt. Registerpaar HL wird gelöscht und Register B als Zählwerk mit 8 geladen, weil der Multiplikator 8 Bits hat.
Der weitere Gang der Rechnung soll an einem konkreten Fall erläutert werden.
Der Multiplikator sei 10101010 B = 170 D, der Multiplikand sei 01010101 B = 85 D.
Die handschriftliche Multiplikation sieht dann so aus:

```
                 01010101 × 10101010
1)               01010101
2)                00000000
3)                 01010101
4)                  00000000
5)                   01010101
6)                    00000000
7)                     01010101
8)                      00000000
                 011100001110010 B = 14450 D
```

Nun zum Programmbeispiel. Bevor DAD H zum ersten Mal eingeholt wird, hat man die Registerinhalte

(B) 00001000 (A) 10101010 (HL) 00000000 00000000

DAD H verlangt eine Verdoppelung des Inhaltes von HL; dies ist noch ohne Bedeutung, aber das Übertragsbit wird zurückgesetzt, weil kein Übertrag entsteht. Das gilt für jeden der 8 Schritte. Infolgedessen werden durch RAL von rechts immer Nullen nachgeschoben (was aber für das Resultat keine Bedeutung hat). Der Zweck von RAL ist, die Bits des Multiplikators von oben herunter durchzuprüfen.
Das höchste Bit des Multiplikators ist 1. Deshalb wird JNC WEITR nicht ausgeführt, sondern der Multiplikand durch DAD D in das Registerpaar HL hineinaddiert; dies entspricht der ersten Zeile der handschriftlichen Rechnung oben.
Nun wird das Register B um 1 heruntergezählt und, da dies noch nicht 0 ergibt, wieder zu SHIFT gesprungen. Die Verdoppelung des Inhaltes von HL bedeutet eine Linksverschiebung um eine Ziffer; hierdurch wird die Verschiebung zwischen erster und zweiter Zeile im handgerechneten Fall hergestellt.
Der Sprung nach WEITR wird diesmal ausgeführt, da das Bit Nr. 6 = 0 ist. Diesmal wird also nichts zum Inhalt von HL addiert. So geht es weiter. Hier folgt der Inhalt der Register, jeweils vor der Dekrementierung von Register B:

(B)	(A)	(HL)
00001000	01010100	00000000 01010101
00000111	10101000	00000000 10101010
00000110	01010000	00000001 10101001
00000101	10100000	00000011 01010010
00000100	01000000	00000110 11111001
00000011	10000000	00001101 11110010
00000010	00000000	00011100 00111001
00000001	00000000	00111000 01110010

Bei der letzten Dekrementierung wird der Inhalt von B zu 0. Im Registerpaar HL steht das Resultat (siehe bei der schriftlichen Rechnung).

Fünftes Beispiel: Eine zweistellige Dezimalzahl wird an einem Eingabegerät willkürlich eingestellt. Sie liegt in BCD-Form (siehe Abschnitt 4.7.2) am Eingabekanal 1. Am Ausgabekanal 2 liegt eine Anzeigevorrichtung für zweistellige Dezimalzahlen. Sie soll von der eingestellten Zahl aufwärts bis 99 hochzählen und dabei jede Zahl etwa 0,5 Sekunden lang zeigen. Danach soll wieder bei der eingestellten Zahl begonnen werden.

Das Abzählen der gewünschten Zeitspanne erfolgt wie in Abschnitt 6.1 (nahe dem Schluß) beschrieben durch eine Doppelschleife. Nach den 0,5 Sekunden wird jeweils der Akkuinhalt inkrementiert. Es muß dafür gesorgt werden, daß nach dem Erreichen von 99 jeweils zurückgesprungen wird. Hierzu wird nach der Inkrementierung DAA (Abschnitt 4.7.2) angeordnet. Die Zahl 99 wird in BCD durch 10011001 ausgedrückt. Nach der Inkrementierung wird daraus 10011010.
Die DAA-Operation addiert infolgedessen erst 00000110 und dann 01100000 zum Akkuinhalt; es ergibt sich 00000000, also die in BCD ausgedrückte Dezimalzahl 00.
Die Befehlsfolge lautet:

```
1233    B7                  ORA   A        ;Rücksetzen von ÜB u. HÜB.
1234    DBØ1      INPUT:    IN    1        ;Eingabeschalter.
1236    D3Ø2      OUPUT:    OUT   2
1238    1EFF                MVI   E,ØFFH   ;Anzeige.
123A    16FF      WAIT1:    MVI   D,ØFFH   ;Zeitschleife ~ 0,5s.
123C    15        WAIT2:    DCR   D
123D    C23C12              JNZ   WAIT2
124Ø    1D                  DCR   E
1241    C23A12              JNZ   WAIT1
1244    3C                  INR   A        ;Erhöhen des Zählers.
1245    27                  DAA
1246    FEØØ                CPI   Ø        ;vgl. dez. 100.
1248    C23612              JNZ   OUPUT
124B    C33412              JMP   INPUT
```

Der Befehl INR A beeinflußt das Hilfsübertragsbit. Es wird immer zurückgesetzt: Die zu inkrementierende niederwertige Hälfte des Akkuinhaltes ist nie größer als 1001. Die anschließende DAA-Operation findet den korrekten Wert des Hilfsübertragsbits vor (Einzelheiten sollen hier nicht besprochen werden). Dagegen kann beim ersten Durchlauf das Übertragsbit vom früheren Programmteil noch zufällig gesetzt sein; es wird ja durch die Transferbefehle und durch DCR und INR nicht beeinflußt. Daher wird es zu Beginn durch ORA A zurückgesetzt. Danach ist dies nicht mehr nötig, weil CPI Ø keinen Übertrag erzeugt und daher das Übertragsbit immer zurücksetzt.
Ohne ORA A würde u.U. gerade durch DAA ein sinnloses Resultat erzeugt werden (wieder ohne Angabe von Einzelheiten).
Der Vollständigkeit halber sei bemerkt, daß der Transferbefehl POP PSW das Übertragsbit durchaus beeinflussen würde; siehe hierzu Tabelle 6-1.

8. Herstellung von Programmen

8.1. Entwicklungssystem

Programme werden normalerweise in ROMs eingeschrieben. Unmittelbar nach dem Herstellen eines Programmes ist es aber zweckmäßig, das Programm zuerst in einen RAM einzuschreiben. Man findet nämlich regelmäßig noch Fehler; das Ändern einzelner Stellen wird durch Speichern in einem RAM sehr erleichtert.

Kapitel 8 befaßt sich mit den Arbeiten, die bis zum Testen eines Programmes im RAM zu leisten sind, insbesondere mit der Erleichterung dieser Arbeiten durch Computerhilfe.

Keine Computerhilfe gibt es für das früheste Stadium: Der Programmierer überlegt, wie er die gestellte Aufgabe lösen will. Hierzu stellt er sich z. B. Programmablaufpläne her. Für die weiteren Arbeiten ist eine Vorentscheidung zu treffen: Man benutzt entweder eine „problemorientierte" oder eine „maschinenorientierte" Programmiersprache.
Bei Benutzung einer problemorientierten, „höheren" Programmiersprache braucht man keine Notiz zu nehmen vom Befehlsvorrat des Computers, auf dem das Programm ablaufen soll.
Bei Benutzung einer maschinenorientierten Programmiersprache muß man den Befehlsvorrat kennen; für den SAB 8080 kann hierzu z. B. die vorliegende Schrift verhelfen. Die maschinenorientierte Programmiersprache heißt **Assemblersprache**; ein Teil dieser Sprache sind die mnemonischen Abkürzungen.

Der Vorteil der höheren Programmiersprachen (z. B. FORTRAN, COBOL, PL/1 für große Computer; BASIC, PL/M für Mikrocomputer) besteht in einem wesentlich geringeren Zeitaufwand des Programmierers. Der Vorteil der Assemblersprache liegt (jedenfalls bei nicht zu langen Programmen) in der wirtschaftlicheren Nutzung von Speicherraum und Arbeitszeit des Computers. Es hängt von den speziellen Bedingungen ab und ist z. T. umstritten, wo die Grenze liegt zwischen überwiegendem Nutzen der einen oder anderen Methode.
In der vorliegenden Schrift wird konsequenterweise das Arbeiten mit Assemblersprache beschrieben; der Leser hätte sich sonst auch den größeren Teil der geleisteten Arbeit sparen können.

Dann kann man die Einzelarbeiten nach der ersten Planung des Programmes wie folgt einteilen:
1. Aufstellen der Befehlsreihenfolge für das Programm mit Hilfe mnemonischer Abkürzungen und symbolischer Adressen.
2. Übersetzung der mnemonischen Abkürzungen in die Maschinensprache sowie Verwaltung und Zuweisung der Absolut-Adressen. Dieser Vorgang heißt **Assemblieren.**
3. Einschreiben des Programmes in einen RAM, Testen und Korrigieren des Programmes.

Jede dieser Arbeiten kann durch Computerunterstützung erleichtert werden. Hierzu dienen die folgenden Hilfsprogramme:
Zu 1. Das Editorprogramm (spezielle Bezeichnung beim SAB 8080; allgemein z. B. „Listenverarbeiter").
Zu 2. Das Assemblerprogramm.
Zu 3. Das Monitorprogramm (allgemein z. B. „Ablaufteil").
Solche Hilfsprogramme können im Prinzip auf jedem Computer ablaufen. Hier wird der Fall behandelt, in dem die Hilfsprogramme ebenfalls auf Computern der Familie SAB 8080 ablaufen. Solche Computer werden dann als **Entwicklungssysteme**

bezeichnet; die Hilfsprogramme heißen dann **residente** Editor-, Assembler- und Monitorprogramme. Die vorliegende Schrift kann nicht alle Einzelheiten dieser Hilfsprogramme behandeln. Es sollen aber einige Bemerkungen zur allgemeinen Arbeitsweise gemacht werden.

Die meisten Arbeiten der obigen Aufstellung kann man auch ohne Hilfe eines Computers durchführen; man braucht dann „nur" mehr Zeit dazu. So kann die Aufstellung der Befehlsreihenfolge schriftlich geschehen. Die Übersetzung in Maschinensprache kann ebenfalls mit Hilfe einer Tabelle schriftlich durchgeführt werden. Auch die Absolutadressen kann der Programmierer durch geeignete Buchführung ermitteln. Er kann auch das Programm mit Hilfe von Tasten und Hexadezimalschaltern in einen RAM einschreiben; solche Möglichkeiten bestehen bei den weit verbreiteten Lerncomputern, meist in Verbindung mit Anzeige durch Lumineszenzdioden.

Nur das Testen des Programmes läßt sich nicht gut ohne Hilfe eines Computers durchführen. Damit ist nicht die triviale Tatsache gemeint, daß zum Ablauf eines Programmes eben ein Computer nötig ist; vielmehr wird noch mehr Hilfe benötigt.
Man kann nämlich aus dem einfachen Ablauf eines ganzen Programmes nicht erkennen, an welcher Stelle sich Programmfehler befinden. Um Fehler zu finden, muß man sie einkreisen. Man muß die Möglichkeit haben, das zu testende Programm in abgegrenzten Portionen ablaufen zu lassen. Ferner muß man die Möglichkeit haben, die dann vorhandenen Register- und Speicherinhalte zu inspizieren.
Diese Möglichkeit zu schaffen, ist die wichtigste Aufgabe des **Monitorprogrammes** („der Monitor", hat nichts zu tun mit dem gleichnamigen Kontroll-Bildschirm).

Wenn man aber schon ein Monitorprogramm zum Testen schafft, dann gibt man ihm zweckmäßigerweise auch gleich die Möglichkeit, den RAM-Inhalt zu ändern; ferner die Möglichkeit, von Eingabegeräten (z. B. Lochstreifenleser oder Lesevorrichtung für Tonband) in den RAM zu schreiben sowie aus dem RAM auf Ausgabegeräte (z. B. Drucker) auszugeben.
Damit kann das Monitorprogramm sämtliche Aufgaben zu Punkt 3 der obigen Liste ausführen: Einschreiben, testen und korrigieren.

Die Notwendigkeit eines **Assemblerprogrammes** zur Erledigung der Arbeiten zu Punkt 2 ist nicht so prinzipieller Natur. Trotzdem ist das Assemblerprogramm („der Assembler") in der Praxis unentbehrlich. Nur bei extrem kurzen Programmen (etwa unter 100 Bytes) ist es zumutbar, die Befehlsliste in Maschinensprache (hexadezimal stenografiert) von Hand aufzustellen. Aber selbst in solchen Fällen benutzt man dankbar die Assemblierungsmöglichkeit.

Wenn man ein Entwicklungssystem mit Monitor und Assembler hat, dann bietet das **Editorprogramm** („der Editor") einen zwar nicht unentbehrlichen, aber sehr angenehmen Komfort. Das Editorprogramm dient der Herstellung von Text im weitesten Sinne; hier wird es speziell zur Erledigung der Arbeiten zu Punkt 1 der obigen Aufstellung verwendet.

Weitere Möglichkeiten eines Entwicklungssystemes wie z. B. das Testen von Hardwareeigenschaften liegen außerhalb des Rahmens dieser Schrift.
Im Entwicklungssystem muß der ROM nicht die untersten Adressen belegen; der als Entwicklungssystem bezeichnete Mikrocomputer wird ja auch nicht nur von einem Programm gelenkt, sondern abwechselnd von den verschiedenen Hilfsprogrammen und auch von dem zu testenden Programm.
Man speichert nur das Monitorprogramm in einem ROM und gibt ihm z. B. die höchsten Adressen, z. B. F8ØØ H bis FFFF H. Will man in das Monitorprogramm eintreten, so sorgt man durch Hardwaremittel dafür, daß der Befehlszähler des Mikroprozessors die Anfangsadresse des Monitors bekommt.
Die anderen Hilfsprogramme und das herzustellende und zu testende Programm werden jeweils von Lochstreifen oder Magnetplatte in den RAM übertragen.

Das Assemblieren eines Programmes erfolgt weitgehend automatisch. Dagegen befindet sich der Bearbeiter beim Benutzen des Monitors oder des Editors „im **Dialog** mit dem Computer." Als Mittel zum Dialog dient eine Eingabetastatur und eine Ausgabemöglichkeit entweder durch Fernschreibmaschine oder durch Datensichtgerät. Die Eingabedaten von der Tastatur bezeichnet man gelegentlich als „Befehle" an Monitor oder Editor, besser jedoch als Anweisungen, weil dies keine Befehle im Sinne von Kapitel 4 sind.

Als Beispiel für eine solche **Anweisung,** und zwar an den Monitor, soll hier dienen

$$G\ Adr_1,\ Adr_2,\ Adr_3.$$

„G" steht für „go" (= gehe). Hierdurch wird der Ablauf eines Teilstückes des zu testenden Programmes veranlaßt.
Die erste Adresse ist die (hexadezimal stenografierte) Adresse desjenigen Operationscodes, mit dem begonnen werden soll. Wird sie weggelassen, so beginnt der Ablauf dort, wo man zuletzt stehengeblieben war. Die zweite und ggf. dritte Adresse gibt denjenigen Operationscode, der als erster nicht mehr beachtet werden soll. Zwei solcher Adressen gibt man an, wenn im zu testenden Teilstück ein bedingter Sprung vorkommt und man nicht weiß, wie es dort weitergehen wird. Der Monitor gibt nach Ablauf des Teilstückes in jedem Fall den letzten Stand des Befehlszählers aus. Im Fall des bedingten Sprunges hat man damit automatisch die wertvolle Information, ob der Sprung ausgeführt wurde oder nicht.

Im Zeitpunkt, zu dem man den Stand des Befehlszählers erfährt, steht aber in Wirklichkeit im Befehlszähler schon wieder eine Adresse des Monitorprogrammes; denn zu diesem Zeitpunkt beherrscht den Mikroprozessor wieder der Monitor. Das Gleiche gilt, wenn man durch eine andere Anweisung nach dem Inhalt anderer Register fragt. Das Monitorprogramm rettet immer bei Übernahme des Mikroprozessors dessen letzten Registerinhalt in den Kellerspeicher; von dort kann diese Information bei entsprechender Anweisung herausgeholt werden.

Die Rückkehr in das Monitorprogramm nach Ablauf eines Teilstückes des zu testenden Programmes wird wie folgt erreicht. Nach der G-Anweisung rettet der Monitor den Inhalt der Schluß-Adresse(n) des Teilstückes in den Kellerspeicher und ersetzt ihn durch einen unbedingten Sprungbefehl in das Monitorprogramm. Nach dem Sprung wird der alte Inhalt der Schluß-Adresse(n) wieder hergestellt.

Das **Editorprogramm** hat manche Aufgaben mit dem Monitorprogramm gemeinsam, nämlich das Ändern von RAM-Daten im weitesten Sinne. Der Unterschied besteht darin, daß der Monitor hierzu die Adressen der Speicherplätze braucht; der Editor reagiert dagegen auf Zeichenfolgen, also auf den Inhalt von Speicherplätzen. Andere Anweisungen veranlassen den Editor zur Aktivität an derjenigen Speicherstelle, auf die ein gedachter Zeiger („Pufferzeiger") zeigt; dieser Zeiger kann willkürlich gesetzt werden. Mit dem Editor kann man Zeichen oder Zeilen so löschen, daß der übrige Text automatisch nachrückt. Man kann auch an beliebiger Stelle Text einfügen.

Das **Assemblerprogramm** nimmt, wie erwähnt, die Übersetzung vor aus mnemonischen Abkürzungen mit symbolischen Adressen in die Maschinensprache mit Absolutadressen. In Abschnitt 4.1 (Bild 4.2) wurde bereits das **Assembler-Listing** vorgestellt. Eine Voraussetzung für das korrekte Arbeiten des Assemblerprogrammes ist das Einhalten gewisser formaler Vorschriften durch den Programmierer. Diese Vorschriften werden hier nicht erörtert.
Wichtiger ist: Der Programmierer schreibt das Quellenprogramm (Bild 4.2) in **Assemblersprache.** Die Assemblersprache besteht aus zwei Arten von Vokabeln. Die eine Art sind die **mnemonischen Abkürzungen.** Die zweite Art von Vokabeln der Assemblersprache ist in dieser Schrift noch nicht vorgekommen. Es handelt sich um Anweisungen für den Assemblierungsvorgang; man bezeichnet sie als **Pseudobefehle.** Der Grund für diese Bezeichnung ist: Während die echten Befehle nach

der Assemblierung weiterhin vorhanden sind (nur übersetzt), sind die Pseudobefehle im assemblierten Objektprogramm nicht mehr vorhanden, da sie ihren Zweck erfüllt haben. Im Assembler-Listing (dem Protokoll also) finden sie sich natürlich.

Zu den Pseudobefehlen gehören Anweisungen für die Absolutadresse des Programmanfanges, für das Ende des Assemblierens, für freizuhaltende RAM-Plätze, für in den ROM zu schreibende Konstanten-Tabellen, für Gleichsetzung einer symbolischen Abkürzung mit irgendeinem (ggf. noch offenen) Datenwort.
Ein besonders wichtiger Pseudobefehl ist die Definition (= Festlegung) eines **„Makro"**. Ein Makro hat eine gewisse Ähnlichkeit mit einem Unterprogramm; er ist ebenfalls eine häufig vorkommende Programmstelle. Aber im Gegensatz zum Unterprogramm wird er jedesmal aufs Neue in den ROM geschrieben. Man spart sich aber Schreibarbeit beim Herstellen des Quellenprogrammes, indem man ihn dort nur einmal schreibt. Danach wird der Makro im Quellenprogramm immer „aufgerufen", wenn er im Objektprogramm vorkommen soll. Dabei kann man sogar die Werte irgendwelcher Daten im Makro von Fall zu Fall neu vorschreiben.

Ein Makro wird i.a. angewendet, wenn eine Programmstelle zu kurz ist oder zu selten auftritt, um ein Unterprogramm zu rechtfertigen.
Aber auch wenn ein Unterprogramm vom Standpunkt der Speicherökonomie gerechtfertigt wäre, kann man ggf. aus anderen Gründen darauf verzichten. Das ist dann der Fall, wenn es sich um einen zeitkritischen Programmteil handelt, wenn also der Computer hinter irgendwelchen realen Ereignissen nicht in Verzug geraten darf. Dann kann man den Zeitverbrauch für Sprungbefehle sparen und dafür mehr Speicherplätze opfern.
Auch in solchen Fällen erleichtert man sich wenigstens das Schreiben des Quellenprogrammes durch einen Makro.

Im folgenden Abschnitt 8.2 wird die Entstehung eines Objektprogrammes an einem kleinen Beispiel vorgeführt.

8.2. Ein Beispiel: Geschwindigkeitsmessung

Mit diesem Beispiel soll der Gang der Projektierung bei einer Mikroprozessor-Lösung gezeigt werden. Auf die im Folgenden geschilderte Aufgabe wurde bereits am Anfang des Kapitels 4 Bezug genommen.
In einer Einbahnstraße soll die Geschwindigkeit von Fahrzeugen gemessen werden. Dazu liegen zwei Lichtschranken in definiertem Abstand über der Straße, siehe Bild 4.1. Der Mikrocomputer wartet, bis ein Fahrzeug die linke Schranke durchbricht; dann läßt er einen Zähler solange laufen, bis auch die rechte Schranke durchbrochen wird. Danach enthält der Zähler eine Zahl, die der Fahrzeit in der Meßstrecke proportional ist. Dieser Wert wird in die Geschwindigkeit umgerechnet.
Geschwindigkeiten zwischen 0 und 99 km/h werden durch eine Leuchtdiodenanzeige ausgegeben. Außerdem kann durch einen Schalter eine Geschwindigkeit (zwischen 0 und 99 km/h) vorgegeben werden, bei deren Überschreiten ein Blitzlichtgerät betätigt wird.
Es folgt der Gedankengang bei der Konzipierung.

1. Zeitbetrachtungen. Im Mikrocomputer werden wie bei jedem Computer die Befehle nacheinander abgearbeitet.
Beim 8080 A dauert die Durchführung eines Befehles im Mittel 4 µsec. Man muß prüfen, ob auch die schnellstmöglichen Vorgänge verfolgt werden können. Würde z.B. der Abstand der Lichtschranken nur 1 cm betragen, so würde ein Fahrzeug mit einer Geschwindigkeit von 100 km/h diese Strecke in 360 µsec zurücklegen. Diese Zeit wäre für eine genügende Meßauflösung zu kurz. Deshalb wird in unserem Beispiel eine Meßstrecke von 1 m zugrundegelegt.

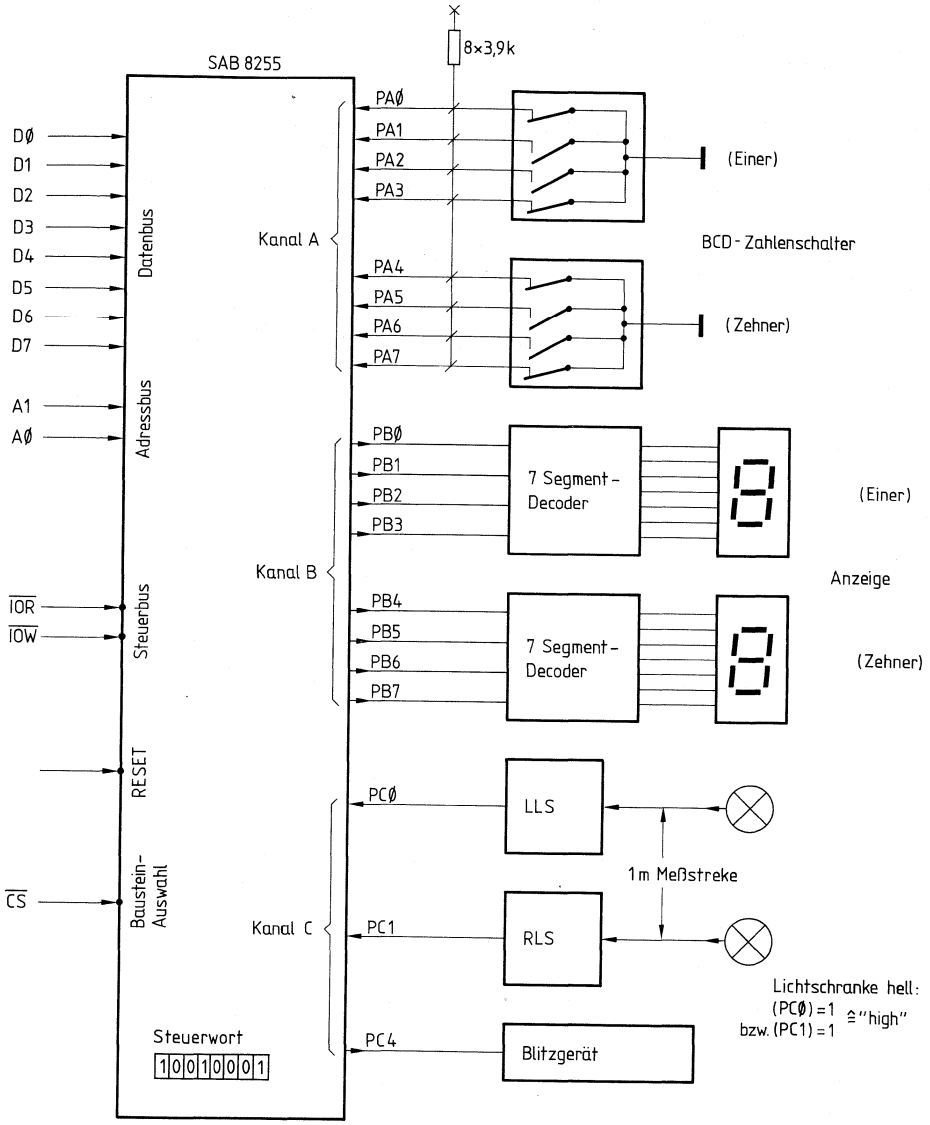


Bild 8.1
Ein- und Ausgabe für die Geschwindigkeitsmessung
Betriebsart („Mode") 0: Einfache Ein- und Ausgabe, Ausgabe gepuffert
Durch Steuerwort 91 H = 10010001 B (über Datenbus) wird dies bewirkt und außerdem:
Kanal A: Eingabe
Kanal B: Ausgabe
Kanal C, Bit 0 bis 3: Eingabe
Kanal C, Bit 4 bis 7: Ausgabe

2. **Hardware-Abschätzung.** Dabei werden die Ein- und Ausgabeleitungen gezählt, entsprechende E/A-Bausteine ausgewählt und ggf. ein Mikrocomputersystem auf einer fertigen Platine ausgesucht.

3. **Systemanalyse.** Dies bedeutet Klären der Aufgabenverteilung zwischen Hardware und Software sowie Festlegen der Schnittstelleneigenschaften; diese Festlegung umfaßt Valenz (Vorzeichen) der Signale, Logikpegel (Arbeitsspannung), Anordnung der Bits in Eingabe- und Ausgabewörtern. Hierzu zeigt Bild 8.1 eine Realisierung der Ein- und Ausgaben zur Geschwindigkeitsmessung. Verwendet wird der Eingabe- und Ausgabebaustein SAB 8255; dies ist ein häufig benutzter, vielseitig verwendbarer Baustein. Seine Arbeitsweise kann softwaremäßig so vorgeschrieben werden, wie dies am Ende des Abschnitts 5.3 geschildert wurde. Als Eingabegeräte treten hier auf Zahlenschalter und Lichtschranken, als Ausgabegeräte Anzeige und Blitzlicht.
Um den Hardwareaufwand niedrig zu halten, werden die Lichtschrankensignale durch Abfrageschleifen ausgewertet.

4. Erstellen eines **Programmablaufplanes,** siehe Bild 8.2.

5. Umsetzen (codieren) des Programmablaufplanes in ein **Quellenprogramm** (siehe Bild 4.2), „editieren" (in korrekter Form schreiben) mit Hilfe des Editors. Das Resultat zeigt Bild 8.3.

6. **Assemblieren,** d.h. Übersetzen in Maschinensprache. Der Assembler zeigt dabei formale Fehler an. Das Resultat (das Assembler-Listing) zeigt Bild 8.4.

7. Bei Fehlermeldungen des Assemblers ggf. neu editieren, d.h. Text korrigieren, und erneut assemblieren.

8. **Testen** des Programmes. Hierbei können logische Fehler entdeckt werden. Die ggf. noch nicht fertige Hardware wird dabei simuliert: Z.B. Lichtschranken ersetzen durch Tasteneingabe am Programmentwicklungsplatz.

9. **Vortesten der Hardware** durch Hilfsprogramme. Hierzu können Testprogramme dienen, die mit wenigen Befehlen die E/A-Geräte ansprechen.

10. Gemeinsames **Testen von Hard- und Software.** Hierzu gibt es eine Möglichkeit der Computerunterstützung am Entwicklungsgerät, Stichwort ETA (= Emulations-Test-Adapter).

11. Überlegungen über **Selbstkontrollen** der Geräte und über **Wartungshilfen.**

12. Protokollieren der Soft- und Hardware-**Unterlagen.**

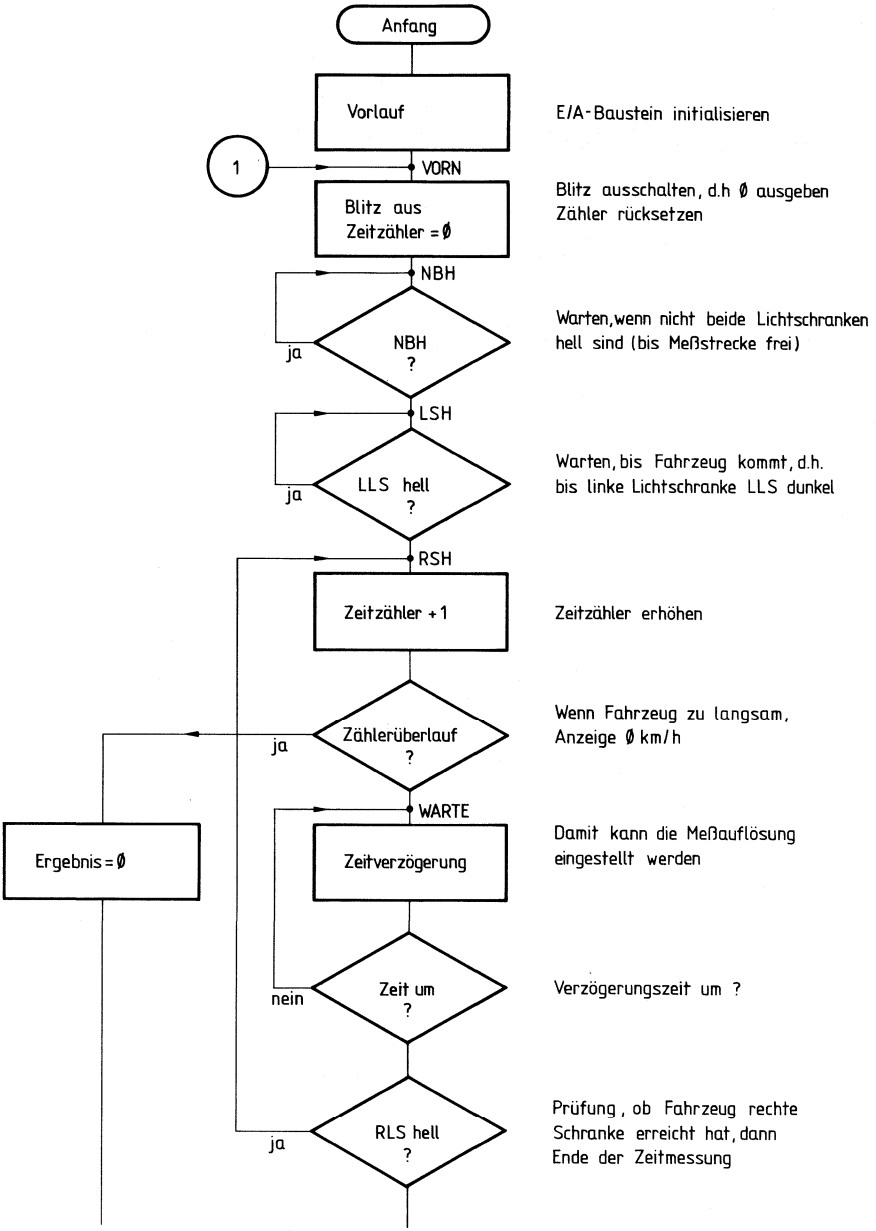


Bild 8.2/1
Programmablaufplan für die Geschwindigkeitsmessung/1. Teil

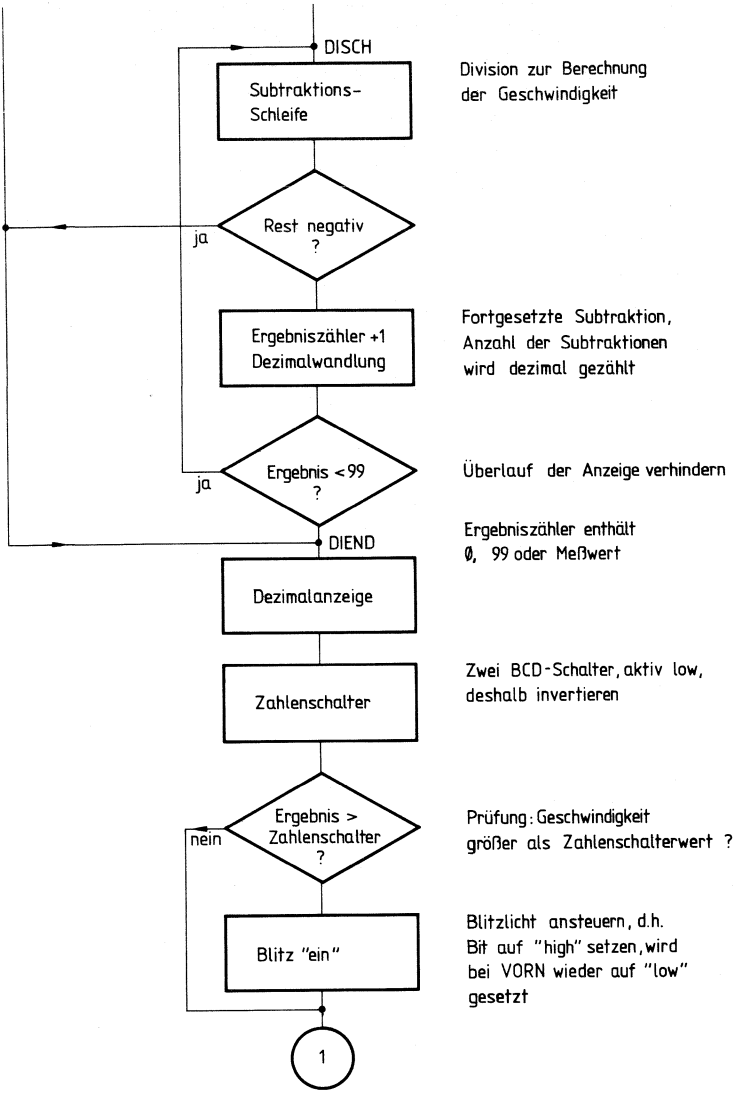


Bild 8.2/2
Programmablaufplan für die Geschwindigkeitsmessung/2. Teil

```
 ;BEISPIEL: GESCHWINDIGKEITSMESSUNG
;LS-ABSTAND 100 CM;ZAEHLEINHEIT=250MMS
        ORG 0       ;PROGRAMMANFANGSADRESSE
        MVI  A,STW ;AKN=EING,BKN=AUSG,C0-C3=EING,C4=AUSG
        OUT  STWPO ;STEUERWORT  FUER 8255
VORN:   SUB A       ;AKKU=0
        OUT BLITZ   ;BLITZ AUSSCHALTEN
        LXI H,0     ;ZAEHLER RUECKSETZEN
NBH:    IN  LIS     ;LICHTSCHRANKE HOLEN
        ANI 3
        CPI 3       ;TESTE BIT 0 UND BIT 1
        JNZ NBH     ;FAHRZEUG IN MESSSTRECKE,DANN WARTEN
LSH:    IN  LIS     ;LS HOLEN
        RRC
        JC  LSH     ;LI LS HELL,DANN WARTEN,BIS FZG KOMMT
        LXI D,1     ;D=0,E=1
RSH:    DAD D       ;ZAEHLER +1
        JC  DIEND   ;UEBERLAUF,FZG ZU LANGSAM,ANZ:0KM/H
        MVI C,29    ;ZEITSCHLEIFE,1 ZAEHLIMP.=250MMS
WARTE:  DCR C       ;DAMIT WIRD MESSAUFLOESUNG
        JNZ WARTE   ;/EINGESTELLT
        IN  LIS
        ANI 2        ;RECHTE LS NOCH HELL
        JNZ RSH      ;/DANN SPRINGE
        LXI B,CONST ;KONSTANTE AUS WEG U. KM/H-FAKTOR
DISCH:  MOV A,C      ;DIVISION,C=UNTERE HAELFTE CONST
        SUB L        ;UNTERE H. ZAEHLER
        MOV C,A
        MOV A,B      ;OBERE H. CONST
        SBB H        ;OBBERE H. ZAEHLER
        MOV B,A
        JC  DIEND    ;UNTERLAUF=DIV.ENDE
        INR D        ;ERHOEHE ERGEBNISZAEHLER D. DIVISION
        ORA A        ;CARRY 0 ,HILFSCARRY 0 SETZEN
        MOV A,D
        DAA          ;DEZIMALKORREKTUR
        MOV D,A
        CPI 99H
        JNZ DISCH    ;NOCH NICHT 99 KM/H
DIEND:  MOV A,D      ;ERGEBNIS IN AKKU,>99KM/H =99KM/H
        OUT ANZ      ;MESSWERT AUF ANZEIGE AUSGEBEN
        IN  ZASCH    ;ZAHLENSCHALTER HOLEN
        CMA          ;KOMPLEMENT DA ZAHLENSCH. NEG
        CMP D        ;VERGLEICH MIT MESSWERT
        JNC VORN     ;MESSW. WAR KLEINER,KEIN BLITZ
        MVI A,10H    ;BIT D4=1
        OUT BLITZ    ;BLITZ AUSLOESEN
        JMP VORN     ;NEUE MESSUNG BEGINNEN
;
STW     EQU 91H  ;MODE 0 ,A=EIN,B=AUS,C0-3=EIN,C4-7=AUS
STWPO   EQU 17H  ;STEUERWORTKANAL FUER 8255
LIS     EQU 16H  ;EING.KANAL C  ;C0,C1=LICHTSCHRANKEN
BLITZ   EQU 16H  ;AUSGABE KANAL C   :C4=BLITZSIGNAL
ZASCH   EQU 14H  ;EINGABE KANAL A   :ZAHLENSCHALTER
ANZ     EQU 15H  ;AUSGABE KANAL B     :MESSWERT-ANZEIGE
CONST   EQU 14400;ERMITTELT  AUS ((100CM):100000CM/KM))*
                 ;*((3600SEC/H):(250MMSEC))
        END
```

Bild 8.3
Das Quellenprogramm für die Geschwindigkeitsmessung

```
                ;BEISPIEL: GESCHWINDIGKEITSMESSUNG
                ;LS-ABSTAND 100 CM;ZAEHLEINHEIT=250MMS
0000                    ORG 0       ;PROGRAMMANFANGSADRESSE
0000 3E91               MVI  A,STW  ;AKN=EING,BKN=AUSG,C0-C3=EING,C4=AUSG
0002 D317               OUT  STWPO  ;STEUERWORT  FUER 8255
0004 97         VORN:   SUB A       ;AKKU=0
0005 D316               OUT BLITZ   ;BLITZ AUSSCHALTEN
0007 210000             LXI H,0     ;ZAEHLER RUECKSETZEN
000A DB16       NBH:    IN  LIS     ;LICHTSCHRANKE HOLEN
000C E603               ANI 3
000E FE03               CPI 3       ;TESTE BIT 0 UND BIT 1
0010 C20A00             JNZ NBH     ;FAHRZEUG IN MESSSTRECKE,DANN WARTEN
0013 DB16       LSH:    IN  LIS     ;LS HOLEN
0015 0F                 RRC
0016 DA1300             JC  LSH     ;LI LS HELL,DANN WARTEN,BIS FZG KOMMT
0019 110100             LXI D,1     ;D=0,E=1
001C 19         RSH:    DAD D       ;ZAEHLER +1
001D DA4300             JC  DIEND   ;UEBERLAUF,FZG ZU LANGSAM,ANZ:0KM/H
0020 0E1D               MVI C,29    ;ZEITSCHLEIFE,1 ZAEHLIMP.=250MMS
0022 0D         WARTE:  DCR C       ;DAMIT WIRD MESSAUFLOESUNG
0023 C22200             JNZ WARTE   ;/EINGESTELLT
0026 DB16               IN  LIS
0028 E602               ANI 2       ;RECHTE LS NOCH HELL
002A C21C00             JNZ RSH     ;/DANN SPRINGE
002D 014038             LXI B,CONST ;KONSTANTE AUS WEG U. KM/H-FAKTOR
0030 79         DISCH:  MOV A,C     ;DIVISION,C=UNTERE HAELFTE CONST
0031 95                 SUB L       ;UNTERE H. ZAEHLER
0032 4F                 MOV C,A
0033 78                 MOV A,B     ;OBERE H. CONST
0034 9C                 SBB H       ;OBBERE H. ZAEHLER
0035 47                 MOV B,A
0036 DA4300             JC  DIEND   ;UNTERLAUF=DIV.ENDE
0039 14                 INR D       ;ERHOEHE ERGEBNISZAEHLER D. DIVISION
003A B7                 ORA A       ;CARRY 0 ,HILFSCARRY 0 SETZEN
003B 7A                 MOV A,D
003C 27                 DAA         ;DEZIMALKORREKTUR
003D 57                 MOV D,A
003E FE99               CPI 99H
0040 C23000             JNZ DISCH   ;NOCH NICHT 99 KM/H
0043 7A         DIEND:  MOV A,D     ;ERGEBNIS IN AKKU,>99KM/H =99KM/H
0044 D315               OUT ANZ     ;MESSWERT AUF ANZEIGE AUSGEBEN
0046 DB14               IN  ZASCH   ;ZAHLENSCHALTER HOLEN
0048 2F                 CMA         ;KOMPLEMENT DA ZAHLENSCH. NEG
0049 BA                 CMP D       ;VERGLEICH MIT MESSWERT
004A D20400             JNC VORN    ;MESSW. WAR KLEINER,KEIN BLITZ
004D 3E10               MVI A,10H   ;BIT D4=1
004F D316               OUT BLITZ   ;BLITZ AUSLOESEN
0051 C30400             JMP VORN    ;NEUE MESSUNG BEGINNEN
                ;
0091            STW     EQU 91H  ;MODE 0 ,A=EIN,B=AUS,C0-3=EIN,C4-7=AUS
```

Bild 8.4/1
Das Assembler-Listing für die Geschwindigkeitsmessung/1. Teil

```
0017            STWPO   EQU 17H  ;STEUERWORTKANAL FUER 8255
0016            LIS     EQU 16H  ;EING.KANAL C  :C0,C1=LICHTSCHRANKEN
0016            BLITZ   EQU 16H  ;AUSGABE KANAL C  :C4=BLITZSIGNAL
0014            ZASCH   EQU 14H  ;EINGABE KANAL A  :ZAHLENSCHALTER
0015            ANZ     EQU 15H  ;AUSGABE KANAL B    :MESSWERT-ANZEIGE
3840            CONST   EQU 14400;ERMITTELT  AUS ((100CM):100000CM/KM))*
                                 ;*(((3600SEC/H):(250MMSEC))
0000                    END
```

Bild 8.4/2
Das Assembler-Listing für die Geschwindigkeitsmessung/2. Teil

```
ANZ    0015    BLITZ  0016    CONST  3840    DIEND  0043
DISCH  0030    LIS    0016    LSH    0013    NBH    000A
RSH    001C    STW    0091    STWPO  0017    VORN   0004
WARTE  0022    ZASCH  0014
```

Bild 8.4/3
Das Assembler-Listing für die Geschwindigkeitsmessung/3. Teil

Zum Programmablauf:
Beim Einschalten der Versorgungsspannung erzeugt der Mikrocomputer ein RESET-Signal (siehe Abschnitt 2.4), das den Befehlszähler des Mikroprozessors auf 0000 setzt. Mit dem Operationscode auf dem Speicherplatz Null beginnt das Programm. Im Programmvorlauf wird der E/A-Baustein SAB 8255 „initialisiert"; d.h. er erhält Steuerinformation, damit er weiß, welche Anschlüsse als Eingaben und welche als Ausgaben arbeiten sollen. Einzelheiten hierzu finden sich im Datenbuch „Siemens Mikroprozessor Bausteine System SAB 8080".
Dann wird das Blitzlichtgerät ausgeschaltet (Signal Null vom Akku ausgegeben) und der Zeitzähler auf Null gesetzt. Um eine Fehlmessung auszuschließen, wird vor Beginn jeder Messung geprüft, ob sich auf der Meßstrecke nicht schon ein Fahrzeug befindet (Schleife NBH).
Sind beide Lichtschranken hell, so wird auf ein Fahrzeug gewartet. Die Zeitdauer der Abfrageschleife LSH begrenzt die Genauigkeit der Messung.
Durchfährt ein Fahrzeug die linke Lichtschranke, so beginnt die Zählschleife RSH. Sie wird duchlaufen, bis entweder die rechte Lichtschranke durchfahren wird oder der Zähler überläuft, weil das Fahrzeug zu langsam ist. Auch die Dauer dieser Schleife begrenzt die Meßgenauigkeit. Mit der Verzögerungsschleife WARTE kann man die Meßauflösung einstellen. Die Zeitverzögerung kann berechnet werden wie in den letzten Beispielen des Abschnitts 6.1. Die Umrechnung des Meßwertes im Registerpaar HL in km/h erfolgt durch eine Division mit Dezimalkorrektur. Die Division erfolgt hier durch fortgesetzte Subtraktion; man zählt die Anzahl der Subtraktionen, bis das Resultat negativ wird, oder bis das Resultat größer als 99 wird. Im letzteren Fall wird 99 ausgegeben. Die Konstante, von der subtrahiert wird, errechnet sich aus der Länge der Meßstrecke und der Dauer der Zeitschleife.

Das Meßergebnis wird schließlich verglichen mit dem eingestellten Zahlenschalterwert; wird dieser durch das Meßergebnis übertroffen, so wird das Blitzlicht angesteuert. Dazu wird ein kurzer Impuls auf Port C, Bit Nr. 4 ausgegeben.

An Stelle der verwendeten Abfrageschleifen hätte man mit etwas mehr Hardware-Aufwand eine Interrupt-Organisation einrichten können; hierdurch wäre der Meßfehler verringert worden.

Das Programm benötigt 45 Befehle auf 83 ROM-Plätzen; ein RAM ist nicht nötig.

9. Zahlensysteme, binäre Addition und Subtraktion

Um die Wirkungsweise von Computern zu verstehen, muß man sich mit dem Binär-Zahlensystem vertraut machen; ferner ist die Kenntnis des Hexadezimal- und des Oktal-Systemes zweckmäßig. Wer diese Begriffe zum ersten Mal hört, sollte sich nicht durch die beunruhigend-mathematisch klingenden Wortbildungen abschrecken lassen. Zum Verständnis der Zahlensysteme benötigt man nur die simple Fähigkeit zu zählen.

9.1. DEZIMAL-System

Für die folgenden Erläuterungen stelle man sich einen Auto-Kilometerzähler vor. Er soll so primitiv sein, daß er keine Anzeige für km-Bruchteile enthält, siehe Bild 9.1.
Der Zähler besteht aus 5 Rädchen; auf jedem der Rädchen sind die 10 Ziffern des Dezimalsystems angeordnet. Das Rädchen ganz rechts ist mit einem Rade des Autos verbunden und ist dasjenige, das direkt die gefahrenen km zählt. Wegen dieser besonderen Rolle wird es hier als Zählrädchen bezeichnet und erhält die Nummer 0. Die anderen Rädchen heißen von rechts nach links „Nr. 1" bis „Nr. 4".

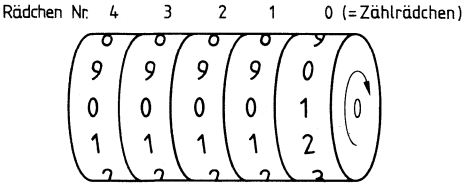


Bild 9.1
Kilometerzähler

Solange nicht mehr als 10 km zurückgelegt sind, zeigt das Zählrädchen direkt die Anzahl der gefahrenen km an. Danach erscheint aber wieder die Ziffer 0; das Zählrädchen muß dann also seine Anzeige wiederholen. Damit trotzdem eine eindeutige Anzeige zustandekommt, dient Rädchen Nr. 1 als „Wiederholungszähler" für das Zählrädchen: Auf Grund eines inneren Mechanismus des km-Zählers rückt Rädchen Nr. 1 immer dann um eine Ziffer weiter, wenn das Zählrädchen zum ersten oder wiederholten Male von Ziffer 9 auf Ziffer 0 weiterrückt. Anfangs gibt die von Rädchen Nr. 1 gezeigte Ziffer die Anzahl ganzer Umdrehungen an, die das Zählrädchen schon gemacht hat. Wenn aber Rädchen Nr. 1 ebenfalls eine ganze Umdrehung zurückgelegt hat, muß es ebenfalls seine Anzeige wiederholen; deshalb muß dann das Rädchen Nr. 2 als Wiederholungszähler für Rädchen Nr. 1 dienen.

Man sieht, wie sich das fortsetzt:

DEZ, a Jedes der Rädchen mit den Nummern 1 bis 4 dreht sich immer dann zur nächsten DEZIMAL-Ziffer weiter, wenn sein rechter Nachbar von der DEZIMAL-Ziffer 9 zur DEZIMAL-Ziffer 0 weiterrückt.

DEZ, b Es rückt um eine DEZIMAL-Ziffer weiter:

Rädchen Nr.	nach jeweils km
0	1
1	10
2	$10 \times 10 = 100$
3	$10 \times 10 \times 10 = 1000$
4	$10 \times 10 \times 10 \times 10 = 10000$

In einer abkürzenden Schreibweise mit den „Exponenten" 1 bis 4 ist

$$\begin{aligned} 10 &= 10^1, \quad (\text{„zehn hoch eins"}) \\ 100 &= 10 \times 10 = 10^2, \\ 1000 &= 10 \times 10 \times 10 = 10^3, \\ 10000 &= 10 \times 10 \times 10 \times 10 = 10^4. \end{aligned}$$

Legt man dann noch fest, daß für jede beliebige Zahl y gelten soll: $y^0 = 1$, so kann man die Aussage (DEZ, b) allgemein formulieren:

DEZ, c Das Rädchen mit der Nummer n rückt nach jeweils 10^n km um eine DEZIMAL-Ziffer weiter (n = 0, 1, 2, 3 oder 4).

Nach dieser Regel entsteht die von km-Zähler angezeigte DEZIMAL-Zahl.

Man muß unterscheiden zwischen der Anzahl der gefahrenen km und der daraufhin vom km-Zähler angezeigten Zahl. Es wird sich zeigen, daß bei gleicher gefahrener Strecke die vom km-Zähler angezeigte Zahl davon abhängt, mit welchem Zahlensystem der Zähler arbeitet. Dagegen ändert sich die Anzahl der gefahrenen km natürlich nicht, wenn man auf ein anderes Zahlensystem übergeht; ebensowenig, wie sich ein Gegenstand ändert, nur weil man ihn in einer andere Sprache beschreibt als zuvor.
Die Umkehrung der Aussage (DEZ, c) führt von der angezeigten Zahl zur Anzahl der km:

DEZ, d Die Anzahl gefahrener km ist die Summe sämtlicher Ausdrücke „10^n mal Anzeige des Rädchens Nr. n", wobei n alle Werte von 0 bis 4 durchlaufen soll.

Wenn etwa das Zählrädchen die Ziffer 5 zeigt, kann es schon 5, 15, 25 usw. km zurückgelegt haben. Die Vorschrift (DEZ, d) berücksichtigt nur die letzten 5 davon. Das ist deshalb richtig, weil die zurückgelegten vollen Umdrehungen des Zählrädchens durch die Anzeige der höher numerierten Rädchen berücksichtigt sind.

Beispiel: Zeigt der km-Zähler an „56387", so ergibt sich nach (DEZ, d) beim Durchlaufen der n-Werte von 0 bis 4 die Summe $7 \times 10^0 + 8 \times 10^1 + 3 \times 10^2 + 6 \times 10^3 + 5 \times 10^4$.

Wie findet man nun die gesuchte Anzahl? Man kann sie sich veranschaulichen, z. B. durch eine Strichliste oder indem man etwa Streichholzpakete von 10, 100, 1000 und 10000 Stück herstellt und in entsprechender Kombination zusammenlegt.
Dagegen würde die Antwort „Das war von vornherein klar, es kommt heraus 56387 km" die Begriffe verwirren. Man geht aus von einer DEZIMAL-Zahl und sucht

die ihr entsprechende Anzahl. Nur weil man so sehr an das Denken in DEZIMAL-Zahlen gewöhnt ist, kann man sich ohne Hilfsmittel die richtige Anzahl sofort vorstellen. Daher ist für die meisten Menschen eine DEZIMAL-Zahl in der Vorstellung identisch mit der ausgedrückten Anzahl, und die Aufgabe im Beispiel sieht aus wie ein Scheinproblem. Dies wird sich bei der Behandlung anderer Zahlensysteme ändern. Es wird dann eine ernsthafte Aufgabe sein, aus einer Zahl des anderen Systems die entsprechende DEZIMAL-Zahl zu ermitteln; damit kann man dann auch die Anzahl als ermittelt betrachten.

Wenn es notwendig ist, zur Vermeidung von Mißverständnissen eine DEZIMAL-Zahl ausdrücklich als solche zu kennzeichnen, hängt man der Zahl ein „D" an; entsprechend B bei Binärzahlen, H bei Hexadezimalzahlen und Q bei Oktalzahlen. Die Anzahl von Punkten : : . wird ausgedrückt durch z. B. die Zahl 5 D.

9.2. OKTAL-System

Man stelle sich intelligente Lebewesen vor, die nur vier Finger an jeder Hand besitzen. Diese Lebewesen würden wahrscheinlich ein Zahlensystem mit nur acht Ziffern einführen, z. B. mit den Ziffern *, +, ÷, ≙, £, $, % und &. Zur Erleichterung der Diskussion soll hier dafür geschrieben werden 0, 1, 2, 3, 4, 5, 6 und 7; diese Zeichen sind jetzt aber nicht Ziffern des DEZIMAL-Systems!
Das Zahlensystem mit acht Ziffern heißt OKTAL-System.

Der km-Zähler für Benutzer des OKTAL-Systems hat auf jedem Rädchen nur acht Ziffern. Deshalb erreicht das Zählrädchen schon beim 8. Kilometer (DEZIMAL ausgedrückt) wieder die OKTAL-Ziffer 0, und das Rädchen Nr. 1 als Wiederholungszähler für das Zählrädchen rückt dann auf die OKTAL-Ziffer 1. Das sich dadurch ergebende Muster 0–0–0–1–0 sollte man nicht als „zehn" aussprechen, sondern als „eins-null OKTAL". Das schreibt man „10 Q".

Es folgen nun die den Aussagen (DEZ, a) bis (DEZ, d) entsprechenden Aussagen für das OKTAL-System. Ihre Begründung läuft ganz analog. Es wird nur, entsprechend der geänderten Anzahl von Ziffern, die Zehn durch die Acht ersetzt.
Bei allen Aussagen über nicht-DEZIMALE Zahlensysteme sollen die Nummern der Zähler-Rädchen ohne Erläuterung immer mit den Zeichen 0, 1, 2, 3 und 4 angegeben werden.

OKT, a Jedes der Rädchen mit den Nummern 1 bis 4 dreht sich immer dann zur nächsten OKTAL-Ziffer weiter, wenn sein rechter Nachbar von der OKTAL-Ziffer 7 zur OKTAL-Ziffer 0 weiterrückt.

OKT, b Es rückt um eine OKTAL-Ziffer weiter:

Rädchen Nr.	nach jeweils km (DEZIMAL)
0	1
1	8
2	$8 \times 8 = 64$
3	$8 \times 8 \times 8 = 512$
4	$8 \times 8 \times 8 \times 8 = 4096$

OKT, c Das Rädchen mit der Nummer n rückt nach jeweils 8^n km (DEZIMAL) um eine OKTAL-Ziffer weiter (n = 0, 1, 2, 3 oder 4).

Beispiel: Wie lautet die OKTAL-Zahl für diejenige Anzahl, die durch 795 D ausgedrückt wird?
Oder von jetzt ab kurz: Welche OKTAL-Zahl ist gleich 795 D? Vergleich mit (OKT, b) zeigt, daß sich Rädchen Nr. 4 nach 795 D km noch nicht bewegt hat, wohl aber Rädchen Nr. 3 einmal. Auf den restlichen 795 D – 512 D Kilometern = 283 D km hat sich

Rädchen Nr. 2 weitere vier Male bewegt (zusätzlich zur früheren vollen Umdrehung), und auf den danach restlichen 283 D – 256 D Kilometern = 27 D km hat sich Rädchen Nr. 1 dreimal, schließlich auf den letzten 27 D – 24 D Kilometern = 3 D km hat sich Rädchen Nr. 0 noch dreimal bewegt. Also folgt 795 D = 1433 Q.

Anschließend eine vergleichende Zusammenstellung von DEZIMAL- und gleichgroßen OKTAL-Zahlen für kleinere Werte. Man erkennt hieran auch, daß die OKTAL-Zahlen genauso lückenlos und eindeutig wie die DEZIMAL-Zahlen jede Anzahl ausdrücken:

DEZ:	0 . . . 7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
OKT:	0 . . . 7	10	11	12	13	14	15	16	17	20	21	22	23	24	25	26	27	30

OKT, d Die Anzahl gefahrener km, DEZIMAL ausgedrückt, ist die Summe sämtlicher Ausdrücke „8^n mal Anzeige des Rädchens Nr. n", wobei n alle Werte von 0 bis 4 durchlaufen soll.

Beispiel: Eine OKTAL-Zahl laute 23725 Q. Das ist DEZIMAL
$2 \times 8^4 + 3 \times 8^3 + 7 \times 8^2 + 2 \times 8^1 + 5 \times 8^0 = 8192 + 1536 + 448 + 16 + 5 = 10197$ D.

9.3. BINÄR-System (auch Dualsystem)

Im BINÄR-System gibt es nur zwei Ziffern. Sie werden mit denselben Zeichen bezeichnet wie die DEZIMAL-Ziffern 0 und 1. Ein km-Zähler für BINÄR-Zahlen trägt auf jedem Rädchen nur die beiden BINÄR-Ziffern. Die Rädchen müssen sich also viel schneller drehen als beim DEZIMAL- oder OKTAL-Zähler.

BIN, a Jedes der Rädchen mit den Nummern 1 bis 4 dreht sich immer dann zur nächsten BINÄR-Ziffer weiter, wenn sein rechter Nachbar von der BINÄR-Ziffer 1 zur BINÄR-Ziffer 0 weiterrückt.

BIN, b Es rückt um eine BINÄR-Ziffer weiter

Rädchen Nr.	nach jeweils km (DEZIMAL)
0	1
1	2
2	$2 \times 2 = 4$
3	$2 \times 2 \times 2 = 8$
4	$2 \times 2 \times 2 \times 2 = 16$

BIN, c Das Rädchen mit der Nummer n rückt nach jeweils 2^n km (DEZIMAL) um eine BINÄR-Ziffer weiter (n = 0, 1, 2, 3 oder 4).

Beispiel: Welche Binärzahl ist gleich 29 D?
In DEZIMALER Schreibweise ist $29 = 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$.
Deshalb ist 29 D = 11101 B.

Es folgt eine vergleichende Zusammenstellung:

DEZ	BIN	DEZ	BIN	DEZ	BIN
00	00000	06	00110	12	01100
01	00001	07	00111	13	01101
02	00010	08	01000	14	01110
03	00011	09	01001	15	01111
04	00100	10	01010	16	10000
05	00101	11	01011	31	11111

Der BINÄRe km-Zähler reicht also nicht weit. Der Vorteil, mit nur zwei Ziffern auszukommen, wodurch u.a. das Rechnen einfacher wird, ist erkauft mit einer größeren Länge der Zahlen.

BIN,d	Die Anzahl gefahrener km, DEZIMAL ausgedrückt, ist die Summe sämtlicher Ausdrücke „2^n mal Anzeige des Rädchens Nr. n", wobei n alle Werte von 0 bis 4 durchlaufen soll.

Beispiel: 10010000 B = (DEZIMAL) $1 \times 2^7 + 1 \times 2^4 = 128 + 16 = 144$ D.

9.4. HEXADEZIMAL-System (auch Sedezimalsystem)

Dies ist ein Zahlensystem mit mehr als zehn Ziffern, und zwar mit sechzehn Ziffern. Weil zu deren Darstellung die zehn Zeichen des DEZIMAL-Systems nicht ausreichen, benutzt man zusätzlich die Buchstaben A bis F:

DEZ:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
HEX:	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Jedes Rädchen des km-Zählers trägt bei Benutzung des HEXADEZIMAL-Systems die sechzehn Ziffern 0 bis F.

HEX,a	Jedes der Rädchen mit den Nummern 1 bis 4 dreht sich immer dann zur nächsten HEXADEZIMAL-Ziffer weiter, wenn sein rechter Nachbar von der HEXADEZIMAL-Ziffer F zur HEXADEZIMAL-Ziffer 0 weiterrückt.
HEX,b	Es rückt um eine HEXADEZIMAL-Ziffer weiter:

Rädchen Nr.	nach jeweils km (DEZIMAL)
0	1
1	16
2	$16 \times 16 = 256$
3	$16 \times 16 \times 16 = 4096$
4	$16 \times 16 \times 16 \times 16 = 65536$

HEX,c	Das Rädchen mit der Nummer n rückt nach jeweils 16^n km (DEZIMAL) um eine HEXADEZIMAL-Ziffer weiter (n = 0, 1, 2, 3 oder 4).

Beispiel: In 45808 D ist (DEZIMAL ausgedrückt) 16^4 noch nicht enthalten, dagegen 16^3 bereits elfmal. Im Rest 752 ist 16^2 zweimal enthalten, im Rest 240 ist 16^1 genau fünfzehnmal enthalten. Daher 45808 D = B2F0 H.

Eine vergleichende Zusammenstellung für kleinere Zahlen:

DEZ:	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
HEX:	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21

HEX,d	Die Anzahl gefahrener km, DEZIMAL ausgedrückt, ergibt sich als Summe sämtlicher Ausdrücke „16^n mal Anzeige des Rädchens Nr. n", wobei n alle Werte von 0 bis 4 durchlaufen soll.

Beispiel: Wie groß ist F0AC H?
Das ist (DEZIMAL) $= 15 \times 16^3 + 10 \times 16^1 + 12 \times 16^0 = 61440 + 160 + 12 =$
$= 61612$ D.

9.5. Abkürzungen von BINÄR-Zahlen durch OKTAL- oder HEXADEZIMAL-Zahlen

Die BINÄR-Zahlen sind verhältnismäßig lang und auch verhältnismäßig leicht zu verwechseln, – beides, weil es nur zwei BINÄR-Ziffern gibt. Das stört nicht, solange BINÄR-Zahlen vom Computer behandelt werden; aber es stört dann, wenn sich ein menschlicher Bearbeiter schriftlich oder mündlich mit ihnen befaßt. Deshalb benutzt man gern eine „Stenografie" mit Hilfe von OKTAL- oder HEXADEZIMAL-Zahlen.
Die Stenografie besteht im Grunde nur darin, daß man eine BINÄR-Zahl ersetzt durch die gleichgroße OKTAL- oder HEXADEZIMAL-Zahl. Aber es gibt noch eine angenehme Vereinfachung. Es genügt nämlich, wenn man die BINÄR-Zahl in Gruppen von je drei BINÄR-Ziffern in je eine OKTAL-Ziffer oder in Gruppen von je vier BINÄR-Ziffern in je eine HEXADEZIMAL-Ziffer umwandelt. Dann ergibt sich im ganzen die richtig übersetzte Zahl. Man beginnt dabei mit den niederwertigsten BINÄR-Ziffern, die ganz rechts stehen. Am hochwertigen (linken) Abschluß der BINÄR-Zahl füllt man notfalls mit Nullen auf.

1. Beispiel: Die BINÄR-Zahl 10010000 B (wie nach BIN, d) soll durch die gleichgroße OKTAL-Zahl ersetzt werden.
Sie ist (DEZIMAL) $= 2^7 + 2^4 = 2 \times 8^2 + 2 \times 8^1$, also $= 220$ Q.

Schneller findet man das so: Man zerlegt in Dreiergruppen und übersetzt einzeln diese Gruppen: 10010000 B wird geschrieben 010 010 000, dafür die OKTAL-Ziffern 2, 2 und 0.

2. Beispiel: Dieselbe Zahl HEXADEZIMAL ausdrücken. Hier gleich mit der schnellen Methode: 10010000 wird geschrieben 1001 0000, daraus die Hexadezimalziffern 9 und 0.

Mit der gleichen Methode erhält man z. B.
11111010 B = FA H,
01011100 B = 5C H,
01101110 B = 6E H.

9.6. Addition und Subtraktion binärer Zahlen

Jedem ist das Addieren zweier positiver Dezimalzahlen auf dem Papier geläufig. Man beginnt ganz rechts mit den niederwertigsten Ziffern; beim jeweils nächsten Ziffernpaar ist der Übertrag vom letzten Ziffernpaar mitzubeachten.

Beispiel:	Erster Summand	3 2 7
	Zweiter Summand	7 3 4
	Übertrag	1 ↖ 0 ↖ 1 ↖
	Resultat	1 0 6 1

Nach dieser Methode kann man auch Zahlen anderer Zahlensysteme addieren, z. B. Binärzahlen.

Beispiel:	Erster Summand	0 1 0 1 1 0 0 1
	Zweiter Summand	1 0 0 1 1 1 0 1
	Übertrag	0 ↖ 0 ↖ 0 ↖ 1 ↖ 1 ↖ 0 ↖ 0 ↖ 1 ↖
	Resultat	1 ↖ 1 ↖ 1 ↖ 1 ↖ 0 ↖ 1 ↖ 1 ↖ 0

Ganz rechts ergibt sich aus dem Ziffernpaar Nr. 0: 1 B + 1 B = 2 D = 10 B, also im Resultat 0 und im Übertrag 1. Beim Ziffernpaar Nr. 4 ergibt sich mit dem Übertrag zusammen sogar 3 D = 11 B, also 1 im Resultat und 1 im Übertrag.

Ein Computer kann die **binäre Addition** leicht durchführen. Er braucht nur eine Kombination von zwei logischen Verknüpfungen, um nacheinander für jedes Ziffernpaar die Resultatziffer und den neuen Übertrag zu ermitteln (wird hier nicht näher ausgeführt). Ein Unterschied zum Rechnen auf dem Papier besteht aber in der begrenzten Ziffernzahl des Computers, z. B. 8 Binärziffern im letzten Beispiel. Dort hätte mit anderen Zahlen der Übertrag aus der höchsten Stelle (Nr. 7) ebensogut auch 1 sein können. Damit solche Überträge nicht verlorengehen, besitzt ein Mikroprozessor ein internes Register für eine Binärziffer (1 Bit), in dem sie gespeichert werden.
Der Inhalt dieses Registers heißt das Übertragsbit. Den darin zu speichernden Übertrag aus der höchsten Stelle bezeichnet man meist kurz als d e n Übertrag.
Das Übertragsbit wird gesetzt (erhält oder behält den Wert 1), wenn die letzte Addition einen Übertrag ergab. Es wird zurückgesetzt (erhält oder behält den Wert 0), wenn die letzte Addition keinen Übertrag ergab.
Auf Ausnahmen von dieser Regel und auf andere Verwendungszwecke des Übertragsbits wird hier nicht eingegangen.

Die **Subtraktion zweier Binärzahlen** soll nun durch das Bild des (dezimalen) km-Zählers vorbereitet werden.
Eine Subtraktion entspricht dem Zurückdrehen des Zählers um eine bestimmte Anzahl von km. Um im Bild zu bleiben, muß jetzt zunächst angenommen werden, daß das Resultat der Subtraktion positiv sei; auf dem km-Zähler gibt es ja nur positive Zahlen. Man kann dasselbe Ergebnis bekommen, indem man den Zähler vorwärtsdreht; aber um wieviel?
Die größte Zahl auf dem Zähler ist 99999. Mit Einschluß der Null gibt es also 100000 verschiedene Zahlen auf dem Zähler. Wenn man zur Anzeige 100000 addiert oder von der Anzeige 100000 subtrahiert, ändert man nichts.

Diese Aussage wäre falsch, wenn auch der km-Zähler eine Übertragsziffer hätte, die dem Übertragsbit des Mikroprozessors entsprechen würde.

Man kann, anstatt von der Anzeige 40000 zu subtrahieren, erst 100000 subtrahieren und dann 60000 addieren. Da die Subtraktion der Zahl 100000 nichts ändert, kann man sie auch unbeachtet lassen.

km-Zähler-Subtraktion:
Wenn $x > y$ (d. h. „x größer als y"), dann
$x - y = x + (100000 - y) - 100000$
(unbeachtet)

Die Zahl $(100000 - y)$ heißt das Zehnerkomplement von y, wenn es sich wie hier um fünfstellige Dezimalzahlen handelt. Bei dreistelligen Dezimalzahlen wäre $(1000 - y)$ das Zehnerkomplement von y.
Nun wird die gezeigte Methode auf Binärzahlen angewendet; damit alles einfach bleibt, sollen es vierstellige Binärzahlen sein. Zu Anfang wird das Übertragsbit noch nicht beachtet.

Es gibt sechzehn verschiedene Binärzahlen von 0000 bis 1111. Deshalb lautet nun die Subtraktionsvorschrift

Binär-Subtraktion ohne Übertragsbit:
Wenn x und y vierstellige Binärzahlen und $x > y$, „x größer als y", dann ist
$x - y = x + (10000\ B - y) - 10000\ B$
(unbeachtet)

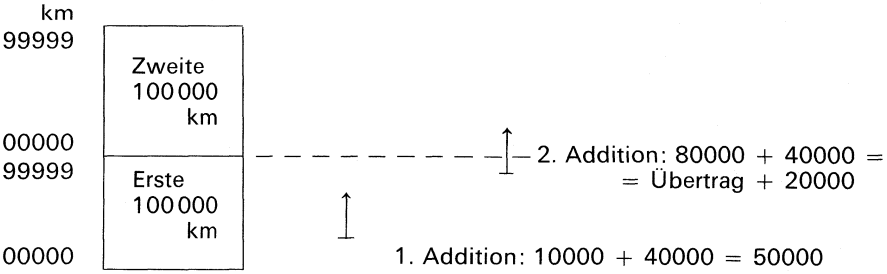
Die Zahl $(10000\ B - y) = (16\ D - y)$ heißt das **Zweierkomplement** von y. Hat man achtstellige Binärzahlen, so ist $(100000000\ B - y) = (256\ D - y)$ das Zweierkomplement von y.

Es ist nicht wichtig, wie dieser Name entstanden ist. Wichtiger ist, daß der Mikroprozessor das Zweierkomplement einer Binärzahl sehr leicht bilden kann. Zuerst „komplementiert" er die Zahl, d.h. ersetzt Einsen durch Nullen und umgekehrt; dies ergibt das sogenannte **Einerkomplement.** Dann addiert er 1 dazu; dies ergibt das Zweierkomplement. Danach ist nur noch der gewöhnliche Additionsprozeß durchzuführen.

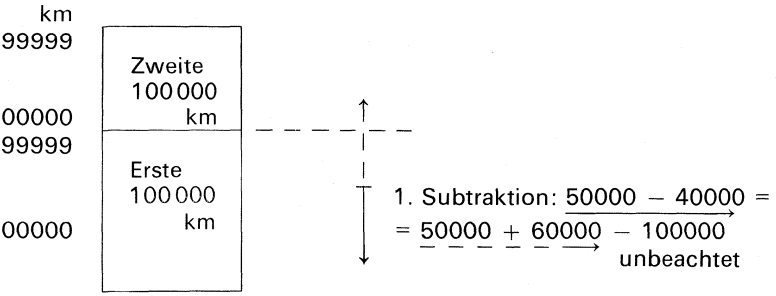
Der Mikroprozessor SAB 8080 ersetzt die Subtraktion einer Binärzahl y von einer Binärzahl x durch die Addition des Zweierkomplements von y zu x.

Beispiel: x − y mit x = 1111 B, y = 0110 B.
Einerkomplement = 1001 B
Zweierkomplement = 1010 B
Dazu addiert x = 1111 B
1001 B
Kontrolle: 15 D − 6 D = 9 D.

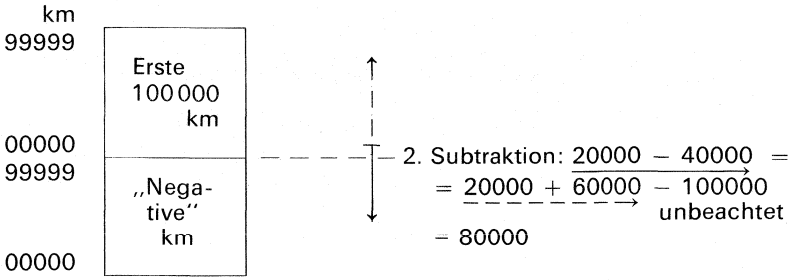
Jetzt kann das Verhalten des Übertragsbit untersucht werden. Dies geschieht der Anschaulichkeit zuliebe an Hand des dezimalen km-Zählers; die Übertragung auf Binärzahlen bereitet keinerlei Schwierigkeiten.
Der Zahlenbereich des km-Zählers erstreckt sich von 00000 bis 99999. Wenn eine Rechenoperation aus diesem Bereich herausführt, d.h. das Resultat größer als 99999 oder negativ ist, dann spricht man von einer „Bereichsüberschreitung" (besser wäre Grenzüberschreitung).
Man wünscht sich, daß das Übertragsbit immer dann gesetzt wird, wenn eine Bereichsüberschreitung vorliegt. Bei der Addition ist dies der Fall:



Wenn der km-Zähler einen Übertrag feststellen könnte, so würde er dies tun bei der bereichsüberschreitenden 2. Addition, dagegen nicht bei der nicht bereichsüberschreitenden 1. Addition. Hier ist also alles in Ordnung. Dagegen bei der Subtraktion:



Hier ergibt die Addition einen Übertrag, obwohl bei der eigentlichen Subtraktion keine Bereichsüberschreitung vorkommt. Der Grund dafür ist das Nichtbeachten von „– 100 000".



Hier ergibt sich kein Übertrag, obwohl bei der Subtraktion eine Bereichsüberschreitung vorkommt. Der Grund liegt wieder im Nichtbeachten von „– 100 000".

Bei Binärzahlen sieht es entsprechend aus. Es werde z. B.
2 D = 00000010 B von 1 D = 00000001 B subtrahiert.
Das Zweierkomplement zu 00000010 B ist 11111110 B:

$$\begin{array}{r} 00000001 \\ +\ 11111110 \\ \hline =\ 11111111 \end{array}$$

Ein Übertrag entsteht nicht, aber offenbar hat eine Bereichsüberschreitung stattgefunden.

Damit nur das eine (von beiden Rechenarten betätigte) Übertragsbit im SAB 8080 alle Bereichsüberschreitungen richtig anzeigt, wird beim Subtrahieren im SAB 8080 das Übertragsbit nachträglich komplementiert, d. h. wenn 1, dann zu 0 gemacht und umgekehrt. Mit dieser Regelung kann man für die bisher behandelten positiven Zahlen ohne Vorzeichen sagen:

> Das Übertragsbit im SAB 8080 wird gesetzt, wenn die Addition vorzeichenloser (und deshalb positiver) Zahlen einen Übertrag ergibt oder wenn die Subtraktion solcher Zahlen einen negativen Übertrag ergibt.

Als Beispiel folgt eine Subtraktion zweier 16 Bit-Zahlen.

$$\begin{array}{r} 11001111\ \ 10011100 \\ -\ 01110010\ \ 11001100 \end{array}$$

Zuerst die niederwertigen Bytes. Das Zweierkomplement von 11001100 ist 00110100:

$$\begin{array}{r} 10011100 \\ +\ 00110100 \\ \hline =\ 11010000 \end{array}$$

Es entsteht kein Übertrag, also wird nach der Regel das Übertragsbit gesetzt.
Nun die höherwertigen Bytes. Das Zweierkomplement wird analog gebildet:

$$\begin{array}{r} 11001111 \\ +\ 10001110 \\ \hline =\ 01011101 \end{array}$$

Es entsteht ein Übertrag, also muß nach der Regel das Übertragsbit zurückgesetzt werden. Nun muß aber noch das Übertragsbit aus den niederwertigen Bytes berücksichtigt werden; hierzu muß in der niedrigsten Stelle des höherwertigen Resultat-

Byte noch 1 subtrahiert werden. Das Gesamtresultat ist

```
  11001111 10011100 B  (= 53148 D)
- 01110010 11001100 B  (= 29388 D)
= 01011100 11010000 B  (= 23760 D)
```

Jetzt können **negative Binärzahlen** eingeführt werden. Dies kann auf verschiedene Art geschehen. Hier soll es so gemacht werden, daß der Mikroprozessor bei „kritikloser" Anwendung seines Rechenmechanismus dasselbe Resultat herausbringt*)

für die Subtraktion $x - y$ und für die Addition $x + (-y)$.

Dann gibt es nur eine Möglichkeit. Da der Mikrozessor die Subtraktion ersetzt durch Addition des Zweierkomplements, muß eine negative Zahl als Zweierkomplement der entsprechenden positiven definiert werden.
Dies wird nun an vierstelligen Binärzahlen durchgeführt. Natürlich kann man nicht mehr 16 nicht-negative Zahlen durch die 16 möglichen Vierergruppen ausdrücken; einige von ihnen müssen negativen Zahlen Platz machen.

Also:

y dezimal	y binär	Einer-kompl.	Zweier-kompl.	Dies soll bedeuten dezimal	und **nicht**
0	0000	1111	0000	0	
+1	0001	1110	1111	−1	+15
+2	0010	1101	1110	−2	+14
+3	0011	1100	1101	−3	+13
+4	0100	1011	1100	−4	+12
+5	0101	1010	1011	−5	+11
+6	0110	1001	1010	−6	+10
+7	0111	1000	1001	−7	+ 9

Bisher sind 15 Zahlen von +7 bis −7 vergeben. Das Zeichen 1000 wird noch als „−8 D" vergeben. Dann hat man den Vorteil, daß alle negativen Zahlen mit 1 beginnen, die positiven und Null mit 0. Daher bezeichnet man auch das höchstwertige Bit als Vorzeichenbit.
Null und −8 D sind ihre eigenen Zweierkomplemente.

Auf diese Weise kann man negative Zahlen für Binärzahlen jeder Länge definieren. Das einfachste Rezept ist: Man beginnt oben mit der größten positiven Zahl, die noch mit einer 0 beginnt. Die negativen Zahlen erhält man durch schrittweises Subtrahieren der 1 von Null abwärts. Wenn eine Subtraktion wieder die größte positive Zahl ergibt, hat man den „Zahlenkreis" umfahren. Bei vierstelligen Zahlen ergibt die Subtraktion der 1 von −8 D = 1000 B die größte positive Zahl +7 D = 0111 B. Bei achtstelligen Zahlen ist die kleinste negative Zahl −128 D = 10000000 B. Subtraktion der 1 ergibt 01111111 B = +127 D.

*) Bis auf das Übertragsbit; es wird am Schluß besprochen.

Zur besseren Übersicht folgt noch eine geordnete Aufstellung für vierstellige Binärzahlen:

DEZ	BIN mit Vorzch.	DEZ	BIN mit Vorzch.
+7	0111	−1	1111
+6	0110	−2	1110
+5	0101	−3	1101
+4	0100	−4	1100
+3	0011	−5	1011
+2	0010	−6	1010
+1	0001	−7	1001
0	0000	−8	1000

Bei vorzeichenbehafteten Binärzahlen dieser Art läßt sich kein einfacher Zusammenhang zwischen Übertragsbit und Bereichsüberschreitung herstellen. Das ergibt sich schon allein aus der Komplementierung des Übertragsbits nach der Subtraktion.
Das Resultat von x − y gleicht demjenigen von x + (−y) nur mit Ausnahme des Übertragsbits; denn dieses wird ja nach der Subtraktion komplementiert, nach der Addition nicht. Schon deshalb kann aus ihm allein keine Bereichsüberschreitung erkannt werden.

Es wäre auch zuviel verlangt. Der Rechenmechanismus ist genau derselbe wie bei den vorzeichenlosen (positiven) Zahlen; auch die positiven Zahlen, die in die vorzeichenbehafteten Zahlen übernommen wurden, sind unverändert. Aber die Bereichsgrenzen haben sich bei Einführung der Vorzeichen verschoben. Im Fall von vierstelligen Binärzahlen erstreckt sich der Zahlenbereich ohne Vorzeichen von Null bis 15 D, mit Vorzeichen von −8 D bis +7 D.

Ohne Vorzeichen ergab z.B. eine Addition 6 D + 3 D = 9 D ganz korrekt kein Setzen des Übertragsbit. Wenn jetzt dieselben Vierergruppen als vorzeichenbehaftete Zahlen +6 D und +3 D aufgefaßt werden, dann ergibt ihre Addition natürlich eine Bereichsüberschreitung; aber mit demselben Rechenmechanismus und denselben Binärziffer-Gruppen kommt dasselbe Übertragsbit heraus wie zuvor, nämlich 0.
In einem solchen Fall muß man deshalb andere Methoden zur Kontrolle der Bereichsüberschreitung einführen.

Sachverzeichnis

	Seite
Abfrage	57, 81
Adresse	10, 15, 16, 24, 25
Adressierung	32
Adreßbus	15, 63
Akkumulator	12, 30
Analogtechnik	9
Arithmetische Operation	24, 36
Anweisung	92
Assembler	90, 91, 92
Assemblerlisting	24, 95
Assemblersprache	90, 92
Assemblieren	90
Aufruf	75
Ausgabegeräte	11
Bausteine	12
BCD	58
Bedingter Sprung	66, 71
Befehle	9, 10, 24
Befehlsregister	64
Befehlszähler	62, 71, 74
Befehlszyklus	9, 61
Bit	10
Bus	15
Byte	16
Datenbus	15, 63
Datenwort	10
Dezimalkorrektur	58
Dialog	92
Digitaltechnik	9
E/A's	11, 14, 32, 57
Editor	90, 91, 92
Eingabegeräte	11
Entwicklungssystem	90
Hardware	9
Hexadezimalziffern	18
Hilfsübertragsbit	58, 66
Interface	11
k	17
Kanal	14, 17, 32
Kapazität	10, 17
Kellerspeicher	76
Konstante	11
Leitungen	15
Logische Operation	24, 53
Makro	93
Maschinensprache	18
Maschinenzyklus	62
Maskierung	56
Mikroprozessor	12
Mnemonische Abkürzungen	18, 92
Monitor	90, 91, 92
Nahtstelle	11
Nullbit	70
Objektprogramm	25
Operand	16
Operationscode	16, 24
Paritätsbit	70
Port	14
Programm	9
Programmablaufplan	66
Programmiersprache	90
Programmsprung	66
Programmzähler	62
Pseudobefehle	92
PSW	71
Quellenprogramm	25
RAM	14, 16
Register	12, 28
ROM	14, 16
Rotation	59
Rückkehrbefehl	76
Schleife	73
Schnittstelle	11
Software	9
Speicher	10, 14
Speicherplatz	10, 15
Sprung	66
Stapelzeiger	76
Statuswort	57, 62
Steuerbus	15, 62
Steuersignale	15, 62
Steuerwort	57
Symbolische Adresse	25
Takt	24, 61
Transfer	23, 26, 33
Übertragsbit	37, 55, 59
Unbedingter Sprung	66, 71
Unterbrechung	80
Unterprogramm	75
Variable	11
Verzweigung	66
Vorzeichenbit	70
Wortlänge	15
Zentraleinheit	9
Zustandsbits	66

Tabellen 4-1 bis 4-4

Tabelle 4-1/1. Teil[1])

Transfer 8 Bit

INPUT/OUTPUT
D3 OUT } Nr
DB IN }

MOVE IMMEDIATE

06	MVI B,	40	MOV B,B	50	MOV D,B	60	MOV H,B	70	MOV M,B
0E	MVI C,	41	MOV B,C	51	MOV D,C	61	MOV H,C	71	MOV M,C
16	MVI D,	42	MOV B,D	52	MOV D,D	62	MOV H,D	72	MOV M,D
1E	MVI E,	43	MOV B,E	53	MOV D,E	63	MOV H,E	73	MOV M,E
26	MVI H,	44	MOV B,H	54	MOV D,H	64	MOV H,H	74	MOV M,H
2E	MVI L,	45	MOV B,L	55	MOV D,L	65	MOV H,L	75	MOV M,L
36	MVI M,	46	MOV B,M	56	MOV D,M	66	MOV H,M		—
3E	MVI A,	47	MOV B,A	57	MOV D,A	67	MOV H,A	77	MOV M,A
		48	MOV C,B	58	MOV E,B	68	MOV L,B	78	MOV A,B
		49	MOV C,C	59	MOV E,C	69	MOV L,C	79	MOV A,C
		4A	MOV C,D	5A	MOV E,D	6A	MOV L,D	7A	MOV A,D
		4B	MOV C,E	5B	MOV E,E	6B	MOV L,E	7B	MOV A,E
		4C	MOV C,H	5C	MOV E,H	6C	MOV L,H	7C	MOV A,H
		4D	MOV C,L	5D	MOV E,L	6D	MOV L,L	7D	MOV A,L
		4E	MOV C,M	5E	MOV E,M	6E	MOV L,M	7E	MOV A,M
		4F	MOV C,A	5F	MOV E,A	6F	MOV L,A	7F	MOV A,A

D8

LOAD/STORE

3A LDA } Adr
32 STA }
0A LDAX B
1A LDAX D
02 STAX B
12 STAX D

Transfer 16 Bit

LOAD IMMEDIATE

01 LXI B,
11 LXI D,
21 LXI H,
31 LXI SP,
} D16

LOAD/STORE

2A LHLD } Adr
22 SHLD }

EB XCHG

STACK

C5	PUSH B	C1	POP B
D5	PUSH D	D1	POP D
E5	PUSH H	E1	POP H
F5	PUSH PSW	F1	POP PSW*
E3	XTHL		
F9	SPHL		

Arithmetische Operation 8 Bit

ADDITION*		SUBTRACTION*		IN-CREMENT**	DE-CREMENT**
80 ADD B	88 ADC B	90 SUB B	98 SBB B	04 INR B	05 DCR B
81 ADD C	89 ADC C	91 SUB C	99 SBB C	0C INR C	0D DCR C
82 ADD D	8A ADC D	92 SUB D	9A SBB D	14 INR D	15 DCR D
83 ADD E	8B ADC E	93 SUB E	9B SBB E	1C INR E	1D DCR E
84 ADD H	8C ADC H	94 SUB H	9C SBB H	24 INR H	25 DCR H
85 ADD L	8D ADC L	95 SUB L	9D SBB L	2C INR L	2D DCR L
86 ADD M	8E ADC M	96 SUB M	9E SBB M	34 INR M	35 DCR M
87 ADD A	8F ADC A	97 SUB A	9F SBB A	3C INR A	3D DCR A
C6 ADI D8	CE ACI D8	D6 SUI D8	DE SBI D8		

Arithmetische Operation 16 Bit

ADDITION+	INCREMENT	DECREMENT
09 DAD B	03 INX B	0B DCX B
19 DAD D	13 INX D	1B DCX D
29 DAD H	23 INX H	2B DCX H
39 DAD SP	33 INX SP	3B DCX SP

[1]) Befehlsliste, geordnet nach Inhalt der Befehle (Erläuterungen im 2. Teil)

Tabelle 4-1/2. Teil[1])

Logische Operation 8 Bit

AND *	X-OR *	OR *	COMPARE *	DECIMAL ADJUST *
A0 ANA B	A8 XRA B	B0 ORA B	B8 CMP B	27 DAA
A1 ANA C	A9 XRA C	B1 ORA C	B9 CMP C	
A2 ANA D	AA XRA D	B2 ORA D	BA CMP D	
A3 ANA E	AB XRA E	B3 ORA E	BB CMP E	
A4 ANA H	AC XRA H	B4 ORA H	BC CMP H	COMPLEMENT
A5 ANA L	AD XRA L	B5 ORA L	BD CMP L	2F CMA
A6 ANA M	AE XRA M	B6 ORA M	BE CMP M	
A7 ANA A	AF XRA A	B7 ORA A	BF CMP A	
E6 ANI D8	EE XRI D8	F6 ORI D8	FE CPI D8	

Sprung

JUMP		CALL		RETURN	RESTART
C3 JMP		CD CALL		C9 RET	C7 RST 0
C2 JNZ		C4 CNZ		C0 RNZ	CF RST 1
CA JZ		CC CZ		C8 RZ	D7 RST 2
D2 JNC		D4 CNC		D0 RNC	DF RST 3
DA JC	Adr	DC CC	Adr	D8 RC	E7 RST 4
E2 JPO		E4 CPO		E0 RPO	EF RST 5
EA JPE		EC CPE		E8 RPE	F7 RST 6
F2 JP		F4 CP		F0 RP	FF RST 7
FA JM		FC CM		F8 RM	
E9 PCHL					

Sonstige

ROTATE +	CONTROL	CARRY +
07 RLC	00 NOP	37 STC
0F RRC	76 HLT	3F CMC
17 RAL	F3 DI	
1F RAR	FB EI	

Erläuterungen zu Tabelle 4-1

D16 = 16-Bit-Konstante; Adr = Speicherplatzadresse;
D8 = 8-Bit-Konstante; Nr = E/A-Kanalnummer.

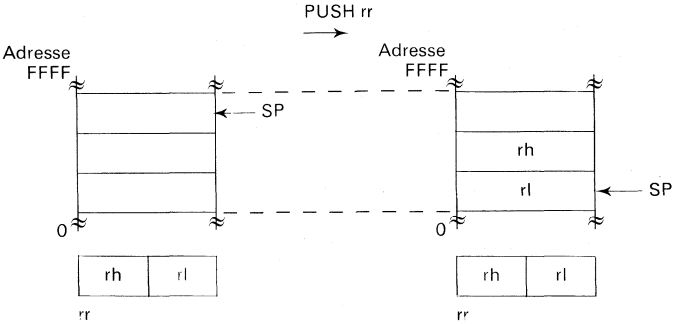
* = beeinflußt alle Zustandsbits,
** = beeinflußt alle Zustandsbits außer Übertragungsbit;
\+ = beeinflußt nur Übertragungsbit.

(Über die Richtung der Beeinflussung ist damit nichts gesagt)

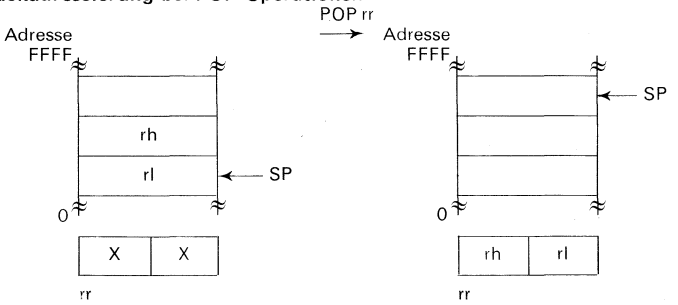
[1]) Befehlsliste, geordnet nach Inhalt der Befehle.

Tabelle 4-2
Befehlsvorrat

Stackadressierung bei PUSH-Operationen



Stackadressierung bei POP-Operationen



Legende zur nachfolgenden Tabelle

Legende zu „Binär-Code"

ddd = Zielregister
rr = Registerpaar
sss = Senderegister

ddd/sss

000 ≙ Reg. B
001 ≙ Reg. C
010 ≙ Reg. D
011 ≙ Reg. E
100 ≙ Reg. H
101 ≙ Reg. L
110 ≙ Speicher (M)
111 ≙ Akku

rr

00 ≙ B/C
01 ≙ D/E
10 ≙ H/L
11 ≙ Stack Ptr. (SP)

Legende zu „Beeinflußte Zustands-Bits"

* = bedingt gesetzt oder rückgesetzt
– = nicht beeinflußt
0 = rückgesetzt
1 = gesetzt

Legende zu „Taktzyklen"

Eine Taktperiode kann beim Mikroprozessor SAB 8080 von 480 ns bis 2 µs variieren, beim Mikroprozessor SAB 8085 von 200 ns bis 2 µs. Wenn zwei Zahlen für die Taktperiode angegeben sind (z. B. 5/11), bedeutet dies, daß die größere Zahl gilt, falls die Bedingung erfüllt ist, und die kleinere, falls die Bedingung nicht erfüllt ist.

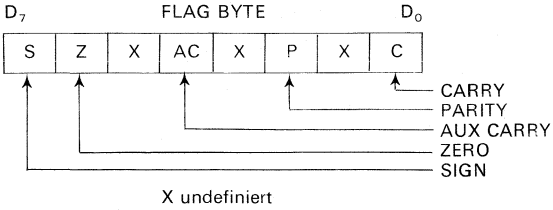
Achtung!

Für arithmetische Operationen mit 2-Byte-Zahlen und zur Adressierung ist das höherwertige Byte das linke Byte. Mit dem PUSH-Befehl wird zuerst das linke Byte in den Stack gespeichert und mit dem POP-Befehl wird zuerst das rechte Byte aus dem Stack geladen.

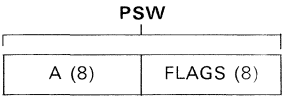
Interne Registerorganisation

A Reg. (8)

B Reg. (8)	C Reg. (8)
D Reg. (8)	E Reg. (8)
H Reg. (8)	L Reg. (8)
Programm Counter (16)	
Stack Pointer (16)	

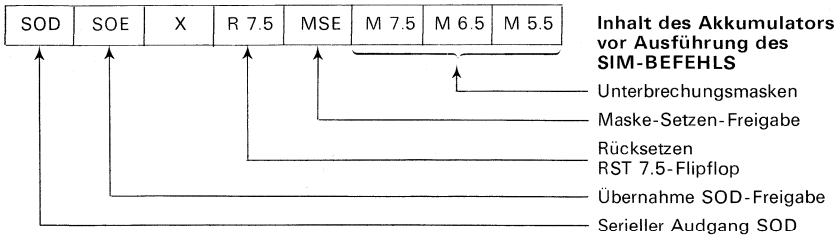


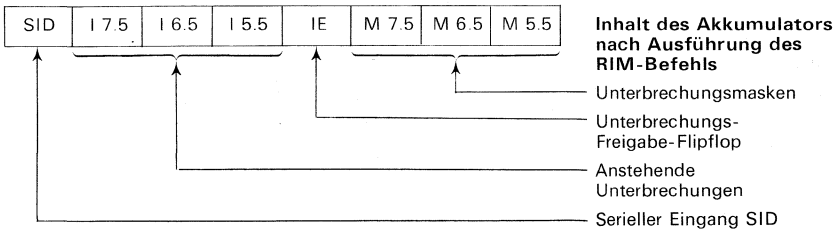
Registerpaarorganisation



B	(B/C)	(16)
D	(D/E)	(16)
H	(H/L)	(16)
Program Counter		(16)
Stack Pointer		(16)

SAB 8085 Prozessor-Unterbrechungsmaske





Mnemonic	Binär-Code	Beeinflußte Zustands-Bits S Z AC P C	Bytes	Taktzyklen 8080	Taktzyklen 8085	Englische Befehlsbeschreibung	Funktion des Befehls	Siehe Seite
TRANSFERBEFEHLE								
a) Register → Register								
MOV r_1, r_2	0 1 d d d s s s	– – – – –	1	5	4	Move register to register	r_1, r_2 = A, B, C, D, E, H oder L: Lade Register r_1 mit dem Inhalt von Register r_2	29
XCHG	1 1 1 0 1 0 1 1	– – – – –	1	4	4	Exchange D & E, H & L	Vertausche Inhalte der Registerpaare (D, E) und (H, L)	36
XTHL	1 1 1 0 0 0 1 1	– – – – –	1	18	16	Exchange top of stack H & L	Vertausche Inhalt des Registerpaares (H, L) und den Inhalt des Wortes, das durch den Stackpointer adressiert ist.	80
SPHL	1 1 1 1 1 0 0 1	– – – – –	1	5	6	H & L to stackpointer	Lade Stackpointer mit dem Inhalt des Registerpaares (H, L)	80
b) Speicher, Peripherie → Register								
MOV r_1, M	0 1 d d d 1 1 0	– – – – –	1	7	7	Move memory to register	r_1 = A, B, C, D, E, H oder L: Lade Register r_1 mit dem Inhalt des Speicherbytes, das durch den Inhalt des Registerpaares (H, L) adressiert ist	29
LDA adr	0 0 1 1 1 0 1 0	– – – – –	3	13	13	Load accu direct	Akkumulator laden mit dem Inhalt der Adresse adr	30
LDAX rp	0 0 r r 1 0 1 0	– – – – –	1	7	7	Load accu indirect	rp = B, D: Akkumulator laden mit dem Inhalt des Speicherplatzes, der durch den Inhalt des Registerpaares rp adressiert ist	31
LHLD adr	0 0 1 0 1 0 1 0	– – – – –	3	16	16	Load H & L direkt	Lade Registerpaar (H, L) mit dem Inhalt der Adressen adr und (adr + 1)	35
POP rp POP PSW	1 1 r r 0 0 0 1 1 1 1 1 0 0 0 1	– – – – – * * * * *	1	10	10	Pop register pair off stack	rp = B, D, H, PSW: Registerpaar rp wird mit dem Wort geladen, das durch den Stackpointer adressiert ist	78
IN nr	1 1 0 1 1 0 1 1	– – – – –	2	10	10	Input	Akkumulator wird mit dem Inhalt des Eingabekanals (Nummer nr $\leqq$ 255) geladen	27

Mnemonic	Binär-Code	Beeinflußte Zustands-Bits S Z AC P C	Bytes	Taktzyklen 8080	Taktzyklen 8085	Englische Befehlsbeschreibung	Funktion des Befehls	Siehe Seite

c) Konstante → Registerpaar

LXI rp, adr	0 0 r r 0 0 0 1	– – – – –	3	10	10	Load reg. pair immediate	rp = B, D, H, SP: Lade Registerpaar rp mit Wert adr	34

d) Register → Speicher, Peripherie

MOV M, r_1	0 1 1 1 0 s s s	– – – – –	1	7	7	Move register to memory	r_1 = A, B, C, D, E, H oder L: Inhalt von Register r_1 auf den Speicherplatz abspeichern, der durch den Inhalt des Registerpaares (H, L) adressiert ist	29
STA adr	0 0 1 1 0 0 1 0	– – – – –	3	13	13	Store accu direct	Akkumulator-Inhalt unter Adresse adr abspeichern	31
STAX rp	0 0 r r 0 0 1 0	– – – – –	1	7	7	Store accu indirect	rp = B, D: Akkumulator in dem Byte abspeichern, das durch den Inhalt des Registerpaares rp adressiert ist	31
SHLD adr	0 0 1 0 0 0 1 0	– – – – –	3	16	16	Store H & L direct	Registerpaar (H, L) unter Adresse adr u. (adr + 1) abspeichern	36
PUSH rp	1 1 r r 0 1 0 1	– – – – –	1	11	12	Push register pair rp on stack	rp = B, D, H, PSW: Inhalt des Registerpaares rp wird in das Wort übertragen, das durch den Stackpointer −2 adressiert ist	78
OUT nr	1 1 0 1 0 0 1 1	– – – – –	2	10	10	Output	Akkumulator-Inhalt wird auf Ausgabekanal (Nummer nr ≦ 255) ausgegeben	27

e) Konstante → Register-Speicher

MVI M, konst	0 0 1 1 0 1 1 0	– – – – –	2	10	10	Move to memory immediate	Lade den Speicherplatz, der durch den Inhalt des Registerpaares (H, L) adressiert ist, mit Konstante (konst ≦ 255)	28
MVI r_1, konst	0 0 d d d 1 1 0	– – – – –	2	7	7	Move to register immediate	r_1 = A, B, C, D, E, H oder L: Lade Register r_1 mit Konstante (konst ≦ 255)	28

Mnemonic	Binär-Code	Beeinflußte Zustands-Bits S Z AC P C	Bytes	Taktzyklen 8080	Taktzyklen 8085	Englische Befehlsbeschreibung	Funktion des Befehls	Siehe Seite
ARITHMETISCHE BEFEHLE								
INR r_1	0 0 d d d 1 0 0	* * * * –	1	5	4	Increment register	r_1 = A, B, C, D, E, H oder L: Zum Inhalt des Registers r_1 wird 1 addiert	38
INR M	0 0 1 1 0 1 0 0	* * * * –	1	10	10	Increment memory	Zum Inhalt des durch Registerpaar (H, L) adressierten Bytes wird 1 addiert	38
DCR r_1	0 0 d d d 1 0 1	* * * * –	1	5	4	Decrement register	r_1 = A, B, C, D, E, H oder L: Vom Inhalt des Registers r_1 wird 1 subtrahiert	38
DCR M	0 0 1 1 0 1 0 1	* * * * –	1	10	10	Decrement memory	Vom Inhalt des durch Registerpaar (H, L) adressierten Bytes wird 1 subtrahiert	38
INX rp	0 0 r r 0 0 1 1	– – – – –	1	5	6	Increment register pair	rp = B, D, H, SP: Der Inhalt des Registerpaares rp wird um 1 erhöht	48
DCX rp	0 0 r r 1 0 1 1	– – – – –	1	5	6	Decrement register pair	rp = B, D, H, SP: Der Inhalt des Registerpaares rp wird um 1 erniedrigt	48
ADD r_1	1 0 0 0 0 s s s	* * * * *	1	4	4	Add register to accu	r_1 = A, B, C, D, E, H oder L: Inhalt von Register r_1 wird zum Inhalt des Akkumulators addiert	38
ADD M	1 0 0 0 0 1 1 0	* * * * *	1	7	7	Add memory to accu	Der Inhalt des Speicherbytes, das durch den Inhalt des Registerpaares (H, L) adressiert ist, wird zum Inhalt des Akkumulators addiert	38
ADC r_1	1 0 0 0 1 s s s	* * * * *	1	4	4	Add register to accu with carry	r_1 = A, B, C, D, E, H oder L: Inhalt des Registers r_1 und Inhalt des Carry-Bits werden zum Inhalt des Akkumulators addiert	38
ADC M	1 0 0 0 1 1 1 0	* * * * *	1	7	7	Add memory to accu with carry	Inhalt des Speicherbytes, das durch den Inhalt des Registerpaares (H, L) adressiert ist und der Inhalt des Carry-Bits werden zum Inhalt des Akkumulators addiert	38
DAD rp	0 0 r r 1 0 0 1	– – – – *	1	10	10	Add register pair to H and L	rp = B, D, H, SP: Inhalt des Registerpaares rp, und der Inhalt des Registerpaares (H, L) werden addiert. Ergebnis in (H, L)	48

Mnemonic	Binär-Code	Beeinflußte Zustands-Bits S Z AC P C	Bytes	Taktzyklen 8080	Taktzyklen 8085	Englische Befehlsbeschreibung	Funktion des Befehls	Siehe Seite
SUB r_1	1 0 0 1 0 s s s	* * * * *	1	4	4	Subtract register from accu	r_1 = A, B, C, D, E, H oder L: Inhalt des Register r_1 wird vom Akkumulator-Inhalt subtrahiert	38
SUB M	1 0 0 1 0 1 1 0	* * * * *	1	7	7	Subtract memory from accu	Inhalt des Speicherbytes, das durch das Registerpaar (H, L) adressiert ist, wird vom Akkumulator subtrahiert	38
SBB r_1	1 0 0 1 1 s s s	* * * * *	1	4	4	Subtract register from accu with borrow	r_1 = A, B, C, D, E, H oder L: Inhalt von Register r_1 und Inhalt des Carry-Bits werden vom Akkumulator-Inhalt subtrahiert	38
SBB M	1 0 0 1 1 1 1 0	* * * * *	1	7	7	Subtract memory from accu with borrow	Inhalt des Speicherbytes, das durch das Registerpaar (H, L) adressiert ist, und der Inhalt des Carry-Bits werden vom Akkumulator subtrahiert	38
ADI konst	1 1 0 0 0 1 1 0	* * * * *	2	7	7	Add immediate to accu	Konstante (konst $\leqq$ 255) wird zum Inhalt des Akkumulators addiert	38
ACI konst	1 1 0 0 1 1 1 0	* * * * *	2	7	7	Add immediate to accu with carry	Zum Akkumulator-Inhalt werden konst $\leqq$ 255 und Carry-Bit addiert	38
SUI konst	1 1 0 1 0 1 1 0	* * * * *	2	7	7	Subtract immediate from accu	konst $\leqq$ 255 wird vom Akkumulator-Inhalt subtrahiert	38
SBI konst	1 1 0 1 1 1 1 0	* * * * *	2	7	7	Subtract immediate from accu with borrow	konst $\leqq$ 255 und der Inhalt des Carry-Bits werden vom Akkumulator-Inhalt subtrahiert	38
DAA	0 0 1 0 0 1 1 1	* * * * *	1	4	4	Decimal adjust accu	Akkumulator-Inhalt wird in eine 2stellige Zahl umgewandelt	58

Mnemonic	Binär-Code	Beeinflußte Zustands-Bits S Z AC P C	Bytes	Taktzyklen 8080	Taktzyklen 8085	Englische Befehlsbeschreibung	Funktion des Befehls	Siehe Seite

LOGISCHE OPERATIONEN

CMA	00101111	– – – – –	1	4	4	Complement accu	Akkumulator-Inhalt wird negiert	58
ANA r_1 für 8085	10100sss	* * * * 0 * * 1 * 0	1	4	4	And register with accu	r_1 = A, B, C, D, E, H oder L: Akkumulator-Inhalt und der Inhalt des Registers r_1 werden UND-verknüpft	55
ANA M für 8085	10100110	* * * * 0 * * 1 * 0	1	7	7	And memory with accu	Der Inhalt des durch Registerpaar (H, L) adressierten Bytes wird mit dem Akkumulator-Inhalt UND-verknüpft	55
ANI konst für 8085	11100110	* * * * 0 * * 1 * 0	2	7	7	And immediate with accu	Akkumulator-Inhalt wird mit der konst ≦ 255 UND-verknüpft	55
ORA r_1	10110sss	* * 0 * 0	1	4	4	Or register with accu	r_1 = A, B, C, D, E, H oder L: Akkumulator-Inhalt wird mit dem Inhalt des Registers r_1 ODER-verknüpft	55
ORA M	10110110	* * 0 * 0	1	7	7	Or memory with accu	Inhalt des über Registerpaar (H, L) adressierten Bytes wird mit dem Akkumulator-Inhalt ODER-verknüpft	55
ORI konst	11110110	* * 0 * 0	2	7	7	Or immediate with accu	Akkumulator-Inhalt wird mit konst ≦ 255 ODER-verknüpft	55
XRA r_1	10101sss	* * 0 * 0	1	4	4	Exclusive Or register with accu	r_1 = A, B, C, D, E, H oder L: Akkumulator-Inhalt wird mit dem Inhalt des Registers r_1 Exklusiv-ODER-verknüpft	55
XRA M	10101110	* * 0 * 0	1	7	7	Exclusive Or memory with accu	Das über Registerpaar (H, L) adressierte Byte wird durch Exklusiv-ODER mit dem Akkumulator verknüpft	55
XRI konst	11101110	* * 0 * 0	2	7	7	Exclusive Or immediate with accu	Der Akkumulator wird mit dem Wert konst ≦ 255 durch Exklusiv-ODER-verknüpft	55
CMP r_1	10111sss	* * * * *	1	4	4	Compare register with accu	r_1 = A, B, C, D, E, H oder L: Akkumulator-Inhalt wird mit dem Inhalt des Registers r_1 verglichen	57
CMP M	10111110	* * * * *	1	7	7	Compare memory with accu	Akkumulator-Inhalt wird mit dem Inhalt des durch Registerpaar (H, L) adressierten Bytes verglichen	57
CPI konst	11111110	* * * * *	2	7	7	Compare immediate with accu	Akkumulator-Inhalt wird mit konst ≦ 255 verglichen	57

Mnemonic	Binär-Code	Beeinflußte Zustands-Bits S Z AC P C	Bytes	Taktzyklen 8080	Taktzyklen 8085	Englische Befehlsbeschreibung	Funktion des Befehls	Siehe Seite

REGISTERANWEISUNGEN

a) Akkumulatorinhalt rotieren

RLC	00000111	– – – – *	1	4	4	Rotate accu left	Akkumulator-Inhalt wird um 1 Bit nach links verschoben. Bit 2^7 wird in das Carry-Bit geschrieben und auch in das Bit 2^0	59
RRC	00001111	– – – – *	1	4	4	Rotate accu right	Akkumulator-Inhalt wird um 1 Bit nach rechts verschoben. Bit 2^0 wird in das Carry-Bit geschrieben und auch in das Bit 2^7	59
RAL	00010111	– – – – *	1	4	4	Rotate accu left through carry	Akkumulator-Inhalt wird um 1 Bit nach links geschoben. Bit 2^7 wird in das Carry-Bit und das Carry-Bit in das Bit 2^0 geschrieben	59
RAR	00011111	– – – – *	1	4	4	Rotate accu right through carry	Akkumulator-Inhalt wird um 1 Bit nach rechts geschoben. Bit 2^0 wird in das Carry-Bit und das Carry-Bit in das Bit 2^7 geschrieben	59

b) Übertragsbit-Anweisungen

CMC	00111111	– – – – *	1	4	4	Complement carry	Carry-Bit wird negiert	60
STC	00110111	– – – – 1	1	4	4	Set carry	Carry-Bit wird gesetzt	60

Mnemonic	Binär-Code	Beeinflußte Zustands-Bits S Z AC P C	Bytes	Taktzyklen 8080	Taktzyklen 8085	Englische Befehlsbeschreibung	Funktion des Befehls	Siehe Seite

SPRUNGBEFEHLE

a) Unbedingte Sprünge

PCHL		11101001	– – – – –	1	5	6	H & L to program counter	Programm wird an der Adresse fortgesetzt, die im Registerpaar (H, L) steht	71
JMP	adr	11000011	– – – – –	3	10	10	Jump unconditional	Programm wird an der Adresse adr fortgesetzt	71

b) Bedingte Sprünge

JC	adr	11011010	– – – – –	3	10	7/10	Jump on carry	Bei Carry-Bit = 1 wird das Programm bei der Adresse adr fortgesetzt	71
JNC	adr	11010010	– – – – –	3	10	7/10	Jump on no carry	Bei Carry-Bit = 0 wird das Programm bei der Adresse adr fortgesetzt	71
JZ	adr	11001010	– – – – –	3	10	7/10	Jump on zero	Bei Zero-Bit = 1 wird das Programm bei der Adresse adr fortgesetzt	71
JNZ	adr	11000010	– – – – –	3	10	7/10	Jump on no zero	Bei Zero-Bit = 0 wird das Programm bei der Adresse adr fortgesetzt	71
JM	adr	11111010	– – – – –	3	10	7/10	Jump on minus	Bei Sign-Bit = 1 wird das Programm bei der Adresse adr fortgesetzt	71
JP	adr	11110010	– – – – –	3	10	7/10	Jump on positiv	Bei Sign-Bit = 0 wird das Programm bei der Adresse adr fortgesetzt	71
JPE	adr	11101010	– – – – –	3	10	7/10	Jump on parity even	Bei Parity-Bit = 1 wird das Programm bei der Adresse adr fortgesetzt	71
JPO	adr	11100010	– – – – –	3	10	7/10	Jump on parity odd	Bei Parity-Bit = 0 wird das Programm bei der Adresse adr fortgesetzt	71

Mnemonic	Binär-Code	Beeinflußte Zustands-Bits S Z AC P C	Bytes	Taktzyklen 8080	Taktzyklen 8085	Englische Befehlsbeschreibung	Funktion des Befehls	Siehe Seite

UNTERPROGRAMMBEHANDLUNG

a) Unterprogrammaufrufe

Bei allen Aufrufbefehlen wird die Rückkehradresse in dem Wort, das durch den Stackpointer adressiert ist, abgelegt

CALL adr	11001101	– – – – –	3	17	18	Call unconditional	Programm wird bei der Adresse adr fortgesetzt	76
CC adr	11011100	– – – – –	3	11/17	9/18	Call on carry	Bei Carry-Bit = 1 wird das Programm bei der Adresse adr fortgesetzt	76
CNC adr	11010100	– – – – –	3	11/17	9/18	Call on no carry	Bei Carry-Bit = 0 wird das Programm bei der Adresse adr fortgesetzt	76
CZ adr	11001100	– – – – –	3	11/17	9/18	Call on zero	Bei Zero-Bit = 1 wird das Programm bei der Adresse adr fortgesetzt	76
CNZ adr	11000100	– – – – –	3	11/17	9/18	Call on no zero	Bei Zero-Bit = 0 wird das Programm bei der Adresse adr fortgesetzt	76
CM adr	11111100	– – – – –	3	11/17	9/18	Call on minus	Bei Sign-Bit = 1 wird das Programm bei der Adresse adr fortgesetzt	76
CP adr	11110100	– – – – –	3	11/17	9/18	Call on positiv	Bei Sign-Bit = 0 wird das Programm bei der Adresse adr fortgesetzt	76
CPE adr	11101100	– – – – –	3	11/17	9/18	Call on parity even	Bei Parity-Bit = 1 wird das Programm bei der Adresse adr fortgesetzt	76
CPO adr	11100100	– – – – –	3	11/17	9/18	Call on parity odd	Bei Parity-Bit = 0 wird das Programm bei der Adresse adr fortgesetzt	76
RST konst	11nnn111	– – – – –	1	11	12	Restart	Programm wird bei der Adresse 8 × konst fortgesetzt ($0 \leq$ konst ≤ 7)	80

Mnemonic	Binär-Code	Beeinflußte Zustands-Bits S Z AC P C	Bytes	Taktzyklen 8080	Taktzyklen 8085	Englische Befehlsbeschreibung	Funktion des Befehls	Siehe Seite
b) Rücksprungbefehle								
RET	11001001	– – – – –	1	10	10	Return	Programm wird an der Adresse fortgesetzt, die in dem Wort steht, das über den Stackpointer adressiert ist	76
RC	11011000	– – – – –	1	5/11	6/12	Return on carry	Bei Carry-Bit = 1 wird das Programm an der Adresse fortgesetzt, die in dem über den Stackpointer adressierten Wort steht	76
RNC	11010000	– – – – –	1	5/11	6/12	Return on no carry	Bei Carry-Bit = 0 wird das Programm an der Adresse fortgesetzt, die in dem über den Stackpointer adressierten Wort steht	76
RZ	11001000	– – – – –	1	5/11	6/12	Return on zero	Bei Zero-Bit = 1 wird das Programm an der Adresse fortgesetzt, die in dem über den Stackpointer adressierten Wort steht	76
RNZ	11000000	– – – – –	1	5/11	6/12	Return on no zero	Bei Zero-Bit = 0 wird das Programm an der Adresse fortgesetzt, die in dem über den Stackpointer adressierten Wort steht	76
RM	11111000	– – – – –	1	5/11	6/12	Return on minus	Bei Sign-Bit = 1 wird das Programm an der Adresse fortgesetzt, die in dem über den Stackpointer adressierten Wort steht	76
RP	11110000	– – – – –	1	5/11	6/12	Return on positiv	Bei Sign-Bit = 0 wird das Programm an der Adresse fortgesetzt, die in dem über den Stackpointer adressierten Wort steht	76
RPE	11101000	– – – – –	1	5/11	6/12	Return on parity even	Bei Parity-Bit = 1 wird das Programm an der Adresse fortgesetzt, die in dem über den Stackpointer adressierten Wort steht	76
RPO	11100000	– – – – –	1	5/11	6/12	Return on parity odd	Bei Parity-Bit = 0 wird das Programm an der Adresse fortgesetzt, die in dem über den Stackpointer adressierten Wort steht	76

Mnemonic	Binär-Code	Beeinflußte Zustands-Bits S Z AC P C	Bytes	Taktzyklen 8080	Taktzyklen 8085	Englische Befehlsbeschreibung	Funktion des Befehls	Siehe Seite
PROGRAMMUNTERBRECHUNG								
EI	11111011	– – – – –	1	4	4	Enable interrupts	INTE-Flipflop wird gesetzt; der Mikroprozessor kann eine Unterbrechungsanforderung annehmen	81
DI	11110011	– – – – –	1	4	4	Disable interrupts	INTE-Flipflop wird rückgesetzt; der Mikroprozessor ignoriert Unterbrechungsanforderungen	81
SONSTIGE BEFEHLE								
HLT	01110110	– – – – –	1	7	5	Halt	Programm hält an, bis eine Unterbrechungsanforderung eintrifft	60
NOP	00000000	– – – – –	1	4	4	No operation	Leerbefehl	60
8085 BEFEHLE								
RIM	00100000	– – – – –	1	–	4	Read interrupt mask	Lesen Unterbrechungsmaske und seriellen Eingang in Akkumulator	–
SIM	00110000	– – – – –	1	–	4	Set interrupt mask	Setzen Unterbrechungsmaske und seriellen Ausgang	–

Tabelle 4-3
Mnemotechnische Operationscodes in alphabetischer Reihenfolge

Mnemonic	Binär-Code	Beeinflußte Zustands-Bits S Z AC P C	Bytes	Taktzyklen 8080	Taktzyklen 8085	Englische Befehlsbeschreibung	Funktion des Befehls	Siehe Seite
ACI konst	1 1 0 0 1 1 1 0	* * * * *	2	7	7	Add immediate to accu with carry	Zum Akkumulator-Inhalt werden konst ≦ 255 und Carry-Bit addiert	38
ADC M	1 0 0 0 1 1 1 0	* * * * *	1	7	7	Add memory to accu with carry	Inhalt des Speicherbytes, das durch den Inhalt des Registerpaares (H, L) adressiert ist und der Inhalt des Carry-Bits werden zum Inhalt des Akkumulators addiert	38
ADC r_1	1 0 0 0 1 s s s	* * * * *	1	4	4	Add register to accu with carry	r_1 = A, B, C, D, E, H oder L: Inhalt des Registers r_1 und Inhalt des Carry-Bits werden zum Inhalt des Akkumulators addiert	38
ADD r_1	1 0 0 0 0 s s s	* * * * *	1	4	4	Add register to accu	r_1 = A, B, C, D, E, H oder L: Inhalt von Register r_1 wird zum Inhalt des Akkumulators addiert	38
ADD M	1 0 0 0 0 1 1 0	* * * * *	1	7	7	Add memory to accu	Der Inhalt des Speicherbytes, das durch den Inhalt des Registerpaares (H, L) adressiert ist, wird zum Inhalt des Akkumulators addiert	38
ADI konst	1 1 0 0 0 1 1 0	* * * * *	2	7	7	Add immediate to accu	Konstante (konst ≦ 255) wird zum Inhalt des Akkumulators addiert	38
ANA r_1 für 8085	1 0 1 0 0 s s s	* * * * 0 * * 1 * 0	1	4	4	And register with accu	r_1 = A, B, C, D, E, H oder L: Akkumulator-Inhalt und der Inhalt des Registers r_1 werden UND-verknüpft	55
ANA M für 8085	1 0 1 0 0 1 1 0	* * * * 0 * * 1 * 0	1	7	7	And memory with accu	Der Inhalt des durch Registerpaar (H, L) adressierten Bytes wird mit dem Akkumulator-Inhalt UND-verknüpft	55
ANI konst für 8085	1 1 1 0 0 1 1 0	* * * * 0 * * 1 * 0	2	7	7	And immediate with accu	Akkumulator-Inhalt wird mit der konst ≦ 255 UND verknüpft	55
CALL adr	1 1 0 0 1 1 0 1	– – – – –	3	17	18	Call unconditional	Programm wird bei der Adresse adr fortgesetzt	76
CC adr	1 1 0 1 1 1 0 0	– – – – –	3	11/17	9/18	Call on carry	Bei Carry-Bit = 1 wird das Programm bei der Adresse adr fortgesetzt	76
CM adr	1 1 1 1 1 1 0 0	– – – – –	3	11/17	9/18	Call on minus	Bei Sign-Bit = 1 wird das Programm bei der Adresse adr fortgesetzt	76
CMA	0 0 1 0 1 1 1 1	– – – – –	1	4	4	Complement accu	Akkumulator-Inhalt wird negiert	58
CMC	0 0 1 1 1 1 1 1	– – – – *	1	4	4	Complement carry	Carry-Bit wird negiert	60

Mnemonic	Binär-Code	Beeinflußte Zustands-Bits S Z AC P C	Bytes	Taktzyklen 8080	Taktzyklen 8085	Englische Befehlsbeschreibung	Funktion des Befehls	Siehe Seite
CMP r_1	1 0 1 1 1 s s s	* * * * *	1	4	4	Compare register with accu	r_1 = A, B, C, D, E, H oder L: Akkumulator-Inhalt wird mit dem Inhalt des Registers r_1 verglichen	57
CMP M	1 0 1 1 1 1 1 0	* * * * *	1	7	7	Compare memory with accu	Akkumulator-Inhalt wird mit dem Inhalt des durch Registerpaar (H, L) adressierten Bytes verglichen	57
CNC adr	1 1 0 1 0 1 0 0	– – – – –	3	11/17	9/18	Call on no carry	Bei Carry-Bit = 0 wird das Programm bei der Adresse adr fortgesetzt	76
CNZ adr	1 1 0 0 0 1 0 0	– – – – –	3	11/17	9/18	Call on no zero	Bei Zero-Bit = 0 wird das Programm bei der Adresse adr fortgesetzt	76
CP adr	1 1 1 1 0 1 0 0	– – – – –	3	11/17	9/18	Call on positiv	Bei Sign-Bit = 0 wird das Programm bei der Adresse adr fortgesetzt	76
CPE adr	1 1 1 0 1 1 0 0	– – – – –	3	11/17	9/18	Call on parity even	Bei Parity-Bit = 1 wird das Programm bei der Adresse adr fortgesetzt	76
CPI konst	1 1 1 1 1 1 1 0	* * * * *	2	7	7	Compare immediate	Akkumulator-Inhalt wird mit konst $\leqq$ 255 verglichen	57
CPO adr	1 1 1 0 0 1 0 0	– – – – –	3	11/17	9/18	Call on parity odd	Bei Parity-Bit = 0 wird das Programm bei der Adresse adr fortgesetzt	76
CZ adr	1 1 0 0 1 1 0 0	– – – – –	3	11/17	9/18	Call on zero	Bei Zero-Bit = 1 wird das Programm bei der Adresse adr fortgesetzt	76
DAA	0 0 1 0 0 1 1 1	* * * * *	1	4	4	Decimal adjust accu	Akkumulator-Inhalt wird in eine 2stellige Zahl umgewandelt	58
DAD rp	0 0 r r 1 0 0 1	– – – – *	1	10	10	Add register pair to H and L	rp = B, D, H, SP: Inhalt des Registerpaares rp, und der Inhalt des Registerpaares (H, L) werden addiert. Ergebnis in (H, L)	48
DCR r_1	0 0 d d d 1 0 1	* * * * –	1	5	4	Decrement register	r_1 = A, B, C, D, E, H oder L: Vom Inhalt des Registers r_1 wird 1 subtrahiert	38
DCR M	0 0 1 1 0 1 0 1	* * * * –	1	10	10	Decrement memory	Vom Inhalt des durch Registerpaar (H, L) adressierten Bytes wird 1 subtrahiert	38
DCX rp	0 0 r r 1 0 1 1	– – – – –	1	5	6	Decrement register pair	rp = B, D, H, SP: Der Inhalt des Registerpaares rp wird um 1 erniedrigt	48

Mnemonic	Binär-Code	Beeinflußte Zustands-Bits S Z AC P C	Bytes	Taktzyklen 8080	Taktzyklen 8085	Englische Befehlsbeschreibung	Funktion des Befehls	Siehe Seite
DI	1 1 1 1 0 0 1 1	– – – – –	1	4	4	Disable interrupts	INTE-Flipflop wird rückgesetzt; der Mikroprozessor ignoriert Unterbrechungsanforderungen	81
EI	1 1 1 1 1 0 1 1	– – – – –	1	4	4	Enable interrupts	INTE-Flipflop wird gesetzt; der Mikroprozessor kann eine Unterbrechungsanforderung annehmen	81
HLT	0 1 1 1 0 1 1 0	– – – – –	1	7	5	Halt	Programm hält an, bis eine Unterbrechungsanforderung eintrifft	60
IN nr	1 1 0 1 1 0 1 1	– – – – –	2	10	10	Input	Akkumulator wird mit dem Inhalt des Eingabekanals (Nummer nr $\leqq$ 255) geladen	27
INR r_1	0 0 d d d 1 0 0	* * * * –	1	5	4	Increment register	r_1 = A, B, C, D, E, H oder L: Zum Inhalt des Registers r_1 wird 1 addiert	38
INR M	0 0 1 1 0 1 0 0	* * * * –	1	10	10	Increment memory	Zum Inhalt des durch Registerpaar (H, L) adressierten Bytes wird 1 addiert	38
INX rp	0 0 r r 0 0 1	– – – – –	1	5	6	Increment register pair	rp = B, D, H, SP: Der Inhalt des Registerpaares rp wird um 1 erhöht	48
JC adr	1 1 0 1 1 0 1 0	– – – – –	3	10	7/10	Jump on carry	Bei Carry-Bit = 1 wird das Programm bei der Adresse adr fortgesetzt	71
JM adr	1 1 1 1 1 0 1 0	– – – – –	3	10	7/10	Jump on minus	Bei Sign-Bit = 1 wird das Programm bei der Adresse adr fortgesetzt	71
JMP adr	1 1 0 0 0 0 1 1	– – – – –	3	10	10	Jump unconditional	Programm wird an der Adresse adr fortgesetzt	71
JNC adr	1 1 0 1 0 0 1 0	– – – – –	3	10	7/10	Jump on no carry	Bei Carry-Bit = 0 wird das Programm bei der Adresse adr fortgesetzt	71
JNZ adr	1 1 0 0 0 0 1 0	– – – – –	3	10	7/10	Jump on no zero	Bei Zero-Bit = 0 wird das Programm bei der Adresse adr fortgesetzt	71
JP adr	1 1 1 1 0 0 1 0	– – – – –	3	10	7/10	Jump on positiv	Bei Sign-Bit = 0 wird das Programm bei der Adresse adr fortgesetzt	71
JPE adr	1 1 1 0 1 0 1 0	– – – – –	3	10	7/10	Jump on parity even	Bei Parity-Bit = 1 wird das Programm bei der Adresse adr fortgesetzt	71

Mnemonic	Binär-Code	Beeinflußte Zustands-Bits S Z AC P C	Bytes	Taktzyklen 8080	Taktzyklen 8085	Englische Befehlsbeschreibung	Funktion des Befehls	Siehe Seite
JPO adr	1 1 1 0 0 0 1 0	– – – – –	3	10	7/10	Jump on parity odd	Bei Parity-Bit = 0 wird das Programm bei der Adresse adr fortgesetzt	71
JZ adr	1 1 0 0 1 0 1 0	– – – – –	3	10	7/10	Jump on zero	Bei Zero-Bit = 1 wird das Programm bei der Adresse adr fortgesetzt	71
LDA adr	0 0 1 1 1 0 1 0	– – – – –	3	13	13	Load accu direct	Akkumulator laden mit Inhalt der Adresse adr	30
LDAX rp	0 0 r r 1 0 1 0	– – – – –	1	7	7	Load accu indirect	rp = B, D: Akkumulator laden mit Inhalt des Speicherplatzes, der durch den Inhalt des Registerpaares rp adressiert ist	31
LHLD adr	0 0 1 0 1 0 1 0	– – – – –	3	16	16	Load H & L direct	Lade Registerpaar (H, L) mit dem Inhalt der Adressen adr und (adr+1)	35
LXI rp, adr	0 0 r r 0 0 0 1	– – – – –	3	10	10	Load register pair immediate	rp = B, D, H, SP: Lade Registerpaar rp mit Wert adr	34
MOV M, r_1	0 1 1 1 0 s s s	– – – – –	1	7	7	Move register to memory	r_1 = A, B, C, D, E, H oder L: Inhalt von Register r_1 auf den Speicherplatz abspeichern, der durch den Inhalt des Registerpaares (H, L) adressiert ist	29
MOV r_1, M	0 1 d d d 1 1 0	– – – – –	1	7	7	Move memory to register	r_1 = A, B, C, D, E, H oder L: Lade Register r_1 mit dem Inhalt des Speicherbytes, das durch den Inhalt des Registerpaares (H, L) adressiert ist	29
MOV r_1, r_2	0 1 d d d s s s	– – – – –	1	5	4	Move register to register	r_1, r_2 = A, B, C, D, E oder L: Lade Register r_1 mit dem Inhalt von Register r_2	29
MVI M, konst	0 0 1 1 0 1 1 0	– – – – –	2	10	10	Move to memory immediate	Lade den Speicherplatz, der durch den Inhalt des Registerpaares (H, L) adressiert ist, mit Konstante (konst ≦ 255)	28
MVI r_1, konst	0 0 d d d 1 1 0	– – – – –	2	7	7	Move to register immediate	r_1 = A, B, C, D, E, H oder L: Lade Register r_1 mit Konstante (konst ≦ 255)	28
NOP	0 0 0 0 0 0 0 0	– – – – –	1	4	4	No operation	Leerbefehl	60
ORA r_1	1 0 1 1 0 s s s	* * 0 * 0	1	4	4	Or register with accu	r_1 = A, B, C, D, E, H oder L: Akkumulator-Inhalt wird mit dem Inhalt des Registers r_1 ODER-verknüpft	55

Mnemonic	Binär-Code	Beeinflußte Zustands-Bits S Z AC P C	Bytes	Taktzyklen 8080	Taktzyklen 8085	Englische Befehlsbeschreibung	Funktion des Befehls	Siehe Seite
ORA M	10110110	* * 0 * 0	1	7	7	Or memory with accu	Inhalt des über Registerpaar (H, L) adressierten Bytes wird mit dem Akkumulator-Inhalt ODER-verknüpft	55
ORI konst	11110110	* * 0 * 0	2	7	7	Or immediate with accu	Akkumulator-Inhalt wird mit konst $\leq$ 255 ODER-verknüpft	55
OUT nr	11010011	– – – – –	2	10	10	Output	Akkumulator-Inhalt wird auf Ausgabekanal (Nummer nr $\leq$ 255) ausgegeben	27
PCHL	11101001	– – – – –	1	5	6	H & L to program counter	Programm wird an der Adresse fortgesetzt, die im Registerpaar (H, L) steht	71
POP rp POP PSW	11 r r 0 0 0 1 11110001	– – – – – * * * * *	1	10	10	Pop register pair off stack	rp = B, D, H, PSW: Registerpaar rp wird mit dem Wort geladen, das durch den Stackpointer adressiert ist	78
PUSH rp	11 r r 0 1 0 1	– – – – –	1	11	12	Push register pair rp on stack	rp = B, D, H, PSW: Inhalt des Registerpaares rp wird in das Wort übertragen, das durch den Stackpointer adressiert ist	78
RAL	00010111	– – – – *	1	4	4	Rotate accu left through carry	Akkumulator-Inhalt wird um 1 Bit nach links geschoben. Bit 2^7 wird in das Carry-Bit und das Carry-Bit in das Bit 2^0 geschrieben	59
RAR	00011111	– – – – *	1	4	4	Rotate accu right through carry	Akkumulator-Inhalt wird um 1 Bit nach rechts geschoben. Bit 2^0 wird in das Carry-Bit und das Carry-Bit in das Bit 2^7 geschrieben	59
RC	11011000	– – – – –	1	5/11	6/12	Return on carry	Bei Carry-Bit = 1 wird das Programm an der Adresse fortgesetzt, die in dem über den Stackpointer adressierten Wort steht	76
RET	11001001	– – – – –	1	10	10	Return	Programm wird an der Adresse fortgesetzt, die in dem Wort steht, das über den Stackpointer adressiert ist	76
RIM	00100000	– – – – –	1	–	4	Read interrupt mask	Lesen Unterbrechungsmaske und seriellen Eingang in Akkumulator	–
RLC	00000111	– – – – *	1	4	4	Rotate accu left	Akkumulator-Inhalt wird um 1 Bit nach links verschoben. Bit 2^7 wird in das Carry-Bit geschrieben, Bit 2^7 nach Bit 2^0	59
RM	11111000	– – – – –	1	5/11	6/12	Return on minus	Bei Sign-Bit = wird das Programm an der Adresse fortgesetzt, die in dem über den Stackpointer adressierten Wort steht	76

Mnemonic	Binär-Code	Beeinflußte Zustands-Bits S Z AC P C	Bytes	Taktzyklen 8080	Taktzyklen 8085	Englische Befehlsbeschreibung	Funktion des Befehls	Siehe Seite
RNC	11010000	– – – – –	1	5/11	6/12	Return on no carry	Bei Carry-Bit = 0 wird das Programm an der Adresse fortgesetzt, die in dem über dem Stackpointer adressierten Wort steht	76
RNZ	11000000	– – – – –	1	5/11	6/12	Return on no zero	Bei Zero-Bit = 0 wird das Programm an der Adresse fortgesetzt, die in dem über dem Stackpointer adressierten Wort steht	76
RP	11110000	– – – – –	1	5/11	6/12	Return on positiv	Bei Sign-Bit = 0 wird das Programm an der Adresse fortgesetzt, die in dem über dem Stackpointer adressierten Wort steht	76
RPE	11101000	– – – – –	1	5/11	6/12	Return on parity even	Bei Paritäts-Bit = 1 wird das Programm an der Adresse fortgesetzt, die in dem über dem Stackpointer adressierten Wort steht	76
RPO	11100000	– – – – –	1	5/11	6/12	Return on parity odd	Bei Paritäts-Bit = 0 wird das Programm an der Adresse fortgesetzt, die in dem über dem Stackpointer adressierten Wort steht	76
RRC	00001111	– – – – *	1	4	4	Rotate accu right	Akkumulator-Inhalt wird um 1 Bit nach rechts verschoben. Bit 2^0 wird in das Carry-Bit geschrieben, Bit 2^0 nach Bit 2^7	59
RST konst	11nnn111	– – – – –	1	11	12	Restart	$0 \leq$ konst ≤ 7 Programm wird auf der Adresse $8 \times$ konst fortgesetzt	80
RZ	11001000	– – – – –	1	5/11	6/12	Return on zero	Bei Zero-Bit = 1 wird das Programm an der Adresse fortgesetzt, die in dem über dem Stackpointer adressierten Wort steht	76
SBB r_1	10011sss	* * * * *	1	4	4	Subtract register from accu with borrow	r_1 = A, B, C, D, E, H oder L: Inhalt von Register r_1 und Inhalt des Carry-Bits werden vom Akkumulator-Inhalt subtrahiert	38
SBB M	10011110	* * * * *	1	7	7	Subtract memory from accu with borrow	Inhalt des Speicherbytes, das durch das Registerpaar (H, L) adressiert ist, und Inhalt des Carry-Bits werden vom Akkumulator subtrahiert	38
SBI konst	11011110	* * * * *	2	7	7	Subtract immediate from accu with borrow	konst ≤ 255 und der Inhalt des Carry-Bits werden vom Akkumulator-Inhalt subtrahiert	38

Mnemonic	Binär-Code	Beeinflußte Zustands-Bits S Z AC P C	Bytes	Taktzyklen 8080	Taktzyklen 8085	Englische Befehlsbeschreibung	Funktion des Befehls	Siehe Seite
SHLD adr	00100010	– – – – –	3	16	16	Store H & L direct	Registerpaar (H, L) unter Adresse adr u. (adr + 1) abspeichern	36
SIM	00110000	– – – – –	1	–	4	Set interrupt mask	Setzen Unterbrechungsmaske und seriellen Ausgang	–
SPHL	11111001	– – – – –	1	5	6	H & L to stackpointer	Lade Stackpointer mit dem Inhalt des Registerpaares (H, L)	80
STA adr	00110010	– – – – –	3	13	13	Store accu direct	Akkumulator Inhalt unter Adresse adr abspeichern	31
STAX rp	00rr0010	– – – – –	1	7	7	Store accu indirect	rp = B, D: Akkumulator in dem Byte abspeichern, das durch den Inhalt des Registerpaares rp adressiert ist	31
STC	00110111	– – – – 1	1	4	4	Set carry	Carry-Bit wird gesetzt	60
SUB r_1	10010sss	* * * * *	1	4	4	Subtract register from accu	r_1 = A, B, C, D, E, H oder L: Inhalt des Registers r_1 wird vom Akkumulator-Inhalt subtrahiert	38
SUB M	10010110	* * * * *	1	7	7	Subtract memory from accu	Inhalt des Speicherbytes, das durch das Registerpaar (H, L) adressiert ist, wird vom Akkumulator subtrahiert	38
SUI konst	11010110	* * * * *	2	7	7	Subtract immediate from accu	konst $\geqq$ 255 wird vom Akkumulator-Inhalt subtrahiert	38
XCHG	11101011	– – – – –	1	4	4	Exchange D & E, H & L	Vertausche Inhalte der Registerpaare (D, E) und (H, L)	36
XRA r_1	10101sss	* * 0 * 0	1	4	4	Exclusive Or register with accu	r_1 = A, B, C, D, E, H oder L: Akkumulator-Inhalt wird mit dem Inhalt des Registers r_1 Exklusiv-ODER-verknüpft	55
XRA M	10101110	* * 0 * 0	1	7	7	Exclusive Or memory with accu	Das über Registerpaar (H, L) adressierte Byte wird durch Exklusiv-ODER mit dem Akkumulator verknüpft	55
XRI konst	11101110	* * 0 * 0	2	7	7	Exclusive Or immediate with accu	Der Akkumulator wird mit dem Wert konst $\leqq$ 255 durch Exklusiv-ODER verknüpft	55
XTHL	11100011	– – – – –	1	18	16	Exchange top of stack and H & L	Vertausche Inhalt des Registerpaares (H, L) und den Inhalt des Wortes, das durch den Stackpointer adressiert ist	80

Tabelle 4-4
Befehlsliste in der Reihenfolge der hexadezimalen Abkürzungen

00	NOP		2B	DCX	H	56	MOV	D,M	81	ADD	C	AC	XRA	H	D7	RST	2
01	LXI	B,D16	2C	INR	L	57	MOV	D,A	82	ADD	D	AD	XRA	L	D8	RC	
02	STAX	B	2D	DCR	L	58	MOV	E,B	83	ADD	E	AE	XRA	M	D9	...	
03	INX	B	2E	MVI	L,D8	59	MOV	E,C	84	ADD	H	AF	XRA	A	DA	JC	Adr
04	INR	B	2F	CMA		5A	MOV	E,D	85	ADD	L	B0	ORA	B	DB	IN	D8
05	DCR	B	30	...		5B	MOV	E,E	86	ADD	M	B1	ORA	C	DC	CC	Adr
06	MVI	B,D8	31	LXI	SP,D16	5C	MOV	E,H	87	ADD	A	B2	ORA	D	DD	...	
07	RLC		32	STA	Adr	5D	MOV	E,L	88	ADC	B	B3	ORA	E	DE	SBI	D8
08	...		33	INX	SP	5E	MOV	E,M	89	ADC	C	B4	ORA	H	DF	RST	3
09	DAD	B	34	INR	M	5F	MOV	E,A	8A	ADC	D	B5	ORA	L	E0	RPO	
0A	LDAX	B	35	DCR	M	60	MOV	H,B	8B	ADC	E	B6	ORA	M	E1	POP	H
0B	DCX	B	36	MVI	M,D8	61	MOV	H,C	8C	ADC	H	B7	ORA	A	E2	JPO	Adr
0C	INR	C	37	STC		62	MOV	H,D	8D	ADC	L	B8	CMP	B	E3	XTHL	
0D	DCR	C	38	...		63	MOV	H,E	8E	ADC	M	B9	CMP	C	E4	CPO	Adr
0E	MVI	C,D8	39	DAD	SP	64	MOV	H,H	8F	ADC	A	BA	CMP	D	E5	PUSH	H
0F	RRC		3A	LDA	Adr	65	MOV	H,L	90	SUB	B	BB	CMP	E	E6	ANI	D8
10	...		3B	DCX	SP	66	MOV	H,M	91	SUB	C	BC	CMP	H	E7	RST	4
11	LXI	D,D16	3C	INR	A	67	MOV	H,A	92	SUB	D	BD	CMP	L	E8	RPE	
12	STAX	D	3D	DCR	A	68	MOV	L,B	93	SUB	E	BE	CMP	M	E9	PCHL	
13	INX	D	3E	MVI	A,D8	69	MOV	L,C	94	SUB	H	BF	CMP	A	EA	JPE	Adr
14	INR	D	3F	CMC		6A	MOV	L,D	95	SUB	L	C0	RNZ		EB	XCHG	
15	DCR	D	40	MOV	B,B	6B	MOV	L,E	96	SUB	M	C1	POP	B	EC	CPE	Adr
16	MVI	D,D8	41	MOV	B,C	6C	MOV	L,H	97	SUB	A	C2	JNZ	Adr	ED	...	
17	RAL		42	MOV	B,D	6D	MOV	L,L	98	SBB	B	C3	JMP	Adr	EE	XRI	D8
18	...		43	MOV	B,E	6E	MOV	L,M	99	SBB	C	C4	CNZ	Adr	EF	RST	5
19	DAD	D	44	MOV	B,H	6F	MOV	L,A	9A	SBB	D	C5	PUSH	B	F0	RP	
1A	LDAX	D	45	MOV	B,L	70	MOV	M,B	9B	SBB	E	C6	ADI	D8	F1	POP	PSW
1B	DCX	D	46	MOV	B,M	71	MOV	M,C	9C	SBB	H	C7	RST	0	F2	JP	Adr
1C	INR	E	47	MOV	B,A	72	MOV	M,D	9D	SBB	L	C8	RZ		F3	DI	
1D	DCR	E	48	MOV	C,B	73	MOV	M,E	8E	SBB	M	C9	RET	Adr	F4	CP	Adr
1E	MVI	E,D8	49	MOV	C,C	74	MOV	M,H	9F	SBB	A	CA	JZ		F5	PUSH	PSW
1F	RAR		4A	MOV	D,D	75	MOV	M,L	A0	ANA	B	CB	...		F6	ORI	D8
20	...		4B	MOV	C,E	76	HLT		A1	ANA	C	CC	CZ	Adr	F7	RST	6
21	LXI	H,D16	4C	MOV	C,H	77	MOV	M,A	A2	ANA	D	CD	CALL	Adr	F8	RM	
22	SHLD	Adr	4D	MOV	C,L	78	MOV	A,B	A3	ANA	E	CE	ACI	D8	F9	SPHL	
23	INX	H	4E	MOV	C,M	79	MOV	A,C	A4	ANA	H	CF	RST	1	FA	JM	Adr
24	INR	H	4F	MOV	C,A	7A	MOV	A,D	A5	ANA	L	D0	RNC		FB	EI	
25	DCR	H	50	MOV	D,B	7B	MOV	A,E	A6	ANA	M	D1	POP	D	FC	CM	Adr
26	MVI	H,D8	51	MOV	D,C	7C	MOV	A,H	A7	ANA	A	D2	JNC	Adr	FD	...	
27	DAA		52	MOV	D,D	7D	MOV	A,L	A8	XRA	B	D3	OUT	D8	FE	CPI	D8
28	...		53	MOV	D,E	7E	MOV	A,M	A9	XRA	C	D4	CNC	Adr	FF	RST	7
29	DAD	H	54	MOV	D,H	7F	MOV	A,A	AA	XRA	D	D5	PUSH	D			
2A	LHLD	Adr	55	MOV	D,L	80	ADD	B	AB	XRA	E	D6	SUI	D8			

D8 = Konstante oder logischer/arithmetischer Ausdruck, der eine 8-Bit-Datengröße darstellt.
D16 = Konstante oder logischer/arithmetischer Ausdruck, der eine 16-Bit-Datengroße darstellt.
Adr = 16-Bit-Adresse

Unsere Geschäftsstellen mit Bauteile-Vertrieb

Bundesrepublik Deutschland und Berlin (West)

Siemens AG
Salzufer 6–8
1000 Berlin 10
☎ (030) 3939-1, Tx 1810-278
FAX (030) 3939-2630

Siemens AG
Schweriner Straße 1
Postfach 7820
4800 Bielefeld 1
☎ (0521) 291-1, Tx 932805
FAX (0521) 291-375

Siemens AG
Contrescarpe 72
Postfach 107827
2800 Bremen
☎ (0421) 364-1, Tx 245451
FAX (0421) 364-687

Siemens AG
Lahnweg 10
Postfach 1115
4000 Düsseldorf 1
☎ (0211) 3030-1, Tx 8581301
FAX (0211) 3030-506

Siemens AG
Rödelheimer Landstraße 5–9
Postfach 111733
6000 Frankfurt 1
☎ (0611) 797-0, Tx 414131
FAX (0611) 797-2253

Siemens AG
Habsburgerstraße 132
Postfach 1380
7800 Freiburg 1
☎ (0761) 2712-1
Tx 772842

Siemens AG
Lindenplatz 2
Postfach 105609
2000 Hamburg 1
☎ (040) 282-1, Tx 215584-0
FAX (040) 282-2210

Siemens AG
Am Maschpark 1
Postfach 5329
3000 Hannover 1
☎ (0511) 199-1, Tx 922333
FAX (0511) 199-2799

Siemens AG
Wittland 2
Postfach 4049
2300 Kiel 1
☎ (0431) 5860-1
Tx 292814

Siemens AG
N 7, 18 (Siemenshaus)
Postfach 2024
6800 Mannheim 1
☎ (0621) 296-1, Tx 462261
FAX (0621) 296-222

Siemens AG
Richard-Strauss-Straße 76
Postfach 202109
8000 München
☎ (089) 9221-0
Tx 0529421-01
FAX (089) 9221-4499

Siemens AG
Von-der-Tann-Straße 30
Postfach 4844
8500 Nürnberg 1
☎ (0911) 654-1, Tx 622251
FAX (0911) 654-3436,
34614, 3716

Siemens AG
Geschwister-Scholl-Straße 24
Postfach 120
7000 Stuttgart 1
☎ (0711) 2076-1, Tx 723941-0
FAX (0711) 2076-706

Siemens AG
Nicolaus-Otto-Straße 4
Postfach 3606
7900 Ulm 1
☎ (0731) 499-1
Tx 712826

Siemens AG
Andreas-Grieser-Str. 30
Postfach 3280
8700 Würzburg 21
☎ (0931) 801-1
Tx 68844

Siemens Bauteile Service
Lieferzentrum Fürth
Postfach 146
8510 Fürth-Bislohe
☎ (0911) 3001-1, Tx 623818

EUROPA

Belgien
Siemens S.A.
chaussée de Charleroi 116
B-1060 Bruxelles
☎ (02) 5373100, Tx 21347

Bulgarien
RUEN,
Büro für Firmenvertretungen und
Handelsvermittlungen bei der
Vereinigung „Interpred"
San Stefano 14/16
BG-1504 Sofia 4
☎ 457082, Tx 22763

Dänemark
Siemens A/S
Borupvang 3
DK-2750 Ballerup
☎ (02) 656565, Tx 35313

Finnland
Siemens Osakeyhtiö
Mikonkatu 8
Fach 8
SF-00101 Helsinki 10
☎ (90), 1626-1, Tx 124465

Frankreich
Siemens S.A.
B.P. 109
F-93203 Saint-Denis CEDEX 1
☎ (16-1) 8206120, Tx 620853

Griechenland
Siemens Hellas E.A.E.
Voulis 7
P.O.B. 601
Athen 125
☎ (01) 3293-1, Tx 216291

Großbritannien
Siemens Ltd.
Siemens House
Windmill Road
Sunburry-on-Thames
Middlesex TW 16 7HS
☎ (09327) 85691, Tx 8951091

Irland
Siemens Limited
8, Raglan Road
Dublin 4
☎ (01) 684727, Tx 5341

Island
Smith & Norland H/F
Nóatún 4
P.O.B. 519
IS-121 Reykjavik
☎ 28322, Tx 2055

Italien
Siemens Elettra S.p.A.
Via Fabio Filzi, K 25/A
Casella Postale 10388
I-20100 Milano
☎ (02) 6248, Tx 330261

Jugoslawien
Generalexport
Ul. Narodnih heroja 43/XV
YU-11070 Novi Beograd
☎ (011) 693-321, Tx 11287

Luxemburg
Siemens S.A.
17, rue Glesener
B.P. 1701
Luxembourg
☎ 49711-1, Tx 3430

Niederlande
Siemens Nederland N.V.
Postb. 16068
NL-2500 BB Den Haag
☎ (070) 782782, Tx 31373

Norwegen
Siemens A/S
Østre Aker vei 90
Postboks 10, Veitvet
N-Oslo 5
☎ (02) 153090, Tx 18477

Österreich
Siemens Aktiengesellschaft
Österreich
Postfach 326
A-1031 Wien
☎ (0222) 7293-0, Tx 131866

Polen
PHZ Transactor S.A.
ul. Stawki 2
P.O.B. 276
PL-00-950 Warszawa
☎ 398910, Tx 815554

Portugal
Siemens S.A.R.L.
Avenida Almirante Reis, 65
Apartado 1380
P-1100 Lisboa-1
☎ (019) 538805, Tx 12563

Rumänien
Siemens birou
de consultaţii tehnice
Strada Edgar Quinet Nr. 1
R-70106 Bucureşti 1
☎ 151825, Tx 11473

Schweden
Siemens AB
Norra Stationsgatan 63-65
Box 23141
S-10435 Stockholm
☎ (08) 161100, Tx 11672

Schweiz
Siemens-Albis AG
Freilagerstraße 28
Postfach
CH-8047 Zürich
☎ (01) 495-3111, Tx 52131

Spanien
Siemens S.A.
Órense, 2
Apartado 155
Madrid 20
☎ (91) 4552500, Tx 42241

Tschechoslowakei
EFEKTIM,
Technisches Beratungsbüro
Siemens AG
Anglická ulice 22, 3. Stock
P.O.B. 1087
CS-12000 Praha 2
☎ 258417, Tx 122389

Ungarn
Sicontact KFT GmbH
Bártfai u. 54
H-1115 Budapest XI
☎ (01) 868044, Tx 224133

Union der Sozialistischen Sowjetrepubliken
Ständige Vertretung der
Siemens AG in Moskau
Internationales Postamt
Postfach 77
SU-Moskau G 34
☎ 2027711, Tx 7413

AFRIKA

Ägypten
Siemens Resident Engineers
26, El Batal Abdel Aziz Street
P.O. Box 775
Cairo-Mohandessin
Arab Republik Egypt
☎ 705673, Tx 93199

Äthiopien
Addis Electrical Engineering Ltd.
P.O.B. 5505
Addis Ababa
☎ 151599, Tx 21052

Algerien
Siemens Algérie S.A.R.L.
3, Viaduc Youghourta
B.P. 224, Alger-Gare
Alger
☎ 615966/67, Tx 52817

Libyen
Siemens Resident Engineers
17, First September Street
P.O.B. 46
Tripoli
☎ 41534, Tx 20029

Marokko
SETEL
Société Electrotechnique
et de Télécommunications S.A.
Immeuble Siemens
km 1, Route de Rabat
Casablanca-Ain Sebâa
☎ 351025, Tx 25914

Nigeria
Siemens Nigeria Ltd.
Siemens House
Industrial estate 3 f,
Block A
P.O.B. 304, Apapa
Oshodi (Lagos)
☎ 842502, Tx 21357

Sudan
National Electrical
& Commercial Company (NECC)
P.O.B. 1202
Khartoum
Republic of Sudan
☎ 80818, Tx 642

Südafrika
Siemens Limited
Siemens House,
P.O.B. 4583
Johannesburg 2000
☎ (011) 7159111, Tx 22524

Tunesien
Sitelec S.A.,
Immeuble Saâdi - Tour C
Route de l'Ariana
Tunis-El Menzah TN
☎ 231526, Tx 12326

Zaire
Siemens Zaire S.A.R.L.
B.P. 9897
6e rue Limité
Kinshasa 1
☎ 77206, Tx 21377

AMERIKA

Argentinien
Siemens S.A.
Avenida Pte. Julio A. Roca 516
Casilla Correo Central 1232
RA-1000 Buenos Aires
☎ 00541/300411, Tx 021812

Bolivien
Sociedad Comercial é Industrial
Hansa Limitada
CalleMercadoesquinaYanacocha
Cajón Postal 1402
La Paz
☎ 320289, Tx 5261

Brasilien
Siemens S.A.
Sede Central
Caixa Postal 1375,
01000 São Paulo-SP
☎ (011) 2610211
Tx 11-23641

Chile
Gildemeister S.A.C.,
Division Siemens
Huerfanos 587
Santiago de Chile
☎ 82523,
Tx TRA SGO 392, TDE 40589
FAX 393421

Ecuador
Siemens S.A.
Panamericana Norte y
Manuel Zambrano
Casilla de Correos 3580
Quito
☎ 537666, Tx 22190

Kanada
Siemens Electric Limited
7300 Trans-Canada Highway
P.O.B. 7300, Pointe Claire,
Québec H9R 4R6
☎ (514) 6957300, Tx 5-822778

Kolumbien
Siemens S.A.
Carrera 65, No. 11–83
Apartado Aéreo 80150
Bogotá 6
☎ 2628811, Tx 44750

Mexico
Siemens S.A.
Poniente 116, No. 590
Col. Pro-Hogar
Apartado Postal 15064
02600 México, D.F.
☎ 5670722, Tx 1772700

Uruguay
Conatel S.A.
Ejido 1690
Casilla de Correo 1371
Montevideo
☎ 917331, Tx 6664

Venezuela
Siemens S.A.
Avenida Don Diego Cisneros
Urbanización los Ruices
Apartado 3616
Caracas 1010 A
☎ (02) 2392133, Tx 25131

Vereinigte Staaten von Amerika
Siemens Corporation
186 Wood Avenue South
Iselin, New Jersey 08830
☎ (201) 3400
Tx WU 844491
TWX WU 7109980588

ASIEN

Afghanistan
Afghan Electrical Engineering and Equipment Limited
Alaudin, Karte 3
P.O.B. 7
Kabul 1
☎ 40446, Tx 35

Bangladesch
Siemens Bangladesh Ltd.
74, Diskusha Commercial Area
P.O.B. 33
Dacca 2
☎ 231381, Tx 642424 bj

Hongkong
Jebsen & Co., Ltd.
Siemens Division
Prince's Building, 24th floor
P.O.B. 97
Hong Kong
☎ 5225111, Tx 73221

Indien
Siemens India Ltd.
Head Office
134-A, Dr. Annie Besant Road, Worli
P.O.B. 6597
Bombay 400018
☎ 379906, Tx 112373

Indonesien
Repräsentative Siemens AG
Jl. Kebon Sirih 4
P.O.B. 2469
Jakarta Pusat
☎ 351051, Tx 46222

Irak
Siemens Iraq Branch
P.O.B. 3120
Baghdad
☎ 98198, Tx 2393

Iran
Siemens Sherkate Sahami Khass
Ave. Ayatolla Taleghani 32
Siemenshaus
Teheran 15
☎ (021) 614-1, Tx 212351

Japan
Siemens K.K.
Delegates to Fuji Electric
c/o Fuji Electric Co. Ltd.
Central P.O.B. 1619
Tokyo 100-91
☎ 2840777, Tx j22130

Korea
Siemens Electrical
Engineering Co., Ltd.
C.P.O.B. 3001
Seoul
☎ 7783431, Tx 23229

Kuwait
National & German Electrical and Electronic Service Company
NGEECO
P.O.Box 6612 Hawalli
Kuwait, Arabia
☎ 831544, Tx 22777

Libanon
Ets. F. A. Kettaneh S.A.
(Kettaneh Frères)
Medawar
P.B. 110242
Beyrouth
☎ 251040, Tx 20614

Malaysia
Electcoms Bumi Engineering
Sdn. Bhd.
Lot 18, Jalan 225
P.O.B. 310
Petaling Jaya/Selangor
☎ 762563, Tx 37418

Pakistan
Siemens Pakistan Engineering
Co. Ltd.
Ilaco House, Abdullah Haroon Road
P.O.B. 7158
Karachi 3
☎ 516061, Tx 2820

Philippinen
Maschinen + Technik Inc. (MATEC)
Greenbelt Mansion, Ground Floor,
Perea Street, Legaspi Village
Makati
P.O.Box 7129-s, ADC, MIA
Manila
☎ 8181321,
Tx TxM1, 63972

Saudi-Arabien
Arabia Electric Ltd.
Head Office
P.O.B. 4621
Jeddah
☎ 009662/6605089
Tx 401864
FAX 6605089

Singapur
Siemens Components Pte. Ltd.
Promotion Office
Block 7
Ayer Rajah Industrial Estate
Singapore 0513
☎ 7760283, Tx RS 21000

Syrien
Syrian Import
Export & Distribution
Co., S.A.S. SIEDCO
Port Saïd Street
P.O.B. 363
Damas
☎ 113431/32, Tx 11267 sy

Taiwan
Tai Engineering Co. Ltd.
6th Floor Central Building
No.108 Chung Shan N. Rd. Sec. 2
P.O.Box 68-1882
Taipei
☎ 5363171, Tx 27860 tai engco

Thailand
B. Grimm & Co., R.O.P.
1643/4, Phetburi Road
(Extension)
G.P.O.B. 66
Bangkok 10
☎ 2524081, Tx bgrim th 82614

Türkei
ETMAŞ Elektrik Tesisati ve Mühendislik A.Ş.
Meclisi Mebusan Caddesi 55/35
Findikli
P.K. 1001 Karakoey
Istanbul
☎ 009011/452090, Tx 24233

Yemen (Arab. Republik)
Tihama Tractors
& Engineering Co. Ltd.
P.O.B. 49
Sanaa
Yemen Arab Republic
☎ 2462, Tx 2217

AUSTRALIEN

Siemens Ltd.
544 Church Street, Richmond
Melbourne, Vic. 3121
☎ (03) 4297111, Tx 30425

Notizen