

PHILIPS

MICROCOMPUTER MASTER LAB



Anleitungsbuch
mit über 150 Programmier-Experimenten

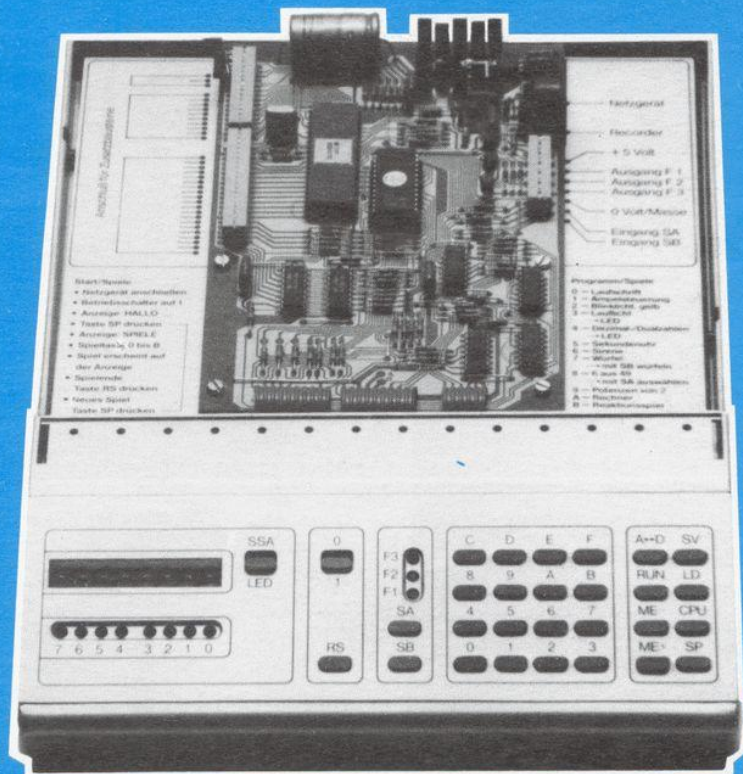
© Philips GmbH, Bereich Hobby-Technik

Alle Rechte vorbehalten. Nachdruck und
fotomechanische Wiedergabe — auch
auszugsweise — nicht gestattet.

Wir übernehmen keine Gewähr, daß die in
diesem Buch enthaltenen Angaben frei
von Schutzrechten sind.

Technische Änderungen vorbehalten.

Anleitungsbuch Microcomputer 6400 Master-Lab



Das renommierte US-Wochenmagazin TIME hat den „Personal Computer“, also den privaten Heimcomputer, zur „Persönlichkeit“ des Jahres 1982 gewählt.

Das ist erstaunlich, denn man hat damit nicht wie bisher einen „Mann bzw. eine Frau des Jahres“ herausgestellt, sondern erstmalig ein „Geräte-System“, das nach Meinung der Jury von Millionen von Amerikanern leidenschaftlich geliebt wird, in den Blickpunkt der Öffentlichkeit gerückt.

Immerhin wurden 1982 nahezu drei Millionen Heimcomputer verkauft. Welche Wachstumstendenzen in diesem Markt stecken, das kann man daran erkennen, daß im Jahr 1980 die Zahl der verkauften Geräte noch unterhalb einer dreiviertel Million lag. Nach dem Aufbruch ins Weltall hat sich — scheinbar in aller Stille — ein gewaltiger Aufbruch ins Computer-Zeitalter vollzogen. So gigantisch die aktuellen Verkaufszahlen auch sind, der „Computer des Jahres“ ist es nicht. Er ist klein (griechisch: mikros) und heißt daher auch in der Regel „Microcomputer“. Klein zwar in den Abmessungen, aber groß in der Leistung.

Der Firma PHILIPS ist Dank dafür auszusprechen, daß ihre Ingenieure in der Version des „Microcomputer MC 6400“ sowohl die Technik als auch die Kosten in den Griff bekommen haben. Noch nie gab es so viele Möglichkeiten, mit einem so kleinen Computer so viele interessante Anwendungen zu verwirklichen. In Verbindung mit einem ausführlichen Anleitungsbuch, auch „Software“ genannt, wird Schritt für Schritt in die Welt des Computers eingeführt und erklärt, wie man es anstellen muß, damit der Computer zum multifunktionalen Partner wird. Ob es nun um elektronische Spiele, oder ob es um die Steuerung der Modelleisenbahn geht. Natürlich kann er als Computer auch rechnen, aber das ist eigentlich nicht so bedeutsam, dafür genügt letztlich auch ein Taschenrechner.

Microcomputer der neuen Generation sind heute so unentbehrlich wie das Telefon und das Fernsehgerät. Im neuen PHILIPS-Microcomputer 6400 steckt eine Technik, die auf der Erfahrung einer Weltfirma aufbaut. Herzstück ist ein moderner Mikroprozessor, ein hochkomplizierter und leistungsfähiger integrierter Schaltkreis (IC), der auf einer Fläche von rund 4 Quadratmillimeter nicht weniger als 38 000 Transistorfunktionen aufweist. Die Datenbreite beträgt acht Bit, d. h. Informationen und Daten mit acht binär codierten Stellen können in einem Schritt verarbeitet werden. Kurz zur Erläuterung: Bit ist das Kürzel für „Binary digit“; ein Bit kann nur den Wert 0 oder 1 haben.

Als externen Speicher benutzt der MC 6400 handelsübliche Compact-Cassetten, wobei jeder preiswerte Cassetten-Re-

corder einsetzbar ist. Den Arbeitsspeicher bildet ein eingebauter Festwertspeicher (EPROM) mit 4 KB (Kilobytes), d. h. es stehen 4096 x 8 Speicherplätze zur Verfügung. Betrieben wird das System mit 9 Volt Gleichspannung. Die Taktfrequenz beträgt wie bei all diesen Geräten 4 MHz.

Ohne Übertreibung kann gesagt werden, daß es im Preis-/Leistungsverhältnis keine Alternative zum MC 6400 gibt. Der MC 6400 ist ein universell einsetzbares System, das zudem leicht zu bedienen ist. Dank seiner vielseitigen Programme respektive Programmiermöglichkeiten ist der MC 6400 für nahezu jede Aufgabe geeignet. Er macht dem Anfänger den Einstieg leicht und bietet auch für den Fortgeschrittenen noch viele technische Möglichkeiten.

In wünsche den Besitzern eines MC 6400 einen guten Start ins Computer-Zeitalter.

Prof. Georg Guertler
Vizepräsident der Fachhochschule Nürnberg

Der Microcomputer 6400	3	6. Vom Halbaddierer zum Addierwerk	30
Inhaltsverzeichnis	4	6.1 Der Halbaddierer	30
Bauteile des Microcomputers	6	6.2 Der Volladdierer	31
Vorwort	7	6.3 Ein vierstelliges Addierwerk	32
Experimentierbox	8	6.4 Aufgabe zu Kapitel 6	32
Bauteile der Experimentierbox	8	6.5 Die Addition von einstelligen Hexadezimalzahlen	33
Vorbereiten der Experimentierbox	9		
Befestigen der Bauteile auf der Grundplatte	10	7. Die ersten Befehle: Addition	34
Bauteile der Grundplatte	11	7.1 „Addiere dazu 3F!“	34
		7.2 16 Bit = 2 Byte	35
1. Die zwölf Spiele	13	7.3 Wir erweitern das Programm	35
1.1 Wir nehmen den Computer in Betrieb	13	7.4 Das Carry/Link-Flag: CY/L	36
1.2 Wir beginnen mit den Spielen	14	7.5 Eine zweite Adressierungsart	37
1.3 Beschreibung der Spiele	14	7.6 Noch mehr Komfort	38
1.3.1 Spiel 1: Ampel	14	7.7 Schriftliche Addition von Hexadezimalzahlen	38
1.3.2 Spiel 2: gelbes Blinklicht	14	7.8 Aufgaben zu Kapitel 7	38
1.3.3 Spiel 3: Lauflicht	14	7.9 Zusammenstellung der bisher besprochenen Befehle	39
1.3.4 Spiel 4: Zähler	14	7.9.1 Der mnemonische Code	39
1.3.5 Spiel 5: Sekunden-Uhr	15	7.9.2 Befehle mit unmittelbarer Adressierung	39
1.3.6 Spiel 6: Sirene	15	7.9.3 Befehle mit direkter Adressierung	39
1.3.7 Spiel 7: elektronischer Würfel	16	7.9.4 Die Calls	39
1.3.8 Spiel 8: Zahlenlotto „6 aus 49“	16		
1.3.9 Spiel 9: Zweier-Potenzen	16	8. Und nun noch einmal alles vierstellig	40
1.3.10 Spiel 10: Taschenrechner	16	8.1 Das Extension-Register E	40
1.3.11 Spiel 11: Reaktionstest	17	8.2 Aus Zwei mach Eins	41
1.3.12 Spiel 12: Laufschrift	17	8.3 Vierstellig, direkt adressiert	43
		8.4 Der Computer kennt keine Dezimalzahlen	44
2. Eingabe und Start eines Programms	18	8.5 Addition im Dezimalsystem	45
2.1 Vorbereitungen für die Programmeingabe	18	8.6 Aufgaben zu Kapitel 8	46
2.2 Programmeingabe	19		
2.3 Kontrolle eines eingegebenen Programms	19	9. Subtraktion	46
2.4 Programmstart	19	9.1 Zunächst die neuen Befehle	46
2.5 Abänderungen des Programms	19	9.2 Schriftliche Subtraktion von Hexadezimalzahlen	47
		9.3 Das CY/L-Flag und die Subtraktion	48
3. Der Computer und seine Peripherie	20	9.4 Das Rechnen mit Komplementen	49
3.1 Die Verbindungen mit der Peripherie	20	9.4.1 Neuner- und Zehnerkomplementen im Dezimalsystem	48
3.2 Die Glühlampe kann auch blinken	20	9.4.2 Fünfezener- und Sechzehnerkomplemente im Hexadezimalsystem	50
3.3 Ein Morseapparat	21	9.4.3 Einer- und Zweierkomplemente im Dualsystem	50
3.4 Eine Lichtschranke	21	9.5 Und so subtrahiert der Mikroprozessor wirklich	51
3.4.1 Aufbau und Überprüfung	21	9.6 Negative Zahlen	51
3.4.2 Diebstahlsicherung	21	9.7 Aufgabe zu Kapitel 9	52
3.4.3 Lichtschranke und Zähler	21		
3.5 Programmstart nur bei Adresse 1000?	22	10. Sprungbefehle	53
3.6 Ein kleiner Ausblick	22	10.1 Der Programmzähler PC	53
3.7 Etwas Elektronik	23	10.2 Der Jump-Befehl	53
3.6.1 Die Schaltung für die Ausgänge	23	10.3 Ein Branch-Befehl	54
3.6.2 Die Schaltung für die Eingänge	23	10.4 Bedingte Sprünge	56
		10.5 Wir lassen die Displacements berechnen	57
4. Dualzahlen und Hexadezimalzahlen	24	10.6 Der Programmablaufplan	58
4.1 Zahlenraten	24	10.7 Aufgaben zu Kapitel 10	58
4.2 Dualzahlen	24		
4.3 Hexadezimalzahlen	25	11. Inkrementieren und Dekrementieren	59
4.3.1 Einstelliger Hexadezimalzähler	25	11.1 Zwei nützliche Befehle	59
4.3.2 Zusammenhang mit Dualzahlen	25	11.2 Wir entwickeln ein Programm	60
4.3.3 Ein Rückwärtszähler	26	11.2.1 Die Aufgabenstellung	60
4.3.4 Mehrstelliger Hexadezimalzähler	26	11.2.2 Planung des Programmablaufs	60
4.3.5 Und noch ein Rückwärtszähler	27	11.2.3 Der Programmablaufplan	61
		11.2.4 Das Programm	61
5. Umwandlungen zwischen Zahlensystemen	27	11.2.5 Erprobung des Programms und Fehlersuche	61
5.1 Vorbemerkungen	27	11.3 Aufgaben zu Kapitel 11	62
5.2 Umwandlungen zwischen dem Dual- und dem Hexadezimalsystem	27		
5.3 Umwandlungen zwischen Dual- und Dezimalsystem	28	12. Unterprogramme	63
5.4 Umwandlungen zwischen dem Hexadezimal- und dem Dezimalsystem	28	12.1 Bemerkungen zum Status-Register S	63
5.5 Aufgaben zu Kapitel 5	28	12.2 Wir programmieren eine Pause	64
5.6 Drei Umwandlungsprogramme	29		
5.6.1 Von dezimal nach dual	29		
5.6.2 Von dezimal nach hexadezimal	29		
5.6.3 Von hexadezimal nach dezimal	29		

12.3	Wir schreiben unser erstes Unterprogramm	65	17.4	Entprellen mit Software	112
12.4	Der Stack oder der Stapelspeicher	67	17.5	Ein Interrupt von der Peripherie	113
12.5	Verschachtelung von Unterprogrammen	68	17.6	Ein gefälschter Würfel	114
12.6	Der Stackpointer	69	17.7	Aufgaben zu Kapitel 17	115
12.7	Aufgaben zu Kapitel 12	70			
13.	Die sechzehn Calls	70	18.	Programme — Programme — Programme	116
13.1	Die Adressierung der Calls	70	18.1	Die Programme aus dem 2. und 3. Kapitel	116
13.2	Beispiel: CALL VERZ (Verzögerung)	71	18.1.1	Das Programm aus dem 2. Kapitel	116
13.3	Ein Ampel-Programm	72	18.1.2	Das Programm „Die Glühlampe kann auch blinken“	116
13.4	Die verwendeten Speicher-Adressen	73	18.1.3	Das Programm „Ein Morseapparat“	116
13.5	Die Codierung der sechzehn Hexadezimalziffern	75	18.1.4	Das Testprogramm für die Lichtschranke	116
13.6	Zum Anzeige-Vorgang	76	18.1.5	Das Programm „Diebstahlsicherung“	116
13.7	Drei weitere Calls	77	18.1.6	Das Programm „Lichtschranke und Zähler“	116
13.8	Codierung beliebiger Symbole	78	18.2	Ergänzungen zu besprochenen Programmen	117
13.9	Die restlichen Calls	79	18.2.1	Die Summe kann fünfstellig sein	117
13.9.1	Der CALL ANZ EIN	79	18.2.2	Die Differenz kann negativ sein	117
13.9.2	Der CALL UEB 4L	80	18.2.3	Das Produkt kann achtestellig sein	118
13.9.3	Der CALL HALT	80	18.2.4	Verbesserte achtestellige Anzeige	118
13.10	Aufgaben zu Kapitel 13	81	18.3	Weitere Programme	119
14.	Logische Operationen	82	18.3.1	„RL EA“ — ein neuer Befehl?	120
14.1	Logische Verknüpfungen von zwei Eingangssignalen	82	18.3.2	Der BCD — Code	120
14.1.1	Die UND-Schaltung	82	18.3.3	Ein JK — Flip-Flop	120
14.1.2	Die ODER-Schaltung	83	18.3.4	Eine Ampel mit Handbedienung	121
14.1.3	Die NAND-Schaltung	83	18.3.5	Das kleine Einmaleins	121
14.1.4	Die NOR-Schaltung	84	18.3.6	Ein Würfelspiel	122
14.1.5	Die Äquivalenzschaltung	84	18.3.7	Messung der Reaktionszeit	122
14.1.6	Die Antivalenzschaltung	85			
14.1.7	Zusammenstellung der logischen Verknüpfungen	85	19.	Programme auf Cassette	123
14.2	Logische Verknüpfungen von zweistelligen Hexadezimalzahlen	86	19.1	Cassettenrecorder-Interface	123
14.3	Die logischen Befehle	86	19.1.1	Interface-Schaltung, 1. Teil	123
14.4	Negation mit Hilfe des XOR-Befehls	87	19.1.2	Interface-Schaltung, 2. Teil	124
14.5	Zusätzliche Sprungbedingungen	88	19.2	Speichern und Laden von Programmen	125
14.6	Maskieren	89	19.2.1	„Save“	126
14.7	Ein Bit löschen — ein Bit setzen	89	19.2.2	„Load“	126
14.8	Die ALU — die arithmetische, logische Einheit	90	19.3	Verschieben von Programmen	126
14.9	Aufgaben zu Kapitel 14	91	19.4.1	Ein Würfelspiel	128
15.	Schiebe- und Rotationsbetrieb	91	19.4.2	Das Spiel „Zahlen erkennen“	128
15.1	Lauflicht rechts	91	19.4.3	Das Spiel „NIMM“	128
15.2	Die sieben Befehle	93	19.4.4	Eine Digitaluhr	130
15.3	Spiel 9: Zweier-Potenzen	94	19.4.5	Der Computer macht Musik	130
15.4	(FL3): = (CY/L)	94			
15.5	Der Zufallszahlengenerator	95	20.	Lösungen zu den Aufgaben	131
15.6	Aufgaben zu Kapitel 15	97		Lösungen zu Kapitel 5	131
16.	Multiplikation und Division	98		Lösungen zu Kapitel 6	131
16.1	Ein erstes Multiplikationsprogramm	98		Lösungen zu Kapitel 7	132
16.2	Schriftliches Multiplizieren	99		Lösungen zu Kapitel 8	132
16.3	Verdoppeln — Halbieren	99		Lösungen zu Kapitel 9	133
16.4	Das zweite Multiplikationsprogramm	100		Lösungen zu Kapitel 10	134
16.5	Das T-Register und der Multiplikationsbefehl	101		Lösungen zu Kapitel 11	135
16.6	Dezimale Multiplikation	103		Lösungen zu Kapitel 12	137
16.7	Dezimale Division	104		Lösungen zu Kapitel 13	138
16.8	Die Calls CALL DEZ-HEX und CALL HEX-DEZ	105		Lösungen zu Kapitel 14	140
16.8.1	Der CALL DEZ-HEX	105		Lösungen zu Kapitel 15	142
16.8.2	Der CALL HEX-DEZ	106		Lösungen zu Kapitel 16	144
16.8.3	Bemerkungen zu diesen Calls	107		Lösungen zu Kapitel 17	146
16.9	Aufgaben zu Kapitel 16	108	A.	Anhang	147
17.	Ergänzungen zum Status-Register	109	A1	Befehlslisten	147
17.1	Das Overflow-Flag	109	A1.1	Zusammenstellung der behandelten Befehle	147
17.2	Endlos-Schleife mit Ausgang	110	A1.2	Befehlsliste, Adressierungsarten	148
17.3	Weitere Untersuchung des Interrupt-Verfahrens	111	A1.3	Befehlsliste, mnemonischer Code	148
			A1.4	Befehlsliste, Operationscode	149
			A1.5	Änderung der Register- und Speicherinhalte bei den sechzehn Calls	150
			A2	Umwandlungstabellen	151
			A2.1	Dezimal — dual — hexadezimal	152
			A2.2	dezimal — hexadezimal	152
			A2.3	hexadezimal — dezimal	152
			A3	Stichwortverzeichnis	153
			A4	Verdrahtungspläne	158
			A5	Blaue Funktionstasten bei Spiel 10: Rechner	160

Bauteile des Microcomputers 6400

Bestell-Nr.	Bezeichnung	Menge
6400.2535	Microcomputer komplett, mit Gehäuse	1
2531	Stecker-Netzteil 9 V, 350 mA	1
2532	Programm-Cassette	1
2533	Überspielkabel	1
2548	Platine mit Bedienungselementen Tastschalter Schiebeschalter Potentiometer, 10.000 Ohm Leuchtdiode, rot Vorwiderstand, 470 Ohm Federleiste, 13-polig Federleiste, 2-polig Netzanschlußbuchse	1
2549	Gehäuse	1
2550	Rahmeneinsatz	1
2552	Frontplatte für Bedienungspult	1
2551	Grundplatte	1
2523	Abdeckhaube	2
2554	Zeigerknopf	1
2515	Lautsprecher	1
2534	Knopfaufsatz für Schiebeschalter	1
2546	Transistor, weiß	1
1004	Widerstand 100 Ohm (braun, schwarz, braun) 4.700 Ohm (gelb, violett, rot) 47.000 Ohm (gelb, violett, orange)	1 1 1
1005	Folien-Kondensator 0,1 μ F	1
1006	Elektrolyt-Kondensator 10 μ F	1
1010	LDR — Lichtempfindlicher Widerstand	1
1014	Glühlampe 6 V, 0,05 A	1
1017	Isolierter Draht	4m
1026	Lampenfassung	1
2526	Klemmen	25
2051	Stiftkontakt	18
2057	Gummitülle	18
1101	Steckhülse	2
2560	Blehschraube	11
2561	Unterlegscheibe	4
2571	Anleitungsbuch	1

Der Computer läßt sich aus unserem Leben nicht mehr wegdenken. Er hat in den letzten Jahren das Berufsleben entscheidend verändert, und er dringt immer mehr in die privaten Haushalte ein. Diese Entwicklung ist noch lange nicht abgeschlossen. Wir können uns heute noch nicht vorstellen, wie der weitere Vormarsch des Computers aussehen wird. Es ist unerläßlich, daß wir uns mit diesem Thema ernsthaft auseinandersetzen.

Vor etwa 40 Jahren wurde der erste Computer gebaut; es wurden zunächst Relais, dann Röhren verwendet. Mit der Erfindung des Transistors wurden digitale Rechenanlagen kleiner. Sie füllten jetzt nicht mehr ein ganzes Zimmer, sondern nur noch ein Gehäuse wie etwa das einer größeren Schreibmaschine. Die Entwicklung ging weiter. Die heutige Mikroelektronik benötigt immer weniger Platz und wird dabei immer billiger und störunanfälliger. Zehntausende von elektronischen Schaltelementen werden heute auf einem Siliziumplättchen von einigen Quadratmillimetern untergebracht. Ein solches Siliziumplättchen befindet sich im Mikroprozessor-IC unseres Computers. Diese Entwicklung — immer mehr Elektronik auf immer kleinerem Raum — ist kaum noch vorstellbar, aber doch ist es wahr.

Computer sind programmierbar. Man gibt ihnen andere Befehle, und schon lösen sie ganz andere Aufgaben. Darin besteht ihre Vielseitigkeit. Der Programmierer formuliert seine Wünsche häufig in einer Sprache, die die Rechenanlage nicht unmittelbar versteht. Eine Programmiersprache (die bekannteste ist wohl BASIC) muß für den Computer in die Maschinensprache übersetzt werden. Bei unserem Computer benutzen wir von vornherein die Maschinensprache. So können wir besser verstehen, wie der Computer arbeitet.

Das Anleitungsbuch soll helfen, diesen Computer kennenzulernen. Es soll darüberhinaus Zusammenhänge erklären, die jedem Computer zugrundeliegen.

Die ersten drei Kapitel beschreiben, wie unser Computer bedient wird. Sie geben einen ersten Einblick in die unterschiedlichsten Anwendungsmöglichkeiten. Die Kapitel 4 und 5 beschäftigen sich mit den verwendeten Zahlensystemen. Wenn das Dual- und das Hexadezimalsystem bekannt sind, können diese Kapitel überschlagen werden. Als Hilfen sind im Anhang Umwandlungstabellen vorhanden. Die Benutzung dieser Zahlensysteme ist aber unerläßlich, wenn ein Mikroprozessor in Maschinensprache programmiert werden soll. Kapitel 6 soll das Prinzip erläutern, wie es einem Computer möglich ist, Additionsaufgaben zu lösen. Alle weiteren arithmetischen Operationen bauen auf der Addition auf.

In den Kapiteln 7 bis 17 werden nach und nach 71 Befehle beschrieben. Diese Befehle werden zunächst mit Hilfe klei-

ner Testprogramme untersucht. Durch Kombination ergeben sich dann immer neue Möglichkeiten für das Programmieren. Nach Bearbeitung dieser Kapitel ist man sicher in der Lage, eigene Programme zu schreiben. Das Kapitel 18 soll dazu Anregungen geben. Neben den Befehlen werden Begriffe aus der Computertechnik eingeführt und erläutert. Diese Begriffe werden an den Stellen beschrieben, an denen sie im Zusammenhang mit den Befehlen benötigt werden. Das ausführliche Stichwortverzeichnis im Anhang dieses Anleitungsbuches soll das Auffinden der Befehls- und Begriffserläuterungen erleichtern.

Unser Computer ist mit einem Cassettenrecorder-Interface ausgerüstet. Dadurch haben wir die Möglichkeit, Programme mit Hilfe eines handelsüblichen Cassettenrecorders zu speichern und bei Bedarf wieder in unseren Computer zu laden. Das 19. Kapitel beschreibt diese Zusammenhänge. Es ist durchaus möglich, sich mit dem Speichern von Programmen auf Cassette und dem Laden von Cassette (vgl. Abschnitte 19.2.1 und 19.2.2) schon dann zu beschäftigen, wenn man sich mit der Handhabung des Computers in den ersten drei Kapiteln vertraut gemacht hat. Man hat dann die Möglichkeit, auch schon bei der Bearbeitung des Anleitungsbuches interessante Programme auf Cassette zu speichern.

Das 20. Kapitel bringt die Lösungen für alle Aufgaben, die in den voranstehenden Kapiteln gestellt wurden.

Das Anleitungsbuch beschreibt die wesentlichen Grundlagen, die zum Arbeiten mit unserem Computer erforderlich sind. Es wurde in diesem Buch darauf verzichtet, den Befehlssatz unseres Mikroprozessors vollständig zu behandeln. Das hätte den Überblick unnötig erschwert. Die besprochenen Befehle reichen sicher aus, um die meisten Probleme zu lösen.

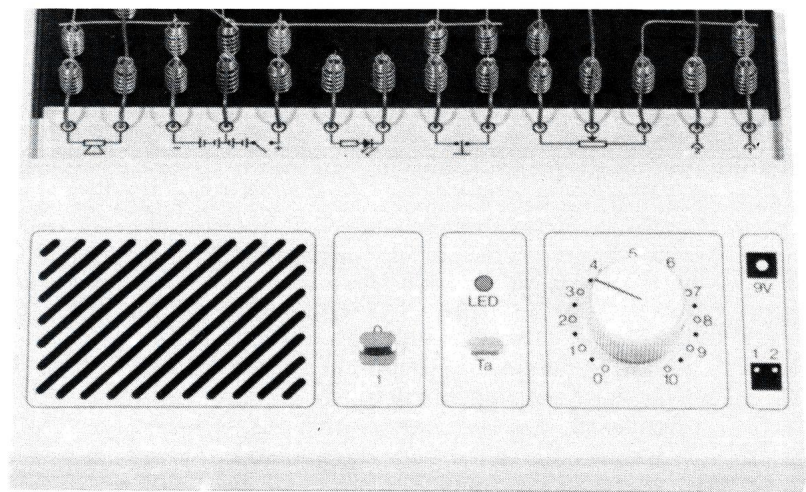
Unser Computer ist so ausgelegt, daß auch hardwaremäßig Erweiterungsmöglichkeiten bestehen. An der linken Federleiste unserer Computerplatine können unterschiedliche Zusatzbausteine angesteckt werden. Aber auch hier gilt das, was oben bei den Befehlen gesagt wurde: man sollte sich zunächst mit dem Computer in der vorliegenden Ausbaustufe beschäftigen, und man sollte die zahlreichen Möglichkeiten kennenlernen, die jetzt schon gegeben sind.

Wer sich auf das Abenteuer „Computer“ einläßt, wird fasziniert sein. Dabei viel Erfolg!

Erhard Meyer

Experimentierbox

Die **Experimentierbox** besteht nach der Montage aus dem **Experimentierfeld** mit der Grundplatte, der Abdeckhaube und dem **Bedienungspult**.



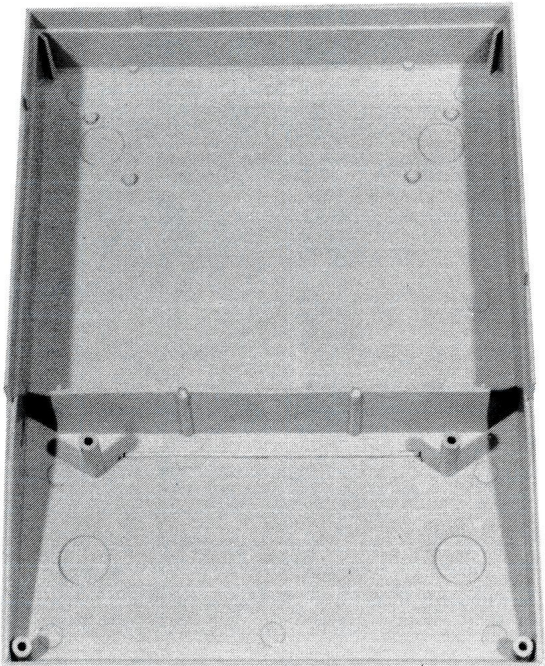
Experimentierfeld

Bedienungspult

Die in diesem Anleitungsbuch beschriebenen Experimente der Perefierie erhalten ihre Stromversorgung durch die Steckerleiste des Microcomputers. Im Verdrahtungsplan sind die entsprechenden Verbindungen eingezeichnet. Sollen später größere Verbraucher angeschlossen werden, ist ein geeignetes Netzteil zu verwenden. Der Pluspol muß am vorderen Kontakt des Klinkensteckers liegen. An den mit + und — gekennzeichneten Klemmen steht dann die Spannung zur Verfügung. Zum Microcomputer selbst dürfen dann nur noch die Minusleitung und die Aus- und Eingänge durchverbunden werden. Die Plusleitung entfällt also.

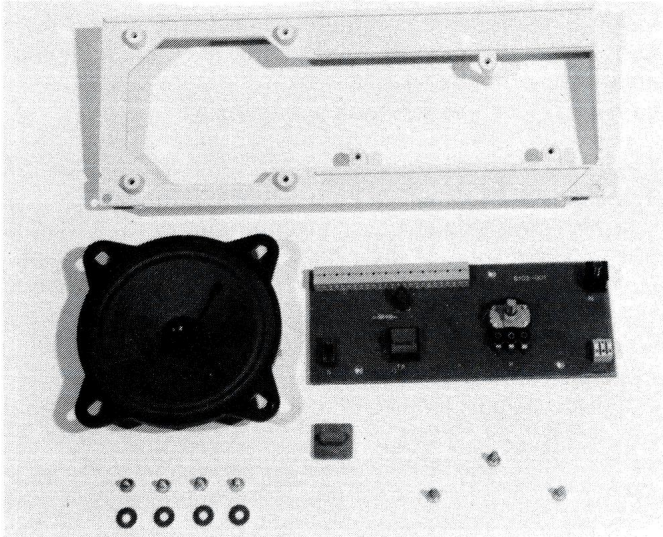
Lautsprecher Ein- Aus- Schalter LED Tastschalter Potentiometer Anschluß für Netzgerät Außenanschlüsse 1 u. 2

Bauteile der Experimentierbox



Gehäuse

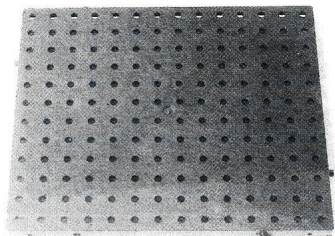
Lautsprecher Rahmeneinsatz Platine mit Bedienungselementen



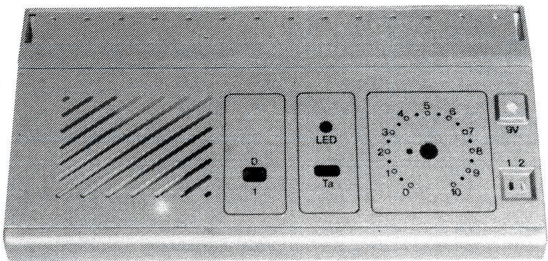
Blechschauben Unterlegscheiben Knopfaufsatz für Schiebeschalter Blechschauben



Zeigerknopf



Grundplatte



Frontplatte für Bedienungspult

Vorbereiten der Experimentierbox

In das Bedienungspult werden die Platine mit den Bedienungselementen und der Lautsprecher eingebaut.

In der vorderen Abteilung wird der **Rahmeneinsatz** eingelegt. Darauf liegt an der rechten Seite leicht schräg die **Platine** mit den fest montierten **Bedienungselementen**. Die drei Bohrlöcher der Platine müssen sich mit denen im Rahmen decken, damit die Platine mit drei selbstschneidenden Schrauben (Blechschauben) fixiert werden kann. An der linken Seite wird der **Lautsprecher** befestigt, wobei darauf zu achten ist, daß seine Anschlüsse zur Platine zeigen. Dazu je eine Unterlegscheibe auf die Schraube stecken, diese von oben durch die 4 Befestigungslöcher des Lautsprechers führen und sie dann in die Löcher des Rahmeneinsatzes drehen.

Um eine feststehende Verdrahtung zwischen dem Bedienungspult und dem Experimentierfeld herzustellen, müssen 11 Drähte von 14 cm Länge abgeschnitten und mit Stiftkontakten versehen werden.

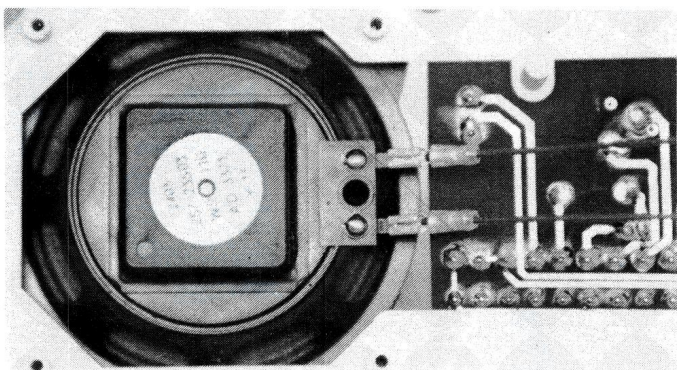
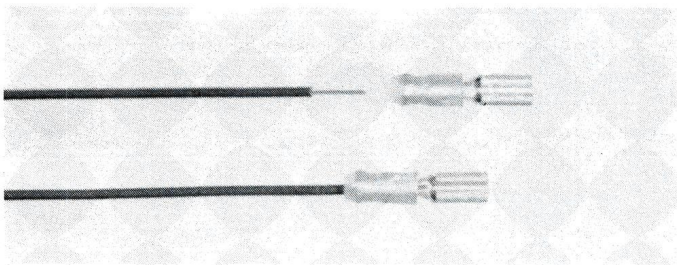
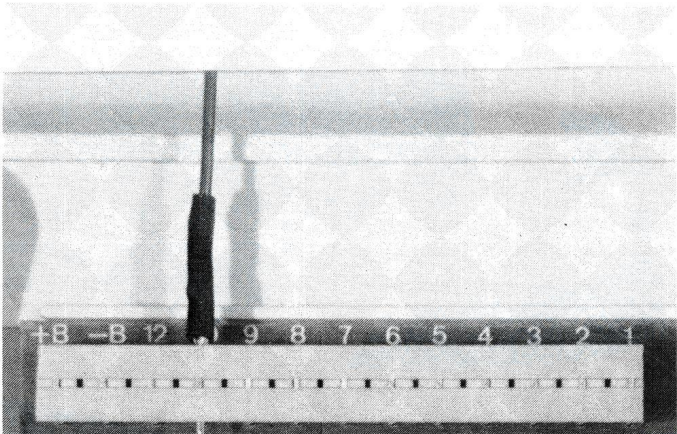
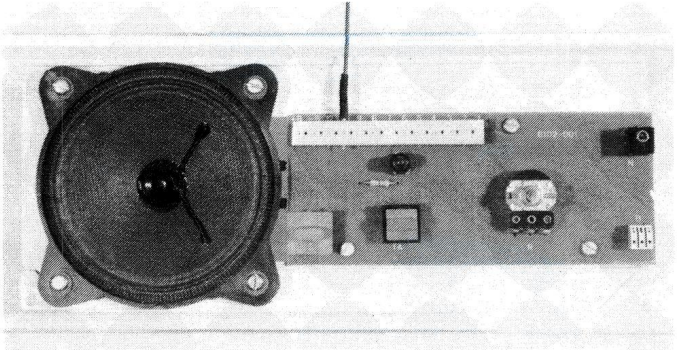
Dazu werden die Drähte an beiden Enden abisoliert. Auf einer Seite mit einer Flachzange den Stiftkontakt festdrücken und eine Gummitülle darüber schieben, damit keine Kurzschlüsse entstehen.

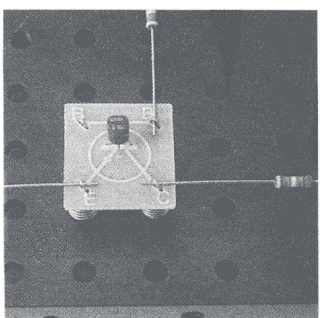
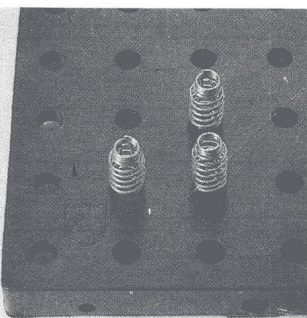
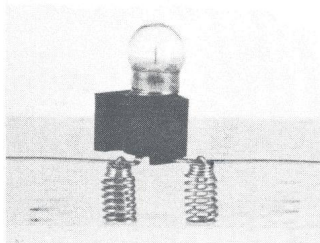
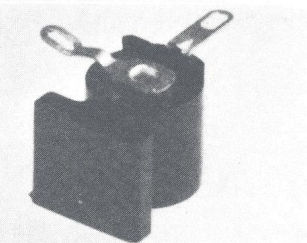
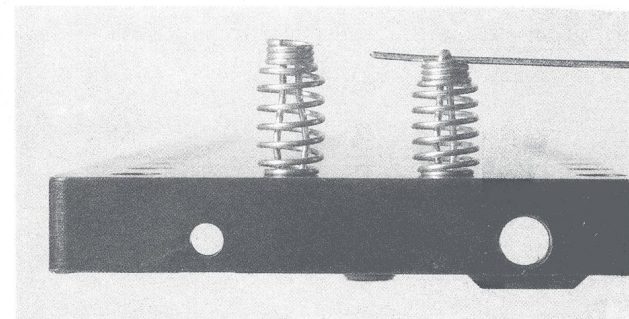
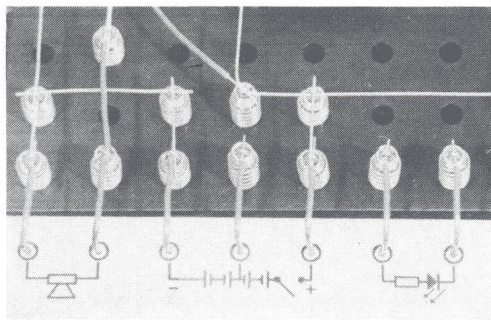
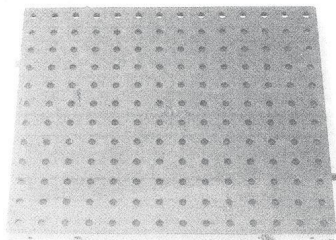
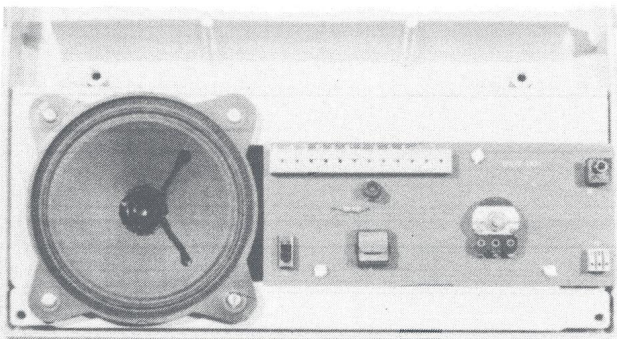
Die 11 Drähte dann mit den Stiftkontakten in die Kontakte 1—10 und 12 der Buchsenleiste auf der Platine schieben (Abb.).

Damit beim späteren Experimentieren diese Drähte nicht aus den Kontakten gerissen werden, in jeden Draht etwa 5 cm vom abisolierten Ende einen Knoten machen.

Der Lautsprecher hat zwei Steckkontakte. Deshalb müssen jetzt zwei abisolierte Drähte, etwa 20 cm lang, mit je einer Flachsteckhülse versehen werden. Ein Ende des Drahtes in die Seite des Steckkontaktes drücken, die mit der Isoliertülle gegen Kurzschluß geschützt ist.

Darauf achten, daß der Draht innen in die Metallzungen reicht, und dann mit einer Flachzange diese Zungen fest um den Draht drücken (Abb.).





Die 11 Verbindungsdrähte der Platine und die zwei des Lautsprechers müssen jetzt von innen durch die Löcher der Frontplatte für das Bedienungspult geführt werden, wobei 1—10 und 12 für die Verbindung mit der Platine vorbehalten sind, 13 und 14 für den Lautsprecher. Das Loch 11 bleibt frei. Alle 13 Drahtenden werden später in entsprechende Klemmen auf der Grundplatte gesteckt.

Abschließend noch den Knopfaufsatz für den Ein-Aus-Schalter aufsetzen (Abb.) und die Frontplatte für das Bedienungspult mit vier langen Schrauben, die von unten eingedreht werden, montieren.

Nachdem der Reglerknopf für das Potentiometer aufgesetzt wurde, ist das Bedienungspult für alle Experimente betriebsbereit.

Das eigentliche Experimentierfeld ist die schwarze Grundplatte mit den vielen Löchern. Sie wird in die hintere Abteilung der Experimentierbox eingelegt. Zur Befestigung der Bauteile und zum Herstellen von Drahtverbindungen dienen Klemmen, die von oben durch ein Loch der Grundplatte gesteckt werden, bis sie einrasten (Abb.). Für feststehende Verbindungen des Bedienteils mit dem Experimentierfeld in die letzte Reihe über dem Bedienungspult auf der Grundplatte 13 Klemmen einsetzen, an die die Drähte 1—10 und 12—14 aus dem Bedienungspult angeschlossen werden (Abb.). Diese Verbindungen bleiben immer bestehen.

Befestigen der Bauteile auf der Grundplatte

Anschlußdrähte
Widerstände
Kondensatoren

Klemmfeder niederdrücken, bis Schlaufe der Haarnadelfeder sichtbar ist.
Draht in die Schlaufe einschieben.
Klemmfeder loslassen.

Glühlampe

Anschlüsse der Fassung niederdrücken.
Anschlüsse auf Klemmen stecken.
Draht in die Schlaufe einschieben.
Klemmfeder loslassen.

Transistor

So viele Klemmen einsetzen, wie Kreise auf dem Verdrahtungsplan sind (z. B. 3 oder 4 beim Transistor).

Haarnadelfedern nach den Schlitten im Plättchen ausrichten.

Plättchen an einer Ecke niederdrücken und Anschlußdraht durch die Schlaufe schieben.

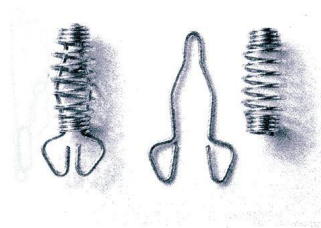
An den anderen Klemmen wiederholen, bis alle Anschlüsse befestigt sind.

Bauteile auf der Grundplatte

Abbildung im
Verdrahtungsplan

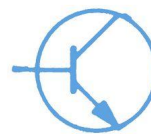
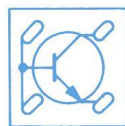
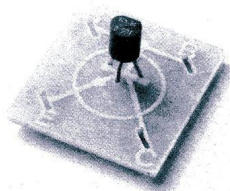
Abbildung im
Schaltplan

Klemme

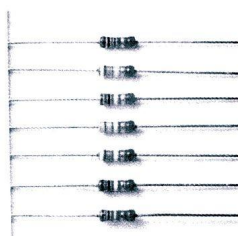


Transistor

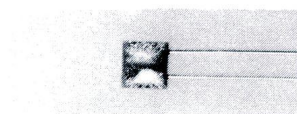
Achtung:
Anschlüsse nicht vertauschen



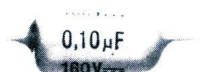
Widerstand



LDR



Folien-
Kondensator



Elektrolyt-Kondensator

Achtung: richtig herum einsetzen



Rille +

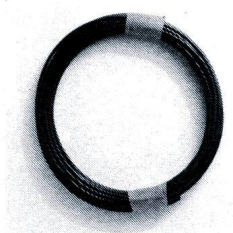


Lampe und Fassung

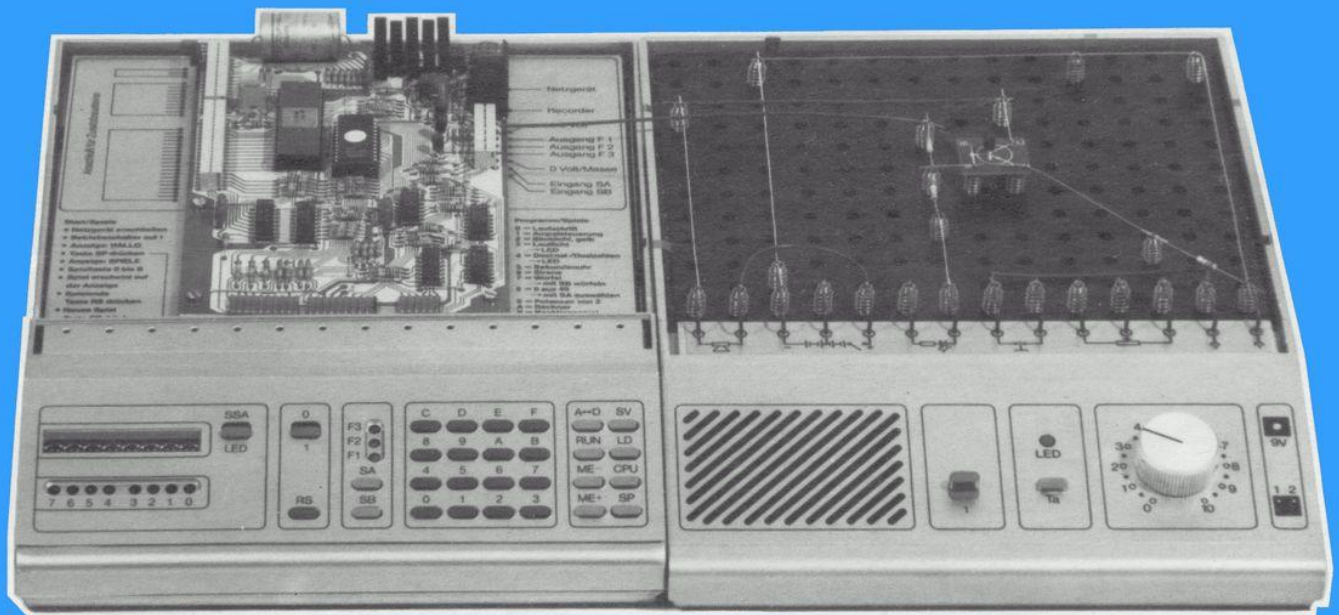


isolierter Draht

Enden abisolieren



Microcomputer 6400



Microcomputer

Experimentierbox

1. Die zwölf Spiele

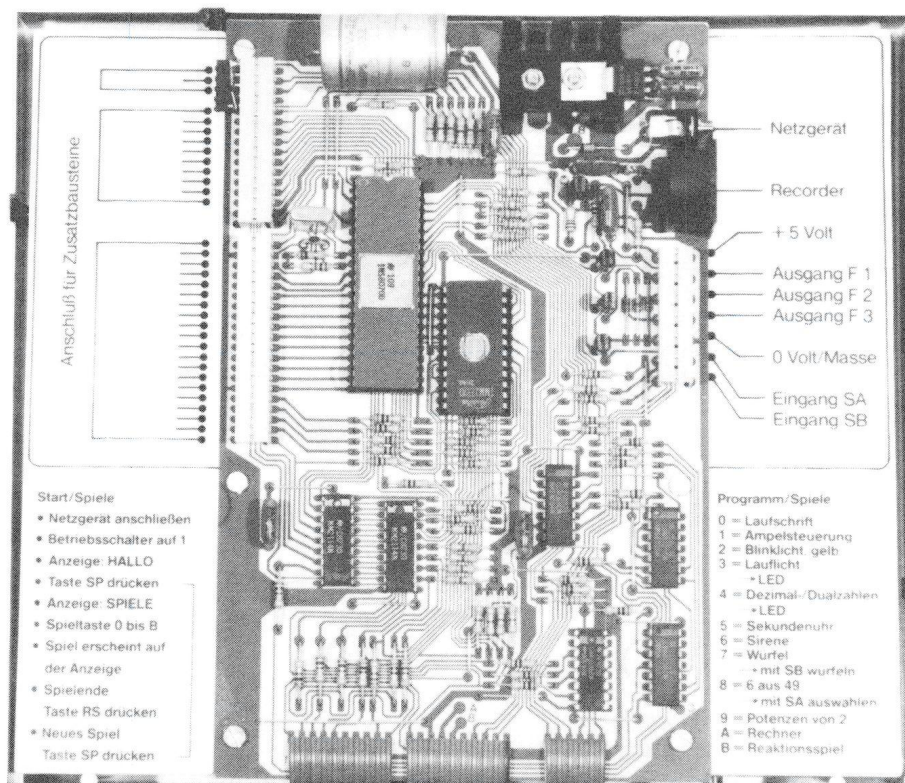
1.1 Wir nehmen den Computer in Betrieb

Vor uns steht unser Computer. Wir unterscheiden die Computer-Platine und das Bedienungspult. Beide Teile sind durch eine Flachbandleitung verbunden.

Die Computer-Platine hat rechts Anschlüsse für das Netzgerät und für einen Cassettenrecorder. Das Netzgerät wer-

den wir gleich anschließen. Mit Hilfe des Cassettenrecorders können Programme vom Computer auf Cassette abgespeichert bzw. von der Cassette wieder eingelesen werden. Das 19. Kapitel wird sich mit diesen Möglichkeiten befassen.

An der rechten Seite der Computer-Platine befindet sich außerdem eine Federleiste für sieben Drahtverbindungen. Hier lassen sich externe Bauteile an den Computer anschließen. Wir kommen darauf bei Spiel 6 und im 3. Kapitel zurück.



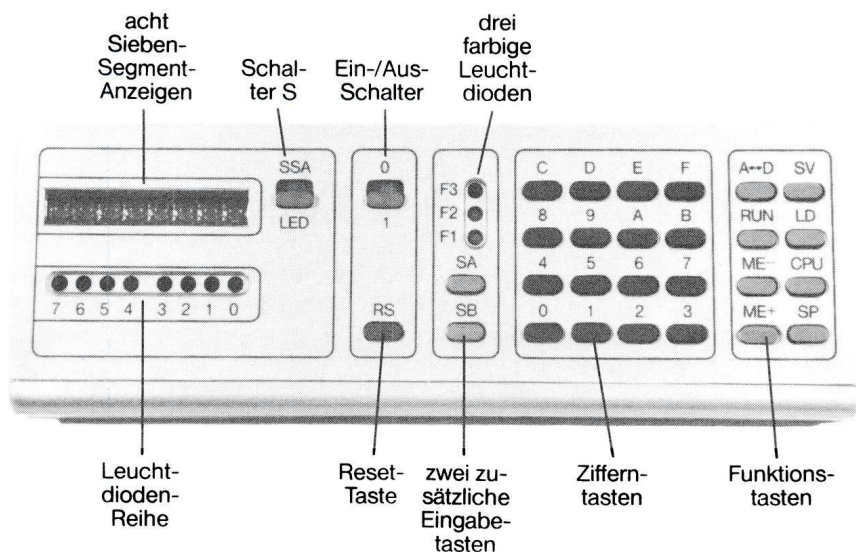
Computer-Platine

Die Federleiste an der linken Seite der Computer-Platine wird erst bei einer weiteren Ausbaustufe unseres Computersystems für den Anschluß von Zusatzeinheiten benötigt. Damit wir uns mit unserem Computer verständigen können, ist das Bedienungspult unerlässlich. Mit Hilfe der Tasten teilen wir ihm unsere Wünsche mit, über die verschiedenen Anzeigemöglichkeiten erfahren wir, was in ihm vorgeht. Wir verbinden das Netzgerät mit unserem Computer und

schalten das Gerät mit dem Ein-Aus-Schalter ein. Der Computer meldet sich mit seinem Namen:

HALLO

Der Schalter S muß hierbei in der hinteren Stellung stehen. Sonst würden außer der verstümmelten Anzeige „HA LO“ noch drei Leuchtdioden in der Leuchtdioden-Reihe leuchten.



Bedienungspult

1.2 Wir beginnen mit den Spielen

Wird bei der Anzeige „HALLO“ die Taste **SP** betätigt, so erscheint die Anzeige „SPIELE“ (Schalter S nach hinten, sonst Anzeige „SPI LE“ und fünf leuchtende Dioden!). Bevor wir uns diese Spiele genauer ansehen, soll gleich etwas klargestellt werden: es handelt sich nicht um Spiele im herkömmlichen Sinn des Wortes, sondern um zwölf festgelegte Programme, die einen kleinen Einblick geben sollen, wie vielseitig unser Computer ist. Darüber hinaus kann er natürlich noch viel mehr, aber das kann sich erst im Laufe dieses Lehrgangs herausstellen.

Wir wollen die zwölf Spiele folgendermaßen bezeichnen:

Spiel Nr.	Bezeichnung	Schalter S
1	Ampel	beliebig
2	gelbes Blinklicht	beliebig
3	Lauflicht	vorn
4	Zähler	vorn
5	Sekunden-Uhr	beliebig
6	Sirene	beliebig
7	elektronischer Würfel	hinten
8	Zahlenlotto „6 aus 49“	hinten
9	Zweier-Potenzen	hinten
10 (A)	Taschenrechner	hinten
11 (B)	Reaktionstest	hinten
12 (C)	Laufschrift	hinten

Ein Spiel – oder eins dieser fest vorgegebenen zwölf Programme – läßt sich folgendermaßen aufrufen: Man betätigt nacheinander die Taste **SP** (mehrfache Betätigung schadet nicht) und eine der Zifferntasten **1** bis **9** **A** **B** oder **C**. Eventuell muß vorher die Taste **RS** betätigt werden. Soll ein Spiel abgebrochen werden, so ist das grundsätzlich mit der **RS**-Taste möglich. Mit der Reset-Taste **RS** kann jedes laufende Programm unterbrochen werden (engl. to reset – zurücksetzen). Diese Methode ist nicht so radikal wie eine Unterbrechung der Versorgungsspannung mit dem Ein-/Aus-Schalter. Im nächsten Abschnitt sollen die zwölf Spiele beschrieben werden. Es wird hier zunächst nur gesagt, wie der Computer bei diesen Spielen – oder bei diesen fest vorgegebenen Programmen – zu bedienen ist. Einzelheiten über den jeweiligen Programmablauf bleiben uns hier noch verborgen. Wir werden später darauf zurückkommen.

1.3 Beschreibung der Spiele

1.3.1 Spiel 1: Ampel

Tasten: (**RS**) **SP** **1**

Die drei verschiedenfarbigen Leuchtdioden leuchten nacheinander wie bei einer Verkehrsampel auf: grün, gelb, rot, rot/gelb und grün. Bemerkenswert ist vielleicht, daß der Computer nicht nur festlegt, welche Leuchtdiode leuchtet, sondern auch, wie lange sie leuchtet. Der Computer versteht also auch etwas von einer Zeitdauer. Die beiden Ampelphasen grün und rot sind länger als die Übergangsphasen gelb und rot/gelb.

1.3.2 Spiel 2: gelbes Blinklicht

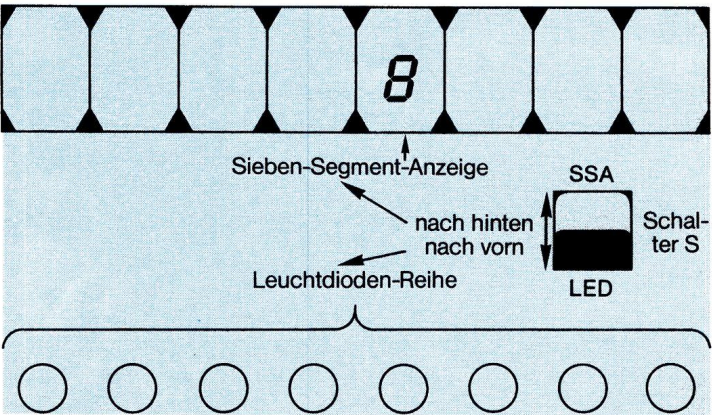
Tasten: (**RS**) **SP** **2**

Hier ist keine weitere Erklärung erforderlich. Die gelbe Leuchtdiode wird im Wechsel an- und ausgeschaltet.

1.3.3 Spiel 3: Lauflicht

Schalter S nach vorn; Tasten: (**RS**) **SP** **3**

Die acht Leuchtdioden der Leuchtdioden-Reihe werden nacheinander kurz angesteuert, so daß man den Eindruck hat, ein Licht lief durch die LED-Reihe (LED ist die Abkürzung für die englische Bezeichnung: Light Emitting Diode; Licht ausstrahlende Halbleiter-Diode, Leuchtdiode). Sollte der Schalter S in der hinteren Stellung stehen, so bleiben die acht Leuchtdioden dunkel. Dafür ergeben sich in der Anzeige darüber eigenartige Erscheinungen. Wir werden darauf bei dem nächsten Spiel zurückkommen. Hier geht der Wechsel der Anzeige so schnell, daß die Beobachtung schwierig ist. Die Bedeutung des Schalters S ist aber jetzt schon deutlich geworden. Mit ihm können wir wählen, ob wir eine bestimmte Anzeige an der LED-Reihe oder an einer Sieben-Segment-Anzeige darüber (vierte Stelle von rechts) wünschen.



1.3.4 Spiel 4: Zähler

Schalter S nach vorn; Tasten: (**RS**) **SP** **4**

Wir sehen, daß unser Zähler bis 15 zählt und dann wieder bei 0 beginnt. Gleichzeitig leuchten einige der vier rechten Leuchtdioden der LED-Reihe. Eingeweihte werden erkennen, daß die Leuchtdioden auch jeweils die Zahl des Zählers anzeigen. In den beiden rechten Sieben-Segment-Anzeigen wird die Zahl im Dezimalsystem dargestellt, die vier rechten Leuchtdioden zeigen dieselbe Zahl im Dual- oder Zweiersystem. Man muß sich nun folgende Zuordnung klar-machen:

Die leuchtenden Leuchtdioden entsprechen von rechts nach links den vier Dezimalzahlen 1, 2, 4 bzw. 8:

entsprechende Dezimalzahl → 8 4 2 1
 ↓ ↓ ↓ ↓
Leuchtdioden-Reihe → ○ ○ ○ ○

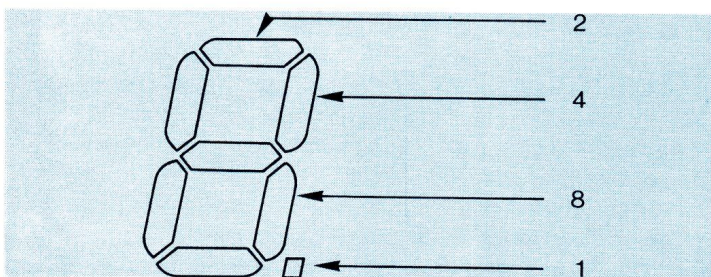
Leuchtet nur eine der vier Dioden, so zeigt der Dezimalzähler gerade die entsprechende Zahl an. Leuchten mehrere Dioden, so läßt sich die angezeigte Zahl aus den Zahlen 1, 2, 4 und 8 durch Addition ermitteln:

Dezimalzahl	Leuchtdioden	Die Dezimalzahl läßt sich aus den entsprechenden Zahlen 8, 4, 2 und 1 ermitteln.
	8 4 2 1 ↓ ↓ ↓ ↓	
0	○ ○ ○ ○	0
1	○ ○ ○ ●	1
2	○ ○ ● ○	2
3	○ ○ ● ●	2 + 1 = 3
4	○ ● ○ ○	4
5	○ ● ○ ●	4 + 1 = 5
6	○ ● ● ○	4 + 2 = 6
7	○ ● ● ●	4 + 2 + 1 = 7
8	● ○ ○ ○	8
9	● ○ ○ ●	8 + 1 = 9
10	● ○ ● ○	8 + 2 = 10
11	● ○ ● ●	8 + 2 + 1 = 11
12	● ● ○ ○	8 + 4 = 12
13	● ● ○ ●	8 + 4 + 1 = 13
14	● ● ● ○	8 + 4 + 2 = 14
15	● ● ● ●	8 + 4 + 2 + 1 = 15

Diese Andeutungen sollen hier zunächst genügen. Wir werden im 4. Kapitel ausführlich auf das Dualsystem zurückkommen.

Wird jetzt der Schalter S nach hinten gestellt, so leuchten statt der vier Leuchtdioden der LED-Reihe vier Leuchtdioden in der vierten Sieben-Segment-Anzeige (von rechts) abwechselnd auf.

Bei der Sieben-Segment-Anzeige lassen sich die sieben Teilstriche (Segmente) und ein Punkt getrennt oder gleichzeitig ansteuern. Es handelt sich also auch um acht Leuchtdioden in ganz bestimmter Anordnung. Die Zuordnung ergibt sich entsprechend wie bei der LED-Reihe:

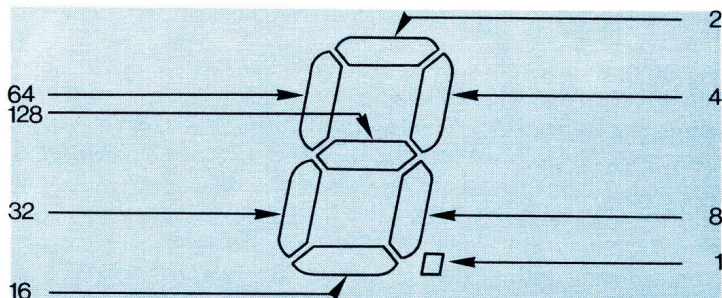


Unser Computer zählt bei diesem Spiel nicht nur. Er prüft jeweils nach, ob die angezeigte Zahl gleich 15 ist. Wenn das der Fall ist und die Taste **SA** wird nicht betätigt, dann folgt nach 15 wieder 0. Wird aber – während 15 angezeigt wird – die Taste **SA** betätigt, dann zählt der Zähler nach 15 weiter bis 255. Erst dann folgt wieder die 0.

Ist der Schalter S in der vorderen Stellung, so werden jetzt auch die linken vier Leuchtdioden der LED-Reihe angesteuert. Dieser Dualzähler zählt jetzt auch bis 255. Die leuchten den linken vier Leuchtdioden entsprechen größeren Dezimalzahlen:

entsprechende Dezimalzahl	→	128	64	32	16	8	4	2	1
LED-Reihe	→	○	○	○	○	○	○	○	○

Entsprechend werden in der hinteren Stellung des Schalters S auch die anderen vier Segmente der Sieben-Segment-Anzeige angesteuert:



Hier ergeben sich nacheinander alle 256 Möglichkeiten für die Ansteuerung der Sieben-Segment-Anzeige (einschließlich der Möglichkeit, daß keine der acht Leuchtdioden angesteuert wird).

Wir erkennen, daß die Ziffern 0, 1, 2, 3, 4, 5, 6, 7, 8 und 9, die Buchstaben A, b, C, d, E, F und weitere Buchstaben H, L, O (vgl. Anzeige HALLO), S, P, I (vgl. Anzeige SPIELE) angezeigt werden können. In den acht Sieben-Segment-Anzeigen könnten daher z. B. auch folgende Wörter dargestellt werden: HILFE, FLASCHE oder ESEL; für viele andere Wörter ist unsere Anzeige aber nicht geeignet, da z. B. die Buchstaben K, M, N oder T nicht darstellbar sind.

1.3.5 Spiel 5: Sekunden-Uhr

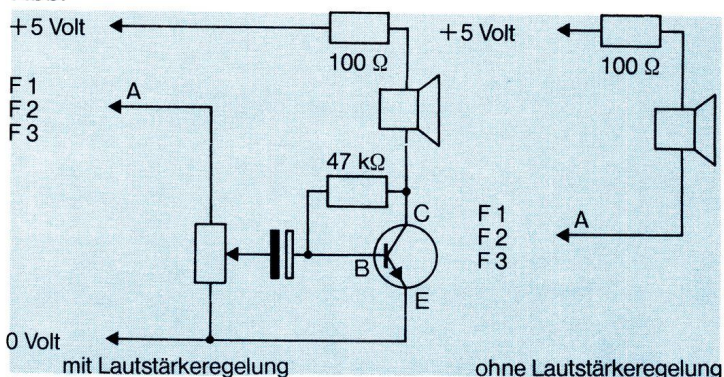
Tasten: **(RS)** **SP** **5**

Unsere Mini-Uhr mißt nur Sekunden. Für die Genauigkeit der Uhr ist letztlich der Quarz auf der Computer-Platine verantwortlich. Der Quarz hat eine Frequenz von 4 000 000 Hz, d. h. er macht in einer Sekunde 4 000 000 Schwingungen. Wenn man jetzt dafür sorgt, daß der Zähler der Sekunden-Uhr jeweils nach 4 000 000 Schwingungen des Quarzes erhöht wird, dann geht die Uhr so genau, wie der Quarz seine Frequenz einhält.

Natürlich könnte unser Computer auch eine komfortablere Uhr steuern. Er müßte dann auch Minuten und Stunden anzeigen; er müßte nach 60 Sekunden die Minutenanzeige und nach 60 Minuten die Stundenanzeige erhöhen. Wir werden eine solche Digitaluhr später besprechen und auch programmieren.

1.3.6 Spiel 6: Sirene

Für dieses Spiel muß der Lautsprecher an den Computer angeschlossen werden. Dazu wird ein Anschluß des Lautsprechers mit F1, F2 oder F3 verbunden, der andere über dem Widerstand 100 Ω mit + 5 Volt der Steckerleiste auf der Computerplatine (Abb. rechts). Eine verbesserte Schaltung, bei der die Lautstärke geregelt werden kann, zeigt die linke Abb.



Tasten: **(RS)** **SP** **6**

Der Ton, den der Lautsprecher erzeugt, hängt von der Geschwindigkeit ab, mit der die Lautsprechermembrane bewegt wird. Der Computer ändert bei diesem Spiel ständig diese Geschwindigkeit und damit die Tonhöhe.

1.3.7 Spiel 7: elektronischer Würfel

Schalter S nach hinten; Tasten: (RS) SP 7

Unser Würfel zeigt zunächst eine 1. Wird die Taste SB betätigt, so zeigen die vier etwas ungewöhnlichen Fragezeichen das Würfeln an. Wird die Taste SB nicht mehr betätigt, dann zeigt unser Würfel wieder eine Zahl zwischen 1 und 6. Wie bei einem richtigen Würfel kommt es vor, daß eine Zahl häufiger nacheinander angezeigt wird oder daß man längere Zeit auf eine bestimmte Zahl warten muß.

1.3.8 Spiel 8: Zahlenlotto „6 aus 49“

Schalter S nach hinten; Tasten: (RS) SP 8

Es erscheint die Anzeige 6 AUS 49. Die Anzeige verschwindet, wenn die Taste SA betätigt wird. Wird die Taste SA wieder losgelassen, so erscheint kurzzeitig die erste Lottozahl. Anschließend ist wieder 6 AUS 49 zu sehen. Die nächste Lottozahl kann ermittelt werden. Nach der sechsten Lottozahl kann die Anzeige 6 AUS 49 nur mit der RS-Taste gelöscht werden.

Der Computer ermittelt die Lottozahlen mit Hilfe eines Zufallszahlengenerators; er überprüft, ob die Zahl zwischen 1 und 49 liegt; er sorgt dafür, daß jede Zahl nur einmal ausgewählt wird; und schließlich läßt er nur die Auswahl von 6 Lottozahlen zu.

1.3.9 Spiel 9: Zweier-Potenzen

Schalter S nach hinten; Tasten: (RS) SP 9

Dieses und das nächste Spiel zeigen, daß die Bezeichnung „Spiele“ nicht für alle fest vorgesehenen Programme zutrifft. Der Computer berechnet hier nacheinander die Zweier-Potenzen von 1 bis 8192 und zurück; er zeigt rechts die Zweier-Potenz, links den Exponenten an.

Man erhält die Zweier-Potenz, indem man zunächst 1, dann jeweils das Ergebnis immer wieder mit 2 multipliziert. Der Exponent gibt an, wie oft mit 2 multipliziert wurde.

Beispiel:

$$(1 \cdot) \underbrace{2 \cdot 2 \cdot 2 \cdot 2 \cdot 2}_{5 \text{ mal}} = 32$$

In der Mathematik schreibt man auch $2^5 = 32$ (sprich: 2 hoch 5 gleich 32). Die 2 nennt man die Basis oder die Grundzahl, die 5 heißt Exponent, das Ergebnis 32 ist die fünfte Potenz von 2. Die nullte Potenz wird gleich 1 gesetzt. So ergeben sich die Zweier-Potenzen, die bei diesem „Spiel“ dargestellt werden:

Zweier-Potenzen bis 8192	
Exponent	Potenz
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192

1.3.10 Spiel 10: Taschenrechner

Schalter S nach hinten; Tasten: (RS) SP A

Dieses Spiel zeigt, daß unser Computer auch Aufgaben wie ein einfacher Taschenrechner lösen kann.

Die Zifferntasten 0 1 9 dienen zur Zahleneingabe; sieben der acht Funktionstasten erhalten für den Taschenrechner folgende Bedeutung (siehe A. 5, Seite 160):

	+ / -	CI	↔ Clear-Taste (Lösch-Taste)
		-	↔ Minus-Taste
CI	÷	+	↔ Plus-Taste
		+ / -	↔ Taste für Vorzeichenwechsel
-	×	÷	↔ Divisions-Taste
		×	↔ Multiplikations-Taste
+	=	=	↔ Ergebnis-Taste)

Unser Taschenrechner ist für alle vier Grundrechenarten (+, -, x, ÷) geeignet. Er rechnet vorzeichenrichtig, kann allerdings nur mit vierstelligen Zahlen rechnen.

Wenn das Taschenrechnerprogramm mit RS SP (=) A gestartet oder wenn bei der Benutzung unseres Taschenrechners die Taste CI betätigt wird, wird in der Anzeige eine 0 angezeigt, der Computer wartet auf die Eingabe der ersten Zahl.

Wird die erste Zahl Ziffer für Ziffer eingegeben, so wird die eingegebene Zahl bei jeder weiteren Ziffer nach vorn geschoben. Bei der Eingabe der fünften Ziffer geht die erste verloren. So kann auch eine irrtümlich falsch eingegebene Zahl durch eine neue Eingabe überschrieben werden.

Jetzt wird normalerweise das Verknüpfungszeichen eingegeben. Werden mehrere Verknüpfungszeichen nacheinander eingegeben, so gilt das zuletzt eingegebene. Bei der Division wird grundsätzlich abgerundet. Unser Taschenrechner kennt kein Komma. Die Nachkommastellen werden verschluckt. Mit der Taste + / - wird das Vorzeichen der zuletzt eingegebenen Zahl verändert.

Eine Rechnung wird mit der =-Taste abgeschlossen. Es erscheint in der Anzeige ein E und das Ergebnis der Rechnung. Wird jetzt eine Zifferntaste betätigt, so wird diese Eingabe als erste Zahl einer neuen Rechnung aufgefaßt. Wird bei der Anzeige eines Ergebnisses ein neues Verknüpfungszeichen gewählt, so verschwindet das E, das Ergebnis der letzten Rechnung gilt als erste Zahl der folgenden Rechnung.

Wird bei der Eingabe einer Zahl statt der erlaubten Tasten 0 1 9 eine der Tasten A B F gewählt, so wird bei der anschließenden Betätigung einer Verknüpfungstaste Error (engl. Irrtum, Fehler) angezeigt. Error erscheint ebenfalls, wenn bei der Berechnung eines Ergebnisses die Zahl 9999 überschritten wird.

Mit RS SP (=) A kann das Taschenrechnerprogramm neu gestartet werden.

Beispiele:

Taste	Anzeige
RS	HALLO
SP	SPIELE
A	0
1	1
7	1 7
+	1 7
4	4
=	E 2 1

$$17 + 4 = 21$$

Taste	Anzeige
5	5
x	5
1	1
0	1 0
+/-	1 0
=	E - 5 0

$$5 \times (-10) = -50$$

Taste	Anzeige
5	5
0	5 0
÷	5 0
3	3
=	E 1 6

$$50 \div 3 = 16,66$$

Taste	Anzeige
3	3
2	3 2
x	3 2
3	3
2	3 2
x	1 0 2 4
1	1
0	1 0
=	E r r o r
RS	HALLO
SP	SPIELE
A	0

$$32 \times 32 \times 10 > 9999$$

(das Zeichen > heißt „größer als“)

Taste	Anzeige
1	1
3	1 3
+/-	1 3
+	1 3
2	2
5	2 5
-	1 2
7	7
+/-	7
x	1 9
9	9
-	1 7 1
2	2
0	2 0
0	2 0 0
=	E - 2 9

$$(-13 + 25 - (-7)) \times 9 - 200 = -29$$

1.3.11 Spiel 11: Reaktionstest

Schalter S nach hinten; Tasten: (RS) SP B

Dieser Reaktionstest ist ein Spiel gegen die Zeit. Zu Beginn steht in der Anzeige 00. Anschließend erscheint achtfach eine der Ziffern 0 bis 9 oder einer der Buchstaben A bis F. Es ist jetzt die Aufgabe, die entsprechende Taste zu drücken. Wurde die richtige Taste gedrückt, so zeigt der Computer 01 an. Er zählt mit, wie oft wir es geschafft haben, in der vorgegebenen Zeit die geforderte Taste zu drücken. Das Spiel wiederholt sich. Unsere Aufgabe ist es nur, immer dann die geforderte Taste zu betätigen, wenn die achtfache Anzeige erscheint.

Ein Zufallszahlengenerator entscheidet, welche Ziffer oder welcher Buchstabe achtfach angezeigt wird. Er entscheidet auch, wie lange die Anzahl der „Richtigen“ angezeigt wird. Die Zeit, die zur Betätigung der geforderten Taste zur Verfügung steht, wird von Mal zu Mal kürzer. Wird eine falsche Taste oder wird in der vorgegebenen Zeit keine Taste betätigt, so ist das Spiel beendet. Der Computer gibt an, wie oft wir den Test bestanden haben, und er zeigt das Wort AUS.

Ein neuer Start wird mit den Tasten RS SP B erreicht. Es ist das Ziel, ein möglichst hohes Ergebnis zu erreichen. Wer sich ein wenig eingespielt hat, wird ohne Schwierigkeiten 10 „Richtige“ erreichen. Ein Könner erreicht leicht 20 „Richtige“.

1.3.12 Spiel 12: Laufschrift

Schalter S nach hinten; Tasten: (RS) SP C

Statt C kann auch eine der Tasten O, oder D, gewählt werden.

Die beiden Wörter SPIELE und HALLO werden nacheinander – mit den entsprechenden Zwischenräumen – ständig von rechts nach links durchgeschoben. Es ließen sich auch andere Wörter wählen, aber ganz frei sind wir in der Wortwahl nicht. Bei Spiel 4 haben wir gesehen, daß uns von der Anzeige her bestimmte Grenzen gesetzt werden.

Die Laufschrift wird – wie alle anderen Spiele – mit der Taste RS abgebrochen. Unser Computer meldet sich wieder mit dem unbewegten HALLO.

2. Eingabe und Start eines Programms

2.1 Vorbereitungen für die Programmeingabe

Im ersten Kapitel haben wir zwölf Programme erprobt, die fest in unserem Computer gespeichert sind. Diese Programme lassen sich durch Betätigung der **SP**-Taste und einer Zifferntaste starten.

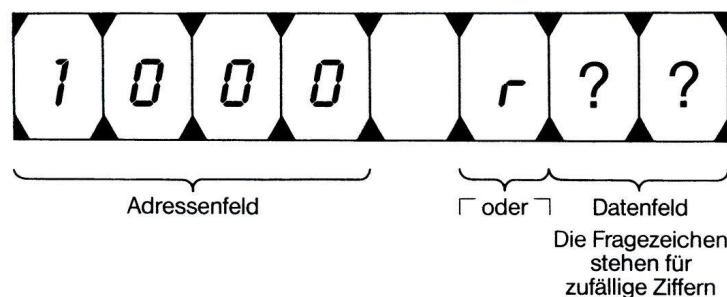
Die Vielseitigkeit eines Computers besteht aber gerade darin, daß er immer wieder andere Programme bearbeiten kann. Man gibt ein Programm ein und gibt dem Computer anschließend den Befehl, dieses Programm auszuführen. Und er tut es auch, stets mit der gleichen Exaktheit!

Was ist ein Programm? Es ist eine Folge von Einzelbefehlen, die der Computer versteht, die er dann nach Programmstart sehr zuverlässig Befehl für Befehl ausführt. Welche Befehle unser Computer versteht, kann erst nach und nach deutlich werden. Zunächst verbergen sich die Befehle hinter einer Folge von Ziffern und Buchstaben.

Wenn noch ein Programm läuft (z. B. eines der Spiele 1 bis 12), dann müssen wir es zunächst mit **RS** unterbrechen. Betätigen wir jetzt die Taste **A ↔ D** einmal (oder eventuell zweimal; mehrfache Betätigung schadet nichts), dann steht in den linken vier Sieben-Segment-Anzeigen 1000 (sprich: eins – null – null – null). 1000 ist die Anfangsadresse, von der ab wir unsere Programme eingeben werden. Davon werden wir nur in Sonderfällen abweichen.

Jetzt haben wir schon wieder eine Bezeichnung benutzt, die erklärt werden muß: die Adresse. Wenn wir eine Folge von Ziffern und Buchstaben (die eigentlich unser Programm darstellen sollen) in unseren Computer eingeben wollen, dann muß das, was eingegeben wird, gespeichert werden. Auf unserer Computer-Platine sind über 3500 Speicherplätze vorhanden. Diese Speicherplätze (oder kurz: Speicher) müssen eine Adresse haben, sonst würden wir das Eingespicherte nur schwer wiederfinden. Oder wir wollen nur den Inhalt in einem bestimmten Speicher ändern. Wie sollten wir das machen, wenn wir nicht seine Adresse wüßten? Die Adressen werden bei unserem Computer grundsätzlich vierstellig angegeben, wobei neben den Ziffern 0 bis 9 auch die Buchstaben A bis F verwendet werden.

Unser Programm soll – wie oben gesagt – bei der Adresse 1000 beginnen. In der Anzeige steht – wenn wir inzwischen nichts verändert haben – links die vierstellige Adresse 1000, rechts steht zweistellig der Inhalt des Speichers, dessen Adresse 1000 ist. Auch der Inhalt eines Speichers – oder die Daten – werden mit den Ziffern 0 bis 9 oder den Buchstaben A bis F angegeben.



Es läßt sich nicht voraussagen, welchen Inhalt der Speicher mit der Adresse 1000 z. Zt. hat. Dieser Inhalt ist beim Einschalten des Computers entstanden. Die beiden Fragezeichen für die beiden rechten Sieben-Segment-Anzeigen sollen diese zufälligen, nicht feststehenden Werte andeuten.

In der dritten Sieben-Segment-Anzeige von rechts wird ein Haken dargestellt: **┐** oder **┌**. Dieses Symbol sagt uns, ob der Computer die nächste Eingabe als Adressen- oder als Dateneingabe erwartet. **┐** bedeutet: Die nächste Eingabe über das Tastenfeld (0 bis 9, A bis F) wird als Adresseneingabe aufgefaßt. Es wird dann der Speicher mit dieser Adresse ausgewählt. Der senkrechte Strich des Symbols **┐** befindet sich links auf der Seite des Adressenfeldes.

┌ bedeutet: Die nächste Eingabe über das Tastenfeld wird als Dateneingabe aufgefaßt. Wird jetzt eine Taste betätigt, so wird dadurch der Inhalt in dem Speicher geändert, dessen Adresse im Adressenfeld links angezeigt wird. Der senkrechte Strich des Symbols **┌** befindet sich rechts auf der Seite des Datenfeldes.

Damit kennen wir die beiden Funktionen der Taste **A ↔ D**. Bei einmaliger (eventuell auch zweimaliger) Betätigung der Funktionstaste **A ↔ D** erscheint im Adressenfeld die Adresse 1000. Jede weitere Betätigung dieser Taste ändert dann nur noch das Symbol in der dritten Sieben-Segment-Anzeige von rechts. Damit wird der Computer gleichzeitig für eine Adresse oder eine Dateneingabe vorbereitet. Daher auch die Bezeichnung **A ↔ D**.

Beginnen wir jetzt bei der Anzeige

1	0	0	0	┐	?	?
---	---	---	---	---	---	---

Diese Anzeige läßt sich – falls sie nicht mehr vorhanden ist – mit **RS** und ein- oder zweimal **A ↔ D** erreichen. Die beiden Fragezeichen stehen für zwei Ziffern oder Buchstaben, die nicht voraussagbar sind.

Taste **1**, Anzeige:

0	0	0	1	┐	2	4
---	---	---	---	---	---	---

Wir haben den Speicher mit der Adresse 0001 angewählt. In diesem Speicher steht der Inhalt 24.

Taste **0**, Anzeige:

0	0	1	0	┐	C	2
---	---	---	---	---	---	---

Wir haben die Adresseneingabe mit 0 fortgesetzt. Wir haben jetzt den Speicher mit der Adresse 0010 angewählt. In diesem Speicher steht der Inhalt C2.

Taste **0**, Anzeige:

0	1	0	0	┐	D	8
---	---	---	---	---	---	---

Im Speicher mit der Adresse 0100 steht der Inhalt D8. D und B werden in der Anzeige als kleine Buchstaben dargestellt, schreiben werden wir diese Buchstaben aber weiterhin groß. Die Inhalte 24, C2 und D8 in den Speichern mit den Adressen 0001, 0010 und 0100 liegen fest, da diese Speicher zu dem Speicherbereich gehören, der auch beim Ausschalten des Gerätes seinen Inhalt behält. Diese Inhalte lassen sich auch nicht ändern.

Taste **0**, Anzeige:

1	0	0	0	┐	?	?
---	---	---	---	---	---	---

Wir erhalten wieder die Anzeige wie am Anfang.

Taste **A ↔ D**, Anzeige:

1	0	0	0	┌	?	?
---	---	---	---	---	---	---

Die nächste Eingabe wird als Dateneingabe interpretiert.

Taste **6**, Anzeige:

1	0	0	0	┌	0	6
---	---	---	---	---	---	---

In dem Speicher mit der Adresse 1000 steht jetzt der Inhalt 06. Der alte Speicherinhalt wird überschrieben. In der vorderen Stelle des Datenfeldes erscheint automatisch eine 0. Bei der Eingabe 06 braucht also nur 6 eingetastet zu werden.

Taste **A**, Anzeige:

1	0	0	0			6	A
---	---	---	---	--	--	---	---

Jetzt wird die 6 nach vorne geschoben, das A wird angefügt. Für den Computer war die Dateneingabe für den Speicher mit der Adresse 1000 nicht abgeschlossen. Diese Dateneingabe läßt sich fortsetzen, bis wir mit dem, was im Speicher mit der Adresse 1000 steht, einverstanden sind.

Taste **3**, Anzeige:

1	0	0	0			A	3
---	---	---	---	--	--	---	---

Taste **A ↔ D**, Anzeige:

1	0	0	0			A	3
---	---	---	---	--	--	---	---

Taste **A ↔ D**, Anzeige:

1	0	0	0			A	3
---	---	---	---	--	--	---	---

Taste **3**, Anzeige:

1	0	0	0			0	3
---	---	---	---	--	--	---	---

Da zwischendurch eine Funktionstaste betätigt wurde, hat jetzt für den Speicher mit der Adresse 1000 eine neue Dateneingabe begonnen.

2.2 Programmeingabe

Wir wollen jetzt in den Computer unser erstes Programm eingeben. Im Speicher mit der Adresse 1000 sollen die Daten C4 stehen.

Tasten: (**RS**) **A ↔ D** (eventuell mehrmals), Anzeige:

1	0	0	0			?	?
---	---	---	---	--	--	---	---

Tasten: **C**, **4**, Anzeige:

1	0	0	0			C	4
---	---	---	---	--	--	---	---

In den nächsten Speicher mit der Adresse 1001 sollen die Daten 02 eingeschrieben werden. Mit den Tasten **A ↔ D**, **1**, **0**, **0**, **1**, **A ↔ D** könnten wir den Computer für die Dateneingabe für den Speicher mit der Adresse 1001 vorbereiten. Um dieses umständliche Verfahren abzukürzen, gibt es die Taste **ME +**. ME ist eine Abkürzung des englischen Wortes memory (Gedächtnis). Mit der Taste **ME +** (memory plus) können wir den nächsten Speicherplatz des „Gedächtnisses“ unseres Computers aufrufen. Entsprechend läßt sich mit der Taste **ME -** (memory minus) der vorhergehende Speicherplatz aufrufen.

Die Anzeige sei jetzt:

1	0	0	1			?	?
---	---	---	---	--	--	---	---

Taste **2**, Anzeige:

1	0	0	1			0	2
---	---	---	---	--	--	---	---

Entsprechend geben wir unser Programm weiter ein:

Tasten:

ME +	7	
ME +	6	
ME +	D	4
ME +	1	0
ME +	6	C
ME +	F	8
ME +	C	4
ME +	8	
ME +	7	
ME +	7	4
ME +	F	6

Anzeige:

1	0	0	2			0	7
1	0	0	3			0	6
1	0	0	4			D	4
1	0	0	5			1	0
1	0	0	6			6	C
1	0	0	7			F	8
1	0	0	8			C	4
1	0	0	9			0	8
1	0	0	A			0	7
1	0	0	B			7	4
1	0	0	C			F	6

2.3 Kontrolle eines eingegebenen Programms

Wenn wir überprüfen wollen, ob das Programm richtig eingegeben wurde, müssen wir der Reihe nach die Speicherinhalte prüfen. In der folgenden Tabelle sind noch einmal die eingegebenen Daten zusammengestellt. Wir wählen mit der Taste **A ↔ D** wieder den Speicherplatz mit der Adresse 1000 an und kontrollieren den eingespeicherten Inhalt. Nach Betätigung der Taste **ME +** zeigt uns der Computer den Speicherinhalt mit der Adresse 1001. So fahren wir fort und kontrollieren das gesamte Programm. Sollte ein Speicherinhalt nicht richtig eingegeben worden sein, kann der Inhalt in diesem Speicher einfach überschrieben werden. Es wird der entsprechende Speicherplatz aufgerufen (mit Hilfe der Adresse), der Computer wird auf Dateneingabe geschaltet (↵), die richtigen Daten werden eingetastet. Die Inhalte in den anderen Speicherplätzen ändern sich dadurch nicht.

ADRESSE	INHALT
1000	C4
1001	02
1002	07
1003	06
1004	D4
1005	10
1006	6C
1007	F8
1008	C4
1009	08
100A	07
100B	74
100C	F6

2.4 Programmstart

Wenn ein Programm bei der Adresse 1000 beginnt, so läßt es sich folgendermaßen starten: Wir wählen mit der Taste **A ↔ D** den Speicher mit der Adresse 1000, wo unser Programm beginnt und betätigen die Taste **RUN** (engl. to run – laufen). Wenn wir das bei unserem Programm tun, so erlischt die Anzeige, die grüne Leuchtdiode leuchtet. Mehr tut sich nicht. Der Computer „beobachtet“ aber die Taste **SA**. Wenn wir diese Taste betätigen, leuchtet statt der grünen Leuchtdiode die rote. Es ist sicher kein aufregendes Programm, aber es ist das erste!

2.5 Abänderung des Programms

Wir können uns hier ansehen, wie leicht eine Änderung des Programms vorzunehmen ist. Wir ersetzen den Inhalt 10 im Speicherplatz mit der Adresse 1005 durch 20.

Tastenfolge:

RS, **ME +**, **ME +**, **ME +**, **ME +**, **ME +**, **ME +**, **2**, **0**

Nach einem neuen Programmstart (**A ↔ D**, **RUN**) wird jetzt vom Computer die Taste **SB** abgefragt. Die rote Leuchtdiode leuchtet, wenn diese Taste gedrückt wird.

Bei einer weiteren Änderung wollen wir in den Speicher mit der Adresse 1001 den Inhalt 04, in den Speicher mit der Adresse 1009 den Inhalt 0A einschreiben.

Tastenfolge:

RS, **ME +**, **4**, **A ↔ D**, **1**, **0**, **0**, **9**, **A ↔ D**, **A**

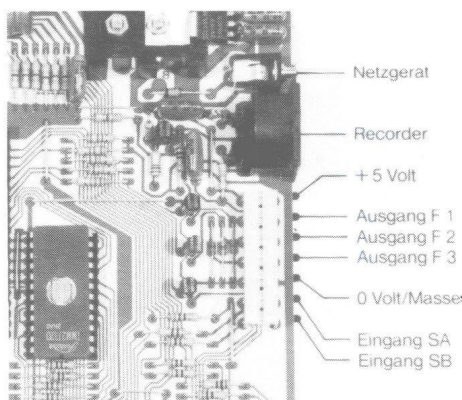
Nach einem neuen Programmstart (**A ↔ D**, **RUN**) leuchtet die gelbe Leuchtdiode. Bei Betätigung der Taste **SB** (wenn im Speicher mit der Adresse 1005 noch der Inhalt 20 vorhanden ist) leuchten die rote und die grüne Leuchtdiode.

3. Der Computer und seine Peripherie

3.1 Die Verbindungen mit der Peripherie

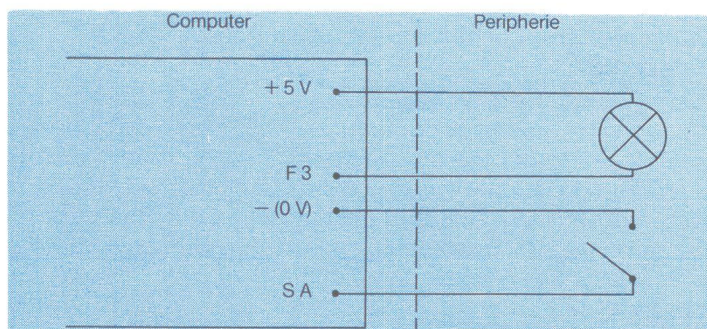
Unser Computer hat in der vorliegenden ersten Ausbaustufe neben den Kontakten für die Spannungsversorgung und der Buchse für den Cassettenrecorder sieben Anschlüsse nach „draußen“. Hier können unterschiedliche Geräte angeschlossen werden. Sie gehören nicht unmittelbar zu unserem Computer, sie gehören zur Peripherie.

Zwei der sieben Anschlüsse sind mit der Betriebsspannung unseres Computers verbunden: – und +5V. Über zwei Anschlüsse kann unser Computer Signale von außen bekommen: SA und SB (Sense A und Sense B, engl. sense – der Sinn, der Fühler). Drei Anschlüsse dienen zur Ausgabe von Signalen: F1, F2 und F3 (Flag 1, Flag 2 und Flag 3, engl. flag – die Flagge, Fahne).



Es wird sicher aufgefallen sein, daß die Bezeichnungen SA und SB auch für zwei Taster bei unserem Computer verwendet wurden. Es ist für die Signalverarbeitung im Computer egal, ob von der Peripherie ein Signal an den Eingang SA geliefert wird oder ob dieses Signal mit Hilfe des Tastschalters **SA** erzeugt wird. Andererseits gehen Signale, die über die Anschlüsse F1, F2 und F3 an die Peripherie geliefert werden, auch an die drei farbigen Leuchtdioden: F1 – grüne Leuchtdiode, F2 – gelbe Leuchtdiode, F3 – rote Leuchtdiode. Im letzten Abschnitt dieses Kapitels sind die Schaltbilder für diese Schaltungsteile angegeben.

Wir sorgen jetzt zunächst dafür, daß das erste Programm von Kapitel 2 wieder eingeschrieben ist (vgl. Abschnitt 2.2 Programmeingabe.). Wir verbinden die beiden Anschlüsse F3 und +5V mit einer Glühlampe, die beiden Anschlüsse SA und – mit einem externen Tastschalter.



Nach Programmstart (vgl. Abschnitt 2.4) betätigen wir den externen Tastschalter, die Glühlampe leuchtet. Der externe Tastschalter ist parallel zum Tastschalter **SA** geschaltet. Es ist egal, welchen Taster wir betätigen. Die Glühlampe

leuchtet jetzt zusätzlich zur schon vorhandenen Leuchtdiode. Verbinden wir die Glühlampe mit F1 und +5V, so leuchtet die Glühlampe immer dann, wenn keiner der beiden Tastschalter betätigt wird.

3.2 Die Glühlampe kann auch blinken

Bei dem folgenden Programm tut der Computer schon ein bißchen mehr. Glühlampe (zwischen F1 und +5V) und Tastschalter (zwischen SA und –) bleiben angeschlossen wie zuletzt beim ersten Programm.

Wir geben das nebenstehende Programm ein (vgl. Abschnitt 2.2); Tastenfolge: **RS**, **A↔D** (eventuell mehrmals), **3**, **9**, **ME+**, **0**, **ME+**, **6**, **ME+**, **D**, **4**, **ME+** usw. Nach Kontrolle (vgl. Abschnitt 2.3) und Programmstart (vgl. Abschnitt 2.4) können wir wieder den Taster betätigen. Die Glühlampe blinkt fünfmal.

ADRESSE	INHALT
1000	39
1001	00
1002	06
1003	04
1004	10
1005	6C
1006	FB
1007	06
1008	D4
1009	10
100A	7C
100B	FB
100C	C4
100D	0A
100E	CD
100F	E0
1010	06
1011	E4
1012	02
1013	07
1014	84
1015	00
1016	00
1017	1D
1018	9D
1019	E0
101A	7C
101B	F4
101C	74
101D	E2

Natürlich kann man auch bei diesem Programm leicht einige Veränderungen vornehmen. Wohlgemerkt: wir verändern nicht die elektrische oder elektronische Schaltung sondern einen Befehl im Programm.

Wir schreiben z. B. in den Speicher mit der Adresse 100D statt 0A den neuen Inhalt 14. Tastenfolge: **RS**, **1**, **0**, **0**, **D**, **A↔D**, **1**, **4**. Nach neuem Programmstart (Tastensequenz: **A↔D**, **RUN**) blinkt die Glühlampe nach Loslassen des Tastschalters zehnmal.

Wir schreiben z. B. in den Speicher mit der Adresse 1016 statt 00 den neuen Inhalt 80. Tastenfolge: **RS**, **1**, **0**, **1**, **6**, **A↔D**, **8**, **0**. Nach neuem Programmstart und Betätigung des Tastschalters blinkt die Glühlampe doppelt so schnell.

Wird in den Speicher mit der Adresse 1016 der Inhalt 40 eingegeben, blinkt die Glühlampe nach Betätigung des Tastschalters abermals doppelt so schnell. Sicher ist es interessant, was sich bei diesen Programmen im Innern des Computers abspielt. Es ist vorstellbar, daß der Computer bei diesem Programm den Eingang **SA** untersucht; daß er dann, wenn der Tastschalter an diesem Eingang losgelassen wird, die Lampe mehrmals hintereinander aufleuchten läßt. Wie er das macht, welche Befehle er nacheinander bearbeiten muß, kann aber erst später untersucht werden. Wir werden auf alle Programme, die wir hier nur eintasten und erproben, später ausführlich zurückkommen.

3.3 Ein Morseapparat

Die beiden folgenden Programme sollen uns zeigen, daß nicht nur Glühlampe und Tastschalter als Peripherie-Geräte erlaubt sind.
Wir schließen statt der Glühlampe den Lautsprecher an (zwischen F1 und + 5V), der Tastschalter bleibt unverändert zwischen SA und - . Damit ist der Morseapparat fertig. Wir müssen nur noch ein anderes Programm eingeben. Zur Eingabe des Programms vgl. die Abschnitte 2.2 und 3.2. Falls das letzte Programm noch gespeichert ist, brauchen erst ab Adresse 1006 neue Daten eingetastet zu werden.

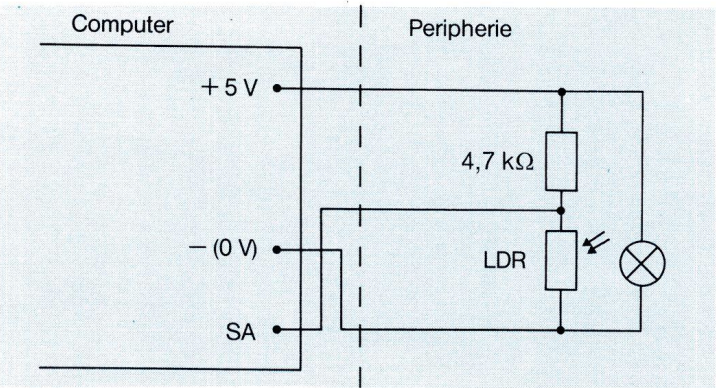
ADRESSE	INHALT
1000	39
1001	00
1002	06
1003	04
1004	10
1005	6C
1006	F9
1007	06
1008	E4
1009	02
100A	07
100B	84
100C	20
100D	00
100E	10
100F	74
1010	F1

Ist das Programm richtig eingegeben und gestartet worden (Tasten: **A↔D**, **RUN**), dann ist der Morseapparat endgültig fertig.
Solange der Taster betätigt wird, ertönt ein Ton bestimmter Höhe. Hier läßt sich natürlich die Tonhöhe ändern. Eine kleinere Zahl als 20 im Speicher mit der Adresse 100C bewirkt einen höheren, eine größere Zahl bewirkt einen tieferen Ton.

3.4 Eine Lichtschranke

3.4.1 Aufbau und Überprüfung

Wir bauen zunächst eine kleine Schaltung auf. Der Widerstand (1 kΩ) und der LDR (lichtabhängiger Widerstand, eng. light dependent resistor) werden hintereinandergeschaltet. Der Widerstand wird an den Anschluß + 5V, der LDR an - angeschlossen. Die Verbindungsstelle zwischen beiden Bauteilen wird mit dem Eingang SA verbunden. Die Glühlampe wird etwa 1 cm bis 2 cm vom LDR befestigt. Die gestreifte Seite des LDR muß zur Glühlampe zeigen.



Wir bereiten noch einen kleinen Pappstreifen (etwa 10 cm × 1 cm) vor, den wir zwischen Glühlampe und LDR halten können. Der LDR ändert seinen Widerstand, wenn er beleuchtet wird oder wenn er wieder abgedunkelt wird. Jetzt muß der Computer nur noch so programmiert werden, daß er merkt, ob sich der Pappstreifen in der Lichtschranke befindet oder nicht.
Zunächst geben wir ein sehr einfaches Programm ein, mit dem wir eigentlich nur die richtige Funktion der Lichtschranke überprüfen wollen. Wenn dieses Programm eingegeben und gestartet ist, sollte die rote der drei farbigen Leuchtdioden leuchten, solange der LDR beleuchtet ist; wird der LDR abgedunkelt, so erlischt die Leuchtdiode. Wenn die Leuchtdiode nicht so wie erwartet reagiert, dann müssen wir den Fehler suchen.

ADRESSE	INHALT
1000	06
1001	04
1002	10
1003	3C
1004	07
1005	74
1006	F9

Eine häufige Fehlermöglichkeit ist das Programm. Wir sollten es noch einmal kontrollieren (**RS**, **ME+**, **ME+**...). Als zweites sollten wir die Schaltung der Peripherie noch einmal überprüfen. Eventuell muß die Glühlampe dichter an die gestreifte Seite des LDR geschoben werden. Statt der Glühlampe kann auch eine helle Schreibtischlampe oder Taschenlampe verwendet werden.

3.4.2 Diebstahlsicherung

Wenn die Lichtschranke wie geplant funktioniert, können wir sie variieren. Wir schließen zusätzlich den Lautsprecher wieder zwischen F1 und +5V an und ändern das Programm:

ADRESSE	INHALT
1000	06
1001	04
1002	10
1003	7C
1004	FB
1005	24
1006	51
1007	04

Bei gut beleuchtetem LDR starten wir jetzt das Programm. Unsere Diebstahlsicherung ist fertig. Sowie der LDR verdunkelt wird, beginnt die Sirene, die wir aus Spiel 6 (vgl. Abschnitt 1.3.6) kennen. Die Sirene läßt sich dann nur mit der Taste **RS** abstellen.

3.4.3 Lichtschranke und Zähler

Unser Computer kann bei Abdunklung des LDR nicht nur die rote Leuchtdiode aus- oder die Sirene einschalten. Er kann noch mehr. Das ist nur eine Frage des Programms. Das folgende etwas längere Programm zeigt, daß der Computer z. B. auch Gegenstände zählen kann, die sich nacheinander durch die Lichtschranke bewegen. Wir tasten das folgende Programm ein und starten es bei der Adresse 1000.

ADRESSE	INHALT
1000	12
1001	C4
1002	00
1003	CD
1004	E0
1005	1C
1006	CD
1007	D8
1008	15
1009	C4
100A	08
100B	11
100C	06
100D	D4
100E	10
100F	7C
1010	F8
1011	C4
1012	08
1013	11
1014	06
1015	D4
1016	10
1017	6C
1018	F8
1019	95
101A	E0
101B	74
101C	E8

In der Anzeige des Computers steht die Zahl 00. Wenn wir jetzt den Pappstreifen einmal durch die Lichtschranke bewegen, zeigt die Anzeige die Zahl 01 usw.: Der Computer zählt mit, wie oft die Lichtschranke durch den Pappstreifen unterbrochen wurde, wie viele Gegenstände sich durch die Lichtschranke bewegt haben.

3.5 Programmstart nur bei Adresse 1000?

Wir haben unsere Programme bisher immer im Speicher mit der Adresse 1000 begonnen. Das hat den Vorteil, daß wir das Programm sehr einfach starten können: **A↔D** (eventuell zweimal), **RUN**.

Vielleicht haben wir dabei schon bedauert, daß wir mit der Eingabe eines neuen Programms das vorherige überschreiben. Vielleicht wollen wir mehrere Programme eintasten, die wir dann im Wechsel starten wollen. Das ist durchaus möglich.

In dem folgenden Ausdruck ist angegeben, wie z. B. die drei Programme zur Lichtschranke aus den letzten drei Abschnitten (3.4.1, 3.4.2 und 3.4.3) gemeinsam eingetastet sein können. Das Programm zur Überprüfung der Lichtschranke beginnt ab Adresse 1000, das Programm „Diebstahlsicherung“ beginnt ab 1100, das Programm „Lichtschranke und Zähler“ beginnt ab Adresse 1200.

Wenn wir diese drei Programme eingetastet haben und mit **A↔D**, **RUN** starten, läuft immer das erste Programm, das Programm ab Adresse 1000.

Das Starten eines Programms mit einer anderen Anfangsadresse als 1000 ist etwas aufwendiger. Das Programm mit der Anfangsadresse 1100 können wir mit folgender Tastenfolge starten:

CPU, 0, 1, 1, 0, 0, **CPU**, **RUN**, **RUN**.

Das Programm mit der Anfangsadresse 1200 starten wir entsprechend mit der Tastenfolge:

CPU, 0, 1, 2, 0, 0, **CPU**, **RUN**, **RUN**. Mit Hilfe dieser Tastenfolge haben wir den Programmzähler PC mit 1100, bzw. mit 1200 geladen und das entsprechende Programm von hier gestartet. Wir werden im Abschnitt 10.1, Seite 53, den Programmzähler ausführlich besprechen.

3.6 Ein kleiner Ausblick

Wir haben jetzt schon einen kleinen Eindruck von dem, was unser Computer kann. Wir haben die zwölf Spiele durchgespielt, und wir haben gesehen, daß wir selbst unterschiedlichste Programme eingeben können. Der Computer kann Informationen von der Peripherie bekommen, kann diese verarbeiten, und er kann wieder Signale an die Peripherie geben. Eine wesentliche Feststellung soll noch einmal betont werden: Bei dem gleichen gerätemäßigen Aufbau (bei der gleichen Hardware) können sich ganz unterschiedliche Dinge abspielen, einfach dadurch, daß das Programm (die Software) geändert wurde. In dieser Vielseitigkeit liegen gerade die großen Vorteile für den Einsatz eines Computers.

„Hardware“ und „Software“ sind zwei wesentliche Begriffe in der Computertechnik. Mit Hardware (engl. hard ware – harte Ware) bezeichnet man den technischen, materiellen Teil des Computersystems. Die Software (engl. soft ware – weiche Ware) bezeichnet das Nichtmaterielle, also im wesentlichen die Programme.

Bei der Software unterscheidet man zwischen System- und Anwender-Software. Die System-Software ermöglicht es, daß wir mit dem Computer arbeiten können. Sie sorgt dafür, daß bestimmte Funktionen ausgeführt werden, wenn wir die Funktionstasten drücken; sie gestattet es, daß wir Daten über die Tastatur eingeben können; sie sorgt dafür, daß wir unter bestimmten Voraussetzungen eine Anzeige erhalten. Die von uns für ein bestimmtes Problem eingegebenen Programme gehören zur Anwender-Software.

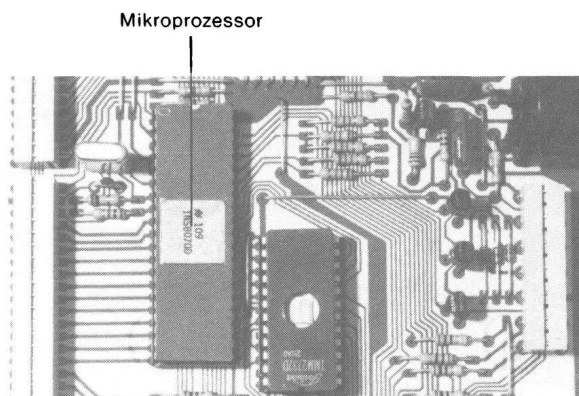
Man kann nicht sagen, daß die Hardware wichtiger sei als die Software oder umgekehrt. Nur im Zusammenspiel zwischen Hardware und Software ergibt sich etwas Sinnvolles, dann sogar etwas besonders Faszinierendes.

	ADRESSE	INHALT
Anfangsadresse 1000 Überprüfung der Lichtschranke	1000	06
	1001	D4
	1002	10
	1003	3C
	1004	07
	1005	74
	1006	F9
Anfangsadresse 1100 Diebstahlsicherung	1100	06
	1101	D4
	1102	10
	1103	7C
	1104	F8
	1105	24
	1106	51
	1107	04
Anfangsadresse 1200 Lichtschranke und Zähler	1200	12
	1201	C4
	1202	00
	1203	CD
	1204	E0
	1205	1C
	1206	CD
	1207	D8
	1208	15
	1209	C4
	120A	08
	120B	11
	120C	06
	120D	D4
	120E	10
	120F	7C
	1210	F8
	1211	C4
	1212	08
	1213	11
	1214	06
	1215	D4
	1216	10
	1217	6C
	1218	F8
	1219	95
	121A	E0
	121B	74
	121C	E8

Wir haben unser Gerät bisher immer als Computer bezeichnet. Etwas treffender wäre sicher die Bezeichnung „Mikroprozessor-System“. Das zentrale IC (engl. Integrated Circuit – integrierte Schaltung) auf unserer Computer-Platine ist der Mikroprozessor.

Ein Mikroprozessor-System erhebt nicht den Anspruch, einen guten Taschenrechner bei mathematischen Problemen zu schlagen. Gewiß, er kann auch Rechnungen durchführen (vgl. Spiel 10: Taschenrechner), aber das ist nicht seine ausschließliche Aufgabe. Ein Mikroprozessor-System kann auf der anderen Seite vieles, was ein Taschenrechner nicht kann. Er kann unterschiedlichste Prozesse steuern und regeln (vgl. unser Programm für die Lichtschranke).

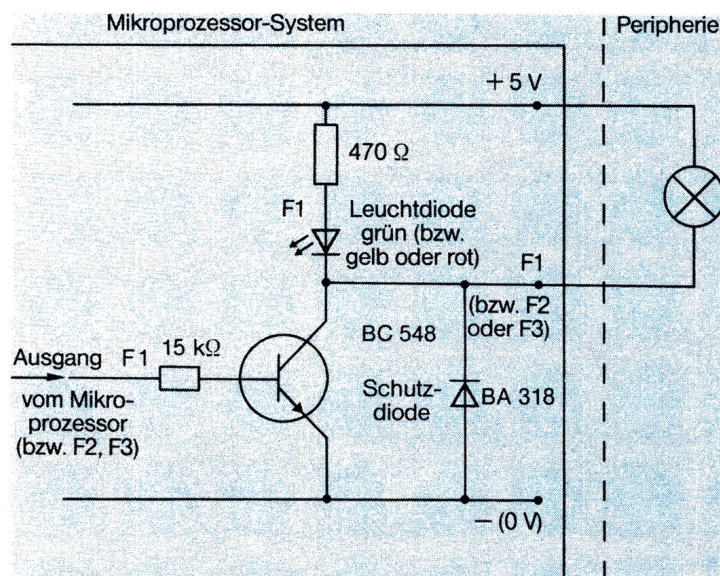
Der Mikroprozessor kann neben der Glühlampe und dem Lautsprecher auch andere elektrische und elektronische Geräte ansteuern, er kann neben Tasten und LDR auch von anderen Schaltungsteilen Signale empfangen. Wir werden darauf zurückkommen. Aber bis dahin führt noch ein langer und vielleicht auch mühsamer Weg. Wir müssen uns erst ansehen, was sich im Innern unseres Computers oder Mikroprozessor-Systems tut; wir müssen erst lernen, welche Befehle unser Computer versteht, und wie wir diese Befehle zu einem sinnvollen Programm kombinieren können.



3.7 Etwas Elektronik

Die folgenden beiden Schaltbilder sollen verständlich machen, wie die Schaltungen für die drei Ausgänge zur Peripherie und wie die beiden Eingänge von der Peripherie in das Mikroprozessor-System aussehen.

3.7.1 Die Schaltung für die Ausgänge

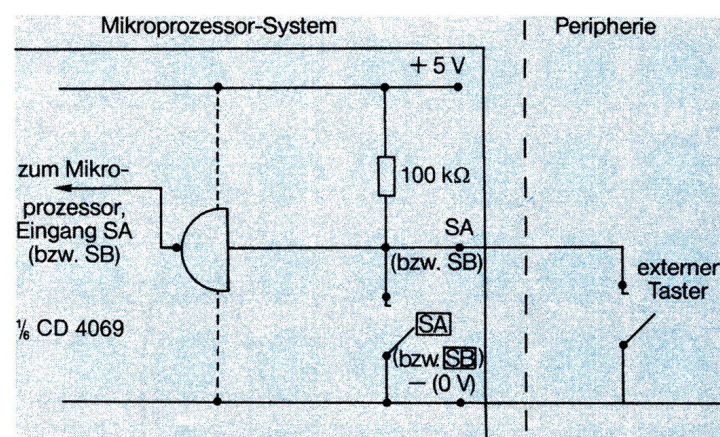


Der Mikroprozessor steuert den Transistor an. Wenn der Transistor durchschaltet, fließt ein Strom durch die Leuchtdiode bzw. durch das parallel geschaltete Bauteil (wie Glühlampe) der Peripherie.

Die Bezeichnung F1 (Flag 1) wird dreimal verwendet (entsprechend gilt das für F2 und F3). Mit F1 wird zunächst ein Ausgang des Mikroprozessors bezeichnet. Außerdem steht die Bezeichnung F1 an der grünen Leuchtdiode und an dem zugehörigen Ausgang der rechten Federleiste. Aus dem Textzusammenhang wird aber immer ersichtlich sein, welchen Punkt der Schaltung wir meinen.

Wir sollten uns merken, daß die grüne Leuchtdiode F1 leuchtet, daß am Ausgang F1 an der Federleiste (zwischen F1 und –) eine niedrige Spannung liegt, wenn der Mikroprozessor an seinem Ausgang F1 eine hohe Spannung liefert, die den Transistor durchsteuert.

3.7.2 Die Schaltung für die Eingänge



Wird der externe Taster bzw. die Taste SA in unserem Mikroprozessor-System betätigt, dann sperrt der Transistor. Der Mikroprozessor bekommt ein anderes Signal als bei durchgesteuertem Transistor.

Hier wird die Bezeichnung SA (Sense A) mehrfach verwendet (entsprechend gilt das für SB). Betätigung des Tasters SA oder des parallel geschalteten externen Tasters bewirkt eine hohe Spannung am Eingang SA des Mikroprozessors. Wird das Signal am Eingang SA der Federleiste (zwischen diesem SA und –) nicht durch einen Taster, sondern durch eine externe Spannung erzeugt wie z. B. bei der Lichtschranke, so müssen wir uns merken: Ist die extern angelegte Spannung hoch, so bekommt der Mikroprozessor an seinem Eingang SA eine niedrige Spannung, und umgekehrt. Das heißt, der Transistor invertiert oder negiert das von außen angelegte Signal.

4. Dualzahlen und Hexadezimalzahlen

4.1 Zahlenraten

Für das Spiel werden die folgenden sechs Karten benötigt.

1

32 33 34 35
36 37 38 39
40 41 42 43
44 45 46 47
48 49 50 51
52 53 54 55
56 57 58 59
60 61 62 63

2

16 17 18 19
20 21 22 23
24 25 26 27
28 29 30 31
48 49 50 51
52 53 54 55
56 57 58 59
60 61 62 63

3

8 9 10 11
12 13 14 15
24 25 26 27
28 29 30 31
40 41 42 43
44 45 46 47
56 57 58 59
60 61 62 63

4

4 5 6 7
12 13 14 15
20 21 22 23
28 29 30 31
36 37 38 39
44 45 46 47
52 53 54 55
60 61 62 63

5

2 3 6 7
10 11 14 15
18 19 22 23
26 27 30 31
34 35 38 39
42 43 46 47
50 51 54 55
58 59 62 63

6

1 3 5 7
9 11 13 15
17 19 21 23
25 27 29 31
33 35 37 39
41 43 45 47
49 51 53 55
57 59 61 63

Spieler A will zeigen, wie gut er Zahlen erraten kann. Er fordert Spieler B auf, sich eine Zahl kleiner als 64 (von 0 bis 63) zu denken. Anschließend zeigt Spieler A seinem Mitspieler nacheinander die sechs Karten und fragt jeweils, ob die gedachte Zahl darauf sei. Daraufhin sagt Spieler A die Zahl. Wie hat er das wissen können? Er hat natürlich nicht geraten. Er hat lediglich ein bißchen gerechnet. Er hat die Zahlen addiert, die links oben auf den Karten standen, auf denen auch die zu ratende Zahl stand.
Beispiel: Die gedachte Zahl sei 39. Sie steht auf den Karten 1, 4, 5 und 6. Auf diesen Karten stehen links oben die Zahlen 32, 4, 2 und 1. Die zu ratende Zahl ist dann $32 + 4 + 2 + 1 = 39$.

Die Erklärung für dieses „Rateverfahren“ ist gar nicht so schwer. Auf der Karte 1 stehen alle Zahlen größer als 31. Falls die gesuchte Zahl auf dieser Karte steht, läßt sich für eine Zerlegung dieser Zahl zunächst der Summand 32 abspalten. Auf Karte 2 stehen die Zahlen von 16 bis 31 und die von 48 bis 63. Falls die gesuchte Zahl auf dieser Karte steht, läßt sich – eventuell außer 32 – der Summand 16 festhalten. Es wird mit den drei nächsten Karten festgestellt, ob 8, 4 und 2 zur Zerlegung der gesuchten Zahl gehören. Mit Karte 6 wird ermittelt, ob die gesuchte Zahl ungerade ist. In diesem Fall muß zu der bisher ermittelten Summe eins addiert werden.

Anders formuliert: Mit Hilfe der Karten wird die gedachte Zahl in eine Summe von Zweierpotenzen zerlegt, bzw. es wird festgestellt, welche Zweierpotenzen in einer Zerlegung dieser Zahl vorkommen, (vgl. Abschnitt 1. 2. 9, Spiel 9: Zweierpotenzen).

Für unser Beispiel gilt: $39 = 32 + 4 + 2 + 1$.
Wir können auch schreiben: $39 = 2^5 + 2^2 + 2^1 + 2^0$,
oder $39 = 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$.

4.2 Dualzahlen

Wir sehen uns zunächst noch einmal den Zähler in Spiel 4 an: Schalter S nach vorn; Tasten: (RS) SP 4; vgl. 1.3.4. Wir wissen noch: Wird bei der Anzeige 15 die Taste SA betätigt, so zählt der Zähler weiter bis 255. In den rechten Sieben-Segment-Anzeigen wird im Dezimal- oder Zehnersy-

stem gezählt. Dieses Zahlensystem ist uns geläufig. An jeder Stelle können zehn verschiedene Ziffern, die Ziffern 0, 1, ..., 9 verwendet werden. Ist beim Aufwärtszählen die höchste Ziffer 9 erreicht, so folgt nach der 9 wieder die 0, es wird zusätzlich ein Übertrag 1 an die voranstehende Stelle geliefert. Es gibt zehn verschiedene einstellige (0, ..., 9), 100 verschiedene zweistellige Dezimalzahlen (00, 01, ... 99) usw. Die Zahl 39 läßt sich im Dezimalsystem ausführlicher schreiben:

$39 = 3 \cdot 10 + 9$,
oder $39 = 3 \cdot 10^1 + 9 \cdot 10^0$.
Die Zahl 1983 läßt sich im Dezimalsystem ausführlich schreiben: $1983 = 1 \cdot 10^3 + 9 \cdot 10^2 + 8 \cdot 10^1 + 3 \cdot 10^0$ ($10^3 = 1000$, $10^2 = 100$, $10^1 = 10$, $10^0 = 1$).

Alle diese Überlegungen lassen sich in jedes andere Zahlensystem übertragen.

Unser Computer arbeitet im Dual-, Binär- oder Zweiersystem. Er kennt jeweils nur zwei Zustände: eine Leuchtdiode kann leuchten oder nicht, eine Spannung kann vorhanden sein oder nicht, durch ein Bauteil kann ein Strom fließen oder nicht. Physikalisch wäre es natürlich möglich, verschieden große Spannungen oder verschieden große Stromstärken zu unterscheiden; das wäre aber zu aufwendig und auch zu teuer. Wir begnügen uns also mit zwei Zuständen und bezeichnen sie mit 0 und 1. 0 heißt z.B.: die Leuchtdiode leuchtet nicht, 1 heißt: die Leuchtdiode leuchtet.

Wir können jetzt die Anzeige auf der LED-Reihe bei Spiel 4 (Zähler) leicht in eine Dualzahl übertragen: Die Anzeige ● ○ ● ● wird durch die Dualzahl 1011 beschrieben.

Verfolgen wir noch einmal den Zählvorgang von Anfang an: Wir beginnen bei 0000. Wird 1 addiert, so ergibt sich 0001. Jetzt sind schon alle möglichen Ziffern an der letzten Stelle verwendet worden. Bei der nächsten Addition von 1 folgt nach der 1 in der letzten Stelle wieder die 0; es wird zusätzlich ein Übertrag an die zweitletzte Stelle geliefert. Wir erhalten: 0010. Dann folgt: 0011. Wird jetzt wieder 1 addiert, so muß in der letzten Stelle eine 0 geschrieben werden. Zu der 1 in der vorletzten Stelle soll der Übertrag 1 von der letzten Stelle addiert werden. Auch hier schreiben wir eine 0 und bringen wieder einen Übertrag 1 in die nächste Stelle: 0100 usw.

Es gibt zwei verschiedene einstellige Dualzahlen (0 und 1), es gibt 4 ($= 2 \times 2$) verschiedene zweistellige (00, 01, 10 und 11), acht ($= 2 \times 2 \times 2$) verschiedene dreistellige (000, 001, ..., 111), 16 ($= 2 \times 2 \times 2 \times 2$) verschiedene vierstellige Dualzahlen (0000, 0001, ..., 1111).

Dezimalzahl	entsprechende Dualzahl
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

1011 bedeutet $1 \cdot 2^3 + 1 \cdot 2^1 + 1 \cdot 2^0 = 8 + 2 + 1 = 11$.
 Das Verfahren lässt sich fortsetzen: 100111 bedeutet
 $1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 32 + 4 + 2 + 1 = 39$
 $(2^5 = 32, 2^4 = 16, 2^3 = 8, 2^2 = 4, 2^1 = 2, 2^0 = 1;$
 vgl. Spiel 9: Zweier-Potenzen).
 Bei dem Zahlenraten (Vgl. 4. 1) hat Spieler B auf die sechs
 Fragen: Ist die gedachte Zahl auf Karte ①, ②, ..., ⑥ sechs-
 mal geantwortet. Bei unserer Beispielzahl 39 hat Spieler B
 der Reihe nach geantwortet: ja, nein, nein, ja, ja, ja. Damit
 hat er seine Zahl verraten. Spieler A braucht nur statt ja
 eine 1, statt nein eine 0 zu setzen. Dann kennt er die ge-
 suchte Zahl, allerdings als Dualzahl. Seine Aufgabe be-
 steht nur noch darin, diese Dualzahl in eine Dezimalzahl zu
 verwandeln.
 Unser Computer arbeitet im wesentlichen mit achtstelligen
 Dualzahlen: 0000 0000 bis 1111 1111. Es gibt $2^8 = 256$ ver-
 schiedene achtstellige Dualzahlen (vgl. wieder das Spiel 4:
 Zähler).

Dezimalzahl	entsprechende Dualzahl
0	0000 0000
15	0000 1111
16	0001 0000
32	0010 0000
64	0100 0000
128	1000 0000
240	1111 0000
248	1111 1000
252	1111 1100
253	1111 1101
254	1111 1110
255	1111 1111

Teilweise arbeitet unser Computer aber auch mit sechzehn-
 stelligen Dualzahlen. Die kleinste ist 0000 0000 0000 0000
 und entspricht der Dezimalzahl 0. Die größte ist
 1111 1111 1111 1111 und entspricht der Dezimalzahl 65 535.
 Es gibt 65 536 verschiedene sechzehnstelligen Dualzahlen
 (einschließlich der 0).

4.3 Hexadezimalzahlen

4.3.1 Einstelliger Hexadezimalzähler

Wir müssen uns noch mit einem weiteren Zahlensystem be-
 beschäftigen: dem Hexadezimal- oder Sechzehnersystem.
 Wir wollen zunächst ein Programm eintasten. Tastenfolge:
RS, **A↔D**, **A↔D**, bis die Anzeige **1 0 0 0 □ □ ? ?**
 erscheint. Dann weiter:

C, **4**,
ME +, **F**, **F**,
ME +, **C**, **D**, usw.

Wenn wir das nebenstehende Programm vollständig einge-
 geben haben, starten wir es: **A↔D**, **RUN**. Zunächst sieht
 es aus, als hätten wir einen Dezimalzähler programmiert.
 Aber nach 9 folgt nicht 10, sondern A. Dann erscheinen die
 weiteren Buchstaben B, C, D, E und F. In der Anzeige wer-
 den die Buchstaben B und D als kleine Buchstaben darge-
 stellt. Wir wissen, weshalb. Unsere Anzeige verfügt nur
 über begrenzte Möglichkeiten. Wir haben uns damit im Zu-
 sammenhang mit Spiel 4 beschäftigt.

Unser Zähler ist ein einstelliger Hexadezimalzähler. Er hat
 für eine Stelle 16 verschiedene Zeichen zur Verfügung: die
 zehn vom Dezimalsystem bekannten Ziffern 0 bis 9 und die
 sechs Buchstaben A bis F. Alle 16 Zeichen nennen wir die

ADRESSE	INHALT
1000	C4
1001	FF
1002	CD
1003	E0
1004	C4
1005	00
1006	CD
1007	D9
1008	95
1009	E0
100A	CD
100B	D8
100C	E4
100D	10
100E	6C
100F	F0
1010	13
1011	00
1012	00
1013	00
1014	00
1015	C4
1016	00
1017	11
1018	74
1019	EE

Ziffern des Hexadezimalsystems. Wir könnten auch ganz
 andere Symbole verwenden, aber diese haben sich für das
 Hexadezimalsystem eingebürgert.

4.3.2 Zusammenhang mit Dualzahlen

Wir ändern unser Programm geringfügig ab:

RS	A↔D	1	0	1	1	A↔D
		C	5			
	ME +	E	0			
	ME +	C	D			
	ME +	C	3			

Wir haben damit in die vier Speicher mit den Adressen 1011
 bis 1014 nacheinander die Inhalte C5, E0, CD und C3 einge-
 geschrieben. Das vollständige Programm ist hier noch einmal
 angegeben. Wir starten es mit **A↔D**, **RUN**. Wenn der
 Schalter S nach vorn gestellt ist, werden jetzt zusätzlich die
 rechten vier Leuchtdioden der LED-Reihe angesteuert.

ADRESSE	INHALT
1000	C4
1001	FF
1002	CD
1003	E0
1004	C4
1005	00
1006	CD
1007	D9
1008	95
1009	E0
100A	CD
100B	D8
100C	E4
100D	10
100E	6C
100F	F0
1010	13
1011	C5
1012	E0
1013	CD
1014	C3
1015	C4
1016	00
1017	11
1018	74
1019	EE

Es gibt 16 verschiedene vierstellige Dualzahlen, es gibt auch 16 verschiedene einstellige Hexadezimalzahlen. Sie entsprechen sich unmittelbar. Die Hexadezimalziffern werden gerade deswegen eingeführt, um vierstellige Dualzahlen kürzer bezeichnen zu können. Der Zusammenhang zwischen den verschiedenen Darstellungen der ersten sechzehn Zahlen in den drei behandelten Zahlensystemen ist in der folgenden Übersicht zusammengestellt.

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

4.3.3 Ein Rückwärtszähler

Im letzten Programm wurde vorwärts gezählt. Es wurde jeweils 1 addiert. Ebenso kann unser Computer rückwärts zählen. Er muß nur jeweils 1 subtrahieren. Gegenüber dem Programm von 4.3.2 müssen nur folgende Speicherinhalte geändert werden:
bei Adresse 1001: 10 statt FF,
bei Adresse 1008, 9D statt 95 und
bei Adresse 100D: FF statt 10.

ADRESSE	INHALT
1000	C4
1001	10
1002	CD
1003	E0
1004	C4
1005	00
1006	CD
1007	D9
1008	9D
1009	E0
100A	CD
100B	D8
100C	E4
100D	FF
100E	6C
100F	F0
1010	13
1011	C5
1012	E0
1013	CD
1014	C3
1015	C4
1016	00
1017	11
1018	74
1019	EE

Es wäre gut, sich die ersten sechzehn Dual- und Hexadezimalzahlen und ihren Zusammenhang untereinander und mit den Dezimalzahlen einzuprägen. Wir werden sie häufiger benutzen. Eine kleine Hilfe: Die Hexadezimalzahl **D** entspricht der Dezimalzahl 13 (dreizehn), die Hexadezimalzahl **F** entspricht der Dezimalzahl 15 (fünfzehn).

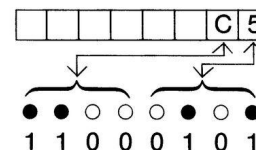
4.3.4 Mehrstellige Hexadezimalzahlen

Das Hexadezimalsystem ist eines von vielen möglichen Zahlensystemen. Natürlich läßt sich auch in diesem System weiter als bis F (dezimal 15) zählen. Wenn der Zeichenvorrat in einer Stelle aufgebraucht ist, wird wieder bei 0 angefangen und ein Übertrag für die nächste Stelle gebildet.

ADRESSE	INHALT
1000	C4
1001	FF
1002	CD
1003	E0
1004	C4
1005	00
1006	CD
1007	D9
1008	95
1009	E0
100A	CD
100B	D8
100C	13
100D	C5
100E	E0
100F	CD
1010	C3
1011	C4
1012	00
1013	11
1014	74
1015	F2

Unser Computer soll uns das in einem weiteren Programm zeigen. Der Schalter S muß nach vorn gestellt werden. Wir sehen: der Zähler zählt bis F. Dann erfolgt ein Übertrag in die nächste Stelle. An der letzten Stelle wird wieder mit 0 begonnen. Ebenso wird beim Weiterzählen nach 1F wieder ein Übertrag gebildet. Es folgt die Zahl 20. Entsprechend müßte nach der Hexadezimalzahl FF die dreistellige Hexadezimalzahl 100 folgen. Unser Computer beginnt aber wieder mit 0. Bei diesem Programm läuft der achtstellige Dualzähler mit. Wir erkennen ohne Schwierigkeiten den Zusammenhang zwischen den achtstelligen Dual- und den zweistelligen Hexadezimalzahlen: jede der beiden Gruppen von vier Dualziffern (die LED-Reihe ist aus diesem Grund in der Mitte unterteilt) entspricht einer Hexadezimalziffer.

Beispiel:



Es wird zweimal dieselbe Zahl dargestellt, nur in unterschiedlichen Zahlensystemen. Es gibt 256 verschiedene zweistellige Hexadezimalzahlen, von 00 bis FF. In der letzten Stelle stehen die Einer, in der vorderen Stelle stehen die Sechzehner. C5 besteht aus zwölf Sechzehnern und fünf Einern. Natürlich lassen sich in diesem Zahlensystem auch größere als zweistellige Zahlen bilden. Vor den Sechzehnern stehen die Zweihundertsechundfünfziger, davor dann die Viertausendsechundneunziger usw. Beispiel: die Hexadezimalzahl 2A07 besteht aus zwei Viertausendsechundneunzigern ($2 \cdot 16^3$), zehn Zweihundertsechundfünfzigern ($10 \cdot 16^2$), 0 Sechzehnern ($0 \cdot 16^1$) und sieben Einern ($7 \cdot 16^0$).

4.3.5 Und noch ein Rückwärtszähler

Wenn das letzte Programm noch eingespeichert ist, brauchen wir nur zu ändern:
bei Adresse 1001: 00 statt FF,
bei Adresse 1008: 9D statt 95.
Andernfalls müßten wir das folgende Programm neu eintasten.

ADRESSE	INHALT
1000	C4
1001	00
1002	CD
1003	E0
1004	C4
1005	00
1006	CD
1007	D9
1008	9D
1009	E0
100A	CD
100B	D8
100C	13
100D	C5
100E	E0
100F	CD
1010	C3
1011	C4
1012	00
1013	11
1014	74
1015	F2

Der Zähler beginnt jetzt bei FF bzw. bei 1111 1111 und zählt hexadezimal und dual rückwärts.

5. Umwandlungen zwischen Zahlensystemen

5.1 Vorbemerkungen

Wir kennen jetzt drei Zahlensysteme, die alle drei im weiteren für uns wichtig sein werden. Uns wird nach wie vor das Dezimalsystem das geläufigste sein. Der Computer rechnet und arbeitet ausschließlich im Dualsystem. Das Hexadezimalsystem dient uns lediglich als Hilfe bei der Eingabe über die Tasten und beim Ablesen an den Sieben-Segment-Anzeigen. Wenn wir z. B., wie im letzten Programm, in den Speicher mit der Adresse 1008 den Inhalt 9D bringen, dann übersetzt der Computer diese beiden Zahlen und schreibt in den Speicher mit der Adresse 0001 0000 0000 1000 den Inhalt 1001 1101. Er berechnet eventuell eine neue Zahl im Dualsystem, übersetzt das Ergebnis auf Wunsch so, daß er es uns überschaubarer anzeigen kann. Für uns sind weiterhin alle Speicheradressen und Speicherinhalte Hexadezimalzahlen, auch wenn der Computer für uns unsichtbar im Dualsystem arbeitet.

Da alle drei Zahlensysteme weiterhin benötigt werden, müssen wir eine Zahl aus dem einen System in das andere umformen können. Das ist das Anliegen dieses Kapitels.

Bevor wir mit dem Umwandeln beginnen, soll noch eine grundsätzliche Bemerkung gemacht werden. Man kann den Zahlen u. U. nicht ansehen, in welchem Zahlensystem sie dargestellt sein sollen. Was bedeutet 101? Diese Frage läßt sich nur beantworten, wenn wir zusätzlich das Zahlensystem angeben. Man schreibt die Grundzahl oder Basis (2, 10 oder 16) als Kennzeichen an die Zahl. Wenn keine Mißverständnisse möglich sind, werden wir auf diese Angabe verzichten.

101_2 ist eine Dualzahl, 101_{10} ist eine Dezimalzahl, 101_{16} ist eine Hexadezimalzahl. Jetzt sind Gleichungen wie $101_2 + 5_{10} = 257_{10}$ verständlich.

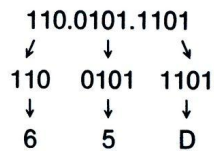
5.2 Umwandlungen zwischen dem Dual- und dem Hexadezimalsystem

Zur Umwandlung einer Dualzahl in eine Hexadezimalzahl (oder umgekehrt) genügt es, die Zuordnung zwischen den 16 vierstelligen Dualzahlen (0000 bis 1111) und den 16 einstelligen Hexadezimalzahlen (0 bis F) zu kennen; vgl. Tabelle im Abschnitt 4.3.2.

Bei einer gegebenen Dualzahl teilt man – von rechts beginnend – Gruppen von je vier Dualziffern ab und ersetzt jede Gruppe durch die entsprechende Hexadezimalziffer.

Beispiel:

gegebene Dualzahl:



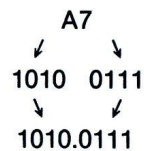
gesuchte Hexadezimalzahl:

Ergebnis: $11001011101_2 = 65D_{16}$.

Der umgekehrte Weg ist genauso einfach. Jetzt wird jede Hexadezimalziffer durch eine Gruppe von vier Dualziffern ersetzt.

Beispiel:

gegebene Hexadezimalzahl:



gesuchte Dualzahl:

Ergebnis: $A7_{16} = 10100111_2$.

5.3 Umwandlungen zwischen dem Dual- und dem Dezimalsystem

Wenn eine Dualzahl in eine Dezimalzahl umgewandelt werden soll, wird sie ausführlich mit Zweier-Potenzen geschrieben. Die Zweier-Potenzen werden im Dezimalsystem berechnet und mit der jeweiligen Ziffer (0 oder 1) multipliziert; schließlich werden die Summanden im Dezimalsystem addiert.

Beispiel:

$$101101_2 = 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$$
$$= 32_{10} + 8_{10} + 4_{10} + 1_{10} = 45_{10}$$

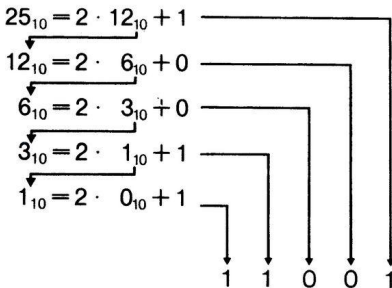
Ergebnis: $101101_2 = 45_{10}$.

Soll umgekehrt eine Dezimalzahl in eine Dualzahl umgewandelt werden, so zerlegt man die gegebene Dezimalzahl in Zweier-Potenzen.

Beispiel: Die Dezimalzahl 25 soll umgewandelt werden. 16 ist die größte Zweier-Potenz, die kleiner ist als 25. Den Rest $25 - 16 = 9$ kann man weiter zerlegen in 8 und 1.

$$25_{10} = 1 \cdot 16_{10} + 9_{10}$$
$$= 1 \cdot 16_{10} + 1 \cdot 8_{10} + 1_{10}$$
$$= 1 \cdot 16_{10} + 1 \cdot 8_{10} + 0 \cdot 4_{10} + 0 \cdot 2_{10} + 1 \cdot 1_{10}$$
$$= 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$$
$$= 11001_2$$

Diese Umwandlung lässt sich schematisieren: Man teilt die gegebene Dezimalzahl und dann das Ergebnis fortlaufend durch 2. Die Reste ergeben die Dualziffern von rechts nach links.



Ergebnis: $25_{10} = 11001_2$.

5.4 Umwandlungen zwischen dem Hexadezimal- und dem Dezimalsystem

Die Umwandlung einer Hexadezimalzahl in eine Dezimalzahl erfolgt nach dem gleichen Verfahren wie bei Dualzahlen: Die Hexadezimalzahl wird ausführlich mit Sechzehner-Potenzen geschrieben. Die Sechzehner-Potenzen werden im Dezimalsystem berechnet und mit jeweiligen Ziffern (0 bis F bzw. 0 bis 15) multipliziert, schließlich werden die Summanden im Dezimalsystem addiert.

Beispiel: $C5_{16} = C \cdot 16^1 + 5 \cdot 16^0$

$$= 12_{10} \cdot 16_{10} + 5_{10} \cdot 1_{10}$$
$$= 197_{10}$$

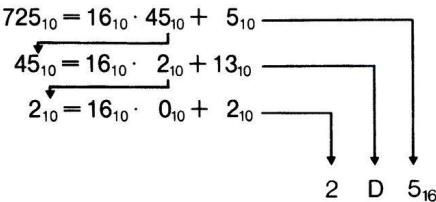
Ergebnis: $C5_{16} = 197_{10}$

Bei der umgekehrten Umwandlung lässt sich eine gegebene Dezimalzahl in Sechzehnerpotenzen zerlegen.

Beispiel: $705_{10} = 2 \cdot 256_{10} + 213_{10}$

$$= 2 \cdot 256_{10} + 13_{10} \cdot 16_{10} + 5_{10}$$
$$= 2 \cdot 16^2 + 13 \cdot 16^1 + 5 \cdot 16^0$$
$$= 2 D 5_{16}$$

Diese Umwandlung lässt sich auch nach dem Schema durchführen, wie wir es im vorangehenden Abschnitt für Dualzahlen kennengelernt haben (vgl. 5.3).



Ergebnis: $725_{10} = 2 D 5_{16}$

5.5 Aufgaben zu Kapitel 5

Hier tauchen zum erstenmal Aufgaben auf. Sie sind dazu gedacht, die behandelten Umwandlungsverfahren einzuüben. Wer Schwierigkeiten hat, findet die Lösungen am Ende dieses Anleitungsbuches. Zum größten Teil lassen sich diese Aufgaben auch mit Hilfe der Computerprogramme in Abschnitt 5.6 lösen. Aber man sollte es doch erst einmal allein versuchen.

- Die folgenden Dualzahlen sollen in Dezimalzahlen und in Hexadezimalzahlen umgewandelt werden:
 - 10111_2
 - 10111101_2
 - 1111101000_2
- Die folgenden Dezimalzahlen sollen in Dualzahlen und in Hexadezimalzahlen umgewandelt werden:
 - 78_{10}
 - 100_{10}
 - 9999_{10}
- Die folgenden Hexadezimalzahlen sollen in Dualzahlen und in Dezimalzahlen umgewandelt werden:
 - $A9_{16}$
 - 100_{16}
 - $AF FE_{16}$

5.6. Drei Umwandlungsprogramme

5.6.1 Von dezimal nach dual

ADRESSE	INHALT
1000	17
1001	85
1002	08
1003	18
1004	01
1005	7C
1006	F9
1007	01
1008	CD
1009	C3
100A	10
100B	00
100C	00
100D	74
100E	F1

Wir tasten zunächst das angegebene Programm ein und starten es. Schalter S nach vorn. Das Programm gestattet es, eine Dezimalzahl bis 255 in die entsprechende Dualzahl umzuwandeln.

In der Anzeige erscheint eine 0. Wir können jetzt eine Dezimalzahl eingeben, indem wir nacheinander die entsprechenden Zifferntasten (0 bis 9) drücken. Betätigen wir dann die Taste RUN, dann erscheint zusätzlich zur eingegebenen Dezimalzahl die entsprechende Dualzahl in der LED-Reihe.

Bei erneuter Betätigung der Taste RUN (oder einer beliebigen Zifferntaste) wird gelöscht. Es erscheint die 0, das Spiel kann von vorn beginnen.

Das Programm ist nicht gegen jede falsche Bedienung geschützt. Bei der Eingabe von vierstelligen Zahlen oder bei der Eingabe der Hexadezimalziffern (A bis F) kann Unverständliches passieren. Wird allerdings eine dreistellige Dezimalzahl größer als 255 eingegeben, dann merkt der Computer das und er verweigert das Umwandeln.

5.6.2 Von dezimal nach hexadezimal

ADRESSE	INHALT
1000	17
1001	85
1002	08
1003	18
1004	8D
1005	08
1006	14
1007	C4
1008	03
1009	CD
100A	E0
100B	26
100C	C8
100D	FF
100E	C6
100F	FF
1010	E4
1011	7E
1012	7C
1013	08
1014	C4
1015	00
1016	CA
1017	00
1018	9D
1019	E0
101A	7C
101B	F2
101C	10
101D	00
101E	00
101F	74
1020	DF

Schalter S nach hinten. Das Programm ist etwas länger als das vorangehende. Es leistet dafür auch mehr.

Nach Eingabe und Start des Programms erscheint wieder die 0. Die Eingabe einer beliebigen vierstelligen Dezimalzahl kann beginnen (bitte nicht die Ziffern A bis F verwenden).

Bei Betätigung der RUN-Taste erscheint zusätzlich zur eingegebenen Dezimalzahl links in der Anzeige die entsprechende Hexadezimalzahl.

Bei erneuter Betätigung der Taste RUN (oder einer beliebigen Zifferntaste) erscheint wieder die 0. Eine neue Dezimalzahl kann eingegeben werden.

5.6.3 Von hexadezimal nach dezimal

ADRESSE	INHALT
1000	17
1001	85
1002	08
1003	1C
1004	8D
1005	08
1006	14
1007	C4
1008	03
1009	CD
100A	E0
100B	26
100C	C8
100D	FF
100E	C6
100F	FF
1010	E4
1011	7E
1012	7C
1013	08
1014	C4
1015	00
1016	CA
1017	00
1018	9D
1019	E0
101A	7C
101B	F2
101C	10
101D	00
101E	00
101F	74
1020	DF

Schalter S nach hinten. Dieses Programm unterscheidet sich vom vorangehenden nur dadurch, daß im Speicher mit der Adresse 1003 hier 1C statt 1B steht

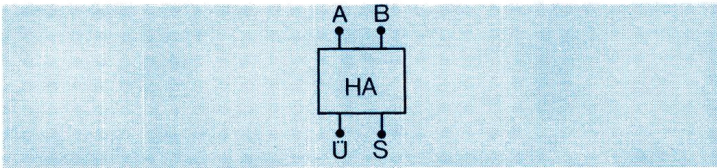
Diese kleine Programmänderung bewirkt, daß unser Computer jetzt eine eingegebene Hexadezimalzahl in die entsprechende Dezimalzahl umwandelt und zusätzlich anzeigt.

Ein kleines Problem: Die Umwandlung ist nur möglich, wenn die Dezimalzahl noch vierstellig ist. Die eingegebene Hexadezimalzahl darf also höchstens 270F sein. Bei einer größeren Hexadezimalzahl meldet sich unser Computer mit Error (engl. Irrtum, Fehler). Mit A·D und RUN oder mit RS und RUN kann das Programm neu gestartet werden.

6. Vom Halbaddierer zum Addierwerk

6.1 Der Halbaddierer

Ein Halbaddierer ist nur ein halber Addierer, aber er ist trotzdem der grundlegende Baustein für alle Rechnungen, die unser Computer durchführen kann. Ein Halbaddierer ist eine elektronische Schaltung mit zwei Eingängen (A und B) und zwei Ausgängen (Ü und S).



Das Innere dieser Schaltung ist so aufgebaut, daß der Halbaddierer zwei einstellige Dualzahlen A und B addieren kann.

Für A und B sind jeweils nur zwei Werte zugelassen. In der Elektronik erlaubt man als Eingangsspannung an den Eingängen A und B entweder eine niedrige Spannung (z. B. zwischen 0V und 1V) oder eine hohe Spannung (z. B. zwischen 3V und 5V). Die beiden erlaubten Eingangsspannungen werden dann mit L (engl. low – niedrig) und mit H (engl. high – hoch) bezeichnet. Da wir hier aber addieren wollen, werden wir statt L und H gleich 0 und 1 schreiben. Für die Ausgänge unseres Halbaddierers gilt Entsprechendes. Die Ausgänge liefern niedrige oder hohe Spannungen, L- oder H- Signale, so daß eine Leuchtdiode dunkel bleiben oder leuchten würde. Wir interpretieren die Ausgangssignale wieder als Dualzahlen: 0 oder 1.

Da A und B jeweils nur zwei Werte annehmen können (sie können jeweils nur 0 oder 1 sein), hat der Halbaddierer nur vier verschiedene Aufgaben zu lösen. Wir kennen natürlich schon die vier Ergebnisse für die Addition, bzw. die Werte für die Ausgangssignale Ü (Übertrag) und S (Summe):

A = 0, B = 0 / 0 + 0 = 0 / Ü = 0, S = 0
A = 0, B = 1 / 0 + 1 = 1 / Ü = 0, S = 1
A = 1, B = 0 / 1 + 0 = 1 / Ü = 0, S = 1
A = 1, B = 1 / 1 + 1 = 10 / Ü = 1, S = 0

Diese Zusammenhänge lassen sich kürzer in einer Verknüpfungstabelle darstellen:

Verknüpfungstabelle für den Halbaddierer			
A	B	Ü	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Jede Zeile dieser Tabelle beschreibt eine der vier möglichen Additionsaufgaben.

Die folgende Additionstabelle gibt auch die vier möglichen Rechnungsergebnisse wieder:

Additionstabelle für zwei einstellige Dualzahlen

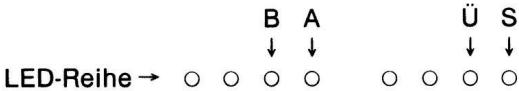
		B	
		0	1
+	A	0	1
		1	10

Wir wollen uns die vier möglichen Verknüpfungen, die ein Halbaddierer ausführen kann, mit Hilfe des Computers ansehen. Diese Aufgabe bedeutet eigentlich für den Computer eine Unterforderung. Er führt normalerweise sehr viel anspruchsvollere Rechnungen durch. Daher wird das Programm länger, als man bei diesem einfachen Problem erwarten würde.

ADRESSE	INHALT
1000	12
1001	39
1002	00
1003	06
1004	04
1005	10
1006	01
1007	06
1008	3C
1009	04
100A	10
100B	70
100C	3C
100D	3C
100E	3C
100F	3C
1010	01
1011	06
1012	70
1013	CD
1014	C3
1015	C4
1016	08
1017	11
1018	74
1019	E6

Schalter S nach vorn. Nach dem Eintasten wird das Programm gestartet. Die Anzeige erlischt.

Die Eingangssignale A und B können wir mit den Tasten **SA** und **SB** eingeben. In der LED-Reihe werden die beiden Eingangssignale A und B und auch die beiden Ausgangssignale Ü und S angezeigt.



0 ≙ Taste nicht betätigt, Leuchtdiode dunkel,
1 ≙ Taste betätigt, Leuchtdiode leuchtet.

Wir werden schnell die vier Möglichkeiten durchprobiert haben.

6.2 Der Volladdierer

Sollen zwei mehrstellige Dualzahlen addiert werden, so ist ein Halbaddierer schon überfordert. Ein Beispiel soll das zeigen.

Wir wollen die beiden Dualzahlen $111_2 (= 7_{10})$ und $101_2 (= 5_{10})$ addieren. Für uns ist das kein Problem. Wir gehen dabei so vor, wie wir es von der Addition im Dezimalsystem gewohnt sind. Wir schreiben die beiden Zahlen untereinander und addieren stellenweise, von rechts nach links.

$$\begin{array}{r} 1\ 1\ 1 \\ 1\ 0\ 1 \\ \hline \end{array} +$$

Wir müssen beachten, daß wir im Dualsystem nur die beiden Ziffern 0 und 1 zur Verfügung haben. In der letzten Stelle addieren wir 1 und 1, das ergibt als Summe eine 0 und einen Übertrag in die nächste Stelle.

$$\begin{array}{r} 1\ 1\ 1 \\ 1\ 0\ 1 \\ \hline 1\ 0\ 1 \\ \hline \end{array} +$$

In der zweiten Stelle haben wir jetzt die drei Ziffern 1, 0 und 1 zu addieren. Das ergibt wieder 0 und einen Übertrag.

$$\begin{array}{r} 1\ 1\ 1 \\ 1\ 0\ 1 \\ \hline 1\ 1\ 0\ 1 \\ \hline \end{array} +$$

In der ersten Stelle müssen jetzt 1, 1 und 1 addiert werden. Das ergibt 1 und erneut einen Übertrag.

$$\begin{array}{r} 1\ 1\ 1 \\ 1\ 0\ 1 \\ \hline 1\ 1\ 1\ 0\ 1 \\ \hline \end{array} +$$

Das Ergebnis unserer Additionsaufgabe ist $1100_2 (= 12_{10})$.

Der Halbaddierer, den wir im letzten Abschnitt behandelt haben, ist nur in der Lage, die Addition in der letzten Stelle auszuführen. Er berechnet von zwei gegebenen einstelligen Dualzahlen die Summe und gegebenenfalls einen Übertrag. Für die Addition in der zweiten und jeder weiteren Stelle benötigen wir einen Addierer, der neben den beiden gegebenen Dualziffern an dieser Stelle noch den Übertrag aus der vorangehenden Stelle berücksichtigt. Der Addierer hat also drei einstellige Dualzahlen zu addieren. Eine elektronische Schaltung, die diese Aufgabe erfüllt, nennt man einen Volladdierer. Man kann sich einen Volladdierer im wesentlichen aus zwei Halbaddierern zusammengesetzt denken. Der erste Halbaddierer addiert die beiden gegebenen Ziffern, der zweite addiert zur Summe den Übertrag aus der vorangegangenen Addition. Kurz: Ein Volladdierer ist eine Schaltung mit drei Eingängen (A, B, C) und zwei Ausgängen (Ü, S). Er berechnet für acht verschiedene Fälle zu drei gegebenen einstelligen Dualzahlen einen Übertrag und eine Summe.

A	B	C	Ü	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Auch das soll uns der Computer wieder vorführen.

ADRESSE	INHALT
1000	26
1001	00
1002	FD
1003	C4
1004	00
1005	CD
1006	E0
1007	84
1008	04
1009	01
100A	8D
100B	E1
100C	C5
100D	E2
100E	0E
100F	CD
1010	E2
1011	CA
1012	00
1013	C2
1014	00
1015	E4
1016	FF
1017	6C
1018	02
1019	95
101A	E0
101B	9D
101C	E1
101D	7C
101E	ED
101F	C5
1020	E0
1021	0E
1022	07
1023	74
1024	DE

Nach dem Start des Programms erlischt wieder die Anzeige. Die Tasten **1**, **2** und **3** dienen zur Eingabe der drei Eingangssignale A, B und C. Wer will, kann stattdessen auch die drei Tasten **A**, **B** und **C** verwenden. Der Zustand der Eingangssignale wird bei diesem Programm nicht zusätzlich angezeigt:

Taste nicht betätigt $\triangleq 0$,
Taste betätigt $\triangleq 1$.

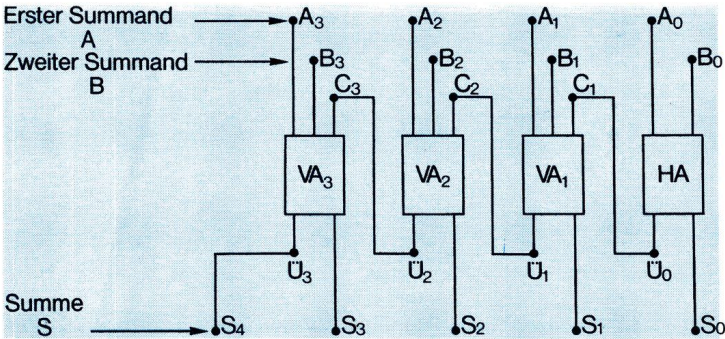
Die Summe wird an der grünen, der Übertrag wird an der gelben Leuchtdiode angezeigt:

Leuchtdiode dunkel $\triangleq 0$,
Leuchtdiode hell $\triangleq 1$.

Die acht möglichen Rechnungen, die der Volladdierer beherrscht, sind schnell durchprobiert.

6.3 Ein vierstelliges Addierwerk

Nach den beiden letzten Abschnitten lässt sich mit Hilfe eines Halbaddierers und dreier Volladdierer ein vierstelliges duales Addierwerk aufbauen.



Die Dualzahlen A und B haben jeweils vier Ziffern: $A = A_3 A_2 A_1 A_0$, $B = B_3 B_2 B_1 B_0$. Der Halbaddierer addiert die beiden letzten Ziffern A_0 und B_0 . Die Summe S_0 ist schon die letzte Stelle des Ergebnisses. Der Übertrag U_0 ist neben A_1 und B_1 das dritte Eingangssignal C_1 des Volladdierers VA_1 . Dieser Volladdierer ermittelt die nächste Stelle S_1 des Ergebnisses und gleichzeitig den Übertrag U_1 , der als Eingangssignal C_2 vom nächsten Volladdierer benötigt wird. Entsprechend geht es weiter. Die Abbildung zeigt, daß die Summe bei der Addition von zwei vierstelligen Dualzahlen fünfstellig werden kann.

Wir wollen uns noch einige Beispiele ansehen (vgl. auch das Beispiel in Abschnitt 6.2).

Beispiel 1: $A = 1101$; $B = 1100$

(A und B bezeichnen hier keine Hexadezimalziffern, sie bezeichnen die beiden Summanden!)

Der Halbaddierer HA addiert die beiden letzten Stellen. Es ergibt sich die Summe 1 und der Übertrag 0.

$$\begin{array}{r} 1 \ 1 \ 0 \ 1 \\ 1 \ 1 \ 0 \ 0 \\ \hline 0 \leftarrow \\ 1 \end{array} +$$

Volladdierer VA_1 rechnet $0 + 0 + 0 = 0$.

$$\begin{array}{r} 1 \ 1 \ 0 \ 1 \\ 1 \ 1 \ 0 \ 0 \\ 0 \leftarrow \\ \hline 0 \ 1 \end{array} +$$

Volladdierer VA_2 berechnet als Summe 0 und als Übertrag 1.

$$\begin{array}{r} 1 \ 1 \ 0 \ 1 \\ 1 \ 1 \ 0 \ 0 \\ 1 \leftarrow \\ \hline 0 \ 0 \ 1 \end{array} +$$

Volladdierer VA_3 ermittelt schließlich die beiden fehlenden Stellen des Ergebnisses.

$$\begin{array}{r} 1 \ 1 \ 0 \ 1 \\ 1 \ 1 \ 0 \ 0 \\ 1 \leftarrow \\ \hline 1 \ 1 \ 0 \ 0 \end{array} +$$

Ergebnis: $1101 + 1100 = 11001$

Beispiel 2: $A = 1010$; $B = 11$

Wie in Beispiel 1 lässt sich schrittweise die Summe ermitteln:

$$\begin{array}{r} 1 \ 0 \ 1 \ 0 \\ \ 1 \ 1 \\ \hline 1 \ 1 \ 0 \ 1 \end{array} +$$

Der Übertrag braucht nur dort eingetragen zu werden, wo er 1 beträgt.

Ergebnis: $1010 + 11 = 1101$

Beispiel 3: $A = 1001$; $B = 111$

$$\begin{array}{r} 1 \ 0 \ 0 \ 1 \\ \ 1 \ 1 \ 1 \\ 1 \leftarrow 1 \leftarrow 1 \leftarrow 1 \leftarrow \\ \hline 1 \ 0 \ 0 \ 0 \ 0 \end{array} +$$

Ergebnis: $1001 + 111 = 10000$

Bei der Addition von vierstelligen Dualzahlen lässt sich natürlich auch der Weg über das Dezimalsystem beschreiten. In Beispiel 1 waren die Dualzahlen 1101 und 1100 zu addieren. Man übersetzt ins Dezimalsystem: $1101_2 = 13_{10}$; $1100_2 = 12_{10}$. Für die Summe ergibt sich im Dezimalsystem $13_{10} + 12_{10} = 25_{10}$. Diese Ergebnis ist dann noch in das Dualsystem zu übertragen: $25_{10} = 16_{10} + 8_{10} + 1_{10} = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 11001_2$

Natürlich kann unser Computer solche Additionsaufgaben lösen. Wir werden in Abschnitt 6.5 ein Programm benutzen, bei dem außer der Addition von vierstelligen Dualzahlen noch ein weiterer Gesichtspunkt betrachtet wird.

6.4 Aufgaben zu Kapitel 6

1. Die folgenden Summen sollen – wie in den Beispielen im Abschnitt 6.3 beschrieben – bestimmt werden.

1.1 $1010 + 0101$

1.2 $1111 + 0011$

1.3 $1111 + 1111$

2.1 bis 2.3. Die drei Summen aus der ersten Aufgabe sollen auf dem Umweg über das Dezimalsystem bestimmt werden. Die beiden Summanden werden dazu in Dezimalzahlen verwandelt. Die im Dezimalsystem berechnete Summe ist dann wieder in eine Dualzahl umzuformen.

3. Wie viele Volladdierer würde man für ein achtstelliges Addierwerk benötigen?

4. Welches ist die größte Summe, die man bei der Addition zweier achtstelliger Dualzahlen erhalten kann? Die Antwort soll im Dual-, im Dezimal- und im Hexadezimalsystem angegeben werden.

6.5 Die Addition von einstelligen Hexadezimalzahlen

Hätten wir ein Addierwerk, wie es im Abschnitt 6.3 beschrieben wurde, so könnten wir mit ihm einstellige Hexadezimalzahlen addieren, und zwar etwa in der Art, wie es unser Computer auch macht. Wir müßten zwei gegebene Zahlen, z. B. D₁₆ und C₁₆ in Dualzahlen verwandeln: D₁₆ = 1101₂ und C₁₆ = 1100₂. Dann könnte das Addierwerk im Dualsystem die Summe bilden:

1101₂ + 1100₂ = 11001₂ (vgl. Beispiel 1, Kap. 6.3)

Das Ergebnis würden wir wieder in eine Hexadezimalzahl verwandeln: 11001₂ = 19₁₆. Ergebnis: D₁₆ + C₁₆ = 19₁₆. In Beispiel 2 würde sich ergeben: A₁₆ + 3₁₆ = D₁₆, in Beispiel 3 würde sich ergeben: 9₁₆ + 7₁₆ = 10₁₆. Wenn wir ohne ein solches Addierwerk die Summe zweier einstelliger Hexadezimalzahlen bestimmen wollen, werden wir uns wohl wieder am Dezimalsystem orientieren:

D₁₆ + C₁₆ = 13₁₀ + 12₁₀
= 25₁₀
= 16₁₀ + 9₁₀
= 19₁₆

A₁₆ + 3₁₆ = 10₁₀ + 3₁₀
= 13₁₀
= D₁₆

9₁₆ + 7₁₆ = 9₁₀ + 7₁₀
= 16₁₀
= 16₁₀ + 0₁₀
= 10₁₆

ADRESSE	INHALT
1000	16
1001	C5
1002	08
1003	CD
1004	C3
1005	10
1006	00
1007	00
1008	C5
1009	08
100A	48
100B	D4
100C	0F
100D	01
100E	3C
100F	3C
1010	3C
1011	3C
1012	70
1013	CD
1014	08
1015	13
1016	C5
1017	08
1018	CD
1019	C3
101A	10
101B	00
101C	00
101D	74
101E	E1

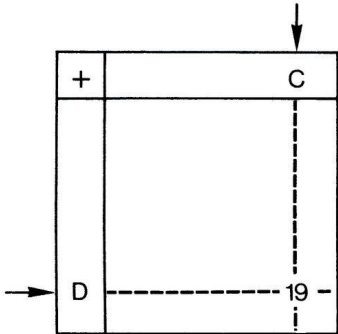
Wir wollen uns jetzt das schon angekündigte Programm ansehen. Es zeigt uns sowohl die Addition von zwei vierstelligen Dualzahlen, als auch die Addition von zwei einstelligen Hexadezimalzahlen.

Schalter S nach vorn. Nach Programmstart erscheint in der Anzeige eine 0. Der Computer wartet auf eine zweistellige Zahleneingabe (hexadezimal). Wir tasten z. B. **D**, **C**. DC erscheint in der Anzeige. Diese beiden Ziffern werden hier als zwei einstellige Hexadezimalzahlen aufgefaßt. Nach Betätigung der Taste **RUN** erscheinen für beide Zahlen die entsprechenden Dualdarstellungen in der LED-Reihe, für D links: 1101, für C rechts: 1100. Das könnte jetzt eine Aufgabenstellung für uns sein: Was ist 1101 + 1100? Was ist D + C?

Wenn wir die beiden Ergebnisse bestimmt haben, können wir den Computer kontrollieren lassen. Wir betätigen wieder die **RUN**-Taste. Die gewünschte Summe wird sowohl dual (11001) als auch hexadezimal (19) angezeigt. Nach erneuter Betätigung der Taste **RUN** kann eine neue Eingabe erfolgen. Wir können noch die beiden anderen Beispiele aus dem letzten Abschnitt überprüfen. Wir können beliebige andere Additionsaufgaben berechnen lassen. Es gibt 256 Möglichkeiten, die in der folgenden Additionstabelle zusammengestellt sind.

+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

Die Summe 19 für die beiden Summanden D und C läßt sich in der Zeile hinter D und in der Spalte unter C ablesen.



7. Die ersten Befehle: Addition

7.1 „Addiere dazu 3F!“

In Abschnitt 6.3 haben wir uns ein Addierwerk überlegt, das vierstellige Dualzahlen addieren kann. Der Mikroprozessor in unserem Computer arbeitet meistens mit achtstelligen, teilweise sogar mit sechzehnstelligen Dualzahlen. Er besitzt also zumindest ein achtstelliges Addierwerk.

Wir wissen schon: Unser Computer erleichtert uns unsere Arbeit. Er erlaubt uns, Zahlen (und auch andere Speicherinhalte) hexadezimal einzugeben. Er übersetzt ins Dualsystem, rechnet in diesem System, übersetzt zurück und erleichtert so wieder das Ablesen des Ergebnisses.

Wir können daher auch sagen: In unserem Computer befindet sich ein Addierwerk für zweistellige Hexadezimalzahlen. Nur, wie sagen wir ihm, daß er addieren soll, wie sagen wir ihm, welche beiden Zahlen er addieren soll, und wo finden wir hinterher das Ergebnis? Wir stellen uns die Aufgabe, dem Computer klarzumachen, daß er die beiden Hexadezimalzahlen 19 und 3F addieren soll. Angenommen, er kennt schon die erste Zahl 19. Dann müssen wir ihm noch sagen: addiere dazu 3F. Es ist schnell einsehbar, daß dieser Auftrag die Anweisung „addiere“ und den zweiten Summanden 3F enthalten muß. Es ist auch einsehbar, daß dieser gesamte Auftrag oder Befehl mit mehr als zwei Hexadezimalziffern codiert werden muß; zwei sind ja schon für die Zahl 3F erforderlich. Der vollständige Operationscode für diesen Befehl „addiere dazu 3F“ heißt F4 3F.

Der Befehl „addiere dazu 3F“ beschreibt die Operation, die ausgeführt werden soll. Der Operationscode ist eine für den Mikroprozessor verständliche Übersetzung dieses Befehls.

Das zentrale Register im Mikroprozessor, der wichtigste Speicher, Arbeits- oder Zwischenspeicher, heißt Akkumulator (engl. accumulator, Abkürzung: AC, oder noch kürzer: A). Wenn der Mikroprozessor seinen Befehl F4 3F liest, interpretiert er ihn viel exakter, als wir ihn mit „addiere dazu 3F“ umschrieben haben. Für ihn heißt der Befehl: Addiere zum Inhalt des Akkumulators die Zahl 3F und bringe das Ergebnis wieder in den Akkumulator. Kurzbeschreibung: (A): = (A) + 3F. A ist die Bezeichnung für den Akkumulator, (A) soll den Inhalt des Akkumulators bezeichnen. Der Inhalt vom Akku und 3F werden addiert, das Ergebnis wird zum neuen Inhalt des Akkus. Der neue Inhalt des Akkus wird gleich der Summe von altem Akku-Inhalt und 3F gesetzt.

Wenn unser Computer die beiden Zahlen 19 und 3F addieren soll, müssen wir zunächst dafür sorgen, daß die Zahl 19 im Akku steht. Jetzt kann der Befehl „addiere dazu 3F“ ausgeführt werden. Das Ergebnis finden wir anschließend im Akku.

Bevor wir den Computer endlich die Rechnung durchführen lassen, noch zwei Bemerkungen:

1. Was macht der Computer, wenn er den Additionsbefehl bearbeitet hat? – Er erwartet einen weiteren Befehl. Wir müssen ihm also sagen, daß wir von ihm gar nicht mehr erwarten. Das geschieht mit dem Operationscode 1E. 1E werden wir immer dann in unser Programm schreiben, wenn wir an dieser Stelle testen wollen, was bis hierher passiert ist.
2. Die Funktionstaste **CPU** gestattet es, immer, wenn unser Computer nicht mit einem Programm beschäftigt ist, die verschiedenen Registerinhalte in der Zentraleinheit zu überprüfen und eventuell zu ändern. Die Zentraleinheit ist der „Kopf“ unseres Computers, sie befindet sich im Mikroprozessor-IC. Zentraleinheit heißt auf englisch

central processing unit (Abkürzung: CPU). Der Akkumulator ist das am häufigsten benutzte Register in der CPU.

So, jetzt können wir endlich beginnen! Wir tasten zunächst unser kleines Programm ein:

ADRESSE	INHALT
1000	F4
1001	3F
1002	1E

Bei der Anzeige

1	0	0	0				
---	---	---	---	--	--	--	--

 tasten wir:

F, **4**,
ME+, **3**, **F**,
ME+, **1**, **E**.

In den beiden Speichern mit den Adressen 1000 und 1001 steht der Additionsbefehl, im Speicher mit der Adresse 1002 findet der Computer den Auftrag, seine Arbeit zunächst zu beenden. Jetzt ist das Programm eingegeben. Bevor wir es starten können, muß die Zahl 19 in den Akku gebracht werden.

Taste: **CPU**, Anzeige:

C	P	U	?				
---	---	---	---	--	--	--	--

Taste: **6**, Anzeige:

		?	?			A
--	--	---	---	--	--	---

Mit **CPU**, **6** wird der Akkuinhalt aufgerufen. Im Akku steht ein nicht vorhersehbarer Inhalt, daher die beiden Fragezeichen. Es ist aber sicher eine zweistellige Hexadezimalzahl.

Taste: **1**, Anzeige:

		0	1			A
--	--	---	---	--	--	---

Taste: **9**, Anzeige:

		1	9			A
--	--	---	---	--	--	---

Taste: **CPU**, Anzeige:

C	P	U	?				
---	---	---	---	--	--	--	--

Damit ist die Eingabe der Zahl 19 in den Akku beendet. Wichtiger Hinweis: die Eingabe in ein Register der CPU muß unbedingt mit der Taste **CPU** abgeschlossen werden, nur dann wird der alte Inhalt überschrieben. Wir starten jetzt das Programm: **A↔D**, **RUN**. Der Computer meldet sich nach Ausführung des Programms mit der Anzeige

1	0	0	2				
---	---	---	---	--	--	--	--

Wir kontrollieren jetzt den Akku-Inhalt: **CPU**, **6** und finden

		5	8			A
--	--	---	---	--	--	---

Der Akkuinhalt ist, wie erwartet, 58: $19_{16} + 3F_{16} = 58_{16}$.

Dieses Additionsverfahren ist sicher sehr umständlich. Es geht aber auch einfacher, wie wir bald sehen werden. Trotzdem sollte man nach dem hier beschriebenen Verfahren noch einige Additionen durchführen. Wir brauchen Übung im Umgang mit dem Computer und seinen Tasten!

Weitere Beispiele: $0A_{16} + 0A_{16} = 14_{16}$
 $17_{16} + 04_{16} = 1B_{16}$
 $77_{16} + 88_{16} = FF_{16}$

Einer der beiden Summanden muß in den Akku gebracht werden, der andere wird in den Speicher mit der Adresse 1001 als Teil des Additionsbefehls geschrieben. Werden nach diesem Verfahren zwei Zahlen addiert, deren Summe größer als FF_{16} wird, so können wir anschließend im Akku nur die beiden letzten Ziffern der Summe finden. Die drittletzte Ziffer (der Übertrag 1) steht in einem anderen Register. Wir kommen darauf zurück.

7.2 16 Bit = 2 Byte

Der Operationscode für den im letzten Abschnitt besprochenen Befehl „Addiere zum Akku-Inhalt die Zahl 3F und bringe die Summe wieder in den Akku“ war F4 3F. Für uns ist er eine vierstellige Hexadezimal-, für den Mikroprozessor ist er eine sechzehnstelligen Dualzahl:

$F43F_{16} = 1111\ 0100\ 0011\ 1111_2$. Dieser Befehlscode enthält für den Mikroprozessor vier Gruppen von Informationen:

1	111 0	100	0011 1111
1.	2.	3.	4.

Man bezeichnet eine einzelne Dualziffer in diesem Zusammenhang auch als ein Bit (Abkürzung der englischen Bezeichnung **binary digit** = binäres oder duales Zeichen). Ein Bit wird auch als Informationseinheit bezeichnet. Man erhält die Information 1 Bit, wenn man erfährt, welcher von zwei möglichen, gleich wahrscheinlichen Werten zutrifft. In unserem Spiel „Zahlenraten“ (vgl. 4.1) erhält Spieler A auf die Frage „Ist die gedachte Zahl auf dieser Karte?“ vom Spieler B die Information 1 Bit, wenn Spieler B mit „ja“ oder „nein“ antwortet. Bei einer Information von 6 Bit weiß Spieler A die gedachte Zahl.

Unser Mikroprozessor benötigt auch eine Information, um zu wissen, welchen Befehl er ausführen soll. Wir hatten oben schon diese Gesamtinformation in vier Gruppen unterteilt.

1. Wenn der Mikroprozessor an der ersten Stelle von vorn im Operationscode eine 1 findet, weiß er, daß ein Befehl folgt, bei dem es acht verschiedene Adressierungsarten gibt. Das klingt jetzt noch sehr rätselhaft, wird aber im weiteren verständlich werden. Es ist aber jetzt schon klar, daß der Mikroprozessor z. B. beim Additionsbefehl wissen muß, wo und wie er die Zahl finden soll, die er zum Akkuinhalt addieren soll. Daß der zweite Summand – wie oben besprochen – Teil des Befehls ist, ist eine von acht Möglichkeiten.
2. Eine Information von 4 Bit ist erforderlich, damit er weiß, daß zum Akkuinhalt addiert werden soll.
3. Wenn es acht verschiedene Adressierungsarten gibt, muß in 3 Bit eindeutig gesagt werden, welche gewünscht ist.
4. Die letzten acht Bit dienen in dem besprochenen Befehl dazu, den zweiten Summanden mitzuteilen. Bei den anderen Adressierungsarten wird dem Mikroprozessor in diesen acht Bit gesagt, wo er den zweiten Summanden findet.

Diese Andeutungen sollen hier genügen. Es sollte hier zunächst nur deutlich werden, daß der bisher besprochene Befehl ein 16-Bit-Befehl ist. Man faßt je acht Bit zu einem Byte zusammen:

16 Bit = 2 Byte.

Unser Befehl ist also ein 2-Byte-Befehl. Da in einem Speicher nur 1 Byte stehen kann, muß unser Befehl in zwei aufeinander folgenden Speichern untergebracht werden. Bei unserem Mikroprozessor gibt es 1-, 2- und 3-Byte-Befehle.

1 Byte = 8 Bit,

3 Byte = 24 Bit.

Wir werden ab sofort bei der Auflistung unserer Programme die Befehle nicht mehr auseinanderreißen. Wir schreiben jetzt die vollständigen Befehle – den vollständigen Operationscode – in eine Zeile und geben jeweils nur die Adresse für das erste Byte an. Es versteht sich von selbst, daß das zweite Byte dieses Befehls dann im Speicher mit der nächsten Adresse steht.

Unser Programm aus 7.1 wird dann folgendermaßen geschrieben:

ADRESSE	OP. CODE
1000	F4 3F
1002	1E

Das Byte 3F steht im Speicher mit der Adresse 1001.

7.3 Wir erweitern das Programm

Wir haben in Abschnitt 7.1 die beiden Hexadezimalzahlen 19 und 3F addiert. Etwas mühsam war die Eingabe des ersten Summanden in den Akkumulator und das Feststellen des Akkuinhalts nach Beendigung der Addition. Wir wollen das Programm aus 7.1 erweitern, um uns diese beiden Teile zu erleichtern.

Wir geben das folgende Programm ein:

ADRESSE	OP. CODE
1000	C4 19
1002	F4 3F
1004	18
1005	1E

Wir erzeugen die Anzeige:

1 0 0 0 □ □ ? ?

und tasten:

	C	4
ME +	1	9
ME +	F	4
ME +	3	F
ME +	1	8
ME +	1	E

Der 2-Byte-Befehl C4 19 steht in den Speichern mit den Adressen 1000 und 1001, der 2-Byte-Befehl F4 3F steht in den Speichern mit den Adressen 1002 und 1003. Im Speicher mit der Adresse 1004 steht der 1-Byte-Befehl 18, der letzte Befehl steht im Speicher mit der Adresse 1005.

Bevor wir die beiden neuen Befehle besprechen, wollen wir das Programm starten: **A·D**, **RUN**. Es erscheint sofort das Ergebnis in der Anzeige. Was bewirken die beiden neuen Befehle? Durch den Befehl mit dem Operationscode „C4 19“ wird die Zahl 19 in den Akku gebracht. Der Befehl bedeutet: „Lade die Zahl 19 in den Akkumulator“. Statt 19 kann natürlich auch jede andere zweistellige Hexadezimalzahl eingesetzt werden.

Der 1-Byte-Befehl mit dem Operationscode 18 heißt: „Zeige den Akkuinhalt an und warte, bis eine Taste gedrückt wird“. Betätigen wir bei der Anzeige

□ □ 5 8 □ □ □ □

eine beliebige Taste (mit Ausnahme der Tasten **A·D** und **RS**), dann erscheint die Anzeige

1 0 0 5 □ □ 1 E

Das ist für uns das Zeichen, daß wir z. B. mit **CPU**, **6** den Akkuinhalt überprüfen können. Die Bearbeitung des Befehls mit dem Operationscode 18 ist jedenfalls beendet.

Wenn wir überprüfen wollen, ob nach der Bearbeitung des ersten Befehls tatsächlich 19 im Akku steht, können wir nach Beendigung dieses Befehls einen Breakpoint (engl. Haltepunkt, Zwischenstop) setzen. Wir überschreiben den Inhalt F4 im Speicher 1002 mit 1E: **A→D**, **A→D**, **ME+**, **ME+**, **1**, **E**. Wenn wir das Programm neu starten (**A→D**, **RUN**), wird nun der erste Befehl bearbeitet. Der Computer meldet sich danach mit der Anzeige

1 0 0 2 7 **1 E**

Wir können uns den momentanen Akkuinhalt ansehen: **CPU**, **6**. Im Akku steht – wie erwartet – 19. Soll das Programm an dieser Stelle fortgesetzt werden, so muß wieder der ursprüngliche Inhalt F4 in den Speicher 1002 geschrieben werden: **RUN**, **RUN**, **F**, **4**, **RUN**.

Eine schwierige Frage: Warum wird das Programm nicht zwischendurch angehalten, wenn wir in den Speicher mit der Adresse 1001 den Breakpoint 1E schreiben? Tun wir es! Nach dem Start läuft das Programm bis zum dritten Befehl, der Computer zeigt uns das Additionsergebnis 5D. Der Computer hat sich ganz kleinlich an seine Anweisungen gehalten. Wir haben ihm nämlich gesagt: „Lade die Zahl 1E in den Akku“, „addiere zum Akkuinhalt die Zahl 3F“, „zeige den Akkuinhalt an“. Da $1E_{16} + 3F_{16} = 5D_{16}$ ist, hat der Computer seine Anweisungen exakt erfüllt. Wir merken uns: Der Breakpoint oder der Haltepunkt darf nur nach einem vollständigen Befehl gesetzt werden, sonst wird er nicht als neuer Befehl interpretiert.

Die Schwierigkeit für uns als Programmierer besteht darin, daß wir uns genauso exakt an die Abmachungen halten müssen, wie es der Computer tut. Der Computer hat alle diese Abmachungen in seinem Kopf. Wir müssen sie mühsam erlernen. Hierfür ein weiteres Beispiel: Wir ersetzen im ursprünglichen Programm den Inhalt F4 von Speicher 1002 durch 18 und starten das Programm. Der Computer lädt die Zahl 19 in den Akku und zeigt sie an. Bis hier ist noch alles in Ordnung. Wenn wir jetzt aber die Taste **RUN** drücken, erhalten wir als Ergebnis nicht 58, sondern 8C. Kein Wunder, der Computer weiß auch gar nichts von einem Additionsauftrag. Er liest nach Betätigung der Taste **RUN** seinen nächsten Auftrag 3F und nicht F4 3F. Das, was ihm damit gesagt wird, tut er. Dann zeigt er völlig richtig den Akkuinhalt an.

Natürlich können wir das Additionsprogramm so ändern, daß uns außer der Summe auch die Summanden angezeigt werden:

ADRESSE	OP. CODE
1000	C4 19
1002	18
1003	C4 3F
1005	18
1006	F4 19
1008	18
1009	1E

Nach Programmstart wird 19 in den Akku geladen, im zweiten Befehl wird 19 angezeigt. Nach Betätigung der Taste **RUN** (oder einer anderen) wird im dritten Befehl 3F geladen und im vierten angezeigt. Würden wir hier gleich den Additionsauftrag geben, könnte 3F nicht angezeigt werden. Nach erneuter Betätigung der Taste **RUN** wird jetzt zur Zahl 3F, die noch im Akku steht, die Zahl 19 addiert. Die Summe wird anschließend angezeigt. Der letzte Befehl 1E ist hier nur eine Sicherheitsmaßnahme. Würde er fehlen, so würde sich der Computer – nach Anzeigen der Summe – bei versehentlicher neuer Betätigung der Taste **RUN** – aus den weiteren Speichern neue Befehle suchen.

7.4 Das Carry/Link-Flag: CY/L

Wir haben im letzten Abschnitt zwei Additionsprogramme besprochen. Das zweite hat den Vorteil, daß auch die Summanden angezeigt werden; es hat den Nachteil, daß der eine Summand zweimal eingegeben werden muß. Diesen Nachteil werden wir im nächsten Abschnitt noch abstellen. Zunächst etwas viel Wichtigeres. Wenn wir nach einem der beiden Additionsprogramme z. B. A3 und 9E addieren, erhalten wir 41, sicher ein zu niedriges Ergebnis. Was müßte sich denn ergeben? Wir addieren schriftlich:

$$\begin{array}{r} A3 \\ 9E \\ \hline 141 \end{array}$$

+

$3_{16} + E_{16} = 11_{16}$

(vgl. Additionstabelle in Abschnitt 6.5.). Wir schreiben die letzte 1 hin und notieren eine 1 als Übertrag. Jetzt addieren wir:

$$\begin{array}{r} A3 \\ 9E \\ \hline 141 \end{array}$$

+

$A_{16} + 9_{16} = 13_{16}$
 $13_{16} + 1_{16} = 14_{16}$. Die Summe ist 4. Es ergibt sich wieder ein Übertrag 1.

Das Additionsergebnis ist also: $A3_{16} + 9E_{16} = 141_{16}$. Das Ergebnis ist eine dreistellige Hexadezimalzahl. Es ist einsehbar, daß die Zahl nicht im Akku stehen kann. Der Akku ist nur ein 8-Bit-Register. Da sich bei der Addition von zwei achtstelligen Dualzahlen höchstens eine neunstellige Dualzahl ergeben kann (vgl. 4. Aufg. zu Kapitel 6), kann auch außer dem Akkuinhalt hier nur eine 1 fehlen, der Übertrag zur nächsten Stelle. Diesen Übertrag (ja oder nein; 1 oder 0) schreibt der Mikroprozessor nach jeder Addition in das Carry/Link-Flag (Abkürzung: CY/L; to carry = tragen, bringen; link = Bindeglied, Verbindungsstück; flag = Flagge, Fahne). Daß das CY/L-Flag etwas mit „tragen“ zu tun hat, ist verständlich: es enthält ja den Übertrag. Es bildet gleichzeitig das Bindeglied bei mehrstelligen Additionen. Doch woran erkennen wir, ob der Mikroprozessor die Übertrags-Fahne gehißt hat oder nicht? Woran erkennen wir, ob im CY/L-Register eine 1 oder eine 0 steht?

Der Mikroprozessor hat neben dem CY/L-Flag sieben weitere 1-Bit-Register, die alle nur eine 1-Bit-Information enthalten. Auch wenn sie alle unterschiedliche Bedeutung haben und bei unterschiedlichen Problemen benutzt werden, werden sie zu einem 8-Bit-Register, dem Status-Register S, zusammengefaßt. An der vordersten Stelle in diesem Register steht das CY/L-Flag:

Status-Register							
CY/L							

Leider können wir jetzt kein Programm wählen, bei dem zum Schluß die Summe (der Akkuinhalt) angezeigt wird. Bei dieser Anzeige wird der Inhalt des Status-Registers verändert. Wir müssen bei den beiden Programmen aus dem letzten Abschnitt den letzten Anzeige-Befehl „18“ mit 1E überschreiben. Oder wir tasten neu ein:

ADRESSE	OP. CODE
1000	C4 A3
1002	F4 9E
1004	1E

Nach dem Programmstart kommen wir zum Haltepunkt (Befehlscode 1E). Jetzt können wir mit CPU, 6 den Akkuinhalt 41 erkennen, mit CPU, 5 das Statusregister untersuchen. Es wird als Inhalt des Status-Registers auch eine zweistellige Hexadezimalzahl angezeigt. Interessant ist für uns, ob diese Zahl größer als 7F ist oder nicht. Alle Zahlen zwischen FF₁₆ und 80₁₆ beginnen, wenn man sie in achtstellige Dualzahlen umwandelt, mit einer 1. Die vollständige Summe ist also 141₁₆. Zur Kontrolle lassen wir noch einmal die Zahlen 19 und 3F addieren:

ADRESSE	OP. CODE
1000	C4 19
1002	F4 3F
1004	1E

Die Summe (im Akku) ist 58, im Status-Register steht jetzt sicherlich eine Zahl zwischen 7F₁₆ und 00₁₆. Es ergab sich also kein Übertrag. Mit 58 ist die richtige Summe bestimmt worden. Ohne daß die folgenden Befehle hier schon erklärt werden, soll unser Programm so verändert werden, daß die Summe wieder in der Anzeige erscheint, und daß zusätzlich die rote Leuchtdiode leuchtet, wenn sich ein Übertrag ergibt.

ADR.	OP. CODE	BEMERKUNGEN
1000	39 00	DIE ROTE LEUCHTDIODE WIRD GEGEBENENFALLS GELOESCHT. ERKLAERUNG FOLGT SPAETER
1002	1C4 A3	
1004	118	DIESER TEIL IST SCHON BEKANNT
1005	1C4 9E	
1007	118	
1008	1F4 A3	
100A	1CD E3	DER INHALT DES CY/L-FLAGS WIRD NACH DER ADDITION AN DER ROTEN LEUCHTDIODE ANGEZEIGT.
100C	106	
100D	1D4 80	
100F	13C	
1010	13C	ERKLAERUNG FOLGT SPAETER
1011	13C	
1012	13C	
1013	107	
1014	1C5 E3	
1016	118	DIESER TEIL IST SCHON BEKANNT
1017	11E	

Nach Programmstart wird der erste Summand A3 angezeigt. Nach erneuter Betätigung der Taste RUN erscheint der zweite Summand 9E. Wird noch einmal die Taste RUN gedrückt, so erscheint die Summe. Die vollständige Summe steht im CY/L-Register und im Akku: 141₁₆. Mit diesem Programm können beliebige zweistellige Hexadezimalzahlen addiert werden.

7.5 Eine zweite Adressierungsart

Bislang haben wir zwei Befehle mit unmittelbarer Adressierung (engl. immediate addressing) besprochen. So nennt man diese Adressierungsart. Ein bißchen eigenartig ist das schon, denn die Befehle „lade die im zweiten Byte angegebene Zahl in den Akku“ und „addiere die im zweiten Byte angegebene Zahl zum Inhalt des Akkus“ enthalten gar keinen Adressenteil. Die zu ladende oder zu addierende Zahl ist unmittelbar Teil des Befehls.

Bei allen anderen Adressierungsarten ist die entsprechende Zahl nicht Teil des Befehls. Sie steht irgendwo in einem Speicher an ganz anderer Stelle. Der Befehl enthält dann einen eindeutigen Hinweis darauf, wo diese Zahl zu finden ist; mit dem Befehl ist eindeutig festgelegt, welche Adresse der Speicher hat, in dem die Zahl steht. Die direkte Adressierung (engl. direct addressing) ist besonders einfach. Leider lassen sich so nicht alle Speicher ansprechen. Die Befehle mit direkter Adressierung sind auch 2-Byte-Befehle. Da das erste Byte die Information über die Operation (z. B. „lade in den Akku“) und die Adressierungsart (z. B. direkte Adressierung) enthält, bleibt für die Adresse nur noch das zweite Byte. Es lassen sich daher nur 256 verschiedene Speicherplätze direkt adressieren. Vom Mikroprozessorhersteller ist festgelegt, daß dieses die Speicher mit den Adressen von FF00 bis FFFF sind. In der ersten Ausbaustufe unseres Computers sind hiervon nur für die Adressen FFC0 bis FFFF 64 Speicher vorhanden. Von diesen Speicherplätzen stehen uns (dem Anwender) nur die 16 mit den Adressen von FFE0 bis FFEF zur freien Verfügung. Die anderen 48 werden z. T. von der Betriebs-Software benutzt, z. T. sind sie für den Stack, den Stapelspeicher, vorgesehen. Hierüber wird natürlich noch zu sprechen sein. Wir wollen uns zunächst nur den Bereich merken, für den direkte Adressierung zugelassen ist:

FFE0	} 16 Speicherplätze
FFE1	
...	
FFEE	
FFEF	

Der Rest ist nun einfach: Der Operationscode für den Befehl „lade die Zahl, die im Speicher mit der Adresse FFE0 steht, in den Akku“ ist C5 E0. Der Operationscode für den Befehl „addiere die Zahl, die im Speicher mit der Adresse FFE7 steht, zum Inhalt des Akkus und bringe das Ergebnis in den Akku“, ist F5 E7. Und gleich ein weiterer Befehl, der mit unmittelbarer Adressierung nicht vorkommen kann: Der Operationscode für den Befehl „bringe den Inhalt des Akkus in den Speicher mit der Adresse FFEF“ ist CD EF. Das letzte Byte vom Operationscode bezeichnet jeweils den Speicher. Da für die direkt adressierbaren Speicher feststeht, daß sie im Bereich von FF00 bis FFFF liegen, genügt die Angabe E0, um den Speicher FFE0 zu bezeichnen. Wandelt man das erste Byte vom Operationscode in eine Dualzahl um, so stellt man schnell fest, daß die drei niedrigsten Dualziffern stets 101 sind (D₁₆ = 1101₂). Das ist der Hinweis auf die direkte Adressierung (vgl. Abschnitt 7.2). Was bewirkt folgendes kleines Programm?

ADRESSE	OP. CODE
1000	C5 E1
1002	F5 E2
1004	CD E3
1006	1E

Im ersten Befehl wird der Inhalt des Speichers FFE1 in den Akku geladen. Dazu wird im zweiten Befehl der Inhalt des Speichers FFE2 addiert. Die Summe wird in den Speicher mit der Adresse FFE3 geschrieben. Wir müssen jetzt vor Programmstart die beiden Summanden in die Speicher mit den Adressen FFE1 und FFE2 eingeben. Nach Ablauf des Programms finden wir die Summe im Speicher mit der Adresse FFE3.

Ein Versuch sollte gewagt werden: Nach Eingabe des Programms wählen wir mit **A↔D**, **F**, **F**, **E**, **1** den Speicherplatz für den ersten Summanden an. Nach **A↔D**, Eingabe 1. Summand, **ME+**, Eingabe 2. Summand, starten wir unser Programm mit **A↔D**, **RUN**. Der Computer meldet sich nach Beendigung seines Auftrages beim Haltepunkt (Adresse 1006; Operationscode 1E). Nach **A↔D**, **F**, **F**, **E**, **3** können wir die Summe als Inhalt des Speichers FFE3 ablesen. Hier fehlt wieder die Berücksichtigung des Übertrages (vgl. 7.4). Wir werden ihn im Programm im nächsten Abschnitt wieder beachten.

7.6 Noch mehr Komfort

	ADRESSE	OP. CODE
1. TEIL	1000	39 00
2. TEIL	1002	16
	1003	C5 D8
	1005	CD E1
	1007	16
	1008	C5 D8
	100A	CD E2
	100C	F5 E1
	100E	CD E3
3. TEIL	1010	06
	1011	D4 80
	1013	3C
	1014	3C
	1015	3C
	1016	3C
	1017	07
4. TEIL	1018	C5 E3
	101A	18
	101B	1E

Nach der Eingabe starten wir das Programm. In der Anzeige erscheint eine 0. Der Computer wartet auf eine Zahleneingabe. Wir können beliebig viele Ziffern eingeben. Der Computer merkt sich nur die beiden letzten. Nach Betätigung einer Funktionstaste (außer **A↔D**) wird die nächste Zahleneingabe erwartet. Betätigen wir dann wieder eine Funktionstaste, so erscheint die Summe der beiden eingegebenen Zahlen. Die rote Leuchtdiode zeigt den Übertrag an (LED leuchtet: 1, LED leuchtet nicht: 0). Mit **A↔D**, **RUN** kann das Programm erneut gestartet werden. Den ersten und den dritten Teil des Programms werden wir später genauer erklären. Im ersten Teil wird die rote Leuchtdiode ausgeschaltet, im dritten Teil wird sie eingeschaltet, wenn sich bei der Addition ein Übertrag ergibt. Bei leuchtender Leuchtdiode ist eine 1 vor die angezeigte Summe zu setzen. Im zweiten Teil ist nur der Befehl mit dem Operationscode 16 neu. Hiermit wird die zweistellige Zahleneingabe aufgerufen. Diese Zahleneingabe wird mit einer Funktionstaste (außer **A↔D**) abgeschlossen. Die eingegebene Zahl steht dann anschließend im Speicher mit der Adresse FFD8. Nach der Eingabe des ersten Summanden wird er in den Akku geladen (C5D8) und von hier in den Speicher FFE1 gebracht (CDE1). Der zweite Summand wird entsprechend in den Speicher FFE2 gebracht (16, C5D8, CDE2). Im vierten Teil wird die Summe (natürlich ohne den möglichen Übertrag) wieder in den Akku geladen, damit sie anschließend angezeigt werden kann. Dann folgt der Breakpoint. Wie wir gesehen haben, werden beide Summanden und die Summe in den Speichern FFE1, FFE2 und FFE3 gespeichert. Wir können diese drei Zahlen nach Beendigung der Addition dort wiederfinden.

7.7 Schriftliche Addition von Hexadezimalzahlen

Im Abschnitt 7.4 haben wir die beiden Hexadezimalzahlen A3 und 9E schriftlich addiert. Wir haben dort gesehen, daß die Addition genauso durchgeführt wird wie im Dezimalsystem, nur mit dem Unterschied, daß wir sechs weitere Ziffern A, 0, ..., F zur Verfügung haben.

Wir wollen uns das noch einmal an einem Beispiel ansehen: $4CE_{16}$ und $A2F_{16}$ sollen addiert werden.

$$\begin{array}{r} 4 \text{ C E} \\ A \text{ 2 F} \\ \hline 1 \leftarrow \\ \text{D} \end{array} +$$

Wir addieren in der letzten Stelle E und F und erhalten 1D. Wer Schwierigkeiten hat, kann in der Additionstabelle für einstellige Hexadezimalzahlen nachsehen (vgl. Abschnitt 6.5.) Wir schreiben D hin und notieren den Übertrag 1.

$$\begin{array}{r} 4 \text{ C E} \\ A \text{ 2 F} \\ \hline 1 \leftarrow \\ \text{F D} \end{array} +$$

Jetzt addieren wir:
 $C + 2 = E$; $E + 1 = F$
 Wir schreiben F hin. Es ergibt sich kein Übertrag.

$$\begin{array}{r} 4 \text{ C E} \\ A \text{ 2 F} \\ \hline 1 \leftarrow \\ \text{E F D} \end{array} +$$

Jetzt addieren wir:
 $4 + A = E$.
 Die Summe lautet also:
 $4CE + A2F = EFD$

Natürlich läßt sich diese Summe auch im Dualsystem ermitteln. Wenn man sich aber an die Hexadezimalzahlen gewöhnt hat, ist der bisher beschriebene Weg der schnellste. Addition im Dualsystem (so macht es unser Computer): Die Addition wird in drei Schritten durchgeführt.

1. Schritt: Umwandlung ins Dualsystem,
2. Schritt: Addition im Dualsystem,
3. Schritt: Umwandlung ins Hexadezimalsystem.

$$\begin{array}{r} \text{1. Schritt} \rightarrow \\ 4CE_{16} = 0100 \ 1100 \ 1110_2 \\ A2F_{16} = 1010 \ 0010 \ 1111_2 \\ \hline 1 1 1 1 \\ 1 1 1 1 \\ \hline EFD_{16} = 1110 \ 1111 \ 1101 \\ \leftarrow \text{3. Schritt} \end{array} \quad \begin{array}{l} \text{2. Schritt} \\ \downarrow \end{array}$$

7.8 Aufgaben zu Kapitel 7

1. Es sind folgende Summen im Hexadezimalsystem zu bestimmen:
 - 1.1 $B5 + 2C$
 - 1.2 $9E + D5$
 - 1.3 $A73F + 58C1$
 - 1.4 $9D3F + 12BF$
2. Natürlich können auch mehr als zwei Summanden addiert werden
 - 2.1 $C5 + 2A + 5D$
 - 2.2 $FF + FF + FF$
 - 2.3 $7B1 + 1C3F + DA + FE6$
3. Wie kann man am ersten Byte des Operationscodes eines 2-Byte-Befehls erkennen, daß es sich um eine direkte Adressierung handelt?

7.9 Zusammenstellung der bisher besprochenen Befehle

7.9.1 Der mnemonische Code

Wenn wir jetzt die bisher benutzen Befehle zusammenstellen, wollen wir gleich den mnemonischen Code mit angeben (Mneme = Erinnerung, Gedächtnis; griechisch). Der mnemonische Code ist ein „Gedächtniscode“; er soll uns helfen, die Befehle leichter zu behalten. Er gibt kurz (und verstümmelt) das an, was wir z. B. mit „addiere den Inhalt vom Speicher der Adresse FFE0 zum Inhalt des Akkus und bringe das Ergebnis wieder in den Akku“ umschrieben haben. Er enthält ein Kürzel für die Operation und Hinweise auf die Operanden. Für „addiere“ wird ADD (engl. to add = addiere), für „lade“ wird LD (engl. to load = laden), für „speichere“ wird ST (engl. to store = lagern, speichern) als Abkürzung gewählt. Nach ADD, LD oder ST wird dann z. B. ein A angegeben, wenn der Akkuiinhalt der erste Operand ist, danach folgt ein Hinweis auf den zweiten Operanden. Diese Andeutungen werden konkreter, wenn wir die Befehle zusammenstellen.

Wir werden später unsere Programme zunächst im mnemonischen Code schreiben und erst dann das fertige Programm für den Computer in den Operationscode übersetzen.

7.9.3 Befehle mit direkter Adressierung

BESCHREIBUNG	MNEM. CODE	IOP. CODE	OPERATION
ADDIERE AUS DEM SPEICHER ZUM AKKU-INHALT DIREKTE ADRESSIERUNG	ADD A,BEZ	IF5 XX	ADR:=FF00+XX (A):=(A)+(ADR); CY/L
LAD E AUS DEM SPEICHER IN DEN AKKUMULATOR DIREKTE ADRESSIERUNG	LD A,BEZ	IC5 XX	ADR:=FF00+XX (A):=(ADR)
BRINGE DEN AKKU-INHALT IN DEN SPEICHER DIREKTE ADRESSIERUNG	ST A,BEZ	ICD XX	ADR:=FF00+XX (ADR):=(A)

BEZ steht für die Bezeichnung eines Speichers oder eines Speicherinhaltes. Dieser Speicher muß eine Adresse zwischen FF00 und FFFF haben. (Für uns frei verfügbar sind die Speicher FFE0 bis FFEF.) Das zweite Byte dieser Adresse ist beim Operationscode mit XX bezeichnet. Die vollständige Adresse ergibt sich dann als Summe FF00 + XX. Der Inhalt des Speichers mit der Adresse ADR, abgekürzt (ADR) wird zum Akkuiinhalt addiert, in den Akku geladen, oder der Akkuiinhalt wird zum Speicherinhalt.

Beispiele:

MNEM. CODE	OP. CODE
LD A,1.ZAHL	C5 E1
ADD A,2.ZAHL	F5 E2
ST A,SUMME	CD E3

7.9.2 Befehle mit unmittelbarer Adressierung

BESCHREIBUNG	MNEM. CODE	IOP. CODE	OPERATION
ADDIERE UNMITTELBAR ZUM AKKU-INHALT	ADD A,=ZAHL	IF4 XX	(A):=(A)+ZAHL; CY/L
LAD E UNMITTELBAR IN DEN AKKUMULATOR	LD A,=ZAHL	IC4 XX	(A):=ZAHL

Im mnemonischen Code deutet das Gleichheitszeichen die unmittelbare Adressierung an. Als Zahl kann jede Dezimalzahl von 0 bis 255 oder jede Hexadezimalzahl von 00 bis FF gesetzt werden. Im Operationscode darf für XX natürlich nur eine zweistellige Hexadezimalzahl stehen. Diese Zahl wird auch als Hexadezimalzahl zum Akkuiinhalt addiert oder in den Akku eingeschrieben. Die Angabe CY/L beim Additionsbefehl soll darauf hinweisen, daß die Operation das CY/L-Flag beeinflusst. Es wird gesetzt oder gelöscht, je nachdem, ob ein Übertrag auftritt oder nicht.

Beispiele:

MNEM. CODE	OP. CODE
ADD A,=10	F4 0A
ADD A,=010	F4 10
LD A,=255	C4 FF
LD A,=036	C4 36

Wir wollen verabreden: Folgt nach dem Gleichheitszeichen im mnemonischen Code zunächst eine 0, dann soll die nachfolgende Zahl als Hexadezimalzahl aufgefaßt werden; andernfalls folgt eine Dezimalzahl; sie muß für den Operationscode noch in eine Hexadezimalzahl verwandelt werden.

Hierzu sind folgende Adressen festgelegt:

DATENSPEICHER, BEZEICHNUNG	ADR.
1. ZAHL	FFE1
2. ZAHL	FFE2
SUMME	FFE3

7.9.4 Die Calls

Die sechzehn 1-Byte-Befehle mit dem Operationscode 10, 11, ..., 1E bzw. 1F sind keine Befehle wie die übrigen. Alle übrigen Befehle bewirken nur eine bestimmte Operation, die der Mikroprozessor direkt ausführt. Mit den Calls (engl. to call = rufen) werden ganze Programmteile aufgerufen, die in der Betriebs-Software fest gespeichert sind. Wir werden später sehen, wie ein solcher Programmteil aussieht, aus welchen Einzelbefehlen er zusammengesetzt ist.

Der Mikroprozessor-Hersteller hat nur die Möglichkeit geschaffen, mit sechzehn Operationscodes sechzehn verschiedene Hilfsprogramme aufzurufen. In der Betriebssoftware sind zwar diese sechzehn Hilfsprogramme gespeichert, aber die Bezeichnungen können wir frei wählen. Wir wollen als mnemonischen Code für diese Calls möglichst kurze Bezeichnungen benutzen, die aber verraten sollen, was sich bei ihrem Aufruf tut.

Wir kennen bisher drei Calls:

BESCHREIBUNG	MMEM.CODE	OP.CODE	OPERATION
ICALL ZWEISTELLIGE EINGABE UEBER TASTATUR	ICALL EIN 2ST	116	(FFD8) := EINGEGEBENE ZAHL

Mit diesem Call rufen wir die Eingabe einer zweistelligen Zahl auf. Sie wird mit Hilfe der Tasten 0 bis F eingegeben. Bei der Eingabe der dritten Ziffer geht die erste verloren. Bei Betätigung einer Funktionstaste (außer **A-D**) ist die Eingabe beendet. Es wird dann der Befehl bearbeitet, der nach dem Call in unserem Programm steht. Die eingegebene zweistellige Zahl steht jetzt im Speicher mit der Adresse FFD8. Vorsicht: Dieser Speicherplatz wird häufig von der Betriebssoftware benutzt. Wir müssen also diese hier zwischengespeicherte Zahl in einen Speicher bringen, der uns allein zur Verfügung steht.

BESCHREIBUNG	MMEM.CODE	OP.CODE	OPERATION
ICALL ANZEIGE VOM AKKU-INHALT	ICALL ANZ A	118	ANZEIGE (A)

Mit diesem Call können wir jederzeit den momentanen Akku-Inhalt anzeigen lassen. Die Anzeige wird durch Betätigung einer Ziffern- oder Funktionstaste (außer **A-D**) beendet. Im Akku steht immer noch der gleiche Inhalt. Der Inhalt des Status-Registers hat sich verändert.

BESCHREIBUNG	MMEM.CODE	OP.CODE	OPERATION
ICALL TEST VON PROGRAMMTEILEN (BREAKPOINT (HALTEPUNKT))	ICALL TEST	11E	

Dieser Call wird verwendet, wenn ein Programm getestet werden soll. Wenn ein Programm nicht wie geplant läuft, überschreibt man an geeigneter Stelle einen Operationscode (genauer: das erste Byte eines Operationscodes) mit 1E. Das Programm wird dann gestartet und läuft bis zu dieser Stelle. Jetzt prüft man, ob die Register-Inhalte im Mikroprozessor bzw. die Speicherinhalte für bestimmte Daten die erwarteten Werte haben. Dieser Call wird daher auch Breakpoint oder Haltepunkt genannt. Man setzt vorübergehend einen Breakpoint, um ein Programm bis zu dieser Stelle zu testen. Anschließend wird der Breakpoint wieder herausgenommen. Wir verwenden diesen Call auch bei unseren einfachen Programmen als letzten Befehl.

8. Und nun noch einmal alles vierstellig!

8.1 Das Extension-Register E

Wir kennen bislang zwei Register der CPU, zwei Register in der Zentraleinheit unseres Mikroprozessors: den Akkumulator und das Status-Register. Wir lernen jetzt ein weiteres 8-Bit-Register in der CPU kennen: Das Extension-Register. „Extension“ ist wieder ein englisches Wort und heißt „Ausdehnung“. Das Extension-Register können wir wohl am besten als Zusatz- oder Erweiterungsregister bezeichnen.

Folgenden Befehle machen deutlich, daß das Extension-Register zunächst einmal so etwas wie ein Speicher ist.

BESCHREIBUNG	MMEM.CODE	OP.CODE	OPERATION
ADDIERE DEN INHALT DES EXTENSION-REGISTERS ZUM AKKU-INHALT	ADD A,E	170	(A) := (A) + (E) ; CY/L
LADEN DEN INHALT DES EXTENSION-REGISTERS IN DEN AKKUMULATOR	LD A,E	148	(A) := (E)
LADEN DEN INHALT DES AKKUMULATORS IN DAS EXTENSION-REGISTER	LD E,A	148	(E) := (A)
TAUSCHE DIE INHALTE VON AKKUMULATOR UND EXTENSION-REGISTER	XCH A,E	101	(A) (E)

Die Beschreibung der Befehle in dieser Übersicht wird wahrscheinlich schon ohne weitere Erklärungen ausreichen. Trotzdem einige Bemerkungen, die z. T. als Wiederholungen gedacht sind:

1. Wenn wir mit A, E, FFE0 oder CY/L ein Register, einen Speicher oder ein Flag bezeichnen, dann meinen wir mit (A), (E), (FFE0) oder (CY/L) deren Inhalt.
2. Die Angabe CY/L-Flag beim Additionsbefehl soll darauf hinweisen, daß die Addition das CY/L-Flag beeinflusst. Es wird gesetzt oder gelöscht, je nachdem, ob ein Übertrag auftritt oder nicht.
3. Beim Lade-Befehl wird zunächst das Register angegeben, das geladen wird. Erst dann folgt eine Angabe über die Herkunft der zu ladenden Zahl. LD A, E heißt ausführlicher: Lade den Akku A mit dem Inhalt des Extension-Registers E.
4. Beim Lade-Befehl bleibt der Inhalt in dem Register (oder Speicher), aus dem geladen wird, unverändert. Bei LD A, E behält das Extension-Register, bei LD E, A behält der Akku seinen Inhalt. Entsprechendes gilt bei anderen Befehlen, z. B. beim Store- oder Additionsbefehl. Bei ADD A, E behält das Extension-Register seinen Inhalt.
5. XCH ist Abkürzung für EXCHANGE (engl. to exchange = austauschen, auswechseln). Der Befehl XCH A, E vertauscht die Inhalte von Akkumulator und Extension-Register.

Wir wollen jetzt in einem kleinen Programm mit diesen Befehlen ein wenig spielen, um dadurch mit diesen vier neuen 1-Byte-Befehlen vertraut zu werden.

MNEM. CODE	ADR.	OP. CODE
LD A,=08	1000	C4 08
LD E,A	1002	48
ADD A,E	1003	70
ADD A,E	1004	70
XCH A,E	1005	01
ADD A,E	1006	70
LD E,A	1007	48
ADD A,E	1008	70
LD E,A	1009	48
ADD A,E	100A	70
XCH A,E	100B	01
LD A,E	100C	40
ADD A,E	100D	70
XCH A,E	100E	01
LD A,E	100F	40
CALL TEST	1010	1E

Wenn wir dieses Programm starten, läuft es bis zum Breakpoint. Wir können jetzt mit **CPU**, **6** den Akkuinhalt, mit **CPU**, **7** den Inhalt des Extension-Registers feststellen. Beide Register-Inhalte sind 00. Wie kommt es dazu? Wir wollen den Programm-Ablauf in einer „Trace“ (engl. Spur, Fahrte) verfolgen. Trace kann hier mit Ablauf-Verfolgung übersetzt werden. Man schreibt dabei auf, wie sich die Inhalte der betreffenden Register (und Speicher) bei jedem Befehl ändern. Das Fragezeichen in der ersten Zeile soll andeuten, daß der entsprechende Inhalt unbekannt ist.

Mnem. Code	Registerinhalt nach Ausführung des Befehls	
	(E)	(A)
LD A, = 08	?	08
LD E,A	08	08
ADD A,E	08	10
ADD A,E	08	18
XCH A,E	18	08
ADD A,E	18	20
LD E,A	20	20
ADD A,E	20	40
LD E,A	40	40
ADD A,E	40	80
XCH A,E	80	40
LD A,E	80	80
ADD A,E	80	00*)
XCH A,E	00	80
LD A,E	00	00
CALL TEST	00	00

*) Der Übertrag, der bei dieser Addition entsteht, wird ins CY/L-Flag geschrieben. Nach Programm-Ablauf läßt sich auch das Status-Register untersuchen (**CPU** **5**). Im Status-Register muß eine Hexadezimalzahl größer als 7F stehen (vgl. Abschnitt 7.4).

Mit der Trace haben wir eine Untersuchungsmethode kennengelernt, die wir auch bei weiteren Programmen anwenden können. Im Zweifelsfall läßt sich dann immer noch der CALL-Test an eine beliebige Stelle ins Programm setzen (vgl. Abschnitt 7.9.4).

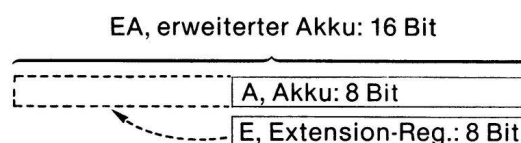
Wir hatten gesagt, daß das Extension-Register Ähnlichkeiten mit einem normalen Speicher hat. Es gibt aber doch wesentliche Unterschiede.

Es ist eben mit 1-Byte-Befehlen ansprechbar. Ein zweites Byte für die Adressierung ist nicht erforderlich. Es gibt nur ein Extension-Register.

Der wesentliche Vorteil – und da wird die Bezeichnung Zusatz- oder Erweiterungsregister erst verständlich – zeigt sich im nächsten Abschnitt.

8.2 Aus zwei mach eins!

Wir haben das Akkumulator- und das Extension-Register als zwei unabhängige Register in der CPU kennengelernt. Für einige Operationen lassen sich diese beiden Register zu einem erweiterten Akkumulator-Register (Extended Accumulator Register) zusammenfassen. Mit Hilfe des Extension-Registers E wird aus dem 8-Bit-Akku A ein 16-Bit-Akku EA.



Die Vorteile liegen auf der Hand: Es lassen sich Operationen mit vierstelligen Hexadezimalzahlen durchführen. Wir wollen uns dazu gleich etwas am Computer ansehen. Wir geben die folgenden Inhalte in die Speicher ab Adresse 1000 ein und starten das Programm:

ADRESSE	OP. CODE
1000	84 12 34
1003	19
1004	1E

Tastenfolge bei Anzeige

1 0 0 0 □ ? ? ? :

	8	4
ME +	1	2
ME +	3	4
ME +	1	9
ME +	1	E
A↔D	RUN	

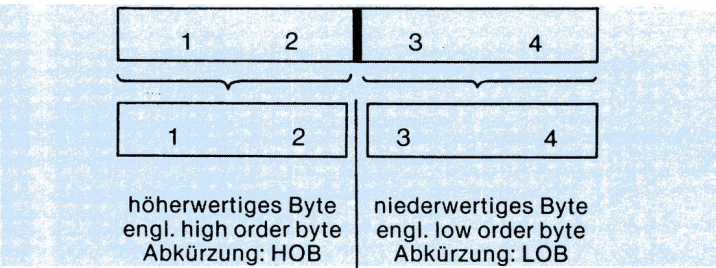
In der Anzeige erscheint die vierstellige Zahl 3412. Wir werden wahrscheinlich schon vermuten, was die beiden neuen Befehle bewirken. Der Befehl mit dem Operationscode 19 ist ein Call (vgl. 7.9.4). Er sorgt dafür, daß der Inhalt des EA-Registers angezeigt wird. Wir bezeichnen ihn im mnemonischen Code mit CALL ANZ EA. Der 3-Byte-Befehl (84 12 34) muß dann wohl ein unmittelbarer Lade-Befehl sein, der dafür sorgt, daß die letzten beiden Bytes 12 und 34 in das EA-Register geladen werden. Nur, warum erscheint in der Anzeige nicht die Zahl 1234?

Wir lassen unser kleines Programm bis zum Breakpoint laufen. Bei der Anzeige 3412 muß eine beliebige Taste (außer **A↔D**) betätigt werden. Jetzt überprüfen wir die Inhalte vom Extension-Register und vom Akku: **CPU**, **7** und **CPU**, **6**. Ergebnis: (E) = 34, (A) = 12. Wir sehen, beim Anzeige-Befehl wird der Inhalt vom Extension-Register vorn, der Akkuinhalt dahinter angezeigt. Das ist auch die richtige Reihenfolge der vier Ziffern der im EA-Register gespeicherten vierstelligen Hexadezimalzahl.

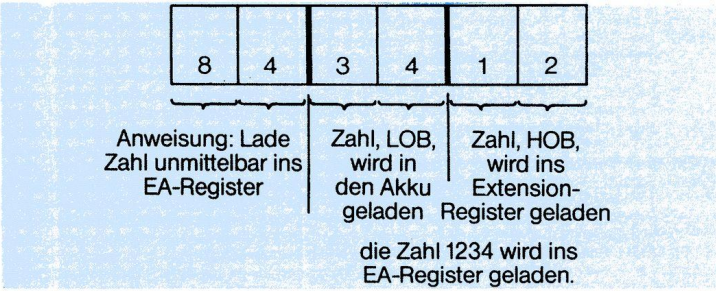
Der Mikroprozessor hat eine Eigenschaft, mit der wir leben müssen – und wohl auch leben können: Bei dem erwähnten Lade-Befehl (und das gilt grundsätzlich auch bei allen weiteren 3-Byte-Befehlen) erwartet der Mikroprozessor von uns zunächst die beiden letzten Ziffern, das niederwertige Byte, anschließend die beiden voranstehenden Ziffern, das höherwertige Byte der vierstelligen Hexadezimalzahl.

Beispiel:

Die vierstellige Hexadezimalzahl sei 1234.



So die Hexadezimalzahl 1234 unmittelbar in EA-Register geladen werden, so heißt der Operationscode:



Wie in Kapitel 7 gibt es neben dem Lade-Befehl auch hier einen Additionsbefehl mit unmittelbarer Adressierung.

BESCHREIBUNG	MNEM.CODE	OP.CODE	OPERATION
ADDIERE UNMITTELBAR ZUM INHALT DES EA-REGISTERS	ADD EA,=ZAHL	B4 YY XX	$\langle EA \rangle := \langle EA \rangle + ZAHL$ CY/L
ILADE UNMITTELBAR IN DAS EA-REGISTER	LD EA,=ZAHL	B4 YY XX	$\langle EA \rangle := ZAHL$

Als Zahl darf jede Dezimalzahl von 0 bis 65535 oder jede Hexadezimalzahl von 0000 bis FFFF gesetzt werden. Im Operationscode wird eine vierstellige Hexadezimalzahl zugrundegelegt. XX bezeichnet das high order byte (HOB), YY das low order byte (LOB) dieser vierstelligen Hexadezimalzahl. Das HOB wird in das Extension-Register geladen oder zum Inhalt des E-Registers addiert. Das LOB wird in den Akku geladen oder zum Akkuinhalt addiert. Die Angabe CY/L beim Additionsbefehl weist wieder darauf hin, daß das CY/L-Register beeinflusst wird.

Beispiele:

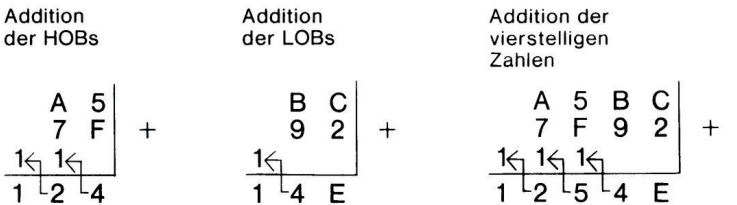
MNEM.CODE	OP.CODE
ADD EA,=1	B4 01 00
ADD EA,=0AB	B4 AB 00
LD EA,=65280	B4 00 FF
LD EA,=0FF00	B4 00 FF

Wir hatten verabredet (vgl. 7.9.2), daß eine 0 nach dem Gleichheitszeichen im mnemonischen Code andeuten soll, daß eine Hexadezimalzahl folgt.

Wir wollen mit diesen Befehlen ein Programm zur Addition von zwei vierstelligen Hexadezimalzahlen schreiben. Die beiden Zahlen A5BC und 7F92 sollen z. B. addiert werden. Wir wissen schon, daß zunächst ein Summand in das EA-Register geladen werden muß, dann muß der andere zum Inhalt des EA-Registers addiert werden. Die erforderlichen Befehle sind also:

MNEM.CODE	ADR.	OP.CODE
LD EA,=0A5BC	1000	B4 BC A5
ADD EA,=07F92	1003	B4 92 7F
CALL TEST	1006	1E

Wenn wir diese Programm starten, finden wir anschließend (CPU, 7 und CPU, 6) im Extension-Register den Inhalt 25, im Akku den Inhalt 4E. Welches Ergebnis erwarten wir? Wir haben ja schon Übung bei der Addition im Hexadezimalsystem:



Dieses Beispiel für die Addition im EA-Register zeigt: Der Übertrag, der bei der Addition der LOBs entsteht, wird bei der Addition der HOBs berücksichtigt. Der jetzt entstehende Übertrag paßt natürlich nicht mehr ins EA-Register. Er steht wieder im CY/L-Flag. Wir können uns davon überzeugen, indem wir uns mit CPU, 5 den Inhalt des Status-Registers in die Anzeige holen.

Wir können unser kurzes Programm wieder so erweitern, daß der Inhalt des EA-Registers und des CY/L-Flags nach der Addition angezeigt werden. Zur Anzeige des EA-Registers benutzen wir den Call: CALL ANZ EA, zur Anzeige von (CY/L) an der roten Leuchtdiode verwenden wir wieder die beiden Programmteile, die noch nicht näher erklärt wurden (vgl. die Programme in den Abschnitten 7.4 und 7.6).

NR.	MNEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.		1000	B9 00	DI E ROTE LEUCHTDIODE WIRD GEGEBENFALLS GELOESCHT. ERKLÄRUNG FOLGT SPÄTER
2.	LD EA,=0A5BC	1002	B4 BC A5	LADE 1. SUMMANDEN
3.	ADD EA,=07F92	1005	B4 92 7F	ADDIERE 2. SUMMANDEN
4.	IST A,LOB	1008	CD E3	$\langle FFE3 \rangle := \langle A \rangle$
5.		100A	06	DER INHALT DES CY/L-FLAGS WIRD NACH DER ADDITION AN DER ROTEN LEUCHTDIODE ANGEZEIGT.
6.		100B	D4 80	
7.		100D	13C	
8.		100E	13C	
9.		100F	13C	
10.		1010	13C	
11.		1011	07	ERKLÄRUNG FOLGT SPÄTER
12.	LD A,LOB	1012	CD E3	$\langle A \rangle := \langle FFE3 \rangle$
13.	CALL ANZ EA	1014	19	$\langle EA \rangle$ WIRD ANGEZEIGT
14.	CALL TEST	1015	1E	

Eine Bemerkung dazu: Im 4. Befehl wird der Akkuinhalt „gerettet“, er wird weggespeichert; im 12. wird er „wiederhergestellt“, er wird wieder geladen. Das ist deswegen erforderlich, weil der Akku im 5. bis 11. Befehl benötigt wird. Bei Beginn des 13. Befehls muß der Akku wieder den Inhalt haben, den er bei der Addition bekommen hat. Der Inhalt des Extension-Registers wird vom 4. bis zum 12. Befehl nicht verändert.

Wir können mit diesem Programm zwei beliebige vierstellige Hexadezimalzahlen addieren. Es soll hier gleich angemerkt werde, daß es für die Addition von mehr als vierstelligen Zahlen keine weiteren Befehle gibt. Für die Addition von größeren Zahlen müßte man die besprochenen Additionsbefehle mehrfach anwenden.

Zu dem verwendeten Call halten wir fest:

BESCHREIBUNG	MNEM. CODE	OP. CODE	OPERATION
ICALL ANZEIGE VOM INHALT DES EA-REGISTERS	ICALL ANZ EA	19	ANZEIGE:<EA>

Mit diesem Call können wir jederzeit den Inhalt des EA-Registers anzeigen lassen. Die Anzeige wird durch Betätigung einer Ziffern- oder einer Funktionstaste (außer **A↔D**) beendet. Im EA-Register steht immer noch der gerade angezeigte Inhalt. Der Inhalt des Status-Registers hat sich verändert.

8.3 Vierstellig, direkt adressiert

Wie bei der Addition zweistelliger Zahlen (vgl. Abschnitt 7.5) läßt sich auch hier eine direkte Adressierung verwenden. Das gilt nicht nur beim Add-, sondern auch beim Load- und beim Store-Befehl.

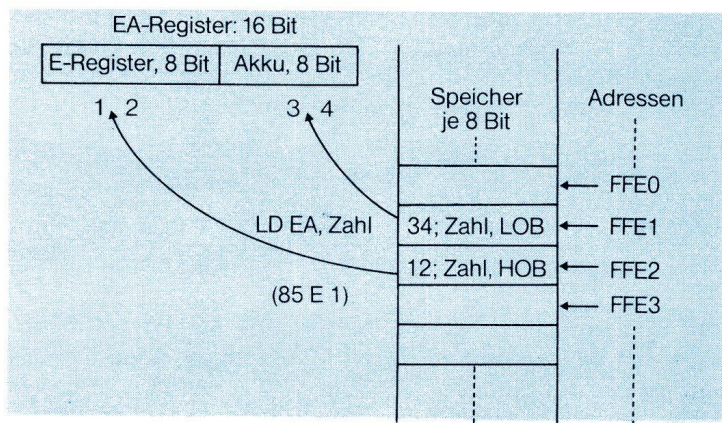
Wir wollen uns das Verfahren wieder beim Laden ansehen. Der Ladebefehl hat z. B. den Operationscode 85 E1. Der erste Teil bedeutet: Lade in das EA-Register bei direkter Adressierung, mit dem zweiten Teil wird der Speicher FFE1 adressiert. Aber wie kann das funktionieren? Ins EA-Register müssen zwei Byte geladen werden, im Speicher mit der Adresse FFE1 steht nur ein Byte. Was macht der Mikroprozessor? Er lädt die beiden Bytes aus dem bezeichneten Speicher FFE1 und aus dem mit der folgenden Adresse FFE1 + 1 = FFE2. Einprägen müssen wir uns, aus welchem Speicher er sich das LOB, aus welchem er sich das HOB holt. Bei der unmittelbaren Adressierung erwartete der Mikroprozessor erst das LOB, dann das HOB. Es ist nur konsequent, wenn er dann auch hier bei dieser Reihenfolge bleibt: er holt sich zunächst das LOB aus dem adressierten Speicher FFE1, dann das HOB aus dem nächsten Speicher FFE2. Wir wollen das probieren:

Wir laden folgenden kurzes Programm:

ADRESSE	OP. CODE
1000	85 E1
1002	19
1003	1E

Jetzt wählen wir mit **A↔D**, **F**, **F**, **E**, **1** den Speicher FFE1 und bereiten ihn mit **A↔D** für die Dateneingabe vor. Dann bringen wir mit **3**, **4**, **ME+**, **1**, **2** die Zahl 34 (LOB) in den Speicher FFE1 und die Zahl 12 (HOB) in den Speicher FFE2.

Nach dem Programmstart wird die vierstellige Zahl 1234 ins EA-Register geladen und anschließend angezeigt.



Die Befehle mit direkter Adressierung lauten:

BESCHREIBUNG	MNEM. CODE	OP. CODE	OPERATION
IADDIERE AUS DEM SPEICHER ZUM INHALT DES EA-REG. DIREKTE ADRESSIERUNG	IADD EA,BEZ	85 XX	ADR:=FF00+XX (EA):= (EA)+(ADR+1,ADR) ICV/L
ILADE AUS DEM SPEICHER IN DAS EA-REGISTER DIREKTE ADRESSIERUNG	ILD EA,BEZ	85 XX	ADR:=FF00+XX (EA):= (ADR+1,ADR)
IBRINGE DEN INHALT DES EA-REG. IN DEN SPEICHER DIREKTE ADRESSIERUNG	IST EA,BEZ	8D XX	ADR:=FF00+XX (ADR+1,ADR):= (EA)

BEZ steht für die Bezeichnung eines Speichers oder eines Speicherinhalts. Dieser Speicher muß eine Adresse zwischen FF00 und FFFE haben (für uns frei verfügbar sind die Speicher FFE0 bis FFE2), denn die Operation betrifft außer diesem 1. Speicher noch einen 2. mit der folgenden Adresse. Das zweite Byte der 1. Adresse ist beim Operationscode mit XX bezeichnet. Die vollständige Adresse ADR ergibt sich dann als Summe FF00 + XX. Der Inhalt der beiden Speicher mit den Adressen ADR + 1 und ADR wird als vierstellige Hexadezimalzahl zum Inhalt des EA-Registers addiert oder in das EA-Register geladen. Beim Store-Befehl wird der Inhalt des EA-Registers in diesen beiden Speichern abgespeichert. Wichtig ist, daß im Speicher mit der kleineren Adresse (ADR) das low order byte (LOB), im Speicher mit der größeren Adresse (ADR + 1) das high order byte (HOB) steht.

Beispiele:

MNEM. CODE	OP. CODE
LD EA,1,ZAHL	85 E1
ADD EA,2,ZAHL	85 E3
ST EA,SUMME	8D E5

Hierzu müssen die Adressen wie folgt festgelegt sein:

DATENSPEICHER, BEZEICHNUNG	ADR.
1. ZAHL	FFE1
1. ZAHL, LOB	FFE1
1. ZAHL, HOB	FFE2
2. ZAHL	FFE3
2. ZAHL, LOB	FFE3
2. ZAHL, HOB	FFE4
SUMME	FFE5
SUMME, LOB	FFE5
SUMME, HOB	FFE6

Diese Beispiele sind natürlich schon im Hinblick auf das nächste Programm ausgewählt. Um ein ähnlich komfortables Programm wie am Ende des 7. Kapitels zu schreiben, müssen wir noch einen Call kennenlernen:

BESCHREIBUNG	MMEM.CODE	OP.CODE	OPERATION
ICALL VIERSTELLIGE EINGABE UEBER TASTATUR	ICALL EIN 4ST	17	(FFD9,FFD8):= EINGEGEBENE ZAHL

Mit diesem Call rufen wir die Eingabe einer vierstelligen Zahl auf. Auch hier wird mit Hilfe der Tasten 0 bis F eingegeben. Bei der Eingabe der fünften Ziffer geht die erste verloren. Bei Betätigung einer Funktionstaste (außer **A-D**) ist die Eingabe beendet. Es wird dann der Befehl bearbeitet, der nach dem Call in unserem Programm steht. Die eingegebene vierstellige Zahl steht jetzt in den Speichern mit den Adressen FFD9 (HOB) und FFD8 (LOB). Vorsicht! Diese Speicherplätze werden häufig von der Betriebssoftware benutzt. Wir müssen die hier zwischengespeicherte Zahl in zwei Speicher bringen, die uns allein zur Verfügung stehen. Der Schiebeschalter muß hier in der oberen Stellung stehen. Das versteht sich eigentlich von selbst, wenn die vier rechten Sieben-Segment-Anzeigen benutzt werden sollen. Wir werden das nicht jedesmal erwähnen, wenn wir diesen Call verwenden.
Und jetzt das Programm (vgl. Abschnitt 7.6):

INR.	MMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.		1000	139 00	(DIE ROTE LEUCHTDIODE WIRD EINGEGEBENFALLS GELOESCHT. ERKLÄRUNG FOLGT SPÄTER)
2.	ICALL EIN 4ST	1002	17	EINGABE 1.ZAHL
3.	ILD EA,ZWSP	1003	185 D8	
4.	IST EA,1.ZAHL	1005	18D E1	
5.	ICALL EIN 4ST	1007	17	EINGABE 2.ZAHL
6.	ILD EA,ZWSP	1008	185 D8	
7.	IST EA,2.ZAHL	100A	18D E3	
8.	IADD EA,1.ZAHL	100C	1B5 E1	SUMME:=2.ZAHL+1.ZAHL
9.	IST EA,SUMME	100E	18D E5	
10.		1010	106	(DER INHALT DES CY/L-FLAGS WIRD NACH DER ADDITION AN DER ROTEN LEUCHTDIODE ANGEZEIGT.
11.		1011	1D4 80	
12.		1013	13C	
13.		1014	13C	
14.		1015	13C	
15.		1016	13C	ERKLÄRUNG FOLGT SPÄTER
16.		1017	107	
17.	ILD EA,SUMME	1018	185 E5	ANZEIGE DER SUMME
18.	ICALL ANZ EA	101A	119	
19.	ICALL TEST	101B	11E	

DATENSPEICHER,BEZ.	ADR.	BEMERKUNGEN
ZWSP	FFD8 (ZWISCHENSPEICHER FFD9)	
1.ZAHL	FFE1 (DIE 1. VIERSTELLIGE ZAHL FFE2 (WIRD IN DEN SPEICHERN FFE1 (LOB) UND FFE2 (HOB) ABGESPEICHERT	
2.ZAHL	FFE3 (2.ZAHL, VIERSTELLIG FFE4	
SUMME	FFE5 (SUMME, VIERSTELLIG FFE6	

Nach Programmstart erscheint in der Anzeige eine 0. Wir geben den ersten Summanden, die 1. Zahl, ein, z. B. A5BC. nach Betätigen der Taste **RUN** erscheint wieder die Anzeige 0. Der Computer wartet auf den zweiten Summanden, die 2. Zahl. Wir geben z. B. 7F92 ein. Nach erneuter Betätigung der **RUN**-Taste erscheint die Summe:

1

(CY/L)

25

(E)

4E

(A)

Die 1 wird an der roten Leuchtdiode angezeigt. Nach **A-D**, **RUN** kann eine neue Addition durchgeführt werden.

8.4 Der Computer kennt keine Dezimalzahlen

Der Mikroprozessor weiß nichts von Dezimalzahlen. Er führt alle Rechnungen (die bisher besprochene Addition und weitere Verknüpfungsarten) im Dualsystem durch. Wir können allenfalls noch sagen, er rechnet im Hexadezimalsystem. Dazu müssen wir je vier Dualziffern geeignet zusammenfassen. Aber vom Dezimalsystem versteht er nichts. Gewiß, wenn er 12 und 24 addiert, wird er auf die Summe 36 kommen. Er hat die Summe richtig im Hexadezimalsystem ermittelt: $12_{16} + 24_{16} = 36_{16}$. Er wird uns jedoch nicht verbieten können, diese Gleichung im Dezimalsystem zu lesen: $12_{10} + 24_{10} = 36_{10}$. Um es gleich festzuhalten, in dieser unterschiedlichen Interpretation der eingegebenen und der berechneten Zahlen besteht die einzige Möglichkeit, Rechnungen im Dezimalsystem durchzuführen. Für den Computer ist und bleibt die Zahl 12 eine Hexadezimalzahl, zumindest dann, wenn sie in einem seiner Speicher steht. Und 36 ist für ihn die Summe im Hexadezimalsystem. Wie wir diese Zahl lesen, ist nicht seine Sache.

Das Verfahren funktioniert leider nicht immer so einfach. Sind die beiden Summanden z. B. 19 und 28, dann rechnet er: $19_{16} + 28_{16} = 41_{16}$. Er zeigt uns als Ergebnis 41. Für ihn ist 41 eine Hexadezimalzahl. Er hat natürlich nichts dagegen, daß wir 41 als Dezimalzahl lesen, aber damit sind wir nicht einverstanden, denn wir würden gern 47 als Summe erhalten. Wir müssen den Computer so manipulieren, daß er irgendwie auf die Hexadezimalzahl kommt, die wir – als Dezimalzahl gelesen – erwarten. Wir könnten ihm bei diesem Beispiel außer dem Auftrag „addiere die beiden Summanden 19 und 28“ noch den Zusatzauftrag geben „addiere zu dieser Summe noch 6 und zeige erst dann das Ergebnis an“. Er würde beide Aufträge schnell und zuverlässig ausführen und 47 anzeigen. Jetzt könnten wir die Anzeige als Summe im Dezimalsystem ablesen.

Noch einmal zusammengefaßt: Der Mikroprozessor kennt keinen Befehl für eine dezimale Addition. Wir können unseren Computer aber mit Zusatzbefehlen dazu bringen, daß er uns ein Ergebnis ermittelt, das wir im Dezimalsystem als Summe ablesen können.

Die Addition von 6, wie oben im Beispiel, ist sicher keine Methode, die bei allen Additionen angewendet werden kann. Eine allgemeingültige Methode, die wir im weiteren anwenden werden, besteht in folgender Überlegung: Wir lassen den Computer das machen, was wir zu tun hätten, wenn wir zwei Dezimalzahlen auf dem Umweg über das Hexadezimalsystem addieren müßten. Wir würden die beiden Dezimalzahlen 19_{10} und 28_{10} zunächst in Hexadezimalzahlen umwandeln, dann würden wir 13_{16} und $1C_{16}$ addieren, das Ergebnis $2F_{16}$ würden wir schließlich in eine Dezimalzahl zurückverwandeln: $2F_{16} = 47_{10}$.

In Anlehnung an diesen Rechengang nennen wir die beiden Calls, die uns unser Computer für die beiden hier erforderlichen Umwandlungen zur Verfügung stellt, CALL DEZ-HEX und CALL HEX-DEZ. Von unserer Warte aus sinnvoll, denn mit dem CALL DEZ-HEX erhalten wir z. B. aus der Zahl 28 die Zahl 1C. Vom Computer her gesehen ist diese Bezeichnung für den Call unsinnig. Wenn wir ihm die Zahl 28 eingeben, dann speichert er die Hexadezimalzahl 28 ($28_{16} = 0010\ 1000_2$). Bekommt er nun den Befehl CALL DEZ-HEX, dann berechnet er eigentlich nach einer von uns gegebenen genauen Anweisung zu der Hexadezimalzahl 28 eine kleinere Hexadezimalzahl 1C. Wir werden uns nun darüber keine Gedanken mehr machen. Die Hauptsache ist, der Computer tut das, was für uns nützlich ist.

Bevor wir nun im nächsten Abschnitt ein Programm zur dezimalen Addition schreiben, sollen noch einige Bemerkungen zu den beiden Calls gemacht werden. Die beiden Umwandlungsprogramme sind für vierstellige Zahlen geschrieben. Da $9999_{10} = 270F_{16}$ gilt, ist einerseits 9999, andererseits 270F die größte Zahl, die umgewandelt werden kann.

BESCHREIBUNG	MNEM. CODE	OP. CODE	OPERATION
CALL UMWANDLUNG VOM DEZIMAL- IN DAS HEXADEZIMALSYSTEM	CALL DEZ-HEX	1B	DEZ := <EA> DEZ → HEX <EA> := HEX

Dieser Call verwandelt eine maximal vierstellige Dezimalzahl in die entsprechende Hexadezimalzahl. Vor der Ausführung dieses Calls, vor Beginn der Umwandlung, steht die Dezimalzahl, nach der Umwandlung die Hexadezimalzahl im EA-Register. Soll nur eine zweistellige Dezimalzahl umgewandelt werden, so wird sie in den Akku geschrieben, dann ist aber zusätzlich das Extension-Register zu löschen, d. h. der Inhalt des E-Registers muß dann 0 sein. Die Bezeichnung Dezimalzahl ist hier nicht ganz korrekt. Gemeint ist eine Ziffernfolge, die nur mit den Ziffern 0 bis 9 geschrieben ist und die wir unmittelbar als Dezimalzahl lesen dürfen.

BESCHREIBUNG	MNEM. CODE	OP. CODE	OPERATION
CALL UMWANDLUNG VOM HEXADEZIM- AL- INS DEZIMALSYSTEM	CALL HEX-DEZ	1C	HEX := <EA> HEX → DEZ <EA> := DEZ

Dieser Call verwandelt eine Hexadezimalzahl (maximal $270F_{16}$) in die entsprechende Dezimalzahl. Vor der Ausführung dieses Calls, vor Beginn der Umwandlung steht die Hexadezimalzahl, nach der Umwandlung die Dezimalzahl im EA-Register. Eine zweistellige Hexadezimalzahl kann zur Umwandlung in den Akku geschrieben werden, dann ist aber zusätzlich das Extension-Register zu löschen. Die Bezeichnung Dezimalzahl ist hier nicht ganz korrekt. Das Umwandlungsprogramm ermittelt eine Ziffernfolge mit den Ziffern 0 bis 9, die wir unmittelbar als Dezimalzahl lesen können. Steht eine größere Zahl als $270F_{16}$ im EA-Register und wird dann CALL HEX-DEZ aufgerufen, so erscheint der Hinweis Error (engl. Irrtum, Fehler). Die Hexadezimalzahl 2710_{16} kann z. B. nicht in eine vierstellige Dezimalzahl umgewandelt werden. Diese Anzeige kann durch Betätigung einer beliebigen Funktionstaste beendet werden. Die beiden Calls lassen sich mit folgenden kleinen Programmen testen. Diese Programme sind auch die Grundlage der Umwandlungsprogramme in den Abschnitten 5.6.2 und 5.6.3. Dort haben wir zusätzlich noch einigen Komfort eingebaut.

NR.	MNEM. CODE	ADR.	OP. CODE	BEMERKUNGEN
1.	CALL EIN 4ST	1000	17	EINGABE DER DEZIMALZAHL
2.	LD EA,ZWSP	1001	85 D8	
3.	CALL DEZ-HEX	1003	1B	
4.	CALL ANZ EA	1004	19	UMWANDLUNG IN EINE HEXADEZIMALZAHL
5.	CALL TEST	1005	1E	ANZEIGE DER HEXADEZIMALZAHL

DATENSPEICHER, BEZ.	ADR.	BEMERKUNGEN
ZWSP	FFD8 FFD9	ZWISCHENSPEICHER

NR.	MNEM. CODE	ADR.	OP. CODE	BEMERKUNGEN
1.	CALL EIN 4ST	1000	17	EINGABE DER HEXADEZIMALZAHL
2.	LD EA,ZWSP	1001	85 D8	
3.	CALL HEX-DEZ	1003	1C	UMWANDLUNG IN EINE DEZIMALZAHL
4.	CALL ANZ EA	1004	19	ANZEIGE DER DEZIMALZAHL
5.	CALL TEST	1005	1E	

DATENSPEICHER, BEZ.	ADR.	BEMERKUNGEN
ZWSP	FFD8 FFD9	ZWISCHENSPEICHER

Bei beiden Programmen wird nach dem Start eingegeben. Beim ersten Programm dürfen wir zur Eingabe nur die Ziffern 0 bis 9 verwenden. Beim zweiten Programm darf die eingegebene Zahl höchstens 270F betragen. Nach Betätigung einer Funktionstaste erscheint dann die umgewandelte Zahl in der Anzeige. Neuer Programmstart ist mit **A-D**, **RUN** möglich. Ein letzter Hinweis: Die beiden Calls CALL DEZ-HEX und CALL HEX-DEZ werden ausführlich im 16. Kapitel besprochen. Dort wird dann auch erklärt, wie diese Umwandlungsprogramme aufgebaut sind.

8.5 Addition im Dezimalsystem

Nach den ausführlichen Vorüberlegungen, die wir im letzten Abschnitt angestellt haben, läßt sich nun leicht ein Programm schreiben, mit dem zwei Dezimalzahlen addiert werden können. Wir wollen uns hier darauf beschränken, die Summanden so zu wählen, daß die Summe höchstens 9999 wird. Dann kann auch kein Übertrag auftreten, der im EA-Register und damit in einer vierstelligen Anzeige keinen Platz hätte. Es darf nur noch angemerkt werden, daß auch ein Programm möglich ist, bei dem größere Summen auftreten. Wir werden darauf zurückkommen.

MNEM. CODE	ADR.	OP. CODE
CALL EIN 4ST	1000	17
LD EA,ZWSP	1001	85 D8
CALL DEZ-HEX	1003	1B
ST EA,1.ZAHL	1004	8D E1
CALL EIN 4ST	1006	17
LD EA,ZWSP	1007	85 D8
CALL DEZ-HEX	1009	1B
ADD EA,1.ZAHL	100A	85 E1
CALL HEX-DEZ	100C	1C
CALL ANZ EA	100D	19
CALL TEST	100E	1E

Nach dem Start des Programms kann der erste Summand eingegeben werden. Dann folgt nach **RUN** die Eingabe des zweiten Summanden. Nach erneuter Betätigung der Taste **RUN** erscheint in der Anzeige die Summe. Mit **A-D** und **RUN** kann dann das Programm neu gestartet werden.

8.6 Aufgaben zu Kapitel 8

1. Warum ist der Befehl ST A, E nicht erforderlich?
2. Angenommen, es gäbe den 3-Byte-Befehl mit dem Operationscode „84 BC A5“ nicht. Wie ließe sich mit 1- oder 2-Byte-Befehlen das gleiche Ergebnis erzielen?
3. Mit welchen Inhalten sind die sechs Speicher FFE1 bis FFE6 belegt, wenn nach dem Programm im Abschnitt 8.3 die Summe von A35E und 7F12 berechnet wurde?
- 4.1 Mit welchen beiden Befehlen kann der Inhalt von Speichern FFE1 ins Extension-Register, der Inhalt von Speicher FFE2 in den Akku geladen werden?
- 4.2 Wie können mit drei Befehlen die Inhalte der beiden Speicher FFEA und FFEB vertauscht werden?
5. Zum Programm im Abschnitt 8.5.
- 5.1 In einer Trace (vgl. auch Abschnitt 8.1) sollen die Inhalte nach jedem Befehl festgehalten werden, wenn nach dem angegebenen Programm die Summe $583 + 319$ berechnet wird.

Mnem. Code	Inhalt nach Ausführung des Befehls			
	(E)	(A)	(FFE2)	(FFE1)
CALL EIN 4ST				
LD EA, ZWSP				
CALL DEZ-HEX				
ST EA, 1. Zahl				
CALL EIN 4ST				
LD EA, ZWSP				
CALL DEZ-HEX				
ADD EA, 1. Zahl				
CALL HEX-DEZ				
CALL ANZ EA				
CALL TEST				

- 5.2 Das Programm soll so umgeschrieben werden, daß bei Programmende die beiden eingegebenen Summanden und die Summe in den sechs Speichern FFE1 bis FFE6 abgespeichert sind.
- 5.3 Das Programm soll so umgeschrieben werden, daß mit ihm drei Dezimalzahlen addiert werden. Die Bedingung, daß die Summe höchstens 9999 beträgt, soll beibehalten werden.

9. Subtraktion

9.1 Zunächst die neuen Befehle

Die Abkürzung von „subtrahiere“ für den mnemonischen Code ist SUB (engl.: to subtract = abziehen, subtrahieren).

Subtraktion: (A) – (E)

BESCHREIBUNG	MNEM. CODE	OP. CODE	OPERATION
SUBTRAHIERE DEN INHALT DES EXTENSION-REGISTERS VOM AKKU-INHALT	SUB A, E	178	$\langle A \rangle := \langle A \rangle - \langle E \rangle$; CY/L

Die Angabe CY/L bedeutet, daß beim Subtraktionsbefehl das CY/L-Flag beeinflusst wird. Es wird gesetzt oder gelöscht, je nachdem, ob die Differenz positiv oder negativ ist.

Subtraktions-Befehle mit unmittelbarer Adressierung:

BESCHREIBUNG	MNEM. CODE	OP. CODE	OPERATION
SUBTRAHIERE UNMITTELBAR VOM AKKU-INHALT	SUB A, =ZAHL	IFC XX	$\langle A \rangle := \langle A \rangle - \text{ZAHL}$; CY/L
SUBTRAHIERE UNMITTELBAR VOM INHALT DES EA-REG.	SUB EA, =ZAHL	IBC YY XX	$\langle EA \rangle := \langle EA \rangle - \text{ZAHL}$; CY/L

Als Zahl kann im 2-Byte-Befehl jede Dezimalzahl von 0 bis 255 oder jede Hexadezimalzahl von 00 bis FF gesetzt werden. Im 3-Byte-Befehl kann als Zahl jede Dezimalzahl von 0 bis 65535 oder jede Hexadezimalzahl von 0000 bis FFFF gesetzt werden. Im Operationscode darf für XX natürlich nur eine zweistellige Hexadezimalzahl stehen. Im 3-Byte-Befehl wird mit der Angabe YY XX eine vierstellige Hexadezimalzahl bezeichnet, XX steht für das high order byte (HOB), YY steht für das low order byte (LOB) dieser vierstelligen Hexadezimalzahl. Die Angabe CY/L bedeutet, daß beim Subtraktionsbefehl das CY/L-Flag beeinflusst wird. Es wird gesetzt oder gelöscht, je nachdem, ob die Differenz positiv oder negativ ist.

Subtraktions-Befehle mit direkter Adressierung:

BESCHREIBUNG	MNEM. CODE	OP. CODE	OPERATION
SUBTRAHIERE AUS DEM SPEICHER VOM AKKU-INHALT (DIREKTE ADRESSIERUNG)	SUB A, BEZ	IFD XX	$\text{ADR} := \text{FF00} + \text{XX}$; $\langle A \rangle := \langle A \rangle - \langle \text{ADR} \rangle$; CY/L
SUBTRAHIERE AUS DEM SPEICHER VOM INHALT DES EA-REGISTERS (DIREKTE ADRESSIERUNG)	SUB EA, BEZ	IBD XX	$\text{ADR} := \text{FF00} + \text{XX}$; $\langle EA \rangle := \langle EA \rangle - \langle \text{ADR} + 1, \text{ADR} \rangle$; CY/L

BEZ steht für die Bezeichnung eines Speichers oder eines Speicherinhalts. Dieser Speicher muß eine Adresse zwischen FF00 und FFFF haben. Das zweite Byte dieser Adresse ist beim Operationscode mit XX bezeichnet. Die vollständige Adresse ADR ergibt sich dann als Summe $\text{FF00} + \text{XX}$. Beim Befehl SUB A, BEZ wird der Inhalt des Speichers mit der Adresse ADR vom Akkuinhalt abgezogen. Beim Befehl SUB EA, BEZ wird der Inhalt der beiden Speicher mit den Adressen $\text{ADR} + 1$ und ADR als vierstellige Hexadezimalzahl vom Inhalt des EA-Registers abgezogen. Die Angabe CY/L bedeutet, daß beim Subtraktionsbefehl das CY/L-Flag beeinflusst wird. Es wird gesetzt oder gelöscht, je nachdem, ob die Differenz positiv oder negativ ist.

Beispiele:

INMEM.CODE	OP.CODE	BEMERKUNGEN
ISUB A,=20	IFC 14	(A):=(A)-14
ISUB A,=020	IFC 20	(A):=(A)-20
ISUB EA,=300	IBC 2C 01	(EA):=(EA)-120
ISUB EA,=01	IBC 01 00	(EA):=(EA)-1
ISUB A,1.ZAHL	IFD E1	1.ZAHL:=(FFE1);(A):=(A)-1.ZAHL
ISUB EA,2.ZAHL	IBD E5	2.ZAHL:=(FFE6,FFE5); (EA):=(EA)-2.ZAHL

Wir könnten jetzt der Reihe nach alle Subtraktionsbefehle in kleinen Programmen durchprobieren – so wie wir es bei der Addition gemacht haben. Wer will, soll es gerne tun. Der Eigentätigkeit sind hier keine Grenzen gesetzt. Wer es sich bequemer machen will, braucht nur in den Programmen der Kapitel 7 und 8 jeweils den vorkommenden Additionsbefehl durch den entsprechenden Subtraktionsbefehl zu ersetzen. In der folgenden Übersicht sind die Additions- und Subtraktionsbefehle zusammengestellt:

Subtraktion		Addition	
INMEM.CODE	OP.CODE	INMEM.CODE	OP.CODE
ISUB A,E	I78	IADD A,E	I70
ISUB A,=ZAHL	IFC XX	IADD A,=ZAHL	IF4 XX
ISUB EA,=ZAHL	IBC YY XX	IADD EA,=ZAHL	IB4 YY XX
ISUB A,BEZ	IFD XX	IADD A,BEZ	IF5 XX
ISUB EA,BEZ	IBD XX	IADD EA,BEZ	IB5 XX

Wir sehen: Um aus dem Additionsbefehl einen Subtraktionsbefehl zu machen, braucht jeweils nur das erste Byte im Operationscode um 8 erhöht zu werden. Hinter der Wahl des Operationscodes steckt ein System (vgl. Abschnitt 7.2). Hier soll zunächst nur ein Subtraktionsprogramm angegeben werden. Mit seiner Hilfe lassen sich zweistellige Hexadezimalzahlen über die Tastatur eingeben und subtrahieren.

INR.	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	ICALL EIN 2ST	1000	I16	DIE 1.ZAHL, DER MINUEND, WIRD EINGEGEBEN
2.	ILD A,ZWSP	1001	IC5 D8	
3.	IST A,1.ZAHL	1003	ICD E0	
4.	ICALL EIN 2ST	1005	I16	DIE 2.ZAHL, DER SUBTRAHEND, WIRD EINGEGEBEN
5.	ILD A,ZWSP	1006	IC5 D8	
6.	IST A,2.ZAHL	1008	ICD E1	
7.	ILD A,1.ZAHL	100A	IC5 E0	DIE DIFFERENZ WIRD ERMITTELT ...UND ANGEZEIGT
8.	ISUB A,2.ZAHL	100C	IFD E1	
9.	ICALL ANZ A	100E	I18	
10.	ICALL TEST	100F	I1E	

DATENSPEICHER, BEZ.	ADR.	BEMERKUNGEN
ZWSP	IFFD8	ZWISCHENSPEICHER
1.ZAHL	IFFE0	MINUEND
2.ZAHL	IFFE1	SUBTRAHEND

Nach Programmstart zeigt der Computer eine 0 an, er erwartet die Eingabe der ersten Zahl, von der etwas abgezogen werden soll (Minuend). Betätigen wir dann die Taste **RUN**, so können wir die zweite Zahl eingeben, die Zahl, die abgezogen werden soll (Subtrahend). Nach erneuter Betätigung der Taste **RUN** erscheint in der Anzeige die Differenz. Ein neuer Programmstart ist mit **A↔D**, **RUN** möglich.

Beispiele:

Nr.	Subtraktionsaufgabe	angezeigte Differenz
1	F – 7	08
2	20 – B	15
3	A3 – 27	7C
4	EB – A7	44
5	2B – E7	44
6	E7 – 2B	BC
7	7 – F	F8
8	0 – 1	FF

Bei diesen Beispielen ergeben sich eine Reihe von Fragen, die wir in diesem Kapitel noch klären müssen.

- 1) Wir können wir die Differenz zweier Hexadezimalzahlen schriftlich berechnen? Wir wollen ja schließlich nachprüfen können, ob der Computer richtig rechnet.
- 2) Im 4. und 5. Beispiel ergibt sich jeweils die Anzeige 44. Die beiden Anzeigen können aber sicher nicht dieselbe Bedeutung haben, denn EB ist größer als A7, 2B ist aber kleiner als E7.
- 3) Beim 8. Beispiel erwarten wir als Ergebnis – 1. Nach dem Computer beträgt die Differenz FF. Ist das dasselbe? Was versteht der Mikroprozessor unter negativen Zahlen?

9.2 Schriftliche Subtraktion von Hexadezimalzahlen

Wir können auch im Hexadezimalsystem (und in jedem anderen Zahlensystem) so subtrahieren, wie wir es vom Dezimalsystem gewohnt sind. Die Subtraktion wird dabei stellenweise – von rechts beginnend – durchgeführt. An das dabei auftretende „Borgen“ haben wir uns im Dezimalsystem gewöhnt.

Im 1. Beispiel gibt es noch keine Probleme. Im Hexadezimalsystem gilt: 7 + 8 = F. Also ist auch F – 7 = 8

F |
7 | –
8

Im 2. Beispiel können wir in der letzten Stelle nicht ohne weiteres B von 0 abziehen. Wir „borgen“ uns von der Sechzehnerstelle eine 1. Jetzt können wir B von 10₁₆ subtrahieren. Wir müssen uns nur merken, daß wir in der nächsten Stelle zusätzlich eine 1 von der gegebenen Ziffer 2 subtrahieren müssen. 10 – B = 5; 2 – 1 = 1

2 0 |
B | –
? |
2 0 |
1 B | –
1 5

Im 3. Beispiel ganz entsprechend:

13 – 7 = C
A – 2 – 1 = 7
Bei der Rechnung für eine einzelne Stelle werden wir wahrscheinlich heimlich wieder das Dezimalsystem zu Hilfe nehmen:
13₁₆ – 7₁₆ = 19₁₀ – 7₁₀
= 12₁₀
= C₁₆

A 3 |
2 7 | –
1 |
C |
A 3 |
2 7 | –
1 |
7 C

Im 4. und im 6. Beispiel erhalten wir auch dieselben Ergebnisse wie unser Computer:

4.

E	B
A	7
4	4

–

6.

E	7
2	B
1	C

–

Dieses Subtraktionsverfahren lässt sich auch bei größeren Zahlen durchführen – jedenfalls dann, wenn der Minuend (die Zahl, von der abgezogen wird) größer ist als der Subtrahend (die Zahl, die abgezogen wird):

Was ist 20F5EC – 82DB1?

2	0	F	5	E	C	
1	8	2	D	B	1	–
1	8	C	8	3	B	

Nun zu den drei noch nicht besprochenen Beispielen von 9.1. Die Aufgabe im 5. Beispiel hieß: Es soll von der Hexadezimalzahl 2B die größere Hexadezimalzahl E7 subtrahiert werden. So ohne weiteres kann das nicht funktionieren, denn man kann von 2B eben höchstens 2B wegnehmen. Wie würden wir die Aufgabe im Dezimalsystem lösen?

$2B_{16} = 2 \cdot 16_{10} + 11_{10}$
 $= 43_{10}$

$E7_{16} = 14 \cdot 16_{10} + 7_{10}$
 $= 231_{10}$

Was ist im Dezimalsystem 43 – 231? Das Ergebnis ist sicher negativ. Um es zu bestimmen, würden wir 43 von 231 abziehen und dann das Ergebnis mit einem Minuszeichen versehen: 231 – 43 = 188; 43 – 231 = – 188. Da $188_{10} = 11 \cdot 16_{10} + 12_{10} = BC_{16}$ ist, hätte der Computer im 5. Beispiel also „– BC“ erhalten müssen, berechnet hat er aber 44. Ebenso hat er im 7. Beispiel F8 statt – 8 und im 8. Beispiel FF statt – 1 berechnet. Wir werden noch sehen, daß wir mit den vom Computer angegebenen Ergebnissen durchaus etwas anfangen können. Es wird vielleicht schon aufgefallen sein, daß sich BC und 44, daß sich 8 und F8, daß sich 1 und FF jeweils zur Hexadezimalzahl 100 ergänzen. Wir werden darauf zurückkommen.

9.3 Das CY/L-Flag und die Subtraktion

Im Abschnitt 9.1 hatte der Computer im 4. und 5. Beispiel jeweils 44 berechnet. Natürlich hat er gemerkt, daß das Ergebnis einmal positiv, einmal negativ zu werten ist. Er hat das auch mitgeteilt, wir haben nur nicht genau hingesehen. Das CY/L-Flag war nämlich einmal gesetzt und einmal gelöscht, wir haben nur dieses Flag bisher noch nicht beachtet.

Wir wollen unser Programm also wieder so erweitern, daß uns der Inhalt des CY/L-Flags zusätzlich angezeigt wird. In den Beispielen, bei denen sich ein positives Ergebnis ergibt, wird das richtige Ergebnis angezeigt, und die Leuchtdiode leuchtet. Sie zeigt an: alles in Ordnung! In den drei Beispielen (5., 7. und 8.), bei denen der Subtrahend größer war als der Minuend, leuchtet die rote Leuchtdiode nicht.

NR.	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.		1000	39 00	DIE ROTE LEUCHTDIODE WIRD (GEBEHNENFALLS GELOESCHT. ERKLÄRUNG FOLGT SPÄTER
2.	ICALL EIN 2ST	1002	16	BEINGABE MINUEND
3.	ILD A,ZWSP	1003	1C5 D8	
4.	IST A,MINU	1005	1CD E0	
5.	ICALL EIN 2ST	1007	116	BEINGABE SUBTRAHEND
6.	ILD A,ZWSP	1008	1C5 D8	
7.	IST A,SUBT	100A	1CD E1	
8.	ILD A,MINU	100C	1C5 E0	BERECHNUNG DER DIFFERENZ
9.	SUB A,SUBT	100E	1FD E1	
10.	IST A,DIFF	1010	1CD E2	
11.		1012	106	DER INHALT DES CY/L-FLAGS WIRD NACH DER SUBTRAKTION IN DER ROTEN LEUCHTDIODE ANGEZEIGT.
12.		1013	104 80	
13.		1015	13C	
14.		1016	13C	
15.		1017	13C	
16.		1018	13C	ERKLÄRUNG FOLGT SPÄTER
17.		1019	107	
18.	ILD A,DIFF	101A	1C5 E2	ANZEIGE DER DIFFERENZ
19.	ICALL ANZ A	101C	118	
20.	ICALL TEST	101D	11E	

DATENSPEICHER, BEZ.	ADR.	BEMERKUNGEN
ZWSP	1FFD8	ZWISCHENSPEICHER
MINU	1FFE0	MINUEND
SUBT	1FFE1	SUBTRAHEND
DIFF	1FFE2	DIFFERENZ

Hier ist das angezeigte Ergebnis mit Vorsicht zu behandeln. Wie macht das der Mikroprozessor? Er rechnet sicher – ob positives oder negatives Ergebnis – mit Hilfe derselben elektronischen Schaltung. Er wird nicht von jeder Subtraktionsaufgabe erst die beiden Zahlen vergleichen und je nach dem Ergebnis dieses Vergleichs einmal so, ein anderes Mal so rechnen. Eine erste, nicht ganz präzise Vorstellung über dieses Subtraktionsverfahren könnte folgendermaßen aussehen:

Bei der Subtraktion zweistelliger Hexadezimalzahlen wird der Minuend (die Zahl, von der abgezogen wird) grundsätzlich um 256 (100_{16}) erhöht. Dann kann es bei der Subtraktion in keinem Fall Schwierigkeiten geben. Die beiden letzten Stellen der Differenz werden in den Akku, die drittletzte Stelle (1 oder 0) kommt ins CY/L-Register. Wie sieht das in unseren Beispielen aus?

Statt EB – A7 wird gerechnet 1 EB – A7:

1	EB	
1	A7	–
1	44	

Von diesem Ergebnis kommt die zweistellige Hexadezimalzahl 44 in den Akku. Die 1 wird in das CY/L-Flag gebracht und signalisiert: Das Ergebnis ist positiv. Wir sehen: An den bisher als richtig erkannten Rechnungen ändert sich überhaupt nichts.

1	2	B	7	
1	E	7	C	–
(0)	4	4	4	

Statt 2B – E7 wird gerechnet 1 2B – E7: Die zweistellige Hexadezimalzahl 44 kommt wieder in den Akku. Jetzt wird aber eine 0 in das CY/L-Flag gebracht.

Nach dieser Vorstellung können wir verstehen, warum die rote Leuchtdiode einmal aufleuchtet, einmal dunkel bleibt. Wir sehen den Zusammenhang zwischen positivem und negativem Ergebnis und Inhalt des CY/L-Flags. Wir verstehen auch, wie der Computer im 5., 7. und 8. Beispiel im Abschnitt 9.1 zu seinen Ergebnissen kommt.

$$7. \begin{array}{r} 1 \rightarrow 0 \rightarrow 7 \\ 1 \downarrow 1 \downarrow \\ (0) | F \quad 8 \end{array} - \quad 8. \begin{array}{r} 1 \rightarrow 0 \rightarrow 0 \\ 1 \downarrow 1 \downarrow \\ (0) | F \quad F \end{array} -$$

Offen ist noch die Frage, wie wir den Akkuinhalt bei negativer Differenz interpretieren sollen. Ein neues Problem ist aufgetaucht: Wenn zum Minuenden 100_{16} addiert wird, so ergibt sich eine dreistellige Hexadezimalzahl (eine neunstellige Dualzahl). Wie kann ein 8-Bit-Akkumulator mit dieser Zahl arbeiten? Auch diese Frage wird später noch beantwortet.

Jetzt wollen wir doch noch gemeinsam einen weiteren Subtraktionsbefehl erproben. Wir wollen uns vergewissern, daß der Mikroprozessor auch bei vierstelliger Subtraktion in gleicher Weise arbeitet.

NR.	INSTR.	CODE	ADR.	OP. CODE	BEMERKUNGEN
1.			1000	39 00	!DIE ROTE LEUCHTDIODE WIRD !GEGEBENENFALLS GELOESCHT. !ERKLÄRUNG FOLGT SPÄTER
2.	ICALL	EIN 4ST	1002	17	!EINGABE MINUEND
3.	ILD	EA,ZWSP	1003	85 D8	
4.	IST	EA,MINU	1005	8D E0	
5.	ICALL	EIN 4ST	1007	17	!EINGABE SUBTRAHEND
6.	ILD	EA,ZWSP	1008	85 D8	
7.	IST	EA,SUBT	100A	8D E2	
8.	ILD	EA,MINU	100C	85 E0	!BERECHNUNG DER DIFFERENZ
9.	SUB	EA,SUBT	100E	8D E2	
10.	IST	EA,DIFF	1010	8D E4	
11.			1012	06	!DER INHALT DES CY/L-FLAGS
12.			1013	04 00	!WIRD NACH DER SUBTRAKTION
13.			1015	13C	!AN DER ROTEN LEUCHTDIODE
14.			1016	13C	!ANGEZEIGT.
15.			1017	13C	
16.			1018	13C	!ERKLÄRUNG FOLGT SPÄTER
17.			1019	107	
18.	ILD	EA,DIFF	101A	85 E4	!ANZEIGE DER DIFFERENZ
19.	ICALL	ANZ EA	101C	119	
20.	ICALL	TEST	101D	11E	

DATENSPEICHER, BEZ.	ADR.	BEMERKUNGEN
ZWSP	1FFD8 1FFD9	!ZWISCHENSPEICHER
MINU	1FFE0 1FFE1	!DER MINUEND WIRD VIERSTELLIG !IN DEN SPEICHERN 1FFE0 (LOB) !UND 1FFE1 (HOB) ABGESPEICHERT
SUBT	1FFE2 1FFE3	!SUBTRAHEND
DIFF	1FFE4 1FFE5	!DIFFERENZ

Wenn wir mit diesem Programm wieder die Beispiele aus Abschnitt 9.1 durchrechnen, stellen wir fest, daß die positiven Ergebnisse lediglich durch voranstehende Nullen zur vierstelligen Hexadezimalzahl werden. Statt 7C erhalten wir jetzt 007C. Die rote Leuchtdiode zeigt nach wie vor an: positiv oder nicht. Bei den Subtraktionsaufgaben mit negativer Differenz wird die bisherige Anzeige vorne durch FF ergänzt und so zu einer vierstelligen Anzeige. Unsere Vorstellung über das Subtraktionsverfahren im Mikroprozessor müssen wir jetzt geringfügig variieren. Der Minuend muß vor der Subtraktion vierstelliger Hexadezimalzahlen grundsätzlich um $65\,536\ (10000_{16})$ erhöht werden. Dann beschreibt unser Modell wieder alle Ergebnisse richtig; z. B. statt $2\,B - E\,7$ wird gerechnet $1002\,B - E\,7$:

$$\begin{array}{r} 1 \rightarrow 0 \rightarrow 0 \rightarrow 2 \quad B \\ 1 \downarrow 1 \downarrow 1 \downarrow \\ (0) | F \quad F \quad 4 \quad 4 \end{array} \quad \begin{array}{l} \text{Die vierstellige Hexadezimalzahl FF44} \\ \text{kommt ins EA-Register. Die 0 wird ins} \\ \text{CY/L-Register gebracht.} \end{array}$$

9.4 Das Rechnen mit Komplementen

Beim Mikroprozessor wird die Subtraktion auf die Addition zurückgeführt. Auch wenn uns diese Feststellung zunächst sehr überraschend erscheint, wir werden zugeben müssen, daß eine solche Methode sehr vorteilhaft ist. Für die Subtraktion kann dann – vielleicht mit geringfügigen Ergänzungen – das Addierwerk benutzt werden, das ohnehin im Mikroprozessor vorhanden ist (vgl. Abschnitt 6.3). Und jetzt zum Verfahren: Es wird nicht der Subtrahend vom Minuenden abgezogen, es wird zum Minuenden das Komplement des Subtrahenden addiert. Das Verfahren ist so allgemeingültig, daß es in jedem Zahlensystem gilt. Wir werden es uns zunächst im Dezimalsystem verständlich machen.

9.4.1 Neuner- und Zehnerkomplemente im Dezimalsystem

Wir wollen die Differenz $11 - 7$ berechnen. Natürlich wissen wir sofort, daß sich 4 ergeben muß. Wir erhalten das Ergebnis auch auf folgendem Weg: Da die größte der beiden Zahlen 11 und 7 zweistellig ist, bilden wir das zweistellige Neunerkomplement vom Subtrahenden 7. Complement ist das englische Wort für Ergänzung. Wir ergänzen die zweistellige Zahl 07 stellenweise zu 9 und erhalten 92 als zweistelliges Neunerkomplement von 07: $0 + 9 = 9$, $7 + 2 = 9$. Aus diesem Neunerkomplement erhalten wir durch Addition von 1 das gesuchte Zehnerkomplement von 7: $92 + 1 = 93$. 92 ergänzt 7 zu 99, 93 ergänzt 7 zu einer Zehnerpotenz, zu 100. Jetzt addieren wir den Minuenden 11 und das ermittelte Zehnerkomplement von 7, nämlich 93.

$$\begin{array}{r} 11 \\ 93 \\ \hline 104 \end{array} +$$

04 ist das gesuchte Ergebnis der Subtraktionsaufgabe. Daß wir bei der Addition eine um 100 zu große Summe erhalten, ist bei diesem Verfahren verständlich. Wir haben ja nicht 7 subtrahiert, sondern $93 (= 100 - 7)$ addiert.

Das Verfahren sieht zunächst sehr umständlich aus. Bei weiteren Beispielen werden wir aber sehen, daß es gar nicht so kompliziert ist: Wie läßt sich die Differenz $231 - 43$ berechnen? Wir rechnen mit dreistelligen Zahlen. Das dreistellige Neunerkomplement der Dezimalzahl 43 ist 956. Das Zehnerkomplement ist dann 957. Wir addieren.

$$\begin{array}{r} 231 \\ 957 \\ \hline 1188 \end{array} +$$

188 ist die gesuchte Differenz. Die Summe bei der Addition ist um 1000 zu groß. Wir haben nicht 43 subtrahiert, sondern $957 (= 1000 - 43)$ addiert.

Was passiert, wenn der Subtrahend größer ist als der Minuend?

Beispiel: $43 - 231$. Das dreistellige Neunerkomplement von 231 ist 768. Das Zehnerkomplement ist dann 769.

$$\begin{array}{r} 043 \\ 769 \\ \hline 812 \end{array} +$$

Im vorigen Beispiel erhielten wir eine Summe größer als 1000. 1000 mußten wir zum Schluß abziehen. Wie machen wir das hier? Auch hier ist die Summe um 1000 zu groß, denn wir haben nicht 231 abgezogen, sondern 769

($= 1000 - 231$) addiert. Aus der Tatsache, daß die Summe kleiner als 1000 ist, entnehmen wir: Die Differenz ist negativ. Außerdem kennen wir auch das Ergebnis: $43 - 231 = 812 - 1000$. Entweder wir begnügen uns hiermit oder wir geben das Ergebnis anders an: Minuszeichen und Zehnerkomplement von 812, also: -188 .

Wir hätten uns hier das Neunerkomplement (Ergänzung zu 99, bzw. zu 999) auch ersparen können. Wir hätten gleich das Zehnerkomplement (Ergänzung zu 100, bzw. zu 1000) bilden können. Beim Dualsystem werden sich diese Vorüberlegungen aber als nützlich erweisen.

9.4.2 Fünfzehner- und Sechzehnerkomplemente im Hexadezimalsystem

Wir gehen ganz entsprechend so vor wie im Dezimalsystem. Beispiel: $EB - A7$. Wir rechnen zweistellig. Das zweistellige Fünfzehnerkomplement (Ergänzung jeweils zu F) von A7 ist 58_{16} : $A + 5 = F$, $7 + 8 = F$. Das Sechzehnerkomplement (Ergänzung zu 100_{16}) ergibt sich dann durch Addition von 1 zum Fünfzehnerkomplement: $58_{16} + 1_{16} = 59_{16}$. Jetzt addieren wir:

$$\begin{array}{r} E \ B \\ 5 \ 9 \\ \hline 1 \ 4 \ 4 \end{array} +$$

Die Summe bei der Addition ist um die Hexadezimalzahl 100_{16} zu groß. Wir haben nicht A7 subtrahiert, sondern 59 ($= 100 - A7$) addiert. Die 1 deutet darauf hin, daß die Differenz positiv ist. Diese positive Differenz ist 44_{16} .

Das nächste Beispiel (vgl. Abschnitt 9.2) zeigt, daß dieses Verfahren für beliebig große Hexadezimalzahlen verwendbar ist: $20F5EC - 82DB1$. Wir rechnen sechsstellig. Das sechsstellige Fünfzehnerkomplement von 082DB1 ist F7D24E. Das zu addierende Sechzehnerkomplement ist dann F7D24F.

$$\begin{array}{r} 2 \ 0 \ F \ 5 \ E \ C \\ F \ 7 \ D \ 2 \ 4 \ F \\ \hline 1 \ 1 \ 8 \ C \ 8 \ 3 \ B \end{array} +$$

Aus der ersten 1 in der Summe folgern wir: Die Differenz ist positiv. Die Differenz ist $18C73B$.

Im folgenden Beispiel, das wir schon häufiger betrachtet haben, ist der Subtrahend kleiner als der Minuend: $2B - E7$. Das zweistellige Fünfzehnerkomplement von E7 ist 18_{16} . Das Sechzehnerkomplement ist dann 19_{16} . Die Addition ergibt 44_{16} .

$$\begin{array}{r} 2 \ B \\ 1 \ 9 \\ \hline 4 \ 4 \end{array} +$$

Da das Ergebnis dieser Addition kleiner als 100_{16} ist, entnehmen wir: Die Differenz ist negativ. Die Differenz beträgt $44_{16} - 100_{16}$. Wenn wir wollen, können wir uns mit diesem Ergebnis begnügen. Oder wir berechnen das Sechzehnerkomplement von 44_{16} ($BB + 1 = BC$) und geben als Ergebnis $-BC$ an.

Es wird niemanden die Feststellung überraschen, daß unser Mikroprozessor auf diese Weise subtrahiert. Wie er die Komplemente bildet, soll gleich noch gesagt werden. Uns wird auffallen sein, daß wir jetzt in allen Beispielen von Abschnitt 9.1 zu befriedigenden Erklärungen gekommen sind, auch in den drei Beispielen mit negativer Differenz (die rote Leuchtdiode leuchtet nicht!):

Nr.	Subtraktionsaufgabe	angezeigte Differenz	Ergebnis
5	$2B - E7$	44	$44 - 100$ oder $-BC$
7	$7 - F$	F8	$F8 - 100$ oder -08
8	$0 - 1$	FF	$FF - 100$ oder -01

Drei Bemerkungen noch:

- 1) Das CY/L-Flag hat hier gar keine andere Funktion als bei der Addition. Auch hier wird schließlich nur addiert. Das CY/L-Flag wird gesetzt (1) oder gelöscht (0), je nachdem, ob ein Übertrag entsteht oder nicht.
- 2) Der Mikroprozessor bildet zwei- oder vierstellige Komplemente, in Abhängigkeit von der Stellenzahl, mit der er rechnet.
- 3) Bei der Berechnung des Sechzehnerkomplements einer Hexadezimalzahl könnten wir uns auch vom Computer helfen lassen. Wir würden doch die Zahl zur geeigneten Sechzehnerpotenz ergänzen oder die Differenz: Sechzehnerpotenz - Zahl berechnen müssen. Wir könnten den Computer einfach die Zahl von 0 subtrahieren lassen. Das Ergebnis ist das gesuchte Komplement. Um das CY/L-Flag brauchen wir uns in diesem Fall nicht zu kümmern.

9.4.3 Einer- und Zweierkomplemente im Dualsystem

Auch im Dualsystem verläuft das Verfahren prinzipiell in der gleichen Weise. Hier ist die Komplementbildung sogar noch einfacher. Zur Bildung des Einerkomplements des Subtrahenden müssen wir stellenweise zu 1 ergänzen: $1 + 0 = 1$, $0 + 1 = 1$. Wo der Subtrahend eine 1 hat, hat das Komplement eine 0; wo der Subtrahend eine 0 hat, hat das Komplement eine 1. Aus dem so bestimmten Einerkomplement ergibt sich durch Addition von 1 dann das Zweierkomplement.

Beispiel: $1101 - 100$. Wir rechnen vierstellig. Das vierstellige Einerkomplement von 0100 ist 1011. Das Zweierkomplement ist $1011 + 1 = 1100$. Wir addieren:

Minuend:

Zweierkomplement von 0100:

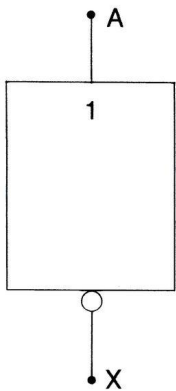
$$\begin{array}{r} 1 \ 1 \ 0 \ 1 \\ 1 \ 1 \ 0 \ 0 \\ \hline 1 \ 1 \ 0 \ 0 \ 1 \end{array} +$$

Wir haben von 1101 nicht 100 subtrahiert, sondern wir haben zu 1101 das Zweierkomplement 1100 ($= 10000 - 100$) addiert. Wir erhalten bei der Addition ein um 10000 zu großes Ergebnis. Der Übertrag, der ganz vorn im Additionsergebnis auftritt, bedeutet wieder: Die Differenz ist positiv. Die Differenz beträgt 1001.

Dieses eine Beispiel soll hier genügen. Weitere Beispiele würden immer wieder in der gleichen Weise ablaufen.

9.5 Und so subtrahiert der Mikroprozessor wirklich

Eigentlich ist nach dem bisher Gesagten alles klar. Um eine Differenz zu berechnen, addiert der Mikroprozessor ein Zweierkomplement. Dazu hat er zunächst ein Einerkomplement zu bilden. Bei der Addition muß er dann zusätzlich 1 addieren. Elektronisch sind diese Aufgaben leicht zu realisieren.



Das nebenstehende Schaltsymbol stellt ein Negationsgatter dar. Es hat einen Eingang und einen Ausgang. Der Kreis am Ausgang deutet die Negation an. Am Eingang darf als Signal nur ein L- oder ein H-Signal liegen. L-Signal bedeutet niedrige Spannung (engl. low = niedrig, z. B. zwischen 0V und 1V), H-Signal bedeutet hohe Spannung (engl. high = hoch, z. B. zwischen 3V und 5V). Am Ausgang liegt dann stets das entgegengesetzte (das negierte) Signal X. Folgende Verknüpfungstabelle zeigt kurz, wie X und A beim Negationsgatter zusammenhängen:

A	X
L	H
H	L

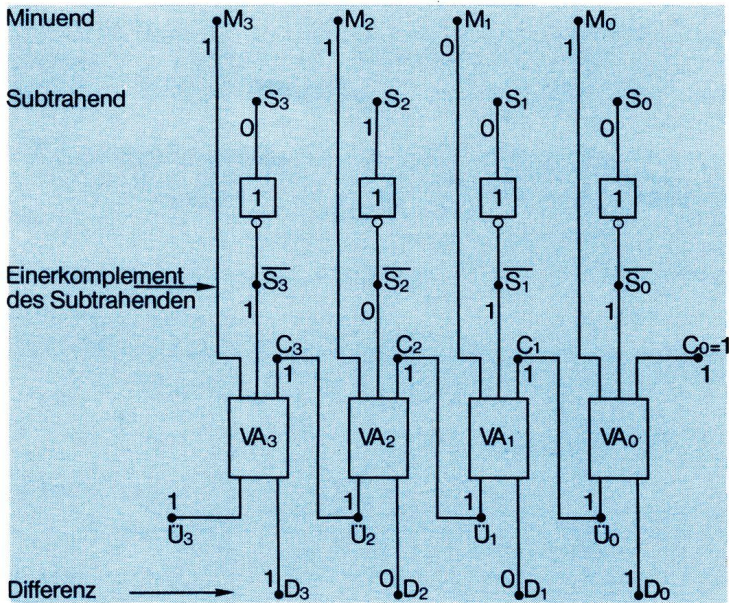
Man schreibt auch: $X = \bar{A}$ (sprich: X gleich A quer). X ist die Negation von A.

Da wir uns mit einer Rechenmaschine beschäftigen, werden wir hier nicht L und H, sondern wieder Dualziffern 0 und 1 verwenden:

A	X
0	1
1	0

Ein Negationsgatter bildet also zu einer einstelligen Dualzahl das einstellige Einerkomplement. Zur Bildung von mehrstelligen Einerkomplementen benötigt man eben entsprechend viele Negationsgatter.

Im folgenden Bild ist das vierstellige duale Addierwerk von Abschnitt 6.3 so ergänzt, daß vierstellige Subtraktionsaufgaben gelöst werden können.



Die vier Negationsgatter bilden zu dem gegebenen Subtrahenden das Einerkomplement des Subtrahenden. Der Halbaddierer des vierstelligen Addierwerks wurde durch einen Volladdierer ersetzt. Mit seiner Hilfe wird bei jeder Subtraktionsaufgabe zusätzlich 1 addiert. Dadurch wird im Prinzip das Einerkomplement zum Zweierkomplement erhöht. Zum Vergleich sind die Dualziffern vom Beispiel im letzten Abschnitt (vgl. 9.4.3) zusätzlich an den Schaltplan angeschrieben. Liefert der Volladdierer VA₃ einen Übertrag $\bar{U}_3 = 1$, so kann die vierstellige positive Differenz an den vier Ausgängen der Volladdierer D₃, D₂, D₁ und D₀ abgelesen werden. Ist $\bar{U}_3 = 0$, so ist das Ergebnis negativ. An den Ausgängen D₃, D₂, D₁ und D₀ steht jetzt das um 10 000 zu große Ergebnis der Subtraktionsaufgabe oder das Zweierkomplement des negativen Ergebnisses.

Wird beim Mikroprozessor statt mit vierstelligen mit acht- oder sechzehnstelligen Dualzahlen gerechnet, so steht das Ergebnis im Akku (8 Bit) oder im EA-Register (16 Bit), der Übertrag (\bar{U} , bzw. \bar{U}_8) steht dann im CY/L-Register (1 Bit). Dort prüfen wir dann nach, ob die Differenz positiv oder negativ ist, dort prüfen wir nach, wie wir den Inhalt im Akku oder im EA-Register zu interpretieren haben.

9.6 Negative Zahlen

Nach so viel Theorie zunächst ein kleines Programm:

INR.	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	LD A,=016	1000	1C 16	$\langle A \rangle := 16$
2.	SUB A,=1	1002	1C 01	$\langle A \rangle := \langle A \rangle - 1$
3.	CALL ANZ A	1004	18	ANZEIGE VON $\langle A \rangle$
4.		1005	24 01 10	

Wir geben das Programm ein und starten es. In der Anzeige erscheint 15. Im ersten Befehl wurde die Hexadezimalzahl 16 in den Akku geladen. Dieser Akkuinhalt wurde im zweiten Befehl durch Subtraktion um 1 erniedrigt. Der dritte Befehl ruft die Anzeige des Akkuinhalts auf. Bis hier ist alles klar. Wenn wir jetzt wieder die Taste **RUN** (oder eine andere Taste außer **A↔D**) betätigen, erscheint die Hexadezimalzahl 14. Es wurde wieder 1 subtrahiert. Der vierte Befehl muß also bewirken, daß unser Programm wieder beim zweiten Befehl fortgesetzt wird. Diesen Befehl werden wir im nächsten Kapitel besprechen. Hier wollen wir uns zunächst mit einem anderen Aspekt unseres kleinen Programms beschäftigen. Bei wiederholter Betätigung der **RUN**-Taste wird jeweils die angezeigte Hexadezimalzahl um 1 erniedrigt. Schließlich kommen wir zur Anzeige 00. Und jetzt? Die nächste Anzeige ist FF. Wenn wir nicht auf den Wert des CY/L-Flags achten, müssen wir FF auch als - 1 bezeichnen können. FF ist auch der Akkuinhalt, der bei Addition von 1 den Akkuinhalt 00 ergibt. Ebenso können wir FE als - 2, FD als - 3, ... F1 als - F usw. bezeichnen. Der Mikroprozessor kennt keine anderen Akkuinhalte als die 256 verschiedenen zweistelligen Hexadezimalzahlen von 00 bis FF (bzw. die entsprechenden achtstelligen Dualzahlen). Wie wir diese Akkuinhalte interpretieren, ist unsere Sache. Wir sind selbst dafür verantwortlich, daß wir nur sinnvolle Interpretationen wählen.

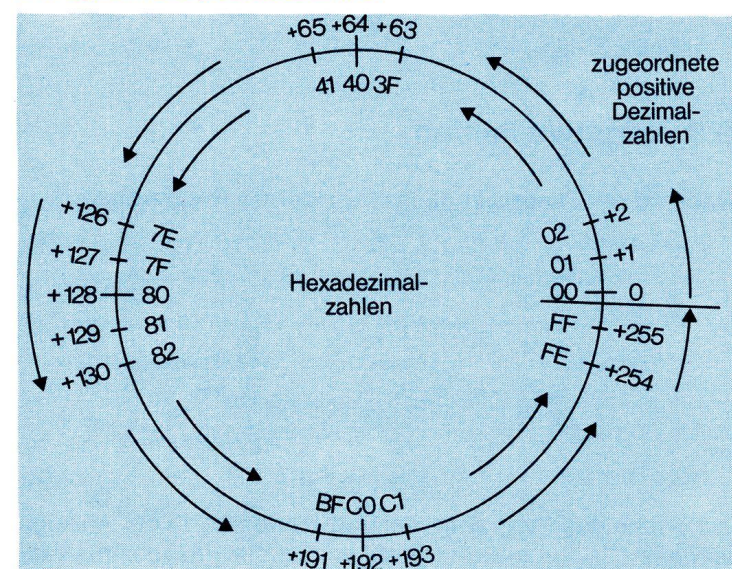
Zum letztenmal wollen wir uns die drei Subtraktionsaufgaben aus Abschnitt 9.1, vgl. auch Abschnitt 9.4.2, ansehen, bei denen die Differenz negativ wurde: Der Inhalt des CY/L-Flags war 0, die rote Leuchtdiode leuchtete nicht. Bei der in diesem Abschnitt angesprochenen Interpretationsmethode stellt der Akkuinhalt dann unmittelbar das negative Ergebnis dar:

Beispiel 8: $0 - 1 = FF = -1$
 Beispiel 7: $7 - F = F8 = -8$
 Beispiel 5: $2B - E7 = 44 = -BC$

Festhalten wollen wir an dieser Stelle noch, wie sich eine negative Zahl berechnen läßt. Wir bilden zu einer Hexadezimalzahl die entsprechende negative Zahl durch Bildung des Sechzehnerkomplements; zu einer Dualzahl die entsprechende negative Zahl durch Bildung des Zweierkomplements.

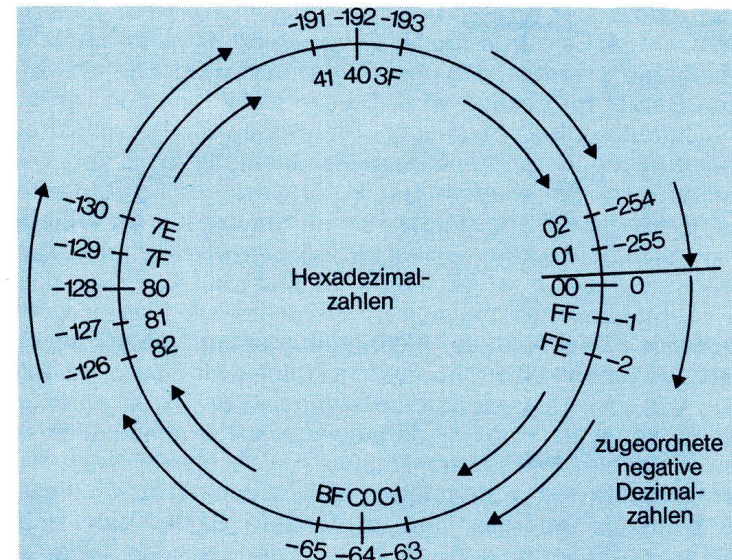
Es bleibt uns in Zukunft überlassen, ob wir den Akkuinhalt als positive oder als negative Zahl interpretieren. Da beim Aufwärtszählen der Akkuinhalt 00 der Zahl FF, da beim Abwärtszählen der Akkuinhalt FF der Zahl 00 folgt, stellt man die Zahlen auch mit Hilfe eines Zahlenkreises dar:

1. Möglichkeit: positive Zahlen



Die Pfeile geben die Richtung beim Aufwärtszählen an.

2. Möglichkeit: negative Zahlen



Die Pfeile geben die Richtung beim Abwärtszählen an.

9.7 Aufgaben zu Kapitel 9

- Die folgenden Differenzen sollen bestimmt werden:
 - $A5B - 9F1$
 - $D70F - 73EA$
 - $C61B2 - A61B4$
 - $3E7 - 41A$
- Zu folgenden Zahlen sind die Sechzehnerkomplemente zu ermitteln:
 - 1F
 - B7C
 - 8D5AE
- Die folgenden zweistelligen Akkuinhalte sollen einmal als positive, zum anderen als negative Zahlen interpretiert werden: Welche positiven oder negativen Dezimalzahlen werden durch den Akkuinhalt dargestellt?
 - E5
 - CA
 - 2F
- Welcher vierstelligen Hexadezimalzahl würde man sinnvollerweise -1 zuordnen?
- Es ist ein Programm zu schreiben, mit dessen Hilfe vierstellige Sechzehnerkomplemente berechnet werden können.
- Es ist ein Programm für die Subtraktion vierstelliger Dezimalzahlen zu schreiben. Es genügt, wenn das Programm positive Differenzen richtig ermittelt.

10. Sprungbefehle

10.1 Der Programmzähler PC

In der CPU gibt es vier 16-Bit-Register, die als Pointer-Register bezeichnet werden (engl. pointer – der Zeiger). Sie werden durchnummeriert von 0 bis 3: Pointer 0, Pointer 1, Pointer 2 und Pointer 3, kurz: P0, P1, P2 und P3. Im Normalfall zeigen diese Pointer auf bestimmte Adressen, daher müssen es auch unbedingt 16-Bit-Register sein. Wir können mit **CPU**, **0** bis **CPU**, **3** diese Register-Inhalte überprüfen. Wir werden feststellen, daß P0 mit PC, daß P1 mit SP bezeichnet wird. Das liegt an der besonderen Funktion dieser beiden Pointer.

Mit dem Pointer P0, bzw PC wollen wir uns in diesem Abschnitt beschäftigen. PC ist die Abkürzung der englischen Bezeichnung **program counter** (Programmzähler). Wegen seiner Funktion kann dieser Pointer auch als Befehlszähler oder Befehlszeiger bezeichnet werden. Er zeigt auf den nächsten Befehl, der in einem Programm zu bearbeiten ist. Genauer: Das Pointer-Register enthält die Adresse des Speichers, aus dem das nächste Befehlsbyte zu holen ist. Zu der Bearbeitung eines jeden Befehls gehört die Erhöhung der im Programmzähler gespeicherten Adresse. Bei 1-, 2- bzw. 3-Byte-Befehlen wird der Inhalt des Programmzählers automatisch um 1, 2 oder 3 erhöht. Nach Bearbeitung eines Befehls zeigt der Programmzähler auf den Speicher, aus dem das erste Byte des nächsten Befehls zu holen ist.

Wir wollen diese Aussagen überprüfen: Wir betätigen zweimal die Taste **A↔D** und sehen uns mit **CPU**, **0** den Inhalt des Programmzählers an: (PC) = 1000. Diese Funktion der Taste **A↔D** hatten wir bisher verschwiegen. Wenn nach der Betätigung dieser Taste in der Anzeige die Adresse 1000 erscheint, ist auch der Inhalt des Programmzählers 1000 geworden. Wenn wir jetzt – es muß natürlich ab Adresse 1000 ein Programm eingetastet sein – die Taste **RUN** betätigen, dann beginnt die Programmbearbeitung wegen (PC) = 1000 mit dem Befehl, dessen erstes Byte im Speicher mit der Adresse 1000 steht. Wir wollen uns überzeugen, daß im Programmzähler auch eine andere Adresse als 1000 stehen kann. Dazu tasten wir z. B. folgende sechs Befehle ein:

NR.	MNEM.CODE	ADR.	OP.CODE
1.	NOP	1000	00
2.	NOP	1001	00
3.	NOP	1002	00
4.	NOP	1003	00
5.	NOP	1004	00
6.	CALL TEST	1005	1E

Bevor wir dieses kleine Programm starten, einige Bemerkungen zu dem neuen 1-Byte-Befehl mit dem mnemonischen Code NOP und dem Operationscode 00. NOP ist die Abkürzung für **no operation** (keine Operation). Man kann z. B. einen Befehl, den man in einem Programm nicht mehr benötigt, durch einen oder mehrere NOP-Befehle überschreiben. Man kann beim Programmieren an einer Stelle zunächst einige NOP-Befehle schreiben, wo man später andere Befehle einfügen will. Beim NOP-Befehl passiert sonst nichts bis auf eins: Der Inhalt des Programmzählers wird um 1 erhöht.

BESCHREIBUNG	MNEM.CODE	OP.CODE	OPERATION
KEINE OPERATION	NOP	00	(PC) := (PC) + 1

Wenn wir nun unser kleines Programm starten, werden die fünf NOP-Befehle ausgeführt. Der Programmzähler zeigt jetzt auf die Adresse 1005. Jetzt wird der CALL TEST ausgeführt. Es erscheint die Anzeige

1005 71E

Wenn wir uns jetzt mit **CPU**, **0** den Inhalt des Programmzählers ansehen, stellen wir fest: (PC) = 1005. Eine Zusatzbemerkung: Eigentlich müßte nach der Bearbeitung des sechsten 1-Byte-Befehls der Programmzähler auf die Adresse 1006 zeigen. Das ist normalerweise der Fall. Eine Ausnahme erreichen wir hier gerade durch unseren speziellen Call. Während der Ausführung dieses Calls wird der PC-Inhalt um 1 vermindert, damit der Pointer – nach Erhöhung um 1 – wieder auf dieselbe Adresse zeigt. Eventuell wollen wir ja diesen Call oder Breakpoint in einem zu testenden Programm wieder durch einen anderen Befehl überschreiben und dann das Programm von hier weiterlaufen lassen (vgl. auch Abschnitt 7.9.4).

10.2 Der Jump-Befehl

Beim letzten Programm von Kapitel 9 hatten wir beim vierten Befehl noch keinen mnemonischen Code angegeben (vgl. 9.6). Wir wollen diesen Befehl jetzt besprechen. Wir vervollständigen das angegebene Programm:

NR.	MARKE	MNEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.		ILD A,=016	1000	1C 16	(A) := 16
2.	WEITER	SUB A,=1	1002	1F 01	(A) := (A) - 1
3.		CALL ANZ A	1004	18	ANZEIGE VON (A)
4.		JMP WEITER	1005	24 01 10	SPRUNG NACH "WEITER"

Unsere Programmdarstellungen werden ab jetzt eine zusätzliche Spalte enthalten, die mit „Marke“ überschrieben ist. Im wesentlichen werden hier Sprungziele angegeben, es werden die Stellen markiert, zu denen beim Programmablauf gesprungen wird. Die Benennung der Marke können wir frei wählen. Bei unserem Programm haben wir die Stelle, an der vom Akkuinhalt 1 subtrahiert wird, mit WEITER bezeichnet. Im vierten Befehl steht der Befehl JUMP WEITER. Nach dem Sprung erfolgt jedesmal wieder die Erniedrigung des Akkuinhalts.

Nun zu dem Sprungbefehl: JMP ist die Abkürzung für den mnemonischen Code (engl. jump = der Sprung, springen). Dieser Sprungbefehl ist ein unbedingter Sprung. Im Abschnitt 10.4 werden wir auch bedingte Sprünge kennenlernen. Dort wird erst eine Bedingung untersucht und daraufhin festgestellt, ob ein Sprung ausgeführt wird oder nicht. Der Jump-Befehl – ein unbedingter Sprungbefehl – wird in jedem Fall ausgeführt.

BESCHREIBUNG	MNEM.CODE	OP.CODE	OPERATION
SPRUNGE UNBEDINGT ZUM PROGRAMMTEIL "MARKE"	JMP MARKE	24 YY XX	ADR := XXYY (PC) := ADR

Mit der Marke bezeichnen wir die Stelle, bei der der Befehl steht, der nach dem Sprungbefehl bearbeitet werden soll. Das erste Byte dieses nächsten Befehls steht im Speicher mit der Adresse ADR + 1. Im Operationscode für den Sprungbefehl muß dann die Adresse ADR angegeben sein. Wie bei den anderen 3-Byte-Befehlen steht im zweiten Byte das LOB (low order byte), im dritten Byte das HOB (high order byte) der vierstelligen Hexadezimalzahl, die hier die Sprungadresse ADR bedeutet.

Die Zusammenhänge sind sofort verständlich, wenn wir uns ansehen, worin die Sprungoperation eigentlich besteht. Bei der Ausführung des Befehls wird die im Operationscode angegebene Adresse ADR (HOB:XX; LOB:YY) in den Programmzähler PC geladen. Vor der Bearbeitung des Sprungbefehls zeigt der Programmzähler auf den Speicher, in dem das erste Byte des Sprungbefehls (24) steht. Der Inhalt des PC-Registers wird dann zum Laden der Sprungadresse zweimal um 1 erhöht. Nach dem Laden der Sprungadresse ADR in den Programmzähler wird dieser Inhalt zum dritten Mal um 1 erhöht. Das Programm wird dann bei der Adresse ADR + 1 festgesetzt.

Beispiele:

INMEM.CODE	OP.CODE	BEMERKUNGEN
JMP WEITER	24 01 10	NÄCHSTER BEFEHL BEGINNT BEI ADRESSE 1002 (WEITER)
JMP SIRENE	24 51 04	NÄCHSTER BEFEHL BEGINNT BEI ADRESSE 0452 (SIRENE)

Das erste Beispiel können wir noch einmal mit dem kleinen Programm vergleichen, das wir in diesem Abschnitt wiederholt haben. Beim Jump-Befehl wird die Adresse 1001 in den Programmzähler geladen. Nach der Erhöhung um 1 wird das Programm mit dem Befehl SUB A, = 1 (Marke WEITER, Adresse 1002) fortgesetzt. Das zweite Beispiel stammt aus dem Abschnitt 3.4.2, Diebstahlsicherung. Bei Verdunklung des LDR wird ein Sprung in das Programm der Betriebssoftware ausgeführt. Die Adresse 0451 wird in den Programmzähler geladen und anschließend um 1 erhöht. Bei der Adresse 0452 beginnt das Spiel Sirene (vgl. 1.3.6). Ebenso können wir aus unserem Programm zu jedem anderen Spiel springen. Das Spiel „gelbes Blinklicht“ beginnt bei der Adresse 03CC. Wie müßte der Operationscode für einen Sprungbefehl mit dem mnemonischen Code JMP BLINK aussehen? Die Zieladresse für den Sprung ist dann 03CB, der Operationscode heißt 24 CB 03. Der Befehl läßt sich leicht überprüfen:

INR.	MARKE	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.		JMP BLINK	1000	24 CB 03	SPRUNG ZUM SPIEL "GELBES BLINKLICHT"

WEITERE MARKE, BEZEICHNUNG	ADR.	BEMERKUNGEN
BLINK	03CC	BEI DIESER ADRESSE BEGINNT DAS SPIEL "GELBES BLINKLICHT"

Nach dem Start dieses Programms, das nur aus dem einen Sprungbefehl besteht, läuft das Spiel 2: gelbes Blinklicht. Das Spiel ist nur mit der Taste **RS** anzuhalten.

10.3 Ein Branch-Befehl

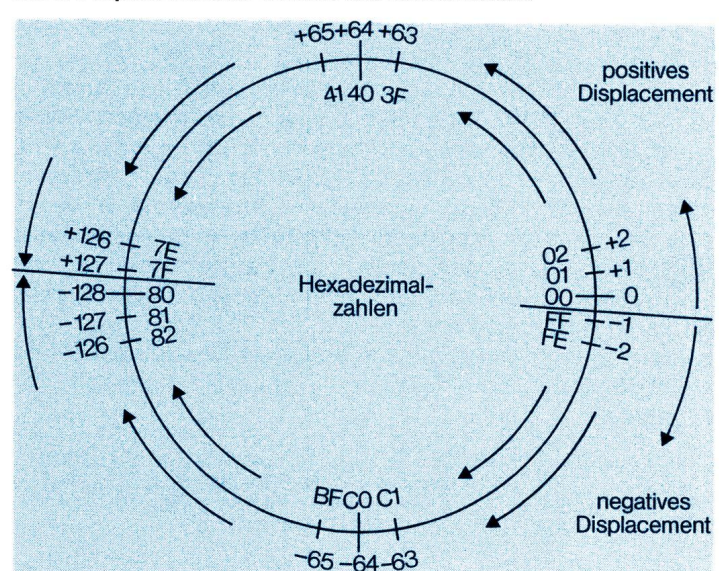
Branch ist auch wieder ein englisches Wort und heißt Zweig oder abzweigen. Der erste Branch-Befehl ist wie der Jump-Befehl ein unbedingter Sprungbefehl. Hier ist „Branch“ eigentlich eine widersinnige Bezeichnung. Bei einem unbedingten Sprung wird ja das Programm nicht verzweigt. Eine Verzweigungsmöglichkeit gibt es erst bei den weiteren drei Branch-Befehlen, die wir uns im nächsten Ab-

schnitt ansehen werden. Bei allen vier Branch-Befehlen haben wir es aber mit einer neuen Adressierungsart, mit der relativen Adressierung, zu tun. Das werden wir zunächst an diesem unbedingten Branch-Befehl erläutern.

BESCHREIBUNG	INMEM.CODE	OP.CODE	OPERATION
VERZWEIGUNG UNBEDINGT ZUM PROGRAMMTEIL "MARKE"	BRA MARKE	74 XX	ADR := (PC) + XX (PC) := ADR

Der Inhalt des Programmzählers wird um die zweistellige Hexadezimalzahl XX verändert. Diese zweistellige Hexadezimalzahl XX wird auch als Displacement (engl. Verschiebung) bezeichnet. Der Programmzähler wird, ausgehend von seiner momentanen Stellung, um das Displacement verschoben. Die Zieladresse für den Sprung wird relativ zum momentanen PC-Inhalt angegeben. Daher relative Adressierung.

Wir hatten im Abschnitt 9.6 festgehalten, daß es uns überlassen ist, eine zweistellige Hexadezimalzahl als positive oder als negative Zahl zu interpretieren. Hier haben wir diese freie Interpretationsmöglichkeit nicht. Da beim Springen sowohl Vorwärts- als auch Rückwärtssprünge möglich sein müssen (d. h. der Inhalt des Programmzählers muß erhöht oder erniedrigt werden können), muß das Displacement einmal als positive, zum anderen als negative Zahl aufgefaßt werden. Für den Mikroprozessor bedeutet eine Zahl von 00 bis 7F ein positives, eine Hexadezimalzahl von 80 bis FF ein negatives Displacement. Wir veranschaulichen die möglichen Displacements wieder am Zahlenkreis:



Wir wollen uns den Sprungbefehl (den unbedingten Branch-Befehl) in einigen Beispielen ansehen.

INR.	MARKE	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.		CALL EIN 2ST	1000	16	EINGABE (A)
2.		BRA BEF 14	1001	74 1B	SPRUNG NACH "BEF 14"
3.			1003		
4.			1004		
5.			1006		
6.			1009		HIER SOLLN IRGENDWELCHE
7.			100C		1-, 2- ODER 3-BYTE-BEFEHLE
8.			100E		STEHEN
9.			1011		
10.			1014		
11.			1017		
12.			1018		
13.			101B		
14.	BEF 14	ILD A, ZWSP	101E	1C5 D8	
15.		CALL ANZ A	1020	18	ANZEIGE (A)
16.		CALL TEST	1021	1E	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZWSP	1FFD8	ZWISCHENSPEICHER

Beispiel 1:
 An diesem Beispiel wollen wir uns klarmachen, wie ein positives Displacement (für einen Vorwärtssprung) berechnet werden kann. Das zweite Byte des Sprungbefehls (also das Displacement) steht im Speicher mit der Adresse 1002. Wenn der Sprungbefehl in den Mikroprozessor eingelesen, aber noch nicht ausgeführt ist, steht im Programmzähler ebenfalls die Adresse 1002. Nach Ausführung der Sprungoperation soll im PC-Register die Adresse 101D stehen. Diese Adresse wird ja dann automatisch noch um 1 erhöht, und das Programm wird mit dem Befehl fortgesetzt, der im Speicher 101E beginnt (dessen erstes Byte im Speicher mit der Adresse 101E steht). Das Displacement für den Sprungbefehl läßt sich also folgendermaßen berechnen: $101D - 1002 = 1B$. Die Operation, die beim Sprungbefehl ausgeführt wird, besteht darin, den PC-Inhalt zusätzlich zur schrittweisen Erhöhung beim Programmablauf um 1B zu vergrößern.

Das Ganze noch einmal in Kürze:
 Der Befehl nach dem Sprung beginnt bei Adresse 101E, die Zieladresse für den Sprung ist dann um 1 kleiner: $ADR = 101E - 1 = 101D$. Der Inhalt des PC-Registers ist 1002 (hier steht das zweite Byte des Sprungbefehls): $(PC) = 1002$. Aus der Gleichung $ADR = (PC) + XX$, ergibt sich für das Displacement: $XX = ADR - (PC) = 101D - 1002 = 1B$.

Die Berechnung des Displacements geschieht ganz entsprechend so, wie wir es auch bei einem positiven Displacement gemacht haben. Der Befehl nach dem Sprung (der Subtraktionsbefehl) beginnt bei Adresse 1002. Die Zieladresse ist um 1 kleiner, also $ADR = 1001$. Das zweite Byte des Sprungbefehls steht im Speicher mit der Adresse 1006: $(PC) = 1006$. Das Displacement wird wieder aufgrund folgender Gleichung berechnet: $ADR = (PC) + XX$. Die Rechnung ergibt:

$$\begin{aligned}
 XX &= ADR - (PC) \\
 &= 1001 - 1006 \\
 &= FFFB \\
 &= -5
 \end{aligned}$$

Als Displacement kommt natürlich nur eine zweistellige Hexadezimalzahl in Frage. Wir brauchen daher auch nur die Differenz in den beiden letzten Stellen zu bilden. Zur Erinnerung:

Entweder (vgl. 9.3):

$$\begin{array}{c}
 1 \rightarrow 0 \rightarrow 1 \\
 \downarrow \quad \downarrow \\
 0 \quad 6 \\
 \downarrow \quad \downarrow \\
 1 \quad 1 \\
 \hline
 F \quad B
 \end{array}$$

oder Sechzehnerkompl. von 06: $F9 + 1 = FA$ (vgl. 9.4.2)

$$\begin{array}{cc}
 0 & 1 \\
 F & A \\
 \hline
 F & B
 \end{array}
 +$$

Ein zusätzlicher Tip: Häufig läßt sich das Displacement durch Abzählen bestimmen. Man zählt einfach die Speicherplätze zwischen dem Sprungbefehl und dem Befehl, der nach dem Sprung ausgeführt werden soll. In unserem Programm ist dieser Teil durch Leerkästen markiert. Es können hier irgendwelche 1-, 2- oder 3-Byte-Befehle stehen. Wenn wir diese Kästen im Hexadezimalsystem abzählen, erhalten wir als Displacement auch 1B.

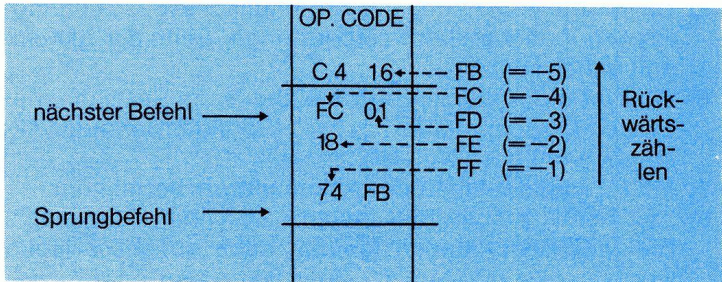
Wenn wir dieses Programm testen wollen, brauchen wir uns um die Inhalte in den Speichern von 1003 bis 101D nicht zu kümmern.

Beispiel 2:
 Was bedeutet der Sprungbefehl mit dem Operationscode 7400? Das Displacement ist 00. Wenn das Displacement zum Inhalt des Programmzählers addiert wird, zeigt der Pointer PC auf den Speicher, in dem das Displacement 00 steht. Anschließend wird der PC-Inhalt um 1 erhöht. Nach diesem Sprungbefehl wird also der unmittelbar folgende Befehl ausgeführt. Mit diesem Sprungbefehl wird das gleiche wie mit zwei NOP-Befehlen erreicht.

Beispiel 3:
 Im nächsten Beispiel soll ein negatives Displacement bestimmt werden. Wir wollen beim ersten Programm von Abschnitt 10.2 (vgl. auch Abschnitt 9.6) statt des Jump-Befehls den Branch-Befehl benutzen.

INR.	MARKE	INMEM.	CODE	ADR.	OP. CODE	BEMERKUNGEN
1.		ILD A,	#016	1000	C4 16	<A>:=16
2.	WEITER	SUB A,	1	1002	FC 01	<A>:=<A>-1
3.		ICALL ANZ A		1004	18	ANZEIGE VON <A>
4.		BRRA WEITER		1005	74 FB	SPRUNG NACH "WEITER"

Auch hier läßt sich das Displacement durch Abzählen bestimmen. Wir müssen nur hexadezimal rückwärts zählen: Wir beginnen beim ersten Byte des Sprungbefehls mit FF und zählen bis zu dem Byte, das vor dem Befehl steht, der nach dem Sprung ausgeführt werden soll.



Natürlich können wir auch dezimal rückwärts zählen: $-1, -2, \dots -5$ und dann die negative Dezimalzahl in das entsprechende negative Displacement umwandeln.

Beispiel 4:
 Was passiert, wenn wir den Sprungbefehl mit dem Operationscode 74 FE wählen? Das Displacement ist FE (oder -2 !). Nach diesem Sprung wird wieder derselbe Sprung ausgeführt und dann wieder usw. Das Programm besteht aus einer Endlosschleife. Wir können den Mikroprozessor bisher nur mit Hilfe der **RS**-Taste oder durch Abschalten der Spannungsversorgung dazu bewegen, diese Schleife zu verlassen.

10.4 Bedingte Sprünge

Die nächsten drei Branch-Befehle sind bedingte Sprungbefehle. Sie haben mit dem im Abschnitt 10.3 behandelten unbedingten Branch-Befehl die relative Adressierung gemeinsam. Im Gegensatz zu diesem Befehl erlauben die drei folgenden Befehle Programmverzweigungen. Hier ist die Bezeichnung „Branch-Befehl“ wirklich angebracht. An einer bestimmten Stelle des Programms gibt es zwei mögliche Fortsetzungen. Es wird eine Bedingung überprüft; je nachdem wie diese Prüfung ausfällt, wird einer von zwei möglichen Wegen eingeschlagen.

Die Bedingungen, die überprüft werden, beziehen sich alle auf den Akkuinhalt. Es wird überprüft, ob der Akkuinhalt positiv, gleich Null oder ungleich Null ist. Alle anderen Sprungbedingungen müssen per Software, also per Programm auf diese drei Sprungbedingungen zurückgeführt werden (und das geht hervorragend!).

Zunächst die Befehle:

BESCHREIBUNG	INMEM.CODE	OP.CODE	OPERATION
VERZWEIGE ZUM PROGRAMM- TEIL "MARKE" NUR WENN AKKU-INHALT POSITIV	BP MARKE	64 XX	INUR WENN (A) POSITIV: ADR := (PC) + XX (PC) := ADR
VERZWEIGE ZUM PROGRAMM- TEIL "MARKE" NUR WENN AKKU-INHALT GLEICH NULL	BZ MARKE	6C XX	INUR WENN (A) GLEICH NULL: ADR := (PC) + XX (PC) := ADR
VERZWEIGE ZUM PROGRAMM- TEIL "MARKE" NUR WENN AKKU-INH. UNGLEICH NULL	BNZ MARKE	7C XX	INUR WENN (A) UNGLEICH NULL: ADR := (PC) + XX (PC) := ADR

Einige Bemerkungen dazu:

- 1) BP ist die Abkürzung für die englische Bezeichnung: Branch if accumulator positiv (Springe, wenn der Akkuinhalt positiv ist).
- 2) BZ ist die Abkürzung für die englische Bezeichnung: Branch if accumulator zero (Springe, wenn der Akkuinhalt gleich Null ist).
- 3) BNZ ist die Abkürzung für die englische Bezeichnung: Branch if accumulator not zero (Springe, wenn der Akkuinhalt ungleich Null ist).
- 4) Mit (A) wird der Inhalt des Akkumulators bezeichnet.
- 5) „ \geq “ heißt „größer oder gleich“; „ \neq “ heißt „ungleich“. Die drei Bedingungen können auch so geschrieben werden: $(A) \geq 00$, $(A) = 00$, $(A) \neq 00$.
- 6) $(A) \geq 0$ gilt, wenn im Akku eine Zahl zwischen 00 und 7F steht;
 $(A) \geq 0$ gilt nicht, wenn im Akku eine Zahl zwischen 80 und FF steht. Der Akkuinhalt wird genauso als positiv oder negativ (nicht positiv) aufgefaßt wie beim Displacement (vgl. Abschnitt 10.3).
- 7) Die Bedingung $(A) = 00$ gilt nur, wenn im Akkuinhalt die zweistellige Hexadezimalzahl 00 steht, wenn alle acht Dualziffern im Akku Null sind.
- 8) die Bedingung $(A) \neq 0$ gilt immer, wenn auch nur eine der acht Dualziffern im Akku ungleich Null ist.
- 9) Wenn jeweils die Bedingung für den bedingten Sprung erfüllt ist, wird der Sprung ausgeführt. Für diesen Sprung gilt das, was über die relative Adressierung beim Branch-Befehl im Abschnitt 10.3 gesagt wurde.
- 10) Ist die Sprungbedingung nicht erfüllt, wird das Programm mit dem Befehl fortgesetzt, der dem bedingten Sprung im Programm unmittelbar folgt.
- 11) Ob ein Sprung ausgeführt wird oder nicht, der Inhalt des Akkumulators wird dabei nicht verändert.

Wir wollen uns jetzt natürlich wieder ein Beispielprogramm ansehen. Die Programme, die wir zur Erklärung der Befehle wählen, sind bewußt sehr einfach. Erst mit weiteren Befehlen werden unsere Programme interessanter. Das folgende Programm soll lediglich die Möglichkeiten zur Programmverzweigung verdeutlichen. Zunächst wird mit Hilfe des Calls CALL EIN 4ST eine maximal vierstellige Zahl eingegeben. Nach Beendigung der Eingabe soll diese Zahl angezeigt werden, und zwar mit Hilfe des Calls CALL ANZ A, wenn die Zahl ein- oder zweistellig ist, mit Hilfe des Calls CALL ANZ EA, wenn sie drei- oder vierstellig ist.

Wie läßt sich diese Entscheidung (Ist die eingegebene Zahl maximal zweistellig oder nicht?) auf die uns zur Verfügung stehenden Sprungbedingungen zurückführen? Wir wissen, nach der Eingabe können wir die Zahl ins EA-Register bringen. Bei einer maximal zweistelligen Zahl steht im Extension-Register 00, sonst ein Inhalt ungleich Null. Wir müßten also als Sprungbedingung untersuchen: $(E) = 0?$ oder $(E) \neq 0?$ Die Lösung des Problems liegt auf der Hand. Da sich die Sprungbedingungen grundsätzlich auf den Akkuinhalt beziehen, müssen wir den Inhalt des Extension-Registers vor dem bedingten Sprung in den Akku laden. Wir müssen dann nur dafür sorgen, daß der Inhalt des EA-Registers nach dem Sprung (nach der Programmverzweigung) und vor der Anzeige wieder seinen ursprünglichen Wert hat. Einen zweiten bedingten Sprung wollen wir noch in das Programm einbauen. Es soll keine Anzeige erfolgen, wenn die eingegebene Zahl 0 ist. In diesem Fall haben wir eine vierstellige Zahl zu untersuchen. Das läßt sich nur mit zwei aufeinander folgenden Sprungentscheidungen durchführen. Zunächst muß der Inhalt des Extension-Registers überprüft werden: Wenn $(E) = 0$ gilt und außerdem $(A) = 0$, dann wird ein Sprung zum Anfang des Programms durchgeführt. Es erfolgt keine Anzeige.

Nach diesen Vorüberlegungen steht das Gerüst unseres geplanten Programms. Für das endgültige Programm gibt es natürlich mehrere Möglichkeiten; z. B. läßt sich ein Sprung programmieren, wenn $(A) = 0$ oder wenn $(A) \neq 0$ ist. Man wird das Programm bei der Entwicklung zunächst im mnemonischen Code schreiben. Anschließend wird man die Adressen festlegen und die Befehle in den Operationscode übersetzen. Dabei kann man die Displacements für die Sprünge zunächst noch offen lassen.

INR.	MARKE	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	CALL EIN 4ST	1000	17	
2.		LD EA,ZWSP	1001	85 DS	
3.		XCH A,E	1003	01	
4.		BNZ VIERST	1004	7C	
					ISPRUNG NACH VIERST. WENN (E) UNGLEICH 00
5.	ZWEIST	XCH A,E	1006	01	
6.		BZ BEGINN	1007	6C	
					ISPRUNG NACH BEGINN. WENN (A) GLEICH 00
7.		CALL ANZ A	1009	18	
8.		BR A BEGINN	100A	174	
					ISPRUNG NACH BEGINN
9.	VIERST	XCH A,E	100C	01	
10.		CALL ANZ EA	100D	19	
11.		BR A BEGINN	100E	174	
					ISPRUNG NACH BEGINN

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZWSP	1FFD8	ZWISCHENSPEICHER
	1FFD9	

Mit etwas Training läßt sich die Bedeutung der elf Befehle am mnemonischen Code erkennen. Man muß sich nach jedem Befehl sehr sorgfältig überlegen, welche Inhalte in den Registern oder Speichern stehen. Nach dem zweiten Befehl steht die eingegebene Zahl im EA-Register, eventuell mit einigen führenden Nullen, auch wenn das bei der Eingabe nicht erkennbar war.

Im dritten Befehl werden die Inhalte vom Akkumulator und vom Extension-Register getauscht, damit beim folgenden bedingten Sprung mit „Ist (A) \neq 0?“ der Inhalt des E-Registers untersucht werden kann. Ist die Sprungbedingung erfüllt, erfolgt der Sprung zur Marke VIERST (vierstellige Anzeige), andernfalls wird das Programm beim unmittelbar folgenden Befehl fortgesetzt. Die Marke ZWEIST (zweistellige Anzeige) ist nicht erforderlich, sie erhöht aber die Übersichtlichkeit. Nach erneutem Tausch von (A) und (E) wird im sechsten Befehl untersucht, ob überhaupt eine Ziffer ungleich Null eingegeben wurde. Ist der Akkuinhalt Null, erfolgt – ohne Anzeige – ein Sprung zum Programmstart. Jetzt zur Berechnung der Displacements. Im vierten Befehl muß ein positives Displacement berechnet werden. Es müssen sechs Speicherplätze übersprungen werden, also: $7C\ 06$. Bei einer Berechnung würden wir von der Zieladresse $100B$ ($100C - 1$) die Adresse 1005 subtrahieren müssen: $100B - 1005 = 06$. Die drei negativen Displacements ließen sich auch durch Abzählen bestimmen. Wir geben kurz die Rechnung an (vgl. Abschnitt 10.3): Es handelt sich in allen drei Fällen um einen Sprung zur Marke BEGINN. Die Zieladresse ist jeweils $0FFF$ ($= 1000 - 1$). Wir haben uns schon überlegt, daß wir nur zweistellig subtrahieren müssen. Wir subtrahieren also von FF :

6. Befehl

$$\begin{array}{r} F\ F \\ 0\ 8 \\ \hline F\ 7 \end{array} -$$

8. Befehl

$$\begin{array}{r} F\ F \\ 0\ B \\ \hline F\ 4 \end{array} -$$

11. Befehl

$$\begin{array}{r} F\ F \\ 0\ F \\ \hline F\ 0 \end{array} -$$

Das vollständige Programm lautet also:

INR.	MARKE	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	ICALL EIN 4ST	1000	17	
2.		ILD EA,ZWSP	1001	85 D8	
3.		IXCH A,E	1003	01	
4.		IBNZ VIERST	1004	7C 06	SPRUNG NACH VIERST, WENN <E> UNGLEICH 00
5.	ZWEIST	IXCH A,E	1006	01	
6.		IBZ BEGINN	1007	6C F7	SPRUNG NACH BEGINN, WENN <A> GLEICH 00
7.		ICALL ANZ A	1009	18	
8.		IBRA BEGINN	100A	74 F4	SPRUNG NACH BEGINN
9.	VIERST	IXCH A,E	100C	01	
10.		ICALL ANZ EA	100D	19	
11.		IBRA BEGINN	100E	74 F0	SPRUNG NACH BEGINN

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZWSP	1FFD8	ZWISCHENSPEICHER
	1FFD9	

Auch wenn bei der Erprobung nichts Aufregendes passiert, werden wir es uns doch nicht nehmen lassen, das Programm zu testen. Nach dem Eintasten der Befehle starten wir das Programm mit der **RUN**-Taste. Der Computer erwartet die Eingabe einer maximal vierstelligen Zahl. Die Zahleneingabe und auch die folgende Anzeige lassen sich mit einer beliebigen Funktionstaste (außer **A \leftrightarrow D**) abschließen.

10.5 Wir lassen das Displacement berechnen

Weshalb sollen wir die Displacements selber berechnen? Wir haben ja unseren Computer. Zumindest können wir unsere Berechnungen der Displacements vom Computer kontrollieren lassen.

INR.	MARKE	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	ICALL EIN 4ST	1000	17	EINGABE DER SPEICHERADRESSE, WO DAS DISPLACEMENT STEHEN SOLL
2.		ILD EA,ZWSP	1001	85 D8	
3.		IST EA,ABSP	1003	8D E0	
4.		ICALL EIN 4ST	1005	17	EINGABE DER SPEICHERADRESSE, WO DER NÄCHSTE BEFEHL BEGINNT
5.		ILD EA,ZWSP	1006	85 D8	
6.		SUB EA,ABSP	1008	8D E0	BERECHNUNG DES DISPLACEMENTS
7.		SUB EA,#1	100A	8C 01 00	
8.		ICALL ANZ A	100D	18	ANZEIGE DES DISPLACEMENTS
9.		IBRA BEGINN	100E	74 F0	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZWSP	1FFD8	ZWISCHENSPEICHER
	1FFD9	
ABSP	1FFE0	ABSPRUNGADRESSE
	1FFE1	

Wenn wir dieses Programm einsetzen, müssen wir sehr genau aufpassen. Als erste Adresse wird die Absprungadresse eingegeben. Damit ist die Adresse des Speichers gemeint, in dem das zweite Byte (das Displacement) des Sprungbefehls steht. Als zweite Adresse geben wir die Speicheradresse ein, wo das erste Byte des Befehls steht, der nach dem Sprung bearbeitet werden soll. Und noch etwas:

Das Ergebnis muß bei einem Vorwärtssprung ein Displacement zwischen 00 und 7F, bei einem Rückwärtssprung ein Displacement zwischen 80 und FF sein. Diese Kontrolle nimmt uns das Programm nicht ab. Wenn wir z. B. die beiden Adressen 100F und 1100 eingeben, dann müßte es sich eigentlich um einen Vorwärtssprung handeln ($100F < 1100$), das berechnete Displacement F0 ist aber ein Displacement für einen Rückwärtssprung. Ein Vorwärtssprung mit dieser Sprungweite ist mit einem Branch-Befehl nicht möglich.

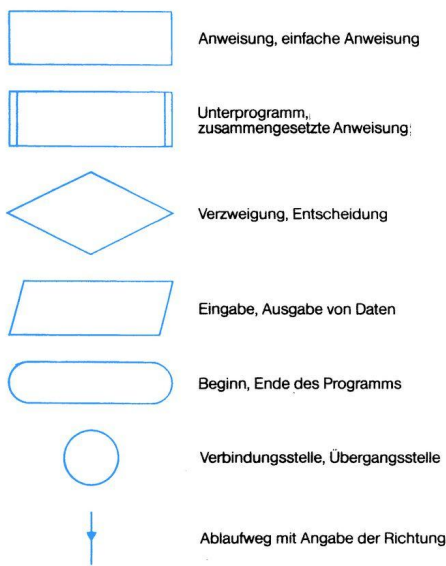
Abgesehen von diesem Problem, kann uns das Programm aber bei der Berechnung der Displacements hilfreich sein. Um das Displacement im 9. Befehl unseres Programms berechnen zu lassen, geben wir als erste Adresse (als Absprungadresse) 100F ein (der Sprungbefehl steht in den Speichern 100E und 100F). Als zweite Adresse geben wir 1000 ein, wir erhalten das Displacement F0.

Als weitere Beispiele können wir alle Sprungbefehle aus diesem Kapitel, Abschnitt 10.3 und 10.4 überprüfen.

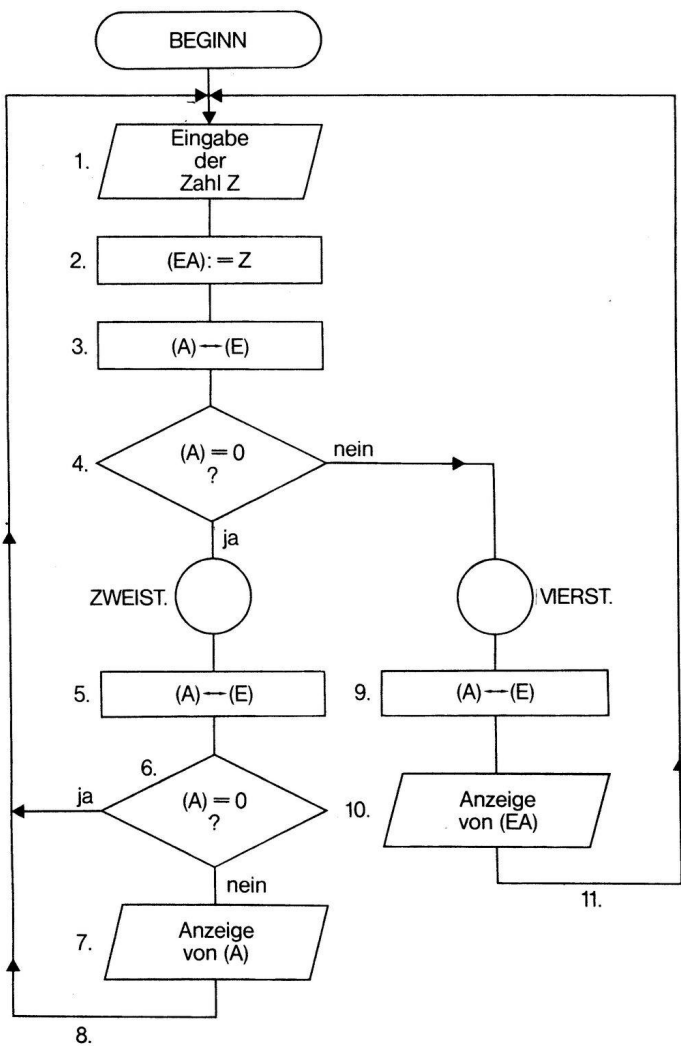
1. Adresse	2. Adresse	Displacement	vgl. Abschnitt
1002	101E	1B	10.3, Beispiel 1
z. B. 100F	z. B. 1010	00	10.3, Beispiel 2
1006	1002	FB	10.3, Beispiel 3
z. B. 101A	z. B. 1019	FE	10.3, Beispiel 4
1005	100C	06	10.4, Programm Seite 56, 57
1008	1000	F7	
100B	1000	F4	
100F	1000	F0	

10.6 Der Programmablaufplan

Ein zusätzliches Mittel zur Darstellung von Programmen (aber auch schon zur Entwicklung von Programmen) ist der Programmablaufplan. In einem solchen Plan wird mit Hilfe von bestimmten Symbolen der Programmablauf beschrieben. Folgende Symbole werden dabei verwendet:



Der Programmablaufplan für das Programm im Abschnitt 10.4 könnte folgendermaßen aussehen:



Zum Vergleich sind die Nummern der entsprechenden Befehle angegeben. Für den Programmablaufplan gibt es natürlich auch verschiedene Möglichkeiten.

Bei der Eingabe der Zahl Z und bei der Anzeige von (A) bzw. (EA) hätten wir auch das Symbol für ein Unterprogramm benutzen können. Wir rufen ja in diesen Fällen mit einem Call ein ganzes Teilprogramm in der Betriebssoftware auf. Bei der ersten Verzweigung hätte die Frage auch „(A) ≠ 0?“ lauten können. Dann müßten die beiden Antworten „ja“ und „nein“ an den beiden Fortsetzungen dieser Verzweigung vertauscht werden. Wichtig ist, daß der Programmablauf eindeutig beschrieben wird. Der Beginn liegt eindeutig fest. Pfeile sind nach Bedarf zu setzen.

10.7 Aufgaben zu Kapitel 10

- 1. Welche Unterschiede bestehen zwischen den beiden Sprungbefehlen „JMP MARKE“ und „BRA MARKE“?
- 2. Im Abschnitt 10.3 haben wir im dritten Beispiel das Displacement FB berechnet. Ein falsch berechnetes Displacement kann überraschende Folgen haben. Wir wollen überlegen oder im Versuch testen, was passieren würde, wenn wir statt FB als Displacement ermittelt hätten:
 - 2.1 FA,
 - 2.2 FC.

Welchen Befehl würde der Computer nach dem Sprungbefehl bearbeiten?

- 3.1 In den beiden Speichern 1000 und 1001 steht ein Sprungbefehl mit dem Operationscode 74 7F. In welchem Speicher müßte das erste Byte des nächsten Befehls stehen?
- 3.2 In den beiden Speichern 1100 und 1101 steht ein Sprungbefehl mit dem Operationscode 7480. In welchem Speicher müßte das erste Byte des nächsten Befehls stehen?
- 4. Die beiden Befehle in Aufgabe 3 sollen durch Jump-Befehle ersetzt werden. Wie müßte der Operationscode in den beiden Fällen heißen?

11. Inkrementieren und Dekrementieren

11.1 Zwei nützliche Befehle

Mit Hilfe der bedingten Sprungbefehle können wir jetzt Programme schreiben, bei denen ein bestimmter Programmteil, eine Schleife, mehrfach durchlaufen und dann nach einer vorgegebenen Anzahl von Durchläufen verlassen wird. Was müssen wir tun? Wir müssen die Durchläufe mitzählen und in jedem Schleifendurchlauf abfragen, ob die vorgegebene Anzahl erreicht ist.

Für solche Aufgaben gibt es – neben den Sprungbefehlen – zwei sehr nützliche Befehle, die natürlich auch in anderem Zusammenhang benutzt werden können: den Inkrement- und den Dekrement-Befehl.

BESCHREIBUNG	MMEM.CODE	OP.CODE	OPERATION
INKREMENTIERE UND LADE DEN SPEICHERINHALT IN DEN AKKUMULATOR DIREKTE ADRESSIERUNG	ILD A,BEZ	95 XX	ADR:=FF00+XX <ADR>:=<ADR>+1 <A>:=<ADR>
DEKREMENTIERE UND LADE DEN SPEICHERINHALT IN DEN AKKUMULATOR DIREKTE ADRESSIERUNG	DLD A,BEZ	9D XX	ADR:=FF00+XX <ADR>:=<ADR>-1 <A>:=<ADR>

Es handelt sich bei diesen beiden Befehlen um die direkte Adressierung, die wir schon mehrfach benutzt haben (vgl. z. B. Abschnitt 7.9.3). BEZ steht für die Bezeichnung eines Speichers oder eines Speicherinhalts mit einer Adresse zwischen FF00 und FFFF. Das zweite Byte dieser Adresse ist beim Operationscode mit XX bezeichnet.

ILD ist die Abkürzung für die vollständige englische Formulierung: increment and load (inkrementiere oder erhöhe und lade). Aus der Beschreibung der Operation ist zu erkennen, daß bei diesem Befehl der Inhalt (ADR) des Speichers mit der Adresse ADR = FF00 + XX um 1 erhöht wird. Der erhöhte Inhalt dieses Speichers steht anschließend zusätzlich im Akkumulator.

DLD ist die Abkürzung für: decrement and load (dekrementiere oder erniedrige und lade). Der Inhalt (ADR) des Speichers mit der Adresse ADR = FF00 + XX wird um 1 erniedrigt. Der erniedrigte Inhalt steht anschließend auch im Akkumulator.

Beispiele zu diesen Befehlen werden wir uns gleich in zwei kleinen Testprogrammen ansehen:

1. Beispiel:

INR.	MARKE	MMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	ILD A,=37	1000	IC4 25	
2.		IST A,1,ZAHL	1002	ICD E0	1.ZAHL:=37
3.		IST A,2,ZAHL	1004	ICD E1	2.ZAHL:=37
4.	SCHL.	ILD A,1,ZAHL	1006	95 E0	1.ZAHL:=1.ZAHL+1
5.		DLD A,2,ZAHL	1008	9D E1	2.ZAHL:=2.ZAHL-1
6.		BNZ SCHL.	100A	7C FA	WIEDERHOLE DIE SCHLEIFE, WENN <A> UNGLEICH 0
7.		ICALL TEST	100C	11E	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
1.ZAHL	FFE0	DIESE ZAHL WIRD INKREMENTIERT
2.ZAHL	FFE1	DIESE ZAHL WIRD DEKREMENTIERT

Wenn wir dieses Programm eingetastet haben und es starten, meldet sich der Computer sofort mit der Anzeige

1 0 0 C 7 1 E

Der Programmablauf ist schon beendet. In dieser nicht wahrnehmbar kurzen Zeit wurde die Schleife siebenunddreißigmal durchlaufen. Die Hexadezimalzahl 25 ($25_{16} = 37_{10}$) wurde zunächst in die Speicher FFE0 und FFE1 geladen. In jedem Schleifendurchlauf wird (FFE0) um 1 erhöht, (FFE1) um 1 erniedrigt. Im Akku steht jeweils nach dem Befehl Nr. 5 dieselbe Zahl wie im Speicher mit der Adresse FFE1. Der Speicherinhalt wird ja dekrementiert und geladen. Im Sprungbefehl wird mit dem Akkuinhalt gleichzeitig der Inhalt des Speichers FFE1 untersucht. Die Schleife wird wiederholt, bis (FFE1) Null geworden ist. Gleichzeitig wurde der Inhalt des Speichers FFE0 verdoppelt.

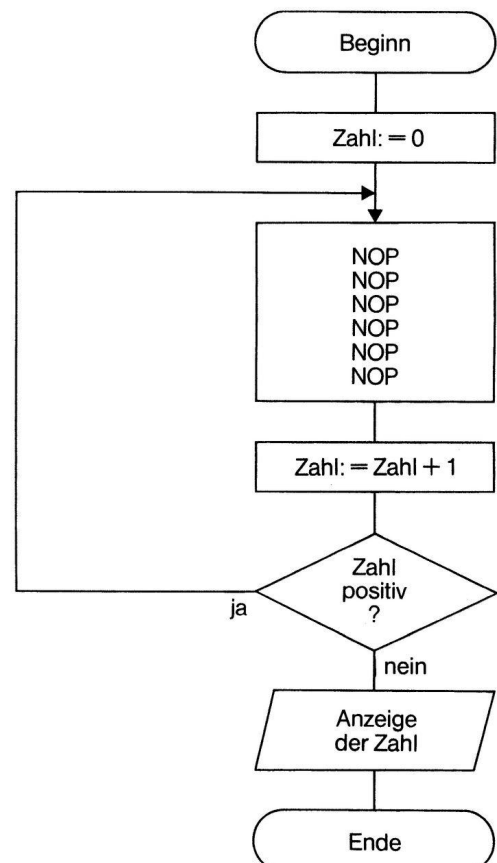
Wir können nach dem Programmablauf die Speicherinhalte überprüfen:
(FFE0) = 4A; (FFE1) = 00.

2. Beispiel:

INR.	MARKE	MMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	ILD A,=0	1000	IC4 00	
2.		IST A,ZAHL	1002	ICD E0	ZAHL:=0
3.	SCHL.	INOP	1004	00	
4.		INOP	1005	00	
5.		INOP	1006	00	
6.		INOP	1007	00	
7.		INOP	1008	00	
8.		INOP	1009	00	
9.		ILD A,ZAHL	100A	95 E0	ZAHL:=ZAHL+1
10.		IBP SCHL.	100C	64 F6	WIEDERHOLE DIE SCHLEIFE, WENN <A> POSITIV
11.		ICALL ANZ A	100E	118	
12.		ICALL TEST	100F	11E	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZAHL	FFE0	

Dazu der Programmablaufplan:



Bis sechs NOP-Befehle stehen hier stellvertretend für einige Befehle, die in jedem Schleifendurchgang bearbeitet werden. Mit ihnen soll angedeutet werden, daß in der Schleife beim Einsatz eines solchen Programms neben dem Zählen der Schleifendurchläufe normalerweise auch noch etwas anderes passiert.

11.2. Wir entwickeln ein Programm

11.2.1 Die Aufgabenstellung

Zunächst muß die Aufgabenstellung ganz klar sein. Es soll zu einer vorgegebenen Dezimalzahl n die Summe der Zahlen von 1 bis n berechnet werden.

Beispiel: Gegeben sei die Zahl $n = 15$. Dann soll die Summe $1 + 2 + 3 + 4 + \dots + 14 + 15$ berechnet werden.

$1 + 2 = 3$	$21 + 7 = 28$	$66 + 12 = 78$
$3 + 3 = 6$	$28 + 8 = 36$	$78 + 13 = 91$
$6 + 4 = 10$	$36 + 9 = 45$	$91 + 14 = 105$
$10 + 5 = 15$	$45 + 10 = 55$	$105 + 15 = 120$
$15 + 6 = 21$	$55 + 11 = 66$	

Es handelt sich um eine Aufgabe, die in der Mathematik häufiger vorkommt. Dort wird gezeigt, daß diese Aufgabe mit Hilfe der Multiplikation schneller gelöst werden kann. Man erhält die Summe auch, wenn man das Produkt $\frac{1}{2} \cdot 15 \cdot (15 + 1)$, allgemein $\frac{1}{2} \cdot n \cdot (n + 1)$ berechnet. Wir werden bei der Besprechung des Multiplikationsbefehls in Kapitel 16 auf diese Aufgabe zurückkommen. Bislang haben wir nur die Möglichkeit, die Summe mit Hilfe wiederholter Addition zu berechnen.

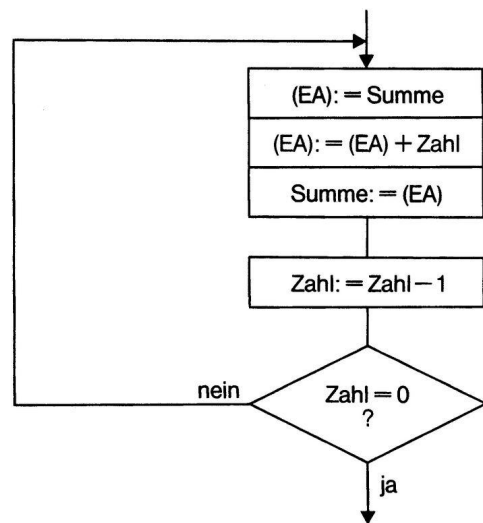
11.2.2 Planung des Programmablaufs

Wir können also mit 1 beginnen, dazu 2 addieren, dann 3 usw. Es müßte jeweils geprüft werden, ob der neue Summand schon gleich n ist; andernfalls wird das Addieren fortgesetzt; sonst ist die gesuchte Summe erreicht. Nur, wie können wir einen bedingten Sprung in Abhängigkeit davon programmieren, ob der Akkuinhalt 15 (allgemein n) ist oder nicht?

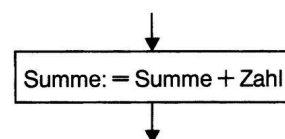
Es gibt für uns zwei Möglichkeiten, dieses Problem zu lösen. Wir können beim Addieren $1 + 2 + \dots$ gleichzeitig einen Zähler, der zunächst gleich n gesetzt wurde, in jedem Schleifendurchlauf einmal dekrementieren und dann das Addieren abbrechen, wenn der Zähler Null geworden ist. Oder – und das ist noch einfacher, und so wollen wir vorgehen – wir beginnen das Summieren mit dem größten Summanden und verlassen die Additionsschleife, wenn der Summand Null geworden ist.

Es gibt für solche Überlegungen kein Patentrezept. Programmieren hat etwas mit Phantasie, aber auch etwas mit Training zu tun. Man muß aber erreichen, daß man das gestellte Problem so formuliert, daß es mit den zur Verfügung stehenden Befehlen lösbar wird.

Der Kernteil unseres Programms, nämlich die Additionsschleife, liegt jetzt im wesentlichen fest. Wir müssen uns nur noch überlegen, wie viele Stellen unsere Zahlen haben sollen. Da $1 + 2 + \dots + 15 = 120$ ist, werden wir uns nicht auf zweistellige Dezimalzahlen beschränken. Unsere Befehle erlauben uns, daß wir ebensogut gleich vierstellig rechnen können. Wir wollen also unser Programm so schreiben, daß jede Zahl n zugelassen ist, für die die berechnete Summe noch eine vierstellige Dezimalzahl wird.



Die Schleife läßt sich im Programmablaufplan beschreiben. Die ersten drei Anweisungen ließen sich zu einer Anweisung zusammenfassen:



Es ist selbstverständlich, daß diese Addition (vierstellig) im EA-Register durchgeführt werden muß.

Vor Eintritt in die Schleife muß die Variable (die veränderliche Größe) „Zahl“ den Wert n haben. Dann wird zunächst n addiert.

Diese Variable wird schrittweise erniedrigt. Nach der Addition von 1 wird die Variable „Zahl“ gleich Null, die Schleife wird verlassen.

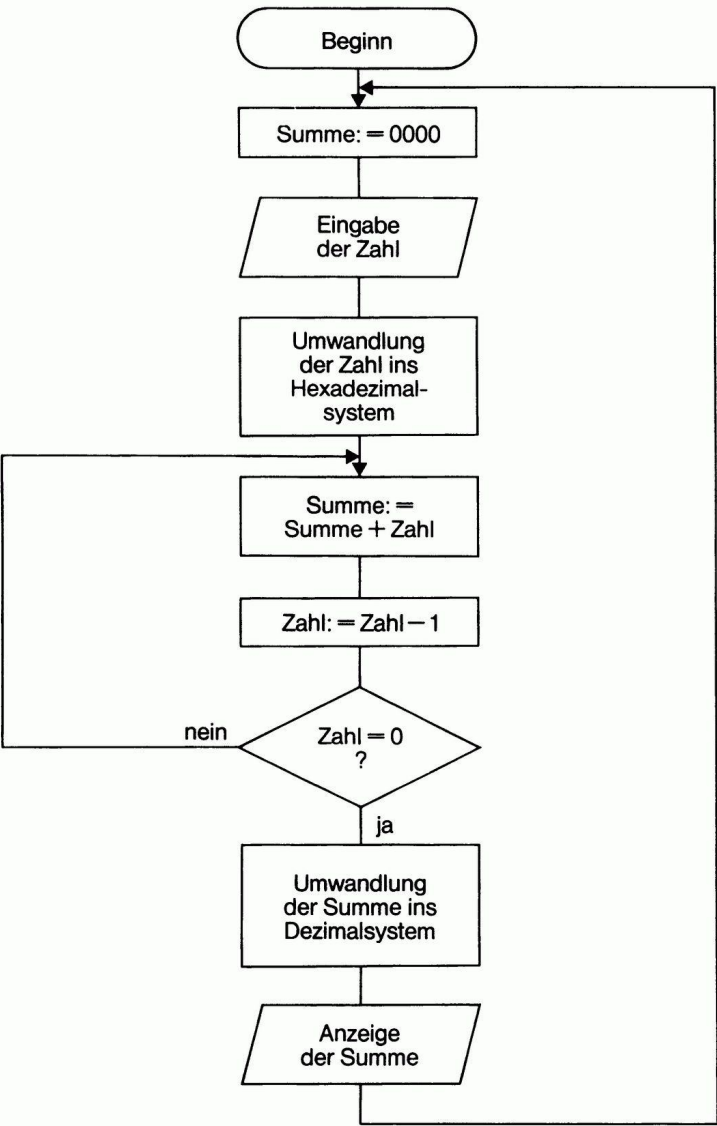
Wir haben uns jetzt noch zu überlegen, welche Vorbereitungen vor Eintritt in die Schleife getroffen werden müssen und welche Befehle nach Verlassen der Schleife ausgeführt werden sollen.

In zwei Speichern (FFE0 und FFE1) werden wir die Summe abspeichern. In diesen beiden Speichern müssen zunächst Nullen stehen. In zwei Speichern (FFE2 und FFE3) steht die Variable „Zahl“. Auch wenn n nur zweistellig ist, wird ja vierstellig addiert. Die Zahl können wir mit Hilfe des Calls CALL EIN 4ST eingeben. Nach fertiger Rechnung können wir die gesuchte Summe anzeigen lassen (CALL ANZ EA).

Wenn wir alle diese Überlegungen in ein Programm übersetzen und das Programm erproben, werden wir sehr schnell feststellen, daß wir etwas Entscheidendes vergessen haben. Nicht immer findet man einen Programmierfehler so schnell wie hier. Wir haben bisher noch nicht berücksichtigt, daß unser Programm für Dezimalzahlen geschrieben sein soll. Es ist am einfachsten, die gegebene Dezimalzahl sofort nach der Eingabe in eine Hexadezimalzahl zu verwandeln. Alle Rechnungen werden im Hexadezimalsystem ausgeführt (ADD-Befehl und DLD-Befehl). Die Verwandelung in eine Dezimalzahl erfolgt erst unmittelbar vor der Ausgabe.

11.2.3 Der Programmablaufplan

Jetzt läßt sich der vollständige Programmablaufplan angeben.



11.2.4 Das Programm

INR.	MARKE	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	ILD EA,=0	1000	04 00 00	SUMME:=0000
2.		IST EA,SUMME	1003	10 E0	
3.		ICALL EIN 4ST	1005	17	EINGABE DER ZAHL
4.		ILD EA,ZWSP	1006	05 08	
5.		ICALL DEZ-HEX	1008	18	
6.		IST EA,ZAHL	1009	10 E2	
7.		IBRA SCHL.	100B	174 02	
8.		INOP	100D	100	
9.		INOP	100E	100	
10.	SCHL.	ILD EA,SUMME	100F	05 E0	
11.		IADD EA,ZAHL	1011	05 E2	SUMME:=SUMME+ZAHL
12.		IST EA,SUMME	1013	10 E0	
13.		IDLD A,ZAHL	1015	10 E2	ZAHL:=ZAHL-1
14.		IBNZ SCHL.	1017	17C F6	WIEDERHOLE DIE SCHLEIFE, WENN <A> NICHT 0
15.		ILD EA,SUMME	1019	05 E0	
16.		ICALL HEX-DEZ	101B	11C	
17.		ICALL ANZ EA	101C	119	ANZEIGE DER SUMME
18.		IBRA BEGINN	101D	174 E1	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZWSP	1FFD8	ZWISCHENSPEICHER
	1FFD9	
SUMME	1FFE0	IN DIESEN SPEICHERN STEHT
	1FFE1	DIE VIERSTELLIGE SUMME
ZAHL	1FFE2	HIER STEHT DIE ZAHL
	1FFE3	

Die Anweisungen aus dem Programmablaufplan werden in den mnemonischen Code übertragen. Es folgt die Festlegung der Adressen und die Übertragung in den Operationscode. Schließlich berechnet man die Displacements für die Sprungbefehle.

Die drei Befehle 7 bis 9 würden wir in einer ersten Erprobung des Programms nicht vorsehen. Sie sind hier eingesetzt, um später eine Ergänzung einfügen zu können.

11.2.5 Erprobung des Programms und Fehlersuche

Wir erproben jetzt unser Programm. Wir stellen fest, daß es für alle Zahlen bis 140 das richtige Ergebnis liefert.

Beispiele:

Zahl	Summe
15	120
100	5050
140	9870

Wir stellen fest, daß sich bei den eingegebenen Zahlen zwischen 141 und 256 die Anzeige „Error“ ergibt. Dafür sorgt der Call CALL HEX-DEZ. Mit Hilfe dieses Calls kann man nur eine Hexadezimalzahl bis 270F in eine Dezimalzahl verwandelt werden. Jede größere Hexadezimalzahl würde – als Dezimalzahl geschrieben – wenigstens fünfstellig sein (vgl. Abschnitt 8.4).

Wenn wir unser Programm weiter testen, stellen wir merkwürdige Dinge fest.

Beispiele:

Zahl	angezeigtes Ergebnis
257	257
258	515
280	6444
1030	6165

Wie kommt es zu diesen Anzeigen? Wenn man die Zahlen von 1 bis 258 addiert, erhält man sicher eine größere Summe als 515. Es müßte sich $\frac{1}{2} \cdot 258 \cdot 259 = 33\,411$ ergeben. Es müßte also die Anzeige „Error“ erscheinen und nicht die Summe 515. Wie können wir dem Fehler auf die Spur kommen?

Hier kann uns der Call „CALL TEST“ helfen. Wir können das Programm an verschiedenen Stellen unterbrechen und uns die Speicherinhalte ansehen.

Wenn wir den Breakpoint (Operationscode 1E) der Reihe nach in die Speicher 100B, 1017 und 1019 schreiben, können wir das Programm nacheinander nach Eingabe der Zahl 258 bis zu diesen drei Stellen laufen lassen. Wir kontrollieren so die Speicherinhalte vor Eintritt in die Schleife, nach einem Schleifendurchlauf und nach dem Verlassen der Schleife.

Zur Erinnerung die Tastenfolge für den ersten Breakpoint. Das Programm sei eingegeben. Die Anzeige sei

1 0 0 0 8 4

Tasten:

1 0 0 B

A→D 1 E

A→D RUN

2 5 8

RUN

A→D F F E 0

ME +

ME +

ME +

A→D 1 0 0 B

A→D 7 4

Anzeige:

1 0 0 B 7 4

1 0 0 B 1 E

0

2 5 8

1 0 0 B 1 E

F F E 0 0 0

F F E 1 0 0

F F E 2 0 2

F F E 3 0 1

1 0 0 B 1 E

1 0 0 B 7 4

NR.	MARKE	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
10.	SCHL.	ILD EA,SUMME	100F	85 E0	
11.		IADD EA,ZAHL	1011	85 E2	SUMME:=SUMME+ZAHL
12.		IST EA,SUMME	1013	8D E0	
13.		ILD A,ZAHL	1015	9D E2	ZAHL:=ZAHL-1
14.		IBNZ SCHL.	1017	7C F6	WIEDERHOLE DIE SCHLEIFE, WENN <A> NICHT 0
15.		ILD EA,SUMME	1019	85 E0	
16.		ICALL HEX-DEZ	101B	1C	
17.		ICALL ANZ EA	101C	19	ANZEIGE DER SUMME
18.		IBRA BEGINN	101D	74 E1	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZWSP	FFD8	ZWISCHENSPEICHER
SUMME	FFD9	
ZAHL	FFE0	IN DIESEN SPEICHERN STEHT DIE VIERSTELLIGE SUMME
	FFE1	
	FFE2	HIER STEHT DIE ZAHL
	FFE3	

Hierbei haben wir zum erstenmal einen weiteren Call benutzt:

BESCHREIBUNG	INMEM.CODE	OP.CODE	OPERATION
ICALL FEHLER	ICALL FEHLER	1A	ANZEIGE: "ERROR"

Der Call sorgt für die Anzeige des Wortes „Error“ (engl., Fehler). Diese Anzeige kann mit einer beliebigen Funktionstaste beendet werden.

eingeegebene Dezimalzahl: 258	Summe, Hex.		Zahl, Hex.	
	(FFE1)	(FFE0)	(FFE3)	(FFE2)
Vor Eintritt in die Schleife nach einem Schleifendurchlauf	00	00	01	02
nach Verlassen der Schleife	01	02	01	01
	02	03	01	00

Vor Eintritt in die Schleife ist die Summe 0, die eingegebene Zahl 102₁₆ (102₁₆ = 258₁₀). Nach einem Schleifendurchlauf ist die Summe 102₁₆, die Zahl 101₁₆. Wird im nächsten Schleifendurchlauf die Zahl 101₁₆ zur bisherigen Summe 102₁₆ addiert, so ergibt sich für die neue Summe 203₁₆ (203₁₆ = 515₁₀), also schon das Endergebnis unserer Rechnung. Die Zahl ist jetzt 100₁₆. Wieso wird hier schon die Schleife verlassen? Warum wird die Schleife nicht solange durchlaufen, bis die Zahl Null geworden ist? Die Antwort liegt auf der Hand, wenn man sich überlegt, was der DLD-Befehl bewirkt. Es wird nur das low order byte, das LOB der Zahl (und der Akkuinhalt) dekrementiert. Die Schleife wird verlassen, wenn der Akkuinhalt Null ist. Der Mikroprozessor macht also genau das, was wir ihm mit den Befehlen vorschreiben.

Damit unser Computer bei allen Fällen, bei denen eine Zahl größer als 256 eingegeben wird, auch „Error“ zeigt, können wir vor Eintritt in die Schleife drei Befehle einfügen. In diesen Fällen ist nach Umwandlung der eingegebenen Dezimalzahl in eine Hexadezimalzahl der Inhalt des Extension-Registers ungleich Null. Also fügen wir einen bedingten Sprung in Abhängigkeit vom Inhalt des E-Registers ein:

NR.	MARKE	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	ILD EA,=0	1000	84 00 00	SUMME:=0000
2.		IST EA,SUMME	1003	8D E0	
3.		ICALL EIN 4ST	1005	17	EINGABE DER ZAHL
4.		ILD EA,ZWSP	1006	85 D8	
5.		ICALL DEZ-HEX	1008	1B	
6.		IST EA,ZAHL	1009	8D E2	
7.		IXCH A,E	100B	01	
8.		IBZ SCHL.	100C	6C 01	
9.		ICALL FEHLER	100E	1A	

11.3 Aufgaben zu Kapitel 11

- 1.1 Welche drei 2-Byte-Befehle können den Befehl IL D A, ZAHL (Operationscode z. B. 95 E0) ersetzen?
- 1.2 Welche drei 2-Byte-Befehle können den Befehl DLD A, ZAHL (Operationscode z. B. 9D E0) ersetzen?
- 2. Wie sieht der Programmablaufplan für das erste Beispielprogramm im Abschnitt 11.1 aus?
- 3. Fragen zum 2. Beispielprogramm im Abschnitt 11.1
- 3.1 Welche Zahl wird bei Programmende angezeigt? Warum?
- 3.2 Wie oft werden die sechs NOP-Befehle ausgeführt?
- 3.3 Wie oft wird der Rücksprung zur Marke SCHL. ausgeführt?
- 4. Es soll begründet werden, warum nach dem Programm im Abschnitt 11.2.4 (vgl. auch Abschnitt 11.2.5) zu der eingegebenen Zahl n = 1030 das Ergebnis 6165 berechnet wird.
- 5. Das in Abschnitt 11.2.5 korrigierte Programm hat noch einen kleinen Schönheitsfehler. Bei der eingegebenen Zahl n = 0 wird auch „Error“ angezeigt. Wie kommt es dazu? Wie läßt sich das Programm so erweitern, daß in diesem Fall die Summe 0 angezeigt wird?

6.

ADRESSE	INHALT
1000	C4
1001	F0
1002	CD
1003	E0
1004	84
1005	00
1006	00
1007	8D
1008	E1
1009	9D
100A	E1
100B	7C
100C	FC
100D	9D
100E	E2
100F	6C
1010	02
1011	74
1012	F6
1013	95
1014	E0
1015	7C
1016	ED
1017	24
1018	CB
1019	03

In den Speichern mit den Adressen 1000 bis 1019 ist ein Programm eingegeben.

Die Aufgabe besteht darin, die Speicherinhalte zum Operationscode zusammenzufassen, die Befehle in den mnemonischen Code umzuschreiben und das Programm zu analysieren. Ob die Aufgabe nun sinnvoll ist oder nicht, das sei dahingestellt. Vielleicht hilft sie aber, sich mit den Befehlen weiter vertraut zu machen.

Wer es nicht abwarten kann, wer das Programm zunächst eintastet und startet, muß knapp 20 Sekunden warten, bis es etwas zu sehen gibt.

7. Das folgende Programm berechnet zu einer eingegebenen zweistelligen Dezimalzahl (0 bis 99) das Quadrat, also z. B. zu 40 die Quadratzahl $40^2 = 40 \cdot 40 = 1600$. Es ist nur der mnemonische Code angegeben. Die Aufgabe besteht darin, die Befehle in den Operationscode zu übersetzen, dann das Programm einzutasten und zu testen. Außerdem soll ein Programmablaufplan erstellt werden. Wie bei allen anderen Aufgaben auch, sollte man sich nicht zu früh Hilfe bei den Lösungen holen.

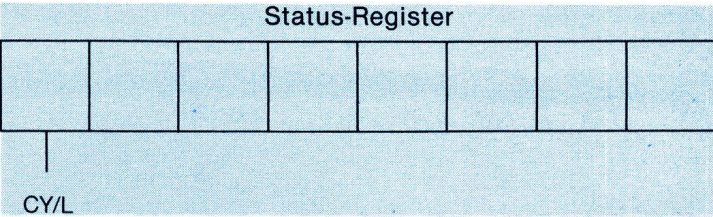
NR.	MARKE	MNEM. CODE
1.	BEGINN	ILD EA,=0
2.		IST EA,QUADRAT
3.		ICALL EIN 2ST
4.		ILD EA,ZWSP
5.		ICALL DEZ-HEX
6.		IST EA,FAKTOR
7.		IST A,ZAEHLER
8.	SCHL.	ILD EA,QUADRAT
9.		ADD EA,FAKTOR
10.		IST EA,QUADRAT
11.		IDLD A,ZAEHLER
12.		BNZ SCHL.
13.		ILD EA,QUADRAT
14.		ICALL HEX-DEZ
15.		ICALL ANZ EA
16.		IBRA BEGINN

12. Unterprogramme

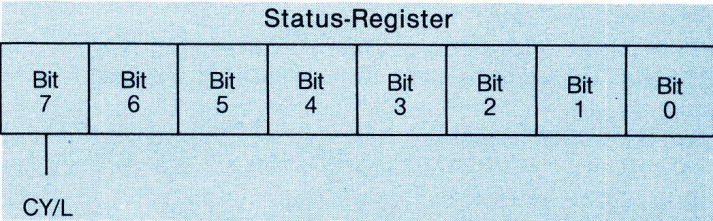
12.1 Bemerkungen zum Status-Register S

Wir wollen uns in diesem Kapitel mit der Unterprogrammtechnik beschäftigen. Wir wollen überlegen, was ein Unterprogramm ist und wie es in einem Hauptprogramm eingesetzt wird. Da wir bei unseren Untersuchungen das Status-Register S benutzen werden, sollen zunächst einige Bemerkungen hierzu gemacht werden.

Im Abschnitt 7.4 haben wir festgehalten, daß das Status-Register S eine Zusammenfassung von acht einzelnen Flags, von acht einzelnen 1-Bit-Registern mit unterschiedlicher Bedeutung ist. An der vordersten Stelle steht in diesem Register der Wert des CY/L-Flags.

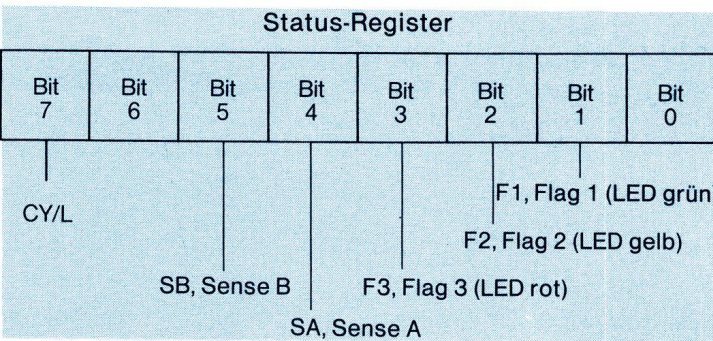


Es ist üblich, die acht Bit eines 8-Bit-Registers oder -Speichers mit 0 bis 7 durchzunummerieren (vgl. Spiel 4, Spiel 9 und die Beschriftung der LED-Reihe bei unserem Computer). Das hängt mit dem Dualsystem zusammen. Hat z. B. Bit 5 den Wert 1, so entspricht die dargestellte Zahl der Dezimalzahl $2^5 = 32$.



Das Bit 7 des Status-Registers zeigt also den Wert des CY/L-Registers an.

Wir haben uns schon mit fünf weiteren Flags des Status-Registers beschäftigt, ohne bisher den Zusammenhang zu diesem Register hergestellt zu haben. In den Kapiteln 3 und 6 haben wir die Eingabetasten (bzw. Eingabeleitungen) **SA** und **SB** benutzt; ebenso haben wir schon mehrfach die drei farbigen Leuchtdioden beobachtet, die – wie wir schon wissen – die Werte der drei Flags F1, F2 und F3 anzeigen. Diese fünf Flags sind neben dem CY/L-Flags sämtlich 1-Bit-Register im 8-Bit-Status-Register. Die Anordnung im S-Register zeigt folgendes Bild.



Die beiden Flags SA und SB dienen ausschließlich dazu, von außen (von der Peripherie) Informationen zu empfangen; sie werden auch als Nur-Lese-Flags bezeichnet. Im Gegensatz dazu könnte man die drei Flags F1, F2 und F3 als

Nur-Schreib-Flags bezeichnen. Sie dienen nur zur Ausgabe von Signalen.

Diese fünf Flags sind unmittelbar mit der Peripherie verbunden, wie wir es in den Schaltplänen im Abschnitt 3.6 angegeben haben. Die Zustände der ankommenden Signale stehen also stets in den Bits 4 und 5 des Status-Registers. Sie können jederzeit dort nachgesehen werden. Das Nachsehen geschieht so, daß der Inhalt des Status-Registers in den Akkumulator geladen wird. Jetzt kann der Akkuinhalt weiter untersucht werden.

Um über die drei Ausgabeleitungen bestimmte Informationen auszugeben, wird umgekehrt ein geeigneter Akkuinhalt zum Statusregister gebracht. Damit sind die drei Flags F1, F2 und F3 in bestimmter Weise gesetzt oder gelöscht. Für den Mikroprozessor ist hiermit seine Ausgabetätigkeit beendet. Von hier wird die Information über die angegebenen elektronischen Schaltungen weitergeleitet.

Für den Datentransport zwischen Akkumulator und Statusregister gibt es die beiden folgenden Befehle:

BESCHREIBUNG	MNEM. CODE	OP. CODE	OPERATION
ILADE DEN INHALT DES STATUS-REGISTERS IN DEN AKKUMULATOR	LD A,S	06	(A) := (S)
ILADE DEN INHALT DES AKKUMULATORS IN DAS STATUS-REGISTER	LD S,A	07	(S) := (A)

Die beiden Befehle bedürfen eigentlich keiner weiteren Erklärung. Es soll nur noch einmal daran erinnert werden, daß im mnemonischen Code erst das Register angegeben wird, in das geladen wird. Das Register, aus dem geladen wird, behält seinen Inhalt.

Wenn wir nur diese beiden Befehle sehen, könnten wir das Status-Register für ein normales 8-Bit-Register halten (wie etwa das Extension-Register), in dem auch vorübergehend der Akkuinhalt abgespeichert werden kann. Das ist aber nicht der Fall. Wir hatten schon mehrfach betont, daß das Status-Register lediglich eine Zusammenfassung von acht einzelnen Flags mit unterschiedlichen Funktionen ist. Restlos wird uns das folgende kleine Programm überzeugen:

NR.	MARKE	MNEM. CODE	ADR.	OP. CODE	BEMERKUNGEN
1.	BEGINN	LD A,=0FF	1000	04 FF	(A) := FF
2.		LD S,A	1002	07	(S) := (A)
3.		LD A,S	1003	06	(A) := (S)
4.		ICALL ANZ A	1004	18	ANZEIGE: (A)
5.		ICALL TEST	1005	1E	

Im ersten Befehl wird die Hexadezimalzahl FF (dual: 1111 1111) in den Akku geladen. Diese Zahl wird dann ins Status-Register gebracht. Im dritten Befehl wird der Inhalt vom S-Register wieder in den Akku geholt und anschließend angezeigt. Die Verblüffung ist groß. Wir erhalten nicht FF, sondern CF (dual: 1100 1111). Was ist passiert? Wir können die Frage untersuchen, indem wir den dritten Befehl durch den CALL TEST ersetzen:

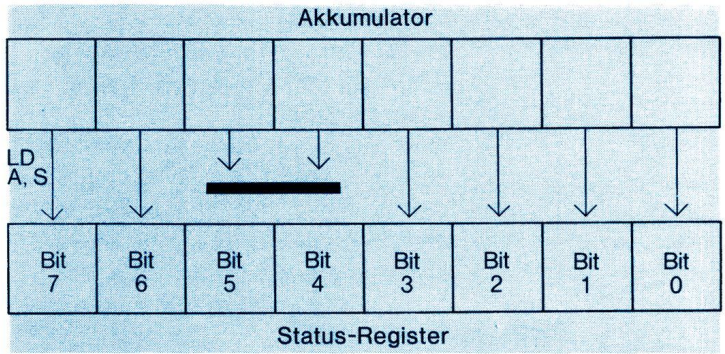
NR.	MARKE	MNEM. CODE	ADR.	OP. CODE	BEMERKUNGEN
1.	BEGINN	LD A,=0FF	1000	04 FF	(A) := FF
2.		LD S,A	1002	07	(S) := (A)
3.		ICALL TEST	1003	1E	

Wenn wir jetzt das Programm neu starten und uns bei der Anzeige

1 0 0 3 1 1 E

mit CPU, 5 den Inhalt des Status-Registers ansehen,

stellen wir fest, daß hier nur die Zahl CF steht. Die beiden Flags Sense A und Sense B sind eben Nur-Lese-Flags für Informationen von außen. Sie lassen sich durch den Befehl LD S, A nicht setzen. Das müssen wir uns bei diesem Befehl merken. Es wird nicht der gesamte Akkuinhalt in das Status-Register geladen, sondern nur der Inhalt von Bit 7, Bit 6 und Bit 3 bis Bit 0.



Auf der anderen Seite haben wir sicher gemerkt, daß die drei farbigen Leuchtdioden bei unseren Testprogrammen angeschaltet wurden. Bit 1, Bit 2 und Bit 3 im Status-Register können durch den Befehl LD A, S gesetzt werden. Diese drei Flags behalten dann ihren Wert, bis eine neue Information kommt, z. B. durch eine neue Datenübertragung vom Akku:

NR.	MARKE	MNEM. CODE	ADR.	OP. CODE	BEMERKUNGEN
1.	BEGINN	LD A,=0E	1000	04 0E	
2.		LD S,A	1002	07	(S) := 0E
3.		ICALL ANZ A	1003	18	
4.		LD A,=0	1004	04 00	
5.		LD S,A	1006	07	(S) := 00
6.		ICALL ANZ A	1007	18	
7.		IBRA BEGINN	1008	74 F6	

Eine letzte Bemerkung zu diesem Thema: Wir sollten das Bit 0 im Status-Register nicht setzen. Eine 1 an dieser Stelle kann ungeahnte Folgen haben. Wir kommen im Kapitel 17 darauf zurück.

12.2 Wir programmieren eine Pause

Wir wollen das Prinzipielle der Unterprogramm-Technik im nächsten Abschnitt an einem einfachen Blink-Programm untersuchen. Wir wissen schon, wie wir die drei farbigen Leuchtdioden ein- und ausschalten können. Wir brauchen nur die Bits 1, 2 und 3 im Status-Register zu setzen oder zu löschen. Wenn wir das folgende kurze Programm eintasten und starten, werden wir feststellen, daß unsere Leuchtdioden noch lange nicht blinken.

NR.	MARKE	MNEM. CODE	ADR.	OP. CODE	BEMERKUNGEN
1.	BEGINN	LD A,=0E	1000	04 0E	
2.		LD S,A	1002	07	(S) := 0E
3.		LD A,=0	1003	04 00	
4.		LD S,A	1005	07	(S) := 00
5.		IBRA BEGINN	1006	74 F8	SPRUNG ZUM BEGINN

Die Leuchtdioden werden zwar mit den ersten beiden Befehlen ein-, mit den nächsten beiden Befehlen ausgeschal-

tet. Das Ein- und Ausschalten wird auch ständig wiederholt, geschieht aber in so schneller Folge, daß wir die Leuchtdioden ständig leuchten sehen.

Was können wir tun? Es muß nach dem Einschalten der Leuchtdioden und nach dem Ausschalten eine Pause gemacht werden. Erst dann werden die LEDs auch für unsere Augen blinken. Nur, der Mikroprozessor kennt keine Pausen. Wir müssen ihm also zwischendurch Arbeit verschaffen, mit der er sich beschäftigen kann, und bei der der Leuchtzustand der Dioden nicht verändert wird.

Wir wollen es dadurch versuchen, daß der Mikroprozessor zwischendurch jeweils einen Speicherinhalt 256mal dekrementiert:

INR.	MARKE	MNEM. CODE	ADR.	OP. CODE	BEMERKUNGEN
1.		BEGINN	ILD A,=0E	1000 1C4 0E	
2.			ILD S,A	1002 107	<S>:=0E
3.		IPAUSE1	ILD A,=0	1003 1C4 00	
4.			IST A,ZAHL	1005 1CD E0	ZAHL:=0
5.		WDH1	IDL A,ZAHL	1007 19D E0	ZAHL:=ZAHL-1
6.			IBNZ WDH1	1009 17C FC	SPRUNG NACH WDH1
7.			ILD A,=0	100B 1C4 00	
8.			ILD S,A	100D 107	<S>:=00
9.		IPAUSE2	ILD A,=0	100E 1C4 00	
10.			IST A,ZAHL	1010 1CD E0	ZAHL:=0
11.		WDH2	IDL A,ZAHL	1012 19D E0	ZAHL:=ZAHL-1
12.			IBNZ WDH2	1014 17C FC	SPRUNG NACH WDH2
13.			IBRA BEGINN	1016 174 E8	SPRUNG ZUM BEGINN

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZAHL	FFE0	

Immer noch kein Erfolg! Wir sehen wieder nur das ständige Leuchten der Dioden. Trotzdem sind wir schon einen gewaltigen Schritt weiter. Auch wenn wir es mit dem Auge nicht feststellen können, die drei Flags (F1, F2 und F3) werden jetzt nur noch gut 100mal in jeder Sekunde gesetzt und gelöscht. Vorher passierte das etwa 300mal so oft. Wir machen also die Pausen noch größer. Wir lassen den Mikroprozessor nicht 256mal dekrementieren, sondern jeweils 2560mal:

INR.	MARKE	MNEM. CODE	ADR.	OP. CODE	BEMERKUNGEN
1.		BEGINN	ILD A,=0E	1000 1C4 0E	
2.			ILD S,A	1002 107	<S>:=0E
3.		IPAUSE1	ILD EA,=0A00	1003 184 00 0A	
4.			IST EA,ZAHL	1005 18D E0	ZAHL 1:=00; ZAHL 2:=0A
5.		WDH1	IDL A,ZAHL 1	1008 19D E0	ZAHL 1:=ZAHL 1 - 1
6.			IBNZ WDH1	100A 17C FC	SPRUNG NACH WDH1
7.			IDL A,ZAHL 2	100C 19D E1	ZAHL 2:=ZAHL 2 - 1
8.			IBNZ WDH1	100E 17C F8	SPRUNG NACH WDH1
9.			ILD A,=0	1010 1C4 00	
10.			ILD S,A	1012 107	<S>:=00
11.		IPAUSE2	ILD EA,=0A00	1013 184 00 0A	
12.			IST EA,ZAHL	1015 18D E0	ZAHL 1:=00; ZAHL 2:=0A
13.		WDH2	IDL A,ZAHL 1	1018 19D E0	ZAHL 1:=ZAHL 1 - 1
14.			IBNZ WDH2	101A 17C FC	SPRUNG NACH WDH2
15.			IDL A,ZAHL 2	101C 19D E1	ZAHL 2:=ZAHL 2 - 1
16.			IBNZ WDH2	101E 17C F8	SPRUNG NACH WDH2
17.			IBRA BEGINN	1020 174 DE	SPRUNG ZUM BEGINN

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZAHL	FFE0	
ZAHL 1	FFE1	
ZAHL 2	FFE1	

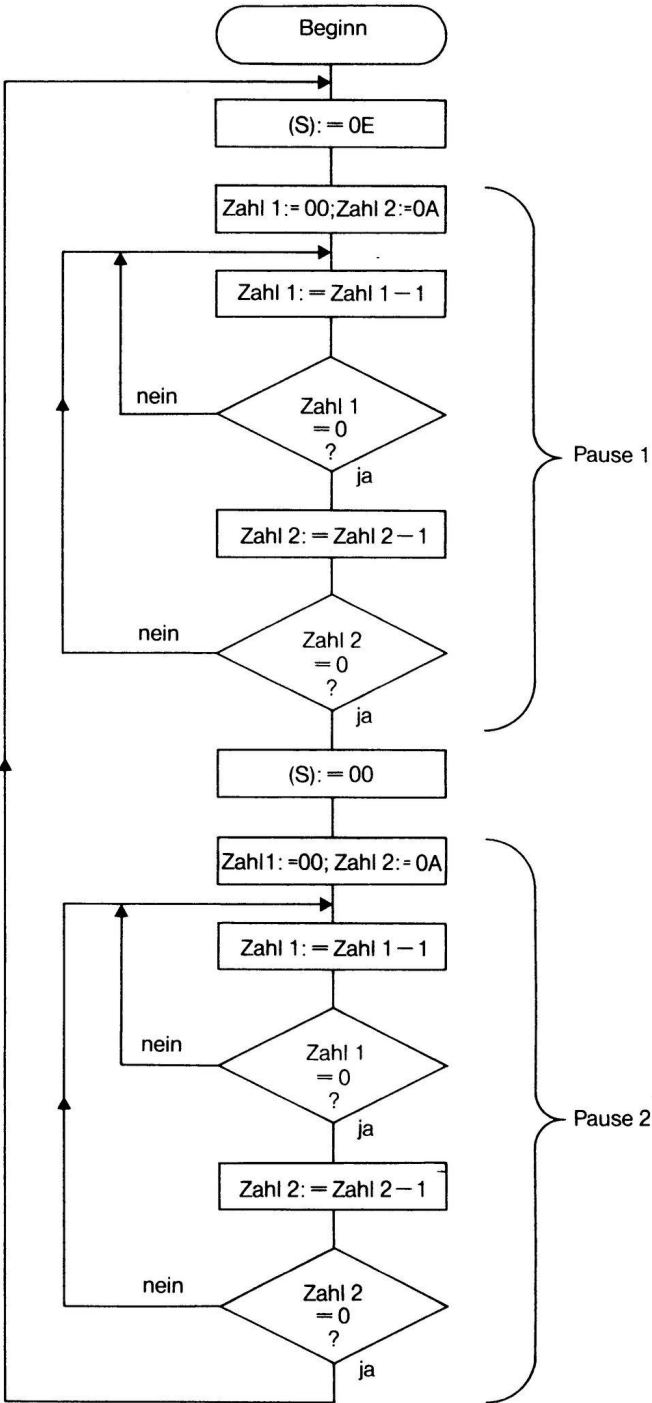
Wenn im Pausenprogramm der Inhalt vom Speicher FFE0 256mal dekrementiert wurde, wird (FFE1) um eins vermindert. Das Dekrementieren von (FFE0) wird wiederholt, und zwar zehnmal, bis auch (FFE1) Null geworden ist. Jetzt erkennen wir endlich ein Flackern der Leuchtdioden. Die drei Flags werden jetzt etwa 10mal pro Sekunde gesetzt und gelöscht. Nebenbei sollten wir unseren Mikroprozessor kurz bestaunen, daß er hier über 100 000 Befehle pro Sekunde

bearbeitet. In einer zehntel Sekunde werden zwei Pausen gemacht. In jedem Pausenprogramm werden über 2500 Dekrementier- und über 2500 Sprungbefehle ausgeführt: $10 \cdot 2 \cdot (2500 + 2500) = 100\,000$.

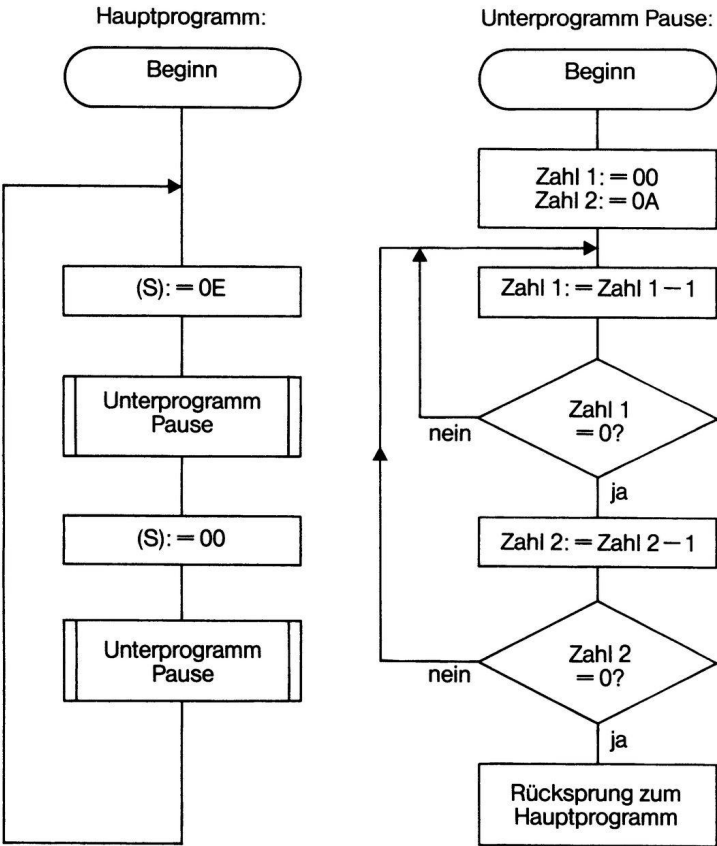
Jetzt ist es kein Problem mehr, das Flackern so zu ändern, daß es zu einem Blinken wird: Wir überschreiben in den Speichern mit den Adressen 1005 und 1015 den Inhalt 0A durch 00. Jetzt wird in jedem Pausenprogramm das Dekrementieren des Speichers FFE0 sogar 65536mal ausgeführt ($256 \cdot 256 = 65\,536$).

12.3 Wir schreiben unser erstes Unterprogramm

Wir haben im letzten Programm den Pausenteil zweimal schreiben müssen. Auch das Ändern der Pausenlänge mußte an zwei Stellen geschehen. Die Befehle 11 bis 16 sind völlig identisch mit den Befehlen 3 bis 8. Das wird auch im Programmablaufplan deutlich.



An diesem Beispiel wollen wir uns die Vorteile der Unterprogrammtechnik klarmachen. Der Programmteil, der mehrfach benutzt wird, kann gesondert als Unterprogramm geschrieben werden. Dieses Unterprogramm wird dann vom Hauptprogramm bei Bedarf aufgerufen. Der Programmablaufplan soll das zunächst verdeutlichen:



NR.	MARKE	MNEM. CODE	ADR.	OP. CODE	BEMERKUNGEN
1.	BEGINN	ILD A,=0E	1100	1C 0E	HAUPTPROGRAMM
2.		ILD S,A	1102	10 7	(S):=0E
3.		IJSR PAUSE	1103	12 0F 10	SPRUNG ZUM UNTERPROGRAMM
4.		ILD A,=0	1106	1C 00	
5.		ILD S,A	1108	10 7	(S):=0
6.		IJSR PAUSE	1109	12 0F 10	SPRUNG ZUM UNTERPROGRAMM
7.		IBRA BEGINN	110C	17 4 F2	ISPRUNG ZUM BEGINN
150.	PAUSE	ILD EA,=0A00	1100	18 00 0A	UNTERPROGRAMM
151.		IST EA,ZAHL	1103	18 D E0	(ZAHL 1:=00; ZAHL 2:=0A
152.	WDH	IDL A,ZAHL 1	1105	1D E0	(ZAHL 1:=ZAHL 1 - 1
153.		IBNZ WDH	1107	17 C FC	ISPRUNG NACH WDH
154.		IDL A,ZAHL 2	1109	1D E1	(ZAHL 2:=ZAHL 2 - 1
155.		IBNZ WDH	110B	17 C F8	ISPRUNG NACH WDH
156.		IRET	110D	15 C	RUECKSPRUNG ZUM HAUPTPROGRAMM
DATENSPEICHER, BEZEICHNUNG			ADR.	BEMERKUNGEN	
ZAHL			11FE0		
ZAHL 1			11FE1		
ZAHL 2			11FE1		

Einige Bemerkungen dazu: das Programm ist gegenüber dem vorangehenden kürzer. Es werden drei Befehle und sechs Speicherplätze weniger benötigt. Wir haben jetzt die Möglichkeit, die Blinkgeschwindigkeit durch eine Änderung vorzunehmen. Es ist nur der Inhalt 0A vom Speicher mit der Adresse 1102 durch eine andere Zahl zu überschreiben.

Es wird auch noch ein Vorteil deutlich, der allgemein für die Unterprogrammtechnik gilt. Die beiden Teile (Hauptprogramm und Unterprogramm) können in ganz unterschiedlichen Speicherbereichen stehen. Wir haben willkürlich das Unterprogramm bei der Adresse 1100 begonnen. Durch diese Trennung der beiden Programme kann jedes Programm für sich in weiten Grenzen geändert werden, ohne daß das andere Programm dadurch betroffen ist. Wir können z. B. auch in die Speicher ab Adresse 1000 (bis Adresse 10FF) ein ganz anderes Hauptprogramm eingeben, in dem das schon getestete Unterprogramm PAUSE ebenfalls benutzt wird.

Eine Bemerkung noch zur Sprungadresse: das Unterprogramm beginnt bei der Adresse 1100. Wir wissen vom JMP-Befehl (vgl. Abschnitt 10.2), daß der Sprung darin besteht, den Inhalt des Programmzählers zu ändern. Hier wird also beim Aufruf des Unterprogramms die Adresse 10FF in das PC-Register (LOB FF; HOB 10) geladen. Anschließend wird dieser Inhalt noch um eins erhöht: 10FF + 1 = 1100. Das erste Byte des nächsten Befehls wird aus dem Speicher mit der Adresse 1100 geholt.

Ein Problem bleibt noch offen: Woher weiß der Mikroprozessor beim Return-Befehl, an welche Stelle er ins Hauptprogramm zurückspringen soll? Einmal folgt nach dem Unterprogramm der vierte, einmal folgt der siebente Befehl. Mit diesem Problem werden wir uns ausführlich in den folgenden Abschnitten beschäftigen.

Das Hauptprogramm wird viel kürzer und übersichtlicher. Der Programmteil Pause muß nur einmal im Unterprogramm ausführlich dargestellt werden.

Was wir hier mit dem Programmablaufplan angedeutet haben, läßt sich ganz entsprechend in ein Programm übertragen. Zwei Dinge müssen dazu nur geklärt werden. Einmal muß vom Hauptprogramm das Unterprogramm aufgerufen werden, zum anderen muß nach Ablauf des Unterprogramms ein Rücksprung ins Hauptprogramm erfolgen. Dafür kennt unser Mikroprozessor zwei spezielle Befehle: den JSR-Befehl (engl. jump to subroutine = springe ins Unterprogramm) und den RET-Befehl (engl. return = zurückkehren). Der JSR-Befehl ist wie der JMP-Befehl ein 3-Byte-Befehl. Er wird auch unbedingt ausgeführt. Die Sprungadresse wird auch mit LOB und HOB vollständig angegeben (vgl. Abschnitt 10.2). Der RET-Befehl ist ein 1-Byte-Befehl.

Bevor wir uns dazu weitere Einzelheiten ansehen, wollen wir zunächst unser Blinkprogramm umschreiben.

12.4 Der Stack oder der Stapelspeicher

Wir wollen uns zunächst das offene Problem noch einmal an dem Programm aus dem letzten Abschnitt veranschaulichen:

Hauptprogramm

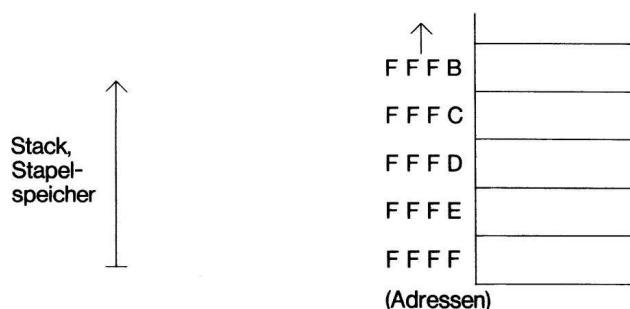
INR.	INMEM.	CODE
1.	ILD	A,=0E
2.	ILD	S,A
3.	JSR	PAUSE
4.	ILD	A,=0
5.	ILD	S,A
6.	JSR	PAUSE
7.	IBRA	BEGINN

Unterprogramm

150.	ILD	EA,=0A00
151.	IST	EA,ZAHL
152.	IDLD	A,ZAHL 1
153.	IBNZ	WDH
154.	IDLD	A,ZAHL 2
155.	IBNZ	WDH
156.	IRET	

Im dritten Befehl erfolgt ein Sprung zum Unterprogramm, anschließend muß zum vierten Befehl zurückgesprungen werden. Erfolgt der Sprung zum Unterprogramm im sechsten Befehl, muß zum siebenten Befehl zurückgesprungen werden. Der Mikroprozessor muß sich also die Stelle, bzw. die Adresse im Hauptprogramm merken, von der er zum Unterprogramm gesprungen ist. Dann findet er auch nach Bearbeitung des Unterprogramms wieder an die richtige Stelle zurück. Beim JSR-Befehl wird also nicht nur die Sprungadresse (die Zieladresse für das Unterprogramm) in den Programmzähler geladen (wie beim JMP-Befehl), es wird zusätzlich der Inhalt, den der Programmzähler zuletzt im Hauptprogramm hatte, abgespeichert. Dieses Speichern geschieht im Stack (engl., Stapelspeicher). Beim Return-Befehl am Ende des Unterprogramms holt sich der Mikroprozessor die im Stack abgelegte Adresse für den Rücksprung; anders ausgedrückt: die im Stack gespeicherte Adresse wird wieder in den Programmzähler geschrieben. Dann wird das Programm automatisch an der richtigen Stelle im Hauptprogramm fortgesetzt.

Der Stapelspeicher besteht nicht aus speziellen Registern, wie etwa die Register A, E, S oder PC. Für den Stapelspeicher können wir einen beliebigen Teil des normalen Speicherbereichs vorsehen. Eine Adresse muß daher in zwei 8-Bit-Speichern abgelegt werden. Wenn keine besonderen Gründe dagegen sprechen, wird der Stack am Ende des adressierbaren Speicherbereichs liegen, also bei den Adressen FFFF, FFFE, FFFD, Nach Betätigung der **RS**-Taste liegt der Stack grundsätzlich an dieser Stelle.



Wir wollen diese Aussagen zunächst am Computer überprüfen. Wir tasten wieder unser Blinkprogramm aus Abschnitt 12.3 ein und überschreiben bei Adresse 1100 den Inhalt 84 mit 1E (Breakpoint). Wir starten das Programm. Die drei farbigen Leuchtdioden werden angeschaltet. Der Computer meldet sich mit der Anzeige

1 1 0 0 1 E

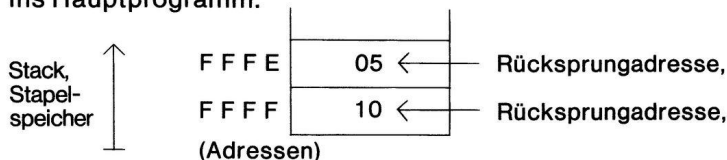
Wir sehen uns den Stapelspeicher an.

A → D, F, F, F, F
ME-

Anzeige

F F F F 1 0
F F F E 0 5

Im Stack steht die Adresse 1005, die Rücksprungadresse ins Hauptprogramm.



An diesem Beispiel wollen wir uns die Vorgänge noch einmal ganz ausführlich klarmachen.

Der zweite Befehl in unserem Programm sei bearbeitet. Der Programmzähler zeigt auf die Adresse 1003, auf den Speicher, in dem das erste Byte des JSR-Befehls steht.

(PC)	Was tut der Mikroprozessor?	(PC): = (PC) + 1
1003	Der Inhalt des Speichers 1003 wird geladen. An dem geladenen Byte 20 erkennt der Mikroprozessor, daß es sich um den JSR-Befehl handelt und daß es ein 3-Byte-Befehl ist. Er benötigt zwei weitere Bytes	1004
1004	Das LOB FF der Sprungadresse wird geladen	1005
1005	Das HOB 10 der Sprungadresse wird geladen. Jetzt hat der Mikroprozessor die vollständige Information für den JSR-Befehl. Der momentane Inhalt des PC-Registers wird zum Stack gebracht: (FFFF): = 10, HOB von (PC) (FFFE): = 05, LOB von (PC). Jetzt wird die Sprungadresse in den Programmzähler geladen: (PC): = 10FF	1100
1100	Das erste Byte des ersten Befehls im Unterprogramm wird geladen.	1101
:	:	:
:	Das Unterprogramm wird bearbeitet	:
:	:	110D
110D	Der Inhalt des Speichers 110D wird geladen. Aus dem geladenen Byte 5C erkennt der Mikroprozessor, daß es sich um den 1-Byte-Befehl RET handelt. Er holt vom Stack zunächst das LOB 05, dann das HOB 10. Diese Adresse 1005 wird in den Programmzähler geschrieben: (PC): = 1005	1006
1006	Das erste Byte vom 4. Befehl im Hauptprogramm wird geladen.	1007

Beim zweiten Aufruf des Unterprogramms im Befehl 6 funktioniert das Verfahren in der gleichen Weise. Wir betätigen die Taste **RS**, überschreiben die drei Bytes des ersten JSR-Befehls in den Speichern 1003, 1004 und 1005 durch drei NOP-Befehle (Operationscode: 00) und starten das Programm neu. Der Computer meldet sich wieder mit der Anzeige

1 1 0 0 1 E

Im Stack steht jetzt in den Speicherstellen FFFF und FFFE die neue Rücksprungadresse 100B. Der siebente Befehl im Hauptprogramm beginnt bei Adresse 100C.

Jetzt haben wir auf mehreren Seiten den Begriff Stapelspeicher benutzt. Aber, was wird hier eigentlich gestapelt? Bislang standen im Stack immer nur die beiden Bytes einer Adresse, die für den Rücksprung am Ende des Unterprogramms benötigt wurden. Unser Stapel im Stack war bislang also nur zwei Bytes hoch. Im nächsten Abschnitt werden wir höhere Stapel bilden. Dann wird die Bezeichnung Stapelspeicher wahrscheinlich verständlicher.

12.5 Verschachtelung von Unterprogrammen

Bei umfangreicheren Programmen kommt es häufig vor, daß in einem Hauptprogramm mehrere Unterprogramme eingesetzt werden. Es kann auch von einem Unterprogramm ein weiteres Unterprogramm aufgerufen werden. Man spricht dann von einer Verschachtelung von Unterprogrammen.

Das folgende Beispiel soll das verdeutlichen. Es ist sicher nicht erforderlich, dieses relativ einfache Blinkprogramm mit Hauptprogramm und drei Unterprogrammen zu schreiben; wir wollen uns aber gerade das Verschachteln von Unterprogrammen ansehen.

Zunächst des Programm:

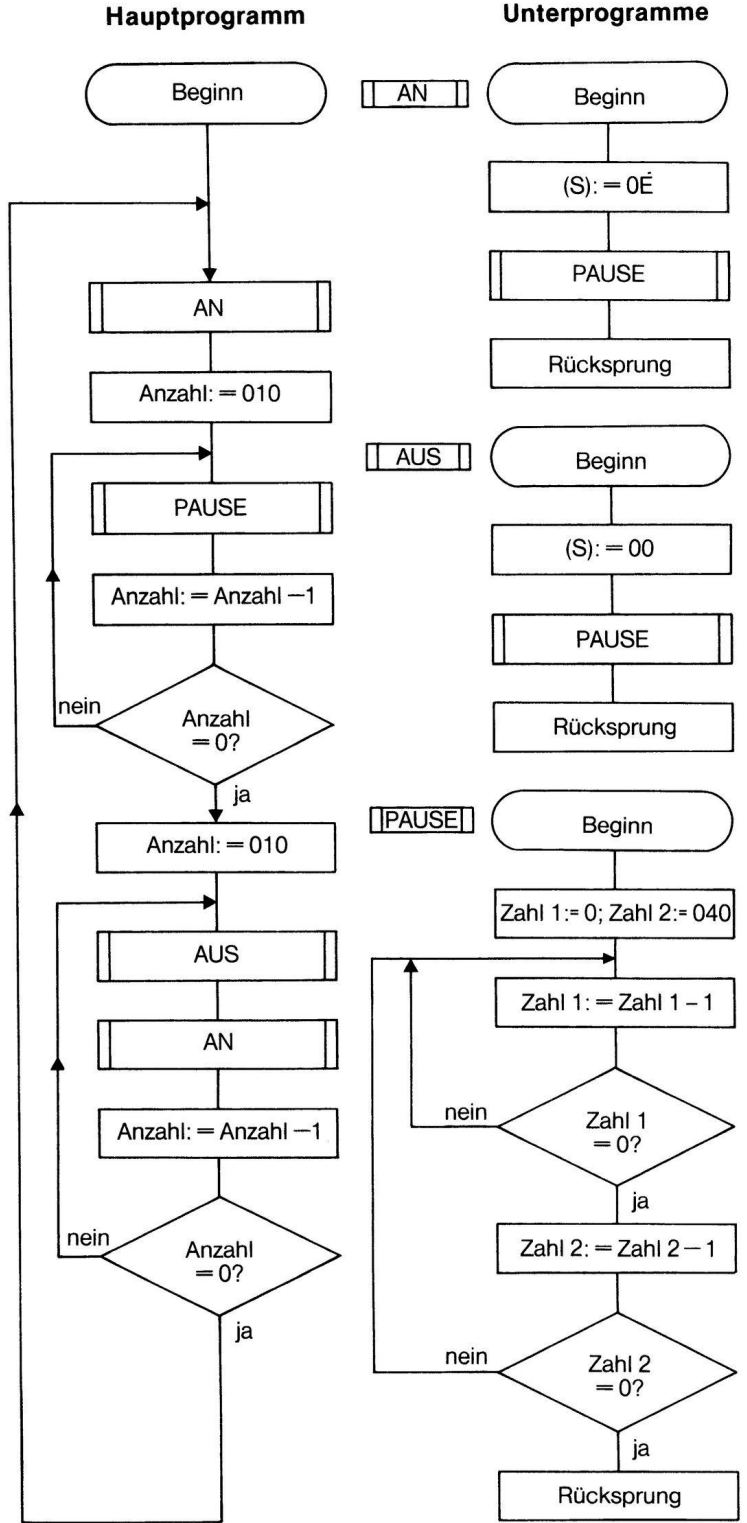
INR.	MARKE	INHEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	JSR AN	1000	20 5F 10	H A U P T P R O G R A M M
2.		LD A,=010	1003	1C4 10	
3.		ST A,ANZAHL	1005	1CD E2	
4.	WDH1	JSR PAUSE	1007	120 FF 10	
5.		LDL A,ANZAHL	100A	19D E2	
6.		BNZ WDH1	100C	17C F9	
7.		LD A,=010	100E	1C4 10	
8.		ST A,ANZAHL	1010	1CD E2	
9.	WDH2	JSR AUS	1012	120 7F 10	
10.		JSR AN	1015	120 5F 10	
11.		LDL A,ANZAHL	1018	19D E2	
12.		BNZ WDH2	101A	17C F6	
13.		BRA BEGINN	101C	174 E2	
30.	AN	LD A,=0E	1060	1C4 0E	UNTERPROGRAMM: A N
31.		LD S,A	1062	107	
32.		JSR PAUSE	1063	120 FF 10	
33.		RET	1066	15C	
40.	AUS	LD A,=0	1080	1C4 00	UNTERPROGRAMM: A U S
41.		LD S,A	1082	107	
42.		JSR PAUSE	1083	120 FF 10	
43.		RET	1086	15C	
50.	PAUSE	LD EA,=04000	1100	184 00 40	UNTERPROGRAMM: P A U S E
51.		ST EA,ZAHL	1103	18D E0	
52.	WDH	LDL A,ZAHL 1	1105	19D E0	
53.		BNZ WDH	1107	17C FC	
54.		LDL A,ZAHL 2	1109	19D E1	
55.		BNZ WDH	110B	17C F8	
56.		RET	110D	15C	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZAHL	FFE0	
	FFE1	
ZAHL 1	FFE0	
ZAHL 2	FFE1	
ANZAHL	FFE2	

Das Unterprogramm PAUSE ist bis auf die Pausenlänge identisch mit dem entsprechenden Programmteil im Abschnitt 12.3. Die beiden anderen Unterprogramme AN und AUS sorgen dafür, daß die Leuchtdioden an- oder ausgeschaltet werden; dann rufen sie das Unterprogramm PAUSE auf.

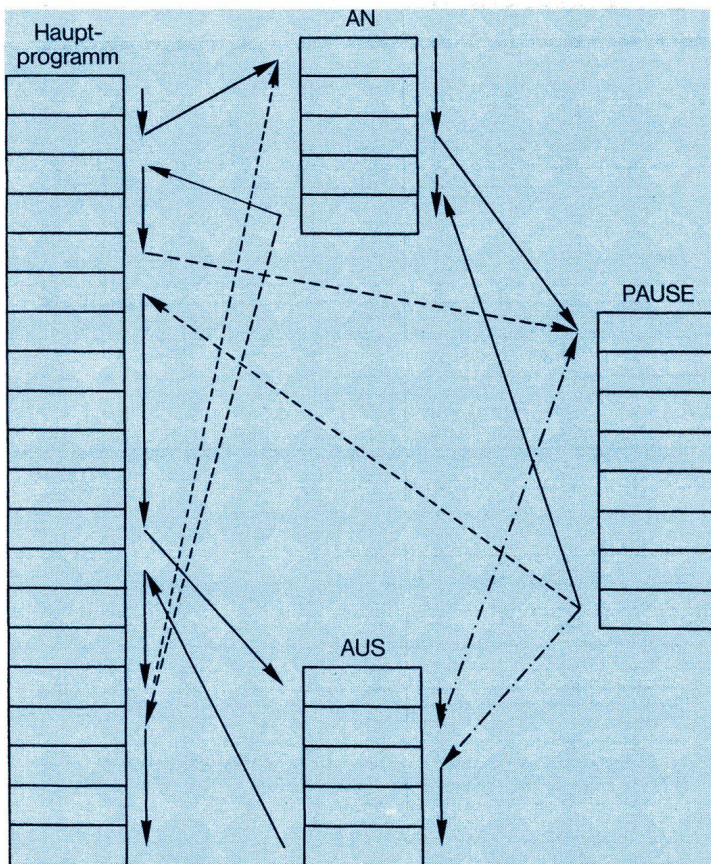
Wenn wir das gesamte Programm starten, sehen wir, daß die Leuchtdioden zunächst eine längere Zeit leuchten, dann beginnen sie zu blinken. Diese beiden Teile werden dann im Wechsel wiederholt.

Der Programmablaufplan dazu:



Im Hauptprogramm wird an vier Stellen ein Unterprogramm aufgerufen. In den beiden Unterprogrammen AN und AUS erfolgt ebenfalls ein Unterprogrammaufruf. Folgende schematische Darstellung kann die Verschachtelung der hier benutzten Unterprogramme und den Weg durch die Programmteile deutlich machen.

Wir sehen, wie verzwick die Geschichte ist. Das Unterprogramm AN wird an zwei Stellen vom Hauptprogramm aufgerufen. Am Ende dieses Unterprogramms erfolgt der Rücksprung an die richtige Stelle des Hauptprogramms. Das Unterprogramm PAUSE wird sogar von drei verschiedenen Stellen aufgerufen. Es erfolgt auch hier der Rücksprung immer dorthin, von wo das Unterprogramm aufgerufen wurde.



dem Stack geholt und in den Programmzähler PC geladen. Nach Erhöhung des PC-Inhalts um eins wird der nächste Befehl bei Adresse 1066 bearbeitet. Das ist auch wieder ein Return-Befehl. Es wird die Adresse 1002 aus dem Stack in den Programmzähler geladen und um eins erhöht. Jetzt erfolgt der nächste Befehl bei Adresse 1003.

12.6 Der Stackpointer

Eine letzte Frage ist offen. Woher weiß der Mikroprozessor, wie hoch der Stapel im Stapelspeicher zur Zeit ist? Diese Information braucht der Prozessor, wenn er eine weitere Rücksprungadresse zusätzlich auf dem Stapel ablegt; er braucht sie, wenn er die oberste Adresse wieder vom Stapel holt.

Hierzu wird der Pointer 1 oder der Stackpointer SP benutzt. Er zeigt auf die oberste Speicherzelle, die im Stack belegt ist. Bitte nicht verwechseln: die oberste belegte Speicherzelle im Stack hat von allen belegten Speichern im Stack die niedrigste Adresse. Wir müssen hier auch zwei Arten von Adressen unterscheiden: Im Stack sind u. U. Adressen für Rücksprünge gespeichert. Die Speicher des Stacks haben ihrerseits als normale Speicher Adressen FFFF, FFFE, ...; auf diese Adressen zeigt der Stackpointer. Wenn unser Programm – nach Betätigung der Taste **RS** – bis zum Breakpoint im Unterprogramm PAUSE gelaufen ist, können wir uns mit **CPU, 1** den Inhalt des Stackpointers (oder des Pointers 1) ansehen. Er zeigt auf die Adresse FFFC, (SP) = FFFC.

SP →	FFFC	65
	FFFD	10
	FFFE	02
	FFFF	10
	(Adressen)	

Wir können dem Mikroprozessor dankbar sein, daß er es uns abnimmt, jeweils die richtige Rücksprungadresse zu finden. Wir schreiben nur an das Ende eines Unterprogramms den Return-Befehl. Um die richtige Ausführung brauchen wir uns dann nicht mehr zu kümmern.

Wir wollen uns hier noch einmal den Stack ansehen, wenn nacheinander zwei Unterprogrammaufrufe durchgeführt wurden. Wir überschreiben wieder in Speicher 1100 den Inhalt 84 durch 1E (Breakpoint) und starten das Programm. Der Computer meldet sich mit der Anzeige

1 1 0 0 1 E

Wir sehen uns den Stack an.

Stack, Stapel- speicher	↑		
	FFFC	65	Rücksprungadresse 2, LOB
	FFFD	10	Rücksprungadresse 2, HOB
	FFFE	02	Rücksprungadresse 1, LOB
	FFFF	10	Rücksprungadresse 1, HOB
	(Adressen)		

Der Mikroprozessor hat hier schon einen höheren Stapel gebildet. Die Bezeichnung Stapelspeicher gewinnt an Bedeutung. Hier sind jetzt zwei Rücksprungadressen abgespeichert. Wir wollen den Weg bis hierhin verfolgen. Im 1. Befehl des Hauptprogramms wurde ein Sprung zum Unterprogramm AN durchgeführt. Die Rücksprungadresse 1002 wurde auf den Stack geschrieben. Jetzt wurden die Befehle 30. und 31. ausgeführt. Beim anschließenden Sprung zum Unterprogramm PAUSE wurde zusätzlich die Rücksprungadresse 1065 auf den Stack geschrieben. Und wie würde es jetzt weitergehen? Am Ende des Programms PAUSE erfolgt der erste Rücksprung. Die Adresse 1065 wird von oben aus

Beim Abspeichern auf den Stack wird der Inhalt des Stackpointers automatisch erniedrigt; im umgekehrten Fall wird (SP) wieder erhöht, wenn Daten vom Stack geholt werden. Wir können jetzt die beiden Befehle vollständig beschreiben, die wir in diesem Kapitel kennengelernt haben.

BESCHREIBUNG	INMEM. CODE	OP. CODE	OPERATION
SPRINGE ZUM UNTERPROGRAMM "MARKE"	JSP MARKE	120 YY XX	$\langle \langle SP \rangle - 1 \rangle := \langle PC \rangle, \text{HOB}$ $\langle \langle SP \rangle - 2 \rangle := \langle PC \rangle, \text{LOB}$ $\langle SP \rangle := \langle SP \rangle - 2$ $\langle PC \rangle := \text{XXYY}$
SPRINGE ZURUECK ZUM HAUPTPROGRAMM	RET	15C	$\langle PC \rangle, \text{LOB} := \langle \langle SP \rangle \rangle$ $\langle PC \rangle, \text{HOB} := \langle \langle SP \rangle + 1 \rangle$ $\langle SP \rangle := \langle SP \rangle + 2$

Ein Hinweis noch zu den Bezeichnungen bei der Beschreibung der Operationen. SP ist der Stackpointer. Er ist ein 16-Bit-Register. Mit (SP) wird der Inhalt des Stackpointers bezeichnet, also eine Adresse, eine vierstellige Hexadezimalzahl, z. B. (SP) = FFFC. (FFFC) oder ((SP)) ist dann der Inhalt des Speichers mit den Adressen FFFC, der Inhalt des Speichers, auf den der Stackpointer zeigt. Dieser Inhalt ist eine zweistellige Hexadezimalzahl, z. B. ein Byte einer Rücksprungadresse. Mit (PC), HOB und (PC), LOB sind das high order byte und das low order byte des Inhalts vom Programmzähler bezeichnet.

Angenommen, der Stackpointer zeigt auf die Adresse FFFA: (SP) = FFFA. Wird jetzt ein weiterer JSR-Befehl ausgeführt, so wird der PC-Inhalt in den beiden Speichern (SP) - 1 = FFF9 und (SP) - 2 = FFF8 abgelegt. (SP) wird um zwei erniedrigt, der Stackpointer zeigt anschließend auf die Adresse FFF8.

Wird bei (SP) = FFFA ein RET-Befehl ausgeführt, so wird der Programmzähler PC mit dem Inhalt der beiden Speicher (SP) = FFFA und (SP) + 1 = FFFB geladen, (SP) wird um zwei erhöht, der Stackpointer zeigt anschließend auf die Adresse FFFC.

Diese Zusammenhänge sind sicher nicht ganz einfach. Eventuell sollte man das Kapitel noch einmal durchlesen. Zwei abschließende Bemerkungen noch: Nach der Betätigung der **RS**-Taste zeigt der Stackpointer grundsätzlich auf die Adresse 0000. Folgt nach dem Start eines Programms jetzt ein JSR-Befehl, so werden die abzuspeichernden Daten in die Speicher (SP) - 1 = 0000 - 1 = FFFF und (SP) - 2 = 0000 - 2 = FFFE gebracht.

Als Programmierer müssen wir darauf achten, daß in einem Programm auf jeden JSR-Befehl der zugehörige Return-Befehl folgt. Sonst können ungewohnte Dinge passieren. Zwei Beispiele dazu: Wir ersetzen in dem Programm von Abschnitt 12.5 den Return-Befehl bei Adresse 1086 durch einen JMP-Befehl mit dem Operationscode 24 14 10 und starten das Programm (evtl. vorher die **RS**-Taste drücken!). Zunächst sieht noch alles ganz normal aus, aber bald wird etwas Unvorhergesehenes passieren. Der Stack wird bei jedem Sprung ins Unterprogramm AUS um zwei weitere Speicherplätze vollgeschrieben, und zwar mit lauter Rücksprungadressen 1014. Das geht solange gut, wie dieser Speicherplatz nicht anders gebraucht wird. Schließlich kollidiert der Stack mit den Speicherplätzen, die wir für die Zählschleife benutzen: FFE0 bis FFE2. Jetzt wird ein Byte im Stack per Programm dekrementiert, und es kommt zu einem Sprung in die Betriebssoftware. Unser Programm ist beendet, da wir die Spielregeln nicht beachtet haben. Das zweite Beispiel ist noch rigoroser, es zeigt noch katastrophalere Folgen. Wir ersetzen den ersten Befehl im Unterprogramm PAUSE durch einen Sprung zum Unterprogramm AN (20 5F10). Der Stack wird jetzt im Wechsel mit den Rücksprungadressen 1065 und 1102 vollgeschrieben. Das geht solange weiter, bis der Stack in den Speicherbereich reicht, in dem unser Unterprogramm PAUSE stand. Wenn dieses Programm überschrieben ist, sind damit für den Mikroprozessor die Befehle verändert worden. Er erhält jetzt Anweisungen, die wir gar nicht gewollt haben.

12.7 Aufgaben zu Kapitel 12

- Das Programm in Abschnitt 12.3 soll verändert werden, und zwar so,
 - daß die gelbe Leuchtdiode blinkt,
 - daß die rote und die grüne Leuchtdiode abwechselnd leuchten,
 - daß die drei Leuchtdioden abwechselnd leuchten (grün, gelb, rot, grün, gelb, rot...).
- Fragen zum Programm von Abschnitt 12.5:
 - Wie oft wird während eines vollständigen Durchlaufs des Hauptprogramms ein JSR-Befehl ausgeführt?
 - Vor Programmbeginn gelte (SP) = 0000. Auf welche Adresse zeigt der Stackpointer, wenn das Unterprogramm PAUSE bearbeitet wird?
 - Welche verschiedenen Werte stehen im Verlauf des Programms in den vier Speichern, die maximal für den Stack benötigt werden?

13. Die sechzehn Calls

13.1 Die Adressierung der Calls

MNEM. CODE	OP. CODE	ZIELADR. STEHT IN DEN SPEICHERN	INHALT DIESER SPEICHER	CALL BEGINNT IM ROM BEI DER ADRESSE
CALL ANZ EIN <CALL 0>	10	0020 UND 0021	6C 01	016C + 1 = 016D
CALL ANZEIGE <CALL 1>	11	0022 UND 0023	66 08	0866 + 1 = 0867
CALL BLANK <CALL 2>	12	0024 UND 0025	94 08	0894 + 1 = 0895
CALL UEB 4R <CALL 3>	13	0026 UND 0027	82 09	0982 + 1 = 0983
CALL UEB 4L <CALL 4>	14	0028 UND 0029	2B 01	012B + 1 = 012C
CALL UEB 2 <CALL 5>	15	002A UND 002B	4B 01	014B + 1 = 014C
CALL EIN 2ST <CALL 6>	16	002C UND 002D	A8 09	09A8 + 1 = 09A9
CALL EIN 4ST <CALL 7>	17	002E UND 002F	BC 09	09BC + 1 = 09BD
CALL ANZ A <CALL 8>	18	0030 UND 0031	3E 09	093E + 1 = 093F
CALL ANZ EA <CALL 9>	19	0032 UND 0033	52 09	0952 + 1 = 0953
CALL FEHLER <CALL 10>	1A	0034 UND 0035	EE 01	01EE + 1 = 01EF
CALL DEZ-HEX <CALL 11>	1B	0036 UND 0037	A0 08	08A0 + 1 = 08A1
CALL HEX-DEZ <CALL 12>	1C	0038 UND 0039	DE 08	08DE + 1 = 08DF
CALL VERZ <CALL 13>	1D	003A UND 003B	5A 08	085A + 1 = 085B
CALL TEST <CALL 14>	1E	003C UND 003D	EF 02	02EF + 1 = 02F0
CALL HALT <CALL 15>	1F	003E UND 003F	E6 02	02E6 + 1 = 02E7

Die sechzehn Calls, von denen wir acht bisher besprochen haben, sind nichts anderes als Unterprogramme. Wir können sie an beliebiger Stelle unseres Programms aufrufen. Nach Bearbeitung eines Calls wird durch einen RET-Befehl erreicht, daß unser Programm bei dem Befehl fortgesetzt wird, der nach dem Call folgt. Die Calls sind Unterprogramme, allerdings mit zwei Unterschieden: einmal sind diese Unterprogramme als Teile der Systemsoftware fest und unwiderruflich in unserem Computer gespeichert, zum anderen können diese sechzehn Calls in besonders einfacher Weise aufgerufen werden.

Wir haben den Begriff Systemsoftware schon häufiger benutzt (vgl. z. B. Abschnitt 3.6). Was stellen wir uns darunter vor? Wir haben schon gesagt, daß sie es uns überhaupt ermöglicht, mit dem Computer zu arbeiten. Die Systemsoftware (oder die Betriebssoftware oder das Betriebsprogramm) ist nichts weiter als ein Programm, das auch nur aus solchen Befehlen besteht, wie wir sie besprochen haben oder noch besprechen werden. Dieses Betriebsprogramm ist allerdings ein recht langes – aber auch recht komfortables – Programm. Es steht in den Speichern mit den Adressen 0000 bis 0A7F (das sind 2688 Speicherplätze!). Wenn wir den Computer einschalten oder wenn wir die

Taste **RS** betätigen, bekommt der Mikroprozessor an einem seiner vierzig Beinchen einen Impuls, das Programm beginnt bei Adresse 0000. Hier folgt dann bald eine Verzweigung. Es wird entweder „HALLO“ oder es werden eine Adresse und die unter dieser Adresse gespeicherten Daten angezeigt. Das Anzeigen müssen wir uns so vorstellen, daß die einzelnen Stellen der Anzeige sehr schnell nacheinander immer wieder angesteuert werden. Unser Auge sieht ständig die vollständige Anzeige. Es läuft also ein Programm, auch wenn wir nichts davon sehen.

In dieser Anzeigeschleife werden auch ständig die Tasten abgefragt. Der Mikroprozessor liest unter bestimmten Adressen, die den Tasten zugeordnet sind, Daten ein. Aus diesen Daten erkennt er, ob eine Taste betätigt ist. Wenn das der Fall ist, folgt eine Verzweigung. Das Programm wird unterschiedlich fortgesetzt, je nachdem welche Taste betätigt wurde. Das Betriebsprogramm läuft weiter, bis wir mit der **RUN**-Taste unser Programm starten. Wir werden einzelne Teile des Betriebsprogramms noch besprechen. Festhalten wollen wir hier nur, daß dort auch nur Befehle benutzt werden, wie wir sie in unseren Programmen verwenden. Das Betriebsprogramm ist allerdings unlöschar gespeichert, und zwar im ROM. ROM ist die Abkürzung der englischen Bezeichnung **Read Only Memory** (Nur-Lese-Speicher, Festspeicher, Festwertspeicher). Die Daten im ROM (Adresse 0000 bis 0A7F) sind nicht überschreibbar. Sie bleiben auch erhalten, wenn die Netzspannung ausgeschaltet wird. In dem ROM stehen u. a. auch die zehn Spielprogramme (vgl. Kap. 1), die sechzehn Unterprogramme, die wir als Calls aufrufen, und einige Daten, die der Mikroprozessor in seinem Betriebsprogramm benötigt.

Jetzt zur Adressierung der Calls: Die sechzehn Calls sind sechzehn Unterprogramme, die im Betriebsprogramm gespeichert sind. Was beim Aufruf eines Calls passieren muß, ist klar. Der momentane Inhalt des Programmzählers wird im Stack gespeichert, eine der sechzehn Adressen, bei denen die Calls beginnen, kommt in den Programmzähler. Dann wird der Call bearbeitet, anschließend folgt mit einem RET-Befehl der Rücksprung in unser Programm. Wir brauchen uns diese sechzehn Adressen nicht zu merken, sie stehen auch im ROM. Der Mikroprozessor holt sich hier selber die richtige Zieladresse für den Sprung bei einem Call-Aufruf.

Der Mikroprozessor-Hersteller hat festgelegt, welche Operation bei einem Call ausgeführt wird:

BESCHREIBUNG	IMNEM.CODE	OP.CODE	OPERATION
CALL (AUFRUF EINES BESTIMMTEN PROGRAMMTEILS IM ROM)	CALL NAME	1X	$\langle \langle SP \rangle - 1 \rangle := \langle PC \rangle, HOB$ $\langle \langle SP \rangle - 2 \rangle := \langle PC \rangle, LOB$ $\langle SP \rangle := \langle SP \rangle - 2$ $ADR := \langle 0021 + 2X, 0020 + 2X \rangle$ $\langle PC \rangle := ADR$

Die ersten drei Zeilen bei der Beschreibung der Operation stimmen mit denen beim JSR-Befehl überein (vgl. 12.6). Der Inhalt vom Programmzähler wird auf dem Stack abgelegt. Der Inhalt des Stackpointers wird um zwei erniedrigt. X bezeichnet eine einstellige Hexadezimalzahl. Die Calls, die deswegen auch mit CALL 0, CALL 1, ..., CALL 15 bezeichnet werden, haben die Operationscodes 10, 11, ..., 1F. Aufgrund der 1 weiß der Mikroprozessor, daß es sich um einen Call handelt, an der zweiten Hexadezimalziffer X erkennt der Mikroprozessor, welcher Call gemeint ist und wo

er die Zieladresse für den Sprung findet. Die Zieladresse ADR steht im ROM, für Call 0 in den Speichern 0020 (ADR, LOB) und 0021 (ADR, HOB), für Call 1 in den Speichern 0020 + 2 · 1 = 0022 und 0021 + 2 · 1 = 0023, ..., für Call 15 in den Speichern 0020 + 2 · F = 0020 + 1E = 003E und 0021 + 1E = 003F. Diese Zieladresse ADR wird in den Programmzähler geladen. Bei der Adresse ADR + 1 beginnt dann die Bearbeitung der Calls.

Soweit sind die Zusammenhänge vom Mikroprozessor-Hersteller festgelegt worden. Bei der Konzeption des Mikroprozessor-Systems oder des Computers wurde zusätzlich entschieden, welche sechzehn Unterprogramme im Betriebsprogramm als Calls aufgerufen werden können. Damit lag dann auch fest, welche Adressen in den 32 Speicherplätzen von 0020 bis 003F stehen müssen. Wir haben jetzt lediglich noch die Freiheit, diesen sechzehn Calls einigermaßen sinnvolle Namen zu geben. Diese Namen sollen für den mnemonischen Code verwendet werden. Sie müssen daher kurz, trotzdem aber möglichst aussagekräftig sein. In der Übersicht am Anfang dieses Kapitels sind alle sechzehn Namen und die Anfangsadressen der sechzehn zugehörigen Unterprogramme im Betriebsprogramm angegeben. Die bisher noch nicht behandelten Calls sollen im weiteren Verlauf dieses Kapitels besprochen werden.

13.2 Beispiel: CALL VERZ (Verzögerung)

Der CALL VERZ (oder CALL 13) hat den Operationscode 1D. VERZ soll andeuten, daß dieser Call eine Verzögerung bewirkt. Die Zieladresse für den Sprung zu diesem Call steht in den Speichern mit den Adressen 0020 + 2 · D und 0021 + 2 · D ($2 \cdot D_{16} = 2 \cdot 13_{10} = 26_{10} = 1A_{16}$), also in den Speichern 003A und 003B. Wenn wir diese beiden Speicher anwählen, können wir die dort gespeicherte Zieladresse feststellen: (003A) = 5A, (003B) = 08. Im Speicher mit der Adresse 003A steht das LOB, im Speicher mit der Adresse 003B das HOB der Zieladresse.

Angenommen, in unserem Programm (ab Adresse 1000) steht im Speicher mit der Adresse 1008 ein CALL VERZ. Der Mikroprozessor habe die ersten Befehle bis hierher bearbeitet. Jetzt liest er den Operationscode 1D ein. Er weiß, es folgt der CALL 13. Der momentane Inhalt des Programmzählers (1008) wird auf den Stack geschrieben, aus den Speichern 003A und 003B wird die Zieladresse 085A geholt, in den Programmzähler geladen und um eins erhöht. Jetzt beginnt die Bearbeitung des Calls ab Adresse 085B. Wir wollen uns genau ansehen, was bei diesem Call passiert. Wir sehen im ROM nach, was dort ab Adresse 085B steht:

ADRESSE	INHALT
085B	BC
085C	01
085D	00
085E	7C
085F	FB
0860	01
0861	6C
0862	03
0863	01
0864	74
0865	F5
0866	5C
0867	CD
0868	DC
0869	26
086A	C0
086B	FF
086C	27
086D	00
086E	E

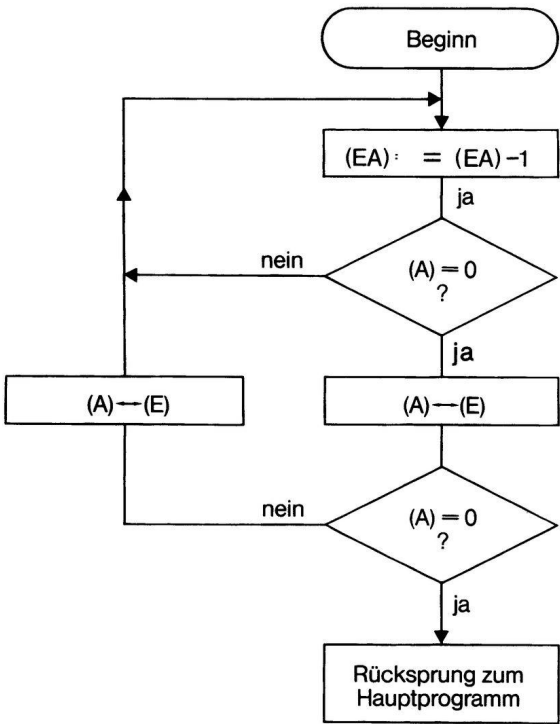
Was bedeuten diese Befehle? Was macht der Mikroprozessor, wenn wir diesen CALL VERZ aufrufen? Wir ordnen die Bytes zu Operationscodes.

ADRESSE	OP. CODE
085B	BC 01 00
085E	7C FB
0860	01
0861	6C 03
0863	01
0864	74 F5
0866	5C

Mit BC beginnt ein Subtraktionsbefehl (3 Byte). Mit 7C, 6C und 74 beginnen Sprungbefehle (2 Byte). 01 ist der Operationscode für den Befehl XCH A,E. 5C ist die Codierung für den RET-Befehl. Daher wissen wir auch, daß das Unterprogramm, der CALL VERZ, an dieser Stelle beendet ist. Unter den oben gemachten Annahmen würde jetzt ein Rücksprung ins Hauptprogramm folgen, es würde die Adresse 1008 vom Stack in den Programmzähler geladen werden. Das Hauptprogramm würde bei Adresse 1009 fortgesetzt werden.
Es ist jetzt keine Schwierigkeit mehr, die Befehle des CALL VERZ in den mnemonischen Code zu übersetzen.

INR.	MARKE	MMEM. CODE	ADR.	OP. CODE	BEMERKUNGEN
150.	IVERZ	ISUB EA,=01	085B	BC 01 00	UNTERPR.: VERZOEGERUNG
151.		IBNZ VERZ	085E	7C FB	(KEIN SPRUNG, WENN <A>=00
152.		IXCH A,E	0860	01	
153.		IBZ RUECK	0861	6C 03	SPRUNG, WENN <E>=00
154.		IXCH A,E	0863	01	
155.		IBRA VERZ	0864	74 F5	
156.	IRUECK	IRET	0866	5C	IRUECKSPRUNG HAUPTPROGRAMM

Bei einer weiteren Analyse dieses Unterprogramms kann der Programmablaufplan hilfreich sein.



Der Inhalt vom EA-Register wird bis 0000 heruntergezählt. In einer inneren Schleife wird solange eins von (EA) subtrahiert, bis der Akkuinhalt null ist. Jetzt wird geprüft, ob (E) auch schon null ist. Dazu muß der Inhalt des Extension-Registers in den Akku geholt werden. Falls (EA) noch nicht null ist, muß vor dem Sprung an den Anfang dieses Unterprogramms der alte EA-Inhalt wiederhergestellt werden, deshalb folgt als drittletzter Befehl wieder: XCH A, E.

Dieser Call hat viel Ähnlichkeit mit dem Unterprogramm PAUSE, das wir in Abschnitt 12.3 benutzt haben. In beiden Fällen wird der Mikroprozessor damit beschäftigt, eine vierstellige Hexadezimalzahl bis 0000 zu erniedrigen, lediglich deshalb, damit sonst in dieser Zeit nichts anderes passiert. Beim Unterprogramm PAUSE wurde am Anfang diese vierstellige Zahl in die Speicher FFE0 und FFE1 geladen, beim CALL VERZ muß diese vierstellige Zahl vor dem Aufruf in das EA-Register geschrieben werden. Hier werden keine Speicherinhalte verändert.

Für den CALL VERZ sollten wir festhalten: Vor Aufruf dieses Unterprogramms muß das EA-Register geeignet geladen werden. Mit (EA) = 0000 erhalten wir die längste Verzögerung von etwa einer Sekunde. Es wird ja erst subtrahiert, dann untersucht, ob der Inhalt des EA-Registers null ist. Mit (EA) = 0001 erhalten wir die kürzeste Verzögerung. Wir wollen uns auch noch merken, daß nach dem Rücksprung ins Hauptprogramm sowohl im Akku als auch im Extension-Register die Hexadezimalzahl 00 steht.

Bei den Programmen im letzten Kapitel, bei denen das Unterprogramm PAUSE aufgerufen wurde, hätten wir genauso gut den CALL VERZ verwenden können. Dieses Unterprogramm zur Verzögerung ab Adresse 085B im ROM kann selbstverständlich auch mit einem JSR-Befehl (Op.Code 20 5A 08) aufgerufen werden, nur der Befehl CALL VERZ (Op. Code 1D) ist eben kürzer und bequemer.

BESCHREIBUNG	MMEM. CODE	OP. CODE	OPERATION
CALL VERZOEGERUNG - PAUSE	CALL VERZ	1D	<EA> := <EA> - 1, BIS <EA> = 0000

13.3 Ein Ampel-Programm

Wir wollen mit Hilfe des Befehls CALL VERZ ein Ampel-Programm schreiben. Das Anschalten der farbigen Leuchtdioden haben wir im letzten Kapitel besprochen. Zur Wiederholung: Die Leuchtdioden zeigen den Zustand der drei Flags F1, F2 und F3 an. Beim Setzen von F1 (LD A, = 02; LD S,A) leuchtet die grüne Leuchtdiode auf. Entsprechendes gilt für die gelbe (F2) und die rote Leuchtdiode (F3). Vor dem Befehl CALL VERZ muß das EA-Register geeignet geladen werden (z. B. mit 0000). Da nach Bearbeitung dieses Calls gilt: (E) = 00, genügt es, das Extension-Register am Anfang des Programms mit 00 zu laden. Vor jedem neuen CALL VERZ hat (E) dann wieder diesen Wert. Bei zwei aufeinander folgenden Befehlen CALL VERZ braucht auch der Akku nicht neu geladen zu werden.

Das Programm ist leicht zu verstehen:

INR.	MARKE	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	ILD A,=0	1000 IC4 00		<A>:=00
2.		ILD E,A	1002 I48		<E>:=00
3.	UMLAUF	ILD A,=02	1003 IC4 02		
4.		ILD S,A	1005 I07		<S>:=02; <F1>:=1; GRUEN
5.		ILD A,=0	1006 IC4 00		
6.		ICALL VERZ	1008 I1D		
7.		ICALL VERZ	1009 I1D		
8.		ICALL VERZ	100A I1D		
9.		ILD A,=04	100B IC4 04		
10.		ILD S,A	100D I07		<S>:=04; <F2>:=1; GELB
11.		ILD A,=0	100E IC4 00		
12.		ICALL VERZ	1010 I1D		
13.		ILD A,=08	1011 IC4 08		
14.		ILD S,A	1013 I07		<S>:=08; <F3>:=1; ROT
15.		ILD A,=0	1014 IC4 00		
16.		ICALL VERZ	1016 I1D		
17.		ICALL VERZ	1017 I1D		
18.		ICALL VERZ	1018 I1D		
19.		ILD A,=0C	1019 IC4 0C		
20.		ILD S,A	101B I07		<S>:=0C; <F2>:=1; <F3>:=1; GELB UND ROT
21.		ILD A,=0	101C IC4 00		
22.		ICALL VERZ	101E I1D		
23.		IBRA UMLAUF	101F I74 E2		SPRUNG ZUR MARKE "UMLAUF"

Während des Umlaufs mit den vier unterschiedlichen Ampelphasen (grün, gelb, rot, gelb/rot) werden jeweils die Flags gesetzt, dann folgt eine Pause, die bei den Phasen grün und rot dreimal so lang ist wie bei den beiden anderen Phasen.

Wenn wir dieses Programm eingetastet und gestartet haben, werden wir feststellen, daß sich hier dasselbe abspielt wie bei Spiel 1: Ampel (vgl. Abschnitt 1.3.1.). Das ist natürlich kein Zufall. Das im ROM von Adresse 03AA bis 03C7 gespeicherte Ampelprogramm ist fast identisch mit unserem Programm.

INR.	MARKE	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	AMPEL	ILD E,A	03AA I48		
2.		ILD A,=02	03AB IC4 02		
3.		ILD S,A	03AD I07		<F1>:=1; GRUEN
4.		ICALL VERZ	03AE I1D		
5.		ICALL VERZ	03AF I1D		
6.		ICALL VERZ	03B0 I1D		
7.		ICALL VERZ	03B1 I1D		
8.		ILD A,=04	03B2 IC4 04		
9.		ILD S,A	03B4 I07		<F2>:=1; GELB
10.		ILD A,=0	03B5 IC4 00		
11.		ICALL VERZ	03B7 I1D		
12.		ILD A,=08	03B8 IC4 08		
13.		ILD S,A	03BA I07		<F3>:=1; ROT
14.		ILD A,=0	03BB IC4 00		
15.		ICALL VERZ	03BD I1D		
16.		ICALL VERZ	03BE I1D		
17.		ICALL VERZ	03BF I1D		
18.		ILD A,=0C	03C0 IC4 0C		
19.		ILD S,A	03C2 I07		<F2>:=1; <F3>:=1; GELB UND ROT
20.		ILD A,=0	03C3 IC4 00		
21.		ICALL VERZ	03C5 I1D		
22.		IBRA AMPEL	03C6 I74 E2		

Drei unwesentliche Unterschiede sind zwischen den beiden Programmen feststellbar.

Wenn das Ampelprogramm im ROM mit den Tasten **SP**, **1** gestartet wird, ist der Akkuinhalt bei Programmstart null. Es kann also gleich mit dem Befehl LD E,A begonnen werden.

Der CALL VERZ wird während der Grün-Phase viermal eingesetzt. Vor dem ersten Aufruf gilt (EA)=0002, d. h. die erste Verzögerung ist sehr kurz (kürzer als eine tausendstel Sekunde) und fällt daher nicht ins Gewicht. Für diesen zusätzlichen Call-Befehl konnte hier der Befehl LD A,=0 gespart werden. Das war erforderlich, um an dieser Stelle einen Speicherplatz zu gewinnen.

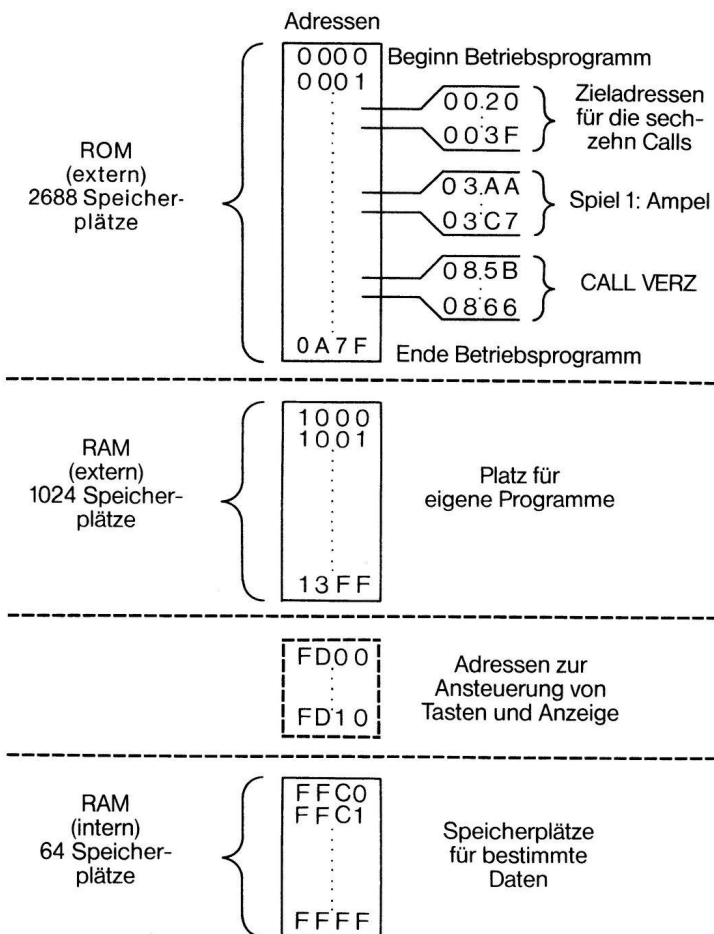
Nach dem Sprung an den Programmanfang wird das Extension-Register wieder mit dem Akkuinhalt geladen. Das wäre nicht erforderlich, da (E) nach dem letzten CALL VERZ ohnehin null ist. Es hat aber keinen Einfluß auf den Programmablauf.

13.4 Die verwendeten Speicher-Adressen

Wir haben neben den Adressen des ROM-Bereichs (0000 bis 0A7F) schon mehrfach andere Adressen benutzt. Unsere Programme haben wir ab Adresse 1000 eingetastet. Bei der direkten Adressierung haben wir Speicher mit den Adressen FFE0, FFE1 usw. verwendet. Die Inhalte in diesen Speichern lassen sich im Gegensatz zu denen im ROM überschreiben. Sie gehen beim Ausschalten der Versorgungsspannung verloren. Beim Einschalten sind zufällige Werte in diesen Speichern.

Diese Speicherplätze bezeichnet man als RAM-Speicherplätze. RAM ist die Abkürzung der englischen Bezeichnung **R**andom **A**ccess **M**emory (deutsche Übersetzung: Speicher mit wahlfreiem Zugriff). Es wird häufiger und treffender die Bezeichnung Schreib-/Lesespeicher mit wahlfreiem Zugriff benutzt. Man kann an einer beliebigen Stelle dieses Speichers etwas einschreiben, abspeichern. Man kann den Inhalt von jeder beliebigen Stelle lesen, also z. B. in den Akku laden oder anzeigen.

Wir wollen uns einen ersten Überblick über alle Speicherplätze verschaffen, die in unserem Mikroprozessorsystem vorhanden sind. Daneben gibt es natürlich noch die Register in der CPU, in denen auch Daten stehen können.

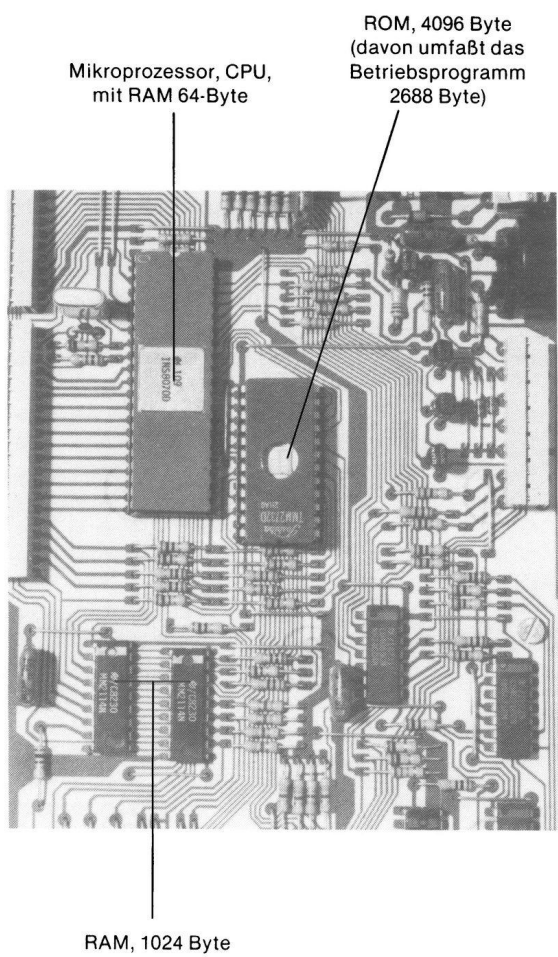


Einige Bemerkungen dazu:

Mit vierstelligen Hexadezimalzahlen könnten 65536 Speicherplätze adressiert werden. Aufgeführt sind in unserer Übersicht nur 3793 Adressen. Wir sehen, es gibt noch genug Möglichkeiten für spätere Ergänzungen. Zwischen den vier angedeuteten Blöcken in der Übersicht ist noch sehr viel Platz. Die nicht aufgeführten Adressen werden in der ersten Ausbaustufe unseres Computers nicht benötigt.

Beim ROM haben wir die drei Teile zusätzlich angegeben, die wir in diesem Kapitel erwähnt haben. Damit haben wir immerhin schon 74 von 2688 Speicherplätzen im ROM untersucht.

Die Angabe „intern“ beim 64-Byte-RAM soll besagen, daß sich dieser Speicherbereich zusammen mit der Zentraleinheit, der CPU, im gleichen Baustein, im Mikroprozessor-IC, befindet. Der 1024 Byte umfassende RAM-Speicher ist in zwei kleinen ICs gesondert untergebracht. Hier tasten wir unsere Programme ein. Der ROM-Bereich befindet sich in einem weiteren IC.



Der Speicherumfang wird häufig in kbyte (sprich: kilobyte) angegeben. k bedeutet normalerweise den Faktor 1000, vgl. 1 km = 1000 m. In der Computertechnik, bei der ja das Dualsystem zugrundegelegt wird, steht das k für $1024 = 2^{10}$. Der RAM-Speicher (von Adresse 1000 bis 13FF) hat einen Umfang von 1 kbyte, das ROM hat eine Größe von 4 kbyte.

Der gestrichelt angegebene Bereich enthält die Adressen von Tastatur und Anzeige. Es handelt sich hier nicht um Speicherplätze wie im ROM oder im RAM. Die elektronische Schaltung auf der Computer-Platine sorgt dafür, daß der Mikroprozessor beim Laden aus diesem Adressenbereich in den Akku eine Information über den Zustand der Tasten enthält, sie sorgt dafür, daß beim Datentransport vom Akku zu diesem Adressenbereich bestimmte Segmente der acht Sieben-Segment-Anzeigen, bzw. bestimmte Leuchtdioden der LED-Reihe aufleuchten. Wir werden uns später mit diesen Zusammenhängen genauer beschäftigen. In diesem Kapitel werden wir Calls kennenlernen, die diesen Datentransport einerseits von der Tastatur zum Akku, andererseits vom Akku zur Anzeige bewirken.

Den 64-Byte-RAM mit den Adressen von FFC0 bis FFFF wollen wir noch einmal ausführlicher darstellen.

F F C 0	— CST0	Codierungen für Stelle 0 bis Stelle 7 bei der Anzeige
F F C 1	— CST1	
F F C 2	— CST2	
F F C 3	— CST3, LED-R	
F F C 4	— CST4	
F F C 5	— CST5	
F F C 6	— CST6	
F F C 7	— CST7	
F F C 8	} (PC)	Speicher, in die die Register-Inhalte aus der CPU gerettet werden
F F C 9		
F F C A	} (SP)	
F F C B		
F F C C	} (P2)	
F F C D		
F F C E	} (P3)	
F F C F		
F F D 0	} (T)	
F F D 1		
F F D 2	— (S)	
F F D 3	— (A)	
F F D 4	— (E)	
F F D 5	} weitere Hilfsspeicher für das Betriebsprogramm	
F F D 6		
F F D 7		
F F D 8	} (AdF), ZWSP	Inhalt des Adressenfeldes bei der Anzeige, Zwischenspeicher bei der Eingabe des Datenfeldes bei der Anzeige
F F D 9		
F F D A		
F F D B	} weitere Hilfsspeicher für das Betriebsprogramm	Speicherplätze, die uns bei unseren Programmen für Daten bei Befehlen mit direkter Adressierung zur Verfügung stehen
F F D C		
F F D D		
F F D E		
F F D F		
F F E 0	} Stack	
F F E 1		
⋮		
⋮		
⋮		
⋮		
⋮		
⋮		
F F E F		
F F F 0		
⋮		
⋮		
⋮		
F F F E		
F F F F		

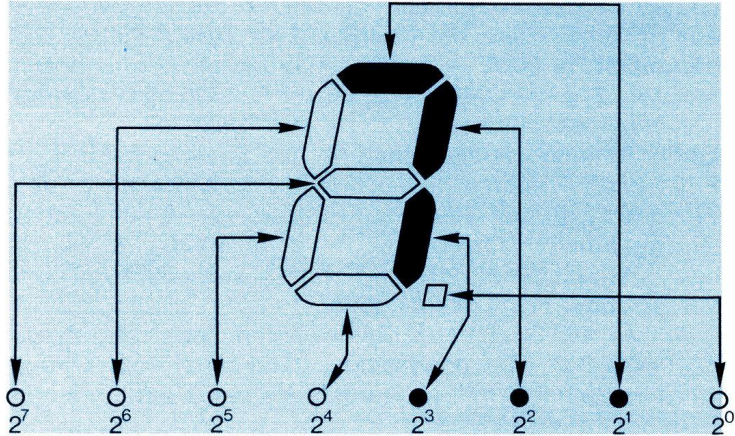
Auch hierzu einige Bemerkungen:

Über die Speicherplätze von FFE0 bis FFFF haben wir uns schon ausführlich Gedanken gemacht. Der Bereich, der uns für das Abspeichern von Daten zur Verfügung steht und der Adressenbereich, der für den Stack (vgl. Abschnitt 12.4) verwendet wird, sind durch eine gestrichelte Linie getrennt. Damit soll angedeutet werden, daß diese Grenze nicht eindeutig festliegt. Es kommt darauf an, wie hoch der Stapelspeicher geladen wird. Wir müssen selber dafür sorgen, daß es nicht zur Überschneidung dieser beiden Bereiche kommt (vgl. 12.6). Wir werden später sehen, daß wir den Stack im Bedarfsfall in einen anderen Adressenbereich legen können. Außerdem stehen uns in der zweiten Ausbaustufe unseres Computers weitere Speicherplätze (FF00 bis FFBF) für direkte Adressierung zur Verfügung.

In den Speichern mit den Adressen FFC8 bis FFD4 werden die Registerinhalte aus der CPU abgespeichert. Wozu ist das erforderlich? Die Register befinden sich in der CPU. In diesen Registern stehen die entsprechenden Inhalte. Die Erklärung ist nicht sehr schwer. Wenn wir z. B. unser Programm mit einem CALL TEST anhalten, dann werden eine Adresse und der unter dieser Adresse gespeicherte Operationscode 1E angezeigt. Für diese Anzeige läuft aber – für uns nicht wahrnehmbar – ein bestimmtes Unterprogramm der Betriebssoftware ab. Für dieses Unterprogramm werden selbstverständlich wieder die Register der CPU benötigt. Daher müssen die Registerinhalte vor Beginn des Calls abgespeichert werden, denn diese Inhalte wollen wir uns ja ansehen. Wenn wir nach der Kontrolle den CALL TEST wieder überschreiben und unser Programm von dieser Stelle ab weiterlaufen lassen wollen, werden zunächst die Register aus den Speichern FFC8 bis FFD4 geladen. Mit diesen Registerinhalten wird dann das Programm fortgesetzt. Die beiden Speicher mit den Adressen FFD8 und FFD9 haben wir bei den Calls CALL EIN 2ST (vgl. Abschnitt 7.6) und CALL EIN 4ST (vgl. Abschnitt 8.3) als Zwischenspeicher benutzt. Nach Beendigung der Eingabe steht die zweistellige Zahl im Speicher mit der Adresse FFD8, die vierstellige Zahl in den beiden angegebenen Speichern. Diese beiden Speicher werden außerdem zusammen mit den Speichern FFDA und FFC0 bis FFC7 ständig für die Anzeige benötigt. Damit werden wir uns in den folgenden Abschnitten ausführlich beschäftigen.

13.5 Die Codierung der sechzehn Hexadezimalziffern

Bei Spiel 4 (vgl. Abschnitt 1.3.4) haben wir uns schon mit diesem Thema beschäftigt. Bei dem dort untersuchten Dezimalzähler bis 15 bzw. bis 255 wurden gleichzeitig die Leuchtdioden der LED-Reihe (Schalter S nach vorn) oder die sieben Segmente und der Punkt in der vierten Sieben-Segment-Anzeige von rechts (Schalter S nach hinten) angesteuert. Wenn ein bestimmtes Symbol angezeigt werden soll, dann kann dieses Symbol nicht unmittelbar zur Anzeige gebracht werden. Es muß erst codiert werden, es muß entsprechend bestimmter Vereinbarungen in eine andere Form übersetzt werden. Diese Vereinbarungen ergeben sich zwangsläufig durch folgende Zuordnung:



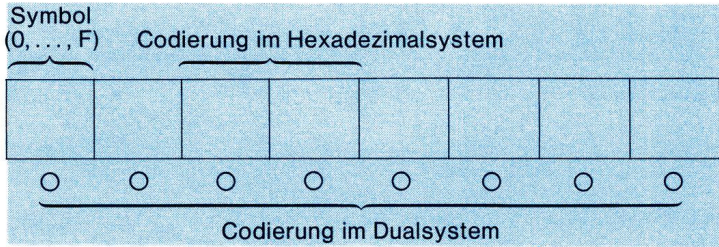
Soll z. B. das Symbol 7 (die Ziffer 7) dargestellt werden, so müssen die drei Segmente leuchten, die den drei Dualzahlen 8, 4 und 2 (2^3 , 2^2 und 2^1) zugeordnet sind. Die Codierung für die Ziffer 7 ist also die Dualzahl 0000 1110, die zweistellige Hexadezimalzahl 0E oder – wie wir bei Spiel 4 erkennen – die Dezimalzahl 14. Entsprechend erhalten wir die Codierungen für alle 256 möglichen Symbole, die darstellbar sind. Die Codierung für das Symbol 0 (Ziffer 0) ist aufgrund der oben angegebenen Zuordnung zwischen Segmenten und Leuchtdioden die Dualzahl 0111 1110 oder die Hexadezimalzahl 7E. Soll nichts angezeigt werden (Anzeige blank, frei, gelöscht), so ist die Codierung hierfür 00.

Das folgende Programm liefert uns die Codierung für die sechzehn Hexadezimalziffern im Dual- und im Hexadezimalsystem.

NR.	MARKE	MMEM. CODE	ADR.	OP. CODE	BEMERKUNGEN
1.	BEGINN	LD A,=-1	1000	1C4 FF	ZAHL:=-1
2.		ST A,ZAHL	1002	1CD E0	
3.		CALL BLANK	1004	112	
4.	WEITER	LD A,ZAHL	1005	195 E0	ZAHL:=ZAHL+1
5.		ST A,ZWSP	1007	1CD D8	
6.		CALL UEB 2	1009	115	CODIEREN DER ZAHL
7.		LD A,CST4	100A	1C5 C4	(A):=(CST4)
8.		ST A,LED-R	100C	1CD C3	(LED-R):=(A)
9.		ST A,CST7	100E	1CD C7	(CST7):=(A)
10.		ST A,ZWSP	1010	1CD D8	(ZWSP):=(A)
11.		CALL UEB 2	1012	115	ERNEUTES CODIEREN
12.		LD A,=0	1013	1C4 00	
13.		CALL ANZEIGE	1015	111	ANZEIGE
14.		CALL ANZEIGE	1016	111	
15.		BRA WEITER	1017	174 EC	NACHSTE ZIFFER

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZAHL	1FFE0	
ZWSP	1FFD8	ZWISCHENSPEICHER
LED-R	1FFC3	LED-REIHE, CST3
CST4	1FFC4	CODIERUNG STELLE 4
CST7	1FFC7	CODIERUNG STELLE 7

Der Schalter S muß in der vorderen Stellung stehen. Wenn wir das Programm starten, erhalten wir nacheinander die Codierungen für die sechzehn Hexadezimalziffern.



Zum Vergleich mit Spiel 4 (vgl. Abschnitt 1.3.4) und im Hinblick auf das Programm im Abschnitt 13.8, ist in der folgenden Tabelle zusätzlich die Codierung im Dezimalsystem angegeben.

dargestelltes Symbol	angezeigte Hexadezimalziffer	Codierung		
		dual	hexadezimal	dezimal
0	0	0111 1110	7E	126
1	1	0000 1100	0C	12
2	2	1011 0110	B6	182
3	3	1001 1110	9E	158
4	4	1100 1100	CC	204
5	5	1101 1010	DA	218
6	6	1111 1010	FA	250
7	7	0000 1110	OE	14
8	8	1111 1110	FE	254
9	9	1101 1110	DE	222
A	A	1110 1110	EE	238
b	B	1111 1000	F8	248
C	C	0111 0010	72	114
d	D	1011 1100	BC	188
E	E	1111 0010	F2	242
F	F	1110 0010	E2	226

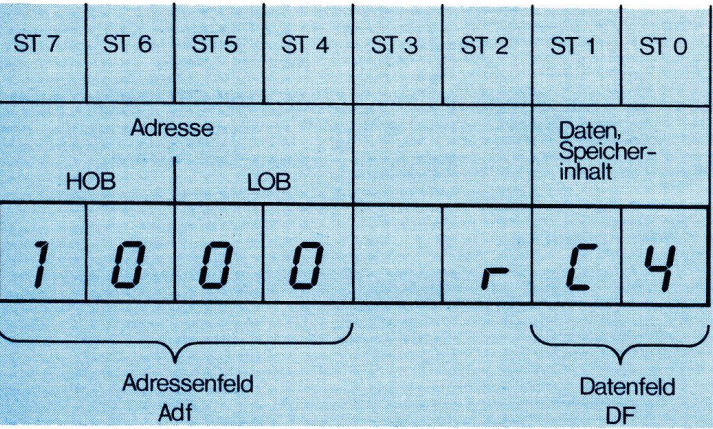
Im übernächsten Abschnitt wollen wir das hier benutzte Programm untersuchen. Im Abschnitt 13.8 sollen weitere Codierungen angegeben werden.

13.6 Zum Anzeige-Vorgang

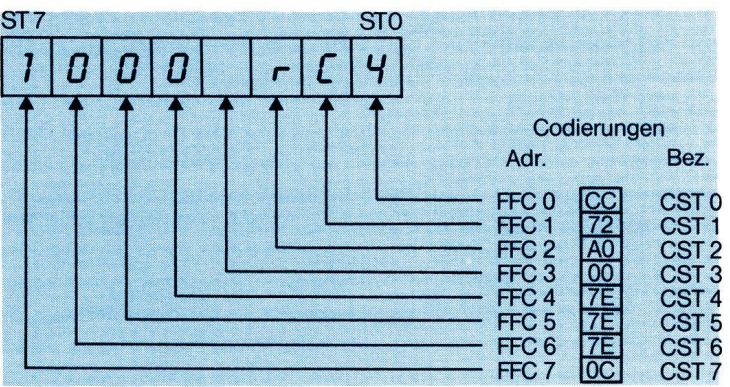
Wir wollen annehmen, daß im Speicher mit der Adresse 1000 der Inhalt C4 steht. Wenn wir mit der Taste **A↔D** die Speicher anwählen, erhalten wir die Anzeige

1 0 0 0 □ C 4 .

Wir wollen einige Bezeichnungen festlegen.

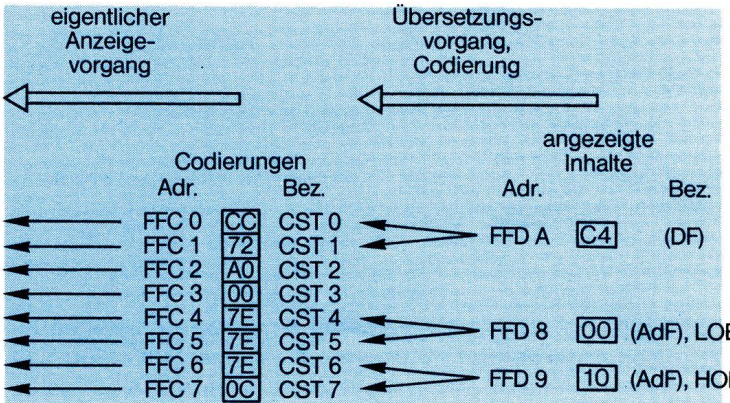


Wir bezeichnen die acht Stellen der Anzeige von rechts nach links mit ST0 bis ST7. Nebenbemerkung: Die acht Leuchtdioden der LED-Reihe sind auch von 0 bis 7 durchnummeriert, sie entsprechen aber nicht den acht Stellen der Anzeige, sondern den sieben Segmenten und dem Punkt von Stelle 3 der Anzeige (vgl. Zuordnung Abschnitt 13.5). Die Adresse 1000 wird im Adressenfeld AdF (ST4 bis ST7), die Daten oder der Speicherinhalt von dem gewählten Speicher werden im Datenfeld DF (ST0 und ST1) dargestellt. Der eigentliche Anzeigevorgang besteht darin, daß die acht Sieben-Segment-Anzeigen ständig der Reihe nach angesteuert werden. Das geschieht so schnell, daß wir ein stehendes Bild sehen. Wie wir aus dem letzten Abschnitt wissen, müssen dazu bestimmte Codierungen für die darzustellenden Symbole und Ziffern zu den acht Stellen der Anzeige gebracht werden. Diese Codierungen für die acht Stellen (CST0 bis CST7) stehen in den acht Speichern FFC0 bis FFC7.



Die Codierung für die Hexadezimalziffern haben wir im letzten Abschnitt angegeben. Daß das Symbol □ mit A0 codiert werden muß, ergibt sich auch aus der Zuordnung im Abschnitt 13.5. Entsprechend wird das Symbol r mit 88 codiert.

Die momentan angezeigte Adresse und die zugehörigen Daten stehen auch im „Klartext“ in bestimmten Speichern. In den beiden Speichern FFD8 und FFD9 steht die angezeigte Adresse, die Daten stehen im Speicher FFDA. Wir müssen uns den gesamten Anzeigevorgang folgendermaßen vorstellen. Zunächst werden die drei Speicher FFD8, FFD9 und FFDA geladen. Jetzt folgt die Codierung der drei Speicherinhalte. Damit werden die Inhalte in den Speichern FFC0 bis FFC7 bestimmt. Dann werden diese Codierungen im eigentlichen Anzeigevorgang zu den acht Stellen der Anzeige gebracht.



Woher weiß unser Computer, wie die sechzehn Hexadezimalziffern zu codieren sind? Im ROM stehen von Adresse 00DE bis 00ED die sechzehn Codierungen. Soll die Ziffer 0 codiert werden, so holt sich der Mikroprozessor den Inhalt aus Speicher 00DE, die Codierung für Ziffer 1 holt er aus dem Speicher 00DE + 1, die Codierung für Ziffer 2 holt er aus dem Speicher 00DE + 2, ... die Codierung für Ziffer F holt er aus dem Speicher 00DE + F = 00ED. Leider können wir die Inhalte in den Speichern FFC0 bis FFC7 und FFD8 bis FFDA nicht ohne weiteres überprüfen. Wenn wir z. B. den Speicher FFDA anwählen, dann ändert sich der Inhalt in diesem Speicher dabei dreimal. Es werden nacheinander die Inhalte der Speicher 000F, 00FF, 0FFD in den Speicher FFDA geschrieben. Wenn wir den Speicher FFD8 anwählen, dann steht da D8, das LOB der jetzt angewählten Adresse FFD8. Ebenso steht im Speicher mit der Adresse FFD9 der Inhalt FF, das HOB der Adresse FFD9.

Das gleiche gilt für die Speicher von FFC0 bis FFC7. Durch das Anwählen wird per Betriebsprogramm ständig der Inhalt in diesen Speichern geändert. In Speicher FFC2 finden wir aber eine der Codierungen A0 oder 88, je nachdem ob gerade das Symbol □ oder ▢ dargestellt wird. Im Speicher FFC3 finden wir die Codierung 00, da bei ST3 nichts angezeigt wird.

Eine Überprüfung der oben gemachten Aussagen gelingt aber per Programm, wenn hierbei die Inhalte der betreffenden Speicher nicht verändert werden.

INR.	MARKE	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	LD A,=-1	1000	C4 FF	(A):=FF
2.		LD A,CST0	1002	C5 C0	(A):=<FFC0>
3.		CALL ANZ A	1004	18	
4.		CALL TEST	1005	1E	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
CST0	FFC0	CODIERUNG FUER STELLE 0

In den Speichern 1000 und 1001 steht ein Load-Befehl, nur deshalb, damit das Programm bei der Anzeige

1	0	0	0	□	C	4
---	---	---	---	---	---	---

gestartet werden kann. Im 2. Befehl wird der Inhalt vom Speicher FFC0 (CST0) in den Akku geladen, im 3. Befehl wird der Akkuinhalt angezeigt. Wenn wir dieses kleine Programm starten, erhalten wir die Anzeige CC, die Codierung für die Ziffer 4. Ebenso können wir die anderen Speicher FFC1 bis FFC7 und FFD8 bis FFDA überprüfen. Wir müssen nur im 2. Befehl C0 der Reihe nach durch C1, C2, ..., C7, D8, D9, DA ersetzen. Die oben angegebenen Speicherinhalte können so bestätigt werden.

Soweit zum Anzeigevorgang, wie er vom Betriebsprogramm benutzt wird. Teile hiervon können wir in unserem Programm als Calls aufrufen, z. B. das Übersetzen der angezeigten Daten in die entsprechenden Codierungen und das eigentliche Anzeigen, der Transport der codierten Daten zu den acht Stellen der Anzeige.

13.7 Drei weitere Calls

Im Programm von Abschnitt 13.5 werden drei Calls benutzt, die wir bisher noch nicht besprochen haben. Um zunächst diese Calls zu untersuchen, wollen wir das Programm vereinfachen. Falls das Programm noch eingetastet ist, ersetzen wir den 7. Befehl (Adressen 100A und 100B) durch einen Sprungbefehl (Operationscode 74 07), einen Sprung zum 12. Befehl.

INR.	MARKE	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	LD A,=-1	1000	C4 FF	ZAHL:=-1
2.		ST A,ZAHL	1002	CD E0	
3.		CALL BLANK	1004	12	
4.	WEITER	LD A,ZAHL	1005	95 E0	ZAHL:=ZAHL+1
5.		ST A,ZWSP	1007	CD D8	
6.		CALL UEB 2	1009	15	CODIEREN DER ZAHL
7.		BRA 12.BEF	100A	74 07	
8.			100C	11 11	
9.			100E	11 11	
10.			1010	11 11	
11.			1012	11 11	
12.	12.BEF	LD A,=0	1013	C4 00	
13.		CALL ANZEIGE	1015	11	ANZEIGE
14.		CALL ANZEIGE	1016	11	
15.		BRA WEITER	1017	74 EC	NACHSTE ZIFFER

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZAHL	FFE0	
ZWSP	FFD8	ZWISCHENSPEICHER

Zunächst wird in den Speicher mit der Adresse FFE0 die Zahl -1 (hexadezimal FF) geladen. Im vierten Befehl wird diese Zahl inkrementiert. Die Zahl 0 wird jetzt zum Speicher mit der Adresse FFD8 (ZWSP oder (AdF), LOB) gebracht. Damit der Inhalt dieses Speichers im rechten Teil des Adressenfeldes angezeigt werden kann, müssen die beiden Hexadezimalziffern aus diesem Speicher FFD8 für die Anzeige codiert werden. Diesen Teil des Betriebsprogramms können wir in unserem Programm als Call aufrufen.

BESCHREIBUNG	INMEM.CODE	OP.CODE	OPERATION
CALL UEBERSETZE FUER ZWEI STELLEN DER ANZEIGE	CALL UEB 2	15	<FFC4>:= CODIERUNG ST.4 <FFC5>:= CODIERUNG ST.5

Der CALL UEB 2 übersetzt die beiden Ziffern aus Speicher FFD8 in die für die Anzeige notwendigen Codierungen und speichert die ermittelten Hexadezimalzahlen in die Speicher mit den Adressen FFC4 und FFC5 (CST4 – Codierung für Stelle 4 der Anzeige; CST5 – Codierung für Stelle 5 der Anzeige).

In unserem Programm steht zunächst 00 im Speicher FFD8. Der CALL UEB 2 codiert die beiden Ziffern 0 und schreibt jeweils 7E in die Speicher FFC4 und FFC5. Damit wird noch nichts angezeigt, es wird nur die Anzeige vorbereitet. Bei einer folgenden Anzeige kämen jetzt die beiden Codierungen 7E zu ST4 und ST5 der Anzeige, es würden zwei Nullen dargestellt werden.

Durch unseren geänderten 7. Befehl (BRA 12. Befehl) erreichen wir jetzt das Anzeigen in den Befehlen 12. bis 14. Anschließend folgt der Rücksprung zur Marke WEITER. Die Zahl (bisher 00) wird inkrementiert: Zahl := 01. Jetzt werden die beiden Ziffern 0 und 1 codiert. 0C wird als Codierung für die Ziffer 1 in den Speicher FFC4, 7E wird in den Speicher FFC5 gebracht. Bei der nächsten Anzeige erscheint 01 usw. Wenn wir dieses veränderte Programm laufen lassen, sehen wir, daß wir einen zweistelligen Hexadezimalzähler programmiert haben.

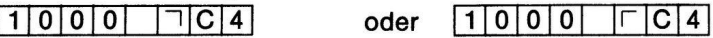
Soweit zum CALL UEB 2. Jetzt zum CALL ANZEIGE.

BESCHREIBUNG	INMEM.CODE	OP.CODE	OPERATION
CALL ANZEIGE DER CODIERUNGEN	CALL ANZEIGE	11	ANZEIGE: <FFC0>...<FFC7>

Dieser Call bringt mehrfach nacheinander die acht Codierungen aus den Speichern FFC0 bis FFC7 zu den acht Sieben-Segment-Anzeigen. Der Inhalt des Akkumulators bei Beginn des Calls bestimmt die Anzeigedauer. Beim Anzeigen wird der Akkuinhalt dekrementiert. Das Anzeigen wird abgebrochen, wenn der Akkuinhalt null geworden ist. Gilt bei Beginn des Calls (A) = 00, so dauert die Anzeige am längsten, etwa 1 Sekunde. (A) wird erst dekrementiert, dann wird untersucht, ob die Bedingung (A) = 00 erfüllt ist. Da der Akkuinhalt nach Bearbeitung dieses Calls wieder null ist, brauchen wir in unserem Programm den Akku vor der zweiten CALL ANZEIGE nicht neu zu laden. Die drei Befehle (12. bis 14.) bewirken jeweils eine Anzeige von etwa zwei Sekunden. Die genaue Dauer werden wir später untersuchen. Noch eine Bemerkung zu dem dritten Call, den wir in diesem Programm zum erstenmal verwendet haben.

BESCHREIBUNG	INMEM.CODE	OP.CODE	OPERATION
CALL LOESCHEN DER ANZEIGE	CALL BLANK	12	<FFC0>:=00 ... <FFC7>:=00

Die Bedeutung dieses Calls erkennen wir am schnellsten, wenn wir ihn durch einen NOP-Befehl ersetzen. Wir überschreiben den Inhalt 12 im Speicher mit der Adresse 1004 durch 00 und starten das Programm neu. Bei Programmstart war die Anzeige



In den Speichern FFC0 bis FFC7 stehen die acht entsprechenden Codierungen. Wenn jetzt im Programm nur die Inhalte in den Speichern FFC4 und FFC5 (rechter Teil des Adressenfeldes) überschrieben werden, so werden die übrigen Symbole beim CALL ANZEIGE weiterhin dargestellt. Der CALL BLANK bewirkt, daß die acht Speicher FFC0 bis FFC7 gelöscht werden, es wird in alle acht Speicher die Hexadezimalzahl 00 geladen.

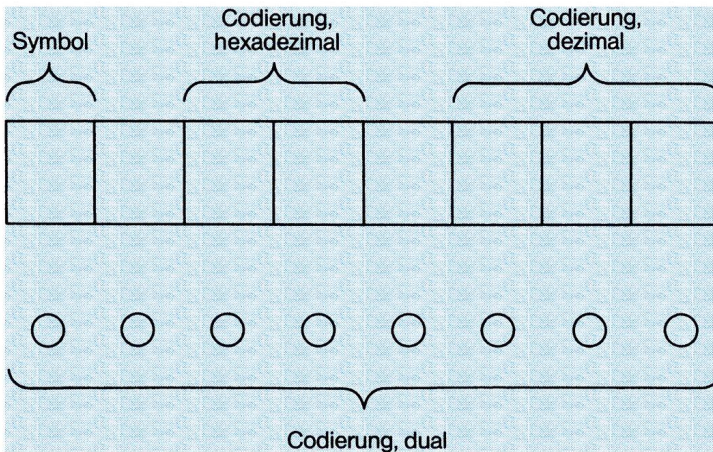
Zurück zum Programm von Abschnitt 13.5. Wir sorgen dafür, daß dieses Programm wieder eingetastet ist. Wir wollen uns noch überlegen, was mit den Befehlen 7. bis 11. erreicht wird. Im 7. Befehl wird die Codierung für die zweite Ziffer der Zahl (die im Speicher FFE0 steht) wieder in den Akku geladen. Sie wird dann in die Speicher FFC3 (Anzeige der Codierung im Dualsystem an der LED-Reihe) und FFC7 (Darstellung der Ziffer in der linken Sieben-Segment-Anzeige) gebracht.

Um z. B. die Codierung 7E für die Darstellung der 0 zusätzlich im Hexadezimalsystem anzeigen zu können, müssen die beiden Ziffern 7 und E noch einmal codiert werden. Das geschieht mit den beiden Befehlen 10 und 11 durch einen erneuten Befehl CALL UEB 2.

13.8 Codierung beliebiger Symbole

Aufgrund der Zuordnung zwischen den Segmenten der Sieben-Segment-Anzeige ST3 und den Leuchtdioden (und den entsprechenden Zweier-Potenzen) im Abschnitt 13.5, liegen die Codierungen für alle möglichen 256 Symbole fest, die an einer Sieben-Segment-Anzeige dargestellt werden können.

Wenn wir die Codierung für ein bestimmtes Symbol ermittelt haben, können wir sie mit Hilfe des folgenden Programms überprüfen. Wenn wir das Programm starten, erwartet der Computer die Eingabe einer Codierung im Hexadezimalsystem. Wird dann die Taste **RUN** betätigt, so erscheinen das codierte Symbol und die Codierung im Dual-, Hexadezimal- und Dezimalsystem. Schalter S muß nach vorn gestellt sein.



INR.	MARKE	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.		ICALL EIN 2ST	1000	16	
2.		ILD EA,ZWSP	1001	85 D8	IEINGABE: COD.HEX.
3.		IST A,CODHEX	1003	1CD E0	
4.		ICALL HEX-DEZ	1005	1C	
5.		IST EA,ZWSP	1006	18D D8	
6.		ICALL UEB 4R	1008	13	CODIERUNG DEZIMAL
7.		ILD A,CODHEX	1009	1C5 E0	
8.		IST A,LED-R	100B	1CD C3	CODIERUNG DUAL
9.		IST A,CST7	100D	1CD C7	DARGESTELLTES SYMBOL
10.		IST A,ZWSP	100F	1CD D8	
11.		ICALL UEB 2	1011	115	CODIERUNG HEXADEZIMAL
12.		ICALL ANZ EIN	1012	110	ANZEIGE
13.		INOP	1013	100	
14.		INOP	1014	100	
15.		IBRA BEGINN	1015	174 E9	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZWSP	1FFD8	ZWISCHENSPEICHER
	1FFD9	
CODHEX	1FFE0	CODIERUNG HEXADEZIMAL
LED-R	1FFC3	LED-REIHE
CST7	1FFC7	CODIERUNG STELLE 7

Beispiele (vgl. auch die Tabelle im Abschnitt 13.5):

dargestelltes Symbol	dual	Codierung hexadezimal	dezimal
blank	0000 0000	00	0
┐	1000 1000	88	136
┌	1010 0000	A0	160
H	1110 1100	EC	236
L	0111 0000	70	112
P	1110 0110	E6	230
U	0111 1100	7C	124
Y	1101 1100	DC	220

Nun zum Programm. Die eingegebene Codierung im Hexadezimalsystem wird zunächst unter der Adresse FFE0 gespeichert, dann ins Dezimalsystem umgewandelt. Jetzt folgt ein weiterer Call:

BESCHREIBUNG	INMEM.CODE	OP.CODE	OPERATION
ICALL UEBERSETZE FUER VIER RECHTE STELLEN DER ANZEIGE	ICALL UEB 4R	13	(FFC0):= CODIERUNG ST.0 ... (FFC3):= CODIERUNG ST.3

Der CALL UEB 4R übersetzt den vierstelligen Inhalt der beiden Speicher FFD9 und FFD8, dann werden die Codierungen für die vier Ziffern in die Speicher FFC3 bis FFC0 gebracht. Dieser Call löscht zusätzlich die vier linken Stellen der Anzeige. Daher ist in diesem Programm der CALL BLANK nicht erforderlich. Außerdem werden bei der maximal vierstelligen Zahl, die rechts angezeigt wird, die führenden Nullen gelöscht. Dieser CALL UEB 4R wird beim CALL EIN 2ST und beim CALL EIN 4ST als Unterprogramm verwendet. Beim Taschenrechnerprogramm (Spiel 10, vgl. Abschnitt 1.3.10) konnte dieser Call mit Erfolg verwendet werden.

Zurück zu unserem Programm. Der folgende Teil ist bekannt. Die Leuchtdioden (ST3) werden für die Anzeige vorbereitet, damit wir dort die Codierung im Dualsystem erhalten. Die Codierung im Hexadezimalsystem kommt in den Speicher FFC7, damit bei ST7 das Symbol erscheint, dessen Codierung wir eingegeben haben. Um zusätzlich die eingegebene Codierung anzeigen zu können, benutzen wir wieder den CALL UEB 2.

Jetzt folgt nach diesen Vorbereitungen die eigentliche Anzeige, wieder mit einem neuen Call.

BESCHREIBUNG	MNEM.CODE	OP.CODE	OPERATION
ICALL ANZEIGE UND EINGABE	ICALL ANZ EIN	10	ANZEIGE: <FFC0>...<FFC7> ABFRAGE DER TASTEN: <A>:=<TASTE> BEI ZIFFERTASTE <PC>:=<PC>+2

Wie beim CALL ANZEIGE werden die Inhalte der Speicher FFC0 bis FFC7 zu den acht Stellen ST0 bis ST7 gebracht. Hier wird allerdings das Anzeigen solange fortgesetzt, bis eine Ziffern- oder eine Funktionstaste betätigt wird. Soweit der Anzeige-Teil dieses Calls. Daß bei diesem Call auch eine Eingabe möglich ist, werden wir im nächsten Abschnitt sehen.

13.9 Die restlichen Calls

13.9.1 Der CALL ANZ EIN

Zunächst noch einmal zu dem Call, den wir im letzten Abschnitt zum erstenmal benutzt haben.

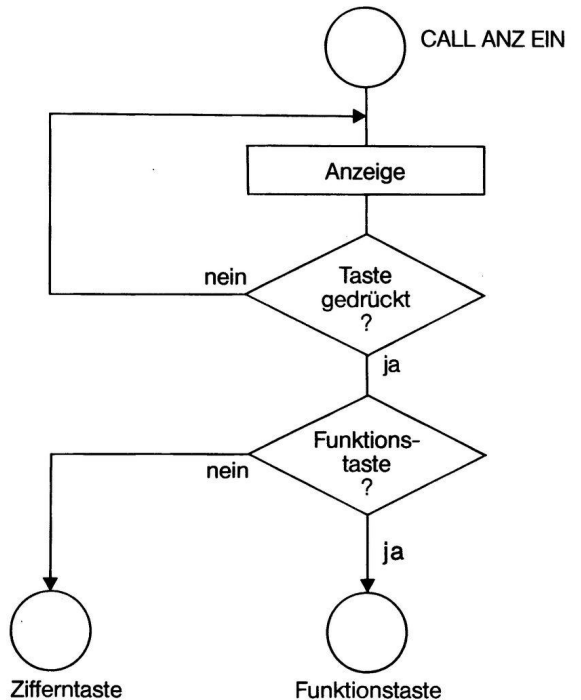
BESCHREIBUNG	MNEM.CODE	OP.CODE	OPERATION
ICALL ANZEIGE UND EINGABE	ICALL ANZ EIN	10	ANZEIGE: <FFC0>...<FFC7> ABFRAGE DER TASTEN: <A>:=<TASTE> BEI ZIFFERTASTE <PC>:=<PC>+2

Den Anzeigen-Teil haben wir kennengelernt. Die Möglichkeiten der Eingabe sollen mit folgendem kleinen Programm erläutert werden. Schalter S nach hinten.

INR.	MARKE	MNEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	ICALL BLANK	1000	12	
2.		ILD EA,=072EE	1001	84 EE 72	
3.		IST EA,CST5	1004	8D C5	ANZEIGE: "CALL 0"
4.		ILD EA,=07070	1006	84 70 70	
5.		IST EA,CST3	1009	8D C3	
6.		ILD A,=07E	100B	1C4 7E	
7.		IST A,CST1	100D	1C0 C1	
8.		ICALL ANZ EIN	100F	10	ABFRAGE DER TASTEN
9.		IBRA FUNKT	1010	174 05	SPRUNG, WENN FUNKTIONSTASTE
10.	ZIFFER	IST A,ZWSP	1012	1C0 D8	ZIFFERTASTE
11.		ICALL BLANK	1014	112	
12.		IBRA UEBERS	1015	174 07	
13.	FUNKT	IST A,ZWSP	1017	1C0 D8	FUNKTIONSTASTE
14.		ICALL BLANK	1019	112	
15.		ILD A,=0E2	101A	1C4 E2	
16.		IST A,CST7	101C	1C0 C7	
17.	UEBERS	ICALL UEB 2	101E	115	UEBERSETZEN
18.		ILD A,=0	101F	1C4 00	
19.		ICALL ANZEIGE	1021	111	... UND ANZEIGEN
20.		IBRA BEGINN	1022	174 DC	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
CST1	1FFC1	CODIERUNG FUER STELLE 1
CST3	1FFC3	CODIERUNG FUER STELLE 3
	1FFC4	... UND FUER STELLE 4
CST5	1FFC5	CODIERUNG FUER STELLE 5
	1FFC6	... UND FUER STELLE 6
CST7	1FFC7	CODIERUNG FUER STELLE 7
ZWSP	1FFD8	ZWISCHENSPEICHER

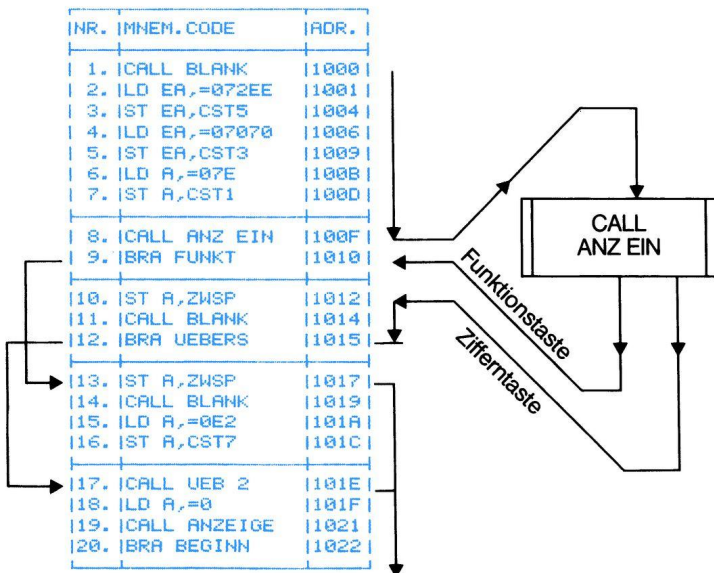
In den ersten sieben Befehlen werden die Speicher FFC0 bis FFC7 so geladen, daß beim achten Befehl – beim CALL ANZ EIN – „CALL 0“ angezeigt wird. Dieser Call kann im wesentlichen durch den folgenden Programmablaufplan beschrieben werden.



Zunächst wird in einer Schleife mit Hilfe der Codierungen in den Speichern FFC0 bis FFC7 die Anzeige angesteuert, und es wird untersucht, ob eine Taste betätigt ist. Ist das nicht der Fall, wird die Schleife weiter durchlaufen. Nach Verlassen der Schleife wird das Programm verzweigt, je nachdem ob eine Zifferntaste oder eine Funktionstaste betätigt wurde. Bei Betätigung der Taste **A↔D** folgt ein Sprung zum Programmstart. Das haben wir in unserem Programm und auch im Ablaufplan nicht berücksichtigt. Bei den sieben anderen Funktionstasten wird das Unterprogramm CALL ANZ EIN mit einem Return-Befehl abgeschlossen. Unser Programm wird mit dem Befehl fortgesetzt, der dem Call in unserem Programm unmittelbar folgt. Wurde eine Zifferntaste betätigt, so wird im Unterprogramm die Rücksprungadresse (auf dem Stack) um zwei erhöht. Daher wird nach Ablauf des Unterprogramms CALL ANZ EIN jetzt ein 2-Byte-Befehl im Hauptprogramm überschlagen.

Der Operationscode 10 für den CALL ANZ EIN steht bei Adresse 100F. Nach dem Call geht es bei Adresse 1010 (Funktionstaste) oder bei Adresse 1012 (Zifferntaste) weiter.

Durch den Sprungbefehl in den beiden Speichern 1010 und 1011 (9. Befehl) erreichen wir die Verzweigung. Im weiteren Verlauf unseres Programms wird für etwa eine Sekunde der Inhalt angezeigt, der beim Rücksprung vom Unterprogramm im Akku vorhanden war. Nach dem Rücksprung von dem CALL ANZEIGE wird bei Betätigung einer Funktionstaste zusätzlich die Codierung für das Symbol F (Funktionstaste) zum Speicher FFC7 gebracht. Der Akkuinhalt zeigt uns



eindeutig, welche Taste betätigt wurde. Beim Rücksprung nach Betätigung einer Zifferntaste steht die entsprechende Ziffer im Akku. Für die Funktionstasten gilt folgende Zuordnung.

Taste	(A)
RUN	01
ME -	02
ME +	03
SV	04
LD	05
CPU	06
SP	07

INR.	MARKE	INMEM. CODE	ADR.	OP. CODE	BEMERKUNGEN
1.	BEGINN	ICALL EIN 4ST	1000	17	EINGABE DER
2.		ILD EA,ZWSP	1001	85 D8	DEZIMALZAHL
3.		ICALL DEZ-HEX	1003	18	
4.		IST EA,ZWSP	1004	8D D8	HEXADEZIMALZAHL
5.		ICALL UEB 4L	1006	14	ZUR ANZEIGE
6.		ICALL ANZ EIN	1007	10	ANZEIGE
7.		INOP	1008	00	
8.		INOP	1009	00	
9.		IBRA BEGINN	100A	174 F4	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZWSP	FFD8	ZWISCHENSPEICHER
	FFD9	

Nach Programmstart geben wir eine Dezimalzahl ein. Diese Eingabe wird mit einer Funktionstaste (außer **A↔D**) abgeschlossen. Die eingegebene Zahl wird in eine Hexadezimalzahl umgewandelt. Zur Anzeige der Hexadezimalzahl werden die Speicher FFD9 und FFD8 geladen, dann wird der CALL UEB 4L aufgerufen. Beim anschließenden CALL ANZ EIN werden die eingegebene Dezimalzahl und die durch Umwandlung gefundene Hexadezimalzahl angezeigt.

Dieses Programm stimmt mit dem ersten und dritten Teil des Programms von Abschnitt 5.6.2 überein. Der dort eingefügte mittlere Teil (Adressen 1007 bis 101B) löscht die führenden Nullen bei der Anzeige der Hexadezimalzahl.

Eine Bemerkung noch zum CALL ANZ EIN. Auf diesen Call folgen in unserem Programm zwei NOP-Befehle. Es soll hier keine Verzweigung vorgenommen werden, wenn nach der Anzeige eine Funktions- oder eine Zifferntaste betätigt wird. In beiden Fällen folgt der Sprung zum Programmbeginn.

13.9.2 Der CALL UEB 4L

BESCHREIBUNG	INMEM. CODE	OP. CODE	OPERATION
ICALL UEBERSATZE FUER VIER LINKE STELLEN DER ANZEIGE	ICALL UEB 4L	14	[FFC4]:= CODIERUNG ST.4 ... [FFC7]:= CODIERUNG ST.7

Dieser Call übersetzt den vierstelligen Inhalt der beiden Speicher FFD9 und FFD8, dann werden die Codierungen für die vier Ziffern in die Speicher FFC7 bis FFC4 gebracht. Im Gegensatz zum CALL UEB 4R werden die vier rechten Stellen der Anzeige und die führenden Nullen bei der Darstellung der vierstelligen Zahl links nicht gelöscht. Als kleines Beispiel-Programm wollen wir die Umwandlung einer maximal vierstelligen Dezimalzahl in eine Hexadezimalzahl wählen.

13.9.3 Der CALL HALT

BESCHREIBUNG	INMEM. CODE	OP. CODE	OPERATION
ICALL HALT - PROGRAMMSTOP	ICALL HALT	1F	

Zunächst wieder ein kleines Programm. Schalter S nach vorn.

INR.	MARKE	INMEM. CODE	ADR.	OP. CODE	BEMERKUNGEN
1.	BEGINN	ILD A,=0	1000	1C 00	ZAHL :=00
2.		IST A,ZAHL	1002	1D E0	
3.	WEITER	ILD A,ZAHL	1004	195 E0	ZAHL :=ZAHL+1
4.		IST A,LED-R	1006	1D C3	DIE ZAHL SOLL DUAL AN DER LED-REIHE ANGEZEIGT WERDEN
5.		ICALL HALT	1008	1F	
6.		IBRA WEITER	1009	174 F9	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZAHL LED-R	FFE0 FFC3	

Wenn wir das Programm gestartet haben, meldet sich der Computer umgehend mit der Anzeige

1 0 0 9 7 7 4

Außerdem leuchtet die rechte Diode der LED-Reihe. Die Zahl im Speicher FFE0, die wir zunächst gleich null gesetzt hatten, wurde einmal inkrementiert und zum Speicher FFC3 gebracht. Es wird also die Zahl (FFE0) an der LED-Reihe dual dargestellt. Außerdem wird die Adresse 1009 und das hier gespeicherte erste Byte des Sprungbefehls angezeigt. Dieser Call hat viel mit dem CALL TEST gemeinsam. In beiden Fällen wird unser Programm angehalten, wir können die Registerinhalte überprüfen. Wir sehen uns z. B. mit CPU, 6 den Akkuinhalt an (anschließend zweimal Taste RUN drücken).

Der Unterschied zwischen beiden Calls besteht darin, daß der Programmzähler beim CALL TEST auf den Speicher zeigt, in dem der Call steht, beim CALL HALT zeigt er auf den folgenden Speicher. Der CALL TEST wird ja vorübergehend in ein zu testendes Programm gesetzt, um an einer bestimmten Stelle die Registerinhalte überprüfen zu können. Nach der Kontrolle dieser Inhalte soll der Call wieder durch den ursprünglichen Befehl überschrieben werden. Mit der Taste RUN wird dann das Programm von hier fortgesetzt. Beim CALL HALT soll z. B. ein Programm in jedem Schleifendurchlauf angehalten werden. Deswegen zeigt der Programmzähler beim Halt auf den nächsten Speicher. Betätigen wir jetzt die Taste RUN, so läuft das Programm weiter, bis es wieder angehalten wird. In unserem kleinen Programm haben wir einen Dualzähler programmiert, der bei jeder Betätigung der RUN-Taste weiterzählt.

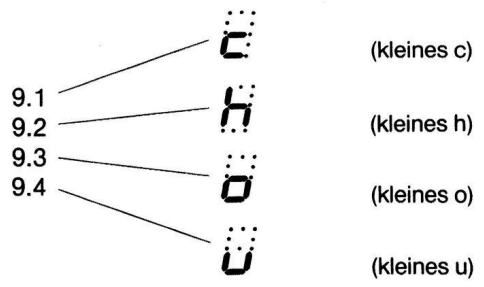
13.10 Aufgaben zu Kapitel 13

- 1. Im Abschnitt 13.2 haben wir den CALL VERZ ausführlich untersucht. Dazu haben wir auch die Befehle im Betriebsprogramm analysiert. In gleicher Weise soll der CALL BLANK untersucht werden.
- 2. Durch welche sechzehn JSR-Befehle könnten wir die sechzehn 1-Byte-Befehle mit den Operationscodes 10, 11, 12, ..., 1F ersetzen, um die sechzehn Unterprogramme aufzurufen, die als Calls in der Betriebssoftware festgelegt sind?
- 3. Das erste Programm in Abschnitt 13.3 soll so geändert werden,
 - 3.1 daß die vier Ampelphasen doppelt so lang werden,
 - 3.2 daß die vier Ampelphasen halb so lang werden.
- 4. Das Spiel 1 (Ampel) kann mit den Tasten SP, 1 aufgerufen werden. Wir können das im ROM gespeicherte Ampelprogramm auch mit einem JMP-Befehl aufrufen. Wie muß der Operationscode für diesen JMP-Befehl lauten?
- 5. Im Abschnitt 4.3.4 ist das Programm für einen Zähler angegeben.
 - 5.1 Das Programm soll in den mnemonischen Code übersetzt werden.
 - 5.2 Wie läßt sich bei diesem Programm die Zählgeschwindigkeit erhöhen?
 - 5.3 Welche Änderungen mußten gegenüber diesem Programm bei dem Programm von Abschnitt 4.3.5 vorgenommen werden?

- 6. Zu dem Programm in Abschnitt 5.6.1 soll ein Programmablaufplan gezeichnet werden.
- 7. Ein Blinkprogramm. Schalter S nach vorne. Gegeben ist nur der mnemonische Code.

IMARKE	IMNEM. CODE
IBEGINN	ICALL BLANK
INEU	ILD A,=0F0
	IST A,LED-R
	ILD A,=0
	ICALL ANZEIGE
	ILD A,=0F
	IST A,LED-R
	ILD A,=0
	ICALL ANZEIGE
	IBRA NEU

- 7.1 Die Befehle sollen in den Operationscode übersetzt werden. Anschließend kann das Programm getestet werden.
- 7.2 Welche Änderungen sind vorzunehmen, wenn alle acht Leuchtdioden gemeinsam im Wechsel ein- und ausgeschaltet werden sollen?
- 7.3 Welche Änderungen sind vorzunehmen, wenn die vier inneren und die vier äußeren Leuchtdioden der LED-Reihe im Wechsel leuchten sollen?
- 8. Welches Symbol wird an der Sieben-Segment-Anzeige dargestellt, wenn die Anzeige mit einer der folgenden Codierungen angesteuert wird?
 - 8.1 80
 - 8.2 01
 - 8.3 90
 - 8.4 05
 - 8.5 A7
 - 8.6 44
- 9. Welche Codierung muß gewählt werden, wenn an einer Sieben-Segment-Anzeige einer der folgenden kleinen Buchstaben dargestellt werden soll?



- 10. Im Abschnitt 13.9.3 hatten wir einen Dualzähler programmiert, der bei jeder Betätigung der RUN-Taste weiterzählte. Wenn in dem Programm der CALL 15 durch den CALL 1 ersetzt wird, ergibt sich ein Dualzähler, der automatisch zählt. Weshalb ist dann die Zählgeschwindigkeit unterschiedlich groß?
- 11. Wir tasten die beiden Befehle ein:

ADRESSE	OP. CODE
1000	11
1001	74 FD

Wenn wir das Programm starten, erhalten wir z. B. die Anzeige

1 0 0 0 7 1 1

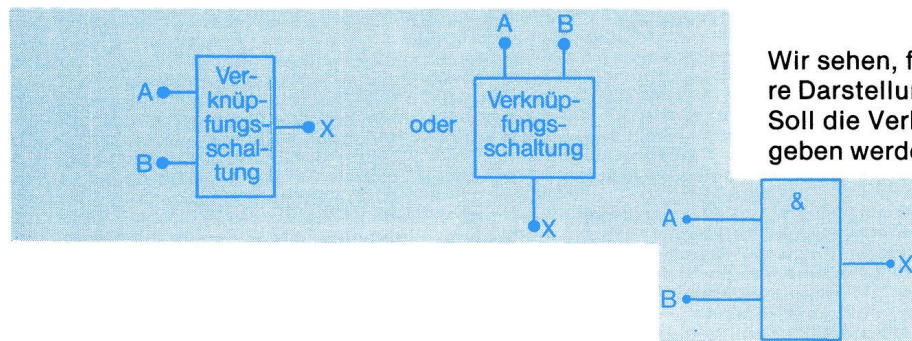
Weshalb reagiert der Computer nun nicht auf die Betätigung der Tasten (außer der Taste RS)?

14. Logische Operationen

14.1 Logische Verknüpfungen von zwei Eingangssignalen

Die Kapitelüberschrift „Logische Operationen“ soll nicht ausdrücken, daß die bisher behandelten Operationen unlogisch sind. Man bezeichnet aber die Operationen, die wir in diesem Kapitel untersuchen wollen als logische Operationen im Gegensatz zu den arithmetischen Operationen wie Addition und Subtraktion. Die Bezeichnung ist dadurch zu erklären, daß die hier untersuchten Verknüpfungen im Prinzip auch im Rahmen der formalen Logik – einem Teilgebiet der Philosophie – behandelt werden.

In diesem Abschnitt wollen wir uns zunächst mit logischen Verknüpfungen von zwei binären Eingangssignalen beschäftigen. Dieses Thema ist Grundlage der Digitalelektronik. Es werden dort Verknüpfungsglieder untersucht, bei denen zwei (oder mehrere) Eingangssignale A, B, ... in bestimmter Weise zu einem Ausgangssignal X (oder Y, ...) verknüpft werden.



Alle Signale können nur zwei verschiedene Zustände annehmen: L oder H (vgl. hierzu Abschnitt 6.1 und Abschnitt 9.5). L-Signal bedeutet niedrige Spannung (engl. low – niedrig, z. B. zwischen 0V und 1V). Ein Taster ist z. B. nicht betätigt, eine Leuchtdiode leuchtet nicht, ein Speicher ist nicht gesetzt usw. H-Signal bedeutet hohe Spannung (engl. high – hoch, z. B. zwischen 3V und 5V). Ein Taster ist betätigt, eine Leuchtdiode leuchtet, ein Speicher ist gesetzt usw.

A	B	X
L	L	
L	H	
H	L	
H	H	

Bei zwei Eingangssignalen gibt es genau vier verschiedene Eingangssignalkombinationen. Ein bestimmtes Verknüpfungsglied ordnet bei einigen Eingangssignalkombinationen dem Ausgangssignal den Wert L, bei den anderen Eingangssignalkombinationen dem Ausgangssignal den Wert H zu. Nach der Art dieser Zuordnung unterscheiden sich die verschiedenen Verknüpfungsschaltungen. Die wichtigsten dieser Schaltungen werden mit bestimmten Namen bezeichnet.

14.1.1 Die UND-Schaltung

Bei einer UND-Schaltung hat das Ausgangssignal X nur dann den Wert H, wenn das Eingangssignal A den Wert H hat u n d wenn das Eingangssignal B den Wert H hat. Dasselbe wird durch die Schaltwerttabelle oder die Verknüpfungstabelle beschrieben. In den vier Zeilen der Tabelle stehen die vier Möglichkeiten, die bei der UND-Verknüpfung der beiden Eingangssignale vorkommen können. A und B können jeweils zwei Werte annehmen (L und H), es gibt vier Eingangssignalkombinationen. Nur in der vierten Zeile hat das Ausgangssignal X den Wert H.

A	B	X
L	L	L
L	H	L
H	L	L
H	H	H

Häufig beschreibt man den Zusammenhang zwischen den drei Signalen auch mit Hilfe einer Gleichung: $X = A \wedge B$, (sprich: X gleich A u n d B). Die Variablen A und B können in dieser Gleichung natürlich wieder jeweils zwei Werte annehmen:

$A = L; B = L; X = L \wedge L = L$

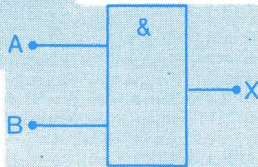
$A = L; B = H; X = L \wedge H = L$

$A = H; B = L; X = H \wedge L = L$

$A = H; B = H; X = H \wedge H = H$

Wir sehen, für den gleichen Zusammenhang gibt es mehrere Darstellungsmöglichkeiten.

Soll die Verknüpfungsschaltung in einem Schaltplan angegeben werden, so benutzt man das folgende Symbolsymbol:

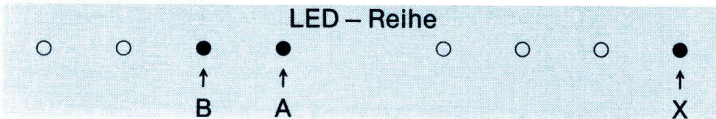


Hier kommt es dann nicht auf die genaue elektronische Schaltung zur Verknüpfung von A und B an, es soll nur angedeutet werden, daß A und B UND-verknüpft werden. Das Symbol & soll die Verknüpfungsart andeuten.

Wozu wird eine solche Schaltung benötigt? Ein Beispiel: eine Schneidepresse darf aus Sicherheitsgründen nur eingeschaltet werden, wenn zwei Taster betätigt werden, die so weit voneinander entfernt sind, daß sie nicht mit einer Hand bedient werden können. Die Schneidepresse wird nur eingeschaltet ($X = H$), wenn der eine Taster (mit der linken Hand) betätigt wird ($A = H$) und wenn der andere Taster (mit der rechten Hand) betätigt wird ($B = H$).

ADRESSE	OP. CODE
1000	12
1001	07
1002	06
1003	D4 10
1005	6C 03
1007	C4 01
1009	48
100A	06
100B	D4 20
100D	6C 02
100F	C4 01
1011	50
1012	00
1013	00
1014	48
1015	06
1016	58
1017	CD C3
1019	C4 00
101B	11
101C	74 E2

Wir wollen uns jetzt in einem Programm die UND-Verknüpfung von zwei Eingangssignalen ansehen. Die Eingangssignale A und B werden hier mit Hilfe der beiden Tasten **SA** und **SB** eingegeben. Das Ausgangssignal wird zusätzlich zu den beiden Eingangssignalen an der LED-Reihe angezeigt. Der Schalter S muß in der vorderen Stellung stehen.



Wir werden dieses Programm im Abschnitt 14.6 ausführlich diskutieren. Hier soll zunächst nur die UND-Verknüpfung untersucht werden. Wenn das Programm eingegeben und gestartet worden ist, lassen sich die vier Möglichkeiten durchprobieren:

Taste SA	Taste SB	rechte Leuchtdiode in der LED-Reihe
nicht betätigt	nicht betätigt	leuchtet nicht
nicht betätigt	betätigt	leuchtet nicht
betätigt	nicht betätigt	leuchtet nicht
betätigt	betätigt	leuchtet

Wir sehen, daß diese Tabelle genau mit der Schaltwert-tabelle übereinstimmt. Die Leuchtdiode für das Ausgangs-signal X leuchtet genau dann, wenn die Taste **SA** betätigt wird u n d wenn die Taste **SB** betätigt wird.

14.1.2 Die ODER-Schaltung

Wir haben schon festgestellt, daß sich die verschiedenen Verknüpfungsschaltungen dadurch unterscheiden, daß sie – jede in bestimmter Weise – dem Ausgangssignal X bei einigen der vier Eingangssignalkombinationen den Wert L, bei den anderen Eingangssignalkombinationen den Wert H zuordnen. Diese Verknüpfungen lassen sich alle mit Hilfe des Computers untersuchen. In dem Programm für die UND-Verknüpfung brauchen wir nur maximal drei Speicher-inhalte zu ändern. Für die ODER-Verknüpfung genügt die Änderung im Speicher mit der Adresse 1011.

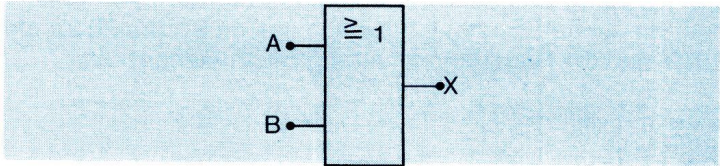
ADRESSE	OP.CODE
1011	50
1012	00
1013	00

Bei der ODER-Schaltung hat das Ausgangssignal X genau dann den Wert H, wenn das Eingangssignal A den Wert H hat o d e r wenn das Eingangssignal B den Wert H hat (oder wenn beide Eingangssignale den Wert H haben).

Schaltwerttabelle für die ODER-Schaltung:

A	B	X
L	L	L
L	H	H
H	L	H
H	H	H

Gleichung: $X = A \vee B$ (sprich: X gleich A o d e r B)
Schaltsymbol:



Das Zeichen ≥ 1 im Schaltsymbol besagt, daß das Aus-gangssignal X dann den Wert H hat, wenn die Anzahl der mit dem Wert H belegten Eingangssignale größer oder gleich 1 ist.

14.1.3 Die NAND-Schaltung

Änderungen im Programm:

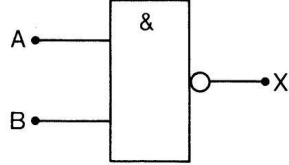
ADRESSE	OP.CODE
1011	50
1012	E4 01

Das Ausgangssignal X hat genau dann den Wert H, wenn n i c h t gilt, daß das Eingangssignal A den Wert H hat u n d daß das Eingangssignal B den Wert H hat. Wir sehen an dieser Formulierung, daß die NAND-Schaltung etwas mit der UND-Schaltung zu tun hat. NAND ist aus den beiden englischen Wörtern NOT AND (nicht und) gebildet worden. Wir können uns also vorstellen, daß die beiden Eingangssig-nale zunächst UND-verknüpft werden, daß dann dieses Zwischenergebnis verneint oder negiert wird. Zur Negation vergleiche Abschnitt 9.5. Die Negation wird durch einen Querstrich angedeutet: $\bar{0} = 1$; $\bar{1} = 0$ (sprich: Null quer gleich 1; Eins quer gleich Null). In der Schaltwerttabelle für die NAND-Schaltung ist zum Vergleich das Ergebnis der UND-Verknüpfung zusätzlich angegeben:

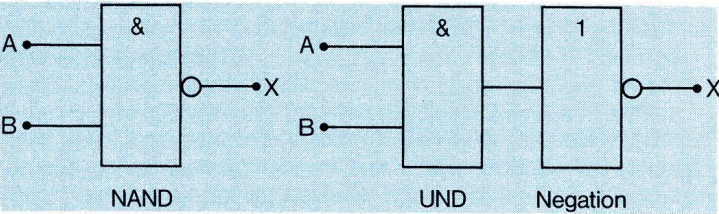
A	B	X	Zum Vergleich $A \wedge B$
L	L	H	L
L	H	H	L
H	L	H	L
H	H	L	H

Gleichung: $X = \overline{A \wedge B}$ (sprich: X gleich A und B quer).

Schaltsymbol:



Gleichung und Schaltsymbol drücken auch aus, daß zunächst die beiden Eingangssignale A und B UND-verknüpft werden, daß dann dieses Ergebnis negiert wird. Der Kreis am Ausgang des Schaltsymbols soll die Negation andeuten. Das Schaltsymbol ist aus den beiden Symbolen für die UND- und die Negationsschaltung zusammengesetzt:



Beide Darstellungen sind gleichbedeutend.

14.1.4 Die NOR-Schaltung

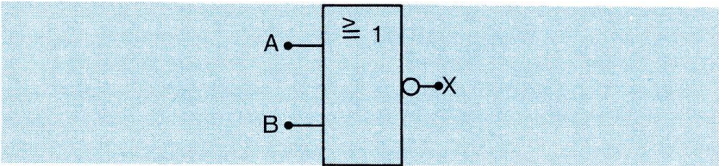
Änderungen im Programm:

ADRESSE	OP. CODE
1011 1012	50 E4 01

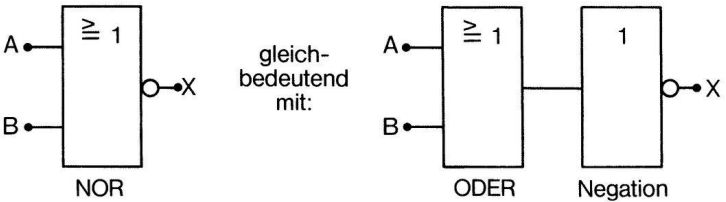
Das Ausgangssignal X hat genau dann den Wert H, wenn nicht gilt, daß das Eingangssignal A den Wert H hat oder daß das Eingangssignal B den Wert H hat. NOR ist aus den beiden englischen Wörtern NOT OR (nicht oder) gebildet worden. In der Schaltwerttabelle für die NOR-Schaltung ist zum Vergleich das Ergebnis der ODER-Verknüpfung zusätzlich angegeben:

A	B	X	zum Vergleich $A \vee B$
L	L	H	L
L	H	L	H
H	L	L	H
H	H	L	H

Gleichung: $X = \overline{A \vee B}$ (sprich: X gleich A oder B quer).
Schaltsymbol:



Auch hier drücken Gleichung und Schaltsymbol aus, daß zunächst die beiden Eingangssignale ODER-verknüpft werden, daß dann dieses Ergebnis negiert wird.



14.1.5 Die Äquivalenzschaltung

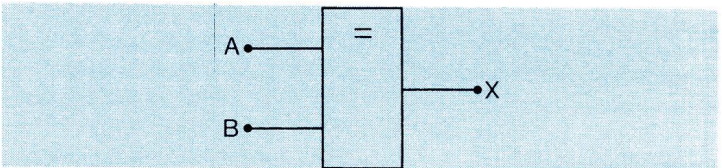
Änderungen im Programm:

ADRESSE	OP. CODE
1011 1012	60 E4 01

Das Ausgangssignal hat genau dann den Wert H, wenn beide Eingangssignale A und B den Wert L haben oder wenn beide Eingangssignale den Wert H haben, kurz, wenn beide Eingangssignale den gleichen Wert haben, wenn beide Eingangssignale äquivalent (gleichwertig) sind. Schaltwerttabelle:

A	B	X
L	L	H
L	H	L
H	L	L
H	H	H

Gleichung: $X = A \equiv B$ (sprich: X gleich A äquivalent B).
Schaltsymbol:



Das Zeichen = im Schaltsymbol besagt, daß das Ausgangssignal X genau dann den Wert H hat, wenn beide Eingangssignale den gleichen Wert haben.

14.1.6 Die Antivalenzschaltung

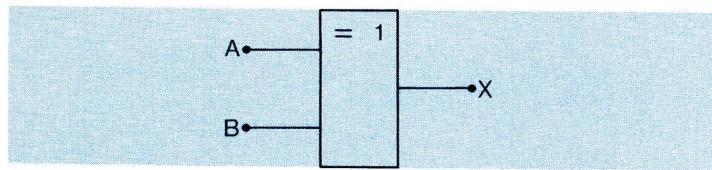
Änderungen im Programm:

ADRESSE	OP. CODE
1011	00
1012	00
1013	00

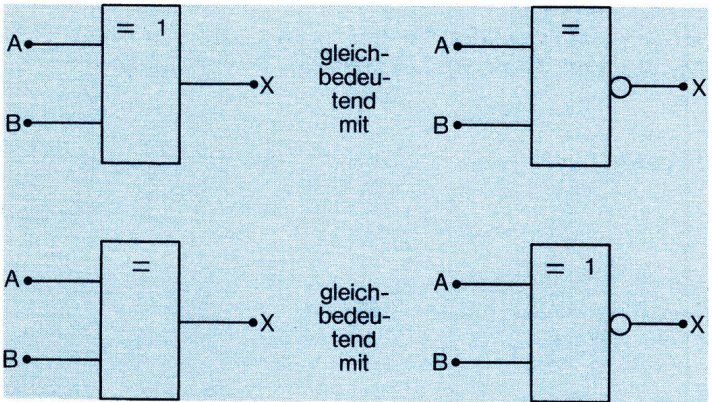
Das Ausgangssignal X hat genau dann den Wert H, wenn die beiden Eingangssignale unterschiedliche Werte haben, wenn beide Eingangssignale antivalent (gegenwertig) sind. In der Schwerttabelle für die Äquivalenzschaltung ist zum Vergleich das Ergebnis der Äquivalenzverknüpfung zusätzlich angegeben:

A	B	X	zum Vergleich $A \equiv B$
L	L	L	H
L	H	H	L
H	L	H	L
H	H	L	H

Gleichung: $X = A \neq B$ (sprich: X gleich A antivalent B).
Schaltsymbol:



Das Zeichen $= 1$ im Schaltsymbol besagt, daß das Ausgangssignal X dann den Wert H hat, wenn die Anzahl der mit dem Wert H belegten Eingangssignale genau gleich eins ist.



Die Antivalenz-Schaltung wird auch als Exklusiv-ODER-Schaltung bezeichnet (ausschließendes ODER). X hat den Wert H, wenn $A = H$ oder wenn $B = H$, aber nicht wenn $A = H$ und $B = H$ ist. Dieser Fall ist ausgeschlossen.


14.1.7 Zusammenstellung der logischen Verknüpfungen

Wir werden sehen, daß sich aus diesen logischen Verknüpfungen sehr viele neue Möglichkeiten für den Mikroprozessor ergeben. Da der Computer mit den beiden Werten 0 und 1 arbeitet, werden wir in der folgenden Zusammenstellung die bisher benutzten Schaltwerte L und H aus der Digital-elektronik wieder mit 0 und 1 bezeichnen. 0 entspricht L, 1 entspricht H.

Bezeichnung	Schalt-tabelle	Gleichung und Schalt-symbol	Beschreibung															
UND	<table><tr><th>A</th><th>B</th><th>X</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	X	0	0	0	0	1	0	1	0	0	1	1	1	$X = A \wedge B$ 	X hat genau dann den Wert 1, wenn A den Wert 1 hat u n d wenn B den Wert 1 hat
A	B	X																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
ODER	<table><tr><th>A</th><th>B</th><th>X</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	X	0	0	0	0	1	1	1	0	1	1	1	1	$X = A \vee B$ 	X hat genau dann den Wert 1, wenn A den Wert 1 hat o d e r wenn B den Wert 1 hat (oder wenn A und B den Wert 1 haben)
A	B	X																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
NAND	<table><tr><th>A</th><th>B</th><th>X</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	X	0	0	1	0	1	1	1	0	1	1	1	0	$X = \overline{A \wedge B}$ 	X hat genau dann den Wert 0, wenn A den Wert 1 hat u n d wenn B den Wert 1 hat
A	B	X																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR	<table><tr><th>A</th><th>B</th><th>X</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	X	0	0	1	0	1	0	1	0	0	1	1	0	$X = \overline{A \vee B}$ 	X hat genau dann den Wert 0, wenn A den Wert 1 hat o d e r wenn B den Wert 1 hat (oder wenn A und B den Wert 1 haben)
A	B	X																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
Äqui-valenz	<table><tr><th>A</th><th>B</th><th>X</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	X	0	0	1	0	1	0	1	0	0	1	1	1	$X = A \equiv B$ 	X hat genau dann den Wert 1, wenn A und B gleiche Werte haben
A	B	X																
0	0	1																
0	1	0																
1	0	0																
1	1	1																
Anti-valenz (Exklusiv-ODER)	<table><tr><th>A</th><th>B</th><th>X</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	X	0	0	0	0	1	1	1	0	1	1	1	0	$X = A \neq B$ 	X hat genau dann den Wert 1, wenn A und B unterschiedliche Werte haben
A	B	X																
0	0	0																
0	1	1																
1	0	1																
1	1	0																

Neben den sechs aufgeführten Verknüpfungen gibt es noch zehn weitere Möglichkeiten, zwei binäre Eingangssignale zu verknüpfen. Diese weiteren Fälle sind aber für die Anwendung uninteressant.

In dieser Zusammenstellung soll noch einmal die Negationsschaltung oder die Nicht-Schaltung erwähnt werden. Es ist eine Schaltung, die ein Eingangssignal A negiert. Sie bildet zu der einstelligen Dualzahl A das einstellige Einerkomplement (vgl. Abschnitt 9.5)

Bezeichnung	Schaltwerttabelle	Gleichung und Schaltsymbol	Beschreibung						
Negation, Nicht	<table border="1"><tr><td>A</td><td>X</td></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	X	0	1	1	0	$X = \overline{A}$ (X gleich A quer) 	X hat genau dann den Wert 1, wenn A den Wert 0 hat.
A	X								
0	1								
1	0								

14.2 Logische Verknüpfungen von zweistelligen Hexadezimalzahlen

Das Ergebnis bei einer logischen Verknüpfung von zweistelligen Hexadezimalzahlen läßt sich am einfachsten auf dem Weg über das Dualsystem bestimmen. Für achtstellige Dualzahlen ergibt sich das Verknüpfungsergebnis dadurch, daß man die im letzten Abschnitt besprochene Verknüpfung von zwei Eingangssignalen (oder zwei einstelligen Dualzahlen) auf jede Stelle der mehrstelligen Dualzahl anwendet. Die Verknüpfung an einer bestimmten Stelle ist völlig unabhängig von der Verknüpfung an den anderen sieben Stellen. Einen Übertrag wie bei der Addition oder der Subtraktion gibt es hier nicht.

Damit ist eigentlich alles gesagt. Wir müssen uns nur jeweils die vier Möglichkeiten klarmachen, die bei einer Verknüpfungsart vorkommen können. Bei der UND-Verknüpfung gibt es nur die vier Fälle $0 \wedge 0 = 0$; $0 \wedge 1 = 0$; $1 \wedge 0 = 0$; $1 \wedge 1 = 1$ (vgl. Zusammenstellung in Abschnitt 14.1.7).

Was ist $F3 \wedge 5A$? Wir wandeln die beiden Hexadezimalzahlen F3 und 5A in Dualzahlen um, verknüpfen die entsprechenden Ziffern der Dualzahlen miteinander und wandeln das Ergebnis wieder in eine Hexadezimalzahl um.

UND	$F3_{16} =$	1	1	1	1	0	0	1	1 ₂
		:	:	:	:	:	:	:	:
		:	:	:	:	:	:	:	:
$F3 \wedge 5A = 52$	$5A_{16} =$	0	1	0	1	1	0	1	0 ₂
		:	:	:	:	:	:	:	:
		:	:	:	:	:	:	:	:
		↓	↓	↓	↓	↓	↓	↓	↓
		0	1	0	1	0	0	1	0 ₂ 52 ₁₆

Äquivalenz	$F3_{16} =$	1	1	1	1	0	0	1	1 ₂
		:	:	:	:	:	:	:	:
		:	:	:	:	:	:	:	:
		↓	↓	↓	↓	↓	↓	↓	↓
$F3 = 5A = 56$	$5A_{16} =$	0	1	0	1	1	0	1	0 ₂
		:	:	:	:	:	:	:	:
		:	:	:	:	:	:	:	:
		↓	↓	↓	↓	↓	↓	↓	↓
		0	1	0	1	0	1	1	0 ₂ = 56 ₁₆

Antivalenz	$F3_{16} =$	1	1	1	1	0	0	1	1 ₂
		≠	≠	≠	≠	≠	≠	≠	≠
		:	:	:	:	:	:	:	:
		:	:	:	:	:	:	:	:
$F3 \neq 5A = A9$	$5A_{16} =$	0	1	0	1	1	0	1	0 ₂
		:	:	:	:	:	:	:	:
		:	:	:	:	:	:	:	:
		↓	↓	↓	↓	↓	↓	↓	↓
		1	0	1	0	1	0	0	1 ₂ = A9 ₁₆

Negation	$F3_{16} =$	1	1	1	1	0	0	1	1 ₂
		—	—	—	—	—	—	—	—
		↓	↓	↓	↓	↓	↓	↓	↓
$F3 = 0C$		0	0	0	0	1	1	0	0 ₂ = 0C ₁₆

Weitere Beispiele: Vgl. 1. Aufgabe zu diesem Kapitel

Entsprechend bei den anderen Verknüpfungsarten:

ODER	$F3_{16} =$	1	1	1	1	0	0	1	1 ₂
		↓	↓	↓	↓	↓	↓	↓	↓
		:	:	:	:	:	:	:	:
$F3 \vee 5A = FB$	$5A_{16} =$	0	1	0	1	1	0	1	0 ₂
		:	:	:	:	:	:	:	:
		:	:	:	:	:	:	:	:
		↓	↓	↓	↓	↓	↓	↓	↓
		1	1	1	1	1	0	1	1 ₂ = FB ₁₆

NAND	$F3_{16} =$	1	1	1	1	0	0	1	1 ₂
		⌋	⌋	⌋	⌋	⌋	⌋	⌋	⌋
		:	:	:	:	:	:	:	:
		:	:	:	:	:	:	:	:
$F3 \sim 5A = AD$	$5A_{16} =$	0	1	0	1	1	0	1	0 ₂
		:	:	:	:	:	:	:	:
		:	:	:	:	:	:	:	:
		↓	↓	↓	↓	↓	↓	↓	↓
		1	0	1	0	1	1	0	1 ₂ = AD ₁₆

NOR	$F3_{16} =$	1	1	1	1	0	0	1	1 ₂
		⌋	⌋	⌋	⌋	⌋	⌋	⌋	⌋
		:	:	:	:	:	:	:	:
		:	:	:	:	:	:	:	:
$F3 \vee 5A = 04$	$5A_{16} =$	0	1	0	1	1	0	1	0 ₂
		:	:	:	:	:	:	:	:
		:	:	:	:	:	:	:	:
		↓	↓	↓	↓	↓	↓	↓	↓
		0	0	0	0	0	1	0	0 ₂ = 04 ₁₆

14.3 Die logischen Befehle

Unser Mikroprozessor kennt AND-, OR- und XOR-Befehle (UND, ODER und Exklusiv-Oder). Bei den ersten drei Befehlen werden die Inhalte vom Extension-Register E und vom Akkumulator A verknüpft:

BESCHREIBUNG	IMNEM.CODE	OP.CODE	OPERATION
AND-VERKNUEPFE DEN INHALT DES EXTENSION-REG ZUM AKKU-INHALT	AND A,E	50	$\langle A \rangle := \langle A \rangle \wedge \langle E \rangle$
OR-VERKNUEPFE DEN INHALT DES EXTENSION-REGISTERS ZUM AKKU-INHALT	OR A,E	58	$\langle A \rangle := \langle A \rangle \vee \langle E \rangle$
XOR-VERKNUEPFE DEN INHALT DES EXTENSION-REG ZUM AKKU-INHALT	XOR A,E	60	$\langle A \rangle := \langle A \rangle \oplus \langle E \rangle$

Der Inhalt des Extension-Registers bleibt dabei unverändert.

Befehle mit unmittelbarer Adressierung:

BESCHREIBUNG	MNEM.CODE	OP.CODE	OPERATION
AND-VERKUEPFE UNMITTELBAR ZUM INHALT DES AKKUMULATORS	AND A,=ZAHL	04 XX	$\langle A \rangle := \langle A \rangle \wedge \text{ZAHL}$
IOR-VERKUEPFE UNMITTELBAR ZUM INHALT DES AKKUMULATORS	IOR A,=ZAHL	0C XX	$\langle A \rangle := \langle A \rangle \vee \text{ZAHL}$
XOR-VERKUEPFE UNMITTELBAR ZUM INHALT DES AKKUMULATORS	XOR A,=ZAHL	0E XX	$\langle A \rangle := \langle A \rangle \oplus \text{ZAHL}$
AND-VERKUEPFE UNMITTELBAR ZUM INHALT DES STATUS-REGISTERS	AND S,=ZAHL	39 XX	$\langle S \rangle := \langle S \rangle \wedge \text{ZAHL}$
IOR-VERKUEPFE UNMITTELBAR ZUM INHALT DES STATUS-REGISTERS	IOR S,=ZAHL	3B XX	$\langle S \rangle := \langle S \rangle \vee \text{ZAHL}$

Mit XX im Operationscode wird die zweistellige Hexadezimalzahl bezeichnet, die mit dem Akkuinhalt bzw. mit dem Inhalt des Status-Registers S verknüpft wird. Bit 4 und Bit 5 im Status-Register werden durch diese Befehle nicht verändert. Sense A (SA) und Sense B (SB) sind Nur-Lese-Flags (vgl. Abschnitt 12.1).

Befehle mit direkter Adressierung:

BESCHREIBUNG	MNEM.CODE	OP.CODE	OPERATION
AND-VERKUEPFE AUS DEM SPEICHER ZUM AKKU-INHALT IDIREKTE ADRESSIERUNG	AND A,BEZ	05 XX	$\text{ADR} := \text{FF00} + \text{XX}$ $\langle A \rangle := \langle A \rangle \wedge \langle \text{ADR} \rangle$
IOR-VERKUEPFE AUS DEM SPEICHER ZUM AKKU-INHALT IDIREKTE ADRESSIERUNG	IOR A,BEZ	0D XX	$\text{ADR} := \text{FF00} + \text{XX}$ $\langle A \rangle := \langle A \rangle \vee \langle \text{ADR} \rangle$
XOR-VERKUEPFE AUS DEM SPEICHER ZUM AKKU-INHALT IDIREKTE ADRESSIERUNG	XOR A,BEZ	0E XX	$\text{ADR} := \text{FF00} + \text{XX}$ $\langle A \rangle := \langle A \rangle \oplus \langle \text{ADR} \rangle$

BEZ steht für die Bezeichnung eines Speichers oder eines Speicherinhalts. Dieser Speicher muß eine Adresse zwischen FF00 und FFFF haben. Das zweite Byte dieser Adresse ist beim Operationscode mit XX bezeichnet. Die vollständige Adresse ADR ergibt sich dann als Summe FF00 + XX. Der Inhalt des Speichers mit der Adresse ADR wird mit dem Akkuinhalt verknüpft.

Damit sind alle logischen Befehle aufgeführt, die unser Mikroprozessor kennt. Die anderen logischen Verknüpfungsarten (NAND, NOR, Äquivalenz, Negation) können mit Hilfe der angegebenen Befehle realisiert werden. Wir kommen darauf in diesem Kapitel zurück. Jetzt wollen wir zunächst einige der angegebenen Befehle erproben. Wir lassen den Mikroprozessor die Verknüpfungen ausführen, für die es unmittelbar Befehle gibt.

NR.	MARKE	MNEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	CALL EIN 2ST	1000	16	IEINGABE: 1. ZAHL
2.		LD A,ZWSP	1001	05 08	
3.		IST A,ZAHL1	1003	0D E1	
4.		CALL EIN 2ST	1005	16	IEINGABE: 2. ZAHL
5.		LD A,ZWSP	1006	05 08	
6.		AND A,ZAHL1	1008	05 E1	$\langle A \rangle := \langle A \rangle \wedge 1. \text{ ZAHL}$
7.		CALL ANZ A	100A	18	
8.		BRA BEGINN	100B	17 4 F3	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZWSP	FFD8	ZWISCHENSPEICHER
ZAHL1	FFE1	1. EINGEGEBENE ZAHL

Mit Hilfe dieses Programms lassen sich zweistellige Hexadezimalzahlen UND-verknüpfen. Nach Programmeingabe und -start wird die erste Zahl eingegeben (z. B. F3). Nach Betätigung einer Funktionstaste (außer **A↔D**) kann die zweite Zahl eingegeben werden (z. B. 5A). Wird jetzt wieder eine Funktionstaste betätigt, so erscheint das Ergebnis der logischen Verknüpfung (z. B. 52).

Überschreiben wir im Speicher 1008 den Inhalt D5 mit DD bzw. mit E5, so können wir zwei weitere Rechnungen aus Abschnitt 14.2 überprüfen: $F3 \vee 5A = FB$; $F3 \oplus 5A = A9$.

14.4 Negation mit Hilfe des XOR-Befehls

Wenn das letzte Programm noch eingetastet ist, lassen wir den Mikroprozessor $F3 \oplus FF$ berechnen. Wir können die Aufgabe natürlich auch allein lösen:

$$F3_{16} = 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1_2$$
$$\begin{matrix} : & : & : & : & : & : & : & : \\ \oplus & \oplus & \oplus & \oplus & \oplus & \oplus & \oplus & \oplus \\ : & : & : & : & : & : & : & : \end{matrix}$$

$$F3 \oplus FF = 0C$$
$$FF_{16} = 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1_2$$
$$\begin{matrix} : & : & : & : & : & : & : & : \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0_2 = 0C_{16} \end{matrix}$$

Wir erhalten dasselbe Ergebnis, das wir in Abschnitt 14.2 für F3 ermittelt hatten. Das ist kein Zufall, denn wenn eine achtstellige Dualzahl A mit 1111 1111 XOR-verknüpft wird, so ergibt sich an jeder Stelle, wo A eine 0 hat, durch Verknüpfung mit 1 eine 1, an jeder Stelle, wo A eine 1 hat, durch Verknüpfung mit 1 eine 0.

Entsprechend bleibt eine zweistellige Hexadezimalzahl A unverändert, wenn sie mit 00₁₆ XOR-verknüpft wird.

Wesentlicher ist aber die folgende Feststellung: Mit Hilfe des XOR-Befehls kann eine Zahl teilweise negiert werden. Wird eine Zahl A z. B. mit 0F XOR-verknüpft, so werden nur die vier niederwertigsten Bits von A negiert. Die anderen vier Bits von A bleiben unverändert. In der Praxis kommt es häufig vor, daß nur ein Bit eines Speicherinhalts geändert oder negiert werden soll. Eine Exklusiv-ODER-Verknüpfung mit 10₁₆ ändert nur Bit 4.

A	=	A ₇	A ₆	A ₅	A ₄	A ₃	A ₂	A ₁	A ₀
10 ₁₆	=	0	0	0	1	0	0	0	0
A \oplus 10 ₁₆	=	A ₇	A ₆	A ₅	A ₄	A ₃	A ₂	A ₁	A ₀

Mit A₀ bis A₇ sind die acht Bits, die acht Dualziffern der Zahl bezeichnet, die mit 10₁₆ verknüpft wird. Die Bits A₀ bis A₃ und A₅ bis A₇ bleiben unverändert. Wird eine 1 mit 0 verknüpft, ergibt sich eine 1 (da antivalent), wird eine 0 mit 0 verknüpft, ergibt sich eine 0 (da nicht antivalent). A₄ wird mit 1 verknüpft. Wenn A₄ den Wert 1 hat, wird $\overline{A_4} = 0$; wenn A₄ den Wert 0 hat, wird $\overline{A_4} = 1$.

In den Abschnitten 14.1.3, 14.1.4 und 14.1.5 wurde in dem Programm mit dem Befehl XOR A, = 01 (Operationscode E4 01 in den Speichern 1012 und 1013) erreicht, daß das einstellige Verknüpfungsergebnis jeweils negiert wurde. Aus der UND-Verknüpfung wurde durch diese Negation die NAND-Verknüpfung. Aus ODER wurde NOR, aus Antivalenz wurde Äquivalenz.

Das Ein- und Ausschalten der gelben Leuchtdiode bei Spiel 2: „gelbes Blinklicht“ wird auch mit einem XOR-Befehl erreicht.

INR.	MARKE	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	ILD A,=0	1000	IC4 00	
2.		ILD S,A	1002	IO7	<S>:=00
3.	BLINK	ILD A,S	1003	IO6	
4.		XOR A,=04	1004	IE4 04	<F2> WIRD GEAENDERT
5.		ILD S,A	1006	IO7	
6.		ILD EA,=0	1007	IS4 00 00	
7.		ICALL VERZ	100A	IID	VERZOEGERUNG
8.		IBRA BLINK	100B	I74 F6	

Der Inhalt des Status-Registers wird im 3. Befehl in den Akku geladen, da der XOR-Befehl nur mit dem Akkuinhalt ausgeführt werden kann. Nach Negation von Bit 2 kommt der veränderte Akkuinhalt wieder in das Status-Register. Bit 2 ist negiert worden, der Leuchtzustand der gelben Leuchtdiode wurde verändert. Dieses Spiel (vgl. Abschnitt 1.3.2) wird mit den Tasten **SP**, **2** aufgerufen. Das zugehörige Programm steht im Betriebsprogramm, im ROM in den Speichern von Adresse 03CC bis 03D8.

14.5 Zusätzliche Sprungbedingungen

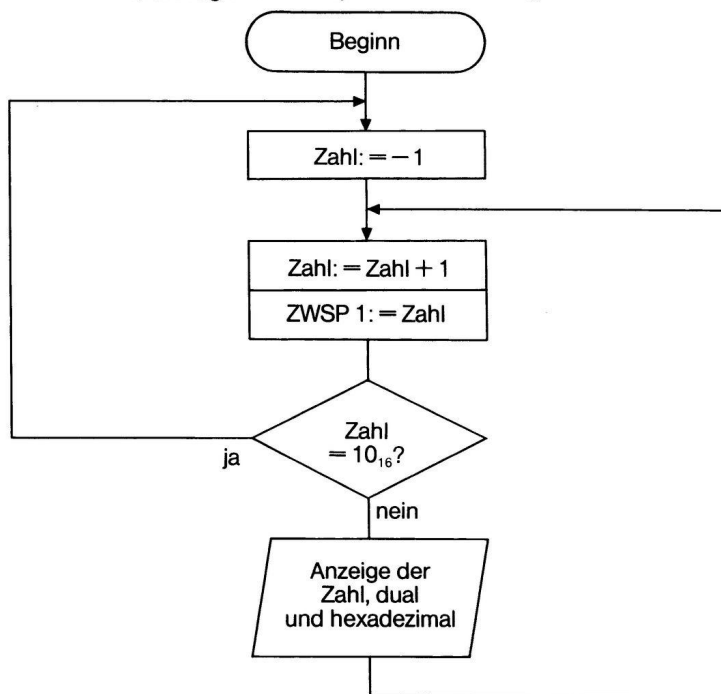
In Abschnitt 10.4 haben wir die drei Befehle für bedingte Sprünge kennengelernt. Ob ein Sprung ausgeführt wird oder nicht, hängt grundsätzlich vom Akkuinhalt ab, es hängt davon ab, ob der Akkuinhalt positiv, gleich Null oder ungleich Null ist. Mit Hilfe des XOR-Befehls können wir jetzt auch einen bedingten Sprung programmieren, der dann ausgeführt wird, wenn der Akkuinhalt einen bestimmten vorgegebenen Wert angenommen hat. Grundlage hierfür ist die folgende Überlegung: wird eine beliebige zweistellige Hexadezimalzahl A mit sich selbst XOR-verknüpft, so ist das Ergebnis in jedem Fall 00: $A \oplus A = 00$. Wird A mit einer beliebigen anderen Zahl XOR-verknüpft, so ist das Ergebnis ungleich Null.

Wir haben dieses Verfahren schon benutzt. In den Programmen in den Abschnitten 4.3.1 und 4.3.2 wurde eine Zahl von 00 bis 0F hochgezählt. Es wurde dabei abgefragt, ob die Zahl schon den Wert 10_{16} erreicht hatte. In diesem Fall wurde der Zähler durch einen Sprung zum Programmbeginn von vorn gestartet. Das ausführliche Programm von Abschnitt 4.3.2 soll noch einmal wiedergegeben werden. Schalter S nach vorn.

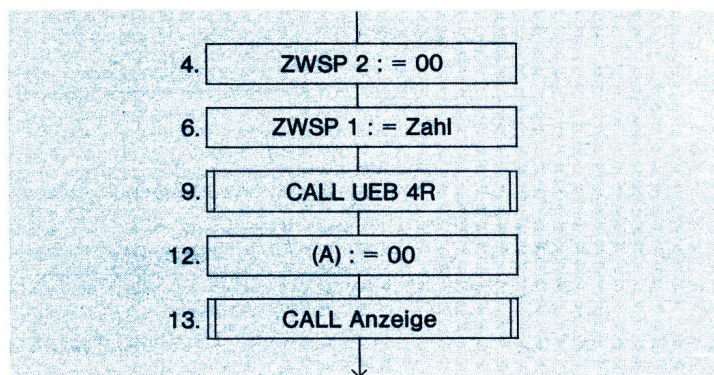
INR.	MARKE	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	ILD A,=0FF	1000	IC4 FF	
2.		IST A,ZAHL	1002	ICD E0	ZAHL:=−1
3.		ILD A,=0	1004	IC4 00	
4.		IST A,ZWSP2	1006	ICD D9	
5.	WEITER	ILD A,ZAHL	1008	I95 E0	ZAHL:=ZAHL+1
6.		IST A,ZWSP1	100A	ICD D0	
7.		XOR A,=010	100C	IE4 10	
8.		IBZ BEGINN	100E	IC6 F0	SPRUNG, WENN ZAHL=010
9.		ICALL UEB 4R	1010	I13	
10.		ILD A,ZAHL	1011	IC5 E0	ZAHL ZUR LED-REIHE
11.		IST A,LED-R	1013	ICD C3	
12.		ILD A,=0	1015	IC4 00	
13.		ICALL ANZEIGE	1017	I11	
14.		IBRA WEITER	1018	I74 EE	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZAHL	FF00	
ZWSP1	FF08 ZWISCHENSPEICHER	
ZWSP2	FF09	
LED-R	FFC3 LED-REIHE	

Der Programmablaufplan für dieses Programm:



Dieser Programmablaufplan gibt nicht alle Befehle einzeln wieder. Die Anzeige der Hexadezimalzahl besteht aus folgenden Befehlen:



Zur Anzeige der Dualzahl wird die Zahl zusätzlich vor dem CALL zum Speicher mit der Adresse FFC3 gebracht.

Jetzt zu der Sprungbedingung. Beim ersten Durchlauf der Schleife wird der Akkuinhalt 00_{16} im 7. Befehl mit 10_{16} XOR-verknüpft. Da das Ergebnis 10_{16} ist, wird der Sprung zum Programmbeginn nicht ausgeführt. Es folgt die Anzeige der Zahl 0. Beim nächsten Schleifendurchlauf wird der Akkuinhalt 01_{16} mit 10_{16} XOR-verknüpft. Da das Ergebnis 11_{16} ist, folgt wieder die Anzeige. Erst im siebzehnten Durchlauf ist das Ergebnis der XOR-Verknüpfung 00 ($10_{16} \oplus 10_{16} = 00_{16}$). Jetzt wird der Sprung zum Programmangefang durchgeführt.

Beim Rückwärtszähler im Abschnitt 4.3.3 wird der Akkuinhalt vor dem bedingten Sprung BZ BEGINN (Adresse 100E und 100F) mit FF XOR-verknüpft. Hier wird der Sprung erst ausgeführt, wenn der Akkuinhalt FF (oder −1) geworden ist.

14.6 Maskieren

Bei dieser Überschrift denkt man ans Theater oder an den Karneval, aber wohl kaum an den Mikroprozessor. Trotzdem verbirgt sich hinter diesem Begriff eine ganz wesentliche Anwendung einer logischen Operation, der UND-Verknüpfung. Wir wollen uns das Verfahren zunächst an einem Programm ansehen.

Schalter S nach vorn.

INR.	MARKE	MMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	CALL BLANK	1000	12	
2.		IST A,ZAHL	1001	CD E0	ZAHL:=00
3.	WEITER	ILD A,ZAHL	1003	95 E0	ZAHL:=ZAHL+1
4.		AND A,=027	1005	D4 27	(A):=(A) ^ 027
5.		IST A,LED-R	1007	CD C3	
6.		ILD A,=020	1009	IC4 20	
7.		CALL ANZEIGE	100B	11	
8.		BRA WEITER	100C	74 F5	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZAHL	FFE0	
LED-R	FFC3	

Wir haben einen Dualzähler mit der Anzeige an der LED-Reihe programmiert. Wenn wir das Programm starten, haben wir den Eindruck, daß vier Leuchtdioden defekt sind, sie leuchten nicht.

Die Zahl im Speicher mit der Adresse FFE0 wird kontinuierlich von 00 bis FF hochgezählt; bevor wir diese Zahl jedoch zur LED-Reihe bringen, maskieren wir sie. Die Zahl sei z. B. gerade 7D. Was passiert, wenn wir sie im 4. Befehl mit 27₁₆ UND-verknüpfen? Das Verknüpfungsergebnis bestimmen wir im Dualsystem:

7D ₁₆ =	0	1	1	1	1	1	0	1 ₂
27 ₁₆ =	0	0	1	0	0	1	1	1 ₂
7D ₁₆ ^ 27 ₁₆ =	0	0	1	0	0	1	0	1 ₂ = 25 ₁₆

Die Zahl 7D wird an den Dualstellen nicht verändert, wo mit einer dualen 1 AND-verknüpft wird, an allen anderen Stellen ergibt sich eine 0. Die Zahl 27₁₆, oder besser das Bit-Muster 0010 0111 wird als Maske bezeichnet. Wir legen der Zahl 7D gewissermaßen diese Maske auf und sehen uns nur das an, was die Maske nicht verdeckt.

7D	→	0	1	1	1	1	1	0	1
Maske	→								
7D mit Maske	→			1		1	0	1	
Bit		7	6	5	4	3	2	1	0

Wir können in unserem Programm natürlich die Maske beliebig verändern. Wählen wir z. B. die Maske 12₁₆, d. h. ersetzen wir den bisherigen 4. Befehl durch „AND A, = 012“ (Operationscode D4 12), so blinken nur noch zwei Leuchtdioden, die eine achtmal so oft wie die andere. In der Anwendung wird sehr häufig ein Bit aus einem Speicher- oder Registerinhalt ausmaskiert. Wenn uns z. B. nur der Wert von Sense A im Statusregister interessiert, wählen wir die Maske 10₁₆. Für Sense B wählen wir entsprechend die Maske 20₁₆.

Sehen wir uns hierzu noch einmal das Programm aus Abschnitt 14.1.1 an.

INR.	MARKE	MMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	CALL BLANK	1000	12	(A):=00; (E):=00
2.		ILD S,A	1001	07	(S):=00
3.	EIN-SA	ILD A,S	1002	06	
4.		AND A,=010	1003	D4 10	MASKIEREN VON BIT 4; SA
5.		IBZ EIN-SB	1005	6C 03	SPRUNG, WENN (SA)=0
6.		ILD A,=1	1007	IC4 01	
7.		ILD E,A	1009	48	(E):=01, WENN (SA)=1
8.	EIN-SB	ILD A,S	100A	06	
9.		AND A,=020	100B	D4 20	MASKIEREN VON BIT 5; SB
10.		IBZ VERKN	100D	6C 02	
11.		ILD A,=1	100F	IC4 01	(A):=01, WENN (SB)=1
12.	VERKN	AND A,E	1011	50	(A):=(A)^(E)
13.		INOP	1012	00	ZWEI SPEICHERPLÄTZE
14.		INOP	1013	00	FÜR XOR-BEFEHL
15.		ILD E,A	1014	48	ERGEBNIS IM EXT.-REG.,
16.		ILD A,S	1015	06	BIT 0
17.		IOR A,E	1016	58	(SA) UND (SB) IM AKKU,
18.		IST A,LED-R	1017	CD C3	BIT 4 UND BIT 5
19.		ILD A,=08	1019	IC4 08	
20.		CALL ANZEIGE	101B	11	
21.		BRA BEGINN	101C	74 E2	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
LED-R	FFC3	

Noch einige Bemerkungen dazu:
Nach Bearbeitung des CALLS BLANK sind Akku und Extension-Register gelöscht. Sie brauchen also nicht mehr mit 00 geladen zu werden.
Im 2. Befehl wird das Status-Register gelöscht. In ihm steht nach dem CALL ANZEIGE ein Inhalt ungleich 00. Man könnte den dritten Befehl für überflüssig halten. Zur Erinnerung soll noch einmal wiederholt werden, daß mit dem Befehl LD S,A nur die Bits 0 bis 3, 6 und 7 vom Akku in das Statusregister geladen werden (vgl. Abschnitt 12.1, Seite 12/4). Im 3. Befehl wird der vollständige Inhalt vom Stausregister in den Akku geladen, also auch die Werte von SA und SB.
Im 4. Befehl wird Bit 4 ausmaskiert. Wurde SA betätigt, so steht jetzt 10₁₆ im Akku, das Extension-Register wird mit 01 geladen. Wurde SA nicht betätigt, so ist (A) nach dem Maskieren 00, es folgt ein Sprung zur Marke EIN-SB (Einlesen der SB-Taste, Eingangssignal B). Die Taste SB wird mit den Befehlen 8 bis 11 entsprechend untersucht.
Im 12. Befehl werden die beiden Eingangssignale verknüpft. In diesem Speicher hatten wir zur Untersuchung der anderen logischen Befehle den Operationscode verändert, gegebenenfalls mit nachfolgender Negation (Speicher 1012 und 1013).
In den Befehlen 15 bis 17 wird der Wert des Ausgangssignals (Bit 0) mit den Werten der Eingangssignale (Bit 4 und Bit 5) ODER-verknüpft. Diese drei Werte werden dann gemeinsam an der LED-Reihe angezeigt.

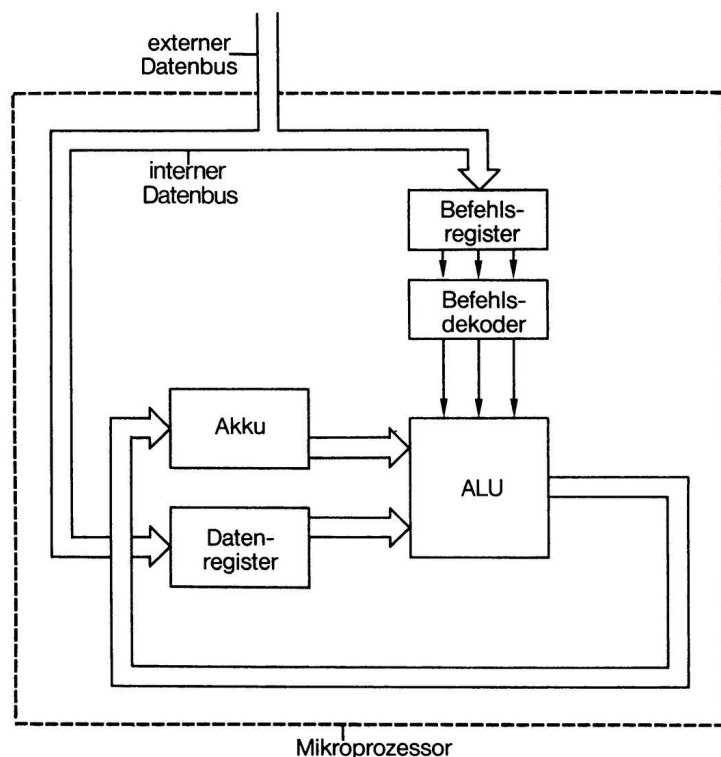
14.7 Ein Bit löschen – ein Bit setzen

Im Abschnitt 14.4 haben wir gesehen, daß mit dem XOR-Befehl der Wert eines Bits verändert oder negiert werden kann, ohne daß die anderen Bits verändert werden. Mit Hilfe der logischen Befehle läßt sich auch ein Bit löschen oder setzen, ohne daß die restlichen sieben Bits „angefast“ werden. Aufgrund der uns zur Verfügung stehenden Befehle ist das im Akku oder im Statusregister möglich (vgl. Abschnitt 14.3). Der Befehl AND A, = 0EF sorgt dafür, daß Bit 4 im Akku gelöscht wird, ohne daß sich die anderen Bits ändern.

Mit dem Befehl OR A, = 040 wird Bit 6 im Akku gesetzt (oder es bleibt gesetzt), egal was vorher im Akku stand, unabhängig davon, was an den anderen Stellen im Akku steht. In dem folgenden Programm setzen und löschen wir im Wechsel Bit 3 im Statusregister. Wir schreiben also ein Programm für ein rotes Blinklicht. Wir erinnern uns: Diese Aufgabe ist auch mit Hilfe des XOR-Befehls zu lösen (vgl. Abschnitt 14.4), aber wir wollen hier eine andere Möglichkeit erproben.

NR.	MARKE	INMEM. CODE	ADR.	OP. CODE	BEMERKUNGEN
1.	BEGINN	AND S,=0F7	1000	39 F7	(F3):=0
2.		CALL VERZ	1002	1D	
3.		OR S,=08	1003	3B 08	(F3):=1
4.		CALL VERZ	1005	1D	
5.		BRA BEGINN	1006	74 F8	

Der 1. Befehl AND S, = 0F7 bildet die UND-Verknüpfung des Inhalts vom Statusregister mit der Dualzahl 1111 0111. Bit 3 ist hinterher in jedem Fall gelöscht, die anderen Bits werden nicht verändert. Der 3. Befehl OR 5, = 08 bildet im Status-Register die ODER-Verknüpfung mit der Dualzahl 0000 1000. Bit 3 ist hinterher in jedem Fall gesetzt, die anderen Bits bleiben unverändert. Da nach Beendigung eines Befehls CALL VERZ das EA-Register den Inhalt 0000 hat, braucht es vor dem nächsten CALL VERZ nicht mit 0000 geladen werden. Wenn wir das Programm starten, blinkt die rote Leuchtdiode, unabhängig davon, welcher Inhalt sonst im Status-Register steht. Es kann z. B. gleichzeitig die gelbe Leuchtdiode angeschaltet sein. Wir erreichen das Leuchten dieser Diode durch folgende Tastenfolge: **RS**, **CPU**, **5**, **4**, **CPU**, **A+D**, **RUN**.



Mit Datenbus (engl. data bus) wird die Gesamtheit von acht parallelen Datenleitungen bezeichnet. In unserem Beispiel wurde auf dem Datenbus zunächst das erste Byte des Operationscodes in das Befehlsregister, anschließend eine achtstellige Dualzahl in das Datenregister geladen. Jetzt können die Inhalte von Akku und Datenregister in der ALU verknüpft werden. Der Befehlsdekodierer entscheidet durch bestimmte Steuersignale, welche Verknüpfung in der ALU durchgeführt wird. Das Ergebnis der Verknüpfung wird dann wieder in den Akku geschrieben.

14.8 Die ALU – die arithmetische, logische Einheit

Wir wissen inzwischen, daß der Mikroprozessor arithmetische und logische Operationen ausführen kann: arithmetische Operation: addieren, subtrahieren; logische Operation: AND-, OR- XOR-Verknüpfungen. Alle diese Operationen werden in der ALU ausgeführt. ALU ist die Abkürzung für die englische Bezeichnung „Arithmetic Logic Unit“ (arithmetische, logische Einheit, Rechenwerk).

Soll eine dieser Operationen ausgeführt werden, so wird zunächst das erste Byte des Operationscodes in das Befehlsregister geladen und dekodiert. Dieses Befehlsregister dient nur dazu, den auszuführenden Befehl im Mikroprozessor zwischenspeichern. Nach der Dekodierung oder Entschlüsselung des Inhaltes im Befehlsregister steht fest, was weiter zu tun ist (vgl. Abschnitt 7.2). Der Mikroprozessor weiß dann, ob z. B. die Daten, die mit dem Inhalt des Akkus verknüpft werden sollen, im Extension-Register stehen oder ob er sie aus einem bestimmten Speicher laden muß. Er weiß, ob eine Verknüpfung mit dem Inhalt des EA-Registers oder mit dem Inhalt des Akkus vorgenommen werden soll. Er weiß, welche Verknüpfung gewünscht wird.

Soll eine Zahl aus einem Speicher im RAM-Bereich mit dem Akkuinhalt verknüpft werden, so wird jetzt diese Zahl in das Datenregister im Mikroprozessor geholt. Jetzt sind die Vorbereitungen abgeschlossen. Die Operation kann ausgeführt werden.

14.9 Aufgaben zu Kapitel 14

1. Es ist jeweils die Hexadezimalzahl zu bestimmen, die sich bei den folgenden Verknüpfungen ergibt:

1.1 $D2 \wedge 97 =$
 $D2 \vee 97 =$
 $\overline{D2} \wedge 97 =$
 $\overline{D2} \vee 97 =$
 $D2 \equiv 97 =$
 $D2 \neq 97 =$

1.2 $A8 \wedge A8 =$
 $A8 \vee A8 =$
 $\overline{A8} \wedge A8 =$
 $\overline{A8} \vee A8 =$
 $A8 \equiv A8 =$
 $A8 \neq A8 =$

1.3 $73 \wedge FF =$
 $73 \vee FF =$
 $\overline{73} \wedge FF =$
 $\overline{73} \vee FF =$
 $73 \equiv FF =$
 $73 \neq FF =$

1.4 $3E \wedge 00 =$
 $3E \vee 00 =$
 $\overline{3E} \wedge 00 =$
 $\overline{3E} \vee 00 =$
 $3E \equiv 00 =$
 $3E \neq 00 =$

2. Im Abschnitt 14.3 wurde ein Programm für die UND-Verknüpfung zweier Hexadezimalzahlen angegeben. Dieses Programm wurde anschließend verändert (ODER-, XOR-Verknüpfung). Das Programm soll jetzt so verändert werden, daß

- 2.1 die NAND-Verknüpfung,
- 2.2 die NOR-Verknüpfung,
- 2.3 die Äquivalenzverknüpfung ausgeführt wird.

3. Es soll ein Programm geschrieben werden, das eine eingegebene zweistellige Hexadezimalzahl negiert und dann das Ergebnis anzeigt.

4. Das Spiel 4: Zähler (vgl. Abschnitt 1.3.4) läßt sich mit Hilfe der Tasten (**RS**,) **SP**, **4** starten. Das zugehörige Programm steht im ROM von Adresse 0405 bis Adresse 0423. Diese Programm soll analysiert werden. Zu den im ROM stehenden Operationscodes sollen die Befehle im mnemonischen Code angegeben werden. Außerdem soll ein Programmablaufplan gezeichnet werden.

15. Schiebe- und Rotationsbefehle

15.1 Lauflicht rechts

Zunächst soll das folgende Programm eingetastet werden:

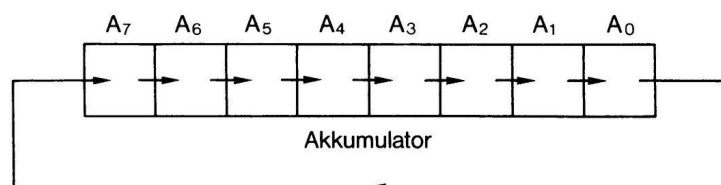
NR.	MARKE	MNEM. CODE	ADR.	OP. CODE
1.	BEGINN	ICALL .BLANK	1000	12
2.		ILD A,=080	1001	1C 80
3.		IST A,ZAHL	1003	1C D0
4.	WEITER	IST A,LED-R	1005	1C D3
5.		ILD A,=010	1007	1C 10
6.		ICALL ANZEIGE	1009	11
7.		ILD A,ZAHL	100A	1C5 E0
8.		IRR A	100C	13E
9.		IST A,ZAHL	100D	1C D0
10.		IBRA WEITER	100F	174 F4

DATENSPEICHER, BEZEICHNUNG	ADR.
ZAHL	FFE0
LED-R	FFC3

Der Schalter S muß in der vorderen Stellung stehen. Nach dem Programmstart erkennen wir, daß wir ein Lauflicht programmiert haben. Die Dioden der LED-Reihe leuchten nacheinander auf. An der LED-Reihe scheint ein Licht nach rechts zu laufen.

In den ersten drei Befehlen wird die Anzeige gelöscht und der Speicher mit der Adresse FFE0 mit der Hexadezimalzahl 80₁₆ geladen. In der Schleife vom 4. bis zum 10. Befehl wird die Zahl aus dem Speicher FFE0 zunächst an der LED-Reihe angezeigt, dann wird diese Zahl verändert, schließlich erfolgt ein Sprung zur Marke WEITER.

Die Veränderung der Zahl geschieht im wesentlichen im 8. Befehl, der mit dem mnemonischen Code RR A bezeichnet ist. RR A ist die Abkürzung für die englische Beschreibung „Rotate Accumulator Right“, rotiere den Akkuinhalt um eine Stelle nach rechts. Das folgende Bild soll diese Operation verdeutlichen:



Mit A₀, A₁, . . . , A₇ sind die acht Stellen (die acht Bit) des Akkuinhaltes bezeichnet. Das Bild spricht für sich. Die acht Bit werden gleichzeitig um eine Stelle nach rechts verschoben. Die Besonderheit dieses Rotationsbefehles ist die, daß der Wert des niederwertigsten Bits (A₀) nicht verloren geht, sondern links wieder in den Akku hineingeschoben wird (A₇).

Im Vergleich mit Spiel 3: Lauflicht (vgl. Abschnitt 1.3.3) fallen zwei Unterschiede auf:

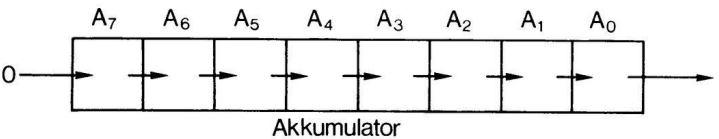
- 1) Bei dem hier untersuchten Programm leuchten die Leuchtdioden nicht so hell,
- 2) zwischen zwei Durchläufen des Lauflichtes wird keine Pause gemacht.

Der erste Unterschied liegt an der anderen Ansteuerungsart der LED-Reihe. Hiermit werden wir uns erst später beschäftigen.

Die Pause zwischen zwei Durchläufen des Lauflichtes können wir durch eine Abänderung unseres Programms auch erreichen. Wir ersetzen zunächst den Inhalt 3E im Speicher mit der Adresse 100C durch den neuen Inhalt 3C. Wenn wir anschließend das veränderte Programm starten, dann werden wir – vielleicht mit einiger Enttäuschung – feststellen, daß unser Lauflicht nur einmal läuft.

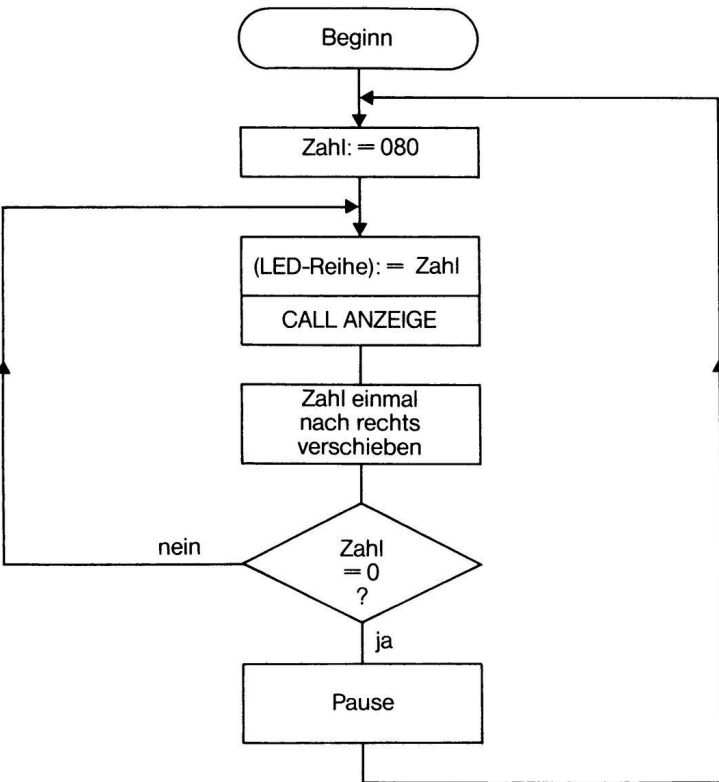
Wir haben den Rotationsbefehl RR A durch einen Schiebepfehl SR A ersetzt. SR A ist die Abkürzung für die englische Beschreibung „Shift Accumulator Right“, schiebe den Akkuinhalt um eine Stelle nach rechts.

Wir verdeutlichen diese Operation wieder durch ein kleines Bild:



Wenn diese Operation einmal ausgeführt wird, steht ganz links im Akku eine 0, der ursprüngliche Wert von Bit A₀ ist verloren.

Wenn diese Operation in unserem Programm achtmal ausgeführt worden ist, ist die zunächst eingegebene Hexadezimalzahl 80₁₆ über 40₁₆, 20₁₆, . . . , 02₁₆, 01₁₆ schließlich Null geworden. Es liegt auf der Hand, was jetzt zu tun ist. Das Programm muß nach der gewünschten Pause wiederholt werden.



Wir ersetzen den unbedingten Sprung (10. Befehl) durch einen bedingten Sprung, programmieren dann die gewünschte Verzögerung und anschließend den unbedingten Sprung zum Programmbeginn.

NR.	MARKE	INMEM.CODE	ADR.	OP.CODE
1.	BEGINN	CALL BLANK	1000	12
2.		ILD A,=080	1001	1C4 80
3.		IST A,ZAHL	1003	1CD E0
4.	WEITER	IST A,LED-R	1005	1CD C3
5.		ILD A,=010	1007	1C4 10
6.		ICALL ANZEIGE	1009	111
7.		ILD A,ZAHL	100A	1C5 E0
8.		ISR A	100C	13C
9.		IST A,ZAHL	100D	1CD E0
10.		BNZ WEITER	100F	17C F4
11.	PAUSE	ILD EA,=0	1011	184 00 00
12.		ICALL VERZ	1014	11D
13.		ICALL VERZ	1015	11D
14.		IBRA BEGINN	1016	174 E8

DATENSPEICHER, BEZEICHNUNG	ADR.
ZAHL	1FFE0
LED-R	1FFC3

15.2 Die sieben Befehle

Bevor wir unser Lauflicht-Programm erneut verändern, sollen alle sieben Schiebe- und Rotationsbefehle zusammengestellt werden.

3 Befehle: Schieben nach rechts

BESCHREIBUNG

ISCHIEBE DEN INHALT VOM AKKUMULATOR
UM EINE STELLE NACH RECHTS

MNEM. CODE

SR A

OP. CODE

3C

OPERATION:

A7 A6 A5 A4 A3 A2 A1 A0

0

Akkumulator

BESCHREIBUNG

ISCHIEBE DIE INHALTE VON CY/L-REGISTER UND
AKKUMULATOR UM EINE STELLE NACH RECHTS

MNEM. CODE

SRL A

OP. CODE

3D

OPERATION:

A7 A6 A5 A4 A3 A2 A1 A0

CY/L

Akkumulator

BESCHREIBUNG

ISCHIEBE DEN INHALT VOM EA-REGISTER
UM EINE STELLE NACH RECHTS

MNEM. CODE

SR EA

OP. CODE

0C

OPERATION:

E7 E6 E5 E4 E3 E2 E1 E0

A7 A6 A5 A4 A3 A2 A1 A0

Extension-Register

Akkumulator

2 Befehle: Schieben nach links

BESCHREIBUNG

ISCHIEBE DEN INHALT VOM AKKUMULATOR
UM EINE STELLE NACH LINKS

MNEM. CODE

SL A

OP. CODE

0E

OPERATION:

A7 A6 A5 A4 A3 A2 A1 A0

0

Akkumulator

BESCHREIBUNG

ISCHIEBE DEN INHALT VOM EA-REGISTER
UM EINE STELLE NACH LINKS

MNEM. CODE

SL EA

OP. CODE

0F

OPERATION:

E7 E6 E5 E4 E3 E2 E1 E0

A7 A6 A5 A4 A3 A2 A1 A0

Extension-Register

Akkumulator

2 Befehle: Rotieren nach rechts

BESCHREIBUNG

IROTIERE DEN INHALT VOM AKKUMULATOR
UM EINE STELLE NACH RECHTS

MNEM. CODE

RR A

OP. CODE

3E

OPERATION:

A7 A6 A5 A4 A3 A2 A1 A0

Akkumulator

BESCHREIBUNG

IROTIERE DIE INHALTE VON CY/L-REGISTER UND
AKKUMULATOR UM EINE STELLE NACH RECHTS

MNEM. CODE

RRL A

OP. CODE

3F

OPERATION:

A7 A6 A5 A4 A3 A2 A1 A0

CY/L

Akkumulator

Einige Bemerkungen dazu:

- 1. Diese Befehle beziehen sich grundsätzlich auf den Akkuinhalt. Die beiden Befehle SR EA und SL EA werden in dem um das Extension-Register erweiterten Akkumulator (engl.: extended accumulator), dem EA-Register, durchgeführt.
- 2. Bei den drei Befehlen SR A, SRL A und SR EA geht das niederwertigste Bit A₀ des Akkuinhaltes verloren.
- 3. Bei den beiden Befehlen SL A und SL EA geht das höchstwertigste Bit verloren: A₇ bzw. E₇ (= EA₁₅).
- 4. Das Verschieben nach rechts bzw. nach links um eine Stelle kann als Dividieren durch 2, bzw. als Multiplizieren mit 2 interpretiert werden. Wir kommen darauf im nächsten Abschnitt und im 16. Kapitel zurück.
- 5. Soll ein bestimmtes Bit eines 8-Bit-Wortes innerhalb des Wortes verschoben werden, sind auch hierfür die Schiebepfehle geeignet: vgl. Abschnitt 15.4.
Es ist jetzt kein Problem, das letzte Programm aus Abschnitt 15.1 so zu verändern, daß das Licht in der LED-Reihe nach links läuft. Wir können statt des 8. Befehls, statt SR A, den Befehl SL A verwenden. Außerdem muß im 2. Befehl dann die Hexadezimalzahl 01₁₆ geladen werden. Gegenüber dem letzten Programm im Abschnitt 15.1 haben wir also folgende Änderungen vorzunehmen:

INR.	IMARKE	IMNEM. CODE	ADR.	OP. CODE
2.		ILD A, ₇ =01	1001	C4 01
8.		SL A	100C	0E

15.3 Spiel 9: Zweier-Potenzen

Wir erinnern uns: Beim Spiel 9 (vgl. Abschnitt 1.3.9) werden nacheinander die Zweier-Potenzen von $2^0 = 1$ bis $2^{13} = 8192$ angezeigt. Dazu muß die Potenz ständig verdoppelt oder halbiert werden. Das geschieht mit den beiden Schiebepfeilen SL EA und SR EA. Das Programm, das wir mit den Tasten (**RS**), **SP**, **9** aufrufen können, steht im ROM von Adresse 052E bis 0573. Wir können es wie folgt ab Adresse 1000 eintasten. Dann muß lediglich die Ansprungsadresse für das benutzte Unterprogramm verändert werden.

NR.	MARKE	MNEM. CODE	ADR.	OP. CODE	BEMERKUNGEN
1.	BEGINN	ILD EA,=0	1000	184 00 00	
2.		IST EA,POT HEX	1003	18D E4	
3.		IST EA,EXP HEX	1005	18D E6	EXP HEX:=00
4.		ILD A,POT HEX	1007	195 E4	POT HEX:=01
5.	AUFW.	JSR UEBANZ	1009	120 24 10	SPRUNG ZUM UNTERPROGRAMM
6.		ILD EA,POT HEX	100C	185 E4	
7.		SL EA	100E	10F	POT HEX := POT HEX * 2
8.		IST EA,POT HEX	100F	18D E4	
9.		ILD A,EXP HEX	1011	195 E6	EXP HEX := EXP HEX + 1
10.		XOR A,=0D	1013	1E4 0D	IST EXP HEX = 0D?
11.		BNZ AUFW.	1015	17C F2	
12.	ABW.	JSR UEBANZ	1017	120 24 10	SPRUNG ZUM UNTERPROGRAMM
13.		ILD EA,POT HEX	101A	185 E4	
14.		SR EA	101C	10C	POT HEX := POT HEX / 2
15.		IST EA,POT HEX	101D	18D E4	
16.		ILD A,EXP HEX	101F	19D E6	EXP HEX := EXP HEX - 1
17.		BNZ ABW.	1021	17C F4	IST EXP HEX = 0?
18.		BRA AUFW.	1023	174 E4	
19.	UEBANZ	ILD EA,POT HEX	1025	185 E4	UNTERPROGRAMM "UEBANZ" UEBERSETZEN UND ANZEIGEN
20.		ICALL HEX-DEZ	1027	11C	
21.		IST EA,ZWSP	1028	18D D8	
22.		ICALL UEB 4R	102A	113	ANZEIGE DER POTENZ RECHTS
23.		ILD EA,EXP HEX	102B	185 E6	
24.		ICALL HEX-DEZ	102D	11C	
25.		IXCH A,E	102E	101	
26.		IST EA,ZWSP	102F	18D D8	
27.		ICALL UEB 4L	1031	114	ANZEIGE DES EXPONENTEN LINKS
28.		ILD EA,=0	1032	184 00 00	
29.		IST EA,CST4	1035	18D C4	IST <CST4>:=00; <CST5>:=00
30.		ILD A,CST7	1037	1C5 C7	
31.		XOR A,=07E	1039	1E4 7E	IST <CST7>:=7E? ANZEIGE: "0"
32.		BNZ WEITER	103B	17C 04	
33.		ILD A,=0	103D	1C4 00	
34.		IST A,CST7	103F	1CD C7	EVENTL. <CST7>:=00
35.	WEITER	ILD A,=0	1041	1C4 00	
36.		ICALL ANZEIGE	1043	111	
37.		ICALL ANZEIGE	1044	111	
38.		IRET	1045	15C	RUECKSPRUNG

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
POT HEX	1FFE4	POTENZ, HEXADEZIMAL
EXP HEX	1FFE5	EXPONENT, HEXADEZIMAL
ZWSP	1FFD8	ZWISCHENSPEICHER
CST4	1FFC4	STELLE 4 WIRD GELOESCHT
CST5	1FFC5	STELLE 5 WIRD GELOESCHT
CST7	1FFC7	STELLE 7 WIRD GELOESCHT. FALLS <CST7>:=7E

In den ersten vier Befehlen wird die Potenz gleich 0001, der Exponent gleich 00 gesetzt. Der Exponent steht vierstellig in den Speichern mit den Adressen 1FFE6 und 1FFE7. Er wird zwar im Laufe des Programms maximal D (= 13₁₀), aber für den CALL HEX DEZ und den CALL UEB 4L sind die führenden Nullen erforderlich.

In den Befehlen 5. bis 11. wird nach jedem Aufruf des Unterprogramms die Potenz durch den SL EA-Befehl verdoppelt, der Exponent durch den ILD-Befehl erhöht. Potenz und Exponent sind hier zunächst noch Hexadezimalzahlen. Das Verdoppeln mit Hilfe des SL EA-Befehls ist besonders dann verständlich, wenn man die Potenz als Dualzahl auffaßt. Sie hat jeweils nur an einer Stelle eine 1 und sonst Nullen. Zum Beispiel wird aus 20₁₆ (= 32₁₀ = 10 0000₂) durch Linkschieben 40₁₆ (= 64₁₀ = 100 0000₂). Die größte Zweier-Potenz, die hier vorkommt, ist 2¹³ = 2000₁₆ (= 8192₁₀ = 10 0000 0000 0000₂).

Diese Begrenzung ist durch den CALL HEX DEZ festgelegt. Die Dezimalzahl muß kleiner oder gleich 9999 sein.

In den Befehlen 12. bis 18. wird entsprechend die Potenz durch den SR EA-Befehl halbiert, der Exponent durch den DLD-Befehl erniedrigt.

Noch einige Bemerkungen zum Unterprogramm: Die Potenz wird in eine Dezimalzahl übersetzt und zur Anzeige an den vier rechten Stellen zum Zwischenspeicher FFD9/FFD8 gebracht. Der Exponent wird auch in eine Dezimalzahl umgewandelt. Damit er in den beiden linken Sieben-Segment-Anzeigen dargestellt wird, folgt der Befehl XCH A, E, bevor mit dem CALL UEB 4L die Anzeige vorbereitet wird. Vor dem Aufruf von CALL ANZEIGE werden schließlich noch die nicht gewünschten Stellen gelöscht.

15.4 (F3): = (CY/L)

Im Programm von Abschnitt 7.4 (vgl. auch Abschnitt 7.6, Abschnitt 8.2, Abschnitt 8.3 und Abschnitt 9.3) haben wir den Inhalt des CY/L-Flags an der roten Leuchtdiode F3 (Flag 3) angezeigt. Dazu mußte im wesentlichen der Inhalt des Status-Registers viermal nach rechts verschoben werden.

Wir geben hier noch einmal das vollständige Programm von Abschnitt 7.4 an.

NR.	MARKE	MNEM. CODE	ADR.	OP. CODE	BEMERKUNGEN
1.	BEGINN	AND S,=0	1000	139 00	<S>:=00, DAMIT: <F3>:=0
2.		ILD A,=0A3	1002	1C4 A3	<A>:=0A3
3.		ICALL ANZ A	1004	118	
4.		ILD A,=09E	1005	1C4 9E	<A>:=09E
5.		ICALL ANZ A	1007	118	
6.		IADD A,=0A3	1008	1F4 A3	<A>:=<A>+0A3
7.		IST A,SUMME	100A	1CD E3	
8.		ILD A,S	100C	106	<A>:=<S>
9.		AND A,=080	100D	1D4 80	
10.		SR A	100F	13C	
11.		SR A	1010	13C	
12.		SR A	1011	13C	
13.		SR A	1012	13C	
14.		ILD S,A	1013	107	<F3>:=<CY/L>
15.		ILD A,SUMME	1014	1C5 E3	<A>:=SUMME
16.		ICALL ANZ A	1016	118	
17.		ICALL TEST	1017	11E	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
SUMME	1FFE3	

In diesem Programm werden die Hexadezimalzahlen A3 und 9E addiert. Den Übertrag zeigt F3. In Abschnitt 7.4 hatten wir den ersten Befehl und die Befehle 7. bis 15. noch nicht besprochen. Im 1. Befehl wird das Status-Register gelöscht. Der im 6. Befehl berechnete Akkuinhalt wird im 7. Befehl zum Speicher FFE3 gebracht, damit er später für die Anzeige im 16. Befehl wieder verfügbar ist. Im 8. bis 14. Befehl wird der Inhalt des Statusregisters in den Akku geladen, Bit 7 (CY/L) wird maskiert und viermal nach rechts verschoben. Wird jetzt der Akkuinhalt wieder in das Statusregister geladen, dann hat Flag 3 den Wert des CY/L-Registers übernommen.

15.5 Der Zufallszahlengenerator

Bei Spiel 8: „Zahlenlotto 6 aus 49“ (vgl. Abschnitt 1.3.8) und bei Spiel 11: „Reaktionstest“ (vgl. Abschnitt 1.3.11) wird ein Zufallszahlengenerator benutzt. Was ist ein Zufallszahlengenerator? Er soll bei jedem Aufruf aus einem vorgegebenen Zahlenbereich eine zufällige Zahl ermitteln. Wir können uns etwa die Ziehungsgeräte für das Lotto vorstellen, die aus dem Fernsehen bekannt sind. Die Wahrscheinlichkeit ist für jede der möglichen Zahlen gleich groß. Ist schon eine Zahl ausgewählt, so hat das keinen Einfluß auf die nächsten Zahlenauswahl. Beim Lotto darf natürlich eine bereits ermittelte Zahl nicht noch einmal ausgewählt werden.

Unser Zufallszahlengenerator ist eigentlich kein echter Zufallszahlengenerator. Er ermittelt nach einem festen Unterprogramm im ROM (Adresse 06CF bis 06EA), also nach einem ganz bestimmten Verfahren, zu dem Inhalt von Speicher FFE0 eine andere Hexadezimalzahl (von 01 bis FF) und schreibt sie in diesen Speicher. Zufällig ist vielleicht, welche Zahl vor Aufruf des Unterprogramms im Speicher FFE0 steht. Welche Zahl im Unterprogramm dann ermittelt wird, erscheint höchstens als Zufall, solange man das Verfahren nicht untersucht, nach dem die neue Zahl bestimmt wird. Wir wollen uns zunächst einige dieser „Zufallszahlen“ ansehen. Um einen definierten Ausgangspunkt festzulegen, bringen wir in dem folgenden kleinen Programm zunächst die Zahl 00 in den Speicher mit der Adresse FFE0. Dann folgt ein Sprung zum angegebenen Unterprogramm im ROM (ab Adresse 06CF). Nach der Anzeige der ermittelten Zahl kann durch Betätigung der Taste **RUN** die folgende Zahl abgerufen werden.

INR.	MARKE	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	LD A,=0	1000	1C4 00	
2.		IST A,ZAHL	1002	1CD E0	ZAHL:=00
3.	WEITER	JSR ZUFALL	1004	120 CE 06	NÄCHSTE ZUFALLSZAHL
4.		CALL ANZ A	1007	118	
5.		BRA WEITER	1008	174 FA	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZAHL	FFE0	

WEITERE MARKE, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZUFALL	06CF	UNTERPROGRAMM: "ZUFALLSZAHL-GENERATOR"

Dieses Programm liefert uns nacheinander die Hexadezimalzahlen 7F, 3F, 1F, 8F, C7, 63, B1, 58, AC usw. Wenn wir diese Reihe fortsetzen, stellen wir fest, daß alle 255 Zahlen von 01 bis FF in ihr vorkommen. Mit dem benutzten Verfahren werden die Zahlen eigentlich nur in bestimmter Weise sortiert.

Wir wollen uns jetzt noch ansehen, was in dem benutzten Unterprogramm geschieht. In dem folgenden Programmausdruck ist das Unterprogramm ab Adresse 1200 angegeben. Wenn man es verändern will, oder wenn man zur Untersuchung an bestimmten Stellen einen Breakpoint einfügen will, muß das Programm im RAM-Bereich stehen.

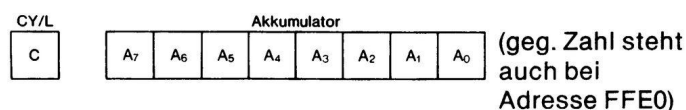
In den ersten drei Befehlen (50. bis 52. Befehle) wird der Akku mit dem Inhalt des Speichers FFE0 geladen. In dem Fall, daß diese Zahl 00 ist, wird sie dekrementiert. Das folgende Verfahren würde die Zahl 00 nicht verändern.

INR.	MARKE	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	LD A,=0	1000	1C4 00	
2.		IST A,ZAHL	1002	1CD E0	ZAHL:=00
3.	WEITER	JSR ZUFALL	1004	120 FF 11	NÄCHSTE ZUFALLSZAHL
4.		CALL ANZ A	1007	118	
5.		BRA WEITER	1008	174 FA	

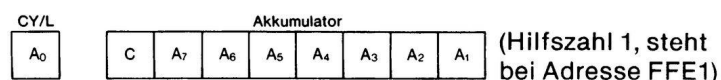
50.	ZUFALL	LD A,ZAHL	1200	1C5 E0	UNTERPROGRAMM: "ZUFALLSZAHL-GENERATOR"
51.		BNZ NEU	1202	17C 02	IFALLS ZAHL=00, DANN ZAHL:=FF
52.		LD A,ZAHL	1204	19D E0	
53.	NEU	RRL A	1206	13F	NEUE ZUFALLSZAHL
54.		IST A,HZ1	1207	1CD E1	
55.		RRL A	1209	13F	
56.		RRL A	120A	13F	
57.		IST A,HZ2	120B	1CD E2	
58.		RRL A	120D	13F	
59.		RRL A	120E	13F	ERKLÄRUNG IM TEXT
60.		XOR A,HZ2	120F	1E5 E2	
61.		XOR A,HZ1	1211	1E5 E1	
62.		XOR A,ZAHL	1213	1E5 E0	
63.		RRL A	1215	13F	
64.		LD A,ZAHL	1216	1C5 E0	
65.		RRL A	1218	13F	
66.		IST A,ZAHL	1219	1CD E0	
67.		RET	121B	15C	RUECKSPRUNG

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZAHL	FFE0	ZUFALLSZAHL
HZ1	FFE1	HILFSZAHL 1
HZ2	FFE2	HILFSZAHL 2

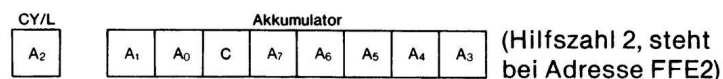
Was passiert jetzt mit einer Zahl ungleich Null? Um das Unterprogramm zu untersuchen, müssen wir die einzelnen Dualziffern bezeichnen:



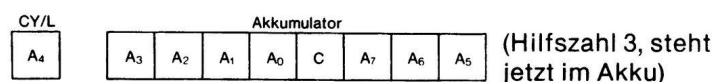
Mit C ist der Inhalt des CY/L-Registers, mit A₇ bis A₀ sind die Dualziffern der gegebenen Zahl bezeichnet. Durch den 53. Befehl (RRL A) entsteht im Akku:



Jetzt folgt zweimal der RRL A-Befehl:

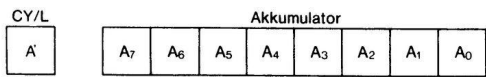


Nach zwei weiteren RRL-Befehlen (58. und 59.) entsteht:

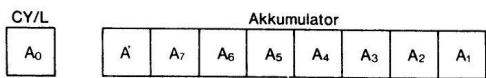


Jetzt folgen drei XOR-Befehle, dann wieder ein RRL-Befehl. Wenn dann anschließend der Akku neu geladen wird (64. Befehl: LD A, ZAHL), geht vom Ergebnis des XOR-Verknüpfens alles verloren, was beim Laden im Akku steht. Benötigt wird also nur das Verknüpfungsergebnis an der nieder-

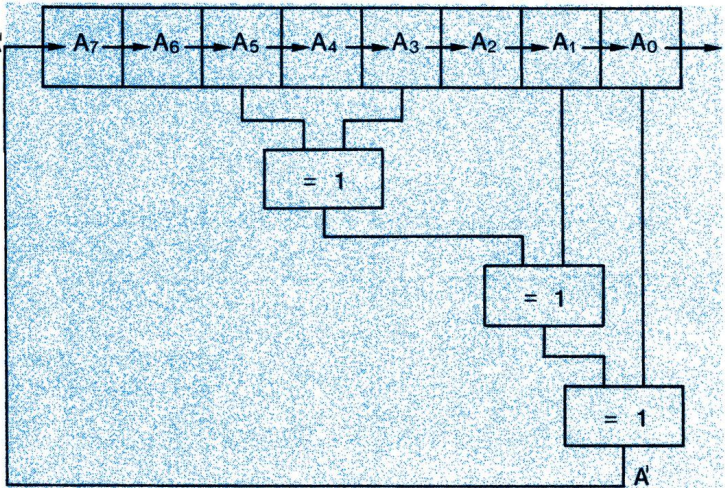
wertigsten Stelle im Akku. Nach den drei XOR-Befehlen steht ganz rechts im Akku eine Dualziffer A'. Es ist $A' = (((A_5 \oplus A_3) \oplus A_1) \oplus A_0)$. Diese Ziffer gelangt im 63. Befehl ins CY/L-Register. Jetzt wird der Akku neu mit dem Inhalt von Speicher FFE0 geladen.



Jetzt folgt der letzte RRL-Befehl.



Im Akku steht jetzt die gesuchte Zufallszahl. Diese Zahl wird zusätzlich in den Speicher mit der Adresse FFE0 geschrieben.
 Was ist im ganzen geschehen? Die ursprüngliche Zahl ist um eine Stelle nach rechts verschoben worden. A₀ ist verloren. Links ist eine neue Ziffer aus vier ursprünglichen Ziffern gebildet worden.
 Folgende Schaltung – in Verbindung mit einem Rechtschieben – würde das gleiche wie unser Software-Programm erreichen:



A' bekommt immer dann den Wert 1, wenn die Anzahl der Einsen bei den vier Dualziffern A₅, A₃, A₁ und A₀ ungerade ist.

Beispiele:

- Die gegebene Zahl sei: $63_{16} = 0110\ 0011_2$.
 Es ergibt sich: $A' = 1$
 Rechtsverschieben und Ergänzen von $A' = 1$ ergibt die neue Zahl:
 $1011\ 0001_2 = B1_{16}$.
- Die gegebene Zahl sei: $B1_{16} = 1011\ 0001_2$
 Rechtsverschieben und Ergänzen von $A' = 0$ ergibt die neue Zahl: $0101\ 1000_2 = 58_{16}$.
- Die gegebene Zahl sei: $58_{16} = 0101\ 1000_2$
 Rechtsverschieben und Ergänzen von $A' = 1$ ergibt die neue Zahl: $1010\ 1100_2 = AC_{16}$.

Erwähnt werden soll noch, daß die Auswahl der sechs Zahlen beim Spiel 8: „Zahlenlotto 6 aus 49“ nicht nur von unserem Zufallszahlengenerator bestimmt wird. Die Auswahl hängt auch davon ab, wann nach Erscheinen der Anzeige „6 AUS 49“ die Taste **SA** betätigt wird.
 Zum Spiel 11: „Reaktionstest“ soll auch noch eine Bemerkung gemacht werden. Hier werden die Zahlen, die der Zufallszahlengenerator ermittelt, im Wechsel für zwei verschiedene Aufgaben verwendet. Mit der ersten (und der dritten usw.) Zufallszahl wird festgelegt, wielange die Anzahl der „Richtigen“ (vgl. Abschnitt 1.3.11) angezeigt wird. Die zweite Hexadezimalziffer der jeweils nächsten Zufallszahl erscheint dann achtfach. Wenn wir noch wissen, welche Zufallszahl bei diesem Spiel als erste gewählt wird, können wir die Folge der angezeigten Ziffern voraussagen. Beim Aufruf dieses Spiels mit **SP**, **B** wird zunächst der momentane Akkuinhalt in den Speicher FFE0 geladen. Zu diesem Speicherinhalt wird dann die nächste Zufallszahl ermittelt. Mit den Tasten **CPU**, **6**, **0**, **CPU** können wir 00 in den Akku laden. Wenn wir jetzt mit **SP**, **SP**, **B** das Spiel 11 starten, werden die Zufallszahlen in der Reihenfolge – mit 7F beginnend – ermittelt, wie wir es beim Programm in Abschnitt 15.5 gesehen haben: 7F, 3F, 1F, 8F, C7, 63, B1, 58, AC usw. . . Die angezeigten Hexadezimalziffern sind dann: F, F, 3, 8, 6, 5 usw.

15.6 Aufgaben zu Kapitel 15

1. In dem folgenden Programm ist ein Lauflicht programmiert, das im Wechsel nach links und nach rechts läuft. Hier ist nur der mnemonische Code angegeben. Die Aufgabe besteht darin, die Befehle in den Operationscode zu übersetzen, dann das Programm einzutasten und zu testen. Der Schalter S muß dabei in der vorderen Stellung stehen. Zusätzlich soll ein Programmablaufplan erstellt werden.

NR.	MARKE	MMEM. CODE
1.	BEGINN	ICALL BLANK
2.	NEU	ILD A,=01
3.		IST A,ZAHL
4.	LINKS	IST A,LED-R
5.		ILD A,=040
6.		ICALL ANZEIGE
7.		ILD A,ZAHL
8.		ISL A
9.		IST A,ZAHL
10.		IBNZ LINKS
11.		ILD A,=040
12.		IST A,ZAHL
13.	RECHTS	ILD A,ZAHL
14.		IST A,LED-R
15.		ILD A,=040
16.		ICALL ANZEIGE
17.		ILD A,ZAHL
18.		ISR A
19.		IST A,ZAHL
20.		IAND A,=01
21.		IBZ RECHTS
22.		IBRA NEU

2. In dieser Aufgabe soll der RR-Befehl im Programm von Abschnitt 15.1 durch den RRL-Befehl ersetzt werden.

2.1 Was passiert, wenn man lediglich den Inhalt 3E im Speicher 100C durch 3F überschreibt? Wie läßt sich diese Erscheinung erklären? – Zusatzfrage als Hilfestellung: Welchen Inhalt hat das CY/L-Register nach dem CALL ANZEIGE (6. Befehl)? Wer es testen möchte, kann in die beiden Speicher 100A und 100B vorübergehend die Operationscode 06 und 18 schreiben.

2.2 Das Programm soll jetzt zusätzlich folgendermaßen verändert werden:
Vor dem 5. Befehl werden die beiden Befehle LDA, S und ST A, HSP (HSP-Hilfsspeicher), nach dem 6. Befehl werden die beiden Befehle LD A, HSP und LD S, A eingefügt. Durch diese Maßnahme wird jetzt das Programm nach dem CALL ANZEIGE mit dem ursprünglichen Inhalt des Status-Registers fortgesetzt. Der Inhalt des Status-Registers (und damit des CY/L-Registers) wird vor dem CALL ANZEIGE „gerettet“, nach dem CALL wiederhergestellt. Wie sieht jetzt das Programm aus? Das Displacement beim Sprungbefehl muß verändert werden.

2.3 Wir sind unserem Ziel schon näher gekommen. Wenn wir die Änderungen von 2.2 durchgeführt haben, ist aber ein doppeltes Lauflicht entstanden. Es leuchten immer zwei benachbarte Dioden in der LED-Reihe. Woran liegt das? Wie ließe sich erreichen, daß immer nur eine Diode leuchtet?

3. Wir haben sieben Schiebe- und Rotationsbefehle kennengelernt. Es sei (CY/L) = 1; (E) = 30; (A) = B8.

3.1 Es soll überlegt werden, welche Inhalte diese drei Register nach jeweils einem der sieben Befehle haben.

(CY/L)	(E)	(A)	Befehl	(CY/L)	(E)	(A)
1	30	B8	SR A			
			SRL A			
			SR EA			
			SL A			
			SL EA			
			RR A			
			RRL A			

3.2 Wie könnten kleine Testprogramme aussehen, mit denen man experimentell die Änderung dieser Registerinhalte überprüfen könnte?

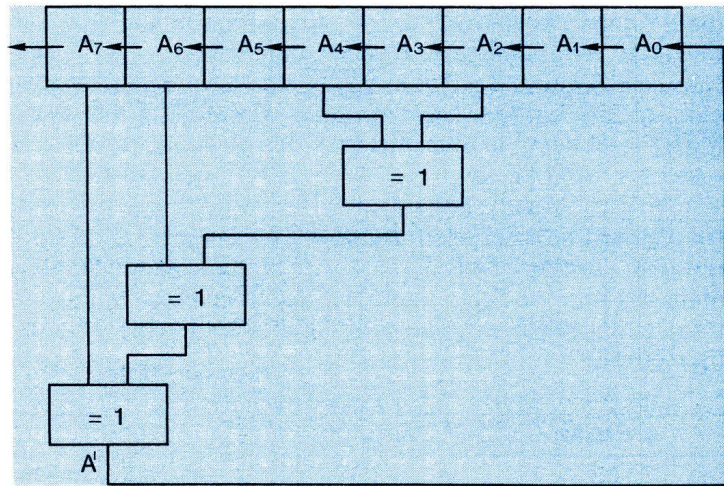
4. Im Abschnitt 6.1 hatten wir einen Halbaddierer programmiert. Dieses Programm soll hier näher untersucht werden.

4.1 Wie lauten die vollständigen Operationscodes der Befehle?

4.2 Wie lauten die Befehle im mnemonischen Code?

4.3 Wie sieht der zugehörige Programmablaufplan aus?

5. für unseren Zufallszahlengenerator hatten wir uns die Ersatzschaltung in Abschnitt 15.5 überlegt. Entsprechend hätte man den Zufallszahlengenerator nach folgender Schaltung programmieren können:



5.1 Wie würden die drei ersten Zahlen heißen, die dieser Zufallszahlengenerator ermitteln würde, wenn er mit der Hexadezimalzahl FF begänne?

5.2 Es soll ein Programm geschrieben werden, das nach diesem „Zufallszahlengenerator“ Zufallszahlen ermittelt (vgl. Programme des Abschnitts 15.5).

16. Multiplikation und Division

16.1 Ein erstes Multiplikationsprogramm

Es ist bekannt, daß die Multiplikation auf die Addition zurückgeführt werden kann:

$$3 \cdot 4 = \underbrace{4 + 4 + 4}_{3\text{mal}} = \underbrace{3 + 3 + 3 + 3}_{4\text{mal}}$$

Diese Überlegung ist die Grundlage für unser erstes Multiplikationsprogramm:

NR.	MARKE	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	ILD EA,=0	1000	84 00 00	
2.		IST EA,PROD	1003	8D E0	PROD:=0000
3.		ICALL EIN 2ST	1005	16	IEINGABE: FAKT1
4.		ILD EA,ZWSP	1006	85 D8	(FFE2):=FAKT1
5.		IST EA,FAKT1	1008	8D E2	(FFE3):=00
6.		ICALL EIN 2ST	100A	16	IEINGABE: FAKT2
7.		ILD A,FAKT2	100B	C5 D8	
8.		IBZ ERGEBN	100D	6C 0A	WENN FAKT2=0
9.	ADDIT.	ILD EA,PROD	100F	85 E0	
10.		ADD EA,FAKT1	1011	85 E2	PROD:=PROD+FAKT1
11.		IST EA,PROD	1013	8D E0	
12.		IDLD A,FAKT2	1015	9D D8	FAKT2:=FAKT2-1
13.		IBNZ ADDIT.	1017	7C F6	
14.	ERGEBN	ILD EA,PROD	1019	85 E0	
15.		ICALL ANZ EA	101B	19	ANZEIGE: PROD
16.		IBRA BEGINN	101C	74 E2	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
PROD	FFE0	PRODUKT
FAKT1	FFE1	
	FFE2	1. FAKTOR
	FFE3	
FAKT2	FFD8	2. FAKTOR
ZWSP	FFD8	ZWISCHENSPEICHER

Wenn wir das Programm eingegeben und gestartet haben, meldet sich der Computer mit der Anzeige „0“. Er wartet auf die Eingabe des ersten Faktors. Als Faktoren sind hier alle maximal zweistelligen Hexadezimalzahlen zugelassen. Nach Betätigung der Taste **RUN** kann der zweite Faktor eingegeben werden. Nach erneuter Betätigung der Taste **RUN** erscheint in der Anzeige das Produkt (PROD := FAKT1 · FAKT2).

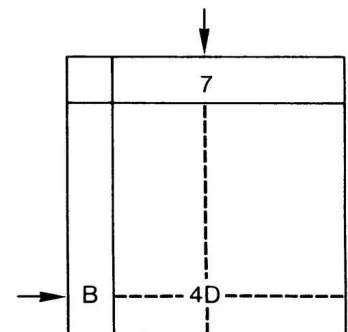
Der Computer rechnet im Dualsystem. Eingabe der Faktoren und Anzeige des Produkts erfolgen im Hexadezimalsystem. Da uns Produkte in diesem Zahlensystem wenig vertraut sind, sollen einige Multiplikationsbeispiele angegeben werden:

1. Faktor	2. Faktor	Produkt
03	04	000C
04	04	0010
73	10	0730
83	B7	5DA5
A3	CE	832A
FF	FF	FE01

Außerdem geben wir hier die Multiplikationstabelle für einstellige Hexadezimalzahlen an:

#	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	0	2	4	6	8	A	C	E	10	12	14	16	18	1A	1C	1E
3	0	3	6	9	C	F	12	15	18	1B	1E	21	24	27	2A	2D
4	0	4	8	C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	0	5	A	F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	0	6	C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	0	7	E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	0	8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	0	9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	0	A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	0	B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	0	C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	0	D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	0	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	0	F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

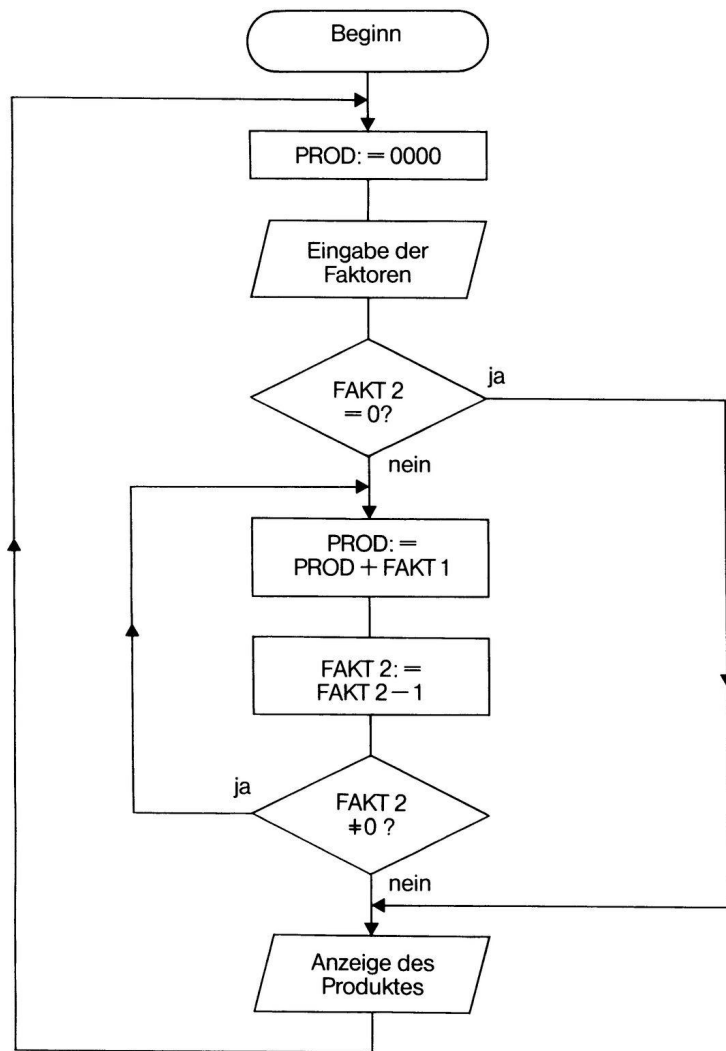
Das Produkt 4D der beiden Faktoren B und 7 läßt sich in der Zeile hinter B und in der Spalte unter 7 ablesen.



Noch einige Bemerkungen zum Programm: In den ersten beiden Befehlen wird das Produkt zunächst gleich Null gesetzt. Dazu werden die beiden Speicher mit den Adressen FFE0 und FFE1 gelöscht. Das Produkt, das bei Programmende in diesem Speicher steht, kann maximal vierstellig werden (FF · FF = FE01).

In den Befehlen 3 bis 6 werden die beiden Faktoren eingegeben. Der erste (maximal zweistellige) Faktor wird mit führenden Nullen vierstellig in die Speicher FFE2 und FFE3 gebracht. Das ist erforderlich, weil die Addition im Kernteil des Programms vierstellig im EA-Register erfolgt. Der zweite Faktor bleibt nach der Eingabe im Speicher FFD8 stehen. In den Befehlen 7 und 8 wird untersucht, ob der zweite Faktor null ist. In diesem Fall erfolgt sofort ein Sprung zur Ergebnisanzeige.

In den Befehlen 9 bis 13 wird die mehrfache Addition ausgeführt. Der erste Faktor wird so oft addiert, wie es der zweite Faktor angibt.


$$\begin{array}{r} \text{A 3} \cdot \text{C E} \\ \hline \text{8 E A} \leftarrow \begin{array}{|c|} \hline \text{ } \\ \hline \end{array} \\ \text{7 A 4} \leftarrow \begin{array}{|c|} \hline \text{ } \\ \hline \end{array} \\ \text{1 1} \\ \hline \text{8 3 2 A} \\ \hline \hline \end{array}$$

Bei diesem Beispiel werden die folgenden Produkte aus dem hexadezimalen kleinen Einmal-Eins benötigt (vgl. Multiplikationstabelle für einstellige Hexadezimalzahlen):

$$\begin{aligned}(3 \cdot E)_{16} &= (3 \cdot 14)_{10} = 42_{10} = 2A_{16} \\ (A \cdot E)_{16} &= (10 \cdot 14)_{10} = 140_{10} = 8C_{16} \\ (3 \cdot C)_{16} &= (3 \cdot 12)_{10} = 36_{10} = 24_{16} \\ (A \cdot C)_{16} &= (10 \cdot 12)_{10} = 120_{10} = 78_{16}\end{aligned}$$

Im Dualsystem ist die entsprechende Rechnung sehr einfach. Es wird nur mit 0 oder 1 multipliziert: $\text{Zahl} \cdot 0 = 0$; $\text{Zahl} \cdot 1 = \text{Zahl}$. Die beiden Zahlen $1010\ 0011_2 (= A3_{16})$ und $1100\ 1110_2 (= CE_{16})$ können im Dualsystem folgendermaßen multipliziert werden:

The diagram illustrates the iterative step of the Longest Common Subsequence (LCS) algorithm. It shows the alignment of two strings, 10100011 and 10100110 , with a grid of arrows indicating the backtracking path from the end of the LCS to the start. The diagram highlights the step where the last character of the LCS is determined by comparing the last characters of the two strings and their predecessors.

Das Ergebnis ist also $1000\ 0011\ 0010\ 1010_2 = 832A_{16}$. Anmerkung: Bei der gleichzeitigen Addition von mehr als zwei Dualzahlen kann sich ein größerer Übertrag als 1 ergeben. Bei der Addition in der achten Stelle von rechts ergibt sich 100_2 . Die letzte Null wird hingeschrieben. Der Übertrag ist 10.

Der erste Faktor wird mehrfach um eine Stelle nach links geschoben. Es wird addiert, wenn die entsprechende Ziffer vom zweiten Faktor 1 ist. Diese Rechnung ist Grundlage für unser zweites Multiplikationsprogramm, das wir in Abschnitt 16.4 untersuchen werden.

16.2 Schriftliches Multiplizieren

Wir würden sicher nicht nach dem Verfahren multiplizieren, wie es unserem ersten Multiplikationsprogramm zugrunde liegt. Wir würden das Produkt $163 \cdot 206$ nicht so berechnen, daß wir den Faktor 163 hinschreiben und ihn 205mal dazu addieren würden.

Wir würden folgendermaßen rechnen: Wir multiplizieren zunächst $163 \cdot 6 = 978$. Dieses Produkt schreiben wir auf. Jetzt multiplizieren wir $163 \cdot 0 = 0$ und schreiben das Ergebnis 0 eine Stelle nach links verschoben unter das Ergebnis 978.

Jetzt multiplizieren wir $163 \cdot 2 = 326$ und schreiben auch dieses Ergebnis wiederum eine Stelle nach links verschoben auf. (Eigentlich hätten wir ja nicht $163 \cdot 2$, sondern $163 \cdot 200$ rechnen müssen). Schließlich addieren wir die Teilergebnisse und erhalten als Produkt 33578.

Nach diesem Verfahren läßt sich auch in jedem anderen Zahlensystem rechnen. Man muß nur das entsprechende kleine Ein-mal-Eins kennen.

16.3 Verdoppeln – Halbieren

Beim Spiel 9: Zweier-Potenzen (vgl. Abschnitt 1.3.9 und Abschnitt 15.3) hatten wir gesehen, daß das Verdoppeln bzw. Halbieren der Zweierpotenzen mit den Befehlen für das Links- bzw. das Rechtsschieben erreicht werden kann. Dort wurden aber nur Dualzahlen verdoppelt und halbiert, die nur an einer einzigen Stelle eine 1 und sonst Nullen haben. Wir wollen uns überzeugen, daß diese Verfahren bei beliebigen Dualzahlen angewendet werden können.

16.4 Das zweite Multiplikationsprogramm

Wir erinnern uns an das, was wir beim schriftlichen Multiplizieren von Dualzahlen gesagt haben. Der erste Faktor wird mehrfach um eine Stelle nach links geschoben. Es wird addiert, wenn die entsprechende Stelle des zweiten Faktors 1 ist. Das Verschieben des ersten Faktors nach links ist das Verdoppeln aus dem letzten Abschnitt. Um beim zweiten Faktor festzustellen, ob an der entsprechenden Stelle eine 1 steht, werden wir ihn (jeweils nach der Abfrage, ob an der letzten Stelle eine 1 steht oder nicht) halbieren, bzw. um eine Dualstelle nach rechts verschieben.

NR.	MARKE	MMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	CALL EIN 2ST	1000	16	IEINGABE DER ZAHL
2.		ICALL BLANK	1001	12	
3.		ILD A,ZWSP	1002	C5 D8	
4.	WEITER	IST A,LED-R	1004	CD C3	IZAHL DUAL
5.		ICALL UEB 4L	1006	14	ZUR LED-REIHE
6.		ICALL ANZ EIN	1007	10	IZAHL HEXADEZIMAL
7.		INOP	1008	00	ZUR ANZEIGE
8.		INOP	1009	00	ANZEIGE
9.		ILD EA,ZWSP	100A	85 D8	
10.		ISL EA	100C	0F	IZAHL:=ZAHL * 2
11.		IST EA,ZWSP	100D	8D D8	
12.		IBRA WEITER	100F	74 F3	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZWSP	1FFD8	ZWISCHENSPEICHER
LED-R	1FFC3	LED-REIHE

Nach Programmstart geben wir z. B. die Hexadezimalzahl B_{16} ein. Betätigen wir dann die Taste **RUN**, dann erscheint links die Anzeige 000B. Außerdem zeigt die LED-Reihe diese Zahl in der dualen Schreibweise: 0000 1011. Wird jetzt wieder die **RUN**-Taste (oder eine andere Funktionstaste außer **A→D**) betätigt, so wird die Zahl verdoppelt. Die Anzeige ist jetzt 0016, die LED-Reihe zeigt: 0001 0110. Wir sehen: durch den Befehl SL EA wurde die Zahl verdoppelt, bzw. mit 2 multipliziert ($2 \cdot B_{16} = 2 \cdot 11_{10} = 22_{10} = 16_{16}$). An der LED-Reihe erkennen wir, daß dieser Verdopplung eine Verschiebung der Dualzahl um eine Stelle nach links entspricht.

Unser Programm ist so geschrieben, daß wir das Verschieben nach links mehrfach hintereinander ausführen können. Wir brauchen nur die Taste **RUN** mehrfach zu betätigen. Nach vier Verschiebungen ist die eingegebene Zahl um vier Dualstellen, um eine Hexadezimalstelle verschoben worden (0B – 16 – 2C – 58 – B0). Nach acht Verschiebungen erhalten wir die Anzeige 0B00. Die LED-Reihe ist jetzt dunkel, da hier nur die beiden letzten Hexadezimalziffern dual dargestellt werden.

Bei der dreizehnten Verschiebung erkennen wir, daß das höchstwertigste Bit des Extension-Registers beim SL EA-Befehl verloren geht.

Wir können unser Programm zum Verdoppeln leicht so ändern, daß es eingegebene Zahlen halbiert. Wir ändern die beiden folgenden Befehle.

NR.	MARKE	MMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	CALL EIN 4ST	1000	17	IEINGABE DER ZAHL
10.		SR EA	100C	0C	IZAHL:=ZAHL / 2

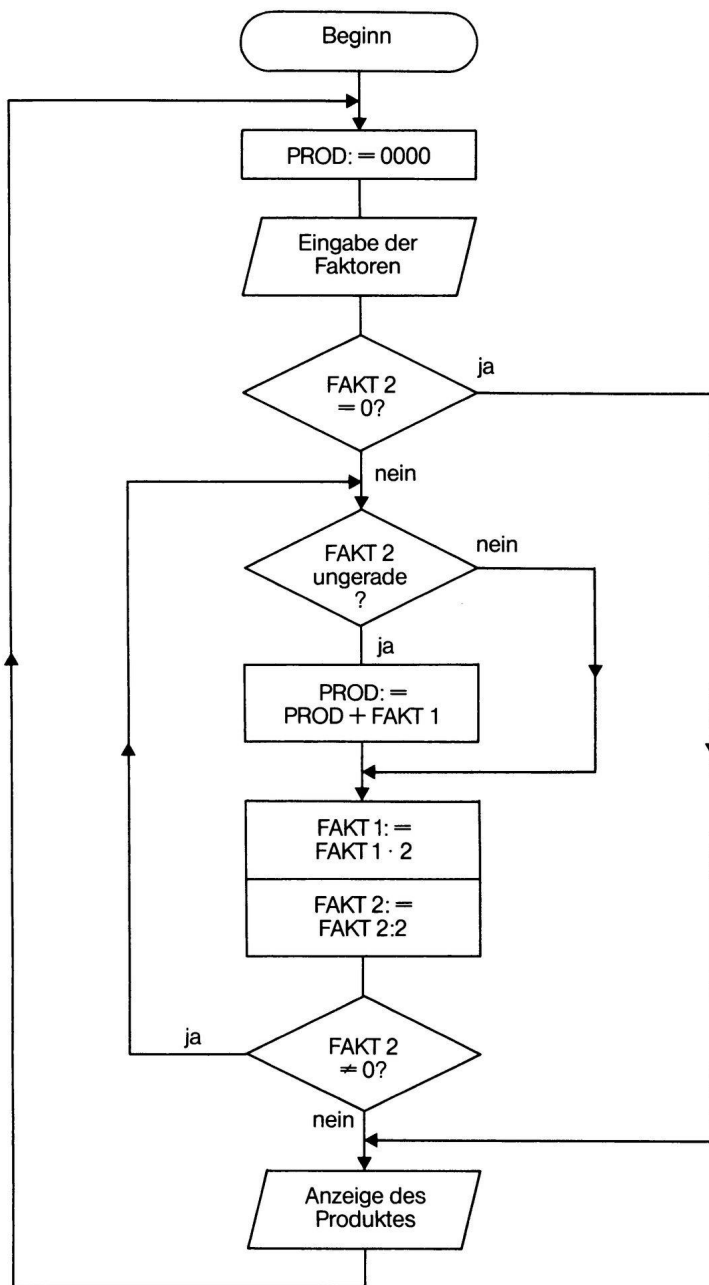
Wir starten das Programm, geben z. B. die Zahl C80 ein und betätigen eine Funktionstaste. Wir erhalten die Anzeige 0C80 und die Darstellung der letzten beiden Hexadezimalziffern an der LED-Reihe in der dualen Schreibweise: $1000\ 0000_2 = 80_{16}$. Jetzt wird bei jeder Betätigung der Funktionstaste die Zahl halbiert, bzw. um eine Dualstelle nach rechts geschoben. Bei der achten Verschiebung (SR EA) nach rechts geht die duale 1, die ganz rechts im Akku steht, verloren. Bei der Halbierung von ungeraden Zahlen wird bei diesem Verfahren grundsätzlich abgerundet: $19_{16} = 25_{10}$; $25_{10} : 2 = 12,5_{10}$; nach dem Verschieben erhalten wir $C_{16} = 12_{10}$.

NR.	MARKE	MMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	ILD EA,=0	1000	84 00 00	
2.		IST EA,PROD	1003	8D E0	IPROD:=0000
3.		ICALL EIN 2ST	1005	16	IEINGABE: FAKT1
4.		ILD EA,ZWSP	1006	85 D8	IFFE2:=FAKT1
5.		IST EA,FAKT1	1008	8D E2	IFFE3:=00
6.		ICALL EIN 2ST	100A	16	IEINGABE: FAKT2
7.		ILD A,FAKT2	100B	C5 D8	
8.		IBZ ERGEBN	100D	6C 16	IWENN FAKT2=0
9.	SCHL.	AND A,=01	100F	D4 01	IFAKT2 UNGERADE?
10.		IBZ SCHIEB	1011	6C 06	
11.		ILD EA,PROD	1013	85 E0	
12.		IADD EA,FAKT1	1015	85 E2	IPROD:=PROD+FAKT1
13.		IST EA,PROD	1017	8D E0	
14.	SCHIEB	ILD EA,FAKT1	1019	85 E2	
15.		ISL EA	101B	0F	IFAKT1:=FAKT1 * 2
16.		IST EA,FAKT1	101C	8D E2	
17.		ILD A,FAKT2	101E	C5 D8	
18.		ISR A	1020	3C	IFAKT2:=FAKT2 / 2
19.		IST A,FAKT2	1021	CD D8	
20.		IBNZ SCHL.	1023	7C EA	
21.	ERGEBN	ILD EA,PROD	1025	85 E0	
22.		ICALL ANZ EA	1027	19	IANZEIGE: PROD
23.		IBRA BEGINN	1028	74 D6	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
PROD	1FFE0	IPRODUKT
FAKT1	1FFE1	
FAKT2	1FFE2	1. FAKTOR
	1FFE3	
ZWSP	1FFD8	2. FAKTOR
	1FFD9	
	1FFD8	ZWISCHENSPEICHER

Die ersten acht und die letzten drei Befehle stimmen mit den entsprechenden aus Abschnitt 16.1 überein. Die Schleife SCHL. wird maximal achtmal durchlaufen. Es wird zunächst die letzte Stelle von (FFD8) untersucht (9. und 10. Befehl). Steht hier eine 1, so wird in den nächsten drei Befehlen (11. bis 13.) addiert. Andernfalls folgt gleich der Programmteil Schieben (SCHIEB). Der erste Faktor wird verdoppelt, der zweite wird halbiert.

Zwei Bemerkungen noch zu diesem Programm: Dieses Programm ist um sieben Befehle länger als unser erstes Multiplikationsprogramm, trotzdem hat es Vorteile. Es ist schneller. Wir werden es bei der Multiplikation von zweistelligen Hexadezimalzahlen noch nicht merken, aber bei der Multiplikation von vierstelligen Zahlen würde der Unterschied deutlich werden. Um $FFFF \cdot FFFF$ zu berechnen, müßten nach dem Additionsverfahren des 1. Programms 65 535, nach dem Schiebverfahren des 2. Programms 16 Schleifendurchläufe ausgeführt werden. Bei der Herstellung eines Taschenrechners spielen solche Überlegungen eine Rolle.



Das Verfahren, das unserem zweiten Multiplikationsprogramm zugrunde liegt, wird als altägyptische Multiplikationsmethode bezeichnet. Die alten Ägypter konnten gut addieren, verdoppeln und halbieren. Sie haben nach dem angegebenen Verfahren multipliziert. Dieses Verfahren gilt auch in jedem anderen Zahlensystem. Als Beispiel wollen wir noch einmal das Produkt $163 \cdot 206$ im Dezimalsystem berechnen.

1. Faktor	2. Faktor
463	206
326	103
652	51
1304	25
2608	12
5216	6
10432	3
20864	1
211	
33578	

Der erste Faktor wird jeweils verdoppelt, der zweite halbiert. Beim Halbieren wird gegebenenfalls abgerundet. Das Verfahren wird beendet, wenn der zweite Faktor 1 geworden ist. Jetzt wird jeweils der erste Faktor gestrichen, wenn der zweite Faktor gerade ist. Die nicht gestrichenen Zahlen in der ersten Spalte werden addiert. So ergibt sich das Produkt.

16.5 Das T-Register und der Multiplikationsbefehl

Unser Mikroprozessor kennt einen speziellen Multiplikationsbefehl. Mit ihm wird unser Multiplikationsprogramm noch viel schneller und auch deutlich kürzer.

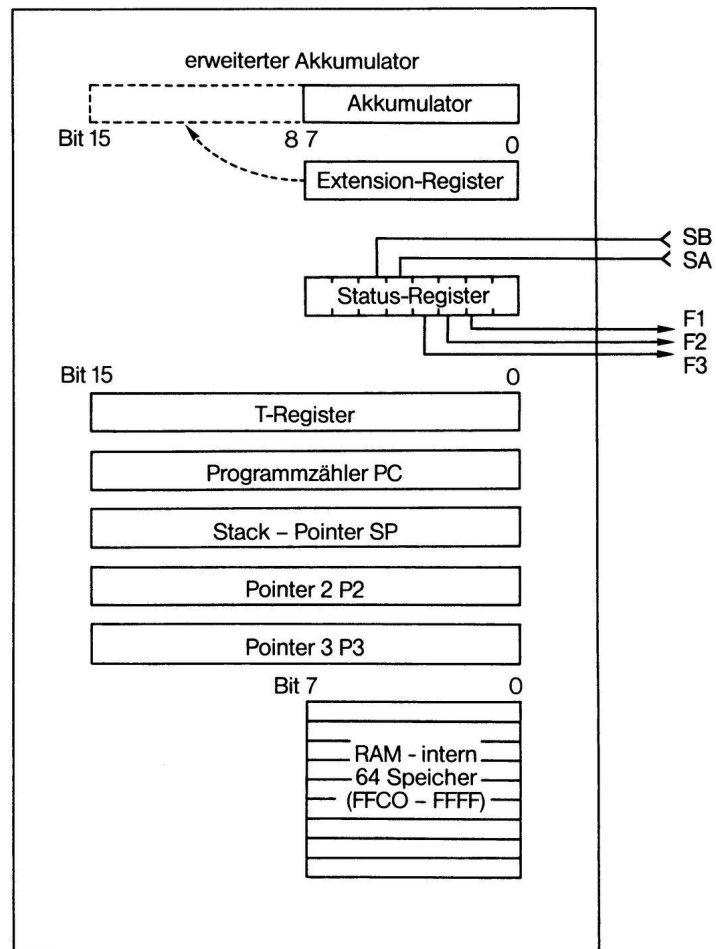
Zunächst das Programm:

INR.	MARKE	MMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	CALL EIN 2ST	1000	16	
2.		LD T,ZWSP	1001	185 D8	(T) := FAKT1
3.		CALL EIN 2ST	1003	16	
4.		LD EA,ZWSP	1004	185 D8	(EA) := FAKT2
5.		MPY EA,T	1006	12C	PROD := (T) * (EA)
6.		LD EA,T	1007	10B	
7.		CALL ANZ EA	1008	119	ANZEIGE: PROD
8.		BRA BEGINN	1009	174 F5	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZWSP	1FFD8 1FFD9	ZWISCHENSPEICHER

Dieses Programm liefert natürlich dieselben Produkte wie unsere anderen Multiplikationsprogramme. Wir können uns davon überzeugen. Um jedoch dieses Programm erklären zu können, müssen wir etwas ausholen. Hier begegnet uns zum erstenmal das T-Register (engl.: Temporary Register; Register für zeitweise, vorübergehende Speicherung). Man kann in diesem 16-Bit-Register selbstverständlich auch über längere Zeit Daten speichern, wenn man nicht den Multiplikations- bzw. den Divisionsbefehl verwendet. Wir kennen jetzt alle Register in der CPU. Im folgenden Bild sind alle diese Register aufgeführt. Daneben sind die Speicher erwähnt, die sich mit der CPU im Mikroprozessor-IC befinden:

Mikroprozessor-IC



In unserem Mikroprozessor gibt es drei 8-Bit-Register: den Akku, das Extension- und das Status-Register. Bei einigen Befehlen können Akku und E-Register zusammen als 16-Bit-Register, als erweiterter Akkumulator oder als EA-Register verwendet werden.

Die fünf 16-Bit-Register (T-Register, PC, SP, P2 und P3) sind jetzt alle erwähnt worden. Mit den beiden Pointer-Registern P2 und P3 werden wir uns erst später ausführlicher beschäftigen.

Die aufgeführten Register sind für uns erreichbar, d. h. wir können in sie Daten speichern und aus ihnen Daten holen. Daneben gibt es im Mikroprozessor Register, die bei jedem Befehl benutzt werden, die nur kurzzeitig als Zwischenspeicher für Daten und Adressen Verwendung finden. Es sind Register, die für uns unsichtbar ihre speziellen Aufgaben erfüllen. Bei jedem Befehl gibt der Mikroprozessor zunächst eine Adresse über ein 16-Bit-Adress-Register auf den Adress-Bus. Dadurch wird ein bestimmter Speicher im ROM- oder RAM-Bereich adressiert, aus dem der nächste Befehl (oder Befehlsteil) geladen werden soll. Der Befehl wird jetzt über den Daten-Bus in das 8-Bit-Befehlsregister geladen. Bei Zwei- oder Drei-Byte-Befehlen gelangen die weiteren Bytes auch über den Datenbus in das 8-Bit-Datenregister (vgl. Abschnitt 14.8). Erst jetzt kann der Befehl im Mikroprozessor ausgeführt werden.

Zurück zum T-Register. Wenn wir dieses Register benutzen wollen, müssen wir wissen, wie das T-Register geladen werden kann, wir müssen wissen, wie wir den Inhalt des T-Registers abfragen können. Es gibt zwei 1-Byte-Befehle für den Datentransport zwischen dem EA- und dem T-Register.

BESCHREIBUNG	MNEM. CODE	OP. CODE	OPERATION
LADEN DEN INHALT DES EA-REGISTERS IN DAS T-REGISTER	LD T,EA	09	$\langle T \rangle := \langle EA \rangle$
LADEN DEN INHALT DES T-REGISTERS IN DAS EA-REGISTER	LD EA,T	0B	$\langle EA \rangle := \langle T \rangle$

Im mnemonischen Code wird zunächst das Register angegeben, in das geladen wird. Das Register, aus dem geladen wird, behält seinen Inhalt.

Weiter gibt es die Möglichkeit, das T-Register unmittelbar mit einer vierstelligen Hexadezimalzahl zu laden.

BESCHREIBUNG	MNEM. CODE	OP. CODE	OPERATION
LADEN UNMITTELBAR IN DAS T-REGISTER	LD T,=ZAHL	04 YY XX	$\langle T \rangle := \text{ZAHL}$

Als Zahl darf jede Dezimalzahl von 0 bis 65 535 oder jede Hexadezimalzahl von 0000 bis FFFF gesetzt werden. Im Operationscode wird eine vierstelligen Hexadezimalzahl zugrundegelegt. XX bezeichnet das high order byte (HOB), YY das low order byte (LOB).

Auch das Laden des T-Registers aus dem Speicherbereich FF00 bis FFFF ist mit direkter Adressierung möglich.

BESCHREIBUNG	MNEM. CODE	OP. CODE	OPERATION
LADEN AUS DEM SPEICHER IN DAS T-REGISTER (DIREKTE ADRESSIERUNG)	LD T,BEZ	05 XX	$\text{ADR} := \text{FF00} + \text{XX}$ $\langle T \rangle := \langle \text{ADR} + 1, \text{ADR} \rangle$

BEZ steht für die Bezeichnung eines Speichers oder eines Speicherinhalts. Dieser Speicher muß eine Adresse zwischen FF00 und FFFE haben, denn die Operation betrifft außer diesem einen Speicher noch einen zweiten mit der

folgenden Adresse. Das zweite Byte der ersten Adresse ist beim Operationscode mit XX bezeichnet. Die vollständige Adresse ADR ergibt sich dann als Summe $\text{FF00} + \text{XX}$. Der Inhalt der beiden Speicher $\text{ADR} + 1$ und ADR wird als vierstellige Hexadezimalzahl in das T-Register geladen. Den Inhalt des T-Registers können wir uns z. B. beim CALL TEST mit Hilfe der Tasten **CPU**, **4** anzeigen lassen. Es erscheint eine vierstellige Hexadezimalzahl und ein Symbol, das wir mit etwas Phantasie als kleines t interpretieren können.

Nachdem wir jetzt wissen, wie das T-Register geladen werden kann, können wir uns den Multiplikationsbefehl ansehen. Es ist ein 1-Byte-Befehl. Die beiden Faktoren müssen vorher in das T- und das EA-Register gebracht werden.

BESCHREIBUNG	MNEM. CODE	OP. CODE	OPERATION
MULTIPLIZIERE DIE INHALTE VON T-REGISTER UND EA-REGISTER	IMPY EA,T	2C	$\text{PROD} := \langle T \rangle * \langle EA \rangle$ $\langle EA \rangle := \text{PROD}$ BIT 31 - BIT 16 $\langle T \rangle := \text{PROD}$ BIT 15 - BIT 0

Der Befehl erlaubt die Multiplikation von zwei vierstelligen Hexadezimalzahlen. Das Ergebnis kann hexadezimal maximal achtstellig werden. Die höheren Ziffern des Produktes PROD stehen anschließend im EA-Register, die weiteren Ziffern stehen im T-Register. Beide Faktoren gehen verloren.

Eine Einschränkung gibt es: Die Hexadezimalzahl, die vor Ausführung der Multiplikation (bei unserem Programm als 2. Faktor) ins EA-Register geladen wird, darf maximal 7FFF sein. Das höchste Produkt, das wir mit dem Multiplikationsbefehl berechnen können, ist $\text{FFFF} \cdot 7\text{FFF} = 7\text{FFE } 8001$. Es kann also eine sechzehnstelligen Dualzahl mit einer fünfzehnstelligen multipliziert werden. Das Produkt ist eine maximal 31stelligen Dualzahl (Bit 30 bis Bit 0).

Erwähnt werden soll noch, daß sich bei der Ausführung des Multiplikationsbefehles der Inhalt des CY/L-Flags ändern kann.

Die Multiplikation verläuft hier etwa so, wie wir es bei unserem zweiten Multiplikationsprogramm gesehen haben: Verdoppeln, halbieren und addieren. Es handelt sich hier aber nicht um ein Unterprogramm, das mit dem Operationscode 2C aufgerufen wird. Im Mikroprozessor befindet sich eine elektronische Schaltung, mit deren Hilfe diese Operation durchgeführt wird. Das geht schneller, weil nicht ständig die weiteren Befehle geladen werden müssen.

Dieses Multiplikationsprogramm ist jetzt geklärt. In den ersten vier Befehlen werden T- und EA-Register geladen. Der 5. Befehl ist der Multiplikationsbefehl. Da in dem Programm nur zweistellige Hexadezimalzahlen multipliziert werden, ist das Ergebnis maximal vierstellig. Es steht also nach der Multiplikation im T-Register. Es wird dann mit Hilfe der beiden Befehle LD EA, T und CALL ANZ EA angezeigt.

Wir wollen das Programm noch so erweitern, daß vierstellige Hexadezimalzahlen multipliziert werden. Bei der Eingabe können wir den CALL EIN 2ST durch den CALL EIN 4ST ersetzen. Aber wie programmieren wir die achtstellige Anzeige? Der CALL ANZ EA ist jetzt nicht geeignet, da bei ihm die vier rechten Sieben-Segment-Anzeigen gelöscht werden. Beim CALL UEB 4R werden die vier linken Sieben-Segment-Anzeigen gelöscht. Es ergibt sich folgende Möglichkeit: Der Inhalt des EA-Registers, der links angezeigt werden soll, muß erst in einen Hilfsspeicher gebracht werden.

Mit dem CALL UEB 4R bereiten wir die Anzeige der vier rechten Stellen des Ergebnisses (Inhalt des T-Registers) vor. Dann folgt die Vorbereitung für die Anzeige der vier linken Stellen mit dem CALL UEB 4L. Beim CALL ANZ EIN werden dann alle acht Stellen angezeigt.

INR.	MARKE	MMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	ICALL EIN 4ST	1000	17	
2.		ILD T,ZWSP	1001	85 D8	(T):=FAKT1
3.		ICALL EIN 4ST	1003	17	
4.		ILD EA,ZWSP	1004	85 D8	(EA):=FAKT2
5.		IMPY EA,T	1006	12C	PROD:=(T)*(EA)
6.		IST EA,HSP	1007	8D E0	
7.		ILD EA,T	1009	0B	
8.		IST EA,ZWSP	100A	8D D8	VORBEREITUNG DER ANZEIGE
9.		ICALL UEB 4R	100C	13	DER RECHTEN STELLEN
10.		ILD EA,HSP	100D	85 E0	
11.		IST EA,ZWSP	100F	8D D8	VORBEREITUNG DER ANZEIGE
12.		ICALL UEB 4L	1011	14	DER LINKEN STELLEN
13.		ICALL ANZ EIN	1012	10	ANZEIGE: PROD, 8-STELLIG
14.		INOP	1013	00	
15.		INOP	1014	00	
16.		IBRA BEGINN	1015	74 E9	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZWSP	1FFD8	ZWISCHENSPEICHER
HSP	1FFD9	
	1FFE0	HILFSSPEICHER
	1FFE1	

Wir wollen einige Multiplikationsbeispiele angeben:

Nr.	1. Faktor (T)	2. Faktor (EA)	erwartetes Produkt	Anzeige
1.	444	2000	88 8000	0 0 8 8 8 0 0 0
2.	F00	F0	E 1000	0 0 0 E 1 0 0 0
3.	FFFF	1000	FFF F000	0 F F F F 0 0 0
4.	FFFF	7FFF	7FFE 8001	7 F F E 8 0 0 1
5.	2710	2710	5F5 E100	0 5 F 5 E 1 0 0
6.	7FFF	FFFF	7FFE 8001	F F F F 8 0 0 1
7.	1111	8000	888 8000	F 7 7 7 8 0 0 0
8.	1	FFFF	FFFF	F F F F F F F F
9.	1000	1000	100 0000	0 1 0 0 0 0 0
10.	200	201	4 0200	0 0 0 4 2 0 0
11.	1	1	1	0 0 0 0 0 0 1

In den ersten fünf Beispielen erhalten wir das erwartete Ergebnis, das allerdings achtestellig mit führenden Nullen angezeigt wird. Der Rechnung im fünften Beispiel entspricht im Dezimalsystem die Rechnung $10.000 \cdot 10.000 = 100.000.000$.

Im sechsten bis achten Beispiel ist der 2. Faktor (Faktor im EA-Register) größer als 7FFF. Das Ergebnis wird falsch, weil wir die Grenze für den maximalen Wert des zweiten Faktors nicht beachtet haben. Der Multiplikationsbefehl wird zwar ausgeführt, aber die elektronische Schaltung ist für diese Produkte nicht geeignet. Das größte Produkt, das mit Hilfe dieses Befehls berechnet werden kann, ist $FFFF \cdot 7FFF$. Oder anders formuliert: Es können maximal sechzehnstelligen Dualzahlen (T) mit maximal fünfzehnstelligen Dualzahlen (EA) multipliziert werden.

Bei den drei letzten Beispielen wird deutlich, daß unsere Anzeige noch verbessert werden müßte. Beim CALL UEB 4R werden die führenden Nullen gelöscht, beim CALL UEB 4L werden die führenden Nullen nicht gelöscht. Für andere Anwendungen waren diese Calls gerade so erwünscht. Hier wären sie anders sinnvoller. Wir wollen diesen Nachteil hier nur erwähnen. Im 18. Kapitel werden wir untersuchen, wie dieser Nachteil abgestellt werden kann.

16.6 Dezimale Multiplikation

Nach der vielen Multiplikation im Hexadezimalsystem soll wenigstens ein Multiplikationsprogramm für Dezimalzahlen angegeben werden. Wir wissen, wie unsere bisherigen Programme abgeändert werden müssen: Nach der Eingabe müssen wir die eingegebenen Zahlen in Hexadezimalzahlen umwandeln. Nach der Multiplikation muß das Ergebnis wieder ins Dezimalsystem übersetzt werden (vgl. Abschnitt 8.5).

Wir werden für die Eingabe der Dezimalzahlen den CALL EIN 4ST benutzen. Wir wollen uns aber damit begnügen, Produkte zu berechnen, die höchstens 9999 sind. Andernfalls könnten wir nicht ohne weiteres den CALL HEX-DEZ benutzen. Wir werden auch hierzu im 18. Kapitel Überlegungen anstellen, wie die Umwandlung zwischen den Zahlensystemen auf größere Zahlen ausgedehnt werden kann.

INR.	MARKE	MMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	ICALL EIN 4ST	1000	17	
2.		ILD EA,ZWSP	1001	85 D8	
3.		ICALL DEZ-HEX	1003	1B	
4.		ILD T,EA	1004	09	(T):=FAKT1
5.		ICALL EIN 4ST	1005	17	
6.		ILD EA,ZWSP	1006	85 D8	
7.		ICALL DEZ-HEX	1008	1B	(EA):=FAKT2
8.		IMPY EA,T	1009	12C	PROD:=(T)*(EA)
9.		ILD EA,T	100A	0B	
10.		ICALL HEX-DEZ	100B	1C	
11.		ICALL ANZ EA	100C	19	ANZEIGE: PROD
12.		IBRA BEGINN	100D	74 F1	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZWSP	1FFD8	ZWISCHENSPEICHER
	1FFD9	

Wenn wir unser Programm starten und z. B. die beiden Faktoren 80 und 25 eingeben, werden wir eine böse Überraschung erleben. Nach unserem Programm ergibt sich das Ergebnis 500. Wir wissen es besser: $80 \cdot 25 = 2000$. Wo ist der Fehler?

Wir benutzen den CALL TEST, um zunächst die Eingabe des ersten Faktors in das T-Register zu überprüfen. Wir schreiben den Operationscode 1E in den Speicher 1005, starten das Programm neu und geben den Faktor 80 ein. Nach Betätigung einer Funktionstaste erscheint die Anzeige

1 0 0 5 1 E

Wir überprüfen mit CPU, 4 den Inhalt des T-Registers: (T) = 50. Bis hier ist alles in Ordnung. Die eingegebene Dezimalzahl 80 wurde in die entsprechende Hexadezimalzahl umgewandelt und im T-Register abgespeichert ($80_{10} = 50_{16}$). Wir ersetzen den Inhalt 1E im Speicher 1005 wieder durch 17 und schreiben den Breakpoint (1E) jetzt in den Speicher mit der Adresse 1009. Wir starten wieder das Programm, geben die Faktoren 80 und 25 ein und überprüfen die Registerinhalte:

CPU, 7 : (E) = 00;
CPU, 6 : (A) = 19;
CPU, 4 : (T) = 14.

Im EA-Register steht die richtige Zahl 0019, denn $25_{10} = 19_{16}$. Aber im T-Register steht jetzt unerwartet eine andere Zahl als 50_{16} . Damit haben wir den Fehler gefunden. Der Inhalt des T-Registers hat sich bei der Ausführung der Befehle 5. bis 7. verändert. Das kann nur beim CALL DEZ-HEX passiert sein, denn die beiden anderen Befehle haben nichts mit dem T-Register zu tun. Von dieser Stelle ab läuft

unser Programm auch wieder erwartungsgemäß. Jetzt werden die beiden Hexadezimalzahlen 14 und 19 multipliziert, das Ergebnis wird in eine Dezimalzahl verwandelt und angezeigt.

$$\begin{array}{r} 14_{16} \cdot 19_{16} \\ \text{B } 4 \\ 1 \text{ 4} \\ \hline 1 \text{ F } 4_{16} \end{array}$$

$$\begin{aligned} 1\text{F}4_{16} &= 256_{10} + 15 \cdot 16_{10} + 4_{10} \\ &= 256_{10} + 240_{10} + 4_{10} \\ &= 500_{10} \end{aligned}$$

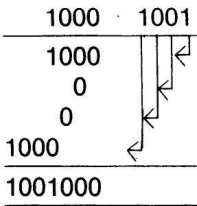
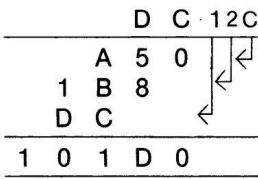
Wenn wir unser Programm jetzt korrigieren wollen, müssen wir dafür sorgen, daß das T-Register für die Multiplikation erst geladen wird, wenn die Umwandlung der Faktoren in Hexadezimalzahlen abgeschlossen ist. Wir merken uns: Beim CALL DEZ-HEX (und auch beim CALL HEX-DEZ) wird das T-Register benutzt. Es verliert dabei den vorherigen Inhalt. Wir werden uns in Abschnitt 16.8 davon überzeugen, wenn wir diese beiden Calls näher untersuchen. Das Programm für die dezimale Multiplikation kann jetzt korrigiert werden:

INR.	MARKE	MMEM.CODE	IADR.	IOP.CODE	BEMERKUNGEN
1.	BEGINN	ICALL EIN 4ST	1000	17	
2.		ILD EA,ZWSP	1001	85 D8	
3.		ICALL DEZ-HEX	1003	1B	
4.		IST EA,HSP	1004	8D E0	(HSP):=FAKT1
5.		ICALL EIN 4ST	1006	17	
6.		ILD EA,ZWSP	1007	85 D8	
7.		ICALL DEZ-HEX	1009	1B	
8.		ILD T,EA	100A	09	(T):=FAKT2
9.		ILD EA,HSP	100B	85 E0	(EA):=FAKT1
10.		IMPY EA,T	100D	12C	PROD:=(T)*(EA)
11.		ILD EA,T	100E	08	
12.		ICALL HEX-DEZ	100F	1C	
13.		ICALL ANZ EA	1010	19	ANZEIGE: PROD
14.		IBRA BEGINN	1011	74 ED	

DATENSPEICHER, BEZEICHNUNG	IADR.	BEMERKUNGEN
ZWSP	1FFD8	ZWISCHENSPEICHER
	1FFD9	
HSP	1FFE0	HILFSSPEICHER
	1FFE1	

Das Programm multipliziert jetzt die beiden Faktoren 80 und 25 richtig. Das Ergebnis ist wie erwartet 2000. Das Programm berechnet auch alle Produkte richtig, die (dezimal) kleiner als 10000 sind. Wenn das T-Register nach der Multiplikation eine Hexadezimalzahl enthält, die größer als 270F ist, wird beim CALL HEX-DEZ die Anzeige „Error“ erzeugt, vgl. Abschnitt 8.4
Aber es gibt Beispiele, bei denen uns die Ergebnis-Anzeige nicht gefällt: Für 220·300 ermittelt das Programm 464, für 4096·4097 ermittelt das Programm 4096. Woran liegt das? Das Produkt ist als Hexadezimalzahl sicher größer als 270F. Um eine Lösung zu finden, rechnen wir die beiden Beispiele nach:

$$\begin{aligned} 220_{10} \cdot 300_{10} &= \text{DC}_{16} \cdot 12\text{C}_{16} \\ &= 1\text{01D0}_{16} \\ &\quad \downarrow \quad \quad \quad \downarrow \\ 66000_{10} &= 65536_{10} + 464_{10} \end{aligned}$$
$$\begin{aligned} 4096_{10} \cdot 4097_{10} &= 1000_{16} \cdot 1001_{16} \\ &= 100\ 1000_{16} \end{aligned}$$



In beiden Fällen verstehen wir jetzt die Anzeige. Im ersten Beispiel steht nach der Multiplikation im Akku der Inhalt 01, im T-Register der Inhalt 01D0. 01D0₁₆ wird in die Dezimalzahl 464₁₀ umgewandelt und angezeigt.
Im zweiten Beispiel erhalten wir nach der Multiplikation die Registerinhalte: (E) = 01, (A) = 00, (T) = 1000. Nur der Inhalt des T-Registers wird umgewandelt und angezeigt.
Wenn wir in diesen Fällen die falsche Ergebnis-Anzeige unterdrücken, wenn wir auch hier die Anzeige „Error“ erzeugen wollen, müssen wir den CALL FEHLER auch dann aufrufen, wenn der Inhalt des EA-Registers nach der Multiplikation ungleich null ist. Diese Programmänderung ist mit drei Befehlen möglich, die wir nach dem Multiplikationsbefehl einfügen können.

INR.	MARKE	MMEM.CODE	IADR.	IOP.CODE
1.	BEGINN	ICALL EIN 4ST	1000	17
10.		IMPY EA,T	100D	12C
11.		IOR A,E	100E	58
12.		IBZ ERGEBN	100F	6C 01
13.		ICALL FEHLER	1011	1A
14.	ERGEBN	ILD EA,T	1012	08
15.		ICALL HEX-DEZ	1013	1C
16.		ICALL ANZ EA	1014	19
17.		IBRA BEGINN	1015	74 E9

16.7 Dezimale Division

Bei der Division wollen wir kürzer fassen. Wir geben gleich den Divisionsbefehl an, und wir werden uns mit einem Divisionsprogramm für Dezimalzahlen begnügen.

BESCHREIBUNG	MMEM.CODE	IOP.CODE	OPERATION
DIVIDIERE DEN INHALT DES EA-REGISTERS DURCH DEN INHALT DES T-REGISTERS	DIV EA,T	0D	(EA):=(EA)/(T)

Vor der Befehlsausführung muß die Zahl, die dividiert werden soll (der Dividend, der Zähler) ins EA-Register, die Zahl, durch die dividiert werden soll (der Divisor, der Nenner), ins T-Register gebracht werden. Beide Zahlen können maximal vierstellige Hexadezimalzahlen sein. Der Divisor darf natürlich nicht 0 sein, er darf auch nicht größer als 7FFF sein. Das Ergebnis wird gegebenenfalls abgerundet. Es kann maximal vierstellig sein. Es steht nach der Befehlsausführung im EA-Register. Im T-Register finden wir nach der Division einen undefinierten Wert, nicht etwa den Rest, der bei der Division entsteht.
Beispiel:
Vor Befehlsausführung: (EA) = 00E8; (T) = 0014;
nach Befehlsausführung: (EA) = 000B; (T) = 0019.

Hierzu ein kleines Testprogramm:

NR.	MARKE	INMEM.CODE	ADR.	OP.CODE
1.	BEGINN	LD EA,=0E8	1000	84 E8 00
2.		LD T,=014	1003	A4 14 00
3.		IDIV EA,T	1006	0D
4.		ICALL TEST	1007	1E

Nachrechnen ergibt:

$$\begin{array}{c} \text{E8 : 14} = \text{B} + \frac{\text{C}}{14} \\ \updownarrow \quad \updownarrow \quad \updownarrow \quad \updownarrow \\ \text{dezimal: } 232 : 20 = 11 + \frac{12}{20} = 11,6 \end{array}$$

Das abgerundete, ganzzahlige Ergebnis B steht im EA-Register, der Rest C ist in keinem Register der CPU vorhanden. Er geht verloren. Bemerkt werden soll auch noch, daß sich der Inhalt des CY/L-Flags ändern kann.

Mit Hilfe des folgenden Programms können Dezimalzahlen dividiert werden. Wer Hexadezimalzahlen dividieren möchte, braucht nur an drei Stellen die Calls CALL DEZ-HEX und CALL HEX-DEZ durch NOP-Befehle zu ersetzen.

NR.	MARKE	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	ICALL EIN 4ST	1000	17	EINGABE: DIVIDEND
2.		ILD EA,ZWSP	1001	85 D8	
3.		ICALL DEZ-HEX	1003	1B	
4.		IST EA,DIVID	1004	8D E0	
5.		ICALL EIN 4ST	1006	17	EINGABE: DIVISOR
6.		ILD EA,ZWSP	1007	85 D8	
7.		ICALL DEZ-HEX	1009	1B	
8.		ILD T,EA	100A	09	
9.		ILD EA,DIVID	100B	85 E0	
10.		IDIV EA,T	100D	0D	(EA):=(EA)/<T>
11.		ICALL HEX-DEZ	100E	1C	
12.		ICALL ANZ EA	100F	19	ANZEIGE: QUOTIENT
13.		IBRA BEGINN	1010	74 EE	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZWSP	FFD8	ZWISCHENSPEICHER
	FFD9	
DIVID	FFE0	DIVIDEND, ZAEHLER
	FFE1	

Die Zahl, die dividiert werden soll (der Dividend, der Zähler) wird zuerst eingegeben. Dann folgt die Eingabe der Zahl, durch die dividiert werden soll (der Divisor, der Nenner). Wir wollen unser Programm noch so ergänzen, daß außer dem (abgerundeten, ganzzahligen) Ergebnis der Rest ermittelt und zusätzlich angezeigt wird.

NR.	MARKE	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	ICALL EIN 4ST	1000	17	EINGABE: DIVIDEND
2.		ILD EA,ZWSP	1001	85 D8	
3.		ICALL DEZ-HEX	1003	1B	
4.		IST EA,DIVID	1004	8D E0	
5.		ICALL EIN 4ST	1006	17	EINGABE: DIVISOR
6.		ILD EA,ZWSP	1007	85 D8	
7.		ICALL DEZ-HEX	1009	1B	
8.		IST EA,DIVIS	100A	8D E2	
9.		ILD T,EA	100C	09	
10.		ILD EA,DIVID	100D	85 E0	
11.		IDIV EA,T	100F	0D	QUOT:=DIVID/DIVIS
12.		IST EA,QUOT	1010	8D E4	
13.		ILD T,EA	1012	09	
14.		ILD EA,DIVIS	1013	85 E2	
15.		IMPV EA,T	1015	2C	
16.		ILD EA,T	1016	08	
17.		IST EA,HSP	1017	8D E6	
18.		ILD EA,DIVID	1019	85 E0	
19.		ISUB EA,HSP	101B	8D E6	REST:=DIVID - QUOT*DIVIS
20.		ICALL HEX-DEZ	101D	1C	
21.		IST EA,ZWSP	101E	8D D8	VORBEREITUNG DER ANZEIGE
22.		ICALL UEB 4R	1020	13	DES RESTES, RECHTS
23.		ILD EA,QUOT	1021	85 E4	
24.		ICALL HEX-DEZ	1023	1C	
25.		IST EA,ZWSP	1024	8D D8	VORBEREITUNG DER ANZEIGE
26.		ICALL UEB 4L	1026	14	DES QUOTIENTEN, LINKS
27.		ICALL ANZ EIN	1027	10	ANZEIGE: QUOTIENT
28.		INOP	1028	00	UND REST
29.		INOP	1029	00	
30.		IBRA BEGINN	102A	74 D4	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZWSP	FFD8	ZWISCHENSPEICHER
	FFD9	
DIVID	FFE0	DIVIDEND, ZAEHLER
	FFE1	
DIVIS	FFE2	DIVISOR, NENNER
	FFE3	
QUOT	FFE4	QUOTIENT
	FFE5	
HSP	FFE6	HILFSSPEICHER
	FFE7	

Die ersten sieben Befehle stimmen mit denen aus dem vorangegangenen Programm überein. Um nach der Division den Rest zu ermitteln, wird von dem eingegebenen Dividenten das Produkt aus Quotient und Divisor subtrahiert. Dieser Rest wird dezimal rechts, der dezimale Quotient wird links angezeigt.

16.8 Die Calls CALL DEZ-HEX und CALL HEX-DEZ

Wir wollen uns am Ende dieses Kapitels die beiden Calls zur Umwandlung zwischen dem Dezimal- und dem Hexadezimalsystem näher ansehen.

Wir erinnern uns: Vor dem Aufruf des Calls muß die umzuwandelnde Zahl im EA-Register stehen. Am Ende dieses Unterprogramms steht die umgewandelte Zahl im EA-Register.

16.8.1 Der CALL DEZ-HEX

Aus Abschnitt 13.1 wissen wir, daß dieses Unterprogramm, das als CALL DEZ-HEX mit dem Operationscode 1B aufgerufen werden kann, im ROM steht und im Speicher mit der Adresse 08A1 beginnt. Wenn wir dieses Programm analysieren wollen, müssen wir zunächst die Speicherinhalte ab (08A1) herausschreiben.

ADRESSE	INHALT
08A1	8D
08A2	D8
08A3	A4
08A4	E8
08A5	03
08A6	84
08A7	00
08A8	00
08A9	C5
08AA	D9
08AB	3C
08AC	3C
08AD	3C
08AE	3C
08AF	2C
08B0	0B
08B1	8D
08B2	DA
08B3	A4
08B4	64
08B5	00
08B6	84
08B7	00
08B8	00
08B9	C5
08BA	

Die angegebene Liste ist nur der Anfang. Wir wissen, daß der Call – wie jedes andere Unterprogramm – mit einem Return-Befehl abgeschlossen wird. Zum Programm gehören also die Speicherinhalte bis zum ersten Return-Befehl mit dem Operationscode 5C. Diesen Inhalt finden wir erst im Speicher 08DE. Es sind also 62 Speicherinhalte zu untersuchen ($08DE - 08A0 = 3E_{16} = 62_{10}$). Im zweiten Schritt müssen die Speicherinhalte zu vollständigen Operationscodes geordnet werden. Mit 8D beginnt ein Store-Befehl (ST EA, ...; 2 Byte), dann beginnt mit A4 ein Load-Befehl (LD T, ...; 3 Byte), usw.

ADRESSE	INHALT	ADRESSE	OP. CODE
08A1	8D	08A1	8D D8
08A2	D8	08A3	A4 E8 03
08A3	A4	08A6	84 00 00
08A4	E8	08A9	C5 D9
08A5	03	08AB	3C
08A6	84	08AC	3C
08A7	00	08AD	3C
08A8	00	08AE	3C
08A9	C5	08AF	2C
08AA	D9	08B0	0B
08AB	3C	08B1	8D DA
08AC	3C	08B3	A4 64 00
08AD	3C	08B6	84 00 00
08AE	3C	08B9	C5 D9
08AF	2C	08BB	D4 0F
08B0	0B	08BD	2C
08B1	8D	08BE	0B
08B2	DA	08BF	B5 DA
08B3	A4	08C1	8D DA
08B4	64	08C3	A4 0A 00
08B5	00	08C6	84 00 00
08B6	84	08C9	C5 D8
08B7	00	08CB	3C
08B8	00	08CC	3C
08B9	C5	08CD	3C
08BA	D9	08CE	3C
08BB	3C	08CF	2C
08BC	3C	08D0	0B
08BD	3C	08D1	B5 DA
08BE	3C	08D3	8D DA
08BF	2C	08D5	84 00 00
08C0	0B	08D8	C5 D8
08C1	8D	08DA	D4 0F
08C2	DA	08DC	B5 DA
08C3	A4	08DE	5C
08C4	64		
08C5	00		
08C6	84		
08C7	00		
08C8	00		
08C9	C5		
08CA	D9		
08CB	3C		
08CC	3C		
08CD	3C		
08CE	3C		
08CF	2C		
08D0	0B		
08D1	B5		
08D2	DA		
08D3	8D		
08D4	DA		
08D5	84		
08D6	00		
08D7	00		
08D8	C5		
08D9	D9		
08DA	D4		
08DB	0F		
08DC	B5		
08DD	DA		
08DE	5C		

NR.	MARKE	MMEM. CODE	ADR.	OP. CODE	BEMERKUNGEN
1.	DEZHEX	ST EA,DEZ	08A1	8D D8	(BEGINN: CALL DEZ-HEX (EA)=DEZIMALZAHL DEZ:=<EA>)
2.	TAUS.	LD T,=1000	08A3	A4 E8 03	HEX:=TAUS.* 03E8
3.		LD EA,=0	08A6	84 00 00	
4.		LD A,DEZHOB	08A9	C5 D9	
5.		SR A	08AB	3C	
6.		SR A	08AC	3C	
7.		SR A	08AD	3C	
8.		SR A	08AE	3C	
9.		IMPY EA,T	08AF	2C	
10.		LD EA,T	08B0	0B	
11.		ST EA,HEX	08B1	8D DA	
12.	HUND.	LD T,=100	08B3	A4 64 00	HEX:=HEX + HUND.* 064
13.		LD EA,=0	08B6	84 00 00	
14.		LD A,DEZHOB	08B9	C5 D9	
15.		AND A,=0F	08BB	D4 0F	
16.		IMPY EA,T	08BD	2C	
17.		LD EA,T	08BE	0B	
18.		ADD EA,HEX	08BF	B5 DA	
19.		ST EA,HEX	08C1	8D DA	
20.	ZEHNER	LD T,=10	08C3	A4 0A 00	HEX:=HEX + ZEHNER* 0A
21.		LD EA,=0	08C6	84 00 00	
22.		LD A,DEZLOB	08C9	C5 D8	
23.		SR A	08CB	3C	
24.		SR A	08CC	3C	
25.		SR A	08CD	3C	
26.		SR A	08CE	3C	
27.		IMPY EA,T	08CF	2C	
28.		LD EA,T	08D0	0B	
29.		ADD EA,HEX	08D1	B5 DA	
30.		ST EA,HEX	08D3	8D DA	
31.	EINER	LD EA,=0	08D5	84 00 00	HEX:=HEX + EINER
32.		LD A,DEZLOB	08D8	C5 D8	
33.		AND A,=0F	08DA	D4 0F	
34.		ADD EA,HEX	08DC	B5 DA	
35.		RET	08DE	5C	(EA)=HEXADEZIMALZAHL

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
DEZ	1FFD8 DEZIMALZAHL	
	1FFD9	
DEZLOB	1FFD8 DEZIMALZAHL, LOW ORDER BYTE	
DEZHOB	1FFD9 DEZIMALZAHL, HIGH ORDER BYTE	
HEX	1FFD8 HEXADEZIMALZAHL	
	1FFD9	

Bei diesen Überlegungen ergeben sich gleichzeitig die mnemonischen Codes für die Befehle. Man muß nur noch sinnvolle Bezeichnungen für die verwendeten Datenspeicher festlegen. Das könnte beispielsweise so geschehen, wie es im folgenden Programmausdruck gemacht ist.

Das Programm für diesen Call ist leicht zu verstehen. Da keine Sprungbefehle vorkommen, brauchten keine Marken gesetzt zu werden. Die angegebenen Marken dienen nur der besseren Übersicht. Diesem Programm liegt die folgende Überlegung zugrunde:

Die vier Stellen der Dezimalzahl DEZ seien mit D_T (Tausender), D_H (Hunderter), D_Z (Zehner) und D_E (Einer) bezeichnet: DEZ = D_T D_H D_Z D_E. Es gilt:

$$1000_{10} = 3E8_{16}; \quad 100_{10} = 64_{16}; \quad 10_{10} = A_{16}.$$

Daher ist die entsprechende Hexadezimalzahl

$$\text{HEX} = D_T \cdot 3E8 + D_H \cdot 64 + D_Z \cdot A + D_E.$$

Der entsprechende Faktor wird ins T-Register geladen. Mit dem Befehl LD EA, = 0 wird das Extension-Register gelöscht (möglich wäre auch: LD A, = 0; LD E, A). Dann wird die gewünschte Dezimalziffer in den Akku gebracht. Mit LD A, DEZ HOB und viermal SR A ist der Inhalt vom EA-Register 000D_T. Mit LD A, DEZ HOB und Maskieren (AND A, = 0F) ist der Inhalt vom EA-Register 000D_H. Weitere Erklärungen sind sicher nicht erforderlich. Bei Programmende steht die gewünschte Hexadezimalzahl im EA-Register.

16.8.2 Der CALL HEX-DEZ

Der CALL HEX-DEZ, der mit dem Operationscode 1C aufgerufen wird, beginnt im ROM im Speicher mit der Adresse 08DF. Wer Lust hat, sollte den folgenden Programmausdruck zunächst zur Seite legen und allein versuchen, den Call zu analysieren. Es kommt nur ein 1-Byte-Befehl (Operationscode 3A) vor, der bisher noch nicht besprochen wurde. Dieser Befehl dient dazu, daß der Stackpointer wieder seinen vorherigen Wert bekommt, wenn das Unterprogramm über den CALL FEHLER (Anzeige „ERROR“) verlassen wird.

In den ersten sieben Befehlen wird geprüft, ob die eingegebene Hexadezimalzahl größer als 270F₁₆ (= 9999₁₀) ist. In dem Fall läßt sie sich nicht in eine vierstellige Dezimalzahl umwandeln. Diese Überprüfung geschieht so, daß von der gegebenen Hexadezimalzahl 2710₁₆ (= 10000₁₀) subtrahiert wird. Ist das Subtraktionsergebnis negativ, kann die Umwandlung beginnen. Wir erinnern uns: Das CY/L-Flag zeigt nach einer Subtraktion mit einer 1 an, daß das Subtraktionsergebnis positiv ist (größer oder gleich null), es zeigt mit einer 0 an, daß das Ergebnis negativ ist. Es wird daher

INR.	MARKE	INHEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.		1HEXDEZ	18DF	18D D6	1BEGINN: CALL HEX-DEZ (EA)=HEXADEZIMALZAHL (HEX)=<EA> (DEZIMALZAHL) > 9999 ?
2.		ISUB EA,=10000	18E1	18C 10 27	
3.		ILD A,S	18E4	186	
4.		IRAND A,=000	18E5	184 80	
5.		IBZ UMWAND	18E7	18C 02	
6.		ICALL FEHLER	18E9	18A	
7.			18EA	11A	"ERROR", WENN HEX > 270F
8.		UMWAND	18EB	185 D6	
9.		ITRAUS.	18ED	1A4 E8 03	(HEX)=HEX - TRAU.S. * 03E8
10.		IDIV EA,T	18F0	180	
11.		IST A,TRAUS.	18F1	1CD DD	
12.		ILD T,=1000	18F3	1A4 E8 03	
13.		IMPY EA,T	18F6	12C	
14.		ILD EA,T	18F7	186	
15.		IST EA,ABZUG	18F8	18D DA	
16.		ILD EA,HEX	18FA	185 D6	
17.		ISUB EA,ABZUG	18FC	18D DA	
18.		IST EA,HEX	18FE	18D D6	
19.		IHUND.	1890	1A4 64 00	(HEX)=HEX - HUND.* 064
20.		IDIV EA,T	1893	180	
21.		IST A,HUND.	1894	1CD DF	
22.		ILD T,=100	1896	1A4 64 00	
23.		IMPY EA,T	1899	12C	
24.		ILD EA,T	189A	186	
25.		IST EA,ABZUG	189B	18D DA	
26.		ILD EA,HEX	189D	185 D6	
27.		ISUB EA,ABZUG	189F	18D DA	
28.		IST EA,HEX	1891	18D D6	
29.		IZEH,EI	18913	1A4 0A 00	(EINER)=HEX - ZEHNER* 0A
30.		IDIV EA,T	18916	180	
31.		IST A,ZEHNER	18917	1CD DC	
32.		ILD T,=10	18919	1A4 0A 00	
33.		IMPY EA,T	1891C	12C	
34.		ILD EA,T	1891D	186	
35.		IST EA,ABZUG	1891E	18D DA	
36.		ILD EA,HEX	18920	185 D6	
37.		ISUB EA,ABZUG	18922	18D DA	
38.		IST A,EINER	18924	1CD DE	
39.		IERGEBEN	18926	185 DC	(E)=TRAUS.; (A)=ZEHNER
40.		ISL EA	18928	18F	
41.		ISL EA	18929	18F	
42.		ISL EA	1892A	18F	
43.		ISL EA	1892B	18F	
44.		IADD EA,EINER	1892C	185 DE	10*ZEHNER + EINER 10*TAUS. + HUND.
45.		IRET	1892E	15C	(EA)=DEZIMALZAHL

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
HEX	1FFD6	(HEXADEZIMALZAHL)
	1FFD7	
ZEHNER	1FFDC	(DEZIMALZAHL)
TRAUS.	1FFDD	
EINER	1FFDE	
HUND.	1FFDF	
ABZUG	1FFDA	
	1FFDB	

16.8.3 Bemerkungen zu diesen Calls

Einige Punkte zu diesen Unterprogrammen CALL DEZ-HEX und CALL HEX-DEZ wollen wir festhalten:

1. Die Calls wandeln den vierstelligen Inhalt des EA-Registers um. Das Umwandlungsergebnis steht wieder im EA-Register.
2. Die größte Zahl, die umgewandelt werden kann, ist 9999_{10} , bzw. $270F_{16}$.
3. Soll eine zweistellige Zahl umgewandelt werden, muß sie vor Aufruf des Calls in den Akku geschrieben werden. Außerdem muß dann das Extension-Register gelöscht werden.
4. Das T-Register wird bei beiden Calls benutzt. Sein Inhalt wird also verändert.
5. Bei beiden Calls wird addiert. Der Inhalt des Status-Registers, zumindest der des CY/L-Flags, kann sich ändern.
6. Als Datenspeicher werden die Speicher mit den Adressen FFD6 bis FFDF benutzt. Die Inhalte in diesen Speichern ändern sich.
7. Die Datenspeicher FFE0, FFE1, ... werden nicht benutzt. Diese Speicherinhalte ändern sich beim Einsatz dieser Calls nicht.
8. Beide Unterprogramme sind lineare Programme (keine Sprünge, keine Verzweigung). Das bedeutet, daß die Umwandlungsprogramme immer die gleiche Zeit dauern, unabhängig davon, wie groß die umgewandelte Zahl ist. Diese Feststellung wird erst später wichtig sein, wenn wir uns mit der Bearbeitungsdauer bestimmter Programmteile beschäftigen werden.

im 3. Befehl der Inhalt des Status-Registers in den Akku geladen und im 4. Befehl das Bit 7 (CY/L) ausmaskiert. Bei (A) = 00 erfolgt jetzt der Sprung zum Programmteil „UMWAND“ (Umwandlung). Andernfalls erfolgt der Aufruf von CALL FEHLER mit der Anzeige „Error“. Die Umwandlung ist die Umkehrung von dem, was beim CALL DEZ-HEX passiert. Zur Bestimmung der Tausender (D_T , Tausenderziffer der gesuchten Dezimalzahl) wird die gegebene Hexadezimalzahl durch $03E8_{16}$ ($= 1000_{10}$) dividiert. Damit ist D_T gefunden, D_T kann gespeichert werden. In den Befehlen 12 bis 18 wird der Rest bestimmt, der bei der Division durch $03E8$ übrigbleibt:

$$\text{HEX} = \underbrace{D_T \cdot 03E8_{16}}_{\text{Abzug}} + \text{Rest}$$

Dieser Rest ergibt die Hexadezimalzahl, die vom 19. Befehl an weiter umgewandelt werden muß. Nach drei Divisionen bleiben die Einer (D_E) übrig. In den Befehlen 39. bis 44. werden die ermittelten Ziffern der Dezimalzahl in geeigneter Weise zusammengesetzt. Aus der vierstelligen Zahl $OD_T OD_Z$ wird durch vierfaches Linksschieben $D_T OD_Z O$. Durch Addition von $OD_H OD_E$ ergibt sich dann die gesuchte Dezimalzahl $DEZ = D_T D_H D_Z D_E$.

16.9 Aufgaben zu Kapitel 16

- 1. Das Programm von Abschnitt 16.3 soll so geändert werden, daß mit ihm Dezimalzahlen verdoppelt werden können. Die eingegebene Zahl soll als Dezimalzahl aufgefaßt werden. Sie soll wie im Programm von Abschnitt 16.3 links angezeigt werden. Bei jeder Betätigung der **RUN**-Taste soll die Dezimalzahl verdoppelt werden. Die LED-Reihe muß hier nicht beachtet werden.
- 2. Bei dem Programm von Abschnitt 16.4 wird die Schleife „SCHL.“ maximal achtmal durchlaufen. Welche Inhalte stehen vor dem 1. Schleifendurchlauf, nach jedem der acht Schleifendurchläufe in den Datenspeichern, wenn mit diesem Programm die Hexadezimalzahlen A3 (1. Faktor) und CE (2. Faktor) multipliziert werden?

Schleifen- durchlauf	1. Faktor (FFE3), (FFE2)	2. Faktor (FFD8)	Produkt (FFE1), (FFE0)
vor dem 1.			
nach dem 1.			
nach dem 2.			
nach dem 3.			
nach dem 4.			
nach dem 5.			
nach dem 6.			
nach dem 7.			
nach dem 8.			

- 3. Im Abschnitt 11.2.1 hatten wir gesagt, daß die dort bestimmte Summe s auch nach der Gleichung $s = \frac{1}{2} \cdot n \cdot (n + 1)$ berechnet werden kann. Das folgende Programm berechnet zu einer Dezimalzahl n (maximal 140) diese Summe mit Hilfe der Multiplikation.

3.1 Es ist nur der mnemonische Code gegeben. Die Befehle sollen in den Operationscode übersetzt werden.

NR.	MARKE	MNEM. CODE
1.	BEGINN	CALL EIN 4ST
2.		ILD EA,ZWSP
3.		CALL DEZ-HEX
4.		ILD T,EA
5.		IXCH A,E
6.		IBZ WEITER
7.		CALL FEHLER
8.	WEITER	ILD EA,T
9.		IADD EA,#1
10.		IMPY EA,T
11.		ILD EA,T
12.		ISR EA
13.		CALL HEX-DEZ
14.		CALL ANZ EA
15.		BRN BEGINN

- 3.2 Wenn die vier Befehle (5. bis 8.) weggelassen werden, wird die Summe für alle Zahlen n bis 140 auch richtig ermittelt. Wir erhalten dann aber bei der eingegebenen Zahl $n = 257$ das falsche Ergebnis 0385. Wie kommt es zu dieser Anzeige?

- 4. In Aufgabe 7 von Kapitel 11 ist ein Programm zur Berechnung der Quadratzahlen (bis 99^2) angegeben. Dieses Programm soll jetzt mit Hilfe des Multiplikationsbefehls kürzer geschrieben werden. Es sind nur neun Befehle erforderlich.
- 5. Wir hatten erwähnt, daß die Division auf die Subtraktion zurückgeführt werden kann. Von der ersten Zahl (Dividend) wird die zweite Zahl (Divisor) so oft abgezogen, wie die Differenz positiv (≥ 0) bleibt. Es wird mitgezählt, wie oft die Subtraktion ausgeführt wurde. So erhält man das Divisionsergebnis. Es soll ein Programm geschrieben werden, das nach diesem Verfahren zwei eingegebene (maximal vierstellige) Hexadezimalzahlen durcheinander dividiert.
- 6. Mit dem folgenden Programm können Potenzen im Dezimalsystem berechnet werden.

ADRESSE	OP. CODE
1000	16
1001	85 D8
1003	18
1004	8D E0
1006	16
1007	85 D8
1009	18
100A	8D E2
100C	7C 05
100E	84 01 00
1011	74 0F
1013	85 E0
1015	09
1016	9D E2
1018	6C 06
101A	85 E0
101C	2C
101D	6C F7
101F	1A
1020	08
1021	1C
1022	8D D8
1024	13
1025	10
1026	00
1027	00
1028	74 D6

Beispiel: $3^4 = \underbrace{3 \cdot 3 \cdot 3 \cdot 3}_{4 \text{ mal}} = 81$

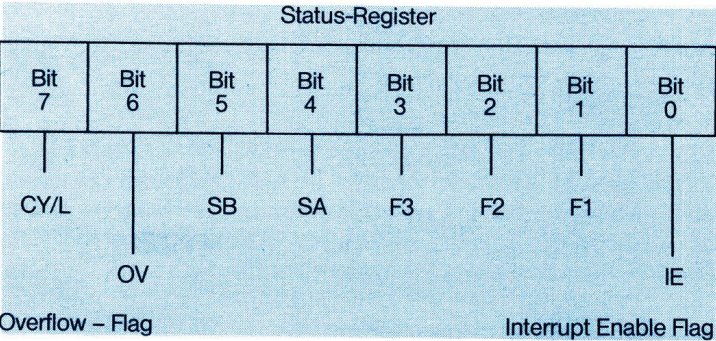
Die 3 heißt Basis oder Grundzahl, die 4 heißt Exponent oder Hochzahl, 3^4 oder 81 ist die Potenz. Das Ergebnis kann bei diesem Programm wieder maximal 9999 sein. Die größte Quadratzahl, die berechnet werden kann, ist $99^2 = 9801$. Die größte Dreierpotenz ist $3^8 = 6561$, die größte Zweierpotenz ist $2^{13} = 8192$ (vgl. Spiel 9, Abschnitt 1.3.9). Bei der Potenzrechnung hat man festgelegt, daß das Potenzieren mit 0 das Ergebnis 1 liefert: z. B.: $25^0 = 1$.

- 6.1 Die Befehle sind nur im Operationscode angegeben. Die Befehle sollen in den mnemonischen Code übersetzt werden.
- 6.2 Der Programmablaufplan soll gezeichnet werden.

17. Ergänzungen zum Status-Register

17.1 Das Overflow-Flag

In Abschnitt 12.1 haben wir sechs Bits des Status-Registers bezeichnet: CY/L-Flag, Sense B, Sense A, Flag 1, Flag 2 und Flag 3. Mit den beiden weiteren Flags wollen wir uns in diesem Kapitel beschäftigen.



Das Overflow-Flag OV hat für unsere Programme keine wesentliche Bedeutung. Dieser Abschnitt kann überschlagen werden. Er ist in dieses Anleitungsbuch aufgenommen worden, weil alle acht Bits des Status-Registers behandelt werden sollten. Die entscheidenden Ergänzungen zum Status-Register folgen ab Abschnitt 17.2. Das Overflow-Flag (engl. overflow – überfließen) wird nur bei arithmetischen Operationen (ADD, SUB, MPY, DIV) gesetzt oder gelöscht. Es spielt nur dann eine Rolle, wenn mit Vorzeichen behafteten Zahlen gerechnet wird (höchstwertiges Bit = 1: negative Zahl; höchstwertiges Bit = 0: positive Zahl; vgl. Abschnitt 10.3). Das Overflow-Flag zeigt das Überschreiten der Grenze zwischen $7F_{16}$ ($= +127_{10}$) und 80_{16} ($= -128_{10}$) bei arithmetischen Operationen an. Wir wollen diese Aussage mit einem Additionsprogramm näher untersuchen.

INR.	MARKE	INMEM. CODE	ADR.	OP. CODE	BEMERKUNGEN
1.	BEGINN	AND S,=0	1000	139 00	DIE LEUCHTDIODEN WERDEN GEBENENFALLS GELOESCHT; <S>:=00
2.		ICALL EIN 2ST	1002	116	INGABE DER 1. ZAHL
3.		ILD A,ZWSP	1003	1C5 D8	
4.		IST A,1.ZAHL	1005	1CD E0	
5.		ICALL EIN 2ST	1007	116	INGABE DER 2. ZAHL
6.		ILD A,ZWSP	1008	1C5 D8	
7.		IST A,2.ZAHL	100A	1CD E1	
8.		ILD A,1.ZAHL	100C	1C5 E0	
9.		ADD A,2.ZAHL	100E	1F5 E1	SUMME:=1.ZAHL+2.ZAHL
10.		IST A,ZWSP	1010	1CD D8	
11.		ILD A,S	1012	106	<A>:=<S>
12.		AND A,=0C0	1013	1D4 C0	MARKIEREN VON BIT 7 UND 6
13.		ISR A	1015	13C	
14.		ISR A	1016	13C	
15.		ISR A	1017	13C	
16.		ISR A	1018	13C	
17.		ILD S,A	1019	107	<F3>:=<CY/L>; <F2>:=<OV>
18.		ICALL BLANK	101A	112	
19.		ICALL UEB 2	101B	115	
20.		ICALL ANZ EIN	101C	110	ANZEIGE
21.		INOP	101D	100	
22.		INOP	101E	100	
23.		IBRA BEGINN	101F	174 DF	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZWSP	1FFD8	ZWISCHENSPEICHER
1. ZAHL	1FFD9	
2. ZAHL	1FFE0	
	1FFE1	

Wie der Inhalt des CY/L-Flags an der roten Leuchtdiode angezeigt wird, haben wir im Abschnitt 15.4 untersucht. Da im 11. und 12. Befehl Bit 7 und Bit 6 des Status-Registers ausmaskiert werden, wird hier zusätzlich der Wert des Overflow-Flags an der gelben Leuchtdiode angezeigt.

In unserem Programm könnte nach Eingabe der zweiten Zahl sofort die erste Zahl aus dem Speicher FFE0 addiert werden. Dann würde das Programm aber nicht so leicht in ein Subtraktionsprogramm zu ändern sein. Der CALL UEB 2 darf nicht unmittelbar hinter dem 10. Befehl (ST A, ZWSP) stehen. Bei diesem Call wird der Inhalt des Status-Registers verändert. Im 11. Befehl soll aber der Inhalt des Status-Registers in den Akku geladen werden, der beim Additionsbefehl entstanden ist. Nun wieder zum Overflow-Flag.

Bei den folgenden Beispielen, die wir mit unserem Programm überprüfen können, sind neben den Summanden und der Summe die Werte angegeben, die das CY/L- und das OV-Flag nach Ausführung der Addition haben.

	1. Zahl	2. Zahl	Summe	(CY/L)	(OV)
1.	18	27	3F	0	0
2.	70	30	A0	0	1
3.	E8	38	(1) 20	1	0
4.	90	90	(1) 20	1	1

Die Bedeutung des CY/L-Flags ist uns bekannt. Das OV-Flag wird gesetzt (vgl. 2. und 4. Beispiel),
– wenn die Summe zweier positiver Zahlen (zwischen 00 und 7F) negativ wird (zwischen 80 und FF),
– wenn die Summe zweier negativer Zahlen (zwischen 80 und FF) positiv wird (zwischen 00 und 7F).

Technisch (hardwaremäßig) wird das Setzen des OV-Flags folgendermaßen realisiert: Das OV-Flag wird gesetzt, wenn bei Addieren der Übertrag von Bit 6 nach Bit 7 und der Übertrag von Bit 7 zum CY/L-Register unterschiedlich sind. Diese Aussage läßt sich im Dualsystem überprüfen. Ein Beispiel soll genügen:

$$\begin{array}{r} 70_{16} = 01110000 \\ 30_{16} = 00110000 \\ \hline 01110000 \\ 00110000 \\ \hline 10100000 \end{array} +$$

Übertrag von Bit 6 nach Bit 7: $\bar{U}_6 = 1$,
Übertrag von Bit 7 zum CY/L-Register: $\bar{U}_7 = 0$.
Das Overflow-Flag OV wird gesetzt, da \bar{U}_6 und \bar{U}_7 unterschiedlich sind.
Entsprechendes gilt für das Overflow-Flag bei der Addition vierstelliger Hexadezimalzahlen. Hier gilt:
positive Zahlen sind: 0000, ..., 7FFF,
negative Zahlen sind: 8000, ..., FFFF.
Das Overflow-Flag wird gesetzt,
– wenn die Summe zweier positiver Zahlen negativ wird,
– wenn die Summe zweier negativer Zahlen positiv wird, oder anders ausgedrückt,
– wenn \bar{U}_4 (Übertrag von Bit 14 nach Bit 15) und \bar{U}_{15} (Übertrag von Bit 15 zum CY/L-Register) unterschiedlich sind.
Ersetzen wir bei unserem Programm den Inhalt F5 im Speicher 100E durch FD, dann können wir das Verhalten des OV-Flags bei der Subtraktion untersuchen.

Das OV-Flag wird gesetzt,

- wenn der Minuend (Zahl, von der subtrahiert wird) positiv, der Subtrahend (Zahl, die subtrahiert wird) und die Differenz negativ sind,
- wenn der Minuend negativ (zwischen 80 und FF), der Subtrahend und die Differenz positiv (zwischen 00 und 7F) sind.

Minuend	Subtrahend	Differenz	(CY/L)	(OV)
70	90	E0 (– 100)	0	1
90	70	20	1	1

Die Erklärung für das Verhalten des OV-Flags bei der Subtraktion ergibt sich zwangsläufig aus den Feststellungen für die Addition. Bei der Subtraktion wird ja das Komplement des Subtrahenden gebildet, dann wird addiert.

17.2 Endlos-Schleife mit Ausgang

Im Abschnitt 10.3 hatten wir im Beispiel 4 einen Sprungbefehl angegeben, durch den eine Endlos-Schleife programmiert werden kann. Nach dem Sprungbefehl mit dem Operationscode 74 FE folgt wieder derselbe Sprungbefehl, usw. Wir hatten festgestellt, daß wir den Mikroprozessor nur mit Hilfe der **RS**-Taste oder durch Abschalten der Spannungsversorgung dazu bewegen können, diese Endlos-Schleife zu verlassen.

Es gibt für das Verlassen einer solchen Endlos-Schleife eine weitere, recht interessante Möglichkeit. Es gibt aus dieser Schleife einen Ausgang (oder genauer zwei Ausgänge).

Bei der Eingabe der folgenden vier Befehle müssen wir genau die angegebenen Adressen beachten.

INR.	MARKE	INMEM. CODE	ADR.	OP. CODE
1.	BEGINN	JOR S,=01	1000	3B 01
2.	ENDLOS	BRA ENDLOS	1002	74 FE
40.		ICALL TEST	1300	1E
180.		ICALL TEST	1380	1E

Wenn wir dieses Programm bei Adresse 1000 starten, erlischt die Anzeige. Es wird ständig der Sprungbefehl (2. Befehl) ausgeführt. Das Programm läuft in der oben beschriebenen Endlos-Schleife. Betätigen wir jetzt die Taste **SA**, so wird die Schleife verlassen. Es wird der Befehl bearbeitet, der bei Adresse 1300 steht. Entsprechend können wir nach einem erneuten Start des Programms ab Adresse 1000 (**RS**, **A↔D**, **A↔D**, **RUN**) mit der Taste **SB** erreichen, daß der Befehl bei Adresse 1380 bearbeitet wird. Die Endlos-Schleife (Adresse 1002 und 1003) könnte in diesem Zusammenhang als Warte-Schleife bezeichnet werden. Der Mikroprozessor wartet auf die Betätigung der Tasten **SA** oder **SB**. Statt der Warte-Schleife hätten wir ein beliebiges Programm schreiben können. Bei Betätigung einer Sense-Taste wird dieses Programm verlassen.

Es geht weiter bei Adresse 1300 oder bei Adresse 1380. Wir wollen festhalten, daß hier nicht ein Jump- oder Branch-Befehl (JMP, JSR, BRA, BP, BZ, BNZ) im Programm selbst den Sprung bewirkt, sondern ein Signal von außen. Man spricht vom Interrupt-Verfahren (engl. interrupt – Unterbrechung). Der Mikroprozessor findet die Adressen 1300 und 1380 im ROM.

ADRESSE	OP. CODE
0000	00
0001	24 09 00
0004	24 FF 12
0007	24 7F 13
000A	08
000E	20 67 09

Wenn der Mikroprozessor eingeschaltet oder mit der **RS**-Taste neu gestartet wird, führt er einen Sprung (24 09 00) zur Adresse 00 09 aus und beginnt das Betriebsprogramm ab Adresse 000A. Beim Interrupt A (Taste **SA**) erfolgt ein Sprung zur Adresse 12FF. Das Programm wird mit dem Befehl im Speicher 12FF + 1 = 1300 fortgesetzt. Beim Interrupt B (Taste **SB**) wird der Sprung (24 7F 13) ausgeführt. Es wird der Befehl ausgeführt, der im Speicher 137F + 1 = 1380 beginnt.

Obwohl der Mikroprozessor – wie wir gesehen haben – die Adressen 1300, bzw. 1380 mit Hilfe von Jump-Befehlen findet, merkt er sich bei einem Interrupt die Stelle, wo er vorher beschäftigt war.

Wenn wir unser kleines Testprogramm mit **RS**, **A↔D**, **A↔D**, **RUN** erneut starten und mit **SA** einen Interrupt verursachen, können wir uns davon überzeugen: **CPU**, **1**: (SP) = FFFE; **A↔D**, **A↔D**, **F**, **F**, **F**, **F**: (FFFF) = 10; **ME –** · (FFFF) = 01. Der Stackpointer ist also um 2 erniedrigt worden. Die Adresse 1001 steht auf dem Stack.

Ein Interrupt hat also Ähnlichkeit mit einem Sprung zum Unterprogramm, nur daß der Sprung nicht durch einen JSR-Befehl, sondern durch ein Signal an Sense A oder an Sense B erfolgt.

Die Programmteile, die ab Adresse 1300 und ab Adresse 1380 im RAM stehen, könnten also durch Return-Befehle abgeschlossen werden. Dann kämen wir nach der Unterbrechung wieder ins ursprüngliche Programm zurück. Wir wollen das probieren:

INR.	MARKE	INMEM. CODE	ADR.	OP. CODE
1.	BEGINN	JOR S,=01	1000	3B 01
2.	ENDLOS	BRA ENDLOS	1002	74 FE
40.	INTP A	ICALL HALT	1300	1F
41.		IRET	1301	5C
180.	INTP B	ICALL HALT	1380	1F
181.		IRET	1381	5C

Wir starten dieses Testprogramm. Die Anzeige erlischt. Mit **SA** erhalten wir jetzt die Anzeige **1 3 0 1** **5 C**. Der CALL HALT (Adresse 1300) zeigt uns den nächsten Speicher (1301) mit dem Return-Befehl (5C). Wir können unmittelbar mit der **RUN**-Taste fortsetzen und sind wieder in der Warteschleife. Doch jetzt ist ein weiterer Interrupt nicht mehr möglich. Der Interrupt ist nicht wieder freigegeben worden.

Sehen wir noch einmal in unser Programm. Im ersten Befehl OR S, = 01 wird Bit 0 im Status-Register gesetzt. Dieses Bit im Status-Register heit Interrupt Enable Flag IE (engl. enable – ermglichen). Ein Interrupt ist nur mglich bei (IE) = 1. Bei (IE) = 0 kann kein Interrupt ausgefhrt werden. Bei der Ausfhrung eines Interrupts wird automatisch das IE-Flag zurckgesetzt. Wenn wir nach einem Interrupt einen weiteren Interrupt zulassen wollen, mssen wir per Programm das IE-Flag wieder setzen. Es gibt zwei Mglichkeiten:

A.

NR.	MARKE	INMEM.CODE	ADR.	OP.CODE
1.	BEGINN	OR S,=01	1000	3B 01
2.		BRA BEGINN	1002	74 FC
40.	INTP A	CALL HALT	1300	1F
41.		RET	1301	5C
180.	INTP B	CALL HALT	1380	1F
181.		RET	1381	5C

B.

NR.	MARKE	INMEM.CODE	ADR.	OP.CODE
1.	BEGINN	OR S,=01	1000	3B 01
2.	ENDLOS	BRA ENDLOS	1002	74 FE
40.	INTP A	CALL HALT	1300	1F
41.		OR S,=01	1301	3B 01
42.		RET	1303	5C
180.	INTP B	CALL HALT	1380	1F
181.		OR S,=01	1381	3B 01
182.		RET	1383	5C

NR.	MARKE	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	LD A,=0	1000	34 00	HAUPTPROGRAMM
2.		ST A,ZAHL	1002	CD E0	ZAHL:=0
3.		OR S,=01	1004	3B 01	INTERRUPT-FREIGABE
4.	SCHL.	CALL BLANK	1006	12	
5.		LD A,ZAHL	1007	95 E0	ZAHL:=ZAHL+1
6.		ST A,ZWSP	1009	CD 08	
7.		CALL UEB 2	100B	15	
8.		LD A,=080	100C	34 80	
9.		CALL ANZEIGE	100E	11	ANZEIGE DER ZAHL
10.		BRA SCHL.	100F	74 F5	
40.	INTP A	LD EA,=0A	1300	34 0A 00	INTERRUPT-PROGRAMM A
41.		ST EA,ZWSP	1303	8D 08	
42.		CALL UEB 4R	1305	13	
43.		LD EA,=0CE6	1306	34 E6 0C	
44.		ST EA,CST1	1309	8D C1	
45.		LD A,=0	130B	34 00	
46.		CALL ANZEIGE	130D	11	ANZEIGE: "IPA"
47.		CALL BLANK	130E	12	
48.		CALL VERZ	130F	1D	
49.		OR S,=01	1310	3B 01	INTERRUPT-FREIGABE
50.		RET	1312	5C	
180.	INTP B	LD EA,=0B	1380	34 0B 00	INTERRUPT-PROGRAMM B
181.		ST EA,ZWSP	1383	8D 08	
182.		CALL UEB 4R	1385	13	
183.		LD EA,=0CE6	1386	34 E6 0C	
184.		ST EA,CST1	1389	8D C1	
185.		LD A,=0	138B	34 00	
186.		CALL ANZEIGE	138D	11	ANZEIGE: "IPB"
187.		CALL BLANK	138E	12	
188.		CALL VERZ	138F	1D	
189.		OR S,=01	1390	3B 01	INTERRUPT-FREIGABE
190.		RET	1392	5C	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZAHL	0FFEH	
ZWSP	0FFDH	ZWISCHENSPEICHER
	0FFD9	
CST1	0FFC1	CODIERUNG STELLE 1
CST2	0FFC2	CODIERUNG STELLE 2

Bei Mglichkeit A wird der Interrupt im Hauptprogramm freigegeben. Der Befehl OR S, = 01 ist jetzt ein Teil der Warteschleife. Bei Mglichkeit B wird der nchste Interrupt in den Programmen ab Adresse 1300, bzw. 1380 vor dem Rcksprung ins Hauptprogramm freigegeben. Wir knnen uns berzeugen. In beiden Fllen lt sich der Interrupt wiederholen.

Ein Hinweis: Wenn wir im Wechsel die Tasten **RUN** und **SA** bettigen, haben wir eventuell den Eindruck, da der Mikroprozessor auf Bettigung der **RUN**-Taste nicht reagiert. Das liegt dann aber nicht an der **RUN**-sondern an der **SA**-Taste. Wenn die **SA**-Taste prellt, also statt eines Signals zwei liefert, dann fhrt der Mikroprozessor eine Interruptanforderung aus, merkt sich aber, da noch ein weiterer Interrupt gewnscht wurde. Bei Bettigung der **RUN**-Taste wird dann der erste Interrupt abgeschlossen, dann sofort der zweite Interrupt ausgefhrt. Daher ndert sich an der Anzeige nichts. Wir mssen in diesem Fall die **RUN**-Taste mehrfach bettigen.

Falls die **SA**-Taste nicht prellen sollte, knnen wir den gleichen Effekt durch mehrfache Bettigung der **SA**-Taste erreichen.

17.3 Weitere Untersuchung des Interrupt-Verfahrens

Wir wollen das im letzten Abschnitt angesprochene Interrupt-Verfahren weiter untersuchen. Wir werden dazu ein ausfhrliches Testprogramm whlen. Wir werden sowohl das Hauptprogramm als auch die beiden Interrupt-Programme so schreiben, da wir beim Programmablauf erkennen knnen, welchen Programmteil der Mikroprozessor gerade bearbeitet. Im Hauptprogramm ist ein Zhler programmiert, in den Interrupt-Programmen wird „IPA“ oder „IPB“ angezeigt.

Wir knnen zunchst die Aussagen erneut berprfen, die sich ber das Interrupt-Verfahren im letzten Abschnitt ergeben haben.

Mit Hilfe der Tasten **SA** oder **SB** lt sich ein Interrupt erreichen. Wird der Interrupt mit der Taste **SA** aufgerufen, wird das Interrupt-Programm A ab Adresse 1300 aufgerufen. Bei Bettigung der Taste **SB** folgt das Interrupt-Programm B ab Adresse 1380.

Ein Interrupt ist nur mglich, wenn das IE-Flag gesetzt ist. Es wird hier beim ersten Start des Hauptprogramms und jeweils am Ende der Interrupt-Programme gesetzt. Das erneute Setzen des IE-Flags ist erforderlich, da dieses Flag bei jedem Interrupt automatisch gelscht wird.

Nach Bearbeitung eines Interrupt-Programms wird das Hauptprogramm dort fortgesetzt, wo es durch den Interrupt unterbrochen wurde. Das erreichen wir mit dem Return-Befehl am Ende der Interrupt-Programme.

Wir werden eventuell sehen, da bei einer Bettigung einer Sense-Taste das zugehrige Interrupt-Programm zweimal ausgefhrt wird. Das liegt am Prellen der Tasten. Wir haben auch diese Erscheinung im letzten Abschnitt schon erwhnt (vgl. Abschnitt 17.2), ausfhrlicher werden wir das Prellen und die Mglichkeiten zum Entprellen im nchsten Abschnitt behandeln.

Bisher haben wir die Aussagen ber das Interrupt-Verfahren aus dem letzten Abschnitt wiederholt. Unser Programm erlaubt weitere Aussagen.

Der Interrupt wird durch einen H-L-bergang an dem entsprechenden Sense-Eingang des Mikroprozessors ausgelst. Wird z. B. die Taste **SA** gedrckt und festgehalten, so liegt am Mikroprozessor ein High-Signal (etwa 5V). Dadurch wird der Interrupt noch nicht bewirkt. Das Hauptprogramm luft weiter. Wird jetzt die Taste losgelassen, dann ndert sich das Signal am Mikroprozessor-Eingang vom High-Zustand zum Low-Zustand (etwa 0V). Durch diesen bergang vom High- zum Low-Zustand, kurz durch diesen H-L-ber-

gang, wird der Interrupt aufgerufen. Bei dieser Überprüfung kann uns auch das Pellen stören. Wenn die Taste beim Niederdrücken prellt, wird entgegen unseren Erwartungen ein Interrupt ausgelöst. Dann können wir aber die Taste festhalten, bis wieder das Hauptprogramm läuft. Wir stellen weiter fest: Eine Interrupt-Anforderung wird auch registriert, wenn keine Interrupt-Freigabe erfolgt ist. Jede der beiden Interrupt-Anforderungen kann zur Zeit nur einmal gespeichert werden. Wir können z. B. vor dem Start des Hauptprogramms die Tasten **SA** und **SB** betätigen. Wird jetzt das Hauptprogramm gestartet, so werden bei Interrupt-Freigabe (3. Befehl: OR S, = 01) die Interrupt-Programme bearbeitet. Wir sehen dabei gleichzeitig, daß die Interrupt – A – Anforderung über Taste **SA** vorrangig bedient wird. Eine Interrupt-Anforderung wird auch registriert, wenn das Interrupt-Programm gerade bearbeitet wird. Bei der Tastenfolge **SA, SA, SB** werden die Programmteile in folgender Reihenfolge bearbeitet: IPA – IPA – IPB – Hauptprogramm. Bei der Tastenfolge **SB, SB, SA** werden die Programmteile in folgender Reihenfolge bearbeitet: IPB – IPA – IPB – Hauptprogramm. Eine letzte – sehr wichtige – Feststellung: Wenn eine Interrupt-Anforderung vorliegt, dann wird nach Interrupt-Freigabe (OR S, = 01) vor dem Sprung zum Interrupt-Programm noch ein weiterer Befehl ausgeführt. Aus diesem Grund sollte am Ende des Interrupt-Programms nach dem Befehl OR S, = 01 unmittelbar der Return-Befehl folgen. Selbst wenn wir jetzt z. B. bei jeder Bearbeitung des Interrupt-Programms B wieder die Taste **SB** betätigen, wird der Inhalt des Stackpointers nicht immer weiter erniedrigt, denn es folgen der Sprung zum Interrupt-Programm B und der Return-Befehl immer im Wechsel. Anders wäre es, wenn zwischen den Befehlen OR S, = 01 und RET noch ein weiterer Befehl stehen würde:

INR.	MARKE	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
189.		IOR S,=01	1390	13B 01	INTERRUPT-FREIGABE
190.		ICALL TEST	1392	11E	
191.		IRET	1393	15C	

SP →	FFEC	EF
	FFED	02
	FFEE	92
	FFEF	13
	FFF0	EF
	FFF1	02
	FFF2	92
	FFF3	13
	FFF4	EF
	FFF5	02
	FFF6	92
	FFF7	13
	FFF8	EF
	FFF9	02
	FFFA	92
	FFFB	13
	FFFC	94
	FFFD	08
	FFFE	06
	FFFF	10

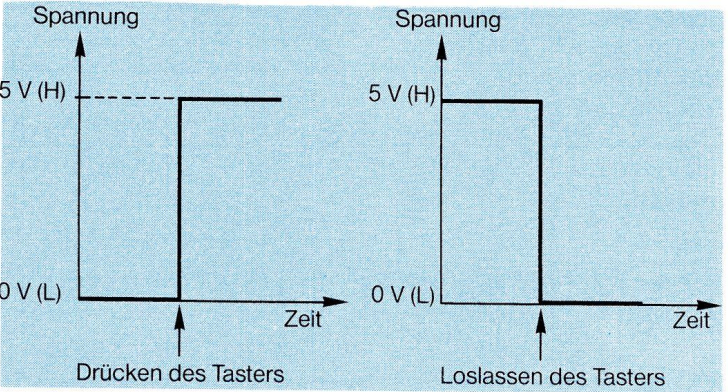
(Adressen)

Wenn nach dieser Programmänderung ständig die Taste **SB** betätigt wird, wird der Stack immer weiter vollgeschrieben. Die nebenstehend angegebenen Stackinhalte sind bei folgender Tastenfolge entstanden: **RS, A↔D, A↔D, SB, RUN, SB, SB, SB, SB**. Die vier letzten Betätigungen der Taste **SB** erfolgten immer bei der Anzeige „IPB“. Mit **CPU, 1** sehen wir den Inhalt des Stackpointers: (SP)=FFEC. Zwanzig Speicherplätze des Stacks sind also mit zehn Adressen beschrieben worden. Wir finden sie durch Betätigung der Tasten **A↔D, A↔D, F, F, F, F, ME–, ME–, ME–, ME–, ...**

Durch Betätigung der **RS**-Taste hatten wir erreicht, daß der Stackpointer auf die Adresse 0000 zeigte. Nach Betätigung der Taste **SB** hatten wir das Hauptprogramm gestartet. Im 3. Befehl wurde die Interrupt-Freigabe erteilt. Der eine Befehl, der vor dem Sprung zum Interrupt-Programm B noch ausgeführt wurde, ist nicht der CALL BLANK (ein Call besteht wie jedes Unterprogramm aus mehreren Einzelbefehlen), sondern nur der Sprung zum CALL BLANK. Auf den Stack wurde zunächst die Adresse 1006 geschrieben (Rücksprungadresse zum Hauptprogramm). Die nächste Adresse auf dem Stack ist 0894 (nach Bearbeitung des Interrupt-Programms müßte der CALL BLANK ausgeführt werden. Vgl. Abschnitt 13.1 und 1. Aufg. von Kapitel 13). Außerdem finden wir auf dem Stack viermal die beiden Adressen 1392 und 02EF im Wechsel. Nach Interrupt-Freigabe im Befehl Nr. 89 wurde jeweils noch der Sprung zum Anfang des CALL TEST ausgeführt (dabei wurde die Adresse 1392 auf den Stack geschrieben). Dann folgte gleich darauf der Sprung zum Interrupt-Programm B (die Anfangsadresse 02EF des Calls wird auf dem Stack notiert).

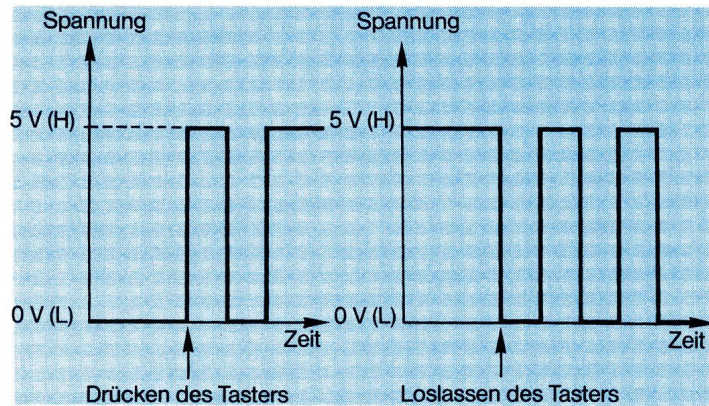
17.4 Entprellen mit Software

Wir haben schon mehrfach das Pellen der Taster erwähnt. Wir wollen hier etwas ausführlicher darauf eingehen. Anschließend wollen wir uns eine Möglichkeit überlegen, wie wir die Nachteile des Prellens per Software unterdrücken können. Bei der Betätigung der Sense-Taster soll die Spannung am Sense-Eingang des Mikroprozessors schlagartig den anderen Wert annehmen.



Hier würde der Mikroprozessor nur beim Loslassen des Tasters eine Interrupt-Anforderung registrieren. In der Realität gibt es bei mechanischen Schaltern häufig ein Pellen. Dann ändert sich die Spannung nicht schlagartig, sondern es gibt einige Spannungsänderungen, bis der gewünschte Spannungswert erreicht ist.

Der Spannungsverlauf sieht z. B. folgendermaßen aus:



Hier würde der Mikroprozessor mehrere Interrupt-Anforderungen registrieren.

Dieses Prellen von mechanischen Tastern läßt sich hardwaremäßig unterdrücken (z. B. mit einem Schmitt-Trigger in Verbindung mit einem Kondensator).

Da wir an der Hardware keine Änderungen vornehmen können, wollen wir es mit der Software versuchen. Um es gleich zu sagen: Wenn der Taster prellt, läßt sich das mit Hilfe der Software nicht ungeschehen machen. Mit der Software können wir nur erreichen, daß das Interrupt-Programm nicht zweimal unmittelbar hintereinander vollständig durchlaufen wird.

INR.	MARKE	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	ILD A,=0	1000	IC4 00	HAUPTPROGRAMM
2.		IST A,ZAHL	1002	ICD E0	ZAHL:=0
3.		IOR S,=01	1004	IB 01	INTERRUPT-FREIGABE
4.	SCHL.	ICALL BLANK	1006	I12	
5.		ILD A,ZAHL	1007	I95 E0	ZAHL:=ZAHL+1
6.		IST A,ZWSP	1009	ICD D8	
7.		ICALL UEB 2	100B	I15	
8.		ILD A,=000	100C	IC4 80	
9.		ICALL ANZEIGE	100E	I11	ANZEIGE DER ZAHL
10.		ILD A,=01	100F	IC4 01	
11.		IST A,FLAG A	1011	ICD EA	IFLAG A:=1
12.		IST A,FLAG B	1013	ICD EB	IFLAG B:=1
13.		IBRA SCHL.	1015	I74 EF	
40.	INTP A	ILD A,FLAG A	1300	IC5 EA	INTERRUPT-PROGRAMM A
41.		IBNZ ANZ A	1302	I7C 03	ISPRUNG, WENN FLAG A = 1
42.		IOR S,=01	1304	IB 01	INTERRUPT-FREIGABE
43.		IRET	1306	I5C	ZURUECK ZUM HAUPTPROGRAMM
44.	ANZ A	ILD A,=0	1307	IC4 00	
45.		IST A,FLAG A	1309	ICD EA	IFLAG A:=0
46.		ILD EA,=0A	130B	I84 0A 00	
47.		IBRA WEITER	130E	I74 30	
60.	WEITER	IST EA,ZWSP	1340	ID D8	
61.		ICALL UEB 4R	1342	I13	
62.		ILD EA,=0CE6	1343	I84 E6 0C	
63.		IST EA,CST1	1346	ID C1	
64.		ILD A,=0	1348	IC4 00	
65.		ICALL ANZEIGE	134A	I11	ANZEIGE: "IPA" ODER "IPB"
66.		ICALL BLANK	134B	I12	
67.		ICALL VERZ	134C	I1D	
68.		IOR S,=01	134D	IB 01	INTERRUPT-FREIGABE
69.		IRET	134F	I5C	ZURUECK ZUM HAUPTPROGRAMM
80.	INTP B	ILD A,FLAG B	1380	IC5 EB	INTERRUPT-PROGRAMM B
81.		IBNZ ANZ B	1382	I7C 03	ISPRUNG, WENN FLAG B = 1
82.		IOR S,=01	1384	IB 01	INTERRUPT-FREIGABE
83.		IRET	1386	I5C	ZURUECK ZUM HAUPTPROGRAMM
84.	ANZ B	ILD A,=0	1387	IC4 00	
85.		IST A,FLAG B	1389	ICD EB	IFLAG B:=0
86.		ILD EA,=0B	138B	I84 0B 00	
87.		IBRA WEITER	138E	I74 B0	

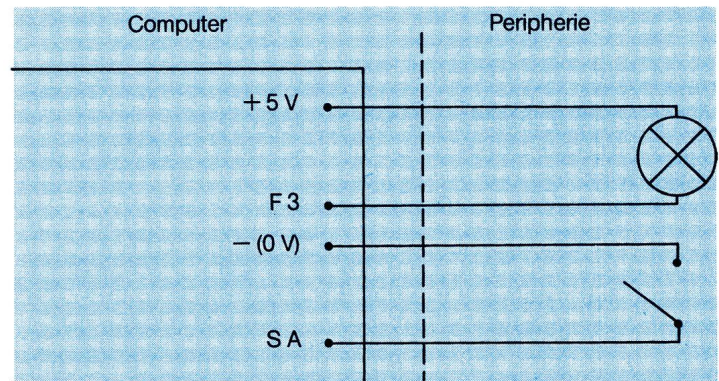
DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZAHL	IFFE0	
ZWSP	IFFD8	ZWISCHENSPEICHER
	IFFD9	
CST1	IFFC1	CODIERUNG STELLE 1
CST2	IFFC2	CODIERUNG STELLE 2
FLAG A	IFFEA	
FLAG B	IFFEB	

Wir erfinden zwei Flags: Flag A (Speicher FFEA) und Flag B (Speicher FFEB). Diese Flags setzen wir im Hauptprogramm ((FFEA)= 01; (FFEB)= 01). Beim ersten Aufruf des Interrupt-Programms löschen wir das entsprechende Flag ((FFEA)= 00, bzw. (FFEB)= 00). Wird dasselbe Interrupt-Programm unmittelbar danach wieder aufgerufen, so wird der Hauptteil dieses Interrupt-Programms nicht bearbeitet. Wir wollen noch einmal festhalten: Beim Prellen der Sense-Taster werden eventuell mehrere Interrupt-Anforderungen registriert. Sie werden auch alle erfüllt. Wir merken es aber nicht. Der Nachteil ist jetzt, daß wir das volle Interrupt-Programm erst wieder aufrufen können, wenn im Hauptprogramm das Flag wieder gesetzt wurde.

Das Programm hat viel Ähnlichkeit mit dem aus Abschnitt 17.3. Außer den besprochenen Änderungen haben wir noch die gleichen Teile beider Interrupt-Programme herausgenommen und nur einmal ab Adresse 1340 programmiert.

17.5 Ein Interrupt von der Peripherie

Wir schließen Glühlampe und Tastschalter an der rechten Federleiste unserer Computer-Platine an, wie wir es im Abschnitt 3.1 gemacht haben.



Das folgende Programm soll uns zeigen, daß ein Interrupt auch von der Peripherie angefordert werden kann. Wir werden nicht sehr überrascht sein, wenn wir uns noch einmal den Schaltplan im Abschnitt 3.7.2 ansehen. Trotzdem kann uns dieses Programm veranschaulichen, wie ein Interrupt bei möglichen Anwendungen eingesetzt wird.

Der Computer ist mit irgendwelchen Dingen beschäftigt, die aber nicht unbedingt jetzt sofort erledigt werden müssen. Mit Hilfe eines Interrupts wird diese Tätigkeit unterbrochen, damit zunächst eine unaufschiebbare Aufgabe erfüllt werden kann. Wir können uns etwa vorstellen, daß der Computer in einem Warenhaus die Kassen betreut (Kassenbeleg-Ausgabe, Überprüfung des Lagerbestandes usw.). Wenn jetzt an einer Stelle im Warenhaus ein Alarmknopf betätigt wird, muß der Computer die Betreuung der Kassen unterbrechen. Er überprüft die Ursache für den Interrupt und entscheidet, welches Alarmsignal gegeben werden muß. Nach der eventuellen Auslösung des Alarms kann sich der Computer wieder den Kassen zuwenden.

Unser Programm ist hier natürlich nur ein recht einfaches Modell. Unser Computer zählt. Wenn über den extern angeschlossenen Tastschalter ein Interrupt angefordert wird, sorgt der Computer dafür, daß sich der Leuchtzustand der extern angeschlossenen Glühlampe verändert. Bemerkenswert ist, daß die Bearbeitung des Interrupts sehr schnell erledigt wird. Wir merken am Zählprogramm gar nicht, daß der Computer zwischendurch noch eine andere Aufgabe bearbeitet hat.

NR.	MARKE	MMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	ILD A,=0	1000	IC4 00	HAUPTPROGRAMM
2.		IST A,ZAHL	1002	ICD E0	ZAHL:=0
3.		OR S,=01	1004	13B 01	INTERRUPT-FREIGABE
4.	SCHL.	ICALL BLANK	1006	112	
5.		ILD A,ZAHL	1007	195 E0	ZAHL:=ZAHL+1
6.		IST A,ZWSP	1009	ICD 08	
7.		ICALL UEB 2	100B	115	
8.		ILD A,=0FF	100C	IC4 FF	
9.		IST A,FLAG A	100E	ICD EA	FLAG A:=FF
10.		ICALL ANZEIGE	1010	111	ANZEIGE DEK ZAHL
11.		IBRA SCHL.	1011	174 F3	
140.	INTP A	ILD A,FLAG A	1300	IC5 EA	INTERRUPT-PROGRAMM A
141.		IBZ WEITER	1302	16C 00	SPRUNG, WENN FLAG A = 0
142.		ILD A,S	1304	106	
143.		XOR A,=08	1305	1E4 08	<F3> RENDERN
144.		ILD S,A	1307	107	
145.		ILD A,=0	1308	IC4 00	
146.		IST A,FLAG H	130A	ICD EA	FLAG A:=0
147.	WEITER	OR S,=01	130C	13B 01	INTERRUPT-FREIGABE
148.		IRET	130E	15C	
180.	INTP B	OR S,=01	1380	13B 01	INTERRUPT-PROGRAMM B
181.		IRET	1382	15C	INTERRUPT-FREIGABE

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZAHL	1FE0	
ZWSP	1FFD8	ZWISCHENSPEICHER
FLAG A	1FFFA	

Ein Interrupt-Programm B ist hier nicht erforderlich. Die beiden Befehle in den Speichern 1380 bis 1382 verhindern lediglich, daß bei einer versehentlichen Betätigung der Taste **SB** etwas Unvorhergesehenes passiert. Das Entprellen der externen Taste ist hier sicher erforderlich, denn beim Prellen dieser Taste würden sonst zwei Interrupt-Anforderungen ausgeführt werden. Zweifache Änderung des Leuchtzustandes der Glühlampe würde aber den Leuchtzustand der Glühlampe gar nicht ändern. Wenn wir diese Aussage überprüfen wollen, können wir das Displacement 08 im Speicher 1303 durch 00 ersetzen. Der volle Durchlauf durch das Interrupt-Programm A ist nur bei Flag A = FF möglich. Flag A bekommt diesen Wert im Hauptprogramm. Vor einer weiteren Leuchtzustandsänderung der Glühlampe muß bis zur Anzeige der nächsten Zahl gewartet werden.

17.6 Ein gefälschter Würfel

Das folgende Programm soll ein Beispiel dafür sein, daß das IE-Flag während eines Programm-Durchlaufs im Wechsel gesetzt und gelöscht werden kann. Ein Interrupt ist dann nur in bestimmten Programmabschnitten möglich. Zunächst das Programm:

Nach Programmstart erlischt die Anzeige. Durch Betätigung der Taste **SA** können wir würfeln. Es erscheint dann eine der Ziffern 1, 2, ..., 6 für eine kurze Zeit. Nach dem Erlöschen der Anzeige kann erneut gewürfelt werden.

NR.	MARKE	MMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	ICALL BLANK	1000	112	
2.	WDH.1	ILD A,ZAHL	1001	19D E0	ZAHL:=ZAHL-1
3.		AND A,=07	1003	1D4 07	MASKIEREN: BIT 2 BIS BIT 0
4.		IST A,ZWSP	1005	ICD 08	
5.		XOR A,=06	1007	1E4 06	
6.		IBNZ WDH.1	1009	17C F6	KEIN SPRUNG, WENN <A>=06
7.		OR S,=01	100B	13B 01	INTERRUPT-FREIGABE
8.	WDH.2	ILD A,ZAHL	100D	19D E0	ZAHL:=ZAHL-1
9.		AND A,=07	100F	1D4 07	MASKIEREN: BIT 2 BIS BIT 0
10.		IST A,ZWSP	1011	ICD 08	
11.		XOR A,=01	1013	1E4 01	
12.		IBNZ WDH.2	1015	17C F6	KEIN SPRUNG, WENN <A>=01
13.		AND S,=0FE	1017	139 FE	INTERRUPT-UNTERDRUECKUNG
14.		IBRA WDH.1	1019	174 E6	
140.	INTP A	ICALL UEB 2	1300	115	INTERRUPT-PROGRAMM A
141.		ILD A,=0	1301	IC4 00	
142.		IST A,CST5	1303	ICD C5	<CST5>:=00
143.		ICALL ANZEIGE	1305	111	WUERFEL-ANZEIGE
144.		OR S,=01	1306	13B 01	INTERRUPT-FREIGABE
145.		IRET	1308	15C	
180.	INTP B	IRET	1380	15C	INTERRUPT-PROGRAMM B

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZAHL	1FE0	
ZWSP	1FFD8	ZWISCHENSPEICHER
CST5	1FFC5	CODIERUNG STELLE 5

Dieser Würfel ist „gefälscht“. Er würfelt die 6 häufiger als die anderen Zahlen. Wir können uns davon überzeugen. In der folgenden Strichliste ist ein mögliches Ergebnis bei hundert Würfelvorgängen angegeben:

	1	2	3	4	5	6
	+++	+++	+++	+++	+++	+++
	+++	+++	+++	+++	1	+++
		1	+++	1		+++
						+++
						+++
						+++
						+++
Anzahl	13	11	17	11	6	42

Vor einem Würfelvorgang können wir nicht voraussagen, welche Zahl erscheint. Wir können aber voraussagen, daß bei häufigem Würfeln die 6 etwa dreimal so oft erscheint wie jede der anderen Zahlen.

Die Erklärung finden wir im Programm. Der Inhalt des Speichers FFE0 wird dekrementiert (im 2. und 8. Befehl). Das Ergebnis wird mit 07 maskiert. In den Speicher FFD8 kommen nacheinander die Zahlen 7, 6, 5, ..., 1, 0. Der Interrupt ist aber nicht freigegeben, wenn 7 oder 0 im Speicher FFD8 steht. Da die Würfelanzeige im Interrupt-Programm geschieht, können 7 und 0 auch nicht gewürfelt werden. Falls die Interrupt-Anforderung erfolgt, wenn unser Zähler bei 0 oder 7 ist, wird der Inhalt von Speicher FFD8 erst noch bis 6 weitergezählt, dann wird der Interrupt ausgeführt, und es erscheint eine 6.

Auf zwei Dinge soll noch hingewiesen werden. Den Befehl zur Interrupt-Freigabe (OR S, = 01) haben wir schon häufig eingesetzt. Die Interrupt-Unterdrückung geschieht mit dem 13. Befehl: AND S, = FE. Dadurch wird Bit 0 im Status-Register in jedem Fall gelöscht, ganz egal, was sonst im Status-Register steht.

Die Interrupt-Freigabe muß im Hauptprogramm stehen, um das Würfeln der sechs gewünschten Augenzahlen zu er-

möglichen. Dieser Befehl steht auch im Interrupt-Programm. Beim Pressen des Tasters wird so die gleiche Anzeige wiederholt. Die Anzeige bleibt lediglich länger stehen. Andernfalls würde auf eine Anzeige 1, . . . , 5 unmittelbar die Anzeige 6 folgen. Der zweite Interrupt, der durch das Pressen angefordert wurde, wird erst wieder im 7. Befehl freigegeben, wenn der Inhalt von Speicher FFD8 wieder 6 ist.

17.7 Aufgaben zu Kapitel 17

1.1 In dem folgenden Zählprogramm werden nach jeder Addition die Inhalte vom CY/L- und OV-Flag an der roten und der gelben Leuchtdiode angezeigt. Wann leuchtet die rote, wann die gelbe Leuchtdiode? Warum?

NR.	MARKE	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	CALL BLANK	1000	12	
2.		ILD A,=0	1001	C4 00	
3.		IST A,ZAHL	1003	CD D8	ZAHL:=0
4.	WEITER	ILD A,ZAHL	1005	C5 D8	
5.		IADD A,=010	1007	F4 10	ZAHL:=ZAHL+010
6.		IST A,ZAHL	1009	CD D8	
7.		ILD A,S	100B	06	
8.		IAND A,=0CA	100C	D4 CA	
9.		ISR A	100E	13C	
10.		ISR A	100F	13C	
11.		ISR A	1010	13C	
12.		ISR A	1011	13C	
13.		ILD S,A	1012	07	(F3):=(CY/L); (F2):=(OV)
14.		ICALL UEB 2	1013	15	
15.		ILD A,=040	1014	C4 40	
16.		ICALL ANZEIGE	1016	11	ANZEIGE
17.		IBRA WEITER	1017	74 EC	

DATENSPEICHER	BEZEICHNUNG	ADR.	BEMERKUNGEN
	ZAHL	FFD8	(ZWISCHENSPEICHER)

- 1.2 Wann leuchtet die rote, wann die gelbe Leuchtdiode, wenn der 5. Befehl durch den Subtraktionsbefehl (SUB A, = 010; FC 10) ersetzt wird? Warum?
2. Zum Programm von Abschnitt 17.4:
- 2.1 Es sollen die Rechnungen angegeben werden, mit denen die Displacements im 47. und 87. Befehl bestimmt werden.
- 2.2 Diese beiden Sprung-Befehle sollen durch Jump-Befehle ersetzt werden. Wie müßten die Operationscodes für diese beiden Befehle lauten?
3. Das Programm von Abschnitt 17.5 ist hier so umgeschrieben, daß die externe Taste ganz normal abgefragt wird. Es wird hier also kein Interrupt ausgelöst.
- 3.1 Es ist nur der Operationscode angegeben. Das Programm soll analysiert werden.

ADRESSE	OP.CODE
1000	C4 00
1002	CD E0
1004	12
1005	95 E0
1007	CD D8
1009	15
100A	C4 FF
100C	11
100D	06
100E	D4 10
1010	6C F2
1012	06
1013	E4 00
1015	07
1016	74 EC

- 3.2 Es soll ein Programmablaufplan gezeichnet werden.
- 3.3 Welche Unterschiede lassen sich zwischen diesen beiden Programmen feststellen? Welche Vorteile hat das Interrupt-Verfahren?
4. Wenn in dem Programm von Abschnitt 17.6 die folgenden zwei Befehle geändert werden, kommt es beim Pressen der Taste SA vor, daß unmittelbar nach Anzeige einer gewürfelten Zahl (1, 2, . . . , 6) die 0 angezeigt wird.

NR.	MARKE	INMEM.CODE	ADR.	OP.CODE
12.		BNZ 7.BEF	1015	7C F4
14.		IBRA 45.BEF	1306	74 00

Wie ist diese Erscheinung zu erklären?

18. Programme – Programme – Programme

18.1 Die Programme aus dem 2. und 3. Kapitel

Im 2. und 3. Kapitel haben wir einige Programme erprobt, ohne auf die einzelnen Befehle näher einzugehen. Diese Programme sollen hier noch einmal aufgeführt werden, wobei die einzelnen Befehle mit mnemonischem Code und vollständigem Operationscode angegeben sind.

18.1.1 Das Programm aus dem 2. Kapitel (vgl. Abschnitt 2.2 und 2.3)

NR.	MARKE	MNEM. CODE	ADR.	OP. CODE	BEMERKUNGEN
1.	BEGINN	ILD A,=02	1000	1C4 02	
2.		ILD S,A	1002	107	<S>:=02; <F1>:=1
3.	TASTE	ILD A,S	1003	106	
4.		IAND A,=010	1003	1D4 10	
5.		IBZ BEGINN	1006	16C F8	SPRUNG, WENN <SA>=0
6.		ILD A,=08	1008	1C4 08	
7.		ILD S,A	100A	107	<S>:=08; <F3>:=1
8.		IBRA TASTE	100B	174 F6	

18.1.2 Das Programm „Die Glühlampe kann auch blinken“ (vgl. Abschnitt 3.2)

NR.	MARKE	MNEM. CODE	ADR.	OP. CODE	BEMERKUNGEN
1.	BEGINN	IAND S,=0	1000	139 00	<F1>:=0
2.	AUF	ILD A,S	1002	106	
3.		IAND A,=010	1003	1D4 10	SCHLEIFE, SOLANGE DER TASTER NICHT BETRÄTIGT WIRD
4.		IBZ AUF	1005	16C FB	
5.	ZU	ILD A,S	1007	106	
6.		IAND A,=010	1008	1D4 10	SCHLEIFE, SOLANGE DER TASTER BETRÄTIGT WIRD
7.		IBNZ ZU	100A	17C FB	
8.	WEITER	ILD A,=10	100C	1C4 0A	
9.		IST A,ZAHL	100E	1CD E0	ZAHL:=10
10.	BLINK	ILD A,S	1010	106	
11.		IXOR A,=02	1011	1E4 02	<F1> WIRD VERÄNDERT
12.		ILD S,A	1013	107	
13.		ILD EA,=0	1014	184 00 00	
14.		ICALL VERZ	1017	11D	
15.		ILD A,ZAHL	1019	19D E0	ZAHL:=ZAHL-1
16.		IBNZ BLINK	101A	17C F4	SPRUNG, WENN ZAHL > 0
17.		IBRA BEGINN	101C	174 E2	SPRUNG, WENN ZAHL = 0

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZAHL	1FFE0	

18.1.3 Das Programm „Ein Morseapparat“ (vgl. Abschnitt 3.3)

NR.	MARKE	MNEM. CODE	ADR.	OP. CODE	BEMERKUNGEN
1.	BEGINN	IAND S,=0	1000	139 00	<F1>:=0
2.	TASTE	ILD A,S	1002	106	
3.		IAND A,=010	1003	1D4 10	
4.		IBZ BEGINN	1005	16C F9	SPRUNG, WENN <SA>=0
5.	TON	ILD A,S	1007	106	
6.		IXOR A,=02	1008	1E4 02	<F1> WIRD VERÄNDERT
7.		ILD S,A	100A	107	
8.		ILD EA,=020	100B	184 20 00	<STATT "20" AUCH "01" ... "FF">
9.		ICALL VERZ	100E	11D	
10.		IBRA TASTE	100F	174 F1	

18.1.4 Das Testprogramm für die Lichtschranke (vgl. Abschnitt 3.4.1)

NR.	MARKE	MNEM. CODE	ADR.	OP. CODE	BEMERKUNGEN
1.	BEGINN	ILD A,S	1000	106	
2.		IAND A,=010	1001	1D4 10	
3.		ISR A	1003	13C	
4.		ILD S,A	1004	107	<F3>:=<SA>
5.		IBRA BEGINN	1005	174 F9	

18.1.5 Das Programm „Diebstahlsicherung“ (vgl. Abschnitt 3.4.2)

NR.	MARKE	MNEM. CODE	ADR.	OP. CODE	BEMERKUNGEN
1.	BEGINN	ILD A,S	1000	106	
2.		IAND A,=010	1001	1D4 10	
3.		IBNZ BEGINN	1003	17C FB	
4.		IBZ SIRENE	1005	124 51 04	SPRUNG ZUM SPIEL "SIRENE", WENN LICHTSCHRANKE DUNKEL

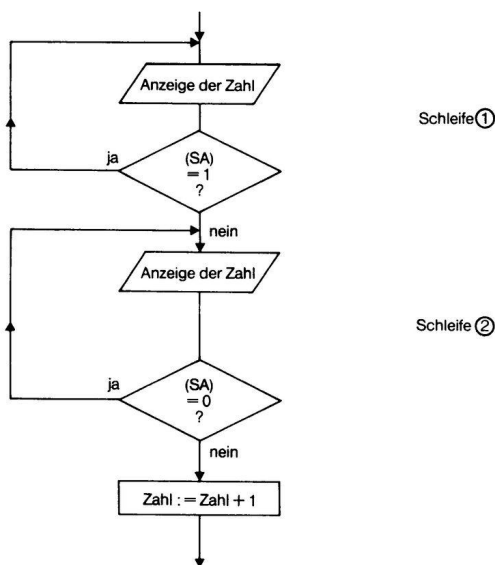
WEITERE MARKE, BEZEICHNUNG	ADR.	BEMERKUNGEN
SIRENE	0452	BEI DIESER ADRESSE BEGINNT DAS SPIEL "SIRENE"

18.1.6 Das Programm „Lichtschranke und Zähler“ (vgl. Abschnitt 3.4.3)

NR.	MARKE	MNEM. CODE	ADR.	OP. CODE	BEMERKUNGEN
1.	BEGINN	ICALL BLANK	1000	112	
2.		ILD A,=0	1001	1C4 00	
3.		IST A,ZAHL	1003	1CD E0	ZAHL:=00
4.	WEITER	ICALL HEX-DEZ	1005	11C	
5.		IST A,ZWSP	1006	1CD D8	
6.		ICALL UEB 2	1008	115	
7.	HELL	ILD A,=08	1009	1C4 08	
8.		ICALL ANZEIGE	100B	111	ANZEIGE DER ZAHL
9.		ILD A,S	100C	106	
10.		IAND A,=010	100D	1D4 10	
11.		IBNZ HELL	100F	17C F8	WDH., WENN HELL
12.	DUNKEL	ILD A,=08	1011	1C4 08	
13.		ICALL ANZEIGE	1013	111	ANZEIGE DER ZAHL
14.		ILD A,S	1014	106	
15.		IAND A,=010	1015	1D4 10	
16.		IBZ DUNKEL	1017	16C F8	WDH., WENN DUNKEL
17.		ILD A,ZAHL	1019	195 E0	ZAHL:=ZAHL+1
18.		IBRA WEITER	101B	174 E8	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZAHL	1FFE0	
ZWSP	1FFD8	ZWISCHENSPEICHER

Eine Bemerkung zu diesem Programm: Die Zahl im Speicher FFE0 wird erst erhöht, wenn der LDR nach einer Abdunklung wieder beleuchtet wird. Dazu sind die beiden Schleifen erforderlich: Schleife ①: 7. bis 11. Befehl; Schleife ②: 12. bis 16. Befehl. Solange der LDR beleuchtet wird, läuft das Programm in der Schleife ①, solange der LDR abgedunkelt ist, läuft das Programm in der Schleife ②. Bei fehlender Schleife ① würde die Zahl bei beleuchtetem LDR ständig erhöht werden.



Im Abschnitt 8.5 durfte bei der Addition zweier vierstelliger Dezimalzahlen die Summe nicht größer als 9999 sein. In diesem Programm ist diese Einschränkung aufgehoben. Jeder Summand kann jetzt maximal 9999 sein. Beide Summanden werden in Hexadezimalzahlen umgewandelt, im Hexadezimalsystem wird dann addiert. Bis hier ist unser Programm noch identisch mit dem aus Abschnitt 8.5. Die Summe im Hexadezimalsystem ist maximal $4E1E_{16}$ ($= 19998_{10}$). Wird jetzt der CALL HEX-DEZ aufgerufen, so wird in allen Fällen, in denen die Summe größer als $270F_{16}$ ist, die Anzeige „Error“ erscheinen. Wir unterscheiden für die Umwandlung ins Dezimalsystem daher zwei Fälle.

1. Fall: Die Summe ist größer als $270F_{16}$ ($= 9999_{10}$), aber sicher nicht größer als $4E1E_{16}$ ($= 19998_{10}$). Subtrahieren wir von dieser Summe 2710_{16} , so erhalten wir eine positive Hexadezimalzahl zwischen $270E_{16}$ ($= 9998_{10}$) und 0. Diese Differenz ergibt nach Umwandlung mit dem CALL HEX-DEZ die vier letzten Ziffern der gesuchten fünfstelligen Summe. Damit bei der Anzeige keine führenden Nullen gelöscht werden, benutzen wir den CALL UEB 4L. Es muß anschließend eine „1“ als fünfte Ziffer von rechts ergänzt werden.
2. Fall: Die Summe ist $270F_{16}$ oder kleiner. Nach Subtraktion von 2710_{16} erhalten wir eine negative Differenz. Wir addieren 2710_{16} wieder, führen die Umwandlung ins Dezimalsystem durch und bereiten mit dem CALL UEB 4R die Anzeige der maximal vierstelligen Summe vor.

Mit SA ist hier der Eingang Sense A des Mikroprozessors gemeint. Bei beleuchtetem LDR ist die am Eingang der Federleiste liegende Spannung niedrig, dann ist $(SA) = 1$; vgl. Abschnitt 3.7.2.

18.2 Ergänzungen zu besprochenen Programmen

18.2.1 Die Summe kann fünfstellig sein

INR.	MARKE	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	CALL EIN 4ST	1000	17	EINGABE: 1. SUMMAND
2.		ILD EA,ZWSP	1001	85 D8	
3.		ICALL HEX-DEZ	1003	1B	
4.		IST EA,HSP	1004	8D E1	
5.		CALL EIN 4ST	1006	17	EINGABE: 2. SUMMAND
6.		ILD EA,ZWSP	1007	85 D8	
7.		ICALL HEX-DEZ	1009	1B	
8.		IADD EA,HSP	100A	85 E1	BERECHNUNG DER SUMME
9.		ISUB EA,=10000	100C	BC 10 27	
10.		IST EA,HSP	100F	8D E1	(HSP):=SUMME-10000
11.		ILD A,S	1011	06	
12.		IAND A,=000	1012	04 80	
13.		IBZ 4-ST.	1014	6C 1C	SPRUNG, WENN (HSP) < 0
14.	5-ST.	ILD EA,HSP	1016	85 E1	
15.		ICALL HEX-DEZ	1018	1C	
16.		IST EA,ZWSP	1019	8D D8	VORBEREITUNG DER ANZEIGE
17.		ICALL UEB 4L	101B	14	DER VIER RECHTEN STELLEN
18.		ILD EA,CST4	101C	85 C4	
19.		IST EA,CST0	101E	8D C0	
20.		ILD EA,CST6	1020	85 C6	
21.		IST EA,CST2	1022	8D C2	
22.		ILD EA,=0	1024	84 00 00	
23.		IST EA,CST6	1027	8D C6	
24.		ILD A,=0C	1029	1C 0C	"1" AN FÜNFTE STELLE
25.		IST EA,CST4	102B	8D C4	VON RECHTS
26.		ICALL ANZ EIN	102D	10	ANZEIGE DER SUMME,
27.		INOP	102E	00	FÜNFSTELLIG
28.		INOP	102F	00	
29.		IBRA BEGINN	1030	17 4C	
30.	4-ST.	ILD EA,HSP	1032	85 E1	
31.		IADD EA,=10000	1034	84 10 27	SUMME:=<HSP>+10000
32.		ICALL HEX-DEZ	1037	1C	
33.		IST EA,ZWSP	1038	8D D8	
34.		ICALL UEB 4R	103A	13	
35.		ICALL ANZ EIN	103B	10	ANZEIGE DER SUMME,
36.		INOP	103C	00	VIERSTELLIG
37.		INOP	103D	00	
38.		IBRA BEGINN	103E	17 4C	

DATENSPEICHER	BEZEICHNUNG	ADR.	BEMERKUNGEN
	ZWSP	1FFD8	ZWISCHENSPEICHER
	HSP	1FFD9	
		1FFE1	HILFSSPEICHER, HIER STEHT
		1FFE2	ZUNÄCHST DER 1. SUMMAND
	CST0	1FFC0	CODIERUNG STELLE 0
		1FFC1	STELLE 1
	CST2	1FFC2	CODIERUNG STELLE 2
		1FFC3	STELLE 3
	CST4	1FFC4	CODIERUNG STELLE 4
		1FFC5	STELLE 5
	CST6	1FFC6	CODIERUNG STELLE 6
		1FFC7	STELLE 7

18.2.2 Die Differenz kann negativ sein

INR.	MARKE	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	ILD A,=0	1000	1C 00	
2.		IST A,VORZ	1002	1D E0	(VORZ):=00
3.		CALL EIN 4ST	1004	17	EINGABE MINUEND
4.		ILD EA,ZWSP	1005	85 D8	
5.		ICALL HEX-DEZ	1007	1B	
6.		IST EA,MINU	1008	8D E1	
7.		CALL EIN 4ST	100A	17	EINGABE SUBTRAHEND
8.		ILD EA,ZWSP	100B	85 D8	
9.		ICALL HEX-DEZ	100D	1B	
10.		IST EA,SUBT	100E	8D E3	
11.		ILD EA,MINU	1010	85 E1	
12.		ISUB EA,SUBT	1012	8D E3	BERECHNUNG DER
13.		IST EA,DIFF	1014	8D E5	DIFFERENZ
14.		ILD A,S	1016	06	
15.		IAND A,=000	1017	04 80	
16.		IBNZ ERGEBN	1019	17 8D	SPRUNG, WENN DIFF. POS.
17.		ILD EA,=0	101B	84 00 00	
18.		ISUB EA,DIFF	101E	8D E5	DIFF:=0000-DIFF
19.		IST EA,DIFF	1020	8D E5	
20.		ILD A,VORZ	1022	1C 50	
21.		IXOR A,=000	1024	1E 40 80	(VORZ):=000
22.		IST A,VORZ	1026	1D E0	
23.	ERGEBN	ILD EA,DIFF	1028	85 E5	
24.		ICALL HEX-DEZ	102A	1C	
25.		IST EA,ZWSP	102B	8D D8	DIFFERENZ UND
26.		ICALL UEB 4R	102D	13	
27.		ILD A,VORZ	102E	1C 50	
28.		IST A,CST4	1030	8D C4	VORZEICHEN WERDEN
29.		ICALL ANZ EIN	1032	10	ANGEZEIGT
30.		INOP	1033	00	
31.		INOP	1034	00	
32.		IBRA BEGINN	1035	17 4C	

DATENSPEICHER	BEZEICHNUNG	ADR.	BEMERKUNGEN
	VORZ	1FFE0	VORZEICHEN
	MINU	1FFE1	MINUEND
		1FFE2	
	SUBT	1FFE3	SUBTRAHEND
		1FFE4	
	DIFF	1FFE5	DIFFERENZ
		1FFE6	
	ZWSP	1FFD8	ZWISCHENSPEICHER
		1FFD9	
	CST4	1FFC4	CODIERUNG STELLE 4

Bei der Lösung zur 6. Aufgabe von Kapitel 9 (vgl. Kapitel 20) haben wir ein Programm angegeben, das positive Differenzen zweier vierstelliger Dezimalzahlen richtig berechnet. Dieses Programm haben wir hier so erweitert, daß auch negative Differenzen richtig bestimmt werden. Im 17. bis 19. Befehl wird zu der negativen Differenz das Komplement gebildet. Zusätzlich wird im 20. bis 22. Befehl das negative Vorzeichen erzeugt.

18.2.3 Das Produkt kann achtstellig sein

Wir wollen das Programm von Abschnitt 16.6 erweitern. Es sollen jetzt beliebige vierstellige Dezimalzahlen multipliziert werden können. Das Programm soll alle Multiplikationen bis $9999 \cdot 9999 = 99980001$ richtig ausführen. Der Programmbeginn ist problemlos. Die beiden Faktoren werden in Hexadezimalzahlen umgewandelt und mit Hilfe des Multiplikationsbefehls multipliziert. Das Problem kommt jetzt erst. Wie läßt sich das hexadezimal Produkt (maximal: $5F592E1_{16}$) in die entsprechende Dezimalzahl umwandeln?

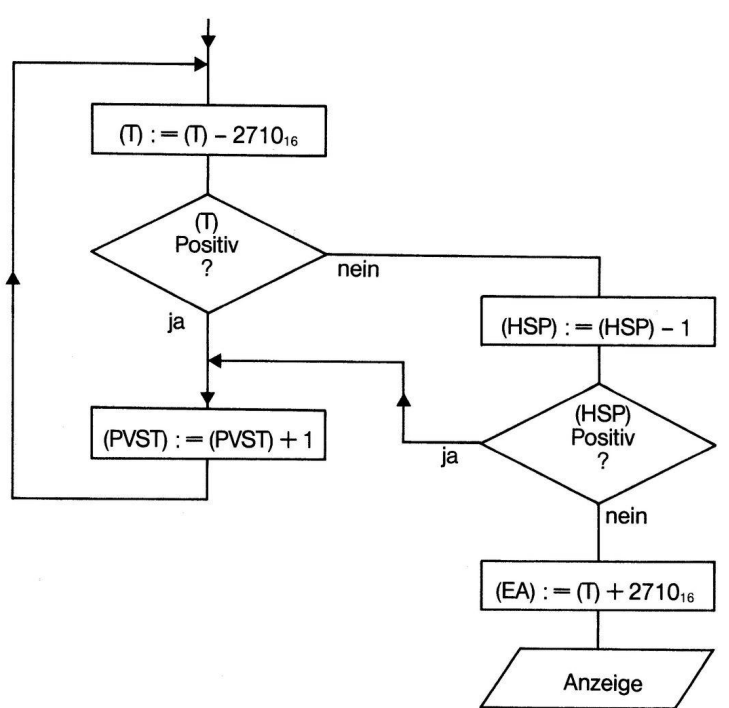
Die Lösungsidee ist folgende: Wir subtrahieren von dem hexadezimalen Produkt die Zahl 2710_{16} ($= 10000_{10}$) so oft wie möglich. Dann wissen wir, wie oft 10000_{10} im dezimalen Produkt enthalten ist. Der bei der Subtraktion verbleibende Rest ergibt die rechten vier Stellen des dezimalen Produktes.

Beispiel:
 $250 \cdot 923 = 230750$
 die Faktoren im Hexadezimalsystem: FA, 39B,
 das Produkt im Hexadezimalsystem: 3855E,
 die Umwandlung:

(PVST)		(HSP)	(T)
0	· 2710	+ 3	855E
1	· 2710	+ 3	5E4E
2	· 2710	+ 3	373E
3	· 2710	+ 3	102E
4	· 2710	+ 2	E91E
⋮	⋮		⋮
E	· 2710	+ 1	627E
F	· 2710	+ 1	3B6E
10	· 2710	+ 1	145E
⋮	⋮		⋮
17	· 2710	+	2EE

Damit steht das Ergebnis fest:
 Im Hexadezimalsystem haben wir erhalten:
 $FA \cdot 39B = 3855E = 17 \cdot 2710 + 2EE.$
 Das bedeutet im Dezimalsystem:
 $250 \cdot 923 = 23 \cdot 10000 + 750$
 $= 230750$

Zu Beginn des Programnteils „Umwandeln“ (14. bis 31. Befehl) stehen die letzten vier Hexadezimalziffern im T-Register, die weiteren Ziffern stehen im Hilfsppeicher HSP.



INR.	MARKE	IMHEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	ILD EA,=0	1000	04 00 00	
2.		IST EA,PVST	1003	0D E0	(PVST):=0000
3.		ICALL EIN 4ST	1005	17	
4.		ILD EA,ZWSP	1006	05 D8	
5.		ICALL DEZ-HEX	1008	1B	
6.		IST EA,HSP	1009	0D E2	
7.		ICALL EIN 4ST	100B	17	
8.		ILD EA,ZWSP	100C	05 D8	
9.		ICALL DEZ-HEX	100E	1B	
10.		ILD T,EA	100F	09	
11.		ILD EA,HSP	1010	05 E2	
12.		IMPY EA,T	1012	12C	
13.		IST EA,HSP	1013	0D E2	
14.	UMWAND	ILD EA,T	1015	0B	
15.		SUB EA,=02710	1016	0C 10 27	
16.		ILD T,EA	1019	09	
17.		ILD A,S	101A	06	
18.		AND A,=000	101B	04 00	
19.		IBZ HSP-1	101D	0C 09	
20.	PVST+1	ILD EA,PVST	101F	05 E0	
21.		IADD EA,=1	1021	04 01 00	
22.		IST EA,PVST	1024	0D E0	
23.		IBRA UMWAND	1026	174 ED	
24.	HSP-1	ILD EA,HSP	1028	05 E2	
25.		SUB EA,=1	102A	0C 01 00	
26.		IST EA,HSP	102D	0D E2	
27.		ILD A,S	102F	06	
28.		AND A,=000	1030	04 00	
29.		IBNZ PVST+1	1032	17C EB	
30.	ANZVOR	ILD EA,T	1034	0B	
31.		IADD EA,=02710	1035	0C 10 27	
32.		ICALL HEX-DEZ	1038	11C	
33.		IST EA,ZWSP	1039	0D D8	
34.		ICALL UEB 4R	103B	113	
35.		ILD EA,PVST	103C	05 E0	
36.		ICALL HEX-DEZ	103E	11C	
37.		IST EA,ZWSP	103F	0D D8	
38.		ICALL UEB 4L	1041	114	
39.	ANZEIG	ICALL ANZ EIN	1042	10	
40.		INOP	1043	00	
41.		INOP	1044	00	
42.		IBRA BEGINN	1045	174 B9	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
PVST	1FFE0	PRODUKT, VORDERE STELLEN
	1FFE1	
ZWSP	1FFD8	ZWISCHENSPEICHER
	1FFD9	
HSP	1FFE2	HILFSSPEICHER
	1FFE3	

Vom Inhalt des T-Registers wird die Hexadezimalzahl 2710_{16} ($= 10000_{10}$) subtrahiert. Ist das Subtraktionsergebnis positiv, so wird der „Zehntausender-Zähler“ PVST (Produkt vordere Stellen) um 1 erhöht. Ist das Subtraktionsergebnis negativ, so wird der Inhalt des Hilfspspeichers um 1 erniedrigt. Folgt jetzt wieder ein Rücksprung in die Schleife, in der

von (T) die Zahl 2710_{16} abgezogen wird, so wird der Inhalt des T-Registers wieder als positive Zahl aufgefaßt. Das können wir so interpretieren, daß gleichzeitig mit der Erniedrigung von (HSP) der Inhalt vom T-Register um 10000_{16} erhöht wurde.

Wenn auf diesem Wege (HSP) negativ geworden ist, haben wir einmal 2710_{16} zu häufig subtrahiert. Wir müssen zum Schluß die Zahl 2710_{16} einmal zum Inhalt des T-Registers addieren. Nach Umwandlung ins Dezimalsystem erhalten wir die vier letzten Ziffern des dezimalen Produktes. Wird der Inhalt des „Zehntausender-Zählers“ PVST in eine Dezimalzahl umgewandelt, so ergeben sich die vorderen Stellen des dezimalen Produktes.

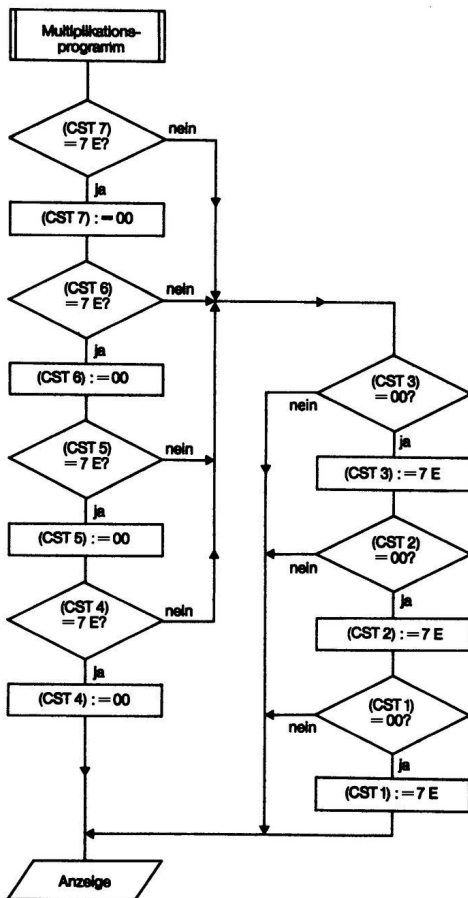
Damit ist die Multiplikation der beiden vierstelligen Dezimalzahlen abgeschlossen. Wir haben das maximal achtstellige Produkt erhalten.

Die Anzeige hat noch einen Schönheitsfehler (vgl. Abschnitt 16.5): links werden führende Nullen mitgeschrieben, rechts nicht. Diesen Nachteil werden wir im nächsten Abschnitt beseitigen. Bitte den Computer nicht ausschalten. Das hier besprochene Multiplikationsprogramm wird ergänzt.

18.2.4 Verbesserte achtstellige Anzeige

Wir gehen hier davon aus, daß die Anzeige für die vier rechten Stellen mit dem CALL UEB 4R, für die vier linken Stellen mit dem CALL UEB 4L vorbereitet worden ist. In den Speichern FFC0, FFC1, . . . , FFC7 stehen also die entsprechenden Codierungen. Die Aufgabe besteht darin, links die führenden Nullen zu löschen, rechts evtl. Nullen zu ergänzen, falls die dargestellte Zahl wenigstens fünfstellig ist.

In dem folgenden Programm werden die einzelnen Stellen nacheinander untersucht. Links wird gegebenenfalls die Codierung „7E“ durch „00“ ersetzt. Falls die dargestellte Zahl wenigstens fünfstellig ist wird rechts gegebenenfalls die Codierung „00“ durch „7E“ ersetzt.



INR.	MARKE	MNEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
139.	LINKS	ILD A,CST7	1042	IC5 C7	EVTL. LOESCHEN VON NULLEN STELLE 7
140.		XOR A,=07E	1044	IE4 7E	
141.		IBNZ RECHTS	1046	IC7 24	
142.		ILD A,=0	1048	IC4 00	
143.		IST A,CST7	104A	ICD C7	STELLE 6
144.		ILD A,CST6	104C	IC5 C6	
145.		XOR A,=07E	104E	IE4 7E	
146.		IBNZ RECHTS	1050	IC7 1A	
147.		ILD A,=0	1052	IC4 00	STELLE 5
148.		IST A,CST6	1054	ICD C6	
149.		ILD A,CST5	1056	IC5 C5	
150.		XOR A,=07E	1058	IE4 7E	
151.		IBNZ RECHTS	105A	IC7 10	STELLE 4
152.		ILD A,=0	105C	IC4 00	
153.		IST A,CST5	105E	ICD C5	
154.		ILD A,CST4	1060	IC5 C4	
155.		XOR A,=07E	1062	IE4 7E	STELLE 3
156.		IBNZ RECHTS	1064	IC7 06	
157.		ILD A,=0	1066	IC4 00	
158.		IST A,CST4	1068	ICD C4	
159.		IBRA ANZEIG	106A	IC7 18	STELLE 2
160.	RECHTS	ILD A,CST3	106C	IC5 C3	
161.		IBNZ ANZEIG	106E	IC7 14	
162.		ILD A,=07E	1070	IC4 7E	
163.		IST A,CST3	1072	ICD C3	STELLE 1
164.		ILD A,CST2	1074	IC5 C2	
165.		IBNZ ANZEIG	1076	IC7 0C	
166.		ILD A,=07E	1078	IC4 7E	
167.		IST A,CST2	107A	ICD C2	STELLE 1
168.		ILD A,CST1	107C	IC5 C1	
169.		IBNZ ANZEIG	107E	IC7 04	
170.		ILD A,=07E	1080	IC4 7E	
171.		IST A,CST1	1082	ICD C1	ANZEIGE, ACHTSTELLIG
172.	ANZEIG	ICALL ANZ EIN	1084	110	
173.		INOP	1085	100	
174.		INOP	1086	100	
175.		JMP BEGINN	1087	124 FF 0F	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
CST7	FFC7	CODIERUNG STELLE 7
CST6	FFC6	STELLE 6
CST5	FFC5	STELLE 5
CST4	FFC4	STELLE 4
CST3	FFC3	STELLE 3
CST2	FFC2	STELLE 2
CST1	FFC1	STELLE 1

WEITERE MARKE, BEZEICHNUNG	ADR.	BEMERKUNGEN
BEGINN	1000	

Dieses Programm wird im Anschluß an das Multiplikationsprogramm ab Adresse 1042 eingetastet. Die vier letzten Befehle des Multiplikationsprogramms werden dabei überschrieben.

18.3 Weitere Programme

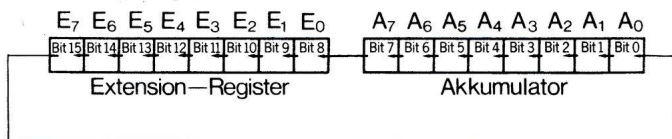
Wenn wir das vorliegende Anleitungsbuch bis hier durchgearbeitet haben, sind wir sicher in der Lage, eigene Programme zu schreiben. Wahrscheinlich haben wir das nebenbei längst gemacht.

In einem weiteren Anleitungsbuch sollen die restlichen Befehle besprochen, die Anschlußmöglichkeiten an der linken Federleiste unserer Mikroprozessor-Platine untersucht werden. Damit ergeben sich dann zusätzlich viele interessante Möglichkeiten, bei denen der Computer einen technischen Ablauf steuert. Der Computer kann nicht nur eine Glühlampe einschalten, einen Lautsprecher ansteuern oder eine Lichtschranke abfragen (vgl. Kapitel 3), er kann viel komplexere technische Schaltungen (z. B. eine Eisenbahnanlage) überwachen.

Aber auch mit den bisher besprochenen Befehlen und in der vorliegenden Ausbaustufe unseres Computers gibt es noch unzählige viele Möglichkeiten für weitere Programme. Die folgenden sieben Beispiele sind als Anregungen gedacht.

18.3.1 „RL EA“ – ein neuer Befehl?

„RL EA“ – das sieht aus wie ein zusätzlicher Rotationsbefehl: Rotiere den Inhalt vom EA-Register um eine Stelle nach links. Den Befehl gibt es aber nicht. Wir haben im Abschnitt 15.2 die beiden Rotationsbefehle angegeben, die im Befehlssatz unseres Mikroprozessors vorkommen. Weitere „Befehle“ können wir uns nur per Programm schaffen. In dem folgenden Programm wird nach Programmstart eine maximal vierstellige Hexadezimalzahl eingegeben und nach Betätigung einer Funktionstaste links angezeigt (vgl. Abschnitt 16.3). Jetzt wird nach jeder erneuten Betätigung einer Ziffern- oder Funktionstaste der Inhalt des EA-Registers um eine Stelle nach links rotiert. Hat das Bit 15 im EA-Register den Wert 0, so genügt der Befehl SL EA, andernfalls wird außerdem das Bit 0 im EA-Register gesetzt. Im ganzen entspricht das einem Rotieren.



INR.	MARKE	MNEM. CODE	ADR.	OP. CODE	BEMERKUNGEN
1.	BEGINN	CALL EIN 4ST	1000 17		<ZWSP> := EINGEGEBENE ZAHL
2.		CALL BLANK	1001 12		
3.	SCHL.	CALL UEB 4L	1002 14		ANZEIGE VON <ZWSP>
4.		CALL ANZ EIN	1003 10		
5.		INOP	1004 00		
6.		INOP	1005 00		
7.		ILD A,ZWSP2	1006 05 D9		BIT 15 VON <EA> ?
8.		AND A,=000	1008 04 80		
9.		BNZ ROTIER	100A 17C 07		
10.	SCHIEB	ILD EA,ZWSP	100C 05 D8		BIT 15 VON <EA> WAR 0!
11.		SL EA	100E 0F		
12.		IST EA,ZWSP	100F 0D D8		BIT 0 VON <EA> WIRD 1!
13.		IBRA SCHL.	1011 174 EF		
14.	ROTIER	ILD EA,ZWSP	1013 05 D8		BIT 15 VON <EA> WAR 1!
15.		SL EA	1015 0F		
16.		ADD A,=1	1016 1F4 01		BIT 0 VON <EA> WIRD 1!
17.		IST EA,ZWSP	1018 0D D8		
18.		IBRA SCHL.	101A 174 E6		

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZWSP	FFD8	ZWISCHENSPEICHER
ZWSP1	FFD9	
ZWSP2	FFD8	ZWISCHENSPEICHER, RECHTS
	FFD9	ZWISCHENSPEICHER, LINKS

18.3.2 Der BCD – Code

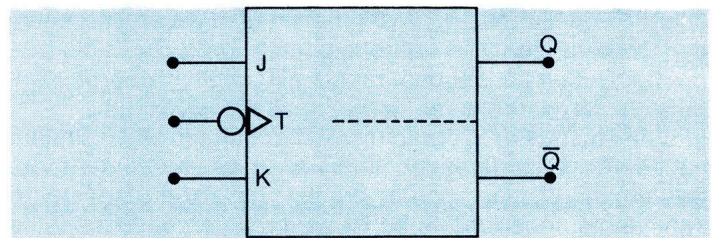
INR.	MARKE	MNEM. CODE	ADR.	OP. CODE	BEMERKUNGEN
1.	BEGINN	CALL BLANK	1000 12		ZAHL := 0
2.		ST A,ZAHL	1001 0D E0		
3.	WEITER	ILD A,ZAHL	1003 05 E0		ZAHL := ZAHL + 1
4.		CALL HEX-DEZ	1005 11C		
5.		ST A,LED-R	1006 0D C3		ZAHL ZUR LED-REIHE
6.		ST A,ZWSP	1008 0D D8		
7.		CALL UEB 2	100A 115		ZAHL ZUR ANZEIGE
8.		ILD A,CST4	100B 05 C4		
9.		ST A,CST1	100D 0D C1		ZAHL = 100?
10.		ILD A,E	100F 140		
11.		ST A,CST4	1010 0D C4		
12.		CALL ANZEIGE	1012 111		
13.		ILD A,ZAHL	1013 05 E0		
14.		XOR A,=100	1015 0E4 64		
15.		IBZ BEGINN	1017 16C E7		
16.		IBRA WEITER	1019 174 E8		

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZAHL	FFE0	
LED-R	FFC3	LED-REIHE
CST1	FFC1	CODIERUNG STELLE 1
CST4	FFC4	CODIERUNG STELLE 4
ZWSP	FFD8	ZWISCHENSPEICHER

Wenn wir dieses Programm starten – Schalter S muß in der vorderen Stellung stehen – sehen wir, daß wir einen Zähler programmiert haben. Es werden eine zweistellige Dezimalzahl und eine zugehörige Codierung an der LED-Reihe dargestellt. Bis zur Anzeige 09 bzw. 0000 1001 verläuft alles wie gewohnt. Dann folgt die Dezimalzahl 10. Die LED-Reihe zeigt jetzt die Codierung 0001 0000. Im BCD-Code (binär codiertes Dezimalsystem) wird jede Ziffer einer Dezimalzahl durch vier binäre oder duale Zeichen dargestellt. Der Vorteil besteht darin, daß die binäre Zeichenfolge leicht in die entsprechende Dezimalzahl umzuwandeln ist, z. B. stellt die Zeichenfolge 0110 1001 die Dezimalzahl 69 dar. Der Nachteil besteht u. a. darin, daß mit acht binären Zeichen nur 100 (und nicht 256) verschiedene Dezimalzahlen codiert werden können.

18.3.3 Ein JK – Flip-Flop

Ein Flip-Flop ist ein Speicher für ein Bit mit zwei stabilen Zuständen. Es kann also nur eine 1 oder eine 0 speichern. Wir können uns einen 8-Bit-Speicher in unserem Mikroprozessor-System aus acht solchen Flip-Flops zusammengesetzt denken. Das JK – Flip-Flop hat drei Eingänge: einen J-, einen K- und einen Takteingang. Der Ausgang Q gibt den Zustand des Flip-Flops an, der Ausgang \bar{Q} zeigt dazu den negierten Wert.



Das JK – Flip-Flop kann seinen Zustand nur bei einem 1-0-Übergang des Taktsignals (bei einer abfallenden Taktfllanke) ändern. Ob und wie sich dieser Zustand ändert, hängt von den Werten der beiden Eingangssignale J und K ab, die diese beim 1-0-Übergang des Taktsignals haben.

J	K	Beim 1-0-Übergang des Taktsignals
0	0	behält das Flip-Flop seinen Zustand
0	1	wird das Flip-Flop gelöscht (oder es bleibt gelöscht)
1	0	wird das Flip-Flop gesetzt (oder es bleibt gesetzt)
1	1	ändert das Flip-Flop seinen Zustand

In dem folgenden Programm wird ein solches JK – Flip-Flop simuliert. Dabei ist folgende Zuordnung gewählt worden:

J-Eingang	---	Sense A
K-Eingang	---	Sense B
Takt-Eingang	---	beliebige Ziffern- oder Funktionstaste (außer A ↔ D)
Q-Ausgang	---	Flag 3 (rot)
\bar{Q} -Ausgang	---	Flag 1 (grün)

Durch Betätigung einer Taste erzeugen wir eine 1 an dem entsprechenden Eingang, bei Nichtbetätigung eine 0; den 1-0-Übergang erreichen wir beim Loslassen der Taste. Die Leuchtdioden leuchten beim Wert 1.

NR.	MARKE	MMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	LD A,=00	1000	IC4 00	
2.		ILD S,A	1002	IC7 07	<F3>:=1; <F1>:=0
3.		ILD EA,=0E270	1003	IC4 70 E2	ANZEIGE: "FLIPFLOP"
4.		IST EA,CST6	1006	ICD C6	
5.		IST EA,CST2	1008	ICD C2	
6.		ILD EA,=060E6	100A	IC4 E6 06	
7.		IST EA,CST4	100D	ICD C4	
8.		ILD EA,=07EE6	100F	IC4 E6 7E	
9.		IST EA,CST0	1012	ICD C0	
10.	ANZ.	ICALL ANZ EIN	1014	IC10	
11.		INOP	1015	IC00	
12.		INOP	1016	IC00	
13.		ILD A,S	1017	IC06	
14.		IAND A,=030	1018	ICD 30	TASTE GEDRUECKT?
15.		IBZ ANZ.	101A	IC6 F8	
16.		IXOR A,=010	101C	IC4 10	<SA> = 1?
17.		IBNZ WEITER	101E	IC7 05	
18.	SETZEN	ILD A,=08	1020	IC4 08	SETZEN DES FLIPFLOPS:
19.		ILD S,A	1022	IC7 07	<F3>:=1; <F1>:=0
20.		IBRA ANZ.	1023	IC74 EF	
21.	WEITER	IXOR A,=030	1025	IC4 30	<SB> = 1?
22.		IBNZ AEND.	1027	IC7 05	
23.	LOESCH	ILD A,=02	1029	IC4 02	LOESCHEN DES FLIPFLOPS:
24.		ILD S,A	102B	IC7 07	<F3>:=0; <F1>:=1
25.		IBRA ANZ.	102C	IC74 E6	
26.	AEND.	ILD A,S	102E	IC06	
27.		IXOR A,=0A	102F	IC4 0A	AENDERN VON <F3> UND <F1>
28.		ILD S,A	1031	IC7 07	
29.		IBRA ANZ.	1032	IC74 E0	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
CST0	FFC0	CODIERUNG STELLE 0
CST2	FFC2	STELLE 2
CST4	FFC4	STELLE 4
CST6	FFC6	STELLE 6

Auf eine Besonderheit des Programms soll noch hingewiesen werden: Wir brauchen den Inhalt des Status-Registers nur einmal einzulesen und können trotzdem die vier möglichen Fälle erkennen. Bei (SA)=0 und (SB)=0 erfolgt nach einem 1-0-Übergang des Taktsignals im 15. Befehl wieder ein Sprung zur Marke ANZ. Der Zustand des Flip-Flops wird nicht verändert.

Ist eine oder sind beide Sense-Tasten betätigt, so wird der Akkuinhalt im 16. Befehl mit der Zahl 10₁₆ XOR-verknüpft. Es gibt drei Möglichkeiten:

(A)	(A) \oplus 10 ₁₆
10	00
20	30
30	20

Im ersten Fall wird das Flip-Flop gesetzt (oder es bleibt gesetzt). Wenn wir im zweiten oder dritten Fall den neuen Akkuinhalt jetzt mit der Zahl 30₁₆ XOR-verknüpfen (21. Befehl), erkennen wir, ob das Flip-Flop gelöscht oder ob sein Zustand geändert werden soll.

18.3.4 Eine Ampel mit Handbedienung

NR.	MARKE	MMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	LD A,=02	1000	IC4 02	<F1>:=1; GRUEN
2.		ILD S,A	1002	IC7 07	
3.		ICALL HALT	1003	IC1F	HALT
4.		ILD A,=04	1004	IC4 04	<F2>:=1; GELB
5.		ILD S,A	1006	IC7 07	
6.		ILD EA,=0	1007	IC4 00 00	
7.		ICALL VERZ	100A	IC1D	
8.		ILD A,=08	100B	IC4 08	<F3>:=1; ROT
9.		ILD S,A	100D	IC7 07	
10.		ICALL HALT	100E	IC1F	HALT
11.		ILD A,=0C	100F	IC4 0C	<F2>:=1; <F3>:=1; GELB UND ROT
12.		ILD S,A	1011	IC7 07	
13.		ILD EA,=0	1012	IC4 00 00	
14.		ICALL VERZ	1015	IC1D	
15.		IBRA BEGINN	1016	IC74 E0	

In diesem Programm wird der CALL HALT angewendet. Das Programm kann nach jedem HALT mit Hilfe der **RUN**-Taste fortgesetzt werden.

Wir können uns vorstellen, daß die Ampel per Hand je nach Verkehrslage umgeschaltet werden soll.

Das Programm soll deutlich machen, welche Vorteile der CALL HALT (neben der Programmüberprüfung) bietet: Man kann einen Programmteil bearbeiten lassen und dann entscheiden, wann der nächste Teil gestartet werden soll.

18.3.5 Das kleine Einmaleins

Nach Programmstart meldet sich der Computer mit der Anzeige „1. 1“. Er wartet jetzt auf Betätigung einer der Tasten 1, 2, ..., 9.

NR.	MARKE	MMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	CALL BLANK	1000	IC12	
2.		ILD A,=0C	1001	IC4 0C	"1.1" ZUR ANZEIGE
3.		IST A,CST0	1003	ICD C0	
4.		IST A,CST2	1005	ICD C2	
5.		ILD A,=01	1007	IC4 01	
6.		IST A,CST1	1009	ICD C1	
7.	EING.	ICALL ANZ EIN	100B	IC10	EINGABE DER ZAHL
8.		IBRA EING.	100C	IC74 FD	
9.		IST A,ZWSP1	100E	ICD D8	
10.		IST A,ZAHL	1010	ICD E0	
11.		ISUB A,=0A	1012	ICF 0A	ZAHL > 9?
12.		ILD A,S	1014	IC06	
13.		IAND A,=000	1015	ICD 00	
14.		IBZ ANZ.	1017	IC6 01	
15.		ICALL FEHLER	1019	IC1A	
16.	ANZ.	ICALL UEB 2	101A	IC15	
17.		ILD A,=0	101B	IC4 00	
18.		IST A,CST5	101D	ICD C5	
19.		ICALL ANZEIGE	101F	IC11	ANZEIGE MIT EINGEGEBENER ZAHL
20.		ICALL ANZEIGE	1020	IC11	
21.		ILD A,S	1021	IC06	
22.		IST A,VIELF	1022	ICD E1	
23.		IST A,ZAEHLER	1024	ICD E3	
24.	SCHL.	ILD A,ZAEHLER	1026	IC5 E3	ZAEHLER:=ZAEHLER+1
25.		IXOR A,=0A	1028	IC4 0A	ZAEHLER = A?
26.		IBNZ ADD.	102A	IC7 04	
27.		ILD A,=010	102C	IC4 10	ZAEHLER:=10
28.		IST A,ZAEHLER	102E	ICD E3	
29.	ADD.	ILD A,VIELF	1030	IC5 E1	
30.		IADD A,ZAHL	1032	ICF E0	VIELF:=VIELF+ZAHL
31.		IST A,VIELF	1034	ICD E1	
32.		ICALL HEX-DEZ	1036	IC1C	
33.		IST EA,ZWSP	1037	ICD D8	VIELF ZUR ANZEIGE
34.		ICALL UEB 4R	1039	IC13	
35.		ILD A,ZAEHLER	103A	IC5 E3	
36.		IST A,ZWSP2	103C	ICD D9	ZAEHLER ZUR ANZEIGE
37.		ILD A,ZAHL	103E	IC5 E0	
38.		IST A,ZWSP1	1040	ICD D8	ZAHL ZUR ANZEIGE
39.		ICALL UEB 4L	1042	IC14	
40.		ILD A,CST7	1043	IC5 C7	
41.		IXOR A,=07E	1045	IC4 7E	
42.		IBNZ WEITER	1047	IC7 04	
43.		ILD A,=0	1049	IC4 00	EVTL. "0" LOESCHEN
44.		IST A,CST7	104B	ICD C7	
45.	WEITER	ILD A,=01	104D	IC4 01	
46.		IST A,CST5	104F	ICD C5	"." ZUR ANZEIGE
47.		ILD A,=090	1051	IC4 90	
48.		IST A,CST2	1053	ICD C2	"=" ZUR ANZEIGE
49.		ILD A,=0	1055	IC4 00	
50.		ICALL ANZEIGE	1057	IC11	
51.		ICALL ANZEIGE	1058	IC11	
52.		ILD A,ZAEHLER	1059	IC5 E3	
53.		IXOR A,=010	105B	IC4 10	ZAEHLER = 10?
54.		IBNZ SCHL.	105D	IC7 C7	WENN ZAEHLER < 10
55.		ICALL ANZEIGE	105F	IC11	
56.		IBRA BEGINN	1060	IC74 9E	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZWSP	FFD8	ZWISCHENSPEICHER
	FFD9	
ZWSP1	FFD8	
ZWSP2	FFD9	
CST0	FFC0	CODIERUNG STELLE 0
CST1	FFC1	STELLE 1
CST2	FFC2	STELLE 2
CST3	FFC3	STELLE 3
CST7	FFC7	STELLE 7
ZAHL	FFE0	
VIELF	FFE1	VIELFACHE DER ZAHL
	FFE2	
ZAEHLER	FFE3	

18.3.6 Ein Würfelspiel

NR.	MARKE	INMEM. CODE	ADR.	OP. CODE	BEMERKUNGEN
1.	BEGINN	ILD A, \rightarrow 0	1000	184 00 00	
2.		IST A,AUGEN	1003	1CD E3	AUGEN:=00
3.		IST A,SUMME	1005	18D E4	SUMME:=0000
4.	SCHL.	JSR ANZVOR	1007	120 79 10	
5.	ANZ.	ILD A, \rightarrow 8	1008	1C4 08	
6.		ICALL ANZEIGE	100C	111	
7.		ILD A,S	100D	106	
8.		AND A, \rightarrow 030	100E	1D4 30	(SA)=1?; (SB)=1?
9.		IBZ ANZ.	1010	16C F8	WENN NICHT, WDH.
10.		XOR A, \rightarrow 010	1012	1E4 10	
11.		IBZ ENDE	1014	16C 46	SPRUNG, WENN (SA)=1
12.	WURF	JSR ZUFALL	1016	120 CE 06	WUERFELN, WENN (SB)=1
13.		AND A, \rightarrow 7	1019	1D4 07	
14.		IBZ WURF	101B	16C F9	WDH., WENN AUGEN=0
15.		IST A,AUGEN	101D	1CD E3	
16.		XOR A, \rightarrow 7	101F	1E4 07	
17.		IBZ WURF	1021	16C F3	WDH., WENN AUGEN=7
18.		ILD A,S	1023	106	
19.		AND A, \rightarrow 020	1024	1D4 20	
20.		IBZ WURF	1026	17C EE	WDH., SOLANGE (SB)=1
21.		ILD A,SUMME	1028	1C5 E4	
22.		IADD A,AUGEN	102A	1F5 E3	SUMME:=SUMME+AUGEN
23.		IST A,SUMME	102C	1CD E4	
24.		ISUB A, \rightarrow 21	102E	1FC 15	
25.		ILD A,S	1030	106	
26.		AND A, \rightarrow 080	1031	1D4 80	
27.		IBZ SCHL.	1033	16C D2	SPRUNG, WENN SUMME<21
28.	FERTIG	ILD A, \rightarrow 010	1035	1C4 10	BLINKEN DER ANZEIGE
29.		IST A,ZAEHLER	1037	1CD E6	
30.	BLINK	JSR ANZVOR	1039	120 79 10	
31.		ILD A, \rightarrow 020	103C	1C4 20	
32.		ICALL ANZEIGE	103E	111	
33.		ILD A, \rightarrow 0	103F	184 00 00	
34.		IST A,CST0	1042	18D C0	
35.		ILD A, \rightarrow 010	1044	1C4 10	
36.		ICALL ANZEIGE	1046	111	
37.		ILD A,ZAEHLER	1047	19D E6	
38.		IBZ BLINK	1049	17C EE	WDH., WENN ZAEHLER<0
39.	ERGEBN	ILD A, \rightarrow 010	104B	1C4 10	
40.		IST A,ZWSP	104D	1CD D8	
41.		ICALL UEB 2	104F	115	"10" ZUR ANZEIGE
42.		ILD A,SUMME	1050	1C5 E4	
43.		XOR A, \rightarrow 21	1052	1E4 15	
44.		IBZ PUNKTE	1054	16C 15	SPRUNG, WENN SUMME=21
45.		ILD A, \rightarrow 080	1056	1C4 80	
46.		IST A,CST6	1058	1CD C6	"-" ZUR ANZEIGE
47.		IBRA PUNKTE	105A	174 0F	
48.	ENDE	ILD A,SUMME	105C	1C5 E4	BERECHNUNG DER PUNKTE
49.		ISUB A, \rightarrow 15	105E	1FC 0F	
50.		IST A,ZWSP	1060	1CD D8	
51.		AND A, \rightarrow 0F0	1062	1D4 F0	
52.		IBNZ ERGEBN	1064	17C E5	SPRUNG, WENN SUMME<15
53.		ICALL UEB 2	1066	115	
54.		ILD A, \rightarrow 0	1067	1C4 00	
55.		IST A,CST5	1069	1CD C5	
56.	PUNKTE	ILD A, \rightarrow 0E67D	106B	184 7D E6	"PU." ZUR ANZEIGE
57.		IST A,CST0	106E	18D C0	
58.	WARTEN	ILD A, \rightarrow 8	1070	1C4 08	
59.		ICALL ANZEIGE	1072	111	ANZEIGE, SOLANGE (SB)=0
60.		ILD A,S	1073	106	
61.		AND A, \rightarrow 020	1074	1D4 20	
62.		IBZ WARTEN	1076	16C F8	
63.		IBRA BEGINN	1078	174 86	
64.	ANZVOR	ILD A,SUMME	107A	185 E4	UNTERPROGRAMM:
65.		ICALL HEX-DEZ	107C	11C	VORBEREITUNG DER ANZEIGE
66.		IST A,ZWSP	107D	18D D8	
67.		ICALL UEB 4R	107F	113	
68.		ILD A,AUGEN	1080	1C5 E3	
69.		IST A,ZWSP	1082	1CD D8	
70.		ICALL UEB 2	1084	115	
71.		ILD A, \rightarrow 0	1085	1C4 00	
72.		IST A,CST5	1087	1CD C5	
73.		IRET	1089	15C	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZAH1	FFFE0	DREI SPEICHER WERDEN VOM
ZAH2	FFFE1	ZUFALLSZAHL-GENERATOR
	FFFE2	BENÖTIGT.
AUGEN	FFFE3	
SUMME	FFFE4	
ZAEHLER	FFFE5	
	FFFE6	
CST0	FFFC0	CODIERUNG STELLE 0
CST5	FFFC5	CODIERUNG STELLE 5
CST6	FFFC6	CODIERUNG STELLE 6
ZWSP	FFFD8	ZWISCHENSPEICHER

WEITERE MARKE, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZUFALL	06CF	UNTERPROGRAMM:
		ZUFALLSZAHL-GENERATOR

Dieses längere Programm braucht nicht eingetastet zu werden. Es ist als erstes Beispiel auf der beiliegenden Cassette abgespeichert.

Ein wichtiger Hinweis: Wenn das Programm von Cassette geladen worden ist, muß vor dem Programmstart das Überspielkabel abgezogen werden. Durch Anstecken dieses Kabels wird Sense B auf 1 gezogen. Da im Würfelspiel die Taste **SB** zum Würfeln benutzt wird, läuft das Programm mit angestecktem Überspielkabel ständig in der Würfelschleife (12. bis 20. Befehl).

Nun zu den Spielregeln: Wenn wir das Programm starten, meldet sich der Computer mit der Anzeige zweier Nullen. Er wartet auf das Würfeln, das wir mit Hilfe der Taste **SB** vornehmen können. Nach dem Würfeln zeigt der Computer links das letzte Würfelergbnis, rechts die Summe der bisherigen Würfe. Wir können mehrfach hintereinander würfeln.

Es ist das Ziel, die Summe 21 zu erreichen. Das Spiel ist beendet, wenn die Summe 21 erreicht worden ist, wenn die Summe 21 überschritten wird, oder wenn wir vor Erreichen der Summe 21 die Taste **SA** betätigen. Der Computer teilt uns am Ende des Würfelspiels Punkte zu:

Summe	Punkte
größer als 21	- 10
21	10
20	5
19	4
18	3
17	2
16	1
15	0
kleiner als 15	- 10

Nach der Anzeige der erreichten Punkte können wir unmittelbar mit der Taste **SB** ein neues Spiel beginnen.

18.3.7 Messung der Reaktionszeit

Wenn wir das folgende Programm starten, erlischt die Anzeige. Mit Hilfe des Zufallszahlen-Generators wird eine Pausenlänge ermittelt. Nach dieser Pause werden die drei farbigen Leuchtdioden angeschaltet. Jetzt kommt es darauf an, möglichst schnell die Taste **SA** zu betätigen. Die Reaktionszeit – die Zeit vom Aufleuchten der Leuchtdioden bis zur Betätigung der **SA**-Taste – wird in Millisekunden (in tausendstel Sekunden) gemessen und angezeigt.

Das Programm läßt sich erneut mit einer Ziffern- oder Funktionstaste (außer **A \leftrightarrow D**) starten. Falls wir allerdings beim Einschalten der Leuchtdioden schon die Taste **SA** drücken, oder falls wir in zehn Sekunden nicht reagieren, erscheint die Anzeige „Error“. Dann ist ein neuer Programmstart mit den Tasten **RS**, **RUN** möglich.

Mit der Zeitmessung werden wir uns in diesem Anleitungsbuch nicht ausführlich beschäftigen. Für das Verständnis des vorliegenden Programms reicht es, wenn wir wissen, daß die Schleife (16. bis 26. Befehl) genau eine Millisekunde dauert. Da in jedem Schleifendurchlauf der Inhalt in den Speichern FFE3/FFE2 um 1 erhöht wird, können wir nach Verlassen der Schleife die Reaktionszeit in Millisekunden

NR.	MARKE	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	CALL BLANK	1000	12	
2.		RAND S,=0	1001	139 00	(F1):=0; (F2):=0; (F3):=0
3.		JSR ZUFALL	1003	120 CE 06	BESTIMMUNG DER PAUSEN- LAENGE
4.		IST A,ZAHL	1006	1C D E1	
5.		ILD EA,=0	1008	184 00 00	
6.		IST EA,ZEIT	100B	18D E2	ZEIT:=0.000
7.	PAUSE	ILD EA,=0000	100D	184 00 00	PAUSE UNTERSCHIEDLICHER LAENGE
8.		CALL VERZ	1010	11D	
9.		ILD A,ZAHL	1011	19D E1	
10.		IBNZ PAUSE	1013	17C F8	
11.		IOR S,=0E	1015	13B 0E	(F1):=1; (F2):=1; (F3):=1
12.		ILD A,S	1017	106	
13.		RAND A,=010	1018	1D4 10	
14.		IBZ UHR	101A	16C 01	SPRUNG, WENN (SA)=0
15.		CALL FEHLER	101C	11A	SONST ANZEIGE "ERROR"
16.	UHR	ILD EA,ZEIT	101D	185 E2	
17.		IADD EA,=01	101F	1B4 01 00	ZEIT:=ZEIT+0.001
18.		IST EA,ZEIT	1022	18D E2	
19.		CALL HEX-DEZ	1024	11C	
20.		IST EA,ZWSP	1025	18D 08	(ZWSP):=ZEIT.DEZIMAL
21.		ILD EA,=0E	1027	184 0E 00	
22.		CALL VERZ	102A	11D	
23.		ILD A,AUSGZ	102B	195 E0	
24.		ILD A,S	102D	106	
25.		RAND A,=010	102E	1D4 10	
26.		IBZ UHR	1030	16C EB	WDH., WENN (SA)=0
27.	ISTOP	CALL UEB 4R	1032	113	
28.		ILD A,CST3	1033	1C5 C3	
29.		IOR A,=01	1035	1DC 01	BERGRENZUNG VON "."
30.		IST A,CST3	1037	1CD C3	
31.		CALL ANZ EIN	1039	110	ANZEIGE DER ZEIT
32.		INOP	103A	100	
33.		INOP	103B	100	
34.		IBRA BEGINN	103C	174 C2	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
AUSGZ	1FFE0	AUSGANGSZAHL FUER "ZUFALLSZAHLEN-GENERATOR"
ZAHL	1FFE1	ZUFALLSZAHL, WIRD VOM UNTER- PROGRAMM "ZUFALL" ERMITTELT
ZEIT	1FFE2	ZEIT IN MILLISEKUNDEN, HEXADEZIMAL
ZWSP	1FFD8	ZWISCHENSPEICHER ZEIT IN MILLISEKUNDEN, DEZIMAL
CST3	1FFC3	CODIERUNG STELLE 3

WEITERE MARKE, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZUFALL	106CF	UNTERPROGRAMM: "ZUFALLSZAHLEN-GENERATOR"

19. Programme auf Cassette

19.1 Cassettenrecorder – Interface

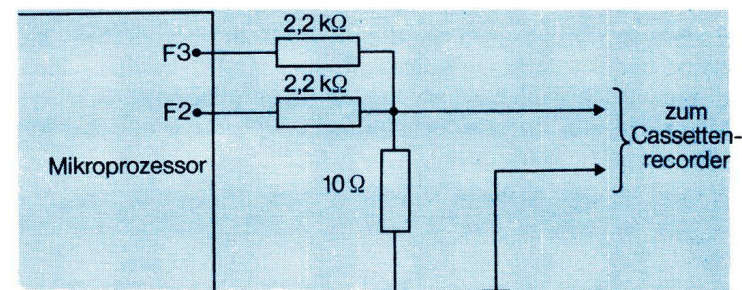
Interface ist eine englische Bezeichnung. Sie wird auch in der deutschen Fachsprache benutzt. Eine mögliche Übersetzung ist „Schnittstelle“. Eine Interface-Schaltung ist eine Schaltung an der Schnittstelle zwischen zwei Systemteilen. Mit dem Cassettenrecorder-Interface meinen wir die elektronische Schaltung zwischen dem Mikroprozessor und dem Cassettenrecorder.

Diese Schaltung besteht aus zwei getrennten Teilen. Der eine Teil wird benutzt, um eine Folge von Einsen und Nullen, die vom Mikroprozessor kommen, in Signale umzuwandeln, die der Cassettenrecorder speichern kann. Der andere Teil der Interface-Schaltung sorgt dafür, daß die Signale, die vom Cassettenrecorder kommen, wieder für den Mikroprozessor lesbar werden.

Dieses Cassettenrecorder-Interface ist die erforderliche Hardware für den Datentransport zwischen Mikroprozessor und Cassettenrecorder. Außerdem ist eine geeignete Software im Betriebsprogramm des Computers erforderlich. Das Programm zum Abspeichern auf die Cassette liest die Inhalte aus dem zu überspielenden Speicherbereich nacheinander (byteweise) in den Akkumulator und gibt pro Byte acht Informationen über das Cassettenrecorder-Interface zum Cassettenrecorder. Das Programm zum Laden von der Cassette untersucht die ankommenden, von der Interface-Schaltung aufbereiteten Signale, interpretiert sie als Einsen oder Nullen und schreibt dann je acht Bit (je ein Byte) in einen Speicher des gewünschten Bereichs im RAM.

19.1.1 Interface-Schaltung, 1. Teil

Die Interface-Schaltung für den Datentransport zur Cassette ist sehr einfach:



Die Schaltungen, die wir im Abschnitt 3.7.1 kennengelernt haben, sind hier nicht mitgezeichnet. Die Schaltungen von Abschnitt 3.7.1 (Ansteuerung der farbigen Leuchtdioden, Verstärkung der Ausgangssignale) bestehen neben dieser Interface-Schaltung. Gegenseitige Störungen dieser Schaltungsteile sind ausgeschlossen.

Es werden zwei Flags (F2 und F3) zur Datenausgabe benutzt. Mit Hilfe der Software und dieser Interface-Schaltung werden geeignete Impulse für den Cassettenrecorder-Eingang erzeugt. Um diese Aussage zu verstehen, sehen wir uns ein Testprogramm an, das mit den Tasten **SP** und **E** aufgerufen werden kann.

Im Unterprogramm „IMPULS“ wird Flag 2 (gelb) gesetzt, dann werden im 23. Befehl Flag 2 und Flag 3 gelöscht, im 26. Befehl wird Flag 3 (rot) gesetzt. Im Hauptprogramm „SPE“ bleibt Flag 3 gesetzt. Im wesentlichen wird im Hauptprogramm nur eine Pause gemacht. Daß durch Betätigung der

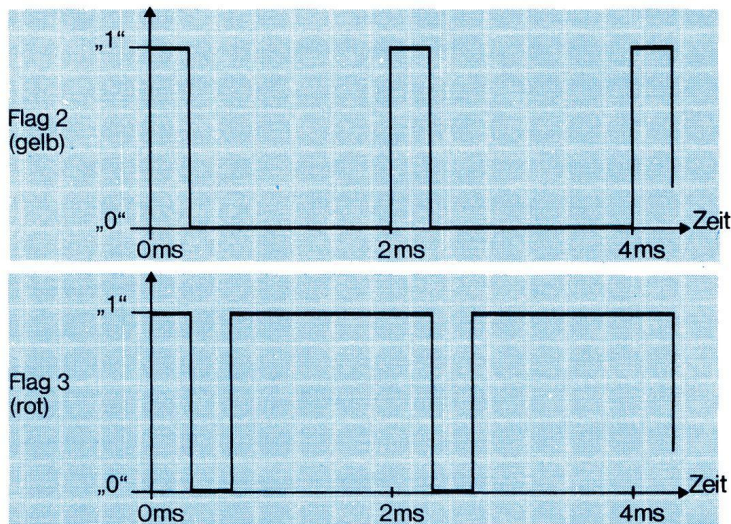
ablesen. Es ist sicher nicht erforderlich, den Befehl CALL HEX-DEZ in jedem Schleifendurchlauf auszuführen. Dadurch wird aber erreicht, daß die Schleife spätestens nach zehn Sekunden (10 000 Millisekunden) mit der Anzeige „Error“ verlassen wird. Außerdem müssen wir den Computer in der Schleife beschäftigen, damit er eine Millisekunde für den Schleifendurchlauf benötigt.

Die Spielregeln lassen sich variieren. Wir können versuchen, möglichst genau die Zeit 5,000 Sekunden zu erreichen. Wir können das Programm ergänzen, so daß der Computer uns die Differenz zwischen der Reaktionszeit und 5,000 Sekunden anzeigt. Der Phantasie sind keine Grenzen gesetzt.

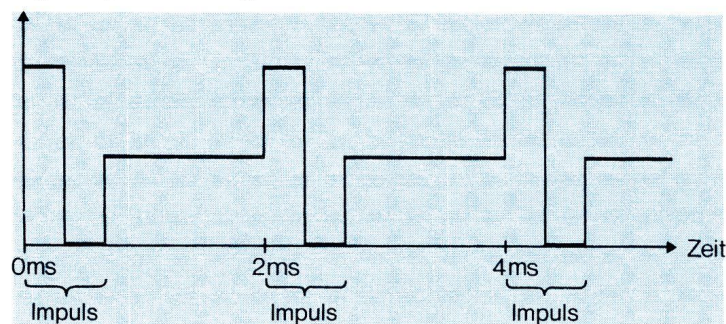
19.1.2 Interface-Schaltung, 2. Teil

NR.	MARKE	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	ISP	E	0A09	20 AF 07	SPRUNG ZUM UNTERPR. "IMPULS"
2.		ILD A, _r =060	0A0C	1C4 69	
3.		JSR PAUSE	0A0E	120 55 08	SPRUNG ZUM UNTERPR. "PAUSE"
4.		ILD A, _r S	0A11	106	
5.		IAND A, _r =010	0A12	1D4 10	
6.		IBZ SP E	0A14	16C F3	WDH., WENN <SA>=0
7.		IBRA SP F	0A16	174 07	SPRUNG NACH "SP F"
11.	ISP	F	0A1F	1C4 0C	
20.	IMPULS	IOR S, _r =04	07B0	13B 04	<F2>:=1
21.		ILD A, _r =0F	07B2	1C4 0F	
22.		JSR PAUSE	07B4	120 55 08	
23.		IAND S, _r =0F3	07B7	139 F3	<F3>:=0; <F2>:=0
24.		ILD A, _r =0F	07B9	1C4 0F	
25.		JSR PAUSE	07BB	120 55 08	
26.		IOR S, _r =08	07BE	13B 08	<F3>:=1
27.		IRET	07C0	15C	
30.	IPAUSE	SUB A, _r =01	0856	1FC 01	
31.		IBNZ PAUSE	0858	17C FC	
32.		IRET	085A	15C	

Taste **SA** eine Beendigung dieses Programms „SP E“ möglich ist, ist für den Programmablauf unwesentlich. An Flag 2 und Flag 3 entstehen bei diesem Testprogramm folgende Signale:

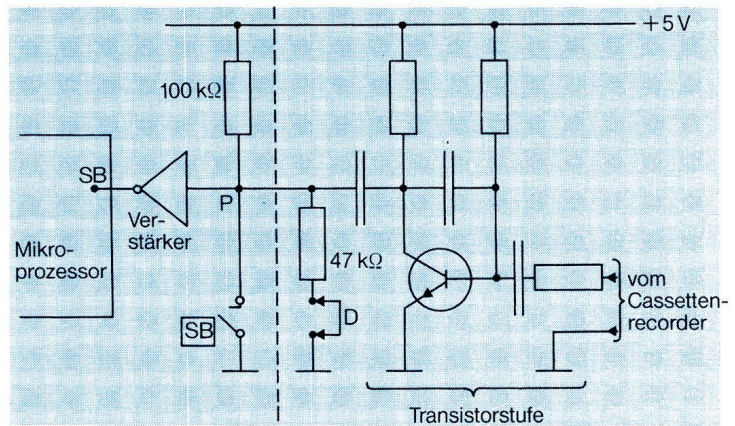


Es ist verständlich, warum die rote Leuchtdiode normal, die gelbe Leuchtdiode nur schwach leuchtet. Die gelbe Leuchtdiode wird jeweils nur sehr kurzzeitig eingeschaltet. Mit Hilfe der Interface-Schaltung werden diese beiden Signale jetzt überlagert.



Die so entstandenen Impulse sind aufgrund der in der Interface-Schaltung verwendeten Widerstände sehr schwach. Sie sind aber für den Eingangsverstärker des Cassettenrecorders ausreichend.

Wir verbinden den Computer mit dem Cassettenrecorder und nehmen einige Minuten lang die vom Computer erzeugte Impulsfolge (Tasten: **SP**, **E**) auf. Das Überspielkabel liegt bei, der gekennzeichnete Stecker kommt in die DIN-Buchse der Mikroprozessor-Platine. Wenn wir bei dieser Aufnahme den Lautsprecher des Cassettenrecorders nicht ausschalten, hören wir einen gleichmäßigen Ton.



Der linke Teil der Schaltung ist uns vom Abschnitt 3.7.2 bekannt. Der Punkt P ist der Anschluß für Sense B an der rechten Steckerleiste. Ohne Drahtbrücke D liegt der Punkt P über den 100 kΩ-Widerstand an +5V. Da der Verstärker sein Eingangssignal negiert, liegt am Mikroprozessor-Eingang SB eine 0. Durch Betätigung von **SB** haben wir den Punkt P auf 0V heruntergezogen und damit eine 1 für den Mikroprozessor erzeugt.

Beim Einlesen der Impulse vom Cassettenrecorder darf der Eingang des Verstärkers (Punkt P) nicht an + 5V liegen. Mit Hilfe der Drahtbrücke D wird der Spannungsteiler aus 100 kΩ und 47 kΩ wirksam, am Punkt P liegt etwa eine Spannung von 1,5 V. Bei dieser Spannung schaltet der Verstärker noch nicht durch. Der Taster **SB** ist jetzt unwirksam, da er ja die Eingangsspannung des Verstärkers nur noch kleiner machen würde. Die vom Cassettenrecorder eintreffenden und von der Transistorstufe verstärkten Impulse können jetzt den Verstärker durchschalten.

Um die Drahtbrücke D brauchen wir uns nicht zu kümmern. Sie ist im gekennzeichneten DIN-Stecker des beiliegenden Verbindungskabels untergebracht. Mit dem Anschließen des Cassettenrecorders wird automatisch diese Drahtbrücke D eingesetzt.



Wir müssen aber an diese Drahtbrücke D denken, wenn wir ein eigenes Programm starten, bei dem die Taste **SB** benutzt wird. Dann muß das Verbindungskabel zum Cassettenrecorder unbedingt abgezogen sein, damit die Drahtbrücke D entfernt ist. Bei den Spielen im letzten Abschnitt dieses Kapitels werden wir darauf zurückkommen.

Wir wollen in einem kleinen Testprogramm feststellen, daß die Drahtbrücke B vorhanden ist. Wir überprüfen damit zugleich, ob der richtige Stecker des Verbindungskabels in der DIN-Buchse der Mikroprozessor-Platine steckt.

NR.	MARKE	IMNEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	CALL BLANK	1000	12	
2.	WDH.	AND S, $\bar{0}$	1001	39 00	
3.		ILD A,S	1003	06	
4.		IST A,LED-R	1004	CD C3	(LED-R):=(S)
5.		IST A,ZWSP	1006	CD D8	(ZWSP):=(S)
6.		ICALL UEB 2	1008	15	
7.		ILD A, $\bar{16}$	1009	CD 10	
8.		ICALL ANZEIGE	100B	11	ANZEIGE
9.		IBRA WDH.	100C	74 F3	

DATENSPEICHER	BEZEICHNUNG	ADR.	BEMERKUNGEN
	LED-R	1FFC3	LED-REIHE
	ZWSP	1FFD8	ZWISCHENSPEICHER

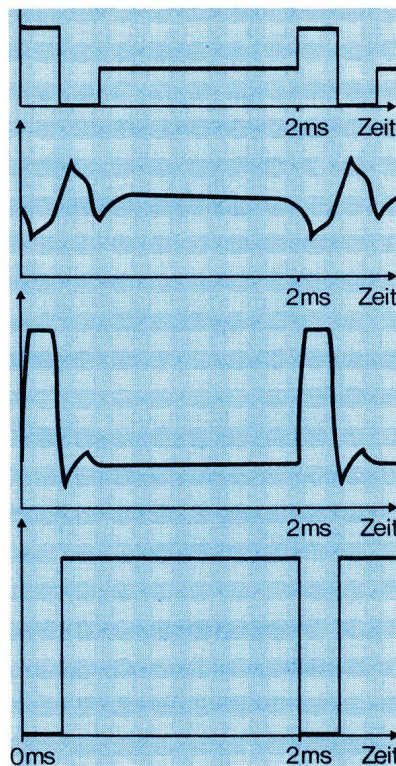
Wenn wir dieses kleine Testprogramm starten, wird der Inhalt des Status-Registers hexadezimal an zwei Sieben-Segment-Anzeigen und dual an der LED-Reihe angezeigt. Ohne angestecktes Verbindungskabel (d.h. ohne Drahtbrücke D) erhalten wir die Anzeige 00. Der Computer registriert eine Betätigung der Tasten **SA** und **SB**. Betätigen wir eine dieser Tasten, so wird das entsprechende Bit im Status-Register gesetzt. Stecken wir das Verbindungskabel zum Cassettenrecorder an, so wird dadurch (SB) = 1. Die Betätigung der Taste **SA** wird wie vorher wahrgenommen, eine Betätigung der Taste **SB** kann nicht mehr erkannt werden. Wenn wir die Impulsfolge, die wir im Abschnitt 19.1.1 mit Hilfe der Tasten **SP** und **E** erzeugt und auf Cassette geschrieben haben, jetzt wieder in das Mikroprozessorsystem einlesen, dann haben sich die Impulse sicherlich in der Form etwas verändert. Wichtig ist nur, daß die Interface-Schaltung die einzelnen Impulse eindeutig aufbereiten kann.

Impulse, wie sie vom Mikroprozessor erzeugt werden, vgl. Abschnitt 19.1.1

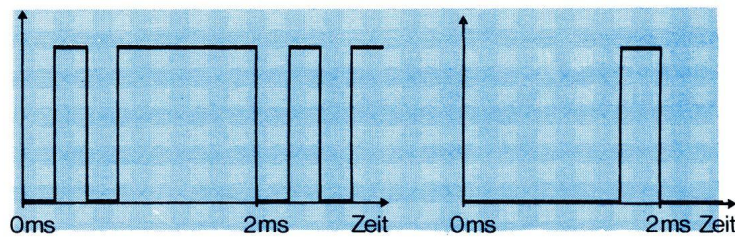
Impulse, die vom Cassettenrecorder zurückgeliefert werden

Impulse am Ausgang der Transistorstufe (Punkt P)

Impulse am SB-Eingang des Mikroprozessors



Wenn der Cassettenrecorder beim Überspielen automatisch gesteuert wird, müßten die Signale eindeutig vom Mikroprozessor erkannt werden. Ist das Überspielen mit Hilfe von Lautstärke- und Klangregler zu beeinflussen (wenn z. B. der Lautsprecher-Ausgang des Cassettenrecorders verwendet wird), so muß eine geeignete Einstellung gefunden werden. Bei zu geringer Lautstärke werden die Signale verzerrt. Es entstehen u. U. in zwei Millisekunden zwei schmale oder sogar ein breites 0-Signal am SB-Eingang des Mikroprozessors.



Den Klangregler sollte man auf möglichst dunkel (Bässe) stellen. Werden die Obertöne zusätzlich übertragen, so führt das auch zur Verdopplung der 0-Signale.

Mit Hilfe eines weiteren Testprogramms, das wir mit den Tasten **SP** und **F** aufrufen können, läßt sich die Lautstärkeeinstellung überprüfen. Nach Betätigung dieser beiden Tasten leuchtet die gelbe Leuchtdiode auf. Wenn wir jetzt vom Cassettenrecorder die vorher aufgezeichnete Impulsfolge zurückspielen, müssen die gelbe und die grüne Leuchtdiode gleichmäßig, aber schwach leuchten. Bei zu geringer Lautstärke leuchtet nur die gelbe Leuchtdiode. Bei Erhöhung der Lautstärke leuchtet zusätzlich die grüne Leuchtdiode. Bei zu großer Lautstärke beginnen die Leuchtdioden zu blinken, schließlich werden sie wieder dunkler.

Die richtige Einstellung ist gefunden, wenn die Lautstärke etwas größer ist als die, bei der die grüne Leuchtdiode aufleuchtet. Die Einstellung des Klangreglers können wir mit diesem Testprogramm nicht überprüfen.

19.2 Speichern und Laden von Programmen

Nicht jeder Cassettenrecorder gestattet das Mithören bei der Aufnahme oder Wiedergabe. Damit auch bei der Benutzung solcher Geräte das Signal akustisch verfolgt werden kann, wird die Zusatzelektronik gem. Abb. auf Seite 15 aufgebaut.

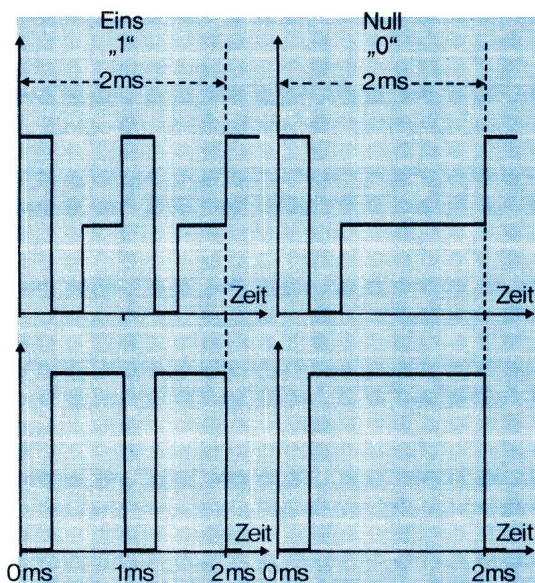
Bei der Aufnahme (speichern des Programms auf Cassette) ist der Punkt A mit F2 des Microcomputers zu verbinden. Bei der Wiedergabe (laden des Programms von Cassette) wird der Punkt A mit SB der Steckerleiste verbunden.

Die Hauptaufgabe des Cassettenrecorders besteht im Zusammenhang mit dem Computer darin, Programme zu speichern. Wenn wir ein Programm entwickelt und eingetastet haben, steht es im RAM. Dort steht es aber nur solange, wie die Spannungsversorgung für den Computer eingeschaltet ist. Beim Ausschalten der Versorgungsspannung geht das schönste Programm verloren; es sei denn, wir schreiben es vorher auf Cassette. Dann steht uns das Programm jederzeit wieder zur Verfügung.

Es wird sicher so kommen, daß wir uns eine Programm-bibliothek anlegen, eine Sammlung von Programmen, die uns besonders gut gefallen. Daneben werden wir den Cassettenrecorder auch bei der Programmentwicklung einsetzen. Auf der Cassette können wir Teilprogramme abspeichern, die wir später ergänzen oder verbessern wollen.

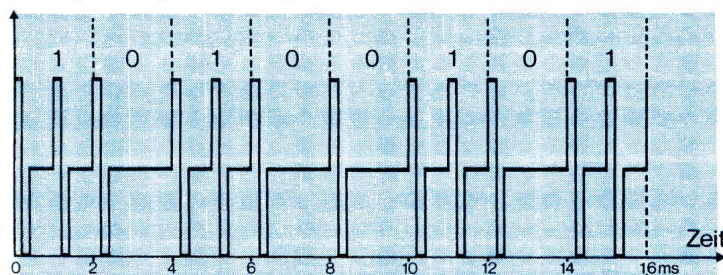
Wie werden jetzt unterschiedliche Speicherinhalte auf der Cassette gespeichert? Wir wissen schon, daß das ganze zu überspielende Programm Byte für Byte, innerhalb eines Bytes Bit für Bit übertragen wird. Das geschieht mit Hilfe von Impulsen, wie wir sie im Abschnitt 19.1 kennengelernt haben. Es müssen jetzt nur noch Einsen und Nullen unterschieden werden. Die beiden Teile des Betriebsprogramms für das Abspeichern und das Laden von Programmen unterscheiden eine Eins und eine Null dadurch: Bei einer Eins werden zwei Impulse in zwei Millisekunden, bei einer Null wird ein Impuls in zwei Millisekunden übertragen.

beim Abspeichern auf
Cassette:

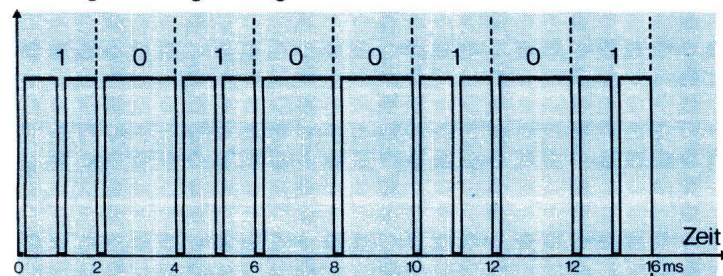


beim Laden von Cassette:

Beim Abspeichern des Bytes „A5“ auf Cassette wird folgende Impulsfolge übertragen:



Beim Laden empfängt der Mikroprozessor entsprechend die folgende Signalfolge:



Wir können den Übertragungsmechanismus auch so formulieren: Jeweils nach zwei Millisekunden wird ein Kontrollbit übertragen. Wird zwischen zwei Kontrollbits ein weiteres Bit gesendet, so entspricht das einer Eins, wird zwischen zwei Kontrollbits kein weiteres Bit gesendet, so handelt es sich um eine Null.

19.2.1 „Save“

Wenn die Verbindung zwischen Computer und Cassettenrecorder hergestellt worden ist (vgl. Abschnitt 19.1), ist das Überspielen einfach. Angenommen im RAM steht das Programm von Abschnitt 18.3.7 „Messung der Reaktionszeit“. Dieses Programm soll auf Cassette gespeichert werden. Der Computer muß dazu von uns wissen, wo das Programm steht. Wir betätigen die Taste **SV** (engl. to save – retten, aufbewahren), der Computer erwartet die Eingabe der Startadresse. Wir geben die Adresse 1000 ein, da hier unser Programm beginnt. Nach erneuter Betätigung der Taste **SV** teilen wir dem Computer die Endadresse mit: 103D. Der letzte Befehl unseres Programms beginnt im Speicher mit der Adresse 103C, aber das Byte „C2“ im Speicher 103D gehört auch noch zum Programm. Jetzt schalten wir den Cassettenrecorder auf Aufnahme und betätigen wieder die

Taste **SV**. Falls wir den Lautsprecher des Recorders nicht ausgeschaltet haben, hören wir zunächst einen gleichmäßig hohen Ton (eine Folge von Kontrollbits), dann ein Schnarren. Der erste Teil ist nur ein Vorspann, die eigentliche Programmüberspielung geschieht im zweiten Teil. Der Computer meldet sich mit der Endadresse und dem Inhalt dieses Speichers, in unserem Beispiel also mit der Anzeige

1 0 3 D 7 C 2 .

Damit haben wir das Programm auf Cassette gerettet.

Auf dem Band stehen jetzt die Anfangsadresse (in unserem Beispiel: 1000), die Anzahl der Bytes ($62_{10} = 3E_{16}$) und eben die Inhalte der 62 Speicher. Natürlich steht auf dem Band nur eine Folge unterschiedlicher Signale, die Nullen und Einsen entsprechen. Es sind etwa 500 Bits übertragen worden. Der eigentliche Überspielvorgang hat ungefähr eine Sekunde gedauert, für die Übertragung eines Bits wird ja eine fünfhundertstel Sekunde benötigt.

19.2.2 „Load“

Zum Laden des Programms von Cassette in den RAM-Speicher unseres Computers benutzen wir die Taste **LD** (engl. to load – laden). Mit dieser Taste rufen wir einen Teil des Betriebsprogramms im ROM auf, das dafür sorgt, daß die von der Cassette kommenden Signale wieder als Nullen und Einsen in die richtigen Speicher geschrieben werden. Wir wollen das Programm, das auf die Cassette abgespeichert wurde, wieder in den Computer laden. Wenn wir die Taste **LD** drücken, fragt uns der Computer nach der Offset-Adresse (engl. offset – wegsetzen, verschieben). Der Computer hatte ja die alte Anfangsadresse 1000 und die Anzahl der Bytes auch auf das Band geschrieben. Im Normalfall werden wir ihm jetzt die Offset-Adresse 0000 geben. Das heißt für ihn, er soll das Programm wieder dorthin schreiben, wo er es ursprünglich vorgefunden hat, er soll das Programm also nicht verschieben.

Nach der Eingabe der Offset-Adresse starten wir den Cassettenrecorder und betätigen noch einmal die Taste **LD**. (Wir können auch erst die Taste **LD** drücken und dann den Recorder starten). Der Computer meldet sich nach dem Ladevorgang mit der Adresse 0300 und dem Inhalt „01“, der im Speicher mit dieser Adresse steht. Wir können jetzt das eingelesene Programm mit Hilfe der Tasten **A↔D**, **RUN** starten.

Falls sich der Computer am Ende der Programmüberspielung nicht meldet oder „Error“ anzeigt, war er mit den empfangenen Signalen nicht zufrieden. Wir müssen alle Einstellungen noch einmal überprüfen (Lautstärkeregler, Klangregler, Überspielkabel) und den Ladevorgang neu starten.

19.3 Verschieben von Programmen

Wir wollen in diesem Abschnitt unterschiedliche Programme bzw. Programmteile verschieben. Es kommt uns hier in erster Linie auf die Technik des Verschiebens an. Zunächst wollen wir das Programm für das Spiel 2: „gelbes Blinklicht“ aus dem ROM (Adresse 03CC bis 03D8) in den RAM (ab Adresse 1000) verschieben. Wir schreiben das Programm auf Cassette: **SV**, **3**, **C**, **C** (Startadresse: 03CC), **SV**, **3**, **D**, **8** (Endadresse: 03D8), Cassettenrecorder auf Aufnahme, **SV**. Jetzt soll das Programm in den RAM geladen werden. Welche Offset-Adresse müssen wir wählen? Im ROM beginnt das Programm ab Adresse 03CC. Diese Anfangsadresse muß so vergrößert (verschoben) werden, daß sich die neue Anfangsadresse 1000 ergibt.

Wir berechnen die Differenz:

$$\begin{array}{r} 1 \rightarrow 0 \rightarrow 0 \rightarrow 0 \\ 1 \rightarrow 1 \rightarrow 1 \rightarrow 1 \\ \hline C \quad 3 \quad 4 \end{array} -$$

Also: alte Anfangsadresse
plus Offset-Adresse

neue Anfangsadresse

$$\begin{array}{r} 0 \quad 3 \quad C \quad C \\ 0 \quad C \quad 3 \quad 4 \\ \hline 1 \leftarrow 1 \leftarrow 1 \leftarrow 1 \\ 1 \quad 0 \quad 0 \quad 0 \end{array} +$$

Wir starten den Ladevorgang (nach entsprechendem Rückspulen der Cassette): **LD**, **C**, **3**, **4** (Offset-Adresse: 0C34), Cassettenrecorder auf Wiedergabe, **LD**.

Das Programm steht jetzt in den Speichern 1000 bis 100C (vgl. Programmausdruck im Abschnitt 14.4). Wir können es mit Hilfe der Tasten **A→D**, **RUN** starten.

Wenn wir jetzt den Inhalt 00 im Speicher mit der Adresse 1009 durch 20 ersetzen, blinkt die gelbe Leuchtdiode schneller. Wir haben damit gesehen, wie ein Programmteil aus dem ROM in den RAM geladen und dort verändert werden kann.

Wir wollen jetzt das veränderte Programm im RAM verschieben, um zu zeigen, daß wir damit Platz für Programm-ergänzungen schaffen können.

„Save“: Startadresse: 1000; Endadresse: 100C, „Load“: Offset-Adresse: 0020.

Unser Programm steht jetzt in den Speichern 1020 bis 102C.

Wir können es zur Kontrolle starten:

CPU, **0**, **1**, **0**, **2**, **0**, **CPU**, **RUN**, **RUN**.

ADRESSE	OP. CODE
1020	C4 00
1022	07
1023	06
1024	E4 04
1026	07
1027	84 00 20
102A	1D
102B	74 F6

Jetzt können wir das Programm ergänzen. Wir schreiben zwei Befehle davor, um einen Zähler zu setzen. Den unbedingten Branch-Befehl am Programmende überschreiben wir. Das veränderte Programm läßt die gelbe Leuchtdiode achtmal aufblinken. Zur Kontrolle können wir auch das testen:

CPU, **0**, **1**, **0**, **1**, **C**, **CPU**, **RUN**, **RUN**.

ADRESSE	OP. CODE
1020	C4 00
1022	07
1023	06
1024	E4 04
1026	07
1027	84 00 20
102A	1D
102B	74 F6

ADRESSE	OP. CODE
101C	C4 10
101E	CD E0
1020	C4 00
1022	07
1023	06
1024	E4 04
1026	07
1027	84 00 20
102A	1D
102B	9D E0
102D	7C F4
102F	1E

Dieses letzte Programm (ohne den Befehl: CALL TEST; 1E) soll im nächsten Programm zweimal eingesetzt werden. Wir retten es auf Cassette.

„Save“: Startadresse: 101C; Endadresse: 102E.

Dieser „gerettete“ Programmteil soll jetzt einmal ab Adresse 1008, zum anderen ab Adresse 1023 geladen werden. Wir berechnen die Offset-Adressen:

$$\begin{array}{r} 1 \rightarrow 1 \rightarrow 0 \rightarrow 0 \rightarrow 8 \\ 1 \rightarrow 1 \rightarrow 1 \rightarrow 1 \rightarrow C \\ \hline 0 \mid F \quad F \quad E \quad C \end{array} - \begin{array}{r} 1 \quad 0 \quad 2 \rightarrow 3 \\ 1 \quad 0 \quad 1 \rightarrow C \\ \hline 0 \quad 0 \quad 0 \quad 7 \end{array} -$$

Im ersten Fall muß die Anfangsadresse hexadezimal um 14_{16} erniedrigt werden ($-14_{16} = FFEC_{16}$), im anderen Fall muß sie um 7_{16} erhöht werden.

Wir laden jetzt den „geretteten“ Programmteil zweimal: einmal mit der Offset-Adresse FFEC, zum anderen mit der Offset-Adresse 0007. Wenn wir jetzt am Anfang fünf, in der Mitte fünf und am Ende einen Befehl ergänzen, wie im folgenden Programmausdruck zu erkennen ist, dann haben wir unser Ziel erreicht. Bei diesem Programm leuchten die rote und die grüne Leuchtdiode im Wechsel, zwischendurch blinkt die gelbe Leuchtdiode jeweils achtmal.

ADRESSE	OP. CODE
1008	C4 08
1002	07
1003	84 00 00
1006	1D
1007	1D
1008	C4 10
100A	CD E0
100C	C4 00
100E	07
100F	06
1010	E4 04
1012	07
1013	84 00 20
1016	1D
1017	9D E0
1019	7C F4
101B	C4 02
101D	07
101E	84 00 00
1021	1D
1022	1D
1023	C4 10
1025	CD E0
1027	C4 00
1029	07
102A	06
102B	E4 04
102D	07
102E	84 00 20
1031	1D
1032	9D E0
1034	7C F4
1036	24 FF 0F

Natürlich können wir dieses fertige Programm wieder auf Cassette bringen. Bei einem Verschieben dieses Programms müssen wir jedoch folgendes beachten: Der letzte Befehl ist ein unbedingter Jump-Befehl, er verursacht einen Sprung zum Speicher mit der Adresse 1000. Wenn wir dieses Programm verschieben, müssen wir anschließend den letzten Befehl verändern.

19.4 Die Programme auf der beiliegenden Cassette

Wir wollen uns jetzt die fünf Programme ansehen, die auf der beiliegenden Cassette gespeichert sind. Wenn uns die Berechnung der Offset-Adresse im letzten Abschnitt Schwierigkeiten gemacht hat, braucht uns das hier nicht zu beunruhigen. Alle fünf Programme sind ab Adresse 1000 geschrieben und auf Band gespeichert. Wir laden sie alle mit der Offset-Adresse 0000. Dann stehen sie wieder ab Adresse 1000 im RAM des Mikrocomputers. Wir verbinden unseren Computer mit dem Cassettenrecorder (gekennzeichnete Stecker des Überspielkabels zum Computer!) und laden jedes der fünf Programme wie folgt: Wir spielen das Band bis zu der Stelle, wo der gleichmäßige hohe Ton beginnt (wer einen Cassettenrecorder mit Zählwerk benutzt, wird sich den Stand des Zählwerkes notieren). Jetzt betätigen wir die Tasten **LD**, **0**, **LD** und schalten den Cassettenrecorder auf Wiedergabe. Wenn sich der Computer mit der Anzeige

0 3 0 0 0 1

meldet, ist das Programm geladen. Wir wollen diese Programme jetzt nicht ausführlich diskutieren. Teilweise werden hier Befehle benutzt, die wir in diesem Anleitungsbuch noch nicht besprochen haben. Wir geben nur zur Kontrolle die Speicherinhalte an, und zwar in folgender Kurzform: hinter einer Adresse stehen acht Speicherinhalte. Das sind die Speicherinhalte des Speichers mit der angegebenen Adresse und der sieben nachfolgenden Speicher.

19.4.1 Ein Würfelspiel

Dieses Programm haben wir im Abschnitt 18.3.6 schon besprochen. Wir wollen nur noch auf einen Punkt hinweisen: Wenn wir das Programm nach dem Laden mit den Tasten **A↔D** und **RUN** starten, erlischt die Anzeige, es sei denn, wir haben daran gedacht, den DIN-Stecker aus der Buchse an der Computer-Platine herauszuziehen. Bei diesem Spiel wird die Taste **SB** zum Würfeln benutzt. Solange das Überspielkabel angesteckt ist, hat der SB-Eingang den Wert 1, vgl. Abschnitt 19.1.2.

ADRESSE	INHALT
1000	84 00 00 CD E3 8D E4 20
1008	79 10 C4 08 11 06 D4 30
1010	6C F8 E4 10 6C 46 20 CE
1018	06 D4 07 6C F9 CD E3 E4
1020	07 6C F3 06 D4 20 7C EE
1028	C5 E4 F5 E3 CD E4 FC 15
1030	06 D4 80 6C D2 C4 10 CD
1038	E6 20 79 10 C4 20 11 84
1040	00 00 8D C0 C4 10 11 9D
1048	E6 7C EE C4 10 CD D8 15
1050	C5 E4 E4 15 6C 15 C4 80
1058	CD C6 74 0F C5 E4 FC 0F
1060	CD D8 D4 F0 7C E5 15 C4
1068	00 CD C5 84 7D E6 8D C0
1070	C4 08 11 06 D4 20 6C F8
1078	74 86 85 E4 1C 8D D8 13
1080	C5 E3 CD D8 15 C4 00 CD
1088	C5 5C

19.4.2 Das Spiel „Zahlen erkennen“

Auch bei diesem Spiel muß das Überspielkabel von der Computer-Platine abgezogen werden. Auch hier wird die Taste **SB** benutzt.

ADRESSE	INHALT
1000	26 00 FD 20 CE 06 C4 FF
1008	CA 00 C4 A7 CA 10 06 D4
1010	20 6C F0 06 D4 20 7C FB
1018	C5 E0 CD D8 15 85 C4 8D
1020	E3 12 C4 01 CD E5 84 00
1028	00 8D E6 C4 08 CD E8 C5
1030	E5 3E CD E5 D5 E4 01 C5
1038	E5 D5 E3 8D C4 C4 20 11
1040	06 D4 20 7C 11 9D E8 7C
1048	E6 85 E6 B4 01 00 8D E6
1050	1C 8D D8 13 74 D5 84 00
1058	00 8D C4 11 16 C5 D8 E5
1060	E0 7C 10 85 E6 1C 8D D8
1068	13 C5 E0 CD D8 15 10 00
1070	00 74 8D 85 E6 1C 8D D8
1078	13 C4 00 11 85 E6 B4 05
1080	00 8D E6 1C 8D D8 13 C4
1088	00 11 74 9F

Wenn wir das Programm starten, werden uns acht Fragezeichen angezeigt. Mit Hilfe des Zufallszahlengenerators wird eine Zufallszahl ermittelt. Dieser Vorgang wird solange wiederholt, bis wir die Taste **SB** drücken. Jetzt wird uns die zweistellige, zufällig gewählte Hexadezimalzahl angezeigt, allerdings in einer ganz ungewohnten und schwer erkennbaren Form. Die entsprechenden Segmente werden uns nacheinander gezeigt. Gleichzeitig wird gezählt, wie oft wir alle Segmente gesehen haben. Wenn wir die Zahl erkannt haben, können wir den Anzeigevorgang mit der Taste **SB** anhalten. Der Computer wartet jetzt auf eine zweistellige Eingabe. Wir tasten zwei Ziffern ein und beenden die Eingabe mit einer Funktionstaste (außer **A↔D**). Haben wir die zweistellige Zahl richtig erkannt und eingegeben, zeigt uns der Computer außer dieser Zahl, wie oft er die einzelnen Segmente angesteuert hatte. Wir können das Spiel mit einer Ziffern- oder Funktionstaste neu starten. Haben wir die zweistellige Zahl nicht richtig erkannt, erhalten wir fünf Strafunkte, und das Spiel läuft weiter.

19.4.3 Das Spiel „NIMM“

Das NIMM-Spiel ist ein Spiel für zwei Personen. Man einigt sich auf eine Anfangsmenge von Streichhölzern (oder von anderen Gegenständen) und legt fest, wie viele Streichhölzer bei einem Spielzug maximal weggenommen werden dürfen (ein Streichholz muß mindestens weggenommen werden). Die Spieler sind abwechselnd an der Reihe. Gewinner oder Verlierer – ja, das hängt auch von den Abmachungen ab – ist derjenige, der das letzte Holz nimmt bzw. nehmen muß. Wir spielen das NIMM-Spiel gegen den Computer. Wir starten das Programm. Es erscheint die Anzeige „1–2“. Jetzt sollen wir uns für eine der beiden möglichen Spielregeln entscheiden. Spielregel 1: Verlierer ist, wer das letzte Streichholz wegnehmen muß, Spielregel 2: Gewinner ist, wer das letzte Streichholz wegnehmen kann.

Wir betätigen eine der Tasten **1** oder **2**. Der Computer erwartet von uns jetzt die Eingabe einer maximal zweistelligen Dezimalzahl (also höchstens 99), er möchte wissen, mit wie vielen Streichhölzern das Spiel begonnen wird. Wenn wir die Eingabe mit einer Funktionstaste (außer **A↔D**) abgeschlossen haben, erwartet er von uns die Festlegung, wie viele Streichhölzer bei einem Spielzug weggenommen werden dürfen. Wir geben eine einstellige Dezimalzahl ein. Jetzt muß nur noch entschieden werden, wer anfängt. Drücken wir die Taste **C**, beginnt der Computer, bei jeder anderen Taste (außer **A↔D**) beginnen wir.

ADRESSE	INHALT
1000	84 02 00 8D 08 13 84 80
1008	0C 8D C1 10 74 FD E4 01
1010	6C 08 E4 03 7C F5 CD E0
1018	74 04 C4 01 CD E0 16 C5
1020	D8 6C FB 20 EF 10 85 D8
1028	1B 8D E1 20 1D 11 84 7E
1030	00 8D C0 10 74 FD 6C FB
1038	CD E3 CD E4 CD E6 20 FB
1040	10 CD E5 95 E4 20 14 11
1048	C4 EE CD C0 10 00 00 E4
1050	0C 6C 31 C4 A7 CD C0 10
1058	74 FD 6C FB CD E7 FD E4
1060	06 D4 80 6C 01 1A 20 0F
1068	11 11 11 11 C5 E1 FD E7
1070	6C 50 06 D4 80 7C 07 12
1078	11 20 14 11 74 D5 C5 E1
1080	FD E7 CD E1 A5 E4 85 E1
1088	0D A5 E4 2C 0B 01 C5 E1
1090	78 48 7C 08 C5 E0 6C 08
1098	C5 E3 74 08 E4 01 7C 04
10A0	C4 01 74 03 40 FD E0 CD
10A8	E6 20 14 11 C4 72 CD C0
10B0	11 11 11 C5 E1 FD E6 CD
10B8	E1 7C BE 12 C5 E0 6C 07
10C0	74 11 12 C5 E0 6C 0C 84
10C8	F2 E6 8D C6 84 EC 72 8D
10D0	C4 74 0A 84 0C DA 8D C6
10D8	84 FA F2 8D C4 10 00 00
10E0	24 FF 0F
10F0	48 D4 F0 FC A0 06 D4 80
10F8	6C 01 1A 40 D4 0F FC 0A
1100	06 D4 80 6C 01 1A 5C
1110	C5 E7 CD D8 13 C5 E6 CD
1118	D8 15 C5 C4 CD C2 85 E1
1120	1C 8D D8 15 C5 E4 7E
1128	7C 04 C4 00 CD C5 5C

1. Beispiel:

1

-

2

Taste **1**, wir entscheiden uns für Spielregel 1: wer das letzte Holz wegnehmen muß, verliert.

Tasten **1**, **5**, **ME+**, die Anfangsmenge besteht aus 15 Streichhölzern.

1

5

Taste **3**, drei Hölzer dürfen bei jedem Spielzug maximal weggenommen werden.

1

5

Taste **3** (oder andere Taste außer **C**), wir beginnen.

1

5

Taste **2**, wir nehmen zwei Streichhölzer weg.

1

5

Es sind noch 13 Streichhölzer vorhanden, der Computer wird 1 Streichholz wegnehmen.

1

3

Es sind jetzt noch 12 Streichhölzer vorhanden.

1

2

Der letzte Computerzug wird weiter angezeigt. Wir sind an der Reihe.

1

2

Taste **3**, wir nehmen drei Hölzer.

Taste **3**, wir nehmen drei Hölzer.

Taste **3**, wir nehmen wieder drei Hölzer.

Der Computer muß das letzte Streichholz nehmen.

Er signalisiert uns den Sieg.

Mit jeder Ziffern- oder Funktionstaste können wir ein neues Spiel starten.

2. Beispiel:

Taste **2**, Spielregel 2: wer das letzte Holz nehmen kann, gewinnt.

Tasten **9**, **9**, **ME+**, Ausgangszahl: 99.

9

9

Taste **9**, 9 Streichhölzer dürfen maximal weggenommen werden.

9

9

Taste **C**, der Computer beginnt.

9

9

Er nimmt 9 Streichhölzer.

9

0

Taste **5**, wir nehmen 5 Hölzer.

9

0

Taste **7**, wir nehmen 7 Hölzer.

Wenn wir uns nicht an die Spielregeln halten und bei einem Spielzug mehr Streichhölzer wegnehmen, als wir es nach der Verabredung dürfen, dann erfolgt die Anzeige „Error“. Wenn wir zum Schluß mehr Hölzchen wegnehmen wollen, als noch vorhanden sind, dann läßt uns der Computer unseren Zug wiederholen.

19.4.4 Eine Digitaluhr

ADRESSE	INHALT
1000	16 85 D8 CD E3 1B 8D E8
1008	FC 18 06 D4 80 6C 01 1A
1010	16 85 D8 CD E2 1B 8D E6
1018	FC 3C 06 D4 80 6C 01 1A
1020	16 85 D8 CD E1 1B 8D E4
1028	FC 3C 06 D4 80 6C 01 1A
1030	C5 E1 CD D8 15 85 C4 8D
1038	C0 85 E2 8D D8 14 85 C4
1040	8D C3 C4 00 CD C5 C4 BB
1048	11 CB 10 95 E4 85 E4 1C
1050	CD E1 E4 60 6C 07 84 10
1058	01 1D 40 74 D3 40 CD E1
1060	CD E4 95 E6 1C CD E2 E4
1068	60 6C 07 84 EC 00 1D 00
1070	74 BE 40 CD E2 CD E6 95
1078	E8 1C CD E3 E4 24 6C 09
1080	84 C6 00 1D C5 E3 00 74
1088	A7 40 CD E3 CD E8 84 C5
1090	00 1D 00 00 74 9A

Wir starten das Programm. Der Computer erwartet von uns die Eingabe der aktuellen Zeit in Stunden, Minuten und Sekunden. Angenommen, wir wollen unsere Uhr genau um 17 Uhr, 45 Minuten und 30 Sekunden starten. Dann tasten wir: **1**, **7**, **ME+**, **4**, **5**, **ME+**, **3**, **0** und starten unsere Uhr genau zu der eingegebenen Zeit durch erneute Betätigung der Taste **ME+**.

1 **7** **4** **5** **3** **0**

Die Uhr läuft, bis wir die **RS**-Taste drücken. Die Genauigkeit der Uhr ist davon abhängig, wie genau der Quarz auf der Mikroprozessorplatine die angegebene Frequenz von 4 MHz (4 000 000 Schwingungen pro Sekunde) einhält. Der Quarz ist in der vorliegenden Schaltung nicht abgestimmt. Seine Frequenz kann bis zu 300 Hz abweichen. Das wäre ein Fehler von weniger als einem hundertstel Prozent. Unsere Uhr kann daher auch ungenau gehen. Pro Tag ist eine Abweichung bis zu sechs Sekunden möglich. Wir können in einem solchen Fall unsere Uhr per Software genauer machen. Im Speicher mit der Adresse 1057 steht der Inhalt 10₁₆. Wir können diesen Inhalt erhöhen (11₁₆, 12₁₆, 13₁₆, ...). Bei einer Erhöhung um 1₁₆ wird unsere Digitaluhr pro Tag 1,5 Sekunden langsamer. Wir können den Speicherinhalt erniedrigen (0F₁₆, 0E₁₆, 0D₁₆, ...). Bei einer Erniedrigung um 1₁₆ wird unsere Digitaluhr pro Tag 1,5 Sekunden schneller. Läuft unsere Uhr z. B. nach dem von der Cassette eingelesenen Programm pro Tag um 6 Sekunden zu schnell, dann schreiben wir in den Speicher mit der Adresse 1057 den Inhalt 14₁₆. Es gibt noch feinere Korrekturmöglichkeiten, auf die wir hier aber nicht eingehen wollen.

19.4.5 Der Computer macht Musik

Wir schließen den Lautsprecher zwischen F1 (oder F2 oder F3) und + 5V an der rechten Federleiste an, vgl. Abschnitt 1.3.6. Wenn wir das Programm eingelesen und gestartet haben, ertönt Musik. Das Lied „Hänschen klein“ werden wir sicher kennen. Das eigentliche Musikprogramm steht in den Speichern 1000 bis 1057. Ab Adresse 1060 stehen die Noten in einer für den Computer verständlichen Form. Es sind bestimmte Codierungen für Tonhöhe und Tonlänge. Findet der Computer in dieser Liste den Speicherinhalt 00, so beginnt er die Melodie nach einer Pause wieder von vorn. Eine zweite Melodie steht im RAM ab Adresse 10D0. Wir ersetzen bei Adresse 1002 den Inhalt 60 durch D0 und starten das Programm neu.

ADRESSE	INHALT
1000	12 26 60 10 C6 02 7C 0E
1008	C4 04 CD E0 84 00 00 1D
1010	9D E0 7C F8 74 EA C4 04
1018	CD E1 C2 FE E4 01 7C 12
1020	C2 FF CD E2 84 67 00 1D
1028	9D E2 7C F8 9D E1 7C F0
1030	74 20 84 00 00 C2 FF CD
1038	E2 06 E4 0E 07 C2 FE 1D
1040	9D E2 6C 05 C4 03 1D 74
1048	F0 74 00 C4 01 1D 9D E1
1050	7C E3 84 01 08 1D 74 AC
1060	5F 20 73 1B 73 35 6C 1C
1068	82 18 82 30 93 15 82 18
1070	73 1B 6C 1C 5F 20 5F 20
1078	5F 40 5F 20 73 1B 73 35
1080	6C 1C 82 18 82 30 93 15
1088	73 1B 5F 20 5F 20 93 40
1090	01 20 82 18 82 18 82 18
1098	82 18 82 18 73 1B 6C 39
10A0	73 1B 73 1B 73 1B 73 1B
10A8	73 1B 6C 1C 5F 40 5F 20
10B0	73 1B 73 35 6C 1C 82 18
10B8	82 30 93 15 73 1B 5F 20
10C0	5F 20 93 40 01 20 00
10D0	82 24 82 0C 73 35 82 30
10D8	5F 40 65 78 82 24 82 0C
10E0	73 35 82 30 53 48 5F 7F
10E8	82 24 82 0C 3B 60 49 50
10F0	5F 30 5F 10 65 3C 73 35
10F8	44 40 44 15 49 50 5F 40
1100	53 48 5F 7F 00

20. Lösungen zu den Aufgaben

Lösungen zu Kapitel 5

1. Aufg. (Kap. 5)

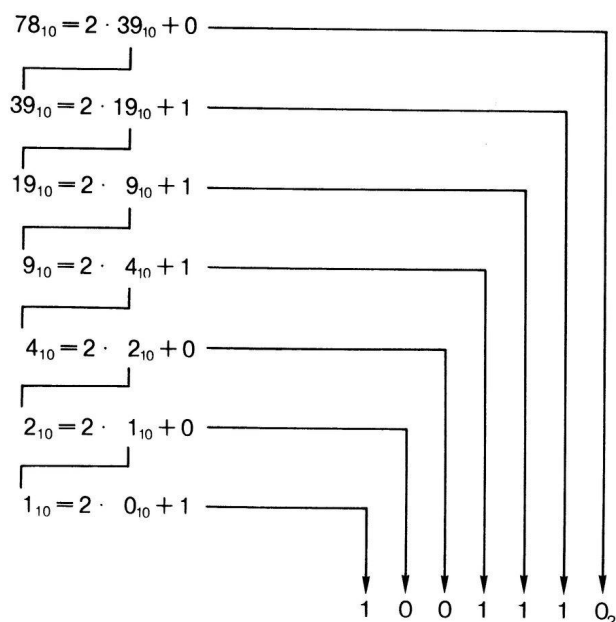
$$\begin{aligned}
 1.1 \quad 10111_2 &= 2^4 + 2^2 + 2^1 + 2^0 \\
 &= 16 + 4 + 2 + 1 \\
 &= 23_{10} \\
 1.2 \quad 10111101_2 &= 2^7 + 2^5 + 2^4 + 2^3 + 2^2 + 2^0 \\
 &= 128 + 32 + 16 + 8 + 4 + 1 \\
 &= 189_{10} \\
 1.3 \quad 1011 \ 1101_2 &= BD_{16} \\
 1111101000_2 &= 2^9 + 2^8 + 2^7 + 2^6 + 2^5 + 2^3 \\
 &= 512 + 256 + 128 + 64 + 32 + 8 \\
 &= 1000_{10} \\
 11 \ 1110 \ 1000_2 &= 3E8_{16}
 \end{aligned}$$

2. Aufg. (Kap. 5)

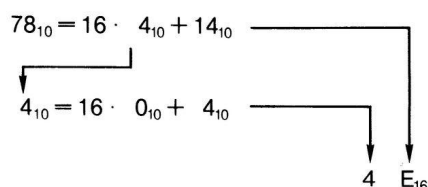
2.1 Umwandlung der Dezimalzahl 78_{10} :
Zerlegung in Zweierpotenzen:

$$\begin{aligned}
 78_{10} &= 64_{10} + 8_{10} + 4_{10} + 2_{10} \\
 &= 2^6 + 2^3 + 2^2 + 2^1 \\
 &= 1001110_2
 \end{aligned}$$

nach dem Schema in Abschnitt 5.3:



nach dem Schema in Abschnitt 5/4 (Zerlegung in Sechzehnerpotenzen)



auf dem Weg über das Dualsystem:

$$78_{10} = 100 \ 1110_2 = 4E_{16}$$

$$\begin{aligned}
 2.2 \quad 100_{10} &= 64_{10} + 32_{10} + 4_{10} \\
 &= 2^6 + 2^5 + 2^2 \\
 &= 1100100_2 \\
 100_{10} &= 110 \ 0100_2 = 64_{16} \\
 2.3 \quad 9999_{10} &= 8192_{10} + 1024_{10} + 512_{10} + 256_{10} + 8_{10} + 2_{10} + 1_{10} \\
 &= 2^{13} + 2^{10} + 2^9 + 2^8 + 2^3 + 2^2 + 2^1 + 2^0 \\
 &= 10011100001111_2 \\
 9999_{10} &= 10 \ 0111 \ 0000 \ 1111_2 \\
 &= 270F_{16}
 \end{aligned}$$

3. Aufg. (Kap. 5)

$$\begin{aligned}
 3.1 \quad A9_{16} &= 1010 \ 1001_2 \\
 A9_{16} &= 10 \cdot 16_{10} + 9_{10} \\
 &= 169_{16} \\
 3.2 \quad 100_{16} &= 1 \ 0000 \ 0000_2 \\
 100_{16} &= 1 \cdot 256_{10} \\
 &= 256_{10} \\
 3.3 \quad AF FE_{16} &= 1010 \ 1111 \ 1111 \ 1110_2 \\
 AF FE_{16} &= 10 \cdot 4096_{10} + 15 \cdot 256_{10} + 15 \cdot 16_{10} + 14_{10} \\
 &= 40960_{10} + 3840_{10} + 240_{10} + 14_{10} \\
 &= 45054_{10}
 \end{aligned}$$

Lösungen zu Kapitel 6

1. Aufg. (Kap. 6)

$$\begin{array}{c|c} 1 & 0 & 1 & 0 \\ \hline 0 & 1 & 0 & 1 \\ \hline 1 & 1 & 1 & 1 \end{array} + \begin{array}{c|c} 1 & 1 & 1 & 1 \\ \hline 0 & 0 & 1 & 1 \\ \hline 1 & 0 & 0 & 1 \end{array} + \begin{array}{c|c} 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 \end{array} + \begin{array}{c|c} 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 \end{array}$$

2. Aufg. (Kap. 6)

$$\begin{aligned}
 2.1 \quad 1010_2 &= 10_{10} \\
 101_2 &= 5_{10} \\
 10 &+ 5 &= 15_{10} &= 1111_2 \\
 2.2 \quad 1111_2 &= 15_{10} \\
 11_2 &= 3_{10} \\
 15 &+ 3 &= 18_{10} &= 10010_2 \\
 2.3 \quad 1111_2 &= 15_{10} \\
 1111_2 &= 15_{10} \\
 15 &+ 15 &= 30_{10} &= 11110_2
 \end{aligned}$$

3. Aufg. (Kap. 6)

Für eine Erweiterung des vierstelligen Addierwerkes von Abschnitt 6.3, für die Addition achtstelliger Dualzahlen sind ein Halbaddierer und sieben Volladdierer erforderlich. Die Summe zweier achtstelliger Dualzahlen kann maximal neunstellig werden.

4. Aufg. (Kap. 6)

Die größte achtstellige Dualzahl ist $1111\ 1111_2$ ($= 255_{10} = FF_{16}$). Die größte Summe bei der Addition zweier achtstelliger Dualzahlen ist dann:

$$1111\ 1111_2 + 1111\ 1111_2 = 1\ 1111\ 1110_2$$

$$\begin{array}{r} 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1 \\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1 \\ \hline 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1 \\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 0 \end{array} +$$

Das Ergebnis läßt sich umwandeln:

$$\begin{aligned} 1\ 1111\ 1110_2 &= 2^8 + 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 \\ &= 256 + 128 + 64 + 32 + 16 + 8 + 4 + 2 \\ &= 510_{10} \\ &= 255_{10} + 255_{10} \\ 1\ 1111\ 1110_2 &= 1FE_{16} \\ &= FF_{16} + FF_{16} \end{aligned}$$

Lösungen zu Kapitel 7

$$\begin{array}{r} B\ 5 \\ 2\ C \\ \hline 1\ 1 \\ E\ 1 \end{array} +$$

$$\begin{array}{r} 9\ E \\ D\ 5 \\ \hline 1\ 1 \\ 1\ 7\ 3 \end{array} +$$

$$\begin{array}{r} A\ 7\ 3\ F \\ 5\ 8\ C\ 1 \\ \hline 1\ 1\ 1\ 1\ 1\ 1 \\ 1\ 0\ 0\ 0\ 0\ 0 \end{array} +$$

$$\begin{array}{r} 9\ D\ 3\ F \\ 1\ 2\ B\ F \\ \hline 1\ 1 \\ A\ F\ F\ E \end{array} +$$

$$\begin{array}{r} C\ 5 \\ 2\ A \\ \hline E\ F \\ 5\ D \\ \hline 1\ 1 \\ 1\ 4\ C \end{array} +$$

oder

$$\begin{array}{r} C\ 5 \\ 2\ A \\ 5\ D \\ \hline 1\ 1 \\ 1\ 4\ C \end{array} +$$

$$\begin{array}{r} F\ F \\ F\ F \\ F\ F \\ \hline 2\ 2 \\ 2\ F\ D \end{array} +$$

$$\begin{array}{r} 7\ B\ 1 \\ 1\ C\ 3\ F \\ D\ A \\ F\ E\ 6 \\ \hline 2\ 2\ 2 \\ 3\ 4\ B\ 0 \end{array} +$$

3. Aufg. (Kap. 7)

Wir wollen die Antwort zunächst für die duale Darstellung des Operationscodes formulieren:

- Das erste Byte muß mit einer 1 beginnen (vgl. Abschnitt 7.2),
- Die drei niederwertigsten Ziffern des ersten Bytes müssen 101 sein (vgl. Abschnitt 7.5).

Zusammengefaßt: Wenn das erste Byte des Operationscodes folgende vier Dualziffern enthält

$$1\ \square\ \square\ \square\ 1\ 0\ 1$$

dann handelt es sich um einen 2-Byte-Befehl mit direkter Adressierung.

Das bedeutet für die hexadezimale Darstellung des Operationscodes:

- Die erste Ziffer des ersten Bytes muß eine Ziffer zwischen 8 und F (einschließlich) sein,
- Die zweite Ziffer des ersten Bytes muß 5 oder D sein.

Es sind daher theoretisch sechzehn 2-Byte-Befehle mit direkter Adressierung möglich: 85 XX, 8D XX, 95 XX, 9D XX, A5 XX, ..., F5 XX, FD XX. Drei dieser Befehle kennen wir bisher. Wir werden in den weiteren Kapiteln dieses Anleitungsbuches sehen, daß bei unserem Mikroprozessor vierzehn der sechzehn Möglichkeiten ausgenutzt werden.

Lösungen zu Kapitel 8

1. Aufg. (Kap. 8)

Der Befehl ST A,E würde den Inhalt des Akkus in das Extension-Register bringen. Dasselbe macht der Befehl LD E,A.

2. Aufg. (Kap. 8)

MNEM. CODE	OP. CODE
LD A,=0A5	C4 A5
LD E,A	48
LD A,=0BC	C4 BC
CALL TEST	1E

Der angegebene 3-Byte-Befehl lädt den Inhalt A5 BC in das EA-Register ((E) = A5; (A) = BC). Gäbe es diesen Befehl nicht, so müßte zunächst die Hexadezimalzahl A5 in den Akku, von dort in das Extension-Register geladen werden. Anschließend müßte der Akku neu mit dem Inhalt BC geladen werden.

Für eine Erprobung mit dem Computer muß als vierter Befehl der CALL TEST zugefügt werden.

3. Aufg. (Kap. 8)

Im 4. Befehl wird die 1. Zahl A3 5E in die Speicher FFE1 und FFE2 gebracht: (FFE1) = 5E; (FFE2) = A3.

Im 7. Befehl wird die 2. Zahl 7F 12 in die Speicher FFE3 und FFE4 gebracht: (FFE3) = 12; (FFE4) = 7F.

Im 9. Befehl wird die Summe ($A3\ 5E + 7F\ 12 = 1\ 22\ 70$) bis auf den entstehenden Übertrag, der ins CY/L-Register kommt, in die Speicher FFE5 und FFE6 gebracht: (FFE5) = 70; (FFE6) = 22.

4. Aufg. (Kap. 8)

4.1 Mit dem Befehl LD EA, DATEN (Operationscode 85 E1) werden die beiden Speicherinhalte ins EA-Register gebracht, allerdings nicht in der gewünschten Reihenfolge: (E) = (FFE2); (A) = (FFE1). Es muß hinterher noch der Befehl XCH A,E ausgeführt werden.

MNEM. CODE	OP. CODE
LD EA,DATEN	85 E1
XCH A,E	01
CALL TEST	1E

4.2 Die drei nebenstehend angegebenen Befehle leisten das Gewünschte.

MNEM. CODE	OP. CODE
LD EA,DATEN	85 EA
XCH A,E	01
ST EA,DATEN	8D EA
CALL TEST	1E

5. Aufg. (Kap. 8)

5.1

Befehl Nr.	Mnem. Code	Inhalt nach Ausführung des Befehls			
		(E)	(A)	(FFE2)	(FFE1)
1.	CALL EIN 4ST	?	?	?	?
2.	LD EA,ZWSP	05	83	?	?
3.	CALL DEZ-HEX	02	47	?	?
4.	ST EA, 1. ZAHL	02	47	02	47
5.	CALL EIN 4ST	?	?	02	47
6.	LD EA,ZWSP	03	19	02	47
7.	CALL DEZ-HEX	01	3F	02	47
8.	ADD EA, 1. ZAHL	03	86	02	47
9.	CALL HEX-DEZ	09	02	02	47
10.	CALL ANZ EA	09	02	02	47
11.	CALL TEST				

Man kann diese Aufgabe dadurch lösen, daß man sich genau überlegt, was der einzelne Befehl bewirkt. Wir können aber auch unseren Computer befragen. Wir schreiben in das Programm nacheinander an die verschiedenen Stellen einen Breakpoint (CALL TEST) und lassen das Programm vom Anfang bis zu dieser Stelle laufen. Dann können wir uns die Inhalte nach dem letzten Befehl ansehen. Wir können auch zwischen je zwei Befehle zusätzlich den Befehl CALL ANZ EA schreiben. Aber Vorsicht! Diesen CALL dürfen wir nach dem 1. und nach dem 5. Befehl nicht einfügen. Die eingegebenen Zahlen stehen noch im Zwischenspeicher (FFD9/FFD8). Diese Speicher benutzt die Betriebssoftware beim CALL ANZ EA. Sie überschreibt dann die von uns eingegebenen Zahlen.

Noch einige Bemerkungen zu der angegebenen Trace:

Im 1. und 5. Befehl werden die Summanden eingegeben. Der Summand steht dann im Zwischenspeicher. Bei dieser Eingabe werden Akku und Extension-Register benutzt. Daher stehen nach der Eingabe in diesen Registern bestimmte Werte, die uns im Zusammenhang mit unserem Additionsprogramm nicht interessieren (deshalb die Fragezeichen).

Im 2. (bzw. im 6.) Befehl wird der Summand in das EA-Register geladen, im 3. (bzw. im 7.) Befehl wird die eingegebene Dezimalzahl in die entsprechende Hexadezimalzahl umgewandelt ($583_{10} = 247_{16}$; $319_{10} = 13F_{16}$). Nach der Addition im Hexadezimalsystem wird die Summe in eine Dezimalzahl umgewandelt ($386_{16} = 902_{10}$).

5.2

Alle Programme, die wir als Lösungen angeben, sind Lösungsvorschläge. Es gibt häufig mehrere Möglichkeiten. Bei dem folgenden Programm werden die eingegebenen Dezimalzahlen und die dezimale Summe in die Speicher FFE1 bis FFE6 gespeichert. Der erste Summand wird außerdem als Hexadezimalzahl in einen Hilfsspeicher (FFE7/FFE8) gebracht.

INR.	MMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	ICALL EIN 4ST	1000	17	
2.	ILD EA,ZWSP	1001	85 D8	EINGABE DES 1. SUMMANDEN
3.	IST EA,1.ZAHL	1003	8D E1	
4.	ICALL DEZ-HEX	1005	1B	
5.	IST EA,HSP	1006	8D E7	(HSP)=1.ZAHL,HEXADEZIMAL
6.	ICALL EIN 4ST	1008	17	
7.	ILD EA,ZWSP	1009	85 D8	EINGABE DES 2. SUMMANDEN
8.	IST EA,2.ZAHL	100B	8D E3	
9.	ICALL DEZ-HEX	100D	1B	
10.	IADD EA,HSP	100E	85 E7	SUMME:=1.ZAHL+2.ZAHL
11.	ICALL HEX-DEZ	1010	1C	
12.	IST EA,SUMME	1011	8D E5	
13.	ICALL ANZ EA	1013	19	ANZEIGE DER SUMME
14.	ICALL TEST	1014	1E	

DATENSPEICHER, BEZ.	ADR.	BEMERKUNGEN
ZWSP	FFD8	ZWISCHENSPEICHER
1. ZAHL	FFD9	
	FFFE1	1. SUMMAND, DEZIMAL
2. ZAHL	FFFE2	
	FFFE3	2. SUMMAND, DEZIMAL
SUMME	FFFE4	
	FFFE5	DEZIMALWERT DER SUMME
HSP	FFFE6	
	FFFE7	HILFSSPEICHER
	FFFE8	1. SUMMAND, HEXADEZIMAL

5.3

INR.	MMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	ICALL EIN 4ST	1000	17	
2.	ILD EA,ZWSP	1001	85 D8	EINGABE DES 1. SUMMANDEN
3.	ICALL DEZ-HEX	1003	1B	
4.	IST EA,1.ZAHL	1004	8D E1	
5.	ICALL EIN 4ST	1006	17	
6.	ILD EA,ZWSP	1007	85 D8	EINGABE DES 2. SUMMANDEN
7.	ICALL DEZ-HEX	1009	1B	
8.	IST EA,2.ZAHL	100A	8D E3	
9.	ICALL EIN 4ST	100C	17	
10.	ILD EA,ZWSP	100D	85 D8	EINGABE DES 3. SUMMANDEN
11.	ICALL DEZ-HEX	100F	1B	
12.	IADD EA,1.ZAHL	1010	85 E1	SUMME:=3.ZAHL+1.ZAHL
13.	IADD EA,2.ZAHL	1012	85 E3	SUMME:=SUMME+2.ZAHL
14.	ICALL HEX-DEZ	1014	1C	
15.	ICALL ANZ EA	1015	19	ANZEIGE DER SUMME
16.	ICALL TEST	1016	1E	

DATENSPEICHER, BEZ.	ADR.	BEMERKUNGEN
ZWSP	FFD8	ZWISCHENSPEICHER
1. ZAHL	FFD9	
	FFFE1	1. SUMMAND, HEXADEZIMAL
2. ZAHL	FFFE2	
	FFFE3	2. SUMMAND, HEXADEZIMAL
	FFFE4	

Lösungen zu Kapitel 9

1. Aufg. (Kap. 9)

1.1

$$\begin{array}{r} A \text{ } 5 \text{ } B \\ 9 \text{ } F \text{ } 1 \\ 1 \text{ } \end{array} - \begin{array}{r} 6 \text{ } A \end{array}$$

1.2

$$\begin{array}{r} D \text{ } 7 \text{ } 0 \text{ } F \\ 7 \text{ } 3 \text{ } E \text{ } A \\ 1 \text{ } \end{array} - \begin{array}{r} 6 \text{ } 3 \text{ } 2 \text{ } 5 \end{array}$$

1.3

$$\begin{array}{r} C \text{ } 6 \text{ } 1 \text{ } B \text{ } 2 \\ A \text{ } 6 \text{ } 1 \text{ } B \text{ } 4 \\ 1 \text{ } 1 \text{ } 1 \text{ } 1 \end{array} - \begin{array}{r} 1 \text{ } F \text{ } F \text{ } F \text{ } E \end{array}$$

1.4 Die Differenz wird negativ. Es gibt mehrere Rechenwege:

$$\begin{array}{r} 4 \text{ } 1 \text{ } A \\ 3 \text{ } E \text{ } 7 \\ 1 \text{ } \end{array} - \begin{array}{r} 3 \text{ } 3 \end{array} \quad \begin{array}{l} \text{Wir subtrahieren die 1. Zahl von der 2.} \\ \text{Das Ergebnis ist dann: } -33_{16} \end{array}$$

$$\begin{array}{r} 1 \text{ } 3 \text{ } E \text{ } 7 \\ 4 \text{ } 1 \text{ } A \\ 1 \text{ } \end{array} - \begin{array}{r} F \text{ } C \text{ } D \end{array}$$

Wir ergänzen den Minuenden um 1000_{16} .
Nach der Subtraktion müssen wir vom Ergebnis wieder 1000_{16} abziehen oder wir müssen das Ergebnis als negative Zahl interpretieren.
Ergebnis: $FCD_{16} - 1000_{16} = -33_{16}$.

Das Fünfezehnerkomplement von 41A ist BE5, das Sechszehnerkomplement von 41A ist dann BE6. Wir addieren:

$$\begin{array}{r} 3 \text{ } E \text{ } 7 \\ B \text{ } E \text{ } 6 \\ 1 \text{ } \end{array} + \begin{array}{r} F \text{ } C \text{ } D \end{array}$$

Ergebnis: $FCD_{16} - 1000_{16}$

2. Aufg. (Kap. 9)

	2.1	2.2	2.3
gegebene Zahl	1F	B7C	8D5AE
Fünfzehnerkomplement	EO	483	72A51
Sechzehnerkomplement	E1	484	72A52

3. Aufg. (Kap. 9)

	3.1	3.2	3.3
Akkuinhalt	E5	CA	2F
entspricht der positiven Dezimalzahl	229	202	47
entspricht der negativen Dezimalzahl	- 27	- 54	- 209

Die negative Zahl findet man z. B. dadurch, daß man von der positiven Zahl 256 abzieht.

4. Aufg. (Kap. 9)

Der vierstelligen Hexadezimalzahl FFFF würde man die Zahl - 1 zuordnen, denn beim Aufwärtszählen folgt auf FFFF die Zahl 0000.

5. Aufg. (Kap. 9)

In diesem Programm wird die eingegebene vierstellige Hexadezimalzahl von 0000 subtrahiert.

NR.	MMEM.CODE	ADR.	OP.CODE
1.	ICALL EIN 4ST	1000	17
2.	ILD EA,=0	1001	84 00 00
3.	SUB EA,ZWSP	1004	BD D8
4.	ICALL ANZ EA	1006	19
5.	ICALL TEST	1007	1E

6. Aufg. (Kap. 9)

Dieses Programm ermittelt positive Differenzen richtig. Ist der Subtrahend größer als der Minuend, dann steht nach der Subtraktion im EA-Register eine vierstellige Hexadezimalzahl, die größer als 270F ist. Beim CALL HEX-DEZ ergibt sich daher die Anzeige „Error“. Im 18. Kapitel werden wir dieses Programm so erweitern, daß auch negative Differenzen richtig berechnet werden.

NR.	MMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	ICALL EIN 4ST	1000	17	EINGABE MINUEND
2.	ILD EA,ZWSP	1001	85 D8	
3.	ICALL DEZ-HEX	1003	1B	
4.	IST EA,MINU	1004	8D E0	
5.	ICALL EIN 4ST	1006	17	EINGABE SUBTRAHEND
6.	ILD EA,ZWSP	1007	85 D8	
7.	ICALL DEZ-HEX	1009	1B	
8.	IST EA,SUBT	100A	8D E2	
9.	ILD EA,MINU	100C	85 E0	BERECHNUNG DER DIFFERENZ
10.	SUB EA,SUBT	100E	BD E2	
11.	ICALL HEX-DEZ	1010	1C	
12.	ICALL ANZ EA	1011	19	
13.	ICALL TEST	1012	1E	ANZEIGE DER DIFFERENZ

DATENSPEICHER, BEZ.	ADR.	BEMERKUNGEN
ZWSP	1FFD8 1FFD9	ZWISCHENSPEICHER
MINU	1FFE0	MINUEND
SUBT	1FFE1 1FFE2 1FFE3	SUBTRAHEND

Lösungen zu Kapitel 10

1. Aufg. (Kap. 10)

Beide Befehle sind unbedingte Sprungbefehle. Es gibt zwei wesentliche Unterschiede:

- Der Jump-Befehl ist ein 3-Byte-Befehl, der Branch-Befehl ist ein 2-Byte-Befehl.
- Beim Jump-Befehl ist die Ziel- oder Sprungadresse unmittelbar im Befehl angegeben (das LOB der Adresse im zweiten Byte, das HOB im dritten Byte). Beim Branch-Befehl wird die Zieladresse relativ zum momentanen PC-Inhalt angegeben. Das Displacement kann maximal $7F_{16}$ ($= + 127_{10}$) für Vorwärtssprünge oder 80_{16} ($= - 128_{10}$) für Rückwärtssprünge sein. Es lassen sich mit diesem Branch-Befehl nicht alle Ziele erreichen.

2. Aufg. (Kap. 10)

- 2.1 Würde der Operationscode im 4. Befehl 74 FA sein, dann würde der Computer nach dem Sprungbefehl den Befehl ausführen, der im Speicher mit der Adresse 1001 steht. Da hier der Inhalt 16 steht, würde der Computer jetzt die Eingabe einer zweistelligen Hexadezimalzahl erwarten (CALL EIN 2ST).
- 2.2 Bei einem Operationscode 74 FC im 4. Befehl würde der Computer den nächsten Befehl nach dem Sprung im Speicher mit der Adresse 1003 suchen. Der Inhalt 01 in diesem Speicher wird als XCH A,E interpretiert.

3. Aufg. (Kap. 10)

- 3.1 Das Displacement 7F steht im Speicher mit der Adresse 1001. 7F wird zu dieser Adresse addiert: $1001_{16} + 7F_{16} = 1080_{16}$. Anschließend wird der Inhalt des Programmzählers um 1 erhöht. Der nächste Befehl müßte bei Adresse 1081 stehen.

NR.	MARKE	MMEM.CODE	ADR.	OP.CODE
1.	BEGINN	BRA WEITER	1000	74 7F
20.	WEITER	ICALL TEST	1081	1E

- 3.2 Das Displacement 80 steht im Speicher mit der Adresse 1101. Das Displacement wird als negatives Displacement aufgefaßt. Da das Sechzehnerkomplement von 80_{16} auch 80_{16} ist (Fünfzehnerkomplement von 80_{16} : $7F_{16}$; $7F_{16} + 1_{16} = 80_{16}$) müssen wir den Inhalt des Programmzählers hexadezimal um 80 erniedrigen: $1101_{16} - 80_{16} = 1081_{16}$. Das erste Byte des nächsten Befehls müßte bei Adresse 1082 ($1081_{16} + 1_{16}$) stehen. Zur Erprobung dieses Sprungbefehles muß das Testprogramm bei Adresse 1100 gestartet werden. Zur Erinnerung die Tastenfolge:

RS, CPU, 0, 1, 1, 0, 0, CPU, RUN, RUN.

NR.	MARKE	MMEM.CODE	ADR.	OP.CODE
20.	NEU	ICALL TEST	1082	1E
40.		BRA NEU	1100	74 80

4. Aufg. (Kap. 10)

NR.	MARKE	MNEM.CODE	ADR.	OP.CODE
1.	BEGINN	JMP WEITER	1000	24 80 10
20.	WEITER	CALL TEST	1081	1E

NR.	MARKE	MNEM.CODE	ADR.	OP.CODE
20.	NEU	CALL TEST	1082	1E
40.		JMP NEU	1100	24 81 10

Lösungen zu Kapitel 11

1. Aufg. (Kap. 11)

Nach den drei gesuchten Befehlen ist jeweils der CALL TEST ergänzt worden. Das ist bei einer Erprobung mit dem Computer erforderlich.

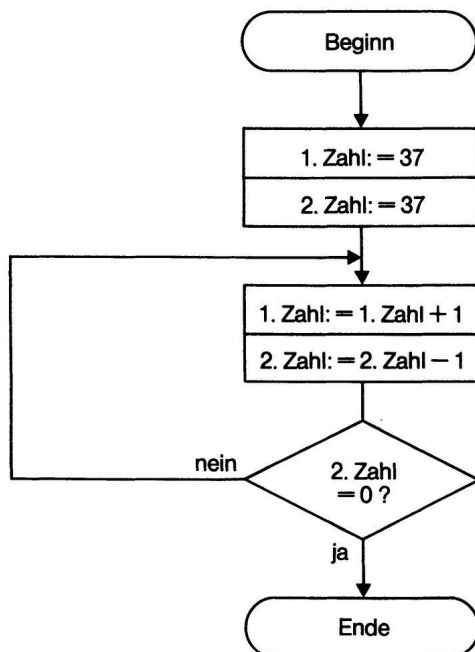
1.1

MNEM.CODE	OP.CODE
LD A,ZAHL	C5 E0
ADD A,=1	F4 01
ST A,ZAHL	CD E0
CALL TEST	1E

1.2

MNEM.CODE	OP.CODE
LD A,ZAHL	C5 E0
SUB A,=1	FC 01
ST A,ZAHL	CD E0
CALL TEST	1E

2. Aufg. (Kap. 11)



3. Aufg. (Kap. 11)

- Die Schleife wird verlassen, wenn die Zahl nicht mehr positiv ist. In diesem Zusammenhang gelten die Akkumultinhalte von 00 bis 7F als positiv. Ist die Zahl 80_{16} geworden, wird die Schleife verlassen. Diese Zahl wird angezeigt.
- Nach der ersten Ausführung der sechs NOP-Befehle wird die Zahl auf 1 erhöht, nach der zweiten Ausführung der sechs NOP-Befehle wird die Zahl auf 2 erhöht, usw. Nach der einhundertachtundzwanzigsten Ausführung der sechs NOP-Befehle wird die Zahl auf 80_{16} ($= 128_{10}$) erhöht, die Schleife wird verlassen. Die sechs NOP-Befehle werden also 128mal ausgeführt.
- Die Zahl wird auf 1 erhöht, dann folgt der erste Rücksprung zur Marke SCHL. Nach der Erhöhung der Zahl auf $7F$ ($= 127_{10}$) folgt der letzte, der 127. Rücksprung.

4. Aufg. (Kap. 11)

Wir wollen annehmen, daß im 3. Befehl die Zahl 1030 eingegeben worden ist. Diese Zahl wird in eine Hexadezimalzahl umgewandelt: $1030_{10} = 406_{16}$. Dann steht 406_{16} in den Speichern FFE3 und FFE2. Bei der Bearbeitung der Schleife wird nur der Inhalt des Speichers FFE2 dekrementiert. Nach sechs Durchläufen durch die Schleife ist der Inhalt von Speicher FFE2 Null geworden. Die Schleife wird verlassen.

406
405
404
403
402
401
+
1815

In der Schleife werden nacheinander die Zahlen 0000_{16} , 0406_{16} , 0405_{16} , 0404_{16} , 0403_{16} , 0402_{16} und 0401_{16} addiert. Das Ergebnis ist 1815_{16} . Diese Hexadezimalzahl wird nach dem Verlassen der Schleife in die Dezimalzahl 6165_{10} umgewandelt und angezeigt.

5. Aufg. (Kap. 11)

Bei der Eingabe der Zahl 0 wird nach dem Programm in Abschnitt 11.2.5 mit der Bearbeitung der Schleife begonnen. In den vier Speichern FFE0 bis FFE3 stehen Nullen. Bei der

NR.	MARKE	MNEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	LD EA,=0	1000	84 00 00	SUMME:=0000
2.		IST EA,SUMME	1003	8D E0	
3.		ICALL EIN 4ST	1005	17	EINGABE DER ZAHL
4.		ILD EA,ZWSP	1006	85 D8	
5.		ICALL DEZ-HEX	1008	1B	
6.		IST EA,ZAHL	1009	8D E2	
7.		IXCH A,E	100B	01	
8.		IBZ NULL	100C	6C 01	
9.		ICALL FEHLER	100E	1A	
10.	NULL	IXCH A,E	100F	01	
11.		IBZ ANZ.	1010	6C 0A	SPRUNG, WENN <A>=0
12.	SCHL.	ILD EA,SUMME	1012	85 E0	
13.		IADD EA,ZAHL	1014	85 E2	SUMME:=SUMME+ZAHL
14.		IST EA,SUMME	1016	8D E0	
15.		IDLD A,ZAHL	1018	9D E2	ZAHL:=ZAHL-1
16.		IBNZ SCHL.	101A	7C F6	WIEDERHOLE DIE SCHLEIFE, WENN <A> NICHT 0
17.	ANZ.	ILD EA,SUMME	101C	85 E0	
18.		ICALL HEX-DEZ	101E	1C	
19.		ICALL ANZ EA	101F	19	ANZEIGE DER SUMME
20.		IBRA BEGINN	1020	74 DE	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZWSP	FFD8	ZWISCHENSPEICHER
SUMME	FFD9	
ZAHL	FFE0	IN DIESEN SPEICHERN STEHT DIE VIERSTELLIGE SUMME
	FFE1	
	FFE2	hier steht die Zahl
	FFE3	

ersten Addition bleibt die Summe Null. Jetzt wird im 13. Befehl der Inhalt von Speicher FFE2 dekrementiert und damit gleich FF. Die Schleife wird anschließend nicht verlassen. Es wird dieselbe Rechnung durchgeführt wie bei der eingegebenen Zahl 255. Als Summe wird die Hexadezimalzahl 7F80₁₆ ermittelt. Bei der Umwandlung in die entsprechende Dezimalzahl ergibt sich die Anzeige „Error“, da die Dezimalzahl größer als 9999 ist.

Dieser Schönheitsfehler läßt sich dadurch beseitigen, daß vor Eintritt in die Schleife noch der Inhalt des Speichers FFE2 überprüft wird. Dieser Inhalt steht nach dem Sprungbefehl BZ SCHL. (8. Befehl) noch im Extension-Register. Ist dieser Inhalt Null, so muß sofort die Ergebnisanzeige folgen.

6. Aufg. (Kap. 11)

Zunächst werden die Speicherinhalte zu vollständigen Operationscodes zusammengefaßt. Man erkennt jeweils am ersten Byte, ob es sich um einen 1-, 2- oder 3-Byte-Befehl handelt.

ADRESSE	OP. CODE
1000	C4 F0
1002	CD E0
1004	84 00 00
1007	8D E1
1009	9D E1
100B	7C FC
100D	9D E2
100F	6C 02
1011	74 F6
1013	95 E0
1015	7C ED
1017	24 CB 03

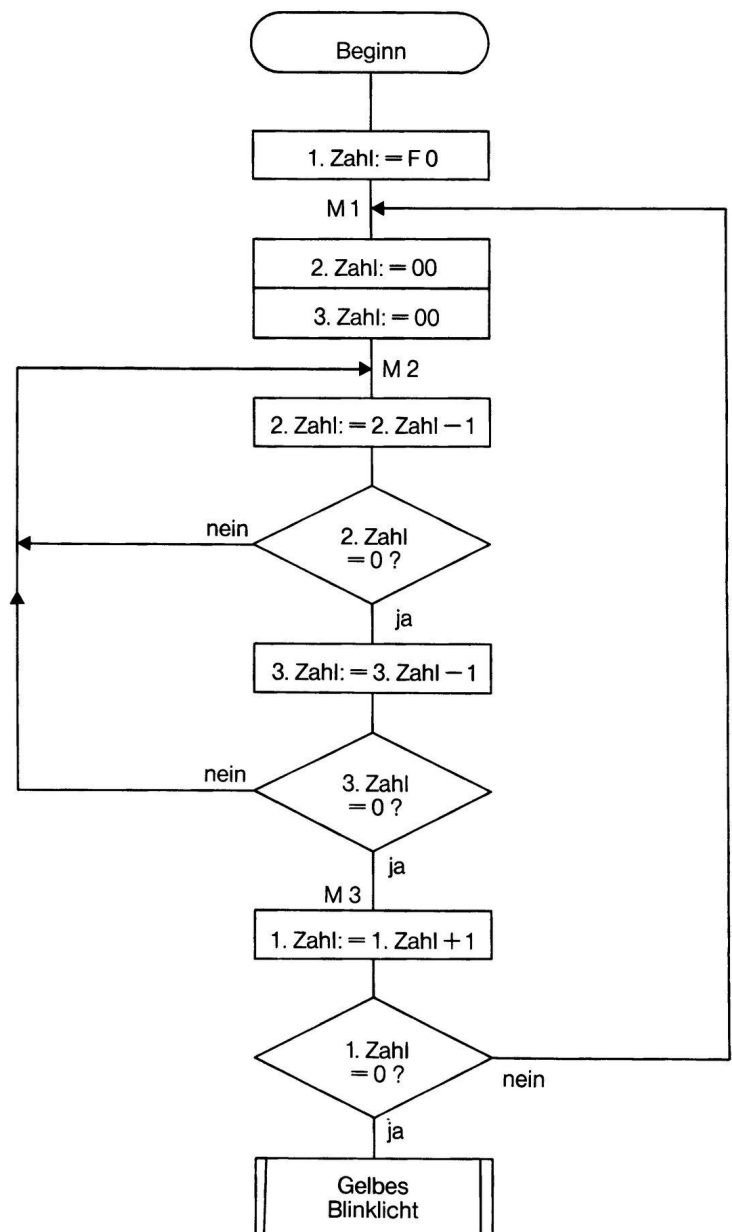
Jetzt müssen die Datenspeicher und die Marken benannt werden. Den Operationscode 24 CB 03 haben wir schon in Abschnitt 10.2 für einen Sprung zum Spiel „Gelbes Blinklicht“ benutzt.

NR.	MARKE	MNEM. CODE	ADR.	OP. CODE
1.	BEGINN	ILD A,=0F0	1000	C4 F0
2.		IST A,1,ZAHL	1002	CD E0
3.	M1	ILD EA,=0	1004	84 00 00
4.		IST EA,2,ZAHL	1007	8D E1
5.	M2	ILD A,2,ZAHL	1009	9D E1
6.		IBNZ M2	100B	7C FC
7.		ILD A,3,ZAHL	100D	9D E2
8.		IBZ M3	100F	6C 02
9.		IBRA M2	1011	74 F6
10.	M3	ILD A,1,ZAHL	1013	95 E0
11.		IBNZ M1	1015	7C ED
12.		JMP BLINK	1017	24 CB 03

DATENSPEICHER, BEZEICHNUNG	ADR.
1. ZAHL	FFE0
2. ZAHL	FFE1
3. ZAHL	FFE2

WEITERE MARKE, BEZEICHNUNG	ADR.
BLINK	03CC

Der zusätzlich angegebene Programmablaufplan vermittelt einen besseren Überblick über das Programm.



7. Aufg. (Kap. 11)

Zunächst müssen für die benutzten Datenspeicher Adressen festgelegt werden. Für den Zwischenspeicher ZWSP müssen die Adressen FFD9 und FFD8 verwendet werden. Für die Festlegung der anderen Adressen gibt es keine verbindliche Vorschrift.

DATENSPEICHER, BEZEICHNUNG	ADR.
ZWSP	FFD8
	FFD9
QUADRAT	FFE0
FAKTOR	FFE1
	FFE2
ZAEHLER	FFE3
	FFE4

Jetzt können Adressen und Operationscodes für das Programm ergänzt werden.

Lösungen zu Kapitel 12

1. Aufg. (Kap. 12)

Alle Änderungen beziehen sich nur auf das Hauptprogramm. Das Unterprogramm PAUSE bleibt unverändert. Wir erinnern aber daran, daß die Pausenlänge vergrößert werden kann, indem der Inhalt des Speichers 1102 vergrößert wird (z. B.: (1102) = 40 statt (1102) = 0A).

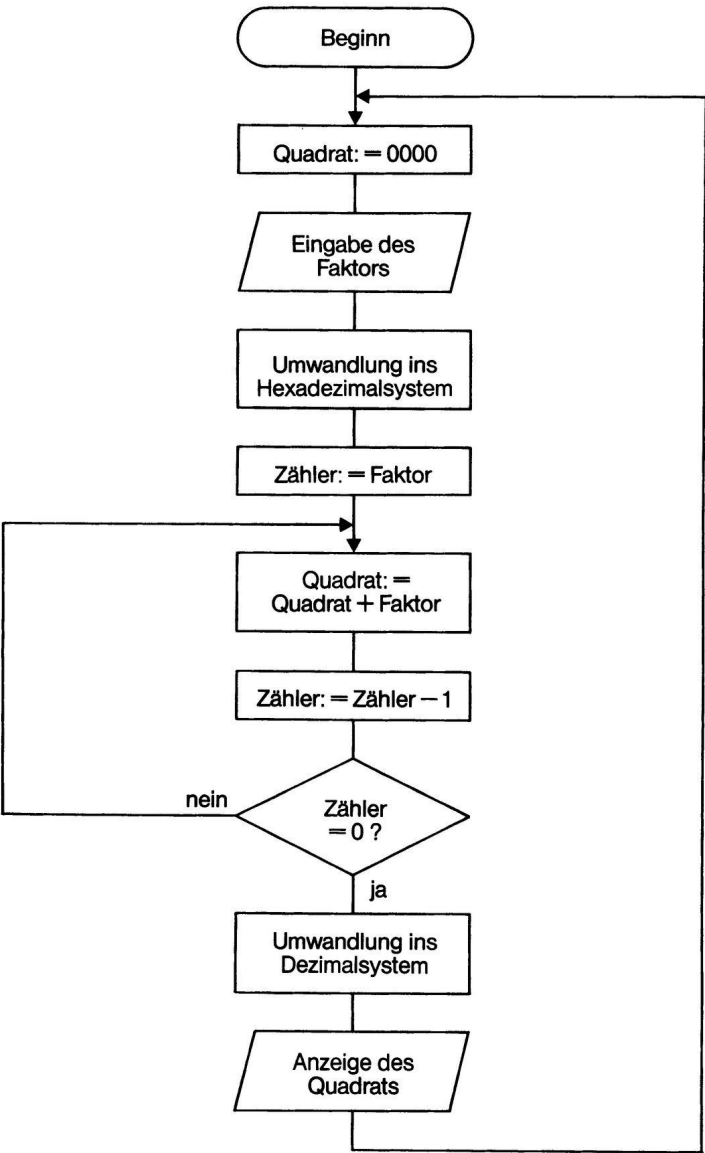
1.1 Es wird nur der 1. Befehl geändert:

NR.	MARKE	MMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	ILD A,=04	1000	1C 04	HAUPTPROGRAMM
2.		ILD S,A	1002	07	(S)=04; (F2)=1

1.2 Es werden der 1. und der 4. Befehl geändert:

NR.	MARKE	MMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	ILD A,=02	1000	1C 02	HAUPTPROGRAMM
2.		ILD S,A	1002	07	(S)=02; (F1)=1; (F3)=0
4.		ILD A,=08	1006	1C 08	(S)=08; (F1)=0; (F3)=1
5.		ILD S,A	1008	07	

Der zugehörige Programmablaufplan:



1.3 Das Hauptprogramm wird erweitert. Da jetzt drei verschiedene Leuchtzustände der drei farbigen Leuchtdioden vorkommen sollen, besteht das Hauptprogramm auch aus drei Teilen. Es wird jeweils der Inhalt des Status-Registers geändert, dann folgt der Aufruf des Unterprogramms PAUSE:

NR.	MARKE	MMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	ILD A,=02	1000	1C 02	HAUPTPROGRAMM
2.		ILD S,A	1002	07	(S)=02; (F1)=1
3.		JSR PAUSE	1003	20 FF 10	SPRUNG ZUM UNTERPROGRAMM
4.		ILD A,=04	1006	1C 04	(S)=04; (F2)=1
5.		ILD S,A	1008	07	(S)=04; (F2)=1
6.		JSR PAUSE	1009	20 FF 10	SPRUNG ZUM UNTERPROGRAMM
7.		ILD A,=08	100C	1C 08	(S)=08; (F3)=1
8.		ILD S,A	100E	07	(S)=08; (F3)=1
9.		JSR PAUSE	100F	20 FF 10	SPRUNG ZUM UNTERPROGRAMM
10.		BRA BEGINN	1012	174 EC	SPRUNG ZUM BEGINN

2. Aufg. (Kap. 12)

2.1 Im 1. Befehl wird das Unterprogramm AN, von ihm wird dann das Unterprogramm PAUSE aufgerufen: also 2 JSR-Befehle.

Der 4. Befehl wird sechzehnmal ausgeführt (10₁₆ = 16₁₀): 16 JSR-Befehle. Der 9. und der 10. Befehl (jeweils 2 JSR-Befehle) werden beide sechzehnmal ausgeführt: weitere 64 JSR-Befehle. Es werden bei einem vollen Durchlauf durch das Hauptprogramm also 82 JSR-Befehle ausgeführt.

Wir können diese Aussage durch den Computer überprüfen lassen. Wir ersetzen den 13. Befehl durch einen CALL TEST, ergänzen vor dem RET-Befehl in jedem der drei Unterprogramme einen Befehl IL D A,ZAEHLER (Operationscode 95 E5), schreiben vor dem Start des Programmes 00 in den Speicher FFE5 und starten das Programm. Beim Breakpoint überprüfen wir den Inhalt des Speichers FFE5: (FFE5) = 52₁₆ = 82₁₀.

2.2 Es gibt zwei Möglichkeiten:

Wird das Unterprogramm PAUSE vom Unterprogramm AN (oder AUS) aufgerufen, zeigt der Stackpointer auf die Adresse FFFC. Auf dem Stack sind dann zwei Rücksprungadressen abgespeichert worden: Rücksprungadresse zum Unterprogramm AN (oder AUS) und Rücksprungadresse zum Hauptprogramm.

Wird das Unterprogramm PAUSE direkt vom Hauptprogramm aufgerufen, zeigt der Stackpointer auf die Adresse FFFE. Auf dem Stack steht die eine Rücksprungadresse zum Hauptprogramm.

2.3 Unten im Stack (in den Speichern FFFF und FFFE) kann nur eine der vier vorkommenden Rücksprungadressen zum Hauptprogramm stehen: 1002, 1009, 1014 bzw. 1017. Wird das Unterprogramm PAUSE vom Unterprogramm AN oder AUS aufgerufen, so steht im Stack zusätzlich die Rücksprungadresse zum entsprechenden Unterprogramm: 1065 bzw. 1085.

In den Speichern FFFF und FFFD kann also nur die Hexadezimalzahl 10 stehen. Im Speicher mit der Adresse FFFE können vier Werte stehen: 02, 09, 14 oder 17. Im Speicher mit der Adresse FFFC kann einer der beiden Werte 65 oder 85 stehen.

2. Aufg. (Kap. 13)

Wie wir bei der ersten Aufgabe gesehen haben, beginnt das Unterprogramm, das mit dem CALL BLANK aufgerufen wird, im ROM ab Adresse 0895. Dieses Unterprogramm, das wir mit BLANK bezeichnen wollen, können wir auch mit einem JSR-Befehl aufrufen: JSR BLANK. Der Operationscode besteht aus drei Byte. Im Operationscode ist die Adresse 0894 enthalten: 20 94 08.

Entsprechendes gilt für die anderen Calls (vgl. Abschnitt 13.1).

MNEM. CODE	OP. CODE
JSR ANZ EIN	20 6C 01
JSR ANZEIGE	20 66 08
JSR BLANK	20 94 08
JSR UEB 4R	20 82 09
JSR UEB 4L	20 2B 01
JSR UEB 2	20 4B 01
JSR EIN 2ST	20 A8 09
JSR EIN 4ST	20 BC 09
JSR ANZ A	20 3E 09
JSR ANZ EA	20 52 09
JSR FEHLER	20 EE 01
JSR DEZ-HEX	20 A0 08
JSR HEX-DEZ	20 DE 08
JSR VERZ	20 5A 08
JSR TEST	20 EF 02
JSR HALT	20 E6 02

Lösungen zu Kapitel 13

1. Aufg. (Kap. 13)

Das Unterprogramm, das wir mit dem CALL BLANK aufrufen können (Operationscode 12), steht im ROM. Die Anfangsadresse finden wir in den Speichern 0021 + 2·2 und 0020 + 2·2, also in den Speichern 0025 und 0024. Dort steht die Adresse 0894. Das mit dem Operationscode 12 aufgerufene Unterprogramm beginnt dann ab Adresse 0895.

ADRESSE	INHALT
0895	84
0896	00
0897	00
0898	8D
0899	C0
089A	8D
089B	C2
089C	8D
089D	C4
089E	8D
089F	C6
08A0	5C

Wir lesen die Speicherinhalte bis zum ersten Return-Befehl ab.

Dieses Unterprogramm ist leicht zu verstehen. Es werden in die Speicher FFC0 bis FFC7 Nullen geschrieben. Wird anschließend z. B. der CALL ANZ EIN oder der CALL ANZEIGE aufgerufen, so bleiben alle Segmente der acht Sieben-Segment-Anzeigen dunkel, die gesamte Anzeige ist gelöscht, sie ist blank.

Nach Bearbeitung dieses Unterprogramms BLANK steht im EA-Register der Inhalt 0000.

INR.	MARKE	MNEM. CODE	ADR.	OP. CODE	BEMERKUNGEN
1.	BLANK	ILD EA,=0	0895 84 00 00		
2.		IST EA,CST0	0898 8D C0		<CST0>:=00; <CST1>:=00
3.		IST EA,CST2	089A 8D C2		<CST2>:=00; <CST3>:=00
4.		IST EA,CST4	089C 8D C4		<CST4>:=00; <CST5>:=00
5.		IST EA,CST6	089E 8D C6		<CST6>:=00; <CST7>:=00
6.		IRET	08A0 5C		

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
CST0	FFC0 CODIERUNG	STELLE 0
CST1	FFC1 CODIERUNG	STELLE 1
CST2	FFC2 CODIERUNG	STELLE 2
CST3	FFC3 CODIERUNG	STELLE 3
CST4	FFC4 CODIERUNG	STELLE 4
CST5	FFC5 CODIERUNG	STELLE 5
CST6	FFC6 CODIERUNG	STELLE 6
CST7	FFC7 CODIERUNG	STELLE 7

3. Aufg. (Kap. 13)

3.1 Um die Ampelphasen doppelt so lang zu machen, muß jeweils die Anzahl der Befehle CALL VERZ verdoppelt werden: sechsmal CALL VERZ statt dreimal, zweimal CALL VERZ statt einmal.

3.2 Die halbe Dauer der Ampelphasen erreichen wir wie folgt: Die Verzögerung ist halb so lang, wenn die Zahl 8000₁₆ bis 0000₁₆ heruntergezählt wird, wenn also das EA-Register vor Aufruf des Verzögerungs-Programms mit 8000₁₆ statt mit 0000₁₆ geladen wird.

Bei der grünen und der roten Ampelphase können wir an die Verzögerung mit halber Dauer unmittelbar eine Verzögerung mit der vollen Dauer anschließen. Da das EA-Register am Ende des ersten Verzögerungs-Programms die Zahl 0000 enthält, muß das EA-Register jetzt nicht neu geladen werden. Die ersten beiden Befehle des ursprünglichen Programms (vgl. Abschnitt 13.3) können somit entfallen, da das Extension-Register bei jeder Ampelphase neu geladen wird.

INR.	MARKE	MNEM. CODE	ADR.	OP. CODE	BEMERKUNGEN
1.	UMLAUF	ILD A,=02	1000 C4 02		
2.		ILD S,A	1002 07		<S>:=02; <F1>:=1; GRÜN
3.		ILD EA,=08000	1003 84 00 80		<E>:=80; <A>:=80
4.		ICALL VERZ	1006 1D		
5.		ICALL VERZ	1007 1D		
6.		ILD A,=04	1008 C4 04		
7.		ILD S,A	100A 07		<S>:=04; <F2>:=1; GELB
8.		ILD EA,=08000	100B 84 00 80		
9.		ICALL VERZ	100E 1D		
10.		ILD A,=08	100F C4 08		
11.		ILD S,A	1011 07		<S>:=08; <F3>:=1; ROT
12.		ILD EA,=08000	1012 84 00 80		
13.		ICALL VERZ	1015 1D		
14.		ICALL VERZ	1016 1D		
15.		ILD A,=0C	1017 C4 0C		
16.		ILD S,A	1019 07		<S>:=0C; <F2>:=1; <F3>:=1; GELB UND ROT
17.		ILD EA,=08000	101A 84 00 80		
18.		ICALL VERZ	101D 1D		
19.		BRA UMLAUF	101E 74 E0		SPRUNG ZUR MARKE "UMLAUF"

4. Aufg. (Kap. 13)

Das Ampelprogramm im ROM beginnt bei Adresse 03AA. Im Sprungbefehl muß also die Adresse 03AA - 1 = 03A9 stehen.

NR.	MARKE	MNEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.		JMP AMPEL	1000	24 A9 03	SPRUNG ZUM SPIEL "AMPEL"

WEITERE MARKE, BEZEICHNUNG	ADR.	BEMERKUNGEN
AMPEL	03AA	BEI DIESER ADRESSE BEGINNT DAS SPIEL "AMPEL"

5. Aufg. (Kap. 13)

5.1

NR.	MARKE	MNEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	LD A, -1	1000	C4 FF	
2.		ST A, ZAHL	1002	CD E0	ZAHL := -1
3.		ILD A, 0	1004	C4 00	
4.		ST A, ZWSP1	1006	CD D9	<ZWSP1> := 00

5.	ISCHL.	ILD A, ZAHL	1008	95 E0	ZAHL := ZAHL + 1
6.		ST A, ZWSP2	100A	CD D8	<ZWSP2> := ZAHL
7.		ICALL UEB 4R	100C	13	
8.		ILD A, ZAHL	100D	C5 E0	
9.		ST A, LED-R	100F	CD C3	<LED-R> := ZAHL
10.		ILD A, 0	1011	C4 00	
11.		ICALL ANZEIGE	1013	11	
12.		IBRA SCHL.	1014	74 F2	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZAHL	FF00	
ZWSP1	FFD9	ZWISCHENSPEICHER 1
ZWSP2	FFD8	ZWISCHENSPEICHER 2
LED-R	FFC3	LEUCHTDIODEN-REIHE

5.2 Die Zählgeschwindigkeit hängt im wesentlichen von der Dauer des CALLs ANZEIGE ab. Wird vor Aufruf dieses Unterprogramms ANZEIGE der Akkumulator mit der Hexadezimalzahl 80 geladen, so dauert dieser Call halb so lange; damit wird die Zählgeschwindigkeit verdoppelt.

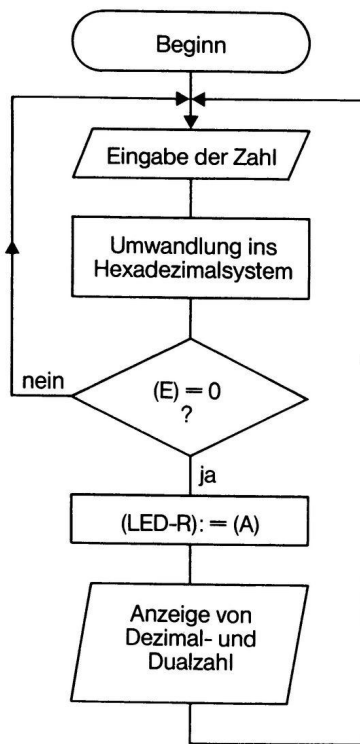
5.3 Im Programm von Abschnitt 4.3.5 wurde ein Rückwärtszähler programmiert. Dazu waren gegenüber dem Programm von Abschnitt 4.3.4 zwei Änderungen erforderlich: Der Befehl ILDA, ZAHL (5. Befehl) wurde durch den Befehl ILDA, 0 ersetzt. Die zweite Änderung betrifft den Anfangswert. Wenn die Zahl in den ersten beiden Befehlen gleich Null gesetzt wird, beginnt der Rückwärtszähler mit FF.

6. Aufg. (Kap. 13)

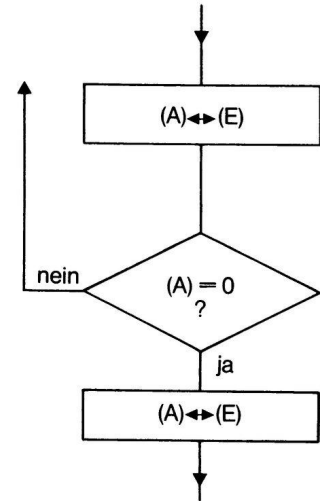
Um den Programmablaufplan zeichnen zu können, müssen zunächst die vollständigen Befehle aufgelistet werden.

MNEM.CODE	OP.CODE
CALL EIN 4ST	17
LD EA, ZWSP	85 D8
CALL DEZ-HEX	1B
XCH A, E	01
BNZ BEGINN	7C F9
XCH A, E	01
ST A, LED-R	CD C3
CALL ANZ EIN	10
NOP	00
NOP	00
BRA BEGINN	74 F1

Ist die eingegebene Dezimalzahl größer als 255, so ist die entsprechende Hexadezimalzahl nach Ausführung des CALLs DEZ-HEX größer als 00FF, die entsprechende Dualzahl läßt sich nicht an der LED-Reihe darstellen. Deshalb wird in diesem Fall (Hexadezimalzahl größer als 00FF) keine Anzeige vorgenommen. Wenn der Inhalt des Extension-Registers ungleich Null ist, erfolgt sofort ein Sprung zum Programmbeginn.



Die Abfrage des Extension-Registers kann ausführlicher folgendermaßen dargestellt werden:



7. Aufg. (Kap. 13)

7.1

NR.	MARKE	MNEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	CALL BLANK	1000	12	
2.	NEU	ILD A, 0F0	1001	C4 F0	
3.		ST A, LED-R	1003	CD C3	
4.		ILD A, 0	1005	C4 00	DIE LINKEN VIER LEUCHTDIODEN LEUCHTEN
5.		ICALL ANZEIGE	1007	11	

6.		ILD A, 0F	1008	C4 0F	
7.		ST A, LED-R	100A	CD C3	
8.		ILD A, 0	100C	C4 00	DIE RECHTEN VIER LEUCHTDIODEN LEUCHTEN
9.		ICALL ANZEIGE	100E	11	
10.		IBRA NEU	100F	74 F0	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
LED-R	FFC3	

7.2 Der 2. und der 6. Befehl müssen folgendermaßen verändert werden:

NR.	MARKE	MNEM.CODE	ADR.	OP.CODE
2.	NEU	ILD A, 0FF	1001	C4 FF
6.		ILD A, 0	1008	C4 00

7.3 Der 2. und der 6. Befehl müssen folgendermaßen verändert werden:

NR.	MARKE	INMEM.CODE	ADR.	OP.CODE
2.	INEU	ILD A, _r =03C	1001	IC4 3C
6.		ILD A, _r =0C3	1008	IC4 C3

8. Aufg. (Kap. 13)

Aufgabe	Codierung		dargestelltes Symbol
	dual	hexadezimal	
8.1	1000 0000	80	0 (Minuszeichen)
8.2	0000 0001	01	1 (Punkt)
8.3	1001 0000	90	0 (Gleichheitszeichen)
8.4	0000 0101	05	! (Ausrufezeichen)
8.5	1010 0111	A7	? (Fragezeichen)
8.6	0100 0100	44	" (Anführungszeichen)

9. Aufg. (Kap. 13)

Aufgabe	dargestelltes Symbol	Codierung	
		dual	hexadezimal
9.1	0	1011 0000	B0
9.2	h	1110 1000	E8
9.3	o	1011 1000	B8
9.4	u	0011 1000	38

10. Aufg. (Kap. 13)

Wenn wir in dem Programm in Abschnitt 13.9.3 den CALL HALT durch den CALL ANZEIGE ersetzen, wird das Programm nicht mehr in jeder Schleife angehalten. Beim Aufruf des CALLs ANZEIGE steht die Zahl, die an der LED-Reihe angezeigt wird, im Akkumulator. Vom Akkuinhalt hängt aber die Anzeigedauer ab, sie ist also unterschiedlich lang. Der Zähler zählt zunächst bei kleinen Akkuinhalten sehr schnell, er wird mit wachsender Zahl langsamer.

11. Aufg. (Kap. 13)

Wenn wir die angegebene Anzeige sehen, sorgt normalerweise das Betriebsprogramm für die Anzeige. Es fragt dann zwischendurch ständig die Tastatur ab und reagiert auf eine betätigte Taste in bestimmter Weise. Wenn wir unser kleines Programm starten, dann wird nur angezeigt. In diesem Programm werden die Tasten nicht abgefragt, es kann also auch keine Reaktion auf das Betätigen der Tasten erfolgen. Unser Computer befindet sich in einer Endlos-Schleife (vgl. Abschnitt 10.3, Beispiel 4). Er verläßt diese Schleife nur, wenn die **RS**-Taste betätigt wird.

Lösungen zu Kapitel 14

1. Aufg. (Kap. 14)

Zur Bestimmung des Verknüpfungsergebnisses werden die beiden gegebenen Hexadezimalzahlen in Dualzahlen umgewandelt. Die Verknüpfung wird dann stellenweise im Dualsystem durchgeführt. Zum Schluß wird das duale Ergebnis in die entsprechende Hexadezimalzahl umgewandelt.

1.1 $D2_{16} = 1101\ 0010$
 $97_{16} = 1001\ 0111$
 $D2_{16} \wedge 97_{16} = 1001\ 0010 = 92_{16}$

$D2 \vee 97 = D7$
 $D2 \vee 97 = 6D$
 $D2 \vee 97 = 28$
 $D2 \oplus 97 = BA$
 $D2 \oplus 97 = 45$

1.2 $A8 \wedge A8 = A8$
 $A8 \vee A8 = A8$
 $A8 \wedge A8 = 57$
 $A8 \vee A8 = 57$
 $A8 \oplus A8 = FF$
 $A8 \oplus A8 = 00$

1.3 $73 \wedge FF = 73$
 $73 \vee FF = FF$
 $73 \wedge FF = 8C$
 $73 \vee FF = 00$
 $73 \oplus FF = 73$
 $73 \oplus FF = 8C$

1.4 $3E \wedge 00 = 00$
 $3E \vee 00 = 3E$
 $3E \wedge 00 = FF$
 $3E \vee 00 = C1$
 $3E \oplus 00 = C1$
 $3E \oplus 00 = 3E$

2. Aufg. (Kap. 14)

2.1 Für die NAND-Verknüpfung zweier Hexadezimalzahlen gibt es keinen unmittelbaren Befehl. Die beiden Zahlen werden zunächst UND-verknüpft. Das Ergebnis der UND-Verknüpfung wird dann anschließend negiert. Wir haben schon gesehen (vgl. Abschnitt 14.4), daß diese Negation am einfachsten mit dem Befehl XOR A, = OFF erreicht werden kann.

NR.	MARKE	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	ICALL EIN 2ST	1000	116	EINGABE: 1.ZAHL
2.		ILD A,ZWSP	1001	IC5 D8	
3.		IST A,ZAHL1	1003	ICD E1	
4.		ICALL EIN 2ST	1005	116	EINGABE: 2.ZAHL
5.		ILD A,ZWSP	1006	IC5 D8	
6.		IAND A,ZAHL1	1008	ID5 E1	$\langle A \rangle := \langle A \rangle \wedge 1.ZAHL$
7.		IXOR A, _r =0FF	100A	IE4 FF	$\langle A \rangle := \langle A \rangle$
8.		ICALL ANZ A	100C	118	
9.		IBRA BEGINN	100D	174 F1	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZWSP	1FFD8	ZWISCHENSPEICHER
ZAHL1	1FFE1	1. EINGEGEBENE ZAHL

2.2 Die NOR-Verknüpfung wird erreicht, indem das Ergebnis der ODER-Verknüpfung negiert wird.

NR.	MARKE	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
6.		IOR A,ZAHL1	1008	ID5 E1	$\langle A \rangle := \langle A \rangle \vee 1.ZAHL$
7.		IXOR A, _r =0FF	100A	IE4 FF	$\langle A \rangle := \langle A \rangle$

2.3 Die Äquivalenz-Verknüpfung wird erreicht, indem das Ergebnis der Antivalenz-Verknüpfung negiert wird.

INR.	MARKE	MNEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
6.		XOR A,ZAHL1	1008	E5 E1	$\langle A \rangle := \langle A \rangle \oplus 1.ZAHL$
7.		XOR A,=0FF	100A	E4 FF	$\langle A \rangle := \langle A \rangle$

3. Aufg. (Kap. 14)

INR.	MARKE	MNEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	CALL EIN 2ST	1000	16	EINGABE DER ZAHL
2.		ILD A,ZWSP	1001	C5 D8	
3.		XOR A,=0FF	1003	E4 FF	NEGATION DER ZAHL
4.		CALL ANZ A	1005	18	ANZEIGE DER ZAHL
5.		BRA BEGINN	1006	74 F8	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZWSP	FFD8	ZWISCHENSPEICHER

Eine andere Möglichkeit besteht darin, daß die eingegebene Hexadezimalzahl von FF subtrahiert wird. Bei Subtraktion der eingegebenen Zahl von 100_{16} (oder von 00_{16}) würden wir das Sechzehner-Komplement erhalten. Subtrahieren wir von FF, dann erhalten wir das Sechzehner-komplement minus 1, also das Fünfzehner-Komplement.

INR.	MARKE	MNEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	CALL EIN 2ST	1000	16	EINGABE DER ZAHL
2.		ILD A,=0FF	1001	C4 FF	
3.		SUB A,ZWSP	1003	FD D8	$\langle A \rangle := FF - ZAHL$
4.		CALL ANZ A	1005	18	ANZEIGE DER ZAHL
5.		BRA BEGINN	1006	74 F8	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZWSP	FFD8	ZWISCHENSPEICHER

4. Aufg. (Kap. 14)

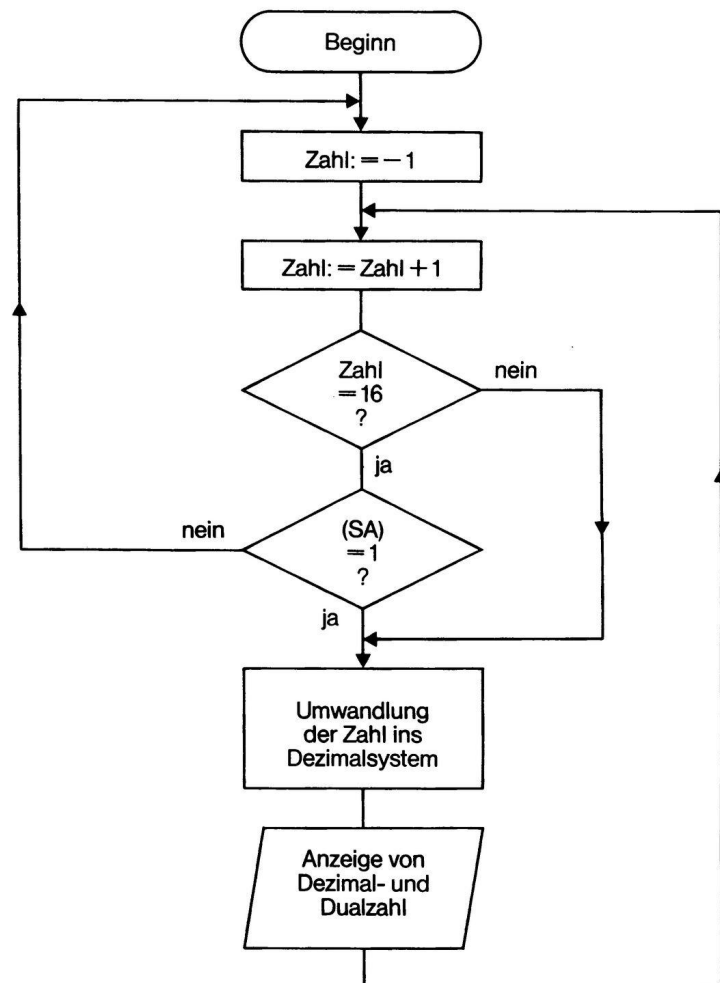
ADRESSE	OP. CODE
0405	84 FF 00
0408	8D D3
040A	95 D3
040C	E4 10
040E	7C 05
0410	06
0411	D4 10
0413	6C F0
0415	85 D3
0417	1C
0418	8D D8
041A	13
041B	C5 D3
041D	CD C3
041F	C4 00
0421	11
0422	74 E6

In dem nebenstehenden Ausdruck sind die Speicherinhalte von Adresse 0405 bis 0423 schon zu vollständigen Operationscodes zusammengefaßt.

Aufgrund der Displacements in den drei vorkommenden Sprungbefehlen sind dabei schon die Sprünge mit Pfeilen markiert.

INR.	MARKE	MNEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	ILD EA,=0FF	0405	84 FF 00	$ZAHL := -1$; $\langle FFD4 \rangle := 00$
2.		IST EA,ZAHL	0408	8D D3	
3.	WEITER	ILD A,ZAHL	040A	95 D3	$ZAHL := ZAHL + 1$
4.		XOR A,=010	040C	E4 10	
5.		BNZ ANZ.	040E	7C 05	KEIN SPRUNG, WENN $ZAHL = 16$
6.		ILD A,S	0410	06	
7.		AND A,=010	0411	D4 10	
8.		IBZ BEGINN	0413	6C F0	SPRUNG, WENN $\langle SA \rangle = 0$
9.	ANZ.	ILD EA,ZAHL	0415	85 D3	
10.		CALL HEX-DEZ	0417	1C	
11.		IST EA,ZWSP	0418	8D D8	
12.		CALL UEB 4R	041A	13	ANZEIGE DER DEZIMALZAHL
13.		ILD A,ZAHL	041B	C5 D3	
14.		IST A,LED-R	041D	CD C3	ANZEIGE DER DUALZAHL
15.		ILD A,=0	041F	C4 00	
16.		CALL ANZEIGE	0421	11	A N Z E I G E
17.		IBRA WEITER	0422	74 E6	

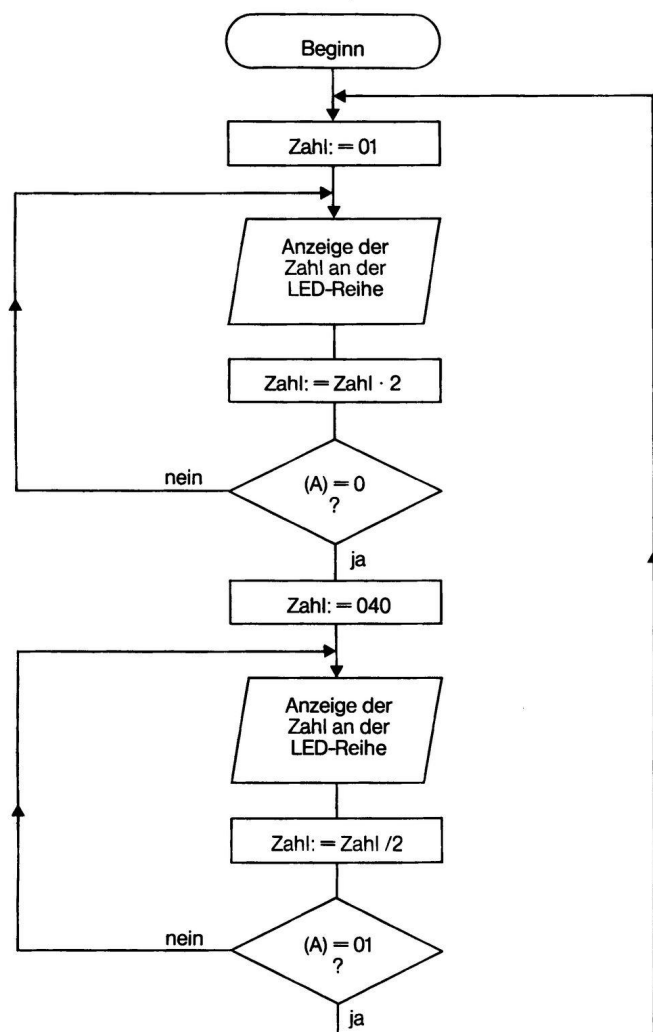
DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZAHL	FFD3	
	FFD4	$\langle FFD4 \rangle = 00$
ZWSP	FFD8	ZWISCHENSPEICHER
	FFD9	
LED-R	FFC3	LED-REIHE



1. Aufg. (Kap. 15)

NR.	MARKE	INMEM.CODE	ADR.	OP.CODE
1.	BEGINN	CALL BLANK	1000	12
2.	NEU	ILD A,=01	1001	1C 01
3.		IST A,ZAHL	1003	1C E0
4.	LINKS	IST A,LED-R	1005	1C C3
5.		ILD A,=040	1007	1C 40
6.		CALL ANZEIGE	1009	11
7.		ILD A,ZAHL	100A	1C5 E0
8.		ISL A	100C	10E
9.		IST A,ZAHL	100D	1C E0
10.		IBNZ LINKS	100F	17C F4
11.		ILD A,=040	1011	1C 40
12.		IST A,ZAHL	1013	1C E0
13.	RECHTS	ILD A,ZAHL	1015	1C5 E0
14.		IST A,LED-R	1017	1C C3
15.		ILD A,=040	1019	1C 40
16.		CALL ANZEIGE	101B	11
17.		ILD A,ZAHL	101C	1C5 E0
18.		ISR A	101E	13C
19.		IST A,ZAHL	101F	1C E0
20.		AND A,=01	1021	1D 01
21.		IBZ RECHTS	1023	16C F0
22.		IBRA NEU	1025	174 DA

DATENSPEICHER, BEZEICHNUNG	ADR.
ZAHL	1FFE0
LED-R	1FFC3



2. Aufg. (Kap. 15)

2.1 Nach Änderung des Befehls (RRL-Befehl statt RR-Befehl) und Start des veränderten Programms leuchten die Leuchtdioden der LED-Reihe von links beginnend nacheinander auf, bis alle acht Leuchtdioden leuchten. Diese Erscheinung ist dadurch zu erklären, daß das CY/L-Flag nach dem CALL ANZEIGE gesetzt ist. Bei jedem RRL-Befehl wird der Wert 1 des CY/L-Flags links in den Akkumulator hineingeschoben, bis die Zahl gleich FF geworden ist, bis alle Leuchtdioden leuchten.

2.2

NR.	MARKE	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	CALL BLANK	1000	12	
2.		ILD A,=080	1001	1C 80	
3.		IST A,ZAHL	1003	1C E0	
4.	WEITER	IST A,LED-R	1005	1C C3	
5.		ILD A,S	1007	106	
6.		IST A,HSP	1008	1C E1	(HSP) := (S)
7.		ILD A,=010	100A	1C 10	
8.		CALL ANZEIGE	100C	11	
9.		ILD A,HSP	100D	1C5 E1	
10.		ILD S,A	100F	107	(S) := (HSP)
11.		ILD A,ZAHL	1010	1C5 E0	
12.		IRRL A	1012	13F	
13.		IST A,ZAHL	1013	1C E0	
14.		IBRA WEITER	1015	174 EE	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZAHL	1FFE0	
LED-R	1FFC3	
HSP	1FFE1	HILFSSPEICHER

2.3 Für das „doppelte Lauflicht“ kann nicht mehr der CALL ANZEIGE verantwortlich sein. Das CY/L-Flag muß schon gesetzt sein, wenn das Programm beginnt. Vor dem Programmstart wird die Adresse 1000 und der Inhalt des Speichers mit dieser Adresse angezeigt. Dabei benutzt das Betriebsprogramm dieselbe Anzeige-Routine, die wir beim CALL ANZEIGE aufrufen. Das CY/L-Flag ist daher beim Programmstart gesetzt. Wenn wir ein „einfaches Lauflicht“ erreichen wollen, haben wir zwei Möglichkeiten: Wir können den Befehl AND S,=00 (Operationscode 39 00) vor dem 1., nach dem 1., 2. oder 3. Befehl ergänzen. Die zweite Möglichkeit: Wir laden im 2. Befehl den Akkumulator mit der Zahl 00. Beim RRL-Befehl wird dann die 1 aus dem CY/L-Flag in den Akkumulator geschoben.

3. Aufg. (Kap. 15)

3.1

(CY/L)	(E)	(A)	Befehl	(CY/L)	(E)	(A)
1	30	B8	SRA	1	30	5C
			SRL A	1	30	DC
			SR EA	0	18	5C
			SL A	1	30	70
			SL EA	1	61	70
			RR A	1	30	5C
			RRL A	0	30	DC

Wenn man die Register-Inhalte dual aufschreibt, lassen sich leicht die Register-Inhalte nach Ausführung eines Schiebe- oder Rotationsbefehles ermitteln.

Das CY/L-Flag ist unmittelbar beim SRL- und beim RRL-Befehl beteiligt. Beim SRL-Befehl kommt die 1 aus dem CY/L-Flag links in den Akku, bleibt aber im CY/L-Flag bestehen. Beim RRL-Befehl wird die 0 von Bit 0 des Akkus in das CY/L-Flag geschoben.

Bei allen anderen Befehlen sollte man erwarten, daß sich der Wert des CY/L-Flags nicht ändert. Wie wir mit dem Testprogramm in Aufg. 3.2 feststellen können, ist das CY/L-Flag nach dem SR EA-Befehl (bei den hier verwendeten Register-Inhalten) gelöscht. Das muß daran liegen, daß der Inhalt des E-Registers durch das CY/L-Flag in den Akku geschoben wird.

3.2 Für das Test-Program gibt es sicher mehrere Möglichkeiten.

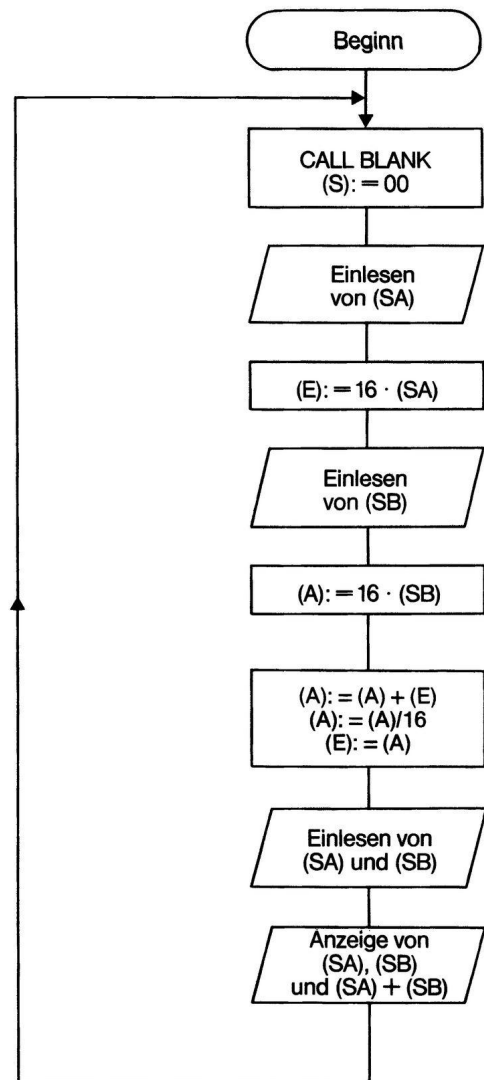
NR.	MARKE	MNEM. CODE	ADR.	OP. CODE
1.	BEGINN	CALL BLANK	1000	12
2.		IOR S,=000	1001	3B 00
3.		ILD EA,=03008	1003	84 B8 30
4.		ISR A	1006	3C
5.		IST EA,ZWSP	1007	8D 08
6.		ILD A,S	1009	06
7.		IST A,LED-R	100A	CD C3
8.		ICALL UEB 4L	100C	14
9.		ICALL ANZ EIN	100D	10
10.		INOP	100E	00
11.		INOP	100F	00
12.		ICALL TEST	1010	1E

DATENSPEICHER, BEZEICHNUNG	ADR.
ZWSP	FFD8
	FFD9
LED-R	FFC3

Der Wert des CY/L-Flags wird links an der LED-Reihe, der Inhalt des EA-Registers wird an den vier linken Sieben-Segment-Anzeigen dargestellt.

Wie wir bei der 2. Aufgabe dieses Kapitels gesehen haben, könnte der 2. Befehl in unserem Testprogramm entfallen. Das CY/L-Flag ist bei Programmstart gesetzt. Zur Erprobung der anderen Schiebe- und Rotationsbefehle muß der 4. Befehl entsprechend abgeändert werden.

4.3



4. Aufg. (Kap. 15)

4.1 und 4.2

NR.	MARKE	MNEM. CODE	ADR.	OP. CODE	BEMERKUNGEN
1.	BEGINN	CALL BLANK	1000	12	
2.		AND S,=0	1001	39 00	(S):=00
3.		ILD A,S	1003	06	
4.		AND A,=010	1004	D4 10	
5.		IXCH A,E	1006	01	(E):=010, WENN (SA)=1
6.		ILD A,S	1007	06	
7.		ISR A	1008	3C	
8.		AND A,=010	1009	D4 10	(A):=010, WENN (SB)=1
9.		ADD A,E	100B	70	(A):=(A)+(E)
10.		ISR A	100C	3C	
11.		ISR A	100D	3C	
12.		ISR A	100E	3C	
13.		ISR A	100F	3C	
14.		IXCH A,E	1010	01	(E):=(SA)+(SB)
15.		ILD A,S	1011	06	
16.		ADD A,E	1012	70	(A):= (SB)*020 + (SA)*010 + (SA)+(SB)
17.		IST A,LED-R	1013	CD C3	(LED-R):=(A)
18.		ILD A,=00	1015	C4 00	
19.		ICALL ANZEIGE	1017	11	ANZEIGE
20.		IBRA BEGINN	1018	74 E6	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
LED-R	FFC3	LED-REIHE

5. Aufg. (Kap. 15)

5.1 Bei diesem Zufallszahlengenerator werden auch vier bestimmte Bits des Akkuinhaltes XOR-verknüpft. Der Akkuinhalt wird nach links um eine Stelle verschoben. Das einstellige Verknüpfungsergebnis kommt rechts in den Akku (Bit 0).

Wenn wir von der Hexadezimalzahl FF ausgehen, wird A'=0, die nächste Zufallszahl wird damit FE. Entsprechend lassen sich die weiteren Zahlen überlegen: FC, F8, F1, E3, C6, usw. Es kommen alle zweistelligen Hexadezimalzahlen (≠ 00) vor.

5.2 Für dieses Programm gibt es sicher auch sehr unterschiedliche Möglichkeiten.

NR.	MARKE	MNEM.CODE	ADR.	OP.CODE
1.	BEGINN	ILD A,=0	1000	C4 00
2.		ST A,ZAHL	1002	CD E0
3.	WEITER	JSR ZUFALL	1004	20 FF 11
4.		CALL ANZ A	1007	18
5.		BRA WEITER	1008	74 FA
150.	ZUFALL	ILD A,ZAHL	1200	C5 E0
151.		BNZ NEU	1202	7C 02
152.		ILD A,ZAHL	1204	9D E0
153.	NEU	SL A	1206	0E
154.		ST A,HZ1	1207	CD E1
155.		SL A	1209	0E
156.		SL A	120A	0E
157.		ST A,HZ2	120B	CD E2
158.		SL A	120D	0E
159.		SL A	120E	0E
160.		XOR A,HZ2	120F	E5 E2
161.		XOR A,HZ1	1211	E5 E1
162.		XOR A,ZAHL	1213	E5 E0
163.		AND A,=000	1215	D4 80
164.		BZ FERTIG	1217	6C 02
165.		ILD A,HZ1	1219	95 E1
166.	FERTIG	ILD A,HZ1	121B	C5 E1
167.		ST A,ZAHL	121D	CD E0
168.		IRET	121F	5C

DATENSPEICHER, BEZEICHNUNG	ADR.
ZAHL	FFE0
HZ1	FFE1
HZ2	FFE2

2. Aufg. (Kap. 16)

Schleifen- durchlauf	1. Faktor (FFE3), (FFE2)		2. Faktor (FFD8)	Produkt (FFE1), (FFE0)	
vor dem 1.	00	A3	CE	00	00
nach dem 1.	01	46	67	00	00
nach dem 2.	02	8C	33	01	46
nach dem 3.	05	18	19	03	D2
nach dem 4.	0A	30	0C	08	EA
nach dem 5.	14	60	06	08	EA
nach dem 6.	28	C0	03	08	EA
nach dem 7.	51	80	01	31	AA
nach dem 8.	A3	00	00	83	2A

Der erste Faktor wird jeweils verdoppelt (vierstellig), der zweite halbiert. Das Produkt wird um den ersten Faktor (in der vorangehenden Zeile) vergrößert, wenn der zweite Faktor (in der vorangehenden Zeile) ungerade ist.

3. Aufg. (Kap. 16)

3.1

NR.	MARKE	MNEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	CALL EIN 4ST	1000	17	EINGABE DER ZAHL
2.		ILD EA,ZWSP	1001	85 D8	
3.		CALL DEZ-HEX	1003	1B	
4.		ILD T,EA	1004	09	
5.		XCH A,E	1005	01	"ERROR", WENN ZAHL > 255
6.		IBZ WEITER	1006	6C 01	
7.		CALL FEHLER	1008	1A	
8.	WEITER	ILD EA,T	1009	0B	ZAHL:=ZAHL+1 PROD:=(T)*(EA) SUMME:=PROD/2
9.		ADD EA,=1	100A	B4 01 00	
10.		IMPY EA,T	100D	2C	
11.		ILD EA,T	100E	0B	
12.		ISR EA	100F	0C	
13.		CALL HEX-DEZ	1010	1C	
14.		CALL ANZ EA	1011	19	
15.		BRA BEGINN	1012	74 EC	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZWSP	FFD8 FFD9	ZWISCHENSPEICHER

3.2 Die entsprechende Hexadezimalzahl zur eingegebenen Dezimalzahl 257_{10} ist 101_{16} . Ohne die Befehle 5. bis 8. würden jetzt die beiden Zahlen 101_{16} und 102_{16} multipliziert werden: $101_{16} \cdot 102_{16} = 10302_{16}$. Anschließend wird der Inhalt des EA-Registers (0302_{16}) halbiert (0181_{16}). Die entsprechende Dezimalzahl ($0181_{16} = 385_{10}$) wird angezeigt.

4. Aufg. (Kap. 16)

NR.	MARKE	MNEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	CALL EIN 2ST	1000	16	EINGABE DER ZAHL
2.		ILD EA,ZWSP	1001	85 D8	
3.		CALL DEZ-HEX	1003	1B	
4.		ILD T,EA	1004	09	
5.		IMPY EA,T	1005	2C	$\langle EA \rangle := ZAHL, HEX$ $\langle T \rangle := ZAHL, HEX$ $\langle T \rangle := QUADRAT, HEX$ $\langle EA \rangle := QUADRAT, HEX$ $\langle EA \rangle := QUADRAT$
6.		ILD EA,T	1006	0B	
7.		CALL HEX-DEZ	1007	1C	
8.		CALL ANZ EA	1008	19	
9.		BRA BEGINN	1009	74 F5	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZWSP	FFD8 FFD9	ZWISCHENSPEICHER

Lösungen zu Kapitel 16

1. Aufg. (Kap. 16)

Das Verdoppeln der Zahl geschieht im Hexadezimal- (oder Dual-) System mit dem Befehl SL EA. Wir müssen die Dezimalzahl vorher in eine Hexadezimalzahl, hinterher die verdoppelte Hexadezimalzahl wieder in eine Dezimalzahl umwandeln.

NR.	MARKE	MNEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	CALL EIN 2ST	1000	16	EINGABE DER ZAHL
2.		CALL BLANK	1001	12	
3.	WEITER	CALL UEB 4L	1002	14	ZAHL ZUR ANZEIGE
4.		CALL ANZ EIN	1003	10	
5.		INOP	1004	00	
6.		INOP	1005	00	
7.		ILD EA,ZWSP	1006	85 D8	
8.		CALL DEZ-HEX	1009	1B	
9.		SL EA	1009	0F	VERDOPPELN DER ZAHL
10.		CALL HEX-DEZ	100A	1C	
11.		ST EA,ZWSP	100B	8D D8	
12.		BRA WEITER	100D	74 F3	

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZWSP	FFD8 FFD9	ZWISCHENSPEICHER

Entsprechend läßt sich die eingegebene Dezimalzahl halbieren (vgl. Abschnitt 16.3).

NR.	MARKE	MNEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	CALL EIN 4ST	1000	17	EINGABE DER ZAHL
2.		CALL BLANK	1001	12	
9.		ISR EA	1009	0C	HALBIEREN DER ZAHL

5. Aufg. (Kap. 16)

6.2

NR.	MARKE	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	ILD EA,=0	1000 04 00 00		
2.		IST EA,QUOT	1003 0D E0		QUOTIENT:=0000
3.		ICALL EIN 4ST	1005 17		EINGABE: DIVIDEND
4.		ILD EA,ZWSP	1006 05 D8		
5.		IST EA,DIVID	1008 0D E2		EINGABE: DIVISOR
6.		ICALL EIN 4ST	100A 17		
7.	SCHL.	ILD EA,DIVID	100B 05 E2		
8.		SUB EA,DIVIS	100D 0D D8		DIVID:=DIVID-DIVIS
9.		IST EA,DIVID	100F 0D E2		
10.		ILD A,S	1011 06		
11.		AND A,=000	1012 04 00		
12.		IBZ FERTIG	1014 0C 09		SPRUNG, WENN DIVID<0
13.		ILD EA,QUOT	1016 05 E0		
14.		ADD EA,=1	1018 04 01 00		QUOT:=QUOT+1
15.		IST EA,QUOT	101B 0D E0		
16.		IBRA SCHL.	101D 174 EC		
17.	FERTIG	ILD EA,QUOT	101F 05 E0		
18.		ICALL ANZ EA	1021 19		ANZEIGE: QUOTIENT
19.		IBRA BEGINN	1022 174 DC		

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZWSP	1FFD8 1FFD9	ZWISCHENSPEICHER
DIVID	1FFE2 1FFE3	DIVIDEND, ZAEHLER
DIVIS	1FFD8 1FFD9	DIVISOR, NENNER
QUOT	1FFE0 1FFE1	QUOTIENT

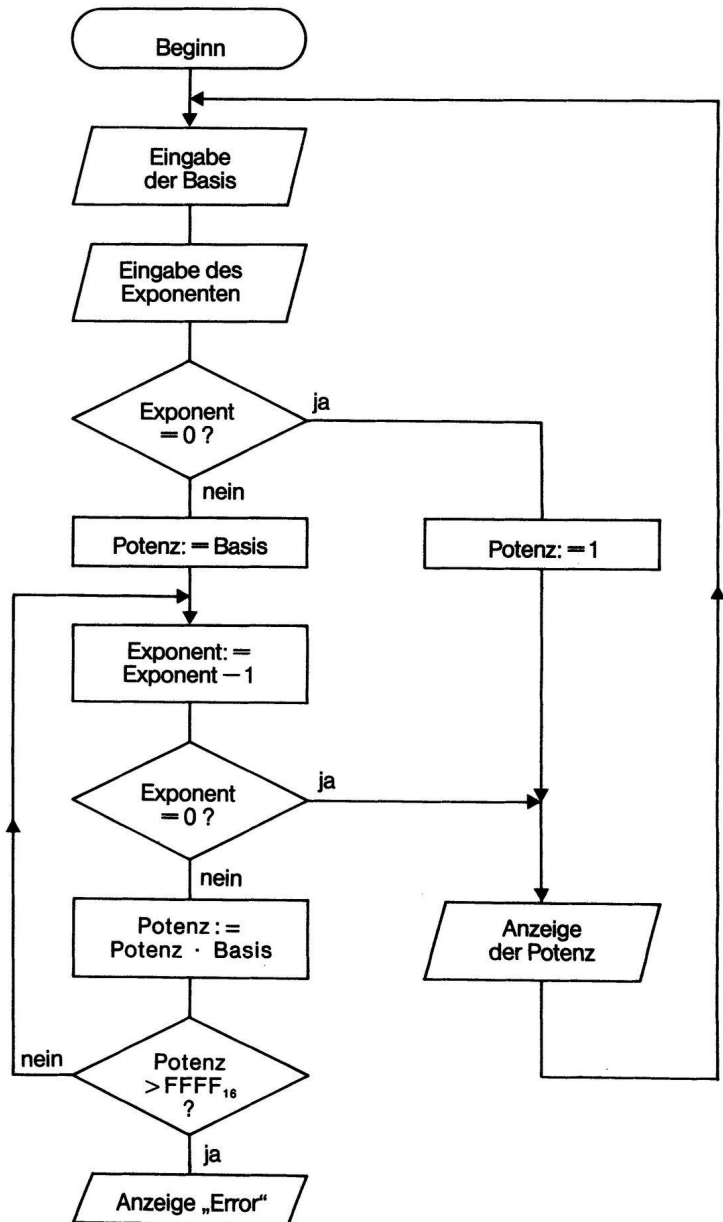
Wir hätten noch eine Sicherheitsmaßnahme einbauen können. Wenn wir mit diesem Programm eine Hexadezimalzahl durch 0 dividieren lassen, rechnet der Computer beliebig lange, ohne zu einem Ergebnis zu kommen.

6. Aufg. (Kap. 16)

6.1

NR.	MARKE	INMEM.CODE	ADR.	OP.CODE	BEMERKUNGEN
1.	BEGINN	ICALL EIN 2ST	1000 16		EINGABE DER BASIS
2.		ILD EA,ZWSP	1001 05 D8		
3.		ICALL DEZ-HEX	1003 1B		
4.		IST EA,BASIS	1004 0D E0		
5.		ICALL EIN 2ST	1006 16		EINGABE DES EXPONENTEN
6.		ILD EA,ZWSP	1007 05 D8		
7.		ICALL DEZ-HEX	1009 1B		
8.		IST EA,EXPON	100A 0D E2		
9.		IBNZ WEITER	100C 17C 05		
10.		ILD EA,=1	100E 04 01 00		WENN EXPON=0, ANZEIGE: "1"
11.		IBRA ANZEIG	1011 174 0F		
12.	WEITER	ILD EA,BASIS	1013 05 E0		
13.		ILD T,EA	1015 09		(T):=BASIS
14.	SCHL.	ILD A,EXPON	1016 19D E2		EXPON:=EXPON-1
15.		IBZ FERTIG	1018 0C 06		
16.		ILD EA,BASIS	101A 05 E0		
17.		IMPY EA,T	101C 12C		POTENZ:=POTENZ*BASIS
18.		IBZ SCHL.	101D 0C F7		
19.		ICALL FEHLER	101F 1A		"ERROR", WENN POTENZ>FFFF
20.	FERTIG	ILD EA,T	1020 0B		
21.		ICALL HEX-DEZ	1021 1C		
22.	ANZEIG	IST EA,ZWSP	1022 0D D8		ANZEIGE DER POTENZ
23.		ICALL UEB 4R	1024 13		
24.		ICALL ANZ EIN	1025 10		
25.		INOP	1026 00		
26.		INOP	1027 00		
27.		IBRA BEGINN	1028 174 D6		

DATENSPEICHER, BEZEICHNUNG	ADR.	BEMERKUNGEN
ZWSP	1FFD8 1FFD9	ZWISCHENSPEICHER
BASIS	1FFE0 1FFE1	BASIS, GRUNDZAHL
EXPON	1FFE2 1FFE3	EXPONENT, HOCHZAHL



1. Aufg. (Kap. 17)

- 1.1 Das CY/L-Flag wird bei der Addition $F0_{16} + 10_{16} = (1)00_{16}$ gesetzt. Die rote Leuchtdiode leuchtet bei der Anzeige der Zahl 00.
Das OV-Flag wird bei der Addition $70_{16} + 10_{16} = 80_{16}$ gesetzt. Die gelbe Leuchtdiode leuchtet bei der Anzeige der Zahl 80.
- 1.2 Das CY/L-Flag wird gesetzt, wenn die Subtraktion „ohne Borgen“ durchführbar ist. Es wird also nur bei der Subtraktion $00_{16} - 10_{16} = F0_{16}$ gelöscht. Die rote Leuchtdiode leuchtet bei allen Anzeigen außer bei der Anzeige der Zahl F0. Das OV-Flag wird bei der Subtraktion $80_{16} - 10_{16} = 70_{16}$ gesetzt. Die gelbe Leuchtdiode leuchtet bei der Anzeige der Zahl 70.

2. Aufg. (Kap. 17)

- 2.1 Das Displacement im 47. Befehl steht im Speicher mit der Adresse 130F. Die Zieladresse ist $ADR = 1340 - 1 = 133F$. Die Subtraktion $133F_{16} - 130F_{16} = 30_{16}$ liefert das Displacement 30_{16} .
Das Displacement im 87. Befehl steht im Speicher mit der Adresse 138F. Die Zieladresse für den Sprung ist wieder 133F. Die Subtraktion läßt sich in verschiedener Weise durchführen:

(1)
$$\begin{array}{r} 3F \\ 8F \\ \hline B0 \end{array} -$$

Sechzehnerkomplement von 8F:

$$\begin{array}{r} 3F \\ 71 \\ \hline B0 \end{array} +$$

$$\begin{array}{r} 138F \\ 133F \\ \hline 50 \end{array} -$$

Das Displacement ist $-50_{16} = B0$

2.2 Die Zieladresse ist 133F. Der Jump-Befehl hat also in beiden Fällen den Operationscode 24 3F 13.

3. Aufg. (Kap. 17)

3.1

INR.	MARKE	INMEM.	CODE	ADR.	OP.	CODE	BEMERKUNGEN
1.	BEGINN	ILD A, \emptyset		1000	IC4	00	
2.		IST A,ZAHL		1002	ICD	E0	ZAHL := \emptyset
3.	SCHL.	ICALL BLANK		1004	I12		
4.		ILD A,ZAHL		1005	I95	E0	ZAHL :=ZAHL+1
5.		IST A,ZWSP		1007	ICD	D8	
6.		ICALL UEB 2		1009	I15		
7.		ILD A, \emptyset FF		100A	IC4	FF	
8.		ICALL ANZEIGE		100C	I11		ANZEIGE DER ZAHL
9.		ILD A,S		100D	I86		
10.		IAND A, \emptyset 10		100E	ID4	10	
11.		IBZ SCHL.		1010	I6C	F2	SPRUNG, WENN <SA>=0
12.		ILD A,S		1012	I86		
13.		IXOR A, \emptyset 8		1013	IE4	08	<F3> ÄNDERN
14.		ILD S,A		1015	I07		
15.		IBRA SCHL.		1016	I74	EC	

DATENSPEICHER	BEZEICHNUNG	ADR.	BEMERKUNGEN
	ZAHL	1FFE0	
	ZWSP	1FFD8	ZWISCHENSPEICHER

3.3 In dem hier vorliegenden Programm wird die Taste **SA** per Programm abgefragt. Das ist innerhalb des Programmablaufes nur an einer bestimmten Stelle (bzw. an bestimmten Stellen) möglich. Eine kurzzeitige Betätigung der Taste **SA** wird in den meisten Fällen nicht registriert, da sich der Computer gerade mit der Anzeige beschäftigt. Das Programm ließe sich natürlich noch so umschreiben, daß die **SA**-Taste häufiger abgefragt wird. Das würde aber nichts am Prinzip ändern.
Beim Interrupt-Verfahren wird eine Betätigung der **SA**-Taste in jedem Fall registriert, egal mit welchem Programmteil sich der Computer gerade beschäftigt. Die Reaktion auf die Interrupt-Anforderung kann entweder sofort oder an einer durch Interrupt-Freigabe markierten Stelle erfolgen.

4. Aufg. (Kap. 17)

Durch die angegebene Änderung der beiden Befehle wird die Interrupt-Freigabe im Hauptprogramm gegeben. Wir wollen annehmen, daß eine Anforderung des Interrupt-Programms A durch ein Pellen der Taste **SA** nach dem 8. oder 9. Befehl des Hauptprogramms erfolgt ist. Dann folgt eine Würfel-Anzeige. Am Ende des CALLS ANZEIGE im Interrupt-Programm ist jetzt der Akkumulator mit 00 geladen. Nach dem Rücksprung ins Hauptprogramm wird der Akkumulatorinhalt 00 im 10. Befehl in den Zwischenspeicher geschrieben. Nach der erneuten Interrupt-Freigabe im 7. Befehl erfolgt wieder der Sprung ins Interrupt-Programm A. Es wird eine 0 angezeigt.

146

A. Anhang

A. 1 Befehlslisten

A. 1.1 Zusammenstellung der behandelten Befehle



BEFEHLE	MNEMONISCHER CODE	OPERATIONS-CODE	BYTE-ZAHL
TRANSPORT-BEFEHLE	LD A,=ZAHL	C4 XX	2
	LD A,BEZ	C5 XX	2
	LD EA,=ZAHL	84 YY XX	3
	LD EA,BEZ	85 XX	2
	LD T,=ZAHL	A4 YY XX	3
	LD T,BEZ	A5 XX	2
	LD A,E	40	1
	LD E,A	48	1
	LD A,S	06	1
	LD S,A	07	1
	LD T,EA	09	1
	LD EA,T	0B	1
	ST A,BEZ	CD XX	2
	ST EA,BEZ	8D XX	2
	XCH A,E	01	1
ARITHMETISCHE BEFEHLE	ADD A,=ZAHL	F4 XX	2
	ADD A,BEZ	F5 XX	2
	ADD EA,=ZAHL	B4 YY XX	3
	ADD EA,BEZ	B5 XX	2
	ADD A,E	70	1
	SUB A,=ZAHL	FC XX	2
	SUB A,BEZ	FD XX	2
	SUB EA,=ZAHL	BC YY XX	3
	SUB EA,BEZ	BD XX	2
	SUB A,E	78	1
	MPY EA,T	2C	1
	DIV EA,T	0D	1
INKREMENT- UND DEKREMENT-BEF.	ILD A,BEZ	95 XX	2
	DLD A,BEZ	9D XX	2
LOGISCHE BEFEHLE	AND A,=ZAHL	D4 XX	2
	AND A,BEZ	D5 XX	2
	AND A,E	50	1
	AND S,=ZAHL	39 XX	2
	OR A,=ZAHL	DC XX	2
	OR A,BEZ	DD XX	2
	OR A,E	58	1
	OR S,=ZAHL	3B XX	2
	XOR A,=ZAHL	E4 XX	2
	XOR A,BEZ	E5 XX	2
SPRUNG-BEFEHLE	XOR A,E	60	1
	JMP MARKE	24 YY XX	3
	BRA MARKE	74 XX	2
	BP MARKE	64 XX	2
	BZ MARKE	6C XX	2
	BNZ MARKE	7C XX	2
	JSR MARKE	20 YY XX	3
	RET	5C	1
DIE CALLS	CALL ANZ EIN	10	1
	CALL ANZEIGE	11	1
	CALL BLANK	12	1
	CALL UEB 4R	13	1
	CALL UEB 4L	14	1
	CALL UEB 2	15	1
	CALL EIN 2ST	16	1
	CALL EIN 4ST	17	1
	CALL ANZ A	18	1
	CALL ANZ EA	19	1
	CALL FEHLER	1A	1
	CALL DEZ-HEX	1B	1
	CALL HEX-DEZ	1C	1
	CALL VERZ	1D	1
	CALL TEST	1E	1
	CALL HALT	1F	1
SCHIEBE- UND ROTATIONS-BEF.	SR A	3C	1
	SRL A	3D	1
	SR EA	0C	1
	SL A	0E	1
	SL EA	0F	1
	RR A	3E	1
	RRL A	3F	1
	NOP	00	1

A. 1.2 Befehlsliste, Adressierungsarten

ADRESSIERUNGSART	MNEMONISCHER CODE	OPERATIONS-CODE
BEFEHLE MIT UNMITTELBARER ADRESSIERUNG	ADD A,=ZAHL	F4 XX
	ADD EA,=ZAHL	B4 YY XX
	AND A,=ZAHL	D4 XX
	AND S,=ZAHL	39 XX
	LD A,=ZAHL	C4 XX
	LD EA,=ZAHL	84 YY XX
	LD T,=ZAHL	A4 YY XX
	OR A,=ZAHL	DC XX
	OR S,=ZAHL	3B XX
	SUB A,=ZAHL	FC XX
	SUB EA,=ZAHL	BC YY XX
	XOR A,=ZAHL	E4 XX
BEFEHLE MIT DIREKTER ADRESSIERUNG	ADD A,BEZ	F5 XX
	ADD EA,BEZ	B5 XX
	AND A,BEZ	D5 XX
	DLD A,BEZ	9D XX
	ILD A,BEZ	95 XX
	LD A,BEZ	C5 XX
	LD EA,BEZ	85 XX
	LD T,BEZ	A5 XX
	OR A,BEZ	DD XX
	ST A,BEZ	CD XX
	ST EA,BEZ	8D XX
	SUB A,BEZ	FD XX
	SUB EA,BEZ	BD XX
	XOR A,BEZ	E5 XX
BEFEHLE MIT ABSOLUTER ADRESSIERUNG	JMP MARKE	24 YY XX
	JSR MARKE	20 YY XX
BEFEHLE MIT PC-RELATIVER ADRESSIERUNG	BNZ MARKE	7C XX
	BP MARKE	64 XX
	BRA MARKE	74 XX
	BZ MARKE	6C XX

A. 1.3 Befehlsliste, mnemonischer Code

MNEMONISCHER CODE	OPERATIONS- CODE
ADD A,=ZAHL	F4 XX
ADD A,BEZ	F5 XX
ADD A,E	70
ADD EA,=ZAHL	B4 YY XX
ADD EA,BEZ	B5 XX
AND A,=ZAHL	D4 XX
AND A,BEZ	D5 XX
AND A,E	50
AND S,=ZAHL	39 XX
BNZ MARKE	7C XX
BP MARKE	64 XX
BRA MARKE	74 XX
BZ MARKE	6C XX
CALL ANZ A	18
CALL ANZ EA	19
CALL ANZ EIN	10
CALL ANZEIGE	11
CALL BLANK	12
CALL DEZ-HEX	1B
CALL EIN 2ST	16
CALL EIN 4ST	17
CALL FEHLER	1A
CALL HALT	1F
CALL HEX-DEZ	1C
CALL TEST	1E
CALL UEB 2	15
CALL UEB 4L	14
CALL UEB 4R	13
CALL VERZ	1D
DIV EA,T	00
DLD A,BEZ	9D XX
ILD A,BEZ	95 XX
JMP MARKE	24 YY XX
JSR MARKE	20 YY XX
LD A,=ZAHL	C4 XX
LD A,BEZ	C5 XX
LD A,E	40
LD A,S	06
LD E,A	48
LD EA,=ZAHL	84 YY XX
LD EA,BEZ	85 XX
LD EA,T	0B
LD S,A	07
LD T,=ZAHL	A4 YY XX
LD T,BEZ	A5 XX
LD T,EA	09

MNEMONISCHER CODE	OPERATIONS- CODE
MPY EA,T	2C
NOP	00
OR A,=ZAHL	DC XX
OR A,BEZ	DD XX
OR A,E	58
OR S,=ZAHL	3B XX
RET	5C
RR A	3E
RRL A	3F
SL A	0E
SL EA	0F
SR A	3C
SR EA	0C
SRL A	3D
ST A,BEZ	CD XX
ST EA,BEZ	8D XX
SUB A,=ZAHL	FC XX
SUB A,BEZ	FD XX
SUB A,E	78
SUB EA,=ZAHL	BC YY XX
SUB EA,BEZ	BD XX
XCH A,E	01
XOR A,=ZAHL	E4 XX
XOR A,BEZ	E5 XX
XOR A,E	60

A. 1.4 Befehlsliste, Operationscode

OPERATIONS- CODE	MNEMONISCHER CODE
00	NOP
01	XCH A,E
06	LD A,S
07	LD S,A
09	LD T,EA
0B	LD EA,T
0C	SR EA
0D	DIV EA,T
0E	SL A
0F	SL EA
10	CALL ANZ EIN
11	CALL ANZEIGE
12	CALL BLANK
13	CALL UEB 4R
14	CALL UEB 4L
15	CALL UEB 2
16	CALL EIN 2ST
17	CALL EIN 4ST
18	CALL ANZ A
19	CALL ANZ EA
1A	CALL FEHLER
1B	CALL DEZ-HEX
1C	CALL HEX-DEZ
1D	CALL VERZ
1E	CALL TEST
1F	CALL HALT
20 YY XX	JSR MARKE
24 YY XX	JMP MARKE
2C	MPY EA,T
39 XX	AND S,=ZAHL
3B XX	OR S,=ZAHL
3C	SR A
3D	SRL A
3E	RR A
3F	RRL A
40	LD A,E
48	LD E,A
50	AND A,E
58	OR A,E
5C	RET
60	XOR A,E
64 XX	BP MARKE
6C XX	BZ MARKE

OPERATIONS- CODE	MNEMONISCHER CODE
70	ADD A,E
74 XX	BRA MARKE
78	SUB A,E
7C XX	BNZ MARKE
84 YY XX	LD EA,=ZAHL
85 XX	LD EA,BEZ
8D XX	ST EA,BEZ
95 XX	ILD A,BEZ
9D XX	DLD A,BEZ
A4 YY XX	LD T,=ZAHL
A5 XX	LD T,BEZ
B4 YY XX	ADD EA,=ZAHL
B5 XX	ADD EA,BEZ
BC YY XX	SUB EA,=ZAHL
BD XX	SUB EA,BEZ
C4 XX	LD A,=ZAHL
C5 XX	LD A,BEZ
CD XX	ST A,BEZ
D4 XX	AND A,=ZAHL
D5 XX	AND A,BEZ
DC XX	OR A,=ZAHL
DD XX	OR A,BEZ
E4 XX	XOR A,=ZAHL
E5 XX	XOR A,BEZ
F4 XX	ADD A,=ZAHL
F5 XX	ADD A,BEZ
FC XX	SUB A,=ZAHL
FD XX	SUB A,BEZ

A. 1.5 Änderung der Register- und Speicherinhalte bei den sechzehn Calls

MNEMONISCHER CODE	OP. CODE	REGISTER IN DER CPU							SPEICHER	
		A	E	S	T	SP	P2	P3	FFC7	FFD9
									BIS FFC0	UND FFD8
CALL ANZ EIN	10	*	*	*	-	-	*	*	-	-
CALL ANZEIGE	11	0	-	*	-	-	*	*	-	-
CALL BLANK	12	0	0	-	-	-	-	-	0	-
CALL UEB 4R	13	*	*	*	-	-	*	-	*	-
CALL UEB 4L	14	*	*	*	-	-	*	-	*	-
CALL UEB 2	15	*	*	*	-	-	*	-	*	-
CALL EIN 2ST	16	*	*	*	-	-	*	*	*	*
CALL EIN 4ST	17	*	*	*	-	-	*	*	*	*
CALL ANZ A	18	-	-	*	-	-	-	-	*	*
CALL ANZ EA	19	-	-	*	-	-	-	-	*	*
CALL FEHLER	1A	*	*	*	-	-	*	*	*	*
CALL DEZ-HEX	1B	*	*	*	*	-	-	-	-	*
CALL HEX-DEZ	1C	*	*	*	*	-	-	-	-	-
CALL VERZ	1D	0	0	*	-	-	-	-	-	-
CALL TEST	1E	-	-	*	-	-	-	-	*	*
CALL HALT	1F	-	-	*	-	-	-	-	*	*

„*“ — der Inhalt wird (eventl.) verändert
 „—“ — der Inhalt wird nicht verändert
 „0“ — der Inhalt ist hinterher Null

Der Inhalt des Status-Registers ändert sich in den meisten Fällen deshalb, weil während des Calls addiert oder subtrahiert wird. Die Änderungen betreffen dann häufig nur das CY/L- und das OV-Flag.

A. 2 Umwandlungstabellen

A.2.1 Umwandlungstabellen:
Dezimal — dual — hexadezimal

(für Dezimalzahlen von 0 bis 255)

DEZ.	DUAL	HEX.	DEZ.	DUAL	HEX.	DEZ.	DUAL	HEX.	DEZ.	DUAL	HEX.
0	0000 0000	00	64	0100 0000	40	128	1000 0000	80	192	1100 0000	C0
1	0000 0001	01	65	0100 0001	41	129	1000 0001	81	193	1100 0001	C1
2	0000 0010	02	66	0100 0010	42	130	1000 0010	82	194	1100 0010	C2
3	0000 0011	03	67	0100 0011	43	131	1000 0011	83	195	1100 0011	C3
4	0000 0100	04	68	0100 0100	44	132	1000 0100	84	196	1100 0100	C4
5	0000 0101	05	69	0100 0101	45	133	1000 0101	85	197	1100 0101	C5
6	0000 0110	06	70	0100 0110	46	134	1000 0110	86	198	1100 0110	C6
7	0000 0111	07	71	0100 0111	47	135	1000 0111	87	199	1100 0111	C7
8	0000 1000	08	72	0100 1000	48	136	1000 1000	88	200	1100 1000	C8
9	0000 1001	09	73	0100 1001	49	137	1000 1001	89	201	1100 1001	C9
10	0000 1010	0A	74	0100 1010	4A	138	1000 1010	8A	202	1100 1010	CA
11	0000 1011	0B	75	0100 1011	4B	139	1000 1011	8B	203	1100 1011	CB
12	0000 1100	0C	76	0100 1100	4C	140	1000 1100	8C	204	1100 1100	CC
13	0000 1101	0D	77	0100 1101	4D	141	1000 1101	8D	205	1100 1101	CD
14	0000 1110	0E	78	0100 1110	4E	142	1000 1110	8E	206	1100 1110	CE
15	0000 1111	0F	79	0100 1111	4F	143	1000 1111	8F	207	1100 1111	CF

DEZ.	DUAL	HEX.	DEZ.	DUAL	HEX.	DEZ.	DUAL	HEX.	DEZ.	DUAL	HEX.
16	0001 0000	10	80	0101 0000	50	144	1001 0000	90	208	1101 0000	D0
17	0001 0001	11	81	0101 0001	51	145	1001 0001	91	209	1101 0001	D1
18	0001 0010	12	82	0101 0010	52	146	1001 0010	92	210	1101 0010	D2
19	0001 0011	13	83	0101 0011	53	147	1001 0011	93	211	1101 0011	D3
20	0001 0100	14	84	0101 0100	54	148	1001 0100	94	212	1101 0100	D4
21	0001 0101	15	85	0101 0101	55	149	1001 0101	95	213	1101 0101	D5
22	0001 0110	16	86	0101 0110	56	150	1001 0110	96	214	1101 0110	D6
23	0001 0111	17	87	0101 0111	57	151	1001 0111	97	215	1101 0111	D7
24	0001 1000	18	88	0101 1000	58	152	1001 1000	98	216	1101 1000	D8
25	0001 1001	19	89	0101 1001	59	153	1001 1001	99	217	1101 1001	D9
26	0001 1010	1A	90	0101 1010	5A	154	1001 1010	9A	218	1101 1010	DA
27	0001 1011	1B	91	0101 1011	5B	155	1001 1011	9B	219	1101 1011	DB
28	0001 1100	1C	92	0101 1100	5C	156	1001 1100	9C	220	1101 1100	DC
29	0001 1101	1D	93	0101 1101	5D	157	1001 1101	9D	221	1101 1101	DD
30	0001 1110	1E	94	0101 1110	5E	158	1001 1110	9E	222	1101 1110	DE
31	0001 1111	1F	95	0101 1111	5F	159	1001 1111	9F	223	1101 1111	DF

DEZ.	DUAL	HEX.	DEZ.	DUAL	HEX.	DEZ.	DUAL	HEX.	DEZ.	DUAL	HEX.
32	0010 0000	20	96	0110 0000	60	160	1010 0000	A0	224	1110 0000	E0
33	0010 0001	21	97	0110 0001	61	161	1010 0001	A1	225	1110 0001	E1
34	0010 0010	22	98	0110 0010	62	162	1010 0010	A2	226	1110 0010	E2
35	0010 0011	23	99	0110 0011	63	163	1010 0011	A3	227	1110 0011	E3
36	0010 0100	24	100	0110 0100	64	164	1010 0100	A4	228	1110 0100	E4
37	0010 0101	25	101	0110 0101	65	165	1010 0101	A5	229	1110 0101	E5
38	0010 0110	26	102	0110 0110	66	166	1010 0110	A6	230	1110 0110	E6
39	0010 0111	27	103	0110 0111	67	167	1010 0111	A7	231	1110 0111	E7
40	0010 1000	28	104	0110 1000	68	168	1010 1000	A8	232	1110 1000	E8
41	0010 1001	29	105	0110 1001	69	169	1010 1001	A9	233	1110 1001	E9
42	0010 1010	2A	106	0110 1010	6A	170	1010 1010	AA	234	1110 1010	EA
43	0010 1011	2B	107	0110 1011	6B	171	1010 1011	AB	235	1110 1011	EB
44	0010 1100	2C	108	0110 1100	6C	172	1010 1100	AC	236	1110 1100	EC
45	0010 1101	2D	109	0110 1101	6D	173	1010 1101	AD	237	1110 1101	ED
46	0010 1110	2E	110	0110 1110	6E	174	1010 1110	AE	238	1110 1110	EE
47	0010 1111	2F	111	0110 1111	6F	175	1010 1111	AF	239	1110 1111	EF

DEZ.	DUAL	HEX.	DEZ.	DUAL	HEX.	DEZ.	DUAL	HEX.	DEZ.	DUAL	HEX.
48	0011 0000	30	112	0111 0000	70	176	1011 0000	B0	240	1111 0000	F0
49	0011 0001	31	113	0111 0001	71	177	1011 0001	B1	241	1111 0001	F1
50	0011 0010	32	114	0111 0010	72	178	1011 0010	B2	242	1111 0010	F2
51	0011 0011	33	115	0111 0011	73	179	1011 0011	B3	243	1111 0011	F3
52	0011 0100	34	116	0111 0100	74	180	1011 0100	B4	244	1111 0100	F4
53	0011 0101	35	117	0111 0101	75	181	1011 0101	B5	245	1111 0101	F5
54	0011 0110	36	118	0111 0110	76	182	1011 0110	B6	246	1111 0110	F6
55	0011 0111	37	119	0111 0111	77	183	1011 0111	B7	247	1111 0111	F7
56	0011 1000	38	120	0111 1000	78	184	1011 1000	B8	248	1111 1000	F8
57	0011 1001	39	121	0111 1001	79	185	1011 1001	B9	249	1111 1001	F9
58	0011 1010	3A	122	0111 1010	7A	186	1011 1010	BA	250	1111 1010	FA
59	0011 1011	3B	123	0111 1011	7B	187	1011 1011	BB	251	1111 1011	FB
60	0011 1100	3C	124	0111 1100	7C	188	1011 1100	BC	252	1111 1100	FC
61	0011 1101	3D	125	0111 1101	7D	189	1011 1101	BD	253	1111 1101	FD
62	0011 1110	3E	126	0111 1110	7E	190	1011 1110	BE	254	1111 1110	FE
63	0011 1111	3F	127	0111 1111	7F	191	1011 1111	BF	255	1111 1111	FF

A. 2.2 Umwandlungstabelle: dezimal — hexadezimal

DEZ /	HEX	0	00	000	0000
1	1	A	64	3E8	2710
2	2	14	C8	7D0	4E20
3	3	1E	12C	BB8	7530
4	4	28	190	FA0	9C40
5	5	32	1F4	1388	C350
6	6	3C	258	1770	EA60
7	7	46	2BC	1858	11170
8	8	50	320	1F40	13880
9	9	5A	384	2328	15F90

Diese Tabelle kann bei der Umwandlung größerer Dezimalzahlen helfen. Unmittelbar lassen sich nur die Hexadezimalzahlen ablesen, die bestimmten Dezimalzahlen (vollen Zehnern, Hundertern, Tausendern, Zehntausendern) entsprechen, z. B.:

↓

Dez /	Hex	00
5	1F4	

$500_{10} = 1F4_{16}$ →

Bei der Umwandlung beliebiger Dezimalzahlen kann die Tabelle mehrfach benutzt werden.

Zum Beispiel soll 9580_{10} umgewandelt werden:

$$\begin{array}{rcl}
 9000_{10} & = & 2328_{16} \\
 500_{10} & = & 1F4_{16} \\
 80_{10} & = & 50_{16} \\
 \hline
 9580_{10} & = & 256C_{16}
 \end{array}$$

A. 2.3 Umwandlungstabelle: hexadezimal — dezimal

HEX /	DEZ	0	00	000	0000
1	1	16	256	4096	65536
2	2	32	512	8192	131072
3	3	48	768	12288	196608
4	4	64	1024	16384	262144
5	5	80	1280	20480	327680
6	6	96	1536	24576	393216
7	7	112	1792	28672	458752
8	8	128	2048	32768	524288
9	9	144	2304	36864	589824
A	10	160	2560	40960	655360
B	11	176	2816	45056	720896
C	12	192	3072	49152	786432
D	13	208	3328	53248	851968
E	14	224	3584	57344	917504
F	15	240	3840	61440	983040

Diese Tabelle kann bei der Umwandlung größerer Hexadezimalzahlen helfen. Unmittelbar lassen sich nur Dezimalzahlen ablesen, die bestimmten Hexadezimalzahlen entsprechen, z. B.:

$$\begin{array}{rcl}
 C00_{16} & = & 3072_{10} \\
 10000_{16} & = & 65536_{10}
 \end{array}$$

↓ ↓

Hex /	Dez	00	0000
1	65536		
C	3072		

→

Bei der Umwandlung beliebiger Hexadezimalzahlen kann die Tabelle mehrfach benutzt werden, vgl. Seite A/11.

A. 3 Stichwortverzeichnis

Bezeichnung/Seiten

- A 25, 34
- A↔D**-Taste 18, 53
- Achtstellige Anzeige 103, 119
- Achtstelliges Produkt 118
- ADD 39
- ADD A, = Zahl 39
- ADD A, BEZ 39
- ADD A, E 40
- ADD EA, = Zahl 42
- ADD EA, BEZ 43
- Addierer 30, 31, 97, 143
- Addierwerk 32, 49, 51
- Addition Dezimal 45, 46, 117, 133
- Addition Dual 30, 31, 33
- Addition Hexadezimal 33, 34, 37, 38, 43, 44, 94, 109
- Additionsbefehl 34, 39, 40, 42, 43
- Additionstabelle 30, 33
- Adreß-Bus 102
- Adreß-Register 102
- Adresse 18
- Adresseneingabe 18
- Adressenfeld 18, 76
- Adressierung der Calls 70, 71
- Adressierungsarten 35, 37, 54, 148
- Äquivalenz-Schaltung 84
- Äquivalenz-Verknüpfung 84, 86, 91, 140
- Akku 34
- Akku-Inhalt 34, 56
- Akkumulator 34, 93
- ALU 90
- Ampel 14, 73
- Ampel mit Handbedienung 121
- Ampel-Programm 73, 81, 138
- AND A, = Zahl 86
- AND A, BEZ 87
- AND A, E 86
- AND S, = Zahl 86
- AND-Befehl 86, 87
- AND-Verknüpfung 86, 87
- Anfangsadresse 18, 22
- Anschluß der Glühlampe 20, 22
- Anschluß des Cassettenrecorders 13, 123, 124
- Anschluß des Lautsprechers 15
- Antivalenz-Schaltung 85
- Antivalenz-Verknüpfung 85, 86, 87
- Anwender-Software 22
- Anzeige achtstellig 119
- Anzeige des Akku-Inhaltes 40
- Anzeige von (EA) 41, 42
- Anzeigen 40, 42, 77, 79
- Arbeitsspeicher 34
- Arithmetisch logische Einheit 90
- Arithmetische Operationen 82, 90, 109
- Ausgänge 23
- Ausgangssignal 82
- BCD-Code 120
- Bedienungspult 13
- Bedingter Sprung 53, 56
- Befehl 18, 34, 147
- Befehls-Decodierer 90
- Befehlsliste 147, 148, 149
- Befehlsregister 90
- Befehlszähler 53
- Berechnung des Displacements 57, 58, 134
- Betriebs-Software 39, 70
- Betriebsprogramm 70, 71, 73
- Betriebsspannung 20
- Binärsystem 24
- Bit 35
- Bit ändern 87
- Bit ausmaskieren 89
- Bit löschen 89
- Bit setzen 89
- Blinken der farbigen Leuchtdioden 66, 70, 137
- Blinken einer Glühlampe 20, 116
- Blinklicht 14, 88
- Blinkprogramme LED-Reihe 81, 139
- BNZ MARKE 56
- BP MARKE 56
- BRA MARKE 54, 58
- BRANCH-Befehl 54, 56
- Breakpoint 36, 39, 61
- Byte 35
- BZ MARKE 56
- Call 39, 70, 71
- CALL ANZ A 40
- CALL ANZ EA 43
- CALL ANZ EIN 79, 80
- CALL Anzeige 77
- CALL BLANK 77, 81, 138
- CALL DEZ-HEX 45, 104, 105, 106, 107
- CALL EIN 2ST 40
- CALL EIN 4ST 44
- CALL FEHLER 62, 107
- CALL HALT 80, 81, 121
- CALL HEX-DEZ 45, 104, 106, 107
- CALL TEST 40, 61, 81
- CALL UEB 2 77
- CALL UEB 4L 80
- CALL UEB 4R 78
- CALL VERZ 71, 72
- Call-Aufruf 70
- Carry-Link-Flag 36, 109
- Cassettenrecorder-Anschluß 13, 123, 124
- Cassettenrecorder-Interface 123
- Codierung beliebiger Symbole 78, 81, 140
- Codierung der Hexadezimalziffern 15, 75
- Computer macht Musik 130
- Computer und Peripherie 20
- Computer-Platine 13
- CPU 34, 40, 41, 53, 74, 80, 101
- CPU**-Taste 19
- CST 0 76
- CST 7 76
- CY/L 39, 40, 42, 46, 63, 93, 95
- CY/L-Flag 36, 63, 109
- CY/L-Flag und Addition 37, 38, 42, 44, 94, 109
- CY/L-Flag und Subtraktion 48, 49, 94
- CY/L-Register 93
- Daten-Bus 90, 102
- Daten-Register 90, 102
- Dateneingabe 18
- Datenfeld 18, 76
- Dekrementier-Befehl 59
- Dezimale Addition 44, 45
- Dezimale Division 102
- Dezimale Multiplikation 103, 104, 118, 119

Dezimale Subtraktion 52, 117, 134
 Dezimalsystem 14
 Diebstahlsicherung 21, 22, 54, 116
 Differenz 47, 51
 Differenz Negativ 48, 49, 52, 117
 Digitaluhr 130
 Direkte Adressierung 37, 38, 39, 43, 47, 59, 74, 102, 132
 Displacement 54, 55, 57
 DIV EA, T 104
 Dividieren durch 2 93, 94, 100
 Division Dezimal 102, 105
 Division Hexadezimal 105, 108, 145
 Division mit Rest 105
 Divisionsbefehl 104
 DLD 59
 DLD A, BEZ 59, 63, 135
 DLD-Befehl 62
 Drahtbrücke D 124
 Dualsystem 14, 24
 Dualzahl 24, 25, 26, 27, 28, 151
 Dualzahlen und Hexadezimalzahlen 25

E 25, 40
 EA-Register 41, 43, 93
 Einer-Komplement 50, 51
 Eingabe einer Zahl 39, 43
 Eingänge 23
 Eingang SA 23
 Eingang SB 23, 124, 125, 128
 Eingangssignal 82
 Einmaleins 121
 Einstelliger Hexadezimalzähler 25
 Elektronischer Würfel 15
 Endadresse 126
 Endlos-Schleife 55, 110, 140
 Endlos-Schleife mit Ausgang 110, 111
 Entprellen 113, 114
 Error 16, 45, 62
 Erweiterter Akkumulator 41, 93
 Erweiterungs-Register 40
 Exchange 40
 Exklusiv-ODER-Schaltung 85
 Exponent 16
 Extension-Register 40, 41, 93
 Externer Taster 20, 23

F 1 63
 F 2 63
 F 3 63
 Farbige Leuchtdioden 13, 14, 23, 63, 64 122
 Federleiste links 13, 120
 Federleiste rechts 13, 15, 20, 23
 Festspeicher 71
 Festwertspeicher 71
 Flag 36
 FLAG 1 20, 23, 63
 FLAG 2 20, 23, 63
 FLAG 3 20, 23, 63
 Flip-Flop 120
 Frequenz 15, 130
 Fünfstellige Summe 117
 Fünfzehner-Komplement 50
 Funktionstaste **A↔D** 18
 Funktionstaste **CPU** 34
 Funktionstaste **LD** 126
 Funktionstaste **ME+** 19
 Funktionstaste **ME—** 19

Funktionstaste **RUN** 19
 Funktionstaste **SP** 14
 Funktionstaste **SV** 126
 Funktionstasten 13, 16, 79 80

Gefälschter Würfel 114
 Gelbes Blinklicht 14, 88, 126
 Glühlampe 20, 23
 Glühlampe blinkt 20, 116

H 82
 H-L-Übergang 111
 H-Signal 51, 52
 Halbaddierer 30, 97, 143
 Halbieren 100, 144
 Haltepunkt 35, 40
 Hardware 23, 123
 Hauptprogramm 66, 68
 Hexadezimalsystem 25
 Hexadezimalzähler 25, 26, 81, 139
 Hexadezimalzahl 25, 26, 27, 28, 51
 Hexadezimalzahlen und Dualzahlen 25
 High 30, 51, 82, 111
 High Order Byte 42
 HOB 42, 43, 46, 53
 Höherwertiges Byte 42

IC 23
 IE-Flag 109, 110, 114
 ILD 59
 ILD A, BEZ 59, 63, 135
 Inhalt des Akkumulators 34, 56
 Inhalt des Extension-Registers 41
 Inhalt des Status-Registers 37
 Inhalt des T-Registers 102, 103
 Inhalt von PC 53
 Inhalt von SP 69, 110
 Inkrementier-Befehl 59
 Integrierte Schaltung 23
 Interface 124
 Interface-Schaltung 124
 Interrupt A 110
 Interrupt B 110
 Interrupt Enable Flag 109, 110
 Interrupt von der Peripherie 114
 Interrupt-Freigabe 110, 112, 114
 Interrupt-Programm 111, 112
 Interrupt-Unterdrückung 114
 Interrupt-Verfahren 110, 111

JK-Flip-Flop 120
 JMP 53
 JMP MARKE 53, 58
 JSR 66
 JSR MARKE 69
 JUMP-Befehl 53

Keine Operation 53
 Kilobyte 71
 Kleines Einmaleins 121
 Komplement 49, 50
 L 82
 L-Signal 51, 82
 Lade-Befehl 35, 39, 40, 42, 43, 64, 102
 Lauflicht 14
 Lauflicht links 93

Lauflicht links-rechts 97, 142
 Lauflicht rechts 91, 97, 142
 Laufschrift 17
 Lautsprecher 15, 21, 130
 LD 39, 80
 LD A, = Zahl 39
 LD A, Bez 39
 LD A, E 40
 LD A, S 64
 LD E, A 40
 LD EA, = Zahl 42
 LD EA, BEZ 43
 LD EA, T 102
 LD S, A 64
 LD T, = Zahl 102
 LD T, BEZ 102
 LD T, EA 102
LD-TASTE 126, 127
 LDR 21
 LED 14
 LED-REIHE 15, 75
 Leuchtdiode 14, 23
 Leuchtdioden-Reihe 13, 14, 75
 Lichtschranke 21, 22, 116
 Lichtschranke und Zähler 21, 22, 116
 LOAD 126
 LOAD-Befehl 39, 40, 42, 43, 64, 102
 LOB 42, 43, 46, 53
 Löschen eines Bits 89
 Logische Befehle 86, 87
 Logische Operationen 82, 83, 87, 90
 Logische Verknüpfungen 82, 85, 86, 87, 89
 Lotto „6 aus 49“ 16
 LOW 30, 51, 82, 111
 Low Order Byte 41

 Marke 53
 Maske 89
 Maskieren 89
 Maskieren eines Bits 89
 ME+ 80
ME+-Taste 19
 ME— 80
ME—-Taste 19
 Mehrstellige Hexadezimalzahlen 26, 81, 139
 Messung der Reaktionszeit 122
 Mikroprozessor 23, 34, 74
 Mikroprozessor-IC 34, 74, 101
 Mikroprozessor-System 23
 Millisekunde 122, 123
 Minuend 47, 51
 Mnemonischer Code 31, 148
 Morseapparat 21, 116
 MPY EA, T 102
 Multiplikation Dezimal 103, 104, 118, 119
 Multiplikation Dual 99
 Multiplikation Hexadezimal 98, 99, 100, 101, 103, 108, 144
 Multiplikationsbefehl 102
 Multiplikationstabelle 98
 Multiplizieren mit 2 93, 94, 100
 Musik 130

 NAND-Schaltung 83
 NAND-Verknüpfung 83, 86, 91, 140
 Negation 51, 83, 84, 85, 87, 88, 91, 141
 Negationsgatter 51
 Negationsschaltung 84, 85

 Negative Differenz 48, 49, 52, 117
 Negative Zahlen 51, 52, 53, 109
 Negatives Displacement 54, 55, 57
 Neuner-Komplement 49
 Nicht-Schaltung 85
 Niederwertiges Byte 42
 NOP 53
 NOR-Schaltung 84
 NOR-Verknüpfung 84, 86, 91, 141
 Nur-Lese-Flag 63, 64
 Nur-Lese-Speicher 71
 Nur-Schreib-Flag 63

 ODER-Schaltung 83
 ODER-Verknüpfung 83, 86, 87
 Offset-Adresse 126, 127
 Operand 39
 Operationscode 34, 149
 OR A, = Zahl 86
 OR A, BEZ 87
 OR A, E 86
 OR S, = Zahl 87
 OR-Befehl 87
 OR-Verknüpfung 86, 87
 OV-Flag 109, 110
 Overflow-Flag 109, 110

 Pause 65, 66, 68
 PC 23, 53
 PC-Register 23, 53, 66
 Peripherie 20, 23
 Pointer 0 23, 53
 Pointer 1 53, 69
 Pointer 2 53
 Pointer 3 53
 Pointer-Register 53
 Positive-Zahlen 52, 109
 Positives Displacement 54, 55
 Potenzen 16, 108, 145
 Prellen einer Taste 111, 112, 113
 Produkt achtstellig 118
 Programm 18
 Programm aus dem 2. Kapitel 19, 116
 Programmablaufplan 58
 Programmänderung 19
 Programmeingabe 19
 Programmkontrolle 19
 Programmstart (ab Adresse 1000) 19
 Programmstart bei beliebiger Adresse 22, 127
 Programmverzweigung 56
 Programmzähler 23, 53

 Quadratzahlen 63, 108, 136, 145
 Quarz 15, 130

 RAM 73, 127
 RAM (EXTERN) 73, 74
 RAM (INTERN) 73, 74, 101
 Reaktionstest 17, 96
 Reaktionszeit 122
 Rechenwerk 90
 Register 34
 Register in der CPU 40, 41, 101
 Relative Adressierung 54, 56
 RESET 70
 RESET-Taste 13, 14
 RET 66, 69

RETURN 69
 RL EA — ein neuer Befehl? 120
 ROM 71, 73, 74, 76, 127
 Rotationsbefehl 91, 92, 93, 97, 120, 142
 RR A 91, 92, 93
 RRL A 93
RS-Taste 13, 14, 70
 Rücksprung zum Hauptprogramm 66
 Rücksprungadresse 67, 69
 Rückwärtssprung 54, 57
 Rückwärtszähler 26, 27, 81, 139
 RUN 80
RUN-Taste 19

 S 37, 63, 109
 S-Register 37, 63, 109
 SA 63, 110, 111
SA-Taste 15, 20, 23
 Save 126
 SB 63, 110
 SB-Eingang 124, 125, 128
SB-Taste 15, 20, 23, 122
 Schalter S „SSA-LED“ 13, 14
 Schaltsymbol 51, 82, 83, 84, 85
 Schaltwerttabelle 51, 82, 83, 84, 85
 Schiebefehl 91, 93, 97, 142
 Schleife 59, 62
 Schleifendurchlauf 59, 62
 Schreib-/Lesespeicher 73
 Sechzehner-Komplement 50, 52, 134
 Sechzehnersystem 25
 Segment 75
 Sekunden-Uhr 15
 SENSE A 20, 23, 63, 64, 110, 111
 SENSE B 20, 23, 63, 64, 110
 Setzen eines Bits 89
 Sieben-Segment-Anzeige 13, 14, 15, 75, 78
 Sirene 15, 21
 SL A 93
 SL EA 93, 100
 Software 23, 123, 130
 SP 53, 69, 71, 80
 SP-Register 53
SP-Taste 14
 Speicher 18, 34, 37
 Speicher mit wahlfreiem Zugriff 73
 Speicher-Adressen 73
 Speicher-Befehl 39, 43
 Speicherplatz 18, 19, 37
 Spiel 0 „Laufschrift“ 17
 Spiel 1 „Ampel“ 14
 Spiel 2 „Gelbes Blinklicht“ 14, 88, 127
 Spiel 3 „Lauflicht“ 14, 92
 Spiel 4 „Zähler“ 14, 91, 141
 Spiel 5 „Sekunden-Uhr“ 15
 Spiel 6 „Sirene“ 15, 21
 Spiel 7 „Elektronischer Würfel“ 16
 Spiel 8 „Zahlenlotto 6 aus 49“ 16, 96
 Spiel 9 „Zweierpotenzen“ 16, 94, 100
 Spiel „Nimm“ 128
 Spiel „Zahlen erkennen“ 128
 Spiel „Zahlenraten“ 24, 25, 35
 Spiel 10 „Taschenrechner“ 16
 Spiel 11 „Reaktionstest“ 17, 96
 Spiel 12 „Laufschrift“ 17
 Spiel E 124
 Spiel F 125

Sprung zum Spiel „Ampel“ 81, 138
 Sprung zum Spiel „Gelbes Blinklicht“ 54
 Sprung zum Spiel „Sirene“ 54
 Sprung zum Unterprogramm 66, 69
 Sprung-Befehl 54, 56
 Sprungadresse 54, 96
 Sprungbedingung 56, 88
 SR A 91, 92, 93, 100
 SRE A 93
 SRL A 93
 ST 39
 ST A, BEZ 39
 ST EA, BEZ 43
 ST0 76
 ST7 76
 Stack 37, 67, 69, 74
 Stackpointer 69, 70, 71, 110
 Stapelspeicher 37, 67, 69
 Startadresse 126
 Status-Register 37, 63, 109, 150
 STORE-Befehl 39, 43
 SUB 46
 SUB A, = Zahl 46
 SUB A, BEZ 46
 SUB A, E 46
 SUB EA, = Zahl 46
 SUB EA, BEZ 46
 Subtrahend 47, 51
 Subtraktion Dezimal 52, 117, 134
 Subtraktion Hexadezimal 47, 49
 Subtraktionsbefehl 46
 Summe „1 + 2 + ... + N“ 61, 62, 63, 108, 144
 Summe 30, 31, 32
 Summe fünfstellig 117
 SV 80
SV-Taste 126
 System-Software 23, 70

 T-Register 101, 102
 Taschenrechner 16, 78
 Taste **A↔D** 18
 Taste **CPU** 34
 Taste **LD** 126
 Taste **ME +** 19
 Taste **ME —** 19
 Taste **RUN** 19
 Taste **SA** 15, 23, 63, 110, 111
 Taste **SB** 15, 23, 63, 110, 122
 Taste **SP** 14
 Taste **SV** 126
 Taster Extern 20, 23
 Tausch-Befehl 46
 Testprogramme für Cassettenrecorder-Anschluß 124, 125
 Trace 41, 46, 133

 Überspielkabel 122, 124, 128
 Übertrag 24, 26, 30, 31, 32, 34, 36, 42, 51, 94, 99, 109
 Uhr 15, 130
 Umwandlung Dezimal—Dual 29, 81, 139, 151
 Umwandlung Dezimal—Hexadezimal 29, 45, 80, 151, 152
 Umwandlung Dual—Dezimal 28, 151
 Umwandlung Dual—Hexadezimal 27, 151
 Umwandlung Hexadezimal—Dezimal 29, 45, 151, 152
 Umwandlung Hexadezimal—Dual 27, 151
 Umwandlungstabelle 151, 152
 Unbedingter Sprung 53, 54
 UND-Schaltung 82, 85

UND-Verknüpfung 82, 86, 87, 89
Unmittelbare Adressierung 37, 39, 42, 46, 102
Unterprogramm 65, 66, 68
Unterprogramm-Technik 66

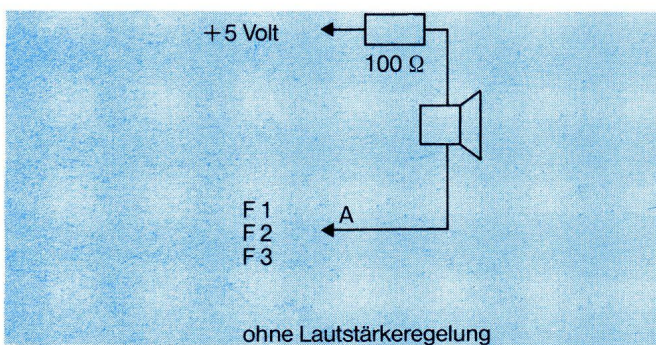
Verdoppeln 100, 108, 144
Verknüpfungsschaltung 82
Verknüpfungstabelle 30, 31, 51, 82
Verschieben von Programmen 126
VERZ 71
Verzweigung 56, 58
Volladdierer 31, 51
Von Dezimal nach Dual 29, 81, 139, 151
Von Dezimal nach Hexadezimal 29, 45, 151, 152
Von Dual nach Dezimal 28, 151
Von Dual nach Hexadezimal 26, 27, 151
Von Hexadezimal nach Dezimal 29, 45, 151, 152
Von Hexadezimal nach Dual 27, 151
Vorwärtssprung 54, 57

Würfel 15, 114, 122
Würfelspiel 122, 128

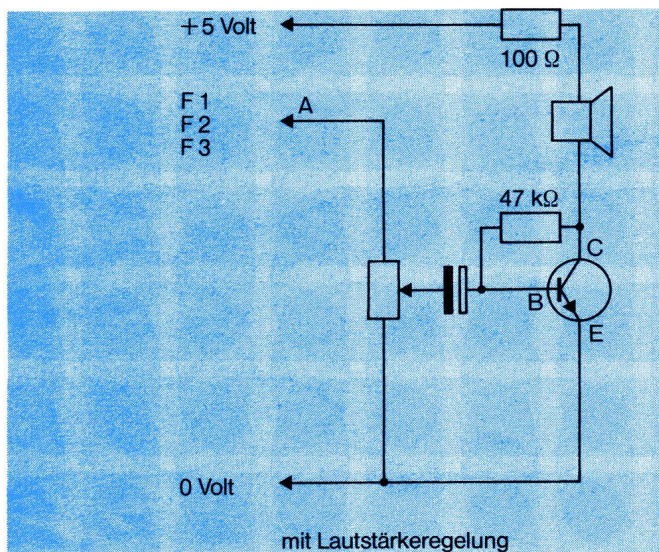
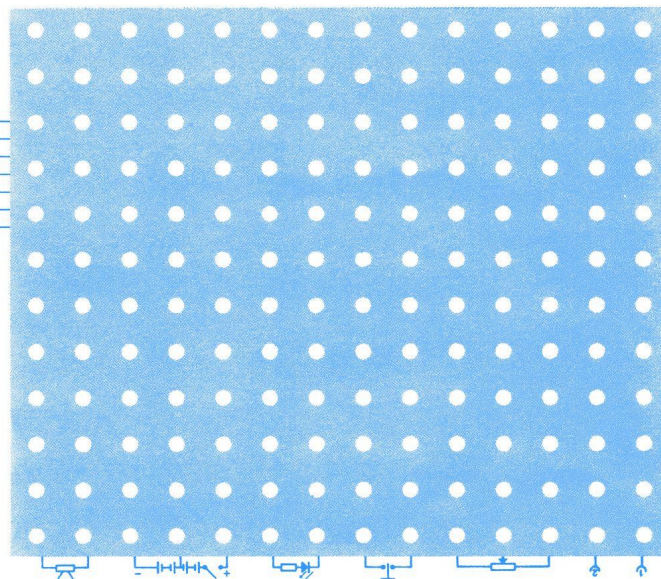
XCH 40
XCH A, E 40
XOR A, =Zahl 87
XOR A, BEZ 87
XOR A, E 86
XOR-Befehl 86, 87
XOR-Verknüpfung 86, 87, 121

Zähler 14, 25, 26, 27, 81, 88, 139
Zähler und Lichtschranke 21, 22, 116
Zahleneingabe 38, 39, 43
Zahlenkreis 52, 54
Zahlenlotto „6 aus 49“ 16
Zahlenraten 24
Zehner-Komplement 49
Zeitmessung 123
Zentraleinheit 34
Zieladresse 54, 55
Zifferntasten 13, 14, 16, 79, 80
Zufallszahlen-Generator 16, 95, 97, 122, 143
Zweier-Komplement 50, 51
Zweierpotenzen 16, 24, 28, 94, 108, 145
Zweiersystem 15, 24

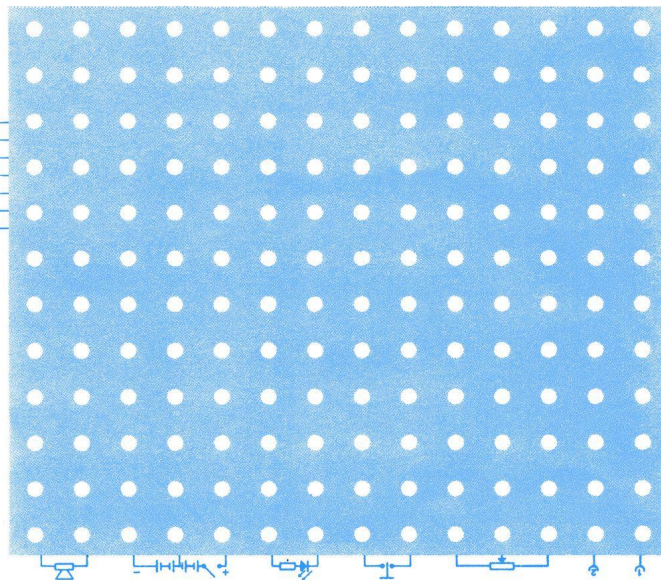
A. 4 Verdrahtungspläne



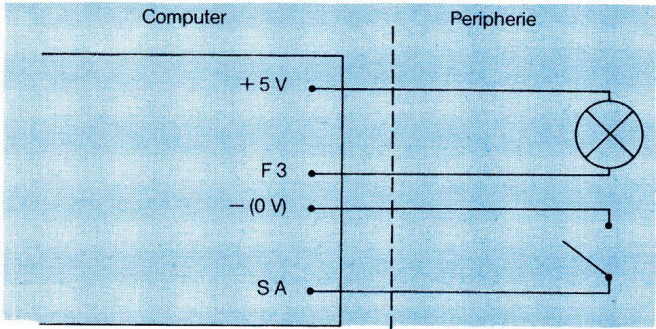
+ 5 Volt
F1
F2
F3
0 Volt
SA
SB



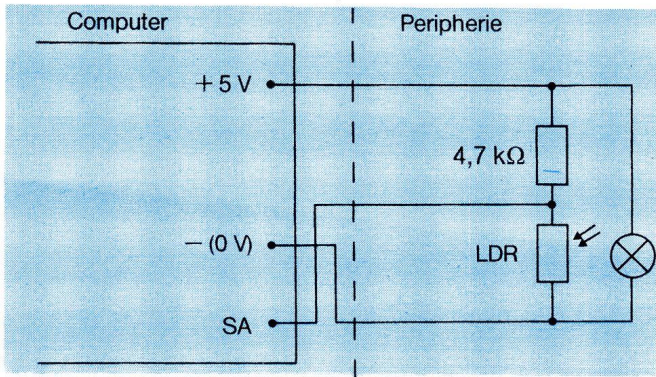
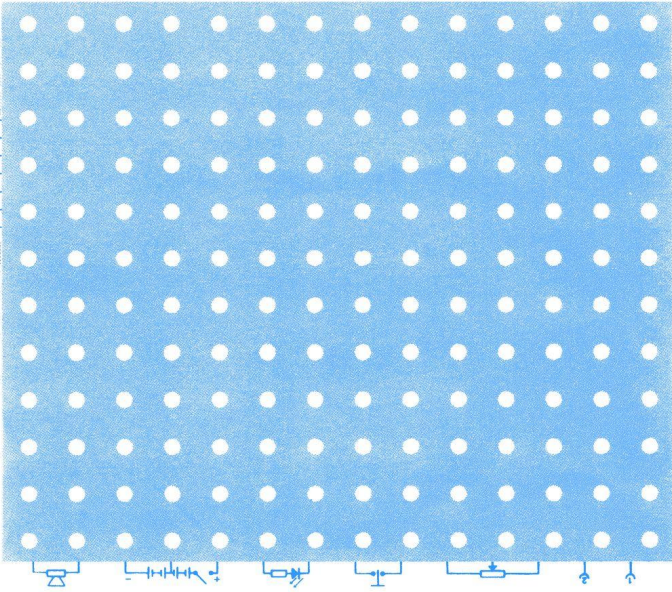
+ 5 Volt
F1
F2
F3
0 Volt
SA
SB



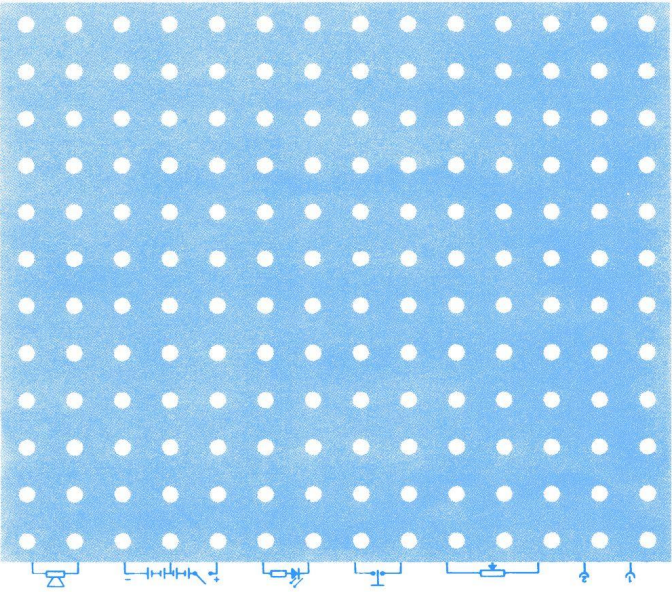
Bevor die Schaltungen der Peripherie auf der Grundplatte aufgebaut werden, empfiehlt es sich, auf den blauen Karten entsprechend Verdrahtungspläne zu entwerfen.



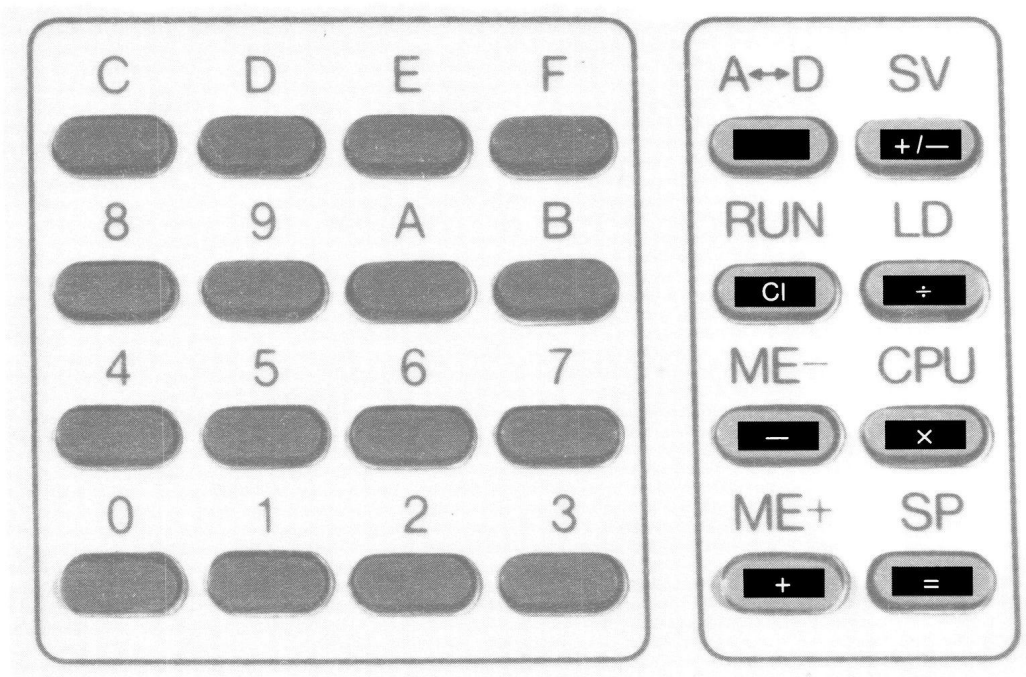
+ 5 Volt
F1
F2
F3
0 Volt
SA
SB



+ 5 Volt
F1
F2
F3
0 Volt
SA
SB



A. 5 Bedeutung der blauen Funktionstasten beim Taschenrechner-Spiel 10



Unsere Anschrift lautet

in Deutschland Georg Adam Mangold GmbH & Co KG
Lange Straße 69—75
8510 Fürth/Bayern

in Österreich Spiel-Sport-Stadlbauer Ges.m.b.H.
Postfach 83
5027 Salzburg

in der Schweiz Witeco Spielwaren AG
Birsstraße 58
4052 Basel

PHILIPS

MICROCOMPUTER MASTER LAB



Der leichte Einstieg in die Computertechnik
mit 12 Festprogrammen und Speichercassette
als Programmierbibliothek.