Allgemeine Hinweise zu den Experimenten

Das Experimentiersystem enthält einen vollständigen Rechner. Das Kernstück ist der INTEL 8080 Mikroprozessor. Als Speicher sind ein 1-k-ROM (1024 Wörter à 8 bit) und ein RAM mit 256 Wörtern à 8 bit vorhanden. In dem ROM sind 7 Programme zur Simulierung von verschiedenen Systemen fest abgespeichert. Welches Programm ablaufen soll, kann mit dem SYSTEM-Schalter (BCD-Schalter auf der linken Seite) festgelegt werden. Den 7 Programmen sind die Nummern 0 bis 6 zugeordnet. Die Stellung 7 des SYSTEM-Schalters ist für eine eventuelle Erweiterung des Systems vorgesehen, die Stellungen 8 und 9 werden nicht verwendet. Bei jeder Schalterstellung können mehrere Experimente durchgeführt werden. In den Stellungen 4, 5 und 6 stehen sogar 3 unterschiedliche Rechner zur Verfügung, die für beliebig viele Experimente benutzt werden können.

Ein Programm wird mit der RESET-Taste gestartet. Diese Taste entspricht in etwa der Löschtaste eines Taschenrechners und **muß am Anfang jedes Experimentes gedrückt werden.** Mit den restlichen Schiebeschaltern können Daten und Steuerinformationen eingegeben werden.

Die Schaltergruppe C_4 bis C_0 wird zur Steuerung des Experimentierablaufes benötigt.

Die Schaltergruppen A_7 bis A_0 und B_7 bis B_0 werden bis auf einige Spezialfälle für die Daten oder Programmeingabe benutzt.

Bei allen Experimenten gelten folgende Festlegungen:

Schalter oben ≙ logisch 1
Schalter unten ≙ logisch 0

Die 2 von 8 Leuchtdioden dienen zur Anzeige der Rechenergebnisse sowie zur Anzeige interner Schaltzustände. Hier gelten folgende Festlegungen:

Leuchtdiode leuchtet ≙ logisch 1
Leuchtdiode dunkel ≙ logisch 0

Alle Schalter, die bei einem bestimmten Experimentiervorgang nicht benötigt werden, müssen auf log. 0 geschaltet werden.

Es ist zu empfehlen, daß zu Beginn eines Experimentes alle Schalter auf log. 0 geschaltet werden, bevor die RESET-Taste gedrückt wird. Ausnahmen hiervon werden bei den einzelnen Experimenten angegeben.

Bei falscher Schalterbetätigung können grundsätzlich keine Schäden am Experimentiersystem entstehen. Allerdings können dadurch die selbst eingegebenen Programme und Daten verändert werden, so daß ein falsches Ergebnis entsteht. Bei umfangreichen und komplizierten Experimenten kann eine falsche Betätigung viel Zeit kosten.

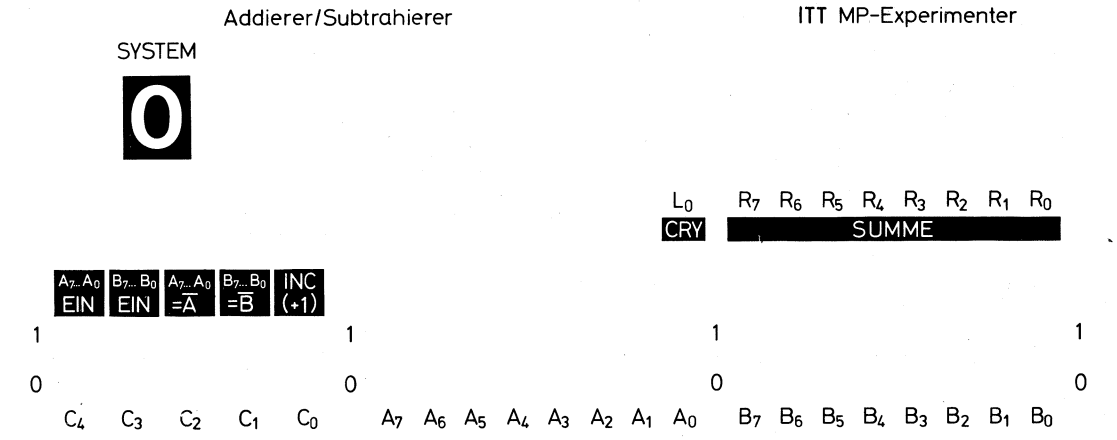
Die grundsätzliche Experimentiervorbereitung ist folgende:

1. Die in der Experimentieranweisung angegebene Schablone auflegen
2. SYSTEM-Schalter auf das verlangte Programm einstellen
3. Alle Schiebeschalter auf Null (unten) stellen
4. RESET-Taste drücken

In den Experimentieranleitungen ist zu Beginn jedes Experimentes auch noch einmal die entsprechende Schablone angegeben. Aus der Schablone kann die Bedeutung bzw. Funktion der einzelnen Schalter entnommen werden.

Zu jedem Experimentierprogramm werden ein oder auch mehrere Musterexperimente durchgeführt. Danach sind Aufgabenstellungen gegeben, die Sie selbst lösen sollen. Die Musterlösungen finden Sie im Experimentieranhang.

Experiment 1: Arbeitsweise eines 8-bit-Ripple-Carry-Addierers



Nach der Experimentiervorbereitung Schalter C_4 und C_3 auf 1. Damit können die an den Schaltern A_7 bis A_0 und B_7 bis B_0 eingestellten Informationen in das System gelangen. Mit den Schaltern $C_2 = \overline{A}$ und $C_1 = \overline{B}$ können die eingegebenen Informationen komplementiert werden (Einerkomplement). Der Schalter C_0 = INC legt bei 1 eine 1 auf den INC-Eingang. Die Schalter C_2 bis C_0 sind zunächst in der Stellung 0 zu belassen. Das Ergebnis der Addition erscheint in den rechten 8 LEDs (R_7 bis R_0). Die LED L_0 in der linken Lampengruppe zeigt einen Übertrag (Carry) an. Bei diesen Programmen haben die LEDs L_7 bis L_1 keine Bedeutung.

1. Beispiel:

A-Schalter:	0 0 0 0 1 0 1 0	$\triangleq 10_{10}$
+ B-Schalter:	0 0 0 0 0 0 1 1	$\triangleq 3_{10}$
Ergebnis:	0 0 0 0 1 1 0 1	$\triangleq 13_{10}$
	R_7 bis R_0	

2. Beispiel:

A-Schalter:	1 1 1 1 1 1 1 1	$\triangleq 255_{10}$
+ B-Schalter:	0 0 0 0 0 0 1 0	$\triangleq 2_{10}$
Ergebnis:	1 | 0 0 0 0 0 0 0 1	$\triangleq 257_{10}$
	L_0 R_7 bis R_0	

3. Beispiel:
Darstellung von negativen Zahlen über das Zweierkomplement

a)

A-Schalter:	0 0 0 0 1 0 1 0	$\triangleq 10_{10} = A$
$\overline{A}$-Schalter 1:	1 1 1 1 0 1 0 1	$\triangleq \overline{A}$
INC-Schalter 1:	1 1 1 1 0 1 1 0	$\triangleq -A = \overline{A} + 1$
	R_7 bis R_0	

b)

A-Schalter:	1 1 1 1 1 1 1 1	$\triangleq 255_{10} = A$
$\overline{A}$-Schalter 1:	0 0 0 0 0 0 0 0	$\triangleq \overline{A}$
INC-Schalter 1:	0 0 0 0 0 0 0 1	$\triangleq -A = \overline{A} + 1$
	R_7 bis R_0	

c)

A-Schalter:	0 0 0 0 0 0 0 1	$\triangleq 1_{10} = A$
$\overline{A}$-Schalter 1:	1 1 1 1 1 1 1 0	$\triangleq \overline{A}$
INC-Schalter 1:	1 1 1 1 1 1 1 1	$\triangleq -A = \overline{A} + 1$
	R_7 bis R_0	

4. Beispiel:
Subtraktion über das Zweierkomplement nach der Beziehung $A + (-B)$

a)

A-Schalter:	0 0 0 0 1 0 1 0	$\triangleq 10_{10}$
B-Schalter:	0 0 0 0 0 1 0 1	$\triangleq 5_{10}$
$\overline{B}$-Schalter 1:	1 | 0 0 0 0 0 1 0 0	$\triangleq A + \overline{B}$
	L_0 R_7 bis R_0	
INC-Schalter 1:	1 | 0 0 0 0 0 1 0 1	$\triangleq A + \overline{B} + 1 = A - B$
	L_0 R_7 bis R_0	
Ergebnis:	0 0 0 0 0 1 0 1	$\triangleq 5_{10}$
	R_7 bis R_0	

b)

A-Schalter:	0 0 0 0 0 1 0 1	$\triangleq 5_{10}$
B-Schalter:	0 0 0 0 1 0 1 0	$\triangleq 10_{10}$
$\overline{B}$-Schalter 1:	1 1 1 1 1 0 1 0	$\triangleq A + \overline{B}$
	R_7 bis R_0	
INC-Schalter 1:	1 1 1 1 1 0 1 1	$= A + \overline{B} + 1 = A - B$
	R_7 bis R_0	
Ergebnis:	1 1 1 1 1 0 1 1	$= -5_{10}$
	R_7 bis R_0	

Kontrollieren Sie dieses Ergebnis entsprechend Beispiel 3.

Lösen Sie experimentell folgende Aufgaben (Angaben in Dezimalzahlen).

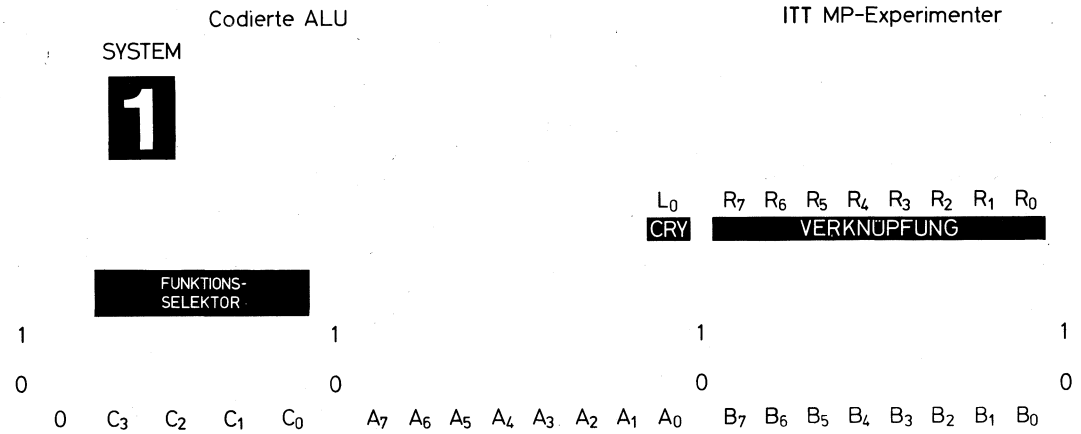
1. a) $125_{10} + 40_{10} =$
   b) $184_{10} + 100_{10} =$

2. Stellen Sie über das Zweierkomplement folgende Zahlen dar:
   a) -120_{10}
   b) -12_{10}
   c) -2_{10}

3. Lösen Sie folgende Subtraktionsaufgaben über die Beziehung A + (−B):
   a) $120_{10} - 100_{10} =$
   b) $130_{10} - 140_{10} =$

Die Lösungen finden Sie auf Seite E20.

Experiment 2: Arbeitsweise einer codierten ALU



Für dieses Experiment ist eine 8-bit-Version der in Bild 3.3.3 dargestellten codierten ALU simuliert worden. Die Funktionen U_3 bis U_0 in Tab. 3.3.1 werden mit den Schaltern C_3 bis C_0 festgelegt. Diese Tabelle ist auch auf der Karte „Codierte ALU" zu finden.
Nach der Experimentiervorbereitung (alle Schalter 0, RESET-Taste drücken) überprüfen wir zunächst alle Funktionen nach Tab. 3.3.1.

	C_3	C_2	C_1	C_0		
1.	0	0	0	0	→	Ein an den A-Schaltern eingestelltes bit-Muster erscheint in der Anzeige R_7 bis R_0. Die B-Schalter haben keine Funktion.
2.	0	0	0	1	→	Unabhängig von der Stellung der A- und B-Schalter erscheint in R_7 bis R_0 eine 1 in der Stelle R_0.
3.	0	0	1	0	→	In R_7 bis R_0 erscheint das Einerkomplement des an A_7 bis A_0 eingestellten bit-Musters. B_7 bis B_0 haben keine Funktion.
4.	0	0	1	1	→	Ein an B_7 bis B_0 eingestelltes bit-Muster erscheint in R_7 bis R_0. A_7 bis A_0 haben keine Auswirkung.
5.	0	1	0	0	→	Beide Schalterreihen haben keine Funktion.
6.	0	1	0	1	→	In R_7 bis R_0 erscheint die an A_7 bis A_0 eingestellte Zahl plus 1.
7.	0	1	1	0	→	In R_7 bis R_0 erscheint die an A_7 bis A_0 eingestellte Zahl minus 1.
8.	0	1	1	1	→	In R_7 bis R_0 erscheint die Summe der in A_7 bis A_0 und B_7 bis B_0 eingestellten Zahlen. Ein Übertrag erscheint in L_0.
9.	1	0	0	0	→	In R_7 bis R_0 erscheint die Differenz $A - B$ der an den Schaltern A_7 bis A_0 und B_7 bis B_0 eingestellten Zahlen (Zweierkomplement beachten!).
10.	1	0	0	1	→	Die in A_7 bis A_0 und B_7 bis B_0 stehenden Informationen werden bit-weise miteinander UND-verknüpft.
11.	1	0	1	0	→	Die in A_7 bis A_0 und B_7 bis B_0 stehenden Informationen werden bit-weise miteinander ODER-verknüpft.
12.	1	0	1	1	→	Die in A_7 bis A_0 und B_7 bis B_0 stehenden Informationen werden bit-weise miteinander EXCLUSIV-ODER-verknüpft.
13.	1	1	0	0	→	Unabhängig von A_7 bis A_0 und B_7 bis B_0 erscheint in R_7 bis R_0 −1 (Zweierkomplement beachten!).

Die angesprochenen Funktionen sind alle ausreichend bekannt und bedürfen daher keiner weiteren Erläuterung. Nachfolgend einige Übungsbeispiele:

1. Beispiel: $-30_{10} - 64_{10} =$

Diese Aufgabe wird durch eine Addition der Zweierkomplemente gelöst

$$-A + (-B) = -A - B$$

Funktion C_3 bis C_0: 0 1 1 1
A-Schalter: 1 1 1 0 0 0 1 0 ≙ -30_{10}
B-Schalter: 1 1 0 0 0 0 0 0 ≙ -64_{10}
1 | 1 0 1 0 0 0 1 0
L_0 R_7 bis R_0
Ergebnis: 1 0 1 0 0 0 1 0 = -94_{10}

2. Beispiel: $34_{10} - 128_{10} =$

Funktion C_3 bis C_0: 1 0 0 0
A-Schalter: 0 0 1 0 0 0 1 0 ≙ 34_{10}
$-B$-Schalter: 1 0 0 0 0 0 0 0 ≙ 128_{10}
1 | 1 0 1 0 0 0 1 0
L_0 R_7 bis R_0
Ergebnis: 1 0 1 0 0 0 1 0 = -94_{10}

3. Beispiel:
In A_7 bis A_0 ist folgendes bit-Muster eingestellt:

0 1 1 0 1 0 0 1

Dieses bit-Muster ist in ein Muster der Form

0 0 0 0 1 0 0 1

abzuändern, ohne dabei die Schalterstellung A_7 bis A_0 zu ändern. Diese Aufgabenstellung, das Ausblenden von bestimmten bit oder bit-Gruppen, läßt sich mit der UND-Funktion leicht lösen, indem an den B-Schaltern eine sog. Maske eingestellt wird. In unserem Beispiel wird:

Funktion C_3 bis C_0: 1 0 0 1
A-Schalter: 0 1 1 0 1 0 0 1
B-Schalter: 0 0 0 0 1 1 0 1 → Maske
0 0 0 0 1 0 0 1
R_7 bis R_0

Eine Änderung der Schalterstellung A_7 bis A_4 hat keinen Einfluß auf das Ergebnis in R_7 bis R_0.

4. Beispiel:
An den A-Schaltern ist folgendes bit-Muster eingestellt:

0 1 1 0 1 0 0 1

Dieses bit-Muster ist in ein Muster der Form

1 0 0 1 1 0 0 1

abzuändern, ohne dabei die Schalterstellung A_7 bis A_0 zu ändern. Diese Aufgabenstellung läßt sich mit der EXCLUSIV-ODER-Funktion lösen.

Funktion C_3 bis C_0: 1 0 1 1
A-Schalter: 0 1 1 0 1 0 0 1
B-Schalter: 1 1 1 1 0 0 0 0
1 0 0 1 1 0 0 1
R_7 bis R_0

5. Beispiel:
Ein an den A-Schalter eingestelltes bit-Muster der Form

1 0 0 0 0 1 0 0

soll in ein Muster der Form

1 0 1 1 0 1 0 1

geändert werden, ohne dabei die Schalterstellung A_7 bis A_0 abzuändern. Diese Aufgabe läßt sich über die ODER-Verknüpfung lösen.

Funktion C_3 bis C_0:	1 0 1 0
A-Schalter:	1 0 0 0 0 1 0 0
B-Schalter:	0 0 1 1 0 0 0 1
	1 0 1 1 0 1 0 1

R_7 bis R_0

Lösen Sie folgende Aufgaben experimentell. Die Daten sind im Hexadezimalsystem angegeben.

1. a) $3\,4_{16} + 2\,1_{16} =$
   b) $1\,4_{16} + (-4\,8_{16}) =$

2. a) $8\,1_{16} - 7\,5_{16} =$
   b) $7\,6_{16} - 8\,4_{16} =$

3. a) Ein bit-Muster in A_7 bis A_0 der Form

   1 1 0 0 1 1 1 1

   ist in die Form

   1 1 0 0 1 1 0 0

   umzuformen, ohne A_7 bis A_0 zu ändern.

   b) Ein bit-Muster in A_7 bis A_0 der Form

   1 1 0 1 0 1 1 0

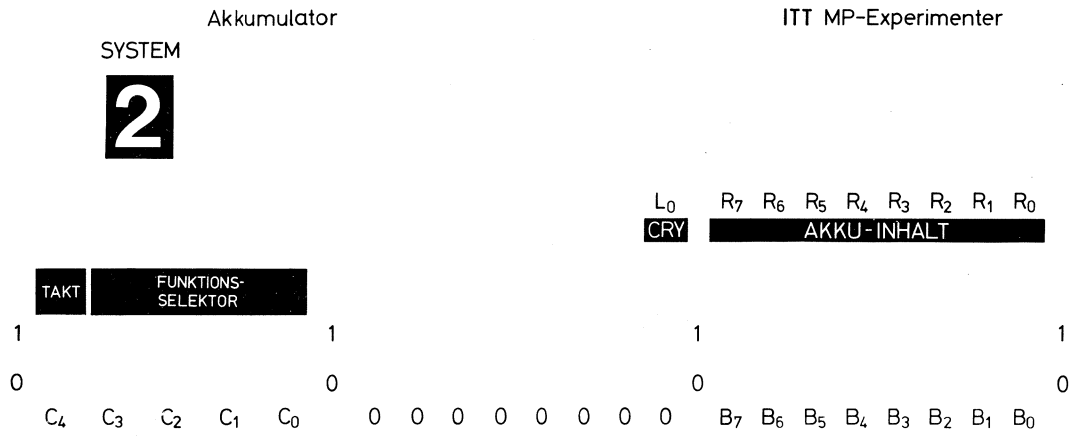
   ist in die Form

   0 0 0 0 0 1 1 0

   umzuformen, ohne dabei A_7 bis A_0 zu ändern.

Die Lösungen dieser Aufgaben finden Sie auf Seite E20.

Experiment 3: Arbeitsweise eines Akkumulators



Mit diesem Programm wird der in Bild 3.4.1 dargestellte Akkumulator simuliert. Die in Tab. 3.4.1 gezeigten Funktionen werden mit den Schaltern C_3 bis C_0 ausgewählt. Der Schalter C_4 dient jetzt als Taktschalter. Durch einmaliges hin- und herschieben wird ein Ergebnis in das Register übernommen und zur Anzeige gebracht. Die A-Schalter werden in diesem Beispiel nicht gebraucht, weil die A-Eingänge der im Akkumulator enthaltenen ALU mit den Ausgängen des Registers verbunden sind. Das Ergebnis bzw. der momentane Inhalt des Akkus wird wieder in R_7 bis R_0 angezeigt, ein Übertrag in L_0.
Zu Beginn eines Experimentes ist der Akku-Inhalt beliebig. Er muß daher zunächst auf Null gebracht werden (Vergleich: Löschtaste eines Taschenrechners). Das Löschen erfolgt laut Tab. 3.4.1 über die CLA-Funktion U_3 bis U_0 = C_3 bis C_0 = 0 1 0 0. Der Vorgang ist folgender:

1. C_3 bis C_0 auf 0 1 0 0 stellen
2. Schalter C_4 einmal takten (einmal hin- und herschieben)
3. Kontrollieren, ob alle LEDs R_7 bis R_0 einschließlich L_0 ausgehen.

Da die A-Schalter keine Funktion haben, werden die entsprechenden Operationen immer zwischen dem Akku-Inhalt und den an den B-Schaltern eingestellten Informationen durchgeführt. Hierzu einige Beispiele:

1. Beispiel:
Die Hexadezimalzahlen 1 5_{16} und 3 3_{16} werden addiert.

Ablauf:
1. 1 5_{16} an den B-Schaltern einstellen
2. LDA (Lade den Zustand der B-Schalter in den Akku) mit C_3 bis C_0 gleich 0 0 1 1 einstellen
3. Mit einem Takt (Schalter C_4) Inhalt der B-Schalter in den Akku laden
4. 3 3_{16} an den B-Schaltern einstellen
5. Mit C_3 bis C_0 = 0 1 1 1 den Befehl ADD wählen
6. Mit C_4 Takten
7. In R_7 bis R_0 steht jetzt das Ergebnis mit 0 1 0 0 1 0 0 0 $\triangleq$ 4 8_{16}

2. Beispiel:
Das Zweierkomplement der Binärzahl 0 1 0 1 1 1 0 1 wird gebildet.

Ablauf:
1. Die Zahl 0 1 0 1 1 1 0 1 an den B-Schaltern einstellen
2. Mit C_3 bis C_0 den Befehl LDA = 0 0 1 1 einstellen
3. Mit C_4 einmal takten
4. Mit C_3 bis C_0 den Befehl CMA = 0 0 1 0 einstellen
5. Mit C_4 takten. In R_7 bis R_0 erscheint jetzt 1 0 1 0 0 0 1 0, also das Einerkomplement der vorher eingestellten Zahl
6. Mit C_3 bis C_0 den Befehl INC = 0 1 0 1 einstellen

7. Mit C_4 takten. In R_7 bis R_0 erscheint die Zahl 1 0 1 0 0 0 1 1, das Zweierkomplement der eingegebenen Zahl

3. Beispiel:
Lösen der Aufgabe 5 1_{16} – 4 B_{16}

Ablauf:
1. Mit dem Befehl CLA den Akkumulator löschen (siehe vorher)
2. 5 1_{16} an B_7 bis B_0 einstellen und über Befehl LDA in den Akku laden
3. 4 B_{16} an B_7 bis B_0 einstellen
4. Mit Befehl SUB = 1 0 0 0 die an B_7 bis B_0 eingestellte Zahl von der im Akku befindlichen Zahl subtrahieren. Als Ergebnis erscheint in R_7 bis R_0 die Zahl 0 6_{16} (kontrollieren Sie selbst das Ergebnis über Dezimalzahlen nach).

4. Beispiel:
Der Akkumulator wird als Aufwärtszähler betrieben, der bei 0 beginnt und mit jedem Takt um 1 weiterzählt.

Ablauf:
1. Mit Befehl CLA den Akkumulator löschen
2. Den Befehl INC einstellen
3. Mit C_4 das System takten. In R_7 bis R_0 erscheint das jeweilige Zählergebnis

5. Beispiel:
Zwischen den beiden bit-Kombinationen 1 0 0 0 0 0 0 1 und 1 1 1 1 0 0 0 1 wird die EXCLUSIV-ODER-Verknüpfung gebildet.

Ablauf:
1. Mit Befehl CLA Akkumulator löschen
2. Kombination 1 0 0 0 0 0 0 1 an B_7 bis B_0 einstellen
3. Mit Befehl LDA Information B_7 bis B_0 in den Akkumulator laden
4. Kombination 1 1 1 1 0 0 0 1 an B_7 bis B_0 einstellen
5. Befehl XOR = 1 0 1 1 einstellen und takten. In R_7 bis R_0 erscheint das Ergebnis 0 1 1 1 0 0 0 0

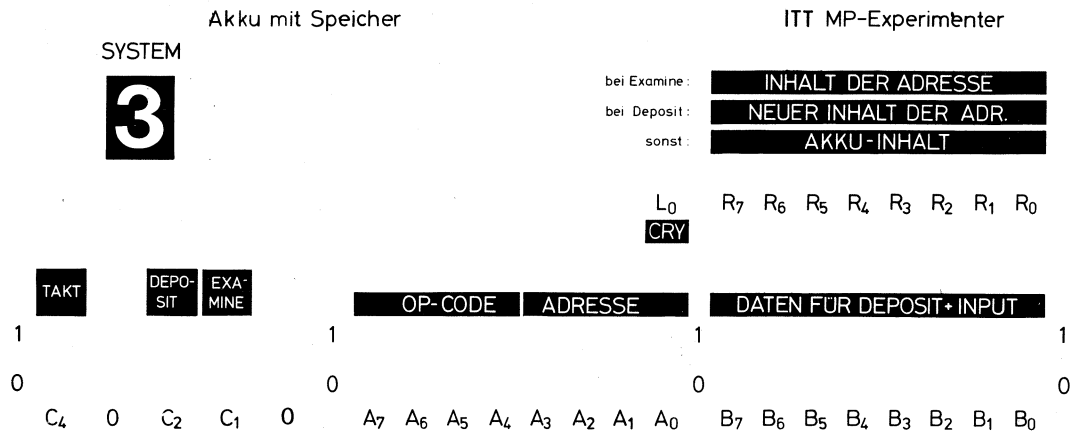
Aufgaben:

1. Subtrahieren Sie folgende Aufgaben:
   a) F 8_{16} – C C_{16} = 2C$_{16}$
   b) 1 5_{16} – 2 2_{16} = F3$_{16}$
   c) –4 8_{16} – 3 2_{16} = 86$_{16}$

2. Bilden Sie die ODER-Verknüpfung zwischen folgenden bit-Kombinationen:
   a) 1 1 0 0 0 1 1 0
   b) 0 1 0 1 0 0 0 0

Die Lösungen finden Sie auf Seite E21.

Experiment 4: Arbeitsweise eines Akkumulators mit Datenspeicher



Das System 3 enthält grundsätzlich die gleichen Funktionen wie das System 2. Der Unterschied besteht darin, daß entsprechend Bild 3.5.1 die Daten nicht mehr von den *B*-Schaltern kommen sondern von einem RAM. Mit dem neuen Befehl STA (Speichere Akku-Inhalt in Adresse *a a a a* ab), können die Daten im RAM zurückgeschrieben werden. Mit dem Befehl INP (Lade *B*-Eingänge in den Akku) werden jetzt die Daten an B_7 bis B_0 in den Akkumulator eingelesen (entspricht Befehl LDA in System 2). Bei allen Befehlen, die den Speicher nutzen, muß jetzt eine bestimmte Adresse spezifiziert werden. Der hier verwendete Speicher hat eine Kapazität von 16 Wörtern à 8 bit. Damit jedes dieser 16 Wörter spezifiziert bzw. adressiert werden kann, werden 4 bit benötigt. Damit besteht ein Befehl jetzt aus insgesamt 8 bit. Hiervon legen 4 bit die Funktion fest, die ausgeführt werden soll. Sie bilden den sog. OP-Code (Operation-Code). Die anderen 4 bit bestimmen die Speicheradresse. Aus diesem Grunde werden jetzt die *A*-Schalter für die Befehlseingabe benutzt.
Aus Tab. 3.5.1 geht hervor, daß es Befehle gibt, die unbedingt die Angabe einer Adresse benötigen (*a a a a*), und andere, die ohne spezielle Adresse auskommen (*x x x x*).
Bevor Befehle, die Daten aus dem Speicher unter einer bestimmten Adresse benötigen, benutzt werden können, müssen die entsprechenden Daten in den Speicher geladen werden. Das Laden einer bestimmten Speicheradresse erfolgt mit dem Schalter C_2 DEPOSIT (Laden). Wird dieser Schalter betätigt, d.h. auf 1 und dann wieder auf 0 geschaltet, werden die Daten, die an B_7 bis B_0 liegen, im Speicher bei der Adresse abgespeichert, die von den Schaltern A_3 bis A_0 spezifiziert ist.

1. Beispiel:
Die Zahl $1\ 5_{16}$ wird in Adresse 0 1 1 1 abgespeichert.

Ablauf:
1. B_7 bis B_0 auf 0 0 0 1 0 1 0 1 einstellen
2. A_7 bis A_0 auf 0 0 0 0 0 1 1 1 einstellen
3. Mit DEPOSIT-Schalter das System takten

Während DEPOSIT = 1 ist, erscheinen in R_7 bis R_0 die abzuspeichernden Daten. Dies ist als Kontrolle gedacht. Ist DEPOSIT wieder gleich 0, erscheinen in R_7 bis R_0 wieder die zufällig im Akku vorhandenen Daten.
Nachdem C_2 bzw. DEPOSIT wieder 0 ist, sind die Daten unter der Adresse 0 1 1 1 im Speicher abgespeichert. Die *B*-Schalter können jetzt beliebig verstellt werden.
Möchte man nachträglich die Daten in Adresse 0 1 1 1 kontrollieren, kann dies über Schalter C_1 EXAMINE (Lese) geschehen.

2. Beispiel:
Der Inhalt der Adresse 0 1 1 1 wird kontrolliert.

Ablauf:
1. A_7 bis A_0 auf 0 0 0 0 0 1 1 1 einstellen

2. Schalter EXAMINE auf 1 stellen
3. In R_7 bis R_0 erscheinen die Daten der Adresse 0 1 1 1

Solange EXAMINE = 1 ist, können mit Hilfe der Schalter A_3 bis A_0 alle Adresseninhalte kontrolliert werden.

3. Beispiel:
Folgene Daten werden unter der angegebenen Adresse abgespeichert:

Adresse	Daten
0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 1	0 0 0 0 0 0 0 1
0 0 1 0	0 0 0 0 0 0 1 0
0 0 1 1	0 0 0 0 0 0 1 1
0 1 0 0	0 0 0 0 0 1 0 0
0 1 0 1	0 0 0 0 0 1 0 1
0 1 1 0	0 0 0 0 0 1 1 0
0 1 1 1	0 0 0 0 0 1 1 1
1 0 0 0	0 0 0 0 1 0 0 0
1 0 0 1	0 0 0 0 1 0 0 1
1 0 1 0	0 0 0 0 1 0 1 0
1 0 1 1	0 0 0 0 1 0 1 1
1 1 0 0	0 0 0 0 1 1 0 0
1 1 0 1	0 0 0 0 1 1 0 1
1 1 1 0	0 0 0 0 1 1 1 0
1 1 1 1	0 0 0 0 1 1 1 1

Ablauf:
1. B_7 bis B_0 auf 0 0 0 0 0 0 0 0 einstellen
2. A_7 bis A_0 auf 0 0 0 0 0 0 0 0 einstellen
3. Mit DEPOSIT-Schalter System takten
4. B_7 bis B_0 auf 0 0 0 0 0 0 0 1 einstellen
5. A_7 bis A_0 auf 0 0 0 0 0 0 0 1 einstellen
6. Mit DEPOSIT-Schalter System takten usw.

Die so abgespeicherten Daten bleiben beliebig lang enthalten. Sie werden nur zerstört bei:
– Stromausfall oder Abschalten des Gerätes
– Abspeichern neuer Daten mit DEPOSIT unter derselben Adresse (alte Daten werden überschrieben)
– Abspeichern neuer Daten mit dem STA-Befehl in derselben Adresse
– Umschalten des SYSTEM-Schalters auf ein neues Experimentierprogramm

4. Beispiel:
Die in Beispiel 3 abgespeicherten Daten werden über den EXAMINE-Schalter nachkontrolliert.

Ablauf:
1. EXAMINE-Schalter auf 1
2. Mit A_3 bis A_0 die verschiedenen Adressen einstellen
3. In R_7 bis R_0 erscheinen die abgespeicherten Daten

5. Beispiel:
Überschreiben des Inhaltes der Adresse 0 1 1 1 mit den neuen Daten 1 1 1 1 0 0 0 0.

Ablauf:
1. B_7 bis B_0 auf 1 1 1 1 0 0 0 0 einstellen
2. A_7 bis A_0 auf 0 0 0 0 0 1 1 1 einstellen
3. Mit DEPOSIT-Schalter System takten

Kontrollieren Sie über EXAMINE nach, ob der neue Inhalt in Adresse 0 1 1 1 tatsächlich vorhanden ist.

6. Beispiel:
Überschreiben des Inhaltes der Adresse 1 0 1 0 mit den Daten 1 1 0 0 1 1 0 0 mit Hilfe des STA-Befehles.

Ablauf:
1. B_7 bis B_0 auf 1 1 0 0 1 1 0 0 einstellen
2. A_7 bis A_0 auf 1 1 0 1 0 0 0 0 einstellen (dies entspricht dem Befehl INP = Lade B-Eingänge in den Akkumulator)
3. Mit Schalter C_4 System takten (in R_7 bis R_0 muß jetzt die Information B_7 bis B_0 erscheinen)
4. A_7 bis A_0 auf 1 1 1 0 1 0 1 0 einstellen (dies entspricht laut Tab. 3.5.1 dem Befehl STA = Speichere Akku in Adresse *a a a a* ab)
5. Mit Schalter C_4 System takten

Kontrollieren Sie über EXAMINE, den neuen Inhalt der Speicheradresse 1 0 1 0.

7. Beispiel:
Den Inhalt der Adresse 1 1 1 0 in den Akkumulator laden.

Ablauf:
1. A_7 bis A_0 auf 0 0 1 1 1 1 1 0 einstellen (entspricht dem Befehl LDA = Lade Inhalt Adresse *a a a a*)
2. Mit Schalter C_4 System takten. In R_7 bis R_0 erscheint 0 0 0 0 1 1 1 0

8. Beispiel:
Addition der Hexadezimalzahlen $1\ 7_{16}$ und $3\ 2_{16}$

Ablauf:
1. B_7 bis B_0 auf 0 0 0 1 0 1 1 1 = $1\ 7_{16}$ einstellen
2. Daten B_7 bis B_0 über DEPOSIT in der Adresse 0 0 0 0 abspeichern
(A_3 bis A_0 auf 0 0 0 0 einstellen!)
3. B_7 bis B_0 auf 0 0 1 1 0 0 1 0 = $3\ 2_{16}$ einstellen
4. Über INP-Befehl die Daten B_7 bis B_0 in den Akku laden
5. A_7 bis A_0 auf 0 1 1 1 0 0 0 0 einstellen (entspricht dem Befehl ADD = Addiere Inhalt der Adresse 0 0 0 0)
6. Mit Schalter C_4 System takten. In R_7 bis R_0 erscheint das Additonsergebnis
0 1 0 0 1 0 0 1 $\triangleq$ $4\ 9_{16}$

Bei den arithmetischen und logischen Funktionen ADD, SUB, AND, IOR und XOR wird also immer der Inhalt einer Adresse mit dem jeweiligen Inhalt des Akkus verknüpft. Es ist gleichgültig, wie dabei die Schalter B_7 bis B_0 stehen. Nur über den Befehl INP an A_7 bis A_0 können die Daten B_7 bis B_0 in den Akku gelangen.
Im nächsten Beispiel wollen wir den Inhalt von 2 unterschiedlichen Adressen EXCLUSIV-ODER-verknüpfen.

9. Beispiel:
Die Inhalte der Adressen 1 0 1 0 und 0 1 1 1 werden EXCLUSIV-ODER-verknüpft.

Anmerkung: Wenn Sie in der Zwischenzeit genau das vorgeschriebene Experimentierprogramm durchgeführt haben bzw. das Experimentiersystem nicht zwischendurch abgeschaltet haben, steht in den Adressen folgender Inhalt:

Adresse 1 0 1 0 → 1 1 0 0 1 1 0 0
Adresse 0 1 1 1 → 1 1 1 1 0 0 0 0

Ist das nicht der Fall, über DEPOSIT die beiden Adressen entsprechend laden.

Ablauf:
1. A_7 bis A_0 auf 0 0 1 1 1 0 1 0 einstellen (entspricht Befehl LDA)
2. Mit Schalter C_4 System takten. In R_7 bis R_0 erscheint der Inhalt der Adresse 1 0 1 0, der jetzt auch Inhalt des Akkus ist

3. A_7 bis A_0 auf 1 0 1 1 0 1 1 1 einstellen (entspricht Befehl XOR)
4. Mit Schalter C_4 System takten. In R_7 bis R_0 erscheint das Ergebnis 0 0 1 1 1 1 0 0

Der neue Akku-Inhalt ist also 0 0 1 1 1 1 0 0. Wenn Sie jetzt z.B. noch einmal mit C_4 takten (A_7 bis A_0 bleiben unverändert), erscheint in R_7 bis R_0 1 1 0 0 1 1 0 0. Bei dem erneuten Takt wird nämlich die XOR-Verknüpfung des neuen Akku-Inhaltes mit dem nach wie vor unveränderten Inhalt der Adresse 0 1 1 1 gebildet. Es ergibt sich somit:

0 0 1 1 1 1 0 0	Inhalt Akku
1 1 1 1 0 0 0 0	Inhalt Adresse 0 1 1 1
1 1 0 0 1 1 0 0	

Alle anderen Befehle sind Ihnen vom Prinzip her bekannt.

Aufgaben:

1. Subtrahieren Sie folgende Aufgaben:
   a) $3\,4_{16} - 2\,1_{16} =$
   b) $-2\,4_{16} - 5\,4_{16} =$

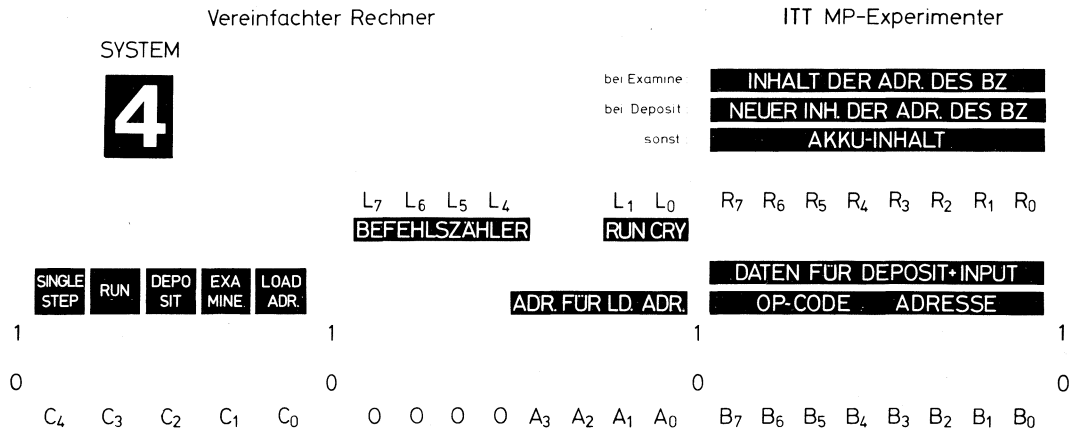
2. Erhöhen Sie den Inhalt der Adresse 0 1 1 1 über Programm um 1.

3. Führen Sie folgende Rechenoperationen durch:

$$3\,5_{16} + 1\,7_{16} - 2\,4_{16} =$$

Die Lösungen finden Sie auf Seite E22.

Experiment 5: Vereinfachter Rechner



Bei diesem Experiment wird ein vereinfachter, aber kompletter Rechner simuliert. Er hat denselben Befehlsvorrat wie der Akkumulator mit Datenspeicher in Experiment 4. Zusätzlich hat er einen HALT-Befehl (HLT), damit der Rechner am Ende eines Programms angehalten werden kann. Im Gegensatz zum Experiment 4 werden im 16-Wort-Speicher nicht nur Daten sondern auch das Programm abgespeichert. Das Programm und die Daten werden mit dem DEPOSIT-Schalter C_2 in den Speicher geladen. Damit ein Programm automatisch ablaufen kann, enthält der simulierte Rechner einen Befehlszähler (BZ). Welche der 16 Adressen gerade selektiert ist, wird durch die LEDs L_7 bis L_4 angezeigt. Die Funktionsweise des Befehlszählers können Sie wie folgt kontrollieren:

– Stellen Sie alle Schalter außer C_4 auf 0. Bei C_4 = 1 arbeitet das System im Single-Step-Betrieb, d.h., der Befehlszähler kann mit Schalter C_3 (RUN) in Einzelschritten getaktet werden
– In L_7 bis L_4 erscheint jetzt eine beliebige Adresse von 0 0 0 0 bis 1 1 1 1.
– Takten Sie das System mit RUN. An L_7 bis L_4 können Sie sehen, daß der Befehlszähler mit jedem Takt um einen Schritt höher springt.

Mit den Schaltern A_3 bis A_0 können Sie den Befehlszähler auf eine bestimmte Adresse laden.

Wenn Sie z.B. A_3 bis A_0 auf 0 1 1 0 einstellen und den Schalter LOAD-ADRESS (C_0) betätigen, wird der Befehlszähler auf diese Adresse gesetzt (Anzeige durch L_7 bis L_4). Wenn Sie jetzt mit dem RUN-Schalter weitertakten, zählt der Zähler von dieser Stellung weiter.

Mit den Schaltern B_7 bis B_0 können OP-Code und Adresse eingegeben werden. Hierbei ist unbedingt zu berücksichtigen, daß es sich um einen **Befehl** handelt, der in einer bestimmten Adresse abgespeichert wird. Wenn Sie z.B. B_7 bis B_0 auf 0 1 1 1 0 0 1 0 einstellen und den Schalter DEPOSIT C_2 takten, wird dieser Befehl in der Adresse abgespeichert, die gerade vom Befehlszähler selektiert ist. Der Befehl 0 1 1 1 0 0 1 0 besagt laut Tab. 3.5.1: Addiere den Inhalt der Adresse 0 0 1 0 zum Inhalt des Akkus. Dies bedeutet – und das ist unbedingt zu beachten – daß bei der Befehlszählerstellung, bei der dieser Befehl eingegeben wurde, diese Rechenoperation durchgeführt wird. Da hier eine neue Denkweise einsetzt, wollen wir das Prinzip an einem Beispiel ausführlich erläutern:

1. Beispiel:

Folgende Aufgabenstellung ist zu programmieren: Die an B_7 bis B_0 eingestellten Daten sollen mit dem Inhalt der Adresse 0 1 0 0 addiert werden.

Ablauf:

1. Alle Schalter zunächst in Stellung 0 bringen
2. Schalter LOAD-ADRESS (C_0) takten. Damit wird der Befehlszähler auf Adresse 0 0 0 0 gesetzt
3. Schalter B_7 bis B_0 auf 0 1 0 0 0 0 0 0 stellen. Dies entspricht dem Befehl CLA = Lösche Akku
4. Mit DEPOSIT-Schalter (C_2) takten. Damit ist der CLA-Befehl in der Adresse 0 0 0 0 gespeichert. Gleichzeitig springt der Befehlszähler auf Adresse 0 0 0 1, d.h., jetzt kann eine Information in dieser Adresse gespeichert werden
5. Schalter B_7 bis B_0 auf 1 1 0 1 0 0 0 0 einstellen. Dies entspricht dem INP-Befehl

6. Mit DEPOSIT-Schalter takten; der INP-Befehl ist in Adresse 0 0 0 1 gespeichert, Befehlszähler springt auf Adresse 0 0 1 0
7. Schalter B_7 bis B_0 auf 0 1 1 1 0 1 0 0 einstellen. Dies entspricht dem Additionsbefehl mit der Adresse 0 1 0 0
8. Mit DEPOSIT-Schalter takten, ADD-Befehl ist in Adresse 0 0 1 0 gespeichert
9. Schalter B_7 bis B_0 auf 1 1 1 1 0 0 0 0 stellen. Dies entspricht dem HALT-Befehl. Wenn am Ende eines Programms dieser Befehl nicht erscheint, rechnet das System unkontrolliert weiter
10. Mit DEPOSIT-Schalter takten
11. Damit sich ein kontrollierbares Ergebnis ergibt, speichern wir in Adresse 0 1 0 0 die Daten 1 1 1 1 0 0 0 0 ab. Hierzu B_7 bis B_0 auf 1 1 1 1 0 0 0 0 einstellen und mit DEPOSIT takten

Jetzt ist der Programmiervorgang abgeschlossen, und der Rechner kann die jeweils an den B-Schaltern eingestellten Daten mit dem Inhalt 1 1 1 1 0 0 0 0 der Adresse 0 1 0 0 addieren. Damit wir die einzelnen Schritte genau verfolgen können, schalten wir C_4 auf 1, d.h. SINGLE-STEP-Betrieb. Da der Befehlszähler jetzt bei Adresse 0 1 0 0 steht, setzen wir ihn durch Takten von C_0 = LOAD-ADRESS auf Adresse 0 0 0 0 zurück (A_3 bis A_0 auf 0 0 0 0). Als erste Aufgabe rechnen wir 0 0 0 0 1 1 1 1 plus Inhalt Adresse 0 1 0 0. Hierzu stellen wir B_7 bis B_0 auf 0 0 0 0 1 1 1 1 ein. Jetzt takten wir einmal mit Schalter RUN. Die Anzeige R_7 bis R_0 muß Null sein, da mit dem 1. Takt der Akkumulator gelöscht wird. Der Befehlszähler springt auf Adresse 0 0 0 1. Jetzt mit Schalter RUN wieder takten. Daten B_7 bis B_0 werden in den Akku geladen und erscheinen in R_7 bis R_0. Mit RUN nochmals takten. Jetzt erfolgt die Addition. Im Akkumulator und in der Anzeige steht das Ergebnis 1 1 1 1 1 1 1 1.
Mit RUN takten. Der Rechner arbeitet nicht weiter, da im Programm ein HALT-Befehl gespeichert ist. Sie können jetzt selbst mit verschiedenen Daten an B_7 bis B_0 das Programm wiederholen. Wesentlich ist, daß zu Beginn einer Aufgabe immer erst der Befehlszähler auf 0 0 0 0 zurückzustellen ist. Wenn Sie Beispiele wählen, die einen Übertrag ergeben, leuchtet die Carry-Anzeige L_0 auf.
Wenn der Schalter SINGLE-STEP auf 0 steht und dann der RUN-Schalter betätigt wird, läuft das Programm automatisch mit einer sehr hohen Systemfrequenz ab. Die einzelnen Zwischenschritte können dann nicht mehr mit dem Auge verfolgt werden. Lediglich ein kurzes Aufleuchten der Anzeige RUN (L_1) signalisiert, daß der Rechner arbeitet.

2. Beispiel:
Die Aufgabenstellung lautet, einen Vorwärtszähler zu simulieren. Hierzu verwenden wir den INC-Befehl.

Ablauf:
1. Befehlszähler auf 0 0 0 0 stellen
2. B_7 bi B_0 auf 0 1 0 1 0 0 0 0 einstellen
3. Mit DEPOSIT das System 16mal takten. Jetzt ist in jeder Adresse der INC-Befehl gespeichert. Da kein HALT-Befehl eingegeben wurde, läuft das System so lange, wie mit RUN getaktet wird. Wenn Sie C_4 = 0 einstellen und den RUN-Schalter auf 1 stellen, läuft der Vorgang automatisch ab. Ab R_3 können Sie ein deutliches Blinken der Lampen erkennen. Die Zählfrequenz läßt sich verringern, wenn Sie nur einen INC-Befehl und z.B. 15 NOP-Befehle eingeben. Wenn jetzt der Befehlszähler läuft, wird nur immer bei einer Adresse der Akku-Inhalt um 1 erhöht, während der übrigen 15 Adressen führt das System keine Operation aus.

3. Beispiel:
Vorwärtszähler mit niedriger Zählfrequenz.

Ablauf:
1. Befehlszähler auf 0 0 0 0 stellen
2. INC-Befehl laden
3. NOP-Befehl 15mal laden

Sie können die wesentlich niedrigere Zählfrequenz an R_7 bis R_0 deutlich erkennen.
Über den DEC-Befehl läßt sich ein Rückwärtszähler simulieren. Erstellen Sie selbst einmal ein Programm für einen langsamen Rückwärtszähler. Dies dürfte an dieser Stelle bestimmt keine Schwierigkeiten mehr bereiten.

Ein wesentliches Merkmal eines Rechners ist, daß er Entscheidungen treffen kann. Wie Sie später noch sehen werden, gibt es hierfür bestimmte Befehle. Der hier simulierte einfache Rechner verfügt nicht über diese Befehle. Über eine geschickte Programmierung ist es jedoch möglich, bestimmte Entscheidungen auch von diesem System treffen zu lassen.

4. Beispiel:
Größer/kleiner-Vergleich von 2 Zahlen.

In diesem Beispiel werden 2 Zahlen *A* und *B* miteinander verglichen. Die jeweils größere Zahl wird dabei zur Anzeige gebracht. Als Zahl *A* wählen wir $20_{16} \triangleq 100000_2$. Die Zahl *A* wird im Speicher abgespeichert. Die Zahl *B* kann an den Schaltern B_7 bis B_0 eingestellt werden. Sie wird dann mit *A* verglichen. Ist *A* größer als *B*, wird *A* angezeigt und umgekehrt. Der Rechner bildet die Differenz $B - A$ und trifft aufgrund des Ergebnisses die Entscheidung, welche Zahl in R_7 bis R_0 angezeigt wird. Der Ablauf ist in einem Flußdiagramm dargestellt (Bild 1).

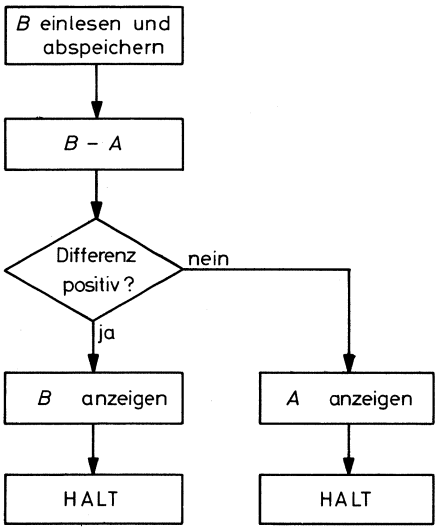


Bild 1
Flußdiagramm zum 4. Beispiel

Wir geben zunächst das Programm für diese Aufgabe an und besprechen anschließend die einzelnen Schritte (Tab. 1).

Adresse		Inhalt		Befehl	Kommentar
hexadez.	binär	hexadez.	binär		
0	0000	D *x*	1101 *xxxx*	INP	*B*-Eingänge in den Akku laden
1	0001	E E	11101110	STA	Inhalt Akku in Adresse E abspeichern
2	0010	8 F	10001111	SUB	Subtraktion *B* minus Inhalt Adresse F
3	0011	9 D	10011101	AND	höchste Stelle ausblenden
4	0100	7 C	01111100	ADD	Entscheidungsaddition
5	0101	E 7	11100111	STA	Ergebnis in Adresse 7 abspeichern
6	0110	3 F	00111111	LDA	Zahl *A* in Akku laden
7	0111	- - -	- - - - - - - -	ADD/HLT	Entscheidungsadresse
8	1000	3 E	00111110	LDA	Zahl *B* in Akku laden
9	1001	F *x*	1111 *xxxx*	HLT	System HALT
A	1010		*xxxxxxxx*	DATEN	nicht belegt
B	1011		*xxxxxxxx*		nicht belegt
C	1100		01110000		Daten für Entscheidungsaddition
D	1101		10000000		Maske für Ausblendung
E	1110		- - - - - - - -		Adresse für Zahl *B*
F	1111		00100000		Zahl *A*

Tab. 1
Programm zum 4. Beispiel

Entsprechend der Aufgabenstellung soll die im Speicher abgespeicherte Zahl $2\,0_{16}$ mit einer an B_7 bis B_0 eingestellten Zahl verglichen werden. Die Zahl A ist in Adresse F abgespeichert. Wird jetzt an B_7 bis B_0 eine Zahl B eingestellt, so muß diese zunächst in den Akku geladen werden (INP-Befehl).
Mit dem STA-Befehl wird dann die Zahl B in der Adresse E abgespeichert. Dabei wird der Inhalt des Akkus nicht verändert, d.h., B steht weiterhin auch im Akku.
Durch den SUB-Befehl wird die Differenz $B - A$ gebildet. Dabei entsteht bei $B > A$ ein positives und bei $B < A$ ein negatives Ergebnis. Nach der Zweierkomplementarithmetik wird eine positive Zahl durch eine 0, eine negative Zahl durch eine 1 in der werthöchsten Stelle gekennzeichnet. Entscheidungskriterium ist also das werthöchste bit des Ergebnisses der Subtraktion.
Aus diesem Grunde wird jetzt mit einem AND-Befehl das werthöchste bit ausgeblendet.
Beispiel:

```
   0 x x x x x x x      beliebige positive Zahl
∧  1 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0

   1 x x x x x x x      beliebige negative Zahl
∧  1 0 0 0 0 0 0 0
   1 0 0 0 0 0 0 0
```

In Adresse D ist 1 0 0 0 0 0 0 0 gespeichert. Der in Adresse 3 gespeicherte AND-Befehl bildet die UND-Verknüpfung zwischen Inhalt Akku und Inhalt Adresse D. Als Ergebnisse können nur 0 0 0 0 0 0 0 0 (positives Ergebnis der Subtraktion) bzw. 1 0 0 0 0 0 0 0 (negatives Ergebnis der Subtraktion im Akku) erscheinen.
Im nächsten Programmschritt wird nun eine Entscheidungsaddition durchgeführt. Entsprechend dem ADD-Befehl in Adresse 4 wird zu dem ausgeblendeten Ergebnis der Inhalt von Adresse C = 0 1 1 1 0 0 0 0 addiert.
Ist $B > A$ erhalten wir:

```
   0 0 0 0 0 0 0 0
+  0 1 1 1 0 0 0 0
   0 1 1 1 0 0 0 0
```

Ist $B < A$ erhalten wir:

```
   1 0 0 0 0 0 0 0
+  0 1 1 1 0 0 0 0
   1 1 1 1 0 0 0 0
```

Entsprechend der Befehlsstruktur des einfachen Rechners stellen die 4 werthöheren bit den OP-Code, die 4 wertniedrigeren bit die Adresse dar. Entscheidend ist, daß der OP-Code 1 1 1 1 den HLT-Befehl darstellt. Wenn wir nun das Ergebnis der Addition über den STA-Befehl in Adresse 7 abspeichern, wird der Rechner beim Abarbeiten des Programmes bei der Adresse 7 entweder anhalten (bei $B < A$) oder, wenn $B > A$ ist, eine Addition zwischen Akku-Inhalt und Inhalt Adresse 0 durchführen. Wenn der Befehlszähler bei $B < A$ gestoppt wird, muß die größere Zahl A angezeigt werden. Dies geschieht dadurch, daß in Adresse 6 der Akku über den LDA-Befehl mit dem Inhalt der Adresse F = $2\,0_{16}$ geladen wird.
Ist dagegen $B > A$, erfolgt bei Adresse 7 eine Addition von Akku-Inhalt und Inhalt Adresse 0, deren Ergebnis aber keine Bedeutung hat. In diesem Falle muß die in Adresse E gespeicherte Zahl B über einen LDA-Befehl in den Akku geladen werden, damit sie in R_7 bis R_0 angezeigt werden kann. In Adresse 9 ist für diesen Fall dann der HLT-Befehl programmiert.
Sicherlich werden Ihnen die hier dargelegten Gedankengänge kompliziert erscheinen. Das liegt ganz einfach daran, daß der vereinfachte Rechner noch keine direkten Entscheidungsbefehle enthält. Wir müssen ihn vielmehr so programmieren, daß er bei einem bestimmten Kriterium selbst einen HALT-Befehl erzeugt. Bevor wir das Programm mit konkreten B-Zahlen noch einmal durchsprechen, muß das Programm zunächst geladen werden:

1. Alle Schalter auf Null stellen
2. Schalter LOAD-ADD takten

3. Schalter B_7 bis B_0 auf 1 1 0 1 einstellen und DEPOSIT takten
4. B_7 bis B_0 auf 1 1 1 0 1 1 1 0 einstellen und DEPOSIT takten
5. B_7 bis B_0 auf 1 0 0 0 1 1 1 1 einstellen und DEPOSIT takten
6. B_7 bis B_0 auf 1 0 0 1 1 1 0 1 einstellen und DEPOSIT takten
7. B_7 bis B_0 auf 0 1 1 1 1 1 0 0 einstellen und DEPOSIT takten
8. B_7 bis B_0 auf 1 1 1 0 0 1 1 1 einstellen und DEPOSIT takten
9. B_7 bis B_0 auf 0 0 1 1 1 1 1 1 einstellen und DEPOSIT takten
10. DEPOSIT takten. Damit springt der Befehlszähler auf Adresse 8
11. B_7 bis B_0 auf 0 0 1 1 1 1 1 0 einstellen und DEPOSIT takten
12. B_7 bis B_0 auf 1 1 1 1 *x x x x* einstellen und DEPOSIT takten
13. DEPOSIT 2mal takten. Damit springt der Befehlszähler auf Adresse C
14. B_7 bis B_0 auf 0 1 1 1 0 0 0 0 einstellen und DEPOSIT takten
15. B_7 bis B_0 auf 1 0 0 0 0 0 0 0 einstellen und DEPOSIT takten
16. DEPOSIT takten, der Befehlszähler springt auf Adresse F
17. B_7 bis B_0 auf 0 0 1 0 0 0 0 0 einstellen und DEPOSIT takten

Damit steht das Programm im Speicher. Der Befehlszähler steht wieder bei Adresse 0 (L_7 bis L_4). Als Beispiel 1 wollen wir einen Vergleich zwischen $B = 7_{16}$ und $A = 2\ 0_{16}$ durchführen. Damit die einzelnen Schritte nachvollzogen werden können, System auf SINGLE-STEP-Betrieb schalten ($C_4 = 1$). An B_7 bis B_0 wird 0 0 0 0 0 1 1 1 $\triangleq$ 7_{16} eingestellt. Jetzt wird das System mit dem RUN-Schalter schrittweise getaktet.

1. Takt: In R_7 bis R_0 erscheinen die Daten B_7 bis B_0
2. Takt: Daten B_7 bis B_0 werden in Adresse E gespeichert
(Kontrollieren Sie über EXAMINE-Funktion. Vergessen sie nicht Befehlszähler über LOAD-ADR wieder auf Adresse 2 zurückzustellen)
3. Takt: R_7 bis R_0 gleich 1 1 1 0 0 1 1 1 $= -1\ 9_{16}$
4. Takt: R_7 bis R_0 gleich 1 0 0 0 0 0 0 0. Das werthöchste bit wird ausgeblendet
5. Takt: R_7 bis R_0 gleich 1 1 1 1 0 0 0 0. Durch die Addition wird der HLT-Befehl für Adresse 7 gebildet
6. Takt: R_7 bis R_0 bleibt, Akku-Inhalt wird in Adresse 7 abgespeichert
7. Takt: Inhalt Adresse F wird in den Akku geladen und erscheint in R_7 bis R_0
8. Takt: Keine Änderung, da HLT-Befehl. Auch ein weiteres Takten hat keinen Einfluß

Jetzt führen wir den Vergleich mit der Zahl $B = 3\ F_{16}$ durch. Über LOAD-ADR Befehlszähler auf Adresse 0 einstellen, und B_7 bis B_0 auf 0 0 1 1 1 1 1 1 $\triangleq$ $3\ F_{16}$ einstellen.

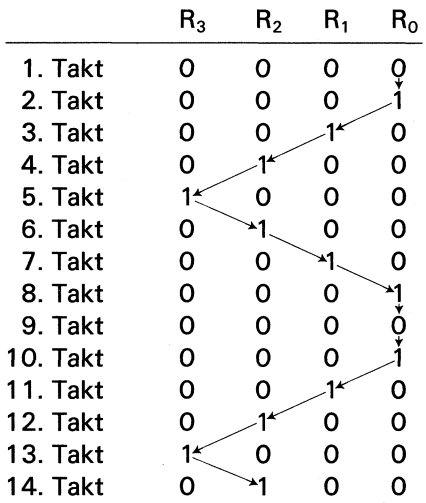
1. Takt: In R_7 bis R_0 erscheinen die Daten B_7 bis B_0
2. Takt: Daten werden in Adresse E gespeichert
3. Takt: R_7 bis R_0 gleich 0 0 0 1 1 1 1 1 $\triangleq$ $1\ F_{16}$
4. Takt: R_7 bis R_0 gleich 0 0 0 0 0 0 0 0 (Ausblenden)
5. Takt: R_7 bis R_0 gleich 0 1 1 1 0 0 0 0. Entspricht hier einem Additionsbefehl für Adresse 7
6. Takt: R_7 bis R_0 bleibt, Akku-Inhalt wird in Adresse 7 gespeichert
7. Takt: Inhalt Adresse F wird in Akku geladen und erscheint in R_7 bis R_0
8. Takt: R_7 bis R_0 gleich 1 1 1 1 0 0 0 0. Dieses Resultat ergibt sich aus der Addition von Akku-Inhalt und Inhalt Adresse 0

Inhalt Adresse 0:		1 1 0 1 0 0 0 0
Inhalt Akku:	+	0 0 1 0 0 0 0 0
		1 1 1 1 0 0 0 0

Es kann auch ein anderes Ergebnis erscheinen, wenn bei der Programmierung der Adresse 0 die bit b_3 bis b_0 einen anderen Wert gehabt haben. Vom Programm ändert sich nichts, da der INP-Befehl keine bestimmte Adresse spezifiziert (*x x x x*)
9. Takt: Die Zahl B (Inhalt Adresse E) wird in den Akku geladen. R_7 bis R_0 gleich 0 0 1 1 1 1 1 1
10. Takt: Keine Änderung, da HLT-Befehl. Auch ein weiteres Takten hat keinen Einfluß

Entscheidend bei diesem Programm ist der Gedankengang, über eine bestimmte Operation in einer bestimmten Adresse unter einer bestimmten Voraussetzung einen HALT-Befehl zu erzeugen.

5. Beispiel:
Das System soll so programmiert werden, daß beim Takten die LEDs R_3 bis R_0 nach folgendem Schema aufleuchten:



Für diese Aufgabenstellung ergibt sich z.B. folgende Programmierungsmöglichkeit (Tab. 2).

Adresse		Inhalt		Befehl	Kommentar
hexad.	binär	hexad.	binär		
0	0 0 0 0	4 *x*	0 1 0 0 *x x x x*	CLA	Lösche Akku
1	0 0 0 1	5 *x*	0 1 0 1 *x x x x*	INC	Incrementiere Akku
2	0 0 1 0	5 *x*	0 1 0 1 *x x x x*	INC	Incrementiere Akku
3	0 0 1 1	7 F	0 1 1 1 1 1 1 1	ADD	Addiere Akku-Inhalt mit 2
4	0 1 0 0	7 E	0 1 1 1 1 1 1 0	ADD	Addiere Akku-Inhalt mit 4
5	0 1 0 1	8 E	1 0 0 0 1 1 1 0	SUB	Subtrahiere von Akku-Inhalt 4
6	0 1 1 0	8 F	1 0 0 0 1 1 1 1	SUB	Subtrahiere von Akku-Inhalt 2
7	0 1 1 1	6 *x*	0 1 1 0 *x x x x*	DEC	Decrementiere Akku
8	1 0 0 0	6 *x*	0 1 1 0 *x x x x*	DEC	Decrementiere Akku
9	1 0 0 1	5 *x*	0 1 0 1 *x x x x*	INC	Incrementiere Akku
A	1 0 1 0	5 *x*	0 1 0 1 *x x x x*	INC	Incrementiere Akku
B	1 0 1 1	7 F	0 1 1 1 1 1 1 1	ADD	Addiere Akku-Inhalt mit 2
C	1 1 0 0	7 E	0 1 1 1 1 1 1 0	ADD	Addiere Akku-Inhalt mit 4
D	1 1 0 1	8 E	1 0 0 0 1 1 1 0	SUB	Subtrahiere von Akku-Inhalt 4
E	1 1 1 0	0 4	0 0 0 0 0 1 0 0	DATEN	
F	1 1 1 1	0 2	0 0 0 0 0 0 1 0		

Tab. 2
Programm zum 5. Beispiel

Dieses Programm ist relativ einfach und bedarf daher keiner umfangreichen Erklärung. Für die Speicherung der Daten werden nur 2 Adressen benötigt, da die Additions- und Subtraktionsbefehle dieselben Daten benötigen. Zu bemerken ist außerdem noch, daß die Aufgabenstellung auch noch über andere Programme möglich ist. So kann z.B. der Befehl INC, der beim 2. Takt das Muster 0 0 0 1 in R_3 bis R_0 erzeugen soll, durch den Befehl SP1 = Setze Akku gleich 1, ersetzt werden.
Störend im Programmablauf ist, daß die Adressen E und F NOP-Befehle darstellen. Dadurch ändert sich bei 2 Takten der Akku-Inhalt nicht. Umfangreichere Mikrorechner lassen sich dagegen so programmieren, daß ein kontinuierlicher Ablauf entsteht.
Vielleicht taucht bei Ihnen jetzt die Frage auf, welche praktische Nutzanwendung ein solches Programm haben könnte. Beispielsweise könnte man anstatt der LEDs R_3 bis R_0 einen

Digital-Analog-Wandler (D/A-Wandler) an das System anschließen. Ein D/A-Wandler wandelt ein digitales Signal in ein analoges Signal um. Er könnte z.B. folgendes Verhalten aufweisen (Bild 2).

R_3	R_2	R_1	R_0	A
0	0	0	0	0 V
0	0	0	1	1 V
0	0	1	0	2 V
0	1	0	0	4 V
1	0	0	0	8 V

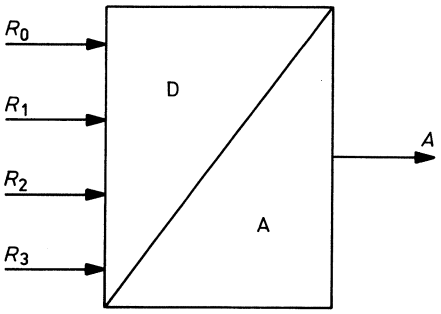


Bild 2
D/A-Wandler

Liegt an seinen Eingängen das Wort 0 0 0 0, beträgt die Ausgangsspannung 0 V, bei 0 0 0 1 ist sie 1 V usw.
Läßt man nun den Befehlszähler mit einer bestimmten Frequenz laufen, so kann man mit einem Oszilloskop am Ausgang des D/A-Wandlers folgendes messen (Bild 3).

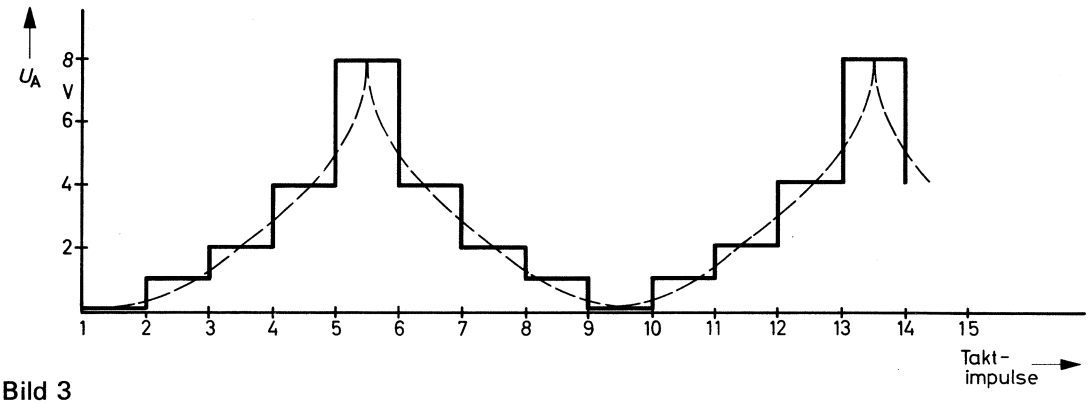


Bild 3
Ausgangsspannung eines D/A-Wandlers

Die so gewonnene Treppenspannung kann durch entsprechende Formung als eine quadratische Funktion benutzt werden, die irgendeinen Vorgang steuert.

Aufgaben:

1. Erstellen Sie die Programme für folgende Aufgaben:
   a) $5_{10} \cdot B =$
   b) $8_{10} \cdot B =$

2. Der 8421-BCD-Code entspricht den Binärzahlen 0 0 0 0 bis 1 0 0 1. Mit 4-bit-Wörtern lassen sich 16 verschiedene Zahlen darstellen. Damit werden für den 8421-BCD-Code die Zahlen von 1 0 1 0 bis 1 1 1 1 nicht benötigt und als Pseudotetraden bezeichnet. Entwerfen Sie ein Programm, das bei Eingabe einer Pseudotetrade mit B_3 bis B_0 in der Anzeige R_7 bis R_0 −1, d.h. 1 1 1 1 1 1 1 1, erscheinen läßt. Handelt es sich bei der Eingabe um keine Pseudotetrade, soll in R_3 bis R_0 die Eingabe B_3 bis B_0 erscheinen.

Anmerkung:
Orientieren Sie sich bei dieser Aufgabe an dem 4. Beispiel.

Die Lösungen finden Sie auf Seite E22.

Lösungen zu Experiment 1

1. a) A-Schalter: $0\,1\,1\,1\,1\,1\,0\,1 \triangleq 125_{10}$
+ B-Schalter: $0\,0\,1\,0\,1\,0\,0\,0 \triangleq 40_{10}$
Ergebnis: $\underbrace{1\,0\,1\,0\,0\,1\,0\,1}_{R_7 \text{ bis } R_0} \triangleq 165_{10}$

b) A-Schalter: $1\,0\,1\,1\,1\,0\,0\,0 \triangleq 184_{10}$
+ B-Schalter: $0\,1\,1\,0\,0\,1\,0\,0 \triangleq 100_{10}$
$\underbrace{1\,|\,0\,0\,0\,1\,1\,1\,0\,0}_{L_0 \quad R_7 \text{ bis } R_0} \triangleq 284_{10}$

2. a) A-Schalter: $0\,1\,1\,1\,1\,0\,0\,0 \triangleq 120_{10}$
$\overline{A}$-Schalter 1: $1\,0\,0\,0\,0\,1\,1\,1 \triangleq \overline{A}$
INC-Schalter 1: $\underbrace{1\,0\,0\,0\,1\,0\,0\,0}_{R_7 \text{ bis } R_0} \triangleq -A = \overline{A} + 1$

b) A-Schalter: $0\,0\,0\,0\,1\,1\,0\,0 \triangleq 12_{10}$
$\overline{A}$-Schalter 1: $1\,1\,1\,1\,0\,0\,1\,1 \triangleq \overline{A}$
INC-Schalter 1: $\underbrace{1\,1\,1\,1\,0\,1\,0\,0}_{R_7 \text{ bis } R_0} \triangleq -A = \overline{A} + 1$

c) A-Schalter: $0\,0\,0\,0\,0\,0\,1\,0 \triangleq 2_{10}$
$\overline{A}$-Schalter 1: $1\,1\,1\,1\,1\,1\,0\,1 \triangleq \overline{A}$
INC-Schalter 1: $\underbrace{1\,1\,1\,1\,1\,1\,1\,0}_{R_7 \text{ bis } R_0} \triangleq -A = \overline{A} + 1$

3. a) A-Schalter: $0\,1\,1\,1\,1\,0\,0\,0 \triangleq 120_{10}$
B-Schalter: $0\,1\,1\,0\,0\,1\,0\,0 \triangleq 100_{10}$
$\overline{B}$-Schalter 1: $\underbrace{1\,|\,0\,0\,0\,1\,0\,0\,1\,1}_{L_0 \quad R_7 \text{ bis } R_0} \triangleq A + \overline{B}$
INC-Schalter 1: $\underbrace{1\,|\,0\,0\,0\,1\,0\,1\,0\,0}_{L_0 \quad R_7 \text{ bis } R_0} \triangleq A + \overline{B} + 1 = A - B$
Ergebnis: $0\,0\,0\,1\,0\,1\,0\,0 = 20_{10}$

b) A-Schalter: $1\,0\,0\,0\,0\,0\,1\,0 \triangleq 130_{10}$
B-Schalter: $1\,0\,0\,0\,1\,1\,0\,0 \triangleq 140_{10}$
$\overline{B}$-Schalter 1: $\underbrace{1\,1\,1\,1\,0\,1\,0\,1}_{R_7 \text{ bis } R_0} \triangleq A + \overline{B}$
INC-Schalter 1: $\underbrace{1\,1\,1\,1\,0\,1\,1\,0}_{R_7 \text{ bis } R_0} = A + \overline{B} + 1 = A - B$
Ergebnis: $1\,1\,1\,1\,0\,1\,1\,0 = -10_{10}$

Lösungen zu Experiment 2

1. a) Funktion C_3 bis C_0: $0\,1\,1\,1$
A-Schalter: $0\,0\,1\,1\,0\,1\,0\,0 \triangleq 3\,4_{16}$
+ B-Schalter: $0\,0\,1\,0\,0\,0\,0\,1 \triangleq 2\,1_{16}$
$\underbrace{0\,1\,0\,1\,0\,1\,0\,1}_{R_7 \text{ bis } R_0} \triangleq 5\,5_{16}$

b) Funktion C_3 bis C_0: 1 0 0 0
A-Schalter: 0 0 0 1 0 1 0 0 ≙ $1\,4_{16}$
$-B$-Schalter: 0 1 0 0 1 0 0 0 ≙ $4\,8_{16}$
1 | 1 1 0 0 1 1 0 0 ≙ $-3\,4_{16}$
L_0 R_7 bis R_0

2. a) Funktion C_3 bis C_0: 1 0 0 0
A-Schalter: 1 0 0 0 0 0 0 1 ≙ $8\,1_{16}$
$-B$-Schalter: 0 1 1 1 0 1 0 1 ≙ $7\,5_{16}$
0 0 0 0 1 1 0 0 ≙ $0\,C_{16}$
R_7 bis R_0

b) Funktion C_3 bis C_0: 1 0 0 0
A-Schalter: 0 1 1 1 0 1 1 0 ≙ $7\,6_{16}$
$-B$-Schalter: 1 0 0 0 0 1 0 0 ≙ $8\,4_{16}$
1 | 1 1 1 1 0 0 1 0 ≙ $-0\,E_{16}$
L_0 R_7 bis R_0

3. a) Funktion C_3 bis C_0: 1 0 1 1
A-Schalter: 1 1 0 0 1 1 1 1
B-Schalter: 0 0 0 0 0 0 1 1
1 1 0 0 1 1 0 0
R_7 bis R_0

b) Funktion C_3 bis C_0: 1 0 0 1
A-Schalter: 1 1 0 1 0 1 1 0
B-Schalter: 0 0 0 0 1 1 1 1
0 0 0 0 0 1 1 0
R_7 bis R_0

Lösungen zu Experiment 3

1. a) 0 0 1 0 1 1 0 0 ≙ 44_{10}
R_7 bis R_0

b) 1 | 1 1 1 1 0 0 1 1 ≙ -13_{10}
L_0 R_7 bis R_0

c) 1 0 0 0 0 1 1 0 ≙ -122_{10}
R_7 bis R_0

Anmerkung: Bei der Aufgabe c) ist darauf zu achten, daß $-4\,8_{16}$ als Zweierkomplement zu bearbeiten ist. Der Rechenablauf ist also folgender:

1. Über CLA löschen
2. $4\,8_{16}$ eingeben
3. Vom Akku-Inhalt das Zweierkomplement bilden
4. $3\,2_{16}$ hiervon über SUB-Befehl subtrahieren

2. 1 1 0 1 0 1 1 0
R_7 bis R_0

Lösungen zu Experiment 4

1. a)
   1. $2\ 1_{16}$ an B_7 bis B_0 einstellen
   2. Über DEPOSIT B-Daten in eine Adresse laden, z.B. Adresse 1 1 1 1
   3. $3\ 4_{16}$ an B_7 bis B_0 einstellen
   4. Über INP-Befehl Akku mit B-Daten laden
   5. SUB-Befehl mit Adresse 1 1 1 1 einstellen

   In R_7 bis R_0 erscheint das Ergebnis 0 0 0 1 0 0 1 1 $\triangleq\ 1\ 3_{16}$

   b)
   1. $5\ 4_{16}$ an B_7 bis B_0 einstellen
   2. Über DEPOSIT B-Daten in eine Adresse laden, z.B. Adresse 1 1 1 1
   3. $2\ 4_{16}$ an B_7 bis B_0 einstellen
   4. Über INP-Befehl Akku mit B-Daten laden
   5. Über CMA-Befehl Akku-Inhalt komplementieren
   6. Über INC-Befehl 1 zum Akku-Inhalt addieren
      In R_7 bis R_0 steht jetzt das Zweierkomplement von $2\ 4_{16}$ (1 1 0 1 1 1 0 0)
   7. SUB-Befehl mit Adresse 1 1 1 1 einstellen und System takten
      In R_7 bis R_0 erscheint das Ergebnis 1 0 0 0 1 0 0 0 $\triangleq\ -78_{16}$

2.
   1. Inhalt der Adresse 0 1 1 1 über EXAMINE feststellen z.B. 0 1 0 1 0 0 0 0
   2. Über LDA-Befehl Inhalt Adresse 0 1 1 1 in Akku laden
   3. Über INP-Befehl Akku-Inhalt plus 1 bilden
   4. Neuen Akku-Inhalt über STA-Befehl in Adresse 0 1 1 1 laden

3.
   1. $2\ 4_{16}$ in Speicher laden, z.B. Adresse 0 0 0 0
   2. $1\ 7_{16}$ in Speicher laden, z.B. Adresse 0 0 0 1
   3. $3\ 5_{16}$ in Akku laden
   4. Über ADD-Befehl mit Adresse 0 0 0 1 die Rechenoperation $3\ 5_{16} + 1\ 7_{16}$ durchführen
   5. Über SUB-Befehl mit Adresse 0 0 0 0 $2\ 4_{16}$ vom Zwischenergebnis subtrahieren
      In R_7 bis R_0 erscheint das Ergebnis 0 0 1 0 1 0 0 0 $\triangleq\ 28_{16}$

Lösungen zu Experiment 5

1. a)

Adresse	Inhalt	Befehl	Kommentar
0	1 1 0 1 *x x x x*	INP	Lade B-Eingänge in Akku
1	1 1 1 0 1 1 1 1	STA	Speichere Akku-Inhalt in Adresse F
2	0 1 1 1 1 1 1 1	ADD	Addiere Akku mit Adresse F
3	0 1 1 1 1 1 1 1	ADD	Addiere Akku mit Adresse F
4	0 1 1 1 1 1 1 1	ADD	Addiere Akku mit Adresse F
5	0 1 1 1 1 1 1 1	ADD	Addiere Akku mit Adresse F
6	1 1 1 1 *x x x x*	HLT	HALT

b)

Adresse	Inhalt	Befehl
0	1 1 0 1 *x x x x*	INP
1	1 1 1 0 1 1 1 1	STA
2	0 1 1 1 1 1 1 1	ADD
3	0 1 1 1 1 1 1 1	ADD
4	0 1 1 1 1 1 1 1	ADD
5	0 1 1 1 1 1 1 1	ADD
6	0 1 1 1 1 1 1 1	ADD
7	0 1 1 1 1 1 1 1	ADD
8	0 1 1 1 1 1 1 1	ADD
9	1 1 1 1 *x x x x*	HLT

2. Kriterium für die Aufgabenstellung ist ein größer/kleiner-Vergleich der Eingabe B_3 bis $B_0 = n$ mit der Zahl $9_{10} \triangleq 1\ 0\ 0\ 1$. Ist $n > 9_{10}$, handelt es sich um eine Pseudotetrade, bei $n < 9_{10}$ handelt es sich um ein Codewort des 8421-Codes. Anhand der Subtraktion $n - 9_{10}$ kann die entsprechende Entscheidung getroffen werden. Damit ergibt sich folgendes Flußdiagramm (Bild 4).

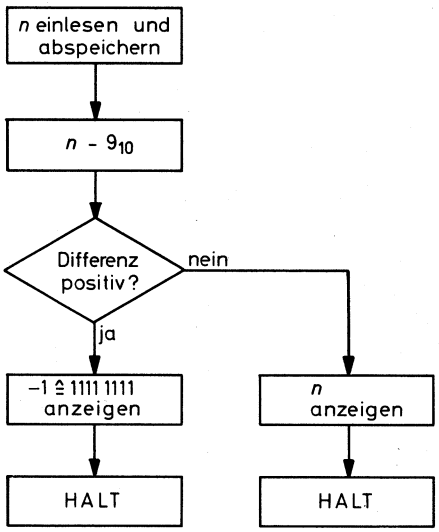


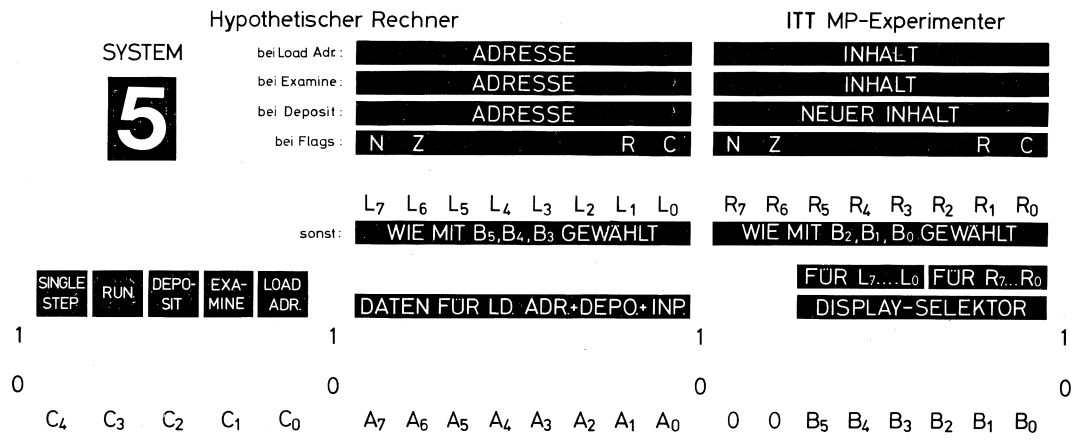
Bild 4
Flußdiagramm zu Aufgabe 2

Adresse	Inhalt	Befehl	Kommentar
0	1 1 0 1 *x x x x*	INP	B_3 bis B_0 in Akku laden
1	1 1 1 0 1 1 1 0	STA	Akku in Adresse E speichern
2	1 0 0 0 1 1 1 1	SUB	Inhalt Adresse F von Akku subtrahieren
3	1 0 0 1 1 1 0 1	AND	höchste Stelle ausblenden
4	0 1 1 1 1 1 0 0	ADD	Entscheidungsaddition
5	1 1 1 0 0 1 1 1	STA	Ergebnis in Adresse 7 abspeichern
6	0 0 1 1 1 1 1 0	LDA	B_3 bis B_0 in Akku laden
7	– – – – – – – –	ADD/HLT	Entscheidungsadresse
8	1 1 0 0 *x x x x*	SM1	Akku −1 setzen
9	1 1 1 1 *x x x x*	HLT	HALT
A	*x x x x x x x x*		nicht belegt
B	*x x x x x x x x*		nicht belegt
C	0 1 1 1 0 0 0 0		DATEN
D	1 0 0 0 0 0 0 0		
E	B_3 bis B_0		
F	0 0 0 0 1 0 0 1		

Da bei diesem Programm als Entscheidungskriterium $n - 9_{10}$ gewählt wurde, entsteht der Nachteil, daß bei B_3 bis B_0 gleich $1\ 0\ 0\ 1 \triangleq 9_{10}$ eine Pseudotetrade angezeigt wird. Das ist aber falsch. Wenn wir als Kriterium $9_{10} - n$ wählen, wird dieser Nachteil aufgehoben. Damit ergibt sich folgendes Programm:

Adresse	Inhalt	Befehl	Kommentar
0	1 1 0 1 *x x x x*	INP	B_3 bis B_0 in Akku laden
1	1 1 1 0 1 1 1 0	STA	Akku in Adresse E speichern
2	0 0 1 1 1 1 1 1	LDA	Inhalt Adresse F in Akku laden
3	1 0 0 0 1 1 1 0	SUB	Inhalt Adresse E von Akku subtrahieren
4	1 0 0 1 1 1 0 1	AND	höchste Stelle ausblenden
5	0 1 1 1 1 1 0 0	ADD	Entscheidungsaddition
6	1 1 1 0 1 0 0 0	STA	Ergebnis in Adresse 8 abspeichern
7	1 1 0 0 *x x x x*	SM1	Akku −1 setzen
8	– – – – – – – –	ADD/HLT	Entscheidungsadresse
9	0 0 1 1 1 1 1 0	LDA	Inhalt Adresse E in Akku laden
A	1 1 1 1 *x x x x*	HLT	HALT
B	*x x x x x x x x*		nicht benutzt
C	0 1 1 1 0 0 0 0		DATEN
D	1 0 0 0 0 0 0 0		
E	B_3 bis B_0		
F	0 0 0 0 1 0 0 1		

Experiment 6: Hypothetischer Mikrorechner



Der hypothetische Mikrorechner bildet das zentrale Thema dieses Lehrganges. Aus diesem Grunde ist es besonders wichtig, daß Sie dieses System einwandfrei beherrschen. Die Versuche in diesem Experiment sollen in erster Linie dazu dienen, Sie mit dem hypothetischen Mikrorechner vertraut zu machen. Im nächsten Lehrheft werden dann ausführlich die Programmiertechniken behandelt.

Bedeutung der Schalter:

Die Schaltergruppe C_4 bis C_0 hat die gleiche Bedeutung wie in Experiment 5. Wie aus der aufgelegten Schablone zu erkennen ist, wird bei LOAD-ADR. gleich 1 die an den Schaltern A_7 bis A_0 eingestellte Adresse angewählt. Solange $C_0 = 1$ ist, erscheint in den LEDs L_7 bis L_0 die angewählte Adresse, der Inhalt dieser Adresse wird in R_7 bis R_0 angezeigt. Das gleiche gilt für die Betätigung von EXAMINE und DEPOSIT. Bei DEPOSIT = 1 erscheint in R_7 bis R_0 der mit dem Takt abgespeicherte neue Adresseninhalt. Werden die erwähnten Schalter wieder in die Stellung 0 zurückgesetzt, erscheinen in den beiden Anzeigenreihen die Daten, die über die Schalter A_5 bis A_0 abgerufen werden. Hierbei kann mit den Schaltern B_5 bis B_3 die Anzeige L_7 bis L_0 und mit den Schaltern B_2 bis B_0 die Anzeige R_7 bis R_0 angewählt werden. Die Schalter B_7 und B_6 haben keine Bedeutung. Bei gleicher Einstellung von B_5 bis B_3 und B_2 bis B_0 erscheinen in R_7 bis R_0 und L_7 bis L_0 die gleichen Daten. Der Anzeigecode (Display-Code) in Tab. 1 spezifiziert die einzelnen Anzeigemöglichkeiten.

B_5 bis B_3 bzw. B_2 bis B_0	Anzeige in L_7 bis L_0 bzw. R_7 bis R_0
0 0 0	Inhalt von R0
0 0 1	Inhalt von R1
0 1 0	Inhalt von R2
0 1 1	Inhalt von R3
1 0 0	Stellung des Befehlszählers PC
1 0 1	Zustände der Flags
1 1 0	Speicherwort, dessen Adresse vom Befehlszähler spezifiziert ist
1 1 1	Speicherwort, dessen Adresse mit A_7 bis A_0 spezifiziert ist

Tab. 1
Anzeigecode

Die Codes 0 0 0 bis 0 1 1 bedürfen keiner weiteren Erklärung.
- Code 1 0 0 zeigt an, welche Adresse vom Befehlszähler gerade angewählt wird.
- Code 1 0 1 gibt Auskunft darüber, welche Zustände die 4 Flags gerade einnehmen.

– Code 1 1 0 gibt den Inhalt der Speicheradresse an, die vom Befehlszähler gerade angewählt ist.
– Code 1 1 1 zeigt den Inhalt einer Speicheradresse an, die mit den Schaltern A_7 bis A_0 frei wählbar ist.
Wesentlich ist, daß die Schalter B_5 bis B_0 keinen Einfluß auf den Funktionsablauf haben. Sie dienen lediglich zur Anzeige bestimmter Daten, wenn ein Programm abläuft.
Damit Sie mit diesen grundsätzlichen Eigenschaften des Rechners vertraut werden, führen Sie nachfolgendes Übungsprogramm Schritt für Schritt durch:

1. Alle Schalter auf Null stellen
2. A_7 bis A_0 auf 4 0_{16} einstellen
3. LOAD-ADR. auf 1 stellen (nicht takten!)
In L_7 bis L_0 erscheint die an A_7 bis A_0 eingestellte Adresse, deren momentaner Inhalt in R_7 bis R_0 angezeigt wird
4. LOAD-ADR. auf 0 stellen, in L_7 bis L_0 und R_7 bis R_0 erscheint der zufällige Inhalt von R0, da mit den Schaltergruppen B_5 bis B_3 und B_2 bis B_0 jeweils der Code 0 0 0 eingestellt ist
5. Schalter B_2 bis B_0 auf 1 0 0 einstellen. In R_7 bis R_0 erscheint die Adresse 4 0_{16}, die über LOAD-ADR. geladen wurde
6. A_7 bis A_0 auf 1 0 0 0 0 1 0 0 einstellen und DEPOSIT auf 1 stellen. In L_7 bis L_0 erscheint die Adresse 4 0_{16}, in R_7 bis R_0 die an A_7 bis A_0 eingestellten Daten. Schalter DEPOSIT auf 0 zurückstellen. Bei B_2 bis B_0 = 1 0 0 erscheint jetzt in R_7 bis R_0 die nächste Adresse 4 1_{16}
7. A_7 bis A_0 auf F E_{16} einstellen und DEPOSIT takten
8. A_7 bis A_0 auf 0 1_{16} einstellen und DEPOSIT takten
9. A_7 bis A_0 auf 1 1_{16} einstellen und DEPOSIT takten
10. A_7 bis A_0 auf 0 6_{16} einstellen und DEPOSIT takten
11. A_7 bis A_0 auf 0 0_{16} einstellen und DEPOSIT takten
12. A_7 bis A_0 auf 4 0_{16} stellen und LOAD-ADR. takten

Wenn Sie alle Eingaben genau vorgenommen haben, ist der Rechner mit einem bestimmten Programm geladen und über Punkt 12 wieder auf die Anfangsadresse 4 0_{16} zurückgestellt. Mit EXAMINE überprüfen wir die Eingaben:

1. Alle Schalter auf 0
2. EXAMINE auf 1 stellen. In L_7 bis L_0 erscheint die Adresse 4 0_{16}, in R_7 bis R_0 deren Inhalt 8 4_{16}, also der erste Befehl des Programms. Hierbei handelt es sich um den Befehl LOAD R0, F E_{16} (lade Akkumulator R0 mit dem Inhalt der Adresse F E_{16}). Diese **Datenadresse** ist in der nächsten Programmadresse abgespeichert
3. EXAMINE über 0 wieder auf 1 stellen. In L_7 bis L_0 erscheint die Adresse 4 1_{16}, der Inhalt F E_{16} erscheint in R_7 bis R_0
4. Punkt 3. wiederholen. In L_7 bis L_0 erscheint Programmadresse 4 2_{16}. Der Inhalt dieser Adresse 0 1_{16} (Anzeige in R_7 bis R_0) entspricht dem Befehl MOVE R1, R0. Dieser Befehl bewirkt, daß die Daten aus R0 in R1 transferiert werden
5. Punkt 4. wiederholen. In L_7 bis L_0 erscheint Adresse 4 3_{16}. Der Inhalt 1 1_{16} entspricht dem Befehl ADDR R1, R0. Hierbei werden die Daten des Registers R0 mit dem Inhalt R1 in R1 addiert
6. Punkt 5. wiederholen. In der Adresse 4 4_{16} ist der Befehl 0 6_{16}, d.h. MOVE R2, R1, abgespeichert. Hierbei werden die Daten von R1 in R2 gebracht
7. Punkt 6. wiederholen. In der Adresse 4 5_{16} ist der HALT-Befehlt abgespeichert

Damit dieses Programm mit definierten Daten ablaufen kann, speichern wir in Adresse F E_{16} die Zahl 1 0_{16}:

1. A_7 bis A_0 auf F E_{16} einstellen
2. Schalter LOAD-ADR. takten
3. A_7 bis A_0 auf 1 0_{16} einstellen
4. DEPOSIT takten

Jetzt stellen wir über LOAD-ADR. das System wieder auf die Prorammadresse 4 0_{16} zurück und schalten über C_4 SINGLE-STEP-Betrieb ein.
Bevor wir das Programm schrittweise ablaufen lassen, stellen wir in Tab. 2 noch einmal die Befehle zusammen.

Adresse (hexadez.)	Inhalt Maschinencode	Befehl	Kommentar
4 0	1 0 0 0 0 1 0 0	LOAD R0, F E	
4 1	1 1 1 1 1 1 1 0		Datenadresse
4 2	0 0 0 0 0 0 0 1	MOVE R1, R0	
4 3	0 0 0 1 0 0 0 1	ADDR R1, R0	
4 4	0 0 0 0 0 1 1 0	MOVE R2, R1	
4 5	0 0 0 0 0 0 0 0	HLT	
.			
.			
.			
F E	0 0 0 1 0 0 0 0	Daten	

Tab. 2
Zusammenfassung der bisher eingegebenen Befehle

Zur Beobachtung des Datenflusses stellen wir B_2 bis B_0 auf 1 0 0 (Stellung PC) und B_5 bis B_3 auf 0 0 0 (Inhalt R0). Mit Schalter RUN wird jetzt das System einmal getaktet. In L_7 bis L_0 erscheint $1\ 0_{16}$, also die Daten aus Adresse $4\ 1_{16}$, da hier (durch MODE 0 1 im Maschinencode bedingt) kein neuer Befehl, sondern eine Datenadresse abgespeichert ist.
Schalter B_5 bis B_3 auf 0 0 1 einstellen, und RUN erneut takten. Die Daten von R0 erscheinen jetzt auch in R1. System mit RUN takten. In L_7 bis L_0 erscheint das Additionsergebnis $2\ 0_{16}$.

Schalter B_5 bis B_3 auf 0 1 0 stellen und mit RUN takten. Die Daten von R1 erscheinen auch in R2. Weiteres Takten hat keinen Einfluß mehr auf das System, da in Adresse $4\ 5_{16}$ ein HALT-Befehl programmiert ist.
Bevor wir jetzt weitere Beispiele programmieren, einige Worte über den Speicher. Insgesamt hat der Speicher (RAM) eine Kapazität von 256 Wörtern à 8 bit. Hierbei ist zu berücksichtigen, daß die Adressen von $0\ 1_{16}$ bis $1\ 8_{16}$ mit einem Unterprogramm fest belegt sind. Sie dürfen also bei einem **Programm nicht** benutzt werden. Eine weitere Besonderheit stellt die Adresse $F\ F_{16}$ dar. Sie wird als INPUT-Adresse der A-Schalter benutzt und darf für ein normales Programm ebenfalls nicht benutzt werden. Über Adresse $F\ F_{16}$ können die an den Schaltern A_7 bis A_0 eingestellten Daten abgerufen werden. Diese Adresse ersetzt praktisch den INP-Befehl aus Experiment 5. Wenn Sie z.B. in Adresse $4\ 1_{16}$ des vorherigen Beispieles anstatt $F\ E_{16}$ die Adresse $F\ F_{16}$ abspeichern, können Sie das Programm mit den jeweils an A_7 bis A_0 eingestellten Daten ablaufen lassen.
Der für Programmierungszwecke zur Verfügung stehende Datenbereich reicht also von Adresse $1\ 9_{16}$ bis $F\ E_{16}$.

Experiment 6a

Nachdem Sie nun grundsätzlich mit der Anwendung des hypothetischen Mikrorechners vertraut sind, wollen wir nachfolgend einige kleine Programme erstellen.

1. Beispiel:
Die Register R0 bis R3 sollen mit folgenden Daten geladen werden

R0: 0 7_{16}
R1: 0 F_{16}
R2: 1 F_{16}
R3: 2 A_{16}

Dann soll der Inhalt R0 mit dem Inhalt R3 in R3 addiert werden. Von dem Ergebnis R3 soll in R2 mit dessen Inhalt die XOR-Verknüpfung gebildet werden. Das jetzt in R2 stehende Ergebnis wird dann in R1 mit dessen Inhalt ODER-verknüpft.

Ablauf:
Als erste Programmadresse wählen wir 4 0_{16}, die Daten werden rückwärts mit der Adresse F E_{16} beginnend abgespeichert (Tab. 3).

Adresse (hexadez)	Inhalt Maschinencode	Befehl
4 0	1 0 0 0 0 1 0 0	LOAD R0, F E
4 1	1 1 1 1 1 1 1 0	
4 2	1 0 0 0 0 1 0 1	LOAD R1, F D
4 3	1 1 1 1 1 1 0 1	
4 4	1 0 0 0 0 1 1 0	LOAD R2, F C
4 5	1 1 1 1 1 1 0 0	
4 6	1 0 0 0 0 1 1 1	LOAD R3, F B
4 7	1 1 1 1 1 0 1 1	
4 8	0 0 0 1 0 0 1 1	ADDR R3, R0
4 9	0 1 0 0 1 1 1 0	XORR R2, R3
4 A	0 0 1 1 1 0 0 1	IORR R1, R2
4 B	0 0 0 0 0 0 0 0	HLT
.		
.		
.		
F B	0 0 1 0 1 0 1 0	Daten
F C	0 0 0 1 1 1 1 1	
F D	0 0 0 0 1 1 1 1	
F E	0 0 0 0 0 1 1 1	

Tab. 3
Programm zum 1. Beispiel

Laden Sie dieses Programm. Wenn Sie beim Programmieren keinen Fehler gemacht haben, erscheint in R1 als Lösung 2 F_{16}. Das Laden der Daten in F B_{16} bis F E_{16} geschieht folgendermaßen:

1. F B_{16} an A_7 bis A_0 einstellen
2. LOAD-ADR. takten
3. 2 A_{16} an A_7 bis A_0 einstellen
4. DEPOSIT takten
5. 1 F_{16} an A_7 bis A_0 einstellen
6. DEPOSIT takten
7. 0 F_{16} an A_7 bis A_0 einstellen

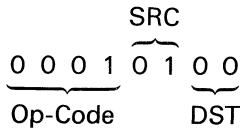
8. DEPOSIT takten
9. 0 7_{16} an A_7 bis A_0 einstellen
10. DEPOSIT takten

Anschließend wird der Befehlszähler über LOAD-ADR. wieder in die Ausgangsstellung 4 0_{16} zurückgebracht. Die Aufgabenstellung hätte auch so gelöst werden können, daß die Register R0 bis R3 zuerst über die Adresse F F_{16} mit den A-Schaltern geladen worden wären. Dies ist natürlich nur im SINGLE-STEP-Betrieb möglich. Der Vorgang ist folgender:

1. Über LOAD-ADR. eine Adresse auswählen, z.B. 8 0_{16}
2. LOAD-Befehl an A_7 bis A_0 einstellen, z.B. 1 0 0 0 0 1 0 0 (LOAD R0, F F_{16}), DEPOSIT takten
3. In Programmadresse 8 1_{16} über DEPOSIT die INP-Adresse F F_{16} abspeichern
4. LOAD-Befehl für R1 einstellen (1 0 0 0 0 1 0 1) und DEPOSIT takten usw. bis Befehl LOAD R3, F F_{16}
5. Über LOAD-ADR. auf Ausgangsadresse zurückstellen
6. Daten für R0 an A_7 bis A_0 einstellen
7. C_4 auf SINGLE-STEP-Betrieb schalten
8. RUN **einmal** takten
9. Daten für R1 an A_7 bis A_0 einstellen
10. RUN **einmal** takten usw. bis R3

Alle Befehle der 1. Befehlsgruppe fühen bekanntlich Operationen zwischen 2 Registern bzw. Akkumulatoren durch. In der Befehlsliste sind diese mit einem R am Ende gekennzeichnet, z.B. ADDR, IORR, XORR usw. Bei Benutzung dieser Befehle ist besonders darauf zu achten, daß in der mnemonischen Befehlsschreibweise der Datenfluß von rechts nach links, im Maschinencode die Programmierung aber in Richtung SRC nach DST erfolgt.

Beispiel:
Der Befehl ADDR R0, R1 besagt, daß in R0 als DST-Register eine Addition mit dem Inhalt R1 des SRC-Registers erfolgen soll. Der Maschinencode lautet hierfür:



Wir weisen an dieser Stelle auf diesen Punkt noch einmal ausdrücklich hin, da hier besonders häufig Programmfehler gemacht werden.
Die Befehle INCR und DECR erhöhen bzw. erniedrigen den Inhalt eines Registers mit jedem Takt um 1. Wenn wir z.B. in Adresse 1 9_{16} einen INCR-Befehl programmieren und die restlichen Speicherplätze von 1 A_{16} bis F E_{16} mit dem NOP-Befehl belegen, haben wir einen langsam zählenden Vorwärtszähler programmiert.

Experiment 6b

Bei den Befehlen der 2. Befehlsgruppe gibt es grundsätzlich 2 Möglichkeiten:
- Direkte Adressierung, z.B. LOAD R2, 9 3
- Immediate Adressierung, z.B. ADDM R2, # 7 5

Bei der direkten Adressierung steht im 2. Byte die Adresse, deren Inhalt als Daten benutzt werden. Der Befehl LOAD R2, 9 3 besagt, daß die Daten der Adresse $9\ 3_{16}$ in Register R2 zu laden sind.
Bei der immediate Adressierung stehen die Daten direkt im 2. Byte. Der Befehl SUBM R2, # 7 5 besagt, daß ~~zum~~ vom Inhalt R2 die Zahl $7\ 5_{16}$ ~~addiert~~ subtrahiert werden soll. Hierbei steht die Zahl $7\ 5_{16}$ im zweiten Byte.

Anmerkung:
Zahlenangaben wie 7 5, A F usw. werden ab jetzt immer hexadezimal verstanden. Ausnahmen im Dezimalsystem werden durch den Index gekennzeichnet.

Wir wollen auch hierfür wieder ein kleines Programmbeispiel durchsprechen.

2. Beispiel:
Die Befehlsliste (Tab. 4) ist zu ergänzen und dann zu programmieren (Lösung Tab. 5).

Adresse (hexadez)	Inhalt Maschinencode	Befehl	Kommentar
4 0		LOAD R0, 8 0	
4 1			
4 2		IORM R0, # 1 7	
4 3			
4 4		ADDM R0, 8 1	
4 5			
4 6		SUBM R0, # 0 4	
4 7			
4 8		XORM R0, # 2 0	
4 9			
4 A		HLT	
.			
.			
.			
.			
8 0	1 0		
8 1	1 1		

Tab. 4
Befehlsliste zum 2. Beispiel

Adresse	Inhalt	Befehl	Kommentar
4 0	1 0 0 0 0 1 0 0	LOAD R0, 8 0	direkte Adressierung
4 1	1 0 0 0 0 0 0 0		Datenadresse
4 2	1 0 1 1 0 0 0 0	IORM R0, # 1 7	immediate Adressierung
4 3	0 0 0 1 0 1 1 1		Daten
4 4	1 0 0 1 0 1 0 0	ADDM R0, 8 1	
4 5	1 0 0 0 0 0 0 1		
4 6	1 0 1 0 0 0 0 0	SUBM R0, # 0 4	
4 7	0 0 0 0 0 1 0 0		
4 8	1 1 0 0 0 0 0 0	XORM R0, # 2 0	
4 9	0 0 1 0 0 0 0 0		
4 A	0 0 0 0 0 0 0 0	HLT	
.			
.			
.			
8 0	0 0 0 1 0 0 0 0		Daten
8 1	0 0 0 1 0 0 0 1		

Tab. 5
Lösung zum 2 Beispiel

Als Endergebnis muß 0 4 in R0 erscheinen. Wenn Sie dieses Programm im SINGLE-STEP-Betrieb abarbeiten, werden Sie feststellen, daß der Befehlszähler immer eine Programmadresse überspringt, da in den Adressen 4 1, 4 3, 4 5, 4 7 und 4 9 keine Befehle, sondern Datenadressen oder Daten abgespeichert sind.

Experiment 6c

Ein wesentlicher Unterschied zum einfachen Rechner in Experiment 5 ist der, daß der hypothetische Mikrorechner Sprungbefehle enthält. Bei den Sprungbefehlen muß grundsätzlich nach
– direkter Adressierung (MODE 0 0) und
– indirekter Adressierung (MODE 0 1)
unterschieden werden.
Bei direkter Adressierung springt der Befehlszähler auf die im 2. Byte spezifizierte Adresse.

Bei indiekter Adressierung dient das 2. Byte als Adresse mit **deren Inhalt** der Befehlszähler geladen wird. Der Befehl JUMP 8 1 bedeutet, daß der Befehlszähler auf die Adresse 8 1 springen soll. Er hat den Maschinencode:

1	1	1	0	0	0	0	0
OP-Code				MODE		Bedingung	

1	0	0	0	0	0	0	1
Sprungadresse							

Der Befehl JUMP @ 8 4 bedeutet, daß der Befehlszähler auf eine Adresse springt, die vom Inhalt der Adresse 8 4 bestimmt wird. Er hat den Maschinencode:

OP-Code				MODE		Bedingung	
1	1	1	0	0	1	0	0

1	0	0	0	0	1	0	0
Adresse, deren Inhalt die Sprungadresse angibt							

In beiden Fällen handelt es sich um unbedingte Sprungbefehle, festgelegt durch die Bedingungbit 0 0. Um bedingte Sprungbefehle handelt es sich bei den Befehlen JMPZ, JMPN und JMPC.

3. Beispiel
Der Rechner ist mit dem in Tab. 6 angegebenen Programm zu laden.

Überprüfen Sie anhand von EXAMINE, ob Ihnen bei der Programmeingabe kein Fehler unterlaufen ist. Wenn alles richtig eingegeben wurde, können Sie mit diesem Programm die Wirkungsweise der Sprungbefehle eindeutig erkennen.
Gehen Sie dabei folgendermaßen vor:

1. Über LOAD-ADR. Befehlszähler auf Ausgangsadresse 4 0 stellen
2. *A*-Schalter auf Null
3. System auf SINGLE-STEP-Betrieb stellen
4. B_2 bis B_0 auf 1 0 0 stellen (R_7 bis R_0 zeigt Stellung des Befehlszählers an)
5. B_5 bis B_3 auf 0 0 0 stellen (L_7 bis L_0 zeigt Inhalt Register R0)
6. Über RUN System Schritt für Schritt takten

Da alle *A*-Schalter auf 0 sind, entsteht bei dem Befehl SUBM R0, 7 1 in Adresse 4 4 ein negatives Ergebnis. Dies bedeutet, daß jetzt der Befehl JUMPN @ 7 2 ausgelöst wird. Der Befehlszähler springt auf Adresse 4 E. In 4 E steht der Befehl LOAD R0, # 0 F, d.h., R0 wird mit 0 F geladen. Im nächsten Befehl wird dann die XOR-Verknüpfung gebildet. Dann folgt der Befehl JUMP 4 0, d.h., der Befehlszähler wird auf die Ausgangsadresse 4 0 zurückgestellt, und der Vorgang beginnt von neuem.
Jetzt wird, wie vorher beschrieben, das Ganze mit einer Zahl an den *A*-Schaltern größer 2 0 (z.B. 2 2) wiederholt. Da jetzt bei dem Befehl SUBM R0, 7 1 kein negatives Ergebnis entsteht, erfolgt an dieser Stelle kein Sprung. Über Befehl LOAD R0, F F wird R 0 mit 2 2

Adresse	Inhalt	Befehl	Kommentar
4 0	1 0 0 0 0 1 0 0	84 LOAD R0, 7 0	
4 1	0 1 1 1 0 0 0 0	70	
4 2	1 0 0 1 0 1 0 0	94 ADDM R0, F F	Daten der *A*-Schalter
4 3	1 1 1 1 1 1 1 1	FF	
4 4	1 0 1 0 0 1 0 0	A4 SUBM R0, 7 1	
4 5	0 1 1 1 0 0 0 1	71	
4 6	1 1 1 0 0 1 1 0	E6 JMPN @ 7 2	bedingter Sprung (indirekt)
4 7	0 1 1 1 0 0 1 0	72	
4 8	1 0 0 0 0 1 0 0	84 LOAD R0, F F	
4 9	1 1 1 1 1 1 1 1	FF	
4 A	1 0 1 0 0 0 0 0	A0 SUBM R0, # 2 0	
4 B	0 0 1 0 0 0 0 0	20	
4 C	1 1 1 0 0 1 0 1	E5 JMPZ @7 3	bedingter Sprung (indirekt)
4 D	0 1 1 1 0 0 1 1	73	
4 E	1 0 0 0 0 0 0 0	80 LOAD R0, # 0 F	
4 F	0 0 0 0 1 1 1 1	0F	
5 0	1 1 0 0 0 0 0 0	C0 XORM R0, # 1 8	
5 1	0 0 0 1 1 0 0 0	18	
5 2	1 1 1 0 0 0 0 0	E0 JUMP 4 0	unbedingter Sprung
5 3	0 1 0 0 0 0 0 0	40	
.			
.			
7 0	0 0 1 0 0 0 0 0	20	
7 1	0 1 0 0 0 0 0 0	40	Daten
7 2	0 1 0 0 1 1 1 0	4E	
7 3	0 1 1 1 0 1 0 1	75	
.			
7 5	0 1 1 0 0 0 0 0	60 INCR R0	
7 6	0 0 0 0 1 1 1 1	0F NOP	
7 7	0 0 0 0 1 1 1 1	0F NOP	
7 8	0 0 0 0 1 1 1 1	0F NOP	
7 9	0 0 0 0 1 1 1 1	0F NOP	
7 A	1 0 0 1 0 1 0 0	94 ADDM R0, F F	Daten der *A*-Schalter
7 B	1 1 1 1 1 1 1 1	FF	
7 C	1 1 1 0 0 0 1 1	E3 JMPC 4 0	bedingter Sprung
7 D	0 1 0 0 0 0 0 0	40	
7 E	1 1 1 0 0 0 0 0	E0 JUMP 7 5	unbedingter Sprung
7 F	0 1 1 1 0 1 0 1	75	

Tab. 6
Programm zum 3. Beispiel

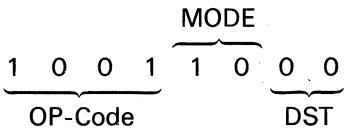
geladen. Davon wird dann 2 0 abgezogen. Da der Befehl JMPZ @ 7 3 nur ausgelöst wird, wenn bei dieser Subtraktion das Ergebnis 0 entsteht, erfolgt hier kein Sprung, das Programm läuft bis Befehl JUMP 4 0 durch und springt dann auf die Ausgangsadresse 4 0 zurück. Im nächsten Beispiel stellen wir die *A*-Schalter auf 2 0 ein. Jetzt wird der Befehl JMPZ @7 3 wirksam. Der Befehlszähler springt auf Adresse 7 5 und führt hier einen INCR-Befehl aus. Danach folgen 4 NOPs. In Adresse 7 A wird der Inhalt R 0 über den Befehl ADDM R0, F F mit den Daten des *A*-Schalters addiert, in unserem Beispiel mit 2 0. In Register R0 steht danach 2 1. Der bedingte Sprungbefehl wird nicht ausgelöst, da noch kein Übertrag vorhanden ist. Bei Adresse 7 E erfolgt ein unbedingter Sprung zur Adresse 7 5 zurück. Hier erfolgt wieder INCR, 4mal NOP usw., bis nach einigen Schleifen ein Übertrag entsteht, der den Befehl JMPC 4 0 auslöst. Jetzt springt der Befehlszähler auf Adresse 4 0 zurück, und der ganze Vorgang läuft von neuem ab.
Der DISPLAY-SELEKTOR gestattet es, alle Einzelschritte genau zu verfolgen. So können Sie auch z.B. durch B_2 bis B_0 gleich 1 0 1 den Zustand der Flags in R_7 bis R_0 anzeigen lassen und somit feststellen, wann ein bedingter Sprung erfolgen muß.

Experiment 6d

Im nächsten Beispiel befassen wir uns mit der **Indexed-Adressierung.** Als Indexregister wird über MODE 1 0 Akku R2 betrieben. Ein Additionsbefehl dieser Adressierungsart in mnemonischer Schreibweise hat die Form:

ADDM R0, @ R2

Der Maschinencode hierfür lautet:



Zuerst programmieren wir das im theoretischen Teil behandelte Beispiel.
Ab jetzt geben wir den Inhalt der Adresse nicht mehr in binärer Schreibweise an, sondern wir verwenden auch hierfür die Hexadezimal-Schreibweise.

4. Beispiel (Tab. 7):

Adresse	Inhalt	Befehl	Kommentar
4 0	8 2	LOAD R2, # 9 0	Adresse der ersten Daten
4 1	9 0		
4 2	8 1	LOAD R1, # 4 1	Anzahl der Datenwörter
4 3	4 1		
4 4	2 0	SUBR R0, R0	DST löschen
4 5	9 8	ADDM R0, @ R2	
4 6	9 2	ADDM R2, # 1	erhöhe Datenadresse
4 7	0 1		
4 8	A 1	SUBM R1, # 1	erniedrige Schleifenzähler
4 9	0 1		
4 A	E 1	JMPZ 4 E	aufhören bei R1 = 0
4 B	4 E		
4 C	E 0	JUMP 4 5	sonst wiederholen
4 D	4 5		
4 E	0 0	HALT	
.			
.			
.			
9 0	0 4		
9 1	0 4		
9 2	0 4		
9 3	0 4		
9 4	0 4		
9 5	0 4		
9 6	0 4		
9 7	0 4		
9 8	0 4		
9 9	0 4		
9 A	0 4		
9 B	0 4		
9 C	0 4		
9 D	0 4		
9 E	0 4		
9 F	0 4		
A 0	0 3		
A 1	0 3		

Adresse	Inhalt	Befehl	Kommentar
A 2	0 3		
A 3	0 3		
A 4	0 3		
A 5	0 3		
A 6	0 3		
A 7	0 3		
A 8	0 3		
A 9	0 3		
A A	0 3		
A B	0 3		
A C	0 3		
A D	0 3		
A E	0 3		
A F	0 3		
B 0	0 2		
B 1	0 2		
B 2	0 2		
B 3	0 2		
B 4	0 2		
B 5	0 2		
B 6	0 2		
B 7	0 2		
B 8	0 2		
B 9	0 2		
B A	0 2		
B B	0 2		
B C	0 2		
B D	0 2		
B E	0 2		
B F	0 2		
C 0	0 5		
C 1	0 5		
C 2	0 5		
C 3	0 5		
C 4	0 5		
C 5	0 5		
C 6	0 5		
C 7	0 5		
C 8	0 5		
C 9	0 5		
C A	0 5		
C B	0 5		
C C	0 5		
C D	0 5		
C E	0 5		
C F	0 5		
D O	0 5		

Tab. 7
Programm zum 4. Beispiel

Nachdem Sie Programm und Daten geladen haben, wird zuerst über LOAD-ADR. der Befehlszähler auf die Anfangsadresse 4 0 zurückgestellt. Lassen Sie dann das Programm über RUN automatisch ablaufen. Der Rechner benötigt ca. 2 Sekunden bis das Ergebnis E 5 in R0 gebildet ist. Verfolgen Sie auch über SINGLE-STEP-Betrieb die Arbeitsweise von Indexregister R2 und Schleifenzähler R1.
Entscheidend ist, daß Sie die Funktion des Befehles ADDM R0, @ R2 genau erkennen. Es werden jeweils die Daten zum Inhalt von R0 addiert, die unter der Adresse des Indexregisters R2 abgespeichert sind. R1 dient als Zähler für die abzuarbeitenden Datenwörter.

Experiment 6e

5. Beispiel:
Dieses Beispiel zeigt die **Auto-Increment-Indexed-Adressierung.** Die Aufgabenstellung entspricht der im 4. Beispiel. Es werden die gleichen Daten wie im 4. Beispiel verwendet. Daher ist nur der Programmteil neu zu laden (Tab. 8).

Adresse	Inhalt	Befehl	Kommentar
4 0	8 3	LOAD R3, # 9 0	Adresse der ersten Daten
4 1	9 0		
4 2	8 1	LOAD R1, # 4 1	Anzahl der Datenwörter
4 3	4 1		
4 4	2 0	SUBR R0, R0	lösche Inhalt im DST R0
4 5	9 C	ADDM R0, @ R3↑	
4 6	A 1	SUBM R1, # 1	erniedrige Schleifenzähler
4 7	0 1		
4 8	E 1	JMPZ 4 C	aufhören, falls R1 = 0
4 9	4 C		
4 A	E 0	JUMP 4 5	sonst wiederholen
4 B	4 5		
4 C	0 0	HALT	
.			
.			
.			
.			
9 0	0 4	}	
.	.		
.	.		
.	.	} Daten	
.	.		
.	.		
D 0	0 5	}	

Tab. 8
Programm zum 5. Beispiel

Lassen Sie das Programm automatisch ablaufen. In R0 erscheint als Ergebnis wieder E 5. Laden Sie wieder Adresse 4 0, und betreiben Sie das System im SINGLE-STEP-Betrieb. Stellen Sie den DISPLAY-SELEKTOR so ein, daß Sie die Zustände von R3 und R0 verfolgen können.
Bei dem Befehl ADDM R0, @ R3↑ in Adresse 4 5 wird der Inhalt R3 jeweils um 1 erhöht, gleichzeitig erfolgt in R0 eine Addition mit den Daten, die in der von R3 spezifizierten Adresse abgespeichert sind. Gegenüber dem 4. Beispiel entfällt der Befehl ADDM R2, # 1.

6. Beispiel:
In diesem Beispiel geht es darum, mit Hilfe der Auto-Increment-Indexed-Adressierung den Inhalt des Speichers auf ein bestimmtes Datenwort bzw. bit-Muster hin zu untersuchen. Das bit-Muster wird über die *B*-Schalter in Adresse F F eingegeben.
Laden Sie das System mit dem in Tab. 9 gezeigten Programm.
Stellen Sie den DISPLAY-SELEKTOR so ein, daß Sie den Zustand von R1 und R3 verfolgen können. Die Anfangsadresse 4 0 laden. Die *B*-Schalter auf F F einstellen, und Programm automatisch ablaufen lassen. Das System hält spätestens an, wenn R3 den Inhalt 4 4 anzeigt. Das System hält früher an, wenn zufällig in einer vorherigen Adresse bereits der Inhalt F F enthalten ist. Wie läßt sich dieses Verhalten erklären?
Mit dem Befehl SUBR R3, R3 wird das Autoindexregister R3 auf die Adresse 0 0 gesetzt. Beim nächsten Befehl LOAD R1, @ R3↑ wird R1 mit **dem Inhalt** der Adresse 0 0 geladen und gleichzeitig R1 auf Adresse 0 1 gesetzt. Vom Inhalt R1 wird mit dem Befehl SUBM R1, F F

Adresse	Inhalt	Befehl	Kommentar
4 0	2 F	SUBR R3, R3	lösche R3
4 1	8 D	LOAD R1, @ R3↑	
4 2	A 5	SUBM R1, F F	
4 3	F F		
4 4	E 1	JMPZ 4 8	
4 5	4 8		
4 6	E 0	JUMP 4 1	
4 7	4 1		
4 8	0 0	HALT	

Tab. 9
Programm zum 6. Beispiel

der an B_7 bis B_0 stehende Wert subtrahiert. Ist das Ergebnis ungleich 0, hat der Befehl JMPZ 4 8 keine Auswirkung, und über JUMP 4 1 erfolgt ein erneuter Befehl LOAD R1, @ R3↑. Hierbei wird jetzt **der Inhalt** von Adresse 0 1 in R1 geladen und R3 auf 0 2 erhöht usw. Dieser Vorgang wiederholt sich so oft, bis spätestens bei R3 = 4 3 R1 mit F F geladen und gleichzeitig R3 auf 4 4 erhöht wird. Der nächste Befehl SUBM R1, F F bewirkt, daß der Inhalt B_7 bis B_0 von R1 subtrahiert wird. Damit entsteht in R1 das Ergebnis 0. Der Befehl JMPZ 4 8 bewirkt, daß ein Sprung in Adresse 4 8 mit dem HALT-Befehl erfolgt.
Stellen Sie selbst beliebige bit-Muster an B_7 bis B_0 ein, und lassen Sie dann das Programm ablaufen.

Experiment 6f

7. Beispiel:
In diesem Beispiel (Tab. 10) sollen die verschiedenen Adressierungsmöglichkeiten der STAC-Befehle erläutert werden (außer Auto-Decrement-Indexed-Adressierung).

Adresse	Inhalt	Befehl	Kommentar
4 0	8 0	LOAD R0, # 4 0	
4 1	4 0		
4 2	7 4	STAC R0, 9 0	
4 3	9 0		
4 4	8 2	LOAD R2, # 9 1	
4 5	9 1		
4 6	8 0	LOAD R0, # 1 0	
4 7	1 0		
4 8	7 8	STAC R0, @ R2	
4 9	8 3	LOAD R3, # 9 2	
4 A	9 2		
4 B	8 2	LOAD R2, # 2 0	
4 C	2 0		
4 D	7 E	STAC R2, @ R3↑	
4 E	7 E	STAC R2, @ R3↑	
4 F	7 E	STAC R2, @ R3↑	
5 0	0 0	HALT	

Tab. 10
Programm zum 7. Beispiel

Geben Sie das Programm ein, und takten Sie das Programm manuell durch. Laden Sie dann Adresse 9 0. Mit EXAMINE ist der Inhalt der Adresse 9 0 bis 9 4 zu überprüfen. Folgender Inhalt muß vorhanden sein:

9 0 → 4 0
9 1 → 1 0
9 2 → 2 0
9 3 → 2 0
9 4 → 2 0

Aus diesem Beispiel geht die Funktion der Indexed- sowie der Auto-Increment-Indexed-Adressierung hervor.

8. Beispiel:
Mit diesem Beispiel wird die Funktion der Auto-Decrement-Indexed-Adressierung erläutert (Tab. 11).
Laden Sie über LOAD-ADR. Adresse 8 E, und überprüfen Sie mit EXAMINE den Inhalt der Adressen 8 E, 8 F und 9 0.
Resultat:

8 E → 2 0
8 F → 2 0
9 0 → 2 0

Dieses Beispiel zeigt, daß im Gegensatz zur Auto-Increment-Indexed-Adressierung zuerst R3 decrementiert wird, bevor der eigentliche Befehl durchgeführt wird. Damit wird der Inhalt von R0 beim ersten STAC-Befehl in Programmadresse 4 4 in Adresse 9 0 gespeichert. Mit den beiden nächsten STAC-Befehlen wird der Inhalt von R0 in 8 F und dann in 8 E gespeichert.

Adresse	Inhalt	Befehl	Kommentar
4 0	8 0	LOAD R0, # 2 0	
4 1	2 0		
4 2	8 3	LOAD R3, # 91	
4 3	9 1		
4 4	7 0	STAC R0, @↓R3	
4 5	7 0	STAC R0, @↓R3	
4 6	7 0	STAC R0, @↓R3	
4 7	0 0	HALT	

Tab. 11
Programm zum 8. Beispiel

Experiment 6g

9. Beispiel:
Dieses Beispiel zeigt die Funktion der Befehle INCR, DECR, RACL und RACR (Tab. 12).

Adresse	Inhalt	Befehl	Kommentar
4 0	8 0	LOAD R0, # 0 1	
4 1	0 1		
4 2	6 0	INCR R0	
4 3	6 0	INCR R0	
4 4	6 0	INCR R0	
4 5	6 0	INCR R0	
4 6	6 4	DECR R0	
4 7	6 4	DECR R0	
4 8	6 4	DECR R0	
4 9	6 8	RACL R0	
4 A	6 8	RACL R0	
4 B	6 8	RACL R0	
4 C	6 8	RACL R0	
4 D	6 8	RACL R0	
4 E	6 8	RACL R0	
4 F	6 8	RACL R0	Carry-Flag überprüfen
5 0	6 8	RACL R0	
5 1	6 8	RACL R0	
5 2	6 8	RACL R0	
5 3	6 C	RACR R0	
5 4	6 C	RACR R0	
5 5	6 C	RACR R0	Carry-Flag überprüfen
5 6	6 C	RACR R0	
5 7	6 C	RACR R0	
5 8	6 C	RACR R0	
5 9	E 0	JUMP 4 0	
5 A	4 0		

Tab. 12
Programm zum 9. Beispiel

Laden Sie Adresse 4 0, und takten Sie das Programm manuell durch. Überprüfen Sie bei den Programmadressen 4 F und 5 5 den Zustand des Carry-Flags.

10. Beispiel:
In diesem Beispiel wird gezeigt, wie mit Hilfe des STAC-Befehles über Auto-Increment-Indexed-Adressierung ein bestimmter Speicherbereich gelöscht werden kann. In diesem Beispiel soll der Speicherbereich 8 0 bis B F gelöscht werden, d.h., es müssen $4\ 0_{16}$ Speicheradressen gelöscht werden (Tab. 13).
Überprüfen Sie über EXAMINE, ob die Adressen 8 0 bis B F tatsächlich den Inhalt 0 0 aufweisen. Über das Programm selbst ist nicht sehr viel zu sagen. Bei Adresse 4 5 wird R 1 auf 0 0 gebracht. Dieser Inhalt wird dann über den Befehl STAC R1, @R3↑ bei Adresse 8 0 beginnend so lange in die darauffolgenden Speicheradressen geladen, bis R2 auf 0 0 angekommen ist. Dann erfolgt der Sprung nach Adresse 4 C mit dem HALT-Befehl.

Adresse	Inhalt	Befehl	Kommentar
4 0	8 3	LOAD R3, # 8 0	Anfang Speicherbereich
4 1	8 0		
4 2	8 2	LOAD R2, # 4 0	Anzahl der Wörter
4 3	4 0		
4 4	2 5	SUBR R1, R1	R1 löschen
4 5	7 D	STAC R1, @ R3↑	
4 6	A 2	SUBM R2, # 0 1	
4 7	0 1		
4 8	E 1	JMPZ 4 C	
4 9	4 C		
4 A	E 0	JUMP 4 5	
4 B	4 5		
4 C	0 0	HALT	

Tab. 13
Programm zum 10. Beispiel

Experiment 6h

11. Beispiel:
Hier soll gezeigt werden, wie man mit einem 8-bit-Mikrorechner Zahlen mit mehr als 8 bit verarbeiten kann. So sollen z.B. zwei 16-bit-Zahlen miteinander addiert werden oder in 2 Hälften in 2 Registern stehen. Wir arbeiten mit den Registern R0 bis R3 und belegen diese, wie Bild 1 zeigt.

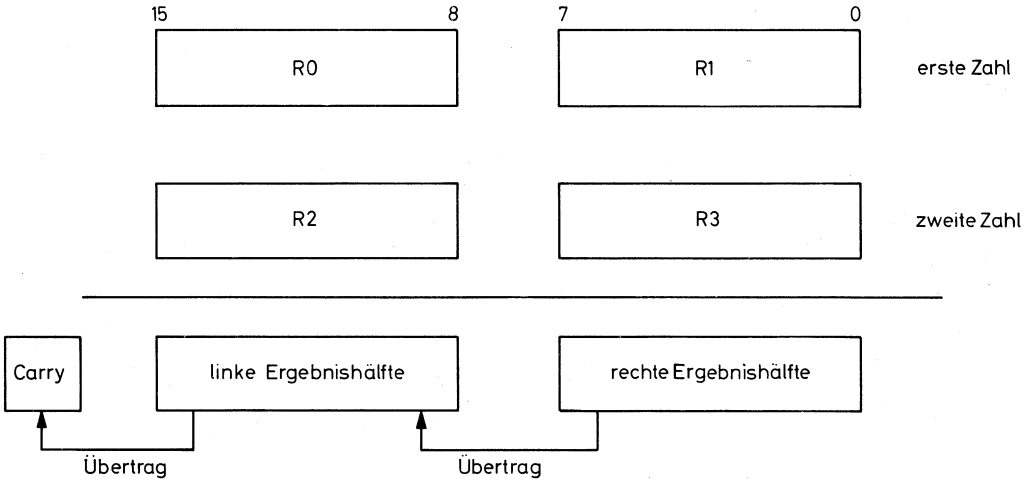


Bild 1
Addition zweier 16-bit-Zahlen mit einem 8-bit-Mikrorechner

Es gibt 2 Möglichkeiten, diese Addition durchzuführen:
a) Es werden zuerst die beiden linken Zahlenhälften addiert und dann die beiden rechten.
b) Es werden zuerst die beiden rechten Zahlenhälften addiert und dann die linken.

Möglichkeit a:
Laden Sie zuerst die einzelnen Register, wie Tab. 14 zeigt.

Adresse	Inhalt	Befehl	Kommentar
3 8	8 0	LOAD R0, # 1 0	
3 9	1 0		
3 A	8 1	LOAD R1, # 9 2	
3 B	9 2		
3 C	8 2	LOAD R2, # 7 2	
3 D	7 2		
3 E	8 3	LOAD R3, # 1 2	
3 F	1 2		

Tab. 14
Laden der einzelnen Register

Laden Sie nun das Programm für die Addition (Tab. 15).
Laden Sie jetzt Adresse 3 8, und takten Sie das System mit SINGLE-STEP. Wenn Sie die Daten und Befehle richtig eingegeben haben, erscheint das Ergebnis:

$$\underbrace{1\ 0\ 0\ 0\ 0\ 0\ 1\ 0}_{R2}\ \underbrace{1\ 0\ 1\ 0\ 0\ 1\ 0\ 0}_{R3}$$

Zu diesem Ergebnis kommen Sie auch, wenn Sie die genannten Zahlen von Hand addieren.
Laden Sie die Register wie folgt neu:

Adresse	Inhalt	Befehl	Kommentar
4 0	1 2	ADDR R2, R0	linke Hälfte addiert
4 1	1 7	ADDR R3, R1	rechte Hälfte addiert
4 2	E 3	JMPC 4 6	Übertrag?
4 3	4 6		
4 4	E 0	JUMP 4 7	
4 5	4 7		
4 6	6 2	INCR R2	
4 7	0 0	HALT	

Tab. 15
Programm für die Addition nach Möglichkeit a

R0 → 1 0
R1 → 9 2
R2 → 7 2
R3 → 8 7

Takten Sie jetzt das System bei Adresse 3 8 beginnend mit SINGLE-STEP. Als Ergebnis erhalten wir:

1 0 0 0 0 0 1 1 | 0 0 0 1 1 0 0 1
R2 | R3

In diesem Falle entsteht bei der Addition der Inhalte von R1 und R3 ein Übertrag. Dieser Übertrag wird dadurch berücksichtigt, daß der Inhalt von R2 durch den INCR-Befehl um 1 erhöht wird.

Möglichkeit b:
Laden Sie das in Tab. 16 gezeigte Programm.

Adresse	Inhalt	Befehl	Kommentar
4 0	1 7	ADDR R3, R1	rechte Hälfte addiert
4 1	6 9	RACL R1	hole Übertrag
4 2	D 1	ANDM R1, # 0 1	isoliere Übertrag
4 3	0 1		
4 4	1 6	ADDR R2, R1	Übertrag plus Inhalt R2
4 5	1 2	ADDR R2, R0	linke Hälfte addiert
4 6	0 0	HALT	

Tab. 16
Programm für die Addition nach Möglichkeit b

Laden Sie jetzt wieder die Register R0 bis R3 mit folgenden Daten:

R0 → 1 0
R1 → 9 2
R2 → 7 2
R3 → 8 7

Wenn Sie dieses Programm (beginnend bei Adresse 3 8) ablaufen lassen, erhalten Sie dasselbe Ergebnis wie bei der Methode a. Der Unterschied gegenüber der zuerst genannten Methode besteht darin, daß ein Übertrag anders verarbeitet wird. Entsteht bei der Addition der Inhalte von R1 und R3 ein Übertrag, so wird dieser durch den Befehl RACL R1 in die

rechte Stelle von R1 geschoben. Berücksichtigen Sie an dieser Stelle, daß die Befehle RACL und RACR den Inhalt vom Carry-Flag mit verschieben (siehe 9. Beispiel). Durch den Befehl ANDM R1, # 0 1 wird die rechte Stelle von R1 ausgeblendet. Ist, wie in unserem Beispiel, ein Übertrag vorhanden, erscheint in R1 das Ergebnis 0 1. Dieser Übertrag wird nun durch den Befehl ADDR R2, R1 in die linke Zahlenhälfte addiert und damit bei dem Befehl ADDR R2, R0 berücksichtigt. Als Ergebnis erhalten wir wieder:

$$\underbrace{1\ 0\ 0\ 0\ 0\ 0\ 1\ 1}_{R2}\ \underbrace{0\ 0\ 0\ 1\ 1\ 0\ 0\ 1}_{R3}$$

Beide hier aufgezeigten Möglichkeiten können ein falsches Ergebnis liefern. Dies hängt damit zusammen, daß beim hypothetischen Mikrorechner kein Befehl vorhanden ist, der es ermöglicht, Überträge aus der rechten und linken Zahlenhälfte **gleichzeitig** zu addieren. Wir wollen hierauf an dieser Stelle nicht weiter eingehen. Bei der Besprechung des 8080-Systems wird diese Problematik behandelt.

12. Beispiel:
In diesem Beispiel wollen wir eine Digitaluhr simulieren. Bedingt dadurch, daß der Mikrorechner mit einem Quarztaktgenerator betrieben wird, läuft das Mikrorechnersystem mit einer praktisch konstanten Geschwindigkeit. Diese Eigenschaft kann zur Zeitmessung herangezogen werden.
Die verschiedenen Befehle eines Mikroprozessors benötigen eine unterschiedliche Anzahl Taktperioden. Bei einem realen Mikrorechnersystem ist die Anzahl der erforderlichen Taktperioden pro Befehl theoretisch und auch durch Zählen leicht festzustellen. Durch die Simulation des hypothetischen Mikrorechners werden dagegen so viele Befehle des 8080-Mikroprozessors ausgeführt, daß ein Abzählen der Takte nicht mehr praktikabel ist. Die Befehlszeit kann hierbei am besten durch Experimentieren ermittelt werden.
Zuerst wollen wir eine bestimmte Wartezeit erzeugen. Hierfür kann das in Tab. 17 gezeigte Programm verwendet werden.

Adresse	Inhalt	Befehl	Kommentar
4 0	8 0	LOAD R0, # 6 0	
4 1	6 0		
4 2	A 0	SUBM R0, # 0 1	
4 3	0 1		
4 4	E 1	JMPZ 4 8	
4 5	4 8		
4 6	E 0	JUMP 4 2	
4 7	4 2		

Tab. 17
Programm zum Erzeugen einer Wartezeit

Bei diesem Programm wird die Zahl 6 0 in R0 geladen. In einer Schleife wird davon so oft 1 abgezogen, bis 0 erreicht ist. Ist das der Fall, wird über JMPZ 4 8 das Programm bei Adresse 4 8 fortgesetzt. Die so erzeugte Zeitverzögerung kann dadurch geändert werden, daß die Anfangszahl 6 0 geändert wird. Durch den Einbau von NOP-Befehlen läßt sich außerdem der Verzögerungsbereich vergrößern. Über den Befehl LOAD R0, F F kann eine mit Hilfe der *A*-Schalter einstellbare Verzögerungszeit eingestellt werden (Tab. 18).
Über die *A*-Schalter muß die Zeitverzögerung dieser Schleife auf 1 Sekunde eingestellt werden. Die erste Schleife muß in eine zweite Schleife eingebaut werden, die 60 Sekunden zählt. Durch eine dritte Schleife können Stunden, durch eine weitere Tage erzeugt werden. Zusätzlich muß die Möglichkeit bestehen, am Anfang des Experimentes die Tageszeit einzustellen. Tab. 19 zeigt das Gesamtprogramm.
Laden Sie dieses Programm sorgfältig und überprüfen Sie es mit EXAMINE. Die Zeitanzeige erfolgt in den Registern:

Adresse	Inhalt	Befehl	Kommentar
4 0	8 4	LOAD R0, FF	
4 1	F F		
4 2	A 0	SUBM R0, # 0 1	
4 3	0 1		
4 4	0 F	NOP	Füllwort
4 5	0 F	NOP	Füllwort
4 6	0 F	NOP	Füllwort
4 7	0 F	NOP	Füllwort
4 8	0 F	NOP	Füllwort
4 9	0 F	NOP	Füllwort
4 A	E 1	JMPZ 4 E	
4 B	4 E		
4 C	E 0	JUMP 4 2	
4 D	4 2		
4 E	0 0	HALT	

Tab. 18
Programm zum Erzeugen einer einstellbaren Verzögerungszeit

R1 (Sekunden)
R2 (Minuten)
R3 (Stunden)

Bevor Sie das Programm ablaufen lassen, einige Erläuterungen:
Die Befehle in den Adressen 4 0 bis 4 6 ermöglichen, am Anfang des Experimentes die Tageszeit in Stunden und Minuten einzustellen. Die Befehle in den Adressen 4 7 bis 5 4 erzeugen die Verzögerungszeit von 1 Sekunde. Die so gewonnenen Sekundentakte werden mit den Befehlen der Adressen 5 5 bis 5 C gezählt. In Adresse 5 5 werden die Sekunden gehalten. In Adresse 5 7 wird von der momentan vorhandenen Sekundenzahl 3 C_{16} Sekunden ≙ 60_{10} Sekunden = 1 Minute subtrahiert. Solange das Ergebnis nicht 0 ist, muß diese Schleife fortgesetzt werden. Damit die tatsächliche Sekundenzahl in R1 erhalten bleibt, wird über den Befehl ADDM R1, # 3 C in Adresse 5 B der vorher abgezogene Wert 3 C wieder zum Inhalt R1 addiert. Sind 60_{10} Sekunden abgelaufen, springt der Befehlszähler auf Adresse 5 F. In dem Programmblock 5 F bis 6 7 werden die Minuten ähnlich aufbereitet wie vorher die Sekunden. Die Stunden werden dann noch im letzten Programmteil behandelt. Die Uhr ist auf 24_{10} Stunden (≙18_{16} Stunden) ausgelegt. Eine Änderung auf 12_{10} Stundentakt ist möglich, indem Adresse 6 C auf 0 C_{16} ≙ 12_{10} geändert wird.
Bevor die Uhr nun in Betrieb genommen werden kann, muß sie zunächst kalibriert werden. Starten Sie das Programm bei Adresse 4 7. Stellen Sie den DISPLAY-SELEKTOR so ein, daß in R2 die Minuten und in R1 die Sekunden angezeigt werden. Mit den *A*-Schaltern muß jetzt über F F eine Zahl eingegeben werden, die bewirkt, daß R2 nach jeder Minute um 1 weiterzählt (Anmerkung: Bei unserem MP-System ist das die Stellung 1 0).
Nachdem die Uhr kalibriert ist, wird die momentane Tageszeit eingestellt. Verfahren Sie dabei wie folgt:

1. Adresse 4 0 laden
2. Stunde an *A*-Schaltern einstellen
3. System mit **RUN** automatisch takten. Der Rechner lädt bei Adresse 4 2.
RUN-Schalter bleibt auf 1
4. Minuten an *A*-Schalter einstellen
5. Mit EXAMINE System takten, damit der HALT-Befehl überschritten wird. Die Minuten erscheinen in R2, und der Rechner bleibt bei Adresse 4 6 stehen
6. *A*-Schalter wieder auf die bei der Kalibrierung ermittelte Zahl einstellen
7. Mit EXAMINE den HALT-Befehl überbrücken. Jetzt kann das Hauptprogramm ablaufen

Es ist zweckmäßig, daß Sie bei der Einstellung der Minuten die Zeit um 1 bis 2 Minuten früher einstellen. Ist die eingestellte Zeit erreicht, kann über kurzes Betätigen von EXAMINE

Adresse	Inhalt	Befehl	Kommentar
4 0	8 7	LOAD R3, F F	Stunden in R3
4 1	F F		
4 2	0 0	HALT	*A*-Schalter einstellen
4 3	8 6	LOAD R2, F F	Minuten in R2
4 4	F F		
4 5	2 5	SUBR R1, R1	Sekunden löschen
4 6	0 0	HALT	*A*-Schalter einstellen
4 7	8 4	LOAD R0, F F	
4 8	F F		
4 9	A 0	SUBM R0, # 0 1	
4 A	0 1		
4 B	0 F	NOP	} Füllwörter
4 C	0 F	NOP	
4 D	0 F	NOP	
4 E	0 F	NOP	
4 F	0 F	NOP	
5 0	0 F	NOP	
5 1	E 1	JMPZ 5 5	
5 2	5 5		
5 3	E 0	JUMP 4 9	
5 4	4 9		
5 5	9 1	ADDM R1, # 0 1	Sekunden incrementieren
5 6	0 1		
5 7	A 1	SUBM R1, # 3 C	3 C Sekunden = 1 Minute
5 8	3 C		
5 9	E 1	JMPZ 5 F	
5 A	5 F		
5 B	9 1	ADDM R1, # 3 C	Sekunden korrigieren
5 C	3 C		
5 D	E 0	JUMP 4 7	
5 E	4 7		
5 F	9 2	ADDM R2, # 0 1	Minuten incrementiren
6 0	0 1		
6 1	A 2	SUBM R2,# 3 C	3 C Minuten = 1 Stunde
6 2	3 C		
6 3	E 1	JMPZ 6 9	
6 4	6 9		
6 5	9 2	ADDM R2, # 3 C	Minuten korrigieren
6 6	3 C		
6 7	E 0	JUMP 4 7	
6 8	4 7		
6 9	9 3	ADDM R3, # 0 1	Stunden incrementieren
6 A	0 1		
6 B	A 3	SUBM R3, # 1 8	1 8 Stunden = 1 Tag
6 C	1 8		
6 D	E 1	JMPZ 4 7	
6 E	4 7		
6 F	9 3	ADDM R3, # 1 8	Stunden korrigieren
7 0	1 8		
7 1	E 0	JUMP 4 7	
7 2	4 7		

Tab. 19
Gesamtprogramm einer Digitaluhr

das Programm ablaufen. Falls die Uhr über längere Zeit betrachtet vor- oder nachläuft, kann mit der *A*-Eingabe eine Korrektur vorgenommen werden. Läuft die Uhr zu langsam, *A*-Eingabe erniedrigen.

Aufgaben:

1.a) Ergänzen Sie nachfolgendes Programm mit dem Inhalt (in hexadezimal)!

Adresse	Inhalt	Befehl
4 0		LOAD R0, F F
4 1		
4 2		MOVE R0, R1
4 3		ADDM R0, # 1 0
4 4		
4 5		INCR R0
4 6		STAC R0, 9 0
4 7		
4 8		LOAD R2, # 5 F
4 9		
4 A		SUBR R0, R0
4 B		DECR R0
4 C		XORM R0, # 8 0
4 D		
4 E		JMPZ @ R2
4 F		JUMP 4 B
5 0		
.		
.		
.		
5 F	6 0	
6 0		ADDM R0, 9 0
6 1		
6 2		HALT

b) Programmieren Sie dieses Programm, und geben Sie das Ergebnis in R0 bei folgenden Eingaben A_7 bis A_0 an:

1. F 1
2. 0 8
3. 4 3

2. Bei welcher Programmadresse weist R0 bereits das Endergebnis auf, ohne daß das ganze Programm durchlaufen wird? Begründen Sie Ihre Antwort!

3. Welcher Programmbereich darf bei der Programmierung des RAMs nicht benutzt werden?

4. Welche Besonderheit weist die Programmadresse F F auf?

5. Laden Sie zunächst folgendes Programm:

Adresse	Inhalt	Befehl
4 0	8 4	LOAD R0, F F
4 1	F F	
4 2	A 4	SUBM R0, F F
4 3	F F	
4 4	0 0	HALT

Betreiben Sie das System im SINGLE-STEP-Betrieb, und führen Sie dann folgende Subtraktionen durch:

a) 0 8 − 1 0 =
b) 4 3 − 2 8 =
c) 0 0 − F E =
d) 0 0 − 7 F =

Geben Sie außer den Ergebnissen auch an, welche Flags ansprechen. Begründen Sie auch hier Ihre Antwort!

6. Laden Sie zunächst folgendes Programm:

Adresse	Inhalt	Befehl
4 0	8 4	LOAD R0, F F
4 1	F F	
4 2	9 4	ADDM R0, F F
4 3	F F	
4 4	0 0	HALT

Betreiben Sie das System wie in Aufgabe 5. Führen Sie folgende Additionen durch:

a) 1 0 + 4 3 =
b) 4 5 + F 1 =
c) 6 0 + 3 F =

Bestimmen Sie die Summen, und begründen Sie das Verhalten der Flags!

7. Laden Sie folgendes Programm:

Adresse	Inhalt	Befehl
4 0		LOAD R0, F F
4 1		
4 2		INCR R0
4 3		STAC R0, 9 1
4 4		
4 5		XORM R0, # *x x*
4 6		
4 7		JMPZ 8 E
4 8		
4 9		JUMP 4 2
4 A		
.		
.		
.		
.		
8 E		LOAD R0, 9 1
8 F		
9 0		HALT

Wählen Sie die Daten *x x* für den Befehl XORM R0, # *x x* so, daß das System unabhängig von der Eingabe F F hält, wenn in R0 die Daten 1 A stehen!

8. Ändern Sie das Programm nach Aufgabe 7 so ab, daß die Schleifendurchläufe vom Register R1 gezählt werden!

Lösungen zu Experiment 6

1. a)

Adresse	Inhalt	Befehl
4 0	8 4	LOAD R0, F F
4 1	F F	
4 2	0 1	MOVE R1, R0
4 3	9 0	ADDM R0, # 1 0
4 4	1 0	
4 5	6 0	INCR R0
4 6	7 4	STAC R0, 9 0
4 7	9 0	
4 8	8 2	LOAD R2, # 5 F
4 9	5 F	
4 A	2 0	SUBR R0, R0
4 B	6 4	DECR R0
4 C	C 0	XORM R0, # 8 0
4 D	8 0	
4 E	E 9	JMPZ @ R2
4 F	E 0	JUMP 4 B
5 0	4 B	
.		
.		
.		
5 F	6 0	
6 0	9 4	ADDM R0, 9 0
6 1	9 0	
6 2	0 0	HALT

b) A_7 bis A_0 → F 1: R0 → 0 2
A_7 bis A_0 → 0 8: R0 → 1 9
A_7 bis A_0 → 4 3: R0 → 5 4

2. Das Endergebnis erscheint erstmalig bei der Programmadresse 4 5 mit dem Befehl INCR R0. Der nächste Befehl STAC R0, 9 0 speichert dieses Ergebnis in Adresse 9 0 ab. Im nachfolgenden Programmteil wird eine Schleife gebildet, die dann verlassen wird, wenn das Zero-Flag anspricht. Dies ist bei R0 = 0 der Fall. Mit R2 als Indexregister springt dann der Programmzähler auf die Adresse 6 0. Der hier gespeicherte Befehl addiert zum Registerinhalt R0 = 0 den Inhalt der Adresse 9 0. Damit muß am Programmschluß in R0 das gleiche Ergebnis erscheinen, wie dies bereits bei der Adresse 4 5 der Fall war.

3. Der Adressenbereich von 0 0 bis 1 8 darf nicht benutzt werden, da dieser für ein Unterprogramm benötigt wird.

4. Über die Adresse F F können die *A*-Schalter zur Dateneingabe benutzt werden.

5. a) 0 8 − 1 0 = F 8
In diesem Fall sprechen Negativ- und Carry-Flag an. Das Carry-Flag spricht an, da der Subtrahend größer als der Minuend ist, das Negativ-Flag, da das Ergebnis im negativen Zahlenbereich liegt.

b) 4 3 − 2 8 = 1 B
Hier spricht kein Flag an, da die Subtraktion im positiven Zahlenbereich stattfindet.

c) 0 0 − F E = 0 2
Hier spricht nur das Carry-Flag an. Die Begründung läßt sich am einfachsten über die rechnerische Lösung geben:

$$\begin{array}{r} 0\,0\,0\,0\,0\,0\,0\,0 \\ -\,1\,1\,1\,1\,1\,1\,1\,0 \\ \hline 1\,0\,0\,0\,0\,0\,0\,1\,0 \end{array}$$

Es entsteht eine Entlehnung, die als Übertrag gewertet wird. Der negative Zahlenbereich ist aber bereits überschritten.

d) 0 0 − 7 F = 8 1
Hier sprechen Carry- und Negativ-Flag an, da einmal eine Entlehnung entsteht und zum anderen das Ergebnis noch im negativen Zahlenbereich liegt.

6. a) 1 0 + 4 3 = 5 3
Es spricht kein Flag an, da weder ein Übertrag entsteht noch der positive Zahlenbereich verlassen wird.

b) 4 5 + F 1 = 3 6 mit Übertrag
Das Carry-Flag spricht an, da die 8-bit-Kapazität des Systems überschritten wird. Dezimal lautet das Ergebnis 310_{10}, mit 8 bit sind aber nur max. 255_{10} darstellbar.

c) 6 0 + 3 F = 9 F
Hier spricht das Negativ-Flag an, da der positive Zahlenbereich verlassen wird.

7.

Adresse	Inhalt	Befehl
4 0	8 4	LOAD R0, F F
4 1	F F	
4 2	6 0	INCR R0
4 3	7 4	STAC R0, 9 1
4 4	9 1	
4 5	C 0	XORM R0, # 1A
4 6	1 A	
4 7	E 1	JMPZ 8 E
4 8	8 E	
4 9	E 0	JUMP 4 2
4 A	4 2	
.		
.		
.		
.		
8 E	8 4	LOAD R0, 9 1
8 F	9 1	
9 0	0 0	HALT

Begründung:
Das Zero-Flag spricht an, wenn der Inhalt von R0 gleich Null ist. Wenn nun die XOR-Verknüpfung mit den Daten 1 A gebildet wird, muß beim Inhalt 1 A des Registers R0 das Zero-Flag ansprechen und so den Befehl JMPZ 8 E auslösen.

8.

Adresse	Inhalt	Befehl
4 0	2 5	SUBR R1, R1
4 1	8 4	LOAD R0, F F
4 2	F F	
4 3	6 0	INCR R0
4 4	7 4	STAC R0, 9 1
4 5	9 1	
4 6	6 1	INCR R1
4 7	C 0	XORM R0, # 1 A
4 8	1 A	
4 9	E 1	JMPZ 8 E
4 A	8 E	
4 B	E 0	JUMP 4 3
4 C	4 3	
.		
.		
.		
8 E	8 4	LOAD R0, 9 1
8 F	9 1	
9 0	0 0	HALT

Experiment 7: Programmschleifen

Dies und auch die nachfolgenden Experimente werden mit dem hypothetischen Rechner ausgeführt. Somit ist wie bei Experiment 6 das System 5 des MP-Experimenters zu verwenden.

1. Beispiel:
Es ist ein Programm zu erstellen, das die ersten *n* Glieder der Reihe 1, 2, 3, 4, 5 . . . addiert. Die Zahl *n*, also wie viele Glieder zu addieren sind, soll am Anfang des Experimentes über Adresse F F eingegeben werden.

Allgemeine Hinweise:
Zur Lösung dieser Aufgabenstellung wird eine laufende Summe benötigt. Bei einer Addition von z.B. $n = 10$ Gliedern ist bei dieser Reihe das höchste Glied der Reihe auch 10, d.h., eine Lösungsmöglichkeit dieser Aufgabe besteht darin, über einen Schleifenzähler (z.B. R2) den Inhalt des Schleifenzählers zum Registerinhalt der laufenden Summe (z.B. R1) zu addieren. Bei Programmbeginn muß dann der Inhalt des Registers R1 gleich 0 sein. Wenn $n = 10$ Schleifen durchlaufen sind, ist das Programm beendet.

Bevor das Programm geschrieben wird, erstellen wir zunächst ein Flußdiagramm (Bild 1).

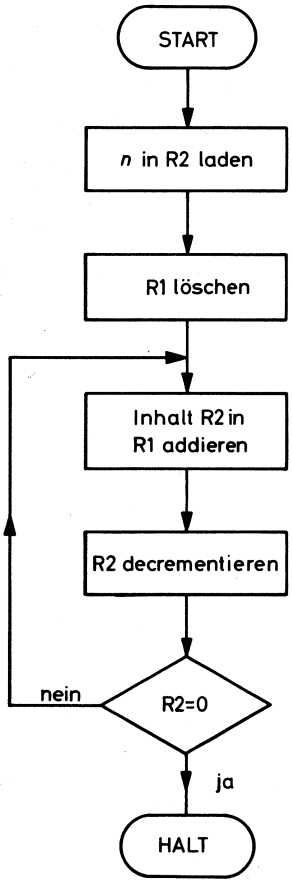


Bild 1
Flußdiagramm zum 1. Beispiel

Dieses Flußdiagramm läßt sich mit dem in Tab. 1 gezeigten Programm realisieren.

Laden Sie dieses Programm und geben Sie für folgende Werte von *n* das Ergebnis an:

a) $n = 5_{10}$
b) $n = 10_{10}$
c) $n = 22_{10}$

Adresse	Inhalt	Befehl	Kommentar
4 0	8 6	LOAD R2, F F	*n* in Schleifenzähler laden
4 1	F F		
4 2	2 5	SUBR R1, R1	R1 löschen
4 3	1 9	ADDR R1, R2	(R2) in R1 addieren
4 4	6 6	DECR R2	
4 5	E 1	JMPZ 4 9	
4 6	4 9		
4 7	E 0	JUMP 4 3	
4 8	4 3		
4 9	0 0	HALT	

Tab. 1
Programm zum 1. Beispiel

Bei richtiger Programmeingabe erhalten Sie folgende Ergebnisse:

a) $0\ F_{16} \triangleq 15_{10}$
b) $3\ 7_{16} \triangleq 55_{10}$
c) $F\ D_{16} \triangleq 253_{10}$

Aufgrund der Rechnerkapazität darf hier *n* nicht größer als $22_{10} \triangleq 1\ 6_{16}$ gewählt werden.

2. Beispiel:
Hier soll ein Programm für das Ziehen der Quadratwurzel erstellt werden. Auch dieses Problem läßt sich über Programmschleifen lösen.

Allgemeine Hinweise:
Die Summe der Reihe

$$1 + 3 + 5 + 7 + 9 + \ldots\ldots$$

ist gleich n^2. Hierbei ist *n* die Anzahl der Glieder. Bei z.B. $n = 5$ ist das Ergebnis $5^2 = 25$. Diese Gesetzmäßigkeit kann zum Ziehen der Quadratwurzel ausgenutzt werden. Soll die Quadratwurzel einer Zahl Z gezogen werden, so muß von dieser Zahl zunächst 1, dann 3, dann 5 usw. abgezogen werden, bis 0 erreicht wird. Die Anzahl der abgezogenen Zahlen (gleich Anzahl der Schleifendurchläufe) ist gleich der Quadratwurzel von Z. Wir beschränken uns bei diesem Beispiel auf die Fälle, bei denen Z eine Quadratwurzel ist, also 1, 4, 9, 16, 25, 36 . . . . usw.
Zur Lösung der Aufgabenstellung wird wieder ein Schleifenzähler benötigt, der bei 0 anfängt. Die Anzahl der Schleifendurchläufe entspricht dem Ergebnis. Die einzelnen Glieder, die von Z subtrahiert werden müssen, lassen sich durch Addition von jeweils 2 zum vorherigen Glied erzeugen. Außerdem wird ein Register benötigt, in dem zu Anfang Z enthalten ist. Mit jedem Schleifendurchlauf muß hiervon das jeweilige Glied abgezogen werden.

Zunächst soll diese Problemstellung wieder in einem Flußdiagramm festgehalten werden. Hierbei werden R1 als Schleifenzähler, R0 als Register zur Erzeugung der einzelnen Glieder und R2 als Subtraktionsregister benutzt (Bild 2).
Dieses Flußdiagramm wird jetzt wieder in ein Programm umgesetzt (Tab. 2).
Laden Sie dieses Programm und testen Sie es.

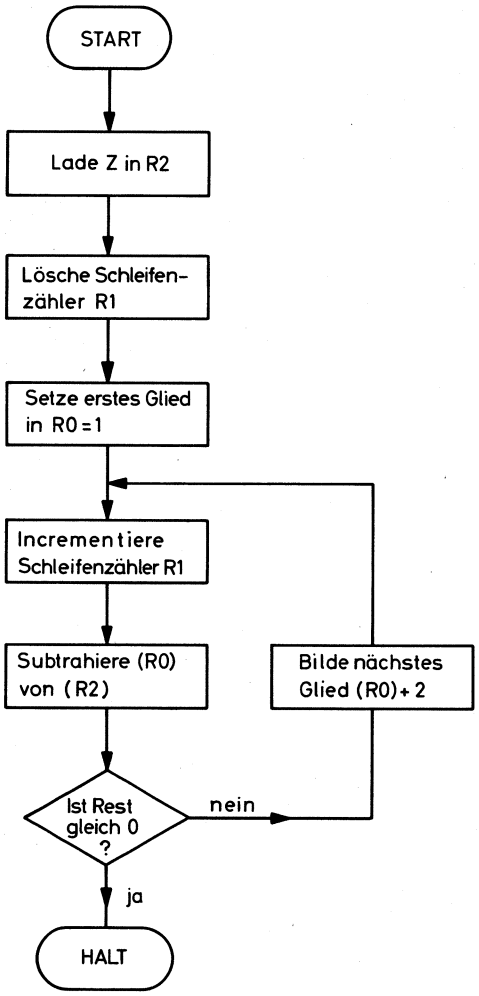


Bild 2
Flußdiagramm zum 2. Beispiel

Adresse	Inhalt	Befehl	Kommentar
4 0	8 6	LOAD R2, F F	Z einlesen
4 1	F F		
4 2	2 5	SUBR R1, R1	Schleifenzähler löschen
4 3	8 0	LOAD R0, # 0 1	1. Glied gleich 1 setzen
4 4	0 1		
4 5	6 1	INCR R1	Erhöhe Schleifenzähler
4 6	2 2	SUBR R2, R0	Ziehe Glied von Rest ab
4 7	E 1	JMPZ 4 D	HALT bei Rest = 0
4 8	4 D		
4 9	9 0	ADDM R0, # 0 2	nächstes Glied bilden
4 A	0 2		
4 B	E 0	JUMP 4 5	Schleife
4 C	4 5		
4 D	0 0	HALT	

Tab. 2
1. Programm zum Ziehen einer Quadratwurzel

Tab. 3 zeigt ein weiteres Programm, das ebenfalls zum Ziehen der Quadratwurzel dient. Laden Sie auch dieses Programm in den MP-Experimenter.

Adresse	Inhalt	Befehl	Kommentar
4 0	8 6	LOAD R2, F F	
4 1	F F		
4 2	2 5	SUBR R1, R1	
4 3	8 0	LOAD R0, # F F	
4 4	F F		
4 5	9 0	ADDM R0, # 0 2	
4 6	0 2		
4 7	6 1	INCR R1	
4 8	2 2	SUBR R2, R0	
4 9	E 1	JMPZ 4 D	
4 A	4 D		
4 B	E 0	JUMP 4 5	
4 C	4 5		
4 D	0 0	HALT	

Tab. 3
2. Programm zum Ziehen einer Quadratwurzel

Testen Sie auch dieses Programm.

Aufgaben:

1. Geben Sie für das Programm in Tab. 3 das Flußdiagramm an!

2. Worin unterscheiden sich die beiden Programme (Tab. 2 und Tab. 3)?

Experiment 8: Computed-JUMP

Diese Befehlsart soll an einem allgemein verständlichen Beispiel erläutert werden.
In einem Produktionsablauf wird das eingehende Material, z.B. Stahlkugeln, auf seine Toleranzen untersucht. Aus dem möglichen Toleranzspektrum der Kugeln sollen diejenigen selektiert werden, die am Ausgang der Meßvorrichtung die Kombinationen 0 0_{16}, 0 1_{16}, 0 2_{16} und 0 3_{16} erzeugen. Weiterhin wird verlangt, daß Kugeln mit den Meßdaten 0 0_{16} ein Signal 0 0_{16}, Kugeln mit 0 1_{16} ein Signal 8 1_{16}, Kugeln mit 0 2_{16} ein Signal 4 2_{16} und Kugeln mit 0 3_{16} ein Signal 0 3 erzeugen. Alle anderen Kugeln sollen das Signal F F_{16} erzeugen. Über die so gewonnenen 5 Ausgangssignale kann dann die entsprechende Weiterleitung der Kugeln bis an bestimmte Fertigungstellen gesteuert werden.
Wir übertragen diese verbale Aufgabenstellung zunächst in ein Flußdiagramm (Bild 1).

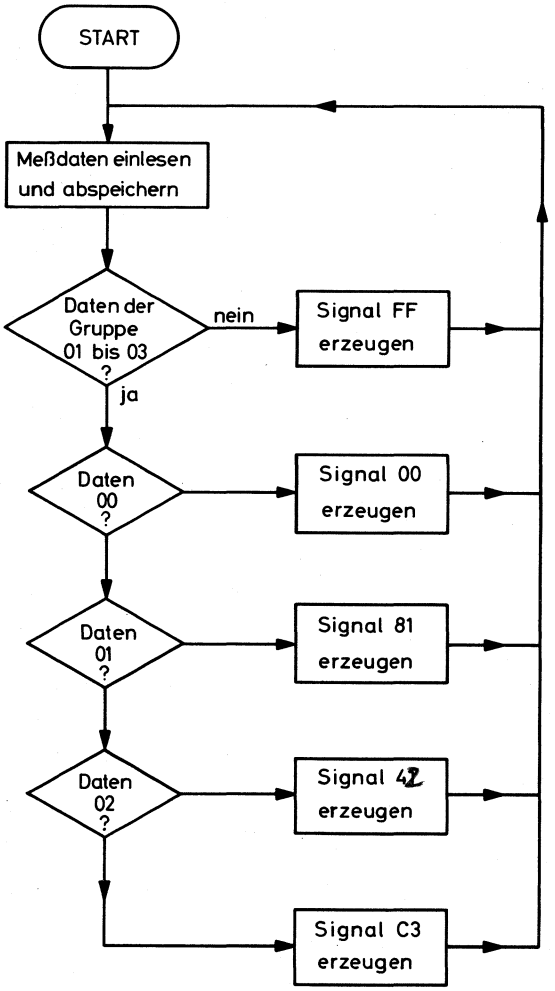


Bild 1
Flußdiagramm zu dem Beispiel

Dieses Flußdiagramm kann z.B. mit nachfolgendem Programm realisiert werden.

Adresse	Inhalt	Befehl	Kommentar
4 0	8 6	LOAD R2, F F	Meßdaten einlesen
4 1	F F		
4 2	7 6	STAC R2, F E	Daten abspeichern
4 3	F E		
4 4	3 A	IORR R2, R2	N-Flag setzen
4 5	E 2	JMPN B 0	Springe, wenn höchste Stelle 1
4 6	B 0		
4 7	A 2	SUBM R2, # 0 3	Daten größer als 0 3?
4 8	0 3		
4 9	E 1	JMPZ 5 0	Daten gleich 0 3
4 A	5 0		
4 B	E 2	JMPN 5 0	Daten 0 0 bis 0 2
4 C	5 0		
4 D	E 0	JUMP B 0	Daten größer als 0 3
4 E	B 0		
.			
.			
5 0	8 6	LOAD R2, F E	Meßdaten einlesen
5 1	F E		
5 2	9 2	ADDM R2, # 6 0	Indexregister laden
5 3	6 0		
5 4	E 8	JUMP @ R2	Indirekter Sprung
.			
.			
6 0	7 0	Adresse für Daten 0 0	
6 1	8 0	Adresse für Daten 0 1	
6 2	9 0	Adresse für Daten 0 2	
6 3	A 0	Adresse für Daten 0 3	
.			
.			
7 0	8 0	LOAD R0, # 0 0	Erzeuge 0 0-Signal
7 1	0 0		
7 2	E 0	JUMP 4 0	
7 3	4 0		
.			
.			
8 0	8 0	LOAD R0, # 8 1	Erzeuge 8 1-Signal
8 1	8 1		
8 2	E 0	JUMP 4 0	
8 3	4 0		
.			
.			
9 0	8 0	LOAD R0, # 4 2	Erzeuge 4 2-Signal
9 1	4 2		
9 2	E 0	JUMP 4 0	
9 3	4 0		
.			
.			
A 0	8 0	LOAD R0, # C 3	Erzeuge C 3-Signal
A 1	C 3		
A 2	E 0	JUMP 4 0	
A 3	4 0		
.			
.			
B 0	8 0	LOAD R0, # F F	Erzeuge F F-Signal
B 1	F F		
B 2	E 0	JUMP 4 0	
B 3	4 0		

Programmerläuterung:
Die an A_7 bis A_0 simulierten Meßdaten werden über Adresse F F eingelesen und in Adresse F E abgespeichert. Mit A_7 bis A_0 lassen sich 256 verschiedene Meßdaten erzeugen. Der Befehl JMPN B 0 wird dann ausgelöst, wenn es sich um Daten der Form 1 *x x x x x x x* handelt, also wenn das werthöchste bit gleich 1 ist. Dies kann vom N-Flag geprüft werden, das ja bekanntlich immer 1 ist, wenn das werthöchste bit gleich 1 ist. Da aber die Befehle LOAD R2, F F und STAC R2, F E das N-Flag nicht initialisieren, muß dies über einen anderen Befehl erfolgen. In diesem Programm wird hierfür der Befehl IORR R2, R2 benutzt. Dieser Befehl verändert nicht den Registerinhalt, setzt aber das N-Flag, wenn das werthöchste bit gleich 1 ist. Von einer Eingabe, die das N-Flag nicht setzt, wird jetzt durch den Befehl SUBM R2, # 0 3 die Zahl 0 3 subtrahiert. Für alle Eingaben, die wertmäßig größer als 3 sind, treffen die Bedingungen für die Befehle JMPZ und JMPN nicht zu. In diesem Falle wird der unbedingte Sprung JUMP B 0 ausgeführt, d.h. das Ausgangssignal F F in Register R0 erzeugt. Bei Eingabedaten von 0 0_{16} bis 0 3_{16} wird entweder der Befehl JMPZ (Eingabe = 0 3_{16}) oder der Befehl JMPN (Eingabe 0 0, 0 1 oder 0 2) ausgelöst, d.h., es erfolgt ein Sprung zur Adresse 5 0. Hier werden jetzt die Ursprungsdaten wieder in R2 geladen. Mit dem Befehl ADDM R2, # 6 0 wird der Computed-JUMP vorbereitet. Die Ausführung erfolgt über JUMP @ R2.
Bei diesem Programm ist z.B. die Adresse, die bei der Eingabe 0 0_{16} das Ausgangssignal 0 0_{16} erzeugen soll, in Adresse 6 0 gespeichert (Adresse 7 0). Wird also an A_7 bis A_0 0 0_{16} eingegeben, so ist der Inhalt von R2 nach dem Befehl ADDM R2, # 6 0 natürlich 6 0. Damit erfolgt ein Programmsprung zur Adresse 7 0. Bei einer Eingabe von z.B. 0 3_{16}, steht im Indexregister R2 jetzt 6 3, so daß ein Sprung zur Adresse A 0 erfolgt.
Laden Sie dieses Programm in den MP-Experimenter, und testen Sie es bei verschiedenen Eingaben.

Aufgaben:

1. Aus welchem Grunde wird bei diesem Programm in Adresse 4 5 der Befehl JMPN B 0 benötigt?

2. Entwickeln Sie ein Programm, das die Aufgabenstellung ohne Computed-JUMP löst!

Experiment 9: Divisionsprogramm

Nachdem Sie bereits im theoretischen Teil ein Multiplikationsprogramm kennengelernt haben, soll hier ein Divisionsprogramm behandelt werden. Als Algorithmus liegt diesem Programm die Divisionsmethode mit Divisorverschiebung und Subtraktion zugrunde (siehe Lehrheft 1, Seite 1.24). Weiterhin ist dieses Programm so ausgelegt, daß bei einer Division, die nicht „aufgeht", in R0 der ganzzahlige Quotiententeil und in R2 der gebrochene Quotiententeil erscheint.
Laden Sie das in Tab. 1 gezeigte Programm.

Adresse	Inhalt	Befehl	Kommentar
4 0	0 0	HALT	
4 1	8 4	LOAD R0, F F	
4 2	F F		
4 3	0 0	HALT	
4 4	8 5	LOAD R1, F F	
4 5	F F		
4 6	0 0	HALT	
4 7	4 A	XORR R2, R2	Ergebnisregister löschen
4 8	4 F	XORR R3, R2 R3	Verschiebezähler löschen
4 9	6 9	RACL R1	
4 A	E 3	JMPC 4 F	
4 B	4 F		
4 C	6 3	INCR R3	
4 D	E 0	JUMP 4 9	
4 E	4 9		
4 F	6 D	RACR R1	
5 0	2 4	SUBR R0, R1	
5 1	E 3	JMPC 8 0	
5 2	8 0		
5 3	6 2	INCR R2	
5 4	6 7	DECR R3	
5 5	E 2	JMPN 5 B	
5 6	5 B		
5 7	6 A	RACL R2	
5 8	6 D	RACR R1	
5 9	E 0	JUMP 5 0	
5 A	5 0		
5 B	7 6	STAC R2, A 0	
5 C	A 0		
5 D	4 A	XORR R2, R2	
5 E	8 3	LOAD R3, # 0 8	
5 F	0 8		
6 0	6 A	RACL R2	
6 1	6 8	RACL R0	
6 2	E 3	JMPC 9 5	
6 3	9 5		
6 4	2 4	SUBR R0, R1	
6 5	E 3	JMPC 9 0	
6 6	9 0		
6 7	6 2	INCR R2	
6 8	6 7	DECR R3	
6 9	E 1	JMPZ 6 D	
6 A	6 D		
6 B	E 0	JUMP 6 0	
6 C	6 0		
6 D	8 4	LOAD R0, A 0	
6 E	A 0		
6 F	E 0	JUMP 4 0	

Adresse	Inhalt	Befehl	Kommentar
7 0	4 0		
.			
.			
8 0	1 4	ADDR R0, R1	
8 1	3 0	IORR R0, R0	
8 2	E 0	JUMP 5 4	
8 3	5 4		
.			
.			
9 0	1 4	ADDR R0, R1	
9 1	3 0	IORR R0, R0	
9 2	E 0	JUMP 6 8	
9 3	6 8		
.			
.			
9 5	2 4	SUBR R0, R1	
9 6	3 0	IORR R0, R0	
9 7	E 0	JUMP 6 7	
9 8	6 7		

Tab. 1
Divisionsprogramm

Überprüfen Sie mit EXAMINE, ob das Programm fehlerfrei geladen wurde.
Wie bei der Digitaluhr in Lehrheft 2 ermöglicht der Programmvorspann von Adresse 4 0 bis 4 6 eine Operandeneingabe mit Hilfe des EXAMINE-Schalters. Der Eingabeablauf ist folgender: Bei der **ersten** Inbetriebnahme des Programms Adresse 4 0 laden, RUN-Schalter auf 1 und Single-Step-Schalter auf 0 stellen. Mit A_7 bis A_{10} ersten Operanden (Dividend) einstellen, und EXAMINE einmal takten. Der Dividend wird somit in R0 geladen. Jetzt zweiten Operanden (Divisor) mit A_7 bis A_0 einstellen, und EXAMINE einmal takten. Jetzt steht der Divisor in R1. Ein weiteres Takten von EXAMINE startet das eigentliche Rechenprogramm. Durch den JUMP-Befehl in Adresse 6 F springt der Programmzähler am Ende einer Division wieder auf die Adresse 4 0 zurück, so daß sofort neue Operanden eingegeben werden können. Testen Sie das Programm mit folgenden Beispielen:

1. $64_{10} : 32_{10} =$

R0								R2							
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

$\triangleq$

$2{,}0_{10}$

2. $50_{10} : 4_{10} =$

R0								R2							
0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0

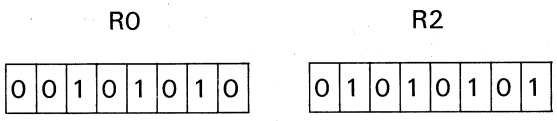
$\triangleq$

$12{,}5_{10}$

Die Stellenwertigkeiten in R0 können als bekannt vorausgesetzt werden. Die Nachkommastellen in R2 haben folgende Wertigkeiten:

2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}
$\triangleq$	$\triangleq$	$\triangleq$	$\triangleq$	$\triangleq$	$\triangleq$	$\triangleq$	$\triangleq$
$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$	$\frac{1}{128}$	$\frac{1}{256}$

3. $127_{10} : 3_{10} =$



$\triangleq$

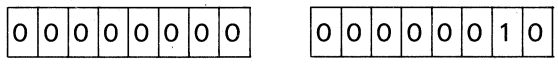
$42{,}332\,031\,25_{10}$

Das korrekte Ergebnis wäre in diesem Fall:

42,33

Die Abweichung vom tatsächlichen Ergebnis ist durch die begrenzte Stellenzahl bedingt. Die wertniedrigste Stelle bestimmt mit ihrer Wertigkeit von $1/256_{10} = 0{,}003\,906\,25_{10}$ die Rechengenauigkeit. Aufgerundet ergibt sich also eine mögliche Abweichung von $0{,}004_{10}$. Einige weitere Beispiele sollen das noch verdeutlichen:

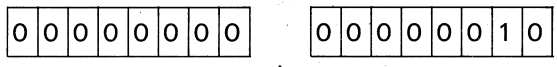
4. $1_{10} : 128_{10} =$



$\triangleq$

$0{,}007\,812\,5_{10}$

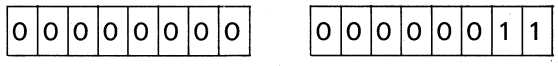
5. $1_{10} : 86_{10} =$



$\triangleq$

$0{,}007\,812\,5_{10}$

6. $1_{10} : 85_{10} =$



$\triangleq$

$0{,}011\,718\,75_{10}$

Beispiel 4 liefert ein exaktes Ergebnis, da $1/128_{10}$ als Stellenwertigkeit vorkommt. Im Beispiel 5 beträgt die Abweichung 0,0038, während Beispiel 6 auf ca. $5 \cdot 10^{-5}$ genau ist.

Programmerläuterung:
Das Divisionsprogramm kann in 3 Abschnitte unterteilt werden.

Teil 1:
Verschieben des Divisors in R1, bis seine erste 1 im höchsten bit von R1 steht. Die Anzahl der notwendigen Verschiebungen wird festgestellt und im Verschieberegister R3 gespeichert (Adressen 4 9 bis 4 F).

Teil 2:
Durch Subtraktion und Verschiebung den ganzzahligen Anteil des Quotienten errechnen und abspeichern (Adressen 5 0 bis 5 B und 8 0 bis 8 3).

Teil 3:
Gebrochenen Quotiententeil errechnen, Ergebnis in R2. Ganzzahligen Anteil zurückholen und in R0 bringen (Adressen 5 D bis 7 0, 9 0 bis 9 3 und 9 5 bis 9 8).

Die Adressen 4 0 bis 4 5 dienen nur zur Operandeneingabe.
Nachfolgend sollen diese Programmteile anhand von Flußdiagrammen besprochen werden (Bilder 1 bis 3).

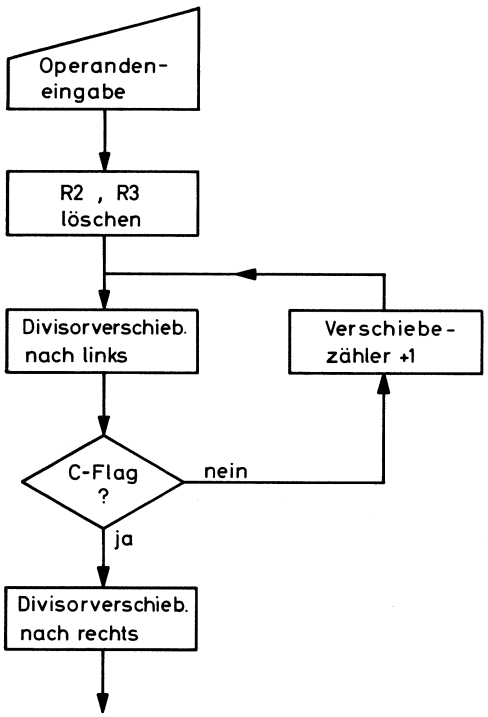


Bild 1
Flußdiagramm zu Teil 1

Zum Löschen der Register R2 und R3 wird der XORR-Befehl verwendet, weil damit auch ein eventuell vorhandenes C-Flag gelöscht wird (siehe Befehlsliste). Nach jeder Divisorverschiebung RACL R1, wird das C-Flag abgefragt. Wenn kein C-Flag vorhanden ist, wird der Verschiebezähler um 1 erhöht und anschließend die nächste Verschiebung durchgeführt. C-Flag = 1 zeigt an, daß bereits eine Verschiebung zuviel durchgeführt wurde. Also wird diese Verschiebung nicht mehr gezählt, sondern vor dem Übergang zum Programmteil 2 durch RACR R1 rückgängig gemacht.

Fragen zu Programmteil 1:

a) Welchen Inhalt haben am Ende des ersten Programmteiles die Register R0 bis R3, wenn als Dividend 7_{10} und als Divisor 3_{10} eingegeben werden?

b) Weshalb muß am Programmanfang das C-Flag gelöscht werden?

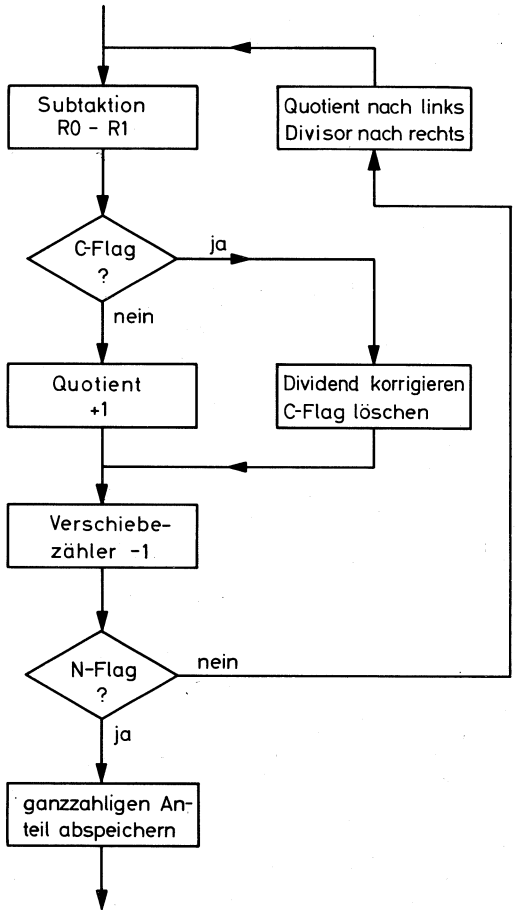


Bild 2
Flußdiagramm zu Teil 2

Das C-Flag nach der Subtraktion R0 – R1 zeigt an, ob der Divisor vom Dividenden abgezogen werden kann. C-Flag = 1 heißt dabei, daß R1 größer ist als R0. Eine Subtraktion war nicht möglich, also muß mit ADDR R0, R1 der Dividend wieder auf seinen ursprünglichen Wert gebracht und außerdem mit IORR R0, R0 das C-Flag gelöscht werden. Diese beiden Befehle finden Sie unter den Adressen 8 0 und 8 1. Mit dem anschließenden JUMP-Befehl erfolgt der Rücksprung in das Hauptprogramm. Wie im Flußdiagramm zu sehen ist, wird dabei der Programmschritt „Quotient + 1" (INCR R2; Adresse 5 3) übersprungen. Falls nach dem Erniedrigen des Verschiebezählers keine N-Flag-Anzeige erscheint, werden der Quotient nach links und der Divisor nach rechts verschoben. Dann erfolgt der nächste Schleifendurchlauf. Eine Subtraktion R0 – R1, die keine C-Flag-Anzeige erzeugt, führt zur Erhöhung des Quotienten. Ist die Anzahl der Schleifendurchläufe gleich der Anzahl der Divisorverschiebungen im Teil 1, wird dies durch das N-Flag angezeigt. Damit steht der ganzzahlige Anteil des Quotienten fest und wird vor dem Übergang zum Teil 3 in A 0 abgespeichert.

Fragen zu Programmteil 2

a) Wie oft wird der zweite Programmteil durchlaufen, wenn als Dividend 68_{10} und als Divisor 9_{10} eingegeben werden?

b) Weshalb muß mit RACL R2 (Adresse 5 7) der Quotient bei jedem Schleifendurchlauf verschoben werden?

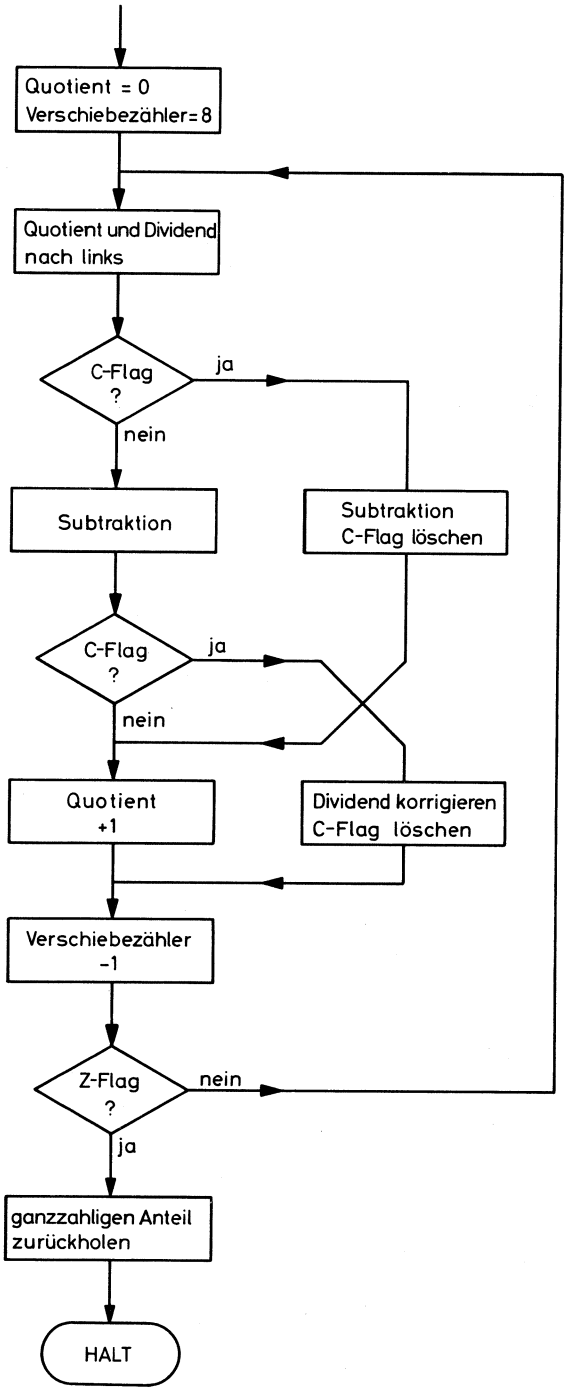


Bild 3
Flußdiagramm zu Teil 3

Vorbemerkungen:
Der Algorithmus für den gebrochenen Anteil des Quotienten unterscheidet sich von dem in Programmteil 2 dadurch, daß nicht mehr der Divisor nach rechts, sondern der Dividend bzw. der Dividendenrest nach links verschoben wird. Ein am Ende des zweiten Programmteiles vorhandener Dividendenrest in R0 ist auf jeden Fall kleiner als der Divisor. Wird nun der Dividendenrest um eine Stelle nach links verschoben, entspricht dies einer Multiplikation des Restes mit 2. Über eine nachfolgende Subtraktion muß jetzt festgestellt werden, ob der mit 2 multiplizierte (nach links verschobene!) Dividend größer als der Divisor ist. Trifft dies zu, muß im Quotientenregister R2 an der Stelle 2^{-1} eine 1 erscheinen, im anderen Falle eine 0. Falls auch nach dieser Subtraktion ein Dividendenrest verbleibt, so ist dieser wiederum kleiner als der Divisor. Eine weitere Verschiebung des Dividendenrestes entspricht einer nochmaligen Multiplikation mit 2. Ist jetzt der Dividend größer als der Divisor – festgestellt durch die

zweite Subtraktion – muß an der Stelle 2^{-2} eine 1 erscheinen, im anderen Falle eine 0. Nach dem gleichen Prinzip lassen sich in 6 weiteren Verschiebeschritten die Stellenwertigkeiten 2^{-3} bis 2^{-8} berechnen. Der Algorithmus zeigt, daß zuerst die Nachkommastelle mit der **größten** Wertigkeit (2^{-1}) berechnet wird. Tritt hierbei das Ergebnis 1 auf, so kann diese 1 durch einen INCR-Befehl in die **niedrigste** Stelle (2^{-8}) des Quotientenregisters R2 gebracht werden. Damit diese 1 aber am Ende der Division an der richtigen Stelle (2^{-1}) erscheint, muß sie **siebenmal** nach links verschoben werden.
Bei einer 8-bit-Stellenzahl für den Nachkommateil muß der vorher beschriebene Vorgang **achtmal** durchgeführt werden.

Nach diesen grundsätzlichen Betrachtungen soll nun das Flußdiagramm diskutiert werden. Der noch in R2 vorhandene Vorkommateil muß zunächst gelöscht werden (XORR R2). Der Verschiebezähler muß auf 0 8 gesetzt werden (LOAD R3, # 08).
Im nächsten Programmteil erfolgt der erste Verschiebevorgang. Da das Quotientenregister noch den Inhalt 0 hat, ändert sich in diesem Register nichts. Der Dividend wird eine Stelle nach links geschoben. Hat der Dividendenrest in der werthöchsten Stelle eine 1, so wird diese 1 nach der Verschiebung in das C-Flag übernommen.
Eine 1 im C-Flag nach der Verschiebung bedeutet, daß der Dividend in seiner höchsten Stelle, nämlich dem C-Flag, die Wertigkeit $2^8 = 256_{10}$ hat. Der Divisor kann aufgrund der 8-bit-Eingabe aber **maximal** den Wert 255_{10} haben, so daß in jedem Falle eine Subtraktion möglich ist. Der Befehl SUBR R0, R1 in Adresse 9 5 (Inhalt R0 = Dividend, Inhalt R1 = Divisor), berücksichtigt bei der Subtraktion **nur** die Inhalte der beiden Register, jedoch **nicht** das bei der Verschiebung von R0 entstandene C-Flag mit der Wertigkeit 2^8. Dadurch wird der Divisor wieder größer als der Dividend, so daß beim Ausführen der Subtraktion eine Entlehnung, angezeigt durch das C-Flag, entsteht. Wenn wir gedanklich die Entlehnung vom Dividenden-C-Flag subtrahieren ist das Ergebnis in der Stelle 2^8 gleich 0. Da der Rechner diese Subtraktion nicht durchführen kann, wird das C-Flag durch den Befehl IORR R0, R0 gelöscht. In R0 steht somit die richtige Differenz.
Die hier besprochene Subtraktion unter Berücksichtigung des C-Flags braucht nur dann durchgeführt werden, wenn der Divisor größer als 128_{10} ist, also in seinem **höchsten** bit und einem weiteren bit eine 1 hat. In allen anderen Fällen wird die Subtraktion durch Befehl SUBR R0, R1 in **Adresse 6 4** durchgeführt (gerader Pfad im Flußdiagramm).
Über den Zustand des C-Flags wird wieder die Entscheidung getroffen, ob als Ergebnis eine 1 oder eine 0 an das Quotientenregister R2 gegeben werden muß. Bei C-Flag gleich 0 treffen sich die beiden Subtraktionspfade und bewirken, daß das Quotientenregister um 1 erhöht wird. War das C-Flag auf dem geraden Subtraktionspfad gleich 1, so darf keine Erhöhung des Quotientenregisters erfolgen (Befehl INCR R2 in Adresse 6 7 darf nicht durchgeführt werden). Stattdessen wird wie in Programmteil 2 der letzte Dividendenwert wieder errechnet und das C-Flag gelöscht (Adresse 9 0 bis 9 3).
Anschließend wird der Verschiebezähler R3 decrementiert. Erst nach 8 Schleifendurchläufen wird das Z-Flag gleich 1 und löst das Zurückholen des Vorkommateiles aus Adresse A 0 in R0 und den Rücksprung auf die Anfangsadresse 4 0 aus.
Wie Sie sehen, erfordert die Bearbeitung der Nachkommastellen eine recht intensive und genaue Beschäftigung mit den Auswirkungen der einzelnen Befehle. Darüber hinaus muß man sich über den Algorithmus genau im Klaren sein. Wir empfehlen Ihnen, diesen Programmteil mit mehreren Beispielen im SINGLE-STEP-Betrieb durchzuführen. Als Aufgabenbeispiele empfehlen wir:

a) $128_{10} : 129_{10} =$
b) $1_{10} : 255_{10} =$
c) $33_{10} : 136_{10} =$

Fragen zu Programmteil 3

1. Welche Auswirkung würde ein versehentliches Laden der Adresse 9 8 mit 6 8 haben?

2. Geben Sie die Wertigkeit der Ergebnisanzeige von R2 an, wenn der Verschiebezähler im Programmteil 3 anstatt auf 0 8 auf 0 4 gesetzt wird!

Experiment 10: Anrufen von Unterprogrammen mit den JUMP-Befehlen

Als Beispiel für den Einsatz von Unterprogrammen soll eine Oberflächenberechnung für einen Quader durchgeführt werden. Zu Beginn sollen über A_7 bis A_0 Höhe, Breite und Tiefe eingegeben (in cm) und in den Adressen 9 0, 9 1 und 9 2 gespeichert werden.
Die erforderlichen Multiplikationen werden in einem Unterprogramm durchgeführt. Wie später noch näher erläutert wird, ist ein Multiplikationsprogramm im ROM-Bereich des Speichers von Adresse 0 1 bis einschließlich 1 8 enthalten. Abgesehen von den Sprungadressen ist dieses Programm identisch mit dem im theoretischen Teil behandelten. Der letzte Befehl in den Adressen 1 7 und 1 8 im ROM-Bereich entspricht dem Befehl in den Adressen 3 6 und 3 7 des theoretisch behandelten Programms. Das bedeutet, daß die Abspeicherung des Ergebnisses nicht mehr vorgegeben ist, sondern beginnend bei Adresse 19 mit geeigneten Befehlen programmiert werden muß.
Überprüfen Sie dieses Multiplikationsprogramm mit EXAMINE.
Die verbale Aufgabenstellung wird zunächst in einem Flußdiagramm festgehalten (Bild 1).

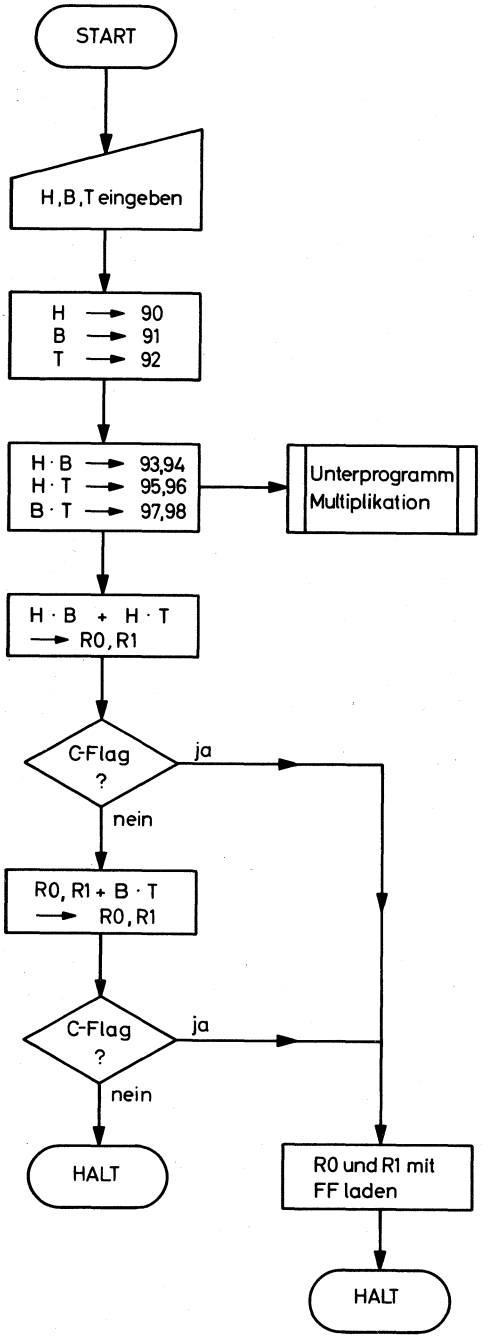


Bild 1
Flußdiagramm für die Oberflächenberechnung eines Quaders

Laden Sie das in Tab. 1 gezeigte Programm.

Adresse	Inhalt	Befehl	Kommentar
1 9	E 4	JUMP @ F 1	Rücksprunganweisung
1 A	F 1		
.			
.			
3 0	0 0	HALT	
3 1	8 4	LOAD R0, F F	
3 2	F F		
3 3	7 4	STAC R0, 9 0	
3 4	9 0		
3 5	0 0	HALT	
3 6	8 4	LOAD R0, F F	Höhe, Breite und Tiefe einlesen
3 7	F F		und abspeichern
3 8	7 4	STAC R0, 9 1	
3 9	9 1		
3 A	0 0	HALT	
3 B	8 4	LOAD R0, F F	
3 C	F F		
3 D	7 4	STAC R0, 9 2	
3 E	9 2		
3 F	8 4	LOAD R0, 9 0	Höhe in R0
4 0	9 0		
4 1	8 5	LOAD R1, 9 1	Breite in R1
4 2	9 1		
4 3	8 2	LOAD R2, # 4 9	Rücksprungadresse in F 1
4 4	4 9		
4 5	7 6	STAC R2, F 1	
4 6	F 1		
4 7	E 0	JUMP 0 1	Sprung ins Unterprogramm
4 8	0 1		
4 9	7 4	STAC R0, 9 3	Ergebnis Höhe · Breite speichern
4 A	9 3		
4 B	7 5	STAC R1, 9 4	
4 C	9 4		
4 D	8 4	LOAD R0, 9 0	Höhe in R0
4 E	9 0		
4 F	8 5	LOAD R1, 9 2	Tiefe in R1
5 0	9 2		
5 1	8 2	LOAD R2, 5 7	Rücksprungadresse in F 1
5 2	5 7		
5 3	7 6	STAC R2, F 1	
5 4	F 1		
5 5	E 0	JUMP 0 1	Sprung ins Unterprogramm
5 6	0 1		
5 7	7 4	STAC R0, 9 5	Ergebnis Höhe · Tiefe speichern
5 8	9 5		
5 9	7 5	STAC R1, 9 6	
5 A	9 6		
5 B	8 4	LOAD R0, 9 1	Breite in R0
5 C	9 1		
5 D	8 5	LOAD R1, 9 2	Tiefe in R1
5 E	9 2		

Adresse	Inhalt	Befehl	Kommentar
5 F	8 2	LOAD R2, # 6 5	} Rücksprungadresse in F 1
6 0	6 5		
6 1	7 6	STAC R2, F 1	
6 2	F 1		
6 3	E 0	JUMP 0 1	Sprung ins Unterprogramm
6 4	0 1		
6 5	7 4	STAC R0, 9 7	} Ergebnis Breite · Tiefe speichern
6 6	9 7		
6 7	7 5	STAC R1, 9 8	
6 8	9 8		
6 9	3 0	IORR R0, R0	Flags löschen
6 A	8 5	LOAD R1, 9 4	} rechte Hälfte addieren
6 B	9 4		
6 C	9 5	ADDM R1, 9 6	
6 D	9 6		
6 E	8 4	LOAD R0, 9 3	linke Hälfte nach R0 bringen
6 F	9 3		
7 0	E 3	JMPC A 0	Übertragsverarbeitung
7 1	A 0		
7 2	9 4	ADDM R0, 9 5	linke Hälfte addieren
7 3	9 5		
7 4	E 3	JMPC 8 2	Aussprung aus dem Programm, wenn
7 5	8 2		Ergebnis zu groß
7 6	9 5	ADDM R1, 9 8	rechte Hälfte addieren
7 7	9 8		
7 8	E 3	JMPC A 5	Übertragsverarbeitung
7 9	A 5		
7 A	9 4	ADDM R0, 9 7	linke Hälfte addieren
7 B	9 7		
7 C	E 3	JMPC 8 2	Aussprung
7 D	8 2		
7 E	6 9	RACL R1	} Multiplikation mit 2 durch Linksschieben
7 F	6 8	RACL R0	
8 0	E 0	JUMP 30	
8 1	3 0		
8 2	8 0	LOAD R0, # FF	} F F nach R0 und R1
8 3	F F		
8 4	0 1	MOVE R0, R1	
8 5	E 0	JUMP 30	
8 6	3 0		
.			
.			
A 0	3 0	IORR R0, R0	
A 1	6 0	INCR R0	
A 2	E 0	JUMP 7 2	
A 3	7 2		
.			
.			
A 5	3 0	IORR R0, R0	
A 6	6 0	INCR R0	
A 7	E 0	JUMP 7A	
A 8	7 A		

Tab. 1
Programm für die Oberflächenberechnung eines Quaders

Testen Sie das Programm mit nachfolgenden Beispielen:

a) Höhe: 2 cm
Breite: 4 cm
Tiefe: 3 cm

Ergebnis:

C-Flag	R0	R1
0	0 0 0 0 0 0 0 0	0 0 1 1 0 1 0 0

≙
52 cm^2

b) Höhe: 128 cm
Breite: 60 cm
Tiefe: 25 cm

Ergebnis:

C-Flag	R0	R1
0	0 1 1 0 0 0 0 0	1 0 1 1 1 0 0 0

≙
24 760 cm^2

c) Höhe: 255 cm
Breite: 128 cm
Tiefe: 192 cm

Ergebnis: R0 und R1 mit FF geladen

Programmerläuterung:
Der erste Programmteil von Adresse 3 0 bis 3 E dient zum Laden und Abspeichern der Daten. Die nächsten 3 Programmteile (3 F bis 4 C, 4 D bis 5 A und 5 B bis 6 8) sind von den Befehlen her identisch. Hier werden mit Hilfe des Unterprogramms „Multiplikation" die 3 Produkte gebildet. Die mit doppelter Genauigkeit errechneten Zwischenergebnisse werden in den Adressen 9 3 bis 9 8 gespeichert.
Im fünften Programmteil (6 9 bis 7 D) werden die 3 Multiplikationsergebnisse wieder mit doppelter Genauigkeit addiert.
Die beiden JMPC-Befehle in den Adressen 7 4, 7 5 und 7 C, 7 D lassen in R0 und R1 jeweils F F_{16} erscheinen. C-Flag = 1 nach einer der beiden Additionen bedeutet nämlich, daß die nachfolgende Multiplikation (7 E und 7 F) mit 2 den zulässigen Zahlenbereich überschreiten und damit ein falsches Ergebnis entstehen würde. Die Addition mit doppelter Genauigkeit wurde im Lehrheft 2 bereits beschrieben. Zunächst werden die beiden rechten Hälften addiert (Adresse 6 3). Entsteht dabei ein Übertrag, erfolgt ein Sprung nach A 0. Dort wird das C-Flag gelöscht und zur **linken** Hälfte in R0 eine 1 addiert. Bei Adresse 7 2 wird dann auch die linke Hälfte addiert. Der Vorgang wiederholt sich von 7 6 bis 7 B.
Bei allen Unterprogrammen muß am Ende noch die richtige Rücksprungadresse gefunden werden. In unserem Programm werden die Rücksprungadressen in F 1 gespeichert, also muß der Rücksprung in das Hauptprogramm mit JUMP @ F 1 erfolgen (Adressen 1 9 und 1 A).

Aufgaben:

1. Welche Auswirkung hat ein falsches Belegen der Adresse 1 9 mit Inhalt E 0 anstatt E 4?

2. Bei einer entsprechenden Eingabe soll nach der Addition der Zwischenergebnisse (nach Programmschritt 7 A) in R0 und R1 folgendes Ergebnis stehen:

R0	R1
1 0 0 1 1 0 0 1	1 1 0 1 1 1 0 0

a) Wie lautet das richtige Endergebnis?

b) Wie lautet das Endergebnis, wenn die Befehle 7 E und 7 F in ihrer Reihenfolge vertauscht werden?

Experiment 11: Anrufen von Unterprogrammen mit den CALL-Befehlen

Die gleiche Aufgabe wie in Experiment 10 soll jetzt unter Verwendung der CALL-Befehle gelöst werden.
Dadurch ergeben sich folgende Programmänderungen:

1. Vor dem ersten CALL-Befehl muß in das als Stack-Pointer arbeitende Register R3 die Anfangsadresse des Stack-Bereiches geladen werden.
2. Das Abspeichern der Zwischenergebnisse wird im Unterprogramm durchgeführt (Adressen 1 9 bis 2 0).
3. Die Verarbeitung eines Übertrages bei der Addition mit doppelter Genauigkeit wird jetzt ebenfalls über ein Unterprogramm durchgeführt.

Das Flußdiagramm unterscheidet sich jedoch nicht von Experiment 10.
Laden Sie das Programm (Tab. 1), und testen Sie es mit den Beispielen aus Experiment 10.

Adresse	Inhalt	Befehl	Kommentar
3 0	0 0	HALT	
3 1	8 3	LOAD R3, # 9 0	Lade Stack-Pointer mit Anfangs-
3 2	9 0		Adresse
3 3	8 2	LOAD R2, # F F	Lade R2 mit der Adresse FF
3 4	F F		
3 5	8 8	LOAD R0, @ R2	Lade R0 mit Höhe
3 6	7 C	STAC R0, @ R3↑	Speichere Höhe in Adr. 90
3 7	0 0	HALT	
3 8	8 8	LOAD R0, @ R2	Lade R0 mit Breite
3 9	7 C	STAC R0, @ R3↑	Speichere Breite in Adr. 91
3 A	0 0	HALT	
3 B	8 8	LOAD R0, @ R2	Lade R0 mit Tiefe
3 C	7 C	STAC R0, @ R3↑	Speichere Tiefe in Adr. 92
3 D	7 7	STAC R3, F 2	Speichere Adr. 92 in F2
3 E	F 2		
3 F	8 3	LOAD R3, # 8 0	Lade R3 (Stack-Pointer) mit Anfangs-
4 0	8 0		adresse
4 1	8 4	LOAD R0, 9 0	Lade R0 mit Höhe
4 2	9 0		
4 3	8 5	LOAD R1, 9 1	Lade R1 mit Breite
4 4	9 1		
4 5	F 0	CALL 0 1	Bilde Produkt aus Höhe u. Breite
4 6	0 1		
4 7	8 4	LOAD R0, 9 0	Lade R0 mit Höhe
4 8	9 0		
4 9	8 5	LOAD R1, 9 2	Lade R1 mit Tiefe
4 A	9 2		
→ 4 B	F 0	CALL 0 1	Bilde Produkt aus Höhe u. Tiefe
4 C	0 1		
4 D	8 4	LOAD R0, 9 1	Lade R0 mit Breite
4 E	9 1		
4 F	8 5	LOAD R0 (handschriftlich: R1), 9 2	Lade R1 mit Tiefe
5 0	9 2		
5 1	F 0	CALL 0 1	Bilde Produkt aus Breite u. Tiefe
5 2	0 1		
5 3	3 0	IORR R0, R0	Lösche R0
5 4	8 5	LOAD R1, 9 4	Lade R1 mit rechter Hälfte des Produkts
5 5	9 4		aus Höhe u. Breite
5 6	9 5	ADDM R1, 9 6	Addiere rechte Hälfte mit rechter Hälfte
5 7	9 6		aus Höhe u. Tiefe
5 8	8 4	? LOAD R0, 9 3	Lade linke Hälfte aus Höhe und Breite
5 9	9 3		
5 A	F 3	CALC A 0	Übertrag? → Unterprogramm

wäre richtiger, da hier ein Carry im [illegible] an der Addition überprüft werden muss. Funktioniert aber auch so, da LOAD keinen Einfluss auf die Flags hat

Adresse	Inhalt	Befehl	Kommentar
5 B	A 0		
5 C	9 4	ADDM R0, 9 5	
5 D	9 5		
5 E	E 3	JMPC 6 C	
5 F	6 C		
6 0	9 5	ADDM R1, 9 8	
6 1	9 8		
6 2	F 3	CALC A 0	
6 3	A 0		
6 4	9 4	ADDM R0, 9 7	
6 5	9 7		
6 6	E 3	JMPC 6 C	
6 7	6 C		
6 8	6 9	RACL R1	
6 9	6 8	RACL R0	
6 A	E 0	JUMP 3 0	
6 B	3 0		
6 C	8 0	LOAD R0, # F F	
6 D	F F		
6 E	0 1	MOVE R0, R1	
6 F	E 0	JUMP 3 0	
7 0	3 0		
A 0-Unterprogramm			
A 0	3 0	IORR R0, R0	
A 1	6 0	INCR R0	
A 2	E C	JUMP @ R3↑	
Ergänzung des Multiplikationsprogramms			
1 9	8 6	LOAD R2, F 2	
1 A	F 2		
1 B	7 8	STAC R0, @ R2	
1 C	6 2	INCR R2	
1 D	7 9	STAC R1, @ R2	
1 E	6 2	INCR R2	
1 F	7 6	STAC R2, F 2	
2 0	F 2		
2 1	E C	JUMP @ R3↑	

Tab. 1
Programm für die Oberflächenberechnung eines Quaders mit CALL-Befehlen

Programmerläuterung:
Bei einem Vergleich der Programmteile zur Dateneingabe in den Experimenten 10 und 11 werden Sie feststellen, daß jetzt zum Laden der mit A_7 bis A_0 eingestellten Daten in einen bestimmten Speicherplatz nur noch 2 Adressen belegt werden müssen. Diese Eingabemethode läßt sich jedoch nur dann anwenden, wenn die Daten in aufeinanderfolgenden Speichern untergebracht werden sollen.
Das Unterprogramm zur Verarbeitung von Überträgen wird über einen bedingten CALL-Befehl angerufen, wenn ein C-Flag erscheint.

Experiment 12: Unterprogramme mit Adreßargumentübergabe

Als Beispiel für eine Adreßargumentübergabe soll der Ausdruck $a^2 + b^2$ berechnet werden. Zur Berechnung der Quadrate wird dabei das Multiplikationsprogramm in Adresse 0 1 bis 1 8 herangezogen. Die Adressen 1 9 bis 2 5 müssen dazu noch mit den Befehlen geladen werden, die zum Abspeichern der Ergebnisse und zum Rücksprung in das Hauptprogramm notwendig sind. Die ebenfalls zum Unterprogramm gehörende Argumentübergabe wird in den Adressen 2 6 bis 3 0 gespeichert. Das Unterprogramm wurde in dieser Form bereits im Abschnitt 5.3 besprochen. Die Befehle können aus der Tabelle 5.3.3 entnommen werden. Für das Hauptprogramm kann jetzt das Flußdiagramm aufgestellt werden, das Bild 1 zeigt.

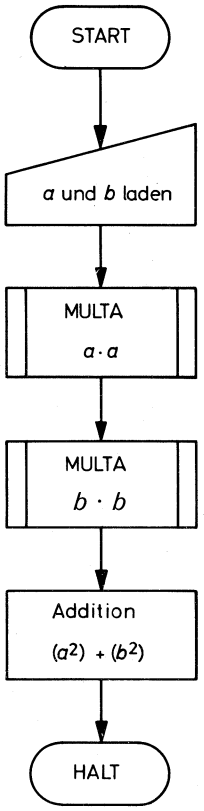


Bild 1
Flußdiagramm zur Multiplikation $a^2 + b^2$

Laden Sie jetzt die Unterprogrammergänzung aus Tabelle 5.3.3 und das in Tab. 1 gezeigte Hauptprogramm.

Adresse	Inhalt	Befehl	Kommentar
5 0	0 0	HALT	
5 1	8 4	LOAD R0, F F	
5 2	F F		*a* in A 0 speichern
5 3	7 4	STAC R0, A 0	
5 4	A 0		
5 5	0 0	HALT	
5 6	8 4	LOAD R0, F F	
5 7	F F		
5 8	7 4	STAC R0, A 1	*b* in A 1 speichern
5 9	A 1		
5 A	8 3	LOAD R3, # F F	Initialisierung Stack-Pointer
5 B	F F		
5 C	F 0	CALL MULTA	
5 D	2 6		
5 E	A 0	ADR Faktor 1	Unterprogrammanruf und Adreßargumente zur Berechnung von a^2
5 F	A 0	ADR Faktor 2	
6 0	B 0	ADR Produkt	
6 1	F 0	CALL MULTA	
6 2	2 6		
6 3	A 1	ADR Faktor 1	Unterprogrammanruf und Adreßargumente zur Berechnung von b^2
6 4	A 1	ADR Faktor 2	
6 5	B 2	ADR Produkt	
6 6	3 0	IORR R0, R0	C-Flag löschen
6 7	9 5	ADDM R1, B 0	
6 8	B 0		
6 9	E 3	JMPC 8 0	Addition und Rücksprung zum Programm-anfang
6 A	8 0		
6 B	9 4	ADDM R0, B 1	
6 C	B 1		
6 D	E 0	JUMP 5 0	
6 E	5 0		
.			
.			
8 0	3 0	IORR R0, R0	C-Flag löschen
8 1	6 0	INCR R0	Übertragsverarbeitung
8 2	E 0	JUMP 6 B	
8 3	6 B		

Tab. 1
Hauptprogramm zur Multiplikation $a^2 + b^2$

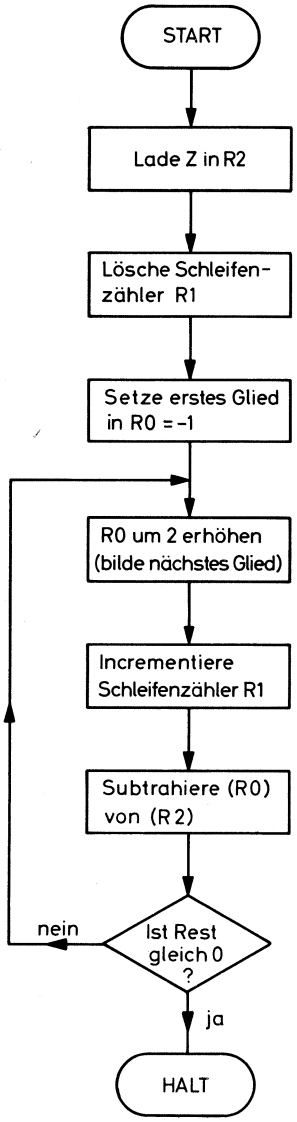
Aufgaben:

1. Weshalb werden in den Adressen 5 E, 5 F und 6 3, 6 4 jeweils dieselben Adressen für das Unterprogramm angegeben?

2. Geben Sie an, in welchen Speicherplätzen die Teilergebnisse a^2 und b^2 (rechte und linke Hälfte je ein Speicherplatz) zu finden sind!

Lösungen zu Experiment 7

1.



2. Das erste Programm entspricht nicht dem in Bild 5.1.1.1 dargestellten Format einer Schleife, da der Hauptteil der Schleife vor und nach der Entscheidung liegt.
Der Programmteil Bildung des nächsten Gliedes (R0) + 2 wird **erstmalig** nach der Entscheidung ausgeführt. Wird z.B. Z = 1 gewählt, wird dieser Teil nicht durchlaufen.
Bei der zweiten Version wird (R0) + 2 bereits beim ersten Schleifendurchlauf ausgeführt. Formal entspricht dieses Programm dem Schleifenformat in Bild 5.1.1.1.

Lösungen zu Experiment 8

1. Das N-Flag spricht immer dann an, wenn das werthöchste bit gleich 1 ist. Ohne diesen Befehl wird dann z.B. bei einer Eingabe 1 *x x x x x x x* bei Adresse 4 B der JMPN-Befehl ausgeführt.

2.

Adresse	Inhalt	Befehl	Kommentar
4 0	8 6	LOAD R2, F F	
4 1	F F		
4 2	0 9	MOVE R1, R2	
4 3	C 2	XORM R2, # 0 0	
4 4	0 0		
4 5	E 1	JMPZ 7 0	
4 6	7 0		
4 7	0 6	MOVE R2, R1	
4 8	C 2	XORM R2, # 0 1	
4 9	0 1		
4 A	E 1	JMPZ 8 0	
4 B	8 0		
4 C	0 6	MOVE R2, R1	
4 D	C 2	XORM R2, # 0 2	
4 E	0 2		
4 F	E 1	JMPZ 9 0	
5 0	9 0		
5 1	0 6	MOVE R2, R1	
5 2	C 2	XORM R2, # 0 3	
5 3	0 3		
5 4	E 1	JMPZ A 0	
5 5	A 0		
5 6	8 0	LOAD R0, # F F	
5 7	F F		
5 8	E 0	JUMP 4 0	
5 9	4 0		
.			
.			
7 0	8 0	LOAD R0, # 0 0	
7 1	0 0		
7 2	E 0	JUMP 4 0	
7 3	4 0		
.			
.			
8 0	8 0	LOAD R0, # 8 1	
8 1	8 1		
8 2	E 0	JUMP 4 0	
8 3	4 0		
.			
.			
9 0	8 0	LOAD R0, # 4 2	
9 1	4 2		
9 2	E 0	JUMP 4 0	
9 3	4 0		
.			
.			
A 0	8 0	LOAD R0, # C 3	
A 1	C 3		
A 2	E 0	JUMP 4 0	
A 3	4 0		

Lösungen zu Experiment 9

Antworten auf die Fragen zu Programmteil 1:

a)

R0:	0 0 0 0 0 1 1 1	Dividend unverändert
R1:	1 1 0 0 0 0 0 0	verschobener Divisor
R2:	0 0 0 0 0 0 0 0	gelöschtes Ergebnisregister
R3:	0 0 0 0 0 1 1 0	Verschiebezähler = 6

b) Das C-Flag würde sonst bei RACL R1 (Adresse 4 6) im niedrigsten bit des Divisors erscheinen.

Antworten auf die Fragen zu Programmteil 2

a) Die 9_{10} muß im ersten Teil 4_{10}-mal verschoben werden, also steht R3 auf 4_{10}. Da der Programmteil 2 erst durch ein N-Flag beendet wird, sind 5 Durchläufe notwendig.

b) Da der Divisor durch die Verschiebungen im ersten Teil in seiner Wertigkeit verändert wurde, muß der Quotient dieser Wertigkeit angepaßt werden.

Antworten auf die Fragen zu Programmteil 3

1. Eine Subtraktion im parallelen Subtraktionspfad würde nicht im Quotientenregister registriert.

2.

				2^{-1}	2^{-2}	2^{-3}	2^{-4}

nicht belegt

Lösungen zu Experiment 10

1. Nach dem ersten Anruf des Multiplikationsprogramms (Adr. 4 7) würde der Rücksprung zur Adresse F 1 erfolgen und nicht wie im Programm verlangt zur Adresse 4 9.

2. a)

C-Flag		
1	0 0 1 1 0 0 1 1	1 0 1 1 1 0 0 0

b)

C-Flag		
1	0 0 1 1 0 0 1 0	1 0 1 1 1 0 0 1

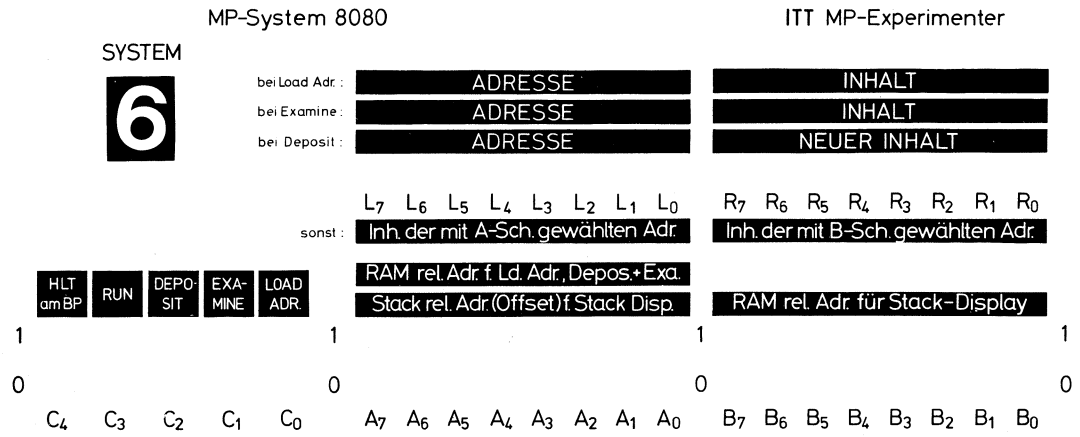
Lösungen zu Experiment 12

1. Da das Unterprogramm nicht geändert werden soll, müssen 2 Faktoradressen angegeben werden. Im Sonderfall des Quadrierens sind beide Faktoren, also auch beide Adressen gleich.

2. a^2: linke Hälfte in B 1
   rechte Hälfte in B 0

   b^2: linke Hälfte in B 3
   rechte Hälfte in B 2

Experiment 13: Handhabung des Monitorprogramms



Vorbereitung:
- Schablone 6 auflegen
- SYSTEM-Schalter auf 6 einstellen
- Alle Schalter auf Null stellen
- RESET drücken

Mit dem Monitorprogramm besteht die Möglichkeit:
- die Registerinhalte im MP 8080 und den Stand des Programmzählers sowie die Inhalte aller Adressen im RAM sichtbar zu machen.
- ähnlich wie im System 5 mit den Schaltern C_2 bis C_0 Daten zu laden und mit EXAMINE zu kontrollieren.

Das Monitorprogramm ist im ROM untergebracht und wird durch Betätigung der RESET-Taste gestartet.
Im Gegensatz zum hypothetischen Rechner, bei dem ein ähnlicher Monitor verwendet wurde, ist es im System 6 nicht mehr möglich, das Anwenderprogramm und den Monitor gleichzeitig zu betreiben. Wenn hier das Anwenderprogramm läuft, ist der Monitor außer Betrieb, d.h., eine Betätigung der Schalter hat keinen sichtbaren Einfluß mehr auf das System. Beim hypothetischen Rechner wurde über das Simulationsprogramm nach jedem Befehl das Monitorprogramm angerufen, so daß jeder Befehl entsprechend sichtbar gemacht werden konnte. In einem Zeitmultiplex-Verfahren wurden Anwender- und Monitorprogramm ständig gemeinsam benutzt. Dies hatte natürlich zur Folge, daß die Rechengeschwindigkeit des hypothetischen Rechners einige hundertmal langsamer war als die im System 6.
Wenn also über den RUN-Schalter der Rechner in das Anwenderprogramm gelangt, wird das Monitorprogramm – dieses liegt im ROM, das Anwenderprogramm liegt im RAM – außer Funktion gesetzt. Es gibt 2 Möglichkeiten, zurück in das Monitorprogramm zu kommen:
- Durch Drücken der RESET-Taste. Dadurch wird der Programmzähler wieder auf die Ausgangsadresse 0 0 0 0 gebracht.
- Durch einen RST 2-Befehl (RESTART) mit dem Code D 7.

Hierzu einige Erläuterungen:
Im MP-System sind ein 1-k-ROM und ein 1/4-k-RAM enthalten. Bedingt durch den 16-bit-Adreßbus muß jeder Speicherplatz mit 2 Byte (16 bit) adressiert werden. Für das ROM sind dabei die Adressen von $0\ 0\ 0\ 0_{16}$ bis $0\ 3\ F\ F$ festgelegt, für das RAM die Adressen $0\ 4\ 0\ 0_{16}$ bis $0\ 4\ F\ F_{16}$. Die Adressen von $0\ 5\ 0\ 0_{16}$ bis $F\ F\ F\ F_{16}$ werden nicht ausgenutzt. Wenn die RESET-Taste betätigt wird und somit der Befehlszähler die Anfangsadresse 0 0 0 0 (ROM) anwählt, gelangt der Rechner in das Monitorprogramm. Er verbleibt so lange im Monitorprogramm, **wie der RUN-Schalter auf Null steht**. Jetzt besteht die Möglichkeit, Daten in das RAM zu laden und entsprechend zu überprüfen.
Mit den Schaltern A_0 bis A_7 kann die Anfangsadresse für ein Programm bestimmt werden, die dann über LOAD-ADR. geladen werden kann. Hierbei ist zu berücksichtigen, daß bei einer Einstellung A_0 bis A_7 von z.B. 0 0 in Wirklichkeit die Adresse 0 4 0 0 angesprochen

wird. Dies wird auch durch die Bezeichnung auf der Schablone „RAM rel. Adr. f. Ld. Adr. Depos. + Exa." ausgedrückt, was besagt, daß sich die Adreßeingabe immer auf die Anfangsadresse 0 4 0 0 des RAMs bezieht.

Beispiel:

absolute Adresse	relative Adresse	Eingabe A_7 bis A_0
0 0 0 0 0 0 0 1 . . . 0 3 F F	nicht über die Eingabe adressierbar	ROM-Bereich
0 4 0 0 0 4 0 1 0 4 0 2 . . . 0 4 F D 0 4 F E 0 4 F F	0 0 0 1 0 2 F D F E F F	0 0 0 1 0 2 RAM-Bereich F D F E F F
0 5 0 0 . . F F F F	in diesem System nicht benutzt bei Systemerweiterung verfügbar	

Laden Sie folgende Daten:

relative Adresse	Inhalt
0 0	8 1
0 1	C 3
0 2	E 7
0 3	F F
0 4	7 E
0 5	3 C
0 6	1 8

Eingabeablauf:
- Stellen Sie alle Schalter auf Null
- RESET-Taste drücken
- LOAD-ADR. takten
- Mit DEPOSIT die Daten (Inhalt) eingeben.

Während DEPOSIT gleich 1 ist, erscheint an der Lampenreihe L_0 bis L_7 die relative Adresse, in R_0 bis R_7 die am A-Schalter eingestellten Daten. Die Adresse wird auch hier beim Takten mit DEPOSIT automatisch um 1 erhöht.
Nach erneutem Laden der Adresse 0 0 können die Speicherinhalte mit EXAMINE kontrolliert werden.
Die eingegebenen Daten können Sie auch mit den B-Schaltern überprüfen. Abgesehen von den Schaltern C_4 bis C_3, die immer noch auf 0 stehen, kann bei C_2 bis C_0 gleich 0 mit den B-Schaltern eine relative Adresse eingestellt werden, deren Inhalt in R_0 bis R_7 angezeigt wird. Überprüfen Sie auch auf diese Weise die geladenen Daten. Auf die Funktion der A-Schalter bei C_2 bis C_0 gleich 0 kommen wir noch zu sprechen.

Das Monitorprogramm wird verlassen, wenn über RUN gleich 1 das Anwenderprogramm angerufen wird.
Laden Sie noch folgendes Programm. Die Bedeutung dieses Programms ist zunächst nebensächlich.

rel. Adresse	Inhalt
0 0	3 A
0 1	0 0
0 2	0 0
0 3	C 6
0 4	4 8
0 5	3 C
0 6	7 6

Überprüfen Sie die Eingabe mit den *B*-Schaltern. Der Inhalt 7 6 in Adresse 0 6 ist ein HALT-Befehl. Wenn Sie den RUN-Schalter auf 1 stellen, springt der Rechner aus dem Monitor- in das eingegebene Programm und arbeitet es ab. Er bleibt bei der rel. Adresse 0 6 stehen. Auch wenn Sie jetzt den RUN-Schalter auf 0 zurückstellen, sind alle Schalter außer Funktion, da der Rechner nicht automatisch in das Monitorprogramm zurückspringt. Durch Drücken der RESET-Taste kann das Monitorprogramm wieder eingeschaltet werden. Der RUN-Schalter muß dabei auf 0 stehen, da sonst nach dem Loslassen der RESET-Taste sofort ein neuer Programmdurchlauf erfolgt.
Bisher haben wir das Monitorprogramm nur zur Eingabe und Kontrolle von Daten benutzt. Nachfolgend sollen nun Registerinhalte angezeigt werden.
Laden Sie folgendes Programm:
(RESET drücken und rel. Adresse 0 0 mit LOAD-ADR. zuerst eingeben)

rel. Adresse	Inhalt	Befehl	Kommentar
0 0	3 E	MVI A, A A	Lade Akku mit A A
0 1	A A		
0 2	0 6	MVI B, 8 1	Lade Reg. B mit 8 1
0 3	8 1		
0 4	0 E	MVI C, C 3	Lade Reg. C mit C 3
0 5	C 3		
0 6	1 6	MVI D, E 7	Lade Reg. D mit E 7
0 7	E 7		
0 8	1 E	MVI E, F F	Lade Reg. E mit F F
0 9	F F		
0 A	2 6	MVI H, 7 E	Lade Reg. H mit 7 E
0 B	7 E		
0 C	2 E	MVI L, 3 C	Lade Reg. L mit 3 C
0 D	3 C		
0 E	D 7	RST 2	Restart 2
0 F	7 6	HLT	Halt

Nach der Eingabe wird erst Reset gedrückt und dann wird RUN auf 1 gestellt!

Anmerkungen:
Die Abkürzung MVI heißt **M**ove **I**mmediate. Mit diesem Befehl erfolgt ein Datenfluß in das jeweils angegebene Register. Durch die Immediate-Adressierung sind dies die Daten des zweiten Befehlsbytes.

In Adresse 0 E ist der Befehl RST 2 gespeichert. Dieser Befehl ruft das Monitorprogramm an (auf Einzelheiten wird später eingegangen), das in diesem Zusammenhang als Unterprogramm angesehen werden kann. Bei Adresse 0 F ist das Anwenderprogramm zu Ende. Überprüfen Sie das Programm über EXAMINE. Wenn Sie jetzt bei C_4 = 0 den RUN-Schalter auf 1 stellen, wird das Programm bis zur rel. Adresse 0 F abgearbeitet, und das Monitorprogramm ist wieder außer Betrieb. Stellen Sie jetzt den RUN-Schalter wieder auf 0

und den Schalter C_4 (HLT an BP) auf 1. Drücken Sie jetzt RESET (Monitor in Betrieb). Starten Sie das Programm erneut mit RUN. Bedingt durch $C_4 = 1$ wird jetzt bei Adresse 0 E durch den Befehl RST 2 das Anwenderprogramm **unterbrochen** und das Monitorprogramm aufgerufen. Dies wird auch durch die Schalterbezeichnung von C_4 mit HLT an BP (Halt am Unterbrechungspunkt) ausgedrückt.
Mit den *B*-Schaltern können, wie vorher beschrieben, die Inhalte aller relativen Adressen aufgerufen werden (Anzeige in R_0 bis R_7).
Wenn durch den Befehl RST 2 das Monitorprogramm angerufen wird, was einer **Programmunterbrechung** entspricht, müssen die Inhalte aller Register incl. Flag-Register sowie die Stellung des Programmzählers abgespeichert werden. Nur so ist sichergestellt, daß bei einer Fortführung des Anwenderprogramms wieder auf die ursprünglichen Daten zurückgegriffen werden kann. Das Monitorprogramm speichert diese Daten in einem Stack (RAM), dessen höchste Adresse F D (relativ) ist. Die Reihenfolge des Abspeicherns liegt durch den Monitor fest und wird in der nachfolgenden Tabelle dargestellt:

absolute Adresse	relative Adresse	Inhalt
0 4 F 4	F 4	Inhalt Register L
0 4 F 5	F 5	Inhalt Register H
0 4 F 6	F 6	Inhalt Register E
0 4 F 7	F 7	Inhalt Register D
0 4 F 8	F 8	Inhalt Register C
0 4 F 9	F 9	Inhalt Register B
0 4 F A	F A	Inhalt Register F (Flag-Register)
0 4 F B	F B	Inhalt Register A (Akkumulator)
0 4 F C	F C	Programmzähler (rechte Hälfte)
0 4 F D	F D	Programmzäähler (linke Hälfte)

Sie können den Inhalt der Register und des Programmzählers dadurch überprüfen, daß Sie die *B*-Schalter auf die relative Adresse einstellen. Da der Programmzähler 16 bit aufweist, werden zur Abspeicherung 2 Bytes benötigt. Bei dem vorgegebenen Programm muß, da über RST 2 das Programm bei Adresse 0 E unterbrochen und bei 0 F wieder fortgesetzt werden muß, der Programmzähler auf Adresse 0 F (relativ) bzw. 0 4 0 F (absolut) stehen. Der Inhalt des Stacks läßt sich auch mit den *A*-Schaltern überprüfen. Durch die feste Anzahl der Register wird für das Abspeichern immer ein Stack mit 10 Plätzen benötigt. Nach dem Abspeichervorgang steht der Stack-Pointer immer auf rel. F 4. Damit kann die rel. Adresse F 4 als Bezugspunkt für den Stack herangezogen werden. Während die Adressierung mit dem *B*-Schalter immer auf die rel. Adresse 0 0 bezogen ist, ist bei den *A*-Schaltern der Bezugspunkt der Stack-Pointer und damit die rel. Adresse F 4. Aus diesem Grunde auch die Bezeichnung „Stack rel. Adr. (Offset) f. Stack Disp.". Um z.B. mit den *B*-Schaltern den Inhalt vom L-Register anzuzeigen, muß die Adresse F 4 eingestellt werden. Die gleiche Anzeige (in L_0 bis L_7) ergibt sich bei der *A*-Schalterstellung 0 0, da F 4 ja die Bezugsadresse ist. Der Akku-Inhalt wird angezeigt, wenn Stack-relative Adresse 0 7 an A_7 bis A_0 eingestellt wird.

absolute Adresse	relative Adresse (*B*-Schalter)	Stack-relative Adresse (*A*-Schalter)	Inhalt
0 4 F 4	F 4	0 0	Inhalt Register L
0 4 F 5	F 5	0 1	Inhalt Register H
0 4 F 6	F 6	0 2	Inhalt Register E
0 4 F 7	F 7	0 3	Inhalt Register D
0 4 F 8	F 8	0 4	Inhalt Register C
0 4 F 9	F 9	0 5	Inhalt Register B
0 4 F A	F A	0 6	Inhalt Register F (Flags)
0 4 F B	F B	0 7	Inhalt Register A (Akku)
0 4 F C	F C	0 8	Programmzähler (rechte Hälfte)
0 4 F D	F D	0 9	Programmzähler (linke Hälfte)

Die Stack-relative Adresse wird auch als Stack-Offset bezeichnet.
Überprüfen Sie mit dem *A*-Schalter die Registerinhalte und den Programmzähler.
Während der hier beschriebenen Vorgänge lief ständig das Monitorprogramm. Wenn der *A*-Schalter HLT am BP auf 0 zurückgestellt wird, wird das Anwenderprogramm bei der rel. Adresse 0 F fortgesetzt, das Monitorprogramm ist nicht mehr wirksam (keine Auswirkung der Schalter). Um den Monitor wieder in Betrieb zu setzen, muß RUN auf 0 und RESET gedrückt werden.
Im nachfolgenden Programm werden mehrere Monitoranrufe durch RST 2 durchgeführt:

relative Adresse	Inhalt	Befehl	Kommentar
0 0	3 E	MVI A, 0 0	Lade Akku mit 0 0
0 1	0 0		
0 2	D 7	RST 2	Restart
0 3	3 C	INR A	Incrementiere Akku
0 4	D 7	RST 2	Restart
0 5	6 F	MOV L, A	(A)→L
0 6	D 7	RST 2	
0 7	3 C	INR A	
0 8	D 7	RST 2	
0 9	6 7	MOV H, A	(A)→H
0 A	D 7	RST 2	
0 B	3 C	INR A	
0 C	D 7	RST 2	
0 D	5 F	MOV E, A	(A)→E
0 E	D 7	RST 2	
0 F	C 3	JMP 0 4 0 0	Rücksprung absolute Adresse
1 0	0 0		
1 1	0 4		

Überprüfen Sie das eingegebene Programm mit EXAMINE.
Schalten Sie bei C_4 = 0 RUN auf 1. Am Ende des Anwenderprogramms erfolgt ein unbedingter Sprung zum Programmanfang (JMP 0 4 0 0). Da der Schalter HLT am BP auf 0 steht, wird dieses Programm ständig durchlaufen. Jedesmal, wenn der Befehl RST 2 angerufen wird, erfolgt ein Sprung ins Monitorprogramm mit anschließendem Rücksprung ins Anwenderprogramm. Wie beim hypothetischen Rechner ist es jetzt möglich, den Programmablauf sichtbar zu machen.
Wird im laufenden Programm der Schalter HLT am BP auf 1 geschaltet, wird das Anwenderprogramm an einem der RST 2-Befehle angehalten. Analog zum SINGLE-STEP-Betrieb im hypothetischen Rechner kann jetzt durch Takten des RUN-Schalters das Programm jeweils bis zum nächsten RST 2-Befehl „weitergeschaltet" werden.
Takten Sie mit RUN bei C_4 = 1 das Programm durch. Der Programmzähler muß dabei der Reihe nach die relativen Adressen 0 3, 0 5, 0 7, 0 9, 0 B, 0 D und 0 F zeigen (rechte Hälfte des Programmzählers PC low).
Ändern Sie den Inhalt der rel. Adresse 0 F in 7 6 (HLT) um.

Anmerkung:
Beim Monitoranruf über RST 2 wird nur der Programmteil im ROM durchlaufen, der eine **Anzeige** der verschiedenen Register- und Speicherinhalte ermöglicht. Zum Laden neuer Daten muß der Monitor mit RESET auf die Anfangsadresse 0 0 0 0 gebracht werden. Der RUN-Schalter muß hierbei unbedingt auf 0 stehen, da sonst das Programm sofort neu abgearbeitet wird.

Wenn Sie jetzt bei C_4 = 0 RUN betätigen, durchläuft der Rechner das Anwenderprogramm und bleibt bei Adresse 0 F stehen. Obschon im Anwenderprogramm das Monitorprogramm 7mal angerufen wird, ist aufgrund der hohen Arbeitsgeschwindigkeit in der Anzeige keine Änderung zu erkennen.
Bei Adresse 0 F erfolgt kein Rücksprung ins Monitorprogramm. Stellen Sie C_4 = 1 und RUN = 0 ein und betätigen Sie RESET. Der Programmzähler muß jetzt auf rel. 0 0 stehen und

das Monitorprogramm in Funktion sein. Jetzt können Sie mit RUN das Programm wieder von RST- zu RST-Befehl durchtakten. Nach dem letzten RST-Befehl wird das Programm mit dem HLT-Befehl in Adresse 0 F fortgesetzt, der Monitor ist außer Betrieb und kann nur über RESET wieder gestartet werden.
Das hier besprochene Monitorprogramm wurde nach didaktischen Gesichtspunkten erstellt, damit der Lernende den Arbeitsablauf des MP möglichst genau verfolgen kann.
Zum Schluß sei noch bemerkt, daß die Handhabung des Monitors absolute Voraussetzung für die Durchführung von Experimenten im System 6 ist.

Experiment 14: Einzelgenauigkeits-Datentransferbefehle

Im nachfolgenden Experiment sollen die Befehle der ersten Gruppe zur Anwendung gebracht werden.
Um die einzelnen Vorgänge besser verfolgen zu können, ist nach jedem Befehl ein Monitoranruf (RST 2) eingebaut.
Laden Sie zunächst das Programm:

relative Adresse	Inhalt	Befehl	Kommentar
0 0	0 6	MVI B, 0 4	} Registerpaar B mit 0 4 8 0 laden
0 1	0 4		
0 2	D 7	RST 2	
0 3	0 E	MVI C, 8 0	
0 4	8 0		
0 5	D 7	RST 2	
0 6	3 E	MVI A, C 3	} Akku mit C 3 laden
0 7	C 3		
0 8	D 7	RST 2	
0 9	0 2	STAX B	(Akku) nach @ BC
0 A	D 7	RST 2	
0 B	0 C	INR C	Increment (C-Register)
0 C	D 7	RST 2	
0 D	6 0	MOV H, B	(B)→H
0 E	D 7	RST 2	
0 F	6 9	MOV L, C	(C)→L
1 0	D 7	RST 2	
1 1	3 E	MVI A, F F	} Akku mit F F laden
1 2	F F		
1 3	D 7	RST 2	
1 4	7 7	MOV M, A	(Akku)→ @ HL
1 5	D 7	RST 2	
1 6	2 D	DCR L	Decrement (L-Register)
1 7	D 7	RST 2	
1 8	5 4	MOV D, H	(H)→D
1 9	D 7	RST 2	
1 A	5 D	MOV E, L	(L)→E
1 B	D 7	RST 2	
1 C	1 A	LDAX D	(@ DE)→A
1 D	D 7	RST 2	
1 E	3 A	LDA 0 4 8 2	(0 4 8 2)→A
1 F	8 2		
2 0	0 4		
2 1	D 7	RST 2	
2 2	7 6	HLT	
.			
.			
.			
8 2	7 E		

Überprüfen Sie das Programm mit EXAMINE. Stellen Sie den HLT am BP-Schalter auf 1, und takten Sie das Programm mit dem RUN-Schalter durch.

Aufgaben:

1. Bei welchem Programmschritt kann der **Endzustand** der Register mit dem Monitorprogramm geprüft werden?

2. Geben Sie die Register- und Speicherinhalte am Programmende an!

Experiment 15: Doppelgenauigkeits-Datentransferbefehle

Die gleichen Datenbewegungen wie in Experiment 14 sollen mit Hilfe von Befehlen aus der zweiten Gruppe durchgeführt werden.
Laden Sie dazu das folgende Programm:

relative Adresse	Inhalt	Befehl	Kommentar
0 0	0 1	LXI B	} lade Registerpaar B mit 0 4 8 0
0 1	8 0		
0 2	0 4		
0 3	D 7	RST 2	
0 4	3 E	MVI A, C 3	} Akku mit C 3 laden
0 5	C 3		
0 6	D 7	RST 2	
0 7	0 2	STAX @ B	(Akku) indirekt über B abspeichern
0 8	D 7	RST 2	
0 9	2 A	LHLD	} Registerpaar H mit 0 4 0 1 laden
0 A	0 1		
0 B	0 4		
0 C	D 7	RST 2	
0 D	2 C	INR L	(Register L) incrementieren
0 E	D 7	RST 2	
0 F	3 E	MVI A, F F	} Akku mit F F laden
1 0	F F		
1 1	D 7	RST 2	
1 2	7 7	MOV M, A	(Akku)→(@ Registerpaar HL)
1 3	D 7	RST 2	
1 4	2 D	DCR L	(Register L) decrementieren
1 5	D 7	RST 2	
1 6	E B	XCHG	vertausche die Inhalte der Registerpaare H und D
1 7	D 7	RST 2	
1 8	1 A	LDAX D	Akku mit (@ Registerpaar DE) laden
1 9	D 7	RST 2	
1 A	3 A	LDA 0 4 8 2	} Akku mit Inhalt der relativen Adresse 8 2 laden
1 B	8 2		
1 C	0 4		
1 D	D 7	RST 2	
1 E	7 6	HLT	
.			
.			
.			
8 2	7 E		

Aufgabe:
Geben Sie die Inhalte aller Register und des Akkumulators an. (Während des Programmablaufs nicht am Ende!)

Experiment 16: IN- und OUT-Befehle

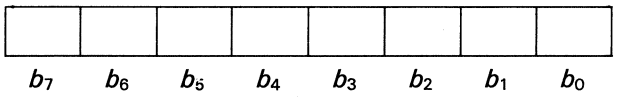
Laden Sie nachfolgendes Programm:

relative Adresse	Inhalt	Befehl	Kommentar
0 0	D B	IN ASHALT	(*A*-Schalter)→Akku
0 1	0 2		
0 2	D 3	OUT LLAMPE	(Akku)→Lampen L_7 bis L_0
0 3	0 2		
0 4	D B	IN BSHALT	(*B*-Schalter)→Akku
0 5	0 1		
0 6	D 3	OUT RLAMPE	(Akku)→Lampen R_7 bis R_0
0 7	0 1		
0 8	D B	IN CSHALT	(*C*-Schalter und Systemschalter)→Akku
0 9	0 4		
0 A	D 3	OUT RLAMPE	(Akku)→Lampen R_7 bis R_0
0 B	0 1		
0 C	C 3	JMP 0 4 0 0	Sprung zur absoluten Adresse
0 D	0 0		0 4 0 0 ≙ relative Adresse 0 0
0 E	0 4		

Mit diesem Schleifenprogramm ist es möglich, alle Schalter des MP-Experimentiersystems zur Anzeige zu bringen. Da kein RST 2-Befehl vorhanden ist, wird das Monitorprogramm nicht durchlaufen, d.h., alle Schalter (bis auf RESET) können beliebig verstellt werden. Über den Befehl IN CSHALT wird in R_7 bis R_0 der jeweilige Zustand der *C*-Schalter sowie des Systemschalters angezeigt (bei B_7 bis B_0 gleich 0).

Aufgaben:

1. Ermitteln Sie die bit-Zuordnung der *C*-Schalter und des Systemschalters:



2. Wie werden die am Systemschalter eingestellten Zahlen wiedergegeben?

3. Wie kann diese Programmschleife wieder verlassen werden?

Experiment 17: Einzelgenauigkeitsadditionsbefehle

Laden Sie nachfolgendes Programm:

relative Adresse	Inhalt	Befehl	Kommentar
0 0	0 6	MVI B	Reg. B mit 1 0 laden
0 1	1 0		
0 2	D 7	RST 2	
0 3	D B	IN BSHALT	(*B*-Schalter)→Akku
0 4	0 1		
0 5	D 7	RST 2	
0 6	8 0	ADD B	(Reg. B) + (Akku)
0 7	D 7	RST 2	
0 8	C 6	ADI	(Akku) + 0 F
0 9	0 F		
0 A	D 7	RST 2	
0 B	4 F	MOV C, A	(Akku)→ Reg. C
0 C	D 7	RST 2	
0 D	D B	IN ASHALT	(*A*-Schalter)→Akku
0 E	0 2		
0 F	D 7	RST 2	
1 0	8 9	ADC C	(Akku) + (Reg. C) + (C-Flag)
1 1	D 7	RST 2	
1 2	C E	ACI 1 1	(Akku) + 1 1 + (C-Flag)
1 3	1 1		
1 4	D 7	RST 2	
1 5	C 3	JMP 0 4 0 0	Sprung zur Anfangsadresse
1 6	0 0		
1 7	0 4		

Durch die RST 2-Befehle kann das Programm im SINGLE-STEP-Betrieb gefahren werden, so daß Sie jeden Befehl verfolgen können.

Aufgaben:

1. Geben Sie den Zustand des C-Flags bei der relativen Adresse 0 A an, wenn die *B*-Schalter auf E F stehen. Geben Sie außerdem den Akku-Inhalt an!

2. Geben Sie den Zustand des C-Flags und des Akkus bei der relativen Adresse 1 1 an, wenn die *A*-Schalter auf 0 7 stehen (*B*-Schalter auf E F)!

3. Wie lautet das Endergebnis bei der relativen Adresse 1 4?

Führen Sie selbst noch weitere Beispiele durch. Berücksichtigen Sie, daß bei dem Befehl ADD zwar ein Übertrag entstehen kann, dieser aber bei dem Befehl ADI nicht mehr berücksichtigt wird.

Experiment 18: Einzelgenauigkeitssubtraktionsbefehle

Laden Sie nachfolgendes Programm:

relative Adresse	Inhalt	Befehl	Kommentar
0 0	0 6	MVI B	Reg. B mit 0 laden
0 1	4 0		
0 2	D 7	RST 2	
0 3	D B	IN BSHALT	(*B*-Schalter)→Akku
0 4	0 1		
0 5	D 7	RST 2	
0 6	9 0	SUB B	(Akku)→(Reg. B)
0 7	D 7	RST 2	
0 8	D 6	SUI 1 0	(Akku)→1 0
0 9	1 0		
0 A	D 7	RST 2	
0 B	4 F	MOV C, A	(Akku)→Reg. C
0 C	D 7	RST 2	
0 D	D B	IN ASHALT	(*A*-Schalter)→Akku
0 E	0 2		
0 F	D 7	RST 2	
1 0	9 9	SBB C	(Akku) – (Reg. C) – (C-Flag)
1 1	D 7	RST 2	
1 2	D E	SBI 2 0	(Akku) – 2 0 – (C-Flag)
1 3	2 0		
1 4	D 7	RST 2	
1 5	C 3	JMP 0 4 0 0	Rücksprung zur relativen
1 6	0 0		Adresse 0 0
1 7	0 4		

Überprüfen Sie mit EXAMINE, ob Sie das Programm richtig eingegeben haben. Stellen Sie den Schalter C_4 „HLT am BP" auf 1, und takten Sie das Programm mit dem RUN-Schalter (vorher RESET drücken). Wählen Sie als Daten für den Befehl IN BSHALT 8 0_{16} und für den Befehl IN ASHALT C 0_{16}. Überprüfen Sie den Zustand des Akkumulators und des C- sowie N-Flags entsprechend nachfolgender Tabelle bei den jeweiligen relativen Adressen.

relative Adresse	Inhalt Akku binär	Inhalt Akku hex.	Flags C-Flag	Flags N-Flag
0 7	0 1 0 0 0 0 0 0	4 0	0	0
0 A	0 0 1 1 0 0 0 0	3 0	0	0
1 1	1 0 0 1 0 0 0 0	9 0	0	1
1 4	0 1 1 1 0 0 0 0	7 0	0	0

Die angegebenen relativen Adressen entsprechen den RST 2-Befehlen, die einem Subtraktionsbefehl folgen. Bei den hier gewählten Daten für die Befehle IN BSHALT mit 8 0_{16} und IN ASHALT mit C 0_{16} ergibt sich nachfolgender Rechengang:

```
  1 0 0 0 0 0 0 0
- 0 1 0 0 0 0 0 0
  1 0 0 0 0 0 0 –    Entlehnung
  0 1 0 0 0 0 0 0    →(Akku) bei rel. Adr. 0 7
- 0 0 0 1 0 0 0 0
  0 1 1 0 0 0 0 –    Entlehnung
  0 0 1 1 0 0 0 0    →(Akku) bei rel. Adr. 0 A
```

```
  1 1 0 0 0 0 0 0
- 0 0 1 1 0 0 0 0
  0 1 1 0 0 0 0 –     Entlehnung
  1 0 0 1 0 0 0 0   →(Akku) bei rel. Adr. 1 1
- 0 0 1 0 0 0 0 0
  1 1 0 0 0 0 0 –     Entlehnung
  0 1 1 1 0 0 0 0   →(Akku) bei rel. Adr. 1 4
```

Aufgaben:

1. Warum ist das N-Flag bei der relativen Adresse 1 1 gleich 1?

2. Führen Sie das Programm mit folgenden Daten durch:

IN BSHALT: 3 0_{16}
IN ASHALT: 0 9_{16}

Tragen Sie die Daten in nachfolgende Tabelle ein.

relative Adresse	Inhalt Akku binär	Inhalt Akku hex.	Flags C-Flag	Flags N-Flag
0 7				
0 A				
1 1				
1 4				

3. Begründen Sie, warum bei dem vorherigen Beispiel in der relativen Adresse 1 1 der Akku-Inhalt 2 9_{16} ist!

Experiment 19: CMP- und CPI-Befehle

Laden Sie nachfolgendes Programm:

relative Adresse	Inhalt	Befehl	Kommentar
0 0	3 E	MVI A	Akku mit 7 4 laden
0 1	7 4		
0 2	0 6	MVI B	Reg. B mit 9 5 laden
0 3	9 5		
0 4	B 8	CMP B	Vergleich (Akku) mit (Reg. B)
0 5	D 7	RST 2	
0 6	F E	CPI 6 8	Vergleich (Akku) mit 6 8
0 7	6 8		
0 8	D 7	RST 2	
0 9	F E	CPI 7 4	Vergleich (Akku) mit 7 4
0 A	7 4		
0 B	D 7	RST 2	
0 C	F E	CPI 7 5	Vergleich (Akku) mit 7 5
0 D	7 5		
0 E	D 7	RST 2	
0 F	C 3	JMP 0 4 0 0	
1 0	0 0		
1 1	0 4		

In nachfolgender Tabelle ist der Zustand der Flags nach den jeweiligen Compare-Befehlen angegeben.

relative Adresse	N-Flag	Z-Flag	0	H-Flag	0	P-Flag	1	C-Flag
0 5	1	0		0		0		1
0 8	0	0		0		1		0
0 B	0	1		1		1		0
0 E	1	0		0		1		1

Wie Sie der Tabelle entnehmen können, werden mit diesem Programm alle Flags beeinflußt.

Relative Adresse 0 5:
Hier ist durch den Befehl CMP B die Subtraktion 7 4_{16} – 9 5_{16} erfolgt. Da der Subtrahend größer als der Minuend ist, entsteht C-Flag = 1. Da ein negatives Ergebnis vorliegt, ist auch das N-Flag = 1, das Z-Flag muß 0 sein, da das Ergebnis **nicht** Null ist.
Das Half-Carry-Flag, auch Hilfs-Carry-Flag genannt, hat den Zustand 0. Dies wird deutlich, wenn wir die Subtraktion durch Zweierkomplementaddition durchführen:

```
  0 1 1 1  0 1 0 0
+ 0 1 1 0  1 0 1 1    Zweierkomplement von 9 5₁₆
  1 1 0[0] 0 0 0 –    Übertrag
  1 1 0 1  1 1 1 1    ≙ – 2 1₁₆
```

Aus dieser Rechnung ist zu erkennen, daß zwischen der 4. und 5. Stelle kein Übertrag entsteht. Damit spricht auch das H-Flag nicht an.
Das P-Flag spricht dann an, wenn im Ergebnis die Anzahl der mit 1 belegten Stellen **gerade** ist. Dies ist bei diesem Ergebnis nicht der Fall, also ist P = 0.

Relative Adresse 0 8:
Der Befehl CPI 6 8 bewirkt die Subtraktion 7 4_{16} – 6 8_{16}. Da das Ergebnis weder 0 noch negativ ist, sind Z-Flag, N-Flag und C-Flag 0. Die Subtraktion über Zweierkomplement-addition ergibt:

```
  0 1 1 1  0 1 0 0
+ 1 0 0 1  1 0 0 0     Zweierkomplement von 6 8₁₆
  1 1 1 0  0 0 0 –     Übertrag
1 0 0 0 0  1 1 0 0
```

Es entsteht ebenfalls kein Übertrag zwischen der 4. und 5. Stelle. Damit ist das H-Flag gleich 0. Das Ergebnis weist eine gerade Anzahl von Einsen auf. Damit ist das P-Flag gleich 1.

Relative Adresse 0 B:
Mit dem Befehl CPI 7 4 wird die Subtraktion 7 4_{16} – 7 4_{16} ausgelöst. Da das Ergebnis jetzt Null sein muß, ist das Z-Flag gleich 1, während das C-Flag und das N-Flag gleich 0 sind. Der Zustand von H- und P-Flag läßt sich wieder durch eine Addition in Zweierkomplementarithmetik erklären:

```
  0 1 1 1  0 1 0 0
+ 1 0 0 0  1 1 0 0     Zweierkomplement von 7 4₁₆
  1 1 1 1  1 0 0 –     Übertrag
1 0 0 0 0  0 0 0 0
```

Relative Adresse 0 E:
Der Befehl CPI 7 5 bewirkt die Subtraktion 7 4_{16} – 7 5_{16}.
Das negative Ergebnis bringt das N- und C-Flag auf 1, das Z-Flag muß 0 sein. Die Subtraktion über das Zweierkomplement ergibt:

```
  0 1 1 1  0 1 0 0
+ 1 0 0 0  1 0 1 1     Zweierkomplement von 7 5₁₆
  0 0 0 0  0 0 0 –     Übertrag
0 1 1 1 1  1 1 1 1
```

Das H-Flag ist 0, das P-Flag 1, da 8 Stellen mit 1 belegt sind.

Experiment 20: DAD-Befehle

Laden Sie nachfolgendes Programm:

relative Adresse	Inhalt	Befehl	Kommentar
0 0	0 1	LXI B, 1 2 3 4	lade Registerpaar BC
0 1	3 4		
0 2	1 2		
0 3	1 1	LXI D, 5 6 7 8	lade Registerpaar DE
0 4	7 8		
0 5	5 6		
0 6	2 1	LXI H, 9 A B C	lade Registerpaar HL
0 7	B C		
0 8	9 A		
0 9	1 9	DAD D	5 6 7 8 + 9 A B C
0 A	D 7	RST 2	
0 B	E B	XCHG	tausche Ergebnis gegen 5 6 7 8 aus
0 C	D 7	RST 2	
0 D	0 9	DAD B	1 2 3 4 + 5 6 7 8
0 E	D 7	RST 2	

Aufgabe:

Geben Sie das Ergebnis nach der Addition DAD D sowie das Ergebnis nach DAD B an!

Experiment 21: DAA-Befehl

Laden Sie nachfolgendes Programm:

relative Adresse	Inhalt	Befehl	Kommentar
0 0	D B	IN ASHALT	(*A*-Schalter)→Akku
0 1	0 2		
0 2	4 7	MOV B, A	(Akku)→Reg. B
0 3	D 7	RST 2	
0 4	D B	IN ASHALT	(*A*-Schalter)→Akku
0 5	0 2		
0 6	8 0	ADD B	(Akku) + (Reg. B)
0 7	D 7	RST 2	
0 8	2 7	DAA	Dezimalkorrektur
0 9	D 7	RST 2	

Überprüfen Sie die in Abschnitt 6.3.6 gebrachten Beispiele, indem Sie die Operanden über die IN ASHALT-Befehle eingeben.

Aufgaben:

1. Bei welchem Summanden wird nach dem ADD B-Befehl das H-Flag gleich 1?

2. Addieren Sie $09_{10} + 09_{10}$. Geben Sie an, welchen Zustand das H-Flag vor und nach dem DAA-Befehl hat!

3. Addieren Sie $07_{10} + 08_{10}$. Geben Sie auch hier den Zustand des H-Flags vor und nach dem DAA-Befehl an!

Experiment 22: Logische Befehle

Laden Sie nachfolgendes Programm:

relative Adresse	Inhalt	Befehl	Kommentar
0 0	D B	IN ASHALT	(*A*-Schalter)→Akku
0 1	0 2		
0 2	4 7	MOV B, A	(Akku)→(Reg. B)
0 3	8 0	ADD B	(Akku) + (Reg. B)
0 4	D 7	RST 2	
0 5	E 6	ANI 0 0	(Akku) Λ 0 0
0 6	0 0		
0 7	D 7	RST 2	
0 8	7 8	MOV A, B	
0 9	8 0	ADD B	
0 A	D 7	RST 2	
0 B	A 0	ANA B	(Akku) Λ (Reg. B)
0 C	D 7	RST 2	
0 D	7 8	MOV A, B	
0 E	8 0	ADD B	
0 F	D 7	RST 2	
1 0	F 6	ORI 0 F	(Akku) V 0 F
1 1	0 F		
1 2	D 7	RST 2	
1 3	7 8	MOV A, B	
1 4	8 0	ADD B	
1 5	D 7	RST 2	
1 6	B 0	ORA B	(Akku) V (Reg. B)
1 7	D 7	RST 2	
1 8	7 8	MOV A, B	
1 9	8 0	ADD B	
1 A	D 7	RST 2	
1 B	E E	XRI F 0	(Akku) $\veebar$ F 0
1 C	F 0		
1 D	D 7	RST 2	
1 E	7 8	MOV A, B	
1 F	8 0	ADD B	
2 0	D 7	RST 2	
2 1	A 8	XRA B	(Akku) $\veebar$ (Reg. B)
2 2	D 7	RST 2	
2 3	7 6	HLT	

Aufgabe:
Geben Sie mit dem Befehl IN ASHALT 8 9_{16} ein, und geben Sie in der nachfolgenden Tabelle den Zustand der Flags sowie den Akku-Inhalt nach Durchführung der angegebenen Befehle an!

relative Adresse	Befehl	Akku-Inhalt	Flags				
			N	Z	H	P	C
0 5	ANI						
0 B	ANA						
1 0	ORI						
1 6	ORA						
1 B	XRI						
2 1	XRA						

Experiment 23: Rotier- und Verschiebebefehle

Laden Sie nachfolgendes Programm:

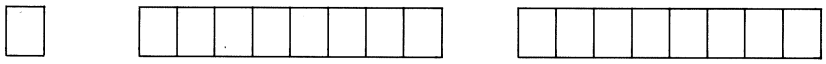
relative Adresse	Inhalt	Befehl	Kommentar
0 0	D B	IN ASHALT	(*A*-Schalter)→Akku
0 1	0 2		
0 2	6 7	MOV H, A	(Akku)→H
0 3	D B	IN BSHALT	(*B*-Schalter)→Akku
0 4	0 1		
0 5	6 F	MOV L, A	(Akku)→L
0 6	D 7	RST 2	
0 7	2 9	DAD H	(HL) + (HL)→HL
0 8	D 7	RST 2	
0 9	1 7	RAL	(Akku) über C-Flag linksschieben
0 A	D 7	RST 2	
0 B	0 7	RLC	(Akku) nach links und **in** C-Flag schieben
0 C	D 7	RST 2	
0 D	1 F	RAR	(Akku) über C-Flag rechtsschieben
0 E	D 7	RST 2	
0 F	1 F	RAR	
1 0	D 7	RST 2	
1 1	1 F	RAR	
1 2	D 7	RST 2	
1 3	0 F	RRC	(Akku) nach rechts und **in** C-Flag schieben
1 4	D 7	RST 2	
1 5	7 6	HLT	

Aufgabe:

Stellen Sie an den *A*-Schaltern 3 C_{16} und an den *B*-Schaltern E 7_{16} ein, und starten Sie das Programm. In die nachfolgende Tabelle sollen die Inhalte von Registern und C-Flag **nach** dem Abarbeiten der entsprechenden Befehle eingetragen werden!

relative Adresse 0 7 Befehl DAD H

C-Flag H-Register L-Register



relative Adresse	Befehl	C-Flag	Akku-Inhalt
0 9	RAL		
0 B	RLC		
0 D	RAR		
0 F	RAR		
1 1	RAR		
1 3	RRC		

Experiment 24: Increment- und Decrement-Befehle

Laden Sie nachfolgendes Programm:

relative Adresse	Inhalt	Befehl	Kommentar
0 0	9 7	SUB A A	Akku löschen
0 1	1 1	LXI D, 0 0 0 0	Registerpaar DE löschen
0 2	0 0		
0 3	0 0		
0 4	3 C	INR A	(Akku) incrementieren
0 5	D 7	RST 2	
0 6	3 D	DCR A	(Akku) decrementieren
0 7	D 7	RST 2	
0 8	3 D	DCR A	(Akku) decrementieren
0 9	D 7	RST 2	
0 A	1 3	INX D	(Registerpaar DE) incrementieren
0 B	D 7	RST 2	
0 C	1 B	DCX D	(Registerpaar DE) decrementieren
0 D	D 7	RST 2	
0 E	1 B	DCX D	(Registerpaar DE) decrementieren
0 F	D 7	RST 2	
1 0	2 1	LXI H, 0 4 2 0	Registerpaar HL mit 0 4 2 0 laden
1 1	2 0		
1 2	0 4		
1 3	D 7	RST 2	
1 4	3 6	MVI M, 0 0	relative Adresse 2 0 mit 0 0 laden
1 5	0 0		
1 6	D 7	RST 2	
1 7	3 4	INR M	(@ Registerpaar HL) incrementieren
1 8	D 7	RST 2	
1 9	3 5	DCR M	(@ Registerpaar HL) decrementieren
1 A	D 7	RST 2	
1 B	3 5	DCR M	(@ Registerpaar HL) decrementieren
1 C	D 7	RST 2	
1 D	C 3	JMP 0 4 0 4	Rücksprung zur relativen Adresse
1 E	0 4		0 4
1 F	0 4		

Anmerkung:
Der Kommentar „(@ Registerpaar HL) incrementieren" bedeutet, daß der Inhalt des Speicherplatzes, dessen Adresse im Registerpaar HL steht, incrementiert wird.

Testen Sie das Programm mit Schalter „HLT am BP" gleich 1.

Aufgaben:

1. Aus welchem Grunde wird nach Durchführung des Befehles DCR A in der relativen Adresse 0 6 das H-Flag gleich 1?

2. Welche Auswirkung hat der Befehl MVI M, 0 0 in den relativen Adressen 1 4 und 1 5?

Experiment 25: Sprungbefehle

Laden Sie nachfolgendes Programm:

relative Adresse	Inhalt	Befehl	Kommentar
0 0	D 7	RST 2	
0 1	2 1	LXI H, 0 4 0 0	lade Registerpaar HL mit 0 4 0 0
0 2	0 0		
0 3	0 4		
0 4	D B	IN BSHALT	(*B*-Schalter)→Akku
0 5	0 1		
0 6	D 7	RST 2	
0 7	4 7	MOV B, A	(Akku)→Reg. B
0 8	D 7	RST 2	
0 9	D B	IN BSHALT	(*B*-Schalter)→Akku
0 A	0 1		
0 B	D 7	RST 2	
0 C	8 0	ADD B	(Akku) + (Reg. B)
0 D	D 7	RST 2	
0 E	D A	JC 0 4 0 0	springe, wenn C-Flag = 1
0 F	0 0		
1 0	0 4		
1 1	D 7	RST 2	
1 2	C A	JZ 0 4 4 0	springe, wenn Z-Flag = 1
1 3	4 0		
1 4	0 4		
1 5	D 7	RST 2	
1 6	C 6	ADI 0 F	(Akku) + 0 F
1 7	0 F		
1 8	D 7	RST 2	
1 9	F A	JM 0 4 6 0	springe, wenn N-Flag = 1
1 A	6 0		
1 B	0 4		
1 C	D 7	RST 2	
1 D	E A	JPE 0 4 8 0	springe, wenn P-Flag = 1
1 E	8 0		
1 F	0 4		
2 0	D 7	RST 2	
2 1	E 9	PCHL	springe (@HL)
.	.		
.	.		
.	.		
4 0	0 4	INR B	incrementiere (Reg. B)
4 1	D 7	RST 2	
4 2	8 0	ADD B	(Akku) + (Reg. B)
4 3	D 7	RST 2	
4 4	E 2	JPO 0 4 6 0	springe, wenn P-Flag = 0
4 5	6 0		
4 6	0 4		
4 7	D 7	RST 2	
4 8	E 9	PCHL	springe (@HL)
.	.		
.	.		
.	.		
6 0	0 5	DCR B	decrementiere (Reg. B)
6 1	0 7	RST 2	
6 2	8 0	ADD B	(Akku) + (Reg. B)
6 3	D 7	RST 2	

relative Adresse	Inhalt	Befehl	Kommentar
6 4	D 2	JNC 0 4 8 0	springe, wenn C-Flag = 0
6 5	8 0		
6 6	0 4		
6 7	D 7	RST 2	
6 8	E 9	PCHL	springe (@ HL)
.	.		
.	.		
.	.		
8 0	E 6	ANI 0 E	
8 1	0 E		
8 2	D 7	RST 2	
8 3	F 2	JP 0 4 ~~8 0~~	springe, wenn N-Flag = 0
8 4	0 8		
8 5	0 4		
8 6	D 7	RST 2	
8 7	E 9	PCHL	springe (@ HL)

Überprüfen Sie die Programmeingabe mit EXAMINE.

Aufgaben:

1. Geben Sie die Sprünge an, wenn bei der relativen Adresse 0 4 die Daten F B und bei 0 9 die Daten 0 2 eingegeben werden!

2. Mit welchem Befehl erfolgt der Rücksprung zur relativen Adresse 0 0, wenn als Daten mit den *B*-Schaltern zuerst F 0 und dann 0 0 eingegeben werden?

Experiment 26: Unterprogrammanrufe und Rücksprungbefehle

Laden Sie nachfolgendes Programm:

relative Adresse	Inhalt	Befehl	Kommentar
0 0	D B	IN BSHALT	(*B*-Schalter)→Akku
0 1	0 1		
0 2	D 7	RST 2	
0 3	A 7	ANA A	setze bzw. lösche Flags
0 4	D 7	RST 2	
0 5	F 4	CP 0 4 3 0	rufe Unterprogramm an, wenn (Akku) plus
0 6	3 0		
0 7	0 4		
0 8	D 7	RST 2	
0 9	F 4	CP 0 4 3 0	rufe Unterprogramm an, wenn (Akku) plus
0 A	3 0		
0 B	0 4		
0 C	D 7	RST 2	
0 D	C 3	JMP 0 4 0 0	springe nach Adresse 0 4 0 0
0 E	0 0		
0 F	0 4		
.	.		
.	.		
.	.		
3 0	0 7	RLC	verschiebe (Akku) nach links
3 1	A 7	ANA A	
3 2	F 8	RM	return, wenn negativ
3 3	0 7	RLC	verschiebe (Akku) nach links
3 4	A 7	ANA A	
3 5	C 9	RET	zurück zum Hauptprogramm

Bei diesem Programm werden zuerst die an den *B*-Schaltern eingestellten Daten in den Akku geladen und mit dem ANA-Befehl die Flags entsprechend gesetzt bzw. gelöscht. Liegt eine positive Zahl vor, wird bei der relativen Adresse 0 5 durch den CP-Befehl das Unterprogramm angerufen. Hier wird der Akku-Inhalt um eine Stelle nach links geschoben. Dann werden die Flags wieder entsprechend gesetzt. Liegt jetzt eine negative Zahl vor, erfolgt durch den RM-Befehl ein Rücksprung zum Hauptprogramm. Bei einer positiven Zahl erfolgt nochmals eine Verschiebung nach links sowie ein Setzen der entsprechenden Flags. Der RET-Befehl bewirkt in diesem Falle einen Rücksprung zum Hauptprogramm. Hier erfolgt dann ein zweiter Unterprogrammanruf, wenn bei der relativen Adresse 0 9 eine positive Zahl vorliegt.

Aufgaben:

1. Mit den *B*-Schaltern wird die Zahl $0\ 1_{16}$ eingegeben. Um wie viele Stellen wird diese Zahl bei einem vollen Programmdurchlauf nach links verschoben?

2. Welche Auswirkung hat ein Austauschen des ANA-Befehles in der relativen Adresse 3 4 gegen einen NOP-Befehl mit dem Code $0\ 0_{16}$?

Experiment 27: RST-Befehle

Da beim MP-Experimenter keine peripheren Geräte vorhanden sind, die ein INT-Signal erzeugen können, lassen sich Interrupts nicht in einem Beispiel durchführen. Die Wirkung der RST-Befehle soll im nachfolgenden Experiment gezeigt werden. In diesem Programm werden auch Adressen im ROM-Bereich verwendet, so daß die RAM-Adressen jetzt mit ihren absoluten Werten angegeben werden. Laden Sie folgendes Programm:

absolute Adresse	Inhalt	Befehl	Kommentar
0 4 0 0	F F	RST 7	rufe Unterprogramm 0 4 1 0 an
0 4 0 1	D 7	RST 2	Monitoranruf
0 4 0 2	C 7	RST 0	rufe Monitor bei Adresse 0 0 0 0 an
.	.		(≙ RESET)
.	.		
.	.		
0 4 1 0	2 1	LXI H, 0 4 2 0	Registerpaar HL mit 0 4 2 0 laden
0 4 1 1	2 0		
0 4 1 2	0 4		
0 4 1 3	3 4	INR M	incrementiere Inhalt Adresse 0 4 2 0
0 4 1 4	C 9	RET	Rücksprung

Mit dem RST 7-Befehl wird ein Unterprogramm bei Adresse 0 0 3 8 angerufen. Hier ist ein JMP-Befehl zur Adresse 0 4 1 0 fest im ROM gespeichert. Bei der Adresse 0 4 1 0 wird Registerpaar HL mit 0 4 2 0 geladen. Der INR M-Befehl incrementiert den Inhalt der Speicheradresse, die vom Registerpaar HL adressiert wird. Mit dem RET-Befehl erfolgt ein Rücksprung zur Adresse 0 4 0 1, in der ein RST 2-Befehl steht. Damit erfolgt der bekannte Monitoranruf. Der nach dem Monitoranruf folgende Befehl RST 0 bewirkt einen Sprung zur Adresse 0 0 0 0, was einem Drücken der RESET-Taste entspricht. Das Betriebsprogramm ist so ausgelegt, daß nicht die nächstfolgende RAM-Adresse 0 4 0 3 als Rücksprungadresse übernommen wird, sondern stattdessen die Adresse 0 4 0 0, also die Anfangsadresse des RAM-Bereiches.
Testen Sie das Programm mit HLT am BP = 1, und sehen Sie sich dabei den Speicherplatz 0 4 2 0 an. Die *B*-Schalter sind hierfür auf die relative Adresse 2 0 einzustellen.

Aufgaben:

1. Warum erfolgt ein Incrementieren des Inhaltes der Adresse 0 4 2 0 nur mit jedem zweiten Takten des RUN-Schalters?

2. Ändern Sie das Programm so ab, daß der Inhalt der Adresse 0 4 2 0 durch den RST 1-Befehl incrementiert wird!

Experiment 28: PUSH- und POP-Befehle

Laden Sie nachfolgendes Programm:

absolute Adresse	Inhalt	Befehl	Kommentar
0 4 0 0	3 E	MVI A, 9 7	Akku mit 9 7 laden
0 4 0 1	9 7		
0 4 0 2	A 7	ANA A	Flags setzen
0 4 0 3	F 5	PUSH PSW	Flags und (Akku) auf Stack
0 4 0 4	9 7	SUB A	Akku löschen
0 4 0 5	F 1	POP PSW	Flags und (Akku) aus dem Stack
0 4 0 6	D 7	RST 2	zurückholen
0 4 0 7	C 3	JMP 0 4 0 0	
0 4 0 8	0 0		
0 4 0 9	0 4		

Dieses Programm lädt zunächst den Akku mit 9 7. Der Befehl ANA A verändert den Akku-Inhalt nicht, er bewirkt lediglich, daß die Flags entsprechend gesetzt werden. Mit dem Befehl PUSH PSW werden der Akku-Inhalt und der Zustand der Flags in den Stack gebracht. Wie bereits erwähnt, ist die höchste Stack-Adresse die Adresse 0 4 F D. In dieser Adresse wird durch den Befehl PUSH PSW der Inhalt des Akkus abgespeichert. Der Zustand der Flags wird in der nächstniedrigeren Adresse 0 4 F C gespeichert.
Mit dem Befehl SUB A wird der Inhalt des Akkus gelöscht sowie die Flags entsprechend neu gesetzt. Der Befehl POP PSW lädt den Akku wieder mit dem Inhalt 9 7 und setzt außerdem die Flags wieder so, wie sie nach dem Befehl ANA A waren. Der Stack-Pointer wird dabei wieder auf 0 4 F E gesetzt. Der Monitoranruf mit RST 2 bedingt wieder eine Stack-Operation, wobei die Inhalte der Speicherplätze 0 4 F D und 0 4 F C, in denen vorher Akku und Flags gespeichert waren, jetzt mit dem Stand des Programmzählers überschrieben werden. Die restlichen Stack-Plätze 0 4 F B bis 0 4 F 4 werden in bekannter Weise belegt. Da bei diesem Programm der RST 2-Befehl nach dem POP-Befehl ausgeführt wird, kann das Monitorprogramm in gewohnter Weise benutzt werden. Dieses ist bei dem nachfolgenden Programm, das die gleichen Befehle erhält, nicht mehr der Fall.
Laden Sie das nachfolgende Programm:

absolute Adresse	Inhalt	Befehl	Kommentar
0 4 0 0	3 E	MVI A, 9 7	
0 4 0 1	9 7		
0 4 0 2	A 7	ANA A	
0 4 0 3	F 5	PUSH PSW	
0 4 0 4	9 7	SUB A	
0 4 0 5	D 7	RST 2	
0 4 0 6	F 1	POP PSW	
0 4 0 7	C 3	JMP 0 4 0 0	
0 4 0 8	0 0		
0 4 0 9	0 4		

Der Unterschied gegenüber dem vorhergehenden Programm besteht darin, daß der Befehl RST 2 **vor** dem POP-Befehl ausgeführt wird. Durch den PUSH-Befehl wird der Stack-Pointer um 2 erniedrigt, so daß beim RST 2-Befehl der Stack-Pointer auf Adresse 0 4 F C steht. Damit verschiebt sich der Stack-Bereich für den Monitor um 2 Adressen. Es ergibt sich somit folgende Zuordnung:

RAM-relative Adresse	Stack-Offset	Register
F 2	0	L
F 3	1	H
F 4	2	E
F 5	3	D
F 6	4	C
F 7	5	B
F 8	6	Flags
F 9	7	Akku
F A	8	PC low
F B	9	PC high
F C	A	Flags } durch PUSH PSW belegt
F D	B	Akku }

Hieraus ist zu entnehmen, daß Akku und Flags je zweimal im Stack erscheinen. Die Inhalte der relativen Adressen F C und F D geben dabei die Zustände **vor** dem PUSH PSW-Befehl wieder, während in den relativen Adressen F 8 und F 9 der Zustand von Akku und Flags **vor** dem RST 2-Befehl wiedergegeben wird.
Weiterhin ist aus der Aufstellung zu entnehmen, daß die Stack-relative Adressierung (Stack-Offset) für den Monitor gleich geblieben ist. Die Inhalte der neu hinzugekommenen Stack-Adressen können mit 0 A und 0 B zur Anzeige gebracht werden. Sollen die gleichen Inhalte mit RAM-relativer Adressierung angezeigt werden, so muß die Adresse gegenüber dem ersten Beispiel um 2 erniedrigt werden.

Aufgabe:

Geben Sie für das nachfolgende Programm den Aufbau des Stacks entsprechend vorhergehendem Beispiel an:

absolute Adresse	Inhalt	Befehl	Kommentar
0 4 0 0	3 E	MVI A, 9 7	
0 4 0 1	9 7		
0 4 0 2	A 7	ANA A	
0 4 0 3	0 1	LXI B, 0 7 0 9	
0 4 0 4	0 9		
0 4 0 5	0 7		
0 4 0 6	F 5	PUSH PSW	
0 4 0 7	C 5	PUSH B	
0 4 0 8	9 7	SUB A	
0 4 0 9	0 1	LXI B, 0 0 0 0	
0 4 0 A	0 0		
0 4 0 B	0 0		
0 4 0 C	D 7	RST 2	
0 4 0 D	C 1	POP B	
0 4 0 E	F 1	POP PSW	
0 4 0 F	C 3	JMP 0 4 0 0	
0 4 1 0	0 0		
0 4 1 1	0 4		

Experiment 29: Initialisierung des Stack-Pointers

Mit dem nachfolgenden Programm soll ein Stack aufgebaut werden, das nicht vom Monitor initialisiert wird.
Laden Sie das Programm:

absolute Adresse	Inhalt	Befehl	Kommentar
0 4 0 0	3 1	LXI SP, 0 4 3 8	Lade SP mit 0 4 3 8
0 4 0 1	3 8		
0 4 0 2	0 4		
0 4 0 3	0 1	LXI B, 1 2 3 4	Lade Registerpaar BC
0 4 0 4	3 4		
0 4 0 5	1 2		
0 4 0 6	C 5	PUSH B	
0 4 0 7	D 7	RST 2	
0 4 0 8	C 1	POP B	
0 4 0 9	C 3	JMP 0 4 0 0	
0 4 0 A	0 0		
0 4 0 B	0 4		

Mit dem Befehl LXI SP, 0 4 3 8 wird der Stack-Pointer mit der Adresse 0 4 3 8 geladen. Der LXI B-Befehl lädt das Registerpaar BC mit 1 2 3 4. Mit dem Befehl PUSH B wird der Inhalt von Register B in Adresse 0 4 3 7, der Inhalt von Register C in Adresse 0 4 3 6 geladen.
Der RST-Befehl bringt die Registerinhalte und Flags wieder in der vom Monitor bestimmten Reihenfolge auf dem Stack unter. Die höchste Adresse ist dabei 0 4 3 5. Bei der Adresse 0 4 2 C ist der Stack-Bereich zu Ende. Nachfolgende Aufstellung verdeutlicht den Sachverhalt:

RAM-relative Adresse	Stack-Offset	Register
2 C	0	L
2 D	1	H
2 E	2	E
2 F	3	D
3 0	4	C
3 1	5	B
3 2	6	Flags
3 3	7	Akku
3 4	8	PC low
3 5	9	PC high
3 6	A	Inhalt Register C
3 7	B	Inhalt Register B

Aufgaben:

1. Geben Sie den Inhalt des Stack-Pointers vor dem POP-Befehl in Adresse 0 4 0 8 an!

2. Geben Sie den Aufbau des Stacks entsprechend vorhergehendem Beispiel an, wenn die Reihenfolge der Befehle RST 2 und POP B vertauscht wird!

3. In welchen Adressen werden die Inhalte des Registerpaares BC im geänderten Programm nach Aufgabe 2 durch den PUSH B-Befehl abgespeichert?

Experiment 30: XTHL-Befehl

Laden Sie folgendes Programm:

Adresse	Inhalt	Befehl	Kommentar
0 4 0 0	2 1	LXI H, 1 2 3 4	
0 4 0 1	3 4		
0 4 0 2	1 2		
0 4 0 3	C D	CALL 0 4 2 0	
0 4 0 4	2 0		
0 4 0 5	0 4		
0 4 0 6	D 7	RST 2	
0 4 0 7	7 6	HLT	
.			
.			
.			
0 4 2 0	E 3	XTHL	
0 4 2 1	D 7	RST 2	
0 4 2 2	E 3	XTHL	
0 4 2 3	C 9	RET	

Mit dem Befehl LXI H, 1 2 3 4 wird das Registerpaar HL und anschließend mit CALL 0 4 2 0 ein Unterprogramm angerufen. Die Rücksprungadresse ins Unterprogramm wird durch den CALL-Befehl im Stack gespeichert (Adressen 0 4 F D und 0 4 F C). Im Unterprogramm wird mit dem Befehl XTHL in Adresse 0 4 2 0 der „Stack-Kopf", das ist die durch den CALL-Befehl abgespeicherte Rücksprungadresse, mit dem **Inhalt** des Registerpaares HL (1 2 3 4) ausgetauscht. Um diesen Austausch kontrollieren zu können, ist in Adresse 0 4 2 1 ein RST 2-Befehl eingefügt. Der zweite XTHL-Befehl stellt wieder die ursprünglichen Verhältnisse her. Damit kann der Rücksprung ins Hauptprogramm mit dem RET-Befehl zur richtigen Adresse erfolgen (0 4 0 6).

Experiment 31: Setzen und Löschen der Flags

In einem Programm sollen die Flags in folgende Zustände gebracht werden:

N-Flag = 1
Z-Flag = 0
H-Flag = 1
P-Flag = 0
C-Flag = 1

Beim Beeinflussen der Flags sollen jedoch die Inhalte der Register nicht verändert werden. Laden Sie nachfolgendes Programm:

Adresse	Inhalt	Befehl	Kommentar
0 4 0 0	E 5	PUSH H	
0 4 0 1	6 7	MOV H, A	
0 4 0 2	2 E	MVI L, 9 3	
0 4 0 3	9 3		
0 4 0 4	E 5	PUSH H	
0 4 0 5	F 1	POP PSW	
0 4 0 6	E 1	POP H	
0 4 0 7	D 7	RST 2	
0 4 0 8	7 6	HLT	

Programmerläuterung:
Die Flags sollen in diesem Programm durch POP PSW gezielt beeinflußt werden, d.h., sie müssen zuvor auf den richtigen Platz im Stack gebracht werden. Da dies über das Registerpaar HL geschehen soll, wird zunächst mit PUSH H der „alte" Inhalt dieses Registerpaares in den Stack gebracht. Nun wird im Registerpaar HL das gewünschte Programm-Status-Word erzeugt. Der Akku-Inhalt soll nicht geändert werden, er wird daher mit dem Befehl MOV H, A in das H-Register geladen. Das Format des Flag-Wortes ist:

N	Z	0	H	0	P	1	C

Mit den anfangs aufgeführten Flag-Zuständen ergibt sich das folgende Wort:

1	0	0	1	0	0	1	1

$\triangleq 9\ 3_{16}$

Mit dem Befehl MVI L, 9 3 wird dieses Wort in das L-Register geladen und mit dem zweiten PUSH H-Befehl (in Adresse 0 4 0 4) auf den Stack gebracht. Mit dem Befehl POP PSW wird nun das zuvor erzeugte neue Program Status Word übernommen und mit dem folgenden POP H der „alte" Zustand des Registerpaares H, L wieder hergestellt.

Aufgaben:

1. Wie lautet das Flag-Wort, wenn das Z-Flag = 1 und alle anderen Flags = 0 sein sollen?

2. Geben Sie mit RAM-relativer Adresse und Inhalt den Aufbau des Stacks **nach** dem zweiten PUSH H-Befehl (Adresse 0 4 0 4) an!

3. Ändern Sie das vorgegebene Programm so ab, daß je ein PUSH- und POP-Befehl durch XTHL ersetzt werden!

Experiment 32: Experimentierprogramm im ROM

Im Abschnitt 6.3.3 wurde bereits erwähnt, daß das Monitorprogramm nicht geeignet ist, um die Inhalte von Speicherplätzen außerhalb des RAM-Bereiches in die Anzeige zu bringen. Das nachfolgende Programm liest die *A*- und *B*-Schalter als 16-bit-Adresse ein und ermöglicht somit den Zugriff zu **allen** Speicherplätzen.
Laden Sie nun das Programm ANSCHAU:

Adresse	Inhalt	Befehl	Kommentar
0 4 0 0	D B	IN ASHALT	
0 4 0 1	0 2		} *B*-Schalter→Register H
0 4 0 2	6 7	MOV H, A	
0 4 0 3	D B	IN BSHALT	
0 4 0 4	0 1		} *A*-Schalter→Register L
0 4 0 5	6 F	MOV L, A	
0 4 0 6	7 E	MOV A, M	
0 4 0 7	D 3	OUT RLAMPE	} (@ H, L)→Anzeige
0 4 0 8	0 1		
0 4 0 9	C 3	JMP 0 4 0 0	Rücksprung zum Anfang
0 4 0 A	0 0		
0 4 0 B	0 4		

Wenn Sie jetzt an den *A*- und *B*-Schaltern die Adressen 0 4 0 0 bis 0 4 0 B einstellen, werden die einzelnen Befehle des ANSCHAU-Programms in den rechten Lampen angezeigt.

Aufgabe:

1. „Lesen" Sie die ROM-Adressen 0 0 0 0 bis 0 0 0 7, und interpretieren Sie die Inhalte!

Zweites Beispiel aus dem ROM-Bereich soll ein Unterprogramm sein, das bei Adresse 0 0 2 0 beginnt. Dieses Unterprogramm kann mit einem RST 4-Befehl angerufen werden. Aufgabe dieses Programms ist es, den Inhalt des B-Registers zu maskieren und mit einer Vergleichszahl zu vergleichen.
Die Maske wird im 1. Byte, die Vergleichszahl im 2. Byte nach dem RST 4-Befehl gespeichert. Der darauf folgende Befehl (im nachfolgenden Beispiel CMA) wird je nach Ergebnis des Vergleiches bearbeitet oder übersprungen.
Das Programmbeispiel liest die *C*-Schalter ein und isoliert durch die Maske 0 8 den Schalter C_3. Abhängig von der Stellung dieses Schalters wird dann der von den *A*-Schaltern eingelesene Wert negiert oder nicht negiert an den linken Lampen ausgegeben.
Laden Sie nachfolgendes Programm:

Adresse	Inhalt	Befehl	Kommentar
0 4 0 0	D B	IN CSHALT	
0 4 0 1	0 4		} *C*-Schalter→Register B
0 4 0 2	4 7	MOV B, A	
0 4 0 3	D B	IN ASHALT	
0 4 0 4	0 2		} *A*-Schalter→Akku
0 4 0 5	E 7	RST 4	Unterprogrammanruf
0 4 0 6	0 8		Maske
0 4 0 7	0 8		Vergleichszahl
0 4 0 8	2 F	CMA	
0 4 0 9	D 3	OUT LLAMPE	} Akku→Anzeige
0 4 0 A	0 2		
0 4 0 B	C 3	JMP 0 4 0 0	
0 4 0 C	0 0		} Rücksprung zum Anfang
0 4 0 D	0 4		

Aufgabe:

2. Ändern Sie Maske und Vergleichszahl so ab, daß an den Schaltern C_0 bis C_2 die Kombination 1 0 1 eingestellt werden muß, um den CMA-Befehl zur Ausführung zu bringen!

Ein ähnlich aufgebautes Unterprogramm beginnt bei Adresse 0 0 3 0, kann also mit RST 6 angerufen werden. Im Gegensatz zum vorhergehenden Programm werden jetzt 3 Bytes übersprungen, die auf Maske und Vergleichszahl folgen.
Laden Sie folgendes Programm:

Adresse	Inhalt	Befehl	Kommentar
0 4 0 0	D B	IN BSHALT	} *B*-Schalter→Register B
0 4 0 1	0 1		
0 4 0 2	4 7	MOV B, A	
0 4 0 3	D B	IN ASHALT	} *A*-Schalter→Akku
0 4 0 4	0 2		
0 4 0 5	F 7	RST 6	Unterprogrammanruf
0 4 0 6	0 7		Maske
0 4 0 7	0 2		Vergleichszahl
0 4 0 8	3 C	INR A	} Befehlsfolge, die übersprungen werden kann
0 4 0 9	1 7	RAL	
0 4 0 A	1 8	RAL	
0 4 0 B	D 3	OUT LLAMPE	} Akku→Anzeige
0 4 0 C	0 2		
0 4 0 D	C 3	JMP 0 4 0 0	
0 4 0 E	0 0		
0 4 0 F	0 4		

Programmerläuterung:
In diesem Beispiel soll, wenn die Schalter B_0, B_1 und B_2 auf 0 1 0 eingestellt sind, der Akku-Inhalt incrementiert und zweimal nach links verschoben werden. Dazu wird, nachdem die *B*-Schalter eingelesen wurden, der Akku mit den *A*-Schaltern geladen. Der OUT-Befehl in Adresse 0 4 0 B bringt dann entweder den ursprünglichen oder (wenn die Schalter B_2 bis B_0 auf 0 1 0 eingestellt sind) den geänderten Akku-Inhalt in den linken Lampen zur Anzeige.

Experiment 33: Mehr-Byte-Addition

Sollen in einem Programm größere Zahlen verarbeitet werden, so erfolgt deren Unterbringung in mehreren hintereinanderliegenden Speicherplätzen. Das nachfolgende Programm soll zwei 3-Byte-Zahlen addieren und das Ergebnis in die Speicherplätze der ersten Zahl zurückschreiben.
Laden Sie das nachfolgende Programm:

Adresse	Inhalt	Befehl	Kommentar
0 4 0 0	D 7	RST 2	
0 4 0 1	0 E	MVI C, 3	Anzahl der Bytes→C
0 4 0 2	0 3		
0 4 0 3	1 1	LXI D, 0 4 2 0	Anfangsadresse Zahl 1 und Ergebnis→DE
0 4 0 4	2 0		
0 4 0 5	0 4		
0 4 0 6	2 1	LXI H, 0 4 3 0	Anfangsadresse Zahl 2→HL
0 4 0 7	3 0		
0 4 0 8	0 4		
0 4 0 9	A F	XRA A	C-Flag = 0
0 4 0 A	1 A	LDAX D	(@ D) + (@ H)→ @ D
0 4 0 B	8 E	ADC M	
0 4 0 C	1 2	STAX D	
0 4 0 D	0 D	DCR C	
0 4 0 E	C A	JZ 0 4 0 0	Aussprung, wenn Byte-Zähler C = 0
0 4 0 F	0 0		
0 4 1 0	0 4		
0 4 1 1	1 3	INX D	Erhöhen der Byte-Adressen
0 4 1 2	2 3	INX H	
0 4 1 3	C 3	JMP 0 4 0 A	nächstes Byte bearbeiten
0 4 1 4	0 A		
0 4 1 5	0 4		

Programmerläuterung:
Der eigentliche Rechenteil des Programms beginnt mit dem Löschen des C-Flags durch den XRA A-Befehl in Adresse 0 4 0 9. Um diesen Teil allgemein verwendbar zu halten, muß im Vorspann mit dem Befehl MVI C (Adresse 0 4 0 1) die Zahl der zu addierenden Bytes geladen werden. Ebenso müssten mit den beiden folgenden LXI-Befehlen die Adressen der ersten Bytes beider Zahlen geladen werden. Nach der eigentlichen Addition mit dem Befehl ADC M (Adresse 0 4 0 B) muß ein eventuell entstehender Übertrag bis zur nächsten Addition bestehen bleiben. Überprüfen Sie anhand der Befehlsliste, ob das C-Flag in der Programmschleife nicht verändert wird.

Aufgabe:
Ändern Sie das Programm so ab, daß es mit dem Befehl CALL 0 4 6 0 als Unterprogramm angerufen werden kann. Der Vorspann soll mit einem RST 2-Befehl beginnen und die Addition von zwei 4-Byte-Zahlen veranlassen, deren Anfangsadressen wieder 0 4 2 0 bzw. 0 4 3 0 lauten sollen. Nach dem Abarbeiten des Unterprogramms Rücksprung nach Adresse 0 4 0 0.

Experiment 34: Multiplikation

In diesem Programm werden die Inhalte der Register C und D miteinander multipliziert. Das Ergebnis erscheint im Registerpaar BC.
Der verwendete Algorithmus fragt alle 8 bit des Multiplikators im Register C auf ihren Inhalt ab. Ist der Inhalt 1, so muß der Multiplikand stellenwertrichtig addiert werden. Begonnen wird mit der wertniedrigsten Stelle des Multiplikators.

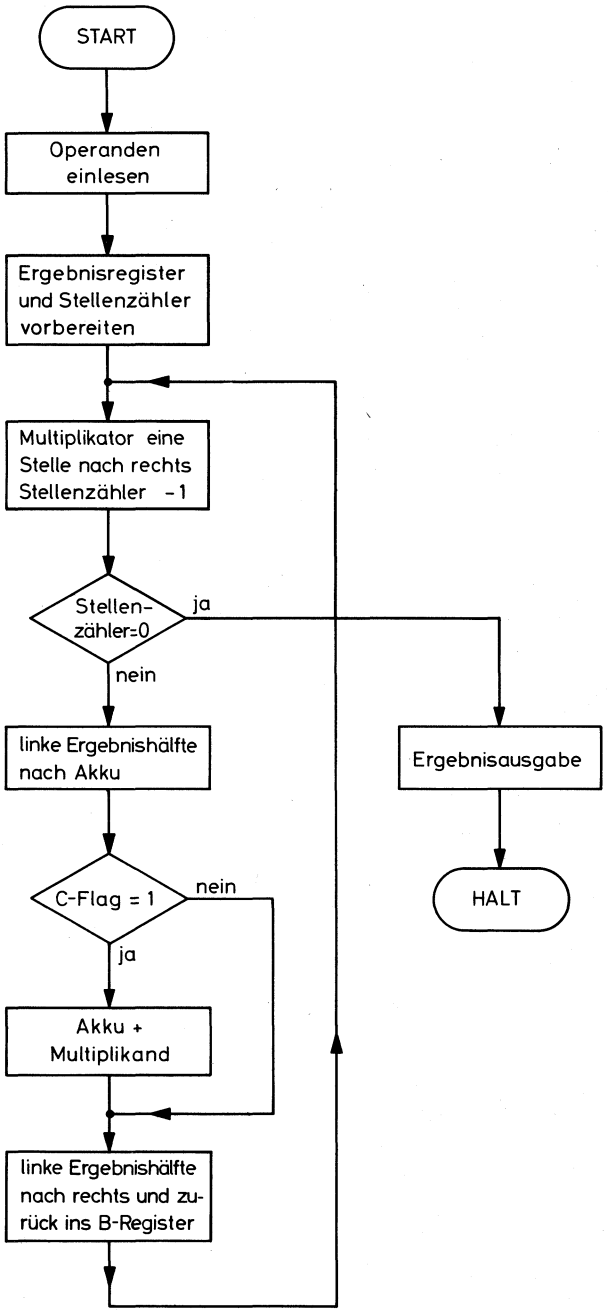


Bild 1
Flußdiagramm des Multiplikationsprogramms

Programmerläuterung (Bild 1):
Im ersten Programmteil bis zur Adresse 0 4 0 A werden Multiplikand und Multiplikator eingelesen, der Stellenzähler (Register E) wird gesetzt und die linke Hälfte des Ergebnisregisters gelöscht.
Die 3 Befehle in den Adressen 0 4 0 B bis 0 4 0 C verschieben den Inhalt des Registers C um eine Stelle nach rechts. Das wertniedrigste bit des Multiplikators befindet sich jetzt im C-Flipflop. Wenn der Stellenzähler noch nicht Null ist, wird mit dem Befehl MOV A, B (Adresse 0 4 1 2) die linke Hälfte des Ergebnisregisters in den Akkumulator geholt. Der

Befehl ADD D (Adresse 0 4 1 6) wird nur dann ausgeführt, wenn das C-Flag = 1 ist. Das bedeutet, daß jede 1 im Multiplikator die Addition des Multiplikanden zur linken Ergebnishälfte veranlaßt. Die linke Ergebnishälfte, die jetzt noch im Akkumulator steht, wird mit RAR (Adresse 0 4 1 7) um eine Stelle nach rechts verschoben und dann mit MOV B, A wieder in das B-Register gebracht. Bei dem Schiebevorgang gelangt das wertniedrigste Ergebnis-bit in das Carry-Flipflop. Der Sprungbefehl zur Adresse 0 4 0 B beeinflußt das C-Flag nicht, so daß mit dem Schiebebefehl für das Register C (MOV A, C; RAR; MOV C, A) das wertniedrigste Ergebnis-bit in der werthöchsten Stelle des Registers C erscheint. Gleichzeitig wird das zweite bit des Multiplikators in das C-Flipflop geschoben. Von hier ab wiederholt sich der Vorgang, bis alle 8 Multiplikatorstellen bearbeitet sind.
Der Stellenzähler muß auf 9 gesetzt werden, da der Aussprung aus der Programmschleife **vor** dem Verschieben der linken Ergebnishälfte erfolgt, diese also nur 8mal durchgeführt wird.
Am Ende der Multiplikation wird das Registerpaar BC in der rechten und linken Lampenreihe zur Anzeige gebracht.
Nach dem Laden können an den *A*- und *B*-Schaltern die Operanden eingestellt werden. Mit dem RUN-Schalter wird dann das Programm gestartet. Weitere Programmdurchläufe mit geänderten Operanden können durch Drücken der RESET-Taste veranlaßt werden.
Laden Sie nun das Programm und probieren Sie es mit einigen Beispielen aus.
Alle angegebenen Zahlen sind Hexadezimal zu verstehen.

1. 0 7 · 0 7 = 0 0 3 1
2. 0 A · 2 1 = 0 1 4 A
3. F 7 · 6 8 = 6 4 5 8
4. F F · F F = F E 0 1

Adresse	Inhalt	Befehl	Kommentar
0 4 0 0	0 0	NOP	
0 4 0 1	D B	IN BSHALT	einlesen von Multiplikand und Multiplikator
0 4 0 2	0 1		
0 4 0 3	4 F	MOV C, A	
0 4 0 4	D B	IN ASHALT	
0 4 0 5	0 2		
0 4 0 6	5 7	MOV D, A	
0 4 0 7	0 6	MVI B, 0 0	linke Hälfte des Ergebnisregisters löschen
0 4 0 8	0 0		
0 4 0 9	1 E	MVI E, 0 9	Stellenzähler setzen
0 4 0 A	0 9		
0 4 0 B	7 9	MOV A, C	Register C um eine Stelle nach rechts
0 4 0 C	1 F	RAR	
0 4 0 D	4 F	MOV C, A	
0 4 0 E	1 D	DCR E	Stellenzähler verringern
0 4 0 F	C A	JZ 0 4 1 C	Aussprung, wenn alle Stellen bearbeitet sind
0 4 1 0	1 C		
0 4 1 1	0 4		
0 4 1 2	7 8	MOV A, B	linke Hälfte Ergebnisregister→Akku
0 4 1 3	D 2	JNC 0 4 1 7	Addition nicht ausführen, wenn C-Flag = 0
0 4 1 4	1 7		
0 4 1 5	0 4		
0 4 1 6	8 2	ADD D	
0 4 1 7	1 F	RAR	Ergebnis eine Stelle nach rechts und wieder in Register B abspeichern
0 4 1 8	4 7	MOV B, A	
0 4 1 9	C 3	JMP 0 4 0 B	
0 4 1 A	0 B		
0 4 1 B	0 4		
0 4 1 C	7 8	MOV A, B	Inhalt des Ergebnisses in die Anzeige bringen
0 4 1 D	D 3	OUT LLAMPE	
0 4 1 E	0 2		
0 4 1 F	7 9	MOV A, C	
0 4 2 0	D 3	OUT RLAMPE	
0 4 2 1	0 1		
0 4 2 2	7 6	HLT	

Experiment 35: Zähler als Zufallsgenerator

In der Rechentechnik werden häufig Zufallszahlen benötigt, um z.B. externe Zufallsprozesse zu simulieren oder Algorithmen mit willkürlichen Zahlenwerten zu testen. Das folgende Programm soll einen einfachen Weg zur Erzeugung sogenannter Pseudozufallszahlen zeigen. Es handelt sich hier nicht um „echte" Zufallszahlen, da nur die verhältnismäßig lange Reaktionszeit des Menschen eine gewollte Beeinflussung des Ergebnisses unmöglich macht. Der Schalter C_3 soll bei jedem Takten eine neue Zufallszahl entstehen lassen. Als Zähler wird das Register C verwendet.
Laden Sie nun das folgende Programm:

Adresse	Inhalt	Befehl	Kommentar
0 4 0 0	D B	IN CSHALT	*C*-Schalter einlesen
0 4 0 1	0 4		
0 4 0 2	0 C	INR C	Zufallszähler erhöhen
0 4 0 3	E 6	ANI 08	Schalter C_3 isolieren
0 4 0 4	0 8		
0 4 0 5	C 2	JNZ 0 4 0 0	Rücksprung, wenn $C_3 = 1$
0 4 0 6	0 0		
0 4 0 7	0 4		
0 4 0 8	D B	IN CSHALT	
0 4 0 9	0 4		
0 4 0 A	0 C	INR C	
0 4 0 B	E 6	ANI 08	
0 4 0 C	0 8		
0 4 0 D	C A	JZ 0 4 0 8	Rücksprung, wenn $C_3 = 0$
0 4 0 E	0 8		
0 4 0 F	0 4		
0 4 1 0	D 7	RST 2	Monitoranruf
0 4 1 1	C 3	JMP 0 4 0 0	
0 4 1 2	0 0		
0 4 1 3	0 4		

Programmerläuterung:
Das Programm besteht aus 2 gleich aufgebauten Schleifen. Die erste Schleife von Adresse 0 4 0 0 bis 0 4 0 7 wird durchlaufen, solange der Schalter C_3 auf 1 steht. Die Isolierung des C_3-Schalters wird mit dem Befehl ANI 0 8 durchgeführt. (0 8 ≙ 0 0 0 0 1 0 0 0, die 1 steht an der dem Schalter C_3 zugeordneten Stelle, siehe auch Experiment 16). Wird der Schalter C_3 auf 0 zurückgenommen, erfolgt der Übergang in die zweite Schleife von Adresse 0 4 0 8 bis 0 4 0 F. In beiden Schleifen wird das C-Register erhöht. Wird der Schalter C_3 jetzt wieder auf 1 gebracht, erfolgt **ein** Monitordurchlauf und anschließend der Rücksprung in die erste Schleife. Der Monitoranruf soll die Anzeige des Inhaltes des C-Registers ermöglichen. Dazu können z.B. die *A*-Schalter auf 0 4 (Stack-relative Adresse) eingestellt werden.

Anmerkung:
Da der Schalter C_3 im Monitorprogramm als RUN-Schalter verwendet wird, braucht das Programm nicht gesondert gestartet zu werden.

Experiment 36: Reaktionszeitmessung

Mit diesem Programm können Reaktionszeiten bis zu 300 ms gemessen werden.
Um die einzelnen Programmteile in ihrer Bedeutung besser verstehen zu können, soll zunächst die Funktion des Programms erklärt werden. Wenn hier von „Anzeige" gesprochen wird, so sind damit alle 16 LEDs der rechten **und** linken Lampenreihe gemeint. Es lassen sich also 4stellige Hexadezimalzahlen zur Anzeige bringen.

Funktionsbeschreibung:
Nach dem Laden und Starten des Programms wird zunächst mit Hilfe eines Zählers (siehe auch vorhergehendes Experiment) eine Zufallszahl erzeugt. Mit dem Zurücknehmen des C_3-Schalters von 1 auf 0 wird F F F F_{16} angezeigt, und damit der Beginn der zufälligen Wartezeit. Die Wartezeit ist beendet, wenn durch wiederholtes decrementieren die Zufallszahl zu Null geworden ist. In diesem Augenblick erlöschen sämtliche Anzeigelampen und die eigentliche Meßzeit beginnt. Jetzt erscheint in der Lampe R_0 eine „1", die alle 20 ms um eine Stelle nach links verschoben wird. Die Meßzeit wird gestoppt, wenn der Schalter C_3 wieder von 0 auf 1 gebracht wurde. Aus der Anzahl der Verschiebungen läßt sich die Reaktionszeit errechnen. Wurde die „1" über die Lampe L_7 hinaus in das C-Flipflop verschoben, so heißt das, daß die Reaktionszeit größer als 300 ms war. In diesem Fall wird F F 0 0_{16} angezeigt, und das Programm kann nur durch Drücken der RESET-Taste wieder gestartet werden.

Programmerläuterung (Bild 1):
Teil 1 des Programms dient zur Erzeugung einer Zufallszahl. Register B wird incrementiert, solange der Schalter C_3 auf 1 bleibt. C_3 wird wie im vorhergehenden Experiment durch den Befehl ANI 0 8 isoliert. Wird C_3 auf 0 zurückgenommen, findet kein Rücksprung zum Programmanfang statt, sondern der Übergang zum Teil 2 des Programms.

Teil 2 beginnt bei Adresse 0 4 0 8 und bringt zunächst mit den beiden OUT-Befehlen F F F F in die Anzeige. Die zufällige Wartezeit wird durch das Rückwärtszählen des Registers B (Befehl DCX B) in Adresse 0 4 1 4 bestimmt. Die größtmögliche Zufallszahl ist F F_{16}, so daß diese sogenannte äußere Schleife bis zu 127_{10} Durchläufe benötigt, um bei B = 0 zum 3. Teil des Programms überzugehen. Um die Schleifendurchläufe zu verlangsamen, wurde zusätzlich die innere Schleife (Adresse 0 4 1 0 bis 0 4 1 3) eingebaut. Da der Akkumulator in Adresse 0 4 0 E und 0 4 0 F mit 4 0_{16} geladen wird, muß diese innere Schleife 4 0_{16} ($\triangleq 64_{10}$) mal durchlaufen werden, bis die äußere Schleife wieder bearbeitet wird. Damit ergäbe sich für die äußere Schleife eine Durchlaufzeit von ca. 1 ms. Da diese Zeit immernoch zu kurz ist, wird nicht die Zufallszahl im Register B decrementiert, sondern mit dem Befehl DCX B (Adresse 0 4 1 4) das Registerpaar BC. Das Register C stellt die wertniedrigste Hälfte des Inhaltes vom Registerpaar BC dar. Daher muß der Befehl DCX B 256_{10}-mal ausgeführt werden, bevor der Inhalt des Registers B um 1 verringert wird. Register C wird gewissermaßen als Vorteiler mit dem Teilverhältnis 256_{10} : 1 verwendet. Die Wartezeiten können somit ca. 0,25 bis 255 · 0,25 s betragen. Das Decrementieren eines Registerpaares mit dem Befehl DCX beeinflußt keines der Flags. Um also das Kriterium [(B) = 0] zum Übergang in den Prograammteil 3 zu erhalten, müssen die Flags mit dem Befehl CMP B (Adresse 0 4 1 5) wieder gesetzt werden. Z-Flag = 1 löst dann den eigentlichen Meßvorgang aus.

Teil 3 des Programms signalisiert zunächst den Beginn der Meßzeit durch 0 0 0 0_{16} in der Anzeige. Da am Ende von Teil 2 der Akkumulator immer den Inhalt 0 0 haben muß, können am Anfang von Teil 3 sofort 2 OUT-Befehle (Adressen 0 4 1 9 und 0 4 1 B) stehen. Das Registerpaar HL wird mit 0 0 0 1_{16} geladen.
Auch in diesem Programmteil kann man wieder zwischen einer inneren und einer äußeren Schleife unterscheiden.
Mit dem Befehl LXI D (Adresse 0 4 2 0) wird das Registerpaar DE mit dem Wert 0 1 8 0_{16} ($\triangleq 384_{10}$) geladen. Die innere Schleife wird verlassen, wenn entweder das Registerpaar DE bis zu 0 decrementiert wurde oder der Schalter C_3 auf 1 gebracht wurde. Die Abfrage des C_3-Schalters wird in bekannter Weise in den Adressen 0 4 2 3 bis 0 4 2 9 durchgeführt. Um das Registerpaar DE auf den Inhalt 0 0 0 0 zu untersuchen, wird die eine Hälfte mit dem Befehl MOV A, D (Adresse 0 4 2 B) in den Akkumulator gebracht. Die anschließende ODER-Verknüpfung von Akkumulator und Register E (Register E stellt die zweite Hälfte dar)

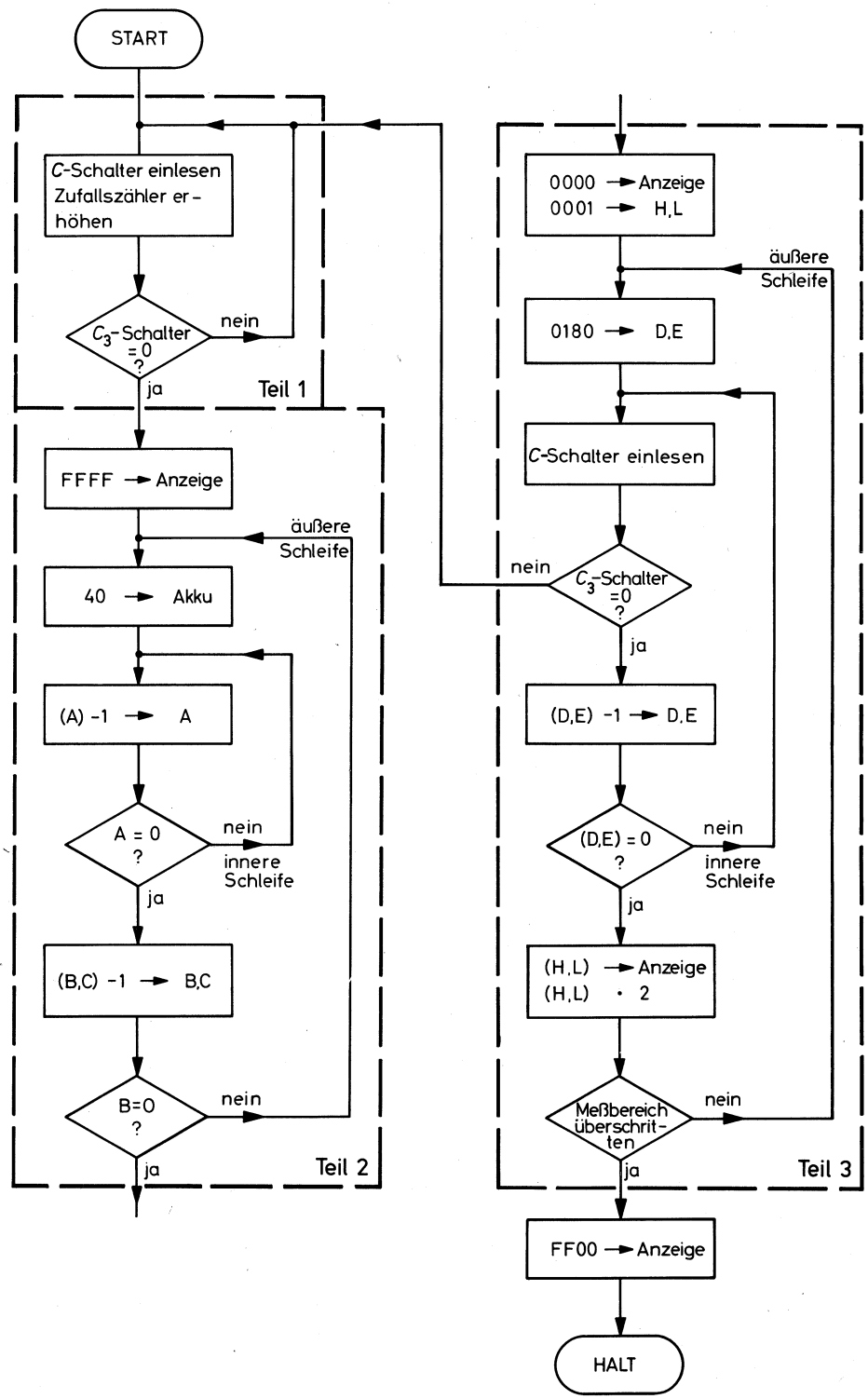


Bild 1
Flußdiagramm des Programms zur Reaktionszeitmessung

kann nur dann Null als Ergebnis liefern, wenn beide Hälften für sich bereits Null sind. Da in diesem Fall das Z-Flag = 1 ist, wird jetzt die äußere Schleife weiter bearbeitet.
Mit den MOV- und OUT-Befehlen in den Adressen 0 4 3 0 bis 0 4 3 5 wird der Inhalt des Registerpaares HL in die Anzeige gebracht. Mit dem Befehl DAD H wird dann der Inhalt dieses Registerpaares verdoppelt, so daß scheinbar die mit LXI H, 0 0 0 1 (Adresse 0 4 1 D) geladene 1 bei jedem Schleifendurchlauf um eine Stelle nach links geschoben wird.
Ein Überschreiten des Meßbereiches, das nach 15 Durchläufen der äußeren Schleife stattfindet, läßt nach dem DAD H-Befehl (Adresse 0 4 3 6) einen Übertrag entstehen (C-Flag = 1). Der bedingte Sprung JNC in Adresse 0 4 3 7 wird dann nicht mehr ausgeführt. Mit den anschließenden Befehlen wird dann F F 0 0_{16} zur Anzeige gebracht und das Programm gestoppt.

Adresse	Inhalt	Befehl	Kommentar
0400	DB	IN CSHALT	C-Schalter einlesen
0401	04		
0402	04	INR B	Zufallszähler erhöhen
0403	E6	ANI 08	C_3 isolieren
0404	08		
0405	C2	JNZ 0400	Rücksprung wenn $C_3 = 1$
0406	00		
0407	04		
0408	3E	MVI A, FF	F F F F in die Anzeige bringen
0409	FF		
040A	D3	OUT LLAMPE	
040B	02		
040C	D3	OUT RLAMPE	
040D	01		
040E	3E	MVI A, 40	äußere Schleife in Teil 2
040F	40		
0410	3D	DCR A	innere Schleife in Teil 2
0411	C2	JNZ 0410	
0412	10		
0413	04		
0414	0B	DCX B	
0415	B8	CMP B	
0416	C2	JNZ 040E	
0417	0E		
0418	04		
0419	D3	OUT LLAMPE	0 0 0 0 in die Anzeige bringen
041A	02		
041B	D3	OUT RLAMPE	
041C	01		
041D	21	LXI H, 0001	
041E	01		
041F	00		
0420	11	LXI D, 0180	Rücksprungziel für die äußere Schleife in Teil 3
0421	80		
0422	01		
0423	DB	IN CSHALT	innere Schleife in Teil 3
0424	04		
0425	E6	ANI 08	
0426	08		
0427	C2	JNZ 0400	Aussprung nach erfolgter Reaktion
0428	00		
0429	04		
042A	1B	DCX D	
042B	7A	MOV A, D	(DE) auf Null abfragen
042C	B3	ORA E	
042D	C2	JNZ 0423	
042E	23		
042F	04		
0430	7C	MOV A, H	(HL) in die Anzeige bringen
0431	D3	OUT LLAMPE	
0432	02		
0433	7D	MOV A, L	
0434	D3	OUT RLAMPE	
0435	01		
0436	29	DAD H	„1" in HL eine Stelle nach links
0437	D2	JNC 0420	Rücksprungbefehl für die äußere Schleife in Teil 3
0438	20		
0439	04		

Adresse	Inhalt	Befehl	Kommentar
0 4 3 A	3 E	MVI A, F F	F F 0 0 in die Anzeige bringen
0 4 3 B	F F		
0 4 3 C	D 3	OUT LLAMPE	
0 4 3 D	0 2		
0 4 3 E	9 7	SUB A	
0 4 3 F	D 3	OUT RLAMPE	
0 4 4 0	0 1		
0 4 4 1	7 6	HLT	

Experiment 37: Überwachung von 8 Grenzwertmeldern

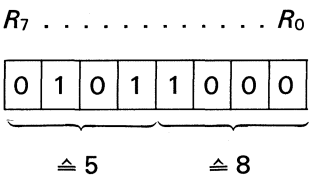
Mit Hilfe dieses Programms sollen 8 Grenzwertmelder (z.B. Druckgeber) überwacht werden. Dabei sind folgende Bedingungen zu erfüllen:

1. Ordnungsgemäße Funktion soll durch 8 8_{16} in den rechten Lampen angezeigt werden.
2. Die einzelnen Melder sind von 0 bis 7 numeriert (entsprechend den *B*-Schaltern, die als Melder dienen sollen). Wird eine Grenzwertüberschreitung gemeldet, soll die Nummer des entsprechenden Melders als Zahl im 8421-Code in die rechten Lampen gebracht werden. Es können also 2 Überschreitungen signalisiert werden.
3. Wenn mehr als 2 Überschreitungen gemeldet werden, soll dies durch F F in den rechten Lampen angezeigt werden.
4. Funktion der Melder:

1 ≙ zulässiger Wert
0 ≙ Grenzwert überschritten

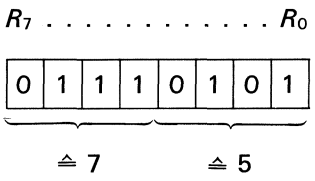
Programmerläuterung (Bild 1):
Im Register D werden die anzuzeigenden Daten gespeichert (8 8 oder die Nummern der Melder). Wenn nur **ein** Melder eine Grenzwertüberschreitung anzeigt, wird dessen Nummer in den linken 4 bit der rechten Lampenreihe angezeigt (R_7 bis R_4), die rechten 4 bit zeigen noch die Zahl 8.

1. Beispiel:
Schalter B_5 auf 0, alle anderen *B*-Schalter auf 1 muß folgende Anzeige ergeben:



Sprechen 2 Melder an, steht die höhere Nummer in den linken 4 bit.

2. Beispiel:
Wenn nach Melder 5 der Melder 7 (Schalter B_7) anspricht, erscheint die folgende Anzeige:



Um jeden einzelnen Melder isolieren zu können, wird ein Zeiger benötigt, d.h. ein Register, in dem **eine** 1 enthalten ist, die über alle Stellen verschoben wird. Durch eine AND-Verknüpfung dieser 1 mit den Melderzuständen kann jeder einzelne Melder auf seinen Zustand geprüft werden.

Teil 1 des Programms prüft, ob alle Melder = 1 sind. Dazu werden die eingelesenen Zustände negiert (CMA in Adresse 0 4 0 4). Um die Flags zu erzeugen, wird anschließend eine AND-Verknüpfung des Akkumulators mit sich selbst durchgeführt. Wenn jetzt das Z-Flag = 1 ist, bedeutet das, daß alle Melder = 1 waren. In diesem Fall erfolgt ein Sprung zur Adresse 0 4 3 3 und der Inhalt des Registers D (8 8) wird in den rechten Lampen angezeigt. Z-Flag = 0 bedeutet, daß **mindestens** eine 0 in den eingelesenen Daten enthalten war. Vor dem Übergang zum Teil 2 des Programms wird durch einen zweiten CMA-Befehl (Adresse 0 4 0 9) der ursprüngliche Meldezustand wieder erzeugt und in Register B abgespeichert.

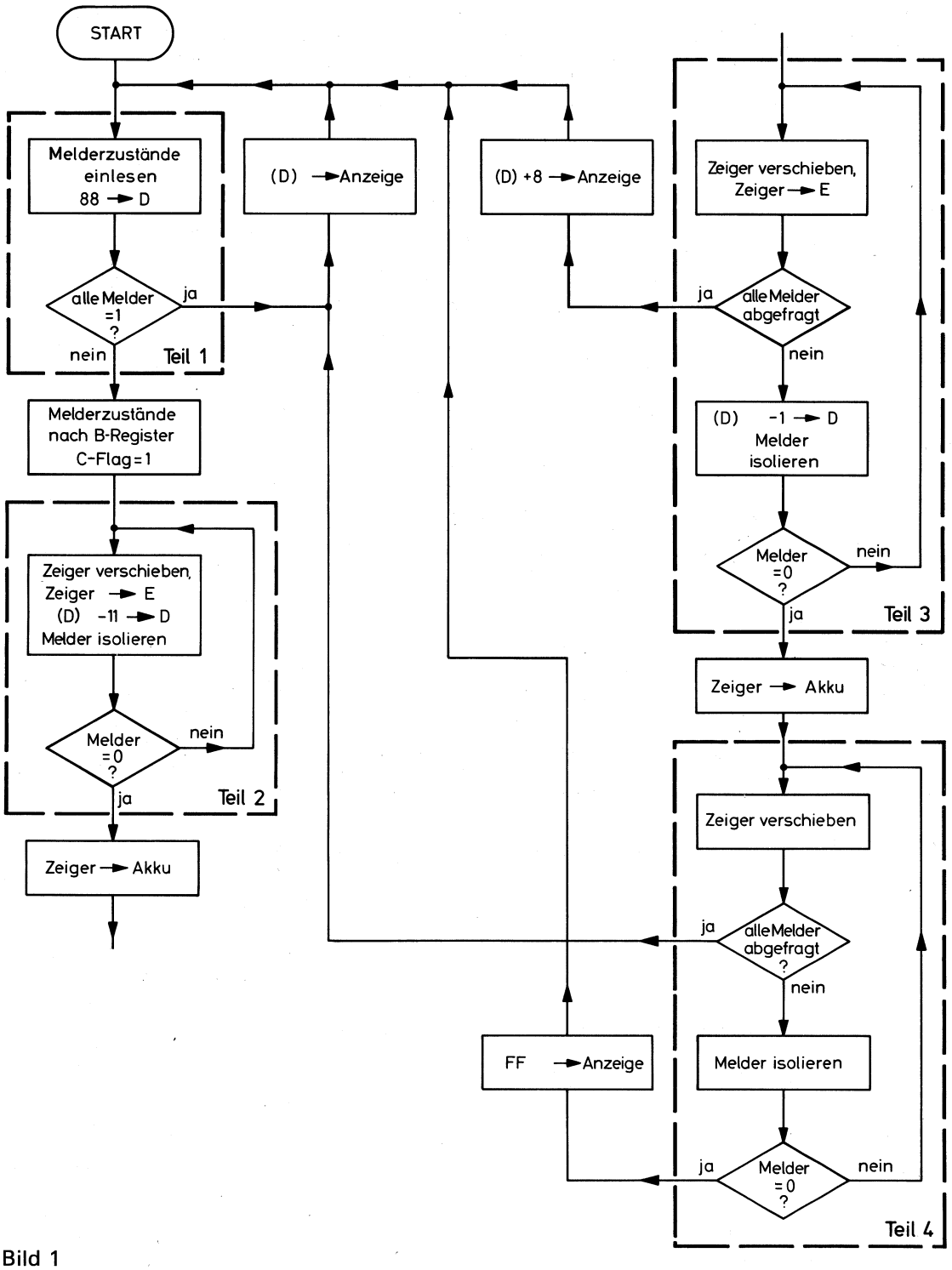


Bild 1
Flußdiagramm des Programms zur Überwachung von 8 Grenzwertmeldern

Der Zeiger wird erzeugt, indem zunächst der Akku gelöscht wird. Mit dem Befehl STC wird das C-Flag = 1 gesetzt. Diese 1 wird mit dem RAR-Befehl (Adresse 0 4 0 D) in die höchste Stelle des Akkumulators geschoben.

Teil 2 beginnt mit diesem RAR-Befehl. Der Zeiger wird dann in Register E gespeichert. Die Befehle in den Adressen 0 4 0 F bis 0 4 1 2 verringern den Inhalt der Register D bei jedem Schleifendurchlauf um $1\ 1_{16}$, so daß nacheinander die Zahlen 7 7; 6 6; 5 5; usw. in diesem Register stehen. Da der RAR-Befehl in Adresse 0 4 0 D ebenfalls in dieser Schleife liegt, wird parallel zum Verringern des D-Registers auch der „Zeiger" auf die entsprechende Stelle verschoben. Wird ein Melder mit dem Zustand 0 gefunden, dann wird der

Sprungbefehl in Adresse 0 4 1 5 nicht mehr ausgeführt. Stattdessen wird der Zeiger wieder in den Akkumulator geholt.

Teil 3 beginnt, wie auch der zweite Teil, mit dem Verschieben und Abspeichern des Zeigers. Der Ablauf im Teil 3 soll unter verschiedenen Voraussetzungen betrachtet werden:

a) **nur ein Melder** hat angesprochen (Schalter B_5 auf 0)
Dieser Melder wurde in Teil 2 festgestellt, so daß das Register D den Inhalt 5 5 hat. In der Schleife in Teil 3 wird das Register D nur noch decrementiert (DCR-Befehl in Adresse 0 4 1 E). Die Inhalte dieses Registers sind daher 5 4; 5 3; 5 2; 5 1; 5 0. Beim letzten Schleifendurchlauf wird der Melder 0 (Schalter B_0) isolierend abgefragt. Da er sich auf 1 befindet, wird der Sprungbefehl in Adresse 0 4 2 9 ausgeführt. Die 1 im Zeiger wird, da sie jetzt in der niedrigsten Stelle steht, in das C-Flipflop geschoben. Somit erfolgt durch den JC-Befehl in Adresse 0 4 1 B der Aussprung aus diesem Programmteil. Zum Inhalt des Registers D wird 0 8 addiert, so daß nach dem OUT-Befehl in Adresse 0 4 3 9 die Zahlen 5 8 in der Anzeige zu sehen sind.

b) **mindestens 2 Melder** haben angesprochen
In diesem Fall wird, nachdem der zweite Melder mit dem Zustand 0 festgestellt wurde, im Teil 4 geprüft, ob ein weiterer Melder angesprochen hat. Ist das der Fall, wird der Sprungbefehl in Adresse 0 4 2 9 nicht ausgeführt und F F angezeigt.
Wenn nur 2 Melder angesprochen haben, wird nach der Abfrage des Melders 0 der Sprungbefehl in Adresse 0 4 2 5 ausgeführt.
Um ein dauerndes Abfragen der Melder zu ermöglichen, wird nach der Anzeige ein Rücksprung zum Programmanfang durchgeführt.

Adresse	Inhalt		Kommentar
0 4 0 0	D B	IN BSHALT	Melderzustände einlesen
0 4 0 1	0 1		
0 4 0 2	1 6	MVI D, 8 8	Anzeigeregister laden
0 4 0 3	8 8		
0 4 0 4	2 F	CMA	
0 4 0 5	A 7	ANA A	
0 4 0 6	C A	JZ 0 4 3 1	} Sprung, wenn Melder = 1
0 4 0 7	3 1		
0 4 0 8	0 4		
0 4 0 9	2 F	CMA	} Melderzustände→Register B
0 4 0 A	4 7	MOV B, A	
0 4 0 B	9 7	SUB A	Akku löschen
0 4 0 C	3 7	STC	Zeiger vorbereiten
0 4 0 D	1 F	RAR	} Zeiger verschieben und in
0 4 0 E	5 F	MOV E, A	} Register E abspeichern
0 4 0 F	7 A	MOV A, D	
0 4 1 0	D 6	SUI 1 1	} (D) – 1 1→D
0 4 1 1	1 1		
0 4 1 2	5 7	MOV D, A	
0 4 1 3	7 B	MOV A, E	Zeiger→Akku
0 4 1 4	A 0	ANA B	Melder isolieren
0 4 1 5	C 2	JNZ 0 4 0 D	} Sprung, wenn Melder = 1
0 4 1 6	0 D		
0 4 1 7	0 4		
0 4 1 8	7 B	MOV A, E	Zeiger→Akku
0 4 1 9	1 F	RAR	} Zeiger verschieben und in
0 4 1 A	5 F	MOV E, A	} Register E abspeichern
0 4 1 B	D A	JC 0 4 3 5	} Sprung, wenn nur ein Melder = 0
0 4 1 C	3 5		
0 4 1 D	0 4		
0 4 1 E	1 5	DCR D	(D) – 1→D
0 4 1 F	A 0	ANA B	Melder isolieren

Adresse	Inhalt	Befehl	Kommentar
0 4 2 0	C 2	JNZ 0 4 1 9	
0 4 2 1	1 9		} Sprung, wenn Melder = 1
0 4 2 2	0 4		
0 4 2 3	7 B	MOV A, E	Zeiger→Akku
0 4 2 4	1 F	RAR	Zeiger verschieben
0 4 2 5	D A	JC 0 4 3 1	
0 4 2 6	3 1		} Sprung, wenn nur zwei Melder = 0
0 4 2 7	0 4		
0 4 2 8	A 0	ANA B	Melder isolieren
0 4 2 9	C 2	JNZ 0 4 2 4	
0 4 2 A	2 4		} Sprung, wenn Melder = 1
0 4 2 B	0 4		
0 4 2 C	3 E	MVI A, F F	F F→Akku
0 4 2 D	F F		
0 4 2 E	C 3	JMP 0 4 3 8	
0 4 2 F	3 8		
0 4 3 0	0 4		
0 4 3 1	7 A	MOV A, D	(D)→Akku
0 4 3 2	C 3	JMP 0 4 3 8	
0 4 3 3	3 8		
0 4 3 4	0 4		
0 4 3 5	7 A	MOV A, D	(D)→Akku
0 4 3 6	C 6	ADI 0 8	Korrektur der rechten Hälfte
0 4 3 7	0 8		
0 4 3 8	D 3	OUT RLAMPE	
0 4 3 9	0 1		} Ergebnis→Anzeige und
0 4 3 A	C 3	JMP 0 4 0 0	} Rücksprung zum Anfang
0 4 3 B	0 0		
0 4 3 C	0 4		

Lösungen zu Experiment 14

1. Bei Programmschritt 2 1, da bei 2 2 (HLT-Befehl) der Monitor nicht in Betrieb ist.

2.

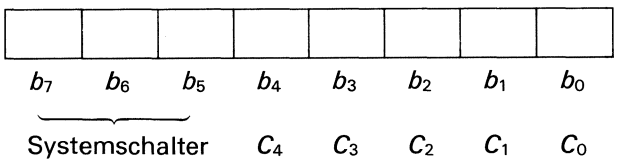
Register B	= 0 4
Register C	= 8 1
Register D	= 0 4
Register E	= 8 0
Register H	= 0 4
Register L	= 8 0
Akku	= 7 E
Speicher 0 4 8 0	= C 3
Speicher 0 4 8 1	= F F
Speicher 0 4 8 2	= 7 E

Lösung zu Experiment 15

(B) = 0 4 8 1
(D) = 0 4 8 0
(H) = 0 4 8 0
(A) = 7 E

Lösungen zu Experiment 16

1. Es besteht folgende Zuordnung:



2. Die an den *C*-Schaltern eingegebenen Zahlen werden komplementiert wiedergegeben, d.h. $7_{10} \triangleq 0\ 0\ 0_2$ oder $0_{10} \triangleq 1\ 1\ 1_2$.

3. Durch Betätigen der RESET-Taste, da hierdurch das Monitorprogramm angerufen wird.

Lösungen zu Experiment 17

1. Das C-Flag ist 1, und im Akku steht 0 E.

2. Im Akku steht jetzt das Zwischenergebnis 1 6. Das C-Flag ist zu diesem Zeitpunkt 0. Addiert man manuell das Zwischenergebnis bei der relativen Adresse 0 A (0 E) mit dem eingegebenen Wert 0 7, so lautet das neue Zwischenergebnis:

$$\begin{array}{r} 1\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 0 \\ +\quad 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1 \\ \hline 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 1 \end{array} \quad \triangleq 1\ 5_{16} \text{ mit Übertrag}$$

Im Akku steht aber als Ergebnis 0 0 0 1 0 1 1 0 ≙ 1 6 ohne Übertrag. Der Übertrag wird also durch den Befehl ADC mit in den Akku addiert.

3. Das Endergebnis beträgt bei diesem Beispiel 2 7. Dies können Sie z.B. feststellen, wenn Sie an dem *B*-Schalter F E einstellen. Manuell gerechnet müßte dann bei der rel. Adr. 0 A folgendes Ergebnis vorliegen:

$$\begin{array}{r} 0\,0\,0\,1\;0\,0\,0\,0 \\ +\;\;1\,1\,1\,1\;1\,1\,1\,0 \\ \hline 1\;0\,0\,0\,0\;1\,1\,1\,0 \\ +\;\;0\,0\,0\,0\;1\,1\,1\,1 \\ \hline 1\;0\,0\,0\,1\;1\,1\,0\,1 \end{array} \quad \triangleq 1\,D$$

Im Akku steht zwar 1 D, aber das C-Flag ist jetzt gleich 0.

Lösungen zu Experiment 18

1. Da auch der MP 8080 in Zweierkomplementarithmetik arbeitet, wird bei einer 1 im höchsten bit dieses Byte als negativ gewertet.

2.

relative Adresse	Inhalt Akku binär	hex.	Flags C-Flag	N-Flag
0 7	1 1 1 1 0 0 0 0	F 0	1	1
0 A	1 1 1 0 0 0 0 0	E 0	0	1
1 1	0 0 1 0 1 0 0 1	2 9	1	0
1 4	0 0 0 0 1 0 0 0	0 8	0	0

3. Der Befehl SBB C bewirkt, daß vom Akku-Inhalt $0\,9_{16}$ (IN ASHALT) der Inhalt des Registers C mit E 0 subtrahiert wird. E 0 entspricht einer negativen Zahl von $-2\,0_{16}$. Damit ergibt sich:

$$0\,9_{16} - (-2\,0_{16}) = 0\,9_{16} + 2\,0_{16} = 2\,9_{16}$$

Lösung zu Experiment 20

Ergebnis nach DAD D:

$$\overbrace{1\,1\,1\,1\,0\,0\,0\,1}^{\text{Reg. H}}\;\overbrace{0\,0\,1\,1\,0\,1\,0\,0}^{\text{Reg. L}}$$

$$\triangleq$$

$$F\,1\,3\,4_{16}$$

Ergebnis nach DAD B:

$$\overbrace{0\,1\,1\,0\,1\,0\,0\,0}^{\text{Reg. H}}\;\overbrace{1\,0\,1\,0\,1\,1\,0\,0}^{\text{Reg. L}}$$

$$\triangleq$$

$$6\,8\,A\,C_{16}$$

Lösungen zu Experiment 21

1. Das H-Flag wird immer dann 1. wenn die rechte Hälfte der Summe größer als 15_{10} wird.

2. Da die Summe größer als 15_{10} ist, ist das H-Flag vor der Korrektur gleich 1, nach der Korrektur gleich 0. Begründung:

$09_{10} \triangleq$	0 0 0 0 1 0 0 1	
$09_{10} \triangleq$	+ 0 0 0 0 1 0 0 1	
	0 0 0 [1] 0 0 1 –	Übertrag
	0 0 0 1 0 0 1 0	unkorrigiertes Ergebnis
	+ 0 0 0 0 0 1 1 0	Korrekturaddition durch DAA
	0 0 0 [0] 1 1 0 –	
$18_{10} \triangleq$	0 0 0 1 1 0 0 0	korrigiertes Ergebnis

3. Vor dem DAA-Befehl ist das H-Flag gleich 0, danach 1.
Begründung:

$7_{10} \triangleq$	0 0 0 0 0 1 1 1	
$8_{10} \triangleq$	+ 0 0 0 0 1 0 0 0	
	0 0 0 [0] 0 0 0 –	Übertrag
	0 0 0 0 1 1 1 1	unkorrigiertes Ergebnis (> 09)
	+ 0 0 0 0 0 1 1 0	Korrekturaddition
	0 0 0 [1] 1 1 0 –	
$15_{10} \triangleq$	0 0 0 1 0 1 0 1	korrigiertes Ergebnis

Lösung zu Experiment 22

relative Adresse	Befehl	Akku-Inhalt	Flags N	Z	H	P	C
0 5	ANI	0 0	0	1	0	1	0
0 B	ANA	0 0	0	1	1	1	0
1 0	ORI	1 F	0	0	0	0	0
1 6	ORA	9 B	1	0	0	0	0
1 B	XRI	E 2	1	0	0	1	0
2 1	XRA	9 B	1	0	0	0	0

Aus dieser Tabelle ist zu erkennen, daß das C-Flag mit allen logischen Befehlen auf 0 gesetzt wird. Mit Ausnahme des Befehles ANA trifft dies auch für das H-Flag zu. Bei ANA wird das H-Flag nicht beeinflußt. Die Flags N, Z und P werden dem Ergebnis entsprechend gesetzt.

Lösung zu Experiment 23

relative Adresse 0 7 Befehl DAD H

C-Flag	H-Register	L-Register
0	0 1 1 1 1 0 0 1	1 1 0 0 1 1 1 0
	7 9	C E

relative Adresse	Befehl	C-Flag	Akku-Inhalt
0 9	RAL	1	1 1 0 0 1 1 1 0
0 B	RLC	1	1 0 0 1 1 1 0 1
0 D	RAR	1	1 1 0 0 1 1 1 0
0 F	RAR	0	1 1 1 0 0 1 1 1
1 1	RAR	1	0 1 1 1 0 0 1 1
1 3	RRC	1	1 0 1 1 1 0 0 1

C E
9 D
C E
E 7
7 3
B 9

Lösungen zu Experiment 24

1. Bei diesem Befehl wird vom Inhalt 0 1 des Akkus 1 subtrahiert. In Zweierkomplementarithmetik bedeutet dies:

```
  0 0 0 0  0 0 0 1   Inhalt des Akkus
+ 1 1 1 1  1 1 1 1
  1 1 1 1  1 1 1 0
1 0 0 0 0  0 0 0 0
```

Damit entsteht zwischen dem 4. und 5. bit eine 1, die vom H-Flag angezeigt wird.

2. Dieser Befehl bewirkt, daß der Inhalt der Speicheradresse, die im HL-Registerpaar steht, 0 0 wird. In diesem Falle ist es die relative Adresse 2 0.

Lösungen zu Experiment 25

1. Bei der relativen Adresse 1 D ist das P-Flag = 1, da der Akku eine gerade Anzahl Einsen aufweist (0 C ≙ 0 0 0 0 1 1 0 0). Damit erfolgt ein Sprung zur relativen Adresse 8 0. Die UND-Verknüpfung mit 0 E verändert den Inhalt des Akkus nicht. Da das N-Flag = 0 ist, erfolgt ein Sprung zur relativen Adresse 0 8. Hier können über die *B*-Schalter neue Daten eingegeben werden.

2. Der Rücksprung erfolgt mit dem Befehl PCHL in der relativen Adresse 6 8.

Lösungen zu Experiment 26

1. Es erfolgt insgesamt eine Verschiebung um 4 Stellen, d.h., im Akku steht dann die Zahl $1\,0_{16}$.

2. In diesem Falle werden die Flags nach der zweiten Linksverschiebung nicht mehr entsprechend dem neuen Akku-Inhalt gesetzt. Damit kann das N-Flag noch 0 sein, obschon im werthöchsten bit des Akkus eine 1 steht, so daß eine positive Zahl vorgetäuscht wird, die einen weiteren Unterprogrammanruf auslöst.

Lösungen zu Experiment 27

1. Mit HLT am BP = 1 wird die Bearbeitung eines Programms wie bekannt mit einem RST 2-Befehl unterbrochen. Eine Unterbrechung findet ebenfalls statt, wenn der Programmzähler über RESET oder den RST 0-Befehl auf die Adresse 0 0 0 0 gesetzt wird.

2.

absolute Adresse	Inhalt	Befehl	Kommentar
0 4 0 0	C F	RST 1	rufe Unterprogramm 0 4 0 8
0 4 0 1	D 7	RST 2	
0 4 0 2	C 7	RST 0	
.	.		
.	.		
.	.		
0 4 0 8	2 1	LXI H, 0 4 2 0	
0 4 0 9	2 0		
0 4 0 A	0 4		
0 4 0 B	3 4	INR M	
0 4 0 C	C 9	RET	

Lösung zu Experiment 28

RAM-relative Adresse	Stack-Offset	Register
F 0	0	L
F 1	1	H
F 2	2	E
F 3	3	D
F 4	4	C
F 5	5	B
F 6	6	Flags
F 7	7	Akku
F 8	8	PC low
F 9	9	PC high
F A	A	Register C
F B	B	Register B
F C	C	Flags
F D	D	Akku

Wird also der Monitor nach mehreren PUSH-Befehlen durch den RST 2-Befehl angerufen, verschiebt sich der Stack-Offset pro PUSH-Befehl um 2 Adressen nach unten.

Lösungen zu Experiment 29

1. Der Stack-Pointer muß den Inhalt 0 4 3 6 haben, da durch den PUSH-Befehl die initialisierte Adresse 0 4 3 8 um 2 verringert wurde.

2.

RAM-relative Adresse	Stack-Offset	Register
2 E	0	L
2 F	1	H
3 0	2	E
3 1	3	D
3 2	4	C
3 3	5	B
3 4	6	Flags
3 5	7	Akku
3 6	8	PC low
3 7	9	PC high

3. Der Inhalt wird in den Adressen 0 4 3 7 (Reg. B) und 0 4 3 6 (Reg. C) zwischengespeichert. Da anschließend sofort ein POP-Befehl folgt, wird der Stack-Pointer wieder zur initialisierten Adresse 0 4 3 8 zurückgesetzt, so daß mit dem RST 2-Befehl die beiden Speicherplätze mit PC low und PC high überschrieben werden.

Lösungen zu Experiment 31

1. Das Flag-Wort lautet:

$$0\,1\,0\,0\;\;0\,0\,1\,0 \triangleq 4\,2_{16}$$

2.

RAM-relative Adresse	Inhalt	
F D	(H-Register)	} „alter" Registerinhalt
F C	(L-Register)	
F B	(Akku)	} „neues" PSW
F A	(Flag-Wort)	

3.

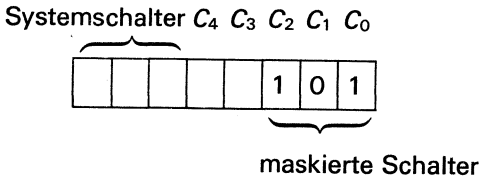
Adresse	Inhalt	Befehl	Kommentar
0 4 0 0	E 5	PUSH H	
0 4 0 1	6 7	MOV H, A	
0 4 0 2	2 E	MVI L, 9 3	
0 4 0 3	9 3		
0 4 0 4	E 3	XTHL	
0 4 0 5	F 1	POP PSW	
0 4 0 6	D 7	RST	
0 4 0 7	7 6	HLT	

Lösungen zu Experiment 32

1.

Adresse	Inhalt	Befehl	Kommentar
0 0 0 0	3 1	LXI SP, 0 4 F E	} Stackpointer mit 0 4 F E laden
0 0 0 1	F E		
0 0 0 2	0 4		
0 0 0 3	D B	IN CSHALT	} *C*-Schalter einlesen
0 0 0 4	0 4		
0 0 0 5	C 3	JMP 0 0 9 D	} unbedingter Sprung nach 0 0 9 D
0 0 0 6	9 D		
0 0 0 7	0 0		

2. Mit der Maske 0 7 und der Vergleichszahl 0 5 wird die gestellte Forderung erfüllt. 0 7 isoliert die Schalter C_0, C_1 und C_2. Die Vergleichszahl 0 5 entspricht der Schalterstellung 1 0 1.



Lösung zu Experiment 33

Adresse	Inhalt	Befehl	Kommentar
0400	D7	RST 2	
0401	0E	MVI C, 4	
0402	04		
0403	11	LXI D, 0420	
0404	20		
0405	04		
0406	21	LXI H, 0430	
0407	30		
0408	04		
0409	CD	CALL 0460	
040A	60		
040B	04		
040C	C3	JMP 0400	
040D	00		
040E	04		
.			
.			
.			
.			
0460	AF	XRA A	
0461	1A	LDAX D	
0462	8E	ADC M	
0463	12	STAX D	
0464	0D	DCR C	
0465	C8	RZ	
0466	13	INX D	
0467	23	INX H	
0468	C3	JMP 0461	
0469	61		
046A	04		

Anmerkung:
Wenn nach der Addition ADCM mit dem Befehl DAA eine Dezimalkorrektur durchgeführt wird, können auch Dezimalzahlen im 8421-Code verarbeitet werden.

	Maschinencode	Hex.-Code	Mnemonische Abkürzung			Flags N Z H P C	Bytes	Zyklen
Doppelgenauigkeits- und Dezimalarithmetikbefehle	0 0 0 0 1 0 0 1	0 9	DAD B	Double precision add	(HL) + (BC) →HL	– – – – x	1	10
	0 0 0 1 1 0 0 1	1 9	DAD D		(HL) + (DE) →HL	– – – – x	1	10
	0 0 1 0 1 0 0 1	2 9	DAD H		(HL) + (HL) →HL	– – – – x	1	10
	0 0 1 1 1 0 0 1	3 9	DAD SP		(HL) + (SP) →HL	– – – – x	1	10
	0 0 1 0 0 1 1 1	2 7	DAA	Decimal adjust Accu		x x x x x	1	4
Logische Befehle	1 0 1 0 0 s s s		ANA r	AND accumulator	(A) ∧ (s s s) →A	x x x x 0	1	4
	1 0 1 1 0 s s s		ORA r	OR accumulator	(A) ∨ (s s s) →A	x x 0 x 0	1	4
	1 0 1 0 1 s s s		XRA r	EXCLUSIVE-OR accu	(A) ∀ (s s s) →A	x x 0 x 0	1	4
	1 0 1 0 0 1 1 0	A 6	ANA M	AND memory to accu	(A) ∧ (@HL) →A	x x x x 0	1	7
	1 0 1 1 0 1 1 0	B C	ORA M	OR memory to accu	(A) ∨ (@HL) →A	x x 0 x 0	1	7
	1 0 1 0 1 1 1 0	A E	XRA M	EXCL.-OR memory to accu	(A) ∀ (@HL) →A	x x 0 x 0	1	7
	1 1 1 0 0 1 1 0	E 6	ANI	AND immediate	(A) ∧ (2. Byte) →A	x x 0 x 0	2	7
	1 1 1 1 0 1 1 0	F 6	ORI	OR immediate	(A) ∨ (2. Byte) →A	x x 0 x 0	2	7
	1 1 1 0 1 1 1 0	E E	XRI	EXCL.-OR immediate	(A) ∀ (2. Byte) →A	x x 0 x 0	2	7
Rotier- und Verschiebebefehle	0 0 0 0 0 1 1 1	0 7	RLC	Rotate left	(bit 7) →bit 0 und C	– – – – x	1	4
	0 0 0 0 1 1 1 1	0 F	RRC	Rotate right	(bit 0) →bit 7 und C	– – – – x	1	4
	0 0 0 1 0 1 1 1	1 7	RAL	Rotate left through C	(bit 7) →C; (C) →bit 0	– – – – x	1	4
	0 0 0 1 1 1 1 1	1 F	RAR	Rotate right through C	(bit 0) →C; (C) →bit 7	– – – – x	1	4
	0 0 1 0 1 0 0 1	2 9	DAD H	Double precision add	≙ linksschieben des Registerpaares H, L	– – – – x	1	10
Increment- und Decrement-Befehle	0 0 d d d 1 0 0		INR r	Increment register	(d d d) + 1 →d d d	x x x x –	1	5
	0 0 d d d 1 0 1		DCR r	Decrement register	(d d d) – 1 →d d d	x x x x –	1	5
	0 0 1 1 0 1 0 0	3 4	INR M	Increment memory	(@HL) + 1 →@HL	x x x x –	1	10
	0 0 1 1 0 1 0 1	3 5	DCR M	Decrement memory	(@HL) – 1 →@HL	x x x x –	1	10
	0 0 0 0 0 0 1 1	0 3	INX B	Increment Register pair	(BC) + 1 →BC	– – – – –	1	5
	0 0 0 1 0 0 1 1	1 3	INX D		(DE) + 1 →DE	– – – – –	1	5
	0 0 1 0 0 0 1 1	2 3	INX H		(HL) + 1 →HL	– – – – –	1	5
	0 0 0 0 1 0 1 1	0 B	DCX B	Decrement Register pair	(BC) – 1 →BC	– – – – –	1	5
	0 0 0 1 1 0 1 1	1 B	DCX D		(DE) – 1 →DE	– – – – –	1	5
	0 0 1 0 1 0 1 1	2 B	DCX H		(HL) – 1 HL	– – – – –	1	5
	0 0 1 1 0 0 1 1	3 3	INX SP	Incr. Stack-Pointer	(SP) + 1 →SP	– – – – –	1	5
	0 0 1 1 1 0 1 1	3 B	DCX SP	Decr. Stack-Pointer	(SP) – 1 →SP	– – – – –	1	5

	Maschinencode	Hex.-Code	Mnemonische Abkürzung			Flags N Z H P C	Bytes	Zyklen
Einzelgenauigkeits-Datentransferbefehle	0 1 d d d s s s		MOV r1, r2	Move register to register	(s s s) →d d d	keine Beeinflussung der Flags	1	5
	0 1 d d d 1 1 0		MOV r, M	Move from memory	(@HL) →d d d		1	7
	0 1 1 1 0 s s s		MOV M, r	Move to memory	(s s s) →@HL		1	7
	0 0 d d d 1 1 0		MVI r	Move immediate to register	(2.Byte)→d d d		2	7
	0 0 1 1 0 1 1 0	3 6	MVI M	Move immediate to memory	(2.Byte)→@HL		2	10
	0 0 1 1 0 0 1 0	3 2	STA adr	Store accumulator	(A) →(3. u. 2. Byte)		3	13
	0 0 1 1 1 0 1 0	3 A	LDA adr	Load accumulator	((3. u. 2. Byte))→A		3	13
	0 0 0 0 0 0 1 0	0 2	STAX B	Store accumulator indexed	(A) →@BC		1	7
	0 0 0 1 0 0 1 0	1 2	STAX D		(A) →@DE		1	7
	0 0 0 0 1 0 1 0	0 A	LDAX B	Load accumulator indexed	(@BC)→A		1	7
	0 0 0 1 1 0 1 0	1 A	LDAX D		(@DE)→A		1	7
Doppelgenauigkeits-Datentransferbefehle	0 0 0 0 0 0 0 1	0 1	LXI B	Load register pair immediate	(2.Byte)→C; (3. Byte)→B		3	10
	0 0 0 1 0 0 0 1	1 1	LXI D		(2.Byte)→E; (3. Byte)→D		3	10
	0 0 1 0 0 0 0 1	2 1	LXI H		(2.Byte)→L; (3. Byte)→H		3	10
	0 0 1 0 0 0 1 0	2 2	SHLD	Store H and L direct	(L) →(3. u. 2. Byte) (H) →(3. u. 2. Byte + 1)		3	16
	0 0 1 0 1 0 1 0	2 A	LHLD	Load H and L direct	((3. u. 2. Byte))→L ((3. u. 2. Byte + 1))→H		3	16
	1 1 1 0 1 0 1 1	E B	XCHG	Exchange HL with DE	(HL) →DE; (DE)→HL		1	4
Ein/Ausgabebefehle	1 1 0 1 1 0 1 1	D B	IN	Input	(Eingabekanal im 2. Byte) →A		2	10
	1 1 0 1 0 0 1 1	D 3	OUT	Output	A→Ausgabekanal im 2. Byte		2	10
Einzelgenauigkeits-arithmetikbefehle	1 0 0 0 0 s s s		ADD r	Add register	(A) + (s s s) →A	alle Flags werden entsprechend den Ergebnissen gesetzt	1	4
	1 0 0 0 0 1 1 0	1 6	ADD M	Add memory	(A) + (@HL) →A		1	7
	1 1 0 0 0 1 1 0	C 6	ADI	Add immediate	(A) + (2. Byte) →A		2	7
	1 0 0 0 1 s s s		ADC r	Add register with carry	(A) + (s s s) + (C) →A		1	4
	1 0 0 0 1 1 1 0	8 E	ADC M	Add memory with carry	(A) + (@HL) + (C) →A		1	7
	1 1 0 0 1 1 1 0	C E	ACI	Add immediate with carry	(A) + (2. Byte) + (C) →A		2	7
	1 0 0 1 0 s s s		SUB r	Subtract register	(A) − (s s s) →A		1	4
	1 0 0 1 0 1 1 0	9 6	SUB M	Subtract memory	(A) − (@HL) →A		1	7
	1 1 0 1 0 1 1 0	D 6	SUI	Subtract immediate	(A) − (2. Byte) →A		2	7
	1 0 0 1 1 s s s		SBB r	Sub. reg. with borrow	(A) − (s s s) − (C) →A		1	4
	1 0 0 1 1 1 1 0	9 E	SBB M	Sub. mem. with borrow	(A) − (@HL) − (C) →A		1	7
	1 1 0 1 1 1 1 0	D E	SBI	Sub. imm. with borrow	(A) − (2. Byte) − (C) →A		2	7
	1 0 1 1 1 s s s		CMP r	Compare register	(A) − (s s s) setzt Flags		1	4
	1 0 1 1 1 1 1 0	B E	CMP M	Compare memory	(A) − (@HL) setzt Flags		1	7
	1 1 1 1 1 1 1 0	F E	CPI	Compare immediate	(A) − (2. Byte) setzt Flags		2	7

	Maschinencode	Hex.-Code	Mnemonische Abkürzung			Flags N Z H P C	Bytes	Zyklen
Stack-Befehle	1 1 0 0 0 1 0 1	C 5	PUSH B	Push reg. pair BC	(BC) →Stack	– – – – –	1	11
	1 1 0 1 0 1 0 1	D 5	PUSH D	Push reg. pair DE	(DE) →Stack	– – – – –	1	11
	1 1 1 0 0 1 0 1	E 5	PUSH H	Push reg. pair HL	(HL) →Stack	– – – – –	1	11
	1 1 1 1 0 1 0 1	F 5	PUSH PSW	Push progr. status word	(Accu, Flags) →Stack	– – – – –	1	11
	1 1 0 0 0 0 0 1	C 1	POP B	Pop reg. pair BC	(Stack) →BC	– – – – –	1	10
	1 1 0 1 0 0 0 1	D 1	POP D	Pop. reg. pair DE	(Stack) →DE	– – – – –	1	10
	1 1 1 0 0 0 0 1	E 1	POP H	Pop reg. pair HL	(Stack →HL	– – – – –	1	10
	1 1 1 1 0 0 0 1	F 1	POP PSW	Pop progr. status word	(Stack) →Accu, Flags	x x x x x	1	10
	0 0 1 1 0 0 0 1	3 1	LXI SP	Load immediate Stack-Pointer	(3. u. 2. Byte)→SP	– – – – –	3	10
	1 1 1 1 1 0 0 1	F 9	SPHL	Stack-Pointer from HL	(HL) →SP	– – – – –	1	5
	0 0 1 1 0 0 1 1	3 3	INX SP	Incr. Stack-Pointer	(SP) + 1 →SP	– – – – –	1	5
	0 0 1 1 1 0 1 1	3 B	DCX SP	Decr. Stack-Pointer	(SP) − 1 →SP	– – – – –	1	5
	0 0 1 1 1 0 0 1	3 9	DAD SP	Double prec. add SP	(HL) + (SP) →HL	– – – – x	1	10
	1 1 1 0 0 0 1 1	E 3	XTHL	Exchange HL with top of stack	(HL) mit ((SP)) vertauschen	– – – – –	1	18
Sonstige Befehle	0 1 1 1 0 1 1 0	7 6	HLT	Halt			1	7
	0 0 0 0 0 0 0 0	0 0	NOP	No operation	**keine** Operation		1	4
	0 0 1 0 1 1 1 1	2 F	CMA	Complement accu	(A)→A		1	4
	0 0 1 1 0 1 1 1	3 7	STC	Set carry	1→C-Flag	1	1	4
	0 0 1 1 1 1 1 1	3 F	CMC	Complement carry	(C-Flag)→C-Flag	x	1	4

Registercode für s s s bzw. d d d:

0 0 0 ≙ Register B
0 0 1 ≙ Register C
0 1 0 ≙ Register D
0 1 1 ≙ Register E
1 0 0 ≙ Register H
1 0 1 ≙ Register L
1 1 0 ≙ Memory (nicht als s s s **und** d d d verwenden!)Set carry
1 1 1 ≙ Accumulator

Abkürzungen:

x = Flag wird beeinflußt
– = Flag wird nicht beeinflußt
(. . .) = Inhalt von . . .
((. . .)) = Inhalt von . . . ist die Adresse, deren Inhalt verarbeitet wird

	Maschinencode	Hex.-Code	Mnemonische Abkürzung			Flags N Z H P C	Bytes	Zyklen
Sprungbefehle	1 1 0 0 0 0 1 1	C 3	JMP	Jump unconditionally	(3. und 2. Byte) → PC	keine Beeinflussung der Flags	3	10
	1 1 0 0 0 0 1 0	C 2	JNZ	Jump if not zero	Adresse im 3. u. 2. Byte wird in den Programm-zähler gebracht, wenn: Z = 0		3	10
	1 1 0 0 1 0 1 0	C A	JZ	Jump if zero	Z = 1		3	10
	1 1 0 1 0 0 1 0	D 2	JNC	Jump if carry not set	C = 0		3	10
	1 1 0 1 1 0 1 0	D A	JC	Jump if carry set	C = 1		3	10
	1 1 1 0 0 0 1 0	E 2	JPO	Jump if parity odd	P = 0		3	10
	1 1 1 0 1 0 1 0	E A	JPE	Jump if parity even	P = 1		3	10
	1 1 1 1 0 0 1 0	F 2	JP	Jump if plus	N = 0		3	10
	1 1 1 1 1 0 1 0	F A	JM	Jump if minus	N = 1		3	10
	1 1 1 0 1 0 0 1	E 9	PCHL	Programm-Counter from HL	(HL) → PC		1	5
Unterprogrammanrufe und Rücksprungbefehle	1 1 0 0 1 1 0 1	C D	CALL	Call unconditionally	(3. und 2. Byte) → PC		3	17
	1 1 0 0 0 1 0 0	C 4	CNZ	Call if not zero	(3. u. 2. Byte) in PC Rücksprung-adresse auf den Stack, wenn: Z = 0		3	11/17
	1 1 0 0 1 1 0 0	C C	CZ	Call if zero	Z = 1		3	11/17
	1 1 0 1 0 1 0 0	D 4	CNC	Call if carry not set	C = 0		3	11/17
	1 1 0 1 1 1 0 0	D C	CC	Call if carry set	C = 1		3	11/17
	1 1 1 0 0 1 0 0	E 4	CPO	Call if parity odd	P = 0		3	11/17
	1 1 1 0 1 1 0 0	E C	CPE	Call if parity even	P = 1		3	11/17
	1 1 1 1 0 1 0 0	F 4	CP	Call if plus	N = 0		3	11/17
	1 1 1 1 1 1 0 0	F C	CM	Call if minus	N = 1		3	11/17
	1 1 0 0 1 0 0 1	C 9	RET	Return unconditionally	Rückspr.adr. vom Stack → PC		1	10
	1 1 0 0 0 0 0 0	C 0	RNZ	Return if not zero	Rücksprung-adresse vom Stack in den Programm-zähler, wenn: Z = 0		1	5/11
	1 1 0 0 1 0 0 0	C 8	RZ	Return if zero	Z = 1		1	5/11
	1 1 0 1 0 0 0 0	D 0	RNC	Return if carry not set	C = 0		1	5/11
	1 1 0 1 1 0 0 0	D 8	RC	Return if carry set	C = 1		1	5/11
	1 1 1 0 0 0 0 0	E 0	RPO	Return if parity odd	P = 0		1	5/11
	1 1 1 0 1 0 0 0	E 8	RPE	Return if parity even	P = 1		1	5/11
	1 1 1 1 0 0 0 0	F 0	RP	Return if plus	N = 0		1	5/11
	1 1 1 1 1 0 0 0	F 8	RM	Return if minus	N = 1		1	5/11
Interrupt-befehle	1 1 1 1 1 0 1 1	F B	EI	Enable interrupt	nach EI Interrupt möglich		1	4
	1 1 1 1 0 0 1 1	F 3	DI	Disable interrupt	nach DI Interrupt nicht möglich		1	4
	1 1 a a a 1 1 1		RST	Restart	(PC) → stack, a a a x 8 → PC		1	11

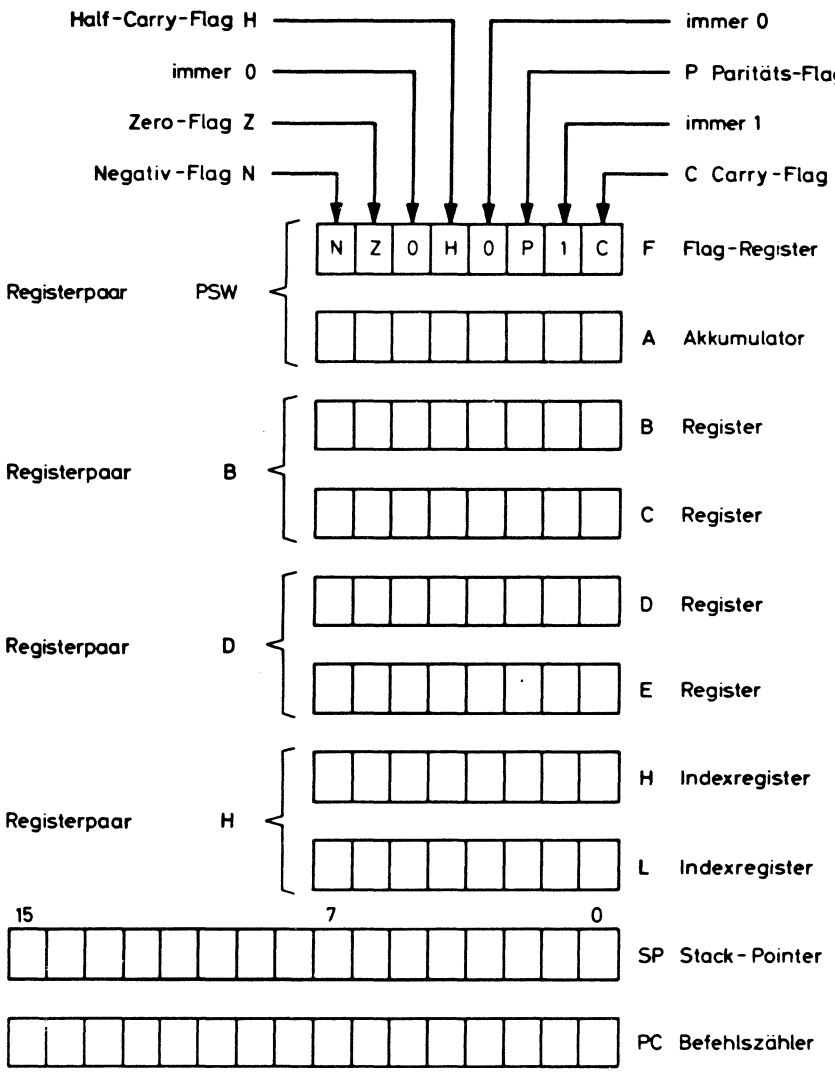


Bild 6.2.1
Die Register und Flags des MP 8080

menfassen. Diese Betriebsmöglichkeit ist notwendig, da der MP 8080 zwar mit 8 bit Datenwortlänge, jedoch mit 16 bit Adreßwortlänge arbeitet.
Das Registerpaar HL kann (ähnlich wie R3 im hypothetischen Rechner) als Indexregister eingesetzt werden.
Der Stack-Pointer SP und der Programmzähler PC sind im MP 8080 mit 16 bit Länge ausgeführt.
Da beim MP 8080 normalerweise keine Möglichkeit besteht, sich durch Anzeigen über die Inhalte der einzelnen Register zu informieren – es können ja nicht direkt Anzeigen angeschlossen werden – ist für Lern- und Prüfzwecke ein **Monitor** erforderlich. Hierbei handelt es sich um ein Programm, das dem Anwender die Möglichkeit bietet, bestimmte „innere" Daten zur Anzeige zu bringen. Damit Sie in der Lage sind, die nachfolgenden Abhandlungen experimentell nachzuvollziehen, ist es notwendig, die Möglichkeiten und die Handhabung des Monitorprogramms zu kennen. Aus diesem Grunde ist jetzt zunächst das Experiment 13 durchzuführen.

Exp. 13

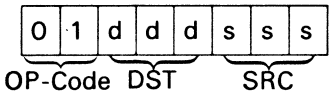
6.3 Befehlsvorrat des MP 8080

Der MP 8080 hat Befehle, die 1, 2 oder 3 Bytes benötigen. Nachfolgend werden die einzelnen Befehle besprochen. Eine Zusammenstellung aller Befehle finden Sie am Ende dieses Lehrheftes. In dieser Tabellen ist die Anzahl der Bytes sowie die Anzahl der Taktzyklen für

jeden Befehl angegeben. Da die Taktfrequenz des MP-Experimenters bei 8,867 238 MHz liegt (Quarzfrequenz) dauert eine Taktperiode 1,015 µs (Teilerverhältnis 1 : 9!). Mit der Angabe der benötigten Taktzyklen läßt sich somit die Ausführungszeit eines Befehles und somit eines Programms berechnen. Hierbei ist zu berücksichtigen, daß bei einigen Befehlen die Anzahl der Taktzyklen unterschiedlich sein kann. So benötigen z.B. die bedingten CALL-Befehle 11 oder 17 Taktzyklen, je nachdem ob die Bedingung erfüllt wurde oder nicht. Wenn ein Befehl aus mehr als einem Byte besteht, so bedeutet dies, daß nach dem eigentlichen Maschinencode (1. Byte) nachfolgende Daten oder Adressen angegeben sind.
Adressen werden immer mit 16 bit angegeben und benötigen deshalb dafür 2 Byte (sog. 3-Byte-Befehle). Die rechte Adressenhälfte steht im zweiten Byte, die linke Hälfte im dritten Byte. Bei Doppelgenauigkeitsarithmetik wird ebenfalls zuerst die rechte Hälfte und dann die linke Hälfte angegeben. Wie bereits anhand von Bild 6.2.1 angedeutet wurde, können bestimmte Register zu Registerpaaren r_P zusammengefaßt werden. So ergibt z.B. die Zusammenfassung der 8-bit-Register H und L das Registerpaar H mit einer Kapazität von 16 bit. Damit ist es möglich, 16-bit-Operationen durchzuführen. Für Stack-Operationen werden der Akku A und das Flag-Register F zum Registerpaar PSW (Programm Status Word) zusammengefaßt. Das Registerpaar PSW kann nicht für arithmetische Operationen benutzt werden. Eine Zusammensetzung anderer Register wie z.B. B und H ist nicht möglich. Nachfolgend werden aus Übersichtlichkeitsgründen Befehle und ähnliche Funktionen zusammen in Gruppen erklärt. Jede Gruppe wird durch ein kleines Experiment verdeutlicht. Diese Experimente sind nicht sehr praxisnah, da sie nur die Wirkungsweise der Befehle zeigen sollen.

6.3.1 Einzelgenauigkeits-Datentransferbefehle

In dieser Gruppe gibt es 6 unterschiedliche Befehle, die Daten zwischen 2 Registern oder zwischen einem Register und dem Speicher bewegen. Die Flags werden durch diese Befehle **nicht** beeinflußt. Der erste Befehl (MOV) hat folgendes Format:



MOV

Der MOV-Befehl bewegt (kopiert) den Inhalt des SRC-Registers in das DST-Register. Als SRC- bzw. DST-Register können der Akku A und die Register B, C, D, E, H und L verwendet werden. Für die Adressierung des SRC- und DST-Registers stehen 3 bit zur Verfügung. Damit können 8 Register spezifiziert werden. Da aber nur 7 Register angesprochen werden können, wird ein Codewort nicht benötigt. In Tabelle 6.3.1.1 ist die Zuordnung zwischen Code und Registern gezeigt.

s s s , d d d	Register
0 0 0	B
0 0 1	C
0 1 0	D
0 1 1	E
1 0 0	H
1 0 1	L
1 1 0	MEMORY
1 1 1	A (Akku)

Tab. 6.3.1.1
Registercode

Der Code 1 1 0 ist mit MEMORY gekennzeichnet. Wenn bei einem Befehl für s s s der Code 1 1 0 steht, dann wird das mit d d d gekennzeichnete Register mit dem Speicherinhalt geladen, dessen Adresse im Registerpaar H steht. Es handelt sich also um Indexed-Adressierung über Registerpaar H. Wird umgekehrt d d d durch Code 1 1 0 gebildet und in s s s ein bestimmtes Register festgelegt, wird der Registerinhalt in der Adresse gespeichert, die im Indexregisterpaar H steht. Werden s s s und d d d mit 1 1 0 belegt, so liegt als

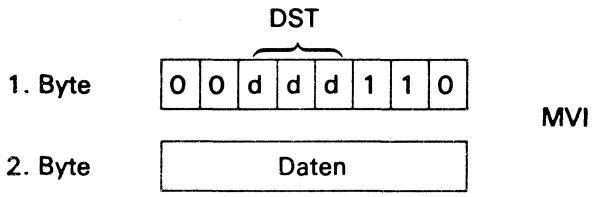
Maschinencode 0 1 1 1 0 1 1 0 ≙ 7 6_{16} vor. In Experiment 13 haben Sie diesen Befehl bereits als HLT-Befehl kennengelernt. Es ist nicht möglich, einen bestimmten Speicherplatz gleichzeitig als Datenquelle und Datenziel zu verwenden. Nachfolgend sind einige MOV-Befehle in der mnemonischen Schreibweise mit dem entsprechenden Maschinencode und Kommentar aufgeführt:

mnemonische Schreibweise	Maschinencode	Hex.-Code	Kommentar
MOV A, H	0 1 1 1 1 1 0 0	7 C	(H)→A
MOV H, A	0 1 1 0 0 1 1 1	6 7	(A)→H
MOV M, L	0 1 1 1 0 1 0 1	7 5	(L)→ @ HL
MOV L, M	0 1 1 0 1 1 1 0	6 E	(@ HL) →L

Da es für s s s und d d d 8 verschiedene Möglichkeiten gibt, sind theorethisch 64_{10} MOV-Befehle möglich.

Anmerkung:
Die mnemonische Schreibweise ist dem Datenbuch der Firma INTEL entnommen. Gegenüber der Mnemonik beim hypothetischen Rechner bestehen Unterschiede.

Der zweite Befehl dieser Gruppe MVI ist ein 2-Byte-Befehl. MVI steht für Move Immediate. Er hat das Format:

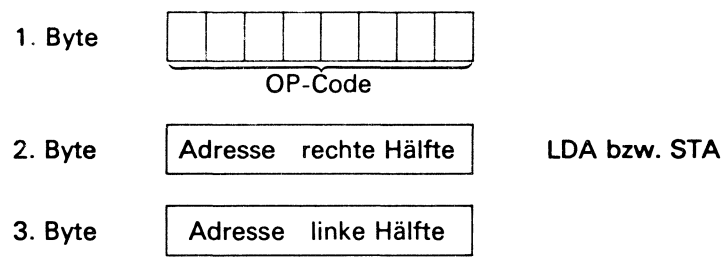


Da die Daten im 2. Byte angegeben sind, muß hier nur das Datenziel (DST) definiert werden. Dabei gilt wieder der Code nach Tab. 6.3.1.1.

Beispiele:

mnemonische Schreibweise	Maschinencode	Hex.-Code	Kommentar
MVI L	0 0 1 0 1 1 1 0	2 E	(2. Byte)→L
MVI B	0 0 0 0 0 1 1 0	0 6	(2. Byte)→B
MVI M	0 0 1 1 0 1 1 0	3 6	(2. Byte)→ @ HL

Bei dem Befehl MVI M wird der Inhalt des 2. Bytes in den durch Registerpaar H angegebenen Speicherplatz gebracht.
Alle weiteren Befehle dieser Gruppe arbeiten mit dem Akkumulator und einem Speicherplatz als Datenquelle oder Datenziel. Die Angabe von SRC oder DST entfällt also. Stattdessen muß die Speicheradresse angegeben werden. Die Befehle STA (Store Accumulator) und LDA (Load Accumulator) sind 3-Byte-Befehle. Ihr Format ist:



Mit dem Befehl STA wird der Inhalt des Akumulators unter der im 2. und 3. Byte angegebenen

Adresse abgespeichert. Bei LDA wird der Inhalt der angegebenen Adresse in den Akkumulator gebracht:

mnemonische Schreibweise	Maschinencode	Hex.-Code	Kommentar
STA	0 0 1 1 0 0 1 0	3 2	(A)→Adresse
	x x x x x x x x		} Adresse
	x x x x x x x x		
LDA	0 0 1 1 1 0 1 0	3 A	(Adresse)→A
	x x x x x x x x		} Adresse
	x x x x x x x x		

Die letzten 4 Befehle dieser Gruppe benutzen wieder den Akkumulator als Datenquelle oder Datenziel. Sie bestehen aus einem Byte, da die Adressierung mit den Registerpaaren B oder D als Indexregister erfolgt. Folgende 4 Befehle sind möglich:

mnemonische Schreibweise	Maschinencode	Hex.-Code	Kommentar
STAX B	0 0 0 0 0 0 1 0	0 2	(A)→@B, C
STAX D	0 0 0 1 0 0 1 0	1 2	(A)→@D, E
LDAX B	0 0 0 0 1 0 1 0	0 A	(@B, C)→A
LDAX D	0 0 0 1 1 0 1 0	1 A	(@D, E)→A

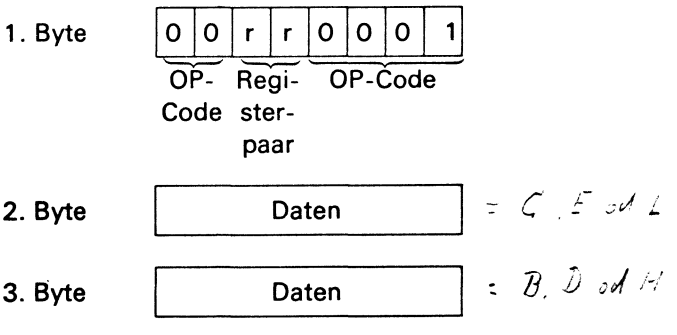
Dabei ist zu beachten, daß in den Registern B bzw. D jeweils die linke Hälfte und in C bzw. E die rechte Hälfte der Adresse steht.

Exp. 14

Die Anwendung dieser Befehlsgruppe soll Exp. 14 anzeigen.

6.3.2 Doppelgenauigkeits-Datentransferbefehle

Diese Befehlsgruppe ist für den Betrieb des MP 8080 sehr nützlich, da Datenübertragungen von 2-Byte-Wörtern (z.B. Adressen) mit nur **einem** Befehl durchgeführt werden können (In Experiment 14 waren 2 Befehle nötig, um ein Registerpaar zu laden).
Die ersten 4 Befehle haben das folgende Format:



Der Code für die Registerpaare wird in Tabelle 6.3.2.1 gezeigt.

r r		Bezeichnung
0 0	Registerpaar BC	B
0 1	Registerpaar DE	D
1 0	Registerpaar HL	H
1 1	Stack-Pointer	SP

Tab. 6.3.2.1
Registerpaarcode

Das 2. Byte wird dabei jeweils in die Speicher C, E und L bzw. in die **rechte** Hälfte des

Stack-Pointers gebracht. Das 3. Byte geht in die Register B, D, H bzw. in die **linke** Hälfte des Stack-Pointers.

Beispiele:

mnemonische Schreibweise	Maschinencode	Hex.-Code	Kommentar
LXI B	0 0 0 0 0 0 0 1	0 1	lade Registerpaar B immediate
		A A	(C) = A A } willkürlich
		B B	(B) = B B } gewählte Daten
LXI D	0 0 0 1 0 0 0 1	1 1	lade Registerpaar D immediate
		7 7	(E) = 7 7 } willkürlich
		8 8	(D) = 8 8 } gewählte Daten
LXI H	0 0 1 0 0 0 0 1	2 1	lade Registerpaar H immediate
		E E	(L) = F F } willkürlich
		F F	(H) = E E } gewählte Daten
LXI SP	0 0 1 1 0 0 0 1	3 1	lade Stack-Pointer immediate
		0 7	} (SP) = 0 9 0 7
		0 9	

Die nächsten beiden 3-Byte-Befehle werden zum direkten Laden bzw. Abspeichern des Registerpaares H verwendet. Mit dem Befehl SHLD (Store HL Direct) wird der Inhalt des L-Registers in der durch das zweite und dritte Byte spezifizierten Adresse abgelegt, während der Inhalt des H-Registers in die nächsthöhere Adresse kommt.

Beispiel:

(H) = 1 2
(L) = 3 4 } angenommener Registerinhalt

SHLD 0 4 7 0 Befehl

(0 4 7 0) = 3 4
(0 4 7 1) = 1 2 Speicherinhalte nach Ausführung des Befehles

Auch der Befehl LHLD (Load HL Direct) soll durch ein Beispiel erläutert werden.

Beispiel:

(0 4 5 0) = A B
(0 4 5 1) = C D } angenommene Speicherinhalte

LHLD 0 4 5 0 Befehl

(H) = C D
(L) = A B } Registerinhalte nach Ausführung des Befehles

Der letzte Befehl dieser Gruppe ist wieder ein 1-Byte-Befehl, der den Austausch der Inhalte der Registerpaare H und D auslöst:

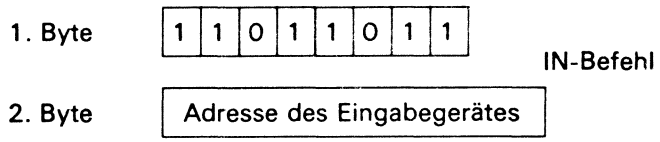
mnemonische Schreibweise	Maschinencode	Hex.-Code	Kommentar
XCHG	1 1 1 0 1 0 1 1	E B	(H)→D (D)→H (L)→E (E)→L

Im Experiment 15 sollen auch diese Befehle zum Einsatz gebracht werden.

Exp. 15

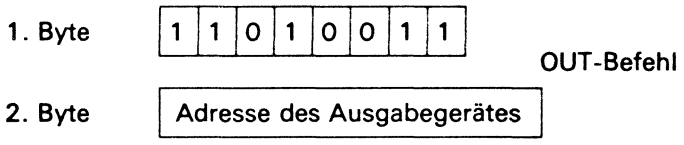
6.3.3 Ein- und Ausgabebefehle

Damit das MP-System Daten ein- und auslesen kann, werden der IN- und der OUT-Befehl benötigt. Bei beiden Befehlen wird der Akkumulator grundsätzlich als Datenziel oder Datenquelle benutzt. Nachfolgend ist das Format des IN-Befehles dargestellt:



Im zweiten Byte steht eine Adresse, die angibt, welches Eingabegerät Daten eingeben muß. Mit einem Takt wird dieses zweite Byte auf den Adreßbus gelegt. Das angesprochene Eingabegerät erkennt diesen Code und liefert jetzt Daten auf den Datenbus, die dann vom Akkumulator übernommen werden.
Bedingt durch die 8 bit können 256_{10} Eingabegeräte mit dem MP 8080 verbunden werden. Welche Adresse welchem Gerät zugeordnet ist, bleibt dem Systementwickler überlassen.

Der OUT-Befehl hat ein ähnliches Format:



Bei der Ausführung dieses Befehles gibt der MP 8080 Daten auf den Datenbus und die Adresse des Ausgabegerätes auf den Adreßbus. Das angesprochene Ausgabegerät übernimmt die Daten vom Datenbus.
Beide Befehle haben keinen Einfluß auf die Flags.
Auch die Anzeigen und Schalter des MP-Experimenters sind in diesem Sinne Ein- und Ausgabegeräte. Bei der Auslegung des Systems wurden folgende Adressen festgelegt:

Peripheriegerät	mnemonische Abkürzung	Adresse	
A-Schalter	ASHALT	02_{16}	für IN
B-Schalter	BSHALT	01_{16}	
C- und Systemschalter	CSHALT	04_{16}	
rechte Anzeige (R_7 bis R_0)	RLAMPE	01_{16}	für OUt
linke Anzeige (L_7 bis L_0)	LLAMPE	02_{16}	

Exp. 16

Wenn man diese Zuordnung kennt, kann man über ein einfaches Programm den gesamten ROM- und RAM-Bereich zur Anzeige bringen. Mit dem beschriebenen Monitorprogramm läßt sich ja nur der RAM-Bereich kontrollieren.
Nachfolgendes Programm kann hierfür verwendet werden:

relative Adresse	Inhalt	Befehl	Kommentar
0 0	D B	IN ASHALT	(*A*-Schalter)→Akku
0 1	0 2		
0 2	6 7	MOV H, A	(Akku)→H-Register
0 3	D B	IN BSHALT	(*B*-Schalter)→Akku
0 4	0 1		
0 5	6 F	MOV L, A	(Akku)→L-Register
0 6	7 E	MOV A, M	(@ HL)→Akku
0 7	D 3	OUT RLAMPE	(Akku)→rechte Anzeige R_7 bis R_0
0 8	0 1		
0 9	C 3	JMP 0 4 0 0	Sprung zur absoluten Adresse 0 4 0 0 ≙ relative Adresse 0 0
0 A	0 0		
0 B	0 4		

Die Funktion dieses Programms ist leicht überschaubar. Die Daten der *A*- und *B*-Schalter werden über 2 IN-Befehle in den Akku geladen. Durch die MOV-Befehle MOV H, A und MOV L, A gelangen diese Daten in das Registerpaar H (Register H und L). Der Befehl MOV A, M bewirkt, daß die Daten in den Akku geladen werden, deren Adresse im Registerpaar H steht. Der Befehl OUT RLAMPE bringt dann den Akku-Inhalt in der rechten Lampenreihe R_7 bis R_0 zur Anzeige. In einem späteren Versuch werden wir auf dieses Programm noch zurückkommen.

6.3.4 Einzelgenauigkeitsarithmetikbefehle

In dieser Gruppe gibt es 4 Additionsbefehle, 4 Subtraktionsbefehle und 2 Vergleichsbefehle. Alle Befehle dieser Gruppe beeinflussen die Flags entsprechend dem Ergebnis. Die Vergleichsbefehle sind den Subtraktionsbefehlen ähnlich und beeinflussen nur die Flags. Das Ergebnis der Subtraktion wird nicht in den Akku zurückgeschrieben. Die einzelnen Befehle werden nachfolgend beschrieben.
Der ADD-Befehl addiert den Inhalt des SRC-Registers zum Akku-Inhalt. Er hat folgendes Format:

1	0	0	0	0	s	s	s

ADD

Mit s s s kann eines der in Tab. 6.3.1.1 genannten Register spezifiziert werden. So bedeutet z.B. der Maschinencode

1 0 0 0 0 0 1 1

daß der Inhalt des E-Registers in den Akku addiert wird. Die mnemonische Schreibweise lautet daher ADD E.
Der Befehl ADD M mit dem Maschinencode

1 0 0 0 0 1 1 0

addiert den Inhalt der Speicheradresse, die im Registerpaar HL steht, in den Akku.
Der Befehl ADD A mit dem Maschinencode

1 0 0 0 0 1 1 1

addiert den Akku-Inhalt mit sich selbst, d.h., der Akku-Inhalt wird verdoppelt.
Der Befehl ADI (Add Immediate) addiert den Inhalt des 2. Bytes zum Akku-Inhalt. Er hat das Format:

1	1	0	0	0	1	1	0

ADI

DATEN

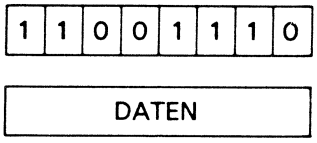
Die bisher besprochenen Additionsbefehle sind mehr oder weniger identisch mit den Additionsbefehlen des hypothetischen Rechners. Dagegen sind die beiden nachfolgenden Additionsbefehle neu.
Der Befehl ADC (Add with Carry) bewirkt, daß zusätzlich zum ADD-Befehl noch der Inhalt des C-Flags in den Akku addiert wird. Er hat das Format:

1	0	0	0	1	s	s	s

ADC

Würde z.B. eine Addition mit dem ADD-Befehl im Akku ein Ergebnis von 6 8 erzeugen, so würde das Ergebnis mit dem Befehl ADC 6 9 betragen, wenn das C-Flag 1 ist. Im anderen Falle, also bei C = 0, würde auch dieses Ergebnis 6 8 lauten. Dieser Befehl wird für Arithmetik mit mehrfacher Genauigkeit benutzt.

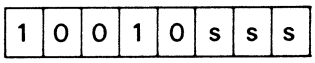
Der Befehl ACI (Add with Carry Immediate) ist ähnlich. Er addiert den Inhalt des zweiten Bytes und den Inhalt des C-Flags in den Akku. Er hat das Format:



ACI

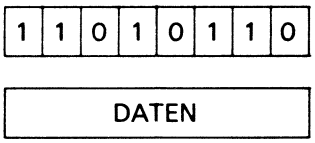
Exp. 17

Entsprechend den 4 Additionsbefehlen gibt es auch 4 Subtraktionsbefehle. Der SUB-Befehl subtrahiert den Inhalt des SRC-Registers vom Inhalt des Akkus und schreibt das Ergebnis in den Akku zurück. Er hat das Format:



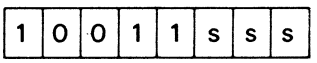
SUB

So subtrahiert z.B. der Befehl SUB L den Inhalt des L-Registers vom Inhalt des Akkus. Ein Sonderfall ist der Befehl SUB A. Dieser Befehl subtrahiert den Akku-Inhalt vom Akku-Inhalt, d.h., er löscht den Akku.
Der Befehl SUI (Subtrakt Immediate) subtrahiert den Inhalt des zweiten Bytes vom Inhalt des Akkus. Er hat folgendes Format:



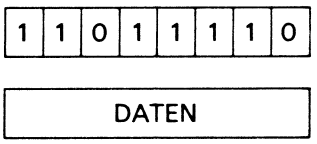
SUI

Der Befehl SBB (Subtract with Borrow = Subtrahiere mit Borge bzw. Übertrag) subtrahiert den Inhalt des SRC-Registers und den Inhalt des C-Flags vom Inhalt des Akkumulators. Er hat das Format:



SBB

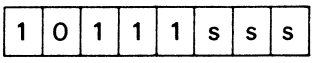
Der Befehl SBI (Subtract with Borrow Immediate) subtrahiert den Inhalt des zweiten Bytes sowie den Inhalt des C-Flags vom Inhalt des Akkumulators. Er hat das Format:



SBI

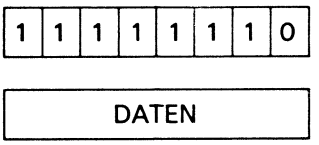
Exp. 18

Die Befehle SBB und SBI werden in ähnlicher Art wie die ADC- und ACI-Befehle für Arithmetik mit mehrfacher Genauigkeit benutzt.
Der CMP-Befehl (Compare oder Vergleich) subtrahiert wie der SUB-Befehl den Inhalt des SRC-Registers vom Inhalt des Akkus. Er schreibt aber das Ergebnis nicht in den Akku zurück, d.h., der Inhalt des Akkus bleibt erhalten. Die Flags werden wie bei den Subtraktionsbefehlen beeinflußt. Damit ist es möglich, 2 Zahlen in ihrer Größe zu vergleichen, ohne diese dabei zu verändern. Der CMP-Befehl hat das Format:



CMP

Der Befehl CPI (Compare Immediate) subtrahiert den Inhalt des zweiten Bytes vom Inhalt des Akkus und setzt die Flags entsprechend. Auch hier wird der Inhalt des Akkus nicht verändert. Sein Format ist:



CPI

Die Funktion läßt sich anhand von Experiment 19 erkennen.

Exp. 19

6.3.5 Doppelgenauigkeitsarithmetikbefehle

Mit den Befehlen DAD B und DAD D (Double precision Add) können die Inhalte der Registerpaare BC bzw. DE in das Registerpaar HL addiert werden. Dies entspricht einer Addition mit 16 bit Wortlänge. Beide Befehle beeinflussen das C-Flag. Sie haben das Format:

0	0	0	0	1	0	0	1	DAD B
0	0	0	1	1	0	0	1	DAD D

Exp. 20

6.3.6 Dezimalarithmetik

In Lehrheft 1, Abschnitt 2.7 wurde gezeigt, wie Dezimalzahlen in BCD-Zahlen umgewandelt werden können. Dies wird beim MP 8080 durch den Befehl DAA (Decimal Adjust Accumulator) erreicht. Er hat das Format:

0	0	1	0	0	1	1	1	DAA

Grundsätzlich kann der MP 8080 arithmetische Operationen nur in Binärarithmetik durchführen. Erfolgt die Eingabe zweier Operanden im BCD-Code, so kann beispielsweise bei einer Addition nach folgenden Fällen unterschieden werden:

1. Die Summe der BCD-Zahlen ist in jeder Stelle kleiner als 10_{10}.
2. Die Summe der BCD-Zahlen ist in einer oder in beiden Stellen größer als 9.

Im ersten Falle kann das Binärergebnis als Dezimalergebnis im BCD-Code gelesen werden.

Beispiel:

Dezimal		**BCD-Eingabe**	
32_{10}	≙	0 0 1 1 0 0 1 0	
27_{10}	≙	+ 0 0 1 0 0 1 1 1	
		0 1 0 0 1 1 0 –	Übertrag
59_{10}	≙	0 0 1 0 1 1 0 0 1	binäres Ergebnis

Das binäre Ergebnis entspricht bei BCD-Betrachtung dem richtigen Ergebnis. Würden bei diesem Beispiel die Eingaben und das Ergebnis binär betrachtet, ergäbe sich folgende Addition:

0 0 1 1 0 0 1 0	≙	50_{10}
+ 0 0 1 0 0 1 1 1	≙	39_{10}
0 1 0 0 1 1 0 –		
0 1 0 1 1 0 0 1	≙	89_{10}

Im zweiten Falle kann das Binärergebnis erst nach einer Korrekturaddition mit 0 1 1 0 als BCD-Ergebnis gelesen werden.

Beispiele:

a)

Dezimal		**BCD-Eingabe**	
37_{10}	≙	0 0 1 1 0 1 1 1	
44_{10}	≙	+ 0 1 0 0 0 1 0 0	
		0 0 0 0 1 0 0 –	Übertrag
		0 1 1 1 1 0 1 1	binäres Ergebnis
		≙	
		7	keine BCD-Zahl

Damit dieses Ergebnis als richtige BCD-Zahl erscheint, muß die Korrekturaddition mit 0 1 1 0 erfolgen:

```
  0 1 1 1  1 0 1 1
+ 0 0 0 0  0 1 1 0   Korrekturzahl
  1 1 1 1  1 1 0 –   Übertrag
  1 0 0 0  0 0 0 1
    ≙        ≙
   8₁₀      1₁₀
```

Damit liegt das richtige BCD-Ergebnis vor.

b)

Dezimal		**BCD-Eingabe**
69_{10}	≙	0 1 1 0 1 0 0 1
78_{10}	≙	+ 0 1 1 1 1 0 0 0
		1 1 1 1 0 0 0 – Übertrag
		1 1 1 0 0 0 0 1
		keine BCD-Zahl 1_{10} → falsch

In diesem Beispiel ist zwar die rechte Ergebnishälfte eine BCD-Zahl, jedoch vom Ergebnis her falsch. Dies wird durch einen Übertrag von der 4. zur 5. Stelle angezeigt (H-Flag gleich 1). In diesem Falle muß also in beiden Ergebnishälften die Korrekturzahl 0 1 1 0 addiert werden:

```
  1 1 1 0  0 0 0 1
+ 0 1 1 0  0 1 1 0   Korrekturzahl
  1 1 0 0  0 0 0 –   Übertrag
1|0 1 0 0  0 1 1 1
    ≙        ≙
   4₁₀      7₁₀
```

Der entstandene Übertrag hat die Wertigkeit $10^2 = 100_{10}$, so daß als Ergebnis die richtige Zahl 147_{10} entsteht.

c)

Dezimal		**BCD-Eingabe**
98_{10}	≙	1 0 0 1 1 0 0 0
98_{10}	≙	+ 1 0 0 1 1 0 0 0
		0 0 1 1 0 0 0 – Übertrag
		1|0 0 1 1 0 0 0 0
		3_{10} 0_{10} → falsch

In diesem Falle bedingen C- und H-Flag, daß beide Ergebnishälften korrigiert werden.

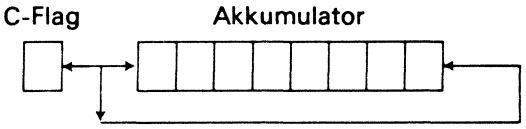
```
1|0 0 1 1  0 0 0 0
+ 0 1 1 0  0 1 1 0   Korrekturzahl
  1 1 0 0  0 0 0 –   Übertrag
1|1 0 0 1  0 1 1 0
    ≙        ≙
   9₁₀      6₁₀
```

Das richtige Ergebnis lautet durch diese Korrektur 196_{10}.

Anhand der Beispiele zeigt sich, daß eine Korrektur immer dann erforderlich ist, wenn
- eine Ergebnishälfte größer als 9_{10} ist,
- das H-Flag gleich 1 ist oder/und wenn
- das C-Flag gleich 1 ist.

Der DAA-Befehl bewirkt entsprechend den vorgenannten Kriterien, daß eine Korrektur automatisch durchgeführt wird. Dadurch ist es möglich, im zugelassenen BCD-Bereich BCD-

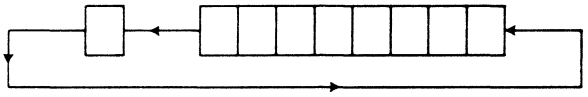
RLC-Befehl (Rotate Left into Carry)
Die Wirkung dieses Befehles läßt sich am besten anhand einer Skizze erläutern:



Das links stehende bit des Akkumulators wird in das C-Flag **und** gleichzeitig in das rechte bit des Akkumulators geschoben. Dieser Befehl hat folgendes Format:

0	0	0	0	0	1	1	1	RLC

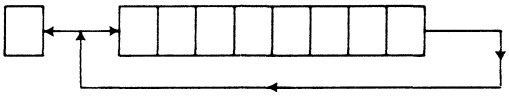
RAL-Befehl (Rotate Accu Left)
Das Funktionsprinzip dieses Befehles ist:



Hier wird das links stehende bit ebenfalls in das C-Flag geschoben. In das rechte bit des Akkumulators wird jedoch das C-Flag geschoben, so daß hier insgesamt 9 bit verschoben werden. Das Format dieses Befehles ist:

0	0	0	1	0	1	1	1	RAL

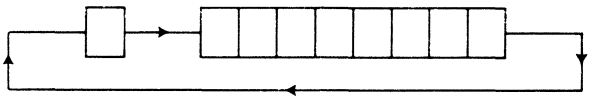
RRC-Befehl (Rotate Right into Carry)
Bei diesem Befehl erfolgt die Verschiebung nach folgendem Schema:



Die Verschiebung ist bei diesem Befehl entgegengesetzt zum RLC-Befehl. Sein Format ist:

0	0	0	0	1	1	1	1	RRC

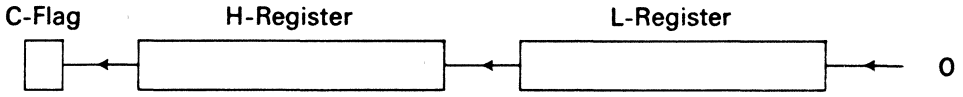
RAR-Befehl (Rotate Accu Right)
Das Funktionsprinzip zeigt, daß dieser Befehl das Gegenstück zum RAL-Befehl ist.



Format:

0	0	0	1	1	1	1	1	RAR

DAD H-Befehl (Double precision Add)
Dieser Befehl kann mit DAD B und DAD D im Abschnitt 6.3.5 verglichen werden. Er addiert den Inhalt des Registerpaares HL in das Registerpaar HL, d.h., er **verdoppelt** den Inhalt von HL. Eine Verdoppelung einer Binärzahl entspricht jedoch auch einer Verschiebung dieser Zahl um eine Stelle nach links. Aus diesem Grunde haben wir diesen Befehl in die Gruppe Rotier- und Verschiebebefehle eingeordnet.
Dieser Befehl hat **keine** Rückkopplung zu einem Ring.
Sein Funktionsprinzip ist:



Sein Format ist:

0	0	1	0	1	0	0	1

DAD H

Exp. 23

6.3.9 Increment- und Decrement-Befehle

Mit den Befehlen dieser Gruppe ist es möglich, die vorhandenen Register oder die Registerpaare BC, DE und HL zu in- und decrementieren.

INR-Befehl (Increment Register)

Dieser Befehl incrementiert das durch d d d spezifizierte Register. Bei d d d = 1 1 0 wird der Speicherplatz incrementiert, dessen Adresse im Registerpaar HL steht. Das Format dieses Befehles ist:

0	0	d	d	d	1	0	0

INR

DCR-Befehl (Decrement Register)

Mit diesem Befehl können wie mit INR alle Register oder Speicherplätze decrementiert werden. Format:

0	0	d	d	d	1	0	1

DCR

Mit dem INR- und dem DCR-Befehl werden alle Flags **außer** dem C-Flag beeinflußt. Die weiteren Befehle beziehen sich jeweils auf Registerpaare und beeinflussen die Flags **nicht**.

INX-Befehl (Increment Register Pair)

Mit diesem Befehl können die eingangs erwähnten Registerpaare incrementiert werden. Format:

0	0	r	r	0	0	1	1

INX

Es ergeben sich folgende Möglichkeiten:

r r = 0 0	INX B	Registerpaar BC wird incrementiert
r r = 0 1	INX D	Registerpaar DE wird incrementiert
r r = 1 0	INX H	Registerpaar HL wird incrementiert
r r = 1 1	INX SP	der Stack-Pointer wird incrementiert

Auf den Befehl INX SP werden wir bei der Behandlung der Stack-Befehle noch zurückkommen.

DCX-Befehl (Decrement Register Pair)

Dieser Befehl ist das Gegenstück zum INX-Befehl. Sein Format ist:

0	0	r	r	1	0	1	1

DCX

Für r r gilt die gleiche Zuordnung wie beim INX-Befehl.
Beide Befehle werden häufig zur Listenverarbeitung benutzt.

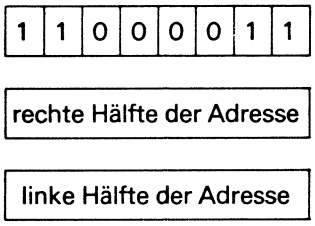
Exp. 24

6.3.10 Sprungbefehle

Der MP 8080 hat insgesamt 10 unterschiedliche Sprungbefehle. Bis auf eine Ausnahme handelt es sich um 3-Byte-Befehle, da der MP 8080 ja mit 16-bit-Adressen arbeitet. Alle Sprungbefehle beeinflussen die Flags nicht. Jedoch ist es so, daß 8 Befehle durch den Zustand der Flags gesteuert werden.

JMP-Befehl

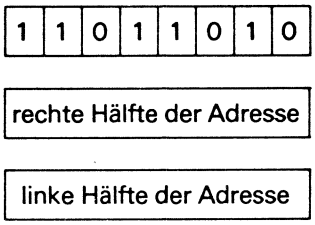
Diesen Befehl haben wir in den vorhergehenden Programmen schon mehrfach benutzt. Es handelt sich um einen unbedingten Sprungbefehl mit folgendem Format:



So bedeutet z.B. der Sprungbefehl JMP 0 4 3 0, daß der Befehlszähler auf die absolute Adresse 0 4 3 0 bzw. auf die relative Adresse 3 0 springt.
Die meisten der 8 Sprungbefehle werden durch bestimmte Zustände der Flags initialisiert.

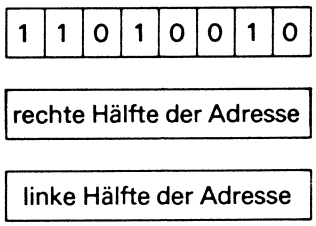
JC-Befehl (Jump if Carry)

Dieser Befehl führt dann einen Sprung aus, wenn das C-Flag gleich 1 ist.
Format:



JNC-Befehl (Jump if No Carry)

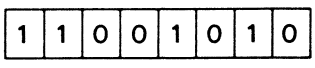
Dieser Befehl löst dann einen Sprung aus, wenn das C-Flag **nicht** 1 sondern 0 ist. Er hat das Format:



Die weiteren 6 bedingten Sprungbefehle haben alle das gleiche Format (3-Byte-Befehle). Wir werden deshalb nur noch die verschiedenen Codes angeben.

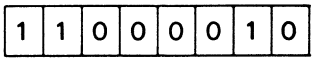
JZ-Befehl (Jump if Zero)

Ein Sprung wird ausgelöst, wenn das Z-Flag gleich 1 ist.

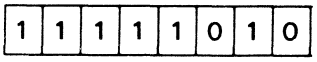


JNZ-Befehl (Jump if Not Zero)

Ein Sprung wird ausgelöst, wenn das Z-Flag gleich 0 ist.

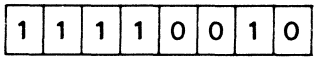


JM-Befehl (Jump if Minus)
Ein Sprung wird ausgelöst, wenn das N-Flag gleich 1 ist.



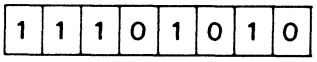
JM

JP-Befehl (Jump if Plus)
Ein Sprung wird ausgelöst, wenn das N-Flag gleich 0 ist.



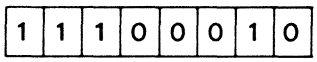
JP

JPE-Befehl (Jump if Parity Even)
Ein Sprung wird ausgelöst, wenn das P-Flag gleich 1 ist.



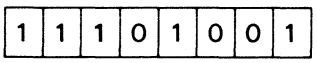
JPE

JPO-Befehl (Jump if Parity Odd)
Ein Sprung wird ausgelöst, wenn das P-Flag gleich 0 ist.



JPO

Alle bisher angesprochenen Sprungbefehle benutzen direkte Adressierung. Diese Befehle sind daher für Computed-JUMP nicht geeignet. Hierfür eignet sich der PCHL-Befehl (Program Counter from Hand L). Es handelt sich um einen 1-Byte-Befehl mit dem Format:



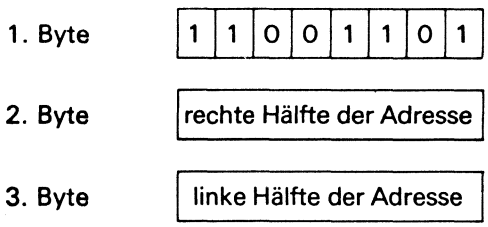
PCHL

Dieser Befehl lädt den Programmzähler mit dem Inhalt des Registerpaares HL, d.h., es erfolgt ein Sprung zu der Adresse, die in HL steht.

Exp. 25

6.3.11 Unterprogrammanrufe und Rücksprungbefehle

Beim MP 8080 werden Unterprogrammrücksprungadressen wie beim hypothetischen Rechner in einem Push-Down-Stack aufbewahrt. Aus diesem Grunde enthält der MP 8080 einen Stack-Pointer SP, der automatisch immer in Auto-Increment- bzw. Auto-Decrement-Mode betrieben wird. Der Stack-Pointer SP muß am Anfang zuerst initialisiert werden, bevor Unterprogramme angerufen werden können. Da Adressen immer 16 bit benötigen, müssen sie in 2 Hälften auf dem Stack abgespeichert werden. Der Stack arbeitet also immer mit 16-bit-Wörtern, so daß der Stack-Pointer SP immer in 2 Schritten incrementiert bzw. decrementiert werden muß.
Wie beim hypothetischen Rechner erfolgt der Unterprogrammanruf mit CALL-Befehlen. Alle CALL-Befehle sind grundsätzlich 3-Byte-Befehle, da ja 2 Bytes für die Adresse benötigt werden. Der **unbedingte Unterprogrammanruf** geschieht mit dem CALL-Befehl. Er hat das Format:



CALL

Bei diesem Befehl wird zunächst die Rücksprungadresse im Stack abgespeichert, und dann

erfolgt ein Sprung zu der Adresse, die im 2. und 3. Byte definiert ist. Der Stack-Pointer wird dabei um 2 erniedrigt.
Entsprechend den bedingten Sprungbefehlen gibt es auch **bedingte CALL-Befehle**. Diese haben alle das gleiche Format wie der CALL-Befehl.

CC-Befehl (Call if Carry)
Es erfolgt dann ein Unterprogrammanruf, wenn das Carry-Flag gleich 1 ist. Maschinencode:

1 1 0 1 1 1 0 0

CNC-Befehl (Call if No Carry)
Es erfolgt dann ein Unterprogrammanruf, wenn das Carry-Flag gleich 0 ist. Maschinencode:

1 1 0 1 0 1 0 0

CZ-Befehl (Call if Zero)
Es erfolgt dann ein Unterprogrammanruf, wenn das Zero-Flag gleich 1 ist. Maschinencode:

1 1 0 0 1 1 0 0

CNZ-Befehl (Call if Not Zero)
Es erfolgt dann ein Unterprogrammanruf, wenn das Zero-Flag gleich 0 ist. Maschinencode:

1 1 0 0 0 1 0 0

CM-Befehl (Call if Minus)
Es erfolgt dann ein Unterprogrammanruf, wenn das Negativ-Flag gleich 1 ist. Maschinencode:

1 1 1 1 1 1 0 0

CP-Befehl (Call if Plus)
Es erfolgt dann ein Unterprogrammanruf, wenn das Negativ-Flag gleich 0 ist. Maschinencode:

1 1 1 1 0 1 0 0

CPE-Befehl (Call if Parity Even)
Es erfolgt dann ein Unterprogrammanruf, wenn das Paritäts-Flag gleich 1 ist. Maschinencode:

1 1 1 0 1 1 0 0

CPO-Befehl (Call if Parity Odd)
Es erfolgt dann ein Unterprogrammanruf, wenn das Paritäts-Flag gleich 0 ist. Maschinencode:

1 1 1 0 0 1 0 0

Alle CALL-Befehle beeinflussen die Flags nicht.
Der Rücksprung von einem Unterprogramm erfolgt mit dem sogenannten RET-Befehl (RET von Return = zurück). Es handelt sich dabei um eine besondere Art eines Sprungbefehles, da die normalen Sprungbefehle den Stack-Pointer nicht als Autoindexregister verwenden können. Er hat das Format:

1	1	0	0	1	0	0	1	RET

Bei diesem Befehl wird die zuletzt abgespeicherte Rücksprungadresse vom Stack in den Programmzähler PC geladen. Der Stack-Pointer SP wird dabei um 2 erhöht.
Außer dem unbedingten Rücksprungbefehl RET hat der MP 8080 auch noch 8 bedingte Rücksprungbefehle. Diese sind

RC (Return if Carry); Code: 1 1 0 1 1 0 0 0

RNC (Return if No Carry); Code: 1 1 0 1 0 0 0 0
RZ (Return if Zero); Code: 1 1 0 0 1 0 0 0
RNZ (Return if Not Zero); Code: 1 1 0 0 0 0 0 0
RM (Return if Minus); Code: 1 1 1 1 1 0 0 0
RP (Return if Plus); Code: 1 1 1 1 0 0 0 0
RPE (Return if Parity Even); Code: 1 1 1 0 1 0 0 0
RPO (Return if Parity Odd); Code: 1 1 1 0 0 0 0 0

Auch die Return-Befehle beeinflussen die Flags nicht.

Exp. 26

6.3.12 Interrupts

Unter Interrupt versteht man eine Programmunterbrechung, die von einem peripheren Gerät ausgelöst wird bzw. ausgelöst werden kann. Im Gegensatz zu einem Unterprogrammanruf mit einem CALL-Befehl wird bei einem Interrupt unter noch näher zu definierenden Bedingungen das Hauptprogramm durch ein externes Steuersignal unterbrochen und dann ein bestimmtes Interruptprogramm ausgeführt. Anschließend wird das Hauptprogramm fortgesetzt. Außerdem kann die CPU an jeder beliebigen Stelle durch ein Interruptsignal bei der Ausführung eines Programms unterbrochen werden, d.h., die externen Steuersignale können völlig asynchron zum CPU-Takt anfallen. Die CPU prüft vor jedem Befehl, ob eine Interruptanforderung vorliegt. Ist dies der Fall, so wird die Interruptbehandlung durchgeführt. Liegt keine Interruptanforderung vor, so wird der Befehl vollständig abgearbeitet, so daß ein Interrupt frühestens vor dem nächsten Befehl erfolgen kann.
Beim MP 8080 erfolgt eine Interruptanforderung durch ein 1-Signal am INT-Eingang (Interrupt request). Die CPU akzeptiert einen Interrupt nur dann, wenn das Interruptflipflop in der CPU auf 1 gesetzt ist, d.h. der INTE-Ausgang (INTE = Interrupt Enable = Interruptfreigabe) gleich 1 ist. Damit der Rechner bei der Bearbeitung eines Programms einen Interrupt aktzeptieren kann, muß das INTE-Flipflop durch einen EI-Befehl (Enable Interrupt) gesetzt werden. Dieser Befehl hat das Format:

| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |

EI

Soll dagegen die Bearbeitung wichtiger Programmteile nicht durch Interrupts unterbrochen werden, so kann dies durch den DI-Befehl (Disable Interrupt) mit Format

| 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |

DI

verhindert werden.
Hierbei ist noch zu berücksichtigen, daß bei Betätigung von RESET das INTE-Flipflop auf 0 gesetzt wird, so daß danach kein Interrupt akzeptiert wird.
In Abschnitt 6.1 wurde auch noch ein INTA-Signal (Interrupt Acknowledge) angesprochen. Dieses Steuersignal zeigt an, ob der Rechner eine Programmunterbrechung akzeptiert hat. Es quittiert also einen Interrupt und wird dazu benutzt, der Schaltung, von der die Interruptanforderung kam, die Anschaltung an das Bussystem zu ermöglichen.
Erfolgt nun von einem Peripheriegerät bei INTE = 1 eine Interruptanforderung, so ergibt sich folgender prinzipieller Ablauf:
- Der gerade ausgeführte Befehl wird beendet
- Das INTE-Flipflop wird zurückgesetzt (INTE = 0)
- Das unterbrechende Peripheriegerät erhält über INTA eine Quittung für den Interruptzustand des Rechners und liefert einen entsprechenden Befehl, der das vorgesehene Interruptprogramm anwählt. Dieses Interruptprogramm kann grundsätzlich irgendwo im Speicher stehen.

Bei dem hier beschriebenen Ablauf wird **vor der Ausführung** dieses Befehles der Programmzähler PC nicht erhöht. Soll dieser Befehl allerdings den Programmzähler mit einer bestimmten Speicheradresse laden (1. Adresse des entsprechenden Interruptprogramms), so ist hierfür ein 3-Byte-Befehl erforderlich. Ein solcher Befehl ist natürlich von dem Peripheriegerät nur

umständlich zu erzeugen, so daß normalerweise von der Gerätesteuerung ein RST-Befehl (Restart) geliefert wird. Dieser Befehl ist ein 1-Byte-Befehl, der sich für Interruptbetrieb besonders gut eignet. Den Befehl RST 2 haben wir bisher schon sehr häufig für das Monitorprogramm benutzt. An einem Programmbeispiel soll der Ablauf eines Interrupts noch näher erläutert werden (Tab. 6.3.12.1).

Adresse	Inhalt	Befehl	Kommentar
0 4 2 0	D B	IN BSHALT	
0 4 2 1	0 1		
0 4 2 2	A 7	ANA A	Interruptanforderung von Gerät *x*
0 4 2 3	0 7	RLC	Gerät *x* sendet RST-Befehl (z.B. RST 0) ↓ Der RST-Befehl bewirkt, daß der Inhalt des Programmzählers (0 4 2 3) im Stack abgespeichert wird. Damit ist die Rücksprungadresse fixiert ↓ Bei RST 0 wird dann der Programmzähler auf Adresse 0 0 0 0 gesetzt
0 0 0 0	*x x*	1. Befehl	
0 0 0 1	*x x*	2. Befehl	
.	Für den Fall, daß beim RST 0-Befehl die 8 zur Vefügung stehenden Bytes nicht zur Abarbeitung des Interrupts ausreichen, kann über CALL- oder JMP-Befehle ein anderer Speicherbereich angesprochen werden		
.			
.			
.			
.			
.			
0 0 0 7	C 9	RET	Rücksprung zum Hauptprogramm ↓ Der RET-Befehl bewirkt, daß der Programmzähler mit der Rücksprungadresse (0 4 2 3) geladen wird
0 0 0 8		Zieladresse für den Befehl RST 1	

Tab. 6.3.12.1
Ablauf eines Interrupts anhand eines Programmbeispieles

Bei diesem Beispiel trifft während des Befehles ANA A von einem Peripheriegerät *x* eine Interruptanforderung ein. Dieser Befehl wird auf jeden Fall noch ausgeführt, und PC springt auf die nächste Befehlsadresse. Unter der Voraussetzung, daß INTE = 1 ist, wird der Interrupt akzeptiert. Das Gerät *x* sendet einen RST-Befehl (hier RST 0), der einmal den Inhalt von PC im Stack abspeichert und zum anderen einen Sprung zur Programmadresse 0 0 0 0 auslöst. Wie nachfolgend noch näher erläutert wird, gibt es insgesamt 8 verschiedene RST-Befehle, die jeweils 8 verschiedene Zieladressen ansprechen. Der Befehl RST 0 hat die Zieladresse 0_{10}, der Befehl RST 1 die Adresse 8_{10}, RST 2 Adresse 16_{10} usw. RST 7 die Adresse 56_{10}. Bis auf den Befehl RST 7 stehen somit pro RST-Befehl 8 Byte für ein Programm zur Verfügung. Reicht diese Speicherkapazität für eine Interruptbehandlung nicht aus, können über JMP- oder CALL-Befehle auch andere Speicherbereiche angewählt werden.
Nachdem das entsprechende Interruptprogramm abgeschlossen ist, erfolgt mit einem RET-Befehl der Rücksprung zum Hauptprogramm.
Nachdem ein peripheres Gerät eine Interruptanforderung gestellt hat und diese akzeptiert wurde, wird der Rechner selbst für die Abarbeitung des entsprechenden Interruptprogramms benutzt. Da sich hierbei die aus dem Hauptprogramm stammenden Registerinhalte und

Flag-Zustände ändern können, müssen diese durch entsprechende Befehle im Stack abgespeichert werden. Diese Befehle werden im nächsten Abschnitt eingehend behandelt. Ebenso müssen zum Schluß eines Interruptprogramms die im Stack abgespeicherten Daten wieder in die CPU gebracht werden, damit der Rechner mit dem abschließenden RET-Befehl das Hauptprogramm fortsetzen kann. Damit hat ein Interruptprogramm normalerweise die in Tab. 6.3.12.2 gezeigte Form.

Adresse	Inhalt	Befehl	Kommentar
x x x x	F 5	PUSH PSW	Flags, Akku- und Registerinhalt im Stack abspeichern
x x x x	C 5	PUSH B	
x x x x	D 5	PUSH D	
x x x x	E 5	PUSH H	
.			
.			
.		eigentliches Interruptprogramm	
.			
.			
x x x x	E 1	POP H	Ausgangszustand auch von Flags Akku- und Registerinhalt wieder herstellen
x x x x	D 1	POP D	
x x x x	C 1	POP B	
x x x x	F 1	POP PSW	
x x x x	C 9	RET	Rücksprung zum Hauptprogramm

Tab. 6.3.12.2
Interruptprogramm

Wie bereits erwähnt, wird bei akzeptierten Interrupts das INTE-Flipflop auf 0 zurückgesetzt. Damit ist zunächst kein weiterer Interrupt möglich. Ein neuer Interrupt kann erst wieder nach einem EI-Befehl zugelassen werden. Wenn dieser Befehl zu Beginn des eigentlichen Interruptprogramms (im Beispiel nach dem PUSH H-Befehl) angeordnet ist, kann während des laufenden Interruptprogramms ein neuer Interrupt erfolgen. Wird dagegen dieser Befehl am Ende des Interruptprogramms (im Beispiel vor dem RET-Befehl) angeordnet, so kann das laufende Interruptprogramm nicht unterbrochen werden. Im nachfolgenden Hauptprogramm ist dann jedoch wieder ein Interrupt möglich.
Eine Besonderheit beim MP 8080 ist, daß bei einem Peripheriegerät, das nicht in der Lage ist, ein entsprechendes RST-Signal zu erzeugen, aber ein INT-Signal gesendet hat, das akzeptiert wurde, der Datenbus auf 1 1 1 1 1 1 1 1 geschaltet wird. Dieser Code entspricht dem RST 7-Befehl, der somit in der vorher beschriebenen Art die Adresse 56_{10} anwählt. Damit ist es möglich, ohne zusätzliche Hardware einfachen Interruptbetrieb zuzulassen.
Nun noch einige Worte zu den 8 RST-Befehlen. Es handelt sich um 1-Byte-Befehle mit dem Format:

1	1	a	a	a	1	1	1	RST

Mit a a a wird angegeben, welcher der RST-Befehle gemeint ist. So handelt es sich z.B bei a a a = 0 0 0 um den RST 0-Befehl und bei a a a = 1 1 1 um den RST 7-Befehl.
Folgende Adressen werden durch die einzelnen RST-Befehle angewählt

RST 0-Befehl: Adresse $0_{10} \triangleq 0\ 0\ 0\ 0_{16}$
RST 1-Befehl: Adresse $8_{10} \triangleq 0\ 0\ 0\ 8_{16}$
RST 2-Befehl: Adresse $16_{10} \triangleq 0\ 0\ 1\ 0_{16}$
RST 3-Befehl: Adresse $24_{10} \triangleq 0\ 0\ 1\ 8_{16}$
RST 4-Befehl: Adresse $32_{10} \triangleq 0\ 0\ 2\ 0_{16}$
RST 5-Befehl: Adresse $40_{10} \triangleq 0\ 0\ 2\ 8_{16}$
RST 6-Befehl: Adresse $48_{10} \triangleq 0\ 0\ 3\ 0_{16}$
RST 7-Befehl: Adresse $56_{10} \triangleq 0\ 0\ 3\ 8_{16}$

Damit ist es möglich, 8 verschiedene Interrupts ohne zusätzliche Hardware durchzuführen. Bei den Befehlen RST 0 bis RST 6 stehen jeweils nur 8 Byte für ein Interruptprogramm **unmittelbar** zur Verfügung. Der RST 7-Befehl ermöglicht aber darüber hinaus auch längere Interruptprogramme. Wesentlich ist noch, daß alle RST-Befehle den Inhalt des Programmzählers PC im Stack abspeichern und somit in Verbindung mit dem RET-Befehl nach einer Programmunterbrechung die Rücksprungadresse automatisch wieder in den Programmzähler geladen wird.
Beim ITT MP-Experimenter sind für das Betriebsprogramm die meisten RST-Befehle bereits benutzt. Für Anwenderprogramme stehen noch die Befehle RST 1 und RST 7 zur Verfügung. Ein RST 1-Befehl bewirkt einen Sprung zur ROM-Adresse 0 0 0 8. Hier ist ein unbedingter Sprungbefehl (JMP 0 4 0 8) zur relativen RAM-Adresse 0 8 gespeichert. Der RST 7-Befehl bewirkt einen Sprung zur relativen RAM-Adresse 1 0.

Exp. 27

6.3.13 Stack-Befehle

Der Stack, der in erster Linie für die Abspeicherung von Unterprogramm-Rücksprungadressen gedacht ist, kann auch als Zwischenspeicher für Daten benutzt werden. Ermöglicht wird dies durch je 4 sog. PUSH- und POP-Befehle, mit denen die Daten abgespeichert bzw. zurückgeholt werden können. Die 4 PUSH-Befehle, die Daten im Stack abspeichern, haben das Format:

1	1	r	r	0	1	0	1

PUSH

Mit einem entsprechenden Code für r r können folgende Registerpaare zwischengespeichert werden:

Code r r	Registerpaar	Bezeichnung	Hex.-Code
0 0	BC	PUSH B	C 5
0 1	DE	PUSH D	D 5
1 0	HL	PUSH H	E 5
1 1	Akku + Flags	PUSH PSW	F 5

Eine Besonderheit stellt der Befehl PUSH PSW dar, bei dem der Inhalt des Akkus sowie der Zustand der Flags abgespeichert werden (PSW = Program Status Word).
An welcher Stelle im Stack die Daten abgelegt werden, bestimmt der Stack-Pointer SP. Zeigt z.B. der Stack-Pointer auf die Adresse 0 4 8 8, so bewirkt der Befehl PUSH B, daß der Inhalt des B-Registers in Adresse 0 4 8 7 und der Inhalt des C-Registers in Adresse 0 4 8 6 abgespeichert werden. Beim Befehl PUSH PSW würde der Inhalt des Akkus in Adresse 0 4 8 7 der Zustand der Flags in Adresse 0 4 8 6 zwischengespeichert. Allgemein läßt sich sagen, daß die Inhalte der erstgenannten Register (B, D, H und Akku) in der Adresse SP minus 1 und die Inhalte der zweitgenannten Register (C, E, L und Flags) in Adresse SP minus 2 gespeichert werden.
Das Zurückholen der abgespeicherten Daten kann mit den POP-Befehlen erfolgen. Sie haben das Format:

1	1	r	r	0	0	0	1

POP

Der Code für r r entspricht dem der PUSH-Befehle.
Bedingt durch die Arbeitsweise des Stacks ist unbedingt zu berücksichtigen, daß abgespeicherte Daten in der entgegengesetzten Reihenfolge wieder aus dem Stack herausgelesen werden, wie sie hineingegeben werden. Beispiel:

```
PUSH D   }
PUSH H   }  Befehlsfolge für die
PUSH PSW }  Eingabe
.
.
```

.
.
.

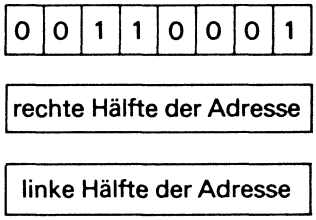
POP PSW
POP H
POP D

} Befehlsfolge für das Zurückholen

Dabei ist die Reihenfolge der PUSH-Befehle beliebig, während die Reihenfolge der POP-Befehle an die gewählte Eingabefolge angepaßt sein muß.

Exp. 28

Soll in einem Programm ein Stack verwendet werden, so muß über den Stack-Pointer ein freier Speicherbereich mit genügender Anzahl Bytes reserviert werden. Der Anfang des Stacks (höchste Stack-Adresse) kann mit dem Befehl LXI SP (Load SP Immediate) festgelegt werden. Dieser hat das Format:



LXI SP

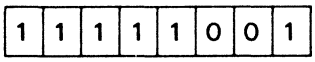
Soll beispielsweise der Stack-Bereich bei der Adresse 0 4 A 0 beginnen, so muß mit dem Befehl LXI SP der Stack-Pointer 0 4 A 1 gesetzt werden. Ein im Programm nachfolgender PUSH D-Befehl würde dann den Inhalt des D-Registers nach Adresse 0 4 A 0 und den Inhalt des E-Registers in Adresse 0 4 9 F bringen.

Nach einem PUSH-Befehl wird der Inhalt von SP um 2 erniedrigt, nach einem POP-Befehl um 2 erhöht.

Beim MP-Experimenter wird über das Monitorprogramm der Stack-Pointer grundsätzlich mit der Adresse 0 4 F E geladen. Damit beginnt der eigentliche Stack-Bereich (höchste Adresse) bei 0 4 F D.

Exp. 29

Der Stack-Pointer kann auch mit dem Befehl SPHL (Stack-Pointer from H and L) geladen werden. Dieser Befehl bringt den Inhalt des Registerpaares HL in den Stack-Pointer. Er hat das Format:

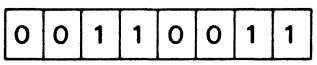


SPHL

Zur Gruppe der Stack-Befehle gehören auch noch die beiden Befehle:

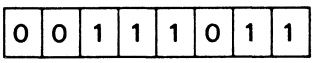
INX SP und
DCX SP

Der INX SP-Befehl incrementiert den Inhalt des Stack-Pointers um 1. Er hat das Format:



INX SP

Der DCX SP-Befehl decrementiert den Inhalt des Stack-Pointers. Sein Format ist:



DCX SP

Ein Befehl, mit dem der Inhalt des Stack-Pointers in das Registerpaar HL addiert werden kann, ist der DAD SP-Befehl (Double precision Add Stack-Pointer). Hierbei wird der Inhalt des Stack-Pointers zum Inhalt des HL-Registers addiert. Entsteht ein Überlauf, wird das Carry-Flag gesetzt. Die anderen Flags werden nicht beeinflußt.

Die 4 Befehle SPHL, INX SP, DCX SP und DAD SP werden in normalen Programmen nur selten benutzt, da sie sehr leicht zu Programmfehlern führen können. Ihre Anwendung bleibt auf Spezialprogramme begrenzt.

Ein weiterer Befehl, der mit dem Stack arbeitet und bei Argumentübergabe von Unterprogrammen sehr nützlich ist, ist der XTHL-Befehl (Exchange Top of Stack with H and L = tausche den „Stack-Kopf" gegen den Inhalt von H und L). Dieser Befehl tauscht die letzten 2 Bytes, die auf dem Stack geschrieben wurden, mit dem Inhalt des Registerpaares HL aus. Er hat das Format:

1	1	1	0	0	0	1	1	XTHL

Exp. 30

Der Inhalt des durch den Stack-Pointer adressierten Bytes wird mit dem Inhalt des L-Registers ausgetauscht, der Inhalt der Adresse SP + 1 mit dem Inhalt des H-Registers.

6.3.14 Weitere Befehle des MP 8080

Die folgenden 5 Befehle sind die letzten aus dem Befehlsvorrat des MP 8080.
Der Befehl NOP (No Operation = keine Operation) beeinflußt den Prozessor nicht. NOP-Befehle können z.B. eingesetzt werden, um Programmteile voneinander zu trennen, oder bestimmte Wartezeiten zwischen 2 Befehlen zu erzeugen, die dann durch die systemtypische Lesezeit für den NOP-Befehl bestimmt sind. Das Format dieses Befehles ist:

0	0	0	0	0	0	0	0	NOP

Der HLT-Befehl (Halt) stoppt den Prozessor. Das Erhöhen des Befehlszählers wird dabei noch ausgeführt. Der Prozessor kann diesen Zustand entweder durch RESET oder einen Interrupt verlassen. Der Wiederstart mit einem Interrupt kann durch einen DI-Befehl (Disable Interrupt) im Programm verhindert werden. Format des HLT-Befehles:

0	1	1	1	0	1	1	0	HLT

Der CMA-Befehl (Complement Accumulator) bildet das Einerkomplement des Akkumulators, d.h., er invertiert jedes einzelne bit des Akkumulatorinhaltes und schreibt das Ergebnis in den Akku zurück. Die Zustände der Flags werden nicht beeinflußt. Das Format des CMA-Befehles ist:

0	0	1	0	1	1	1	1	CMA

Die beiden letzten Befehle beeinflussen nur das Carry-Flag. Mit dem Befehl STC (Set Carry) wird das C-Flag auf 1 gesetzt und mit dem Befehl CMC (Complement Carry) invertiert. Formate der beiden Befehle:

0	0	1	1	0	1	1	1	STC

0	0	1	1	1	1	1	1	CMC

Exp. 31

Die restlichen Flags können nur durch geeignete Operationen oder über den Stack beeinflußt werden.