ELO
Sonderheft Nr. 76
Preis 8.– DM
8.– sfr, 62.– öS

ELO

Laborbrief Nr. 1

Erläuterungen und Beispiele zum 8085 Befehlssatz

Franzis-Verlag, München

1983

Franzis-Verlag GmbH, Karlstraße 37–41, 8000 München 2.

Bearbeitet von der Redaktion der Zeitschrift ELO. Für den Text verantwortlich: Reinhard Gößler

ISSN 0172-2786

Druck: Franzis-Druck GmbH, München. Printed in Germany, Imprimé en Allemagne.

ZV-Artikel-Nr. 76031 · F/ZV/383/750/2'

Reihenfolge der Befehls-vorstellung

Datentransportbefehle

1 Konstante laden
MVI r,xx; MVI m,xx; LXI rp,xxyy
2 Registerinhalt transportieren
MOV z,r; MOV r,m; MOV m,r
3 Akkumulator-Zugriff
LDA xxyy; STA xxyy; LDAX rp; MOV A,m; STAX rp; MOV m,A
4 Operationen mit Registerpaaren
PUSH rp; POP rp; XCHG
5 Operationen mit dem Registerpaar H&L
LHLD xxyy; SHLD xxyy; XCHG; XTHL; SPHL
6 Daten-Ein- und Ausgabe
IN xx; OUT xx; RIM; SIM

Arithmetisch/Logische Befehle

1 Zählbefehle
INR r; INR m; DCR r; DCR m; INX rp; DCX rp
2 Additionsbefehle
ADI xx; ACI xx; ADD r; ADC r; ADD m; ADC m; DAD rp; DAA
3 Subtraktionsbefehle
SUI xx; SBI xx; SUB r; SBB r; SUB m; SBB m
4 Befehle zur Bitmanipulation
ANI xx; ANA r; ANA m;
ORI xx; ORA r; ORA m;
XRI xx; XRA r; XRA m;
CMA; STC; CMC
5 Vergleichsbefehle
CPI xx; CMP r; CMP m
6 Schiebebefehle
RLC; RAL; RRC; RAR

Beeinflussung des Programmablaufs

1 Sprungbefehle
JMP xxyy; PCHL; NOP
JZ xxyy; JC xxyy; JPE xxyy; JM xxyy
JNZ xxyy; JNC xxyy; JPO xxyy; JP xxyy
2 Unterprogramm-Aufruf
CALL xxyy
CZ xxyy; CC xxyy; CPE xxyy; CM xxyy
CNZ xxyy; CNC xxyy; CPO xxyy; CP xxyy
RST n
3 Unterprogramm-Rücksprung
RET
RZ; RC; RPE; RM
RNZ; RNC; RPO; RP
4 Programmunterbrechungen
RIM; SIM; EI; DI; HLT

Der Befehlssatz des 8085

am Beispiel erläutert

Die ELO hat in den Heften 3 und 4/1983 den Befehlssatz des Mikroprozessors 8085 auf zwei großformatigen Postern vorgestellt. Im vorliegenden ELO-Labor-Brief sind als Ergänzung dazu Programm- und Anwendungsbeispiele zusammengefaßt, die die einzelnen Befehle am praktischen Beispiel erläutern.

Die Befehle tragen die Hersteller-übliche Assemblerbezeichnung, bei der die Variablen (z.B. "m" für eine Speicherzelle) stets klein geschrieben sind. 16-Bit-Werte, in der Regel Speicheradressen, werden durch "xxyy" dargestellt; es ist dabei zu beachten, daß bei den zugehörigen Dreiwort-Befehlen in Maschinensprache die untere Adreßhälfte "yy" als zweites und die obere Adreßhälfte "xx" als drittes Wort hinter dem Befehlscode folgen.

Wie in der Literatur üblich, wird der Inhalt einer Speicherzelle (bzw. eines Registers) dadurch gekennzeichnet, daß die Adresse (bzw. der Register-Name) in Klammern gesetzt wird: folglich ist unter "2352" die hexadezimale Adresse einer Speicherzelle und unter "(2352)" deren Inhalt zu verstehen.

Im 8085-Befehlssatz existieren Ein-, Zwei- und Dreiwort-Befehle, die im Speicher entsprechend ein, zwei oder drei aufeinander folgende Bytes belegen. Mehrwort-Befehle werden nacheinander ausgelesen, aber als Einheit ausgeführt. Dies hat nichts mit den Doppelwort-Operationen zu tun, bei denen die Operandenlänge 16 Bit beträgt (im Gegensatz zum Standard-8-Bit-Format).

Bedingt durch den Aufbau einiger Befehle kommt es zu Überschneidungen (z.B. MOV B,B = Datentransport aus dem B-Register ins B-Register), die programmtechnisch bedeutungslos sind. Andere Befehle dagegen können dazu eingesetzt werden, vor einer Abfrage die Zustandsbits zu setzen; ein Beispiel dafür ist der Einwort-Befehl "ORA A", der am Inhalt des Akkumulators nichts ändert, jedoch dazu führt, daß die Zustandssignale im FLAG-Register entsprechend dem Akkumulator-Inhalt gesetzt werden.

Im Gegensatz zu einigen anderen Prozessor-Typen werden beim 8
die Zustandsbits nur nach einer arithmetisch/logischen Operat
verändert, wobei einige Ausnahmen zu beachten sind. Führt e
derartige Operation zum Ergebnis Null, wird das Zustandsbit "
ro" auf HIGH gesetzt; ist im Ergebnis das höchstwertige Bit
HIGH, geht auch das Vorzeichen-Bit "Sign" auf HIGH; hat das
gebniswort eine gerade Anzahl von HIGH-Zuständen, ist dies
HIGH-Zustand des Paritätsbits "Parity" erkennbar (es geht a
bei Null HIGH-Zuständen im Ergebniswort auf HIGH); tritt bei
ner Addition oder Subtraktion ein Überlauf auf, zeigt dies
HIGH-Zustad des"Carry"-Bits an. Diese vier Zustandsbits las
sich mit einer Reihe von Befehlen "abfragen", bei denen eine
dingung enthalten ist (z.B. Befehlsausführung nur bei gesetz
Zero-Bit). Ein fünftes Zustandssignal "Auxiliary Carry" geht
HIGH, wenn im Ergebniswort ein Übertrag von der unteren zur ob
ren 4-Bit-Hälfte aufgetreten ist (z.B. bei hexadezimal OF a
10); dieser Hilfsübertrag läßt sich programmtechnisch nicht a
fragen.

Obwohl im FLAG-Register nur die eben beschriebenen fünf Bits b
legt sind, hat es die systemübliche Wortlänge von acht Bit; d
Verhalten der restlichen drei Bits ist herstellerseitig nic
spezifiziert, ebensowenig wie freie Befehlscodes ("unbekann
8085-Instruktionen"), die zwar bei einer Reihe verschiedener Pr
zessor-Typen zu definierten Reaktionen führen, von den Herste
lern offiziell aber nicht publiziert worden sind.

Das Carry-Bit C (oder auch CY) befindet sich an der niedrigstwe
tigen Position im FLAG-Register (Least Significant Bit, LSB
Bit 0, ganz rechts); im Bit 2 ist das Paritäts-Bit P und
Bit 6 das Null-Bit Z angeordnet. An der höchstwertigen Stel
des FLAG-Registers (Most Significant Bit, MSB; Bit 7, gan
links) steht das Vorzeichen-Bit S. Das Hilfsübertrags-Bit AC be
findet sich in Bitposition 4 des FLAG-Registers.

Der Befehlssatz des 8085 erläutert am Beispiel

1 Datentransportbefehle
1.1 Konstante laden

Diese Befehle dienen dazu, ein Register, ein Registerpaar od eine Speicherzelle mit einem konstanten Wert (Datenwort) zu l den; das Datenwort ist Bestandteil des Befehls. Der Befe "LXI SP,xxyy" legt den Beginn des Stapelspeichers (Stack Top) RAM fest.

MVI r,xx — **move immediate to registe**

Das Register r mit dem Datenwort xx laden.

	r	A	B	C	D	E	H	L
MVI r,xx:		3E xx	06 xx	0E xx	16 xx	1E xx	26 xx	2E xx

MVI m,xx — **move immediate to memory**

Eine Speicherzelle mit dem Datenwort xx laden; die Adresse de Speicherzelle steht im Registerpaar H&L.

MVI m,xx:	36 xx

LXI rp,xxyy — **load register pair immediate**

Das Registerpaar rp mit dem 16-Bit-Wert xxyy laden (Doppelwort-Operation). Das im zweiten Byte stehende Wort "yy" gelangt dabei in das "untere" Register (C, E, L bzw. untere Hälfte des Stack-Pointers), und das im dritten Byte stehende Wort "xx" ins "obere" Register (B, D, H bzw. obere Hälfte des Stack-Pointers).

	rp	B	D	H	SP
LXI rp,xxyy:		01 yy xx	11 yy xx	21 yy xx	31 yy xx

Anm.: rp B = B&C; rp D = D&E; rp H = H&L; SP = Stack-pointer

Keiner dieser Befehle beeinflußt eins der Zustandsbits im FLAG-Register.

Beispiel 1.1.1:

Ab Adresse 2FE0 sollen (dezimal) 24 fortlaufende RAM-Zellen gelöscht (mit "00" aufgefüllt) werden (vgl. Beispiel 3.1.4).

```
CLEAR   MVI  C,18      Schleifenzähler = 24 dez.
        LXI  H,2FE0    Anfangsadresse nach H&L
LOP111  MVI  m,00      00 in RAM-Zelle
        INX  H         Adreßregister erhöhen
        DCR  C         Schleifenzähler erniedrigen
        JNZ  LOP111    nicht Null:weiter bei LOP111
        xx   ...       sonst hier weiter
```

Beispiel 1.1.2:

Der Befehl **LXI SP,xxyy** definiert den Beginn des Stapelspeichers im RAM. Dieser Speicherbereich baut sich von der angegebenen Adresse aus nach unten auf (d.h. zu niedrigeren Adressen hin), und er wird von einem eigenen Adreßregister, dem Stack-Pointer, verwaltet.

Im Stack werden bei Unterprogramm-Aufrufen die Rücksprungadressen abgelegt und nach Durchlaufen des Unterprogramms von dort zurückgeholt. Es gibt aber auch die Möglichkeit, dort Daten zwischenzuspeichern und wieder abzurufen (vgl. Beispiele 1.4.1 und 1.4.2).

Bei jeder Stack-Aktivierung werden immer zwei aufeinanderfolgende RAM-Zellen belegt (mit einer 16-Bit-Adresse oder einem 16-Bit-Datenwort). Der Inhalt des Stack-Pointers enthält immer diejenige Stack-Adresse, in die zuletzt eingeschrieben wurde. Vor jedem neuen Einschreiben wird der Stack-Pointer um Eins erniedrigt und nach jedem Auslesen um Eins erhöht. Da Einschreiben und Auslesen immer mit zwei Bytes erfolgen, verändert jede Stack-Aktivierung den Stack-Pointer um Zwei. Diese Verwaltung erfolgt vollautomatisch, ohne daß sich der Anwender darum kümmern muß. Es ist allerdings darauf zu achten, daß sich der Stack bei ineinander verschachtelten Unterprogrammen schnell so weit ausdehnen kann, daß er andere Daten oder Programmteile überschreibt.

1.2 Registerinhalt transportieren

Mit diesen Befehlen wird der Inhalt eines Registers oder ei
Speicherzelle in ein anderes Register oder eine Speicherze
transportiert, ohne daß sich dabei am Inhalt der Ursprungsda
etwas ändert. Durch den Befehlsaufbau existieren Überschneid
gen (z.B. MOV B,B), die programmtechnisch keine Bedeutung habe

MOV z,r — move to register z from register

Ins Zielregister z den Inhalt des Ursprungregisters r transp
tieren.

	r	A	B	C	D	E	H	L
z								
MOV	A,r:	7F	78	79	7A	7B	7C	7D
MOV	B,r:	47	40	41	42	43	44	45
MOV	C,r:	4F	48	49	4A	4B	4C	4D
MOV	D,r:	57	50	51	52	53	54	55
MOV	E,r:	5F	58	59	5A	5B	5C	5D
MOV	H,r:	67	60	61	62	63	64	65
MOV	L,r:	6F	68	69	6A	6B	6C	6D

MOV r,m — move to register from memc

Den Inhalt einer Speicherzelle ins Register r transportieren;
die Adresse der Speicherzelle steht im Registerpaar H&L.

	r	A	B	C	D	E	H	L
MOV	r,m:	7E	46	4E	56	5E	66	6E

Anm.: Dieser Befehl ist auch unter 1.3 aufgeführt.

MOV m,r — move to memory from regis

Den Inhalt des Registers r in eine Speicherzelle transportier
die Adresse der Speicherzelle steht im Registerpaar H&L.

	r	A	B	C	D	E	H	L
MOV	m,r:	77	70	71	72	73	74	75

Anm.: Dieser Befehl ist auch unter 1.3 aufgeführt.

Keiner dieser Befehle beeinflußt eins der Zustandsbits im Fl
Register.

Beispiel 1.2:

Es soll der Adreßbereich 2901...2A00 um eine Adresse nach unten verschoben werden.

```
MOVE    LXI   H,2A00    oberste Adresse nach H&L
        MVI   A,00      Vorbereitung für Endabfrage
        MOV   B,m       erste RAM-Zelle nach B
LOP12   DCX   H         Adresse erniedrigen
        MOV   C,m       nächste RAM-Zelle nach C
        MOV   m,B       alten Wert an neuen Platz
        MOV   B,C       neuen Wert nach B
        CMP   L         ist A(=00) gleich L?
        JNZ   LOP12     nein: dann weiter bei LOP12
        xx    ...       sonst hier weiter
```

1.3 Akkumulator-Zugriff

Der Akkumulator (Register A) besitzt gegenüber den anderen Registern eine besondere Bedeutung, weil es eine Reihe von B len gibt, die unmittelbar auf den Akkumulator-Inhalt zugre Bei den beiden hier aufgeführten Befehlen steht die Ziel- Quell-Adresse für den Datentransport als zweites und dr Wort mit im Befehl.

LDA xxyy — **load accumulator di**

Den Inhalt der Speicherzelle xxyy in den Akkumulator tran tieren.

LDA xxyy: 3A yy xx

STA xxyy — **store accumulator direc**

Den Akkumulator-Inhalt in die Speicherzelle xxyy transportier

STA xxyy: 32 yy xx

Keiner dieser Befehle beeinflußt eins der Zustandsbits im F Register.

Beispiel 1.3.1:

Die Speicherzelle 2FCF dient in einem Anwendungsbeispiel als Indikator dafür, wenn in einem Programm von der einen Betriebsart (Mode 1) auf eine andere (Mode 2) umgeschaltet wird. Mode 1 ist für das Programm daran erkennbar, daß in der Speicherzelle 2FCF das Bit 0 auf HIGH ist, während im Mode 2 das Bit 1 auf HIGH gesetzt ist.

Zur Umschaltung von einer Betriebsart auf die andere holt das Programm den Inhalt der Speicherzelle 2FCF (LDA-Befehl), löscht das eine und setzt das andere Bit, und anschließend transportiert es den modifizierten Wert zurück nach 2FCF (STA-Befehl; vgl. Beispiele 2.2.1 und 2.3.1).

```
CHANGE   LDA  2FCF     Mode-Anzeiger holen
         ANI  FC       Bits 0 und 1 löschen
         ORI  01       Bit 0 (=Mode 1) setzen
         STA  2FCF     Mode-Anzeiger zurück ins RAM
         xx   ...      ...
```

noch 1.3: Vier Befehle ermöglichen die indirekte Adres beim Datentransport aus dem bzw. in den Akkumulator; es sich hierbei um Einwort-Befehle, bei denen die Ziel- bzw. adresse in einem der Registerpaare steht.

LDAX rp — load accumulator inc

Den Inhalt einer Speicherzelle in den Akkumulator transp ren; die Adresse der Speicherzelle steht im Registerpaar rp

	rp	B	D
LDAX	rp:	0A	1A

Anm.: rp B = B&C; rp D = D&E

MOV A,m — move from m

Den Inhalt einer Speicherzelle in den Akkumulator transp ren; die Adresse der Speicherzelle steht im Registerpaar H&L

MOV A,m: 7E

Anm.: Dieser Befehl ist auch unter 1.2 aufgeführt.

STAX rp — store accumulator indi

Den Akkumulator-Inhalt in eine Speicherzelle transportieren; Adresse der Speicherzelle steht im Registerpaar rp.

	rp	B	D
STAX	rp:	02	12

Anm.: rp B = B&C; rp D = D&E

MOV m,A — move to mem

Den Akkumulator-Inhalt in eine Speicherzelle transportieren; Adresse der Speicherzelle steht im Registerpaar H&L.

MOV m,A: 77

Anm.: Dieser Befehl ist auch unter 1.2 aufgeführt.

Keiner dieser Befehle beeinflußt eins der Zustandsbits im F Register.

Beispiel 1.3.2:

Um in einem Mikrocomputer eine Siebensegmentanzeige anzusteuern, müssen die binär vorliegenden Daten derart umcodiert werden, daß sie zur Ansteuerung der Leuchtbalken geeignet sind. Das zu codierende Datenwort steht im Beispiel hier in der Speicherzelle 2FCC; es wird mit Hilfe der indirekten Adressierung über B&C (LDAX-B-Befehl) in den Akkumulator geholt, wo zunächst die obere Hälfte unterdrückt (auf Null gesetzt) wird (ANI-Befehl).

Die Codes für die Zeichen 0...F stehen in den 16 Zellen der Tabelle SSGTB, zu deren Adressierung das Registerpaar D&E dient. Der in A stehende Wert 0...F gibt an, wie weit der zu holende Code vom Tabellenanfang entfernt steht; addiert man den Akkumulator-Inhalt zur Tabellen-Anfangsadresse (hier in D&E), dann zeigt Registerpaar D&E genau auf diejenige Speicherzelle, aus der der zum augenblicklichen Akkumulator-Inhalt passende Code geholt werden muß (indirekte Adressierung über D&E; LDAX-D-Befehl).

```
CODE    LXI   B,2FCC   Adresse für das Datenwort
        LXI   D,081F   Tabellen-Anfangsadresse
        LDAX  B        Datenwort in den Akku holen
        ANI   0F       obere 4 Bits löschen
        ADD   E        (A) und (E) addieren
        MOV   E,A      Ergebnis zurück nach E
                       (D&E=aktuelle Tabellenadr.)
        LDAX  D        Code aus Tabelle holen
        OUT   10       SSG-Code ausgeben
        xx             ...

SSGTAB  3F             SSG-Code für "0"
        06             SSS-Code für "1"
        xx             ...
                       weiter bis
        xx             ...
        71             SSG-Code für "F"
```

1.4 Operationen mit Registerpaaren

Die PUSH-Befehle dienen dazu, jeweils zwei Register-Inhalt
Registerpaar) ins RAM zu überschreiben (zu "retten"); mi
POP-Befehlen werden die Register-Inhalte wieder zurückgeho
handelt sich hierbei um Einwort-Befehle, bei denen die
bzw. Quelladresse für den Datentransport im Stack-Pointer s

PUSH rp — push register pair to

Den Inhalt des Registerpaars rp in den Stack (RAM) transpo
ren (Doppelwort-Operation).

	rp	B	D	H	PSW
PUSH rp:		C5	D5	E5	F5

Anm.: rp B = B&C; rp D = D&E; rp H = H&L; PSW (Prozessor-Sta
Wort) faßt den Akkumulator und das Register F (Status-Bits)
sammen.

POP rp — pop register pair from s

Den Inhalt der beiden durch den Stack-Pointer adressierten
Zellen in das Registerpaar rp transportieren (Doppelwort-O
tion).

	rp	B	D	H	PSW
POP rp:		C1	D1	E1	F1

Anm.: rp B = B&C; rp D = D&E; rp H = H&L; PSW (Prozessor-Sta
Wort) faßt den Akkumulator und das Register F (Status-Bits
sammen; bei POP PSW wird das Register F mit den im Stack ge
cherten Daten geladen.

XCHG — exchange H&L with

Die Inhalte der Registerpaare H&L und D&E miteinander ver
schen (Doppelwort-Operation).

XCHG: EB

Anm.: Dieser Befehl ist auch unter 1.5 aufgeführt.

Die Zustandsbits werden nur durch den Befehl POP PSW beeinfl
sie nehmen dabei die Werte an, die sie bei Ausführung des
fehls PUSH PSW hatten (Retten des Prozessor-Status-Wortes).

Beispiel 1.4.1:

Beim Sprung in ein Unterprogramm könnten dort Registerinhalte verändert werden, die anschließend in der alten Form benötigt werden. Um dies zu verhindern, überschreibt man vor dem Sprung ins Unterprogramm sämtliche Registerinhalte (paarweise) in den Stack (PUSH-Befehl) und holt sie nach der Rückkehr ins Hauptprogramm von dort wieder zurück (POP-Befehl).

Da die zuletzt in den Stack eingeschriebene Information als erste wieder ausgelesen wird, müssen die korrespondierenden PUSH- und POP-Befehle genauso angeordnet sein wie ineinander verschachtelte Klammern in der Algebra (vgl. Beispiel 1.1.2).

```
SAVE    PUSH B        (B&C) in den Stack
        PUSH D        (D&E) in den Stack
        PUSH H        (H&L) in den Stack
        PUSH PSW      (A) und FLAGs in den Stack
        CALL SUBRUT   Unterprogramm aufrufen
        POP  PSW      (A) und FLAGs zurückholen
        POP  H        (H&L) zurückholen
        POP  D        (D&E) zurückholen
        POP  B        (B&C) zurückholen
        XX   ...      ...
```

Beispiel 1.4.2:

Man kann durch Kombination verschiedener PUSH- und POP-Befehle den Inhalt von Registerpaaren vertauschen; so existiert beispielsweise kein Befehl, um auf den Inhalt des FLAG-Registers mit den Zustandsbits zuzugreifen. Auf dem Umweg über die folgende Sequenz ist dies dennoch möglich (vgl. Beispiel 1.1.2):

```
READF   PUSH PSW      (A) und FLAGs in den Stack
        POP  H        (A) nach H und FLAGs nach L
```

1.5 Operationen mit dem Registerpaar H&L

Mit den beiden hier aufgeführten Befehlen kann man den Inhalt des Registerpaars H&L in zwei aufeinanderfolgende RAM-Zellen transportiert bzw. von dort holen; die untere der beiden RAM-Adressen steht explizit im zweiten und dritten Byte dieser Dreiwort-Befehle.

LHLD xxyy — **load H&L direct**

Den Inhalt der Speicherzelle xxyy nach L und den Inhalt der Speicherzelle xxyy+1 nach H transportieren (Doppelwort-Operation).

LHLD xxyy: 2A yy xx

SHLD xxyy — **store H&L direct**

Den Inhalt von L in die Speicherzelle xxyy und den Inhalt von H in die Speicherzelle xxyy+1 transportieren (Doppelwort-Operation).

SHLD xxyy: 22 yy xx

Keiner dieser Befehle beeinflußt eins der Zustandsbits im FLAG-Register.

Beispiel 1.5.1:

Zahlreiche Operationen mit indirekter Adressierung laufen über das Registerpaar H&L ab. Um dieses Registerpaar nicht ständig zu blockieren, kann man seinen Inhalt (direkt adressiert) im RAM ablegen bzw. von dort zurückholen.

Im Beispiel hier steht in den Speicherzellen 2FE0 und 2FE1 eine 16-Bit-Adresse; sie wird deshalb nicht als Absolutadresse angegeben, weil sie von anderen Programmteilen modifiziert werden kann. Diese Adresse wird ins Registerpaar H&L geholt (LHLD-Befehl), wo sie beim anschließenden indirekten Datentransport (MOV-Befehl) als Zieladresse dient.

Wenn in den Speicherzellen 2FE0,1 beispielsweise der Inhalt 27B4 steht, dann bewirkt die folgende Sequenz einen Datentransport des Akkumulator-Inhalts in die Speicherzelle 27B4:

```
STORE   LHLD 2FE0     (2FE0,1) nach H&L holen;
                      (H&L) = 27B4
        MOV  m,A      (A) nach 27B4
        SHLD 2FE0     (H&) zurück nach 2FE0,1
        xx   ...      ...
```

noch 1.5: Drei Einwort-Befehle ermöglichen es, den Inhalt des Registerpaares H&L mit dem Inhalt eines anderen Registerpaars bzw. mit dem Inhalt zweier RAM-Zellen zu vertauschen bzw. dorthin zu transportieren.

XCHG — exchange H&L with D&E

Die Inhalte der Registerpaare H&L und D&E miteinander vertauschen (Doppelwort-Operation).

XCHG: EB

Anm.: Dieser Befehl ist auch unter 1.4 aufgeführt.

XTHL — exchange H&L with stack top

Den Inhalt des Registerpaars H&L mit dem Inhalt derjenigen beiden RAM-Zellen vertauschen, die durch den Stack-Pointer adressiert werden (Doppelwort-Operation).

XTHL: E3

SPHL — move H&L to SP

Den Inhalt des Registerpaars H&L in den Stack-Pointer transportieren (Doppelwort-Operation).

SPHL: F9

Keiner dieser Befehle beeinflußt eins der Zustandsbits im FLAG-Register.

Beispiel 1.5.2:

Beim RETURN-Befehl wird die Programmausführung bei derjenigen Adresse fortgesetzt, die im Stack abgelegt ist, und auf die der Stack-Pointer zum Zeitpunkt der Befehlsausführung zeigt. Um bei einer anderen Adresse fortzufahren (etwa resultierend aus einer vorangegangenen Operation), kann man diese Zieladresse beispielsweise ins Registerpaar H&L transportieren (LXI-Befehl) und mit dem XTHL-Befehl an diejenige Stelle im Stack übertragen, an der zuvor die ursprüngliche Rücksprungadresse gestanden hat. Bei Erreichen des RETURN-Befehls erfolgt dann die Verzweigung zu der neu eingegebenen Adresse (hier 2A61).

```
BRANCH   LXI  H,2A61    neue Zieladresse nach H&L
         XTHL           (H&L) mit (Stack) tauschen
         xx             ...
         xx
         xx             weitere Verarbeitung
         xx
         xx             ...
         RET            Verzweigung nach 2A61
```

Beispiel 1.5.3:

Wegen der sehr effektiven Zugriffsmöglichkeiten zu den im Stack gespeicherten Daten (PUSH- und POP-Befehle) kann man außer dem System-Stack ohne weiteres auch einen weiteren Stack-Bereich definieren, etwa um sämtliche Register mit dort abgelegten Datensätzen zu laden. Angenommen, im Registerpaar D&E steht eine Variable (kleiner als 1FFF), die mit Acht zu multiplizieren ist, um auf eine Tabelle im RAM zu zeigen; dann lassen sich mit der folgenden kurzen Sequenz alle Register mit den in der Tabelle enthaltenen Daten laden:

```
LOAD     XCHG           (D&E) mit (H&) tauschen
         DAD  H         (H&L) verdoppeln  (*2)
         DAD  H         (H&L) verdoppeln  (*4)
         DAD  H         (H&L) verdoppeln  (*8)
         SPHL           (H&L) in den Stack-Pointer
         POP  B         RAM-Daten nach B&C
         POP  D         RAM-Daten nach D&E
         POP  H         RAM-Daten nach H&L
         POP  PSW       RAM-Daten nach A und F
         xx   ...       ...
```

1.6 Daten-Ein- und -Ausgabe

Die beiden Befehle IN und OUT sind die einzigen, bei deren Ausführung die CPU-Leitung IO/M auf HIGH geht; sie signalisiert damit der Adreßdecodierung, daß auf dem Adreßbus keine Speicher-, sondern eine Portadresse ausgegeben wird. Die Portadresse ist acht Bit breit und erscheint dupliziert auf dem 16-Bit-Adreßbus.

IO/M Unterscheidung zwischen Speicher und Portadresse

IN xx — **input from port**

Die am E/A-Kanal xx anliegende Information in den Akkumulator transportieren.

IN xx: DB xx

Anm.: xx ist die Adresse des E/A-Kanals; während der Befehlsausführung geht die CPU-Leitung IO/M auf HIGH.

OUT xx — **output to port**

Den Akkumulator-Inhalt zum E/A-Kanal xx transportieren.

OUT xx: D3 xx

Anm.: xx ist die Adresse des E/A-Kanals; während der Befehlsausführung geht die CPU-Leitung IO/M auf HIGH.

Keiner dieser Befehle beeinflußt eins der Zustandsbits im FLAG-Register.

Beispiel 1.6.1:

Periphere Stellen (z.B. Tastatur oder Anzeige) lassen sich in einem Mikrocomputer ohne weiteres genauso ansprechen (adressieren) wie Speicherzellen (Memory Mapped I/O), einschließlich der damit verbundenen Vorteile wie z.B. die indirekte Adressierung. Der Nachteil dieses Verfahrens besteht darin, daß dadurch Teile des verfügbaren Adreßraums für die Peripherie verlorengehen. Darum gibt es den IN- und den OUT-Befehl, die beide eine 8 Bit breite Adresse enthalten und damit (zusätzlich zum 64-K-Adreßraum) weitere 256 Ziele ansprechen können (I/O Mapped I/O).

Bei der folgenden Sequenz werden in einer Endlosschleife die Daten vom Eingabekanal "EC" eingelesen und am Ausgabekanal "00" wieder ausgegeben:

```
INOUT   IN   EC      Kanal EC nch A einlesen
        OUT  00      (A) über Kanal 00 ausgeben
        JMP  INOUT   zurück zum Anfang
```

noch 1.6: Die Befehle RIM und SIM haben eine Doppelfunktion; sie dienen einerseits zum Datenverkehr zwischen dem Akkumulator und den seriellen CPU-Anschlüssen SID bzw. SOD und andererseits lesen bzw. laden sie das Interrupt-Masken-Register. In diesem Register wird festgelegt, welcher der drei CPU-Unterbrechungs-Eingänge RST5.5, RST6.5 bzw. RST7.5 aktiviert ("maskiert") wird (vgl. Abschnitt 3.4).

RIM — read interrupt mask

Den Pegel des seriellen CPU-Eingangs SID ins MSB des Akkumulators transportieren und die drei zu den Unterbrechungseingängen RST5.5, RST6.5 sowie RST7.5 gehörenden Masken-Bits in die drei unteren Akkumulator-Bits transportieren.

RIM: 20

Anm.: Dieser Befehl ist auch unter 3.4 aufgeführt.

SIM — set interrupt mask

Den MSB-Pegel des Akkumulators zum seriellen CPU-Ausgang SOD transportieren; dazu muß das Bit 6 (zweithöchstes) auf HIGH sein. Außerdem: Die Möglichkeit, die drei Unterbrechungseingänge RST5.5, RST6.5 RST7.5 freizugeben (zu "maskieren"), indem das korrespondierende Bit 0...2 auf LOW gesetzt wird; zu diesem Setzvorgang muß das Bit 3 auf HIGH sein (vgl. Abschnitt 3.4).

SIM: 30

Anm.: Dieser Befehl ist auch unter 3.4 aufgeführt.

Keiner dieser Befehle beeinflußt eins der Zustandsbits im FLAG-Register.

Beispiel 1.6.2:

Um den Pegel am seriellen CPU-Eingang SID per Programm abzufragen, kann man ihn beispielsweise ins MSB des Akkumulators und von dort ins Carry-Bit transportieren, um dann einen bedingten Sprung anzufügen (vgl. Beispiels 2.6.1, 3.1.2 und 3.1.3):

```
SIDIN   RIM             SID-Pegel nach A (MSB)
        NOP             weiter beim nächsten Befehl
        RLC             (A), MSB ins C-Bit schieben
        JC    CTE162    SID=1: weiter bei CTE162
        xx    ...       sonst hier weiter
```

Beispiel 1.6.3:

Der Pegel des höchstwertigen Akkumulator-Bits erscheint nach dem SIM-Befehl am seriellen CPU-Ausgang SOD (wenn gleichzeitig das Akkumulator-Bit 6 auf HIGH ist). Um beispielsweise den von SID eingelesenen Pegel <invertiert> wieder an SOD auszugeben, bedient man sich folgender Sequenz (vgl. Beispiele 2.4.1 und 3.1.2):

```
SRINOT  RIM             SID-Pegel nach A (MSB)
       <CMA>            (A) bei Bedarf invertieren
        ANI   80        Bits 0...6 löschen
        ORI   40        Bit 6 setzen
        SIM             (A), MSB an SOD ausgeben
        JMP   SRINOT    zurück zum Anfang
```

2 Arithmetisch/Logische Befehle
2.1 Zählbefehle

Mit diesen Befehlen ist es möglich, den Inhalt eines Registers oder einer Speicherzelle um Eins zu erhöhen bzw. zu erniedrigen.

INR r — **increment register**

Den Inhalt des Registers r um Eins erhöhen.

	r	A	B	C	D	E	H	L
INR	r:	3C	04	0C	14	1C	24	2C

INR m — **increment memory**

Den Inhalt einer Speicherzelle um Eins erhöhen; die Adresse der Speicherzelle steht im Registerpaar H&L.

INR m: 34

DCR r — **decrement register**

Den Inhalt des Registers r um Eins erniedrigen.

	r	A	B	C	D	E	H	L
DCR	r:	3D	05	0D	15	1D	25	2D

DCR m — **decrement memory**

Den Inhalt einer Speicherzelle um Eins erniedrigen; die Adresse der Speicherzelle steht im Registerpaar H&L.

DCR m: 35

FLAG-Beeinflussung: S (Sign), Z (Zero), P (Parity) und AC (Aux. Carry); das Carry-Bit wird hiervon nicht beeinflußt.

Beispiel 2.1.1:

Ein in der Speicherzelle 2B9F stehender Wert soll so weit vergrößert werden, daß er gleich groß ist wie der Akkumulator-Inhalt; dabei ist die Randbedingung einzuhalten, daß das Erhöhen nach maximal zehn Schritten abzubrechen ist, um beispielsweise in einem Regelkreis Überschwinger zu vermeiden.

```
COUNT    LXI   H,2BF9    Adresse nach H&L
         MVI   C,0A      Schrittzähler auf dezimal 10
LOP211   CMP   m         (A) mit (2BF9) vergleichen
         JZ    CTE211    gleich: weiter bei CTE211
         INR   m         sonst (2BF9) erhöhen
         DCR   C         Schrittzähler erniedrigen
         JNZ   LOP211    nicht Null:weiter bei LOP211
CTE211   xx              sonst hier weiter
```

noch 2.1: Mit diesen beiden Befehlen läßt sich der Inhalt von Registerpaaren als 16-Bit-Zähler um Eins erhöhen bzw. erniedrigen.

INX rp — increment register pair

Den Inhalt des Registerpaars rp um Eins erhöhen (Doppelwort-Operation).

	rp	B	D	H	SP
INX	rp:	03	13	23	33

Anm.: rp B = B&C; rp D = D&E; rp H = H&L; SP = Stack-Pointer

DCX rp — decrement register pair

Den Inhalt des Registerpaars rp um Eins erniedrigen (Doppelwort-Operation).

	rp	B	D	H	SP
DCX	rp:	0B	1B	2B	3B

Anm.: rp B = B&C; rp D = D&E; rp H = H&L; SP = Stack-Pointer

Es wird keins der Zustandsbits im FLAG-Register beeinflußt, obwohl es sich bei diesen Befehlen um eine arithmetische Operation handelt!

Beispiel 2.1.2:

Bei einer Datenübertragung (Ausgabe über Port 00) stehen im Registerpaar H&L die Anfangs- und im Registerpaar D&E die Endadresse des zu übertragenden Speicherbereichs. Nach jeder Ausgabe eines Datenbytes wird der Inhalt von H&L um Eins hochgezählt und im Unterprogramm COMPAR mit dem Inhalt von D&E verglichen, um das Übertragungsende zu erkennen; beim Rücksprung aus COMPAR ist das Zero-Bit HIGH, wenn die Inhalte der Registerpaare D&E und H&L gleich sind (vgl. Beispiele 2.5.2 und 3.2.1).

```
SERIEL   LXI  H,2800   Anfangsadresse nach H&L
         LXI  D,2AFF   Endadresse nach D&E
LOP212   MOV  A,m      erstes Datenwort nach A
         OUT  00       (A) über Kanal 00 ausgeben
         INX  H        Adresse erhöhen
         CALL COMPAR   Endadresse erreicht?
         JNZ  LOP212   nein: weiter bei LOP212
         xx   ...      sonst hier weiter
```

2.2 Additionsbefehle

Zwei Befehle sind dazu vorgesehen, zum Akkumulator-Inhalt eine Konstante zu addieren; dies kann sowohl ohne als auch mit Einbeziehung des Carry-Bits erfolgen.

ADI xx — **add immediate**

Zum Akkumulator-Inhalt das Datenwort xx addieren.

ADI xx: C6 xx

ACI xx — **add immediate with carry**

Zum Akkumulator-Inhalt das Datenwort xx sowie das C-Bit addieren.

ACI xx: CE xx

FLAG-Beeinflussung: S (Sign), Z (Zero), P (Parity), C (Carry) und AC (Aux. Carry).

Beispiel 2.2.1:

Ein zuvor errechneter Wert, der in der Speicherzelle 2937 steht, ist um den konstanten Betrag von (dezimal) 18 zu erhöhen, bevor er weiterverarbeitet wird. Tritt bei der Addition jedoch ein Überlauf auf, so ist der Maximalwert "FF" weiterzugeben (vgl. Beispiel 2.2.3).

```
ADIER1   LDA   2937      (2937) nach A
         ADI   12        (A) um dezimal 18 erhöhen
         JNC   CTE221    kein Überlauf: zu CTE221
         MVI   A,FF      Maximalwert FF nach A
CTE221   STA   2937      neuen Wert nach 2937
         xx    ...       ...
```

noch 2.2: Zum Akkumulator-Inhalt wird der Inhalt eines Registers addiert, wahlweise ohne oder mit Einbeziehung des Carry-Bits.

ADD r — add register

Zum Akkumulator-Inhalt den Inhalt des Registers r addieren.

	r	A	B	C	D	E	H	L
ADD	r:	87	80	81	82	83	84	85

ADC r — add register with carry

Zum Akkumulator-Inhalt den Inhalt des Registers r sowie das C-Bit addieren.

	r	A	B	C	D	E	H	L
ADC	r:	8F	88	89	8A	8B	8C	8D

FLAG-Beeinflussung: S (Sign), Z (Zero), P (Parity), C (Carry) und AC (Aux. Carry).

Beispiel 2.2.2:

Es sollen die Inhalte der Registerpaare B&C und D&E addiert werden (Ergebnis in D&E); das Registerpaar H&L ist dabei nicht frei.

Die Addition erfolgt in zwei Teilen, beginnend mit den unteren Hälften der Summanden (Inhalte der Register C und E; ADD-Befehl). Da hierbei ein Übertrag auftreten kann (Carry-Bit HIGH), muß die Addition der oberen Hälften der Summanden den Zustand des C-Bits mit einbeziehen (ADC-Befehl). Nach diesem Schema lassen sich Additionen mit beliebig langer Wortlänge durchführen (vgl. Beispiel 2.3.2).

```
ADDRP   MOV   A,E      untere Hälfte des ersten
                       Summanden nach A
        ADD   C        (C) zu (A) addieren
        MOV   E,A      untere Ergebnishälfte nach E
        MOV   A,D      obere Hälfte des ersten
                       Summanden nach A
        ADC   B        (B) und Carry zu (A) addier.
        MOV   D,A      obere Ergebnishälfte nach D
        xx    ...      ...
```

noch 2.2: Zum Akkumulator-Inhalt wird der Inhalt einer Speicherzelle addiert, wahlweise ohne oder mit Einbeziehung des Carry-Bits; die Adressierung des Summanden erfolgt hierbei indirekt über das Registerpaar H&L.

==========

ADD m — **add memory**

==========

Zum Akkumulator-Inhalt den Inhalt einer Speicherzelle addieren; die Adresse der Speicherzelle steht im Registerpaar H&L.

ADD m: 86

==========

ADC m — **add memory with carry**

==========

Zum Akkumulator-Inhalt den Inhalt einer Speicherzelle sowie das C-Bit addieren; die Adresse der Speicherzelle steht im Registerpaar H&L.

ADC m: 8E

FLAG-Beeinflussung: S (Sign), Z (Zero), P (Parity), C (Carry) und AC (Aux. Carry).

Beispiel 2.2.3:

Ein zuvor errechneter Wert, der in der Speicherzelle 2937 steht, ist um den konstanten Betrag von (dezimal) 18 zu erhöhen, bevor er weiterverarbeitet wird (indirekte Adressierung über das Registerpaar H&L). Tritt bei der Addition jedoch ein Überlauf auf, so ist der Maximalwert "FF" weiterzugeben (vgl. Beispiel 2.2.1).

```
ADIER2   LXI   H,2937    Adresse nach H&L
         MVI   A,12      Summand (dezimal 18) nach A
         ADD   m         Speicherinhalt addieren
         JNC   CTE223    kein Übertrag: zu CTE223
         MVI   A,FF      Maximalwert FF nach A
CTE223   MOV   m,A       neuen Wert nach 2937
         xx              ...
```

noch 2.2: Der DAD-Befehl dient zur Doppelwort-Addition mit Registerpaaren; zum Inhalt des Registerpaars H&L wird der Inhalt eines weiteren Registerpaars addiert.

DAD rp — add register pair to H&L

Den Inhalt des Registerpaars rp zu H&L addieren (Doppelwort-Operation).

rp	B	D	H	SP
DAD rp:	09	19	29	39

Anm.: rp B = B&C; rp D = D&E; rp H = H&L; SP = Stack-pointer

Es wird hierdurch nur das Carry-Bit beeinflußt.

Beispiel 2.2.4:

Der im Register E stehende Wert soll mit Fünf multipliziert werden; ein eventuell auftretender Übertrag ist im Register D abzulegen (das Ergebnis steht folglich im Registerpaar D&E).

```
MULT    MVI   C,05      Multiplikator nach C
        LXI   H,0000    Ergebnisregister löschen
        MVI   D,00      obere Hälfte von D&E löschen
LOP224  DAD   D         (D&E) zu (H&L) addieren
        DCR   C         Schleifenzähler erniedrigen
        JNZ   LOP224    nicht Null:weiter bei LOP224
        XCHG            Ergebnis von H&L nach D&E
        xx              ...
```

Beispiel 2.2.5:

Es existiert zwar ein Befehl, um den Inhalt des Registerpaars H&L in den Stack-Pointer zu transportieren, die umgekehrte Instruktion aber (Inhalt des Stack-Pointers in ein Registerpaar transportieren) gibt es im 8085-befehlssatz nicht. Auf folgendem Umweg kommt man dennoch an den Inhalt des Stack-Pointers heran:

```
LOADSP  LXI   H,0000    H&L löschen
        DAD   SP        (SP) zu (H&L) addieren; in
                        H&L steht der Inhalt des SP
        xx    ...       ...
```

noch 2.2: Für die BCD-Arithmetik steht der DAA-Befehl zur Verfügung; er dient dazu, den Akkumulator-Inhalt in zwei BCD-Digits umzusetzen.

Dazu wird zum Akkumulator-Inhalt hexadezimal 06 addiert, wenn die unteren vier Bits größer als 9 (binär 1001) sind; sind die oberen vier Akkumulator-Bits größer als 9 (binär 1001), wird hexadezimal 60 addiert.

==========

DAA **decimal adjust accumulator**

==========

Dezimalkorrektur des Akkumulators nach einer arithmetischen Operation (Umsetzung in zwei BCD-Digits).

DAA: 27

Anm.: Die Addition von hexadezimal 06 <bzw. 60> erfolgt auch dann, wenn, resultierend aus einer vorhergehenden Befehlsausführung, das AC-Bit <bzw. das C-Bit> gesetzt ist.

FLAG-Beeinflussung: S (Sign), Z (Zero), P (Parity), C (Carry) und AC (Aux. Carry).

Beispiel 2.2.6:

Der Akkumulator-Inhalt soll nicht hexadezimal (z.B. 29, 2A, 2B...), sondern in Form von zwei BCD-Digits hochgezählt werden (also z.B. Übertrag von 29 auf 30).

```
ADJUST   MVI  A,29      (A) = 29
         INR  A         (A) = 2A (binäres Zählen)
         DAA            (A) = 30 (Dezimalkorrektur)
         xx   ...       ...
```

Achtung! Bei Ausführung des DAA-Befehls im Beispiel hier erfolgt eine Dezimalkorrektur (Übertrag von der unteren zur oberen Hälfte des Akkumulators), so daß dadurch das AC-Bit (Aux. Carry) gesetzt wird. Würde man hinter den ersten einen weiteren DAA-Befehl setzen (was programmtechnisch keinen Sinn ergäbe), würde der Akkumulator-Inhalt dadurch auf 36 erhöht werden, weil aufgrund einer vorangegangenen Operation (erster DAA-Befehl) das AC-Bit auf HIGH liegt und demzufolge 06 zum Akkumulator-Inhalt addiert wird (vgl. nebenstehende Anmerkung).

Im Zusammenhang mit dem DAA-Befehl ist daher darauf zu achten, daß das Carry- oder AC-Bit durch keinen anderen als den INR-A-Befehl modifiziert wird.

2.3 Subtraktionsbefehle

Zwei Befehle sind dazu vorgesehen, vom Akkumulator-Inhalt eine Konstante zu subtrahieren; dies kann sowohl ohne als auch mit Einbeziehung des Carry-Bits erfolgen. Bei einem (negativen) Subtraktions-Übertrag bezeichnet man das Carry-Bit als "Borrow".

SUI xx — **subtract immediate**

Vom Akkumulator-Inhalt das Datenwort xx subtrahieren.

SUI xx: D6 xx

SBI xx — **subtract immediate with borrow**

Vom Akkumulator-Inhalt das Datenwort xx sowie das C-Bit subtrahieren.

SBI xx: DE xx

FLAG-Beeinflussung: S (Sign), Z (Zero), P (Parity), C (Carry) und AC (Aux. Carry).

Beispiel 2.3.1:

Ein zuvor errechneter Wert, der in der Speicherzelle 2937 steht, ist um den konstanten Betrag von (dezimal) 18 zu verringern, bevor er weiterverarbeitet wird. Tritt bei der Subtraktion jedoch ein (negativer) Übertrag auf, so ist der Minimalwert "00" weiterzugeben (vgl. Beispiel 2.3.3).

```
SUBTR1   LDA   2937      (2937) nach A
         SUI   12        von (A) dezimal 18 abziehen
         JNC   CTE231    kein Übertrag: zu CTE231
                         (Ergebnis nicht negativ)
         MVI   A,00      Minimalwert 00 nach A
CTE231   STA   2937      neuer Wert nach 2937
         xx    ...       ...
```

noch 2.3: Vom Akkumulator-Inhalt wird der Inhalt eines Registers subtrahiert, wahlweise ohne oder mit Einbeziehung des Carry-Bits.

SUB r — subtract register

Vom Akkumulator-Inhalt den Inhalt des Registers r subtrahieren.

	r	A	B	C	D	E	H	L
SUB r:		97	90	91	92	93	94	95

SBB r — subtract register with borrow

Vom Akkumulator-Inhalt den Inhalt des Registers r sowie das C-Bit subtrahieren.

	r	A	B	C	D	E	H	L
SBB r:		9F	98	99	9A	9B	9C	9D

FLAG-Beeinflussung: S (Sign), Z (Zero), P (Parity), C (Carry) und AC (Aux. Carry).

Beispiel 2.3.2:

Es soll der Inhalt des Registerpaars B&C vom Inhalt des Registerpaars D&E subtrahiert werden (Ergebnis in D&E).

Die Subtarktion erfolgt in zwei Teilen, beginnend mit den unteren Hälften der Operanden (Inhalte der Register C und E; SUB-Befehl). Da hierbei ein (negativer) Übertrag auftreten kann (Carry-Bit HIGH), muß die Subtraktion der oberen Hälften der Operanden den Zustand des C-Bits mit einbeziehen (SBB-Befehl). Nach diesem Schema lassen sich Subtraktionen mit beliebig langer Wortlänge durchführen (vgl. Beispiel 2.2.2).

```
SUBRP   MOV   A,E     untere Hälfte des Minuenden
                      nach A
        SUB   C       (C) von (A) subtrahieren
        MOV   E,A     untere Ergebnishälfte nach E
        MOV   A,D     obere Hälfte des Minuenden
                      nach A
        SBB   B       (B) und Carry von (A) subtr.
        MOV   D,A     obere Ergebnishälfte nach D
        xx    ...     ...
```

noch 2.3: Vom Akkumulator-Inhalt wird der Inhalt einer Speicherzelle subtrahiert, wahlweise ohne oder mit Einbeziehung des Carry-Bits; die Adressierung des Subtrahenden erfolgt indirekt über das Registerpaar H&L.

==========

SUB m — **subtract memory**

==========

Vom Akkumulator-Inhalt den Inhalt einer Speicherzelle subtrahieren; die Adresse der Speicherzelle steht im Registerpaar H&L.

SUB m: 96

==========

SBB m — **subtract memory with borrow**

==========

Vom Akkumulator-Inhalt den Inhalt einer Speicherzelle sowie das C-Bit subtrahieren; die Adresse der Speicherzelle steht im Registerpaar H&L.

SBB M: 9E

FLAG-Beeinflussung: S (Sign), Z (Zero), P (Parity), C (Carry) und AC (Aux. Carry).

Beispiel 2.3.3:

Ein zuvor errechneter Wert, der in der Speicherzelle 2937 steht, ist um den konstanten Betrag von (dezimal) 18 zu verringern, bevor er weiterverarbeitet wird. Tritt bei der Subtraktion jedoch ein (negativer) Übertrag auf, so ist der Minimalwert "00" weiterzugeben (vgl. Beispiel 2.3.1).

```
SUBTR2   LXI   H,2937    Adresse nach H&L
         MOV   A,m       Minuend nach A
         MVI   m,12      Subtrahend in Speicherzelle
         SUB   m         RAM-Inhalt von A subtahieren
         JNC   CTE233    kein Übertrag: zu CTE233
                         (Ergebnis nicht negativ)
         MVI   A,00      Minimalwert 00 nach A
CTE233   MOV   m,A       neuer Wert nach 2937
         xx    ...
```

2.4 Befehle zur Bitmanipulation

Die zu den logischen Befehlen gehörenden Instruktionen dienen dazu, einzelne Bits oder Bit-Gruppen zu beeinflussen; dazu gehört das gezielte Setzen, Löschen oder Invertieren von Bits. Bei der UND-Verknüpfung zweier 8-Bit-Worte bleiben im Akkumulator nur diejenigen Bitpositionen auf HIGH, die in beiden Operanden HIGH waren.

ANI xx — AND immediate

Den Akkumulator-Inhalt mit dem Datenwort xx logisch UND-verknüpfen.

ANI xx:	E6 xx

ANA r — AND register

Den Akkumulator-Inhalt mit dem Inhalt des Registers r logisch UND-verknüpfen.

	r	A	B	C	D	E	H	L
ANA r:		A7	A0	A1	A2	A3	A4	A5

ANA m — AND memory

Den Akkumulator-Inhalt mit dem Inhalt einer Speicherzelle logisch UND-verknüpfen; die Adresse der Speicherzelle steht im Registerpaar H&L.

ANA m:	A6

FLAG-Beeinflussung: S (Sign), Z (Zero), P (Parity), C (Carry) und AC (Aux. Carry).

Beispiel 2.4.1:

Nach dem Transport des SID-Pegels ins MSB des Akkumulators (RIM-Befehl) soll dieser Zustand invertiert am seriellen CPU-Ausgang SOD ausgegeben werden (SIM-Befehl; vgl. Beispiel 2.4.4).

Damit die bei SIM möglichen anderen Auswirkungen unterbleiben (Beeinflussung des Interrupt-Masken-Registers), werden die Akkumulator-Bits 0...6 mit dem ANI-Befehl gelöscht; Bit 7 wird hierdurch nicht verändert, weil das korrespondierende Bit 7 im Operanden "80" auf HIGH liegt (vgl. Beispiel 1.6.3).

```
SRINOT   RIM          SID-Pegel nach A (MSB)
         CMA          (A) invertieren
         ANI  80      Bits 0...6 löschen
         ORI  40      Bit 6 setzen
         SIM          (A), MSB an SOD ausgeben
         xx   ...     ...
```

noch 2.4: Bei der ODER-Verknüpfung werden im Akkumulator diejenigen Bitpositionen auf HIGH gesetzt, die in einem der beiden Operanden HIGH waren.

ORI xx — OR immediate

Den Akkumulator-Inhalt mit dem Datenwort xx ODER-verknüpfen.

ORI xx: F6 xx

ORA r — OR register

Den Akkumulator-Inhalt mit dem Inhalt des Registers r logisch ODER-verknüpfen.

	r	A	B	C	D	E	H	L
ORA r:		B7	B0	B1	B2	B3	B4	B5

ORA m — OR memory

Den Akkumulator-Inhalt mit dem Inhalt einer Speicherzelle logisch ODER-verknüpfen; die Adresse der Speicherzelle steht im Registerpaar H&L.

ORA m: B6

FLAG-Beeinflussung: S (Sign), Z (Zero), P (Parity), C (Carry) und AC (Aux. Carry).

Beispiel 2.4.2:

Nach dem Transport des SID-Pegels ins MSB des Akkumulators (RIM-Befehl) soll dieser Zustand invertiert am seriellen CPU-Ausgang SOD ausgegeben werden (SIM-Befehl).

Damit die bei SIM möglichen anderen Auswirkungen unterbleiben (Beeinflussung des Interrupt-Masken-Registers), werden die Bits 0...5 sowie 7 mit dem ANI-Befehl gelöscht.

Um den SOD-Pegel bei Ausführen des SIM-Befehls beeinflussen zu können, muß das Bit 6 im Akkumulator auf HIGH sein; mit dem ORI-Befehl wird es gezielt gesetzt, weil Bit 6 im Operanden "40" auf High liegt. Die übrigen Bits 0...5 sowie 7 werden hierdurch nicht verändert (vgl. Beispiel 1.6.3).

```
SRINOT   RIM          SID-Pegel nach A (MSB)
         CMA          (A) invertieren
         ANI   80     Bits 0...6 löschen
         ORI   40     Bit 6 setzen
         SIM          (A), MSB an SOD ausgeben
         xx    ...    ...
```

noch 2.4: Die Exklusiv-ODER-Verknüpfung invertiert im Akkumulator diejenigen Bitpositionen, die im zweiten Operanden auf HIGH sind. Der Befehl XRA A (Akkumulator-Inhalt mit sich selbst Exklusiv-ODER-verknüpfen) führt zum Löschen sämtlicher Bits im Akkumulator.

XRI xx — XOR immediate

Den Akkumulator-Inhalt mit dem Datenwort xx Exklusiv-ODER-verknüpfen.

XRI xx:	EE xx

XRA r — XOR register

Den Akkumulator-Inhalt mit dem Inhalt des Registers r Exklusiv-ODER-verknüpfen.

	r	A	B	C	D	E	H	L
XRA r:		AF	A8	A9	AA	AB	AC	AD

XRA m — XOR memory

Den Akkumulator-Inhalt mit dem Inhalt einer Speicherzelle Exklusiv-ODER-verknüpfen; die Adresse der Speicherzelle steht im Registerpaar H&L.

XRA m:	AE

FLAG-Beeinflussung: S (Sign), Z (Zero), P (Parity), C (Carry) und AC (Aux. Carry).

Beispiel 2.4.3:

Bei einem Verfahren der Analog/Digital-Umsetzung (sukkzessive Approximation) wird der Akkumulator-Inhalt zunächst gelöscht (hier mit XRA A; zum Löschen läßt sich auch der Einwort-Befehl SUB A einsetzen). Dann wird der Akkumulator-Inhalt mit einem "Bit-Zeiger" (Datenwort mit nur einem HIGH-Bit, hier in Register B) ODER-verknüpft und an den Digital/Analog-Umsetzer ausgegeben (Ausgabekanal 27). Wenn der so gebildete Analogwert größer ist als der zu digitalisierende (Abfrage eines Komparators am Eingabekanal 28), muß das zuletzt gesetzte Bit wieder gelöscht werden (XRA-B-Befehl).

```
ADU       XRA  A       A löschen (Einwortbefehl)
          MVI  B,80     Bit-Zeiger laden (MSB HIGH)
LOP243    ORA  B       in A ein Bit setzen
          MOV  C,A     Wert nach C retten
          OUT  27      (A) an D/A-Umsetzer ausgeben
          IN   28      Komparator-Ausgang einlesen
          RLC          und ins C-Bit verschieben
          MOV  A,C     alten Wert zurückholen
                       (Carry wird nicht verändert)
          JC   CTE243  dig. Wert zu klein:zu CTE243
          XRA  B       gesetztes Bit wieder löschen
CTE243    MOV  A,B     Bit-Zeiger holen,
          RRC          rechts verschieben
          MOV  B,A     Bit-Zeiger zurück nach B
          MOV  A,C     aktuellen Wert zurückholen
          JNC  LOP243  noch nicht fertig: zu LOP243
          xx   ...     fertig: hier weiter
```

noch 2.4: Der Akkumulator-Inhalt läßt sich mit dem Einwort-Befehl CMA invertieren. Zum Setzen und Invertieren des Carry-Bits ist je ein eigener Befehl vorgesehen.

CMA — complement accumulator

Den Akkumulator-Inhalt invertieren.

CMA: 2F

STC — set carry

Das C-Bit auf HIGH setzen.

STC: 37

CMC — complement carry

Den Zustand des C-Bits invertieren.

CMC: 3F

Mit Ausnahme der direkten Carry-Beeinflussung erfolgt keine Veränderung der Zustandsbits.

Beispiel 2.4.4.:

Gelegentlich werden Pegelzustände durch Leuchtdioden dargestellt (HIGH-Pegel: leuchtende Diode, LOW-Pegel: dunkle Diode). Wenn die Treiberschaltungen nichtinvertierend sind, muß der ansteuernde Pegel zuvor invertiert werden; beim Akkumulator-Inhalt kann man dies mit dem Einwort-Befehl **CMA** bewerkstelligen (zum selben Ergebnis, allerdings mit einem Zweiwort-Befehl, führt die Exklusiv-ODER-Verknüpfung des Akkumulator-Inhalts mit "FF": XRI FF; vgl. Beispiel 2.4.1).

Beispiel 2.4.5:

Als Indikator für ein bestimmtes Ergebnis, das die Zustandsbits nicht beeinflußt hat, soll ein Unterprogramm an das übergeordnete Hauptprogramm das Carry-Bit definiert übergeben.

Zum Setzen dient der Befehl **STC**, und zum Löschen muß dem STC-Befehl die Anweisung **CMC** (Carry-Bit invertieren) nachgestellt werden.

2.5 Vergleichsbefehle

Die Vergleichsbefehle dienen dazu, zwei Operanden, von denen einer im Akkumulator steht, miteinander zu vergleichen und resultierend daraus die Zustandsbits zu beeinflussen, um das Ergebnis abfragen zu können. Die FLAG-Beeinflussung erfolgt dabei so, als ob der zum Vergleich herangezogene Operand vom Akkumulator-Inhalt subtrahiert worden wäre.

CPI xx — compare immediate

Den Akkumulator-Inhalt mit dem Datenwort xx vergleichen.

CPI xx:	FE xx

CMP r — compare register

Den Akkumulator-Inhalt mit dem Inhalt des Registers r vergleichen.

r	A	B	C	D	E	H	L
CMP r:	BF	B8	B9	BA	BB	BC	BD

CMP m — compare memory

Den Akkumulator-Inhalt mit dem Inhalt einer Speicherzelle vergleichen; die Adresse der Speicherzelle steht im Registerpaar H&L.

CMP m:	BE

FLAG-Beeinflussung: S (Sign), Z (Zero), P (Parity), C (Carry) und AC (Aux. Carry).

Beispiel 2.5.1:

Eine Tastatur liefert über den Eingabekanal 04 den Tastencode "42" (ASCII-Code für "B") an den Mikrocomputer; das zugehörige Programm muß jede Eingabe daraufhin überprüfen, ob ein "B" angekommen ist, weil dann zum Programmteil "CTEB" verzweigt werden soll.

Die Abfrage nach einem beliebigen Bitmuster (Datenwort) erfolgt über den CPI-Befehl, nach dessen Ausführung das Zero-Bit HIGH ist, wenn Akkumulator-Inhalt und das zum Vergleich herangezogene Datenwort gleich sind.

```
KEYIN   IN   04      Tastencode nach A einlesen
        CPI  42      (A) mit "42" vergleichen
        JZ   CTEB    Z=1 bei (A)=42:
                     weiter bei CTEB
        xx   ...     sonst hier weiter
```

Beispiel 2.5.2:

Ein Unterprogramm soll den Inhalt der Registerpaare D&E und H&L miteinander vergleichen, um beispielsweise festzustellen, ob bei einer Datenübertragung die Endadresse erreicht ist.

Beim Rücksprung aus dem hier aufgeführten Unterprogramm COMPAR ist das Zero-Bit auf HIGH, wenn die Inhalte beider Registerpaare gleich sind (vgl. Beispiele 2.1.2 und 3.2.1).

```
COMPAR  MOV  A,E      (E) nach A
        CMP  L       (A) mit (L) vergleichen
        RNZ          ungleich: Rücksprung mit Z=0
        MOV  A,D     (D) nach A
        CMP  H       (A) mit (H) vergleichen
        RET          nur bei Gleichheit ist Z=1
```

2.6 Schiebebefehle

Mit den Schiebebefehlen wird der Akkumulator-Inhalt zyklisch um eine Bitposition links bzw. rechts verschoben; beim Linksschieben gelangt das MSB, beim Rechtsschieben das LSB ins Carry-Bit. Bei den erweiterten Schiebebefehlen RAL und RAR wird das Carry-Bit als neuntes Bit mit in den Schiebezyklus einbezogen.

RLC — **rotate left**

Den Akkumulator-Inhalt zyklisch um ein Bit links verschieben.

RLC: 07

Anm.: Das MSB gelangt ins C-Bit und ins LSB.

RAL — **rotate left through carry**

Den Akkumulator-Inhalt unter Einbeziehung des C-Bits zyklisch um ein Bit links verschieben.

RAL: 17

Anm.: Das MSB gelangt ins C-Bit, und das C-Bit ins LSB.

RRC — **rotate right**

Den Akkumulator-Inhalt zyklisch um ein Bit rechts verschieben.

RRC: OF

Anm.: Das LSB gelangt ins C-Bit und ins MSB.

RAR — **rotate right through carry**

Den Akkumulator-Inhalt unter Einbeziehung des C-Bits zyklisch um ein Bit rechts verschieben.

RAR: 1F

Anm.: Das LSB gelangt ins C-Bit und das C-Bit ins MSB.

Es wird, wie oben beschrieben, nur das Carry-Bit beeinflußt.

Beispiel 2.6.1:

Um den Pegel am seriellen CPU-Eingang SID per Programm abzufragen, kann man ihn beispielsweise mit dem RIM-Befehl ins MSB des Akkumulators transportieren und anschließend mit einem Links-Schiebebefehl ins Carry-Bit bringen, um einen bedingten Sprung (JUMP ON CARRY/ JUMP ON NO CARRY) anzuschließen (vgl. Abschnitte 1.6.2 und 3.1.2).

```
SIDIN   RIM             SID-Pegel nach A (MSB)
        NOP             weiter beim nächsten Befehl
        RLC             (A), MSB ins C-Bit schieben
        JC    CTE261    SID=HIGH: weiter bei CTE261
        xx    ...       sonst hier weiter
```

Beispiel 2.6.2:

Um den Akkumulator-Inhalt mit Vielfachen von Zwei zu multiplizieren <bzw. zu dividieren>, braucht man ihn nur entsprechend oft (pro Zweier-Potenz einmal) nach links <bzw. rechts> zu verschieben.

Die Multiplikation des Akkumulator-Inhalts mit Zwei erreicht man auch durch Einsatz des ADD-A-Befehls der den Akkumulator-Inhalt (vgl. Beispiel 3.1.1).

3 Beeinflussung des Programmablaufs
3.1 Sprungbefehle

Es gibt zwei Befehle, durch die die Programmausführung an einer anderen als der nächstfolgenden Adresse fortgesetzt wird. Beim JMP-Befehl steht die Zieladresse explizit im zweiten und dritten Wort des Befehls; bei PCHL wird zu derjenigen Adresse verzweigt, die im Registerpaar H&L steht. Der NOP-Befehl hat nur die Aufgabe, Speicherzellen zu reservieren (Platzhalte-Befehl).

JMP xxyy — **jump**

Sprung zur Adresse xxyy (unbedingter Sprung).

JMP xxyy: C3 yy xx

PCHL — **move H&L to PC**

Den Inhalt des Registerpaars H&L in den Programmzähler PC transportieren (unbedingter, indirekter Sprung).

PCHL: E9

NOP — **no operation**

Programmfortsetzung beim nächsten Befehl.

NOP: 00

Keiner dieser Befehle beeinflußt eins der Zustandsbits im FLAG-Register.

Beispiel 3.1.1:

In einem Programmteil kann der Akkumulator die Werte 0...3 annehmen. Resultierend hieraus soll das Programm zu einer der vier Adressen 012F, 02BD, 030A bzw. 0479 verzweigen (PCHL-Befehl). Die Zieladressen stehen nacheinander in einer Sprungtabelle JMPTAB (die untere Adreß-hälfte jeweils zuerst; vgl. Beispiel 3.3.2).

```
BRANCH   LXI   H,JMPTAB  Anfangsadresse der Sprung-
                        tabelle nach H&L
         ADD   A        (A) verdoppeln
         ADD   L        2*(A) zur Tabellen-Anfangs-
                        adresse addieren
         MOV   L,A      (A)+(L) zurück nach L
         MOV   E,m      Zieladresse (untere Hälfte)
         INX   H        Adreßregister erhöhen
         MOV   D,m      Zieladresse (obere Hälfte)
                        (D&E) = Adresse Sprungziel
         XCHG           (D&E) nach (H&L)
         PCHL           Zieladresse in den Programm-
                        zähler (unbedingter Sprung)

JMPTAB   2F             erstes Ziel (untere Hälfte)
         01             erstes Ziel (obere Hälfte)
         BD             zweites Ziel (untere Hälfte)
         02             zweites Ziel (obere Hälfte)
         0A             drittes Ziel (untere Hälfte)
         03             drittes Ziel (obere Hälfte)
         79             viertes Ziel (untere Hälfte)
         04             viertes Ziel (obere Hälfte)
```

Beispiel 3.1.2:

Im Beispiel 1.6.3 sollen zwei Programm-Versionen möglich sein (Pegel entweder invertiert oder nicht invertiert ausgeben). Im zweiten Fall wird der CMA-Befehl nicht benötigt, und man kann ihn (z.B. während einer Testphase) durch den NOP-Befehl ersetzen, um den restlichen Teil des Programms nicht neu schreiben (bzw. verschieben) zu müssen.

noch 3.1: Die bedingten Sprungbefehle JZ, JC, JPE und JM werden nur dann ausgeführt, wenn das betreffende Zustandsbit Z, C, P bzw. S gesetzt ist; andernfalls geht die Programmausführung beim nächsten Befehl weiter.

JZ xxyy — jump on zero

Sprung zur Adresse xxyy, wenn das Z-Bit gesetzt (HIGH) ist (bedingter Sprung).

JZ xxyy: CA yy xx

JC xxyy — jump on carry

Sprung zur Adresse xxyy, wenn das C-Bit gesetzt (HIGH) ist (bedingter Sprung).

JC xxyy: DA yy xx

JPE xxyy — jump on parity even

Sprung zur Adresse xxyy, wenn das P-Bit gesetzt (HIGH) ist (bedingter Sprung).

JPE xxyy: EA yy xx

JM xxyy — jump on minus

Sprung zur Adresse xxyy, wenn das S-Bit gesetzt (HIGH) ist (bedingter Sprung).

JM xxyy: FA yy xx

Keiner dieser Befehle beeinflußt eins der Zustandsbits im FLAG-Register.

Beispiel 3.1.3:

Resultierend aus dem Pegel am seriellen CPU-Eingang SID soll ein Programm unterschiedliche Reaktionen ausführen: Bei HIGH-Pegel an SID soll es zur symbolischen Adresse CTE313 springen und bei LOW mit dem nächsten Befehl fortfahren.

Dazu kann man beispielsweise den SID-Pegel ins MSB des Akkumulators bringen (RIM-Befehl), alle übrigen Bits 0...6 löschen (ANI-Befehl; Beeinflussung der Zustandsbits) und bei gesetztem MSB (damit ist auch das Sign-Bit HIGH) zu CTE313 verzweigen (Befehl JUMP ON MINUS; vgl. Beispiel 1.6.2).

```
SIDIN2   RIM          SID-Pegel nach A (MSB)
         ANI  80      Bits 0...6 löschen
         JM   CTE313  MSB=HIGH: weiter bei CTE313
         xx   ...     sonst hier weiter
```

noch 3.1: Die bedingten Sprungbefehle JNZ, JNC, JPO und JP werden nur dann ausgeführt, wenn das betreffende Zustandsbit Z, C, P bzw. S nicht gesetzt ist; andernfalls geht die Programmausführung beim nächsten Befehl weiter.

JNZ xxyy — jump on not zero

Sprung zur Adresse xxyy, wenn das Z-Bit nicht gesetzt (LOW) ist (bedingter Sprung).

JNZ xxyy: C2 yy xx

JNC xxyy — jump on no carry

Sprung zur Adresse xxyy, wenn das C-Bit nicht gesetzt (LOW) ist (bedingter Sprung).

JNC xxyy: D2 yy xx

JPO xxyy — jump on parity odd

Sprung zur Adresse xxyy, wenn das P-Bit nicht gesetzt (LOW) ist (bedingter Sprung).

JPO xxyy: E2 yy xx

JP xxyy — jump on plus

Sprung zur Adresse xxyy, wenn das S-Bit nicht gesetzt (LOW) ist (bedingter Sprung).

JP xxyy: F2 yy xx

Keiner dieser Befehle beeinflußt eins der Zustandsbits im FLAG-Register.

Beispiel 3.1.4:

Eine Operation, beispielsweise das Auffüllen von Speicherzellen mit "00", beginnend bei einer bestimmten Adresse (hier bei 2FE0), soll mehrere Male wiederholt werden (Durchlaufen einer Schleife; hier dezimal 24 Mal).

Dann lädt man dazu vor dem Schleifen-Einsprung einen Zähler (hier Register C) mit einem Multiplikator, der pro Schleifendurchlauf um Eins verringert wird, und der damit bestimmt, wie oft die Schleife durchlaufen wird.

Die Endabfrage (Ist der Schleifenzähler Null?) wird über den bedingten Sprung JUMP ON NOT ZERO vorgenommen (zurück zum Schleifenanfang, solange der Multiplikator noch nicht auf Null gezählt ist; vgl. Beispiel 1.1.1).

```
CLEAR    MVI   C,18      Schleifenzähler = 24 dez.
         LXI   H,2FE0    Anfangsadresse nach H&L
LOP314   MVI   m,00      00 in RAM-Zelle
         INX   H         Adreßregister erhöhen
         DCR   C         Schleifenzähler erniedrigen
         JNZ   LOP314    nicht Null:weiter bei LOP314
         xx    ...       sonst hier weiter
```

3.2 Unterprogramm-Aufruf

Beim Unterprogramm-Aufruf wird das aufrufende Programm nur vorübergehend verlassen, d.h. nach dem Durchlaufen des Unterprogramms erfolgt automatisch der Rücksprung ins aufrufende Programm. Dazu wird (wiederum automatisch) vor dem Sprung ins Unterprogramm diejenige Adresse in den Stack überschrieben, bei der die Programmausführung fortgesetzt werden soll (bei dem auf CALL folgenden Befehl).

CALL xxyy **call**

Sprung zu dem bei Adresse xxyy beginnenden Unterprogramm (unbedingter UP-Aufruf).

CALL xxyy: CD yy xx

Es werden hierdurch keine Zustandsbits im FLAG-Register beeinflußt.

Beispiel 3.2.1:

Bei einer Datenübertragung (Ausgabe über den Kanal 00) muß nach jeder Ausgabe geprüft werden, ob die im Registerpaar D&E stehende Endadresse erreicht ist (Vergleich von D&E und H&L).

Da dieser Vergleich auch an anderen Stellen des Programms benötigt wird, kleidet man diese Aktivität zweckmäßigerweise in ein Unterprogramm ein. Beim Rücksprung aus diesem Unterprogramm COMPAR ist das Z-Bit HIGH, wenn die Inhalte der Registerpaare D&E und H&L gleich sind (vgl. Beispiele 2.1.2 und 2.5.2).

```
SERIEL   LXI  H,2800   Anfangsadresse nach H&L
         LXI  D,2AFF   Endadresse nach D&E
LOP321   MOV  A,m      Datenwort holen
         OUT  00       (A) über Kanal 00 ausgeben
         INX  H        Adresse erhöhen
         CALL COMPAR   Endadresse erreicht?
         JNZ  LOP321   nein: weiter bei LOP321
         xx   ...      sonst hier weiter
```

noch 3.2: Bei den bedingten CALL-Befehlen CZ, CC, CPE und CM erfolgt der Sprung ins Unterprogramm nur dann, wenn das betreffende Zustandsbit Z, C, P bzw. S gesetzt ist; andernfalls geht die Programmausführung beim nächsten Befehl weiter.

==========

CZ xxyy **call on zero**

==========

Sprung zu dem bei Adresse xxyy beginnenden Unterprogramm, wenn das Z-Bit gesetzt (HIGH) ist (bedingter UP-Aufruf).

CZ xxyy: CC yy xx

==========

CC xxyy **call on carry**

==========

Sprung zu dem bei Adresse xxyy beginnenden Unterprogramm, wenn das C-Bit gesetzt (HIGH) ist (bedingter UP-Aufruf).

CC xxyy: DC yy xx

==========

CPE xxyy **call on parity even**

==========

Sprung zu dem bei Adresse xxyy beginnenden Unterprogramm, wenn das P-Bit gesetzt (HIGH) ist (bedingter UP-Aufruf).

CPE xxyy: EC yy xx

==========

CM xxyy **call on minus**

==========

Sprung zu dem bei Adresse xxyy beginnenden Unterprogramm, wenn das S-Bit gesetzt (HIGH) ist (bedingter UP-Aufruf).

CM xxyy: FC yy xx

Keiner dieser Befehle beeinflußt eins der Zustandsbits im FLAG-Register.

Beispiel 3.2.2:

Die einzelnen Bits des Akkumulators steuern seriell nacheinander eine Peripherieschaltung an (Ausgabe über Kanal 80).

Bei HIGH-Pegel benötigt die periphere Schaltung zum Einschwingen doppelt so lange Zeit (2 ms) wie bei LOW-Pegel (1 ms). Je nach Pegel wird daher das Unterprogramm zur Zeitverzögerung ein- oder zweimal aufgerufen (bedingter Unterprogramm-Aufruf CALL ON CARRY).

```
OUTSLW   LXI   H,2A00    Anfangsadresse nach H&L
         MOV   A,m       Datenwort nach A
         OUT80           (A) über Kanal 80 ausgeben
         RLC             MSB ins Carry-Bit schieben
         CC    DELY1     nur bei C=1 das Unterpro-
                         gramm DELY1 aufrufen
         CALL  DELY1     DELY1 aufrufen
         xx    ...

DELY1    MVI   A,8E      1 ms Laufzeit (bei 4,19 MHz)
DELOP    DCR   A         (A) erniedrigen
         JNZ   DELOP     nicht Null: weiter bei DELOP
         RET             sonst Rücksprung
```

noch 3.2: Bei den bedingten CALL-Befehlen CNZ, CNC, CPO und CP erfolgt der Sprung ins Unterprogramm nur dann, wenn das betreffende Zustandsbit Z, C, P bzw. S nicht gesetzt ist; andernfalls geht die Programmausführung beim nächsten Befehl weiter.

CNZ xxyy — call on not zero

Sprung zu dem bei Adresse xxyy beginnenden Unterprogramm, wenn das Z-Bit nicht gesetzt (LOW) ist (bedingter UP-Aufruf).

CNZ xxyy: C4 yy xx

CNC xxyy — call on no carry

Sprung zu dem bei Adresse xxyy beginnenden Unterprogramm, wenn das C-Bit nicht gesetzt (LOW) ist (bedingter UP-Aufruf).

CNC xxyy: D4 yy xx

CPO xxyy — call on parity odd

Sprung zu dem bei Adresse xxyy beginnenden Unterprogramm, wenn das P-Bit nicht gesetzt (LOW) ist (bedingter UP-Aufruf).

CPO xxyy: E4 yy xx

CP xxyy — call on plus

Sprung zu dem bei Adresse xxyy beginnenden Unterprogramm, wenn das S-Bit nicht gesetzt (LOW) ist (bedingter UP-Aufruf).

CP xxyy: F4 yy xx

Keiner dieser Befehle beeinflußt eins der Zustandsbits im FLAG-Register.

Beispiel 3.2.3:

Das Unterprogramm DIVIDE führt eine Division durch, bei der der Divisor im Registerpaar H&L steht. Dieses Unterprogramm darf jedoch nur dann aufgerufen werden, wenn eine Prüfung ergeben hat, daß der Inhalt von H&L ungleich Null ist (in diesem Fall ist das Zero-Bit LOW!); andernfalls soll die Programmausführung beim nächsten Befehl weitergehen.

```
DIV     MOV   A,L      untere Divisor-Hälfte nach A
        ORA   H        (A) und (H) ODER-verküpfen
                       (Zustandsbits beeinflussen)
        CNZ   DIVIDE   nur bei Z=0 das Unterpro-
                       gramm DIVID aufrufen
        xx    ...      sonst hier weiter
```

Beachten Sie bitte, auf welche effektive Weise hier der Inhalt des Registerpaars H&L auf Null abgefragt wird: Ohne an H&L etwas zu verändern, wird der Inhalt eines Registers in den Akkumulator transportiert und mit dem Inhalt des anderen Registers ODER-verknüpft; das im Akkumulator gebildete Ergebnis ist nur dann Null, wenn der Inhalt beider Register H und L Null ist. In diesem Fall wird das Zero-Bit gesetzt, was mit dem anschließenden bedingten Unterprogramm-Aufruf (CALL ON NOT ZERO) abgefragt werden kann.

noch 3.2: Die acht Restart-Befehle sind Einwort-Sprungbefehle ins Unterprogramm, bei denen die Zieladresse im Befehl implizit enthalten ist. Auch hierbei wird, wie bei allen Unterprogramm-Aufrufen, die Adresse des nächsten Befehls in den Stack überschrieben, damit nach dem Rücksprung aus dem Unterprogramm die Programmausführung bei dieser Adresse fortgesetzt werden kann.

Diese Befehle können ohne weiteres im Anwenderprogramm verwendet werden, obwohl sie primär für die Verarbeitung externer Programmunterbrechungen am CPU-Eingang INTR vorgesehen sind. Als Reaktion auf eine Aktivierung an diesem Eingang unterbricht die CPU die laufende Programmausführung, liest die auf dem Datenbus anliegende Information ein und verarbeitet sie als Befehl. Wenn die periphere, unterbrechende Stelle zu diesem Zeitpunkt einen der Restart-Befehle auf den Datenbus schaltet, lassen sich ohne großen Hardware-Aufwand acht verschiedene Sprungziele unterscheiden, zu denen als Reaktion auf einen Interrupt verzweigt werden soll.

RST n **restart**

Einwort-Unterprogramm-Aufruf mit fester Zieladresse.

n	0	1	2	3	4	5	6	7
RST n:	C7	CF	D7	DF	E7	EF	F7	FF
Zieladresse:	0000	0008	0010	0018	0020	0028	0030	0038

Keiner dieser Befehle beeinflußt eins der Zustandsbits im FLAG-Register.

Beispiel 3.2.4:

Sofern im Programm der Befehl EI (Enable Interrupt) durchlaufen worden ist, akzeptiert die CPU externe Programmunterbrechungen, die durch HIGH-Pegel an den Unterbrechungseingängen ausgelöst werden.

Wenn beispielsweise vier externe Stellen zum INTR-Eingang Zugriff haben (über eine Hardware-ODER-Verknüpfung), können diese mit einfachsten Mitteln einen Interrupt-Vektor (Sprungziel für die Interrupt-Service-Routine) erzeugen.

Nach Aktivierung des INTR-Eingangs erzeugt der Prozessor (außer dem INTA-Signal) einen RD-Impuls, mit dem er die Information vom Datenbus einliest und als Befehl verarbeitet.

Liegen zu diesem Zeitpunkt alle Datenbits auf HIGH (z.B. über hochohmige Pull-up-Widerstände), wird "FF" eingelesen, was dem **RST-7**-Befehl entspricht (festes Sprungziel 0038 für eine der unterbrechenden Stellen).

Legt die unterbrechende Stelle zeitgleich mit dem RD-Impuls eins der Datenbits 3, 4 oder 5 auf LOW, liest die CPU entsprechend "F7", "EF" bzw. "DF" ein, was den Befehlen **RST 6**, **RST 5** bzw. **RST 3** entspricht mit den zugehörigen Sprungzielen für drei weitere unterbrechende Stellen.

3.3 Unterprogramm-Rücksprung

Der Return-Befehl schließt in der Regel jedes Unterprogramm ab; bei seiner Ausführung wird (automatisch) diejenige Adresse aus dem Stack geholt, die beim Sprung ins Unterprogramm dorthin überschrieben worden ist. Diese Adresse wird in den Programmzähler geladen, woraufhin die Programmausführung dort fortgesetzt wird.

RET **return**

Rücksprung aus dem Unterprogramm (unbedingter Rücksprung).

RET: C9

Eine Beeinflussung der Zustandsbits erfolgt hierdurch nicht.

Beispiel 3.3.1:

Der am Schluß eines Unterprogramms (dazu gehört auch jede Interrupt-Service-Routine!) stehende Befehl RETURN bewirkt den automatischen Rücksprung ins übergeordnete, aufrufende Programm (vgl. Beispiel 3.2.2). Das geschieht dadurch, daß der Inhalt der zu diesem Zeitpunkt vom Stack-Pointer adressierten Speicherzellen direkt in den Programmzähler transportiert wird, was zur Programmfortsetzung bei dieser Adresse führt.

Beispiel 3.3.2:

Den eben geschilderten Sachverhalt kann man auch für einen völlig anderen Zweck ausnutzen: Im Beispiel 3.1.1 verzweigt das Programm zu einer von vier möglichen Zieladressen, ohne (wie z.B. bei einem Unterprogramm) zu einem zentralen Punkt zurückzukehren, an dem sich alle möglichen Sprungziele wieder treffen.

Wenn man vor Errechnen des Sprungziels die Adresse des Fortsetzungspunktes (hier CTE332) in den Stack überträgt (PUSH-Befehl), kann man die vier Programmzweige mit RETURN abschließen, und bei Ausführung dieses RET-Befehls erfolgt dann dort die Programmfortsetzung.

```
          BRANCH   LXI  H,CTE332  Fortsetzungsadresse nach H&L
                   PUSH H         CTE322-Adresse in den Stack
                   LXI  H,JMPTAB  ...
                   xx   ...       Adreßberechnung
                   xx   ...       (vgl. Beispiel 3.1.1)
                   PCHL           ...
          CTE332   xx   ...       Programmmfortsetzung nach
                                  Rückkehr aus den vier Teilen
```

Die vier Programmzweige, zu denen zu denen das Hauptprogramm mit dem PCHL-Befehl verzweigen kann, müssen in diesem Fall mit dem Befehl **RET** abgeschlossen werden.

noch 3.3: Die bedingten Rücksprungbefehle RZ, RC, RPE und RM werden nur dann ausgeführt, wenn das betreffende Zustandsbit Z, C, P bzw. S gesetzt ist; andernfalls geht die Programmausführung beim nächsten Befehl weiter.

RZ — return on zero

Rücksprung aus dem Unterprogramm, wenn das Z-Bit gesetzt (HIGH) ist (bedingter Rücksprung).

RZ: C8

RC — return on carry

Rücksprung aus dem Unterprogramm, wenn das C-Bit gesetzt (HIGH) ist (bedingter Rücksprung).

RC: D8

RPE — return on parity even

Rücksprung aus dem Unterprogramm, wenn das P-Bit gesetzt (HIGH) ist (bedingter Rücksprung).

RPE: E8

RM — return on minus

Rücksprung aus dem Unterprogramm, wenn das S-Bit gesetzt (HIGH) ist (bedingter Rücksprung).

RM: F8

Keiner dieser Befehle beeinflußt eins der Zustandsbits im FLAG-Register.

Beispiel 3.3.3:

Das Unterprogramm BINBC setzt eine im Akkumulator stehende Binärzahl (die kleiner ist als dezimal 100) in zwei BCD-Digits um.

Dazu Subtrahiert es vom Akkumulator-Inhalt so oft 0A (dezimal 10), bis das Ergebnis dadurch negativ wird. In diesem Fall erfolgt der Rücksprung ins Hauptprogramm (Befehl RETURN ON MINUS) mit der Zehner-Stelle in Register B und der Einer-Stelle in Register C.

```
BINBC    MVI   B,00     Zehnerstelle löschen
LOP333   MOV   C,A      (A) nach C retten
         SUI   0A       von (A) dezimal 10 abziehen
ENDSUB   RM             bei negativem Ergebnis (S=1)
                        Rücksprung
         INR   B        sonst Zehner erhöhen
         JMP   LOP333   und weiter subtrahieren
```

noch 3.3: Die bedingten Rücksprungbefehle RNZ, RNC, RPO und RP werden nur dann ausgeführt, wenn das betreffende Zustandsbit Z, C, P bzw. S nicht gesetzt ist; andernfalls geht die Programmausführung beim nächsten Befehl weiter.

RNZ — **return on not zero**

Rücksprung aus dem Unterprogramm wenn das Z-Bit nicht gesetzt (LOW) ist (bedingter Rücksprung).

```
RNZ: C0
```

RNC — **return on no carry**

Rücksprung aus dem Unterprogramm, wenn das C-Bit nicht gesetzt (LOW) ist (bedingter Rücksprung).

```
RNC:  D0
```

RPO — **return on parity odd**

Rücksprung aus dem Unterprogramm, wenn das P-Bit nicht gesetzt (LOW) ist (bedingter Rücksprung).

```
RPO:  E0
```

RP — **return on plus**

Rücksprung aus dem Unterprogramm, wenn das S-Bit nicht gesetzt (LOW) ist (bedingter Rücksprung).

```
RP:  F0
```

Keiner dieser Befehle beeinflußt eins der Zustandsbits im FLAG-Register.

Beispiel 3.3.4:

Nach Aufruf des Unterprogramms COMPAR (vgl. Beispiele 2.1.2 und 3.2.1) soll das Zero-Bit auf HIGH gesetzt sein, wenn die Inhalte der Registerpaare D&E und H&L gleich sind.

Diese Bedingung ist nur dann erfüllt, wenn beide Teilvergleiche (Inhalt der Register E und L sowie Inhalt der Register D und H gleich) positiv ausfallen.

Führt bereits der erste Vergleich zu einem negativen Resultat (d.h. ungleiche Register-Inhalte), wird das Unterprogramm vorzeitig verlassen (Befehl RETURN ON NOT ZERO). Am LOW-Zustand des Zero-Bits erkennt das Hauptprogramm in diesem Fall, daß keine Gleichheit vorlag.

Nach dem zweiten Teilvergleich (CMP-H-Befehl) kann unmittelbar der Rücksprung erfolgen; diese Stelle wird nur dann erreicht, wenn der erste Teilvergleich positiv ausgefallen ist. Ist dies auch beim zweiten Teilvergleich der Fall, erfolgt der Rücksprung mit gesetztem Zero-Bit, andernfalls ist das Z-Bit LOW; das aufrufende Hauptprogramm bekommt auf diese Weise die geforderte Information über Gleichheit bzw. Ungleichheit der Registerinhalte.

```
COMPAR   MOV   A,E      (E) nach A
         CMP   L        (A) mit (L) vergleichen
         RNZ            ungleich: Rücksprung mit Z=0
         MOV   A,D      (D) nach A
         CMP   H        (A) mit (H) vergleichen
         RET            nur bei Gleichheit ist Z=1
```

3.4 Verarbeitung von Programmunterbrechungen (Interrupts)

Die Befehle RIM und SIM haben eine Doppelfunktion; sie dienen einerseits zum Datenverkehr zwischen dem Akkumulator und den seriellen CPU-Anschlüssen SID bzw. SOD, und andererseits lesen bzw. laden sie das Interrupt-Masken-Register (vgl. Abschnitt 1.6). In diesem Register wird festgelegt, welcher der drei CPU-Unterbrechungseingänge RST5.5, RST6.5 bzw. RST7.5 für externe Unterbrechungen vorbereitet ("maskiert") wird.

Bei Aktivierung eines der CPU-Unterbrechungs-Eingänge TRAP, RST7.5, RST6.5, RST 5.5 oder INTR unterbricht der Prozessor nach Beendigung des augenblicklich in der Ausführung befindlichen Befehls sein laufendes Programm und springt zu einer festen Zieladresse (bei TRAP nach 0024, bei RST7.5 nach 003C, bei RST6.5 nach 0034 und bei RST5.5 nach 002C; bei INTR wird in der Regel ein Restart-Befehl eingelesen und entsprechend verzweigt (vgl. dazu Abschnitt 3.2).

Voraussetzung für jede Programmunterbrechung ist es, daß Interrupts generell freigegeben worden sind (Befehl "Enable Interrupt"); für die Unterbrechnungseingänge RST7.5, RST6.5 und RST5.5 ist zusätzlich noch das zugehörige Maskenbit im Interrupt-Masken-Register auf LOW zu setzen (vgl. SIM-Befehl).

RIM — **read interrupt mask**

Den Inhalt des Interrupt-Masken-Registers in den Akkumulator transportieren.

RIM: 20

Anm.: RIM transportiert außerdem den Pegel des seriellen CPU-Eingangs SID ins MSB des Akkumulators (vgl. Abschnitt 1.6).

SIM — **set interrupt mask**

Den Akkumulator-Inhalt ins Interrupt-Maskenregister transportieren; bevor externe Unterbrechungen möglich sind, muß zusätzlich noch der Befehl "Enable Interrupt" ausgeführt werden.

SIM: 30

Anm.: SIM kann außerdem den MSB-Pegel des Akkumulators zum seriellen CPU-Ausgang SOD transportieren (vgl. Abschnitt 1.6).

Keiner dieser Befehle beeinflußt eins der Zustandsbits im FLAG-Register.

Beispiel 3.4.1:

Nach Ausführung des Befehls **RIM** stehen im Akkumulator folgende Informationen:

- In den unteren drei Bits 0...2 stehen die mit den Unterbrechungseingängen RST5.5, RST6.5 und RST7.5 korrespondierenden Bits des Interrupt-Masken-Registers (HIGH bedeutet, daß der betreffende Unterbrechungseingang gesperrt ist).
- In den Bits 4...6 ist der Zustand der Eingänge RST5.5, RSt6.5 und RST7.5 ablesbar, unabhängig davon, welche Maskenbits gesetzt sind. **Achtung!** Zur Aktivierung des RST7.5-Eingangs genügt ein kurzer Impuls.
- Bit 7 hat denselben Pegel wie der serielle CPU-Eingang SID.

Beispiel 3.4.2:

Um die Unterbrechungseingänge RST5.5, RST6.5 und RST7.5 gezielt freigeben oder sperren zu können (SIM-Befehl, auf dem Umweg über den Akkumulator), muß man die drei unteren Bits im Interrupt-Masken-Register löschen (entspricht "freigeben") oder setzen (entspricht "sperren"). Zu dieser Modifikation muß Bit 3 auf HIGH sein, unabhängig davon, welcher der Unterbrechungseingänge freigegeben oder gesperrt ist.

Eine RST7.5-Auslösung kann mit HIGH-Pegel in Bit 4 wieder gelöscht werden, unabhängig davon, welche Maskenbits gesetzt sind.

Die Bits 5...7 beeinflussen den seriellen CPU-Ausgang SOD (vgl. Beispiel 1.6.3); im hier betrachteten Fall sollten sie daher auf LOW bleiben.

Um beispielsweise den RST5.5- und den RST7.5-Unterbrechungseingang freizugeben, sind die Bits 0 und 2 im Interrupt-Masken-Register auf LOW zu setzen (vgl. Beispiel 3.4.3):

```
ENINT   MVI  A,0A     Bits 0 und 2 auf LOW
        SIM  (A) ins Interrupt-Masken-
                     Register
        EI           Interrupts freigeben
        xx   ...     ...
```

noch 3.4: Externe Programmunterbrechungen sind erst dann möglich, wenn der hinter "Enable Interrupt" stehende Befehl ausgeführt worden ist. Nach einem Hardware-Reset und nach jeder externen Programmunterbrechung sind Interrupts generell gesperrt; am Ende einer Interrupt-Service-Routine muß daher jeweils erneut der Befehl "Enable Interrupt" durchlaufen werden. Er steht in der Regel unmittelbar vor dem Rücksprungbefehl RET.

Um nach der Freigabe von Interrupts externe Unterbrechungen wieder auszuschließen, muß der Befehl "Disable Interrupt" ausgeführt werden; unmittelbar nach dessen Ausführung reagiert die CPU nicht mehr auf Aktivierungen der Unterbrechungs-Eingänge.

Der Halt-Befehl dient dazu, Programme definiert abzuschließen, wenn vom Computer keine Aktivität mehr gefordert wird; dieser befehl kann aber auch eingefügt werden, wenn der Prozessor auf eine externe Programm-Unterbrechung warten soll.

EI — **enable interrupt**

Externe Programmunterbrechungen ermöglichen.

EI: FB

DI — **disable interrupt**

Externe Programmunterbrechungen sperren.

DI: F3

HLT — **halt**

Programmausführung bis zu einem Interrupt oder (Hardware-)Reset stoppen.

HLT: 76

Anm.: Nach Ausführung des HLT-Befehls gehen sämtliche Busleitungen in den hochohmigen Zustand.

Keiner dieser Befehle beeinflußt eins der Zustandsbits im FLAG-Register

Beispiel 3.4.3:

Nach einem Hardware-RESET sind im Interrupt-Masken-Register die unteren drei Bits 0...2 auf HIGH, d.h. die drei Unterbrechungseingänge RST5.5, RST6.5 und RST7.5 sind, wie auch die übrigen Interrupt-Eingänge, automatisch gesperrt.

Der EI-Befehl hebt diese generelle Sperre auf und setzt dazu ein internes IE-Flipflop (Interrupt Enable); damit sind bereits externe Programmunterbrechungen über die Eingänge TRAP und INTR möglich.

Unterbrechungen über die Eingänge RST5.5, RST6.6 oder RST7.5 sind erst dann möglich, wenn zusätzlich zum EI-Befehl die korrespondierenden Bits im Interrupt-Masken-Register auf LOW gesetzt worden sind (vgl. Beispiel 3.4.2).

```
ENINT   MVI  A,0A     RST5.5 und RST7.5 freigeben
        SIM            (A) ins Interrupt-Masken-
                       Register
        EI             vom übernächsten Befehl an
                       Interrupts zulassen
        xx   ...       ...
```

Beispiel 3.4.4:

Wenn bestimmte Stellen eines Programms nicht durch einen Interrupt unterbrochen werden sollen, wird mit dem Befehl **DI** das interne IE-Flipflop zurückgesetzt, wodurch sämtliche Interrupts gesperrt sind; am Zustand des Interrupt-Masken-Registers ändert sich dadurch nichts.

Beispiel 3.4.5:

Ein Programm soll zum Akkumulator-Inhalt den Inhalt des Registers B addieren und das Ergebnis am Ausgabekanal 00 bereitstellen; weitere Aktivitäten werden nicht verlangt, so daß der Prozessor anschließend mit dem HLT-Befehl in den Wartezustand gebracht werden kann; aus diesem Zustand kommt er nur durch einen Interrupt oder Hardware-RESET wieder heraus.

```
WAIT    ADD  B         (A) und (B) addieren
        OUT  00        (A) über Kanal 00 ausgeben
        HLT            Prozessor anhalten
```

Adresse				Maschinencode								Label	Assemblercode
				1.Byte		2.Byte		3.Byte		4.Byte			