Mikroprozessoren und Mikrorechner

Lehrheft 4

ITT Fachlehrgänge, Pforzheim

ITT

1., 2. und 3. Auflage, Mai 1977

Druck: Druckerei Seiter, 7535 Königsbach-Stein

Mikroprozessoren und Mikrorechner

Verfasser:
Dr. Jürgen Gerlach
C. D. Nabavi, B. Sc.
Forschungszentrum der
Standard Elektrik Lorenz AG
Stuttgart

6. 8080-Mikroprozessor-System

Nachdem Sie in den beiden vorherigen Lehrheften ausführlich mit dem hypothetischen Mikrorechner vertraut gemacht wurden, soll jetzt ein realer Mikroprozessor besprochen werden. Es wird das 8080-System behandelt, das heute mehr oder weniger Industriestandard geworden ist und bereits von mehreren Halbleiterherstellern gefertigt wird. Um den Einsatz des 8080-Systems einfacher zu machen, werden eine Reihe von weiteren ICs angeboten, damit letztlich ein funktionierender Mikrorechner entsteht. Hierzu gehören Speicher, Taktgeneratoren, Ein/Ausgabe-Bausteine, Systemcontroller, um nur die wichtigsten zu nennen. Wir werden auf diese Zusatzschaltungen noch näher eingehen.

6.1 Hardware-Eigenschaften des 8080-Mikroprozessors

Es handelt sich um einen 8-bit-Mikroprozessor, der in einem 40poligen Gehäuse untergebracht ist. Er ist in n-Kanal-Silicon-Gate-MOS-Technologie aufgebaut. Als Stromversorgungen werden −5 V, +5 V und +12 V benötigt. Die Pin-Belegung ist in Bild 6.1.1 dargestellt.

Bis auf die Takteingänge können die Ein- und Ausgänge direkt mit TTL-Bausteinen verbunden werden. Dies bedeutet aber nicht, daß sie mit TTL voll kompatibel sind.
Der Mikroprozessor 8080, nachfolgend kurz MP 8080 genannt, wird mit 2 sich nicht überlappenden Taktimpulsen Φ_1 und Φ_2 gesteuert (Pin 22 und 15). An diese Taktversorgung werden bestimmte Anforderungen gestellt, die jeweils im Datenblatt genau definiert sind. Die grundsätzliche Form der Taktversorgung zeigt Bild 6.1.2.
Aus diesem Bild können Sie errechnen, daß die Standardausführung des MP 8080 mit einem Takt zwischen 480 ns und 2 µs arbeitet. Dies entspricht einer Frequenz von 500 kHz bis ca. 2 MHz. In der Praxis wird für die Taktversorgung ein integrierter Taktgenerator (8224) angeboten, der über einen externen Quarz auf der gewünschten Taktfrequenz betrieben werden kann. Hierzu ist zu bemerken, daß dieser Generator aus einem Oszillator und einem Zählerteil besteht. Der Zählerteil teilt die Oszillatorfrequenz auf 1/9, d.h., der Quarz muß für eine 9mal höhere Frequenz ausgelegt sein als die gewünschte Taktfrequenz. Soll z.B. die Taktfrequenz 1 MHz betragen, so ist ein 9-MHz-Quarz erforderlich.
Die durch den Quarz festgelegte Taktfrequenz bestimmt die Arbeitsgeschwindigkeit des Rechnersystems. Zum Abrufen und Durchführen eines Befehles benötigt der Rechner eine ganz bestimmte Zeit. Diese Zeit wird als **Befehlszyklus** definiert. Wie später noch eingehend erläutert wird, enthält der MP 8080 1-, 2- und 3-Byte-Befehle. Je nach Befehlsart muß während des Abrufes der entsprechende Befehl aus dem Speicher geholt und in ein entsprechendes Register übertragen werden. Während der Ausführungsphase des Befehlszyklus wird der Befehl entsprechend dekodiert. Während eines Befehlszyklus muß der MP eine oder mehrere Operationen durchführen (1 bis 5 Operationen), wie zum Beispiel:
- Ausgabe der Adresse und der Statusinformation
- Erzeugen von Wartetakten
- Datenein- und -ausgabe
- Bearbeitung der Ready-Information
- Takte für interne Verarbeitung

Ein solcher Operationszyklus besteht wieder aus 3 bis 5 Taktzyklen. Damit kann ein voller Befehlszyklus je nach Art des Befehles aus 4 bis 18 Taktzyklen bestehen. Ein Taktzyklus ist dabei als Zeit t_C definiert.
Obwohl der MP 8080 ein 8-bit-Mikroprozessor ist, haben einige Register und der Befehlszähler eine Länge von 16 bit. Damit ist es möglich, 2^{16} = 65 536 Speicherwörter zu adressieren. Wie Sie aus Bild 6.1.1 entnehmen können, sind damit 16 Anschlüsse für die Speicheradressierung erforderlich (Adreßbus). Für die Datenübertragung (Ein- und Ausgabe) werden 8 Anschlüsse benötigt (Datenbus). Damit ist der Datenbus bidirektional, was durch die Doppelpfeile in Bild 6.1.1 gekennzeichnet ist. Inklusive Stromversorgung (4 Anschlüsse mit Masse), Taktversorgung (2 Anschlüsse), Adreßbus (16 Anschlüsse) und Datenbus (8 Anschlüsse) sind bereits 30 Pins des MP-8080-Bausteines belegt. Es bleiben nur 10 Anschlüsse für die notwendigen Steuerfunktionen übrig. Diese reichen aber nicht aus, so daß noch einige Steuersignale über den Datenbus im Zeitmultiplexbetrieb ausgegeben werden müssen. Dies ist in Bild 6.1.3 prinzipiell dargestellt.

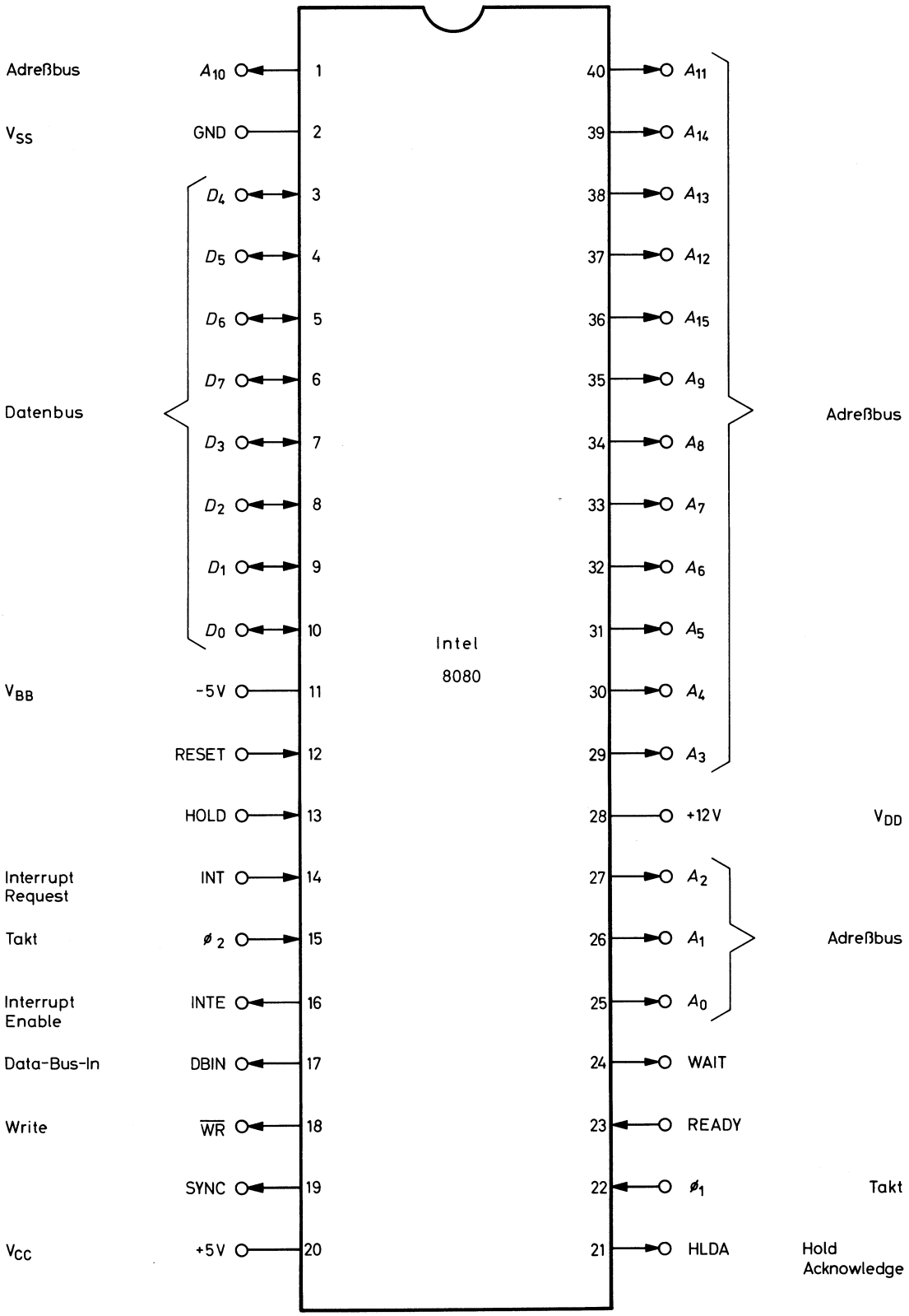
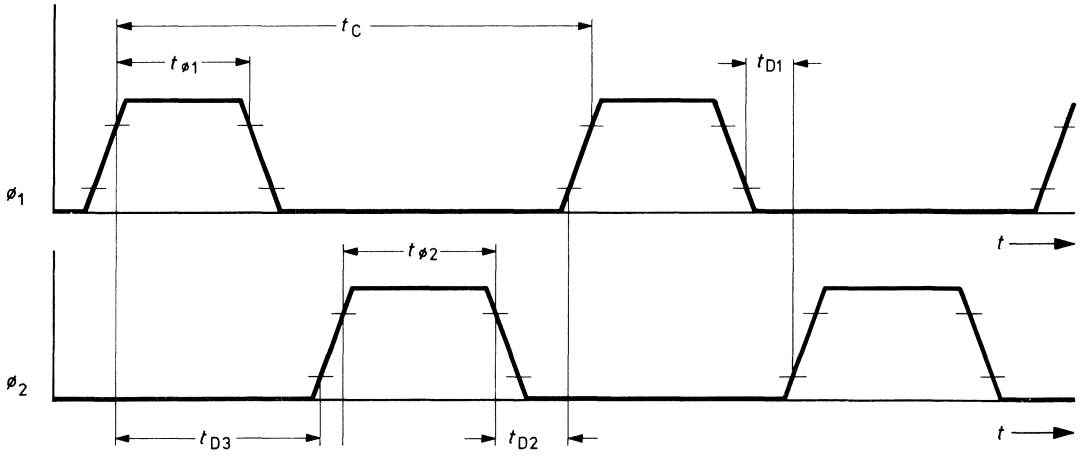


Bild 6.1.1
Intel-8080-Pin-Belegung

Die zusätzlichen Steuerfunktionen erscheinen zu bestimmten Zeitpunkten auf dem Datenbus. Damit sie zeitlich ausgeblendet werden können, liefert der MP 8080 ein Synchronsignal (SYNC), das 8 Flipflops (Latches) mit diesen Steuerfunktionen lädt. Da die Steuerfunktionen den momentanen Zustand (Status) des MP-Systems kennzeichnen, werden diese Informationen als Statusinformationen oder Statussignale bezeichnet. Am Ausgang der Status-Latches stehen folgende Statussignale zur Verfügung:



t_C = Taktperiode: min. 0,48 µs ; max. 2 µs
$t_{\phi 1}$ = Impulsdauer ϕ_1: min. 60 ns
$t_{\phi 2}$ = Impulsdauer ϕ_2: min. 220 ns
t_{D1} = Verzögerungszeit ϕ_1 bis ϕ_2: min. 0 ns
t_{D2} = Verzögerungszeit ϕ_2 bis ϕ_1: min. 70 ns
t_{D3} = Verzögerung gegen Vorderflanke: min. 80 ns

Bild 6.1.2
Taktversorgung des MP 8080 (Standardausführung)

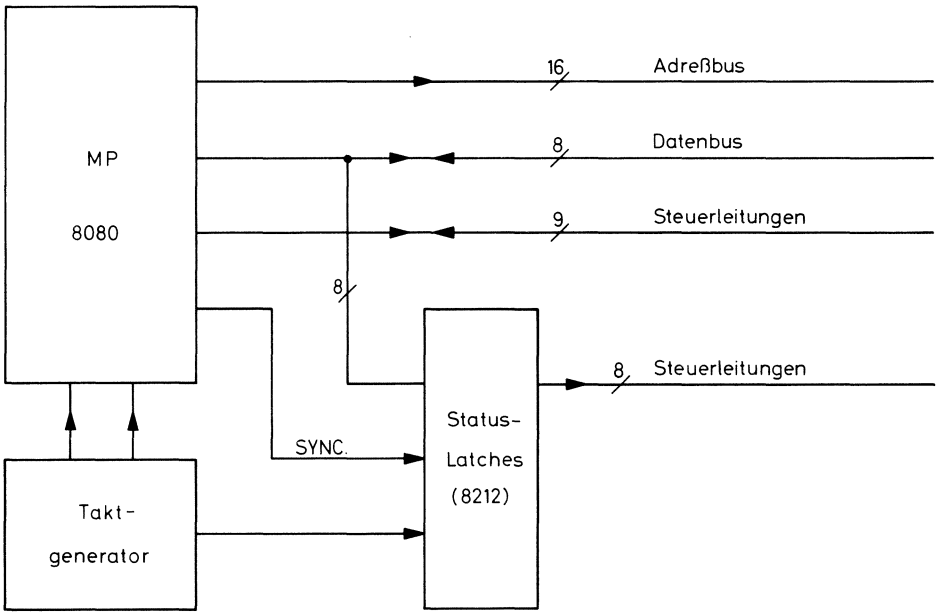


Bild 6.1.3
Prinzipschaltung zur Aufbereitung von 8 weiteren Steuersignalen

INTA (Interrupt Acknowledge):
Quittung für Interrupt-Versuch. Unter Interrupt versteht man eine Programmunterbrechung. Während der Programmunterbrechung wird eine sog. Interrupt-Service-Routine bearbeitet und danach im unteren Programm weitergefahren. Auf Interrupts kommen wir noch näher zu sprechen.

$\overline{WO}$ (Write or Output):
Diese Funktion besagt, daß der 8080 einen Speicherschreibzyklus ausführt oder eine Datenausgabe erfolgt.

STACK:
Bei STACK = H erfolgt ein Datentransport vom oder zum Stack-Bereich. Auf dem Adreßbus liegt die Adresse aus dem Stack-Pointer.

HLTA (Halt Acknowledge):
Der 8080 hat einen HALT-Befehl ausgeführt und somit seinen normalen Arbeitsrhythmus unterbrochen.

OUT:
Der MP 8080 gibt Daten zu einem Peripheriegerät aus.

M1:
Kennzeichnet das Lesen des ersten Bytes eines Befehles.

INP (Input):
Zeigt an, daß Daten von einem Peripheriegerät eingelesen werden.

MEMR (Memory Read):
Zeigt an, daß der Datenbus zum Lesen von Daten aus dem Speicher benutzt wird.

Die restlichen Anschlüsse des MP 8080 selbst haben folgende Funktionen:

RESET:
Löscht den Inhalt des Befehlszählers und startet den MP bei Adresse 0.

HOLD:
Bei HOLD = H wird der MP vorübergehend angehalten. Adreß- und Datenbus werden in den Hochimpedanzzustand geschaltet, so daß Peripheriegeräte das Bussystem benutzen können. Dies ist z.B. bei DMA-Betrieb (Direct-Memory-Access) der Fall, um Daten direkt in den Speicher zu laden.

HLDA (Hold Acknowledge):
Zeigt an, daß der MP sich im HOLD-Zustand befindet.

INT (Interrupt):
Unterbricht das laufende Programm und löst ein Interrupt-Subroutine aus.

INTE (Interrupt Enable):
Zeigt an, ob Interrupt (Programmunterbrechung) möglich ist.

DBIN (Data-Bus-In):
Zeigt den Zeitraum der Dateneingabe in den MP an.

$\overline{\text{WR}}$ (Write):
Solange $\overline{\text{WR}}$ im L-Zustand ist, gibt der MP Daten an Speicher oder Peripherie aus. Dieses Signal kann als Taktsignal für Speicher und Peripherie verwendet werden.

WAIT:
Dieses Signal gibt an, daß der MP sich im Wartezustand befindet.

READY:
Zeigt an, daß auf dem Datenbus Speicher- oder Eingabedaten bereitstehen, die vom MP eingelesen werden können.

Nachdem die Belegung der einzelnen Pins des MP 8080 besprochen wurde, wird in Bild 6.1.4 die Architektur des MP 8080 als Blockbild gezeigt.
Der MP 8080 bildet das Kernstück eines Mikrorechners. Wie bereits erwähnt, kann dieser Baustein jedoch nicht ohne zusätzliche Funktionseinheiten betrieben werden. In jedem Falle müssen Speicher- und Ein/Ausgabemöglichkeiten gegeben sein. Alle diese Funktionen können mit normaler Digitallogik realisiert werden. Um jedoch den Systementwurf zu vereinfachen,

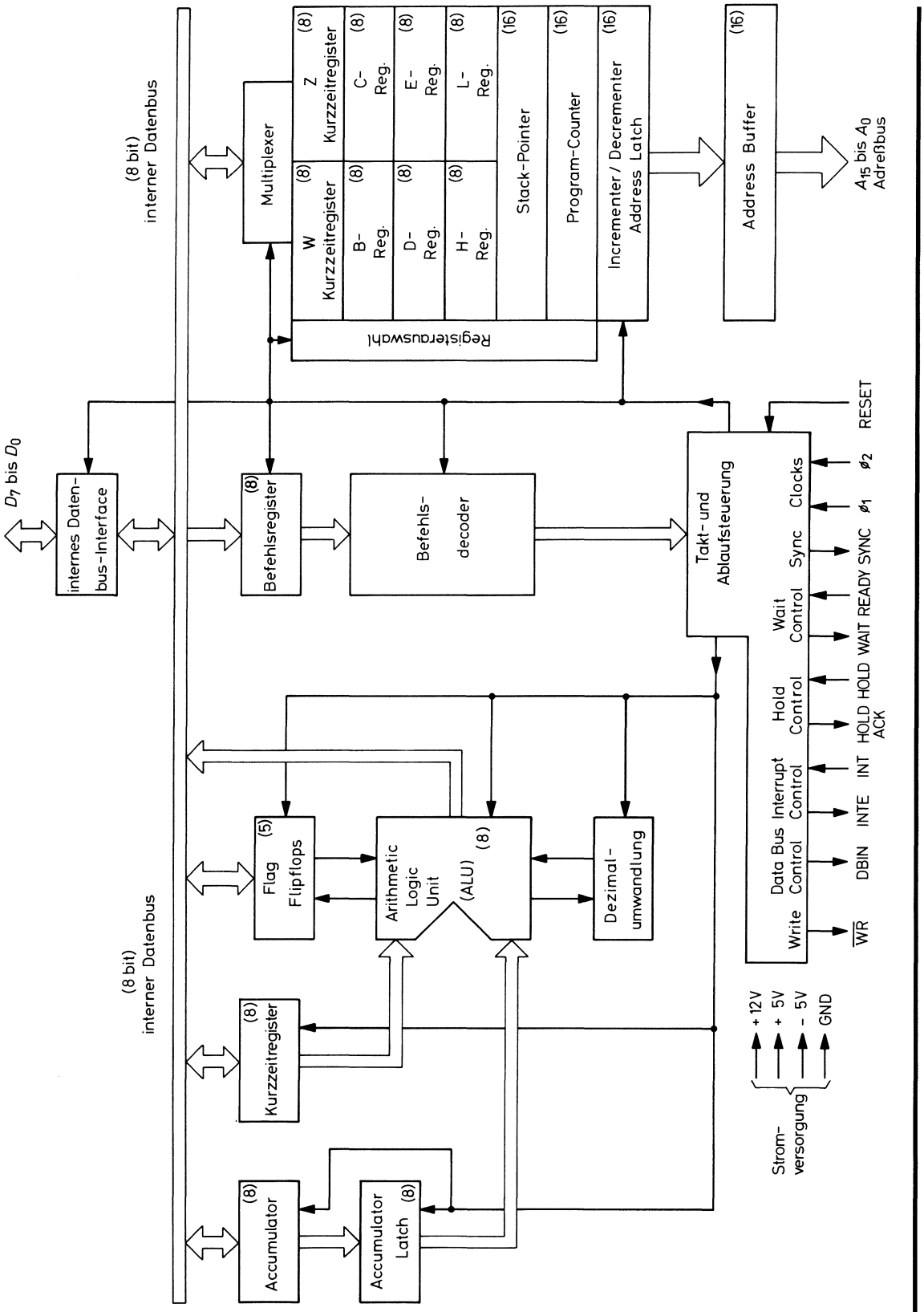


Bild 6.1.4
Blockbild des MP 8080

werden von den Halbleiterherstellern hochintegrierte Schaltungen angeboten, die den Systementwurf wesentlich erleichtern. In Bild 6.1.5 ist der prinzipielle Aufbau eines Mikrorechners dargestellt.
Die grundsätzliche Arbeitsweise eines solchen Systems ist folgende:

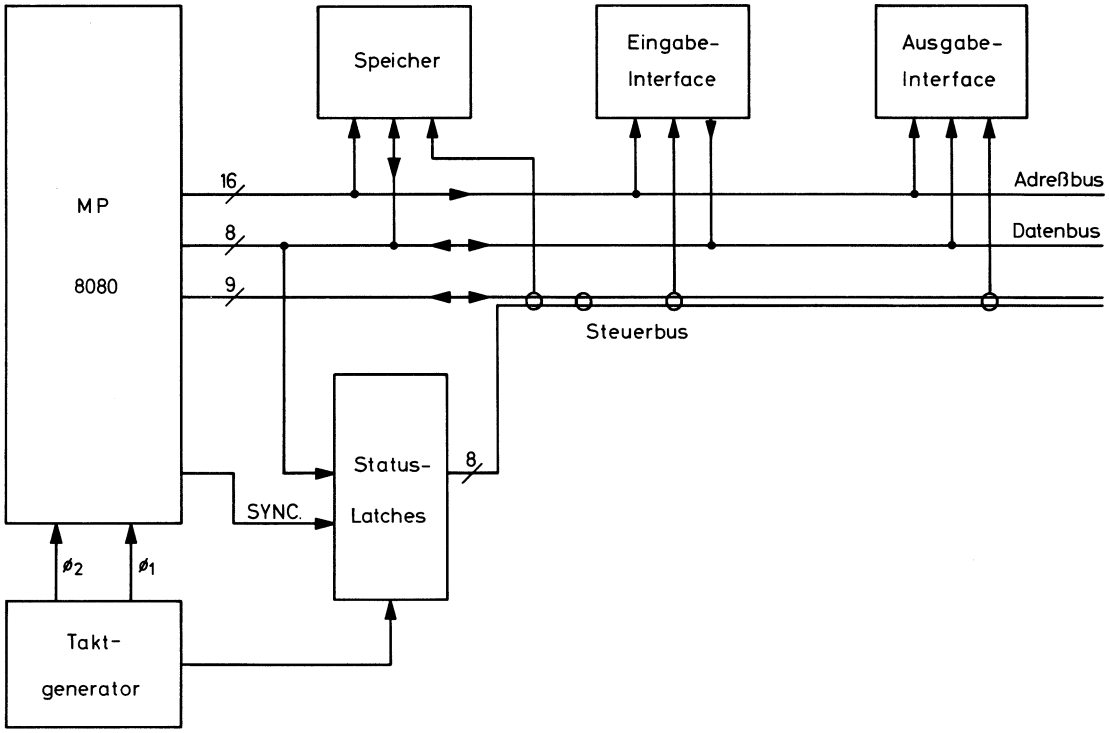


Bild 6.1.5
Blockbild eines einfachen Mikrorechners

– Der MP 8080 aktiviert über den Steuerbus Speicher- oder Ein/Ausgabeeinheiten, die über eine bestimmte Adresse (Adreßbus) gekennzeichnet werden.
– Der MP 8080 empfängt oder sendet Daten bzw. führt Operationen aus.
– Nach Beendigung eines Datentransfers oder einer Operation beginnt der MP 8080 mit dem gleichen Ablauf von neuem.

Der hier grob geschilderte Vorgang setzt sich aus mehreren Einzelschritten zusammen. Dabei spielt die Art der auszuführenden Befehle eine entscheidende Rolle. In einem späteren Kapitel werden wir auf weitere Einzelheiten zurückkommen.

6.2 Software-Eigenschaften

Die Software-Eigenschaften des MP 8080 unterscheiden sich vom hypothetischen Mikrorechner zunächst durch den wesentlich größeren Befehlsvorrat. Das beruht zum Teil auf der größeren Anzahl verfügbarer Register, da viele Befehle Operationen zwischen einzelnen Registern auslösen. Mehr Register ermöglichen andererseits aber auch einen flexibleren Programmaufbau.
Anhand von Bild 6.2.1 sollen die Eigenschaften der Register und Flags des MP 8080 erklärt werden.
Das erste Register (Flag-Register) führt in den angegebenen Stellen die bereits bekannten Flags Zero, Negativ und Carry. Neu hinzugekommen sind:
– Paritäts-Flag
Ist die Anzahl der Einsen in einem Codewort gerade, geht dieses Flag in den High-Zustand.
– Half-Carry-Flag
Dieses Flag zeigt einen Übertrag in der Wortmitte an und wird für Dezimalarithmetik benötigt.

Die nicht benötigten 3 Stellen in diesem Register sind konstant mit 0 bzw. 1 belegt. Der MP 8080 enthält nur ein Register (Register A), das als echter Akkumulator betrieben werden kann. Das Flag-Register und Register A können durch eine spezielle Adressierung als Registerpaar PSW (**P**rogramm **S**tatus **W**ord) angesprochen werden. Ebenso lassen sich auch die Register B und C bzw. D und E bei Bedarf zu Registerpaaren BC bzw. DE zusam-

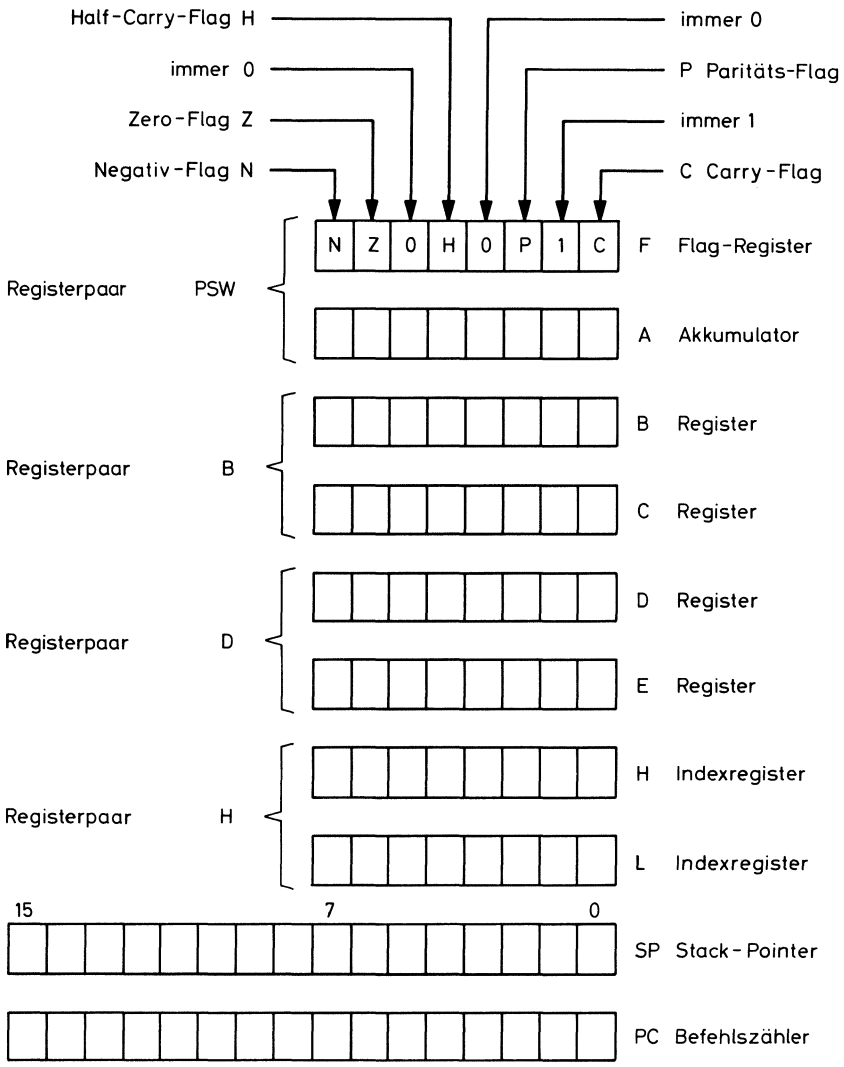


Bild 6.2.1
Die Register und Flags des MP 8080

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

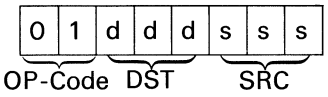
6.3 Befehlsvorrat des MP 8080

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 diesen 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

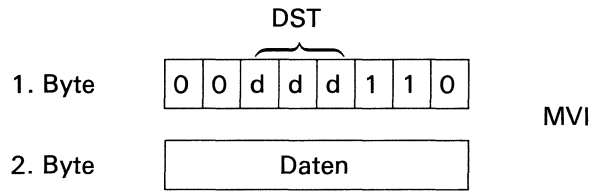
Maschinencode 0 1 1 1 0 1 1 0 ≙ 7 6_{16} vor. In Experiment 13 haben Sie diesen Befehl bereits als HLT-Befehl kennengelernt. Es ist nicht möglich, einen bestimmten Speicherplatz gleichzeitig als Datenquelle und Datenziel zu verwenden. Nachfolgend sind einige MOV-Befehle in der mnemonischen Schreibweise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

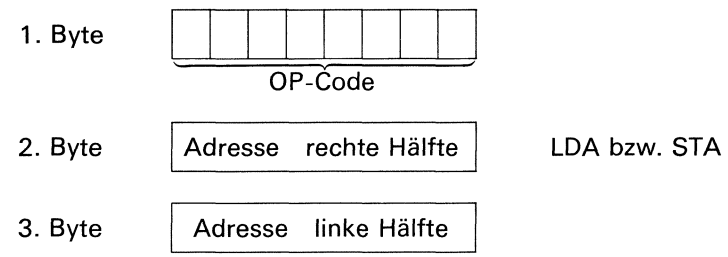


Da die Daten im 2. Byte angegeben sind, muß hier nur das Datenziel (DST) definiert werden. Dabei gilt wieder der Code nach Tab. 6.3.1.1.

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.

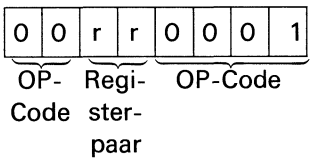
Exp. 14

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:

1. Byte



2. Byte — Daten

3. Byte — Daten

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:

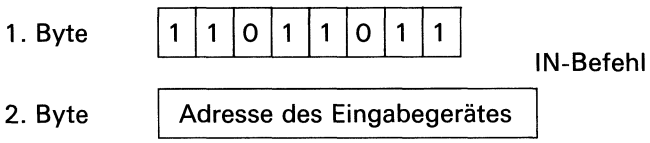
mnemonische Schreibweise	Maschinencode	Hex.-Code	Kommentar
XCHG	1 1 1 0 1 0 1 1	E B	(H)→D (D)→H (L)→E (E)→L

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

Exp. 15

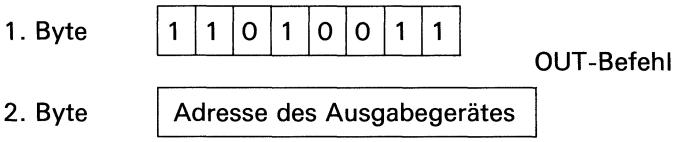
6.3.3 Ein- und Ausgabebefehle

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

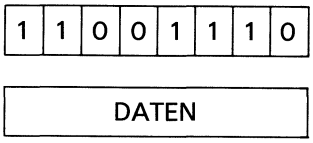
Die bisher besprochenen Additionsbefehle sind mehr oder weniger identisch mit den Additionsbefehlen des hypothetischen Rechners. Dagegen sind die beiden nachfolgenden Additionsbefehle neu.
Der Befehl ADC (Add with Carry) bewirkt, daß zusätzlich zum ADD-Befehl noch der Inhalt des C-Flags in den Akku addiert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

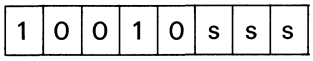
Der Befehl ACI (Add with Carry Immediate) ist ähnlich. Er addiert den Inhalt des zweiten Bytes und den Inhalt des C-Flags in den Akku. Er hat das Format:



ACI

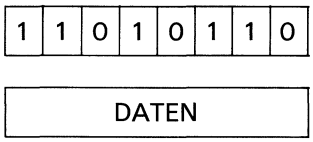
Exp. 17

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:



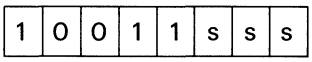
SUB

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:



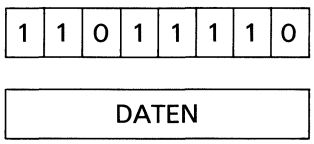
SUI

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

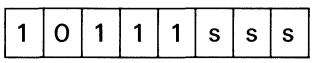
Der Befehl SBI (Subtract with Borrow Immediate) subtrahiert den Inhalt des zweiten Bytes sowie den Inhalt des C-Flags vom Inhalt des Akkumulators. Er hat das Format:



SBI

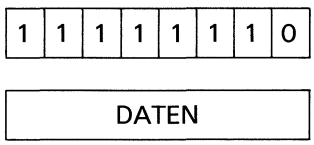
Exp. 18

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.

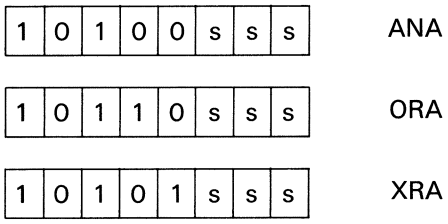
Der DAA-Befehl bewirkt entsprechend den vorgenannten Kriterien, daß eine Korrektur automatisch durchgeführt wird. Dadurch ist es möglich, im zugelassenen BCD-Bereich BCD-

Arithmetik durchzuführen. In einem Programm zur Addition von BCD-Zahlen ist hierfür nach dem normalen Additionsbefehl ein DAA-Befehl zu setzen.
Der DAA-Befehl wird in der Praxis wenig benutzt, weil er nicht ohne weiteres für Subtraktionen angewendet werden kann. In den meisten Fällen wird daher die Binärarithmetik vorgezogen.

Exp. 21

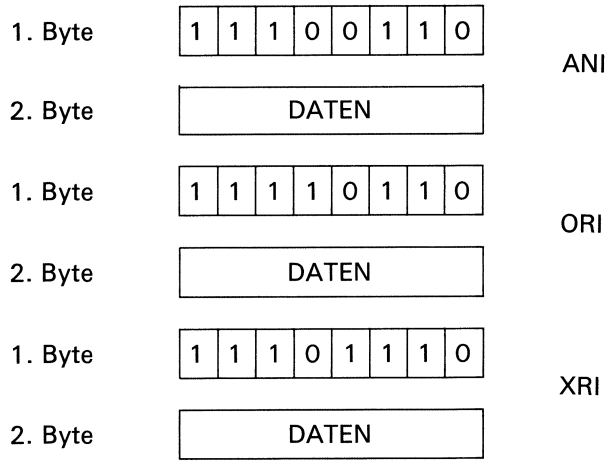
6.3.7 Logische Befehle

Der MP hat 3 logische Befehle (UND, ODER und EXCLUSIV-ODER). Die Verknüpfungen werden zwischen dem Akkumulator und einem mit s s s spezifizierten Register durchgeführt, wobei das Ergebnis immer in den Akkumulator zurückgeschrieben wird. Damit ergeben sich drei 1-Byte-Befehle folgender Formate:



ANA = AND Accumulator = (A) $\wedge$ (s s s)→A
ORA = OR Accumulator = (A) V (s s s)→A
XRA = EXCLUSIV-ODER Accumulator = (A) $\veebar$ (s s s)→A

Wird als s s s der Code 1 1 0 (Memory) gewählt, so erfolgt die logische Verknüpfung zwischen dem Inhalt des Akkumulators und dem Inhalt eines Speicherplatzes, dessen Adresse im Registerpaar HL steht.
Die gleichen Verknüpfungen können auch immediate durchgeführt werden. Auch hier wird das Ergebnis in den Akkumulator zurückgeschrieben. Diese Befehle haben folgende Formate:



ANI = AND Immediate = (A) $\wedge$ (2. Byte)→A
ORI = OR Immediate = (A) V (2. Byte)→A
XRI = EXCLUSIV-OR Immediate = (A) $\veebar$ (2. Byte)→A

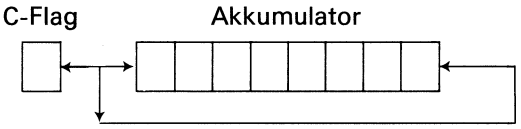
Das Verhalten der Flags bei diesen 6 Befehlen wird anhand von Experiment 22 gezeigt.

Exp. 22

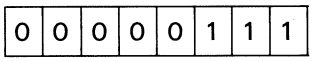
6.3.8 Rotier- und Verschiebebefehle

Zu dieser Gruppe gehören 5 Befehle, die alle das C-Flag, jedoch keines der anderen Flags beeinflussen. Die einzelnen Befehle werden nachfolgend behandelt.

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

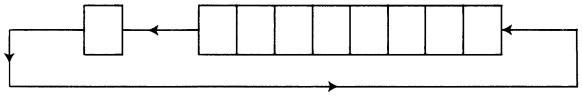


Das links stehende bit des Akkumulators wird in das C-Flag **und** gleichzeitig in das rechte bit des Akkumulators geschoben. Dieser Befehl hat folgendes Format:

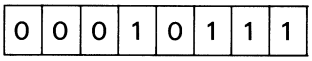


RLC

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

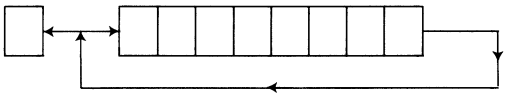


Hier wird das links stehende bit ebenfalls in das C-Flag geschoben. In das rechte bit des Akkumulators wird jedoch das C-Flag geschoben, so daß hier insgesamt 9 bit verschoben werden. Das Format dieses Befehles ist:

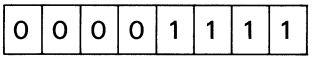


RAL

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

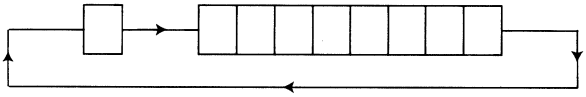


Die Verschiebung ist bei diesem Befehl entgegengesetzt zum RLC-Befehl. Sein Format ist:

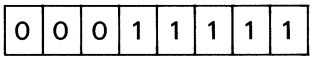


RRC

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

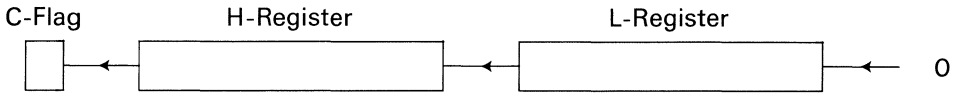


Format:



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.

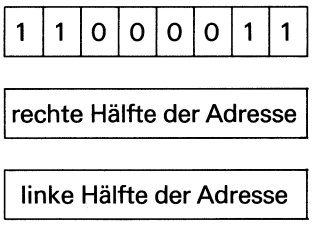
Exp. 24

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

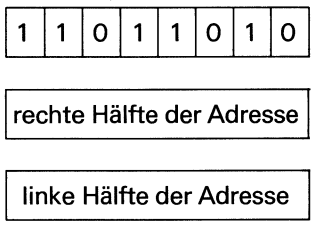
Diesen Befehl haben wir in den vorhergehenden Programmen schon mehrfach benutzt. Es handelt sich um einen unbedingten Sprungbefehl mit folgendem Format:



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.

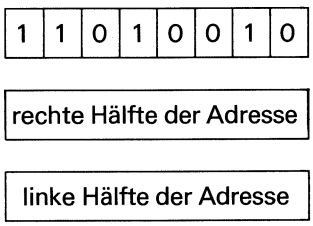
JC-Befehl (Jump if Carry)

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



JNC-Befehl (Jump if No Carry)

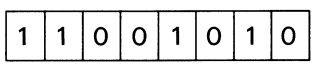
Dieser Befehl löst dann einen Sprung aus, wenn das C-Flag **nicht** 1 sondern 0 ist. Er hat das Format:



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

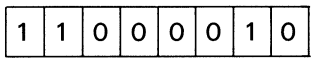
JZ-Befehl (Jump if Zero)

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

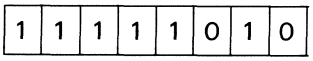


JNZ-Befehl (Jump if Not Zero)

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

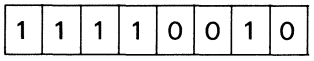


JM-Befehl (Jump if Minus)
Ein Sprung wird ausgelöst, wenn das N-Flag gleich 1 ist.



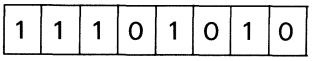
JM

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



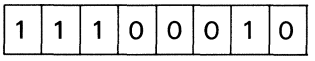
JP

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



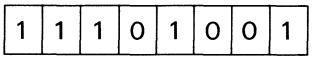
JPE

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:



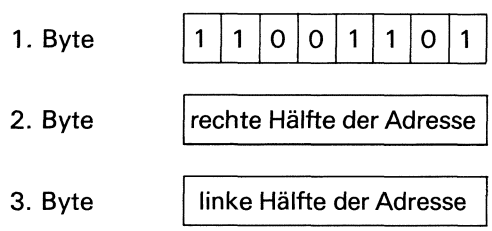
PCHL

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

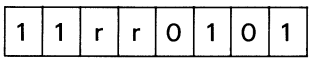
RST 0-Befehl: Adresse $0_{10} \triangleq 0\ 0\ 0\ 0_{16}$
RST 1-Befehl: Adresse $8_{10} \triangleq 0\ 0\ 0\ 8_{16}$
RST 2-Befehl: Adresse $16_{10} \triangleq 0\ 0\ 1\ 0_{16}$
RST 3-Befehl: Adresse $24_{10} \triangleq 0\ 0\ 1\ 8_{16}$
RST 4-Befehl: Adresse $32_{10} \triangleq 0\ 0\ 2\ 0_{16}$
RST 5-Befehl: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

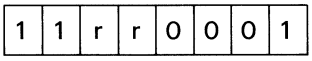


PUSH

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:



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
.
.
```

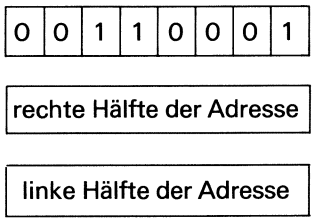
.
.
.

```
POP PSW  }
POP H    }  Befehlsfolge für das
POP D    }  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

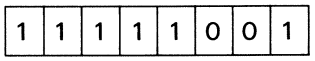
Soll beispielsweise der Stack-Bereich bei der Adresse 0 4 A 0 beginnen, so muß mit dem Befehl LXI SP der Stack-Pointer 0 4 A 1 gesetzt werden. Ein im Programm nachfolgender PUSH D-Befehl würde dann den Inhalt des D-Registers nach Adresse 0 4 A 0 und den Inhalt des E-Registers in Adresse 0 4 9 F bringen.

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:

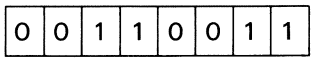


SPHL

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

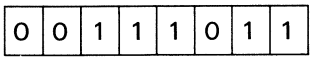
INX SP und
DCX SP

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.

Schlußbemerkungen zu Lehrheft 4

Im Kapitel 6.3. wurde der Befehlsvorrat des MP 8080 behandelt. Die Experimente 14 bis 31 sollten die Anwendung der einzelnen Befehle zeigen und ihre Funktion vertiefen.

Experiment 32 und die folgenden sind Programmbeispiele, die den Umgang mit dem MP 8080 und dessen gesamten Befehlsvorrat zeigen sollen. Diese Beispiele wurden willkürlich gewählt und stellen auch keine optimalen Lösungen für die einzelnen Aufgaben dar.

Um Ihr Wissen noch weiter zu vertiefen, empfehlen wir Ihnen, eigene Programme zu entwickeln bzw. die Programme aus den Experimenten zu modifizieren. Eine gewisse Routine in der Programmerstellung läßt sich nur durch eigenes Üben erreichen.

Exp. 32

bis

Exp. 37

Hinweis:

Am Schluß dieses Lehrheftes finden Sie auf 2 abtrennbaren Blättern nochmals den gesamten Befehlsvorrat des MP 8080. Die Angabe des Hex.-Codes, der Byte- und Zykluszahlen soll die Programmerstellung erleichtern. Die einzelnen Befehlsgruppen lehnen sich an die Reihenfolge der Besprechung im Lehrheft an. Einige Befehle sind mehrfach vorhanden, da sie sich nicht eindeutig einer Gruppe zuordnen lassen.