ITT FACHLEHRGÄNGE

– Elektronik-Seminare –

"Einführung in die Mikroprozessortechnik"

Teil II

ITT Fachlehrgänge
753 Pforzheim, Postfach 1570

Sehr geehrter Lehrgangsteilnehmer!

Wir möchten Sie darauf aufmerksam machen, daß diese als Manuskript gedruckten Arbeitsunterlagen nur für Ihren persönlichen Gebrauch bestimmt sind. Eine Weitergabe oder Vervielfältigung bedarf daher unserer schriftlichen Zustimmung.

Wir wünschen Ihnen einen angenehmen Aufenthalt und hoffen auf eine gute Zusammenarbeit während des Seminars.

Ihre ITT Fachlehrgänge

(Westerholt) (Imhoff)

ITT Fachlehrgänge

Einführung in die Mikroprozessortechnik (Teil 2)

Der Hypothetische Mikrorechner

MP-System 8o8o

ITT Fachlehrgänge

Hypothetischer Mikrorechner (System 5)

Der "Hypothetische Mikrorechner" wurde aus folgenden Gründen entwickelt und in den MP-Experimenter übernommen:

- es sollte ein Lernrechner geschaffen werden, der nicht auf bestimmte käufliche Prozessortypen zugeschnitten ist.

- möglichst viele der gängigen Befehls- und Adressierungsarten sollten vertreten sein.

- Befehlsstrukturen und -formate sollten aus didaktischen Gründen einfacher und klarer gegliedert sein als bei " echten " Rechnern.

- der Zugriff auf alle bei einem Programmlauf interessanten Vorgänge und Daten sollte möglich sein.

- Hilfsfunktionen zur Fehlersuche und -korrektur sollten im System vorhanden sein.

- die Hardwarestruktur sollte ebenfalls Beispielhaft für verschiedene Rechnertypen sein.

1. Wiederholung zum System 5

Die im ersten Seminar behandelten Befehle und Adressierungsarten des hypothetischen Rechners sowie dessen Hardwarestruktur sollen zunächst als Wiederholung kurz zusammengefasst werden.

ITT Fachlehrgänge

Hardwarestruktur:

8 bit Datenwortlänge(Datenbus)	ALU
8 bit Adresswortlänge(Adressbus)	Befehlszähler
4 Register(Akkumulatoren)	Hilfsregister
3 Flags(Zero, Negativ, Carry)	

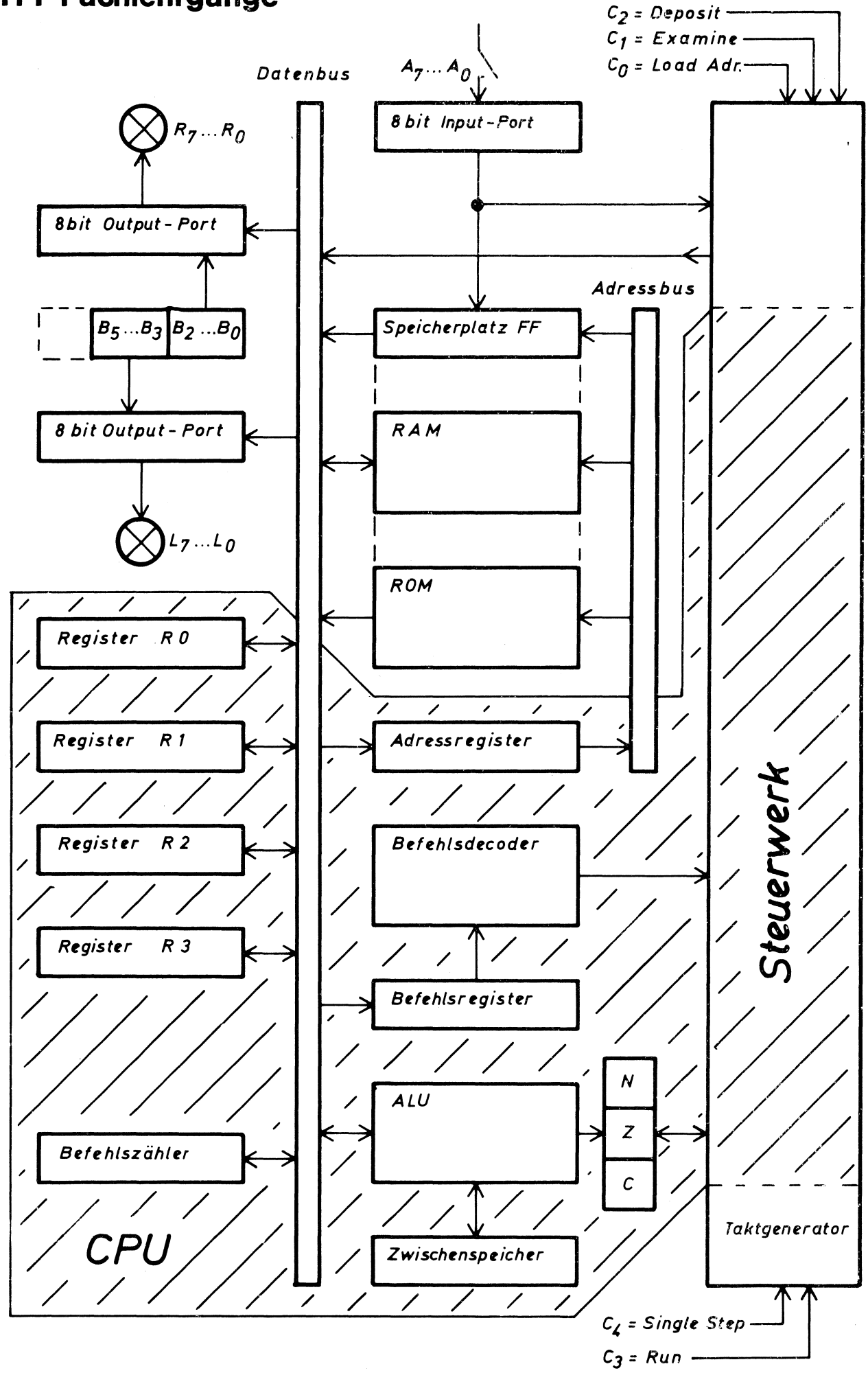
25 byte	ROM-Bereich	(00...18)
230 "	RAM- "	(19...FE)
1 "	Geräteadresse für die A-Schalter	(FF)

Befehlsvorrat und Adressmodes:

Befehl			Adressmode
MOVE	Datentransfer	Register → Register	Register
LOAD		Speicher → Register	# , direkt
STAC		Register → Speicher	direkt
ADD, SUB, IOR, XOR, AND	Arithmetik/Logik	Register - Register	Register
		Register - Speicher	# , direkt
INCR, DECR, RACL, RACR	Registerbefehle		Register
JUMP	Sprungbefehle		direkt

(Die nachfolgende Blockschaltung sowie die Befehlstafeln waren bereits in den Unterlagen zu Seminar I enthalten.)

ITT Fachlehrgänge



ITT Fachlehrgänge *Befehlstafeln System 5 Blatt 1*

Datentransfer Register ⟶ Register (1)

Move to register from register

*MOVE R_,R_*

Destin. ↑

Source ↑

DESTINATION \ SOURCE	R 0	R 1	R 2	R 3
R 0	HALT	04	08	0C
R 1	01	05	09	0D
R 2	02	06	0A	0E
R 3	03	07	0B	NOP

Datentransfer Speicher ⟶ Register (2)

Load to register from memory

LOAD R_, #data

" R_, adr.

" R_, @ R2

R_, @ R3 ↑

Destin. ↑

DESTINATION \	#	dir.	@ R2	@ R3 ↑	„INPUT" (A-Sch.)
R 0	80 data	84 adr.	88	8C	84 FF
R 1	81 data	85 adr.	89	8D	85 FF
R 2	82 data	86 adr.	8A	8E	86 FF
R 3	83 data	87 adr.	8B	8F	87 FF

Datentransfer Register ⟶ Speicher (3)

Store to memory from register (accu)

*STAC @↓R3,R_*

*" adr. ,R_*

*" @ R2 ,R_*

*" @ R3↑,R_*

Source ↑

SOURCE \	R 0	R 1	R 2	R3
@ ↓ R3	70	71	72	73
direkt	74 adr.	75 adr.	76 adr.	77 adr.
@ R2	78	79	7A	7B
@ R3 ↑	7C	7D	7E	7F

ITT Fachlehrgänge *Befehlstafeln System 5 Blatt 2*

Arithmetik / Logik Register - Register (4)

ADDR = 1 _
SUBR = 2 _
IORR = 3 _
XORR = 4 _
ANDR = 5 _

DESTINATION \ SOURCE	R0	R1	R2	R3
R0	0	4	8	C
R1	1	5	9	D
R2	2	6	A	E
R3	3	7	B	F

Arithmetik / Logik Register - Speicher (5)

ADDM = 9 _
SUBM = A _
IORM = B _
XORM = C _
ANDM = D _

DESTINATION \	# (2. Byte)	direkt ((2. Byte))	@R2	@R3↑
R0	0	4	8	C
R1	1	5	9	D
R2	2	6	A	E
R3	3	7	B	F

(6)

	SOURCE R0	R1	R2	R3
INCR	60	61	62	63
DECR	64	65	66	67
RACL	68	69	6A	6B
RACR	6C	6D	6E	6F

(7)

JUMP / CALL		Adressmodes dir.	@	@R2	@R3↑
unbed.		E0 / F0	E4 / F4	E8 / F8	EC /
Bedingung	Z	E1 / F1	E5 / F5	E9 / F9	ED /
	N	E2 / F2	E6 / F6	EA / FA	EE /
	C	E3 / F3	E7 / F7	EB / FB	EF /

2.) Weitere Adressierungsarten im System 5

Als weitere Adressierungsarten stehen im System 5 die folgenden 3 Möglichkeiten zur Verfügung:

a) @ R2 (indirekt über R2)

b) @ R3↑(" " R3 mit anschließendem increment in R3)

c) @↓R3 (" " R3 " vorhergehendem decrement in R3)

Für diese sogenannte " Register-Indirekte " Adressierung können nur die Register R2 und R3 verwendet werden.

Das Symbol @ wird zweckmäßigerweise als "indirekt über" gelesen.
Für Datentransfer- und Arithmetik/Logik-Befehle ergibt sich bei Verwendung der Register-Indirekten Adressierung folgendes Verhalten:

Die Adresse der Daten ist Inhalt eines Registers!

(Zur Erinnerung: Register-Adressierung bedeutete:
Die Daten sind Inhalt eines Registers!)

Beispiel: (R2) = 76 (R0) = 20

Adr.	Inhalt
48	ADDM R0,@R2
.	
.	
.	
76	24

Unter den gegebenen Voraussetzungen wird in diesem Beispiel der Inhalt der Adresse 76 zum Inhalt des Registers R0 addiert. Das Ergebnis in R0 muß 44 lauten.

Weshalb das Erhöhen bzw. Erniedrigen des Registers R3 nach bzw. vor der Ausführung des Befehls durchgeführt wird, soll im Zusammenhang mit dem Aufbau von Stacks (Stapelspeichern) erläutert werden.

Für die Sprung-Befehle ergibt sich folgendes Verhalten:

direkt	: Zieladresse im 2.Byte
indirekt	: Adresse der Zieladresse im 2.Byte
indirekt über R2	: Adresse der Zieladresse in R2
indirekt über R3 ↑	: Adresse der Zieladresse in R3 nach dem Sprung wird R3 incrementiert

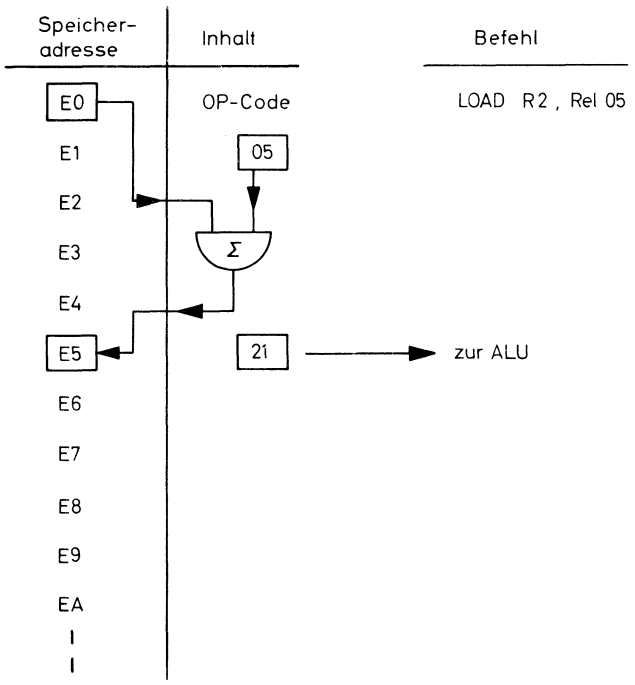
In den nachfolgenden Kapiteln werden einige nicht im System verwendete Adressierungsarten beschrieben, die jedoch in verschiedenen Prozessoren vertreten sind.

3. Relative Adressierung

Bei dieser Adressierungsart enthält der Adreßteil des Befehles nur den Unterschied zwischen der gewünschten Adresse und der Adresse des Befehles selbst (Bild 4.9.1.1).
Der Ablauf ist folgender: Wenn der Befehlszähler die Adresse E 0 spezifiziert, wird über den OP-Code der Befehl LOAD R2, Rel 0 5 ausgelöst. Entscheidend ist hierbei, daß der Inhalt des zweiten Bytes, hier 0 5, zur eigentlichen Befehlsadresse, hier E 0, addiert werden muß (symbolisiert durch das Summenzeichen Σ), um die gewünschte Adresse zu erhalten. Die Summe E 0 + 0 5 = E 5 gibt die Adresse an, deren Inhalt (im Beispiel 2 1) in das Register R2 geladen werden soll. Der Inhalt des zweien Bytes ist also die Differenz zwischen der gewünschten Adresse und der eigentlichen Programmadresse. Diese Differenz wird auch als **Offset** oder **Displacement** bezeichnet. Wenn, wie in der Praxis häufig üblich, die Daten für ein bestimmtes Programmstück in dessen Nähe gespeichert sind, ist diese Differenz relativ klein und benötigt somit wenig Speicherplatz.
In der Befehlsschreibweise ist diese Adressierungsart durch den Vorsatz Rel gekennzeichnet. In der Praxis hätte man hier ein Sonderzeichen verwendet oder einfach LOAD R2, E 5 geschrieben. Wie später noch näher erläutert wird, ist es Aufgabe des Assemblers, eine entsprechende Umwandlung vorzunehmen.
Wenn ein Programm ausschließlich mit relativer Adressierung geschrieben wird, so kann dieses Programm innerhalb des Speicherbereiches beliebig versetzt werden, d.h., das Programm ist relocatable oder bewegbar. Im Gegensatz zur direkten oder indirekten Adressierung, bei denen bei einer Versetzung die Adressen in den Befehlen geändert werden müssen, kann bei ausschließlich relativer Adressierung das Programm ohne Änderung weiterbenutzt werden.
In Mikrorechnern findet diese Adressierungsart hauptsächlich bei Sprungbefehlen Anwendung.

ITT Fachlehrgänge

Bild 4.9.1.1
Prinzip der relativen Adressierung



4. Base-Relative-Adressierung

Diese Adressierungsart ist eine Sonderform der Indexed-Adressierung. Auch hierbei wird die Adresse aus der Summe von 2 Teilen gebildet. Diese beiden Teile sind der Offset und der Inhalt eines Indexregisters, wie z.B. R3 beim hypothetischen Mikrorechner. Das Prinzip zeigt Bild 4.9.3.1.

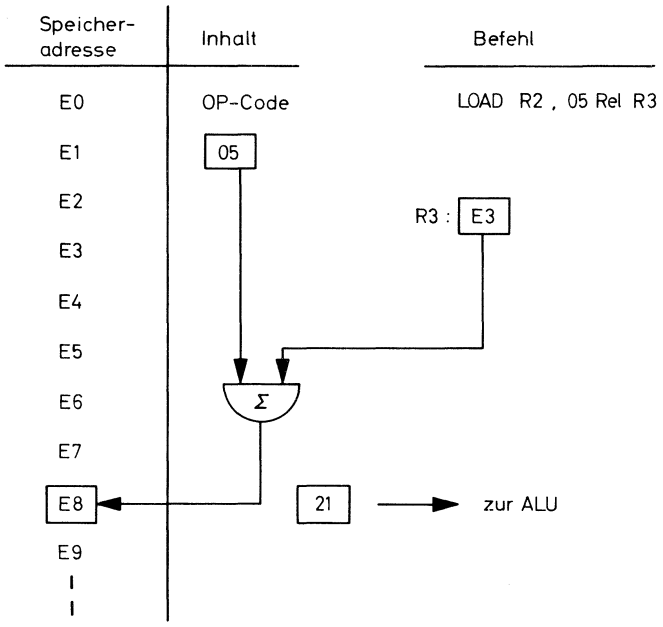


Bild 4.9.3.1
Prinzip der Base-Relative-Adressierung

Die Datenadresse E 8 mit dem Inhalt 2 1 ist hierbei die Summe aus Offset (0 5) und Inhalt des Indexregisters (E 3).
Diese Adressierungsart eignet sich besonders gut für Listenverarbeitung.

ITT Fachlehrgänge

5. Pre-Indexed-Adressierung

Diese Adressierungsart ist eine **indirekte Version** der Base-Relative-Adressierung. Der Inhalt des Indexregisters wird zum Offset addiert. Die so entstandene Summe ergibt eine Adresse, mit der die eigentliche Datenadresse spezifiziert werden kann (Bild 4.9.4.1).

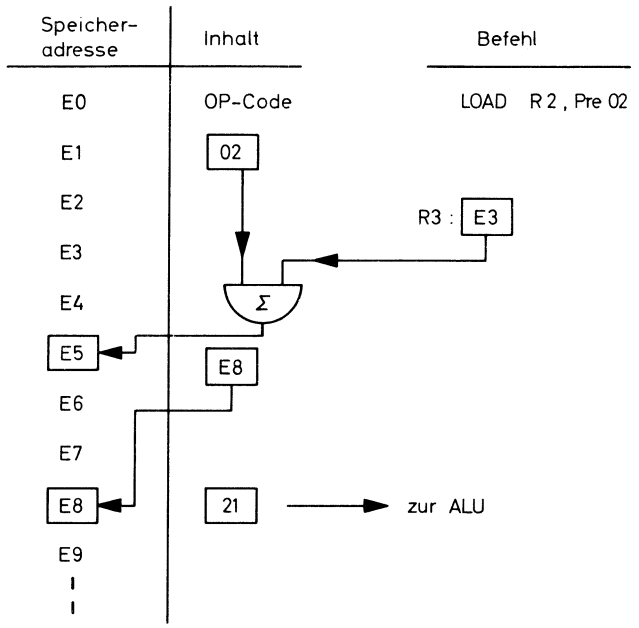


Bild 4.9.1
Prinzip der Pre-Indexed-Adressierung

Zunächst wird aus dem Offset des Befehlswortes (0 2) und dem Inhalt des Indexregisters (E 3) die Summe gebildet (E 5). Diese Summe wird quasi als Zwischenadresse benutzt. Der Inhalt der Adresse E 5 (E 8) ist die eigentliche Datenadresse. Die Daten der Adresse E 8 (2 1) werden über die ALU in das Register R2 geladen.
Die Adressierungsart erscheint zwar etwas kompliziert, hat aber Vorteile beim Arbeiten mit Unterprogrammen. Hierauf werden wir in einem späteren Abschnitt noch näher eingehen.

6. Post-Indexed-Adressierung

Diese Adressierungsart ähnelt sehr der Pre-Indexed-Adressierung. Der Unterschied besteht darin, daß die Summe früher gebildet wird (Bild 4.9.5.1).

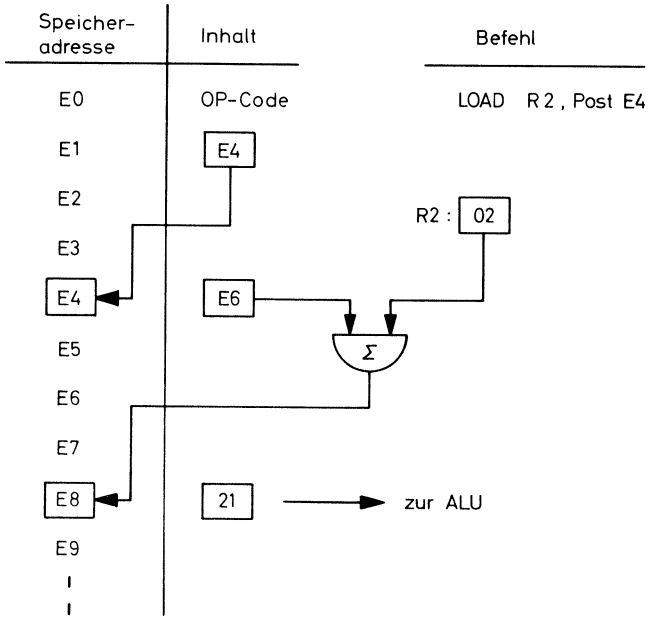


Bild 4.9.5.1
Prinzip der Post-Indexed-Adressierung

Die Anwendung dieser Adressierungsart ist nicht sehr häufig.

Die hier vorgestellten Adressierungsarten sind Grundformen, die durch Abwandlungen und Kombinationen noch einige andere Möglichkeiten erlauben. Weitere Adressierungsarten, die für das praktische Arbeiten von Bedeutung sind, werden im System 6 im Zusammenhang mit entsprechenden Experimenten behandelt.

7.) Stapelspeicher (Stack)

Das Stack (teilweise auch als Push-Down-Stack bezeichnet) ist eine Speicherart, bei der die Daten wie auf einem Stapel abgelegt werden. Das zuletzt eingeschriebene Wort kann (und muß) als erstes wieder gelesen werden. Von diesem Verhalten leitet sich auch die Bezeichnung LIFO (last in first out) ab.

Als"Hardwarestack" könnte man sich ein Schieberegister vorstellen, bei dem serielle Ein- und Ausgänge am gleichen Ende liegen. Werden alle Einschreibvorgänge z.B. rechtsschiebend durchgeführt, müsste in diesem Fall das Lesen des Stacks linksschiebend erfolgen. Die Reihenfolge der Daten wird dadurch natürlich umgekehrt. Ein "Hardwarestack" ist durch die fest vorgegebene Anzahl der Speicherplätze in seiner Aufnahmefähigkeit begrenzt. Festverdrahtete Stacks werden daher nur eingesetzt, wenn nicht zu große Datenmengen zu speichern sind.

Ein "Softwarestack" ist in seiner Kapazität variabel und kann der jeweiligen Aufgabenstellung angepasst werden. Der Anwender eines Rechners kann durch festlegen der Adressen einen beliebigen RAM-Bereich für ein Stack reservieren.

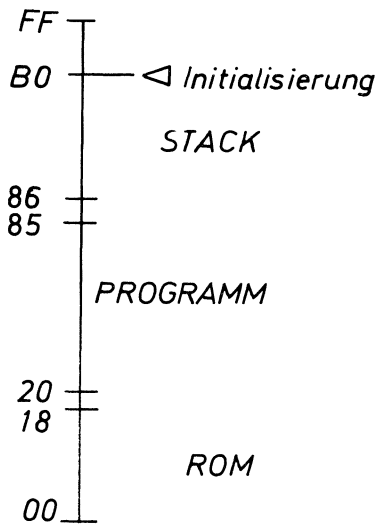
Die Vorstellung von einem Schieberegister stimmt hier jedoch nichtmehr. Anstelle der im Schieberegister be-

wegten Daten wird hier lediglich ein "Zeiger", der sogenannte Stackpointer, bewegt. Im System 5 arbeitet das Register R3 als Stackpointer. Die Daten verbleiben auf den ihnen zugewiesenen Stackplätzen.

Zum Betrieb eines Stack müssen Autoincrement- und Autodecrementadressierung verwendet werden. Der Übliche Ablauf vollzieht sich folgendermaßen:

a) Initialisierung des Stackpointers

Die Initialisierung des Stackpointers sollte sinnvoller Weise gleich zu Beginn eines Programms erfolgen. Damit wird festgelegt, in welchem Speicherbereich das Stack gebaut werden darf.



Im nebenstehenden Beispiel wird der Stackpointer (R3) auf B0 initialisiert. Somit stehen die Speicherplätze bis zur Adresse 86 für das Stack zur Verfügung.

Die Initialisierung kann z.B. mit dem Befehl LOAD R3,# B0 durchgeführt werden.

b) Daten auf das Stack bringen

Im vorhin angenommenen Beispiel soll Register R2 zu einem beliebigen Zeitpunkt den Inhalt 71 haben. Wenn jetzt zum ersten Mal der Inhalt von R2 auf das Stack gebracht werden soll, so löst der Befehl

STAC @↓ R3,R2 folgende Vorgänge aus:

- der Stackpointer wird von B0 auf AF decrementiert

- der Inhalt von R2 wird in der Adresse AF gespeichert, d.h. in der Adresse (SP)-1

c) Daten vom Stack holen

Am vorhergehenden Ablauf ist zu erkennen, daß das Rückholen von Daten zuerst abgeschlossen sein muß, bevor anschließend der Stackpointer wieder incrementiert wird. Der Befehl

LOAD R2, @R3↑

stellt in folgenden Schritten den ursprünglichen Zustand wieder her:

- der Inhalt der im Stackpointer stehenden Adresse wird nach R2 gebracht

- der Stackpointer wird incrementiert

8.) Unterprogramme - Subroutines

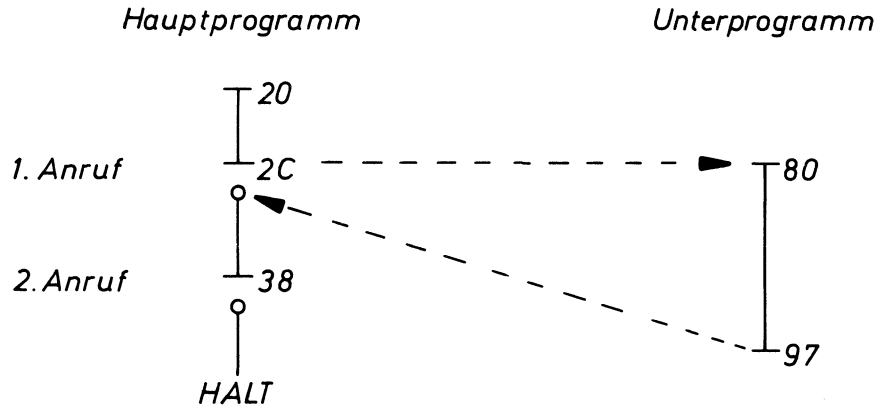
Als Unterprogramme werden Programmstücke bezeichnet, die

- Teilaufgaben eines Programmes erfüllen
- nur beim Auftreten bestimmter Bedingungen bearbeitet werden
- mehrfach innerhalb eines Hauptprogrammes verwendet werden

Das Arbeiten mit Unterprogrammen erleichtert die Programmgestaltung wesentlich. Lösungen von Detailaufgaben müssen nicht in das relativ grobe Ablaufschema eines vorläufigen Flußdiagramms mit eingebaut werden, sondern können zunächst als "black box" erscheinen. Dadurch wird auch die Übersicht über kompliziertere Programme erleichtert.

Die Problemaik beim Einsatz von Unterprogrammen liegt, wenn nur JUMP-Befehle verwendet werden könnten, im Auffinden der Rücksprungziele(Rücksprungadressen)´im Hauptprogramm.

Im nachfolgenden Beispiel wird das Unterprogramm mit der Anfangsadresse 80 zweimal aufgerufen. Die Rücksprungziele beider Anrufe müssen verschieden sein. Um automatisch an die richtige Stelle im Hauptprogramm zurückkehren zu können, werden Unterprogramme mit CALL-Befehlen angerufen. (CALL = rufen)



ITT Fachlehrgänge

Ein CALL-Befehl löst folgende Wirkungen aus:

1.) das Rücksprungziel wird im Stack gespeichert.
Adressierungsart ist @↓ R3.

2.) die Anfangsadresse des Unterprogramms wird in den Befehlszähler gebracht.

zu 1.)
Mit den im System 5 möglichen Adressierungsarten sind CALL-Befehle als 1- oder 2-Byte-Befehle ausführbar. Für den CALL-Befehl, der im vorangegangenen Beispiel in Adresse 2C steht, ergeben sich somit folgende drei Möglichkeiten:

1.) direkt adressiert

Adresse	Inhalt	;Kommentar
2C	CALL	
2D	80	
2E	nächster Befehl	

2.) indirekt adressiert

Adresse	Inhalt	;Kommentar
2C	CALL @	
2D	75	
2E	nächster Befehl	

3.) registerindirekt adressiert

Adresse	Inhalt	;Kommentar
2C	CALL @ R2	
2D	nächster Befehl	

ITT Fachlehrgänge

Um am Ende eines Unterprogramms das Rücksprungziel wiederzufinden, wird mit dem Befehl

JUMP @R3↑ = Rücksprungbefehl aus Unterprogramm

die auf dem Stack abgespeicherte Rücksprungadresse in den Programmzähler gebracht.

Programmbeispiel: Register-Indirekte Adressierung

Das folgende Programm hat keinen praktischen Hintergrund. Es soll lediglich an einigen Beispielen die Wirkungsweise von register-indirekt-adressierten Befehlen gezeigt werden.

Programm:

Adresse	Inhalt	Befehl	Display-Selector-Einstellung
2o	83		
21	9E		
22	20		
23	7C		
24	7C		
25	7C		
26	7C		
27	A3		
28	o2		
29	81		
2A	55		
2B	71		
2C	oo		

Nach der Bearbeitung des Befehls in Adresse 26 sind folgende Speicherplätze gelöscht:

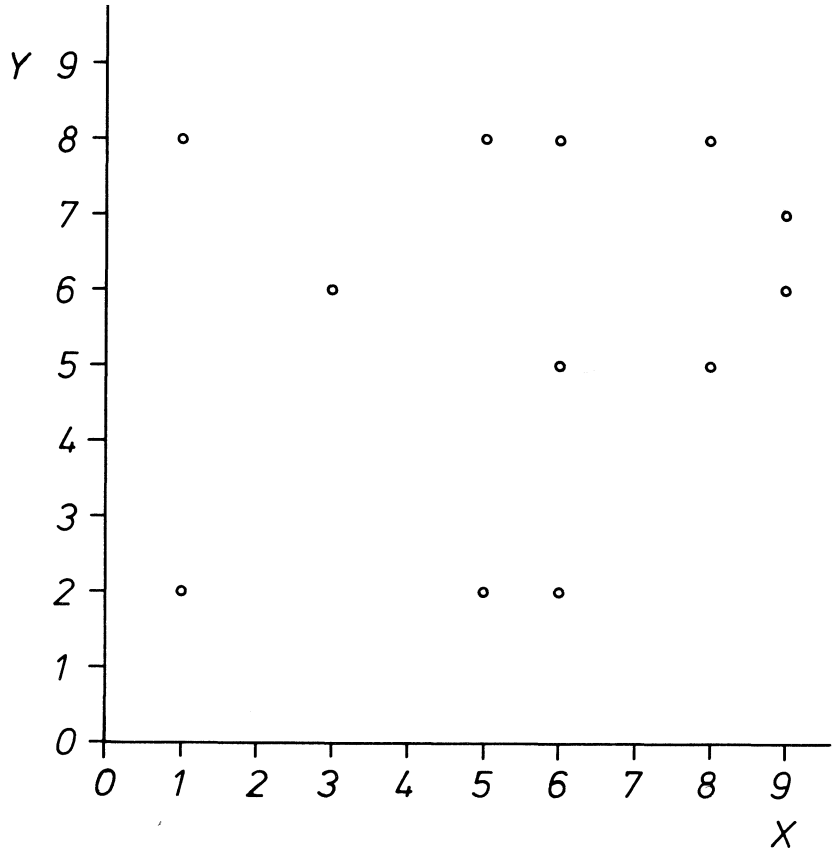
Nach dem Befehl in Adresse 27 steht der Stackpointer auf:

Nach dem Befehl in Adresse 2B erscheint der in Register R1 geladene Wert im Speicherplatz:

ITT Fachlehrgänge

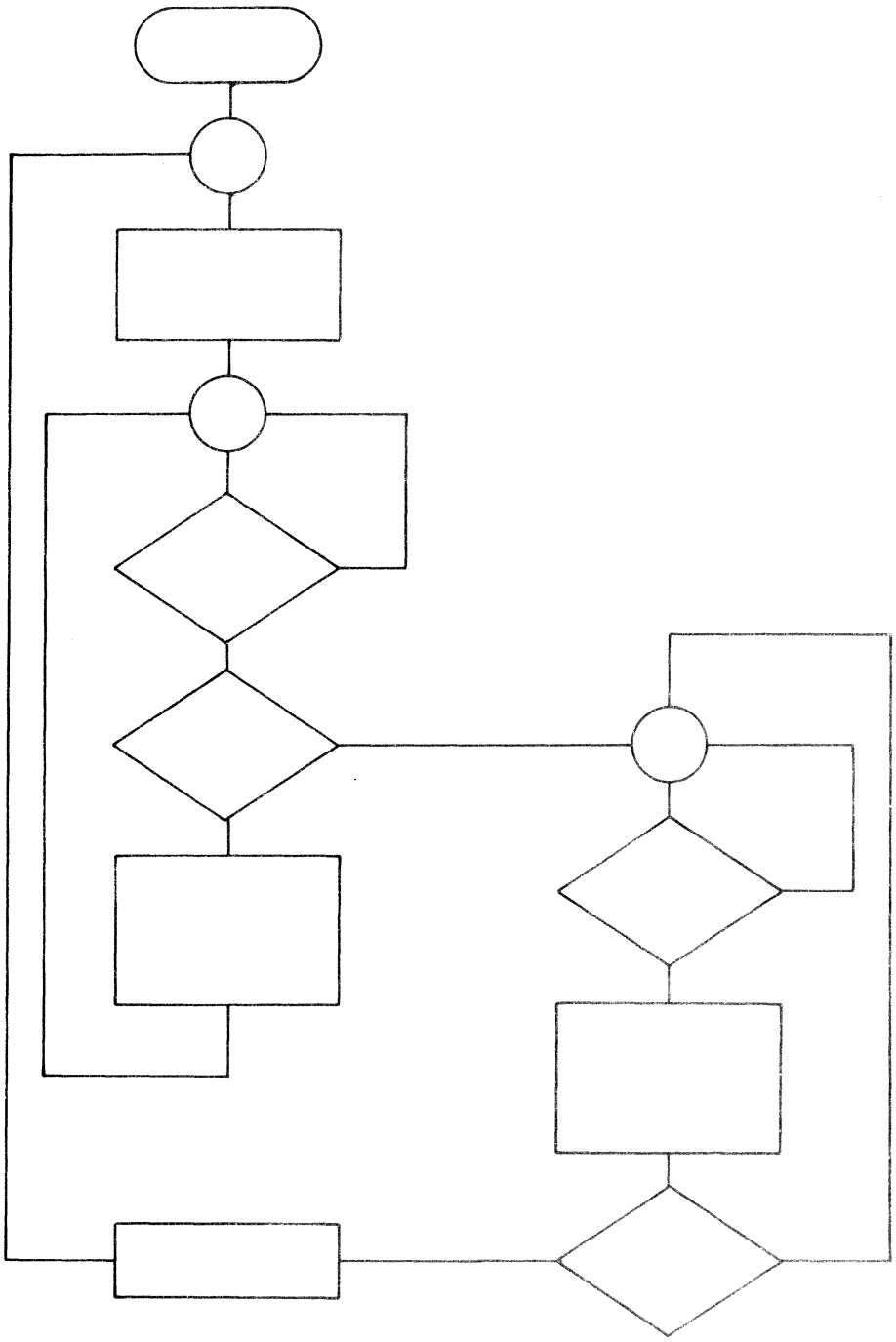
Programmbeispiel: Koordinatenspeicher

Eine Maschinensteuerung soll in einer 1ox1o-Matrix eine fast beliebige Anzahl von Koordinaten anfahren können. Die Koordinaten werden in der umgekehrten Reihenfolge der Eingabe angefahren. Die Ausgabe der gespeicherten Werte muß also ebenfalls in umgekehrter Reihenfolge stattfinden.
Das Ende der Dateneingabe wird durch EE (=Eingabe-Ende) signalisiert.
Das Ende der Datenausgabe wird durch AF (=alles fertig) signalisiert.
Zum Ein- und Ausgeben der Koordinaten muß der Examine-schalter je Koordinate einmal getaktet werden.



X	Y
6	5
8	5
9	6
9	7
8	8
6	8
6	2
5	2
5	8
3	6
1	8
1	2
A	F

ITT Fachlehrgänge



NAME: .... BLATT: ....

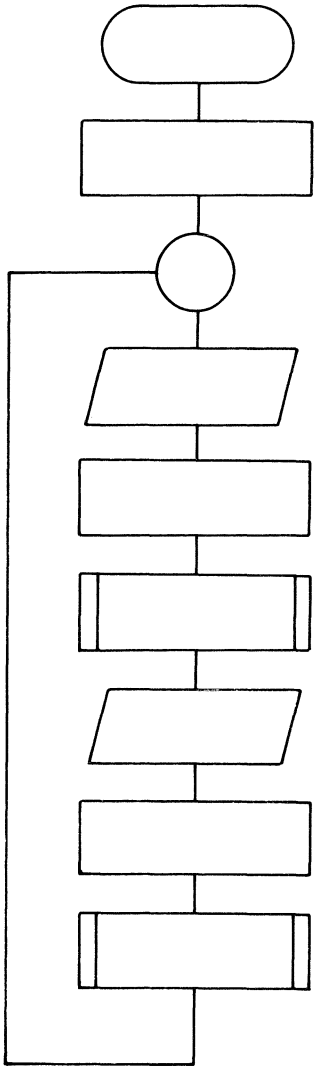
ITT Fachlehrgänge

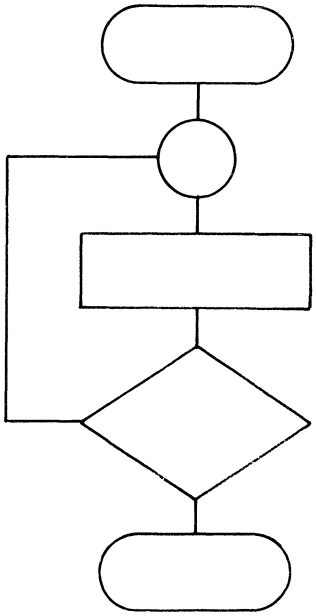
PROGRAMM: ....

LABEL	Adresse	Inhalt	Befehl	; Kommentar

Programmbeispiel: Argumentübergabe an Unterprogramme

In diesem Beispiel sollen an eine Zeitschleife zwei verschiedene Argumente übergeben werden, um so einen Multivibrator mit programmierbarem Tastverhältnis realisieren zu zu können.





NAME: .... BLATT: ....

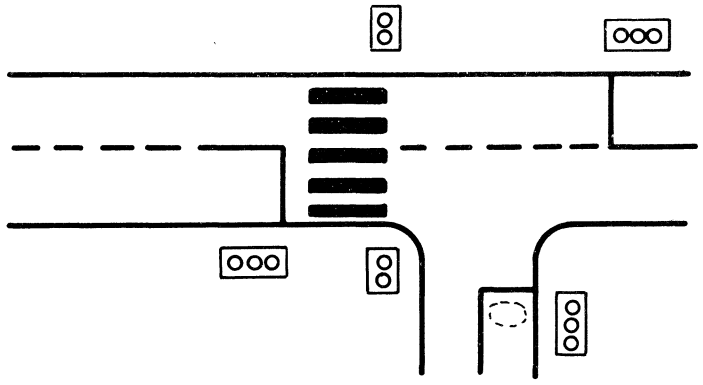
ITT Fachlehrgänge

PROGRAMM: ....

LABEL	Adresse	Inhalt	Befehl	; Kommentar

ITT Fachlehrgänge

Programmbeispiel: Ampelsteuerung



Die Ampelanlage einer Fabrikausfahrt mit Fußgängerüberweg soll durch ein Programm gesteurt werden. Als Geber, deren Signale verarbeitet werden müssen, sind Tasten an den Fußgängerampeln und eine Induktionsschleife in der Fabrikausfahrt vorhanden.

Das Programm soll folgende Bedingungen erfüllen:

1.	Grün der Hauptstraße mindestens	3o sec.
2.	Gelb " "	3 sec.
3.	Alle rot	3 sec.
4.	Rot/gelb der Ausfahrt Gleichzeitig Grün für die Fußgänger.	2 sec.
5.	Grün für Ausfahrt und Fußgänger	5 sec.
6.	Grün nur für die Ausfahrt	5 sec.
7.	Gelb " " "	3 sec.
8.	Alle rot	4 sec.
9.	Rot/gelb für die Hauptstraße	2 sec.

Diese neun Punkte stellen einen Durchlauf durch das Steuerprogramm dar.

ITT Fachlehrgänge

Ein Durchlauf darf frühestens nach dem Ende der 3o sec Grün für die Hauptstraße gestartet werden können. Gestartet wird durch:

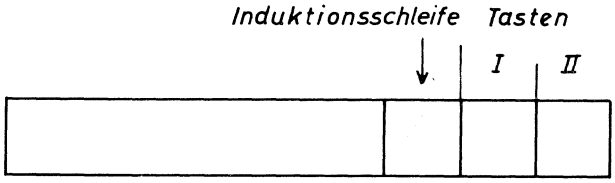
a) einen gespeicherten Tastendruck der Fußgänger, der während eines Durchlaufs erfolgte.

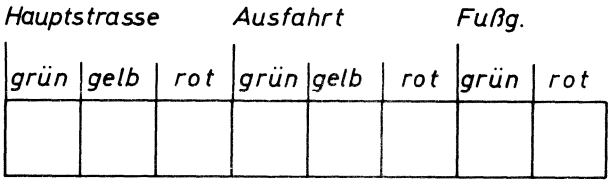
b) einen Tastendruck der Fußgänger nachEnde der 3o sec Grün für die Hauptstraße.

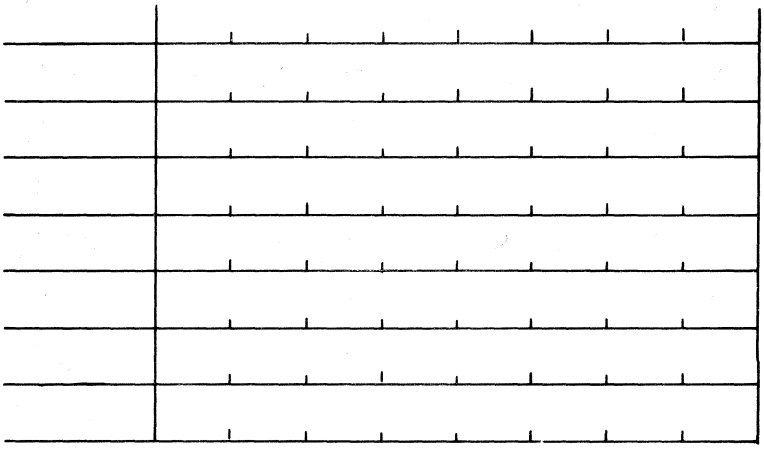
c) ein Signal beim Auffahren auf die Induktionsschleife.

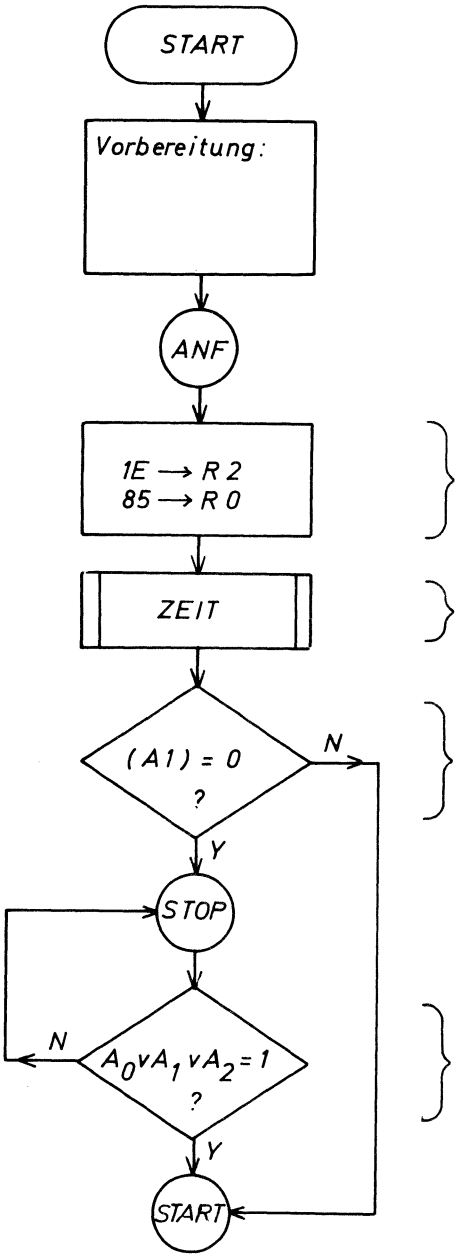
Eine weitere Forderung ist die Verlängerung der Grünphase Fabrikausfahrt von zweimal 1o sec.Die Verlängerung wird eingeleitet, wenn am Ende der Ersten oder Zweiten 1o sec Grün die Induktionsschleife noch besetzt ist. Für die Fabrikausfahrt können also maximal 3o sec Grün zustande kommen.

ITT Fachlehrgänge

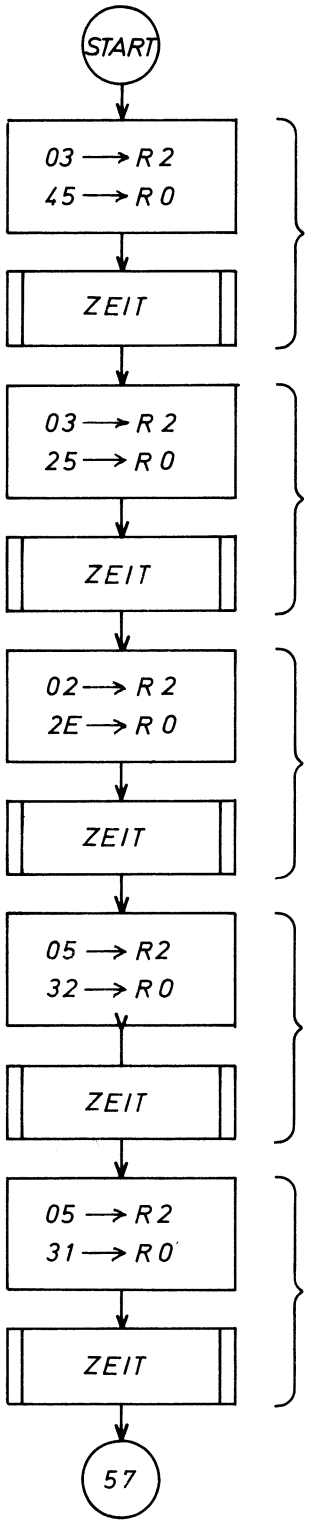




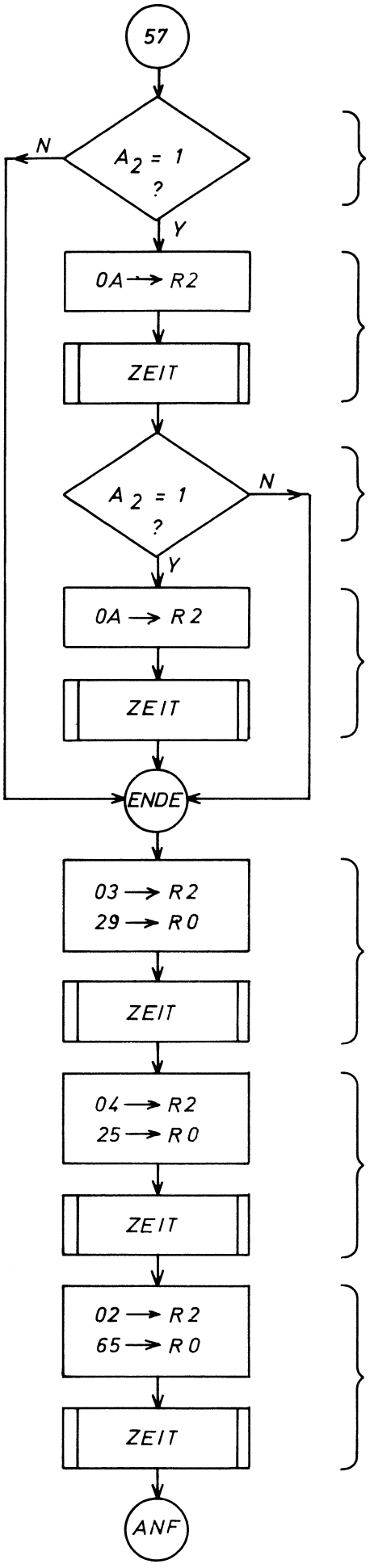


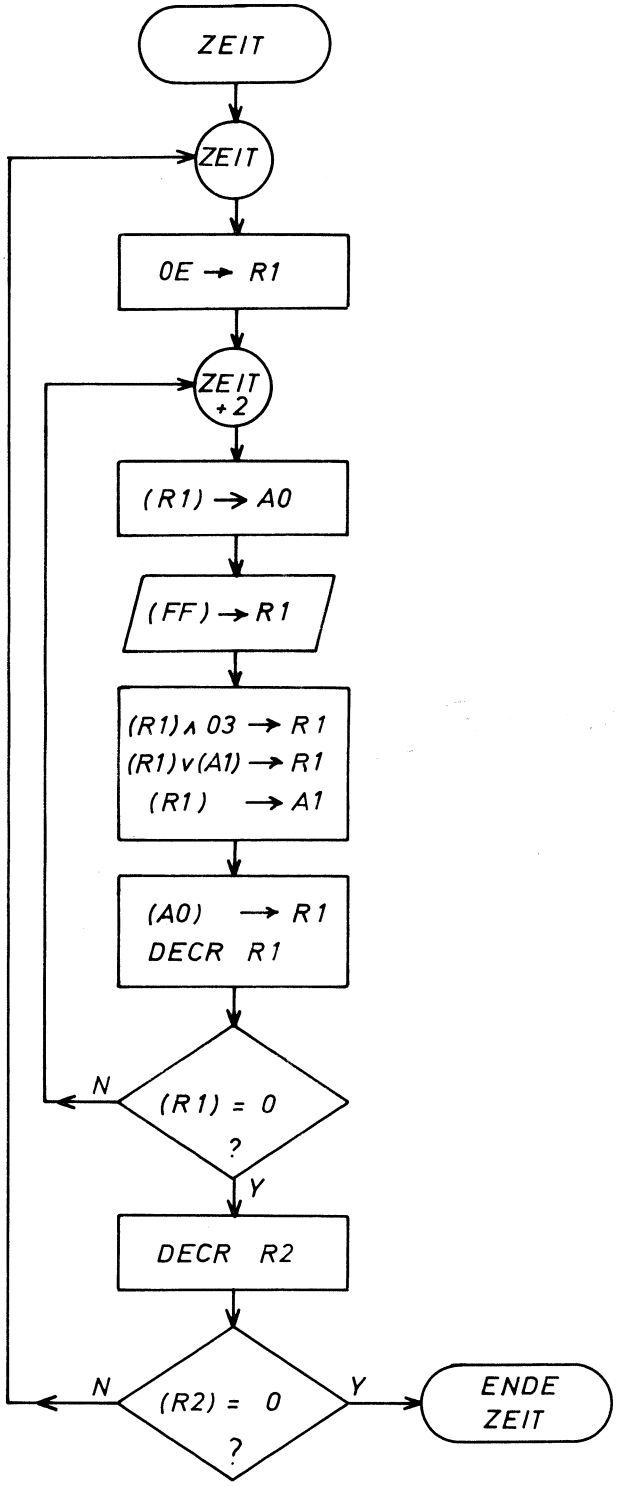


ITT Fachlehrgänge



ITT Fachlehrgänge




ZEIT
ZEIT
0E → R1
ZEIT +2
(R1) → A0
(FF) → R1
(R1) ∧ 03 → R1
(R1) v (A1) → R1
(R1) → A1
(A0) → R1
DECR R1
N
(R1) = 0 ?
Y
DECR R2
N
(R2) = 0 ?
Y
ENDE ZEIT

ITT Fachlehrgänge

NAME: ..... BLATT: 1

PROGRAMM: Ampelsteuerung

LABEL	Adresse	Inhalt	Befehl	; Kommentar
	20	83	LOAD R3,#A8	
	1	A8		
	2	2A	SUBR R2,R2	
	3	76	STAC A1,R2	
	4	A1		
ANF:	25	82	LOAD R2,#1E	
	6	1E		
	7	80	LOAD R0,#85	
	8	85		
	9	F0	CALL 80	
	A	80		
	B	85	LOAD R1,A1	
	C	A1		
	D	D1	ANDM R1,#03	
	E	03		
	F	E1	JMPZ STOP	
	30	33		
	1	E0	JUMP START	
	2	39		
STOP:	33	85	LOAD R1,FF	
	4	FF		
	5	D1	ANDM R1,#07	
	6	07		
	7	E1	JMPZ STOP	
	8	33		
START:	39	82	LOAD R2,#03	
	A	03		
	B	80	LOAD R0,#45	
	C	45		
	D	F0	CALL 80	
	E	80		
	F	82	LOAD R2,#03	
	40	03		
	1	80	LOAD R0,#25	
	2	25		
	3	F0	CALL 80	
	4	80		
	5	82	LOAD R2,#02	
	6	02		
	7	80	LOAD R0,#2E	
	8	2E		
	9	F0	CALL 80	
	A	80		

ITT Fachlehrgänge

NAME: ..... BLATT: 2

PROGRAMM: Ampelsteuerung

LABEL	Adresse	Inhalt	Befehl	; Kommentar
	4B	82	LOAD R2,#05	
	C	05		
	D	80	LOAD R0,#32	
	E	32		
	F	F0	CALL 80	
	50	80		
	1	82	LOAD R2,#05	
	2	05		
	3	80	LOAD R0,#31	
	4	31		
	5	F0	CALL 80	
	6	80		
	7	85	LOAD R1,FF	
	8	FF		
	9	D1	ANDM R1,#04	
	A	04		
	B	E1	JMPZ ENDE	
	C	6B		
	D	82	LOAD R2,#0A	
	E	0A		
	F	F0	CALL 80	
	60	80		
	1	85	LOAD R1,FF	
	2	FF		
	3	D1	ANDM R1,#04	
	4	04		
	5	E1	JMPZ ENDE	
	6	6B		
	7	82	LOAD R2,#0A	
	8	0A		
	9	F0	CALL 80	
	A	80		
ENDE:	6B	82	LOAD R2,#03	
	C	03		
	D	80	LOAD R0,#29	
	E	29		
	F	F0	CALL 80	
	70	80		
	1	82	LOAD R2,#04	
	2	04		
	3	80	LOAD R0,#25	
	4	25		
	5	F0	CALL 80	
	6	80		

ITT Fachlehrgänge

NAME: ..... BLATT: 3

PROGRAMM: Ampelsteuerung

LABEL	Adresse	Inhalt	Befehl	; Kommentar
	77	82	LOAD R2,#02	
	78	02		
	79	80	LOAD R0,#65	
	7A	65		
	7B	F0	CALL 80	
	7C	80		
	7D	E0	JUMP ANF	
	7E	25		
ZEIT:	80	81	LOAD R1,#0E	
	81	0E		
	82	75	STAC A0,R1	
	83	A0		
	84	85	LOAD R1,FF	
	85	FF		
	86	D1	ANDM R1,#03	
	87	03		
	88	B5	IORM R1,A1	
	89	A1		
	8A	75	STAC A1,R1	
	8B	A1		
	8C	85	LOAD R1,A0	
	8D	A0		
	8E	65	DECR R1	
	8F	E1	JMPZ DECR S	
	90	93		
	91	E0	JUMP ZEIT+2	
	92	82		
DECR S:	93	66	DECR R2	
	94	ED	JMPZ @R3↑	
	95	67	DECR R3	
	96	E0	JUMP ZEIT	
	97	80		
	A0			
	A1			
	A8			

MP-System 8080 (System 6)

Hardware System 6

Im System 6 wird als Beispiel eines realen Mikrorechners der Typ 8080 behandelt. Anhand des nachfolgenden Blockschaltbildes und der Gegenüberstellung sollen zunächst die Hardwareeigenschaften des Systems behandelt werden. Dabei sind die Möglichkeiten des 8080 und die im System 6 verfügbare Ausbaustufe nebeneinander aufgeführt.

MP 8080		System 6
	8 bit Datenbus	
	16 bit Adressbus	
64 k Adressen = 65 536		1 k ROM 0000 .... 03FF 1/4 k RAM 0400 .... 04FF
	8 bit I/O-Adressen	
256 x IN 256 x OUT		3 x I-Port (A- B- C-Schalter) 2 x O-Port (R- L-Lampen) isolierte Adressierung je bit ein Port, somit 8 Möglichkeiten Adresszuordnung B-Schalter = 01 A-Schalter = 02 C-Schalter 04 rechte Lampen = 01 linke Lampen = 02

Der Steuerbus des Systems wird soweit notwendig noch eingehender erklärt.

Die in der CPU enthaltenen Funktionsblöcke sind für den 8080 und das System 6 identisch, da der Anwender auf diese Konfiguration keinen Einfluß hat.

Registersatz:

6 allgemeine Register mit 8 bit
paarweise als 16 bit Register verwendbar

1 Akkumulator

Flags:

N, Z, C wie bisher

P = parity = Parität

H = half carry = Übertrag in Wortmitte
(AC = auxilliary carry)

sonstiges:

PC = program counter = Befehlszähler

SP = stack pointer

Die fünf Flags sind nichtmehr in einzelnen Flipflops gespeichert, sondern werden in einem Flag-Register zusammengefasst. Die Anordnung der Flags in diesem Register ist der folgenden Abbildung zu entnehmen.

Der Akkumulator und das Flagregister können bei bestimmten Befehlen zu einem 16 bit-Wort, dem

PSW = program status word,

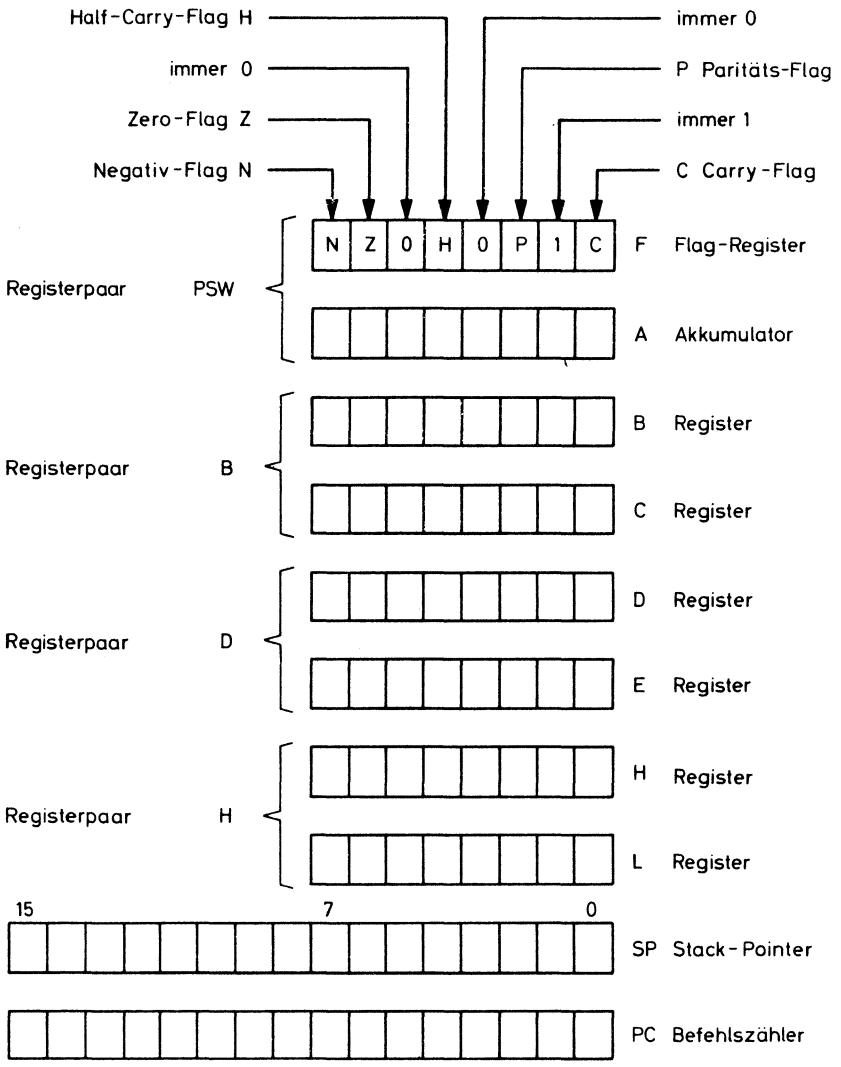
zusammengefasst werden.

ITT Fachlehrgänge

Eine zusammenfassende Darstellung der Eigenschaften der ALU des 8080 soll erst dann gegeben werden, wenn alle Befehlsarten behandelt wurden.

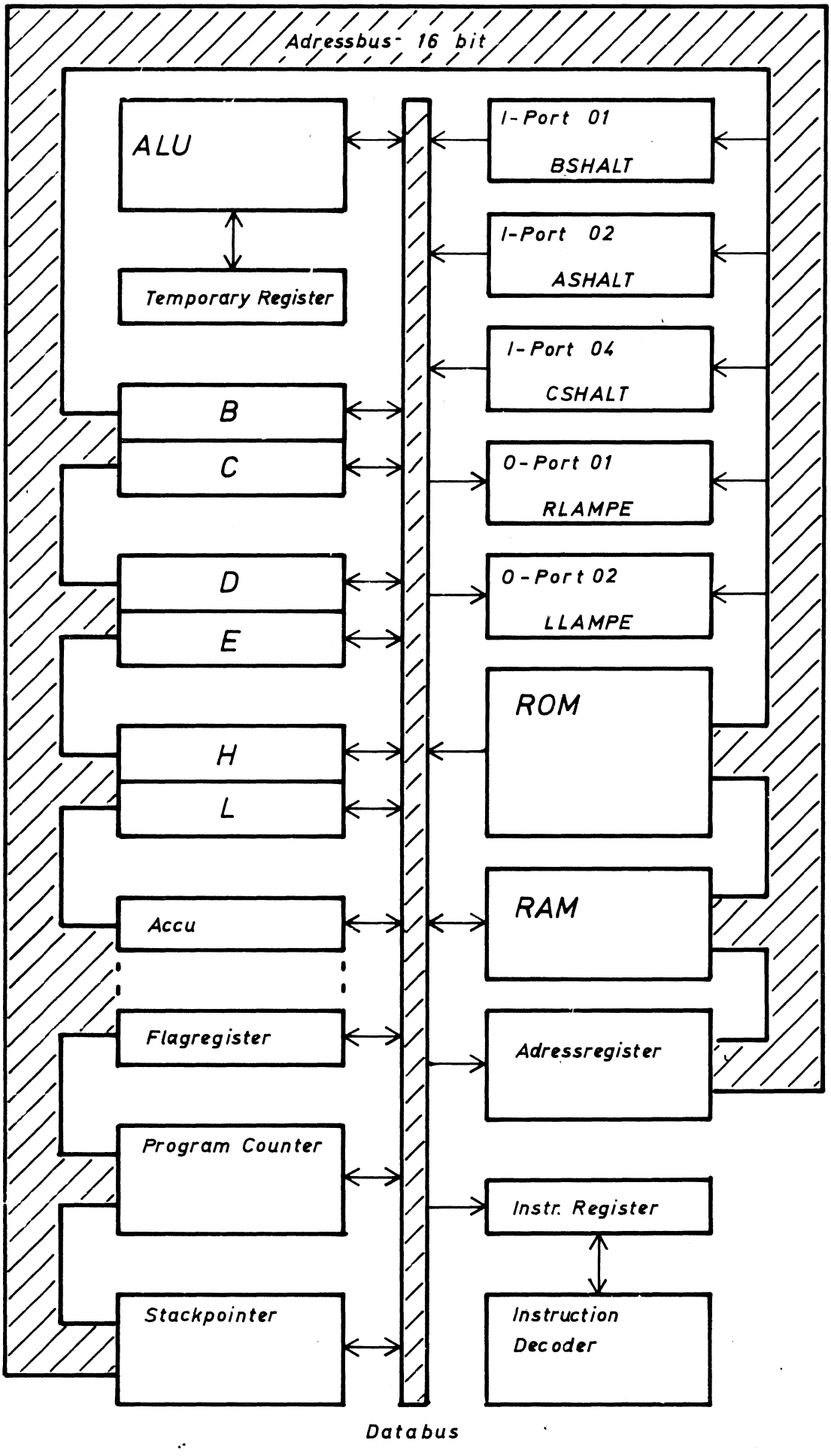
Die Blockschaltung des Systems enthält aus Platzgründen kein Steuerwerk. Hier soll der Hinweis genügen, daß

- das Steurwerk Zugriff zu allen vorhandenen Funktionsblöcken haben muß.
- das Steuerwerk den kompliziertesten Teil der Hardware darstellt.



Die Register und Flags des MP 8080

ITT Fachlehrgänge *Blockschaltung System 6*

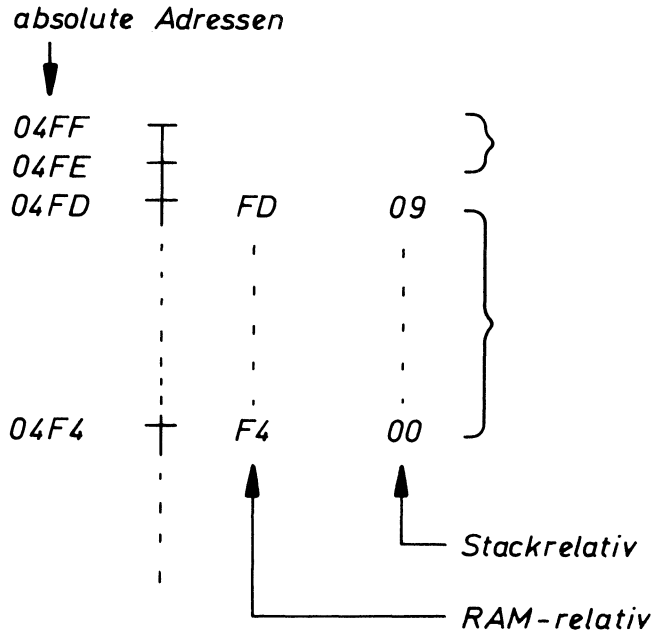


Betriebsprogramm (Monitor) des System 6

Ein Rechner kann ohne die Softwareunterstützung eines Betriebsprogrammes nicht funktionieren. In diesem Programmpaket wird die Steuerung Koordination und Überwachung von Programmabläufen durchgeführt. Außerdem sind Funktionen wie Ein/Ausgabe-Steuerung oder Datenverwaltung enthalten. Im Folgenden sollen die Monitorfunktionen im System 6 näher erläutert werden.

Display-Funktion

Um bei einem Programmlauf zu Kontrollzwecken Zugriff auf die in der CPU vorhandenen Daten zu haben, wird in festgelegten RAM-Adressen ein Display-Stack aufgebaut. Das nachfolgende Schema soll die verschiedenen möglichen Adressierungsarten für dieses Stack verdeutlichen.Die Zuordnung der Stack-Plätze zu den Registern, dem Akkumulator und den Flags ist der kleinen blauen Karte mit den Display-Codes zu entnehmen. Der Stand des echten Programmzählers wird ebenfalls in das Stack gebracht.



RUN- und SINGLE-STEP-Funktion

RUN

Beim hochschieben des RUN-Schalters startet das Programm mit der Adresse 0400. Ein vorhergehendes Load Adr. hat dabei keine Wirkung, d.h. es ist im System 6 nicht möglich Programme bei beliebigen Adressen zu starten.

SINGLE STEP bzw. Halt am Breakpoint (HLT am BP)
Im Unterschied zum System 5 können Daten im System 6 nur dann dargestellt werden,

- wenn der Rechner im Monitorprogramm läuft oder
- durch Datenausgabebefehle eine Anzeige ermöglicht wird.

Durch den Programmstart mit RUN wird die Monitorfunktion zunächst ausgeschaltet. Ein erneutes abarbeiten des Monitors kann ausgelöst werden durch

- das Betätigen der RESET-Taste. Der RUN-Schalter muß dabei auf "0" stehen, da sonst beim Loslassen der RESET-Taste das Programm sofort wieder starten würde.
- den Befehl Monitoranruf = D7. Auf die genaue Bedeutung und Wirkung dieses Befehls wird später eingegangen.

Der Monitoranruf löst ein einmaliges bearbeiten des Monitorprogramms aus, und bringt somit die zum Zeitpunkt seines Auftretens vorhandenen Daten aus der CPU in das Display-Stack. Der Programmablauf wird durch einen Monitoranruf nicht gestört. Eine SINGLE-STEP-ähnliche Funktion wird erreicht, wenn der Schalter C_4 = HLT am BP auf "1" steht. Der Rechner arbeitet das Programm dann bis zu einem Monitoranruf ab und bleibt nach einem solchen Anruf im Monitor. Durch Weitertakten mit dem RUN-Schalter kann eine Programmbearbeitung bis zum nächsten Monitoranruf ausgelöst werden. Im Extremfall müsste also nach jedem Befehl ein Monitoranruf folgen, wenn eine dem SINGLE-STEP entsprechende Arbeitsweise erreicht werden soll.

LOAD ADR.- ,DEPOSIT- und EXAMINE-Funktion

Das RAM im System 6 liegt im Adressbereich 0400 bis 04FF. Mit den A-Schaltern wird bei LOAD ADR. nur die niederwertige Hälfte der tatsächlichen (absoluten) Adresse eingegeben. Der höherwertige Teil, der immer 04 sein muß, wird durch das Monitorprogramm selbständig hinzugefügt. Das bedeutet einerseits eine Erleichterung der Adresseingabe, verhindert jedoch das Verlassen des vorgegebenen RAM-Bereiches. So können z.B. die ROM-Inhalte nicht mit EXAMINE kontrolliert werden.
In der Anzeige erscheint beim Benutzen einer der Funktionen jetzt nur noch die niederwertige Hälfte der Adresse.
Ansonsten erfolgt die Bedienung wie im System 5.

Programmbeispiel: 8 bit-Zähler mit Monitoranruf

Das folgende Programm soll Arbeitsweise und Arbeitsgeschwindigkeit des Monitors demonstrieren. Dazu werden folgende Befehle benutzt:

INR = increment register (ddd) + 1 ⟶ ddd

Format:

0	0	d	d	d	1	0	0

Als Destination = ddd kann eines der sechs Register oder der Akkumulator gesetzt werden. Mit dem Code 110 = M(Memory) ist auch das Incrementieren von Speicherplätzen möglich.

Registercodes:
000 = B
001 = C
010 = D
011 = E
100 = H
101 = L
110 = M(Memeory)
111 = A(Akkumulator)

Als Adressierungsart liegt bei diesem Befehl Registeradressierung vor. Nur beim Code 110 wird Register-Indirekt adressiert. Adressregister ist das Registerpaar HL.

D7 = Monitoranruf

JMP = jump unconditionally (2.Byte) ⟶ PC low
(3.Byte) ⟶ PC high

Format:

1	1	0	0	0	0	1	1	1.Byte
niederwertige Adr.								2. "
höherwertige Adr.								3. "

Für die Jump- und Call-Befehle ist beim 8080 nur direkte Adressierung möglich.

ITT Fachlehrgänge

Programm:

Adresse	Inhalt	Befehl	;Kommentar
0400	3C		
0401	D7		
0402	C3		
0403	00		
0404	04		

Ungefähre Arbeitsgeschwindigkeit des Monitors:

Programmzählerstand bei HLT am BP:

Verhalten des Programms wenn der Monitoranruf in Adresse 0401 durch den Befehl 00 ≙ NOP ersetzt wird:

Programmbeispiel: 16 bit-Zähler ohne Monitoranruf

Zum Arbeiten mit 16 bit-Registern stehen im System 6 einige Befehle zur Verfügung bei denen oft ein X in der Mnemonic enthalten ist. Dieses X kann meist als "Register-Paar" gelesen werden.
Zur Realisierung dieses Programmbeispiels müssen folgende neuen Befehle verwendet werden:

INX = increment registerpair (rr) + 1 ⟶ rr

Format:

0	0	r	r	0	0	1	1

Als Registerpaar = rr können BC , DE , HL oder der Stackpointer gesetzt werden. Alle Flags bleiben unbeeinflusst!

Registerpaarcodes: 00 = B
01 = D
10 = H
11 = SP

Die gleichen Aussagen gelten auch für DCX = decrement reg.pair.

MOV = move (sss) ⟶ ddd

Format:

0	1	d	d	d	s	s	s

Als Source oder Destination darf auch M(=Memory) gesetzt werden, d.h. ein über HL adressierter Speicherplatz. Wenn für Source und Destination der Code 110 gesetzt wird, entsteht der OP-Code des HLT-Befehls(76).

OUT = output = Ausgabebefehl (A) ⟶ (2.Byte)

Format:

1	1	0	1	0	0	1	1	1.Byte
0-Port-Adresse (8 bit)								2.Byte

Datenausgaben können nur über den Akkumulator erfolgen.

ITT Fachlehrgänge

Programm:

Adresse	Inhalt	Befehl	;Kommentar	Zyklen
0400	03			
0401	78			
0402	D3			
0403	02			
0404	79			
0405	D3			
0406	01			
0407	C3			
0408	00			
0409	04			

Die Zyklusdauer im System 6 errechnet sich folgendermaßen:

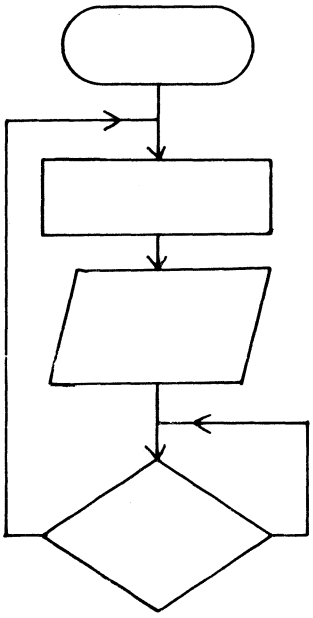
Quarzfrequenz = 8,89 MHz
Taktfrequenz = o,987 MHz (Teilung 9:1 im clock generator)
Zyklusdauer = 1,012 µs

Für unsere Berechnungen wird 1 µs als Zyklusdauer angenommen. Die Anzahl der benötigten Taktzyklen zur Bearbeitung eines Befehls kann der Befehlsliste im Anhang entnommen werden.

Programmbeispiel: Zähler mit Stopschalter

Der im vorhergehenden Beispiel entworfene Zähler soll so erweitert werden, daß er durch einen Schalter angehalten werden kann. Als Schalter soll C_1 verwendet werden.

$C_1 = 0$ Zähler läuft
$C_1 = 1$ Zähler steht



Zur Erweiterung des Programms durch die Schalterabfrage, Maskierung und Verzweigung werden folgende neuen Befehle benötigt:

IN = input = Einlesebefehl ((2.Byte)) ⟶ A

Format:

1	1	0	1	1	0	1	1	1.Byte
I-Port-Adresse (8 bit)								2.Byte

Dateneinlesen kann nur über den Akkumulator erfolgen.

ITT Fachlehrgänge

ANI = AND immediate (A) ∧ (2.Byte) ⟶ A

Format:

1	1	1	0	0	1	1	0	1.Byte
Datenwort (8 bit)								2.Byte

Alle immediate adressierten Arithmetik- und Logik-Befehle benutzen den Akkumulator als Operand und Destination.

J.. = jump conditionally = bedingter Sprung

(2.Byte) PC low

(3.Byte) PC high

Format:

1	1	b	b	b	0	1	0	1.Byte
niederwertige Adresse								2. "
höherwertige Adresse								3. "

Als Branch-Condition = bbb (= Verzweigungsbedingung) kann einer der folgenden Brach-Condition-Codes gesetzt werden.

Branch-Codes:

000 = JNZ = jump if not zero
001 = JZ = " zero
010 = JNC = " no carry
011 = JC = " carry
100 = JPO = " parity odd
101 = JPE = " parity even
110 = JP = " plus
111 = JM = " minus

Nur direkte Adressierung möglich!

ITT Fachlehrgänge

Programm:

Adresse	Inhalt	Befehl	;Kommentar
0400	03	INX B	
0401	78	MOV A,B	
0402	D3	OUT 02	
0403	02		
0404	79	MOV A,C	
0405	D3	OUT 01	
0406	01		
0407	DB		
0408	04		
0409	E6		
040A	02		
040B	C2		
040C	07		
040D	04		
040E	C3		
040F	00		
0410	04		

Zusammenfassung der 1 Byte-Datentransferbefehle

Die nachfolgende Darstellung zeigt die möglichen 1 Byte-Datentransferwege für den MP 8o8o. Unterteilt nach den Adressierungsarten ergeben sich somit die Befehle:

1. Registeradressierung

MOV _,_ = move to register from register

2. Immediate Adressierung

a) Destination: M = memory

MVI M,data = move to memory immediate

b) Destination: Register

MVI _,data = move to register immediate

3. Direkte Adressierung

Nur zum und vom Akkumulator möglich!

a) Destination: M = memory

STA = store accu direct

b) Destination: Akku

LDA = load accu direct

c) I/O-Transfers:

Für die Befehle IN und OUT wird im 2.Byte die 8-bit-Adresse des I/O-Port angegeben, mit dem der Akkumulator arbeiten soll.

ITT Fachlehrgänge

4. Registerindirekte Adressierung

a) Destination: M = memory
MOV M,_

STAX = store accu indirect

b) Destination: Register
MOV _,M

LDAX = load accu indirect

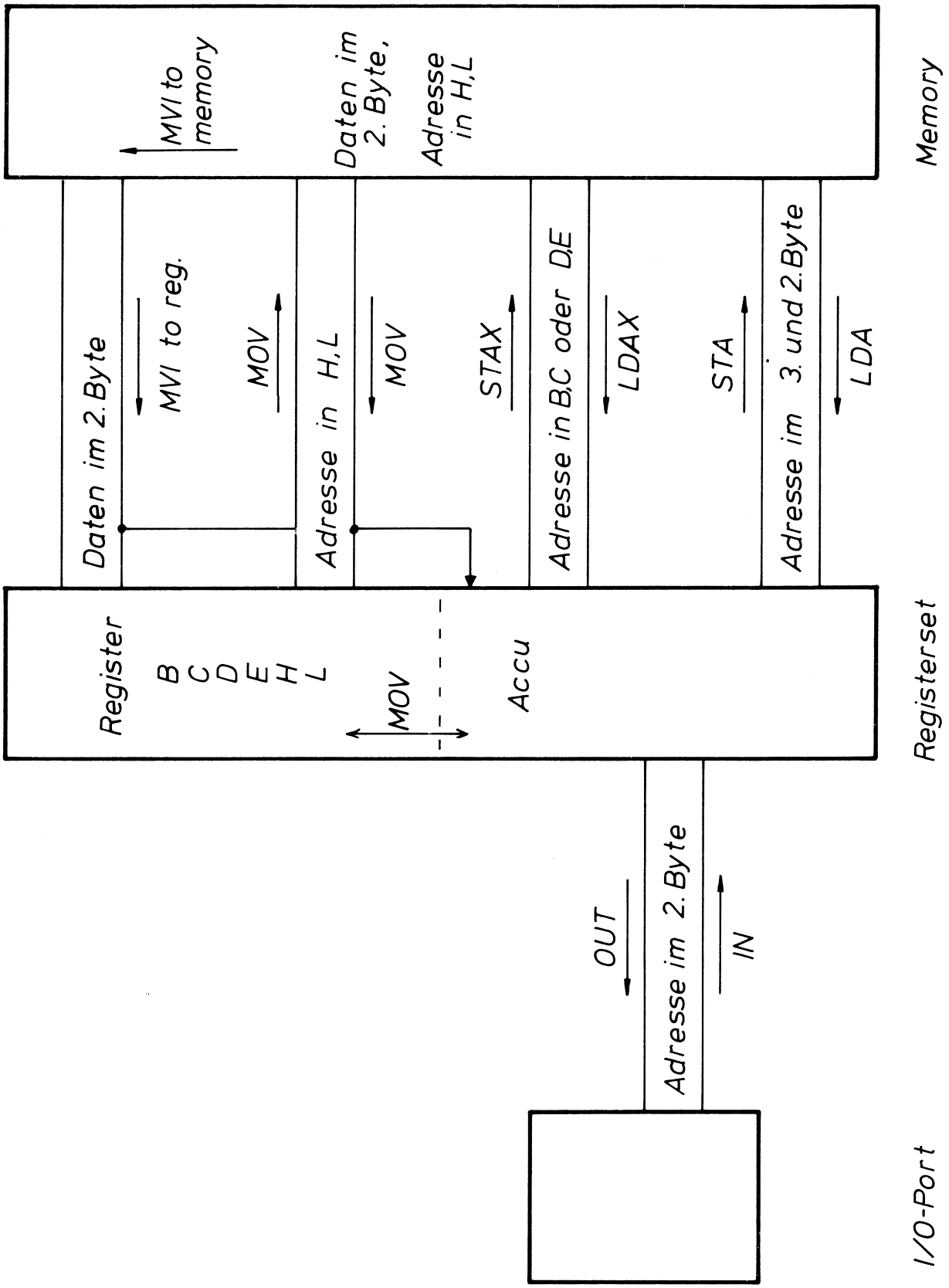
Die Programmiertafel 1 enthält die Befehle MOV und MVI für alle möglichen Source- und Destinationfälle.

1

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

ITT Fachlehrgänge

1 Byte Datentransfer

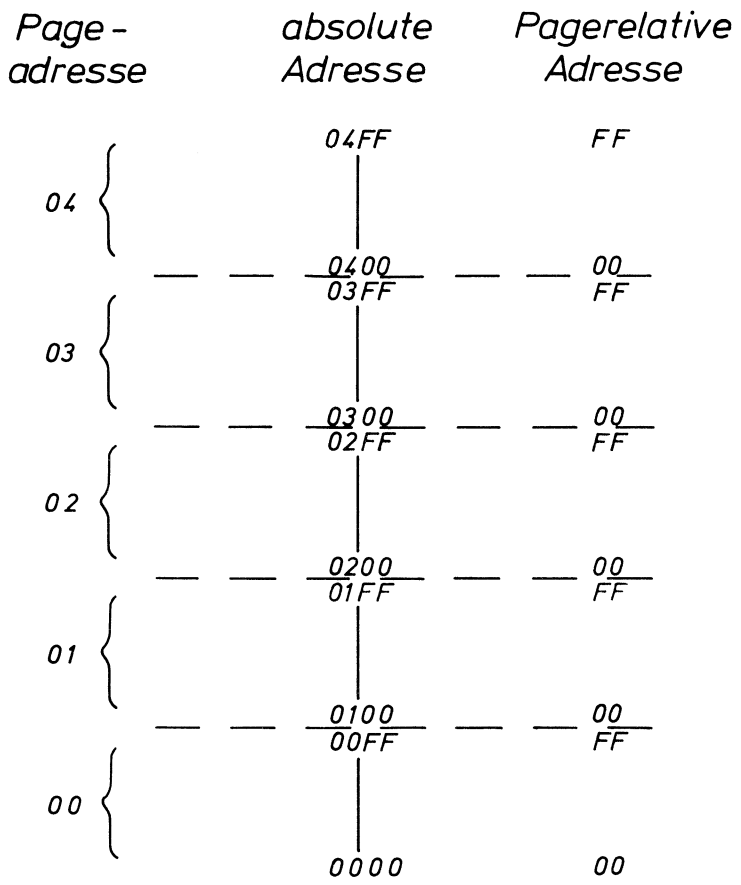


ITT Fachlehrgänge

Page-relative Adressierung

Zur besseren Überschaubarkeit und zur Vereinfachung der Adressierung können Speicher in sogenannte Pages (Page = Seite) unterteilt werden. Die Adresswortlänge wird dabei reduziert, da nur noch der niederwertige Teil der Adresse angegeben werden muß. Die Adressierung im Monitor für die Funktionen LOAD ADR., EXAMINE und DEPOSIT arbeitet auf eine ähnliche Art und Weise.
Das im System 6 verfügbare RAM liegt im Adressbereich o4oo bis o4FF, d.h. in Page o4. Die bereits erwähnte RAM-relative Adressierung entspricht also einer Page-relativen Adressierung für nur eine Page.

Die Größe einer Page ist 1/4 K = 256 Byte. Für den 8o8o würde also die Adresswortlänge bei Page-relativer Adressierung auf die Hälfte reduziert. Lediglich beim Wechseln zu anderen Pages müssten neue Befehle diesen Wechsel veranlassen.



Programmbeispiel: Minimonitor

Der Monitor im System 6 erlaubt nur den Zugriff zu den Speicherplätzen im RAM. Das nachfolgende Programm soll eine EXAMINE-Funktion für alle Adressen des 8o8o-Adressbereichs ermöglichen.

Funktionen:

1.) LOAD ADR. = high

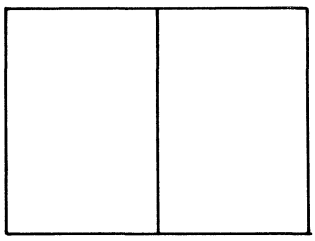
Die in den A-Schaltern stehende Zahl wird als High-Addresse (Page) gelesen und in den rechten Lampen angezeigt. Eine Änderung der Page ist solange möglich, wie der LOAD ADR.-Schalter in High-Stellung bleibt.

2.) LOAD ADR. = low

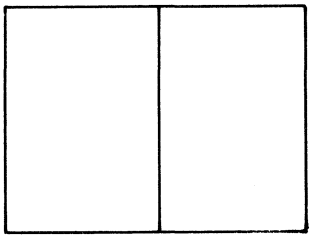
Die in den A-Schaltern stehende Zahl wird als Low-Addresse (Page-relativ) gelesen. Der Inhalt der zusammengesetzten Adresse wird in den linken Lampen angezeigt.

Für die Hexadezimal-Displays ergibt sich damit folgende Zuordnung:

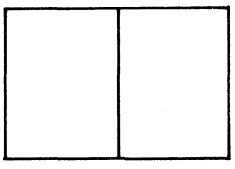
Inhalt der angewählten Adresse



Pageadresse



Pagerelative Adresse



Programmbeispiel: Minimonitor

Flußdiagramm:

ITT Fachlehrgänge

NAME:..... BLATT:.....

PROGRAMM:.....

LABEL	Adresse	Inhalt	Befehl	; Kommentar

Programmbeispiel: Minimonitor

Mit dem Minimonitor sollen die Inhalte folgender Speicherplätze im ROM ermittelt werden:

Adresse	Inhalt	Befehl
0000		
0001		
0002		
0003		
0004		
0005		
0006		
0007		
0008		
0009		
000A		
0038		
0039		
003A		

RST-Befehle (Restart-Befehle)

Die RST-Befehle sind von ihrer Funktion her CALL-Befehle, können jedoch nur acht verschiedene Zieladressen ansprechen. Ein Vorteil liegt darin, daß nur 1 Byte für das Abspeichern des Befehls benötigt wird. Im Vergleich mit dem CALL-Befehl entfallen dadurch zwei Speicherzugriffe im Instruction-Fetch (Befehl holen). Der RST-Befehl benötigt 11 Taktperioden, während ein CALL-Befehl in 17 Taktperioden ausgeführt wird.

RST = restart

Format:

1	1	a	a	a	1	1	1

(PCH) ⟶ (SP)-1
(PCL) ⟶ (SP)-2
oo ⟶ PCH
aaa x 8 ⟶ PCL

Für aaa können Werte von ooo bis 111 gesetzt werden. Damit ergeben sich folgende Zieladressen für die acht RST-Befehle:

Befehl	Hexcode	Zieladresse
RST o	C7	oooo
RST 1	CF	ooo8
RST 2	D7	oo1o
RST 3	DF	oo18
RST 4	E7	oo2o
RST 5	EF	oo28
RST 6	F7	oo3o
RST 7	FF	oo38

Alle angegebenen Zieladressen liegen im ROM-Bereich. Um die Befehle RST 1 und RST 7 für eigene Programme verwenden zu können, sind in den entsprechenden Zieladressen JUMP-Befehle zu den RAM-Adressen o4o8 und o41o untergebracht. Alle anderen RST-Befehle sind im Monitorprogramm verwendet.

Der Befehl RST 2 (Hexcode D7) nimmt als Monitoranruf eine Sonderstellung ein.

Interrupt-Signale und -Befehle

Interrupts sind Programmunterbrechungen die durch Signale (Hardware) ausgelöst werden. Ein Interrupt, d.h. die Unterbrechung eines Hauptprogrammes, löst in der Regel die Bearbeitung eines Unter-Programmes (interrupt service routine) aus. Das Interruptprogramm kann entweder im Programmspeicher des Prozessors gespeichert sein oder von dem Peripheriegerät geliefert werden, das den Interrupt ausgelöst hat.

Nach einem akzeptierten Interrupt leitet der Prozessor einen instruction fetch (Befehl holen) ein. Dabei wird der Befehl jedoch nicht aus dem Speicher geholt, sondern der aktuelle Zustand des Datenbus wird als Befehl interpretiert. Das bedeutet, daß der erste Befehl nach einem Interrupt immer von der Hardware geliefert werden muß.

Dieser erste Befehl sollte zweckmäßigerweise ein RST- oder CALL-Befehl sein, damit die Zieladresse für den Rücksprung in das Hauptprogramm gespeichert wird.

Um die Behandlung von Interrupts zu vereinfachen, liefert der System-Controller-Baustein 8228 nach einem akzeptierten Interrupt automatisch einen RST 7 Befehl. In der Zieladresse dieses RESTART-Befehls ist ein Sprungbefehl (JMP o41o) in den Adressbereich des RAM gespeichert. Ein bei der Adresse o41o beginnendes Unterprogramm kann also durch einen Interrupt gestartet werden.

Signale und Befehle

INTE = interrupt enable

Dieses Signal zeigt an, ob der Prozessor Interrupts akzeptiert (INTE = high) oder nicht.

DI = disable interrupt

Der Befehl DI setzt INTE = "0". Das Gleiche geschieht bei RESET oder beim Akzeptieren eines Interrupt.

EI = enable interrupt

Der Befehl EI muß in einem Programm enthalten sein, um nach dem Einschalten des Gerätes oder RESET Interrupts zuzulassen. Sollen nach einem Interrupt neue Unterbrechungen zugelassen werden, muß ebenfalls der Befehl EI erscheinen.

INT = interrupt

Dieser Eingang muß high sein, um dem Prozessor anzuzeigen, daß ein externes Gerät einen Interrupt anfordert.

$\overline{\text{INTA}}$ = interrupt acknowledge

Mit diesem Signal quitiert der Prozessor einen akzeptierten Interrupt.

Die hier aufgeführten Signale und Befehle beziehen sich auf den 8o8o direkt. Die Verhältnisse im Lehrsystem sollen anhand des folgenden Schaltbildes gezeigt und erläutert werden.

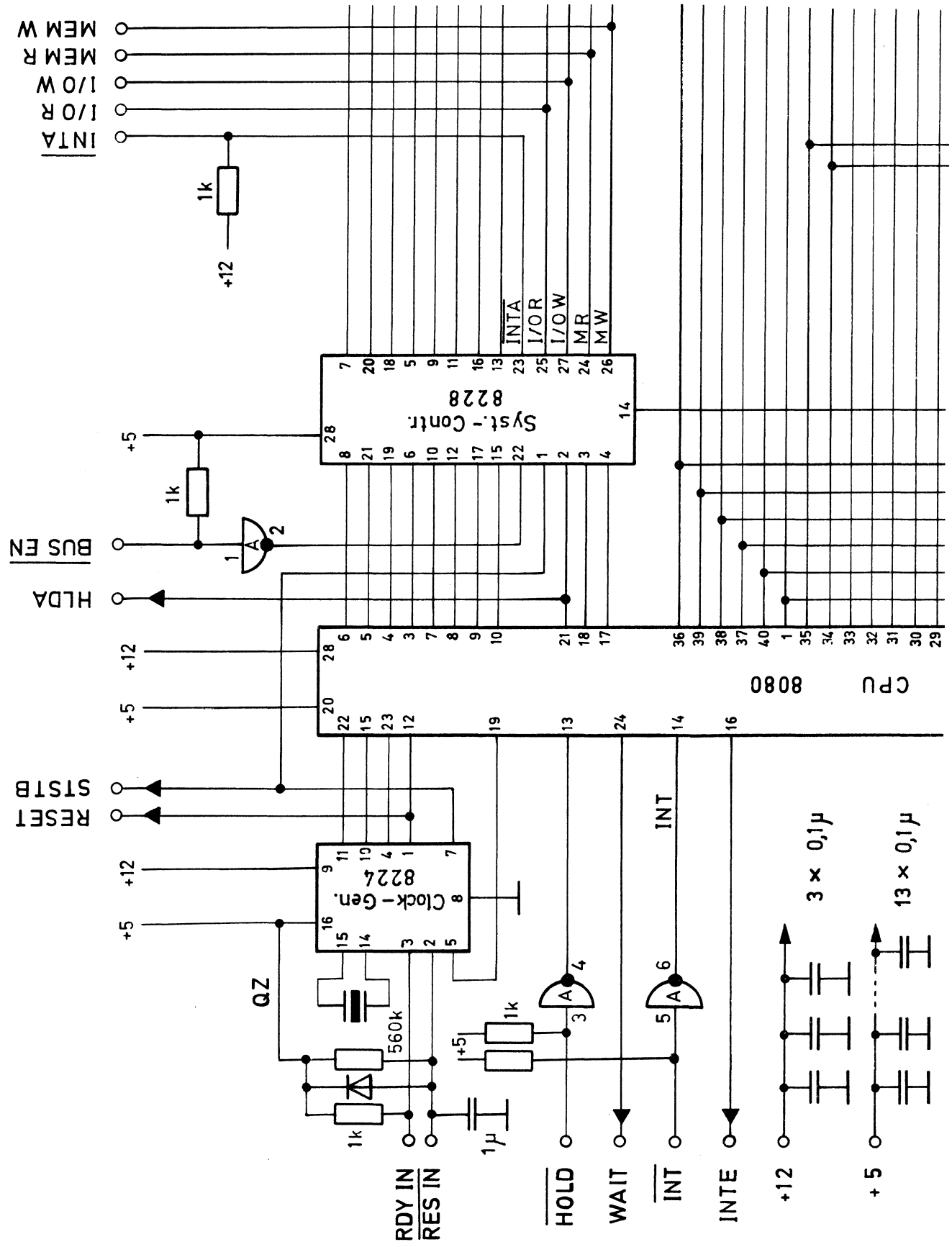
ITT Fachlehrgänge

Der INT-Eingang der CPU (pin 14) wird durch den vorgeschalteten Inverter ständig auf "0" gehalten.
Der $\overline{\text{INT}}$-Eingang des Systems muß also "0" sein,um einen Interrupt auszulösen.

Der INTE-Ausgang liefert bei freigegebenem Interrupt ein High-Signal von ca. 3,5 V und ein Fan out von 1 1/4 LE (ca. 2 mA).

Der $\overline{\text{INTA}}$-Ausgang quitiert eine akzeptierte Interruptanforderung mit einem Low-Signal. Wird der $\overline{\text{INTA}}$-Ausgang jedoch über 1 kOhm an + 12 V gelegt, erzeugt der System-Controller einen RST 7 -Befehl. Das System kennt somit nur eine Zieladresse (Vektor) für Interrupt.

Schaltungsauszug MP-Lehrsystem



Zusammenfassung der Arithmetik/logik-Befehle

Alle Arithmetik- und Logik-Befehle benutzen den Akkumulator als Operanden und gleichzeitig als Destination für das Ergebnis. (Ausgenommen davon sind nur die Vergleichsbefehle.)
Somit gilt:

$$(A) \left\{ \begin{matrix} + \\ - \\ \wedge \\ \vee \\ \not\vee \end{matrix} \right\} (sss) \longrightarrow A$$

Für sss = Source können alle Register (einschließlich des Akkumulators selbst) oder ein Speicherplatz stehen.

Unterteilt nach Adressierungsarten ergeben sich folgende Möglichkeiten:

1.) Immediate Adressierung

$$(A) \left\{ \begin{matrix} + \\ - \\ \wedge \\ \vee \\ \not\vee \end{matrix} \right\} (2.Byte) \longrightarrow A$$

ADI = add immediate
SUI = subtract immediate
ANI = AND immediate
ORI = OR immediate
XRI = Exclusive-OR immediate

Bei den Arithmetik-Befehlen besteht außerdem noch die Möglichkeit ein zuvor entstandenes Carry-Flag (bzw. eine Entlehnung) zu berücksichtigen.

ACI = add with carry immediate

SBI = subtract with borrow immediate

ITT Fachlehrgänge

2.) Register-Adressierung

$$(A) \left\{ \begin{matrix} + \\ - \\ \wedge \\ \vee \\ \not\vee \end{matrix} \right\} (SSS) \longrightarrow A \qquad SSS \neq 110$$

ADD
SUB
ANA
ORA
XRA

Auch hier besteht wieder die Möglichkeit das Carry-Flag zu berücksichtigen.

ADC = add with carry
SBB = subtract with borrow

3.) Registerindirekte Adressierung

$$(A) \left\{ \begin{matrix} + \\ - \\ \wedge \\ \vee \\ \not\vee \end{matrix} \right\} ((HL)) \longrightarrow A$$

Hier wird wie bei allen Memorybefehlen das Registerpaar HL als Adressregister benutzt. Die Mnemonic bleibt die gleiche wie für Register-Adressierung.

Die Vergleichsbefehle stellen einen Sonderfall dar, da hier eine Subtraktion ausgeführt wird, deren Ergebnis nirgends erscheint. Es werden lediglich (entsprechend dem Subtraktionsergebnis) die Flags gesetzt. Beide Operanden bleiben also unverändert !

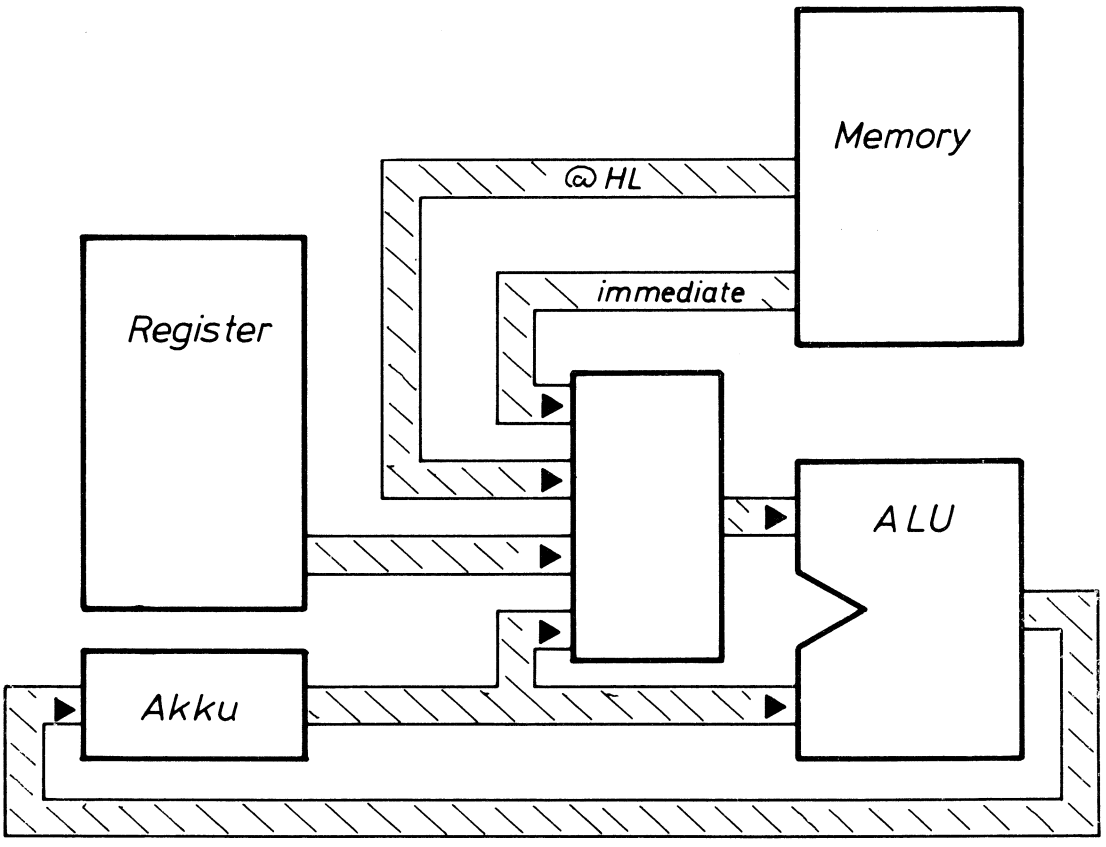
CPI = compare immediate (A) - (2.Byte)
CMP = compare (A) - (sss)

In der folgenden Programmiertafel sind auch die Increment- und Decrement-Befehle für alle Register enthalten. Mit INR M und DCR M ist es außerdem auch möglich Speicherplätze als Zähler zu benutzen. Adressregister ist auch hier wieder das Registerpaar HL.

ITT Fachlehrgänge

Programmiertafel und Funktionsschaltung Arithmetik/Logik

		SOURCE							
		B	C	D	E	H	L	M	A
ADI = C6 SUI = D6 ANI = E6 ORI = F6	ADD = 8_ SUB = 9_ ANA = A_ ORA = B_	_0	_1	_2	_3	_4	_5	_6	_7
ACI = CE SBI = DE XRI = EE CPI = FE	ADC = 8_ SBB = 9_ XRA = A_ CMP = B_	_8	_9	_A	_B	_C	_D	_E	_F
	INR	04	0C	14	1C	24	2C	34	3C
	DCR	05	0D	15	1D	25	2D	35	3D



ITT Fachlehrgänge

Zusammenfassung der 2 Byte-Datentransfer-Befehle

Bei Operationen mit Adressen müssen beim 8080 16 bit-Worte verarbeitet werden. Dazu ist im Befehlsvorrat eine Reihe von 2 Byte-Transferbefehlen vorhanden. Auch hier sollen die Befehle wieder getrennt nach Adressierungsarten besprochen werden.

1.) Registeradressierung (Registerpaaradressierung)

XCHG = exchange (DE) ⟶ HL
(HL) ⟶ DE

Dies ist einer der wenigen Befehle bei dem sich die Inhalte von Source und Destination verändern.

SPHL = stackpointer from HL (HL) ⟶ SP

PCHL = program counter from HL (HL) ⟶ PC

Der Befehl PCHL entspricht in seiner Wirkung einem Sprung-Befehl.

2. Immediate Adressierung

LXI = load registerpair immediate
(3.Byte) ⟶ höherwertiges Register
(2.Byte) ⟶ niederwertiges Register

3. Direkte Adressierung

In dieser Adressierungsart besteht nur die Möglichkeit das Registerpaar HL zu laden, bzw. seinen Inhalt abzuspeichern.

SHLD = store HL direkt
(H) ⟶ (3.Byte)(2.Byte) + 1
(L) ⟶ (3.Byte)(2.Byte)

LHLD = load HL direkt
((3.Byte)(2.Byte) + 1) ⟶ H
((3.Byte)(2.Byte)) ⟶ L

4. Registerindirekte Adressierung

Im 8080 kann nur der Stack-Pointer für Registerindirekte Adressierung benutzt werden.

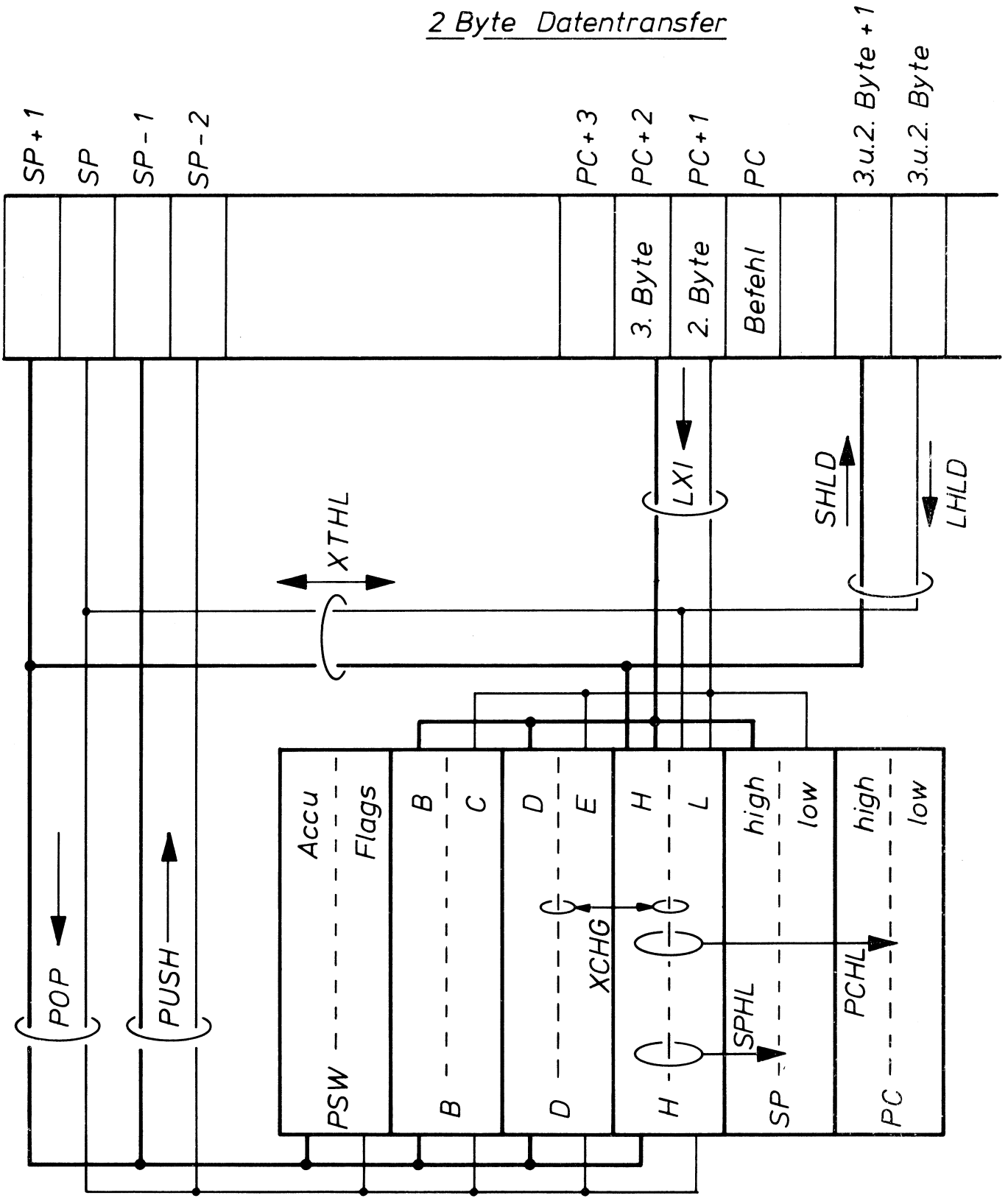
PUSH = push register pair onto stack
(höherwertiges Register) ⟶ (SP)-1
(niederwertiges Register) ⟶ (SP)-2

POP = pop off stack to register pair
((SP)+1) ⟶ höherwertiges Register
((SP)) ⟶ niederwertiges Register

Für die PUSH- und POP-Befehle gilt:

Registerpaarcode = 11 ≙ PSW (program status word)

Die Darstellung auf der folgenden Seite soll die Datentransferwege für 16 bit-Worte nochmals verdeutlichen.


2 Byte Datentransfer
SP + 1
SP
SP - 1
SP - 2
PC + 3
PC + 2
PC + 1
PC
3.u.2. Byte + 1
3.u.2. Byte
3. Byte
2. Byte
Befehl
LXI
SHLD
LHLD
XTHL
POP
PUSH
XCHG
SPHL
PCHL
Accu
Flags
PSW
B
C
D
E
H
L
high
low
SP
PC

Dezimalarithmetik

Zum Arbeiten im dezimalen Zahlensystem hat die CPU 8o8o nur den Befehl DAA zur Verfügung.

DAA = decimal adjust accumulator

Die Wirkung dieses Befehls ist vom Zustand des Half-Carry-Flag (H = Übertrag in Wortmitte von bit 3 nach bit 4) abhängig.

Voraussetzung für die Anwendung des DAA-Befehls ist, daß der Akkumulator die Summe zweier Dezimalzahlen enthält.

Die folgende Tabelle soll den Ablauf der Korrektur unter verschiedenen Bedingungen zeigen.

$bit_{4..7}$ 9 oder C-Flag	$bit_{o..3}$ 9 oder H-Flag	Wirkung	Kommentar
0	0		
0	1		
1	0		
1	1		

Programmbeispiel: Dezimaladdierer

Entwickeln Sie ein Programm das nachfolgende Bedingungen erfüllt:

1.) Das Registerpaar BC soll als 4-stellig dezimales Summenregister arbeiten.

2.) Durch betätigen der DEPOSIT-Taste soll ein in der Tastatur eingegebener Dezimalwert aufaddiert und die Summe angezeigt werden.

3.) Der Ablauf der Addition und die Resultatanzeige soll mit der positiven Flanke des DEPOSIT-Signals erfolgen.

4.) Der Schalter C_o (LOAD ADR.) soll als CLEAR-Schalter zum Löschen der Summe verwendet werden.

Programmbeispiel: Dezimaladdierer

Flußdiagramm:

ITT Fachlehrgänge

NAME:.................... BLATT:........

PROGRAMM:................................

LABEL	Adresse	Inhalt	Befehl	; Kommentar

ITT Fachlehrgänge **Befehlsliste 8o8o**

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

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

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

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

Registercode für s s s bzw. d d d:

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

Abkürzungen:

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

ITT Fachlehrgänge

DESTINATION	SOURCE: MOV B	C	D	E	H	L	Accu	Memory @ HL	MVI # 2.Byte
Register B	40	41	42	43	44	45	47	46	06
" C	48	49	4A	4B	4C	4D	4F	4E	0E
" D	50	51	52	53	54	55	57	56	16
" E	58	59	5A	5B	5C	5D	5F	5E	1E
" H	60	61	62	63	64	65	67	66	26
" L	68	69	6A	6B	6C	6D	6F	6E	2E
Accu	78	79	7A	7B	7C	7D	7F	7E	3E
Memory @ HL	70	71	72	73	74	75	77	HLT	36

ADI = C6
SUI = D6
ANI = E6
ORI = F6

ACI = CE
SBI = DE
XRI = EE
CPI = FE

	SOURCE: B	C	D	E	H	L	M	A
ADD = 8_ SUB = 9_ ANA = A_ ORA = B_	_0	_1	_2	_3	_4	_5	_6	_7
ADC = 8_ SBB = 9_ XRA = A_ CMP = B_	_8	_9	_A	_B	_C	_D	_E	_F
INR	04	0C	14	1C	24	2C	34	3C
DCR	05	0D	15	1D	25	2D	35	3D

JMP = C3
CALL = CD
RET = C9

	CONDITION: NZ	Z	NC	C	PO	PE	P	M
J...	C2	CA	D2	DA	E2	EA	F2	FA
C...	C4	CC	D4	DC	E4	EC	F4	FC
R	C0	C8	D0	D8	E0	E8	F0	F8

not zero, zero, no carry, carry, parity odd, parity even, plus, minus

	B	D	H	SP
LXI	01	11	21	31
INX	03	13	23	33
DCX	0B	1B	2B	3B
DAD	09	19	29	39

	B	D	H	PSW
PUSH	C5	D5	E5	F5
POP	C1	D1	E1	F1

ITT Fachlehrgänge

Befehlsliste 8080

Hexadezimal geordnet

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

ITT Fachlehrgänge

Befehlsliste 8080

Alphabetisch geordnet

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

ITT Fachlehrgänge

Befehlsliste 8080

Nach Funktionsgruppen geordnet

Data Transfer

HEX	MNEMONIC	HEX	MNEMONIC	HEX	MNEMONIC	HEX	MNEMONIC
40	MOV B,B	58	MOV E,B	70	MOV M,B	1A	LDAX D
41	MOV B,C	59	MOV E,C	71	MOV M,C	2A	LHLD
42	MOV B,D	5A	MOV E,D	72	MOV M,D	3A	LDA
43	MOV B,E	5B	MOV E,E	73	MOV M,E	02	STAX B
44	MOV B,H	5C	MOV E,H	74	MOV M,H	12	STAX D
45	MOV B,L	5D	MOV E,L	75	MOV M,L	22	SHLD
46	MOV B,M	5E	MOV E,M	77	MOV M,A	32	STA
47	MOV B,A	5F	MOV E,A	78	MOV A,B	01	LXI B
48	MOV C,B	60	MOV H,B	79	MOV A,C	11	LXI D
49	MOV C,C	61	MOV H,C	7A	MOV A,D	21	LXI H
4A	MOV C,D	62	MOV H,D	7B	MOV A,E	31	LXI SP
4B	MOV C,E	63	MOV H,E	7C	MOV A,H	F9	SPHL
4C	MOV C,H	64	MOV H,H	7D	MOV A,L	E3	XTHL
4D	MOV C,L	65	MOV H,L	7E	MOV A,M	EB	XCHG
4E	MOV C,M	66	MOV H,M	7F	MOV A,A	D3	OUT
4F	MOV C,A	67	MOV H,A	06	MVI B	DB	IN
50	MOV D,B	68	MOV L,B	0E	MVI C	C5	PUSH B
51	MOV D,C	69	MOV L,C	16	MVI D	D5	PUSH D
52	MOV D,D	6A	MOV L,D	1E	MVI E	E5	PUSH H
53	MOV D,E	6B	MOV L,E	26	MVI H	F5	PUSH PSW
54	MOV D,H	60	MOV L,H	2E	MVI L	C1	POP B
55	MOV D,L	6D	MOV L,L	36	MVI M	D1	POP D
56	MOV D,M	6E	MOV L,M	3E	MVI A	E1	POP H
57	MOV D,A	6F	MOV L,A	0A	LDAX B	F1	POP PSW

Arithmetic

HEX	MNEMONIC	HEX	MNEMONIC	HEX	MNEMONIC	HEX	MNEMONIC
80	ADD B	C6	ADI	9E	SBB M	3C	INR A
81	ADD C	CE	ACI	9F	SBB A	03	INX B
82	ADD D	90	SUB B	D6	SUI	13	INX D
83	ADD E	91	SUB C	DE	SBI	23	INX H
84	ADD H	92	SUB D	09	DAD B	33	INX SP
85	ADD L	93	SUB E	19	DAD D	05	DCR B
86	ADD M	94	SUB H	29	DAD H	0D	DCR C
87	ADD A	95	SUB L	39	DAD SP	15	DCR D
88	ADC B	96	SUB M	27	DAA	1D	DCR E
89	ADC C	97	SUB A	04	INR B	25	DCR H
8A	ADC D	98	SBB B	0C	INR C	2D	DCR L
8B	ADC E	99	SBB C	14	INR D	35	DCR M
8C	ADC H	9A	SBB D	1C	INR E	3D	DCR A
8D	ADC L	9B	SBB E	24	INR H	0B	DCX B
8E	ADC M	9C	SBB H	2C	INR L	1B	DCX D
8F	ADC A	9D	SBB L	34	INR M	2B	DCX H
						3B	DCX SP

Logical

HEX	MNEMONIC	HEX	MNEMONIC	HEX	MNEMONIC	HEX	MNEMONIC
A0	ANA B	A9	XRA C	B2	ORA D	BB	CMP E
A1	ANA C	AA	XRA D	B3	ORA E	BC	CMP H
A2	ANA D	AB	XRA E	B4	ORA H	BD	CMP L
A3	ANA E	AC	XRA H	B5	ORA L	BE	CMP M
A4	ANA H	AD	XRA L	B6	ORA M	BF	CMP A
A5	ANA L	AE	XRA M	B7	ORA A	FE	CPI
A6	ANA M	AF	XRA A	F6	ORI	07	RLC
A7	ANA A	EE	XRI	B8	CMP B	0F	RRC
E6	ANI	B0	ORA B	B9	CMP C	17	RAL
A8	XRA B	B1	ORA C	BA	CMP D	1F	RAR
						2F	CMA

Branching

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

Control

HEX	MNEMONIC
00	NOP
76	HLT
F3	DI
FB	EI
37	STC
3F	CMC

Programmbeispiel: Schrittmotor

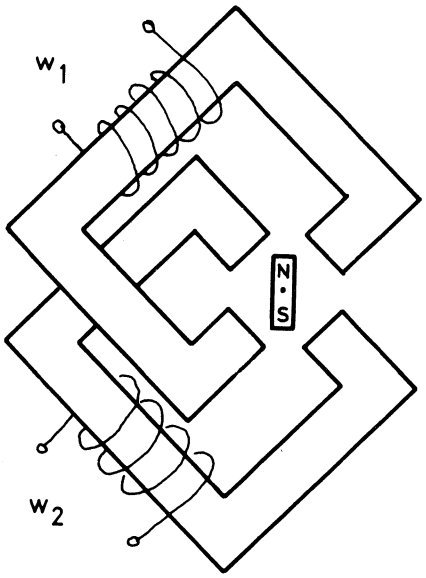
Das folgende Programmbeispiel sol die Anwendung eines Mikroprozessors zur Steurung eines Schrittmotors zeigen.

Wichtiger Programmpunkt ist dabei die Erzeugung des benötigten Impulsmusters.

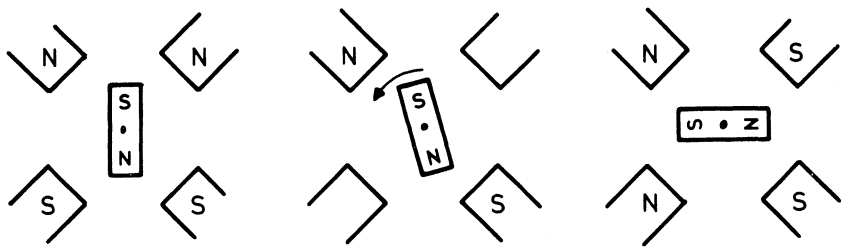
Die Möglichkeiten eines Mikroprozessors sind durch ein solches Programm natürlich längst nicht ausgenutzt. Selbst bei der schnellsten vorgesehenen Geschwindigkeit werden zwischen zwei Schritten 8184 Befehle in einer Zeitschleife vergeudet. Daraus ist zu ersehen, daß auch umfangreichere Zusatzaufgaben zwischen zwei Schritten erledigt werden könnten.

ITT Fachlehrgänge

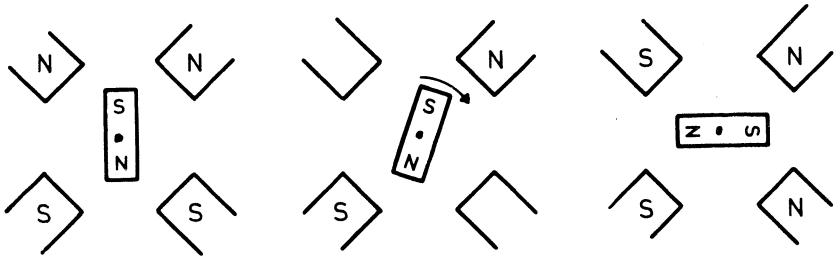
Prinzipieller Aufbau und Funktionsweise eines Schrittmotors



Linkslauf:

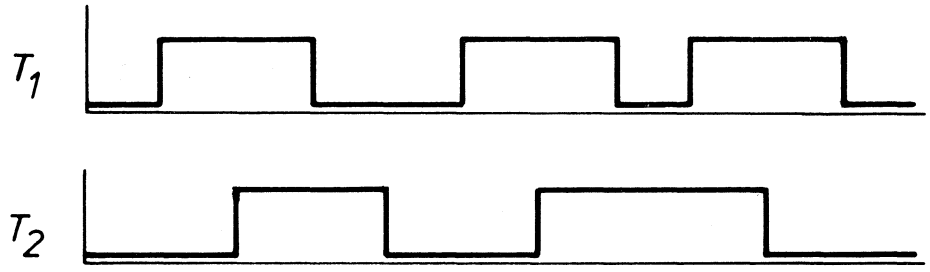


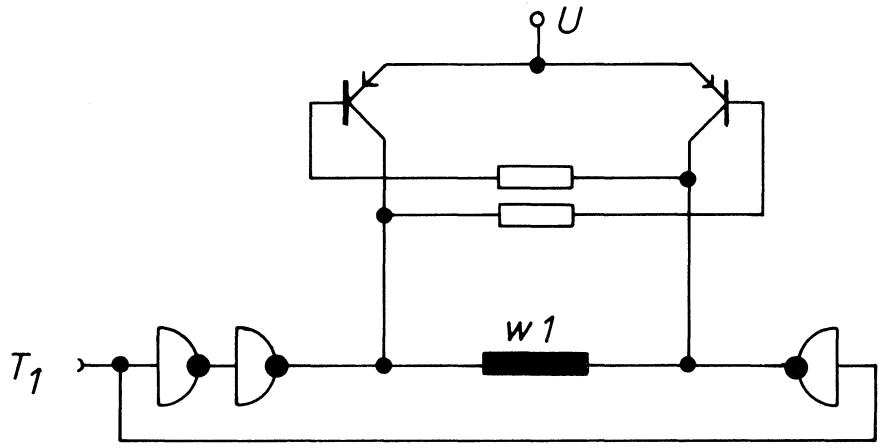
Rechtslauf:



Pulsdiagramm und Interface

Die Untenstehende Schaltung zeigt die Ansteuerung für eine der beiden Wicklungen.

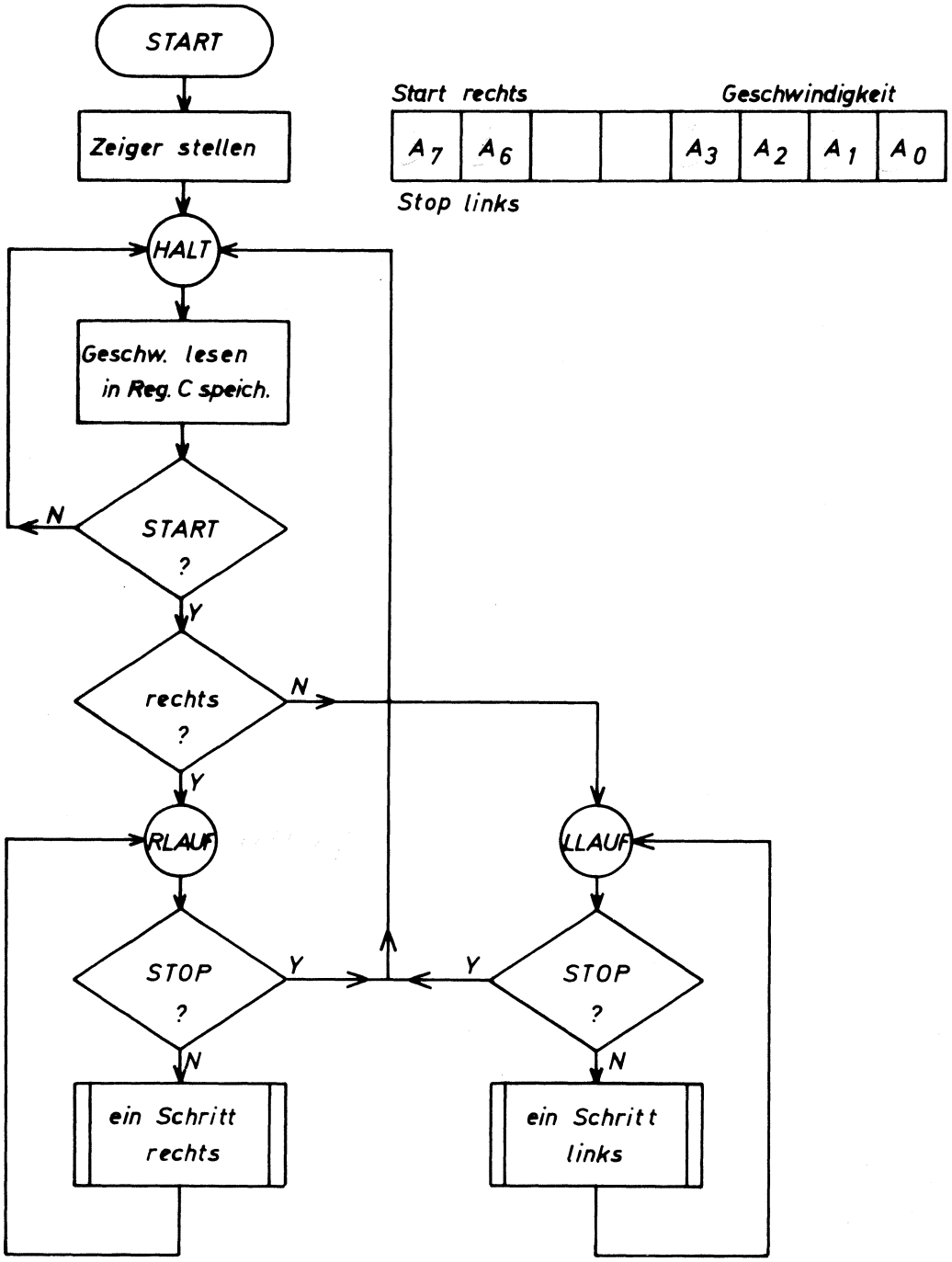




ITT Fachlehrgänge

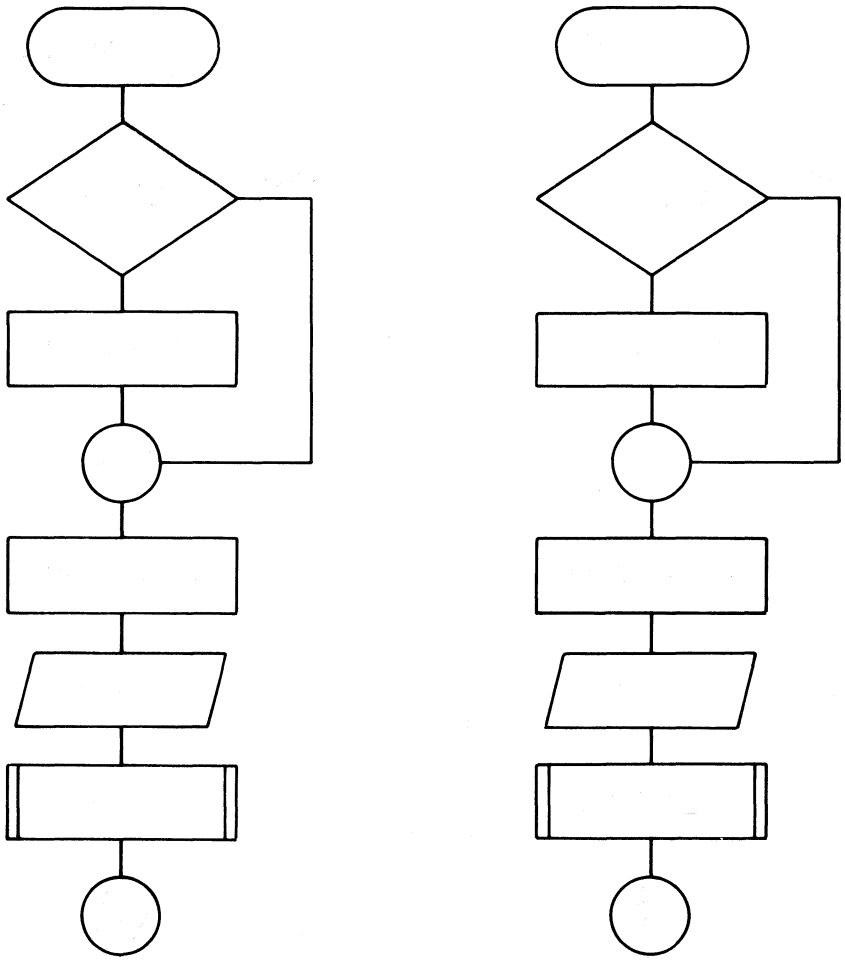
Flußdiagramm: Schrittmotorsteuerung

Gesamtübersicht:



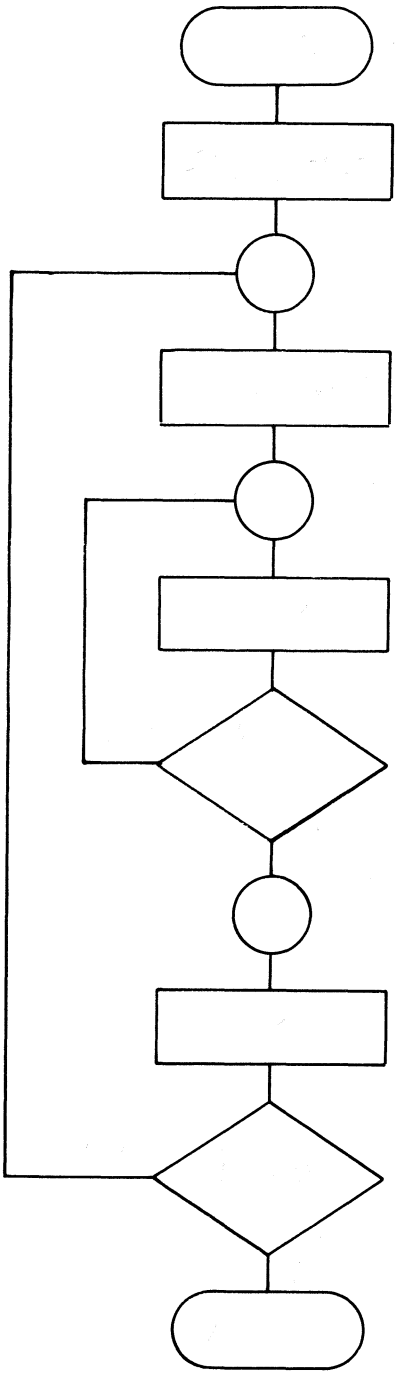
Flußdiagramm: Schrittmotorsteuerung

Teilprogramme für Rechts- und Linkslauf:



Flußdiagramm: Schrittmotorsteuerung

Teilprogramm für die Geschwindigkeitsbestimmung:



ITT Fachlehrgänge

NAME: ..... BLATT: 1

PROGRAMM: Schrittmotor

LABEL	Adresse	Inhalt	Befehl	; Kommentar
	0400	11	LXI D,04A0	
	01	A0		
	02	04		
	03	DB	IN BSHALT	
	04	01		
	05	47	MOV B,A	
	06	E6	ANI 0F	
	07	0F		
	08	4F	MOV C,A	
	09	78	MOV A,B	
	0A	B7	ORA A	
	0B	F2	JP	
	0C	03		
	0D	04		
	0E	E6	ANI 40	
	0F	40		
	0410	CA	JZ	
	11	2B		
	12	04		
	13	DB	IN BSHALT	
	14	01		
	15	B7	ORA A	
	16	F2	JP	
	17	03		
	18	04		
	19	7B	MOV A,E	
	1A	FE	CPI A3	
	1B	A3		
	1C	C2	JNZ	
	1D	21		
	1E	04		
	1F	1E	MVI E,9F	
	0420	9F		
	21	13	INX D	
	22	1A	LDAX D	
	23	D3	OUT LLAMPE	
	24	02		
	25	CD	CALL	
	26	50		
	27	04		
	28	C3	JMP	
	29	13		
	2A	04		

ITT Fachlehrgänge

NAME: ..... BLATT: 2

PROGRAMM: Schrittmotor

LABEL	Adresse	Inhalt	Befehl	; Kommentar
	042B	DB	IN BSHALT	
	C	01		
	D	B7	ORA A	
	E	F2	JP	
	F	03		
	30	04		
	1	7B	MOV A,E	
	2	FE	CPI A0	
	3	A0		
	4	C2	JNZ	
	5	39		
	6	04		
	7	1E	MVI E,A4	
	8	A4		
	9	1B	DCX D	
	A	1A	LDAX D	
	B	D3	OUT LLAMPE	
	C	02		
	D	CD	CALL	
	E	50		
	F	04		
	40	C3	JMP	
	1	2B		
	2	04		
	0450	41	MOV B,C	
	1	21	LXI H,08FF	
	2	FF		
	3	08		
	4	2B	DCX H	
	5	7C	MOV A,H	
	6	FE	CPI 00	
	7	00		
	8	C2	JNZ	
	9	54		
	A	04		
	B	05	DCR B	
	C	78	MOV A,B	
	D	FE	CPI 00	
	E	00		
	F	C8	RZ	
	60	C3	JMP	
	1	51		
	2	04		
	04A0	00		
	1	01		
	2	03		
	3	02		

Hardwaresteuerung des MP-Systems

Unter MP-System soll in diesem Zusammenhang die CPU, der Clock-Generator und der System-Controller zu verstehen sein.
(Siehe auch Schaltungsauszug Seite 57)

Die zur Programmablaufsteuerung erforderlichen Signale werden zum Teil über den Clock-Generator geführt, um eine Synchronisierung mit dem Taktsignal zu erreichen.

1.) RESET

Das RESET-Signal für die CPU wird dem 8o8o am Pin 12 zugeführt und ist am Anschluß 79 der linken Steckerleiste messbar. es kann auch verwendet werden, um Peripheriegeräte in den Reset-Vorgang mit einzubeziehen(Fan out = 1 1/4 LE).

Die Reset-Taste ist an RES IN angeschlossen und arbeitet als Schließkontakt nach Masse. Die Funktionen eines Reset-Vorganges sind folgende:

a) alle Funktionen unterdrücken
b) PC löschen
c) Befehls-Register löschen
d) Interrupt-Controll-FF löschen (INTE = O)
e) Hold-Acknowledge-FF löschen (HLDA = O)
f) u.U. HALT-Zustand beenden
g) nach Ende des Reset: Instruction Fetch aus Adresse 0000

2.) HOLD

Der HOLD-Eingang der CPU (Pin 13) wird durch einen vorgeschalteten Inverter auf "O" gehalten. Der Systemeingang muß daher wieder mit HOLD bezeichnet werden.

HOLD-Eingang = "0" löst folgende Wirkungen aus:

a) Der laufende Maschinenzyklus wird beendet
b) Der HOLD-Zustand wird durch HLDA (hold acknowledge) quitiert
c) Daten- und Adressbus werden in den Hoch-Impedanz-Zustand geschaltet
d) Der Prozessor "wartet" bis der HOLD-Eingang wieder High wird

Der HOLD-Zustand kann z.B. für DMA (direct memory access) ausgenutzt werden. Die Kontrolle von Daten- und Adressbus wird dann durch periphere Geräte ausgeübt.

3.) READY

Der Eingang RDY IN (ready in) wird ebenfalls über den Clock-Generator synchronisiert. Ein Low-Signal an RDY IN löst folgende Vorgänge aus:

a) Ganzzahlige Vielfache von Wartezyklen (wait cycles) werden in den Befehlsablauf eingeschoben bis RDY IN wieder in den High-Zustand geht
b) Der WAIT-Zustand wird vom Prozessor durch das Signal WAIT quitiert

Wait-Zyklen können benutzt werden, wenn mit langsamer Peripherie gearbeitet wird, oder ausgelöste Vorgänge vor der weiteren Bearbeitung des Programms geprüft werden sollen.

Wichtig: Daten- und Adressbus bleiben aktiv!

4.) HALT

Der HALT-Zustand des Prozessors kann zwar nicht durch ein Signal (sondern nur durch ausführen des HALT-Befehls) hervorgerufen werden, er läßt sich aber nur durch Signale beenden oder unterbrechen.

Zum Unterbrechen oder Beenden des HALT-Zustandes gibt es folgende Möglichkeiten:

a) Ein Interrupt beendet den Halt-Zustand und löst die Bearbeitung einer Interrupt-Service-Routine aus
b) RESET kann den Halt-Zustand beenden und einen neuen Programmstart auslösen
c) Mit dem HOLD-Signal können Daten-und Adressbus in den Hochohmigen Zustand gebracht werden. Der Halt-Zustand wird dadurch jedoch nur unterbrochen und nicht beendet.

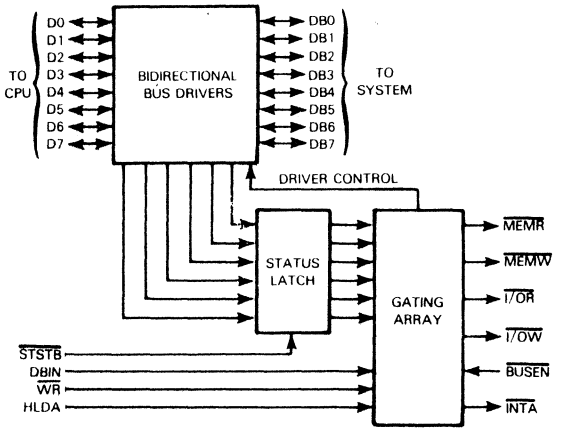
System Controller 8228

Der Einsatz eines System-Controllers vereinfacht den Aufbau eines MP-Systems, da in diesem Baustein Funktionen enthalten sind, die sonst durch Hardware realisiert werden müßten.

Die wesentlichen Funktionen des 8228 sind:

1. Aufbereiten der Steuersignale zum Lesen und Schreiben von Speicherplätzen und I/O-Ports.
2. Bidirektionales Puffern des Datenbus

Weitere Funktionen dienen zur Vereinfachung der Interruptbehandlung und zur externen Datenbussteuerung.



Zu 1.) Die CPU benutzt zu bestimmten Zeiten den Datenbus um Informationen über ihren aktuellen Zustand (processor status) nach "draußen" zu geben. Diese Informationen werden durch das Signal STSTB = status strobe synchronisiert. Da die Informationen bis zur nächsten Statusausgabe zur Verfügung stehen müssen, werden sie in Speichern (status latch) aufgefangen. Eine Codierschaltung (gating array) sorgt dann dafür, daß folgende Steuersignale entstehen:

$\overline{\text{MEMR}}$ = memory read
wird low, wenn der Inhalt eines Speicherplatzes auf den Datenbus gegeben werden soll

$\overline{\text{MEMW}}$ = memory write
wird low, wenn Daten vom Datenbus in einen Speicherplatz geschrieben werden sollen

$\text{I}\overline{/\text{OR}}$ = input/output read
wird low, wenn die an einem Input-Port liegenden Daten gelesen werden sollen

$\overline{\text{I/OW}}$ = input/output write
wird low, wenn Daten über ein Output-Port ausgegeben werden sollen

Zu 2.) Die bidirektionalen Bus-Treiber trennen die Datenbusse von System und Prozessor. Für die Datentransfers durch den Systemcontroller ergeben sich drei Möglichkeiten

1. Daten vom System-Bus zum Prozessor
2. Daten vom Prozessor zum System-Bus
3. Daten vom Prozessor in die Status-Latches

Die weiteren Funktionen sind:

Interruptunterstützung:

Der System-Controller erzeugt das Signal $\overline{\text{INTA}}$ als Interruptquittung. Wird der INTA-Ausgang jedoch über 1kOhm an + 12V gelegt entsteht bei einem Interrupt automatisch ein RST 7-Befehl.

Datenbussteuerung:

$\overline{\text{BUSEN}}$ = bus enable kann benutzt werden, um den Datenbus des Systems in den Hochimpedanz-Zustand zu bringen.

ITT Fachlehrgänge

Input/Output-Ports 8212

Der Baustein 8212 kann sowohl Input- wie auch Output-Port sein.
Im Baustein enthalten sind folgende Funktionsgruppen:

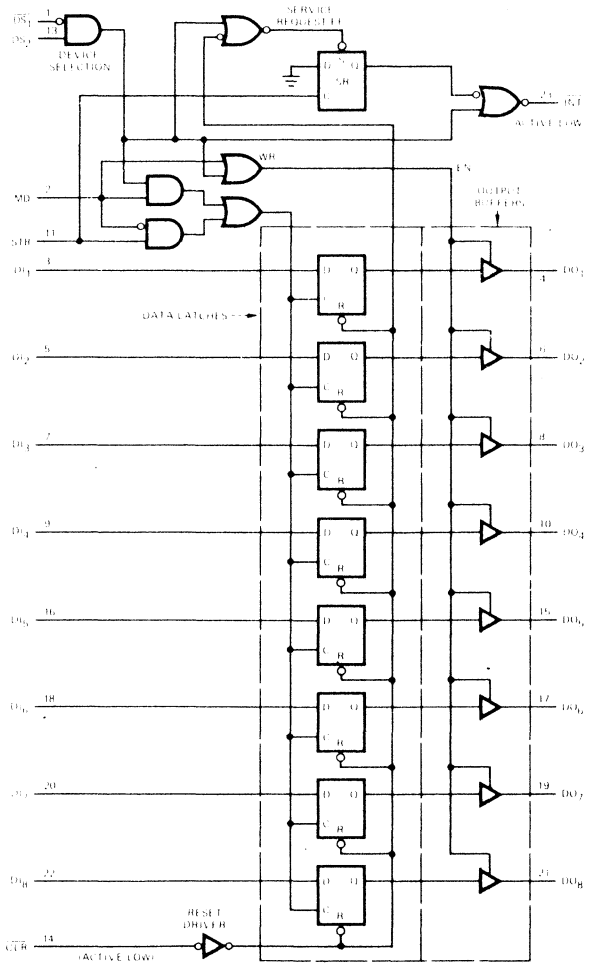
1.) Acht D-FF mit gemeinsamem Löscheingang $\overline{CLR}$

2.) Die Ausgänge aller FF sind über Three-State-Treiber geführt

3.) Zur Interruptverarbeitung ist eine Verknüpfungslogik und ein FF zum Speichern einer Interruptanforderung enthalten

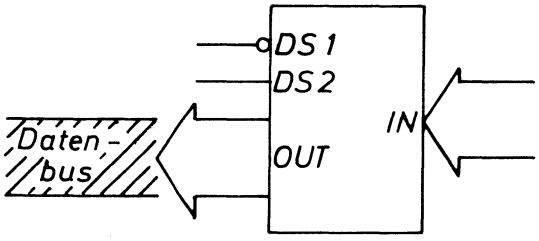
4.) Verknüpfungslogik zur Wahl der Betriebsart (MD = mode) und zur Bausteinadressierung (DS = device select)

Das nachfolgende Logik-Diagramm
zeigt die Innenschaltung des 8212.



ITT Fachlehrgänge

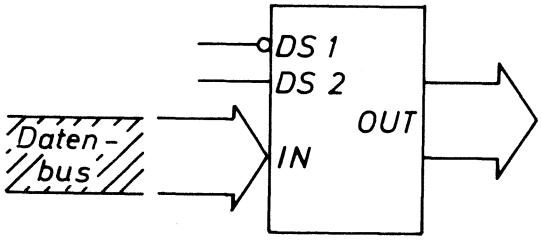
Input-Port-Funktion:



Bei dieser Betriebsart dürfen die Treiber zum Bus hin nur dann getaktet (aktiv geschaltet) werden,wenn:
a) der Baustein adressiert ist und
b) vom Prozessor das Signal I/OR kommt.

Die Adresse und das Signal I/OR werden an DS 1 und DS 2 gelegt und im Baustein verknüpft. Der MD-Eingang muß dabei ständig an "0" liegen.

Output-Port-Funktion:



In dieser Betriebsart sind die Ausgangstreiber ständig aktiv. Daten vom System werden nur dann in die FF's hineingetaktet, wenn:
a) der Baustein adressiert ist und
b) vom Prozessor das Signal I/OW kommt.

Der MD-Eingang liegt jetzt ständig an "1". Der Eingang STB = strobe wird im Lehrsystem nicht benutzt und liegt ständig an "1".

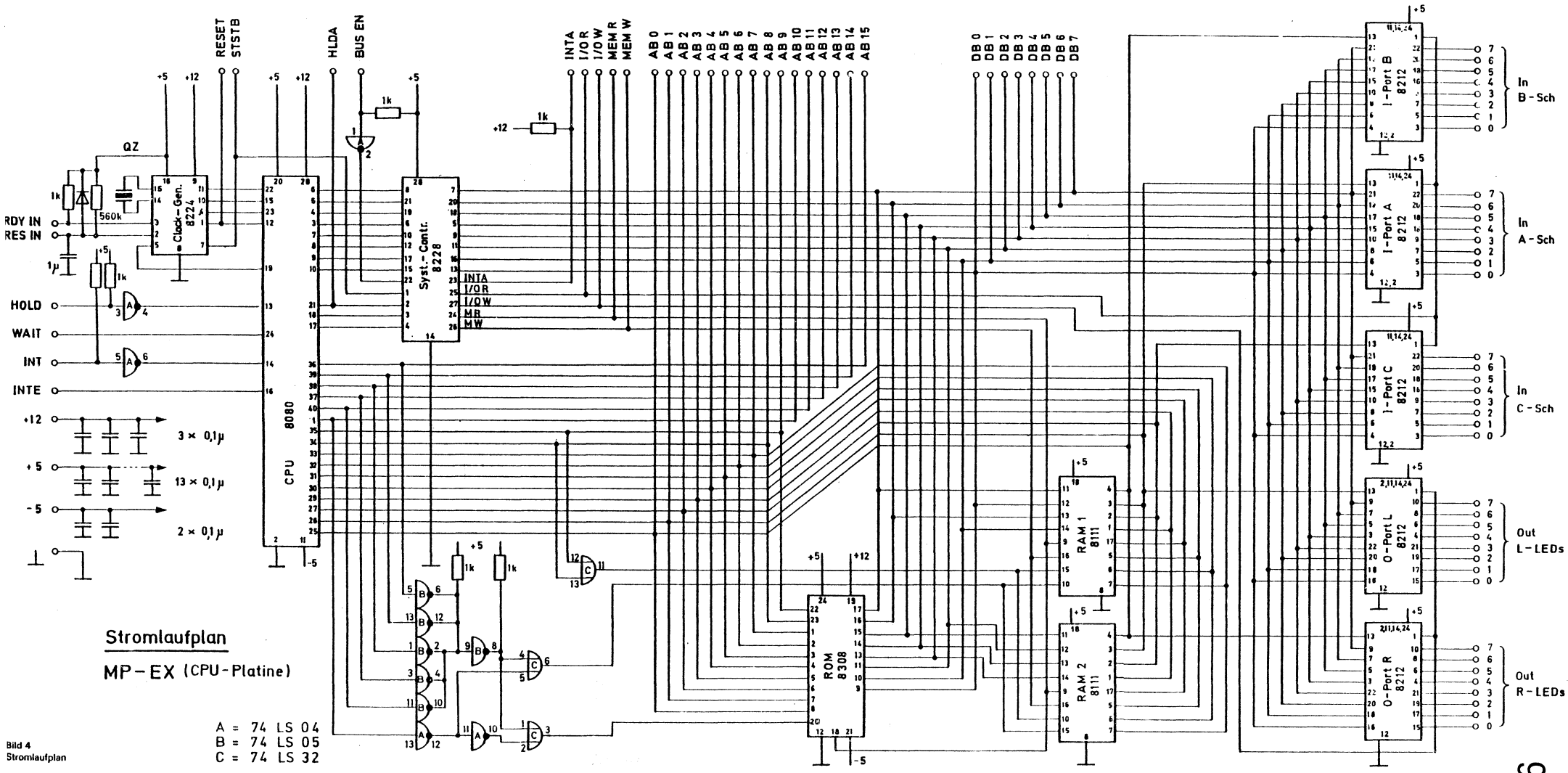
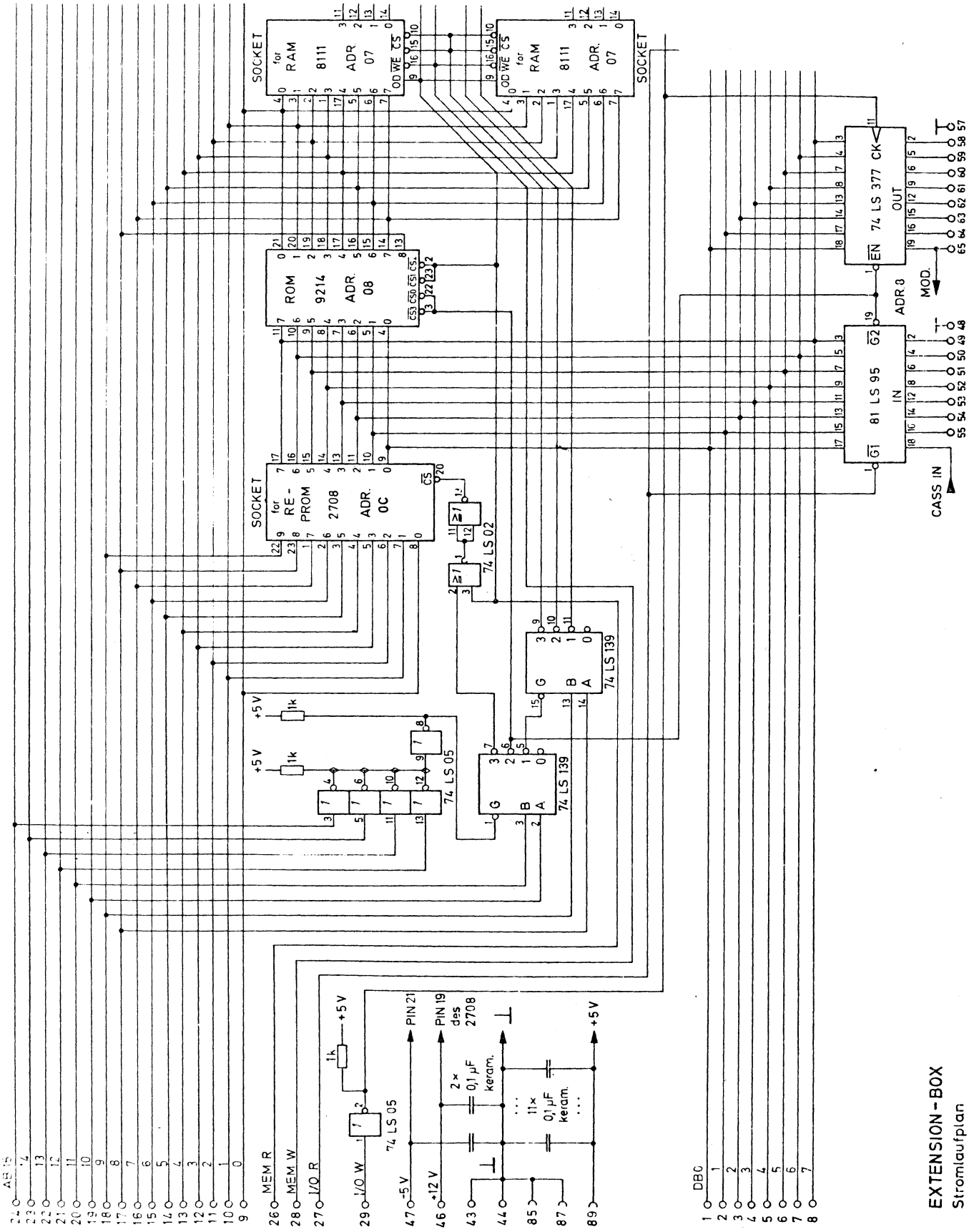


Bild 4
Stromlaufplan



EXTENSION-BOX

Stromlaufplan