Befehl	Maschinen-Code OP-Code Adressierung	Byte-Anzahl	Funktion	Flags N Z C
HALT	0 0 0 0 0 0 0 0	1	hält Rechner an	– – –
NOP	0 0 0 0 1 1 1 1	1	keine Operation	– – –
MOVE	0 0 0 0 s s d d	1	(s s)→d d	– – –
ADDR	0 0 0 1 s s d d	1	(s s) + (d d) →d d	↕ ↕ ↕
SUBR	0 0 1 0 s s d d	1	(d d) – (s s) →d d	↕ ↕ ↕
IORR	0 0 1 1 s s d d	1	(s s) V (d d) →d d	↕ ↕ 0
XORR	0 1 0 0 s s d d	1	(s s) ∀ (d d) →d d	↕ ↕ 0
ANDR	0 1 0 1 s s d d	1	(s s) Λ (d d) →d d	↕ ↕ 0
R . . .	0 1 1 0 e e s s	1	(f (s s))→s s	*)
STAC	0 1 1 1 m m s s	1/2	(s s) →(m m)	– – –
LOAD	1 0 0 0 m m d d	1/2	(m m)→d d	– – –
ADDM	1 0 0 1 m m d d	1/2	(m m) + (d d)→d d	↕ ↕ ↕
SUBM	1 0 1 0 m m d d	1/2	(d d) – (m m)→d d	↕ ↕ ↕
IORM	1 0 1 1 m m d d	1/2	(m m) V (d d)→d d	↕ ↕ 0
XORM	1 1 0 0 m m d d	1/2	(m m) ∀ (d d)→d d	↕ ↕ 0
ANDM	1 1 0 1 m m d d	1/2	(m m) Λ (d d)→d d	↕ ↕ 0
JMP . . .	1 1 1 0 m m c c	1/2	(m m)→PC Sprung	– – –
CAL . . .	1 1 1 1 m m c c	1/2	(PC)→(R3), (m m)→PC	– – –

Register-Adresse

s s/d d	Register
0 0	R0
0 1	R1
1 0	R2
1 1	R3

OP-Code-Erweiterung e e *)

e e	Befehl	Reg.-Funktion	N Z C
0 0	INCR	(s s) + 1→s s	↕ ↕ –
0 1	DECR	(s s) – 1→s s	↕ ↕ –
1 0	RACL	ROT s s u. C links	– – ↕
1 1	RACR	ROT s s u. C rechts	– – ↕

– = Flag wird nicht beeinflußt
↕ = Flag wird beeinflußt, wird 0 oder 1
0 = Flag wird auf 0 gesetzt

Adreßmode m m

Befehls-gruppe	m m = 0 0	m m = 0 1	m m = 1 0	m m = 1 1
LOAD ADDM SUBM IORM XORM ANDM	# IMMEDIATE 2-Byte-Befehl 2. Byte = Daten	ABSOLUT oder DIREKT 2-Byte-Befehl 2. Byte = Adr.	@ R2 INDEXED über R2 1-Byte-Befehl R2 enth. Adr.	@ R3 ↑ AUTO INCRE- MENT INDEXED über R3 1-Byte-Befehl R3 enth. Adr.
STAC	@↓ R3 INDEXED AUTO DECREM. über R3 1-Byte-Befehl R3 enth. Adr.	ABSOLUT oder DIREKT 2-Byte-Befehl 2. Byte = Adr.	@ R2 INDEXED über R2 1-Byte-Befehl R2 enth. Adr.	@ R3 ↑ AUTO INCRE- MENT INDEXED über R3 1-Byte-Befehl R3 enth. Adr.

Fortsetzung der Tabelle auf Seite 14!

JMP . . .	ABSOLUT oder DIREKT 2-Byte-Befehl 2. Byte = Spr.-Adr.	@ INDIREKT 2-Byte-Befehl 2. Byte = Adr. d. Spr.-Adr.	@ R2 INDEXED über R2 1-Byte-Befehl R2 enth. Spr.-Adr.	@ R3 ↑ AUTO INCRE- MENT INDEXED über R3 1-Byte-Befehl R3 enth. Spr.-Adr.
CAL . . .	ABSOLUT oder DIREKT 2-Byte-Befehl 2. Byte = Spr.-Adr.	@ INDIREKT 2-Byte-Befehl 2. Byte = Adr. d. Spr.-Adr.	@ R2 INDEXED über R2 1-Byte-Befehl R2 enth. Spr.-Adr.	nicht verwendet

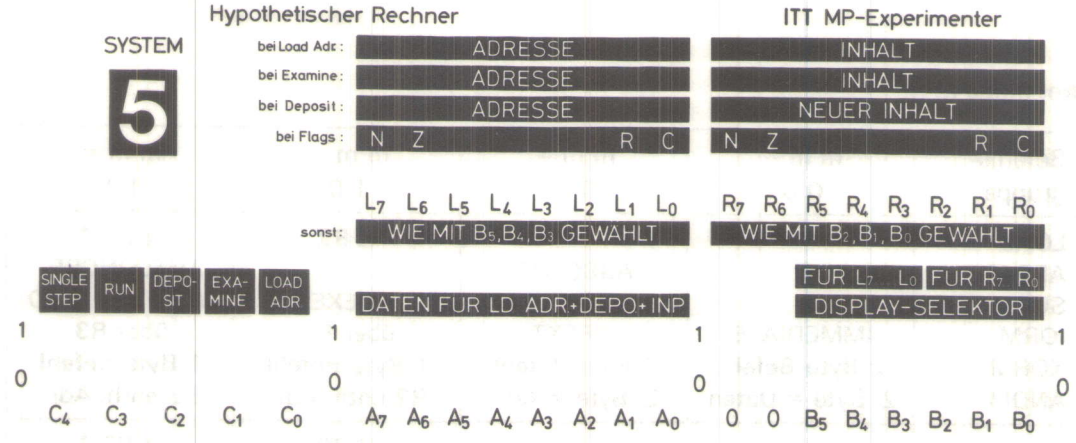
Sprung-Bedingungen c c

c c	Befehl	Sprung- u. Aufrufbedingung
0 0	JUMP, CALL	unbedingt
0 1	. . . Z	falls Z-Flag = 1
1 0	. . . N	falls N-Flag = 1
1 1	. . . C	falls C-Flug = 1

Der Befehl „R . . ." (Registermanipulation) hat noch eine 2-bit-Op-Code-Erweiterung mit dem Code „e e". Damit gibt es hier 4 Befehle, wie die Tabelle zeigt.
Auch bei den Befehlen „JMP . . ." und „CAL . . ." wird der Op-Code mit den beiden Bedingungs-bits „c c" erweitert, so daß jeweils 4 verschiedene Sprungbefehle entstehen.

Mit der Adressierung „s s" bzw. „d d" wird je eins von den 4 vorhandenen Registern zum Quellregister (source, Code s s) bzw. zum Zielregister (destination, Code d d) ernannt.

Die Codierung „m m" legt den eigentlichen Adressierungsmode entsprechend der Tabelle fest. Damit können alle Befehle, die m m enthalten, je nach Wahl des Modes, 1- oder 2-Byte-Befehle werden.
Alle über R2 oder R3 indexierten Adressierungen sind 1-Byte-Befehle, alle immediate, absoluten (direkten) und indirekten ergeben 2-Byte-Befehle.



Bedeutung der Schalter:
Die Schaltergruppe C_4 bis C_0 hat die gleiche Bedeutung wie im System 4. Wie aus der aufgelegten Schablone zu erkennen ist, wird bei LOAD-ADR. = 1 die an den Schaltern A_7 bis A_0 eingestellte Adresse angewählt. Solange C_0 = 1 ist, erscheint in den LEDs L_7 bis L_0 die angewählte Adresse, der Inhalt dieser Adresse wird in R_7 bis R_0 angezeigt. Das gleiche gilt für die Betätigung von EXAMINE und DEPOSIT. Bei DEPOSIT = 1 erscheint in R_7 bis R_0 der mit dem Takt abgespeicherte neue Adresseninhalt. Werden die erwähnten Schalter wieder

in die Stellung 0 zurückgesetzt, erscheinen in den beiden Anzeigenreihen die Daten, die über die Schalter A_5 bis A_0 abgerufen werden. Hierbei kann mit den Schaltern B_5 bis B_3 die Anzeige L_7 bis L_0 und mit den Schaltern B_2 bis B_0 die Anzeige R_7 bis R_0 angewählt werden.

Die Schalter B_7 und B_6 haben keine Bedeutung. Bei gleicher Einstellung von B_5 bis B_3 und B_2 bis B_0 erscheinen in R_7 bis R_0 und L_7 bis L_0 die gleichen Daten. Der Anzeigecode (Display-Code) spezifiziert die einzelnen Anzeigemöglichkeiten:

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

Die Codes 0 0 0 bis 0 1 1 bedürfen keiner weiteren Erklärung.
- Code 1 0 0 zeigt an, welche Adresse vom Befehlszähler gerade angewählt wird.
- Code 1 0 1 gibt Auskunft darüber, welche Zustände die 4 Flags gerade einnehmen.
- Code 1 1 0 gibt den Inhalt der Speicheradresse an, die vom Befehlszähler gerade angewählt ist.
- Code 1 1 1 zeigt den Inhalt einer Speicheradresse an, die mit den Schaltern A_7 bis A_0 frei wählbar ist.

Wesentlich ist, daß die Schalter B_5 bis B_0 keinen Einfluß auf den Funktionsablauf haben. Sie dienen lediglich zur Anzeige bestimmter Daten, wenn ein Programm abläuft.

Damit Sie mit diesen grundsätzlichen Eigenschaften des Rechners vertraut werden, führen Sie nachfolgendes Übungsprogramm Schritt für Schritt durch:

1. Alle Schalter auf Null stellen
2. A_7 bis A_0 auf 4 0_{16} einstellen
3. LOAD-ADR. auf 1 stellen (nicht takten!)

In L_7 bis L_0 erscheint die an A_7 bis A_0 eingestellte Adresse, deren momentaner Inhalt in R_7 bis R_0 angezeigt wird

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

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

1. Alle Schalter auf 0
2. EXAMINE auf 1 stellen. In L_7 bis L_0 erscheint die Adresse 4 0_{16}, in R_7 bis R_0 deren Inhalt 8 4_{16}, also der erste Befehl des Programms. Hierbei handelt es sich um den Befehl LOAD R0, F E_{16} (lade Akkumulator R0 mit dem Inhalt der Adresse F E_{16}). Diese **Datenadresse** ist in der nächsten Programmadresse abgespeichert

3. EXAMINE über 0 wieder auf 1 stellen. In L_7 bis L_0 erscheint die Adresse 4 1_{16}, der Inhalt F E_{16} erscheint in R_7 bis R_0
4. Punkt 3. wiederholen. In L_7 bis L_0 erscheint Programmadresse 4 2_{16}. Der Inhalt dieser Adresse 0 1_{16} (Anzeige in R_7 bis R_0) entspricht dem Befehl MOVE R1, R0. Dieser Befehl bewirkt, daß die Daten aus R0 in R1 transferiert werden
5. Punkt 4. wiederholen. In L_7 bis L_0 erscheint Adresse 4 3_{16}. Der Inhalt 1 1_{16} entspricht dem Befehl ADDR R1, R0. Hierbei werden die Daten des Registers R0 mit dem Inhalt R1 in R1 addiert
6. Punkt 5. wiederholen. In der Adresse 4 4_{16} ist der Befehl 0 6_{16}, d.h. MOVE R2, R1, abgespeichert. Hierbei werden die Daten von R1 in R2 gebracht
7. Punkt 6. wiederholen. In der Adresse 4 5_{16} ist der HALT-Befehl abgespeichert

Damit dieses Programm mit definierten Daten ablaufen kann, speichern Sie in Adresse F E_{16} die Zahl 1 0_{16}:
1. A_7 bis A_0 auf F E_{16} einstellen
2. Schalter LOAD-ADR. takten
3. A_7 bis A_0 auf 1 0_{16} einstellen
4. DEPOSIT takten

Jetzt stellen Sie über LOAD-ADR. das System wieder auf die Programmadresse 4 0_{16} zurück und schalten über C_4 SINGLE-STEP-Betrieb ein.

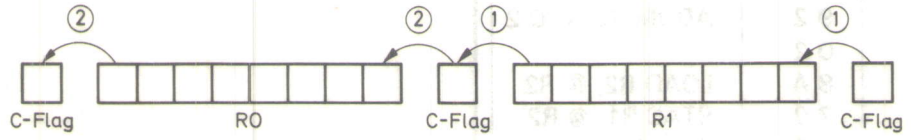
Adresse (hexadez.)	Inhalt Maschinencode	Befehl	Kommentar
4 0	1 0 0 0 0 1 0 0	LOAD R0, F E	
4 1	1 1 1 1 1 1 1 0		Datenadresse
4 2	0 0 0 0 0 0 0 1	MOVE R1, R0	
4 3	0 0 0 1 0 0 0 1	ADDR R1, R0	
4 4	0 0 0 0 0 1 1 0	MOVE R2, R1	
4 5	0 0 0 0 0 0 0 0	HLT	
.			
.			
.			
F E	0 0 0 1 0 0 0 0	Daten	

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

Schalter B_5 bis B_3 auf 0 1 0 stellen und mit RUN takten. Die Daten von R1 erscheinen auch in R2. Weiteres Takten hat keinen Einfluß mehr auf das System, da in Adresse 4 5_{16} ein HALT-Befehl programmiert ist.
Wie bereits erwähnt, ist im Adreßbereich 0 0_{16} bis 1 8_{16} des ROMs ein Betriebsprogramm untergebracht. Dieses Betriebsprogramm ist ein Multiplikationsprogramm, das als Unterprogramm benutzt werden kann:

Adresse	Inhalt	Befehl	Kommentar	
0 1	7 5	STAC R1, F 0	speichere Multiplikand	1)
0 2	F 0			
0 3	4 5	XORR R1, R1	lösche Zwischensumme	
0 4	8 2	LOAD R2, # 0 8	setze Schleifenzähler	
0 5	0 8			
0 6	6 9	RACL R1	verschiebe rechte Hälfte Zwischensumme	2)
0 7	6 8	RACL R0	verschiebe linke Hälfte Zwischensumme	
0 8	E 3	JMPC 0 C	prüfe Carry-Flag	3)
0 9	0 C			
0 A	E 0	JUMP 1 4		
0 B	1 4			
0 C	9 5	ADDM R1, F 0	C-Flag war 1, addiere	
0 D	F 0			
0 E	E 3	JMPC 1 2		4)
0 F	1 2			
1 0	E 0	JUMP 1 4		
1 1	1 4			
1 2	9 0	ADDM R0, # 0 1	C-Flag war 1, addiere	5)
1 3	0 1			
1 4	6 6	DECR R2	erniedrige Schleifenzähler	
1 5	E 1	JMPZ 1 9	Schleifenende?	
1 6	1 9			
1 7	E 0	JUMP 0 6		
1 8	0 6			

1. Der Multiplikand steht in R1. Dieses Register wird später für die rechte Hälfte der Zwischensumme benötigt. Aus diesem Grunde muß der Multiplikand abgespeichert werden. Dies geschieht in unserem Beispiel unter der Adresse F 0.
2. Die Verschiebung der Zwischensumme erfolgt in 2 Schritten. Zuerst wird die rechte Hälfte (R1) verschoben. Das herausgeschobene bit kommt in das Carry-Flag. Danach wird die linke Hälfte (R0) verschoben. Der Inhalt des Carry-Flags wird dabei von rechts in R0 hineingeschoben, so daß insgesamt eine Verschiebung mit doppelter Wortlänge durchgeführt wird:



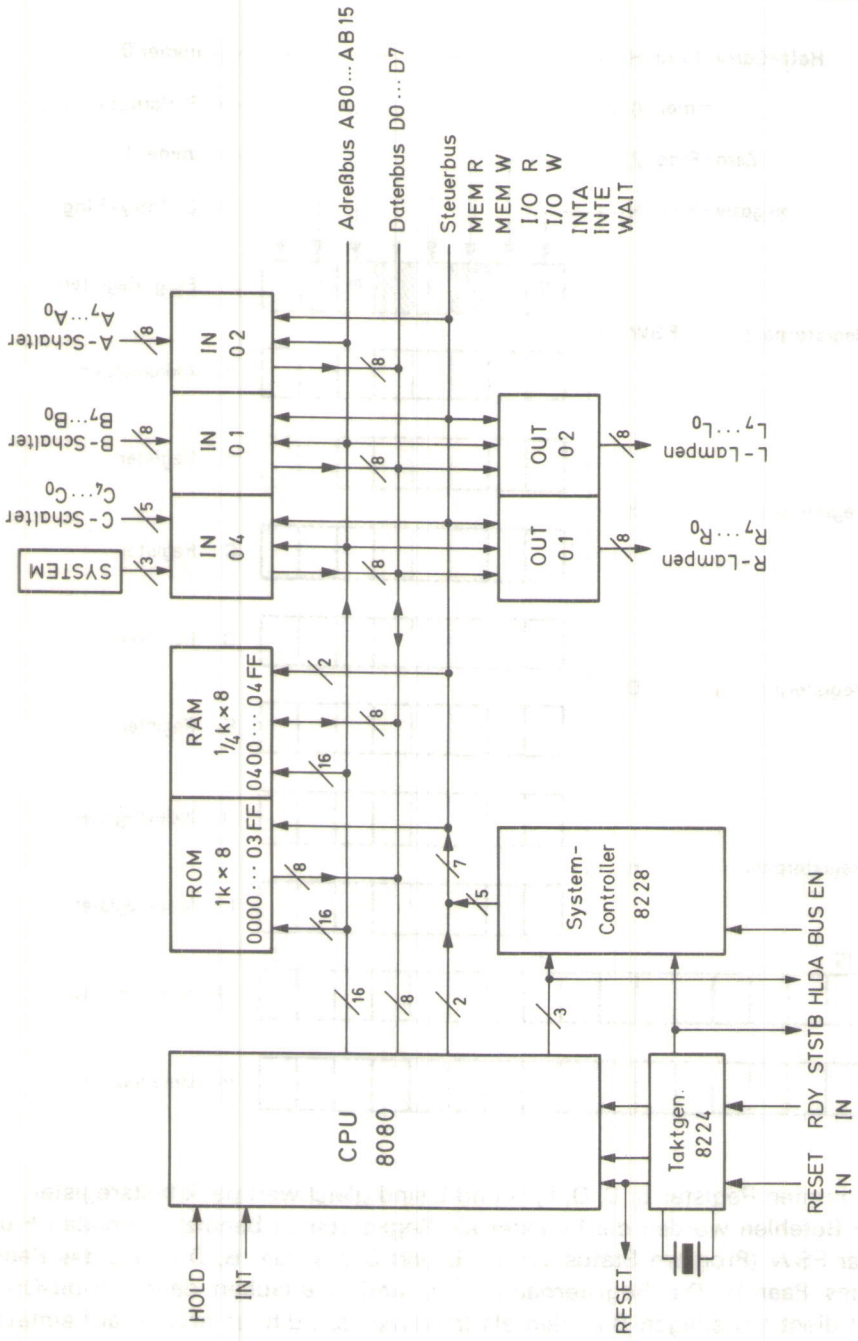
Dabei ist noch zu beachten, daß durch den Befehl RACL 2 1 der Inhalt des Carry-Flags von rechts in R1 hineingeschoben wird. Über das Programm muß deshalb sichergestellt werden, daß vor dieser Instruktion das Carry-Flag 0 enthält. Dies wird beim ersten Programmdurchlauf durch die Instruktion XORR R1, R1 garantiert. Dieser Befehl bewirkt nämlich ein Löschen des Carry-Flags. Auch in der Schleife muß diese Bedingung erfüllt sein.
3. und 4. Die folgenden Programmschritte (Addition) müssen übersprungen werden, wenn das Carry-Flag Null ist. Ein solcher Befehl ist im hypothetischen Rechner nicht vorhanden. Er muß daher durch einen bedingten und unbedingten Sprung realisiert werden, was natürlich zu einem längeren Programm führt.
5. Dieser Befehl addiert 1 zur linken Hälfte der Zwischensumme und löscht gleichzeitig das Carry-Flag, da sich bei dieser Addition kein Übertrag ergeben kann. Damit ist sichergestellt, daß im nächsten Schleifendurchlauf mit dem Befehl RACL R1 eine Null von rechts in die Zwischensumme geschoben wird. Zum Erhöhen von R0 könnte auf den ersten Blick auch der Befehl INCR R0 verwendet werden. Dies ist nicht möglich, da dieser Befehl das Carry-Flag nicht beeinflußt.

Dieses Unterprogramm kann, wie das folgende Beispiel zeigt, in ein Gesamtprogramm für Multiplikationen eingebaut werden:

Adresse	Inhalt	Befehl	Kommentar
4 0	8 3	LOAD R3, # F F	Initialisiere Stack-Pointer
4 1	F F		
4 2	F 0	CALL MULTA	
4 3	2 6		
4 4	5 0	ADR F 1	Adresse von Faktor 1
4 5	5 1	ADR F 2	Adresse von Faktor 2
4 6	5 2	ADR P	Produktadresse
4 7	0 0	HALT	
2 6	0 E	MOVE R2, R3	Unterprogramm
2 7	8 A	LOAD R2, @ R2	
2 8	8 A	LOAD R2, @ R2	
2 9	8 8	LOAD R0, @ R2	
2 A	0 E	MOVE R2, R3	
2 B	8 A	LOAD R2, @ R2	
2 C	6 2	INCR R2	
2 D	8 A	LOAD R2, @ R2	
2 E	8 9	LOAD R1, @ R2	
2 F	E 0	JUMP 0 1	
3 0	0 1	JUMP 0 1	
0 1		Multiplikations-programm im ROM-Bereich	
0 2			
.			
.			
.			
.			
1 7			
1 8			
1 9	0 E	MOVE R2, R3	
1 A	8 A	LOAD R2, @ R2	
1 B	9 2	ADDM R2, @ 0 2	
1 C	0 2		
1 D	8 A	LOAD R2, @ R2	
1 E	7 9	STAC R1, @ R2	
1 F	6 2	INCR R2	
2 0	7 8	STAC R0, @ R2	
2 1	8 E	LOAD R2, @ R3 ↑	
2 2	9 2	ADDM R2, # 0 3	
2 3	0 3		
2 4	7 2	STAC R2, @ ↓ R3	
2 5	E C	JUMP @ R3 ↑	

3.8 Mikrorechner-System 8080 (SYSTEM 6)

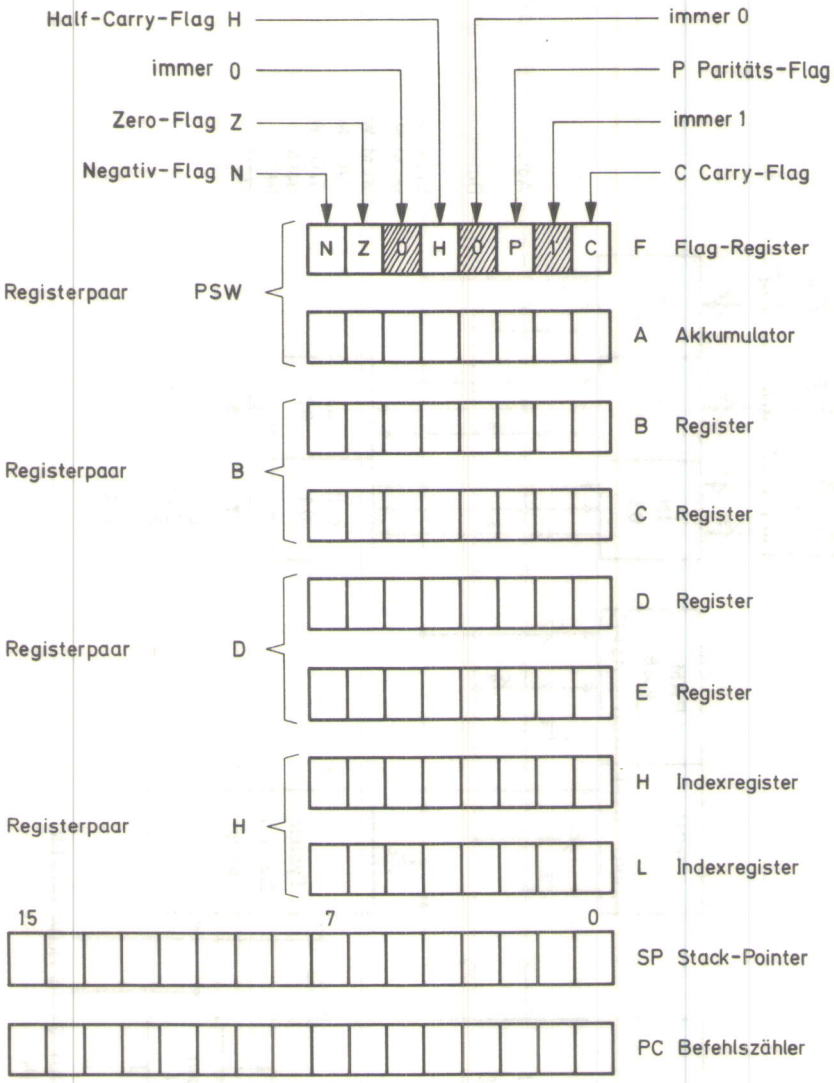
Das nachstehende Blockschaltbild zeigt den Aufbau des Mikrorechners mit der CPU 8080.



Die CPU 8080 mit dem Taktgenerator 8224 und dem System-Controller 8228 liefert alle Adreß-, Daten- und Steuersignale, die auf der linken Anschlußleiste verfügbar sind.
Im 1-k-ROM mit dem Adreßbereich 0 0 0 0_{16} bis 0 3 F F_{16} ist das Monitorprogramm dieses Systems (und der niederen Systeme) untergebracht. Zur Verfügung des Anwenders steht das 1/4-k-RAM im absoluten Adreßbereich 0 4 0 0_{16} bis 0 4 F F_{16}, seine Adressen werden vom Monitor in die relativen Adressen 0 0_{16} bis F F_{16} umgesetzt. Am Datenbus liegen 3 8-bit-Input-Ports und 2 8-bit-Output-Ports, die mit isolierter I/O-Adressierung betrieben werden. Sie werden mit einzelnen Adreßbits adressiert, so daß maximal je 8 Input- bzw. Output-Ports adressiert werden können (Adressen 0 1, 0 2, 0 4, 0 8, 1 0, 2 0, 4 0, 8 0 davon 0 1, 0 2, 0 4 bzw. 0 1, 0 2 verwendet).
Dem Benutzer stehen die im nachstehenden Bild dargestellten Register der CPU zur Verfügung.

Im Flag-Register (F) sind die Flags „N" (Negativ), „Z" (Zero), „H" (Half-Carry für Überträge vom 3. ins 4. bit, Verwendung für BCD-Operationen), „P" (Parität, d.h. bei einer geraden

Anzahl von 1 in einem Register wird P = 1) und „C" (Carry) vorhanden. Die übrigbleibenden 3 bits sind fest mit 0 bzw. 1 belegt. Das 8-bit-A-Register ist das Haupt-Arbeitsregister (Akkumulator), das gegenüber den übrigen Registern mit einigen besonderen Befehlen arbeiten kann.



Die je 8 bit breiten Register B, C, D, E, H und L sind gleichwertige Arbeitsregister.
Bei einigen Befehlen werden die Register als Registerpaare benutzt, so bilden F und A das Registerpaar PSW (Program Status Word), B und C das Paar B, D und E das Paar D sowie H und L das Paar H. Die Registerpaare B, D und H erlauben dann 16-bit-Operationen. Das Paar H dient bei einigen Befehlen als Indexregister, d.h. es erlaubt auf einfache Weise eine Indexadressierung.
Weiterhin steht ein 16-bit-Stack-Pointer (Stapelzeiger) zur Verfügung, der bei Programmsprüngen usw. die Adresse des Speicherplatzes anzeigt, in dem die Rücksprungadresse steht.

Ein 16-bit-Programm-Zähler (PC) dient zur Adressierung von insgesamt 64 k (65 536) möglichen Speicherplätzen.
Das 8080-System hat ein 8-bit-Befehlsformat. Dabei ist insgesamt nicht zwischen Op-Code und Adressierung zu unterscheiden, d.h. man muß alle 8 bits als Op-Code verstehen. Damit ergäben sich theoretisch $2^8 = 256$ Befehle, von denen jedoch 12 Möglichkeiten nicht verwendet werden, so daß 244 Befehle übrigbleiben.
Durch Zusammenfassen zu Befehlsgruppen mit zusätzlicher Codierung (innerhalb der 8 bits) ergibt sich eine übersichtliche Form des Befehlsvorrates.
Aufgrund unterschiedlicher Adressierung können die Befehle 1-Byte-, 2-Byte- oder 3-Byte-Befehle sein. Die folgende Tabelle zeigt alle Befehle in der komprimierten Form der Befehlsliste unter Verwendung zusätzlicher Codierungen.

Befehl	Maschinen-Code	Byte	Funktion	Flags N	Z	H	P	C
NOP	0 0 0 0 0 0 0 0	1	keine Operation	–	–	–	–	–
MOV d, s	0 1 d d d s s s	1	(s s s) →d d d	–	–	–	–	–
MVI d, # k.	0 0 d d d 1 1 0	2	konst. →d d d	–	–	–	–	–
LDA adr.	0 0 1 1 1 0 1 0	3	(adr.) →A	–	–	–	–	–
STA adr.	0 0 1 1 0 0 1 0	3	(A) →adr.	–	–	–	–	–
LDAX rp*	0 0 r r 1 0 1 0	1	(@rp) →A	–	–	–	–	–
STAX rp*	0 0 r r 0 0 1 0	1	(A) →@rp	–	–	–	–	–
LHLD adr.	0 0 1 0 1 0 1 0	3	(adr.) →HL	–	–	–	–	–
SHLD adr.	0 0 1 0 0 0 1 0	3	(HL) →adr.	–	–	–	–	–
LXI rp, # k	0 0 r r 0 0 0 1	3	konst. →rp	–	–	–	–	–
XCHG	1 1 1 0 1 0 1 1	1	(HL) ↔(DE)	–	–	–	–	–
IN A, adr.	1 1 0 1 1 0 1 1	2	(adr.) →A	–	–	–	–	–
OUT adr., A	1 1 0 1 0 0 1 1	2	(A) →adr.	–	–	–	–	–
ADD A, s	1 0 0 0 0 s s s	1	(s s s) + (A) →A	↕	↕	↕	↕	↕
ADC A, s	1 0 0 0 1 s s s	1	(s s s) + (A) + (C) →A	↕	↕	↕	↕	↕
ADI A, # k	1 1 0 0 0 1 1 0	2	(A) + konst. →A	↕	↕	↕	↕	↕
ACI A, # k	1 1 0 0 1 1 1 0	2	(A) + konst. + (C) →A	↕	↕	↕	↕	↕
SUB A, s	1 0 0 1 0 s s s	1	(A) − (s s s) →A	↕	↕	↕	↕	↕
SBB A, s	1 0 0 1 1 s s s	1	(A) − (s s s) − (C) →A	↕	↕	↕	↕	↕
SUI A, # k	1 1 0 1 0 1 1 0	2	(A) − konst. →A	↕	↕	↕	↕	↕
SBI A, # k	1 1 0 1 1 1 1 0	2	(A) − konst. − (C) →A	↕	↕	↕	↕	↕
CMP A, s	1 0 1 1 1 s s s	1	(A) − (s s s)	↕	↕	↕	↕	↕
CPI A, # k	1 1 1 1 1 1 1 0	2	(A) − konst.	↕	↕	↕	↕	↕
DAD H, rp	0 0 r r 1 0 0 1	1	(HL) + (rp) →HL	–	–	–	–	↕
DAA A	0 0 1 0 0 1 1 1	1	(A) + korr. →A	↕	↕	↕	↕	↕
ANA A, s	1 0 1 0 0 s s s	1	(A) ∧ (s s s) →A	↕	↕	↕	↕	0
ANI A, # k	1 1 1 0 0 1 1 0	2	(A) ∧ konst. →A	↕	↕	↕	↕	0
ORA A, s	1 0 1 1 0 s s s	1	(A) ∨ (s s s) →A	↕	↕	0	↕	0
ORI A, # k	1 1 1 1 0 1 1 0	2	(A) ∨ konst. →A	↕	↕	0	↕	0
XRA A, s	1 0 1 0 1 s s s	1	(A) ∀ (s s s) →A	↕	↕	0	↕	0
XRI A, # k	1 1 1 0 1 1 1 0	2	(A) ∀ konst. →A	↕	↕	0	↕	0
CMA A, $\overline{A}$	0 0 1 0 1 1 1 1	1	($\overline{A}$) →A	–	–	–	–	–
INR d	0 0 d d d 1 0 0	1	(d d d) + 1 →d d d	↕	↕	↕	↕	–
DCR d	0 0 d d d 1 0 1	1	(d d d) − 1 →d d d	↕	↕	↕	↕	–
INX rp	0 0 r r 0 0 1 1	1	(rp) + 1 →rp	–	–	–	–	–
DCX rp	0 0 r r 1 0 1 1	1	(rp) − 1 →rp	–	–	–	–	–
R . . . A	0 0 0 n n 1 1 1	1	(A_n) →A	–	–	–	–	↕
PCHL	1 1 1 0 1 0 0 1	1	(HL) →PC	–	–	–	–	–
JMP adr.	1 1 0 0 0 0 1 1	3	(adr.) →PC	–	–	–	–	–
J . . . adr.	1 1 b b b 0 1 0	3	(adr.) →PC, wenn	–	–	–	–	–
CALL adr.	1 1 0 0 1 1 0 1	3	(adr.) →PC, (PC) →SP	–	–	–	–	–
C . . . adr.	1 1 b b b 1 0 0	3	(adr.) →PC, wenn (PC) →SP	–	–	–	–	–
RET	1 1 0 0 1 0 0 1	1	((SP)) →PC	–	–	–	–	–
R . . .	1 1 b b b 0 0 0	1	((SP)) →PC, wenn	–	–	–	–	–
RST a	1 1 a a a 1 1 1	1	8 x a →PC	–	–	–	–	–
POP**	1 1 r r 0 0 0 1	1	((SP)) →rp	(↕	↕	↕	↕	↕)
PUSW**	1 1 r r 0 1 0 1	1	(rp) →(SP)	–	–	–	–	–
SPHL	1 1 1 1 1 0 0 1	1	(HL) →SP	–	–	–	–	–
XTHL	1 1 1 0 0 0 1 1	1	((SP)) ↔(HL)	–	–	–	–	–
EI	1 1 1 1 1 0 1 1	1	Interrupt ermöglicht	–	–	–	–	–
DI	1 1 1 1 0 0 1 1	1	Interrupt gesperrt	–	–	–	–	–
STC	0 0 1 1 0 1 1 1	1	1 →C	–	–	–	–	1
CMC	0 0 1 1 1 1 1 1	1	($\overline{C}$) →C	–	–	–	–	↕
HLT	0 1 1 1 0 1 1 0	1	Prozessor hält an	–	–	–	–	–

Die verwendeten zusätzlichen Codierungen haben dabei folgende Bedeutungen:

Register-Code			
s s s od. d d d	Register	r r	Reg.-Paar rp
0 0 0	B	0 0	BC = B
0 0 1	C	0 1	DE = D
0 1 0	D	1 0	HL = H
0 1 1	E	1 1	SP
1 0 0	H		
1 0 1	L	*	nur 0 0 und 0 1
1 1 0	M = ((HL))	**	1 1 für PSW dann bei POP Flags ↕
1 1 1	A		

s s s = Quellregister
d d d = Zielregister
M = Speicherplatz
↕ = Flag wird beeinflußt
– = Flag wird nicht beeinflußt
1,0 = Flag wird 1,0

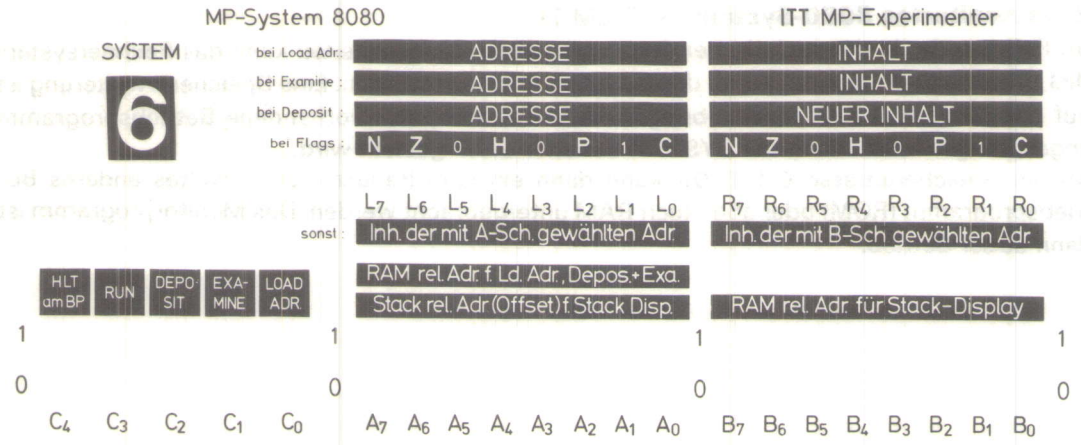
Rotate-Code (R . . .) n n		
n n	Befehl	Funktion
0 0	RLC	$(A_7)\rightarrow A_0$, $(A_7)\rightarrow C$
0 1	RRC	$(A_0)\rightarrow A_7$, $(A_0)\rightarrow C$
1 0	RAL	$(A_7)\rightarrow C$, $(C)\rightarrow A_0$
1 1	RAR	$(A_0)\rightarrow C$, $(C)\rightarrow A_7$

Sprung-Bedingungs-Code b b b		
b b b	Befehlszusatz	Bedingung
0 0 0	. . . NZ	wenn Z = 0, d.h. Inhalt ≠ 0
0 0 1	. . . Z	wenn Z = 1, d.h. Inhalt = 0
0 1 0	. . . NC	wenn C = 0, d.h. kein Übertrag
0 1 1	. . . C	wenn C = 1, d.h. Übertrag
1 0 0	. . . PO	wenn P = 0, d.h. Parität ungerade
1 0 1	. . . PE	wenn P = 1, d.h. Parität gerade
1 1 0	. . . P	wenn N = 0, d.h. Inhalt positiv
1 1 1	. . . M	wenn N = 1, d.h. Inhalt negativ

Für die genaue Kenntnis der Wirkungsweise der einzelnen Befehle sollte man die Datenunterlagen der Hersteller des 8080 oder das schriftliche Lehrmaterial des ITT MP-Lehrsystems zu Rate ziehen.

Um nun mit dem Mikrorechner 8080 arbeiten zu können, ist ein Monitorprogramm erforderlich, über das man Einblick in den Zustand bzw. die Arbeitsweise nehmen kann. Das Monitorprogramm für das 8080-System beim ITT MP-Experimenter ist ein für den Lernzweck mit diesem Gerät speziell zugeschnittenes Programm und nicht identisch mit den Monitorprogrammen der 8080-Hersteller.
Im einzelnen erlaubt dieses Programm:
– die Registerinhalte des 8080, den Stand des Befehlszählers und die Inhalte aller RAM-Adressen sichtbar zu machen
– das RAM zu laden (Funktion DEPOSIT) und den RAM-Inhalt automatisch zu kontrollieren (Funktion EXAMINE)

Das Monitorprogramm ist im ROM untergebracht und kann durch Betätigung der RESET-Taste gestartet werden.
Es wird außer Betrieb gesetzt, d.h. das Anwenderprogramm im RAM wird vom 8080 bearbeitet, wenn der RUN-Schalter (C_3) auf 1 gestellt wird.
Wenn RUN = 1 ist, ist der Monitor außer Betrieb und alle übrigen Schalter haben keinen sichtbaren Einfluß mehr. Auch nach dem Rückschalten von RUN auf 0 bleibt das Anwenderprogramm in Betrieb, d.h. das Monitorprogramm läuft nicht. Das Monitorprogramm ist nur bei RUN = 0 durch Betätigung der RESET-Taste wieder in Betrieb zu setzen.



Solange der Monitor in Betrieb ist, können die Inhalte der Register und der Stand des Befehlszählers zur Anzeige gebracht werden. Der Monitor setzt die echten (absoluten) Adressen des RAM-Bereiches 0 4 0 0_{16} bis 0 4 F F_{16} in relative Adressen – RAM-selektive Adressen – um, die dann 0 0_{16} bis F F_{16} lauten. Die Registerinhalte werden vom Monitor in den STACK-Teil des RAMs gebracht, der die RAM-rel. Adressen F 4_{16} bis F D_{16} belegt. Wählt man diese Adressen mit den B-Schaltern, so werden die Register angezeigt. Dadurch, daß die Adressen des STACK festgelegt sind, kann man die Inhalte der Register mit Hilfe der A-Schalter auch über die STACK-relativen Adressen (STACK-Offset) von 0 0_{16} bis 0 9_{16} (entspricht F 4_{16} bis F D_{16}) zur Anzeige bringen. Die nachstehende Tabelle gibt die Codierung an:

abs. Adresse	RAM-rel. Adresse B-Schalter Anz. in R-LEDs	STACK-rel. Adresse A-Schalter Anz. in L-LEDs	Inhalt
0 4 F 4	F 4	0 0	Reg. L
0 4 F 5	F 5	0 1	Reg. H
0 4 F 6	F 6	0 2	Reg. E
0 4 F 7	F 7	0 3	Reg. D
0 4 F 8	F 8	0 4	Reg. C
0 4 F 9	F 9	0 5	Reg. B
0 4 F A	F A	0 6	Flags
0 4 F B	F B	0 7	Akku
0 4 F C	F C	0 8	PC (bit 7 bis 0)
0 4 F D	F D	0 9	PC (bit 15 bis 8)

In dieser Form ist der Monitor in erster Linie zum Programmladen zu verwenden. Um Ergebnisse (Registerinhalte, Speicherinhalte) nach oder während des Laufes eines Anwenderprogramms sichtbar zu machen, gibt es eine andere Möglichkeit. Das Monitorprogramm wird auch dann aufgerufen, wenn der Prozessor innerhalb des Anwenderprogramms einen RST-2-Befehl (D 7_{16}) vorfindet. Mit dem RST-2-Befehl wird ähnlich eines Interrupts der Monitor als Interrupt-Routine aufgerufen, d.h. alle Registerinhalte auf den STACK gebracht. Der STACK-Inhalt ist dann bei entsprechender Adreßwahl anzeigbar, allerdings nur, wenn „HLT am BP" (C_4) auf 1 steht, da damit der Prozessor im Monitor bleibt, d.h. die Anzeige „steht". Dadurch ist dann eine Art von „Single-Step-Betrieb" möglich:
Steht C_4 auf 1, läuft der Prozessor mit jeder RUN-Betätigung bis zum nächsten RST 2 und bleibt stehen. Will man ein Programm austesten, ersetzt man einfach den auf den letzten Befehl des auszutestenden Programmteilers folgenden Befehl durch RST 2. Ist der Programmteil in Ordnung, wird wieder der Originalbefehl eingesetzt und RST 2 an der nächst interessanten Stelle eingebaut usw.
Der Monitoranruf über RST 2 erlaubt die Anzeige der Registerinhalte, aber nicht das Neuladen von Programmen; dies ist nur möglich, wenn der Monitor über RESET angerufen wird.

3.9 Erweitertes 8080-System (SYSTEM 7)

Im Rahmen der in den technischen Daten genannten Möglichkeiten kann das Rechnersystem des ITT MP-Experimenters durch den Benutzer erweitert werden. Eine Speichererweiterung ist auf der Adresse $0\ 8\ 0\ 0_{16}$ anzubringen, da diese durch das vorhandene Betriebsprogramm angesprungen wird, wenn der SYSTEM-Schalter auf 7 gestellt wird.
Ab der Speicheradresse $0\ 8\ 0\ 0_{16}$ kann dann ein vom Benutzer entwickeltes anderes Betriebsprogramm (ROM) oder aber auch RAM untergebracht werden. Das Monitorprogramm ist dann außer Betrieb.

Hinweis: RAM-Pufferung

Der MP-Experimenter wurde geringfügig abgeändert, so daß die Möglichkeit besteht mit Hilfe eines 2,4-V-NiCd-Akkus den Informationsinhalt des eingebauten RAM's bei Netzausfall bis zu 24 Stunden zu erhalten.
Dazu ist ein 2,4-V-NiCd-Akku (2 Zellen à 1,2 V) mit einer Kapazität von 450 bis 600 mAh mit seinem +-Pol an Buchse 84 und mit seinem −-Pol an Buchse 86 bzw. 90 der linken Anschlußleiste anzuschließen.
Ist der MP-Experimenter am Netz angeschlossen, wird der Akku mit 40 mA geladen.
Wird der MP-Experimenter nicht benutzt und ist eine Informationserhaltung des RAM's nicht erforderlich, ist der Akku abzuklemmen.
Wird eine längere Haltedauer gewünscht, so kann ein Akku größerer Kapazität (z. B. 2 Zellen in Monozellen-Ausführung mit 3500 mAh, d. h. weit mehr als 100 Stunden) verwendet werden. Zur Ladung sollte jedoch dann ein externes Ladegerät benutzt werden.

4. Stromlaufpläne und Anschlußbelegung

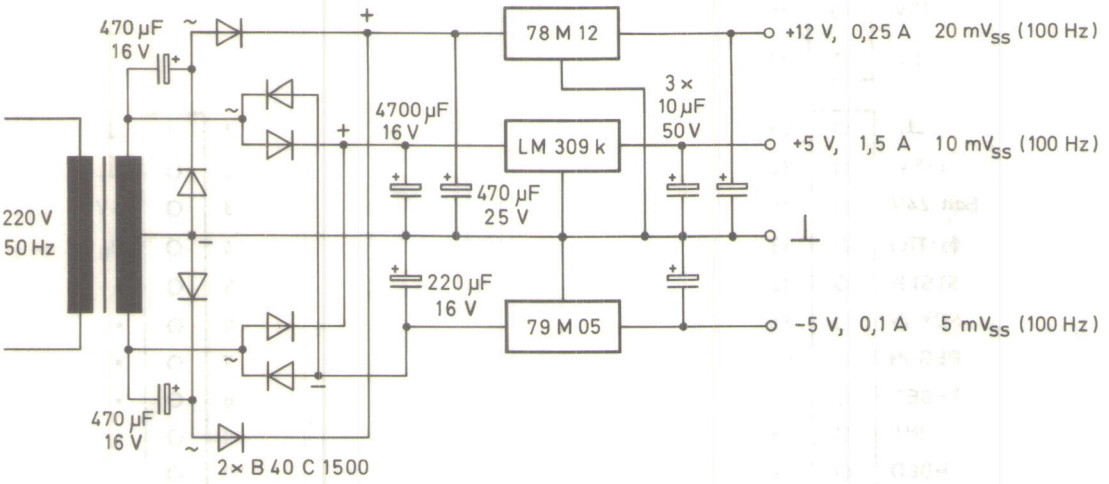


Bild 1
Stromversorgung

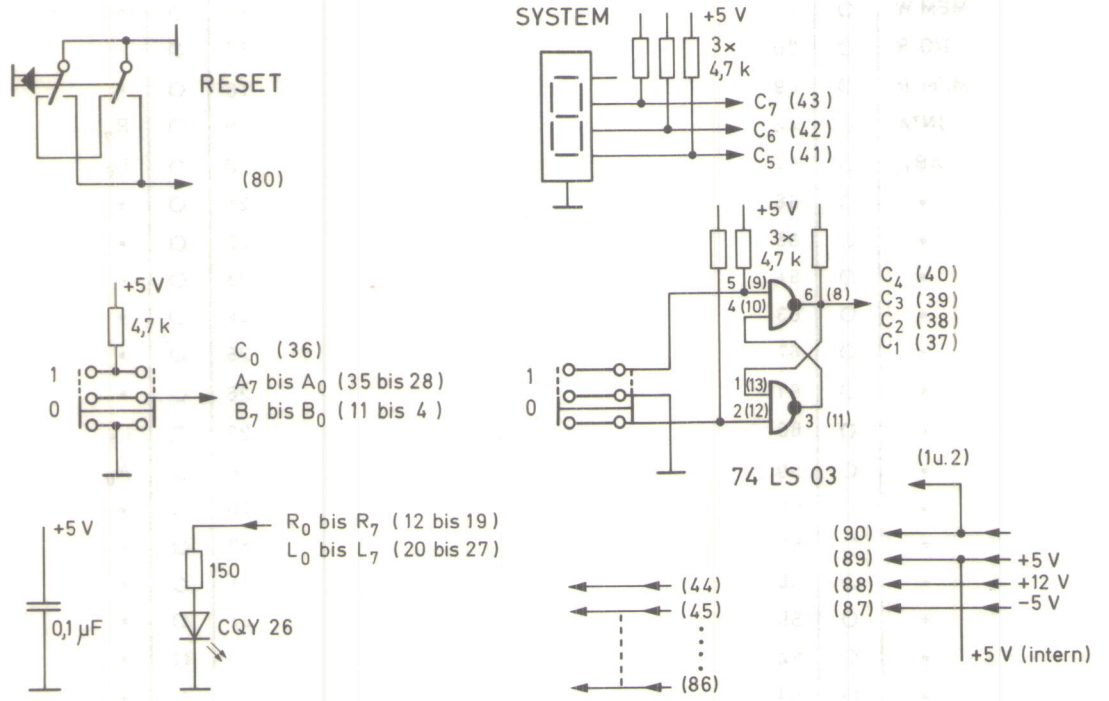
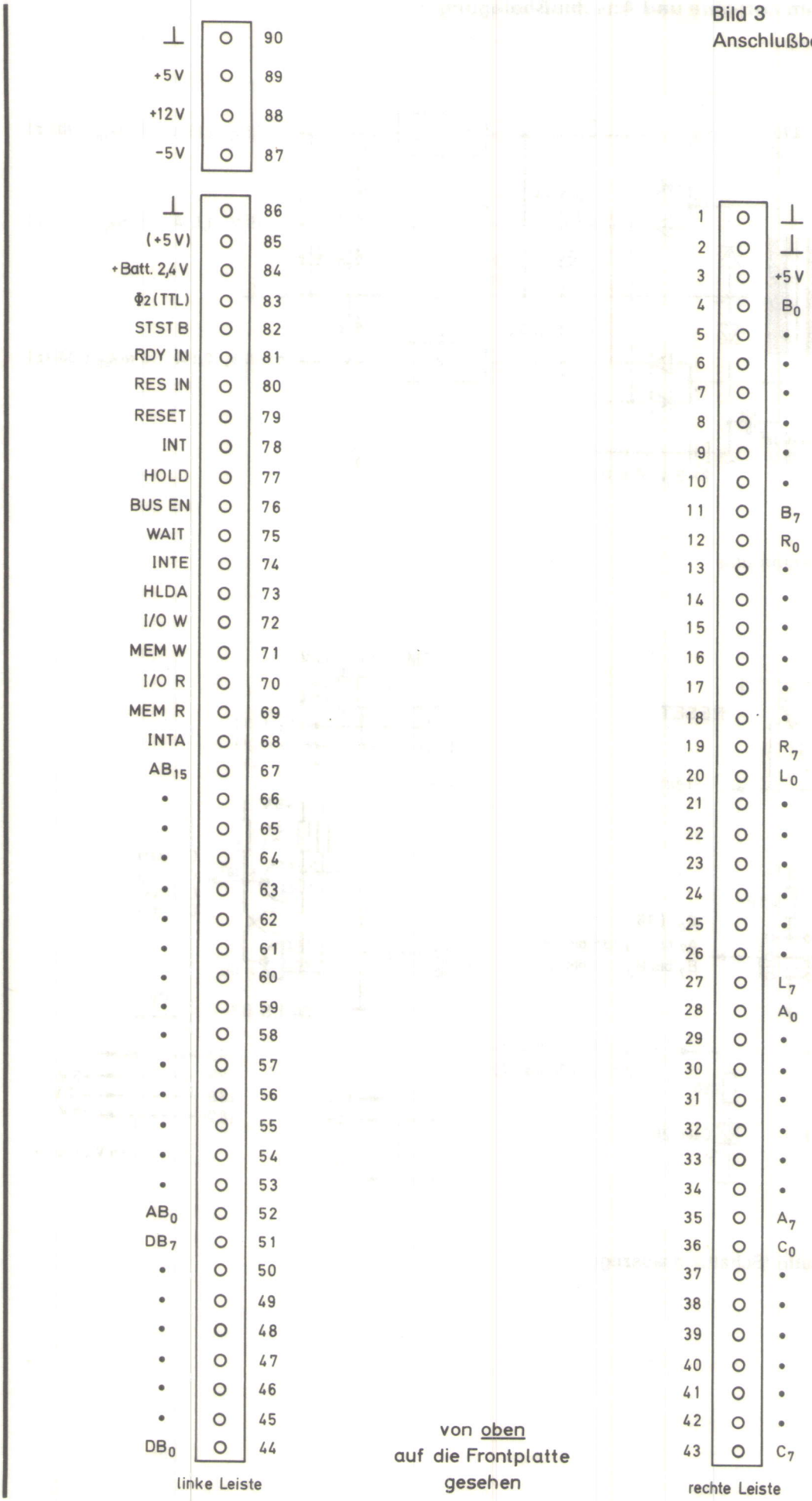
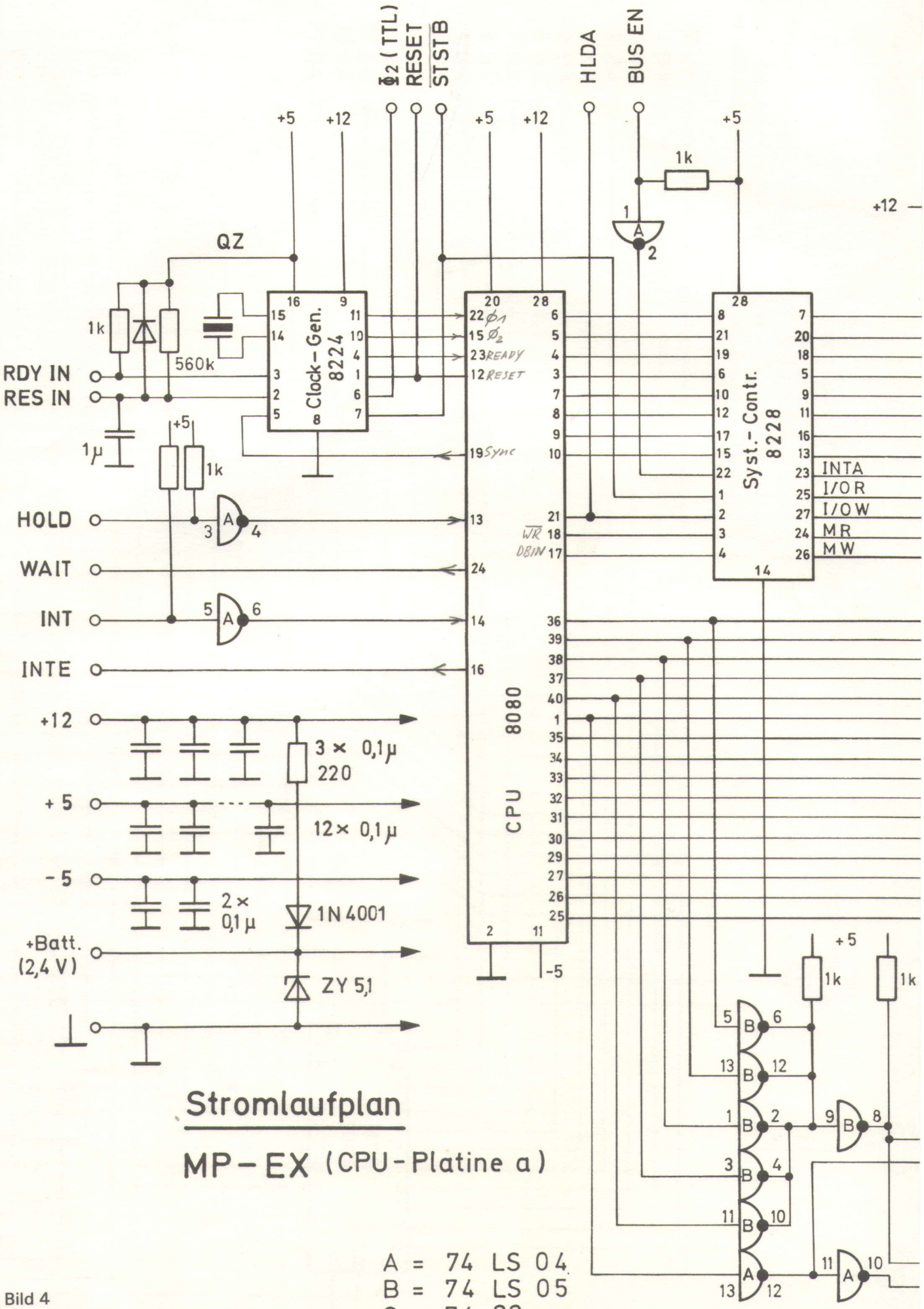


Bild 2
Frontplatte (Schaltungsauszüge)

Bild 3
Anschlußbelegung



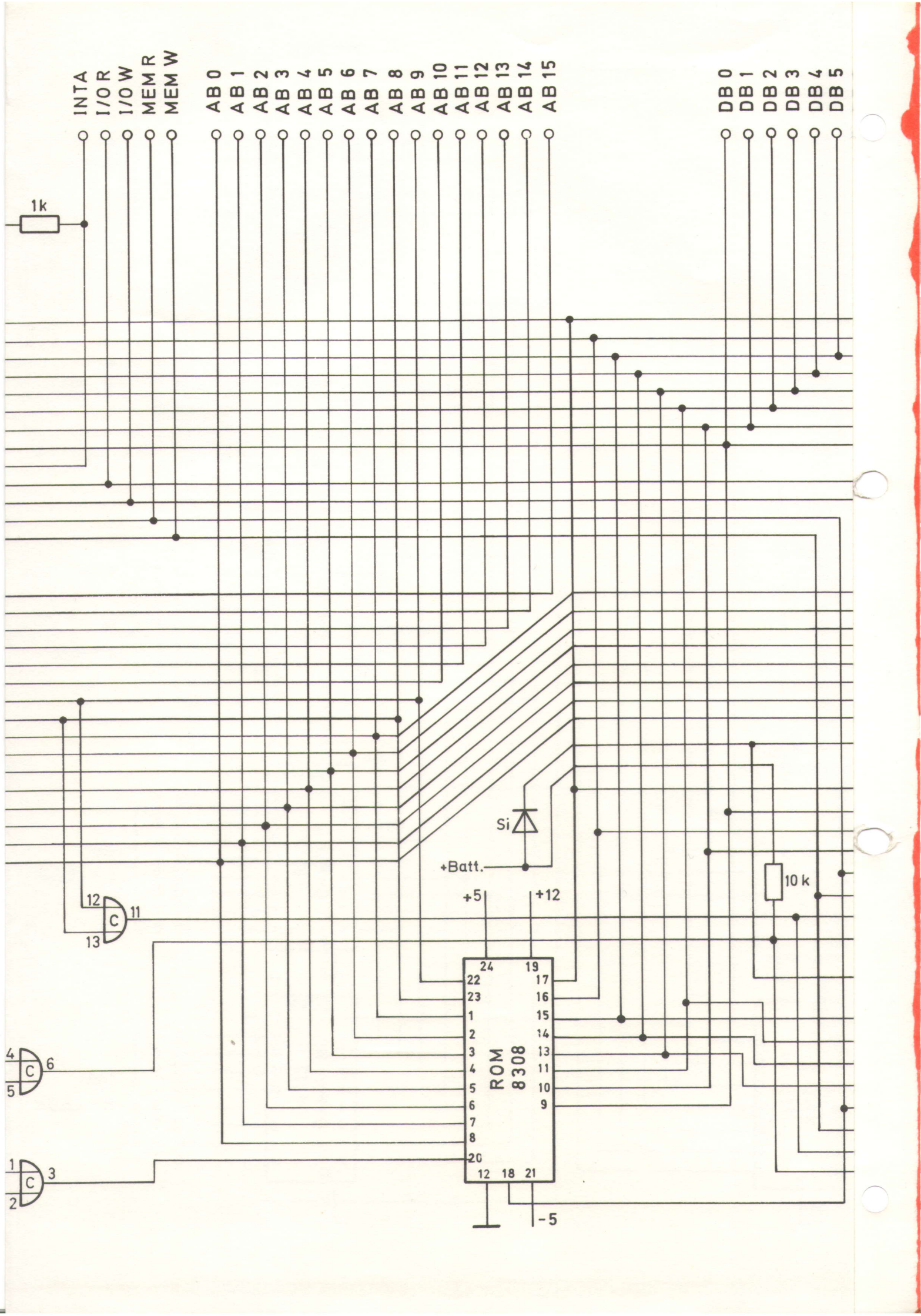


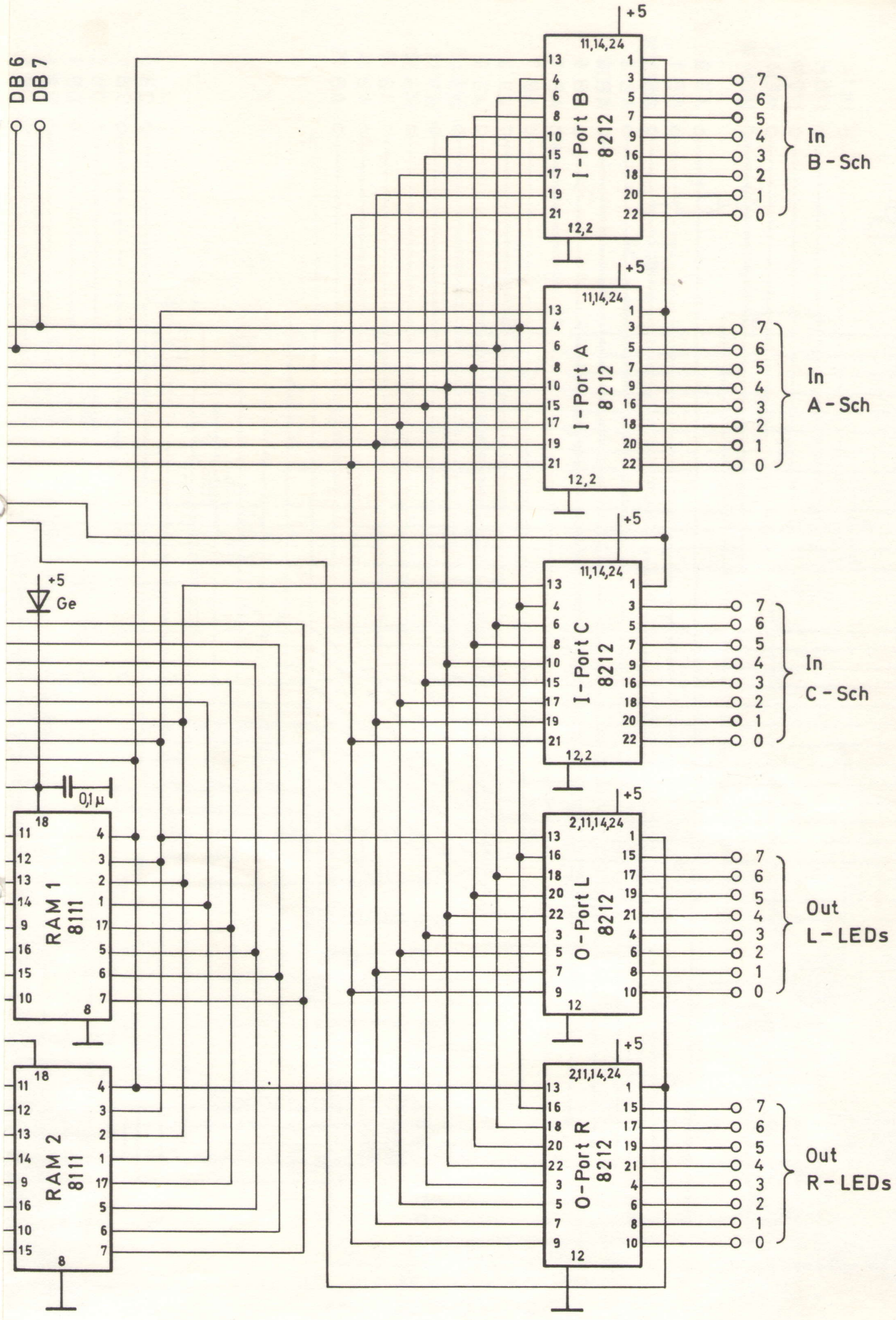
Stromlaufplan

MP – EX (CPU-Platine a)

A = 74 LS 04
B = 74 LS 05
C = 74 32

Bild 4
Stromlaufplan


INTA
I/O R
I/O W
MEM R
MEM W
AB 0
AB 1
AB 2
AB 3
AB 4
AB 5
AB 6
AB 7
AB 8
AB 9
AB 10
AB 11
AB 12
AB 13
AB 14
AB 15
DB 0
DB 1
DB 2
DB 3
DB 4
DB 5
1k
Si
+Batt.
+5
+12
10 k
ROM
8308
-5


DB 6
DB 7
+5
11,14,24
I-Port B 8212
12,2
In B-Sch
I-Port A 8212
In A-Sch
I-Port C 8212
In C-Sch
Ge
0,1 µ
RAM 1 8111
RAM 2 8111
2,11,14,24
O-Port L 8212
Out L-LEDs
O-Port R 8212
Out R-LEDs