FACHSCHUL–
FERNSTUDIUM

Dr. Klaus Geese

MIKRORECHEN–TECHNIK

2

Programmierung des MRS K 1520

Herausgeber:
Ingenieurschule
für Bergbau und Energetik
„Ernst Thälmann"
Senftenberg

03 1020 02 0

Dieser Lehrbrief wurde
verfaßt von:
Dr. Klaus G e e s e
Leiter des ORZ der
Handelshochschule Leipzig

lektoriert von:
Dipl.-Ing. Bernd L a n g e
Fachschullehrer an der Ingenieurschule
für Bergbau und Energetik "Ernst Thälmann"
Senftenberg

bearbeitet von:
FSD Dipl.-Ing. Annemarie H e r t w i g
Beauftragte für die Entwicklung von
Lehrbriefen an der Ingenieurschule für
Bergbau und Energetik "Ernst Thälmann"
Senftenberg

Redaktionsschluß: 31. 12. 1983

● Institut für Fachschulwesen der DDR, Karl-Marx-Stadt
Als Manuskript gedruckt ● Alle Rechte vorbehalten
Printed in the German Democratic Republic
Druck und buchbinderische Verarbeitung:
Zentralstelle für Lehr- und Organisationsmittel des Ministeriums für Hoch- und Fachschulwesen, Zwickau

1. Auflage 1984
4. unveränderter Nachdruck 1988
Ag 613/218/88/1000
Vorzugsschutzgebühr: 2,00 M

Inhaltsverzeichnis

1. Grundlagen der Programmierung des MR K 1520

1.1. Register des MR K 1520

Register sind 8- bzw. 16-Bit-Speicher der zentralen Verarbeitungseinheit (ZVE) und dienen der zeitweiligen Aufnahme von Daten (Zwischenergebnisse, Operanden, Adressen, Steuerinformationen).

Die Nutzung von Registern bei der Abarbeitung von Programmen ist zeiteffektiv, da die sonst erforderlichen Speicherzugriffszeiten entfallen.

Tafel 1: Hauptregistersatz des MR K 1520

Bezeichnung	Anzahl Bit	Aufgaben
Universalregister A, B, C, D, E, H, L (A-Akkumulator)	8	- Rechenoperationen - logische Operationen - Zwischenspeicher
Flagregister F	8	- Speicherung von Ergebniszuständen (Bedingungscode)
Indexregister IX, IY	16	- Speicherung von Adressen, - Adreßrechnung
Interruptregister I	8	- Aufnahme des höherwertigen Teils des Interruptvektors
Refreshregister R	8	- Adressierung des Refreshsignals (keine Bedeutung für Anwenderprogramme)
Stackpointer SP	16	- Aufnahme der aktuellen Adresse des Stack
Befehlszähler PC	16	- Aufnahme der Adresse des aktuellen Befehls (kein Zugriff vom Programm)

Neben diesem Registersatz verfügt der MR K 1520 über einen zweiten Registersatz, der für die Universalregister und das Flagregister je ein äquivalentes Austauschregister besitzt. Durch entsprechende Befehle können die Inhalte des 1. und 2. Registersatzes miteinander ausgetauscht werden.

Die Register BC, DE, HL und AF sind in ausgewählten Befehlen gemeinsam als Doppelregister (16 Bit) verwendbar. Mit Ausnahme von AF dienen sie vor allem zur Aufnahme von Adressen und zur Durchführung von Adreßrechnungen.

Eine besondere Bedeutung für die Programmierung besitzt das Flagregister. In den einzelnen Bit dieses Registers hinterlegt die ZVE bei einer großen Anzahl von Befehlen die durch die Abarbeitung entstandenen Zustände.

Tafel 2: Flags des MR K 1520

Bit im Flagregister	Kurzzeichen	Bedeutung bei gesetztem Flag (Eins)
0	C	- Übertrag nach arithmetischen Operationen - Vergleichsaussage kleiner - herausgeschobenes Bit bei Verschiebebefehlen
2	P/V	- gerade Parität (gerade Anzahl belegter Bit im Ergebnisbyte) - arithmetischer Überlauf (Vorzeichenumkehr)
6	Z	- Nullergebnis nach arithmetischen und logischen Operationen - Vergleichsaussage bei Gleichheit
7	S	- negatives Ergebnis nach arithmetischen Operationen (Bit 7 im Ergebnisbyte = 1)

Die restlichen Bit des Flagregisters sind nicht belegt oder haben nur systemintern Bedeutung.

Durch Auswertung der Flags mit bedingten Sprungbefehlen lassen sich Verzweigungen im Programmablauf realisieren.

1.2. Aufgaben und Arbeitsweise des Stack

Der Stack (Keller) ist ein vom Anwender frei wählbarer RAM-Bereich. Die Endadresse dieses Bereiches muß am Anfang eines Programmes in den Stackpointer (SP) geladen werden. Bei der Abarbeitung von Unterprogrammen bzw. Interruptbehandlungsroutinen bewahrt die ZVE in diesem Stack die Rückkehradressen des jeweils übergeordneten bzw. unterbrochenen Programmes auf (einkellern und auskellern). Der Anwender kann gleichfalls diesen Stack benutzen, um dort Registerinhalte bzw. Flagbelegungen zwischenzeitlich sicherzustellen. Das Ein- und Auskellern geschieht nach dem LIFO-Prinzip (last in, first out - zuletzt hinein, zuerst heraus). Wenn der Anwender den Stack benutzt, muß er die Einhaltung dieses Prinzips garantieren. Beim Einkellern wird zunächst der Stackpointer um 1 vermindert, dann erfolgt das Abspeichern des niederwertigen Teils der einzukellernden Adresse bzw. des Doppelregisters. Danach wird der Stackpointer erneut um 1 vermindert und der höherwertige Teil der einzukellernden Adresse bzw. des Doppelregisters abgespeichert. Beim Auskellern übernimmt die ZVE das durch den Stackpointer adressierte Byte des Stacks als höherwertigen Teil der Adresse bzw. des Doppelregisters. Dann erfolgt die Erhöhung des Stackpointers um 1 und die Übernahme des nächsten Byte aus dem Stack als niederwertiger Teil der Adresse bzw. des Doppelregisters. Auch danach erfolgt eine nochmalige Erhöhung des Stackpointers, so daß die Ausgangsstellung des Stack wieder gewährleistet ist.

1.3. Interruptbehandlung im Programm

Ist der MR K 1520 auf den Interruptmode 0 eingestellt, so erfolgt bei Auslösung eines Interrupts in Abhängigkeit von den Wickelstiftverbindungen der jeweiligen Anschlußsteuerung eine Verzweigung zu den Adressen Ø, 8, 10, 18, 20, 28, 30 oder 38 (hexadezimal). Dort muß die jeweilige Behandlungsroutine beginnen.

Wurde der Interruptmode 1 eingestellt, so geschieht bei Auslösung eines Interrupts die Verzweigung in eine Behandlungs-

routine, die unbedingt an der Speicheradresse 38 H beginnen muß. Typisch für die Interruptbehandlung am MR K 1520 ist jedoch der Interruptmode 2. Die Adressen der Behandlungsroutinen (Interruptsprungziele) sind im Programm tabellenmäßig anzuordnen:

```
ITAB:   DA IBLL     ;BEHANDLUNGSROUTINE LOCHSTREIFENLESER
        DA IBLS     ;BEHANDLUNGSROUTINE LOCHSTREIFENSTANZER
        DA IBCL1    ;BEHANDLUNGSROUTINE ZEIT 1
        DA IBCL2    ;BEHANDLUNGSROUTINE ZEIT 2
```

Mit dem gemeinsamen höherwertigen Teil der Adresse dieser Tabellenglieder ist das I-Register zu laden (höherwertiger Teil des Interruptvektors). Mit dem unterschiedlichen niederwertigen Teil der Adresse der einzelnen Tabellenglieder sind die zugehörigen Anschlußsteuerungen bzw. die zugehörigen Peripherieschaltkreise (PIO, CTC, SIO) zu programmieren. Bei Auslösung eines Interrupts meldet der jeweilige Peripherieschaltkreis den niederwertigen Teil des Interruptvektors an die ZVE zurück. Diese bildet gemeinsam mit dem I-Register daraus eine gültige Tabellenadresse und holt aus dieser Tabelle die Adresse der Behandlungsroutine, zu der verzweigt werden muß. Auf diese Art und Weise ist eine außerordentlich variable Interruptbehandlung möglich, da sowohl die Interruptvektoren als auch die Adressen der Behandlungsroutinen im Programm bei Bedarf beliebig oft modifiziert werden können.

Im Zusammenhang mit der Interruptbehandlung ist zu beachten, daß nach Anerkennung eines Interrupts durch die ZVE weitere Interrupts verboten sind. Spätestens am Ende der Interruptbehandlungsroutine muß daher durch den Befehl EI Interrupt wieder zugelassen werden.

1.4. Zahlensysteme

Der MR K 1520 arbeitet grundsätzlich bei der Realisierung aller Befehle im dualen Zahlensystem. Letztlich werden alle Operationscodes, alle Adressen von Operanden und alle zur Abarbeitung benötigten Daten dual (binär) als Bitfolge mit den Zuständen (0, 1) verschlüsselt. Um die Programmierung zu erleich-

tern, sind beim MR K 1520 weitere Zahlensysteme für die Codierung von Konstanten, Direktwerten und Adressen zugelassen.

Tafel 3: Zahlensysteme des MR K 1520

Charakteristik	Dual-system	Oktal-system	Dezimal-system	Hexadezimal-system
Basis	2	8	10	16
Wertebereich	0,1	0 ... 7	0 ... 9	0 ... 9 A,B,C,D,E,F
Byteeinteilung	bitweise	Gruppen zu 3 Bit	-	Gruppen zu 4 Bit (Tetraden)
Beispiel	00101100	00 101 100 0 5 4	44	0010 1100 2 C
Schreibweise im Befehl	101100B	54Q	44	2CH

Welches Zahlensystem bei der Codierung von Konstanten, Direktwerten und Adressen im Programm benutzt wird, bleibt dem Programmierer überlassen. Im allgemeinen sollte jedoch dem Hexadezimalsystem der Vorrang eingeräumt werden, da auch viele Systemprogramme derartig orientiert sind.

1.5. Ausgewählte Adressierungsarten

Bei der Programmierung des MR K 1520 kann man die Operanden der auszuführenden Befehle auf vielfältige Weise adressieren:

1. Registeradressierung

Im Befehl werden die symbolischen Kurzzeichen der Register bzw. Doppelregister angegeben.

LD A,B — Der Inhalt des Registers B wird in das Register A umgespeichert.

2. Speicheradressierung über Register

Das im Befehl angegebene Doppelregister enthält eine 16-Bit-Adresse, die als Zeiger auf einen Speicherplatz benutzt wird.

LD A,(HL) — Laden des Akkumulators mit den Daten des Speicherplatzes, der durch das Doppelregister

HL adressiert wird.

LD A,M M = HL (andere Schreibweise)

3. Indizierte Adressierung

Im Befehl sind ein Indexregister sowie eine Fortschaltungsadresse (- 128 ... + 127) angegeben. Beide zusammen bilden die Gesamtadresse, die als Zeiger auf einen Speicherplatz hinweist.

LD A,(IX+7) Laden des Akkumulators mit den Daten des Speicherplatzes, der durch den um sieben erhöhten Inhalt des Indexregisters IX adressiert wird.

4. Direktwertadressierung

Im Befehl steht eine 8- bzw. 16-Bit-Konstante.

LD A,97 Laden des Akkumulators mit dem Direktwert 97.

Eine spezielle Form der Direktwertadressierung tritt auf, wenn die Operandenadresse im Befehl mit ihrem symbolischen Namen angegeben ist. Dann wird durch das Übersetzungsprogramm anstelle des Namens die tatsächliche Speicheradresse des Operanden eingesetzt.

JMP WEI Springe unbedingt zum Programmzweig WEI (zur tatsächlichen Speicheradresse, wo dieser Programmzweig beginnt).

5. Direkte Speicheradressierung

Der im Befehl angegebene Absolutwert bzw. symbolische Name wirkt als Zeiger auf den in die Verarbeitung einzubeziehenden Speicherplatz.

LD A,(ZEI) Der Speicherplatz, der durch die tatsächliche Speicheradresse von ZEI bestimmt ist, wird in den Akkumulator geladen.

6. Relative Adressierung

Für ausgewählte Verzweigungsbefehle kann im Befehl absolut oder symbolisch der Abstand (- 128 ... + 127) zur aktuellen Befehlsadresse angegeben sein.

JRNZ 14 Springe um 14 Byte vorwärts, wenn das Ergebnis ungleich Null war.

JRNZ WEI-# Springe vorwärts oder rückwärts (- 128 ...

+ 127) zum Programmzweig WEI, falls das Ergebnis ungleich Null ist.

7. Bitadressierung

Bei einigen Befehlen lassen sich einzelne Bit als Operanden direkt adressieren.

```
BIT 5,A        Abtesten Bit 5 des Akkumulators auf Null.
```

1.6. Format der Programmanweisungen

Die Anweisungen eines Programmes für den MR K 1520 sind wie folgt bereitzustellen:

```
Name   Operation   Operanden   Kommentar

ZYK:   LD          A, E        ;LADEN AKKUMULATOR
       .
       .
       .
       DJNZ        ZYK-#       ;VERZWEIGEN IM ZYKLUS
```

Die einzelnen Teile einer Programmzeile werden dabei durch Leerzeichen oder Tabulator voneinander getrennt. Jede Zeile ist bei ihrer Übernahme auf einen maschinenlesbaren Datenträger mit einem Endezeichen abzuschließen (bei Lochstreifen mit NL, bei Tastatureingabe am Mikrorechnerentwicklungssystem mit der Endetaste). Eine Programmzeile darf maximal 71 Zeichen lang sein.

Name

- Symbolischer Name von Sprungzielen oder Daten,
- maximal fünf Zeichen (Buchstaben, Ziffern), beginnend mit mindestens einem Buchstaben,
- Abschluß des Namens mit einem Doppelpunkt,
- Registerbezeichnungen dürfen als Namen nicht verwendet werden.

Operation

- Code eines Befehls bzw. Pseudobefehls.

Operanden

- Form und Inhalt der Operanden sind vom jeweiligen Befehl abhängig,
- im Operandenfeld dürfen zwischendurch keine Leerzeichen auftreten.

Kommentar

- Ein Kommentar beginnt immer mit Semikolon,
- Ein Kommentar steht entweder im Anschluß an die Operanden oder sofort von Beginn der Zeile an (Kommentarzeile),
- Tabulator oder Leerzeichen vor einer Kommentarzeile führen im Übersetzungsprotokoll zu einer eingerückten Kommentarzeile.

Ü b u n g e n

Ü 1. Welche der nachfolgenden Schreibweisen sind gleichbedeutend?

a) 377Q	d) 101Q	g) 0FFH
b) 128	e) 'A'	h) 80H
c) 1	f) 1B	i) 255

Ü 2. Ermitteln Sie alle fehlerhaften Namen:

a) 1PZW	d) EINGABE
b) ZYK	e) H-K11
c) Z3	f) PS 13

2. Wirkungsweise der Befehle des MR K 1520

2.1. Ladebefehle

Mit den Ladebefehlen werden 1 Byte (8 Bit) oder 2 Byte (16 Bit) von Register zu Register bzw. zwischen Register und Speicher transportiert. Die Ladebefehle verändern die Stellung der Flags nicht. Eine Ausnahme sind dabei lediglich die Ladebefehle für das Interruptregister. Mit den Ladebefehlen des MR K 1520 lassen sich eine Vielzahl von Transportoperationen lösen:

1. Umspeichern von Registerinhalten

```
LD E,L          ; REG E : = REG L
```

2. Laden von Registern mit Direktwerten

```
LD B,97         ; REG B : = 8-BIT-WERT (97)
LD BC,1000      ; REG BC: = 16-BIT-WERT (1000)
LD IX,TAB       ; REG IX MIT ADRESSE TAB LADEN
                  (Der Wert von TAB entspricht der Speicher-
                  adresse von TAB und wird vom Übersetzungs-
                  programm errechnet.)
```

3. Laden von Registern mit Speicherinhalten

```
LD B,(HL)       ; REG B: = SPEICHERBYTE, DESSEN ADRESSE IM
                           REG HL STEHT
LD A,(ZAHL)     ; REG A: = INHALT VON ZAHL
LD BC,(SPEI)    ; REG BC:= INHALT DER SPEICHERBYTE SPEI
                           UND SPEI + 1
                       B : =(SPEI + 1)  C : = (SPEI)
```

4. Laden Speicherbyte mit Direktwerten (8 Bit)

```
LD (HL),5       ; LADEN SPEICHERBYTE, DESSEN ADRESSE IN HL
                  STEHT, MIT DIREKTWERT 5
LD (IX+7),8AH   ; LADEN SPEICHERBYTE, DESSEN ADRESSE IN IX
                  STEHT UND UM 7 FORTGESCHALTET WIRD, MIT
                  DIREKTWERT - 10001010 -
```

5. Laden Speicherbyte mit Registerinhalten

```
LD (BC),D       ; LADEN SPEICHERBYTE, DESSEN ADRESSE IN BC
                  STEHT, MIT REG D
LD (ZAHL),A     ; LADEN SPEICHERBYTE ZAHL MIT REG A
LD (SPEI),HL    ; LADEN SPEICHERBYTE SPEI UND SPEI + 1 MIT
                  REG HL
                  (SPEI): = L (SPEI + 1): = H
```

6. Sonderladebefehle

```
LD A,I          ; REG A: = INTERRUPTREGISTER
LD I,A          ; INTERRUPTREGISTER: = REG A
LD A,R          ; REG A: = REFRESHREGISTER
LD R,A          ; REFRESHREGISTER: = REG A
LD SP,HL        ; STACKPOINTER:    = REG HL
```

Die Tafeln 4 und 5 der Anlage 1 enthalten alle Möglichkeiten, die dem Programmierer am MR K 1520 zur Realisierung von Ladebefehlen zur Verfügung stehen.

2.2. Arithmetikbefehle

Mit den Arithmetikbefehlen lassen sich 1 Byte (8 Bit) bzw. 2 Byte (16 Bit) durch Addition oder Subtraktion miteinander verknüpfen. Arithmetische Operationen können somit nur im Zahlenbereich 0 ... 255 bzw. 0 ... 65535 mit den Befehlen der ZVE durchgeführt werden. Beim Überschreiten dieses Zahlenbereiches ist der Einsatz von speziellen Standardprogrammen (Festkommaarithmetik, Gleitkommaarithmetik, Mehrbytearithmetik) erforderlich. Alle Additionen und Subtraktionen mit Befehlen der ZVE werden bitweise dual durchgeführt.

```
  01000001        65
+ 00011001      + 25
  01011010        90
```

Überschreitet bzw. unterschreitet das Ergebnis den jeweiligen Zahlenbereich, so wird modulo gerechnet und Übertrag angezeigt (C - Flag = 1).

```
  11111111        255
+ 00000011      +   3
  00000010          2   (258 modulo 256)
        C-Flag = 1
```

Bei der Anwendung der Arithmetikbefehle ist zu beachten, daß die Flags abhängig von der durchgeführten Operation sehr unterschiedlich gesetzt werden. So beeinflussen beispielsweise Inkrement- und Dekrementbefehle das C-Flag nicht. Beim Erhöhen

von Registern bzw. Speicherbyte in Programmzyklen kann daher nicht auf Übertrag abgefragt werden, sondern es ist Nulldurchgang zu prüfen.

Zählwert im Zyklus		11111111	255
Erhöhung durch Inkrement	+	1	1
		00000000	0

Z-Flag = 1 Nulldurchgang
C-Flag nicht beeinflußt

Im einzelnen kann die ZVE des MR K 1520 folgende Arithmetikbefehle realisieren:

1. Akkumulator (A-Register) ± Operand (Direktwert, 8-Bit-Register, Speicherbyte)

```
ADD 5           ; REG A: = REG A + DIREKTWERT 5
ADC B           ; REG A: = REG A + REG B + C-FLAG
SUB M           ; REG A: = REG A - SPEICHERBYTE
                ; DESSEN ADRESSE IN HL STEHT
SBC (IX+7)      ; REG A: = REG A - SP - C-FLAG
                ; SP = SPEICHERBYTE, DESSEN ADRESSE
                ; IN IX STEHT UND UM 7 VORGESCHALTET
                ; WIRD
```

2. Addition bzw. Subtraktion von 16-Bit-Registern

```
ADD HL,DE       ; REG HL: = REG HL + REG DE
ADC HL,HL       ; REG HL: = REG HL + REG HL + C-FLAG
SBC HL,BC       ; REG HL: = REG HL - REG BC - C-FLAG
```

Es ist zu beachten, daß es für 16-Bit-Register nur Subtraktionsbefehle mit Berücksichtigung des C-Bit gibt.

3. Erhöhen (Inkrement) bzw. Vermindern (Dekrement) von 8-Bit-Registern, 16-Bit-Registern und Speicherbyte

```
INC B           ; REG B: = REG B + 1
INC SP          ; ERHOEHEN STACKPOINTER UM 1
INC M           ; ERHOEHEN SPEICHERBYTE, DESSEN
                ; ADRESSE IN HL STEHT, UM 1
DEC L           ; REG L: = REG L - 1
DEC SP          ; VERMINDERN STACKPOINTER UM 1
DEC (IX+7)      ; VERMINDERN SPEICHERBYTE UM 1,
```

```
                ; DESSEN ADRESSE IN IX STEHT UND UM
                ; 7 VORGESCHALTET WIRD
```

Den Tafeln 6, 7 und 8 der Anlage 1 sind alle zulässigen Arithmetikbefehle des MR K 1520 zu entnehmen.

2.3. Logische Befehle

Die ZVE des MR K 1520 realisiert als logische Befehle die Verknüpfung des Akkumulatorinhalts (A-Register) mit einem 8-Bit-Direktwert bzw. mit dem Inhalt eines Registers oder Speicherbyte. Als Verknüpfungsvorschriften existieren:

1. Verknüpfung durch UND

Akkumulator	Operand	Ergebnis im Akkumulator
0	0	0
0	1	0
1	0	0
1	1	1

- Die belegten Bit im Operanden verändern den ursprünglichen Inhalt des Akkumulators nicht.
- Die unbelegten Bit im Operanden setzen die entsprechenden Bit im Akkumulator auf Null.

Ein häufiger Anwendungsfall für das logische UND ist daher das Ausblenden (Löschen) von Bit.

```
AND 11110000B       ; AUSBLENDEN RECHTES HALBBYTE
AND OFH             ; AUSBLENDEN LINKES HALBBYTE
AND B               ; UND-VERKNUEPFUNG MIT REG B
AND A               ; AKKUMULATOR UNVERAENDERT
                    ; ABER FLAGS GESETZT
```

2. Verknüpfung durch ODER

Akkumulator	Operand	Ergebnis im Akkumulator
0	0	0
0	1	1
1	0	1
1	1	1

- Die belegten Bit im Operanden setzen das entsprechende Bit des Akkumulators auf Eins.
- Die unbelegten Bit lassen den Akkumulatorinhalt unverändert.

Das logische ODER wird deshalb häufig benutzt, um einzelne Bit auf Eins zu setzen, ehe sie beispielsweise an den Prozeß als Signal weitergegeben werden.

```
OR 128          ; SETZEN BIT 7 DES AKKUMULATORS
OR 1            ; SETZEN BIT 0 DES AKKUMULATORS
```

3. Verknüpfung durch EXKLUSIV-ODER

Akkumulator	Operand	Ergebnis im Akkumulator
0	0	0
0	1	1
1	0	1
1	1	0

- Die belegten Bit im Operanden drehen die Belegung des entsprechenden Akkumulatorbit um.
- Die unbelegten Bit im Operanden lassen den Akkumulatorinhalt unverändert.

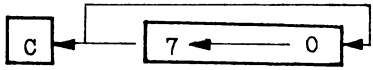
```
XOR 7           ; UMDREHEN DER AKKU-BIT 0, 1, 2
XOR A           ; LOESCHEN AKKUMULATORINHALT
```

Zu beachten ist, daß alle Logikbefehle das C-Flag löschen. Die Gesamtheit der Anwendungsmöglichkeiten für die genannten Logikbefehle ist der Tafel 9 der Anlage 1 zu entnehmen.

2.4. Verschiebebefehle

Mit den Verschiebebefehlen werden die Bit eines Registers bzw. Speicherbyte jeweils um eine Stelle nach links oder rechts verschoben. Dabei wird häufig das Ziel verfolgt, einfache arithmetische Operationen zu realisieren oder das herausgeschobene Bit abzutesten.

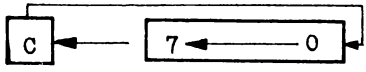
1. Linksrotation



C-Flag

Das herausgeschobene Bit 7 wird in das C-Flag und in das Bit 0 gesetzt.

Befehle: RLC , RLCA

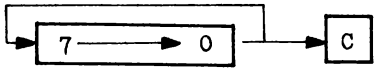


unter voller Einbeziehung des C-Flag

Das herausgeschobene Bit 7 kommt ins C-Flag, während der alte Inhalt des C-Flag als Bit 0 übernommen wird.

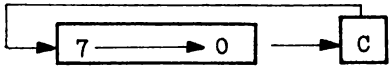
Befehle: RL , RLA

2. Rechtsrotation



Das herausgeschobene Bit 0 gelangt ins C-Flag und wird gleichzeitig als Bit 7 wieder eingezogen.

Befehle: RRC , RRCA

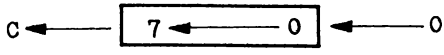


unter voller Einbeziehung des C-Flag

Das herausgeschobene Bit 0 wird ins C-Flag gesetzt und der alte Inhalt des C-Flag als Bit 7 verwendet.

Befehle: RR, RRA

3. Linksverschiebung

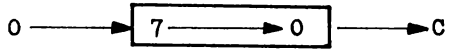


Das herausgeschobene Bit 7 steht nach der Operation im C-Flag, das Bit 0 wird auf Null gesetzt. Dies entspricht der Multiplikation des im Register befindlichen Wertes mit 2. Mit mehreren Verschiebebefehlen und durch Addition der Zwischenergebnisse lassen sich andere Multiplikationen ebenso realisieren, sofern sie in Potenzreihen zur Basis 2 zerlegbar sind.

Befehl: SLA

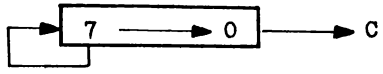
```
                        ; B-REG MIT 10 MULTIPLIZIEREN
SLA   B                 ; B-REG = N x 2
LD    A,B               ; (N x 2) IN DEN AKKUMULATOR
SLA   B                 ; B-REG = (N x 2 x 2) = (N x 4)
SLA   B                 ; B-REG = (N x 2 x 2 x 2) = (N x 8)
ADD   B                 ; (N x 2) + (N x 8) = (N x 10)
LD    B,A               ; B-REG ERHAELT ERGEBNIS
```

4. Rechtsverschiebung



Das herausgeschobene Bit 0 wird in das C-Flag gesetzt und Bit 7 erhält den Wert Null.

Befehl: SRL

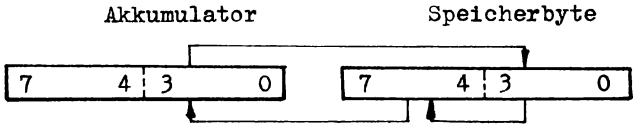


Auch hier wird das herausgeschobene Bit 0 in das C-Flag gesetzt. Auf Bit 7 wird jedoch der Wert (Eins oder Null) nachgezogen, den dieses Bit vor der Verschiebung hatte. Dadurch bleibt das Vorzeichen (Bit 7) erhalten. Dieser Befehl kann als Division durch 2 verwendet werden.

Befehl: SRA

Neben den Rotations- und Verschiebebefehlen verfügt der MR K 1520 auch über Befehle zum Halbbytetausch zwischen Akkumulator und dem durch das Doppelregister HL adressierten Speicherbyte.

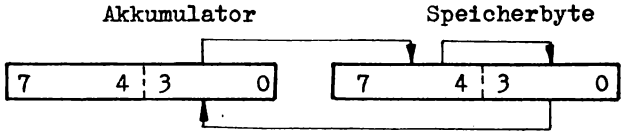
5. Halbbytetausch links



Akku-Bit A0 ... A3 wird Speicherbyte M0 ... M3
Speicherbyte M0 ... M3 wird Speicherbyte M4 ... M7
Speicherbyte M4 ... M7 wird Akku-Bit A0 ... A3

Befehl: RLD

6. Halbbytetausch rechts



Akku-Bit A0 ... A3 wird Speicherbyte M4 ... M7
Speicherbyte M4 ... M7 wird Speicherbyte M0 ... M3
Speicherbyte M0 ... M3 wird Akku-Bit A0 ... A3

Befehl: RRD

Alle Verschiebebefehle des MR K 1520 sind in den Tafeln 10 und 11 der Anlage 1 enthalten.

2.5. Vergleichsbefehle

Mit den Vergleichsbefehlen ist es möglich, den Inhalt des Akkumulators (Register A) mit anderen Registerinhalten, 8-Bit-Direktwerten oder mit einem Speicherbyte zu vergleichen. Dabei wird die Differenz zwischen Akkumulator und Operand errechnet. Im Ergebnis der Differenzbildung erfolgt das Setzen der Flags.

C-Flag = 1	Akkumulator $<$ Operand
C-Flag = 0	Akkumulator $\geqslant$ Operand
Z-Flag = 1	Akkumulator = Operand
Z-Flag = 0	Akkumulator $\neq$ Operand

Durch bedingte Verzweigungsoperationen (abhängig von der Belegung der Flags) ist eine Auswertung der Vergleichsaussage im

Programm möglich.

```
     CMP 5              ; AKKUMULATOR MINUS DIREKTWERT
     CMP B              ; AKKUMULATOR MINUS REGISTER
     CMP M              ; AKKUMULATOR MINUS SPEICHERBYTE

     CMP 9              ; AKKUMULATOR MINUS 9
     JPZ EINS           ; AKKUMULATOR = NEUN
     JPC KLEIN          ; AKKUMULATOR < NEUN
GROSS:                  ; AKKUMULATOR > NEUN
          .
          .
```

Alle Möglichkeiten zur Anwendung von Vergleichsbefehlen sind in der Tafel 12 der Anlage 1 aufgeführt.

2.6. Verzweigungsbefehle

Die Programmabarbeitung erfolgt zunächst einmal befehlsweise nacheinander (sequentiell). Bei der Lösung von Aufgaben ist es aber nicht zu umgehen, daß unter bestimmten Bedingungen diese sequentielle Reihenfolge durchbrochen wird. So sind oft einzelne Befehle oder Befehlsgruppen bedingt zu umgehen oder an bestimmten Stellen des Programmes andere, in sich geschlossene Programmzweige aufzurufen und abzuarbeiten.

Als Sprungbedingungen können dabei die Flagbelegungen angegeben sein. Die Verzweigungsbefehle ihrerseits beeinflussen die Flags nicht, so daß durch mehrere nacheinander angewendete bedingte Verzweigungsbefehle unterschiedliche Flags ausgewertet werden können.

1. Bedingte und unbedingte Sprungbefehle

```
     CMP 'A'            ; AKKU < 'A' ?
     JPC KEIBU          ; JA - KEIN BUCHSTABE (C-FLAG = 1)
     CMP 'Z'            ; AKKU > 'Z' ?
     JPZ BU             ; NEIN = 'Z' (Z - FLAG = 1)
     JPNC KEIBU         ; JA - KEIN BUCHSTABE (C-FLAG = 0)
     BU: .              ; PROGRAMMZWEIG FUER BUCHSTABEN
         .
         .
```

```
JMP ZUSAM          ; UNBEDINGTER SPRUNG
KEIBU:
LD A,20H           ; UNZ. ZEICHEN DURCH LEERZEICHEN
                   ; ERSETZEN
ZUSAM:
LD M,A             ; GEMEINSAME FORTSETZUNG
    .              ; ZEICHEN AUF SPEICHERPLATZ
    .
    .
```

Bei diesen Verzweigungen können die Sprungziele vom jeweiligen Befehl beliebig weit entfernt sein. In den meisten Programmen gibt es aber eine Vielzahl von Verzweigungen, die innerhalb relativ begrenzter Programmabschnitte stattfinden. Zur effektiven Realisierung solcher Verzweigungen verfügt der MR K 1520 über Sprungbefehle mit relativer Adressierung. Sie können im Bereich von 128 Byte rückwärts und 127 Byte vorwärts verzweigen. Sie haben den Vorteil, weniger Speicherplatz (2 statt 3 Byte) zu benötigen.

2. Bedingte und unbedingte relative Sprungbefehle

```
CMP 5              ; VERGLEICH AUF WERT 5
JRZ GLEI- #        ; GLEICHHEIT (Z-FLAG = 1)
JRC KLEIN- #       ; KLEINER (C-FLAG = 1)
  .                ; PROGRAMMZWEIG BEI GROESSER
  .
  .
```

Das Übersetzungsgrogramm berechnet "Symbol-#" und setzt die bis zum Sprungziel zu überbrückende Byteanzahl ein.
#- aktueller Stand des Speicherplatzzählers.

Eine besondere Bedeutung haben Verzweigungen innerhalb von Befehlsfolgen, die zyklisch mehrfach abgearbeitet werden müssen. Dies kann durch einen speziellen relativen Verzweigungsbefehl erfolgen. In jedem Zyklusdurchlauf verzweigt dieser Befehl nicht nur, sondern vermindert auch einen im Register B bereitzustellenden Zählwert. Dies geschieht solange, bis der Zählwert Null geworden ist.

3. Zyklische Verzweigung (relativer Sprung)

```
LD B,20          ; LADEN REG B MIT ZAEHLWERT
ZYK:  .          ; BEGINN DER ZYKLISCHEN BEFEHLSFOLGE
      .
      .
      .
DJNZ ZYK-#       ; REG B UM EINS VERMINDERN
      .          ; VERZWEIGUNG BIS REG B = 0
      .          ; PROGRAMMZWEIG NACH DEM ZYKLUS
      .
      .
```

Während der Programmentwicklung ergibt es sich häufig, daß an verschiedenen Stellen eines Programmes gleichlautende Befehlsfolgen programmiert werden müssen. Außerdem ist es möglich, ähnliche Befehlsfolgen zu vereinheitlichen und die notwendigen Unterschiede auf die Übergabe bzw. Übernahme von verschiedenen Eingangs- und Ausgangsgrößen für diese Befehlsfolgen zu beschränken.

In beiden Fällen kann man den Programmier- und Testaufwand dadurch senken, daß solche Befehlsfolgen einmalig als Unterprogramm entwickelt werden. An den verschiedenen Stellen des Programmes, an denen diese Befehlsfolgen zur Abarbeitung notwendig sind, steht dann lediglich der Aufruf des Unterprogrammes, gegebenenfalls verbunden mit Befehlen bzw. Anweisungen zur Übergabe von Parametern.

4. Aufrufen und Beenden von Unterprogrammen

```
PN P1X           ; BEGINN DES PROGRAMMES
 .
 .
CALL UP          ; 1. AUFRUF DES UNTERPROGRAMMES UP
 .               ; FORTSETZUNG NACH RUECKKEHR
 .
 .
CAZ UP           ; 2. AUFRUF DES UP BEI Z-FLAG = 1
 .               ; FORTSETZUNG NACH RUECKKEHR
 .
 .
UP:              ; BEGINN DES UNTERPROGRAMMES
 .
 .
```

```
RET                 ; ENDE DES UNTERPROGRAMMES -
                    ; RUECKSPRUNG
```

Neben dem geringeren Programmier- und Testaufwand durch die Anwendung der Unterprogrammtechnik ist auch der geringe Speicherplatzbedarf hervorzuheben, da die entsprechenden Befehlsfolgen nur einmalig vorhanden sind. Im Gegensatz dazu tritt eine geringfügige Laufzeiterhöhung ein, weil der Aufrufbefehl und der Rücksprungbefehl zusätzlich abzuarbeiten sind.

Mit dem Aufruf eines Unterprogrammes geschieht nicht nur die Verzweigung in die betreffende Befehlsfolge, sondern gleichzeitig durch die ZVE die Einspeicherung der Rückkehradresse in den Stack. Unter der Rückkehradresse ist dabei die Speicheradresse des dem Aufruf folgenden Befehles zu verstehen.

Nach der Verzweigung in das Unterprogramm werden die einzelnen Befehle des Unterprogrammes solange abgearbeitet, bis ein Rücksprungbefehl erreicht ist. Dann holt die ZVE selbständig die Rückkehradresse aus dem Stack und verzweigt wieder in das übergeordnete rufende Programm.

Aufrufbefehle und Rücksprungbefehle können sowohl unbedingt als auch in Abhängigkeit vom Inhalt des Flagregisters bedingt erfolgen. Sie selbst verändern die Flags jedoch nicht. Außerdem können Unterprogramme ineinander geschachtelt werden, d.h. innerhalb eines Unterprogrammes erfolgt der Aufruf eines weiteren Unterprogrammes. Zu beachten ist jedoch, daß in der gleichen Reihenfolge, wie der Aufruf geschieht, auch die Rückkehr abgesichert sein muß. Das ist notwendig, damit alle von der ZVE in den Stack eingekellerten Rückkehradressen ordnungsgemäß wieder ausgekellert werden. Ist dies aus programmtechnischen Gründen nicht einzuhalten, so muß eine Stackbereinigung durch den Programmierer erfolgen, indem die Adresse im Stackpointer (Register SP) entsprechend korrigiert wird.

```
PN    P2            ; HAUPTPROGRAMM P2
CALL  UP1           ; AUFRUF UNTERPROGRAMM UP1
 .                  ; FORTSETZUNG NACH RUECKKEHR
 .
 .
UP1:                ; BEGINN UNTERPROGRAMM UP1
 .
 .
CALL UP2            ; AUFRUF UP2 (SCHACHTELUNG)
 .                  ; FORTSETZUNG NACH RUECKKEHR
 .
 .
RET                 ; RUECKSPRUNG INS HAUPTPROGRAMM
UP2:                ; BEGINN UNTERPROGRAMM UP2
 .
 .
JPC WEI             ; VERZWEIGUNG BEI C-FLAG = 1
 .
 .
 .
RET                 ; NORMALER RUECKSPRUNG INS UP1
WEI:
 .
 .
INC SP              ; STACKBEREINIGUNG IM 2 BYTE
INC SP              ; UEBERGEHEN RUECKSPRUNGSADRESSE
RET                 ; RUECKSPRUNG INS HAUPTPROGRAMM
```

Eine besondere Art von Unterprogrammen sind die Interruptbehandlungsroutinen. Der Aufruf dieser Befehlsfolgen geschieht entweder hardwareseitig oder nach Auswertung des Interruptvektors durch die ZVE. Dabei wird ein laufendes Programm, das sich gegebenenfalls auch im HALT-Zustand befinden kann, durch eine anliegende Interruptforderung unterbrochen. Die Adresse des Befehles, der ohne Unterbrechung als nächster abgearbeitet worden wäre, wird als Rückkehradresse von der ZVE in den Stack eingekellert. Danach erfolgt die Verzweigung in die Interruptbehandlungsroutine und ihre befehlsweise Abarbeitung als Unterprogramm. Zum Verlassen dieser Routine stehen als spezielle Rücksprungbefehle

- RETI (mit Rücksetzung des interruptauslösenden Peripherieschaltkreises - bei maskierbarem Interrupt)
- RETN (mit Interruptfreigabe bei nichtmaskierbarem Interrupt)

zur Verfügung.

Eine weitere Besonderheit ist der Aufruf von Unterprogrammen mit dem RST-Befehl. Dieser Befehl benötigt zum Aufruf lediglich 1 Byte, ist demzufolge äußerst speicherplatzeffektiv und darüber hinaus auch sehr zeitgünstig. Die jeweiligen Unterprogramme müssen aber unbedingt an den Adressen ϕ, 8, 10, 18, 20, 28, 30, 38 (hexadezimal) beginnen. Zu beachten ist außerdem, daß die Adresse 0 bereits durch das Einschaltinterrupt belegt ist, daß beim Interruptmode 0 und 1 weitere dieser Adressen gebunden sind und daß auch viele Systemprogramme sich dieser Befehle bereits bedienen. Der RST-Befehl ist deshalb für den Anwender nur beschränkt zugänglich. Alle Verzweigungsbefehle, die für den MR K 1520 benutzt werden können, sind in den Tafeln 13 und 14 der Anlage 1 zusammengefaßt.

2.7. Stackbefehle und Registertauschbefehle

Die Stackbefehle dienen zum Ein- und Auskellern des Inhaltes von 16-Bit-Registern in den Stack.

Das Einkellern beispielsweise geschieht in Unterbrechungsbehandlungsroutinen bzw. in Unterprogrammen, um Zustände (Regsterinhalte, Flagstellungen) des unterbrochenen Programmes bzw. des übergeordneten Programmes aufzubewahren. Nach dem Einkellern sind die entsprechenden Register in der jeweiligen Routine beliebig verwendbar. Vor dem Rücksprung in das unterbrochene bzw. übergeordnete Programm können die ursprünglichen Zustände durch Auskellern wieder hergestellt werden. Dabei ist auf die richtige Reihenfolge zu achten (LIFO-Prinzip).

```
UP:   PUSH HL          ; EINKELLERN HL
      PUSH AF          ; EINKELLERN AF
        .              ; HL, AF AB HIER BELIEBIG NUTZBAR
        .
        .
      POP AF           ; AUSKELLERN AF
      POP HL           ; AUSKELLERN HL
      RET              ; RUECKSPRUNG
```

Mit den Stackbefehlen lassen sich auch nicht vorhandene Ladebefehle realisieren:

```
      PUSH HL          ; EINKELLERN HL
      POP BC           ; AUSKELLERN NACH BC
                       ; BC : = HL
```

Bei den Tauschbefehlen haben vor allem die Befehle eine große Bedeutung, die zum Tausch des Inhalts des Hauptregistersatzes mit dem zweiten Registersatz führen. Auch diese Befehle können zum Retten und Wiederherstellen von Zuständen im Programm verwendet werden.

```
      EXAF             ; AKKUMULATOR UND FLAGS DES ERSTEN
                       ; UND ZWEITEN REGISTERSATZES
                       ; TAUSCHEN
      EXX              ; BC, DE, HL MIT ENTSPRECHENDEN
                       ; REGISTERN DES 2. SATZES TAUSCHEN
```

Die am MR K 1520 zugelassenen Stack- und Registertauschbefehle sind Tafel 15 der Anlage 1 zu entnehmen.

2.8. Blocktransport- und Blocksuchbefehle

Der MR K 1520 verfügt über einige Befehle, die eine Mehrbytearbeit im Speicher ermöglichen. Dazu gehören:

1. Geschlossener Transport von Zeichen (Blocktransport)

   Bedingungen: - Laden des Doppelregisters HL mit der Adresse des Sendebereiches
   - Laden des Doppelregisters DE mit der Adresse des Empfangsbereiches

- Laden des Doppelregisters BC mit der Anzahl der zu transportierenden Zeichen

```
LD HL,BER          ; SENDEADRESSE LADEN
LD DF,SPEI         ; EMPFANGSADRESSE LADEN
LD BC,64           ; 64 ZEICHEN UMSPEICHERN
LDIR               ; VON BER NACH SPEI
```

2. Transport von Zeichen mit zyklischer Unterbrechung je Byte

Es wird jeweils nur ein Zeichen transportiert, die Sende- und Empfangsadresse fortgeschaltet und die Anzahl vermindert. Vor dem Transport des nächsten Zeichens können anwenderspezifische Befehle zur Abarbeitung gebracht werden. Anhand des PV-Flags läßt sich abtesten, ob noch weitere Zeichen zu transportieren sind (P/V-Flag = 1).

```
LD HL,BER          ; SENDEADRESSE LADEN
LD DE,SPEI         ; EMPFANGSADRESSE LADEN
LD BC,64           ; MAXIMAL 64 ZEICHEN UMSPEICHERN
   .
   .
   .
LD A,1EH           ; ENDEZEICHEN LADEN (NL)
ZYK: CMP (HL)      ; ENDEZEICHEN IM SENDEBEREICH?
JRZ FERT-#         ; JA - VORZEITIGES ENDE
LDI                ; TRANSPORT EINES ZEICHENS
                   ; ADRESSE FORTSCHALTEN
                   ; ANZAHL VERMINDERN
JPPE ZYK           ; P/V-FLAG = 1, WEITER IM ZYKLUS
FERT:    .         ; GEMEINSAMES ENDE DURCH ANZAHL
         .         ; BZW. ENDEZEICHEN
         .
```

3. Geschlossenes Suchen eines Zeichens in einer Zeichenmenge (Blocksuchen)

Bedingungen:
- Laden des Doppelregisters HL mit der Adresse des zu durchsuchenden Bereiches,
- Laden des Doppelregisters BC mit der Anzahl der zu durchsuchenden Zeichen,
- Laden des Akkumulators mit dem Suchzeichen.

```
LD A,'X'            ; GESUCHTES ZEICHEN -X- LADEN
LD HL,BER           ; ADRESSE DES SUCHBEREICHES
LD BC,64            ; MAX. 64 ZEICHEN DURCHSUCHEN
CPIR                ; SUCHBEFEHL
JRZ GEF-#           ; GEFUNDEN - Z-FLAG = 1
    .               ; PROGRAMMZWEIG "NICHT GEFUNDEN"
    .
    .
GEF:                ; PROGRAMMZWEIG "GEFUNDEN"
    .
    .
```

4. Suchen eines Zeichens mit zyklischer Unterbrechung je Byte

Es wird immer nur ein Zeichen verglichen. Ehe das jeweils nächste Zeichen an der Reihe ist, können zusätzliche anwenderspezifische Befehle abgearbeitet werden.

```
LD A,1EH            ; SUCHBEGRIFF - ENDEZEICHEN (NL)
LD HL,BER           ; ADRESSE SUCHBEREICH
LD BC,64            ; MAX, 64 ZEICHEN ZU DURCHSUCHEN
ZYK:  .             ; AKTUELLES ZEICHEN BEARBEITEN
      !             ; (ZUM BEISPIEL CODEWANDLUNG)
      .
CPI                 ; VERGLEICH AUF ENDEZEICHEN
                    ; ADRESSE SUCHBEREICH FORTSCHALTEN
                    ; ANZAHL VERMINDERN
JRZ GEF-#           ; ENDE DURCH ENDEZEICHEN
JPPE ZYK            ; P/V-FLAG = 1 - WEITER IM ZYKLUS
GEF:  .             ; GEMEINSAMES ENDE DURCH ANZAHL
                    ; BZW. ENDEZEICHEN
```

Die unter 1. bis 4. beschriebenen Befehle arbeiten mit aufsteigenden Speicheradressen (vorwärts). Darüber hinaus gibt es noch die Befehle LDDR, LDD, CPDR und CPD, die mit absteigenden Speicheradressen (rückwärts) arbeiten. Alle anderen Abläufe bleiben dabei unverändert.

```
LD HL, BER + 63        ; SENDEADRESSE LADEN (ENDADRESSE)
LD DE, SPEI + 63       ; EMPFANGSADRESSE LADEN (ENDADRESSE)
LD BC, 64              ; ANZAHL FUER DAS UMSPEICHERN
LDDR                   ; 64 BYTE RUECKWAERTS UMSPEICHERN
```

Die genannten Befehle sind in Tafel 16 der Anlage 1 zusammengefaßt.

2.9. Ein- und Ausgabebefehle

Durch die Ein- und Ausgabebefehle erfolgt der Datenaustausch mit der Peripherie. Dabei dienen die Eingabebefehle zum Einlesen von Daten sowie zum Einlesen von Statusinformationen über die beteiligten peripheren Geräte.

Die Ausgabebefehle hingegen realisieren die physische Übergabe der Daten an die peripheren Einrichtungen sowie die Übertragung von Kommandos zur Steuerung der Ein- und Ausgabeprozesse.

Die beteiligten Ein- und Ausgabegeräte werden über Ein- und Ausgabeadressen (Kanaladressen) angesprochen. Die nachfolgend beschriebenen Befehle sind für sich allein betrachtet oftmals nicht in der Lage, komplette Ein- und Ausgabeprozesse auszulösen. Dazu ist in der Regel eine ganze Folge von Ein- und Ausgabebefehlen erforderlich.

Alle Möglichkeiten der Ein- und Ausgabebefehle des MR K 1520 sind der Tafel 17 der Anlage 1 zu entnehmen.

2.9.1. Eingabebefehle

Der MR K 1520 verfügt über folgende Befehlsvarianten zur Eingabe:

1. Einlesen Einzelbyte

```
IN 84H          ; EINGABE UEBER KANALADRESSE 84 H
                ; BYTE STEHT IM AKKUMULATOR
```

```
LD C,84H        ; LADEN KANALADRESSE 84 H INS REG C
IN B            ; BYTEEINGABE IN DAS REG B
                ; KANALADRESSE DARF NUR IM REG C
                ; STEHEN
```

2. Geschlossene Eingabe mehrerer Byte (Blockeingabe)

Bedingungen:
- Doppelregister HL enthält Speicherempfangsadresse,
- Register B enthält Anzahl der einzulesenden Byte (max. 255),
- Register C enthält Kanaladresse.

```
LD HL,SPEI      ; LADEN SPEICHERADRESSE
LD B,10         ; LADEN ANZAHL
LD C,84H        ; EINSTELLEN KANALADRESSE
LDIR            ; 10 BYTE NACH SPEI EINLESEN
```

3. Eingabe mehrerer Byte mit zyklischer Unterbrechung

Es wird jeweils nur ein Byte eingelesen, die Speicherempfangsadresse fortgeschaltet und die Anzahl der einzulesenden Byte vermindert.

Vor der nächsten Eingabe können anwenderspezifische Befehle abgearbeitet werden.

Sind bereits alle Byte eingelesen, so ist dies über das Z-Flag abtestbar (Z-Flag = 1).

```
LD HL,SPEI      ; LADEN SPEICHERADRESSE
LD B,10         ; LADEN ANZAHL
LD C,84H        ; EINSTELLEN KANALADRESSE
ZYK: LDI        ; 1 BYTE EINLESEN
                ; SPEICHERADRESSE FORTSCHALTEN
                ; ANZAHL VERMINDERN
JRZ FERT-#      ; ALLES EINGELESEN
    .           ; ANWENDERSPEZIFISCHE BEFEHLE
    .
JR ZYK-#        ; WEITER EINLESEN
FERT:  .
       .
       .
```

Die unter 2. und 3. genannten Befehle speichern die eingelesenen Byte mit aufsteigender Speicheradresse (vorwärts). Außerdem gibt es noch die Befehle IND und INDR, die genauso ablaufen, aber die eingelesenen Byte mit absteigender Adresse speichern (rückwärts).

2.9.2. Ausgabebefehle

Auch für die Ausgabe besitzt der MR K 1520 analoge Möglichkeiten wie für die Eingabe.

1. Ausgabe Einzelbyte

```
OUT 85H        ; AUSGABE UEBER KANALADRESSE 85 H
```

```
LD C,85H       ; LADEN KANALADRESSE 85 H INS REG C
OUT B          ; AUSGABE DES INHALTS VON REG B
               ; KANALADRESSE MUSS IM REG C STEHEN
```

2. Geschlossene Ausgabe mehrerer Byte (Blockausgabe)

   Bedingungen:
   - Laden Doppelregister HL mit Speichersendeadresse,
   - Laden Register B mit Anzahl der auszugebenden Byte (max. 255),
   - Laden Register C mit Kanaladresse.

```
LD HL,BER      ; LADEN DER SPEICHERSENDEADRESSE
LD B,10        ; LADEN ANZAHL
LD C,85H       ; EINSTELLEN KANALADRESSE
OTIR           ; AUSGABE 10 BYTE VON BER
```

3. Ausgabe mehrerer Byte mit zyklischer Unterbrechung

   In jedem Zyklus wird nur ein Byte ausgegeben sowie die Speichersendeadresse fortgeschaltet und die Anzahl der auszugebenden Byte vermindert.

   Vor der Ausgabe des jeweiligen Byte können außerdem noch anwenderspezifische Befehle zur Abarbeitung kommen. Sind

im Zyklus alle Byte ausgegeben, so ist das Z-Flag gesetzt.

```
LD HL,BER          ; LADEN SPEICHERSENDEADRESSE
LD B,10            ; LADEN ANZAHL
LD C,85H           ; EINSTELLEN KANALADRESSE
ZYK:    .          ; ANWENDERSPEZIFISCHE BEFEHLE
        .          ; (Z.B. CODEUMSCHLUESSELUNG)
OUTI               ; 1 BYTE AUSGEBEN
                   ; SPEICHERADRESSE FORTSCHALTEN
                   ; ANZAHL VERMINDERN
JRNZ ZYK-#         ; WEITER AUSGEBEN BIS ANZAHL = 0
        .          ; AUSGABE ABGESCHLOSSEN
        .
```

Die Befehle OTIR und OUTI realisieren die Byteausgabe mit steigenden Speichersendeadressen. Darüber hinaus kann mit den Befehlen OTDR und OUTD ebenfalls eine Ausgabe mehrerer Byte bei absteigender Speichersendeadresse durchgeführt werden.

2.10. Spezielle Befehle

In dieser Befehlsgruppe sind die Bitmanipulationsbefehle besonders zu beachten. Mit ihnen kann direkt ein ausgewähltes Bit innerhalb eines Registers bzw. Speicherbyte abgetestet, gesetzt oder gelöscht werden.

```
BIT 5,B            ; ABTESTEN BIT 5 IM REG B
JRZ NULL-#         ; Z-FLAG = 1, WENN BIT 5 = 0 IST
SET 7,A            ; SETZEN BIT 7 IM AKKUMULATOR
RES 2,M            ; LOESCHEN BIT 2 IM SPEICHERBYTE
```

Ein weiterer wichtiger Befehl dient zur Dezimalkorrektur bei Addition bzw. Subtraktion von BCD-Zahlen (gepackte Zahlen). Diese beinhalten je Byte zwei Ziffern, d.h. je Tetrade eine Ziffer.

```
1 0 0 0  0 1 0 1
   8        5
```

Werden solche BCD-Zahlen beispielsweise miteinander addiert, so geschieht dies bitweise. Dadurch können Pseudotetraden entste-

hen.

```
  1 0 0 0      0 1 0 1      8   5
+ 0 0 1 1    + 1 0 0 1    + 3   9
  1 0 1 1      1 1 1 0     12   4
    B            E
```

Durch den Befehl DAA erfolgt dann eine Korrektur der Pseudotetraden, indem auf jede vorhandene Pseudotetrade dual eine 6 addiert wird.

```
  1 0 1 1      1 1 1 0
+ 0 1 1 0    + 0 1 1 0
  0 0 1 0      0 1 0 0
    2            4     und gesetztes C-Flag (Übertrag)
```

Eine Addition bzw. Subtraktion von BCD-Zahlen ist daher stets wie folgt durchzuführen:

```
ADD B          ; ADDITION BCD-ZAHLEN REG A + REG B
DAA            ; DEZIMALKORREKTUR
```

Die anderen Befehle dieser Befehlsgruppe bedürfen keiner weiteren Erläuterung. Sie sind in den Tafeln 18 und 19 der Anlage 1 aufgeführt.

3. Pseudobefehle des MR K 1520

3.1. Definitionsanweisungen

3.1.1. Definition von Datenbytes (Konstanten)

name: DB n

Es wird jeweils ein Byte mit dem Inhalt von n belegt. Textkonstanten benötigen dagegen soviel Speicherplatz, wie Textzeichen vorhanden sind.

```
KON 1:   DB 377Q        ; DEFINIERE 1 BYTE -11111111-
KON 2:   DB 1           ; DEFINIERE 1 BYTE -00000001-
KON 3:   DB 0AH         ; DEFINIERE 1 BYTE -00001010-
KON 4:   DB 'FEHLER'    ; DEFINIERE 6 BYTE (TEXT)
```

3.1.2. Definition von Adressen

```
name:     DA nn
```

nn-16-Bit-Adreßkonstante

-symbolische Adresse (wird vom Übersetzer in die tatsächliche Speicheradresse umgewandelt).

```
ADR 1:    DA 1000        ;NW: 11101000 - HW: 00000011
ADR 2:    DA TAB         ;NW VON TAB - HW VON TAB
```

Jede Adresse belegt 2 Byte Speicherplatz in der Reihenfolge niederwertiges Byte (NW) und höherwertiges Byte (HW)

NW 2^7 ..... 2^0 HW 2^{15} ..... 2^8

Dadurch lassen sich Adressen im Wertbereich von 0 ... 65535 bilden.

3.1.3. Reservierung (Freihalten) von Speicherplatz

```
name:     BER nn
```

nn - Byteanzahl

```
SPEI:     BER 100        ;RESERVIEREN 100 Byte
```

Der reservierte Speicherplatz ist inhaltlich undefiniert.

3.2. Ausgewählte Assembleranweisungen

Diese Anweisungen haben nur für den Übersetzungsvorgang Bedeutung. Sie beeinflussen zwar gegebenenfalls das abarbeitbare Maschinencodeprogramm, werden aber selbst nicht in den Maschinencode umgewandelt.

1. Definiere Programmname

```
PN name
```

name - 2 Zeichen signifikant

```
PN P1          ;DAS PROGRAMM HEISST P1
```

Der auszugebende Maschinencode wird mit diesem Programmnamen gekennzeichnet. Die Anweisung ist nicht zwingend notwendig.

2. Setzen Speicherplatzzähler

```
ORG nn              nn - Adresse von 0 ... 65535
ORG 1000H           ;SPEICHERPLATZZAEHLER AUF
                     1000 H
```

Alle nachfolgenden Befehle und Daten werden ab der angegebenen Speicheradresse angeordnet. Fehlt diese Anweisung, beginnen Befehle und Daten entweder ab Adresse 0 oder ab einer zum Übersetzungszeitpunkt einzugebenden Basisadresse. Es sollte beachtet werden, daß bei Benutzung der Anweisung ORG das jeweilige Programm nicht mehr verschieblich ist.

3. Sonstige Assembleranweisungen

TITL ' ... '	Programmtitel mit maximal 61 Zeichen; erscheint auf jeder Übersetzungsseite.
EJEC	An dieser Stelle wird in der Übersetzungsliste ein Blattwechsel durchgeführt.
END	Anweisung steht am physischen Ende eines Programmes. - Ende - Hinweis für das Übersetzungsprogramm.

Übungen

Ü 3. Wie lautet die Bitbelegung der nachfolgend definierten Konstanten?

```
KON 5:      DB 40Q
KON 6:      DB 'A'
KON 7:      DB 255
KON 8:      DB 31H
KON 9:      DB '1'
```

4. Programmierbeispiele

4.1. Programmierung des Zähler-/Zeitgeber-Schaltkreises (CTC)

Die Funktionsweise des CTC-Schaltkreises läßt sich vom Anwender über Kommandos einstellen. Dies geschieht durch Ausgabebefehle, mit denen die jeweiligen Kommandos bitweise verschlüsselt von der ZVE an den Schaltkreis übertragen werden. Arbeitet der Schaltkreis im Interruptmode 2, so ist als erstes der Interruptvektor zu übermitteln. Dieser muß folgende Bitbelegung besitzen:

```
xxxxx 00 0
      |  |
      |  +-- KZ für Interrupt-Vektor
      +----- wird bei Rückmeldung des Interrupt-Vektors mit
             der Kanaladresse 0 - 3 belegt (interruptauslö-
             sender Kanal)
```

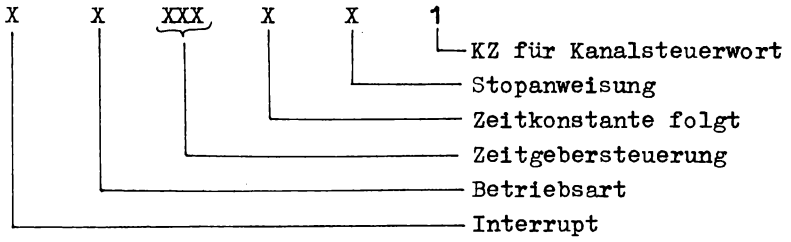
Auf der Grundlage dieser Bitbelegung sind maximal vier Interruptbehandlungsroutinen auslösbar, falls jeder Kanal des CTC-Schaltkreises vom Anwender getrennt als Zähler bzw. Zeitgeber genutzt wird.

Die Adressen der zugehörigen Interruptbehandlungsroutinen müssen entsprechend in die Tabelle für die Interruptsprungziele eingeordnet werden, d.h. die Zieladresse für ein Interrupt des Kanales 0 muß auf einer durch acht teilbaren Speicheradresse stehen.

```
ITAB:       .           ; BEGINN DER TABELLE FUER INTERRUPT-
            .           ; SPRUNGZIELE
            .
        DA  IBKA0       ; INT-BEHANDLUNG KANAL 0
        DA  IBKA1       ; INT-BEHANDLUNG KANAL 1
        DA  IBKA2       ; INT-BEHANDLUNG KANAL 2
        DA  IBKA3       ; INT-BEHANDLUNG KANAL 3
```

Durch acht teilbare Speicheradresse

Nach dem Interruptvektor ist das Kanalsteuerwort an den Schaltkreis auszugeben. Die Bitbelegung des Kanalsteuerworts läßt folgende Verschlüsselungen zu:



Bit 0 - 1 Kennzeichen Kanalsteuerwort
Bit 1 - 0 Kanal arbeitet
1 Kanalarbeit unterbrochen
Bit 2 - 0 es folgt keine Zeitkonstante
1 nächstes Byte ist Zeitkonstante/Zählwert
Bit 3 - 0 Zeitmessung ab Anfang Maschinenzyklus
1 Zeitmessung ab nächster positiver/negativer Flanke
Bit 4 - 0 Start bei negativer Flanke von CLK/TRG
1 Start bei positiver Flanke von CLK/TRG
Bit 5 - 0 Vorteiler 1 : 16
1 Vorteiler 1 : 256
Bit 6 - 0 Betriebsart Zeitgeber C = t x P x T

t = Systemtakt
P = Vorteiler
T = Zeitkonstante 1 - 256

1 Betriebsart Zähler
Bit 7 - 0 Interrupt für Kanal gesperrt
1 Interrupt für Kanal erlaubt

Das Kanalsteuerwort ist für jeden Kanal, der im Programm genutzt wird, an den Schaltkreis auszugeben. Ist im Kanalsteuerwort eine Zeitkonstante avisiert, so muß sie unbedingt mit dem nächsten Ausgabebefehl übermittelt werden.

B e i s p i e l

Ein CTC-Schaltkreis ist wie folgt als Zeitgeber zu nutzen:
- Kanal 0 arbeitet als Zeitgeber für Maximalzeit
- Ausgang Kanal 0 taktet Zähler Kanal 1
- Nulldurchgang des Zählers Kanal 1 löst Interrupt aus.

```
LD   A,11111000B    ; INTERRUPTVEKTOR
                    ; RUECKMELDUNG -
                    ; 11111010 (KANAL 1)
LD   A,11000101B    ; KANALSTEUERWORT KANAL 1
OUT  81H
LD   A,0            ; ZAEHLWERT 0 = 256
OUT  81H
LD   A,00111101B    ; KANALSTEUERWORT KANAL 0
OUT  80H            ; VORTEILER 256
LD   A,0            ; ZEITKONSTANTE 256
OUT  80H
```

Z = 0,4 /µs x 256 x 256 x 256

Vor-	Zeit-	Zähler
tei-	kon-	Kanal 1
ler	stante	

Ü b u n g

Ü 4. Alle 15 min soll Interrupt ausgelöst werden.

Vorteiler: 256

Zähler: Kanal 0 = 255
Kanal 1 = 241
Kanal 2 = 143

Programmieren Sie den CTC-Schaltkreis.

4.2. Programmierung des PIO-Schaltkreises

Der PIO-Schaltkreis kann gleichfalls durch Anwenderkommandos auf eine bestimmte Betriebsweise eingestellt werden. Dazu sind von der ZVE über Ausgabebefehle die bitweise verschlüsselten Kommandos an den Schaltkreis zu übermitteln.

Arbeitet der MR K 1520 im Interruptmode 2, so ist zunächst das Laden des Interruptvektors erforderlich:

```
xxxxxxx 0
        └──Kennzeichen Interruptvektor
```

Die Adresse der zugehörigen Interruptbehandlungsroutine muß in der Tabelle für Interruptsprungziele an einer durch zwei teil-

baren Speicheradresse stehen. Der Interruptvektor ist für jeden Port (Kanal) auszugeben, der im Interruptbetrieb arbeiten soll.

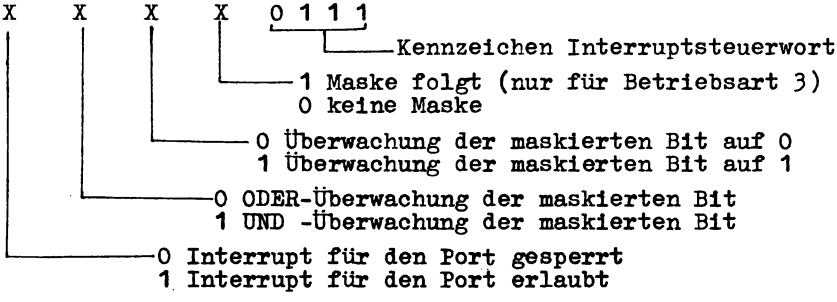
Danach geschieht das Einstellen der Betriebsart durch die Ausgabe eines der nachstehenden Kommandos:

00001111	Byte-Ausgabe
01001111	Byte-Eingabe
10001111	Byte-Ein- und Ausgabe (nur Port A)
11001111	Bitweise Ein- und Ausgabe

Falls eine bitweise Ein- und Ausgabe vorgenommen wird, ist im Anschluß an die Einstellung der Betriebsart durch ein weiteres Kommando (E/A-Steuerwort) festzulegen, welche der acht Bit des jeweiligen Port für die Eingabe oder für die Ausgabe vorgesehen sind. Dabei erfolgt die Kennzeichnung der Ausgabebit mit 0 und die der Eingabebit mit 1.

10001111	Bit 0, 1, 2, 3, 7 - Eingabe
	Bit 4, 5, 6 - Ausgabe

Des weiteren muß bei allen Betriebsarten ein Interruptsteuerwort an den PIO-Schaltkreis übergeben werden.



Arbeitet der Schaltkreis in der Betriebsart 3 (bitweise E/A), so kann im Anschluß an das Interruptsteuerwort noch ein Maskenbyte ausgegeben werden. In dieser Maske sind für alle Eingabebit des Port, die einer Überwachung bedürfen, die entsprechenden Maskenbit auf Null zu setzen.

0 1111110	Von den Eingabebit sind lediglich die Bit 0 und 7 zu überwachen.

Tritt die Überwachungsbedingung ein, so wird Interrupt ausgelöst.

B e i s p i e l

Der Kanal A einer PIO (E/A-Adresse 86 H für Kommandos) arbeitet nach der Betriebsart 3 (bitweise Ein- und Ausgabe).
Bit 0, 1 - Stellsignale an den Prozeß
Bit 2, 3, 4 - Prozeßsignale für Ausnahmesituationen
Bit 5, 6, 7 - Prozeßeingaben
Jedes Prozeßsignal für Ausnahmesituationen löst durch Interrupt ein Havarieprogramm aus.

```
        LD   HL,ITAB      ; LADEN ADRESSE INTERRUPTSPRUNGZIEL-
                          ; TABELLE
        LD   A,H          ; HOEHERWERTIGEN ADRESSTEIL
        LD   I,A          ; INS INTERRUPTREGISTER LADEN
        LD   A,L          ; NIEDERWERTIGEN ADRESSTEIL
        OUT  86H          ; AN PIO-PORT A AUSGEBEN
        IM2               ; EINSTELLEN INTERRUPTMODE 2
        LD   A,11001111B  ; BETRIEBSARTENSTEUERWORT (BA = 3)
        OUT  86H
        LD   A,11111100B  ; E/A-STEUERWORT
        OUT 86H
        LD   A,10110111B  ; INTERRUPTSTEUERWORT
        OUT  86H          ; UEBERWACHUNG AUF 1
                          ; ODER-UEBERWACHUNG, MASKE FOLGT
        LD   A,1110001B   ; BIT 2, 3, 4 UEBERWACHEN
        OUT  86H
        .
        .
ITAB:   DA   HAVPR        ; INTERRUPTSPRUNGZIELTABELLE
        .                 ; HAVPR = INT. BEHANDLUNGSPROGRAMM
HAVPR:  .                 ; BEGINN HAVARIEPROGRAMM
        .
        .
```

Ü b u n g e n

Ü 5. Über Port B eines PIO-Schaltkreises ist eine byteweise Ausgabe zu realisieren.

Der Ausgabeprogrammzweig trägt den symbolischen Namen AUSGA. Sobald von der peripheren Einrichtung ein Byte ausgegeben wurde, soll die Fertigmeldung ein Interrupt auslösen und so von der ZVE in den genannten Programmzweig gesprungen werden, um weitere Byte auszugeben. Programmieren Sie hierzu den PIO-Schaltkreis.

Ü 6. Acht Tasten sind an die Bit des Port B eines PIO-Schaltkreises angeschlossen.

Nach dem Einstellen dieser Tasten (entspricht frei wählbarer Bitkombination) kann durch das Drücken einer Sondertaste Interrupt ausgelöst werden (entspricht Fertigmeldung).

Die eingestellte Bitkombination ist einzulesen und über Port A unverändert auszugeben. Die Bit des Port A sind mit Anzeigedioden direkt verbunden, so daß die Bitkombination visuell sichtbar wird.
Stellen Sie dazu das erforderliche Programm auf.

4.3. Programmierung einer Meßwertübernahme

An einen PIO-Schaltkreis mit den E/A-Adressen

84H Daten Port A
85H Daten Port B

ist ein Meßgerät angeschlossen, das einen 8-Bit-Wert an den Port B liefert. Das Gerät wird zur Übergabe dieses Meßwertes veranlaßt, wenn es über Bit 0 von Port A ein Startsignal erhält. Nachdem ein Meßwert vollständig an den PIO-Schaltkreis übergeben wurde, löst dieser Interrupt aus. In der Interruptbehandlungsroutine erfolgt dann das Einlesen des Meßwertes. Von jeweils 16 Meßwerten ist der Mittelwert zu bilden. Es wird vorausgesetzt, daß die Programmierung des PIO-Schaltkreises bereits erfolgt ist.

```
        ; PROGRAMM MESSWERTUEBERNAHME
        ; MESSWERTSUMME = 0, ZAEHLER = 16
        XOR A               ; AKKU LOESCHEN
        LD  D,A             ; REG DE FUER MESSWERTSUMME
        LD  E,A             ; LOESCHEN
        LD  B,16            ; ANZAHL = 16
        ; STARTSIGNAL ZUR MESSWERTUEBERGABE
ZYK1:   LD  A,1             ; PIN 0, PORT A = 1
        OUT 84H             ; ALS STARTSIGNAL AUSGEBEN
        HALT                ; WARTEN AUF INTERRUPT
                            ; (AUF VORHANDENEN MESSWERT)
        ; MESSWERTSUMME BILDEN
        ; HIER ERFOLGT FORTSETZUNG NACH RUECKKEHR AUS DER
        ; INTERRUPTBEHANDLUNGSROUTINE
        LD   HL,(WERT)      ; MESSWERT LADEN
        ADD  HL,DE          ; + ALTE SUMME
        LD   D,H            ; = NEUE MESSWERTSUMME
        LD   E,L            ;
        DJNZ ZYK1- #        ; 16 MAL, BIS REG B = 0
; MITTELWERT BILDEN
ZYK2:   LD   B,4            ; VERSCHIEBEZAEHLER LADEN (4-MAL
                            ; ENTSPRICHT DIVISION MIT 16)
        SRA  D              ; RECHTSVERSCHIEBEN HOEHERWERTIGER
                            ; TEIL - C-BIT WIRD GELADEN
        RR   E              ; RECHTSVERSCHIEBEN NIEDERWERTIGER
                            ; TEIL - C-BIT WIRD NACHGEZOGEN
                            ; DADURCH VERSCHIEBUNG IM DOPPEL-
                            ; REGISTER DE
        DJNZ ZYK2- #        ; 4 MAL VERSCHIEBEN
        LD   (MIWE),DE      ; MITTELWERT ABSPEICHERN
        •                   ; WEITERE VERARBEITUNG
        •
        •
; INTERRUPTBEHANDLUNGSROUTINE
IBEH:   IN   85H            ; EINLESEN MESSWERT
        LD   (WERT),A       ; ZWISCHENSPEICHERN (AUFFUELLEN
                            ; AUF 16 BIT)
        EI                  ; INTERRUPT WIEDER ERLAUBEN
```

```
        RETI                ; RUECKSPRUNG MIT RUECKSETZEN
                            ; PERIPHERIESCHALTKREIS
WERT:   BER 1               ; MESSWERT
        DB  0               ; HOEHERWERTIGER TEIL VON WERT
MIWE:   BER 2               ; MITTELWERT
        END
```

Lösungen der Übungen

Ü 1. a,g und i
b und h
d und e

Ü 2. a,d,e und f

Ü 3.

KON5	00100000
KON6	01000001
KON7	11111111
KON8	00110001
KON9	00110001

Ü 4. $t = 256 \text{ (V)} \times 255 \text{ (KO)} \times 241 \text{ (K1)} \times 143 \text{ (K2)} \times 0{,}4 \text{ (Taktzeit)} \times 10^{-6}$

V - Vorteiler
KO - Zeitkonstante Kanal 0
K1 - Zählwert Kanal 1
K2 - Zählwert Kanal 2

```
        LD   A,0D5      ; 11010101 Kanalsteuerwort
        OUT  82H        ; Ausgabe an Kanal 2
        LD   A,8FH      ; Zählkonstante 143 an Kanal 2
        OUT  82H        ; ausgeben

        LD   A,55H      ; 01010101 Kanalsteuerwort
        OUT  81H        ; Ausgabe an Kanal 1
        LD   A,0F1H     ; Zählkonstante 241 an Kanal 1
        OUT  81H        ; ausgeben

        LD   A,0D8H     ; 11011000 Interruptvektor
        OUT  80H        ; Ausgabe an Kanal 0
        LD   A,35H      ; 00110101 Kanalsteuerwort
        OUT  80H        ; Ausgabe an Kanal 0
        LD   A,0FFH     ; Zeitkonstante 255
```

t = 899,89 s

Ü 5.

```
        LD   A,0FFH     ; Laden I-Reg mit HW-Adresse
        LD   I,A        ; der Interrupttabelle

        LD   A,0FEH     ; Interruptvektor über
        OUT  87H        ; Kanal B ausgeben

        LD   A,3FH      ; Betriebsartensteuerwort für
        OUT  87H        ; byteweise Ausgabe
        LD   A,83H      ; Interruptsteuerwort
        OUT  87H        ; Interrupt erlaubt
        .
        .
        .
        ORG  0FFFEH     ; Interrupttabelle
ITAB:   DA   (AUSGA)    ; Sprungziel bei Interrupt
```

Ü 6.

```
        PN   P1
        LD   SP,65534   ; RAM-Kelleradresse laden
        LD   A,0FFH     ; Laden I-Reg mit HW-Adresse
        LD   I,A        ; der Interrupttabelle
        LD   A,4FH      ; Betriebsartensteuerwort
        OUT  87H        ; für Eingabe über Port B
        LD   A,0FEH     ; NW-Adresse des Interruptvektors
        OUT  87H        ; über Port B ausgeben
        LD   A,83H      ; Interruptsteuerwort
        OUT  87H        ; Interrupt erlaubt
        LD   A,0FH      ; Betriebsartensteuerwort
        OUT  86H        ; für Ausgabe über Port A
        IM2             ; Interruptmodus 2
        EI              ; Interrupt erlaubt
EING:   HALT            ; Warten auf Tastaturinterrupt
        OUT  84H        ; Anzeige der gesetzten Bits
        JR   EING-#     ; nächste Eingabe
IBEH:   IN   85H        ; Eingabe Tastaturbits
        EI              ; neues Interrupt zulassen
        RETI            ; Ende Interruptbehandlung
        ORG  0FFEH      ; Interrupttabelle
        DA   (IBEH)     ; Sprungziel bei Tastaturinterrupt
        END
```

Literaturverzeichnis

Der Inhalt dieses Lehrbriefes stützt sich auf:

1. Betriebsdokumentation des Mikrorechners K 1520 des VEB Robotron-Elektronik Zella-Mehlis

2. Sprachbeschreibung Assemblersprache SYPS K 1520
   Systemunterlagendokumentation
   VEB Zentrum für Forschung und Technik im
   Kombinat Robotron

Anlage 1

Befehlsliste des MR K 1520

In den nachfolgenden Befehlstafeln gelten als vereinbart:

r - Register A, B, C, D, E, H, L
n - Direktwert im Wertebereich 0 ... 255 (8 Bit)
nn - Direktwert im Wertebereich 0 ... 65535 (16 Bit)
M - Speicherplatz, dessen Adresse in HL steht
- entspricht (HL)
d - Fortschaltung einer Adresse (- 128 ... + 127)
e - relative Sprungadresse (- 128 rückwärts ... + 127 vorwärts)
x - Flag wird durch Befehl beeinflußt
. - Flag wird durch Befehl nicht beeinflußt
? - Flag nach dem Befehl undefiniert
V - P/V-Flag zeigt arithmetischen Überlauf an
P - P/V-Flag zeigt gerade Parität an
IFF - Interruptflipflop

Tafel 4: Ladebefehle für 1 Byte (8 Bit)

Pos.	Mnemonik	Flags C	Z	P/V	S	Anzahl Byte	Takte	Bemerkungen
1	LD r_1,r_2	.	.	.	.	1	4	Umspeichern Registerinhalte
2	LD r,n	.	.	.	.	2	7	Laden Register mit 8-Bit Direktwert
3	LD r,M M = (HL)	.	.	.	.	1	7	Laden Register mit Speicherbyte
4	LD r,(IX+d)	.	.	.	.	3	19	
5	LD r,(IY+d)	.	.	.	.	3	19	
6	LD A,(BC)	.	.	.	.	1	7	
7	LD A,(DE)	.	.	.	.	1	7	
8	LD A,(nn)	.	.	.	.	3	13	
9	LD M,r	.	.	.	.	1	7	Laden Speicherbyte
10	LD (IX+d),r	.	.	.	.	3	19	mit Registerinhalt
11	LD (IY+d),r	.	.	.	.	3	19	
12	LD (BC),A	.	.	.	.	1	7	
13	LD (DE),A	.	.	.	.	1	7	
14	LD (nn),A	.	.	.	.	3	13	
15	LD M,n	.	.	.	.	2	19	Laden Speicherbyte
16	LD (IX+d),n	.	.	.	.	4	19	mit 8-Bit Direktwert
17	LD (IY+d),n	.	.	.	.	4	19	
18	LD A,I	.	x	IFF	x	2	9	Umspeichern Interrupt-
19	LD I,A	.	x	IFF	x	2	9	vektorregister
20	LD A,R	.	.	.	.	2	9	Umspeichern Refresh-
21	LD R,A	.	.	.	.	2	9	register

Tafel 5: Ladebefehle für 2 Byte (16 Bit)

Pos.	Mnemonik	Flags C	Z	P/V	S	Anzahl Byte	Takte	Bemerkungen
1	LD BC, nn	.	.	.	.	3	10	Laden Doppelregister,
2	LD DE, nn	.	.	.	.	3	10	Indexregister und
3	LD HL, nn	.	.	.	.	3	10	Stackpointer mit
4	LD SP, nn	.	.	.	.	3	10	16-Bit Direktwert
5	LD IX, nn	.	.	.	.	4	14	
6	LD IY, nn	.	.	.	.	4	14	
7	LD HL,(nn)	.	.	.	.	3	16	Laden Doppelregister,
8	LD BC,(nn)	.	.	.	.	4	20	Indexregister und
9	LD DE,(nn)	.	.	.	.	4	20	Stackpointer mit 2
10	LD SP,(nn)	.	.	.	.	4	20	Speicherbyte
11	LD IX,(nn)	.	.	.	.	4	20	
12	LD IY,(nn)	.	.	.	.	4	20	
13	LD (nn),HL	.	.	.	.	3	16	Laden 2 Speicherbyte
14	LD (nn),BC	.	.	.	.	4	20	mit dem Inhalt von
15	LD (nn),DE	.	.	.	.	4	20	Doppelregistern, Index-
16	LD (nn),SP	.	.	.	.	4	20	registern und Stack-
17	LD (nn),IX	.	.	.	.	4	20	pointer

Pos.	Mnemonik	Flags C	Z	P/V	S	Anzahl Byte	Takte	Bemerkungen
18	LD (nn),IY	.	.	.	.	4	20	
19	LD SP,HL	.	.	.	.	1	6	Laden Stackpointer
20	LD SP,IX	.	.	.	.	2	10	
21	LD SP,IY	.	.	.	.	2	10	

Tafel 6: Arithmetikbefehle für 1 Byte (8 Bit)

Pos.	Mnemonik	Flags C	Z	P/V	S	Anzahl Byte	Takte	Bemerkungen
1	ADD A,r	x	x	v	x	1	4	Addition Akkumulator mit Register
2	ADD A,n	x	x	v	x	2	7	Addition Akkumulator mit 8-Bit Direktwert
3	ADD A,M	x	x	v	x	1	7	Addition Akkumulator
4	ADD A,(IX+d)	x	x	v	x	3	19	mit Speicherbyte
5	ADD A,(IY+d)	x	x	v	x	3	19	
6	ADC A,r	x	x	v	x	1	4	Addition Akkumulator mit Register und C-Flag
7	ADC A,n	x	x	v	x	2	7	Addition Akkumulator mit 8-Bit Direktwert und C-Flag
8	ADC A,M	x	x	v	x	1	7	Addition Akkumulator
9	ADC A,(IX+d)	x	x	v	x	3	19	mit Speicherbyte und
10	ADC A,(IY+d)	x	x	v	x	3	19	C-Flag
11	SUB r	x	x	v	x	1	4	Subtraktion Akkumulator mit Register
12	SUB n	x	x	v	x	2	7	Subtraktion Akkumulator mit 8-Bit Direktwert
13	SUB M	x	x	v	x	1	7	Subtraktion Akkumulator
14	SUB (IX+d)	x	x	v	x	3	19	mit Speicherbyte
15	SUB (IY+d)	x	x	v	x	3	19	
16	SBC A,r	x	x	v	x	1	4	Subtraktion Akkumulator mit Register und C-Flag
17	SBC A,n	x	x	v	x	2	7	Subtraktion Akkumulator mit 8-Bit Direktwert und C-Flag
18	SBC A,M	x	x	v	x	1	7	Subtraktion Akkumulator
19	SBC A,(IX+d)	x	x	v	x	3	19	mit Speicherbyte
20	SBC A,(IY+d)	x	x	v	x	3	19	und C-Flag

Tafel 7: Arithmetikbefehle für 2 Byte (16 Bit)

Pos.	Mnemonik	Flags C	Z	P/V	S	Anzahl Byte	Takte	Bemerkungen
1	ADD HL,BC	x	.	.	.	1	11	Addieren Doppelregi-
2	ADD HL,DE	x	.	.	.	1	11	ster, Indexregister
3	ADD HL,HL	x	.	.	.	1	11	und Stackpointer
4	ADD HL,SP	x	.	.	.	1	11	
5	ADD IX,IX	x	.	.	.	2	15	
6	ADD IX,BC	x	.	.	.	2	15	
7	ADD IX,DE	x	.	.	.	2	15	
8	ADD IX,SP	x	.	.	.	2	15	
9	ADD IY,IY	x	.	.	.	2	15	
10	ADD IY,BC	x	.	.	.	2	15	
11	ADD IY,DE	x	.	.	.	2	15	
12	ADD IY,SP	x	.	.	.	2	15	
13	ADC HL,BC	x	x	v	x	2	15	Addieren Doppelregi-
14	ADC HL,DE	x	x	v	x	2	15	ster und Stackpointer
15	ADC HL,HL	x	x	v	x	2	15	unter Berücksichtigung
16	ADC HL,SP	x	x	v	x	2	15	C-Flag
17	SBC HL,BC	x	x	v	x	2	15	Subtrahieren Doppelre-
18	SBC HL,DE	x	x	v	x	2	15	gister und Stackpoin-
19	SBC HL,HL	x	x	v	x	2	15	ter unter Berücksich-
20	SBC HL,SP	x	x	v	x	2	15	tigung C-Flag

Tafel 8: Inkrement- und Dekrementbefehle

Pos.	Mnemonik	Flags C	Z	P/V	S	Anzahl Byte	Takte	Bemerkungen
1	INC r	.	x	v	x	1	4	Erhöhen Register um 1
2	INC M	.	x	v	x	1	11	Erhöhen Speicherbyte
3	INC (IX+d)	.	x	v	x	3	23	um 1
4	INC (IY+d)	.	x	v	x	3	23	
5	INC BC	.	.	.	.	1	6	Erhöhen Doppelregister,
6	INC DE	.	.	.	.	1	6	Indexregister und
7	INC HL	.	.	.	.	1	6	Stackpointer um 1
8	INC SP	.	.	.	.	1	6	
9	INC IX	.	.	.	.	2	10	
10	INC IY	.	.	.	.	2	10	
11	DEC r	.	x	v	x	1	4	Vermindern Register um 1
12	DEC M	.	x	v	x	1	11	Vermindern Speicher-
13	DEC (IX+d)	.	x	v	x	3	23	byte um 1
14	DEC (IY+d)	.	x	v	x	3	23	
15	DEC BC	.	.	.	.	1	6	Vermindern Doppelregi-
16	DEC DE	.	.	.	.	1	6	ster, Indexregister
17	DEC HL	.	.	.	.	1	6	und Stackpointer um 1
18	DEC SP	.	.	.	.	1	6	
19	DEC IX	.	.	.	.	2	10	
20	DEC IY	.	.	.	.	2	10	

Tafel 9: Logikbefehle

Pos.	Mnemonik	Flags C	Z	P/V	S	Anzahl Byte	Takte	Bemerkungen
1	AND r	0	x	P	x	1	4	logisches UND mit Register
2	AND n	0	x	P	x	2	7	mit 8-Bit Direktwert
3	AND M	0	x	P	x	1	7	mit Speicherbyte
4	AND (IX+d)	0	x	P	x	3	19	
5	AND (IY+d)	0	x	P	x	3	19	
6	OR r	0	x	P	x	1	4	logisches ODER mit Register
7	OR n	0	x	P	x	2	7	mit 8-Bit Direktwert
8	OR M	0	x	P	x	1	7	mit Speicherbyte
9	OR (IX+d)	0	x	P	x	3	19	
10	OR (IY+d)	0	x	P	x	3	19	
11	XOR r	0	x	P	x	1	4	logisches EXKLUSIV-ODER mit Register
12	XOR n	0	x	P	x	2	7	mit 8-Bit Direktwert
13	XOR M	0	x	P	x	1	7	mit Speicherbyte
14	XOR (IX+d)	0	x	P	x	3	19	
15	XOR (IY+d)	0	x	P	x	3	19	

Tafel 10: Verschiebebefehle (Rotation)

Pos.	Mnemonik	Flags C	Z	P/V	S	Anzahl Byte	Takte	Bemerkungen
1	RLC r	x	x	P	x	2	8	Linksrotation - Register A,B,C,D,E,H,L
2	RLC M	x	x	P	x	2	15	-Speicherbyte
3	RLC (IX+d)	x	x	P	x	4	23	
4	RLC (IY+d)	x	x	P	x	4	23	
5	RLCA	x	.	.	.	1	4	-Akkumulator
								Linksrotation unter Einbeziehung C-Flag
6	RL r	x	x	P	x	2	8	-Register A,B,C,D,E,H,L
7	RL M	x	x	P	x	2	15	-Speicherbyte (HL)
8	RL (IX+d)	x	x	P	x	4	23	
9	RL (IY+d)	x	x	P	x	4	23	
10	RLA	x	.	.	.	1	4	-Akkumulator
								Rechtsrotation
11	RRC r	x	x	P	x	2	8	-Register A,B,C,D,E,H,L
12	RRC M	x	x	P	x	2	15	-Speicherbyte
13	RRC (IX+d)	x	x	P	x	4	23	
14	RRC (IY+d)	x	x	P	x	4	23	
15	RRCA	x	.	.	.	1	4	-Akkumulator
								Rechtsrotation unter Einbeziehung C-Flag
16	RR r	x	x	P	x	2	8	-Register A,B,C,D,E,H,L
17	RR M	x	x	P	x	2	15	-Speicherbyte
18	RR (IX+d)	x	x	P	x	4	23	

Pos.	Mnemonik	Flags C	Z	P/V	S	Anzahl Byte	Takte	Bemerkungen
19	RR (IY+d)	x	x	P	x	4	23	
20	RRA	x	.	.	.	1	4	-Akkumulator

Tafel 11: Verschiebebefehle

Pos.	Mnemonik	Flags C	Z	P/V	S	Anzahl Byte	Takte	Bemerkungen
								arithmetisches Linksverschiebung
1	SLA r	x	x	P	x	2	8	-Register A,B,C,D,E,H,L
2	SLA M	x	x	P	x	2	15	-Speicherbyte
3	SLA (IX+d)	x	x	P	x	4	23	
4	SLA (IY+d)	x	x	P	x	4	23	
								arithmetische Rechtsverschiebung vorzeichengerecht
5	SRA r	x	x	P	x	2	8	-Register A,B,C,D,E,H,L
6	SRA M	x	x	P	x	2	15	-Speicher
7	SRA (IX+d)	x	x	P	x	4	23	
8	SRA (IY+d)	x	x	P	x	4	23	
								logische Rechtsverschiebung
9	SRL r	x	x	P	x	2	8	-Register A,B,C,D,E,H,L
10	SRL M	x	x	P	x	2	15	-Speicher
11	SRL (IX+d)	x	x	P	x	4	23	
12	SRL (IY+d)	x	x	P	x	4	23	
13	RLD	.	x	P	x	2	18	Halbbytetausch links A0...A3 wird M0...M3 M0...M3 wird M4...M7 M4...M7 wird A0...A3
14	RRD	.	x	P	x	2	18	Halbbytetausch rechts A0...A3 wird M4...M7 M4...M7 wird M0...M3 M0,,,M3 wird A0...A3

Tafel 12: Vergleichsbefehle

Pos.	Mnemonik	Flags C	Z	P/V	S	Anzahl Byte	Takte	Bemerkungen
1	CMP r	x	x	v	x	1	4	Vergleich Akkumulator mit Register
2	CMP n	x	x	v	x	2	7	Vergleich Akkumulator mit Direktwert
3	CMP M	x	x	v	x	1	7	Vergleich Akkumulator
4	CMP (IX+d)	x	x	v	x	5	19	mit Speicherbyte
5	CMP (IY+d)	x	x	v	x	5	19	

Tafel 13: Bedingte und unbedingte Sprungbefehle

Pos.	Mnemonik	Flags C	Z	P/V	S	Anzahl Byte	Takte	Bemerkungen
1	JMP nn	.	.	.	.	3	10	unbedingter Sprung nach einer Speicheradresse
2	JMP M	.	.	.	.	1	4	
3	JMP (IX)	.	.	.	.	2	8	
4	JMP (IY)	.	.	.	.	2	8	
5	JR e	.	.	.	.	2	10	unbedingter Sprung mit relativer Adressierung (Bereich: - 128 ... + 127 Byte)
								bedingter Sprung nach einer Speicheradresse
6	JPNZ nn	.	.	.	.	3	10	- Z-Flag = 0
7	JPZ nn	.	.	.	.	3	10	- Z-Flag = 1
8	JPNC nn	.	.	.	.	3	10	- C-Flag = 0
9	JPC nn	.	.	.	.	3	10	- C-Flag = 1
10	JPPO nn	.	.	.	.	3	10	- P/V-Flag = 0
11	JPPE nn	.	.	.	.	3	10	-P/V-Flag = 1
12	JPP nn	.	.	.	.	3	10	- S-Flag = 0
13	JPM nn	.	.	.	.	3	10	- S-Flag = 1
								bedingter Sprung mit relativer Adressierung (Bereich: - 128 ... + 127 Byte)
14	JRNZ e	.	.	.	.	2	12	- Z-Flag = 0
15	JRZ e	.	.	.	.	2	12	- Z-Flag = 1
16	JRNC e	.	.	.	.	2	12	- C-Flag = 0
17	JRC e	.	.	.	.	2	12	- C-Flat = 1
18	DJNZ e	.	.	.	.	2	13	bedingte Verzweigung mit relativer Adressierung solange REG B ≠ 0 ist

Die angegebenen Taktzeiten beziehen sich auf die erfüllten Verzweigungsbedingungen. Bei nicht durchzuführenden Verzweigungen sind die Verarbeitungszeiten 5 Takte geringer.

Tafel 14: Unterprogrammaufrufe und Rücksprungbefehle

Pos.	Mnemonik	Flags C	Z	P/V	S	Anzahl Byte	Takte	Bemerkungen
1	CALL nn	.	.	.	.	3	17	unbedingter Aufruf eines Unterprogramms
								bedingter Aufruf eines Unterprogramms
2	CNZ nn	.	.	.	.	3	17	- Z-Flag = 0
3	CZ nn	.	.	.	.	3	17	- Z-Flag = 1
4	CNC nn	.	.	.	.	3	17	- C-Flag = 0
5	CC nn	.	.	.	.	3	17	- C-Flag = 1
6	CPO nn	.	.	.	.	3	17	- P/V-Flag = 0

Pos.	Mnemonik	Flags C	Z	P/V	S	Anzahl Byte	Takte	Bemerkungen
7	CAPE, nn	.	.	.	.	3	17	- P/V-Flag = 1
8	CAP, nn	.	.	.	.	3	17	- S-Flag = 0
9	CAM, nn	.	.	.	.	3	17	- S-Flag = 1
10	RET	.	.	.	.	1	10	unbedingter Rücksprung
11	R[illegible]	.	.	.	.	1	11	bedingter Rücksprung - Z-Flag = 0
12	RZ	.	.	.	.	1	11	- Z-Flag = 1
13	RNC	.	.	.	.	1	11	- C-Flag = 0
14	RC	.	.	.	.	1	11	- C-Flag = 1
15	RPO	.	.	.	.	1	11	- P/V-Flag = 0
16	RPE	.	.	.	.	1	11	- P/V-Flag = 1
17	RP	.	.	.	.	1	11	- S-Flag = 0
18	RM	.	.	.	.	1	11	- S-Flag = 1
19	RETI	.	.	.	.	2	14	unbedingter Rücksprung nach maskierbarem Interrupt
20	RETN	.	.	.	.	2	14	unbedingter Rücksprung nach nichtmaskierbarem Interrupt
21	RST 0	.	.	.	.	1	11	spezieller unbedingter Unterprogrammaufruf
22	RST 8	.	.	.	.	1	11	
23	RST 10 H	.	.	.	.	1	11	
24	RST 18 H	.	.	.	.	1	11	
25	RST 20 H	.	.	.	.	1	11	
26	RST 28 H	.	.	.	.	1	11	
27	RST 30 H	.	.	.	.	1	11	
28	RST 38 H	.	.	.	.	1	11	

Die angegebenen Taktzeiten gelten für erfüllte Verzweigungsbedingungen. Findet keine Verzweigung statt, sind die Verarbeitungszeiten 7 (Aufruf) bzw. 6 (Rücksprung) Takte geringer.

Tafel 15: Stack- und Registertauschbefehle

Pos.	Mnemonik	Flags C	Z	P/V	S	Anzahl Byte	Takte	Bemerkungen
1	PUSH dd	.	.	.	.	1	11	Einkellern dd = BC,DE, HL,AF
2	PUSH ii	.	.	.	.	2	15	ii = IX,IY $(SP-1) := dd_H$ $(SP-2) := dd_L$
3	POP dd	.	.	.	.	1	10	Auskellern dd = BC,DE,HL,AF
4	POP ii	.	.	.	.	2	14	ii = IX,IY $dd_L := (SP)$ $dd_H := (SP+1)$
		POP AF veränderte Flags						

Pos.	Mnemonik	Flags C	Z	P/V	S	Anzahl Byte	Takte	Bemerkungen
5	EXAF	x	x	x	x	1	4	Tausch mit 2. Registersatz AF mit AF'
6	EXX	.	.	.	.	1	4	BC mit BC' DE mit DE' HL mit HL'
7	EX DE,HL	.	.	.	.	1	4	Tausch DE mit HL
								Tausch Stackinhalt mit Doppel-/Indexregister
8	EX(SP),HL	.	.	.	.	1	19	H mit (SP+1) L mit (SP)
9	EX(SP),IX	.	.	.	.	2	23	IX_H mit (SP+1) IX_L mit (SP)
10	EX(SP),IY	.	.	.	.	2	23	IY_H mit (SP+1) IY_L mit (SP)

Tafel 16: Blocktransport- und Blocksuchbefehle

Pos.	Mnemonik	Flags C	Z	P/V	S	Anzahl Byte	Takte	Bemerkungen
								Transportieren Zeichenmenge mit zyklischer Unterbrechung
1	LDI	.	.	x	.	2	16	-vorwärts
2	LDD	.	.	x	.	2	16	-rückwärts
								geschlossener Blocktransport
3	LDIR	.	.	0	.	2	21	-vorwärts
4	LDDR	.	.	0	.	2	21	-rückwärts
								Durchsuchen Zeichenmenge mit zyklischer Unterbrechung
5	CPI	.	x	x	x	2	16	-vorwärts
6	CPD	.	x	x	x	2	16	-rückwärts
								Geschlossenes Durchsuchen einer Zeichenmenge
7	CPIR	.	x	x	x	2	21	-vorwärts
8	CPDR	.	x	x	x	2	21	-rückwärts

Die angegebenen Taktzeiten beziehen sich jeweils nur auf 1 Byte der zu transportierenden bzw. zu durchsuchenden Datenmenge.

Tafel 17: Ein- und Ausgabebefehle

Pos.	Mnemonik	Flags C	Z	P/V	S	Anzahl Byte	Takte	Bemerkungen
1	IN	.	.	.	.	2	11	Eingabe Einzelbyte über Kanaladresse in den Akku
2	IN	.	x	P	x	2	12	Eingabe Einzelbyte in ein Register A,B,D,E, H,L
								geschosssene Blockeingabe
3	INIR	.	1	?	?	2	21	-vorwärts
4	INDR	.	1	?	?	2	21	-rückwärts
								Eingabe mehrerer Byte mit zyklischer Unterbrechung
5	INI	.	x	?	?	2	16	-vorwärts
6	IND	.	x	?	?	2	16	-rückwärts
7	OUT	.	.	.	.	2	11	Ausgabe Einzelbyte (Akku-Inhalt) über Kanaladresse
8	OUT	.	.	.	.	2	12	Ausgabe Registerinhalt A,B,C,D,E,H,L
								geschossene Blockausgabe
9	OTIR	.	1	?	?	2	21	-vorwärts
10	OTDR	.	1	?	?	2	21	-rückwärts
								Ausgabe mehrerer Byte mit zyklischer Unterbrechung
11	OUTI	.	x	?	?	2	16	-vorwärts
12	OUTD	.	x	?	?	2	16	-rückwärts

Tafel 18: Spezielle Befehle (Bitmanipulation, Dezimalkorrektur)

Pos.	Mnemonik	Flags C	Z	P/V	S	Anzahl Byte	Takte	Bemerkung
1	BIT b,r	.	x	?	?	2	8	Abtesten des Bit b - im Register A,B,C, D,E,H,L
2	BIT b,M	.	x	?	?	2	12	- im Speicherbyte
3	BIT b,(IX+d)	.	x	?	?	4	20	
4	BIT b,(IY+d)	.	x	?	?	4	20	
5	SET b,r	.	.	.	.	2	8	Setzen des Bit b -im Register A,B,C,D, E,H,L
6	SET b,M	.	.	.	.	2	15	- im Speicherbyte
7	SET b,(IX+d)	.	.	.	.	4	23	
8	SET b,(IY+d)	.	.	.	.	4	23	

Pos.	Mnemonik	Flags C	Z	P/V	S	Anzahl Byte	Takte	Bemerkungen
9	RES b,r	.	.	.	.	2	8	Löschen des Bit b -im Register A,B,C,D,E, H,L
10	RES b,M	.	.	.	.	2	15	im Speicherbyte
11	RES b,(IX+d)	.	.	.	.	4	23	
12	RES b,(IY+d)	.	.	.	.	4	23	
13	DAA	x	x	P	x	1	4	Dezimalkorrektur (Befehl muß nach Addition und Subtraktion von BCD-Operanden im Akkumulator angewendet werden)

Tafel 19: Sonstige spezielle Befehle

Pos.	Mnemonik	Flags C	Z	P/V	S	Anzahl Byte	Takte	Bemerkungen
14	CPL	.	.	.	.	1	4	Bitweise Komplementierung des Akkumulators
15	NEG	x	x	v	x	2	8	Bildung Zweierkomplement im Akkumulator
16	SCF	1	.	.	.	1	4	Setzen C-Flag
17	CCF	x	.	.	.	1	4	Komplementierung C-Flag
18	DI	.	.	.	.	1	4	maskierbaren Interrupt sperren
19	EI	.	.	.	.	1	4	maskierbaren Interrupt freigeben
20	IM 0	.	.	.	.	2	8	Einstellen Interruptmode 0
21	IM 1	.	.	.	.	2	8	Einstellen Interruptmode 1
22	IM 2	.	.	.	.	2	8	Einstellen Interruptmode 2
23	HALT	.	.	.	.	1	4	HALT-Zustand der CPU (Unterbrechung durch Interrupt möglich)
24	NOP	.	.	.	.	.	.	keine Operation