Sprachbeschreibung

Assemblersprache SYPS K 1520

VEB Robotron
Zentrum für Forschung
und Technik

Dok.-Nr. 1.78.019 030.0/78

Die vorliegende Systemunterlagendokumentation entspricht dem Stand vom 30.06.1979.

Die Ausarbeitung der Unterlagen erfolgte durch ein Kollektiv des VEB Robotron, Zentrum für Forschung und Technik.

Im Interesse einer ständigen Weiterentwicklung der System-unterlagen werden alle Leser gebeten, ihre Vorschläge bzw. Hinweise zur Verbesserung dem Herausgeber mitzuteilen.

Herausgeber:
VEB Robotron Zentrum für
Forschung und Technik
801 Dresden, PSF 330 Ag 706/HB/22/83 776-1.0

Vorwort

Die vorliegende Sprachbeschreibung der Assemblersprache SYPS K 1520 bezieht sich auf die Assemblerschreibweise aller durch den Prozessor U 880 realisierten Maschinenbefehle und auf die Pseudooperationen, die durch die Assembler des MRES A 5601 und die Cross-Assembler KRS 4200 und ESER realisiert werden.
Die Bedienanleitung für diese Assembler sind den Dokumentationen

- Bedienungsanleitung MEOS 1521 Dok.-Nr. 1.78.019 040.0/53
- Wirtsrechnersystemunterlagen K 1520 für ESER-Rechner
- Wirtsrechnersystemunterlagen K 1520 für KRS 4200

zu entnehmen.
Darüberhinaus gestatten die einzelnen Assembler zusätzlichen Service und Möglichkeiten, die aus den genannten Schriften zu ersehen sind. Grundlage aller Assembler ist jedoch die voll kompatible Übersetzung in die Maschinensprache des U 880, um damit alle Einsatzfälle des K 1520 programmtechnisch zu unterstützen.

Inhaltsverzeichnis **Seite**

Seite

1. Sprachcharakteristik der Assemblersprache SYPS K 1520

Die Assemblersprache SYPS K 1520 ist eine maschinenorientierte Programmiersprache für das Mikrorechnersystem K 1520. Der wesentliche Vorteil dieser Programmierungsform besteht darin, daß der Programmierer unter Beachtung einfacher Regeln und unter einprägsamen (mnemonischen) Operationscodes und symbolischen Adressen mit größerer Effektivität Programme zu schreiben vermag, als in maschineninterner Form.
Andererseits gestattet die Assemblersprache durch die 1:1 Übersetzung in den Maschinencode die gleiche maximale Zugriffsmöglichkeit zu allen Maschinenfunktionen wie der direkte Maschinencode im Gegensatz zu den höheren problemorientierten Programmiersprachen. Damit ist die Assemblersprache das ideale Programmierwerkzeug für den Systemprogrammierer und für die Programmierung zeitkritischer und/oder speicheroptimaler Programme.
Das Vorhandensein wirkungsvoller Testhilfen und Fehlersuchprogramme (vgl. Bedienungsanleitung des MRES A 5601 Dok.-Nr. 1.78.019 040.0/53) gestattet auch für diese maschinennahe Programmierung einen effektiven Programmtest und die einfache Korrektur der Assembler- und Maschinencodeprogramme. Die Assemblersprache bildet im Zusammenwirken mit leistungsfähigen Betriebssystemen die Voraussetzung zur Schaffung von Übersetzungsprogrammen (Compiler/Interpreter) für höhere problemorientierte Programmiersprachen zur breiten Unterstützung der Anwender des Mikrorechnersystems K 1520.

2. Elemente und Struktur der Sprache

2.1. Eigenschaften und Aufbau

Die Assemblersprache SYPS K 1520 ist eine maschinenorientierte symbolische Programmiersprache. Sie ist auf die Erfordernisse des Systems K 1520 zugeschnitten.
Die Sprache hat folgende Haupteigenschaften:
- freies Format der Quellprogrammzeilen
- Verarbeitung von alphanumerischen Zeichen
- Verarbeitung von zusammengesetzten Ausdrücken im Operandenfeld
- Verarbeitung von Direktoperanden
- Möglichkeit einer abschnittsweisen Programmierung
- Aufrüstung zur MAKRO-Assemblersprache
- Anwendung von Steuerprogrammaufrufen und Pseudoanweisungen

Die Assemblersprache K 1520 besitzt folgende Vorteile:
- symbolische Programmierung unter Beibehaltung der Eigenschaften Flexibilität, Geschwindigkeit und Kürze einer Maschinensprache
- Zuweisung symbolischer Adressen zu Speicherplätzen
- keine Berücksichtigung der Sektoren-Aufteilung des Speichers bei der Programmierung
- Verwendung von Pseudoanweisungen zusätzlich zu den Maschinenbefehlen

Jede Zeile des Quellprogramms, die einen symbolischen Befehl enthält, besteht aus maximal vier Feldern:

Namensfeld
Operationsfeld
Operandenfeld
Kommentarfeld

Da die Assemblersprache SYPS K 1520 nicht formatisiert ist, könne die Felder an beliebiger Stelle der Quellprogrammzeile beginnen. Die vier Felder werden bei Vorhandensein durch Sonder- bzw. Leerzeichen voneinander getrennt. Wird ein Feld nicht belegt, so kann auch das Sonder- bzw. Leerzeichen entfallen. Jede Programmzeile ist durch ein Zeilenendezeichen abzuschließen. Dafür sind zugelassen:

- NL = Neue Zeile (New line)
- CR LF = Wagenrücklauf / Zeilenschaltung (Carriage return / line feed)

Eine Quellprogrammzeile besteht aus max. 71 Zeichen. Dazu kommt das Zeilenendezeichen.

2.1.1. Namensfeld

Das Namensfeld umfaßt 2 bis 6 Zeichen. Es wird benutzt, um einem Befehl oder einer Konstanten eine symbolische Adresse zuzuweisen. Die symbolischen Adressen bestehen aus 1 bis 5 Buchstaben oder Ziffern. Mindestens das 1. Zeichen muß ein Buchstabe sein. Als Abschluß muß in jedem Fall ein Doppelpunkt stehen. Der kleinste Name besteht aus einem Buchstaben und einem Doppelpunkt. Die Zuweisung der Speicherplätze zu den symbolischen Adressen erfolgt bei der Übersetzung.
Registernamen sind für den Assembler reserviert und dürfen nicht als Namen verwendet werden.
Enthält die Quellprogrammzeile kein Namensfeld, beginnt sie sofort mit dem Operationsfeld.

2.1.2. Operationsfeld

Das Operationsfeld umfaßt 2 bis 4 Zeichen. Es enthält den mnemonischen Operationscode, die Pseudoanweisung oder den Makroaufruf.

2.1.3. Operandenfeld

Das Operandenfeld kann je nach dem Charakter des Befehls verschiedenen Inhalt besitzen. Es enthält Informationen, die zusammen mit dem Operationsfeld benutzt werden, um die durch den Befehl auszuführende Operation zu definieren. In Abhängigkeit vom Inhalt des Operationsfeldes kann das Operandenfeld fehlen oder aus einem bzw. aus zwei durch Komma getrennte Operanden bestehen. Das Operandenfeld muß grundsätzlich durch Leerzeichen oder Tabulator vom Operationsfeld getrennt werden. Die indirekte Adressierung wird im Operandenfeld durch das Einklammern der Adresse angewiesen.
Der Aufbau des Operationsfeldes bei den Pseudoanweisungen wird unter Punkt 3.2., Pseudo-/Steueranweisungen, erläutert.
Im Operandenfeld sind, außer in Zeichenketten, Leerzeichen nur vor dem ersten Zeichen des ersten Operanden oder nach Operandenende zugelassen.

2.1.4. Kommentarfeld

Das Kommentarfeld beginnt im Normalfall nach dem Operandenfeld durch Kennzeichnung mit einem Semikolon. Erfolgt in einem Befehl keine Angabe von Operanden, kann das Kommentarfeld sofort nach dem Operationsfeld beginnen. Eine ganze Quellprogrammzeile gilt als Kommentarzeile, wenn als 1. Zeichen ein Semikolon steht. Anschließend können max. 70 Zeichen Kommentar gegeben werden. Den Abschluß eines Kommentarfeldes bzw. einer Kommentarzeile bildet eine Zeilenendezeichen (NL, CR, LF).
Kommentare haben auf die Übersetzung keinen Einfluß, werden aber in der Übersetzungsliste ausgedruckt.
Werden in einer Kommentarzeile vor dem Semikolon mindestens 1 Leerzeichen oder Tabulator erfaßt, so wird diese Zeile als eingerückte Kommentarzeile gewertet. Die Kommentarlänge beträgt dann max. 60 Zeichen.

2.2. Syntax, Semantik und Zeichensatz

2.2.1. Zeichen

Für den Aufbau von Quellprogrammzeilen in der Sprache SYPS K 1520 sind alle druckbaren Zeichen und die Steuerzeichen NL, CR, LF, HT des Codes nach TGL 23207/02 zugelassen. Folgende Zeichen haben in der Sprache SYPS K 1520 eine besondere Bedeutung:

Zeichen	Bedeutung	Name
+ -	Operatoren in Ausdrücken	Plus, Minus
#	laufender Stand des Speicherplatz-zuweisungszählers	Doppelkreuz
,	Trennungszeichen für Elemente der im Operandenfeld angegebenen Operanden und Variablen	Komma
;	Kennzeichnung für Kommentar	Semikolon
'	Begrenzung für Literale (Direktoperanden)	Apostroph
:	Abschluß des Symbols im Namensfeld (Abgrenzung von Namens- und Operationsfeld)	Doppelpunkt
.	Begrenzung von logischen Operationen, Trennung des Programmnamens vom Symbol)	Punkt
( )	Kennzeichen für indirekte oder indizierte Adressierung	Klammern
NL CR LF	Zeilenendekennzeichen bzw. Abschluß jeder Quellprogrammzeile	Neue Zeile Wagenrücklauf/ Zeilenschaltung.
HT	Trennungszeichen	Horizontaltabulator

2.2.2. Zahlendarstellung

Prinzipiell können in der Sprache SYPS K 1520 Dezimal-, Oktal-, Hexadezimal- oder Binärzahlen verwendet werden. Die Kennzeichnung der verwendeten Zahlen erfolgt nach folgendem Modus:

Zahl	Kennzeichen	Beispiel
Dezimal	ohne	161
Oktal	O oder Q	241O 241Q
Hexadezimal	H	0A1H
Binär	B	10100001B

Bei negativem Vorzeichen wird das Zweierkompliment gebildet. Bei Hexadezimalzahlen muß das erste Zeichen eine Ziffer sein.

2.2.3. Mnemoniks

Die symbolischen Abkürzungen der Maschinenbefehle, Pseudoanweisungen und Steuerprogrammaufrufe werden als Mnemoniks bezeichnet. Sie können 2 bis 4 Zeichen umfassen. Mnemoniks werden nur im Operationsfeld angewendet.
Die von der Sprache SYPS K 1520 als Maschinenbefehle zugelassenen Mnemoniks werden unter Pkt. 3, Befehls-/Anweisungsbeschreibung, mit der Funktion des jeweiligen Befehls beschrieben.

2.2.4. Symbole

Symbole können sowohl im Namensfeld als auch im Operandenfeld stehen. Ein Symbol verkörpert die Adresse des Befehls, in dessen Namensfeld er steht. Ausnahmen bilden die durch die Pseudoanweisungen EQU und DEF definierten Symbole. In diesem Fall erhält das Symbol den Wert des im Operandenfeld stehenden Ausdrucks. Taucht ein Symbol zum zweiten Mal im Namensfeld auf, so erfolgt bei der Übersetzung ein Fehlerausdruck im Übersetzungsprotokoll.
Symbole bestehen aus 1 bis 5 Zeichen, von denen das erste Zeichen ein Buchstabe sein muß. Als Zeichen sind die Großbuchstaben von A bis Z, außer Registerbezeichnungen, und die Ziffern 0 bis 9 zugelassen.

Beispiele für Symbole

zulässige Symbole	verbotene Symbole	
A1	A1?	(?)
AB	A	(Registerbezeichnung)
START	STARTE	(6 Zeichen)
PSA3	6PA2	(beginnt mit Ziffer)

2.2.5. Externe Symbole

Externe Symbole erlauben den Zugriff auf Adressen, die symbolisch in einem anderen Programm definiert sind. Externe Symbole bestehen aus dem Programmnamen, gefolgt von einem Punkt und dem Symbol.

Beispiel:

```
          JMP       P2.ZANE
                      |   |
                      |   |____Symbol im Programm P2
                      |
                      |________Programmname
```

Externe Symbole werden als Symbolwert gewertet.

2.2.6. Ausdrücke

Ausdrücke treten nur als Operanden im Operandenfeld auf. Ausdrücke bestehen aus Elementen und Operatoren und eventuell aus dem Kennzeichen für indirekte Adressierung.
Es wird zwischen einfachen Ausdrücken (nur ein Element) und zusammengesetzten Ausdrücken (zwei oder mehr Elemente) unterschieden. Bei Verwendung externer Symbole sind maximal 2 Ele-

mente zugelassen. Die Elemente sind durch Operatoren miteinander verbunden. Es ist nur ein Operator zwischen zwei Elementen zugelassen (+ oder -). Ausdrücke können sowohl Adressen, Verschiebungszahlen als auch Bestandteile von Pseudoanweisungen sein. Eine Ausnahme bilden die Pseudoanweisungen DEF und IF. Im verschieblichen Übersetzungsmodus sind Symbole, soweit sie nicht durch EQU oder DEF anders definiert sind, und das Kennzeichen für den aktuellen Befehlszählerstand (#) verschiebliche Operandenelemente.
Verschiebliche Adressenausdrücke erhalten Werte relativ zur Anfangsadresse des Programms.

Zulässige Elemente:

Symbole	- verschieblich
# (Stand des Speicherplatzzuweisungszählers)	- verschieblich
ganze Dezimalzahlen	- absolut
ganze Oktalzahlen	- absolut
ganze Hexadezimalzahlen	- absolut
Binärzahlen	- absolut

Zulässige Operatoren:

+	Plus
-	Minus

Die Länge eines Ausdruckes wird durch das Operandenfeld beschränkt (max. 61 Zeichen).
Enthält ein Ausdruck mehrere Elemente, so wird zwischen absolut und verschieblich folgendermaßen unterschieden:

- Die Anzahl der positiven verschieblichen Elemente innerhalb eines Ausdruckes muß 0 oder 1 sein.

Anzahl positiver Elemente minus Anzahl negativer Elemente = A

A = 0	absoluter Ausdruck
A = 1	verschieblicher Ausdruck
A ≠ 0 oder 1	falscher Ausdruck

Beispiel:
Berechnung von Ausdrücken

Element	Wert (dezimal)
# (Speicherplatzzuweisungszähler)	200
V (Symbol, verschieblich)	5
START (Symbol, verschieblich)	10

Ausdruck	Wert (dezimal)	Art des Ausdruckes
V + 5	10	verschieblicher Ausdruck
START + 2	12	verschieblicher Ausdruck
#	200	verschieblicher Ausdruck
#- 3	197	verschieblicher Ausdruck
START + # - 2	208	falscher Ausdruck
10 + 144Q	110	absoluter Ausdruck

Beispiele:
Es seien NAME und SYMB verschiebliche Symbole, dann sind:

NAME + 5	verschieblicher Ausdruck
NAME - SYMB	absoluter Ausdruck
NAME + SYMB	falscher Ausdruck
-NAME - SYMB	falscher Ausdruck
SYMB - #	absoluter Ausdruck

2.2.7. Operandenformen

Die in den folgenden Punkten aufgeführten Operandenformen sind alternativ möglich.

2.2.7.1. 2-Byte-Operanden (nn)

1. ± ADR
2. - ADR + ADR
3. Literal
   'x'
   x ist ein druckbares Zeichen aus dem Zeichenvorrat nach TGL 23207/02

ADR kann sein:

- Dezimalwert
  0 ≤ ADR ≤ 65535
- Oktalwert
  0Q ≤ ADR ≤ 177777Q
- Hexadezimalwert
  0H ≤ ADR ≤ 0FFFFH
- Binärwert
  0B ≤ ADR ≤ 1111111111111111B
- Symbol
- externes Symbol
- Aktueller Befehlszählerstand #

Wenn für einen 2-Byte-Operanden nur ein 1-Byte-Wert definiert ist, werden die oberen 8 Bits als 0 angenommen. Ausdrücke werden modulo 1000H berechnet.

2.2.7.2. 1-Byte-Operanden (n)

Folgende Ausdrücke für n sind möglich:

1. 0 ≤ n ≤ 255
2. L(nn) niederwertiger Adreßteil
3. H(nn) höherwertiger Adreßteil

2.2.7.3. Relative Sprungadresse (e)

Bei relativen Sprüngen wird das Sprungziel über den Wert e errechnet.
Für e sind alle Operandenformen, mit Ausnahme des externen Symbols, zugelassen.
Für e muß sich ein absoluter Wert im Bereich

$$-126 \leq e \leq 129$$

ergeben.

2.2.7.4. Verschiebung (d) bei Adressierung über Indexregister

Die Verschiebung d wird verwendet, um die in den Indexregistern IX bzw. IY enthaltene Basisadresse zu modifizieren. Für d sind alle Operandenformen, mit Ausnahme des externen Symbols, zugelassen.
Für d muß sich ein absoluter Wert im Bereich

$$-128 \leq d \leq 127$$

ergeben.

2.2.8. Registerbezeichnungen (r)

Die folgenden Registerbezeichnungen sind im Assembler fest definierte symbolische Adressen und damit für anderweitige Verwendung gesperrt.

Universalregister:

A	A - Register (Akkumulator)	\
B	B - Register	|
C	C - Register	|
D	C - Register	|
E	C - Register	> 1-Byte-Register (r)
H	C - Register	|
L	C - Register	|
M oder HL	Speicherplatz, der durch HL adressiert wird	/

BC	Register B und C	\
DE	Register E und E	> 2-Byte-Register (dd)
HL	Register H und L	/
AF	Register A und F	/

Sonderregister:

I	Interrupt-Vektor-Register	\
R	Refresh-Register	> 1-Byte-Register (r)
F	Flagregister	/
IX	Indexregister	\
IY	Indexregister	> 2-Byte-Register (dd)
SP	Stackpointer	/

Für die Universalregister und das Flagregister existieren Hintergrundregister, die mit Hochkomma gekennzeichnet sind. Der Zugriff zu den Hintergrundregistern ist durch die Registeraustauschbefehle (Pkt. 3.1.10. Registeraustauschbefehle) möglich.

2.2.9. Arbeit mit den Bedingungsbits (Flags)

Das Flagregister (F) gibt über das Ergebnis der letzten Prozessoroperation Auskunft und dient im wesentlichen dazu, bedingte Programmverzweigungen bzw. bedingte Unterprogrammaufrufe oder -rücksprünge auszuführen. Die Stellung der einzelnen Bedingungsbits (Flags) im Flagregister zeigt folgende Skizze:

Flagregister	7	6	5	4	3	2	1	0
	S	Z	X	H	X	P/V	N	CY

S - Vorzeichenbit (Sign-Flag)
Z - Nullbit (Zero-Flag)
H - Halbbyteüberlaufbit (Half-Carry-Flag)
P/V - Paritäts-Übertragsbit (Parity/Overflow-Flag)
N - Additions-Subtraktionsbit (Add/Subtract-Flag)
CY - Übertragsbit (Carry-Flag)
X - nicht belegt

Die Wirkung der einzelnen Befehle auf die jeweiligen Bedingungsflags ist zusammenfassend und übersichtlich in Anlage 3, Zusammenstellung der Befehle nach Funktionsgruppen, dargestellt.

2.2.9.1. Vorzeichenbit (S-Flag)

In das S-Flag wird bei bestimmten Befehlen das höchstwertige Bit des Akkumulatorinhaltes geladen. Bei der Ausführung von arithmetischen Befehlen mit vorzeichenbehafteten Zahlen wird eine positive Zahl durch eine 0 und eine negative Zahl durch eine 1 in der höchstwertigen Bitstelle gekennzeichnet.

2.2.9.2. Nullbit (Z-Flag)

Bei 1-Byte arithmetischen und logischen Operationen wird das Z-Flag gesetzt, wenn das Ergebnisbyte des Akkumulators 0 ist. Sonst wird das Z-Flag zurückgesetzt.
Bei Vergleichs- und Suchbefehlen wird das Z-Flag gesetzt, sobald der Vergleich positiv ausgeht.
Bei dem BIT-Befehl wird das Z-Flag mit dem komplementären Wert des getesteten Bits geladen.
Bei Übertragung eines Bytes zwischen einer Speicherstelle und einer E/A-Schnittstelle (INI, IND, OUTI, OUTD) wird das Z-Flag 1 gesetzt, sobald der Wert des Zählregisters 0 wird.
Bei den Befehlen IN r und INF wird das Z-Flag gesetzt, wenn die eingezogenen bzw. am E-Tor anliegenden Daten den Wert 0 haben.

2.2.9.3. Halbbyte-Übertragsbit (H-Flag)

Das H-Flag wird entsprechend dem Übertragsergebnis zwischen den Bits 3 und 4 einer arithmetischen 1-Byte-Operation gesetzt (falls Übertrag) oder rückgesetzt (falls kein Übertrag). Es wird beim Befehl DAA verwendet, um das Ergebnis einer gepackten BCD-Addition bzw. Subtraktion zu korrigieren.

Das H-Flag wird wie folgt gesetzt bzw. rückgesetzt:

H	Addition	Subtraktion
1	Übertrag von Bit 3 zu Bit 4	negativer Übertrag von Bit 4
0	kein Übertrag von Bit 3 zu Bit 4	kein negativer Übertrag von Bit 4

2.2.9.4. Paritäts-/Überlaufbit (P/V-Flag)

Das P/V-Flag wird unterschiedlich benutzt.
Bei arithmetischen Befehlen wird das P/V-Flag gesetzt, wenn im Ergebnis das höchste Bit des Akkumulators gesetzt wird.
Bei logischen Operationen und Verschiebebefehlen dient das P/V-Flag zur Überprüfung der Parität des Ergebnisses. Ist die Anzahl der gesetzten Bits im angesprochenen Byte gerade (d.h., 0, 2, 4, 6, 8), so wird das P/V-Flag gesetzt. Ansonsten wird es rückgesetzt.
Bei den Blocktransport- (LDI, LDIR, LDD, LDDR) und Blocksuchbefehlen (CPI, CPIR, CPD, CPDR) gibt das P/V-Flag Auskunft über den Stand des Bytezählers.
Das P/V-Flag wird rückgesetzt, wenn nach Dekrementieren des Bytezählers (=<BC>) als Ergebnis 0 entsteht. In allen übrigen Fällen bleibt das P/V-Flag 1.

2.2.9.5. Additions-/Subtraktionsbit (N-Flag)

Das N-Flag wird intern bei dem DAA-Befehl benutzt, um zwischen Additions- und Subtraktionsbefehlen zu unterscheiden. Bei allen Additonsbefehlen wird das N-Flag rückgesetzt, Subtraktionsbefehle setzen das N-Flag.

2.2.9.6. Übertragsbit (CY-Flag)

Das Setzen/Rücksetzen des CY-Flags wird je nach ausgeführter Operation verschieden behandelt.

Das C-Flag wird gesetzt, falls
- bei Additionsbefehlen ein Übertrag
- bei Subtraktionsbefehlen ein negativer Übertrag entsteht.

Es wird zurückgesetzt bei
- Additionsbefehlen, die keinen Übertrag
- Subtraktionsbefehlen, die keinen negativen Übertrag erzeugen.

Bei den Verschiebebefehlen RLA, RRA, RL und RR wird das C-Flag als Zwischenspeicher für die Übertragung des niederwertigsten bzw. höchstwertigen Bits eines CPU-Registers bzw. Speicherplatzes benutzt.
Bei den Befehlen RLCA, RLC und SLA enthält das C-Flag den Wert des höchstwertigen Bits, das durch den Befehl aus dem behandelten Register bzw. Speicherplatz hinausgeschoben wurde.
Bei den Befehlen RRCA, RRC, SRA und SRL enthält das C-Flag analog den Wert des niederwertigsten Bits.

Die logischen Befehle AND, OR und XOR setzen das C-Flag grundsätzlich zurück.
Die speziell für das C-Flag vorgesehenen Befehle SCF (Set C-Flag) und CCF (Complement C-Flag) erlauben das Setzen bzw. Komplementieren des C-Flags.

2.2.10. Adressierungsarten

Die meisten Assemblerbefehle arbeiten mit Daten, die entweder direkt vom Programmierer vorgegeben werden (Direktoperand 1 oder 2 Byte), in internen CPU-Registern (Einzel-, Doppel-, Basisregister) gespeichert sind, auf externen Speichern oder den E/A-Kanälen abgelegt sind.
Welche Adressierungsarten für die einzelnen Befehle zulässig sind, ist Pkt. 3, Befehls-/Anweisungsbeschreibung oder der Anlage 3. Zusammenstellung der Befehle nach Funktionsgruppen, zu entnehmen.

2.2.10.1. 1-Byte-Direktoperand

Der Operand wird direkt im Befehl angegeben.

```
Beispiel:      LD   r,n          (symbolisch)
               LD   A,100
```

2.2.10.2. 2-Byte-Direktoperand

Der Operand wird direkt im Befehl angegeben.

```
Beispiel:      LD   dd,nn        (symbolisch)
               LD   BC,0FF10H
```

2.2.10.3. Registeradressierung

Der Operand steht in einem Register oder Registerpaar.

```
Beispiel:      LD   r1,r2        (symbolisch)
               LD   A,E
               LD   SP,HL
```

2.2.10.4. Speicheradressierung über Register

Die Adresse des Operanden steht in einem Register bzw. Registerpaar. Die Kennzeichnung erfolgt durch Klammerung der Register- bzw. Registerpaarbezeichnung. Für (HL) darf auch M geschrieben werden.

```
Beispiel:      LD   A,(HL)    =^   LD   A,M
               LD   E,(BC)
               LD   A,(DE)
```

2.2.10.5. Direkte Speicheradressierung

Die Speicheradresse des Operanden wird im Befehl als Direktwert oder symbolisch angegeben. Die Kennzeichnung erfolgt durch Klammerung der Adressangabe.

```
Beispiel:      LD   A,(nn)
               LD   A,(MARKE)
```

2.2.10.6. Indizierte Adressierung

Die Basisadresse eines Speicherbereiches steht in einem der Indexregister IX oder IY. Die Modifizierung der Adresse erfolgt durch die Verschiebung d, die im Befehl angegeben wird.
Die Kennzeichnung erfolgt durch Klammerung des Ausdruckes.

```
Beispiel:      LD   A,(IX+16)
               LD   r,(IX+d)
               LD   C,(IY)
```

2.2.10.7 Relative Adressierung

Die relative Adressierung wird nur bei den Sprungbefehlen angewendet und kennzeichnet den Abstand zum gerade anliegenden Befehlszählerstand. Dieser vorzeichenbehaftete Abstand e liegt zwischen + 127 und - 128 und kennzeichnet die absolute Adresse: Befehlszähler des dem relativen Sprungbefehl folgenden Befehl ± e.

```
Beispiel:      JRZ  e
               JRNC -16
               JR   MARKE-#
```

2.3. Pseudoanweisungen

Neben der Angabe von Maschinenbefehlen der CPU in Assemblerschreibweise gestattet die Assemblersprache eine Reihe von Anweisungen, die der Steuerung des Übersetzungsprogramms (Assembler) selbst dienen oder die für eine anwendergerechte Aufbereitung des Übersetzungsprotokolls, die Reservierung von Speicherplätzen oder das Einstellen von Programmladeadressen benötigt werden.
Diese Pseudoanweisungen werden nicht in Maschinenbefehle des K 1520 übersetzt.
Eine Auflistung der umfangreichen Programmiermöglichkeiten über Pseudoanweisungen wird in Pkt. 3.2., Pseudo-/Steueranweisungen, gegeben.

3. Befehls-/Anweisungsbeschreibung

Anlage C enthält die Befehlsliste nach Funktionsgruppen geordnet.

3.1. Maschinenbefehle

3.1.1. Ladebefehle

Die Ladebefehle transportieren Daten intern zwischen den Registern oder zwischen Registern und dem Schreib-/Lesespeicher (RAM). Die Befehle müssen eine Ausgangsadresse und eine Zieladresse enthalten. Der Quellspeicherplatz wird durch den Ladebefehl nicht verändert.

3.1.1.1. 1-Byte-Ladebefehle

Lfd.Nr.	MNEMONIK	Wirkungsweise der Befehle
1	LD r1,r2	Der Inhalt des Registers r2 wird in das Register r1 umgespeichert. (r2 und r1 können die Register A, B, C, D, E, H oder L sein)
2	LD r,n	Ein durch n definiertes Wort oder ein Direktoperand wird in das Register r geladen. (r können die Register A, B, C, D, E, H oder L sein)
3	LD r,M	Der Inhalt des durch Registerpaar HL adressierten Speicherplatzes M wird in das Register r geladen. Das Register L beinhaltet dabei die niederwertigen 8 Bits und das Register H die höherwertigen 9 Bits der Adresse. (r können die Register A, B, C, D, E, H oder L sein)
4	LD r,(IX+d)	Der Inhalt des durch das Register IX plus Verschiebung adressierten Speicherplatzes wird in das Register r geladen. (r können die Register A, B, C, D, E, H oder L sein. d ist die Verschiebung im Zahlenbereich $-128 \leq d \leq 127$)
5	LD r,(IY+d)	Der Inhalt des durch das Register IY plus Verschiebung adressierten Speicherplatzes wird in das Register r geladen. (r können die Register A, B, C, D, E, H oder L sein. d ist die Verschiebung im Zahlenbereich $-128 \leq d \leq 127$)

Lfd.Nr.	MNEMONIK	Wirkungsweise der Befehle
6	LD M,r	Dieser Befehl bringt die Daten aus dem Register r an einen Speicherplatz M, dessen Adresse durch den Inhalt des Registerpaares HL spezifiziert ist. Register L beinhaltet dabei die niederwertigen 8 Bits und Register H die höherwertigen 8 Bits der Adresse. (r können die Register A, B, C, D, E, H oder L sein)
7	LD (IX+d),r	Dieser Befehl bringt die Daten aus dem Register r an einen Speicherplatz, dessen Adresse durch den Inhalt des IX-Registers plus Verschiebung spezifiziert ist. (r können die Register A, B, C, D, E, H oder L sein. d ist die Verschiebung im Zahlenbereich $-128 \leq d \leq 127$)
8	LD (IY+d),r	Dieser Befehl bringt die Daten aus dem Register r an einen Speicherplatz, dessen Adresse durch den Inhalt des IY-Registers plus Verschiebung spezifiziert ist. (r können die Register A, B, C, D, E, H oder L sein. d ist die Verschiebung im Zahlenbereich $-128 \leq d \leq 127$)
9	LD M,n	Dieser Befehl bewirkt den Transport des mit n definierten Wortes oder Direktoperanden an einen Speicherplatz M, dessen Adresse durch den Inhalt des Registerpaares HL definiert ist. (Register L beinhaltet dabei die niederwertigen 8 Bits und das Register H die höherwertigen 8 Bits der Adresse)
10	LD (IX+d),n	Dieser Befehl bewirkt den Transport des mit n definierten Wortes oder Direktoperanden an einen Speicherplatz, dessen Adresse durch den Inhalt des IX-Registers plus Verschiebung spezifiziert ist. (d ist die Verschiebung im Zahlenbereich $-128 \leq d \leq 127$)
11	LD (IY+d),n	Ein mit n definiertes Wort oder ein Direktoperand wird an einen Speicherplatz transportiert, dessen Adresse durch den Inhalt des IY-Registers plus Verschiebung spezifiziert ist. (d ist die Verschiebung im Zahlenbereich $-128 \leq d \leq 127$)
12	LD A,(BC)	Der Inhalt des durch das Registerpaar BC adressierten Speicherplatzes wird in den Akkumulator geladen. Das Register C beinhaltet die niederwertigen 8 Bits und das Register B die höherwertigen 8 Bits der Adresse.

Lfd.Nr.	MNEMONIK	Wirkungsweise der Befehle
13	LD A,(DE)	Der Inhalt des durch das Registerpaar DE adressierten Speicherplatzes wird in den Akkumulator geladen. Das Register E beinhaltet die niederwertigen 8 Bits und das Register D die höherwertigen 8 Bits der Adresse.
14	LD A,(nn)	Der Inhalt des durch nn adressierten Speicherplatzes wird in den Akkumulator geladen, nn kann eine 16-Bit-Konstante oder ein symbolischer Name sein. Das auf den Operationscode folgende Byte stellt den niederwertigen Adreßteil dar.
15	LD (BC),A	Der Inhalt des Akkumulators wird auf den Speicherplatz geladen, dessen Adresse im Registerpaar BC definiert ist. Das Register C beinhaltet die niederwertigen 8 Bits und das Register B die höherwertigen 8 Bits der Adresse.
16	LD (DE),A	Der Inhalt des Akkumulators wird auf den Speicherplatz geladen, dessen Adresse im Registerpaar DE definiert ist. Das Register E beinhaltet die niederwertigen 8 Bits und das Register D die höherwertigen 8 Bits der Adresse.
17	LD (nn),A	Der Inhalt des Akkumulators wird auf den Speicherplatz geladen, der mit nn adressiert ist. nn kann eine 16-Bit-Konstante oder ein symbolischer Name sein. Das auf den Operationscode folgende Byte stellt den niederwertigen Adreßteil dar.
18	LD A,I	Der Registerinhalt von I wird in den Akkumulator geladen. I ist das Interrupt-Vektor-Register.
19	LD A,R	Der Registerinhalt von R wird in den Akkumulator geladen. R ist das Refresh-Register.
20	LD I,A	Der Inhalt des Akkumulators wird in das Interrupt-Vektor-Register geladen.
21	LD R,A	Der Akkumulatorinhalt wird in das Refresh-Register geladen.

3.1.1.2. 2-Byte-Ladebefehle

Lfd.Nr.	MNEMONIK	Wirkungsweise der Befehle
1	LD dd,nn	Eine 16-Bit-Konstante wird in ein Doppelregister geladen. (dd können die Doppelregister BC, DE, HL oder SP sein)
2	LD IX,nn	Eine 16-Bit-Konstante wird in das Register IX geladen.
3	LD IY,nn	Eine 16-Bit-Konstante wird in das Register IY geladen.
4	LD HL,(nn)	Der Inhalt der durch nn und nn+1 adressierten Speicherplätze wird in das Doppelregister HL geladen: Inhalt von nn+1 --> Register H Inhalt von nn --> Register L
5	LD dd,(nn)	Der Inhalt der durch nn und nn+1 adressierten Speicherplätze wird in ein Doppelregister geladen. (dd können die Doppelregister BC, DE, HL oder SP sein) Inhalt von nn+1 --> höherwertiges Regist. (B, D, H, SP_H) Inhalt von nn --> niederwertiges Reg. (C, E, L, SP_L)
6	LD IX,(nn)	Der Inhalt des durch nn und nn+1 adressierten Speicherplätze wird in das Indexregister IX geladen: Inhalt von nn+1 --> Register IX_H Inhalt von nn --> Register IX_L
7	LD IY,(nn)	Der Inhalt des durch nn und nn+1 adressierten Speicherplätze wird in das Indexregister IY geladen: Inhalt von nn+1 --> Register IY_H Inhalt von nn --> Register IY_L
8	LD (nn),HL	Der Inhalt des Doppelregisters HL wird auf die Adressen nn und nn+1 transportiert: Inhalt von Register H --> Inhalt der Adresse nn+1 Inhalt von Register L --> Inhalt der Adresse nn
9	LD (nn),dd	Der Inhalt eines Registerpaares wird auf die Adressen nn und nn+1 transportiert. (dd kann sein: BC, DE, SP, HL) Inhalt des höherwertigen Registers --> Inhalt der Adresse nn (B, D, SP_H, H) Inhalt des niederwertigen Registers --> Inhalt der Adresse nn (C, E, SP_L, L)

Lfd.Nr.	MNEMONIK	Wirkungsweise der Befehle
10	LD (nn),IX	Der Inhalt des Indexregisters IX wird auf die Adressen nn und nn+1 transportiert. Inhalt IX_H --> Inhalt der Adresse nn+1 Inhalt IX_L --> Inhalt der Adresse nn
11	LD (nn),IY	Der Inhalt des Indexregisters IY wird auf die Adressen nn und nn+1 transportiert. Inhalt IY_H --> Inhalt der Adresse nn+1 Inhalt IY_L --> Inhalt der Adresse nn
12	LD SP,IX	Der Inhalt des Doppelregisters HL wird in den Stackpointer übertragen: Inhalt Register H --> SP_H Inhalt Register L --> SP_L
13		Der Inhalt des Indexregisters IX wird in den Stackpointer übertragen: Inhalt Register IX_H --> SP_H Inhalt Register IX_L --> SP_L
14	LD SP,IY	Der Inhalt des Indexregisters IY wird in den Stackpointer übertragen: Inhalt Register IY_H --> SP_H Inhalt Register IY_L --> SP_L

3.1.2. Arithmetische Operationen

Die arithmetischen und logischen Befehle arbeiten mit Daten, die sich im Akkumulator und in anderen Universalregistern oder auf Speicherplätzen befinden. Die Ergebnisse dieser Operationen werden im Akkumulator untergebracht. Entsprechende Flags werden in Abhängigkeit vom Ergebnis der Operationen gesetzt.

3.1.2.1. 1-Byte-Arithmetik

Lfd.Nr.	MNEMONIK	Wirkungsweise der Befehle
1	ADD r	Der Registerinhalt r wird zum Akkumulatorinhalt addiert. (r können die Register A, B, C, D, E, H oder L sein.)
2	ADD M	Der durch das Registerpaar HL adressierte Inhalt des Speicherplatzes M wird zum Inhalt des Akkumulators addiert.
3	ADD n	Ein durch n definiertes Wort oder ein Direktoperand wird zum Inhalt des Akkumulators addiert.

Lfd.Nr.	MNEMONIK	Wirkungsweise der Befehle
4	ADD (IX+d)	Der Inhalt des durch das Register IX plus Adressenverschiebung adressierten Speicherplatzes wird zum Inhalt des Akkumulators addiert. Das Ergebnis steht nach der Operation im Akkumulator. (d ist die Verschiebung im Zahlenbereich $-128 \leq d \leq 127$)
5	ADD (IY+d)	Der Inhalt des durch das Register IY plus Adressenverschiebung adressierten Speicherplatzes wird zum Inhalt des Akkumulators addiert. Das Ergebnis steht nach der Operation im Akkumulator. (d ist die Verschiebung im Zahlenbereich $-128 \leq d \leq 127$)
6	ADC r	Der Registerinhalt r plus Carry-Flag (CY) wird zum Akkumulatorinhalt addiert. (r können die Register A, B, C, D. E, H oder L sein)
7	ADC M	Der durch das Registerpaar HL adressierte Inhalt des Speicherplatzes M plus CY wird zum Inhalt des Akkumulators addiert.
8	ADC n	Ein durch n definiertes Wort oder ein Direktoperand plus CY werden zum Inhalt des Akkumulators addiert.
9	ADC (IX+d)	Der Inhalt des durch das Register IX plus der Adressenverschiebung d adressierten Speicherplatzes plus CY werden zum Inhalt des Akkumulators addiert. (d ist die Verschiebung im Zahlenbereich $-128 \leq d \leq 127$)
10	ADC (IY+d)	Der Inhalt des durch das Register IY plus der Adressenverschiebung d adressierten Speicherplatzes plus CY werden zum Inhalt des Akkumulators addiert. (d ist die Verschiebung im Zahlenbereich $-128 \leq d \leq 127$)
11	SUB r	Der Registerinhalt r wird vom Akkumulatorinhalt subtrahiert. (r können die Register A, B, C, D, E, H oder L sein.)
12	SUB M	Der durch das Registerpaar HL adressierte Inhalt des Speicherplatzes M wird vom Inhalt des Akkumulators subtrahiert.
13	SUB n	Ein durch n definiertes Wort oder ein Direktoperand wird vom Inhalt des Akkumulators subtrahiert.

Lfd.Nr.	MNEMONIK	Wirkungsweise der Befehle
14	SUB (IX+d)	Der Inhalt des durch das Register IX plus Adressenverschiebung adressierten Speicherplatzes wird vom Inhalt des Akkumulators subtrahiert. (d ist die Verschiebung im Zahlenbereich $-128 \leq d \leq 127$)
15	SUB (IY+d)	Der Inhalt des durch das Register IY plus Adressenverschiebung adressierten Speicherplatzes wird vom Inhalt des Akkumulators subtrahiert. (d ist die Verschiebung im Zahlenbereich $-128 \leq d \leq 127$)
16	SBC r	Das Carry-Flag (CY) wird zum Registerinhalt r addiert und dieses Ergebnis vom Inhalt des Akkumulators subtrahiert. (r können die Register A, B, C, D, E, H oder L sein)
17	SBC M	Das Carry-Flag (CY) wird zum Inhalt des durch das Registerpaar HL adressierten Speicherplatzes M addiert und dieses Ergebnis vom Inhalt des Akkumulators subtrahiert.
18	SBC n	Ein durch n definiertes Wort plus Carry-Flag (CY) wird vom Inhalt des Akkumulators subtrahiert.
19	SBC (IX+d)	Das Carry-Flag (CY) wird zum Inhalt des durch das Register IX plus Verschiebung adressierten Speicherplatzes addiert und vom Inhalt des Akkumulators subtrahiert. (d ist die Verschiebung im Zahlenbereich $-128 \leq d \leq 127$)
20	SBC (IY+d)	Das Carry-Flag (CY) wird zum Inhalt des durch das Register IY plus Verschiebung adressierten Speicherplatzes addiert und vom Inhalt des Akkumulators subtrahiert. (d ist die Verschiebung im Zahlenbereich $-128 \leq d \leq 127$)
21	INC r	Der Inhalt des Registers r wird um 1 erhöht. (r können die Register A, B, C, D, E, H oder L sein)
22	INC M	Der Inhalt des durch das Registerpaar HL adressierten Speicherplatzes wird um 1 erhöht.
23	INC (IX+d)	Der Inhalt des durch das Register IX plus Verschiebung adressierten Speicherplatzes wird um 1 erhöht. (d ist die Verschiebung im Zahlenbereich $-128 \leq d \leq 127$)

Lfd.Nr.	MNEMONIK	Wirkungsweise der Befehle
24	INC (IY+d)	Der Inhalt des durch das Register IY plus Verschiebung adressierten Speicherplatzes wird um 1 erhöht. (d ist die Verschiebung im Zahlenbereich $-128 \leq d \leq 127$)
25	DEC r	Der Inhalt des Registers r wird um 1 vermindert. (r können die Register A, B, C, D, E, H oder L sein)
26	DEC M	Der Inhalt des durch das Registerpaar HL adressierten Speicherplatzes wird um 1 vermindert.
27	DEC (IX+d)	Der Inhalt des durch das Register IX plus Verschiebung adressierten Speicherplatzes wird um 1 vermindert. (d ist die Verschiebung im Zahlenbereich $-128 \leq d \leq 127$)
28	INC (IY+d)	Der Inhalt des durch das Register IY plus Verschiebung adressierten Speicherplatzes wird um 1 vermindert. (d ist die Verschiebung im Zahlenbereich $-128 \leq d \leq 127$)

3.1.2.2. 2-Byte-Arithmetik

Lfd.Nr.	MNEMONIK	Wirkungsweise der Befehle
1	ADD HL,dd	Der Inhalt des Doppelregisters dd wird zum Inhalt des Registerpaares HL addiert. (dd können die Doppelregister BC, DE, HL oder SP sein.)
2	ADD IX,IX	Der Inhalt des Registers IX wird mit sich selbst addiert. Diese Verdoppelung ist gleichbedeutend mit einer Linksverschiebung der 16 Bits um eine Bitposition.
3	ADD IY,IY	Der Inhalt des Registers IY wird mit sich selbst addiert. Diese Verdoppelung ist gleichbedeutend mit einer Linksverschiebung der 16 Bits um eine Bitposition.
4	ADD IX,pp	Der Inhalt des Doppelregisters pp wird zum Inhalt des Doppelregisters IX addiert. (pp können die Doppelregister BC, DE oder SP sein)
5	ADD IY,pp	Der Inhalt des Doppelregisters pp wird zum Inhalt des Doppelregisters IY addiert. (pp können die Doppelregister BC, DE oder SP sein)

Lfd.Nr.	MNEMONIK	Wirkungsweise der Befehle
6	ADD HL,dd	Der Inhalt des Doppelregisters dd wird zum Inhalt des Registerpaares HL addiert. Der Inhalt des Carry-Flags (CY) wird ebenfalls addiert. (dd können die Doppelregister BC, DE, HL oder SP sein)
7	SBC HL,dd	Das Carry-Flag (CY) wird zu dem Doppelregister dd addiert und dieses Ergebnis vom Inhalt des Registerpaares HL subtrahiert. (dd können die Doppelregister BC, DE, HL oder SP sein)
8	INC dd	Der Inhalt des Doppelregisters dd wird um 1 erhöht. (dd können die Doppelregister BC, DE, HL, SP sowie die Register IX oder IY sein)
9	DEC dd	Der Inhalt des Doppelregisters dd wird um 1 vermindert. (dd können die Doppelregister BC, DE, HL, SP sowie die Register IX oder IY sein)

3.1.2.3. 1-Byte-Logikbefehle

lfd.Nr.	MNEMONIK	Wirkungsweise des Befehls
1	AND s	Logisches UND eines Registers, Direktwertes oder Speicherbytes mit dem Akkumulator. Das spezifizierte Byte s wird bitweise mit dem Inhalt des Akkumulators konjunktiv verknüpft. Das logische UND zweier Bits ist nur dann eins, wenn beide Bits 1 sind. z.B. Akkumulator : 1111 1100 FCH Byte s : 0000 1111 0FH Resultat im Akkumulator: 0000 1100 0CH Bedeutung von s: s kann sein: r, n, M, (IX+d), (IY+d) r können die Register A, B, C, D, E, H, oder L sein n ist eine 8-Bit-Konstante M ist der 8-Bit-Inhalt einer Speicherstelle (IX+d) und (IY+d) sind Indexregister plus Verschiebung d ist die Verschiebung im Zahlenbereich $-128 \leq d \leq 127$

Lfd.Nr.	MNEMONIK	Wirkungsweise der Befehle
2	OR s	Logisches ODER eines Registers, Direktwertes oder Speicherbytes mit dem Akkumulator. Das spezifizierte Byte s wird bitweise mit dem Inhalt des Akkumulators disjunktiv verknüpft. Das logische ODER zweier Bits ist nur dann 0, wenn beide Bits 0 sind. z.B. Akkumulator : 1111 1100 FCH Byte s : 1111 0001 F1H Resultat im Akkumulator: 1111 1101 FDH Bedeutung von s : siehe AND s
3	XOR s	Exklusives ODER eine Registers, Direktwertes oder Speicherbytes mit dem Akkumulator. Das spezifizierte Byts s wird bitweise mit dem Inhalt des Akkumulators exklusiv verknüpft. Das exklusive ODER ist dann gleich 1, wenn ein Bit = 0 und ein Bit = 1 ist. z.B. Akkumulator : 1111 1100 FCH Byte s : 1111 0001 F1H Resultat im Akkumulator: 0000 0001 0DH Bedeutung von s : siehe AND s
4	CMP s	Der Inhalt von s wird mit dem Akkumulatorinhalt verglichen. Der ursprüngliche Inhalt von A bleibt erhalten. Als Vergleichsergebnis werden entsprechende Flags gesetzt. Bedeutung von s : siehe AND s

3.1.3. Sprungbefehle

Bei den Sprungbefehlen ist zu unterscheiden zwischen unbedingten und bedingten Sprüngen. Es existieren weiterhin relative Sprünge, die zur Adressenbildung anstelle von zwei Bytes nur ein Byte benötigen.
Bei den bedingten Sprüngen werden Sprungbedingungen getestet. Diese Sprungbedingungen werden vom Flagregister F abgefragt. In Abhängigkeit von den Bedingungsflags können die Sprungbedingungen erfüllt sein oder nicht.
Bei einer erfüllten Sprungbedingung wird der Befehlszähler entsprechend der Adressenangabe im Sprungbefehl verändert. Bei nichterfüllter Sprungbedingung wird der Sprungbefehl ignoriert. Das Sprungziel nn kann eine 16-Bit-Konstante oder ein symbolischer Name sein. Im assemblierten Befehl steht die Adresse nn in umgekehrter Reihenfolge, d.h., das auf den Operationscode folgende Byte ist das niederwertige Adreßbyte.
Bei den relativen Sprüngen wird das Sprungziel über den Wert e errechnet. Die Sprungweite e wird zum aktuellen Stand des Befehlszählers (= Befehlszählerstand nach dem relativen Sprungbefehl) addiert und ermöglicht einen Sprung im Bereich zwischen -126 und 129 Bytes.
Der Wert e kann eine 8-Bit-Kostante oder eine symbolische Adresse sein.

lfd.Nr.	MNEMONIK	Wirkungsweise der Befehle
1	JMP nn	unbedingter Sprung nach Adresse nn
2	JPNZ nn	Sprung nach Adresse nn, wenn Z-Flag = 0
3	JPZ nn	Sprung nach Adresse nn, wenn Z-Flag = 1
4	JPNC nn	Sprung nach Adresse nn, wenn CY-Flag = 0
5	JPC nn	Sprung nach Adresse nn, wenn CY-Flag = 1
6	JPPO nn	Sprung nach Adresse nn, wenn P/V-Flag = 0
7	JPPE nn	Sprung nach Adresse nn, wenn P/V-Flag = 1
8	JPP nn	Sprung nach Adresse nn, wenn S-Flag = 0
9	JPM nn	Sprung nach Adresse nn, wenn S-Flag = 1
10	JR e	Unbedingter relativer Sprung
11	JRNZ e	Relativer Sprung um Verschiebung e, wenn Z-Flag = 0 ist
12	JRZ e	Relativer Sprung um Verschiebung e, wenn Z-Flag = 1 ist
13	JRNC e	Relativer Sprung um Verschiebung e, wenn CY-Flag = 0 ist
14	JRC e	Relativer Sprung um Verschiebung e, wenn CY-Flag = 1 ist
15	JMP M	Unbedingter Sprung zur Adresse, die in Register HL steht
16	JMP (IX)	Unbedingter Sprung zur Adresse, die im Indexregister IX steht
17	JMP (IY)	Unbedingter Sprung zur Adresse, die im Indexregister IX steht
18	DJNZ e	Der Inhalt des Registers B wird um 1 vermindert. Bedingter relativer Sprungbefehl um Verschiebung e, wenn der Inhalt des Registers B ungleich 0 ist.

3.1.4. Blocktransportbefehle

Mit einem einzigen Blocktransportbefehl kann ein beliebig großer zusammenhängender Block des Speichers zu einem anderen Speicherplatz transportiert werden.
Die Blocktransportbefehle sind für die Verarbeitung von großen Datenblöcken sehr sinnvoll.

lfd.Nr.	MNEMONIK	Wirkungsweise des Befehls
1	LDIR	Transport von mehreren Datenbytes ab der Speicherstelle, die durch das Registerpaar HL adressiert wird, nach der Speicherstelle, die durch das Registerpaar DE adressiert wird. Die Bytezahl ist im Registerpaar BC enthalten. Nach jeder Byteübertragung wird der Inhalt von HL und DE um 1 erhöht und BC um 1 vermindert. Die Übertragung endet, wenn (BC) gleich 0 ist. Beispiel: LD HL,DATA Startadr. d. Quellbereiches --> HL LD DE,PUF Startadr. d. Zielbereiches --> DE LD BC,737 Länge der Datenkette --> BC LDIR Transport der Datenkette nach Zielbereich, Erhöhen HL und DE, Vermindern BC, wiederholen bis BC gleich 0
2	LDI	Transport **eines** Datenbytes von der Speicherstelle, die durch das Register HL adressiert wird nach der Speicherstelle, die durch das Registerpaar DE adressiert wird. Die Register DE und HL werden um 1 erhöht und das Register BC um 1 vermindert. Beispiel: LD HL,DATA Startadr. d. Quellbereiches --> HL LD DE,PUF Startadr. d. Zielbereiches --> DE LD BC,132 max. Kettenlänge --> BC LD A,'$' Endemerkmal --> A LOOP:CMP (HL) Vergleich Speicherinhalt mit Endemerkmal JRZ END-# Sprung zu END, wenn Zeichen gleich LDI Zeichentransport (HL) zu (DE), erhöhen HL und DE, vermindern BC JPPE LOOP Sprung zu LOOP, wenn noch Zeichen zu transportieren sind, sonst gehe weiter. P/V-Flag gleich 1, solange BC ungleich 0 ist END: JMP HALT
3	LDDR	Der Befehl wirkt wie LDIR, nur die Register DE und HL werden um 1 vermindert.
4	LDD	Der Befehl wirkt wie LDI, nur die Register DE und HL werden um 1 vermindert

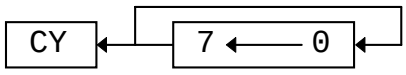
3.1.5. Blocksuchbefehle

Die Blocksuchbefehle sind für die Verarbeitung von großen Datenmengen geeignet. Mit einem einzigen Befehl kann ein Speicherblock von beliebiger Größe nach einem bestimmten 1-Byte-Zeichen durchsucht werden. Wenn das Zeichen gefunden wurde, ist der Befehl automatisch beendet.

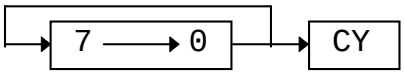
lfd.Nr.	MNEMONIK	Wirkungsweise des Befehls
1	CPI	Vergleich des Inhaltes des durch das Registerpaar HL adressierten Speicherplatzes mit dem Inhalt des Akkumumlators. Das Registerpaar BC kann als Bytezähler arbeiten. Das Registerpaar HL wird um 1 erhöht, das Registerpaar BC um 1 vermindert.
2	CPIR	Vergleich des Inhaltes des Akkumulators mit dem Inhalte eines adressierten Speicherbereiches. Die Startadresse des Bereiches ist in dem Registerpaar HL enthalten, die Länge des Bereiches in dem Registerpaar BC. Die zu suchende Konstante steht im Akkumulator. Der Vergleich endet, wenn der Akkumulator gleich (HL) ist oder wenn BC gleich 0 ist. Der Befehl sucht, indem er Register HL um 1 erhöht und Register BC um 1 vermindert.
3	CPD	Der Befehl wirkt wie CPI, nur die Registerpaare HL und BC werden **beide** vermindert.
4	CPIR	Der Befehl wirkt wie CPIR, nur die Registerpaare HL und BC werden um 1 vermindert.

3.1.6. Verschiebebefehle

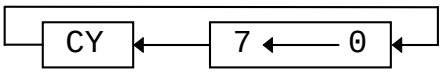
Durch diese Befehle ist die Möglichkeit gegeben, im Akkumulator, in einem Universalregister oder in einem Speicherplatz Daten bitweise, einfach oder zyklisch, zu verschieben.

lfd.Nr.	MNEMONIK	Wirkungsweise des Befehls
1	RLCA	Linksrotation des Akkumulatorinhaltes. Der Inhalt des Akkumulators wird um eine Bitposition nach links verschoben. Das höchstwertige Bit 7 wird zum Inhalt des niederwertigen Bits 0 und des Carry-Glags. 

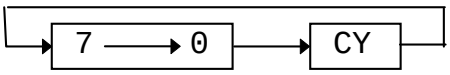
lfd.Nr.	MNEMONIK	Wirkungsweise des Befehls
2	RRCA	Rechtsrotation des Akkumulatorinhaltes. Der Inhalt des Akkumulators wird um eine Bitposition nach rechts verschoben. Das niederwertigste Bit 0 wird zum Inhalt des höchstwertige Bits 7 und des Carry-Flags.



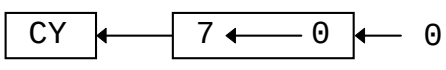
3	RLA	Linksrotattion des Akkumulatorinhaltes durch CY. Der Inhalt des Akkumulators wird um eine Bitposition nach links verschoben. Das höchstwertige Bit 7 ersetzt das Carry-Flag, während das Carry-Flag das niederwertigste Bit 0 des Akkumulators ersetzt.



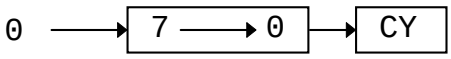
4	RRA	Rechtsrotation des Akkumulatorinhaltes durch CY. Der Inhalt des Akkumulators wird um eine Bitposition nach rechts verschoben. Das niederwertigste Bit 0 ersetzt das Carry-Flag während das höchstwertige Bit 7 durch das Carry-Flag ersetzt wird.



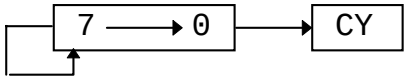
5	RLC s	Linksrotation von s analog dem Befehl RLCA. s steht für r, (HL), (IX+d) und (IY+d)
6	RRC s	Rechtsrotation von s analog dem Befehl RRCA. s steht für r, (HL), (IX+d) und (IY+d)
7	RL s	Linksrotation von s durch CY analog dem Befehl RLA. s steht für r, (HL), (IX+d) und (IY+d)
8	RR s	Rechtsrotation von s durch CY analog dem Befehl RRA. s steht für r, (HL), (IX+d) und (IY+d)
9	SLA s	Linksverschiebung von s um 1 Bit durch CY. Das niederwertige Registerbit 0 wird 0. s steht für r, (HL), (IX+d) und (IY+d)



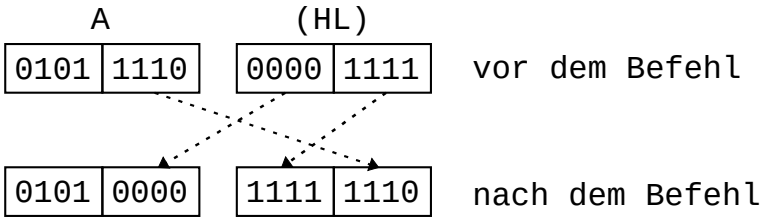
10	SRL s	Rechtsverschiebung von s um 1 Bit durch CY. Das höchstwertige Registerbit 7 wird 0. s steht für r, (HL), (IX+d) und (IY+d)



lfd.Nr.	MNEMONIK	Wirkungsweise des Befehls
11	SRA s	Rechtsverschiebung von s um 1 Bit durch CY. Der Inhalt von Bit 7 bleibt erhalten. s steht für r, (HL), (IX+d) und (IY+d)

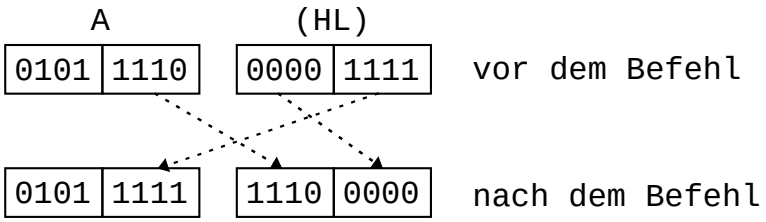


12	RLD	Zyklische Verschiebung nach links zwischen dem Akkumulator und dem Inhalt des durch das Registerpaar HL adressierten Speicherplatzes.



Die unteren 4 Bits der durch das Registerpaar HL adressierten Speicherstelle werden in die oberen 4 Bitstellen übertragen und diese ihrerseits in die unteren 4 Bitstellen des Akkumulators.
Die unteren 4 Bits des Akkumulators werden in die unteren 4 Bits der Speicherstelle transportiert.

13	RRD	Zyklische Verschiebung nach rechts zwischen dem Akkumulator und dem Inhalt des durch das Registerpaar HL adressierten Speicherplatzes.



Die unteren 4 Bits der durch das Registerpaar HL adressierten Speicherstelle werden in die unteren 4 Bitstellen des Akkumulators übertragen und diese in die oberen 4 Bitstellen der durch das Registerpaar HL adressierten Speicherstelle.
Die oberen 4 Bits aus der durch das Registerpaar HL adressierten Speicherstelle werden in die unteren 4 Bitstellen transportiert.

3.1.7. Eingabebefehle

Die Ein-/Ausgabebefehle gestatten den Datentransport zwischen Speicherplätzen oder den Universalregistern der CPU einerseits und den externen E/A-Geräten andererseits. Die Eingabebefehle setzen automatisch das Flag-Register, so daß keine zusätzlichen Befehle nötig sind, um den Status der Eingabedaten zu ermitteln.
Die externen E/A-Geräte werden über zugeordnete Kanaladressen ausgewählt und angesprochen.

lfd.Nr.	MNEMONIK	Wirkungsweise des Befehls
1	IN n	Die Kanaladresse wird mittels eines Direktoperanden eingestellt. Das Zielregister ist der Akkumulator A <-- (n)
2	IN r	Die Kanaladresse wird indirekt über das Register C eingestellt. Das Zielregister ist r (r = A, B, C, D, E, H, L) r <-- (C)
3	INF	Es werden nur Flags gesetzt. Dieser Befehl ist ein Sonderfall von IN r (r = 110)
4	INI	Die Kanaladresse wird indirekt über das Register C eingestellt, die Zieladresse über das Registerpaar HL. B kann als Bytezähler arbeiten. B wird dekrementiert, HL inkrementiert. (HL) <-- (C) B <-- B-1 HL <-- HL+1
5	INIR	Die Kanaladresse wird indirekt über das Register C eingestellt, die Zieladresse über das Registerpaar HL. B arbeitet als Bytezähler. B wird dekrementiert, HL inkrementiert. Es wird eine Blockübertragung durchgeführt bis B = 0 ist. (HL) <-- (C) B <-- B-1 HL <-- HL+1 Wiederholung bis B = 0
6	IND	Die Kanaladresse wird indirekt über das Register C eingestellt, die Zieladresse über das Registerpaar HL. B kann als Bytezähler arbeiten. B und HL werden dekrementiert (HL) <-- (C) B <-- B-1 HL <-- HL-1

lfd.Nr.	MNEMONIK	Wirkungsweise des Befehls
7	INDR	Die Kanaladresse wird indirekt über das Register C eingestellt, die Zieladresse über das Registerpaar HL. B arbeitet als Bytezähler. B und HL werden dekrementiert Es wird eine Blockübertragung durchgeführt bis B = 0 ist. (HL) <-- (C) B <-- B-1 HL <-- HL-1 Wiederholen bis B = 0.

Die Kanaladresse liegt auf der unteren Hälfte des Adreßbusses A0-A7. Auf der oberen Hälfte des Adreßbusses liegt bei IN n der Akkumulatorinhalt, bei den restlichen Befehlen der Inhalt des Registers B.

3.1.8. Ausgabebefehle

lfd.Nr.	MNEMONIK	Wirkungsweise des Befehls
1	OUT n	Die Kanaladresse wird mittels eines Direktoperanden eingestellt. Das Quellregister ist der Akkumulator. (n) <-- A
2	OUT r	Die Kanaladresse wird indirekt über das Register C eingestellt. Das Quellregister ist r (R = A, B, C, D, E, H, L) (C) <-- r
3	OUTI	Die Kanaladresse wird indirekt über das Register C eingestellt, die Quelladresse über das Registerpaar HL. B kann als Bytezähler arbeiten. B wird dekrementiert, HL inkrementiert. (C) <-- (HL) B <-- B-1 HL <-- HL+1
4	OTIR	Die Kanaladresse wird indirekt über das Register C eingestellt, die Quelladresse über das Registerpaar HL. Das Register B arbeitet als Bytezähler. B wird dekrementiert, HL inkrementiert. Es wird eine Blockübertragung durchgeführt, bis B = 0 ist. (C) <-- (HL) B <-- B-1 HL <-- HL+1 Wiederholung bis B = 0.

lfd.Nr.	MNEMONIK	Wirkungsweise des Befehls
5	OUTD	Die Kanaladresse wird indirekt über das Register C eingestellt, die Quelladresse über das Registerpaar HL. B kann als Bytezähler arbeiten. B und HL werden dekrementiert. (C) <-- (HL) B <-- B-1 HL <-- HL-1
6	OTDR	Die Kanaladresse wird indirekt über das Register C eingestellt, die Quelladresse über das Registerpaar HL. B arbeitet als Bytezähler. B und HL werden dekrementiert. Es wird eine Blockübertragung durchgeführt bis B = 0 ist. (C) <-- (HL) B <-- B-1 HL <-- HL-1

3.1.9. Spezielle Akkumulator- und Flagbefehle

lfd.Nr.	MNEMONIK	Wirkungsweise des Befehls
1	DAA	Der DAA-Befehl korrigiert nach der Addition/ Subtraktion zweier gepackter BCD-Zahlen den Akkumulatorinhalt so, daß im Akkumulator wieder gepackte BCD-Darstellung erreicht wird. Beispiel 1: 6|7 0LL0 0LLL + 1|5 000L 0L0L Akkumulator: 7|C 0LLL LL00 Akkumulator nach DAA: 8|2 L000 00L0 Beispiel 2: 6|5 0LL0 0L0L + 5|7 0L0L 0LLL Akkumulator: B|C L0LL LL00 Akkumulator nach DAA: 2|2 00L0 00L0 CY <-- L
2	CPL	Bitweises Negieren (Komplementieren) des Akkumulatorinhaltes
3	NEG	Subtraktion des Akkumulatorinhaltes von 0. Entspricht wertmäßig Zweierkomplement
4	CCF	Komplementieren des CY-Flags
5	SCF	Setzen des CY-Flags

3.1.10. Registeraustauschbefehle

Die Registeraustauschbefehle sind ein Byte lang, ausgenommen ist der Austausch zwischen dem Stackpointer (SP) und jeweils einem Basisregister. Durch die geringe Befehlslänge werden kurze Interruptantwortzeiten möglich.

lfd.Nr.	MNEMONIK	Wirkungsweise des Befehls
1	EX DE,HL	Die 16-Bit-Inhalte der Registerpaare DE und HL werden ausgetauscht. DE <--> HL E <--> L D <--> H
2	EXAF	Die 16-Bit-Inhalte der Registerpaare AF und AF' werden ausgetauscht. AF' besteht aus den Registern A' und F'. AF <--> AF' A <--> A' F <--> F'
3	EXX	Die 16-Bit-Inhalte der nachstehenden Register werden in folgender Weise getauscht: BC <--> BC' DE <--> DE' HL <--> HL'
4	EX (SP),HL	Der Inhalt des Registerpaares HL wird gegen den Inhalt der durch den Stackpointer SP und SP+1 adressierten Speicherstelle ausgetauscht. H <--> (SP+1) L <--> (SP)
5	EX (SP),IX	Der Inhalt des Registers IX wird gegen den Inhalt der durch den Stackpointer SP und SP+1 adressierten Speicherstelle ausgetauscht. IX_H <--> (SP+1) IX_l <--> (SP)
6	EX (SP),IY	Der Inhalt des Registers IY wird gegen den Inhalt der durch den Stackpointer SP und SP+1 adressierten Speicherstelle ausgetauscht. IY_H <--> (SP+1) IY_l <--> (SP)

3.1.11. Indirekte Registeroperationen

3.1.11.1. PUSH-Befehle

Bei den PUSH-Befehlen wird der Inhalt der Registerpaare qq (qq = AF, BC, DE, HL) oder der Register IX bzw. IY in einen durch den Stackpointer adressierten RAM-Bereich (Stack) übertragen, der als LIFO-Datei (last in, first out) organisiert ist. Der Stackpointer SP enthält dabei ständig eine 2-Byte-Adresse, die der aktuellen niedrigsten Adresse des Stack-Bereiches entspricht. Der PUSH-Befehl subtrahiert 1 vom Inhalt des Stackpointers SP und lädt das höherwertige Byte des

Registerpaares bzw. des Registers in die Speicherstelle, die durch den Inhalt des Stackpointers SP adressiert ist. Danach wird der Inhalt des Stackpointers SP nochmals dekrementiert und das niederwertige Byte wird in die Speicherstelle eingetragen, die jetzt durch den Inhalt des Stackpointers SP adressiert ist.

lfd.Nr.	MNEMONIK	Wirkungsweise des Befehls
1	PUSH AF	Erniedrigen SP (SP-1) <--> A Erniedrigen SP (SP-2) <--> F
2	PUSH BC	analog lfd. Nr. 1
3	PUSH DE	analog lfd. Nr. 1
4	PUSH HL	analog lfd. Nr. 1
5	PUSH IX	Erniedrigen SP (SP-1) <--> IX_H Erniedrigen SP (SP-2) <--> IX_I
4	PUSH HL	analog lfd. Nr. 5

3.1.11.2. POP-Befehle

Bei den POP-Befehlen wird der Inhalt der vom Stackpointer SP und von (SP+1) adressierten 2 Bytes des externen Stacks in ein Registerpaar qq bzw. in ein Register IX oder IY übertragen.
Der POP-Befehl überträgt zunächst den Inhalt der Speicherstelle, die durch den aktuellen Wert des Stackpointers adressiert ist in den niederwertigen Teil des Registerpaares bzw. Registers. Danach wird der Stackpointer inkrementiert und der Inhalt der jetzt adressierten Speicherstelle wird in den höherwertigen Teil des Registerpaares bzw. Registers übertragen. Der Stackpointer wird erneut inkrementiert.

lfd.Nr.	MNEMONIK	Wirkungsweise des Befehls
7	POP AF	F <-- (SP) Erhöhen SP A <-- (SP+1) Erhöhen SP
8	POP BC	analog lfd. Nr. 7
9	POP DE	analog lfd. Nr. 7
10	POP HL	analog lfd. Nr. 7
11	POP IX	IY_H <-- (SP+1) IY_I <-- (SP)
12	POP IY	analog lfd. Nr. 11

3.1.12. Unterprogrammaufrufbefehle

Es ist zwischen unbedingten und bedingten Unterprogrammaufrufen zu unterscheiden. Bei dem unbedingten Unterprogrammaufruf wird der dem Unterprogrammaufruf folgenden Befehlszählerstand in den Stack gerettet.
Der höherwertige Adreßteil im Befehlszähler wird nach der Adresse Stackpointer minus 1 und der niederwertige Adreßteil nach der Adresse Stackpointer minus 2 gebracht.

PC_H --> (SP-1)
PC_L --> (SP-2)

Die im Befehl angegebene Unterprogrammstartadresse nn wird vom Befehlszähler übernommen.

nn --> PC

Ein Unterprogramm wird durch einen Sprungbefehl beendet. Bei den bedingten Unterprogrammaufrufen wird bei erfüllter Sprungbedingung analog dem unbedingten Unterprogrammaufruf verfahren und bei nicht erfüllter Sprungbedingung wird der Befehl ignoriert.

lfd.Nr.	MNEMONIK	Wirkungsweise des Befehls
1	CALL nn	Unbedingter Unterprogrammaufruf (SP-1) <-- PC_H (SP-2) <-- PC_L PC <-- nn
2	CANZ nn	Unterprogrammaufruf, wenn Z-Flag = 0 ist.
3	CAZ nn	Unterprogrammaufruf, wenn Z-Flag = 1 ist.
4	CANC nn	Unterprogrammaufruf, wenn CY-Flag = 0 ist.
5	CAC nn	Unterprogrammaufruf, wenn CY-Flag = 1 ist.
6	CAPO nn	Unterprogrammaufruf, wenn P/V-Flag = 0 ist.
7	CAPE nn	Unterprogrammaufruf, wenn P/V-Flag = 1 ist.
8	CAP nn	Unterprogrammaufruf, wenn S-Flag = 0 ist.
9	CAM nn	Unterprogrammaufruf, wenn S-Flag = 1 ist.
10	RST p	Der RST-Befehl ist ein spezieller Unterprogrammaufruf. Für p sind folgende Angaben zugelassen: 00H, 08H, 10H, 18H, 20H, 28H, 30H, 38H. Dabei erfolgt ein Sprung zu der angegebenen Adresse. Der RST-Befehl entspricht in der weiteren Wirkung dem unbedingten Unterprogrammaufruf.

3.1.13. Unterprogrammrücksprungbefehle

Ein Rücksprungbefehl beendet ein Unterprogramm. Es wird unterschieden zwischen unbedingten Rücksprüngen, bedingten Rücksprüngen und Rücksprüngen aus Interrupt-Behandlungsroutinen. Bei einem unbedingten Rücksprung und bei bedingten Rücksprüngen, wenn die Sprungbedingung erfüllt ist, wird der beim Aufruf des Unterprogramms in den Stack gerettete Befehlszählerinhalt wieder in den Befehlszähler zurückgeschrieben.

PC_L <-- (SP)
PC_H <-- (SP+1)
SP <-- SP+2

Bei nichterfüllter Sprungbedingung wird der dem Rücksprung folgende Befehl abgearbeitet.

lfd.Nr.	MNEMONIK	Wirkungsweise des Befehls
1	RET	Unbedingter Rücksprung
2	RNZ	Unterprogrammrücksprung, wenn das Z-Flag = 0 ist.
3	RZ	Unterprogrammrücksprung, wenn das Z-Flag = 1 ist.
4	RNC	Unterprogrammrücksprung, wenn das CY-Flag = 0 ist.
5	RC	Unterprogrammrücksprung, wenn das CY-Flag = 1 ist.
6	RPO	Unterprogrammrücksprung, wenn das P/V-Flag = 0 ist.
7	RPE	Unterprogrammrücksprung, wenn das P/V-Flag = 1 ist.
8	RP	Unterprogrammrücksprung, wenn das S-Flag = 0 ist.
9	RM	Unterprogrammrücksprung, wenn das S-Flag = 1 ist.
10	RETI	Es erfolgt ein Rücksprung aus einer Interrupt-Behandlungsroutine. Dem Peripheriebaustein, der den Interrupt anmeldete, wird das Ende seines Programmes mitgeteilt. Der Baustein gibt daraufhin die von ihm blockierte Interrupt-Kette (DAISY-CHAIN) wieder frei und ermöglicht damit die Abarbeitung niederwertiger Interrupts. Der Inhalt von IFF2 wird nach IFF1 übertragen. Durch die RETI-Anweisung wird der maskierbare Interrupt nicht freigegeben. Es sollte grundsätzlich vor jeder RETI-Anweisung ein EI-Befehl stehen, der die Annahme später folgender Interruptanforderungen ermöglicht.

lfd.Nr.	MNEMONIK	Wirkungsweise des Befehls
11	RETN	Es erfolgt ein Rücksprung aus einer Interrupt-Behandlungsroutine, die durch einen nicht-maskierbaren Interrupt (NMI) ausgelöst wurde. Die Anweisung wirkt zunächst wie die RET-Anweisung. Zusätzlich wird der Inhalt von IFF2 nach IFF1 übertragen, so daß die Abarbeitung maskierbarer Interruptanforderungen unmittelbar nach Ausführung des RETN-Befehls freigegeben ist, falls sie vor der NMI-Anforderung freigegeben war.

3.1.14. Bitmanipulationsbefehle

Die Bitmanipulationsbefehle erlauben, Bits in einem Register oder auf einem Speicherplatz zu setzen, zu löschen und zu testen.

lfd.Nr.	MNEMONIK	Wirkungsweise des Befehls
1	SET b,r	Die durch b gekennzeichnete Bitposition wird in dem Register r gesetzt. b = 7,6, ... 0 r können die Register A, B, C, D, E, H oder L sein.
2	SET b,M	Die durch b gekennzeichnete Bitposition wird in der Speicherstelle gesetzt, die durch das Registerpaar HL adressiert ist. b = 7,6, ... 0
3	SET b,(IX+d)	Die durch b gekennzeichnete Bitposition wird in der Speicherstelle gesetzt, die durch des Doppelregisters IX plus Verschiebung adressiert ist. b = 7,6, ...0 d = Verschiebung im Zahlenbereich $-128 \leq d \leq 127$
4	SET b,(IY+d)	Die durch b gekennzeichnete Bitposition wird in der Speicherstelle gesetzt, die durch des Doppelregisters IY plus Verschiebung adressiert ist. b = 7,6, ...0 d = Verschiebung im Zahlenbereich $-128 \leq d \leq 127$
5	RES b,s	Die durch b gekennzeichnete Bitposition in s wird gelöscht. b = 7,6, ... 0 s kann r, M, (IX+d) oder (IY+d) sein.
6	BIT b,s	Die durch b gekennzeichnete Bitposition in s wird getestet. Das Komplement des zu testenden Bits wird in das Z-Flag geladen. b = 7,6, ... 0 s kann r, M, (IX+d) oder (IY+d) sein..

3.1.15. CPU-Steuerbefehle

Diese Steuerbefehle lösen verschiedene Bedingungen und Betriebsarten bei der CPU aus. Diese Gruppe enthält auch Befehle, die das Ein- und Ausschalten des Interrupt-Flipflops bewirken oder die Betriebsart für das Interruptverhalten setzen.

lfd.Nr.	MNEMONIK	Wirkungsweise des Befehls
1	NOP	Die CPU führt keine Operation aus. Es werden Refresh-Zyklen erzeugt.
2	HALT	Die CPU führt solange eine Folge von NOP-Befehlen aus, bis ein Interrupt- oder der RESET-Eingang an der CPU aktiv wird. Es werden Refresh-Zyklen erzeugt.
3	DI	Der maskierbare Interrupt wird durch Rücksetzen der Interrupt-Flipflops der CPU gesperrt. Nichtmaskierbare Interrupts werden anerkannt.
4	EI	Der maskierbare Interrupt wird durch Setzen der Interrupt-Flipflops der CPU freigegeben. Während der Ausführung des Befehls EI und des darauffolgenden Befehls akzeptiert die CPU keine Interruptanforderungen.
5	IM0	Der Befehl bringt die CPU in den Interruptmodus 0.
6	IM1	Der Befehl bringt die CPU in den Interruptmodus 1.
7	IM2	Der Befehl bringt die CPU in den Interruptmodus 2.

3.2. Pseudo- / Steueranweisungen

Pseudoanweisungen dienen zur Steuerung der Übersetzung des Quellprogramms in das Objektprogramm. Pseudoanweisungen werden nicht in Maschinenbefehle des K1520 übersetzt. Die Pseudoanweisungen sind jedoch wie ausführbare Anweisungen aufgebaut, d.h., sie können mit symbolischen Adressen und Kommentaren versehen sein.

3.2.1. Definition von Datenbytes DB

Namensfeld	Operationsfeld	Operandenfeld
name (wird nach den unter Pkt. 2.1.1., Namensfeld, angegebenen Regeln gebildet)	DB	n 1-Byte-Konstante oder Adresse 'XX...X' ISO-Zeichenkette

```
Beispiele:       DB 'FEHLER'
          ABC1: DB 77H
```

3.2.2. Definition von Adressen DA

Namensfeld	Operationsfeld	Operandenfeld
siehe 3.2.1.	DA	nn

Die DA-Anweisung definiert eine 2-Byte-Konstante oder -Adresse in der Reihenfolge L (nn), H (nn).

```
Beispiele: Adresse von WORT in 014AH
           BD2:  DA WORT
           Abspeicherung: Adresse   Inhalt
                          BD2       4A
                          BD2+1     01
```

3.2.3. Reservierung von Speicherplatz BER

Namensfeld	Operationsfeld	Operandenfeld
siehe 3.2.1.	BER	nn

Die BER-Anweisung reserviert Speicherplätze. Der reservierte Speicherbereich wird nicht belegt. Bei Verwendung von Symbolen für nn müssen diese vorher definiert sein und einen absoluten Wert ergeben.

```
Beispiel:  KAP1:  BER 100          Reservierung von 100 Bytes
           KAP2:  DEF 200
           KAP3:  BER KAP2         Reservierung von 200 Bytes
```

3.2.4. Setzen des Befehlszählers ORG

Namensfeld	Operationsfeld	Operandenfeld
leer	ORG	nn

Die ORG-Anweisung legt den absoluten Programmanfang fest. Die Anweisung steht am Programmanfang und muß einen absoluten Wert ergeben, verwendete Symbole müssen vorher definiert werden. Fehlt die Anweisung, so wird das Programm relativ zu Null übersetzt, es erhält den verschieblichen Modus.

3.2.5. Symboldefinition einmalig EQU

Namensfeld	Operationsfeld	Operandenfeld
name	EQU	nn

Mit der EQU-Anweisung werden im Namensfeld stehenden Symbolen Werte zugewiesen. Das Namensfeld muß belegt sein, die Zuweisung darf im Gegensatz zur DEF-Anweisung (Pkt. 3.2.6.) nur einmalig erfolgen.
Die Angabe im Operandenfeld kann sowohl einen verschieblichen als auch einen absoluten Wert ergeben, verwendete Symbole müssen vorher definiert sein. Externe Symbole sind nicht erlaubt.

3.2.6. Symboldefinition veränderlich DEF

Namensfeld	Operationsfeld	Operandenfeld
name	DEF	nn

Mit der DEF-Anweisung werden im Namensfeld stehenden Symbolen absolute Werte zugeordnet. Das Namensfeld muß belegt sein. Die Zuweisung gilt solange, bis dem Symbol durch eine neuerliche Zuweisung ein neuer Wert angewiesen wird. Verwendete Symbole müssen vorher definiert sein, externe Symbole sind nicht erlaubt.
Als Operatoren sind zugelassen: +, -, .AND., .OR., .XOR.
Sie werden mit folgender Priorität bearbeitet:

1. +, -
2. AND
3. OR, XOR

Die Länge des Ausdruckes wird durch die maximal zulässige Zeichenanzahl begrenzt (max. 61 Zeichen).

3.2.7. Programmname PN

Namensfeld	Operationsfeld	Operandenfeld
leer	PN	Programmname

Mit PN wird dem Programm ein Name zugewiesen, wobei von der Angabe im Operandenfeld die ersten beiden Zeichen gewertet werden.
Das 1. Zeichen muß ein Buchstabe sein.

Beispiel: PN AUSGABE

AU wird als Name des Programms geführt.

3.2.8. Druck des Programmtitels TITL

Namensfeld	Operationsfeld	Operandenfeld
leer	TITL	'XX...X' ISO-Zeichenkette

Der Druck des Programmtitels, 'XX...X' im Kopf der Assembler-druckliste, wird angewiesen. Der Programmtitel steht im Operandenfeld. Er kann, einschließlich der Hochkommas, 61 Zeichen umfassen.

3.2.9. Formularvorschub EJEC

Namensfeld	Operationsfeld	Operandenfeld
leer	EJEC	leer

Mit dieser Anweisung wird beim Druck der Übersetzungsliste ein Formularvorschub veranlaßt.

3.2.10. Programmende END

Namensfeld	Operationsfeld	Operandenfeld
leer	END	leer

END ist die letzte Anweisung eines Quellprogramms.

3.2.11 Bedingte Assemblierung

Durch die bedingte Assemblierung wird ermöglicht, daß bestimmte Anweisungen bzw. Bereiche von Anweisungen von der Übersetzung ausgeschlossen werden. Die Bedingung wird im Operandenfeld angegeben. Der bedingte Bereich beginnt mit der Pseudoanweisung IF und endet mit der Pseudoanweisung ENIF. Die Schachtelung von IF-Anweisungen ist möglich. Wenn nn=0 ist, werden alle Anweisungen innerhalb des bedingten Bereiches nicht übersetzt. Wenn nn≠0 ist, werden die Anweisungen so übersetzt, als wären IF und ENIF nicht vorhanden. Für nn gelten die gleichen Festlegungen wie bei der DEF-Anweisung (Pkt. 3.2.6.).

Beispiel:

```
     KON: DEF  100
           .
           .
          IF   KON
          JMP  MARK1 ;Befehl wird übersetzt
          ENIF
           .
           .
     KON: DEF  0
           .
           .
          IF   KON
          JMP  MAT1  ;Befehl wird nicht übersetzt
          ENIF
           .
           .
           .
```

3.3. Makro-Programmiertechnik

Programmabschnitte, die unverändert an mehreren Stellen des Programmes benötigt werden, können als Unterprogramm oder als Makro geschrieben werden.
Unterprogramme werden vorzugsweise dann eingesetzt, wenn minimale Programmspeicherkapazität gefordert wird. Makros sind vorteilhaft verwendbar, wenn maximale Verarbeitungsgeschwindigkeiten und eine Flexibilität der Programmabschnitte durch einen Parameteraustausch benötigt werden. Die Begründung dafür liegt in der Tatsache, daß ein Makro während der Übersetzung bei jedem Aufruf in seiner ganzen Länge in das Anwenderprogramm eingebunden wird und das Unterprogramm dagegen ein separater Programmabschnitt bleibt, der bei jedem Aufruf angesprungen wird. Demzufolge werden die Anwenderprogramme bei Verwendung von Makros länger, es entfallen jedoch die Zeiten für das Umorganisieren des Befehlszählers.

Definition verwendeter Begriffe:

Makrodefinitionen: kennzeichnet die Befehlsfolge, die durch den Makronamen repräsentiert werden kann.

Makroaufruf: Programmstelle, an der ein Makro eingefügt werden soll.

Makroerweiterung: Programmabschnitt, den der Assembler anstelle jedes Makroaufrufs im Arbeitsprogramm generiert.

Makroparameter: Parameter, die in der Operandenliste der Makrodefinition angegeben werden. Beim Makroaufruf werden ihnen durch die Liste im Operandenfeld aktuelle Werte zugewiesen.

Makrokörper: Gesamtheit der Anweisungen, die in der Makrodefinition enthalten sind, begrenzt durch die Pseudoanweisungen MACR und ENDM.

Formalparameter: Plätze innerhalb des Makrokörpers, an denen bei der Assemblierung die aktuellen Parameter eingesetzt werden.

Bei der Makroprogrammiertechnik ist zu beachten:

- Makros müssen vor ihrem Aufruf definiert sein.
- Im Namensfeld steht der Name des Makros, mit dem er im Programm aufgerufen wird. Im Operationsfeld steht die Mnemonik zur Kennzeichnung der Makrodefinition. Im Operandenfeld werden die formalen Parameter angegeben.

Die allgemeine Form lautet:

Namensfeld	Operationsfeld	Operandenfeld
makroname:	MACR	$f_0,f_1,...,f_n$

- Der Makroname darf maximal 4 Zeichen lang sein und keine gültige Mnemonik sein.
- Die Anzahl der Parameter wird durch die maximal mögliche Zeichenanzahl des Operandenfeldes begrenzt.
- Die formalen Parameter können entfallen.
  Wenn sie vereinbart werden, müssen folgende Regeln beachtet werden:
  Jeder Parameter muß ein Symbol und darf maximal 5 Zeichen lang sein. Mnemoniks, Registernamen und Ausdrücke sind unzulässig.
- Im Makrokörper können im Namensfeld, Operationsfeld und Operandenfeld formale Parameter stehen. Die Symbole im Namensfeld haben lokale Bedeutung. Der Abschluß des Makrokörpers erfolgt durch die Anweisung ENDM im Operationsfeld.
- In einer Makrodefinition darf keine weitere Makrodefinition stehen.
- Makroaufrufe innerhalb eines Makros sind erlaubt. Es könne maximal 9 Makros ineinander verschachtelt werden. Die Parameterübergabe hat dabei aus der nächst höheren Ebene zu erfolgen.

Der Makroaufruf erfolgt durch Verwendung eines Namens entsprechend der eines Operationscodes. Die allgemeine Form lautet:

```
        name    makroname       a0,a1,,...,an
```

Dabei gilt:

- Die Verwendung des Namens ist zulässig, aber nicht erforderlich.

- Die hinter dem Makronamen stehende Zeichenfolge stellt die aktuellen Parameter dar, durch die Formalparameter im Makrokörper ersetzt werden. Dabei sind die einzelnen aktuellen Parameter voneinander durch Komma getrennt.
- Die Zuweisung der aktuellen Parameter beim Aufruf entspricht der Reihenfolge der Formalparameter in der Makrodefinition.
- Werden beim Makroaufruf weniger aktuelle Parameter übergeben als in der Makrodefinition aufgeführt werden, erhalten die übrigen Parameter den Wert 0. Werden mehr aktuelle Parameter angegeben als formale Parameter in der Definition vereinbart wurden, werden die übrigen Angaben ignoriert.
- Für die aktuellen Parameter sind Symbole, Zahlen und Zeichenketten erlaubt. Ausdrücke werden sofort aufgelöst und führen, falls sie einen verschieblichen Wert ergeben, zu Fehlern.
  Parameter mit negativem Vorzeichen sind unzulässig.

Beispiel:
```
MOVE:   MACR START,ZIEL,ANZ
        EXX
        LD   HL,START
        LD   DE,ZIEL
        LD   BC,ANZ
        LDIR
        EXX
        ENDM
```

Mit dem Macroaufruf:

```
TRANS:MOVE SYMS,SYMZ,SANZ
```

wird folgendes Programmstück ab der Stelle des Aufrufs in das laufende Programm eingefügt:

```
        .
        .
        .
        EXX
        LD  HL,SYMS
        LD  DE,SYMZ
        LD  BC,SANZ
        LDIR
        EXX
        .
        .
        .
```

Anlage 1: Aufbau der Quellprogrammzeilen

Befehlszeile:

	name	mnemonik	operand(,operand)	;kommentar	NL
Bedeutung:	Namensfeld	Operationsfeld	Operandenfeld	Kommentarfeld	Zeilenendezeichen
Zeichenanz.:	2-6 oder entf.	2-4	2-61 oder entfällt	1-61 oder entfällt	
			← max, 61 Zeichen →		
	← max. 71 Zeichen →				

Kommentarzeile:

	;kommentar	NL
Bedeutung:	Kommentarfeld	Zeilenendezeichen
Zeichenanz.:	1-71 (einschließlich Semikolon)	

Kommentarzeile eingerückt:

	b;kommentar		NL
Bedeutung:	Leerz.	Kommentarfeld	Zeilenendezeichen
Zeichenanz.:	1	1-61	

Leerzeile:

		NL
Bedeutung:	leer	Zeilenendezeichen

Anlage 2: Alphabetisch geordnete Zusammenstellung der mnemonischen Operationscodes

Mnemonischer Operationscode	Operanden	Deutsche Befehls-bezeichnung	Seite
ADD	HL,BC HL,DE HL,HL HL,SP	Addition	36
ADD	IX,IX		36
ADD	IY,IY		36
ADD	IX,BC IX,DE IX,SP		36
ADD	IY,BC IY,DE IY,SP		37
ADD	r M W (IX+d) (IY+d)		32
ADC	HL,BC HL,DE HL,HL HL,SP		37
ADC	r M n (IX+d) (IY+d)		33
AND	r M n (IX+d) (IY+d)	Logisches UND	38
BIT	r M b,(IX+d) (IY+d)	Bitposition testen	62
BER	nn	Speicherplatzreservierung	65
CAC	nn	Unterprogrammaufruf (wenn C-Flag=1)	58
CALL	nn	Unbedingter Unterpro-grammaufruf	58
CANC	nn	Unterprogrammaufruf (wenn C-Flag=0)	58

Mnemonischer Operationscode	Operanden	Deutsche Befehls-bezeichnung	Seite
CANZ	nn	Unterprogrammaufruf (wenn Z-Flag=0)	58
CAM	nn	Unterprogrammaufruf (wenn S-Flag=1)	58
CAP	nn	Unterprogrammaufruf (wenn S-Flag=0)	58
CAPE	nn	Unterprogrammaufruf (wenn P/V-Flag=1)	58
CAPO	nn	Unterprogrammaufruf (wenn P/V-Flag=0)	58
CAZ	nn	Unterprogrammaufruf (wenn Z-Flag=1)	58
CCF		Flagbefehl	53
CMP	r n M (IX+d) (IY+d)	Vergleich	39
CPL		Bitweises Negieren	53
CPD		Blocksuchbefehl	46
CPDR		Blocksuchbefehl	46
CPI		Blocksuchbefehl	45
CPIR		Blocksuchbefehl	45
DA	nn	Adressendedefinition	64
DAA		Akkumulatorbefehl	53
DB	n 'XX...X'	Datenbytedefinition	64
DEC	r M (IX+d) (IY+d)	Dekrementieren	35/36
DEC	dd	Dekrementieren	37
DEF	nn	Symboldefinition	66
DI		CPU-Steuerbefehl	62
DJNZ	e	Bedingter relativer Sprung	42
EI		CPU-Steuerbefehl	62

Mnemonischer Operationscode	Operanden	Deutsche Befehls-bezeichnung	Seite
EJEC		Formularvorschub	67
END		Programmende	67
EQU		Symboldefinition	65
EX	DE,HL (SP),HL (SP),IX (SP),IY	Registeraustauschbefehle	54/55
EXAF		Registeraustauschbefehl	54
EXX		Registeraustauschbefehl	54
HALT		CPU-Steuerbefehl	62
IN	n r	Eingabebefehl	50
IND		Eingabebefehl	51
INDR		Eingabebefehl	51
INF		Eingabebefehl	50
INI		Eingabebefehl	50
INIR		Eingabebefehl	50
IM0		CPU-Steuerbefehl	63
IM1		CPU-Steuerbefehl	63
IM2		CPU-Steuerbefehl	63
INC	r M (IX+d) (IY+d)	Inkrementieren	35
INC	dd	Inkrementieren	37
JMP	M nn (IX) (IY)	Sprungbefehl	41/42
JPC	nn	Sprungbefehl	41
JPNC	nn	Sprungbefehl	41
JPNZ	nn	Sprungbefehl	41
JPM	nn	Sprungbefehl	41
JPP	nn	Sprungbefehl	41
JPPE	nn	Sprungbefehl	41

Mnemonischer Operationscode	Operanden	Deutsche Befehls-bezeichnung	Seite
JPPO	nn	Sprungbefehl	41
JPZ	nn	Sprungbefehl	41
JR	e	Sprungbefehl	41
JRC	e	Sprungbefehl	42
JRNC	e	Sprungbefehl	42
JRNZ	e	Sprungbefehl	41
JRZ	e	Sprungbefehl	41
LD	r1,r2 r,n r,M r,(IX+d) r,(IY+d) M,r (IX+d),r (IY+d),r M,n (IX+d),n (IY+d),n A,(BC) A,(DE) A,(nn)	Ladebefehle (1-Byte)	24-27
LD	(BC),A (DE),A (nn),A A,I A,R I,A R,A	Ladebefehle (1-Byte)	28
LD	dd,nn IX,nn IY,nn HL,(nn) dd,(nn) IX,(nn) IY,(nn) (nn),HL (nn),dd (nn),IX (nn),IY SP,HL SP,IX SP,IY	Ladebefehle (2-Byte)	29-31
LDD		Blocktransportbefehl	45
LDDR		Blocktransportbefehl	44
LDI		Blocktransportbefehl	44
LDIR		Blocktransportbefehl	43

Mnemonischer Operationscode	Operanden	Deutsche Befehls-bezeichnung	Seite
NEG		Akkumulatorbefehl	53
NOP		CPU-Steuerbefehl	62
OR	r n M (IX+d) (IY+d)	Logisches ODER	38
OUT	n r	Ausgabebefehl	51
OUTD		Ausgabebefehl	52
OUTI		Ausgabebefehl	52
OTDR		Ausgabebefehl	52
OTIR		Ausgabebefehl	52
ORG	nn	Setzen des Befehlszählers	65
PUSH	AF BC DE HL IX IY	Indirekte Register-operation	56
POP	AF BC DE HL IX IY	Indirekte Register-operation	56/57
PN	Programmname	Programme zuweisen	66
RC		Unterprogrammrücksprung	59
RES	b,s	Bitmanipulationsbefehl	61
RET		Unterprogrammrücksprung (unbedingt)	59
RETI		Rücksprung aus einer Interrupt-Behandlungs-routine	60
RETN		Rücksprung aus einer Interrupt-Behandlungs-routine	60
RL	r (HL) (IX+d) (IY+d)	Linksrotation	47

Mnemonischer Operationscode	Operanden	Deutsche Befehls-bezeichnung	Seite
RLA		Linksrotation	47
RLC	r (HL) (IX+d) (IY+d)	Linksrotation	47
RLCA		Linksrotation	46
RLD		Zyklische Verschiebung nach links	48
RM		Unterprgrammrücksprung	59
RNC		Unterprgrammrücksprung	59
RNZ		Unterprgrammrücksprung	59
RP		Unterprgrammrücksprung	59
RPE		Unterprgrammrücksprung	59
RPO		Unterprgrammrücksprung	59
RR	r (HL) (IX+d) (IY+d)	Rechtsrotation	47
RRA		Rechtsrotation	47
RRC	r (HL) (IX+d) (IY+d)	Rechtsrotation	47
RRCA		Rechtsrotation	46
RRD		Zyklische Verschiebung nach rechts	49
RST	00H 08H 10H 18H 20H 28H 30H 38H	Unterprogrammaufruf	58
RZ		Unterprogrammrücksprung	59
SBC	n r M (IX+d) (IY+d)	Arithmetische Operation	34
SBC	HL,dd	Arithmetische Operation	37

Mnemonischer Operationscode	Operanden	Deutsche Befehls-bezeichnung	Seite
SCF		C-Flag setzen	54
SET	b,r b,M b,(IX+d) b,(IY+d)	Bitmanipulation	61
SLA	r (HL) (IX+d) (IY+d)	Verschiebebefehl	48
SRA	r (HL) (IX+d) (IY+d)	Verschiebebefehl	48
SRL	r (HL) (IX+d) (IY+d)	Verschiebebefehl	48
SUB	r M n (IX+d) (IY+d)	Subtraktion	33/34
TITL	'XX...X'	Druck des Programmtitels	67
XOR	r n M (IX+d) (IY+d)	Exclusives ODER	39

Anlage 3: Zusammenstellung der Befehle nach Funktionsgruppen

Nr	Mnemo-nik	Symbolische Operation	Flags						Operationscode binär/hexadez.				Anzahl			Bemerkungen
			CY	Z	P/V	S	N	H	1.Byte	2.Byte	3.Byte	4.Byte	Byte	Zyklen	Takte	
1	LD r_1,r_2	$r_1 \leftarrow r_2$	.	.	.	.	.	.	01-r_1--r_2-				1	1	4	r_1,r_2,r_3 Register
2	LD r,n	r ← n	.	.	.	.	.	.	00-r-110	n			2	2	7	0 0 0 B 0 0 1 C 0 1 0 D 0 1 1 E 1 0 0 H 1 0 1 L 1 1 1 A
3	LD r,M	r ← M	.	.	.	.	.	.	01-r-110				1	2	7	
4	LD,r(IX+d)	r ← (IX+d)	.	.	.	.	.	.	DD	01-r-110	d		3	5	19	
5	LD,r(IY+d)		.	.	.	.	.	.	FD	01-r-110	d		3	5	19	
6	LD M,r		.	.	.	.	.	.	01110-r-				1	2	7	M entspricht (HL)
7	LD(IX+d),r		.	.	.	.	.	.	DD	01110-r-	d		3	5	19	
8	LD(IY+d),r		.	.	.	.	.	.	FD	01110-r-	d		3	5	19	
9	LD M,n		.	.	.	.	.	.	36	n			2	3	10	
10	LD(IX+d),n		.	.	.	.	.	.	DD	36	d	n	4	5	19	
11	LD(IY+d),n		.	.	.	.	.	.	FD	36	d	n	4	5	19	
12	LD A,(BC)		.	.	.	.	.	.	0A				1	2	7	
13	LD A,(DE)		.	.	.	.	.	.	1A				1	2	7	

1-Byte-Ladebefehle

Nr	Mnemonik	Symbolische Operation	Flags						Operationscode binär/hexadez.				Anzahl			Bemerkungen
			CY	Z	P/V	S	N	H	1.Byte	2.Byte	3.Byte	4.Byte	Byte	Zyklen	Takte	
14	LD A,(nn)	A ← (nn)	.	.	.	.	.	.	3A	n_L	n_H		3	4	13	Flag-Kennz.:
15	LD (BC),A	(BC)← A	.	.	.	.	.	.	02				1	2	7	. nicht beeinflußt
16	LD (DE),A	(DE ← A	.	.	.	.	.	.	12				1	2	7	0 rückgesetzt
17	LD (nn),A	(nn)← A	.	.	.	.	.	.	32	n_L	n_H		3	4	13	1 gesetzt
18	LD A,I	A ← I	.	↕	IFF	↕	0	0	ED	57			2	2	9	↕ durch Operations-ergebnis beeinflußt
19	LD A,R	A ← R	.	↕	IFF	↕	0	0	ED	5F			2	2	9	IFF IFF2 → P/V
20	LD I,A	I ← A	.	.	.	.	.	.	ED	47			2	2	9	
21	LD R,A	R ← A	.	.	.	.	.	.	ED	4F			2	2	9	

1-Byte-Ladebefehle

Nr	Mnemonik	Symbolische Operation	Flags						Operationscode binär/hexadez.				Anzahl			Bemerkungen
			CY	Z	P/V	S	N	H	1.Byte	2.Byte	3.Byte	4.Byte	Byte	Zyklen	Takte	
1	LD dd,nn	dd ← nn	.	.	.	.	.	.	00dd0001	n_L	n_H		3	3	10	
2	LD IX,nn	IX ← nn	.	.	.	.	.	.	DD	21	n_L	n_H	4	4	14	pp Reg.-Paar
3	LD IY,nn	IY ← nn	.	.	.	.	.	.	FD	21	n_L	n_H	4	4	14	00 BC
4	LD HL,(nn)	H ← (nn+1) L ← (nn)	.	.	.	.	.	.	2A	n_L	n_H		3	5	16	01 DE 11 SP
5	LD pp,(nn)	pp_H ← (nn+1) pp_L ← (nn)	.	.	.	.	.	.	ED	01dd1011	n_L	n_H	4	6	20	dd Reg.-Paar
6	LD IX,(nn)	IX_H ← (nn+1) IX_L ← (nn)	.	.	.	.	.	.	DD	2A	n_L	n_H	4	6	20	00 BC 01 DE
7	LD IY,(nn)	IY_H ← (nn+1) IY_L ← (nn)	.	.	.	.	.	.	FD	2A	n_L	n_H	4	6	20	10 HL 11 SP
8	LD (nn),HL	(nn+1) ← H (nn) ← L	.	.	.	.	.	.	22	n_L	n_H		3	5	16	
9	LD (nn),pp	(nn+1) ← pp_H (nn) ← pp_L	.	.	.	.	.	.	ED	01dd0011	n_L	n_H	4	6	20	Flag-Kennz.: . nicht beeinflußt
10	LD (nn),IX	(nn+1) ← IX_H (nn) ← IX_L	.	.	.	.	.	.	ED	22	n_L	n_H	4	6	20	
11	LD (nn),IY	(nn+1) ← IY_H (nn) ← IY_L	.	.	.	.	.	.	FD	22	n_L	n_H	4	6	20	
12	LD SP,HL	SP ← HL	.	.	.	.	.	.	F9				1	1	6	
13	LD SP,IX	SP ← IX	.	.	.	.	.	.	DD	F9			2	2	10	
14	LD SP,IY	SP ← IY	.	.	.	.	.	.	FD	F9			2	2	10	

2-Byte-Ladebefehle

Nr	Mnemonik	Symbolische Operation	Flags						Operationscode binär/hexadez.				Anzahl			Bemerkungen
			CY	Z	P/V	S	N	H	OP				Byte	Zyklen	Takte	
1	ADD s	A ← A+s	⇕	⇕	V	⇕	0	⇕	000							s steht für
2	ADC s	A ← A+s+CY	⇕	⇕	V	⇕	0	⇕	001							r,n,M,(IX+d),(IY+d)
3	SUB s	A ← A-s	⇕	⇕	V	⇕	1	⇕	010							(siehe unten)
4	SBC s	A ← A-s-CY	⇕	⇕	V	⇕	1	⇕	011							Flag-Kennz.: . nicht beeinflußt
5	AND s	A ← A ∧ s	0	⇕	P	⇕	0	1	100							0 rückgeetzt
6	XOR s	A ← A ⊕ s	0	⇕	P	⇕	0	0	101							1 gesetzt
7	OR s	A ← A v s	0	⇕	P	⇕	0	0	110							⇕ durch Operations-ergebnis beeinflußt
8	CMP s	Flags ← A-s	⇕	⇕	V	⇕	1	⇕	111							P gerade Parität V Überlauf
									1.Byte	2.Byte	3.Byte					
1	OP r		sie	⇕	sie	⇕	sie	⇕	10-OP--r-				1	1	4	OP steht für
2	OP n		he	⇕	he	⇕	he	⇕	11-OP-110	n			2	2	7	ADD, ADC, SUB,
3	OP M			⇕		⇕		⇕	10-OP-110				1	2	7	SDC, AND, XOR,
4	OP (IX+d)		ob	⇕	ob	⇕	ob	⇕	DD	10-OP-110	d		3	5	19	OR, CMP
5	OP (IY+d)		en	⇕	en	⇕	en	⇕	FD	10-OP-110	d		3	5	19	(siehe oben)

1-Byte-Arithmetik-/Logikbefehle

Nr	Mnemonik	Symbolische Operation	Flags						Operationscode binär/hexadez.				Anzahl			Bemerkungen
			CY	Z	P/V	S	N	H	1.Byte	2.Byte	3.Byte		Byte	Zyklen	Takte	
9	INC r	r ← r+1	.	↕	V	↕	0	↕	00-r-100				1	1	4	r Register 000 B 001 C 010 D 011 E 100 H 101 L 111 A
10	INC M	M ← M+1	.	↕	V	↕	0	↕	34				1	3	11	
11	INC (IX+d)	(IX+d) ← (IX+d)+1	.	↕	V	↕	0	↕	DD	34	d		3	6	23	
12	INC (IY+d)	(IY+d) ← (IY+d)+1	.	↕	V	↕	0	↕	FD	34	d		3	6	23	
13	DEC r	r ← r-1	.	↕	V	↕	1	↕	00-r-101				1	1	4	
14	DEC M	M ← M-1	.	↕	V	↕	1	↕	35				1	3	11	M entspricht HL
15	DEC (IX+d)	(IX+d) ← (IX+d)-1	.	↕	V	↕	1	↕	DD	35	d		3	6	23	
16	DEC (IY+d)	(IY+d) ← (IY+d)-1	.	↕	V	↕	1	↕	FD	35	d		3	6	23	

1-Byte-Arithmetik-/Logikbefehle

Nr	Mnemonik	Symbolische Operation	Flags						Operationscode binär/hexadez.				Anzahl			Bemerkungen
			CY	Z	P/V	S	N	H	1.Byte	2.Byte			Byte	Zyklen	Takte	
1	ADD HL,dd	HL ← HL+dd	⇕	.	.	.	0	?	00dd1001				1	3	11	dd Reg.-Paar
2	ADD IX,IX	IX ← IX+IX	⇕	.	.	.	0	?	DD	29			2	4	15	00 BC 01 DE
3	ADD IY,IY	IY ← IY+IY	⇕	.	.	.	0	?	FD	29			2	4	15	10 HL 11 SP
4	ADD HL,dd	HL ← HL+dd+CY	⇕	⇕	V	⇕	0	?	ED	01dd1010			2	4	15	pp Reg.-Paar
5	SBC HL,dd	HL ← HL-dd-CY	⇕	⇕	V	⇕	0	?	ED	01dd0010			2	4	15	00 BC 01 DE
6	ADD IX,pp	IX ← IX+pp	⇕	.	.	.	0	?	DD	00pp1001			2	4	15	11 SP
7	ADD IY,pp	IY ← IY+pp	⇕	.	.	.	0	?	FD	00pp1001			2	4	15	
8	INC dd	dd ← dd+1	.	.	.	.	.	.	00dd0011				1	1	6	Flag-Kennz.:
9	INC IX	IX ← IX+1	.	.	.	.	.	.	DD	23			2	2	10	. nicht beeinflußt
10	INC IY	IY ← IY+1	.	.	.	.	.	.	FD	23			2	2	10	0 rückgesetzt 1 gesetzt
11	DEC dd	dd ← dd-1	.	.	.	.	.	.	00dd1011				1	1	6	⇕ durch Operation beeinflußt
12	DEC IX	IX ← IX-1	.	.	.	.	.	.	DD	2B			2	2	10	V Überlauf ? unbestimmt
13	DEC IY	IY ← IY-1	.	.	.	.	.	.	FD	2B			2	2	10	

2-Byte-Arithmetikkbefehle

Nr	Mnemonik	Symbolische Operation	Flags						Operationscode binär/hexadez.				Anzahl			Bemerkungen
			CY	Z	P/V	S	N	H	1.Byte	2.Byte	3.Byte		Byte	Zyklen	Takte	
1	JMP nn	PC ← nn	.	.	.	.	.	.	C3	n_L	n_H		3	3	10	Flag-Kennz.:
2	JPNZ nn	PC ← nn bei Z-Flag=0	.	.	.	.	.	.	C2	n_L	n_H		3	3	10	. nicht beeinflußt
3	JPZ nn	PC ← nn bei Z-Flag=1	.	.	.	.	.	.	CA	n_L	n_H		3	3	10	bei nicht erfüllter Bedingung wird der nächste Befehl auf-gerufen
4	JPNC nn	PC ← nn bei CY-Flag=0	.	.	.	.	.	.	D2	n_L	n_H		3	3	10	
5	JPC nn	PC ← nn bei CY-Flag=1	.	.	.	.	.	.	DA	n_L	n_H		3	3	10	
6	JPPO nn	PC ← nn bei P/V-Flag=0	.	.	.	.	.	.	E2	n_L	n_H		3	3	10	
7	JPPE nn	PC ← nn bei P/V-Flag=1	.	.	.	.	.	.	EA	n_L	n_H		3	3	10	
8	JPP nn	PC ← nn bei S-Flag=0	.	.	.	.	.	.	F2	n_L	n_H		3	3	10	
9	JPM nn	PC ← nn bei S-Flag=1	.	.	.	.	.	.	FA	n_L	n_H		3	3	10	

Sprungbefehle

Nr	Mnemonik	Symbolische Operation	Flags						Operationscode binär/hexadez.				Anzahl			Bemerkungen
			CY	Z	P/V	S	N	H	1.Byte	2.Byte			Byte	Zyklen	Takte	
10	JR e	PC ← PC+e	.	.	.	.	.	.	18	e-2			2	3	12	
11	JRNZ e	PC ← PC+e bei Z-Flag=0	.	.	.	.	.	.	20	e-2			2 2	3 2	12 7	Bedingung erfüllt Bedingung nicht erfüllt
12	JRZ e	PC ← PC+e bei Z-Flag=1	.	.	.	.	.	.	28	e-2			2 2	3 2	12 7	Bedingung erfüllt Bedingung nicht erfüllt
13	JRNC e	PC ← PC+e bei CY-Flag=0	.	.	.	.	.	.	30	e-2			2 2	3 2	12 7	Bedingung erfüllt Bedingung nicht erfüllt
14	JRC e	PC ← PC+e bei CY-Flag=1	.	.	.	.	.	.	38	e-2			2 2	3 2	12 7	Bedingung erfüllt Bedingung nicht erfüllt
15	JMP M	PC ← HL	.	.	.	.	.	.	E9				1	1	4	
16	JMP (IX)	PC ← IX	.	.	.	.	.	.	DD	E9			2	2	8	
17	JMP (IY)	PC ← IY	.	.	.	.	.	.	FD	E9			2	2	8	
18	DJNZ e	B ← B-1 PC ← PC+e bei B≠0 sonst nächster Befehl	.	.	.	.	.	.	10	e-2			2 2	3 2	13 8	Bedingung erfüllt Bedingung nicht erfüllt

Sprungbefehle

Nr	Mnemonik	Symbolische Operation	Flags						Operationscode binär/hexadez.				Anzahl			Bemerkungen
			CY	Z	P/V	S	N	H	1.Byte	2.Byte			Byte	Zyklen	Takte	
1	LDI	(DE) ← M DE ← DE+1 HL ← HL+1 BC ← BC-1	.	.	⇕ 1)	.	0	0	ED	A0			2	4	16	M entspricht (HL)
2	LDIR	(DE) ← M DE ← DE+1 HL ← HL+1 BC ← BC-1 wiederholen bis BC=0	.	.	.	.	0	0	ED	B0			2 2	5 4	21 16	Flag-Kennz.: . nicht beeinflußt 0 rückgesetzt ⇕ durch Operations-ergebnis beeinflußt
3	LDD	(DE) ← M DE ← DE-1 HL ← HL-1 BC ← BC-1	.	.	⇕ 1)	.	0	0	ED	A8			2	4	16	1) P/V=0, wenn BC=0, sonst P/V=1
4	LDDR	(DE) ← M DE ← DE-1 HL ← HL-1 BC ← BC-1 wiederholen bis BC=0	.	.	0	.	0	0	ED	B8			2 2	5 4	21 16	für BC≠0 für BC=0

Blocktransport- und -suchbefehle

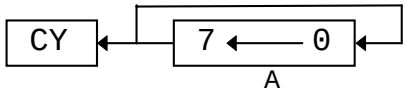
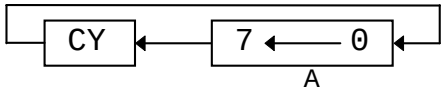
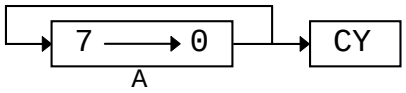
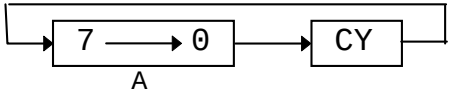
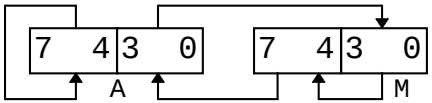
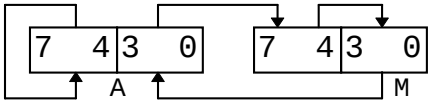
Nr	Mnemonik	Symbolische Operation	Flags						Operationscode binär/hexadez.				Anzahl			Bemerkungen
			CY	Z	P/V	S	N	H	1.Byte	2.Byte			Byte	Zyklen	Takte	
5	CPI	Flags ← A-M HL ← HL+1 BC ← BC-1	.	⇕ 2)	⇕ 1)	⇕	1	⇕	ED	A1			2	4	16	M entspricht (HL)
6	CPIR	Flags ← A-M HL ← HL+1 BC ← BC-1 wiederholen bis A=M oder BC=0	.	⇕ 2)	⇕ 1)	⇕	1	⇕	ED	B1			2 2	5 4	21 16	für BC≠0 und A≠M für BC=0 oder A=M
7	CPD	Flags ← A-M HL ← HL-1 BC ← BC-1	.	⇕ 2)	⇕ 1)	⇕	1	⇕	ED	A9			2	4	16	
8	CPDR	Flags ← A-M HL ← HL-1 BC ← BC-1 wiederholen bis A=M oder BC=0	.	⇕ 2)	⇕ 1)	⇕	1	⇕	ED	B9			2 2	5 4	21 16	für BC≠0 und A≠M für BC=0 oder A=M

Flag-Kennz.:

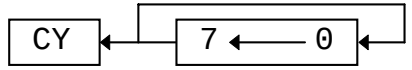
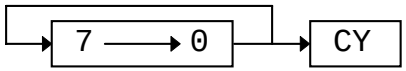
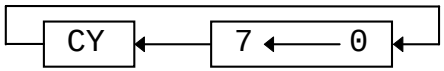
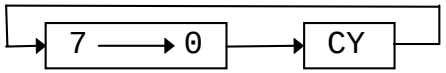
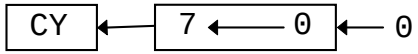
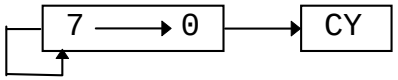
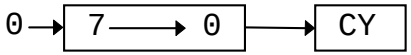
. nicht beeinflußt
1 gesetzt
⇕ durch Operations-ergebnis beeinflußt

1) P/V=0, wenn BC=0, sonst P/V=1
2) Z=1, wenn A=M, sonst Z=0

Blocktransport- und -suchbefehle

Nr	Mnemonik	Symbolische Operation	Flags						Operationscode			Anzahl			Bemerkungen
			CY	Z	P/V	S	N	H	1.Byte	2.Byte		Byte	Zyklen	Takte	
1	RLCA		⇕	.	.	.	0	0	07			1	1	4	Flag-Kennz.:
2	RLA		⇕	.	^ .	.	0	0	17			1	1	4	. nicht beeinflußt 0 rückgesetzt 1 gesetzt
3	RRCA		⇕	.	.	.	0	0	0F			1	1	4	⇕ durch Operations-ergebnis beeinflußt P gerade Parität
4	RRA			.	.	.	0	0	1F			1	1	4	
5	RLD		.	⇕	P	⇕	0	0	ED	6F		2	5	18	
6	RRD		.	⇕	P	⇕	0	0	ED	67		2	5	18	

Verschiebebefehle

Nr	Mnemonik	Symbolische Operation	Flags						Operationscode	Anzahl			Bemerkungen
			CY	Z	P/V	S	N	H		Byte	Zyklen	Takte	
7	RLC s		↕	↕	P	↕	0	0	000				s steht für r,M,(IX+d),(IY+d) (siehe unten)
8	RRC s		↕	↕	P	↕	0	0	001				
9	RL s		↕	↕	P	↕	0	0	010				r Register 000 B 001 C 010 D 011 E 100 H 101 L 111 A
10	RR s		↕	↕	P	↕	0	0	011				
11	SLA s		↕	↕	P	↕	0	0	100				M entspricht (HL)
12	SRA s		↕	↕	P	↕	0	0	101				
13	SRL s		↕	↕	P	↕	0	0	111				

			CY	Z	P/V	S	N	H	1.Byte	2.Byte	3.Byte	4.Byte				
1	OP r		.		P		0	0	CB	00-OP--r-			2	2	8	OP steht für RLC, RRC, RL, RR, SLA, SRA, SRL (siehe oben)
2	OP M		.		P		0	0	CB	00-OP-110			2	4	15	
3	OP(IX+d)		.		P		0	0	DD	CB	d	00-OP-110	4	6	23	
4	OP(IY+d)		.		P		0	0	FD	CB	d	00-OP-110	4	6	23	

Verschiebebefehle

Nr	Mnemonik	Symbolische Operation	Flags						Operationscode binär/hexadez.		Anzahl			Bemerkungen	
			CY	Z	P/V	S	N	H	1.Byte	2.Byte	Byte	Zyklen	Takte		
1	IN n	A ← (n)	·	·	·	·	·	·	DB	n	2	3	11	n auf A0 bis A7 A auf A8 bis A15 D0 bis D7 → A	r Register 000 B 001 C 010 D 011 E 100 H 101 L 111 A
2	IN r	r ← (C)	·	↕	P	↕	0	0	ED	01-r-000	2	3	12	C auf A0 bis A7 B auf A8 bis A15 D0 bis D7 → r	
3	INF	setzt nur Flags	·	↕	P		0	0	ED	70	2	3	12	C auf A0 bis A7 B auf A8 bis A15 D0 bis D7 → Flags	
4	INI	M ← (C) B ← B-1 HL ← HL+1	·	↕ 3)	?	?	1	?	ED	A2	2	4	16	C auf A0 bis A7 B auf A8 bis A15 D0 bis D7 → M	M entspricht (HL)
5	INIR	M ← (C) B ← B-1 HL ← HL+1 wiederholen bis B=0	·	1	?	?	1	?	ED	B2	2	5	21	wenn B≠0 wenn B=0 C auf A0 bis A7 B auf A8 bis A15 D0 bis D7 → M	Flag-Kennz.: . nicht beeinflußt 0 rückgesetzt 1 gesetzt
6	IND	M ← (C) B ← B-1 HL ← HL-1	·	↕ 3)	?	?	1	?	ED	AA	2	4	16	C auf A0 bis A7 B auf A8 bis A15 D0 bis D7 → M	↕ durch Daten beeinflußt P gerade Parität ? unbestimmt 3) Z=1 wenn B=0, sonst Z=0
7	INDR	M ← (C) B ← B-1 HL ← HL-1 wiederholen bis B=0	·	1	?	?	1	?	ED	BA	2 2	5 4	21 16	wenn B≠0 wenn B=0 C auf A0 bis A7 B auf A8 bis A15 D0 bis D7 → M	

Eingabebefehle

Nr	Mnemonik	Symbolische Operation	Flags						Operationscode binär/hexadez.		Anzahl			Bemerkungen	
			CY	Z	P/V	S	N	H	1.Byte	2.Byte	Byte	Zyklen	Takte		
1	OUT n	(n) ← A	·	·	·	·	·	·	D3	n	2	3	11	n auf A0 bis A7 A auf A8 bis A15 A auf D0 bis D7	r Register 000 B 001 C 010 D 011 E 100 H 101 L 111 A
2	OUT r	(C) ← r	·	·	·	·	·	·	ED	01-r-001	2	3	12	C auf A0 bis A7 B auf A8 bis A15 r auf D0 bis D7	
3	OUTI	(C) ← M B ← B-1 HL ← HL+1	·	⇕ 3)	?	?	1	?	ED	A3	2	4	16	C auf A0 bis A7 B auf A8 bis A15 M auf D0 bis D7	
4	OTIR	(C) ← M B ← B-1 HL ← HL+1 wiederholen bis B=0	·	1	?	?	1	?	ED	B3	2 2	5 4	21 16	wenn B≠0 wenn B=0 C auf A0 bis A7 B auf A8 bis A15 M auf D0 bis D7	M entspricht (HL) Flag-Kennz.: . nicht beeinflußt 1 gesetzt
5	OUTD	(C) ← M B ← B-1 HL ← HL-1	·	⇕ 3)	?	?	1	?	ED	AB	2	4	16	C auf A0 bis A7 B auf A8 bis A15 M auf D0 bis D7	
6	OTDR	(C) ← M B ← B-1 HL ← HL-1 wiederholen bis B=0	·	1	?	?	1	?	ED	BB	2 2	5 4	21 16	wenn B≠0 wenn B=0 C auf A0 bis A7 B auf A8 bis A15 M auf D0 bis D7	⇕ durch Operationsergebnis beeinflußt ? unbestimmt 3) Z=1 wenn B=0, sonst Z=0

Ausgabebefehle

Nr	Mnemonik	Symbolische Operation	Flags						Operationscode binär/hexadez.				Anzahl			Bemerkungen
			CY	Z	P/V	S	N	H	1.Byte	2.Byte			Byte	Zyklen	Takte	
1	DAA	Dezimalkorrektur des Akkumulator-inhaltes nach Addition oder Subtraktion mit gepackten BCD-Operanden	⇕	⇕	p	⇕	.	⇕	27				1	1	4	
2	CPL	A ← $\overline{A}$	.	.	.	.	1	1	2F				1	1	4	Bitweise Komplementierung von A
3	NEG	A ← 0-A	⇕	⇕	V	⇕	1	⇕	ED	44			2	2	8	Bildung Zweierkomplement von A (A+1 → A)
4	CCF	CY ← $\overline{CY}$	⇕	.	.	.	0	?	3F				1	1	4	Komplementierung CY-Flag
5	SCF	CY ← 1	1	.	.	.	0	0	37				1	1	4	Setzen CY-Flag
																Flag-Kennz.:
																. nicht beeinflußt 0 rückgesetzt durch Operations- ergebnis beeinflußt P gerade Parität V Überlauf ? unbestimmt

Spezielle Akkumulator- und Flagbefehle

Nr	Mnemonik	Symbolische Operation	Flags						Operationscode binär/hexadez.				Anzahl			Bemerkungen
			CY	Z	P/V	S	N	H	1.Byte	2.Byte			Byte	Zyklen	Takte	
1	PUSH qq	$(SP-2) \leftarrow qq_L$ $(SP-1) \leftarrow qq_H$	.	.	.	.	.	.	11qq0101				1	3	11	qq Reg.-Paar 00 BC 01 DE 10 HL 11 AF
2	PUSH IX	$(SP-2) \leftarrow IX_L$ $(SP-1) \leftarrow IX_H$	.	.	.	.	.	.	DD	E5			2	4	15	
3	PUSH IY	$(SP-2) \leftarrow IY_L$ $(SP-1) \leftarrow IY_H$	.	.	.	.	.	.	FD	E5			2	4	15	
4	POP qq	$qq_H \leftarrow (SP+1)$ $qq_L \leftarrow (SP)$	. 2)	. 2)	. 2)	. 2)	. 2)	. 2)	11qq0001				1	3	10	
5	POP IX	$IX_H \leftarrow (SP+1)$ $IX_L \leftarrow (SP)$	.	.	.	.	.	.	DD	E1			2	4	14	Flag-Kennz.: . nicht beeinflußt
6	POP IY	$IY_H \leftarrow (SP+1)$ $IY_L \leftarrow (SP)$	.	.	.	.	.	.	FD	E1			2	4	14	1) bei EXAF werden die Falgs ausgetauscht
7	EX DE,HL	$DE \leftrightarrow \overline{HL}$	.	.	.	.	.	.	EB				1	1	4	2) bei POP AF werden die Flags überschrieben
8	EXAF	$AF \leftrightarrow \overline{AF'}$	1)	1)	1)	1)	1)	1)	08				1	1	4	
9	EXX	$BC \leftrightarrow \overline{BC'}$ $DE \leftrightarrow DE'$ $HL \leftrightarrow HL'$	.	.	.	.	.	.	D9				1	1	4	
10	EX (SP),HL	$H \leftrightarrow (SP+1)$ $L \leftrightarrow (SP)$	.	.	.	.	.	.	E3				1	5	19	
11	EX (SP),IX	$IX_H \leftrightarrow (SP+1)$ $IX_L \leftrightarrow (SP)$	.	.	.	.	.	.	DD	E3			2	6	23	
12	EX (SP),IY	$IY_H \leftrightarrow (SP+1)$ $IY_L \leftrightarrow (SP)$	.	.	.	.	.	.	FD	E3			2	6	23	

PUSH-, POP- und EXCHANGE-Befehle

Nr	Mnemonik	Symbolische Operation	Flags						Operationscode (binär/hexadez.)				Anzahl			Bemerkungen
			CY	Z	P/V	S	N	H	1.Byte	2.Byte	3.Byte		Byte	Zyklen	Takte	
1	CALL nn	(SP-1) ← PC_H (SP-2) ← PC_L PC ← nn	·	·	·	·	·	·	CD	n_L	n_H		3	5	17	Flag-Kennz.: . nicht beeinflußt
2	CANZ nn	wie CALL bei Z-Flag = 0	·	·	·	·	·	·	C4	n_L	n_H		3 3	5 3	17 10	bei nicht erfüllter Bedingung wird der nächste Befehl aufgerufen
3	CAZ nn	wie CALL bei Z-Flag = 1	·	·	·	·	·	·	CC	n_L	n_H		3 3	5 3	17 10	
4	CANC nn	wie CALL bei CY-Flag = 0	·	·	·	·	·	·	D4	n_L	n_H		3 3	5 3	17 10	Bedingung erfüllt: 17 Takte
5	CAC nn	wie CALL bei CY-Flag = 1	·	·	·	·	·	·	DC	n_L	n_H		3 3	5 3	17 10	Bedingung nicht erfüllt: 10 Takte
6	CAPO nn	wie CALL bei P/V-Flag = 0	·	·	·	·	·	·	E4	n_L	n_H		3 3	5 3	17 10	
7	CAPE nn	wie CALL bei P/V-Flag = 1	·	·	·	·	·	·	EC	n_L	n_H		3 3	5 3	17 10	t p 000 00H 001 08H 010 10H 011 18H 100 20H 101 28H 110 30H 111 38H
8	CAP nn	wie CALL bei S-Flag = 0	·	·	·	·	·	·	F4	n_L	n_H		3 3	5 3	17 10	
9	CAM nn	wie CALL bei S-Flag = 1	·	·	·	·	·	·	FC	n_L	n_H		3 3	5 3	17 10	
10	RST p	(SP-1) ← PC_H (SP-1) ← PC_L PC_H ← 0 PC_L ← p	·	·	·	·	·	·	11-t-111				1	3	11	

Unterprogramm-Aufrufbefehle

Nr	Mnemonik	Symbolische Operation	Flags						Operationscode (binär/hexadez.)				Anzahl			Bemerkungen
			CY	Z	P/V	S	N	H	1.Byte	2.Byte			Byte	Zyklen	Takte	
1	RET	$PC_L \leftarrow (SP)$ $PC_H \leftarrow (SP+1)$	.	.	.	.	.	.	C9				1	3	10	Flag-Kennz.: . nicht beeinflußt
2	RNZ	wie RET bei Z-Flag = 0	.	.	.	.	.	.	C0				1 1	3 1	11 5	
3	RZ	wie RET bei Z-Flag = 1	.	.	.	.	.	.	C8				1 1	3 1	11 5	bei nicht erfüllter Bedingung wird der nächste Befehl aufgerufen
4	RNC	wie RET bei CY-Flag = 0	.	.	.	.	.	.	D0				1 1	3 1	11 5	
5	RC	wie RET bei CY-Flag = 1	.	.	.	.	.	.	D8				1 1	3 1	11 5	Bedingung erfüllt: 11 Takte
6	RPO	wie RET bei P/V-Flag = 0	.	.	.	.	.	.	E0				1 1	3 1	11 5	Bedingung nicht erfüllt: 5 Takte
7	RPE	wie RET bei P/V-Flag = 1	.	.	.	.	.	.	E8				1 1	3 1	11 5	
8	RP	wie RET bei S-Flag = 0	.	.	.	.	.	.	F0				1 1	3 1	11 5	
9	RM	wie RET bei S-Flag = 1	.	.	.	.	.	.	F8				1 1	3 1	11 5	
10	RETI	wie RET nach Interrupt	.	.	.	.	.	.	ED	4D			2	4	14	setzt Peripherie-schaltkreise zurück
11	RETN	wie RET nach nicht maskierbarem Interrupt	.	.	.	.	.	.	ED	45			2	4	14	setzt Interruptfreigabe (IFF1) in den Zusatnd wie vor NMI

Unterprogramm-Rücksprungbefehle

Nr	Mnemonik	Symbolische Operation	Flags						Operationscode (binär/hexadez.)				Anzahl			Bemerkungen
			CY	Z	P/V	S	N	H	1.Byte	2.Byte	3.Byte	4.Byte	Byte	Zyklen	Takte	
1	BIT b,r	$Z \leftarrow \overline{r_h}$	·		?	?	0	1	CB	01-b--r-			2	2	8	BIT setzt Z-Flag mit dem durch b
2	BIT b,M	$Z \leftarrow \overline{M_h}$	·		?	?	0	1	CB	01-b-110			2	3	12	augewählten negierten Bit des
3	BIT b,(IX+d)	$Z \leftarrow \overline{(IX+d)_h}$	·		?	?	0	1	DD	CB	d	01-b-110	4	5	20	Registers r oder des Speicherin-
4	BIT b,(IY+d)	$Z \leftarrow \overline{(IY+d)_h}$	·		?	?	0	1	FD	CB	d	01-b-110	4	5	20	halts M oder des Inhalts des durch
5	SET b,r	$r_h \leftarrow 1$	·	·	·	·	·	·	CB	11-b--r-			2	2	8	IX+d bzw. IY+d adressierten Speicherplatzes
6	SET b,M	$M_h \leftarrow 1$	·	·	·	·	·	·	CB	11-b-110			2	4	15	
7	SET b,(IX+d)	$(IX+d)_h \leftarrow 1$	·	·	·	·	·	·	DD	CB	d	11-b-110	4	6	23	r Register 000 B 001 C 010 D 011 E 100 H 101 L 111 A
8	SET b,(IY+d)	$(IY+d)_h \leftarrow 1$	·	·	·	·	·	·	FD	CB	d	11-b-110	4	6	23	
9	RES b,r	$r_h \leftarrow 0$	·	·	·	·	·	·	CB	10-b--r-			2	2	8	
10	RES b,M	$M_h \leftarrow 0$	·	·	·	·	·	·	CB	10-b-110			2	4	15	
11	RES b,(IX+d)	$(IX+d)_h \leftarrow 0$	·	·	·	·	·	·	DD	CB	d	10-b-110	4	6	23	b Bit 000 0 001 1 010 2 011 3 100 4 101 5 110 6 111 7
12	RES b,(IY+d)	$(IY+d)_h \leftarrow 0$	·	·	·	·	·	·	FD	CB	d	10-b-110	4	6	23	
																M entspricht (HL)

Bitmanipulationsbefehle

Nr	Mnemo-nik	Symbolische Operation	Flags						Operationscode		Anzahl			Bemerkungen
			CY	Z	P/V	S	N	H	1.Byte	2.Byte	Byte	Zyklen	Takte	
1	NOP	keine Operation	.	.	.	.	.	.	00		1	1	4	Flag-Kennzeichen: . nicht beeinflußt
2	HALT	CPU geht in HALT-Zustand bis Interrupt	.	.	.	.	.	.	76		1	1	4	
3	DI	IFF ← 0	.	.	.	.	.	.	F3		1	1	4	maskierbarer Interrupt gesperrt
4	EI	IFF ← 1	.	.	.	.	.	.	FB		1	1	4	maskierbarer Interrupt freigegeben
5	IM0	Interrupt-modus 0	.	.	.	.	.	.	ED	46	2	2	8	Ausführung des Befehls, der bei Interruptbestätigung über D0 bis D7 angelegt wird; Einschaltzustand
6	IM1	Interrupt-modus 1	.	.	.	.	.	.	ED	56	2	2	8	Ausführung des Befehls RST 38H bei Interruptbestätigung
7	IM2	Interrupt-modus 2	.	.	.	.	.	.	ED	5E	2	2	8	Indirekter CALLnn bei Interruptbestätigung H(Zeiger) steht im Register I L(Zeiger) ist über D0 bis D7 anzulegen (D0=0!) n_L = (Zeiger) n_H = (Zeiger+1)

CPU - Steuerbefehle

Abkürzungsverzeichnis

CPU	-	Zentrale Recheneinheit	(central processor)
CR	-	Wagenrücklauf	(carriage return)
HT	-	Horizontaltabulator	
KRS	-	Kleinrechnersystem	
LF	-	Zeilenschaltung	(line feed)
MEOS	-	Betriebssystem des Robotron A 5601	
MRES	-	Mikrorechnerentwicklungssystem	
NL	-	Neue Zeile	(new line)
RAM	-	Schreib-Lese-Speicher	(random access memory)
PC	-	Befehlszähler	
SP	-	Kellerzeiger	(stackpointer)
SYPS	-	Systemprogrammiersprache	