Inhaltsverzeichnis

Lektion 2

1. Der erste Schritt in die Technik des Programmierens

Während die erste Lektion dieses Kurses überwiegend einer allgemeinen Einführung in die Mikroprozessor- und Computertechnik gewidmet war, werden wir uns in dieser Lektion mit den Grundlagen der Programmiersprachen, dem Befehlsvorrat der Mikroprozessoren 8080 und 8085 und dem Aufbau des micromaster beschäftigen.

1.1. Die Bedeutung von Programmiersprachen

Es wurde bereits einige Male das Wort "Programmiersprache" verwendet. Der Gebrauch des Wortes "Sprache" in diesem Zusammenhang ist deshalb geeignet, weil die einzelnen Elemente (Programmiervorschriften, -regeln) wie eine Fremdsprache vom Programmierer gelernt werden müssen, bevor er sie sinnvoll einsetzen kann.

Es gibt eine Vielzahl verschiedener Programmiersprachen, für die wir eine grobe Einteilung aufzeigen wollen. Wir haben in der ersten Lektion gelernt, daß im Speicher Befehle (Anweisungen) für die CPU (Zentraleinheit) nur in Form von Nullen und Einsen, bzw. high- und low-Pegeln abgespeichert werden können. Nur in dieser Form kann der Prozessor die Befehle und auch Daten verarbeiten. Jeder Befehl und jedes Datenwort wird in Kombinationen aus 8 high/low-Werten (8 Bit) codiert. Diese binäre Darstellung läßt sich auch wieder hexadezimal codiert darstellen. Also z.B.

Darstellung im Mikroprozessor	hexadezimale Darstellung	Funktion des Befehls
01110011	73	Transportiere Datenwort A von Ort x nach Ort y

Wenn wir uns das Blockschaltbild eines Mikroprozessorsystems in Erinnerung rufen, können die Orte x und y der Mikroprozessor selbst (eines seiner Register), eine Speicherzelle oder ein Peripheriegerät sein.

M

Ein 8-Bit breites Informationswort, welches einen Befehl oder Datenwort darstellt, wird als 1 Byte bezeichnet.

Da vom Mikroprozessor nur Befehle in der oben genannten Form verarbeitet werden können, bezeichnet man eine auf diese Weise aufbereitete Sprache als "Maschinensprache" (Maschinencode).
(Hier wird die CPU als "Maschine" bezeichnet, weil sie nach Vorgabe eines sinnvollen Programms selbsttätig "arbeiten" kann.) Die Maschinensprache ist praktisch die unterste Kommunikationsebene zwischen Mikroprozessor und dem Anwender.

Wie man sich leicht vorstellen kann, ist es äußerst umständlich und unübersichtlich, größere umfangreiche Programme auf dieser Ebene zu erstellen. Um hier eine erste Vereinfachung zu finden, hat man allen Befehlen eine Kombination aus mehreren Buchstaben zugeordnet, die häufig eine sinnvolle Abkürzung für die Funktion des Befehls darstellen. So z.B.

Abkürzung	abgeleitet aus dem englischen Wort	Funktion
MOV	move	Transportiere Datenwort A von Ort x nach Ort y

M Diese Abkürzungen für die Befehle werden als mnemonischer Code bezeichnet

Die gesamte Menge dieser Befehle wird als Assembler-Sprache bezeichnet. Sie ist maschinennah und maschinenspezifisch, d.h. jede Mikroprozessorfamilie hat ihre eigene Assembler-Sprache. Wir werden uns im nächsten Abschnitt mit der Assembler-Sprache des 8080, bzw. 8085 beschäftigen.

Neben dieser maschinennahen Programmierung gibt es eine ganze Reihe höherer Programmiersprachen. Diese haben die Eigenschaft, daß die Befehle mit Worten aus einer Umgangssprache (meist Englisch) gebildet sind, z.B. DO, FOR, NEXT, IF, GOTO, END, etc. Dadurch sind diese Sprachen leichter zu lernen und einzusetzen.

Die bekanntesten sind heute sicher Algol, Fortran, Cobol, Basic und Pascal. Weitere sind PL/1 und Varianten davon, C, Ada, Forth, usw. (Diese Sprachen

können hier nicht im einzelnen besprochen werden. Der interessierte Leser wird auf die Fachliteratur verwiesen. Siehe auch Zeitschrift "CHIP", Heft Juli 1983.) Diese höheren Sprachen ermöglichen eine komfortable Programmierung auch umfangreicher und komplizierter Problemstellungen. Dabei haben viele Sprachen anwendungsorientierte Verbreitung gefunden, z.B. Algol an technisch-wissenschaftlichen Instituten und Universitäten, Fortran in der Industrie und Cobol in der kaufmännischen Datenverarbeitung. Basic und Pascal wurden bekannt mit der raschen Verbreitung von Personal-Computern.

Jedes Programm, gleich in welcher Hochsprache es geschrieben wurde, muß vor seiner Einsatzfähigkeit im Computer in den Maschinencode übersetzt werden. Dieser Übersetzer wird als Compiler bezeichnet.

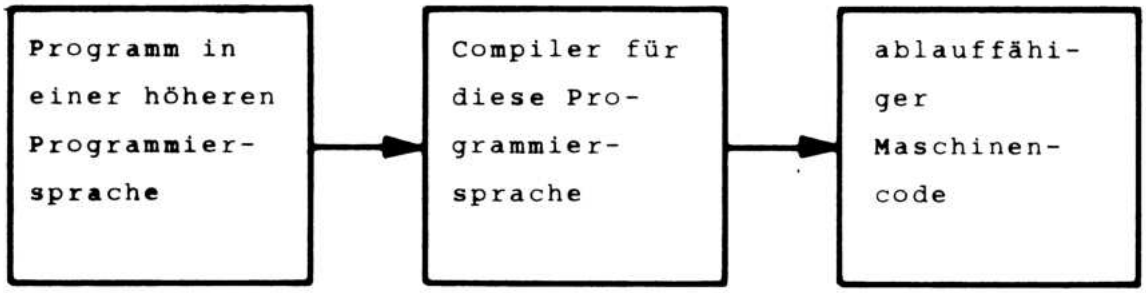


Bild 1

Ein in der Assembler-Sprache geschriebenes Programm muß mit Hilfe des Assemblers in den Maschinencode übersetzt werden. Assembler und Compiler sind Software-Hilfsmittel ohne die in der Computertechnik (fast) nichts geht. (Bei unseren Übungen werden wir das Assemblieren von Hand auf dem Papier durchführen, da im micromaster natürlich kein Assembler zur Verfügung gestellt werden kann.)

Mit der zitierten Verbreitung von Kleincomputern, die ja häufig als Personal-Computer bezeichnet werden, wurden nicht Compiler sondern "interpretative Übersetzer" eingesetzt. Der wesentliche Unterschied ist leicht zu erklären und zu verstehen. Der durch die Compilierung entstandene Maschinencode kann im Computer beliebig oft gestartet werden und ablaufen, ohne daß der Compiler weiterhin zur Verfügung steht. Das Gegenstück dazu ist die Umsetzung des Programms während des Programmablaufs, d.h. der

Interpreter nimmt eine Befehlszeile, erzeugt daraus (interpretiert) den Maschinencode und führt diesen dann aus. Daraus folgt, daß der Interpreter (im Gegensatz zum Compiler) ständig im Computer zur Verfügung stehen muß.

M

Der Compiler übersetzt eine Hochsprache in allein ablauffähigen Maschinencode

Der Assembler übersetzt den Mnemonic-Code der Assemblersprache in allein ablauffähigen Maschinencode

Der Interpreter erzeugt während des Programmablaufs zeilenweise die Befehle im Maschinencode

Ein vom Anwender geschriebenes Programm heißt in seiner ursprünglichen, noch nicht übersetzten Form Quellprogramm (engl. source program). Das vom Assembler oder Compiler übersetzte Programm ist dagegen das Objektprogramm, (engl. object program). Dieses Objektprogramm enthält den Maschinencode.

Neben diesem wesentlichen Unterscheidungsmerkmal zwischen Compilern und Interpretern wollen wir noch kurz Vor- und Nachteile von Programmiersprachen auf verschiedenen Ebenen zusammenfassen:

Programmiersprachen auf höherer Ebene:

Vorteile	Nachteile	Anwendungen
* Effektives und bequemes Schreiben von Programmen	* Sprachen sind anwendungsorientiert (wissenschaftlich, kaufmännisch, etc.)	* Bei Problemen, die lange Programme erfordern oder viele Daten verarbeiten (→ große Speicher)
* Leichte und übersichtliche Dokumentation	* Bei zeitkritischen Abläufen ungeeignet	* Wenn bereits für die spezielle Problemlösung Software-Bibliotheken bestehen
* Unabhängig von der Hardware (CPU) eines Computers	* Nicht Speicherbedarf optimiert	* Übertragbarkeit gegeben sein muß
* Verfügbarkeit von Software-Bibliotheken	* Schwieriger Zugriff auf spezielle Eigenschaften des Computers	* Wenn viel gerechnet wird und wenig Ein/Ausgabeoperationen vorkommen (Prozeß-unabhängige Datenverarbeitung)
* Quellprogramm übertragbar auf verschiedene Systeme	* Die umfangreichen Programmiervorschriften müssen wie eine Fremdsprache gelernt werden	

Programmiersprachen auf niederer Ebene:

Vorteile	Nachteile	Anwendungen
* Zeitoptimale Programmierung möglich	* Detaillierte Hardwarekenntnisse erforderlich (Computer und Peripherie)	* Echtzeitanwendungen bei Prozeßsteuerungen
* Direkter Zugriff auf Computer-Hardware	* Neben den Programmierkenntnissen müssen auch die Steueranweisungen des Assemblers beherrscht werden	* Kleiner bis mittlerer Programmumfang
* Leichte Bindemöglichkeit bereits vorhandener Routinen (Programmteile) aus einer Bibliothek zu einem neuen Programm	* Befehle in Form der Mnemonics schwierig zu merken	* Programme mit geringem Speicherbedarf
* Verfügbarkeit von Code-Wandlungsroutinen	* Nicht übertragbar auf andere Prozessorsysteme	* Programme mit geringem Rechenaufwand
* Direkte Steuerung von Ein/Ausgabe-Routinen		

1.2. Der Aufbau der Assembler-Sprache

Wir wollen an dieser Stelle noch einmal auf die Unterscheidung zwischen der Assembler-Sprache und dem **Assembler** (Übersetzer) hinweisen.
Man muß bei der Verwendung des Wortes Assembler genau auf den Sinn achten. Das vorher bereits verwendete Bild soll in etwas anderer Darstellung den Unterschied noch einmal verdeutlichen.

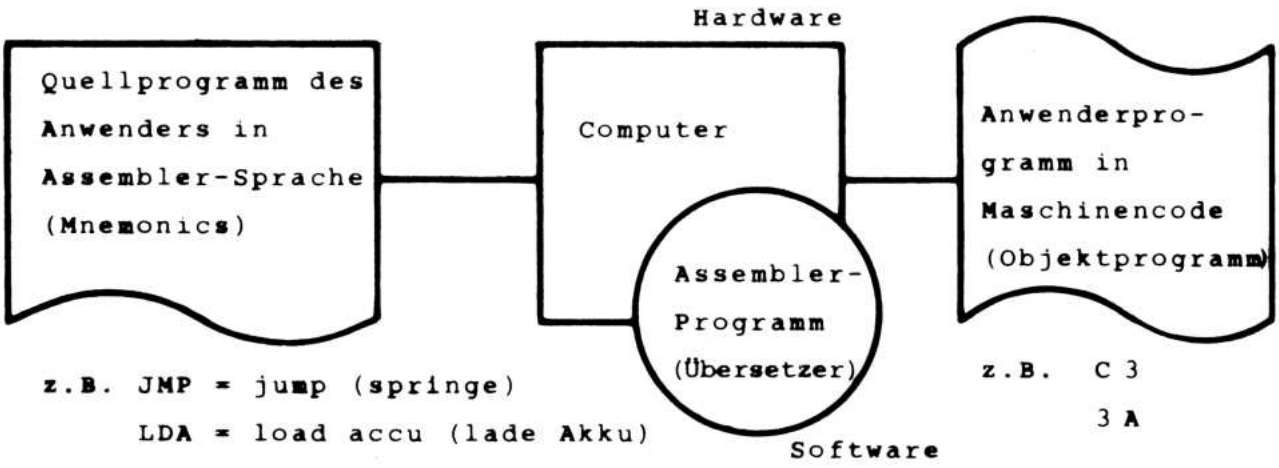


Bild 2

Bevor wir im nächsten Abschnitt den Befehlssatz des 8080/8085 behandeln, müssen wir wissen, in welcher Form diese Befehle beim Programmieren niedergeschrieben werden. Ein Quellprogramm besteht aus einer Folge von Assembler-Befehlen und kann dazwischen auch Anweisungen an den Assembler (Übersetzer) enthalten.
Diese Assembler-Anweisungen werden wir nicht alle im Detail behandeln, da wir wie bereits erwähnt, "von Hand" assemblieren werden. Sie sind zum Erlernen der Befehle und der Programmiertechnik nicht wesentlich. Da sie im eigentlichen Programm nicht wirksam werden, heißen sie Pseudo-Instruktionen.

Das Niederschreiben von Befehlen der Assembler-Sprache muß bestimmte Anforderungen erfüllen. Jede Befehlszeile setzt sich aus vier Feldern zusammen, die alle eine unterschiedliche Bedeutung haben und teilweise "Muß"- bzw. "Kann"-Felder sind. Die vier Felder sind:

M

- das Markenfeld (Label)
- das Mnemonicfeld (Operationscode)
- das Operandenfeld und
- das Kommentarfeld für Bemerkungen

Markenfeld	Mnemonicfeld (Op.-Code)	Operandenfeld	Kommentarfeld

Bild 3

Vollständige Befehlszeile

Bei der Behandlung dieser Felder werden einige formale Angaben mit erwähnt, die für den Assembler sehr wesentlich sind, für uns bei der Hand-Assemblierung aber keine große Bedeutung haben. (Solche Angaben beziehen sich auf den Assembler ASM 80, der von INTEL und SIEMENS für die Mikroprozessoren 8080 und 8085 verwendet wird.)
Es ist auch sinnvoll, sich an den geforderten Formalismus zu halten, da man dann nicht mit dem Lernen neu beginnen muß, wenn ein Computersystem angeschafft wird, auf dem ein Assembler ablaufen kann. Ein solcher Assembler ist z.B. auch in dem weit verbreiteten Betriebssystem CP/M enthalten.

M

Die formalen Anforderungen für die Schreibweise von Befehlen (Anweisungen) wird auch als Syntax bezeichnet.

Das Markenfeld

In dem ersten Feld einer Befehlszeile können Namen als Marken (engl. Label) eingetragen werden, die entweder eine Befehlszeile nur kennzeichnen oder, was viel häufiger vorkommt, als Sprungziel bei Programmver-

zweigungen verwendet werden. In diesem Feld kann ein Name eingetragen werden, muß aber nicht. Das Feld wird manchmal auch als Namensfeld bezeichnet. Weitere Merkmale dieses Feldes sind

M

- Operationscodes (Mnemonics), Pseudoinstruktionen und Namen von Registern dürfen nicht verwendet werden, da sie für den Assembler reserviert sind.

- Die Länge eines sonst frei gewählten Namens darf bis zu 6 Zeichen sein.

- Es können alphanumerische Zeichen verwendet werden, der Name muß aber mit einem Buchstaben (A-Z) oder Sonderzeichen (? , . α ...) beginnen

- Nach dem letzten Zeichen muß ein Doppelpunkt folgen.

Beispiele für richtige und falsche Namen:

richtig	falsch
START:	91:
?NULL:	MARKE
M1:	ADD:
	END:

K Aufgabe: Bei den falschen Beispielen sind ADD ein Operationscode und END eine Pseudoinstruktion, also bereits vergebene Namen. Begründen Sie, warum die beiden Beispiele davor falsch sind.

Zwei weitere Eigenschaften des Markenfeldes müssen wir uns noch merken. Die erstgenannte ist besonders wichtig.

M

- Namen dürfen niemals in einem Programm zwei Mal verwendet werden.

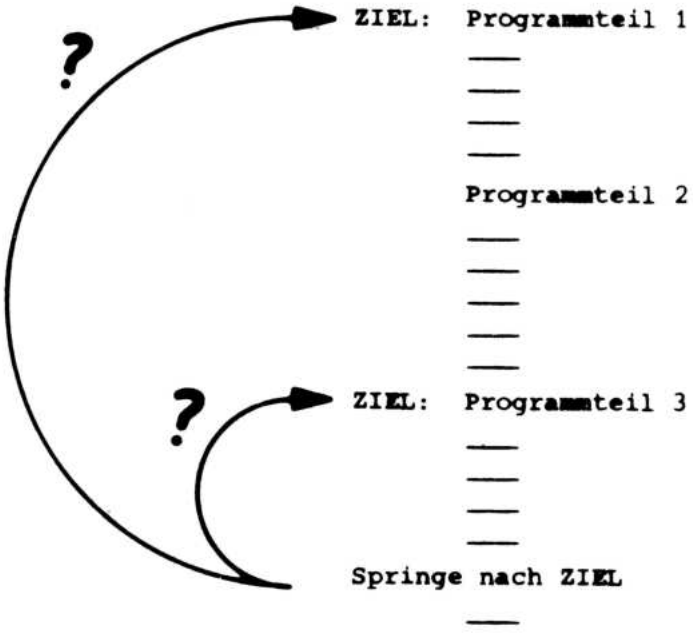
- Nach einem Namen muß nicht notwendiger Weise ein Befehl folgen. Der Name bezieht sich dann auf den Befehl in der nächsten Zeile, die auch zusätzlich einen Namen beinhalten darf.

Der Grund, warum Namen nicht mehrfach zulässig sind, ist einleuchtend.

M

Den Marken (Namen) werden beim Assemblieren Befehlsadressen zugeordnet.

Wird bei einer Programmverzweigung auf einen Namen gesprungen, der mehrfach vorkommt, kann die CPU während des Programmlaufs nicht entscheiden, auf welchen Programmteil verzweigt werden soll.



Ein Beispiel für die mehrfache Vergabe verschiedener Namen für einen Befehl kann so aussehen:

```
ZIEL 1:

ZIEL 2:   MOV ......
          .
          .
          .
          .
          Springe nach ZIEL 1
          .
          .
          .
          .
          Springe nach ZIEL 2
          .
          .
          .
          .
```

In diesem Fall erfolgt die weitere Programmausführung nach den beiden Sprunganweisungen immer mit demselben MOV-Befehl.

Das Mnemonic-Feld (Operationscode)

Beim Programmieren wird in dieses Feld der Assembler-Befehl in Form der bereits geschilderten mnemotechnischen Abkürzung eingetragen. Dieses ist der Operationscode, der der CPU eindeutig sagt, welche Operation auszuführen ist. Die vorher erwähnte Operation "Springe nach Ziel" wird mnemotechnisch mit JMP, abgeleitet von dem englischen Wort "jump", abgekürzt.

Um dieses Feld später richtig bedienen zu können, müssen wir uns noch ausführlich mit dem Befehlssatz auseinandersetzen. Das einzige Merkmal, das wir uns jetzt merken ist, daß

M

- das Operationscodefeld mit einem Zwischenraum-Zeichen, engl. blank, abgeschlossen wird.

Das Operandenfeld

Im vorhergehenden Feld wurde dem Mikroprozessor gesagt, was er machen soll. Jetzt müssen wir ihm noch mitteilen, womit er den Befehl ausführen soll, d.h. für die Operation müssen noch die Operanden bereitgestellt werden. Nicht jeder Befehl erfordert jedoch einen Operanden.

M

Im Operandenfeld sind je nach Befehlstyp keine, ein oder zwei Ausdrücke einzutragen.
Bei zwei Ausdrücken werden diese durch ein Komma getrennt.

Nun sind die Operanden noch zu spezifizieren. Vier verschiedene Informationstypen sind zu unterscheiden, die wir auf neun verschiedene Arten darstellen können. Der einzutragende Operand kann

1. die Kennzeichnung eines Registers sein
2. oder ein Register-Paar ansprechen
3. eine Konstante darstellen
4. oder eine 16 Bit-Speicheradresse sein.

Die Arten der Informationsdarstellung ergeben sich als

1. Hexadezimalcodierte Daten durch Anhängen von H
2. Dezimalcodierte Daten durch Anhängen von D
3. Oktalcodierte Daten durch Anhängen von Q
4. Binärcodierte Daten durch Anhängen von B
5. ASCII-Konstanten durch Einbinden in Hochkomma
6. Befehlszähler durch das Dollarzeichen $ dargestellt
7. Namen, die einer Befehlsadresse zugeordnet sind
8. Namen, die einem Wert zugeordnet sind
9. Ausdrücke

Einige Beispiele für 1-6 sind

Operationscode	Operand	Bedeutung
MVI	A, ØF1H	Lade Akku mit der Hex-Zahl F1
MVI	C, 75D	Lade C-Register mit der dezimalen Zahl 75
MVI	A, 43Q	Lade Akku mit der Oktal-Zahl 43
JMP	1100101100110101B	Springe zur Speicheradresse $CB35_{Hex}$
MVI	A, '#'	Lade Akku mit der ASCII-Darstellung des Sonderzeichens #
JMP	$+ØAH	Springe nach jetzigem Befehlszählerstand plus 10_{Dez}

Außerdem merken wir uns noch

- Beginnt eine Hex-Zahl mit einem Buchstaben, ist ihr eine Ø voranzustellen.

- Bei Dezimalzahlen kann das nachgestellte D entfallen.

Namen, wie unter 7 und 8 erwähnt, bedürfen folgender Erläuterungen:

- Namen (Marken), die im Markenfeld verwendet werden, werden auch als Operatoren eingetragen, z.B.

Name	Operations-code	Operand
ZIEL:	———	
	———	
	———	
	———	
	———	
	JMP	ZIEL

- Namen, denen bereits ein fester Wert zugewiesen ist, sind die Register- und Speicherbezeichnungen

Name	Reg.-Zuordnung	Zugeordnete Dezimalziffer
B	Register B	ø
C	Register C	1
D	Register D	2
E	Register E	3
H	Register H	4
L	Register L	5
M	Speicherzugriff	6
A	Akkumulator	7

Weiterhin können freigewählten Namen beliebige Werte zugewiesen werden. Das kann mit der Pseudoinstruktion EQU geschehen. Wird z.B. am Anfang des Programms

```
ZIEL    EQU    3ØØØD
```

geschrieben, so wird dem Namen ZIEL bei Verwendung im Programm beim Assemblieren immer die dezimale Zahl 3000 zugewiesen.

Manche Befehle, die 16-Bit-Operationen ausführen, sprechen jeweils Registerpaare an, also B und C, D und E sowie H und L.
Es genügt dann die Angabe des ersten Registers. Dazu gibt es noch zwei weitere 16-Bit-Speicherzellen, die wir später noch im Detail besprechen. Es ist zum einen das Byte, welches über den Prozessorstatus (PSW = Prozessorstatuswort) Auskunft gibt und die Bedingungsbits (Zustandsbits) enthält, zusammen mit dem Akkumulator, und zum anderen das sogenannte Stapelzeiger-Register, stack pointer, (SP). Als deutsche Übersetzung wird meist Stapelzeiger verwendet. Diese Speicherzelle enthält eine Speicheradresse. Fassen wir die 16-Bit-Operanden zusammen:

Operanden-Angabe	angesprochen wird Register-Paar
B	BC
D	DE
H	HL
PSW	Status-Register und Register A (Akku)
SP	stack pointer (Stapelzeiger)

Abschließend merken wir uns noch:

> Erfordert ein Operationscode zwei Operanden, die durch ein Komma getrennt werden, so steht vor dem Komma grundsätzlich das Ziel, z.B. bei Transferbefehlen.

Die 9. Möglichkeit, einen Operanden darzustellen, war das Hinschreiben von Ausdrücken. Damit sind arithmetische und logische Ausdrücke gemeint, die wiederum die Darstellungsmöglichkeiten 1. - 8. enthalten können. Da wir diese Form in diesem Kurs nicht einsetzen werden, der Gebrauch ist nur im Zusammenhang mit einem Assembler sinnvoll, verweisen wir auf die spezielle Literatur, z.B. die Bedienungsanleitung des zur Verfügung stehenden Assemblers.

Das Kommentarfeld

Das Kommentarfeld, welches für beliebige Bemerkungen und Notizen vom Programmierer verwendet werden kann, ist eine äußerst sinnvolle Einrichtung. Viele Randbemerkungen und Erklärungen machen ein Programm verständlich und durchschaubar (transparent), sei es wenn man nach langer Zeit das Programm noch einmal lesen muß, um z.B. etwas zu ändern, oder falls jemand, der nicht das Programm geschrieben hat, sich damit befassen und einarbeiten muß.

M Ein Programm wird um so besser, je mehr erklärende Bemerkungen im Kommentarfeld eingetragen werden.

Nutzen Sie diese Möglichkeit besonders am Anfang, wenn Ihnen nicht gleich jede Funktion eines Befehls geläufig ist. Für dieses Feld gilt nur eine Regel

M Das Kommentarfeld muß mit einem Semikolon (Strichpunkt) beginnen.

Das Kommentarfeld kann an beliebiger Stelle und auch am Anfang einer Programmzeile stehen. Will man z.B. zwecks Übersichtlichkeit Leerzeilen schaffen, steht das Semikolon am Anfang einer Zeile ohne folgende Bemerkung. Gültige Zeilen sind

```
        MVI   A, ØFFH    ;Lade Akku mit Kon-
                         ;stante FF
        ;
        ;
ZIEL:   ;Schleifenanfang
        MOV   B,A        ;Lade Register B mit
                         ;den Inhalt des Akkus
```

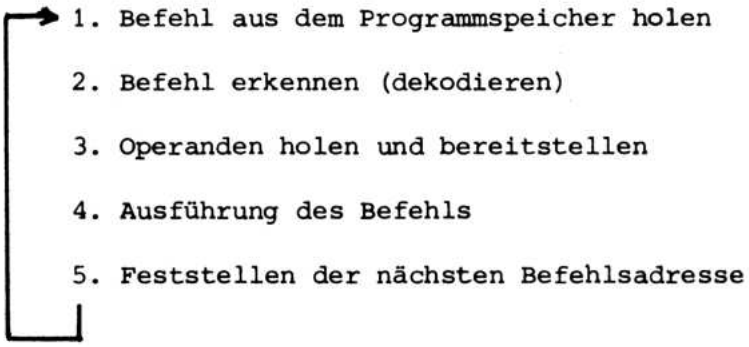
Damit haben wir nun den Aufbau einer Befehlszeile beschrieben. Die genannten Felder werden wir in dem für unsere Programmieraufgaben vorbereiteten Vordruck wiederfinden. Einen solchen Mustervordruck finden Sie im Anhang. Wenn Sie sich diesen für die nächste Lektion einige Male kopieren, können Sie die Blätter als Arbeitsvorlage verwenden.

2. Der Befehlssatz der Mikroprozessorfamilie 8080/8085

Auch in diesem Abschnitt wollen wir wie bisher einige Grundlagen aufbereiten, die für das Verständnis der Assembler-Befehle erforderlich sind. Dabei beginnen wir mit bereits bekanntem und bauen dann schrittweise darauf auf.

2.1. Ablauf einer Befehlsausführung im Mikroprozessor

Das eben erwähnte "bekannte" ist für uns der in der ersten Lektion beschriebene von-Neumann-Zyklus als Definition eines Computers:



Da wir jetzt bereits einiges über die Hardware und das "Drumherum" des Mikroprozessors wissen, können wir uns auch vorstellen, daß die Beschreibung durch den von-Neumann-Zyklus nur grob sein kann. Wir wollen uns jetzt die einzelnen Schritte etwas genauer ansehen.

Zu 1: Befehl holen

Ein Befehl ist als Operationscode im Speicher abgelegt.
Um diesen Code aus dem Speicher auslesen zu können,

- muß erst die Speicherzelle adressiert werden
- müssen Steuersignale, z.B. Speicher lesen, erzeugt werden
- sind Steuersignale zu prüfen, z.B. Hold, Ready
- Befehlscode aus dem Speicher in den Mikroprozessor transportieren

Zu 2: Befehl erkennen

Der in den Prozessor eingelesene Befehlscode muß erkannt werden, da sich daraus die weiteren Schritte ableiten. Dieses Erkennen erfolgt intern im Prozessor durch die Dekodierlogik, wobei extern keinerlei Maßnahmen erforderlich werden. Je nach Befehlstyp gibt es

1-Byte-Befehle
2-Byte-Befehle
3-Byte-Befehle

Zu 3: Operanden in den Mikroprozessor holen:

Nach dem Erkennen des Befehls muß bei einem Mehr-Byte-Befehl der Operand in den Mikroprozessor eingelesen werden. Dieser steht im Speicher nach dem Befehlscode und wird genau so wie ein Operationscode eingelesen.

- Adressierung der Speicherstelle
- Steuersignale erzeugen und prüfen
- Operand aus der Speicherzelle in den Mikroprozessor transportieren
- Wiederholung dieser drei Vorgänge, falls es sich um einen 3-Byte-Befehl handelt, z.B. wenn der Operand eine Adresse ist.

Zu 4: Ausführung des Befehls

Je nach Art des erkannten Befehls wird dieser jetzt ausgeführt, z.B. bei einem Ein-/Ausgabebefehl werden Daten mit der Peripherie ausgetauscht, oder es wird eine Rechenoperation durchgeführt, deren Ergebnis wieder im Akkumulator steht.

Zu 5: Feststellen der nächsten Befehlsadresse

Wenn der zuletzt abgearbeitete Befehl keine "Programmende"-Information enthielt, wird eine neue Befehlsadresse erzeugt und der neue Befehl geholt. Die neue Befehlsadresse ergibt sich durch Inkrementieren (Erhöhung um 1) des Programmzählers oder durch Laden des Programmzählers mit einer neuen

Adresse, z.B. bei einem Sprungbefehl.

Dieser Ablauf im Mikroprozessor setzt sich solange fort, bis er durch eine Hardware-Unterbrechung oder ein Software-Ende in den Halt-Zustand gebracht wird. Ein einzelner Ablauf wird auch als Befehls-Zyklus, engl. instruction cycle, bezeichnet. Unterteilt man den von-Neumann-Zyklus in zwei Phasen (1. - 3.) und (4. - 5.), so kann man diese überschreiben mit

Hol-Phase (engl. fetch-cycle) und
Ausführungs-Phase (engl. execution cycle)

Diese beiden Zyklen werden auch allgemein als Maschinenzyklen bezeichnet

Wer sich in der Digitaltechnik etwas auskennt, weiß, daß für den Betrieb von Schaltwerken ein Takt erforderlich ist. Auch der Mikroprozessor arbeitet taktgesteuert. Für die Abarbeitung der einzelnen Befehlsphasen im Prozessor sind je nach Befehlstyp eine unterschiedliche Anzahl von Taktzyklen erforderlich. Man kann sich das auch so vorstellen, daß jeder Befehl aus einer Anzahl Mikro-Befehlen besteht, die der Hersteller für eine bestimmte Funktion eingebaut hat.

Für zeitkritische Aufgaben, die mit einem Mikroprozessor gelöst werden müssen, kann man die Ablaufzeit von Befehlen errechnen.

Befehlsdauer = Zahl der Taktzyklen x Zeit eines Taktes

Die Zahl der Taktzyklen findet man in der Befehlsliste des Mikroprozessors und die Zeitdauer eines Taktes ergibt sich aus der Frequenz, mit der der Mikroprozessor betrieben wird (meist kann man sie der Gerätebeschreibung entnehmen).

2.2. Beschreibung des Befehlsaufbaus

Wir wissen bereits, daß ein Befehl je nach Art aus einem, zwei oder drei Byte bestehen kann. Das erste Byte stellt dabei immer den Operationscode dar. Im Mikroprozessor ist das eine Folge von 8 Nullen, bzw. Einsen entsprechend den logischen Spannungspegeln H und L.

In der Beschreibung der Befehle werden wir auch die Binär-Codedarstellung der Befehle finden. Da ein gleicher Assembler-Befehl beispielsweise alle Register ansprechen kann, muß in diesem Binärcode auch die Kennzeichnung des verwendeten Registers enthalten sein. So bedeutet z.B.

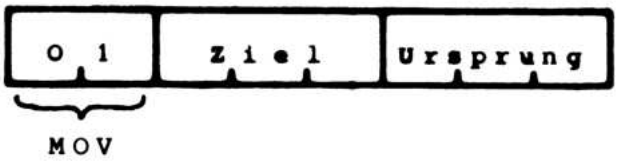
MOV B,M	Bringe Dateninhalt vom Speicher in Register B
	oder
MOV D,M	Bringe Dateninhalt vom Speicher in Register D

Beide Befehle sind identisch bis auf die Angabe des Zielregisters. Der binäre Operationscode für den erwähnten MOV-Befehl ist

0 1 X X X 1 1 0

wobei XXX für die binäre Kennung des gewünschten Registers steht. (Sie finden diese Kennung durch Übersetzen der dezimalen Registerziffern auf Seite 15.) Wir merken uns:

> Der Operationscode wird im Mikroprozessor durch eine Folge von 8 Nullen und Einsen dargestellt, die einen Befehl eindeutig kennzeichnen und variable Blöcke enthalten können, in denen Ziel und Ursprung angegeben werden, z.B. Format des MOV-Befehls



Bei der Beschreibung der Befehle werden außerdem folgende Abkürzungen verwendet:

Symbolische Abkürzung	Erklärung
accumulator	Akkumulator (Register A)
adr	Adresse, 16-Bit
data	Datenwort, 8-Bit
data 16	16-Bit langes Datenwort
port	Adresse eines Ein-/Ausgabe-kanals, 8 Bit
r,r1,r2	Abkürzung für eines der Register A,B,C,D,E,H,L
ddd	Bitmuster des Binärcodes, welches eines der Register kennzeichnet (d von engl. destination, deutsch: Bestimmung, Ziel)
sss	Bitmuster des Binär-Codes, welches eines der Register kennzeichnet (s von engl. source, dtsch.: Quelle, Ursprung).

DDD oder SSS	dezimales Äquivalent	Register-bezeichnung
111	7	A
000	0	B
001	1	C
010	2	D
011	3	E
100	4	H
101	5	L

(Die 6 wird für Speicherzugriff verwendet,
110 ≙ 6 = M (memory)

Symbolische Abkürzung	Erklärung
konst	Konstante, 8-Bit-Wert
M	memory (Speicherbereich)
nr	Nummer
rp	Abkürzung für eines der Registerpaare BC, DE, HL und Register für Stapelzeiger (SP). Der erste Buchstabe kennzeichnet immer das höherwertige Byte
rr	Bezeichnung des Binär-Codes, welches eines der Registerpaare (register-register) kennzeichnet rr — Registerpaar-Bezeichnung 00 — BC 01 — DE 10 — HL 11 — SP (stackpointer) Diese Binär-Codes werden gebildet durch Weglassen der niederwertigsten Stelle der Einzelregister-Bezeichnung
PC	Programmbefehlszähler-Register (programm counter)
PSW	Prozessorstatuswort, enthält Zustandskennzeichenbits Z zero (Null) S signum (Vorzeichen) P Parity (Parität) CY carry (Übertrag) AC auxiliary carry (Hilfsübertrag)

K Aufgabe: Wie muß die vollständige Assemblerschreibweise für einen Befehl lauten, wenn im Prozessor der Binär-Code 01 000 010 steht?

2.3. Unterscheidung verschiedener Befehlstypen

Bevor wir uns mit dem Befehlssatz im Detail beschäftigen, wollen wir mehrere Gruppen von Befehlen unterscheiden. Uns ist bereits bekannt, daß der Transport von Daten in einem Mikroprozessorsystem erforderlich ist, und daß in der ALU arithmetische und logische Operationen ausgeführt werden können. Demzufolge gibt es Befehle

- die den Datentransport bewirken (Transferbefehle)
- die arithmetische Operationen ausführen
- die logische Operationen ausführen

Weitere Befehlsgruppen sind

- Registeranweisungen
- Sprungbefehle
- Befehle zur Unterprogrammbehandlung
- Befehle zur externen Programmunterbrechung
- Sonstige Befehle
- Spezielle 8085-Befehle
- Pseudo-Befehle an den Assembler

Da sich Daten in einem Mikroprozessorsystem an den verschiedenen Stellen (Register, Speicher und Peripherie) befinden können, gliedern sich die Transferbefehle weiter auf

Ursprung	Ziel
a. Register	Register
b. Speicher, Peripherie	Register
c. Konstante	Registerpaar
d. Register	Speicher, Peripherie
e. Konstante	Speicher, Register

Bei den arithmetischen Operationsbefehlen unterscheiden wir

a. Inkrement- und Dekrement-Befehle
b. Additions-Befehle
c. Subtraktions-Befehle

Inkrementieren bedeutet, daß zu einem Wert eine 1 hinzuaddiert wird. Beim Dekrementieren wird ein Wert um 1 verringert. Typische Anwendungsfälle sind, wie wir später sehen werden, sogenannte Zählschleifen.

Die logischen Operationen unterscheiden sich durch den logischen Funktionen, so wie wir sie in der ersten Lektion besprochen haben.

a. Komplementieren (Negation)
b. UND-Verknüpfungen (AND)
c. ODER-Verknüpfungen (OR)
d. Exklusiv-ODER-Verknüpfungen (EX-OR)
e. Logische Vergleiche

Die Registeranweisungen beziehen sich nur auf den Akku und das PSW-Byte

a. Schiebebefehle für Akku
b. Carrybit-(Übertragungsbit-)Anweisungen

Bei den Sprungbefehlen gibt es

a. Unbedingte Sprünge
b. Bedingte Sprünge

Zur Behandlung von Unterprogrammen sind

a. Unterprogrammaufrufe und
b. Rücksprungbefehle

erforderlich.

Programmunterbrechungsbefehle beziehen sich auf die Ein- und Ausschaltmöglichkeit von externen Unterbrechungsanforderungen (engl. interrupts) an den Prozessor.
Beim 8085 gibt es im Gegensatz zum 8080 aufgrund der geringfügig anderen Hardware-Struktur zwei spezielle zusätzliche Befehle. Die Pseudo-Befehle (Pseudo-Instruktionen) sind gleich.

Der Befehlssatz wird im nächsten Abschnitt vollständig erklärt, auch wenn einige Themen, wie zum Beispiel Unterprogrammbehandlung, noch nicht besprochen wurden. Es würde nicht der Übersichtlichkeit dienen, wenn der Befehlssatz über mehrere Lektionen verteilt wäre.

2.4. Beschreibung aller Befehle der Mikroprozessoren 8080/8085

Bei der Beschreibung des Befehlsvorrates der Mikroprozessoren 8080 und 8085 wird jeder Befehl folgendermaßen beschrieben

- Assembler-Befehl, dargestellt durch den mnemonischen Code
- Binär-Code
- Beschreibung der Funktion des Befehls
- veränderte Zustandsbits
- Anzahl der Bytes
- Zahl der Taktzyklen für 8080/8085
  (alle Angaben nach dem Schrägstrich beziehen sich, wie hier angegeben, auf den 8085)
- evtl. Anmerkung und Beispiel

Die Befehle werden nicht in alphabetischer, sondern in Zusammenfassung ihrer Funktion beschrieben. Der 8085-Mikroprozessor unterscheidet sich vom 8080 bezüglich der Software lediglich dadurch, daß er zwei zusätzliche Befehle, den RIM und SIM, besitzt. Ansonsten sind die Befehlssätze identisch. (Der Programmierer muß jedoch auch die Unterschiede bei den Unterbrechungssprungadressen beachten → Datenbuch.)

Hardware-mäßig weißt der 8085 gegenüber dem 8080 folgende Merkmale auf:

- Ausführungszeit kann bis zu 50 % höher sein
- Steuerfunktionen sind integriert
- Nur eine 5V-Versorgungsspannung
- Nicht maskierbare Unterbrechungsmöglichkeit
- Drei separate maskierbare Unterbrechungsmöglichkeiten
- Serielle Ein-/Ausgabe-Kanäle

2.4.1 Beschreibung der Transferbefehle

A. Register nach Register

MOV r1,r2 — 0 1 d d d s s s

Funktion: Lade Register r1 mit dem Inhalt von Register r2.
Diese Register können die Register A, B, C, D, E, H oder L sein

Veränderte Zustandsbits: Keine

Bytes: 1

Taktzyklen: 5/4

XCHG — 1 1 1 0 1 0 1 1

Funktion: Vertauschen des Inhaltes von Registerpaar (D,E) mit Inhalt des Registerpaares (H,L)

Veränderte Zustandsbits: Keine

Bytes: 1

Taktzyklen: 4/4

Beispiel: Inhalt der Registerpaare DE = BA12H und HL = 34EFH
Nach Ausführen des Befehls XCHG ergbit sich DE = 34EFH
und HL = BA12H

XTHL — 1 1 1 0 0 0 1 1

Funktion: Vertauschen des Inhaltes von Registerpaar (HL) mit Inhalt der Speicherzellen, die durch das Stapelzeiger-Register (stack-pointer) adressiert werden

Veränderte Zustandsbits: Keine

Bytes: 1

Taktzyklen: 18/16

Beispiel: Vor der Ausführung des Befehls XTHL seien folgende Inhalte

vorhanden:

Speicher-adresse	Speicher-inhalt	Stackpointer-inhalt	Register-paar H,L
20BCH	0AH	20BCH	70EFH
20BDH	15H		

Nach Ausführung des Befehls:

Speicher-adresse	Speicher-inhalt	Stackpointer	Register-paar H,L
20BCH	EFH	20BCH	150AH
20BDH	70H		

SPHL 1 1 1 1 1 0 0 1

Funktion: Lade das Stapelzeiger-Register mit dem Inhalt des Registerpaares (H,L)

Veränderte Zustandsbits: Keine

Bytes: 1

Taktzyklen: 5/6

Beispiel: Das Registerpaar (H,L) enthalte F043H, nach Ausführung des Befehls SPHL enthält das Stapelzeiger-Register ebenfalls F043H.

B. Speicher und Peripherie nach Register

MOV r1, M 0 1 d d d 1 1 0

Funktion: Lade Register r1 mit dem Inhalt der Speicherstelle, die durch den Inhalt des Registerpaares (H,L) adressiert wird. Register r1 kann das Register A, B, C, D, E, H oder L sein.

Veränderte Zustandsbits: Keine

Bytes: 1

Taktzyklen: 7/7

Beispiel: Vor Ausführung des Befehls sind folgende Inhalte vorhanden

Register r1	Registerpaar H,L	Speicherinhalt unter Adresse 2000H
OOH	2000H	FFH

Nach Ausführung des Befehls

Register r1	Registerpaar H,L	Speicherinhalt unter Adresse 2000 H
FFH	2000H	FFH

LDA adr 0 0 1 1 1 0 1 0

Funktion: Der Akkumulator wird mit dem Byte geladen, welches im Speicher unter der angegebenen Adresse steht

Veränderte Zustandsbits: Keine

Bytes: 3

Taktzyklen: 13/13

Beispiel: Inhalte vor Ausführung des Befehls

Akku-Inhalt	im Befehl angegebene Adresse	Speicherinhalt unter Adresse 20BCH
OOH	20BCH	1AH

Nach Ausführung des Befehls

Akku-Inhalt
1AH

LDAX rp O O r r 1 O 1 O

Funktion: Der Akkumulator wird mit dem Inhalt der Speicherzelle geladen, die durch den Inhalt des Registerpaares rp adressiert wird. rp kann das Registerpaar (B,C) oder (D,E) sein

Veränderte Zustandsbits: Keine

Bytes: 1

Taktzyklen: 7/7

Beispiel: Inhalte vor Ausführung des Befehls LDAX D.

Akku-Inhalt	Inhalt Registerpaar D,E	Speicherinhalt unter Adresse 100AH
11H	100AH	88H

Nach Ausführung des Befehls

Akku-Inhalt
88H

LHLD adr. 0 0 1 0 1 0 1 0

Funktion: Es wird das Registerpaar (H,L) mit den Bytes geladen, die unter der angegebenen Adresse sowie der angegebenen Adresse + 1 stehen.

Veränderte Zustandsbits: Keine

Bytes: 3

Taktzyklen: 16/16

Beispiel: Vor Ausführung des Befehls seien folgende Inhalte vorhanden

Registerpaar H,L	angegebene adr und (adr+1)	Inhalte unter den Adressen 4000H und 4001H
1234H	4000H	1AH
	4001H	2BH

Inhalt von Registerpaar (H,L) nach Ausführung 2B1AH.

POP rp 1 1 r r 0 0 0 1
oder PSW 1 1 1 1 0 0 0 1

Funktion: Das angegebene Registerpaar wird mit den beiden Bytes geladen, die durch den Inhalt und Inhalt+1 des Stapelzeiger-Registers adressiert werden. Mögliche Registerpaare (B,C), (D,E), (H,L) oder PSW

Veränderte Zustandsbits: Keine bei POP rP
Z, S, P, CY, AC bei POP PSW

Bytes: 1

Taktzyklen: 10/10

Beispiel: POP B. Inhalte vor Ausführung des Befehls:

Inhalt Register-paar B,C	Stackpointer	Inhalte unter den Adressen FFOOH und FFO1H
OOOOH	FFOOH	OAH
	FFO1H	7OH

Inhalt des Registerpaares (B,C) nach Ausführung des Befehls 7OOAH.

IN nr 1 1 0 1 1 0 1 1

Funktion: Der Akkumulator wird mit dem Datenwort geladen, welches über den Eingabekanal mit der angegebenen Nummer eingelesen wird (nr ≤ 255)

Veränderte Zustandsbits: Keine

Bytes: 2

Taktzyklen: 10/10

C. Konstante nach Registerpaar

LXI rp,adr 0 0 r r 0 0 0 1

Funktion: Lade das angegebene Registerpaar rp mit dem Wert der angegebenen Adresse adr. Zulässige Registerpaare (B,C), (D,E), (H,L) und SP.

Veränderte Zustandsbits: Keine

Bytes: 3

Taktzyklen: 10/10

Beispiel: LXI H,40FFH. Die Konstante, bzw. der Adresswert 40FFH wird in das H,L-Registerpaar geladen.
LXI SP,FFOOH. Die Adresse FFOOH wird in das Stapelzeiger-Register geladen.

D. Register nach Speicher oder Peripherie

MOV M,r1 0 1 1 1 0 s s s

Funktion: Abspeichern des Inhaltes von Register r1 auf dem Speicherplatz, der durch den Inhalt des Registerpaares (H,L) adressiert ist. r1 kann sein A, B, C, D, E, H oder L.

Veränderte Zustandsbits: Keine

Bytes: 1

Taktzyklen: 7/7

Beispiel: Folgende Inhalte seien vor Ausführung des Befehls vorhanden

Speicherinhalt unter Adresse 3000H	Inhalt Register-paar (H,L)	Inhalt Register r1
1CH	3000H	7FH

Nach Ausführung des Befehls steht im Speicher unter der Adresse 3000H der Inhalt 7FH.

STA adr 0 0 1 1 0 0 1 0

Funktion: Der Akkumulator-Inhalt wird unter der angegebenen Adresse abgespeichert

Veränderte Zustandsbits: Keine

Bytes: 3

Taktzyklen: 13/13

Beispiel: Inhalte vor Ausführung des Befehls

Speicherinhalt unter Adresse 40 ABH	in Befehl angegebene Adresse	Akku-Inhalt
00H	40ABH	FFH

Nach Ausführung des Befehls steht an der Speicherstelle 40ABH der Inhalt FFH.

STAX rp 0 0 r r 0 0 1 0

Funktion: Der Akkumulator-Inhalt wird in der Speicherstelle abgelegt, die durch den Inhalt des Registerpaares rp adressiert wird. rp kann das Registerpaar (B,C) oder (D,E) sein

Veränderte Zustandsbits: Keine

Bytes: 1

Taktzyklen 7/7

Beispiel: Inhalte vor Ausführung des Befehls STAX B.

Inhalt an der Adresse 4711H	Inhalt Register-paar B,C	Akku-Inhalt
2FH	4711H	34H

Nach Ausführung des Befehls steht unter der Adresse 4711H der neue Inhalt 34H.

SHLD adr 0 0 1 0 0 0 1 0

Funktion: Der Inhalt des Registerpaares (H,L) wird auf den Speicherzellen abgelegt, die durch die angegebene Adresse adr sowie (adr+1) bestimmt werden.

Veränderte Zustandsbits: Keine

Bytes: 3

Taktzyklen: 16/16

Beispiel: Vor Ausführung des Befehls seien folgende Inhalte vorhanden

Inhalte unter den Adressen 3000H und 3001H	angegebene adr und (adr+1)	Registerpaar H,L
2EH 44H	3000H 3001H	103AH

Nach Ausführung steht an der Speicherstelle 3000H der Inhalt 3AH und bei 3001 der Inhalt 10H.

PUSH rp 1 1 r r 0 1 0 1

Funktion: Der Inhalt des angegebenen Registerpaares rp wird in die Speicherzellen geladen, die durch den Inhalt und Inhalt+1 des Stapelzeiger-Registers adressiert werden. Mögliche Registerpaare sind (B,C), (D,E), (H,L) oder PSW.

Veränderte Zustandsbits: Keine

Bytes: 1

Taktzyklen: 11/12

Beispiel: PUSH D. Inhalte vor Ausführung des Befehls:

Inhalt unter den Adressen FFOEH und FFOFH	Stackpointer	Inhalt Registerpaar D,E
CDH	FFOEH	20H
30H	FFOFH	OOH

Nach Ausführung des Befehls steht unter der Adresse FFOEH der Inhalt OOH und unter FFOFH der Inhalt 20H.

OUT nr 1 1 0 1 0 0 1 1

Funktion: Der Akkumulator-Inhalt wird auf dem Ausgabekanal mit der angegebenen Nummer ausgegeben (nr ≤ 255)

Veränderte Zustandsbits: Keine

Bytes: 2

Taktzyklen: 10/10

E. Konstante nach Register oder Speicher

MVI M,konst 0 0 1 1 0 1 1 0

Funktion: Der Speicherplatz, der durch das (H,L)-Registerpaar adressiert ist, wird mit der angegebenen Konstanten (konst = Wert zwischen 0 und 255 dezimal) geladen

Veränderte Zustandsbits: Keine

Bytes: 2

Taktzyklen: 10/10

Beispiel: MVI M,15D. Vorausführung des Befehls:

Speicherinhalt bei Adresse 3000H	Inhalt Register-paar H,L
8AH	3000H

Nach Ausführung des Befehls steht bei der Speicheradresse 3000H der Inhalt 0FH (≙ 15D).

MVI r1,konst 0 0 d d d 1 1 0

Funktion: Das angegebene Register r1 wird mit der Konstanten (konst = Wert zwischen 0 und 255 dezimal) geladen. Mögliche Register für r1 sind A, B, C, D, E, H oder L.

Veränderte Zustandsbits: Keine

Bytes: 2

Taktzyklen: 7/7

Beispiel: MVI A,FFH. Enthält der Akku z.B. vor Ausführung des Befehls 11H, so ist der Akku-Inhalt nach Ausführung FFH.

2.4.2. Arithmetische Operationen

INR r1 — 0 0 d d d 1 0 0

Funktion: Zum Inhalt des angegebenen Register r1 wird der Wert 1 addiert. r1 kann das Register A, B, C, D, E, H oder L sein.

Veränderte Zustandsbits: Z, S, P, AC

Bytes: 1

Taktzyklen: 5/4

Beispiel: Vor Ausführung des Befehls enthält der Akku 09H, danach 0AH.

INR M — 0 0 1 1 0 1 0 0

Funktion: Zum Inhalt der durch das Registerpaar (H,L) adressierten Speicherstelle wird der Wert 1 addiert.

Veränderte Zustandsbits: Z, S, P, AC

Bytes: 1

Taktzyklen: 10/10

Beispiel: Vor Ausführung des Befehls seien

Inhalt der Speicherzelle zelle mit Adresse 4000H	Inhalt Register- paar H,L
7AH	4000H

Nach Ausführung des Befehls steht unter der Adresse 4000H der Wert 7BH.

DCR r1 — 0 0 d d d 1 0 1

Funktion: Vom Inhalt des angegebenen Registers r1 wird der Wert 1 abgezogen

Veränderte Zustandsbits: Z, S, P, AC

Bytes: 1

Taktzyklen: 5/4

Beispiel: Vor Ausführung des Befehls enthält der Akku 09H, danach 08H, wenn der Befehl DCR A lautet.

DCR M 0 0 1 1 0 1 0 1

Funktion: Vom Inhalt der durch das Registerpaar (H,L) adressierten Speicherstelle wird der Wert 1 abgezogen.

Veränderte Zustandsbits: Z, S, P, AC

Bytes: 1

Taktzyklen: 10/10

Beispiel: Vor Ausführung des Befehls seien

Inhalt der Speicherstelle mit Adresse 20BCH	Inhalt Register-paar H,L
3CH	20BCH

Nach Ausführung des Befehls steht in der Speicherstelle mit der Adresse 20BCH der Wert 3BH.

INX rp 0 0 r r 0 0 1 1

Funktion: Der Inhalt des angegebenen Registerpaares rp wird um den Wert 1 erhöht. rp kann das Registerpaar (B,C), (D,E), (H,L) oder SP sein.

Veränderte Zustandsbits: Keine

Bytes: 1

Taktzyklen: 5/6

Beispiel: Wenn das Stapelzeiger-Register SP vor Ausführung des Befehls den Wert 3FFEH enthält, ist der Wert danach 3FFFH (INX H).

DCX rp 0 0 r r 1 0 1 1

Funktion: Der Inhalt des angegebenen Registerpaares rp wird um den Wert 1 vermindert. rp kann das Registerpaar (B,C), (D,E), (H,L) oder SP sein.

Veränderte Zustandsbits: Keine

Bytes: 1

Taktzyklen: 5/6

Beispiel: Wenn das (H,L)-Registerpaar vor der Ausführung des Befehls den Wert 3FFEH enthält, ist der Wert danach 3FFDH (DCX H).

ADD r1 1 0 0 0 0 s s s

Funktion: Der Inhalt vom angegebenen Register r1 wird zum Inhalt des Akkumulators addiert. r1 = A, B, C, D, E, H oder L.

Veränderte Zustandsbits: Z, S, P, CY, AC

Bytes: 1

Taktzyklen: 4/4

Anmerkung: Bei der Addition bleibt das Carry-Bit unberücksichtigt. Es wird jedoch gesetzt, wenn ein Überlauf erfolgt.

ADD M 1 0 0 0 0 1 1 0

Funktion: Es wird der Inhalt der Speicherstelle zum Inhalt des Akkumulators addiert, die durch das (H,L)-Registerpaar adressiert wird.

Veränderte Zustandsbits: Z, S, P, CY, AC

Bytes: 1

Taktzyklen: 7/7

Anmerkung: Wie bei ADD r1

Beispiel: (für beide ADD-Befehle):
Der Akku-Inhalt sei 2FH. Das aus einer Speicherzelle oder einem Register zu addierende Byte sei 6BH.

	0010 1111	(2F)
+	0110 1011	(6B)
	1001 1010	(9A)

Der neue Akku-Inhalt ist 9AH. Es wurden folgende Zustandsbits gesetzt

Z = 0 S = 1 P = 1 CY = 0 AC = 1

ADC r1 1 0 0 0 1 s s s

Funktion: Der Inhalt vom angegebenen Register r1 und der Zustand des Carry-Bits werden zum Inhalt des Akkumulators addiert.
r1 = A, B, C, D, E, H oder L.

Veränderte Zustandsbits: Z, S, P, CY, AC

Bytes: 1

Taktzyklen: 4/4

Anmerkung: Das Carry-Bit wird mit addiert und evtl. auch verändert. Anwendung bei Addition von Zahlen über mehrere Bytes.

ADC M 1 0 0 0 1 1 1 0

Funktion: Es wird das Carry-Bit und der Inhalt der Speicherzelle, die durch das (H,L)-Registerpaar adressiert wird, zum Inhalt des Akkumulators addiert.

Veränderte Zustandsbits: Z, S, P, CY, AC

Bytes: 1

Taktzyklen: 7/7

Anmerkung: Wie bei ADC r1

Beispiel: Folgende Addition werde ausgeführt

Akku		1011	1101		1011	1101	(BD)
Register- oder Speicherbyte	+	0100	0000	+	0100	0000	(40)
Carry	+		0	+		1	
		1111	1101		1111	1110	
		= FD			= FE		

Ergebnis

CY = 0 → Akku FDH
Z = 0, S = 1, P = 0, CY = 0, AC = 0

CY = 1 → Akku FEH
Z = 0, S = 1, P = 0, CA = 0, AC = 0

DAD rp 0 0 r r 1 0 0 1

Funktion: Der Inhalt des angegebenen Registerpaares rp und der Inhalt des Registerpaares (H,L) werden addiert. Das Ergebnis steht wieder im (H,L)-Registerpaar. rp = (B,C), (D,E) oder (H,L)

Veränderte Zustandsbits: CY

Bytes: 1

Taktzyklen: 10/10

Anmerkung: Es handelt sich um eine 16-Bit-Addition.

Beispiel: Bei Ausführung des Befehls DAD H wird der Inhalt des (H,L)-Registerpaares mit sich selbst addiert und somit verdoppelt.

SUB M 1 0 0 1 0 1 1 0

Funktion: Es wird der Inhalt der Speicherstelle, die durch das (H,L)-Registerpaar adressiert wird, vom Inhalt des Akkumulators abgezogen.

Veränderte Zustandsbits: Z, S, P, CY, AC

Bytes: 1

Taktzyklen: 7/7

Anmerkung: Wie bei SUB r1.

Beispiel: (für beide SUB-Befehle):
Die Ausführung des Befehls SUB A bewirkt, daß der Akku-Inhalt von sich selbst subtrahiert wird und als Ergebnis im Akku Null steht, z.B. sei der Akku-Inhalt 2FH.

	0010 1111	(2F)
+(-)	1101 0000	Einserkomplement
	1	+ 1 für Zweierkomplement
1	0000 0000	

Es werden folgende Zustandsbits gesetzt
Z = 1, S = 0, P = 1, CY = 0, AC = 1
Das Carry-Bit ist 0, da der Übertrag bei SUB negiert eingetragen wird!

SBB r1 1 0 0 1 1 s s s

Funktion: Der Inhalt des angegebenen Registers r1 das Borrow-Bit ("Carry-Bit") werden vom Inhalt des Akkumulator abgezogen. r1 = A, B, C, D, E, H oder L.

Veränderte Zustandsbits: Z, X, P, CY, AC

Bytes: 1

Taktzyklen: 4/4

Anmerkung: Das Borrow-Bit wird mit subtrahiert und evtl. auch verändert. Anwendung beim Subtrahieren von Zahlen über mehrere Bytes.

SBB M 1 0 0 1 1 1 1 0

Funktion: Der Inhalt der Speicherstelle, die durch das (H,L)-Registerpaar adressiert wird und das Borrow-Bit werden vom Inhalt des Akkumulators abgezogen.

Veränderte Zustandsbits: Z, S, P, CY, AC

Bytes: 1

Taktzyklen: 7/7

Anmerkung: Wie bei SBB r1.

Beispiel: Die durch das (H,L)-Registerpaar adressierte Speicherstelle enthalte 02H. Das Carry-Bit sei auf 1 gesetzt und der Akku-Inhalt sei 05H. Der SBB M bewirkt folgende Rechnung

0000	0010	(02H, Speicherstelle)
0000	0011	02H + Carry-Bit
1111	1100	Einserkomplement
+	1	
1111	1101	Zweierkomplement von 03H
+ 0000	0101	(05H, Akku)
1 0000	0010	

Im Akku steht das Ergebnis 02H. Die Zustandsbits werden wie folgt gesetzt

Z = 0, S = 0, P = 0, CY = 0, AC = 1

Das Carry-Bit ist D, da der aufgetretene Übertrag beim SBB negiert ins Carry-Bit geschrieben wird.

ADI konst 1 1 0 0 0 1 1 0

Funktion: Die angegebene Konstante konst wird zum Inhalt des Akkumulator addiert. (konst ≤ 255 dez.).

Veränderte Zustandsbits: Z, S, P, CY, AC

Bytes: 2

Taktzyklen: 7/7

Beispiel: Akku-Inhalt = 12H. Die Konstante sei 36 (≙ 24H)

```
   0001  0010    (12H)
+  0010  0100    (24H)
-------------
   0011  0110
```

Als Ergebnis steht im Akku 36H. Die Zustandsbits sind wie folgt gesetzt

Z = 0, S = 1, P = 1, CY = 0, AC = 0

ACI konst 1 1 0 0 1 1 1 0

Funktion: Die angegebene Konstante konst und das Carry-Bit werden zum Inhalt des Akkumulator addiert, (konst ≤ 255 dez.).

Veränderte Zustandsbits: Z, S, P, CY, AC

Bytes: 2

Taktzyklen: 7/7

Beispiel: Inhalt des Akku sei 12H und die Konstante 36 dezimal, sowie das Carry-Bit auf 1 gesetzt.

```
   0001  0010    (12H)
   0010  0100    (24H)
+             1   Carry-Bit
-------------
   0011  0111
```

Als Ergebnis steht im Akku 37H. Die Zustandsbits sind wie folgt gesetzt

Z = 0, S = 1, P = 0, CY = 0, AC = 0

SUI konst 1 1 0 1 0 1 1 0

Funktion: Die angegebene Konstante konst wird vom Inhalt des Akkumulators abgezogen. (konst ≤ 255 dez.).

Veränderte Zustandsbits: Z, S, P, CY, AC

Bytes: 2

Taktzyklen: 7/7

Beispiel: Der Inhalt des Akku sei 3EH und die abzuziehende Konstante 62 dez.

	0011	1110	3EH, Akku
+(-)	1100	0001	Einerkomplement von 3EH, Konstante
+		1	+ 1 für Zweierkomplement
1	0000	0000	

Als Ergebnis steht im Akkumulator 00H. Die Zustands-Bits lauten Z = 1, S = 0, P = 1, CY = 0, AC = 1

SBI konst 1 1 0 1 1 1 1 0

Funktion: Die angegebene Konstante konst und das Borrow-Bit werden vom Inhalt des Akkumulator abgezogen. (konst ≤ 255 dez.).

Veränderte Zustandsbits: Z, S, P, CY, AC

Bytes: 2

Taktzyklen: 7/7

Beispiel: Die angegebene Konstante sei 2 dez. und das Carry-Bit = 1. Der Akkumulator soll 04H enthalten.

höherwertigen Halbbyte des Inhalts der Wert 6 addiert.

Beispiele zu 1 und 2:

Im ersten Fall sei vor dem DAA-Befehl folgende Addition zweier BCD-Zahlen durchgeführt

93		1001	0011
+ 08		+ 0000	1000
9BH		1001	1011

Dabei wurde gesetzt CY = 0 und AC = 0. Ein folgender DAA bewirkt, da niederwertiges Halbbyte größer 9, folgendes

9BH		1001	1011
+ 06H		+	0110
A1H		1010	0001

mit CY = 0 und AC = 1. Da jetzt das höherwertige Halbbyte größer 9 ist, addiert der DAA dort noch eine 6 hinzu

A1H		1010	0001
+ 60H		+ 0110	0000
101H		1 0000	0001

Das Carry-Bit und das Hilfscarry-Bit wurden beide mit 1 gesetzt. Der Akkumulator enthält jetzt 01H, entsprechend 01 im BCD-Format. Wird diesem Wert das Carry-Bit vorangestellt, erhält man das dezimale Ergebnis 101.

Im Fall 2 seien die BCD-Zahlen 99 und 88 addiert worden

Es wird folgende Rechnung ausgeführt

	0000	0010	(02H, Konstante)
	0000	0011	02H + Carry-Bit
	1111	1100	Einerkomplement
+		1	
	1111	1101	Zweierkomplement von 03H
+	0000	0100	(04H, Akku)
1	0000	0001	

Im Akku steht das Ergebnis 01H. Die Zustands-Bits werden wie folgt gesetzt

Z = 0, S = 0, P = 0, CY = 0, AC = 1

Das Carry-Bit ist 0, da der aufgetretene Übertrag beim Subtrahieren negiert in das Carry-Bit geschrieben wird.

DAA 0 0 1 0 0 1 1 1

Funktion: Der Akkumulator-Inhalt wird in eine zweistellige Zahl umgewandelt. Der 8-Bit-Wert wird so umgewandelt, daß zwei 4-Bit-binärcodierte Ziffern entstehen.

Veränderte Zustandsbits: Z, S, P, CY, AC

Bytes: 2

Taktzyklen: 4/4

Anmerkung: Die Arbeitsweise des DAA ist folgendermaßen:

1. Wenn das niederwertige Halbbyte des Akkumulators einen Wert größer 9 hat oder das Hilfscarry-Bit auf 1 gesetzt ist, wird zum Inhalt des Akkumulator der Wert 6 addiert.
2. Wenn das höherwertige Halbbyte des Akkumulators einen Wert größer 9 hat oder das Carry-Bit auf 1 gesetzt ist, wird zum

	99		1001 1001
+	88	+	1000 1000
	21H		0010 0001

Dabei wurden die beiden Zustands-Bits Carry und Hilfscarry auf 1 gesetzt.

Folgt jetzt ein DAA-Befehl, so wird auf das niederwertige Halbbyte eine 6 addiert, da das Hilfscarry-Bit gleich 1 ist, und auf das höherwertige Halbbyte ebenfalls eine 6 addiert, weil auch das Carry-Bit gleich 1 ist.

	21H		0010 0001
+	06H	+	0000 0110
	27H		0010 0111
+	60H	+	0110 0000
	87H		1000 0111

Mit Berücksichtigung von CY = 1 ergibt sich zusammen mit dem Akku-Inhalt das Ergebnis 187.

2.4.3. Logische Operationen

CMA — 0 0 1 0 1 1 1 1

Funktion: Der Inhalt des Akkumulators wird negiert.

Veränderte Zustandsbits: Keine

Bytes: 1

Taktzyklen: 4/4

Beispiel: Akku-Inhalt vor dem CMA sei 02H (≙ 0000 0010). Nach Ausführung des Befehls ist der Inhalt FCH (≙ 1111 1101).

ANA r1 — 1 0 1 0 0 s s s

Funktion: Der Inhalt des angegebenen Registers r1 und der Inhalt des Akkumulators werden logisch UND-verknüpft. r1 = A, B, C, D, E, H oder L.

Veränderte Zustandsbits: Z, S, P, CY = 0, AC (für 8080)
Z, S, P, CY = 0, AC = 1 (für 8085)

Bytes: 1

Taktzyklen: 4/4

Anmerkung: Der Befehl wird beispielsweise verwendet, um einzelne Bits des Akkumulators abzuprüfen, in dem im Vergleichsbyte (Maske) das Bit in der zu prüfenden Stelle mit 1 gesetzt wird. Die logische UND-Verknüpfung wird auch verwendet, um einzelne Bits auszublenden.

ANA M — 1 0 1 0 0 1 1 0

Funktion: Der Inhalt des Speicherbytes, das durch das (H,L)-Registerpaar adressiert ist, wird mit dem Akkumulator-Inhalt logisch UND-verknüpft.

Veränderte Zustandsbits: Z, S, P, CY = 0, AC (für 8080)
Z, S, P, CY = 0, AC = 1 (für 8085)

Bytes: 2

Taktzyklen: 7/7

Anmerkung: Wie bei ANA r1

Beispiel: Voraussetzung ist die Kenntnis der logischen UND-Verknüpfung. Im Akkumulator stehe als Ergebnis einer durchgeführten Operation B3H, entsprechend 1011 0011.
Per Befehl soll jetzt beispielsweise geprüft werden, ob in dem Ergebnis in der 5. und 6. Stelle (Bits 2^4 und 2^5) eine 1 enthalten ist. Das Prüfbyte (Maske) muß dann 0011 0000 (= 03H) lauten und kann entweder von einem Register, aus einer Speicherstelle oder als Konstante geliefert werden. Es wird folgende Operation ausgeführt

	1011	0011	(B3H, Akku)
UND	0011	0000	Maske
	0011	0000	Ergebnis im Akku

Im Ergebnis wird in einer Stelle nur dann eine 1 erzeugt, wenn in beiden Bytes in dieser Stelle eine 1 steht.

ORA r1 1 0 1 1 0 s s s

Funktion: Der Inhalt des angegebenen Registers r1 und der Inhalt des Akkumulators werden logisch ODER - verknüpft. r1 = A, B, C, D, E, H oder L.

Veränderte Zustandsbits: Z, S, P, CY = 0, AC = 0

Bytes: 1

Taktzyklen: 4/4

Anmerkung: Der Befehl wird beispielsweise verwendet, um gezielt einzelne Bits in einem Byte mit 1 zu setzen.

ORA M 1 0 1 1 0 1 1 0

Funktion: Der Inhalt des Speicherbytes, das durch das (H,L)-Registerpaar adressiert ist, wird mit dem Akkumulator-Inhalt logisch ODER-verknüpft.

Veränderte Zustandsbits: Z, S, P, CY = 0, AC = 0

Bytes: 1

Taktzyklen: 7/7

Anmerkung: Wie bei ORA r1.

ORI konst 1 1 1 1 0 1 1 0

Funktion: Der Inhalt des Akkumulator wird mit der angegebenen Konstanten konst logisch ODER-verknüpft. (konst ≤ 255 dezimal).

Veränderte Zustandsbits: Z, S, P, CY = 0, AC = 0

Bytes: 2

Taktzyklen: 7/7

Anmerkung: Wie bei ORA r1.

Beispiel: Voraussetzung ist die Kenntnis der logischen ODER-Verknüpfung. Im Akkumulator stehe als Ergebnis einer durchgeführten Operation B3H, entsprechend 1011 0011. Das Bit mit der Stellenwertigkeit 2^6 muß für die weitere Verarbeitung beispielsweise zu 1 gesetzt werden. Dann muß der erforderliche Operand 0100 0000 (= 40H) sein und kann durch ein Register, aus einer Speicherzelle oder als Konstante zur Verfügung gestellt werden. Es wird folgende Operation ausgeführt

	1011	0011	(B3H, Akku)
ODER	0100	0000	Maske
	1111	0011	Ergebnis im Akku

Im Ergebnis wird in einer Stelle eine 1 erzeugt, wenn in einer oder beiden der verknüpften Stellen eine 1 steht.

XRA r1 1 0 1 0 1 s s s

Funktion: Der Inhalt des angegebenen Registers r1 und der Inhalt des Akkumulators werden logisch EXKLUSIV-ODER-verknüpft.
r1 = A, B, C, D, E, H oder L.

Veränderte Zustandsbits: Z, S, P, CY = 0, AC = 0

Bytes: 1

Taktzyklen: 4/4

Anmerkung: Die logische EXKLUSIV-ODER-Verknüpfung erzeugt im Ergebnis dann eine 1, wenn die beiden Bits im zu prüfenden Byte und in der Maske verschieden sind. Häufige Anwendung ist z.B. das Null-Setzen des Akkumulators.

XRA M 1 0 1 0 1 1 1 0

Funktion: Der Inhalt des Speicherbytes, das durch das (H,L)-Registerpaar adressiert ist, wird mit dem Akkumulator-Inhalt EXKLUSIV-ODER-verknüpft.

Veränderte Zustandsbits: Z, S, P, CY = 0, AC = 0

Bytes: 1

Taktzyklen: 7/7

Anmerkung: Wie bei XRA r1.

XRI konst 1 0 1 0 1 1 1 0

Funktion: Der Inhalt des Akkumulators wird mit der angegebenen Konstanten konst EXKLUSIV-ODER-verknüpft. (konst ≤ 255 dezimal).

Veränderte Zustandsbits: Z, S, P, CY = 0, AC = 0

Bytes: 2

Taktzyklen: 7/7

Anmerkung: Wie bei XRA r1.

Beispiel: Voraussetzung ist die Kenntnis der logischen EXKLUSIV-ODER-Verknüpfung. Es soll beispielsweise der Inhalt des Akkumulators gelöscht werden. Der XRA A führt folgende Operation aus, wenn im Akku vorher 3AH steht

	0011	1010	3AH, Akku
EXKLUSIV-ODER	0011	1010	Verknüpfung mit sich selbst
	0000	0000	

Als Ergebnis steht im Akku 00H.

CMP r1 1 0 1 1 1 s s s

Funktion: Der Inhalt des angegebenen Registers r1 und der Inhalt des Akkumulators werden in ihren Werten verglichen.
r1 = A, B, C, D, E, H oder L.

Veränderte Zustandsbits: Z, S, P, CY, AC

Bytes: 1

Taktzyklen: 4/4

Anmerkung: Folgende Ergebnisse werden angezeigt

Z = 1 → Akku- und Registerinhalte gleich

CY = 0 → Akkuinhalt größer als Registerinhalt

CY = 1 → Akkuinhalt kleiner als Registerinhalt

Die Funktion der Carry-Bits dreht sich um, wenn einer der Werte negativ oder komplementiert ist! Die Ursprungsdaten bleiben erhalten. Anwendung aller Vergleichsbefehle z.B. bei der Erzeugung von Verzweigungsbedingungen in Programmen bei Sprüngen. Die Vergleiche werden als Subtraktion ausgeführt.

CMP M 1 0 1 1 1 1 1 0

Funktion: Der Inhalt des Speicherbytes, das durch das (H,L)-Registerpaar adressiert ist, wird mit dem Inhalt des Akkumulators wertmäßig verglichen.

Veränderte Zustandsbits: Z, S, P, CY, AC

Bytes: 1

Taktzyklen: 7/7

Anmerkung: Wie bei CMP r1.

CPI konst 1 1 1 1 1 1 1 0

Funktion: Der Inhalt des Akkumulators und die angegebene Konstante konst werden in ihren Werten verglichen. (konst ≤ 255 dezimal).

Veränderte Zustandsbits: Z, S, P, CY, AC

Bytes: 2

Taktzyklen: 7/7

Anmerkung: Wie bei CMP r1.

Beispiel: Es soll der Befehl CMP D ausgeführt werden mit folgenden Inhalten

Akku = OAH Reg. D = O5H

```
   0000  1010   (OAH, Akku)
+  1111  1011   (Zweierkomplement-Addition von O5H)
-------------
   0000  0101
```

Das Carry-Bit wird negiert und somit sind das Zero- und Carry-Bit beide O, d.h. der Akkumulator-Inhalt ist größer als der des Registers D.

2.4.4. Registeranweisungen

A. Schiebebefehle für Akkumulator

RLC O O O O O 1 1 1

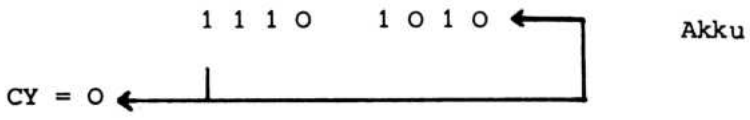
Funktion: Der Inhalt des Akkumulators wird um 1 Bit nach links geschoben. Das höchstwertige Bit (Stelle 2^7) wird in das Carry-Bit und in die niederwertigste Stelle (2^0) geschrieben.

Veränderte Zustandsbits: CY

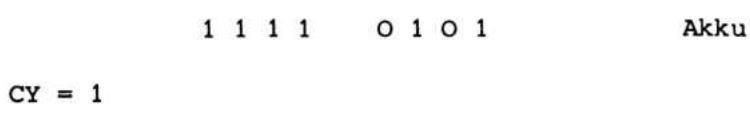
Bytes: 1

Taktzyklen: 4/4

Beispiel: Vor dem RLC sei der Akkumulator-Inhalt EAH und das Carry-Bit O



Nach RCL:



Nach 8 RLC-Anweisungen steht im Akku der alte Inhalt.

RRC O O O O 1 1 1 1

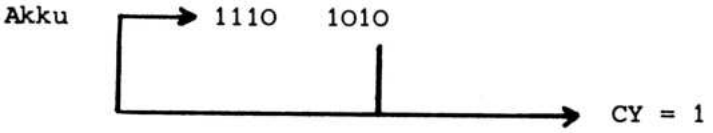
Funktion: Der Inhalt des Akkumulator wird um 1 Bit nach rechts geschoben. Das niederwertigste Bit (Stelle 2^0) wird in das Carry-Bit und in die höchstwertige Stelle (2^7) geschreiben.

Veränderte Zustandsbits: CY

Bytes: 1

Taktzyklen: 4/4

Beispiel: Vor dem RRC sei der Akkumulator-Inhalt EAH und das Carry-Bit 1



Nach RRC

0111 0101

CY = 0

Nach 8 RRC-Anweisungen steht im Akku wieder der alte Inhalt.

RAL 0 0 0 1 0 1 1 1

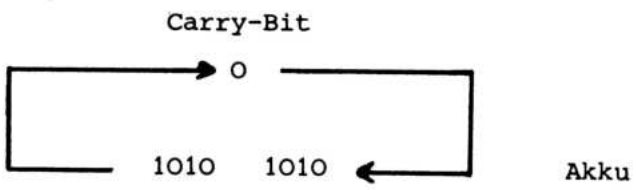
Funktion: Linksschieben durch das Carry-Bit. Der Inhalt des Akkumulators wird um 1 Bit nach links geschoben. Das Bit aus der Stelle 2^7 wird in das Carry-Bit und dieses selbst in die Stelle 2^0 geschrieben.

Veränderte Zustandsbits: CY

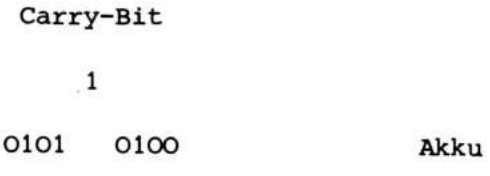
Bytes: 1

Taktzyklen: 4/4

Beispiel: Vor dem RAL sei der Akkumulator-Inhalt AAH und das Carry-Bit 0



Nach RAL



RAR O O O 1 1 1 1 1

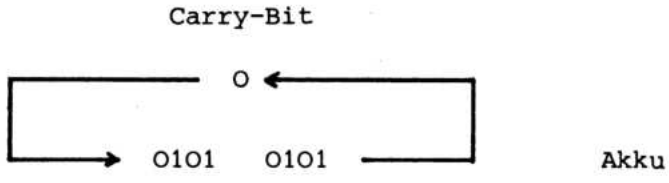
Funktion: Rechts-Schieben durch das Carry-Bit. Der Inhalt des Akkumulators wird um 1 Bit nach rechts geschoben. Das Bit aus Stelle 2^0 wird in das Carry-Bit und dieses selbst in die Stelle 2^7 geschrieben.

Veränderte Zustandsbits: CY

Bytes: 1

Taktzyklen: 4/4

Beispiel: Vor dem RAR sei der Akkumulator-Inhalt 55H und das Carry-Bit O



Nach RAR

B. Übertragsbit-(CY)-Anweisungen

CMC O O 1 1 1 1 1 1

Funktion: Das Carry-Bit wird negiert.

Veränderte Zustandsbits: CY

Bytes: 1

Taktzyklen: 4/4

Beispiel: War nach einer Operation das Carry-Bit mit 1 gesetzt, so ist es nach der CMC-Anweisung gleich O.

STC 0 0 1 1 0 1 1 1

Funktion: Das Carry-Bit wird unabhängig seines vorherigen Zustandes mit 1 gesetzt.

Veränderte Zustandsbits: CY = 1

Bytes: 1

Taktzyklen: 4/4

2.4.5. Sprungbefehle

A. Unbedingte Sprünge

PCHL 1 1 1 0 1 0 0 1

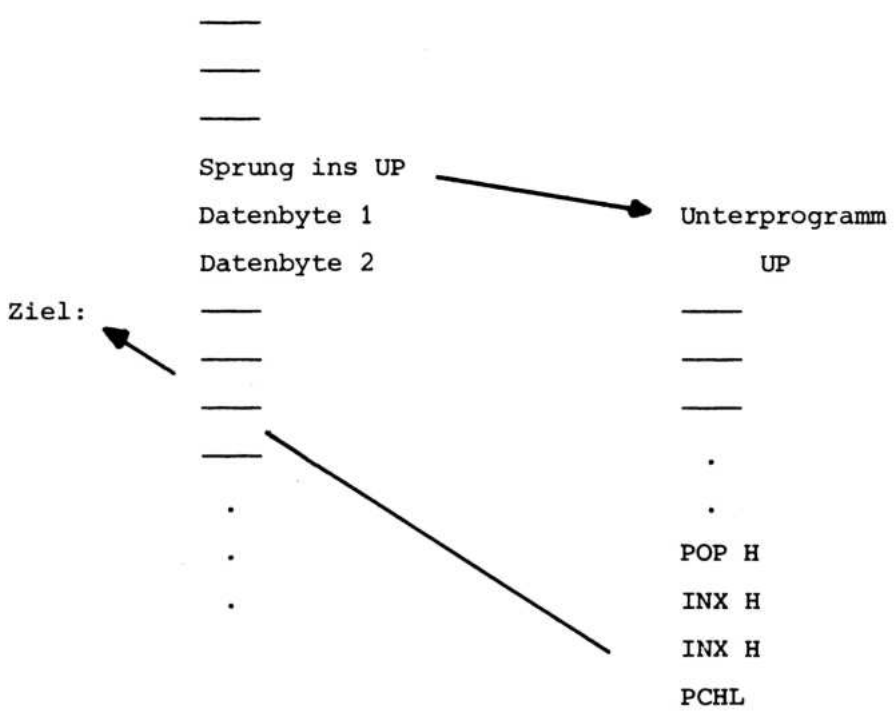
Funktion: Der Inhalt des (H,L)-Registerpaares wird in den Programmzähler (Befehlszähler) geladen und das Programm an dieser Stelle fortgesetzt.

Veränderte Zustandsbits: Keine

Bytes: 1

Taktzyklen: 5/6

Beispiel: Nach einem Unterprogrammaufruf sollen zwei Datenbytes stehen und der Rücksprung aus dem Unterprogramm an die nächste Adresse erfolgen. Dann kann folgender Ablauf programmiert werden



Die Anweisung PCHL im Unterprogramm bewirkt, daß im Hauptprogramm nach Ziel: gesprungen wird und nicht nach Datenbyte 1. Sehen Sie sich die vier Befehle und ihre Funktion im Unterprogramm UP genau an.

JMP adr 1 1 0 0 0 0 1 1

Funktion: Der Wert der angegebenen Adresse adr wird in den Programmzähler geladen und das Programm an dieser Stelle fortgesetzt.

Veränderte Zustandsbits: Keine

Bytes: 3

Taktzyklen: 10/10

B. Bedingte Sprünge

Funktion: Bei allen folgenden bedingten Sprungbefehlen wird der Programmablauf an der angegebenen Adresse fortgesetzt, wenn die entsprechende Bedingung erfüllt ist.

Veränderte Zustandsbits: Keine

Bytes: 3

Taktzyklen: 10/7(10)

JC	11011010	Wenn Carry-Bit = 1
JNC	11010010	Wenn Carry-Bit = 0
JZ	11001010	Wenn Zero-Bit = 1
JNZ	11000010	Wenn Zero-Bit = 0
JM	11111010	Wenn Signum-Bit = 1
JP	11110010	Wenn Signum-Bit = 0
JPE	11101010	Wenn Parity-Bit = 1
JPO	11100010	Wenn Parity-Bit = 0

Anmerkung: Die bedingten Sprungbefehle werden eingesetzt, um von der "Vorgeschichte" abhängige Programmverzweigungen auszuführen.

2.4.6. Unterprogrammbehandlung

A. Unbedingter Programmaufruf

CALL adr 1 1 0 0 1 1 0 1

Funktion: Das Programm wird in dem an der angegebenen Adresse adr stehenden Unterprogramm fortgesetzt.

Veränderte Zustandsbits: Keine

Bytes: 3

Taktzyklen: 17/18

B. Bedingte Programmaufrufe

Funktion: Bei allen folgenden bedingten Unterprogrammaufrufen wird der Programmablauf bei dem an der angegebenen Adresse stehenden Unterprogramm fortgesetzt, wenn die entsprechende Bedingung erfüllt ist.

Veränderte Zustandsbits: Keine

Bytes: 3

Taktzyklen: (11/17)/(9/18)

CC	1101100	Wenn Carry-Bit = 1
CNC	1101100	Wenn Carry-Bit = 0
CZ	1100100	Wenn Zero-Bit = 1
CNZ	1100100	Wenn Zero-Bit = 0
CM	1111100	Wenn Signum-Bit = 1
CP	1111100	Wenn Signum-Bit = 0
CPE	1110100	Wenn Parity-Bit = 1
CPO	1110100	Wenn Parity-Bit = 0

Anmerkung: Die bedingten Unterprogrammaufrufe werden einge-

setzt, um von der "Vorgeschichte" abhängige Unterprogrammaufrufe ausführen zu können.

RST konst. 1 1 n n n 1 1 1

Funktion: Spezieller Unterprogrammaufruf um in Unterbrechungs-Routinen zu verzweigen. Das Programm wird bei der Adresse 8 x konst fortgesetzt. (konst. ≤ 7 dezimal)

Veränderte Zustandsbits: Keine

Bytes: 3

Taktzyklen: 11/12

Anmerkung: Häufiger Einsatz des RST-Befehls ist die Unterbrechungsbehandlung mittels spezieller Programmteile, wenn Peripheriegeräte den RST an den Prozessor senden. Dabei schiebt dieser den 3-Bit-Adressencode in die Bits 3, 4 und 5 des Programmzählers (≙ einer Multiplikation mit 8). Der Programmablauf wird an dieser neuen Adresse fortgesetzt. Die dabei zur Verfügung stehenden 8 Byte können die Unterbrechungsroutine oder, falls der Platz zu klein ist, ein Unterprogrammaufruf enthalten.

C. Rücksprungbefehle

RET 1 1 0 0 1 0 0 1

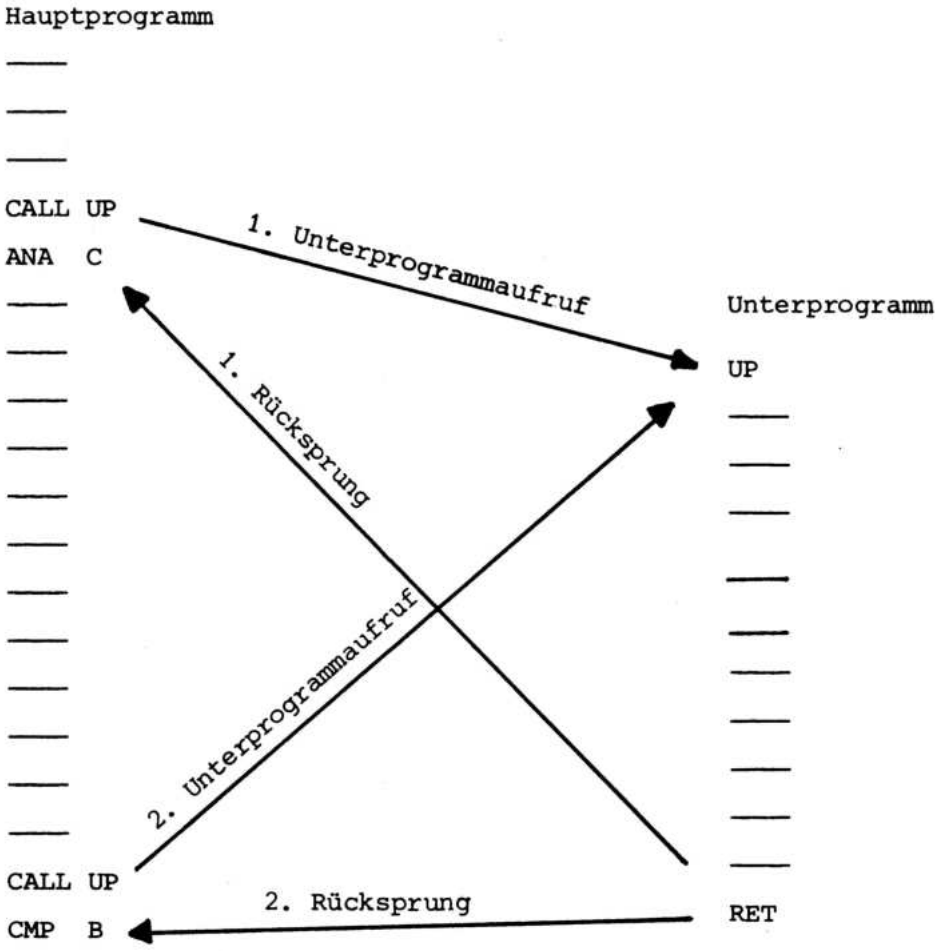
Funktion: Das Programm wird bei Beendung eines Unterprogramms an der Adresse fortgesetzt, die in den Speicherstellen steht, welche durch das Stapelzeiger-Register adressiert werden.

Veränderte Zustandsbits: Keine

Bytes: 1

Taktzyklen: 10/10

Anmerkung: Der RET-Befehl wird fast ausschließlich in Zusammenhang mit dem CALL-Befehl bzw. den bedingten Call-Aufrufen verwendet.



Ein Unterprogramm kann in einem Hauptprogramm mehrfach aufgerufen werden, wie vorher dargestellt. Außerdem können Unterprogramme geschachtelt werden, d.h. ein Unterprogramm kann ein weiteres Unterprogramm aufrufen, welches wiederum ein Unterprogramm aufruft. Beim Programmieren ist auf eine einwandfreie Steuerung der Schachtelung zu achten. Eine große Schachtelungstiefe verringert die Übersicht.

Bedingte Rücksprungbefehle

Funktion: Bei allen folgenden bedingten Rücksprungbefehlen wird das Programm nur dann fortgesetzt, wenn die entsprechende Bedingung erfüllt ist. Die Rücksprungadresse steht in den Speicherstellen, die über das Stapelzeiger-Register adressiert werden.

Veränderte Zustandsbits: Keine

Bytes: 1

Taktzyklen: (5/11)/(6/12)

Anmerkung: Wie bei RET

RC	1101100	Wenn Carry-Bit = 1
RNC	1101000	Wenn Carry-Bit = 0
RZ	1100100	Wenn Zero-Bit = 1
RNZ	1100000	Wenn Zero-Bit = 0
RM	1111100	Wenn Signum-Bit = 1
RP	1111000	Wenn Signum-Bit = 0
RPE	1110100	Wenn Parity-Bit = 1
RPO	1110000	Wenn Parity-Bit = 0

2.4.7. Programmunterbrechung

EI	1 1 1 1 1 0 1 1
Funktion:	Das Interrupt-(Unterbrechungs-)Flipflop wird gesetzt. Dadurch kann der Mikroprozessor Unterbrechungsanforderungen annehmen.
Veränderte Zustandsbits:	Keine
Bytes:	1
Taktzyklen:	4/4
Anmerkung:	Anwendung, wenn in bestimmten Programmteilen eine Unterbrechung von außen auftreten darf. Die Anweisung DI sperrt die Interrupt-Eingänge.

DI	1 1 1 1 0 0 1 1
Funktion:	Das Interrupt-Flipflop wird rückgesetzt und sperrt die Interrupt-Eingänge (außer TRAP-Eingang).
Veränderte Zustandsbits:	Keine
Bytes:	1
Taktzyklen:	4/4
Anmerkung:	Anwendung, wenn in bestimmten Programmteilen keine Unterbrechungen von außen auftreten dürfen, die eine sofortige Änderung des Programmablaufs verursachen würden. Die Anweisung EI macht die Unterbrechungsanforderungen wieder zulässig.

2.4.8. Sonstige Befehle

HLT O 1 1 1 O 1 1 O

Funktion: Die Halt-Anweisung hält den Mikroprozessor an, der Programmzähler behält die Adresse des nächsten Befehls und die Register bleiben unverändert.

Veränderte Zustandsbits: Keine

Bytes: 1

Taktzyklen: 7/5

Anmerkung: Im Halte-Zustand kann der Mikroprozessor durch eine externe Unterbrechung neu gestartet werden. Dazu ist vorher der Interrupt freizugeben (EI-Anweisung). Anwendung z.B. um Wartepausen für langsame Peripheriegeräte einzufügen. (Während dieser Pausen steht der Prozessor nicht für andere Aufgaben zur Verfügung!)

NOP O O O O O O O O

Funktion: Leerbefehl. Es wird keine Funktion ausgeführt.

Veränderte Zustandsbits: Keine

Bytes: 1

Taktzyklen: 4/4

Anmerkung: Verwendung zum Beispiel bei der Programmierung von Zeitschleifen (Füllbefehl).

2.4.9. Spezielle 8085-Befehle

RIM O O 1 O O O O O

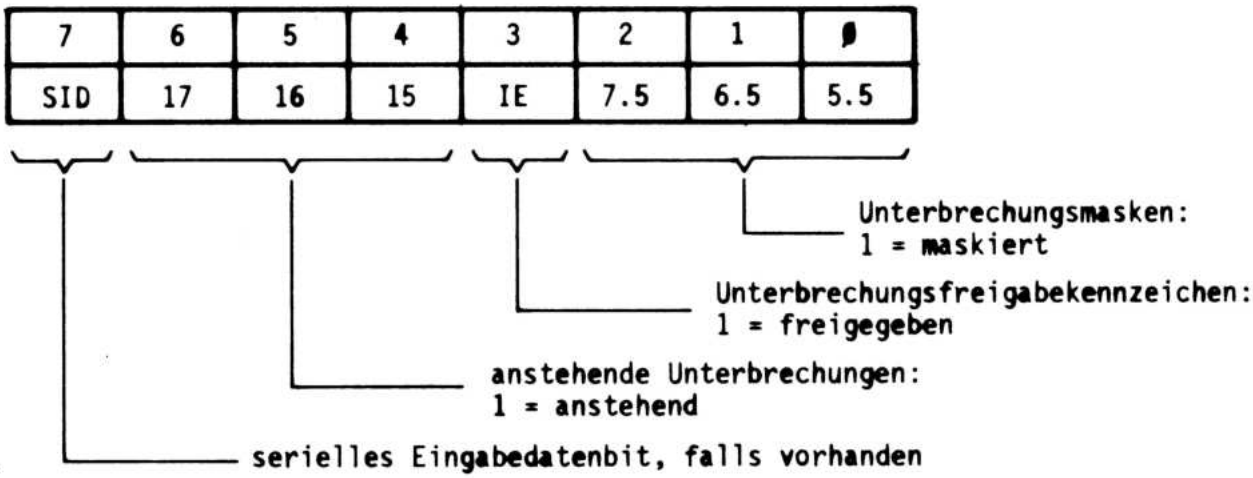
Funktion: Lesen der Interrupt-Maske. Der Befehl lädt 8 Bits in den Akkumulator, die aus dem Stand der Unterbrechungsmaske, dem Stand des Unterbrechungskennzeichens, den evtl. anstehenden Unterbrechungen und dem Bit des seriellen Eingabekanals abgeleitet werden.

Veränderte Zustandsbits: Keine

Bytes: 1

Taktzyklen: -/4

Anmerkungen:



SIM O O 1 1 O O O O

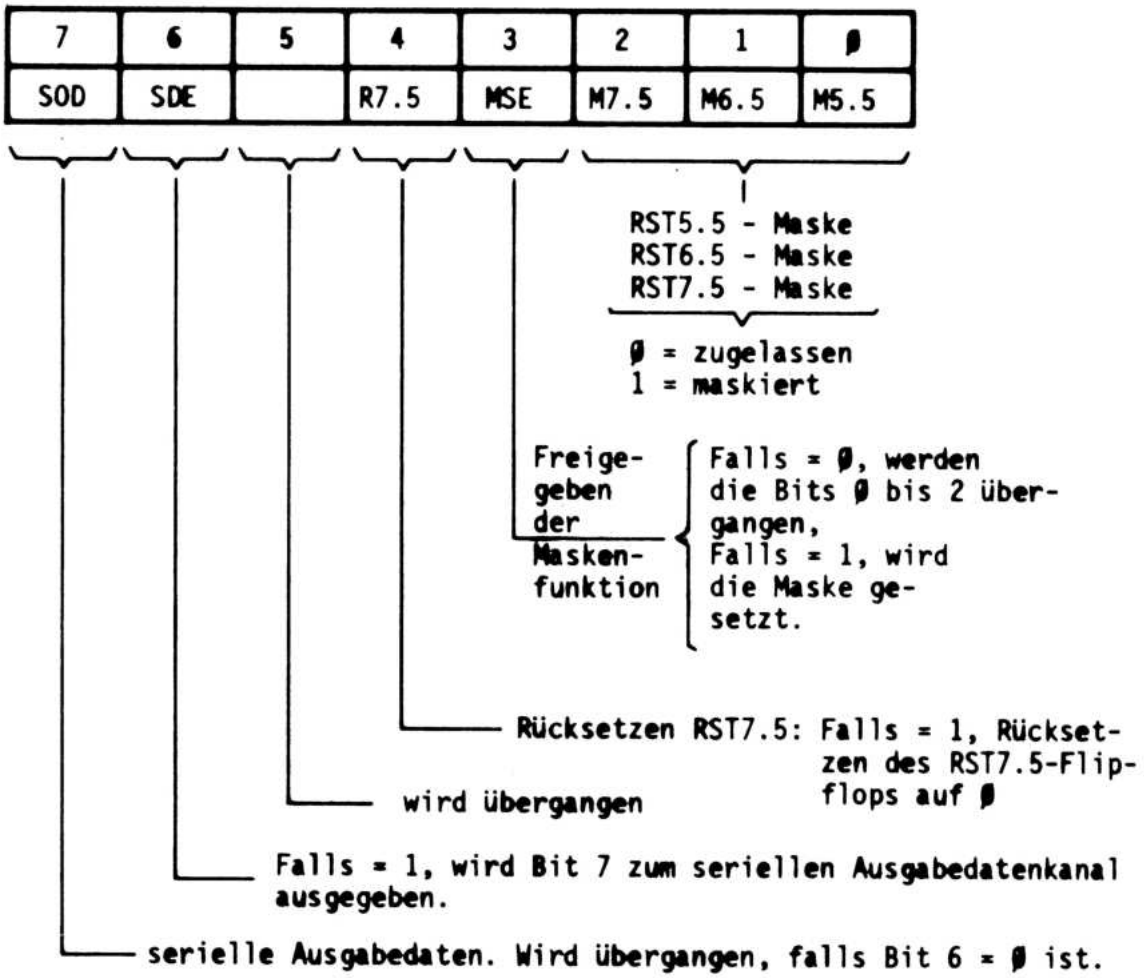
Funktion: Setzen der Interrupt-Maske. Mit dem Inhalt des Akkumulators werden folgende Funktionen ausgeführt: Die Unterbrechungsmaske für die RST-Hardware-Unterbrechungen wird gesetzt, Rücksetzen des RST 7,5-Eingangs und Ausgabe des Bit 7 auf dem seriellen Ausgabekanal.

Veränderte Zustandsbits: Keine

Bytes: 1

Taktzyklen: -/4

Anmerkung:



2.4.10. Schlußbemerkungen zum Befehlssatz

Die gesamten Befehle in ihrer Vielfalt werden Ihnen beim Lesen verwirrend und vielleicht in ihrer Funktion nicht sofort verständlich vorgekommen sein. Wir werden in der nächsten Lektion die Funktion der Befehle in ihren unterschiedlichen Typen in den Programmierübungen vertiefen. Dabei werden es erst ganz kurze Aufgaben sein, die zum Teil nur aus ein paar wenigen Befehlen bestehen und dann immer weiter ausgebaut werden.

Um Ihr Verständnis zu wecken, weisen wir an dieser Stelle auch noch einmal darauf hin, wie der Mnemonic-Code für die Befehle gebildet wurde. Alle Kürzel sind irgendwie aus der englischen verbalen Beschreibung des Befehls abgeleitet. Beispielsweise

Mnemonic	Englisch	Deutsch
XCHG	exchange	vertauschen
STA	store accu	Akku speichern
INR	increment register	Register inkrementieren
JMP	jump	springe
JNZ	jump no zero	springe, wenn nicht Null
RET	jump return	Rücksprung
RAL	rotate accu left	Akku links rotieren

Wenn man die Befehle häufiger gebraucht und den Umfang auch etwas im Griff hat, wird man beim Programmieren nicht mehr wegen jedem Byte nachlesen müssen. Neben den von uns erstellten Befehlslisten wollen wir auch noch auf eine Befehlsliste verweisen, die von Fa. SIEMENS zu ihren Mikroprozessoren 8080/8085 vertrieben wird, (Bestell-Nr. B/2279). Diese Liste enthält neben den Mnemonics, Hex- und Binär-Codes der Befehle auch Angaben über Zustandsänderungen, Taktzyklen, englische und deutsche Beschreibungen der Befehle, ASCII-Zeichen-Tabelle usw. Als ergänzende Literatur gibt es ebenfalls von SIEMENS das Buch "Makroassembler-Programmiersprache System 8080/8085", Bestell-Nr. B2263.

3. Weitere Einführung in die Bedienung des "micromaster"

Im folgenden werden wir zwei weitere Kommandos für die Bedienung des micromaster kennenlernen. Bei der Inbetriebnahme beachten Sie bitte die Hinweise der ersten Lektion (Abschnitt 5.2.).

Übungen mit dem Kommando "1":

Das Kommando 1 führt folgende Funktion aus: Ein Speicherbyte oder auch ein ganzer Speicherbereich wird mit einem gewünschten Wert (Konstante) geladen, d.h. nach Eingabe des Kommandos 1 fragt der micromaster nach einer Startadresse, der Endadresse und der Konstanten, mit der der Speicher gefüllt werden soll. Ein häufiger Anwendungsfall ist z.B. das Rücksetzen eines Speicherbereiches indem die Konstante OOH eingeschrieben wird, oder auch das Setzen eines definierten Anfangszustandes.

In einem Beispiel soll der Umgang mit dem Kommando 1 geübt werden. Dazu werden wir auch das Kommando O benötigen, um vor und nach Ausführung von Kommando 1 die Speicherinhalte auszulesen. Setzen Sie den "micromaster" jetzt in Betrieb.

"micromaster"	Anwender
Der "micromaster" erwartet eine Eingabe	
	Kommando O eingeben
In der Anzeige erscheint O. X X X X = X X	
	Geben Sie im Adressenfeld 1800 (Anfang des RAM-Speicherbereiches) ein und notieren Sie den Dateninhalt
In der Anzeige steht jetzt O. 1 8 O O = X X (XX = zufälliges Zeichen)	

Drücken Sie 16 mal die (+)-Abschlußtaste und notieren Sie nach jedem Drücken die Adresse und das zugehörige Datenwort.

In der Anzeige steht jetzt

0. 1 8 0 F = X X

Jetzt beginnen wir mit dem Kommando 1. Schieben Sie das Blinkzeichen mit einer der "Cursor"-Tasten L oder R in die Kommandostelle (erste Schreibstelle von links). Geben Sie eine 1 ein und drücken Sie anschließend die Abschlußtaste (+).

Der "micromaster" antwortet mit

1. A A = X X X X

Adressfeld

Der "micromaster" will von Ihnen die Anfangsadresse A A des Speicherbereichs wissen, der mit einer Konstanten gefüllt werden soll. Geben Sie wieder 1 8 0 0 ein und drücken dann die Abschlußtaste (+).

In der Anzeige steht jetzt

1. E A = X X X X

Der "micromaster" erwartet von Ihnen die Eingabe der Endadresse, bis zu der der Speicherbereich mit einer Konstanten gefüllt werden soll. Geben Sie 1 8 0 F ein und drücken anschließend wieder die Abschlußtaste (+).

Der "micromaster" gibt die Aufforderung zur Eingabe der Konstanten aus

1. d b = X X

Die Angabe d b für Datenbyte kennzeichnet die Konstante, die wir jetzt eingeben. Wählen Sie zwei beliebige Hex-Ziffern. (Bei der ersten Verwendung des Kommandos 1 ist es sinnvoll, zwei gleiche Ziffern, z.B. O O, einzugeben, damit sich der neue Speicherinhalt deutlich von dem Inhalt unterscheidet, der vor Ausführung des Kommandos bestand.) Abschlußtaste (+) drücken.

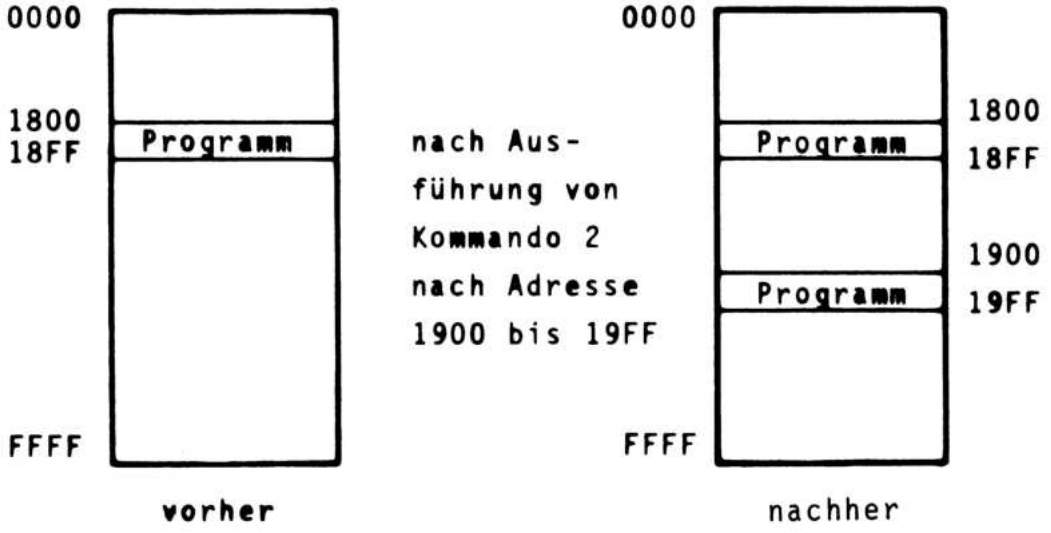
Nach Ausführung des Kommandos meldet sich der "micromaster" mit einer 1 in der Kommandostelle.

Nach der Ausführung des Kommandos 1 verwenden wir wieder das Kommando O, um den neuen Speicherinhalt anzuschauen. Gehen Sie wie am Anfang dieser Übungs beschrieben vor und vergleichen Sie den neuen Speicherinhalt mit dem, den Sie anfangs notiert haben.

Wiederholen Sie diese Übung, indem Sie in demselben Speicherbereich eine andere Konstante einschreiben und abschließend wieder einen Vergleich anstellen.

Übungen mit dem Kommando "2":

Das Kommando 2 führt folgende Funktion aus: Ein freiwählbarer Speicherbereich wird in einen anderen Speicherbereich kopiert. Der Ursprungsbereich wird dabei nicht verändert. Es ergibt sich für ein Beispiel folgende Speicherbelegung nach Ausführung des Kommandos 2:



In diesem Kommando wollen wir auch eine Übung durchführen. Um zuvor einen bekannten definierten Zustand vor Ausführung des Kommandos herzustellen, füllen wir die Speicherbereiche

1800 bis 18FF mit der Konstanten FF

und

1900 bis 19FF mit der Konstanten 00

unter Ausführung von Kommando 1. Um dann die Übertragung des Speicherbereichs zu bewerkstelligen, ist folgender Ablauf erforderlich:

"micromaster"	Anwender
	Da der "micromaster" eingeschaltet ist, können wir gleich die Schreibmarke in die Kommandostelle bringen. Wir geben eine 2 für das auszuführende Kommando ein und drücken die Abschlußtaste (+).
Der "micromaster" antwortet mit 2. A A = X X X X Adressfeld	
	In das Adressfeld geben wir die Anfangsadresse (AA) des zu verschiebenden Bereiches ein, also 1 8 0 0 und drücken die Abschlußtaste (+).
Der "micromaster" will jetzt die Endadresse wissen 2. E A = X X X X	
	Jetzt geben wir in das Adressfeld die Endadresse (EA) des zu verschiebenden Bereiches ein, 1 8 F F . Abschlußtaste (+).
Der "micromaster" erwartet eine Zielangabe (Bestimmungsadresse, BA) 2. b A = X X X X	
	Laut Aufgabenstellung war der Anfang des Zielspeicherbereiches 1 9 0 0. Wir geben diese Adresse ein und drücken die Abschlußtaste (+).

Nach Ausführung des Kommandos meldet sich der "micromaster" wieder in der Kommandostelle mit der blinkenden 2.

Mit Kommando O kontrollieren wir den Speicherbereich von 1900 bis 19FF. Hier muß jetzt auch in jedem Speicherbyte der Wert FF stehen.

Die Angabe einer Endadresse für den Zielbereich war nicht erforderlich, da der zu übertragende Umfang bereits durch die beiden Adressen des Ursprungsbereiches eindeutig bestimmt war. In der nächsten Lektion werden wir 6 weitere Kommandos und deren Funktion kennenlernen. Bis dahin sollten Ihnen die Funktion und der Gebrauch der Kommandos O, 1, 2 und 7 geläufig sein.

4. Die Hardware-Beschreibung des "micromaster"

Die Hardware-Beschreibung des Übungsgerätes soll die Erläuterungen über Mikroprozessoren und deren Einsatz in Computern aus der ersten Lektion vertiefen. Um Ihnen den Überlick über die Hardware möglichst einfach zu machen, finden Sie 3 Bilder in diesem Abschnitt

a. noch einmal den Lageplan der Bauelemente

b. ein Blockschaltbild und

c. einen detaillierten Stromlaufplan (im Anhang)

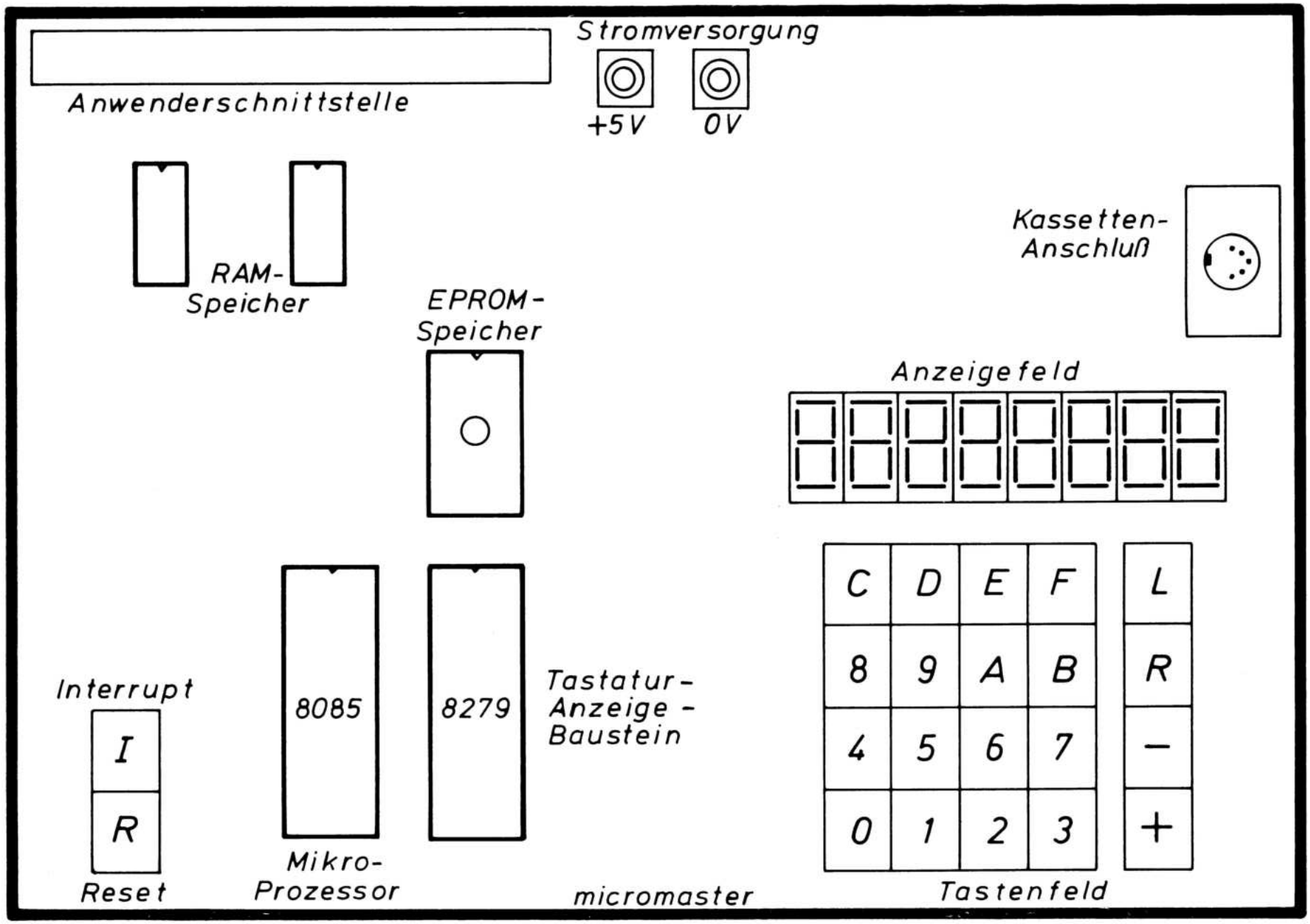
Der auf einer Leiterplatte im Doppel-Europa-Format aufgebaute micromaster enthält als Mikrocomputer die Komponenten Zentraleinheit (8085), RAM-Speicher (2114), Festspeicher (EPROM 2716), ein Tasten- und Anzeigenfeld, eine Schnittstelle zum Anschließen eines Kassetten- oder Tonbandgerätes sowie eine Anwenderschnittstelle.

Der Mikroprozessor 8085 steuert alle Funktionen des micromaster über die Bus-Leitungen. Beim Einschalten des Gerätes wird der Monitor automatisch ab Adresse OOOOH gestartet. Diese "power on jump"-Funktion (springe nach Adresse ..., wenn Gerät eingeschaltet wird) wird mit einer RC-Kombination am Reset-In-Eingang des 8085 ausgelöst.

Alle Leitungen des Mikroprozessors werden auch an der Anwenderschnittstelle zur Verfügung gestellt, so daß beliebige Anwender-Funktionen mit dem Board realisiert werden können. Zu diesem Zweck sind auch die beiden Speicher (RAM und EPROM) auf dem Board durch einfaches Entfernen von Steckbrücken abschaltbar. Es kann dann der gesamte Speicherumfang von 64K extern adressiert werden.

Der RAM-Speicherbereich von 1 KByte dient dem Speichern von Ihren Anwenderprogrammen (Übungsprogrammen) und Ergebnissen, bzw. Zwischenergebnissen (Daten). In dem EPROM ist die Betriebssoftware (Monitor) abgelegt, die von Ihnen später teilweise mit in Ihre Anwendersoftware einbezogen wird, indem Sie bereits programmierte Funktionen in Form von Unterprogrammen aufrufen.

Über das Tasten- und Anzeigefeld wird der Informationsaustausch zwischen dem Mikrocomputer und dem Anwender ermöglicht, ohne daß ein externes



Bauelemente Lageplan

Gerät, wie z.B. ein Datensichtgerät, erforderlich ist. In den 22 Tasten sind

16 Tasten für Hex-Eingabe,

4 Steuertasten,

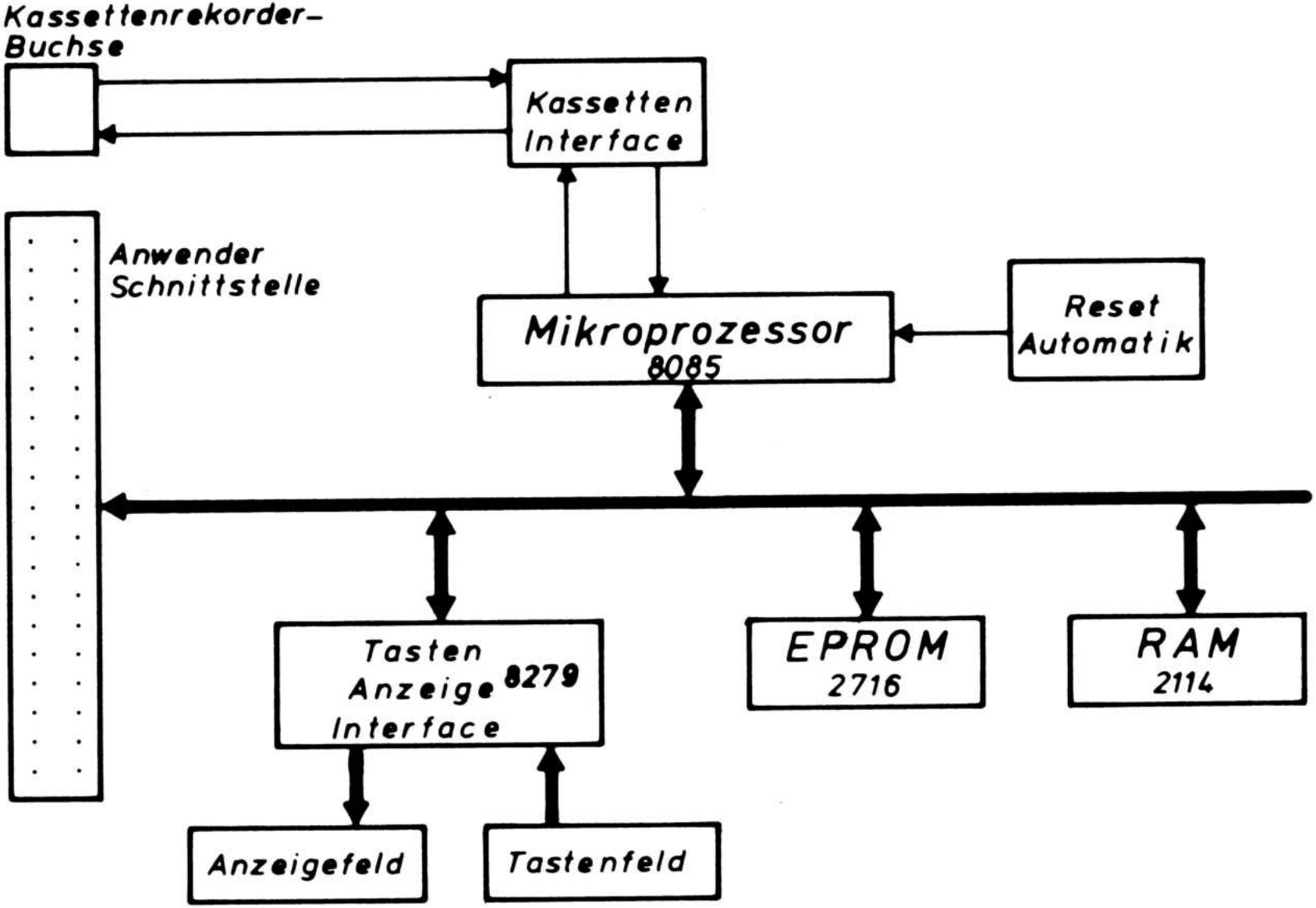
1 Reset-Taste und

1 Interrupt-Taste

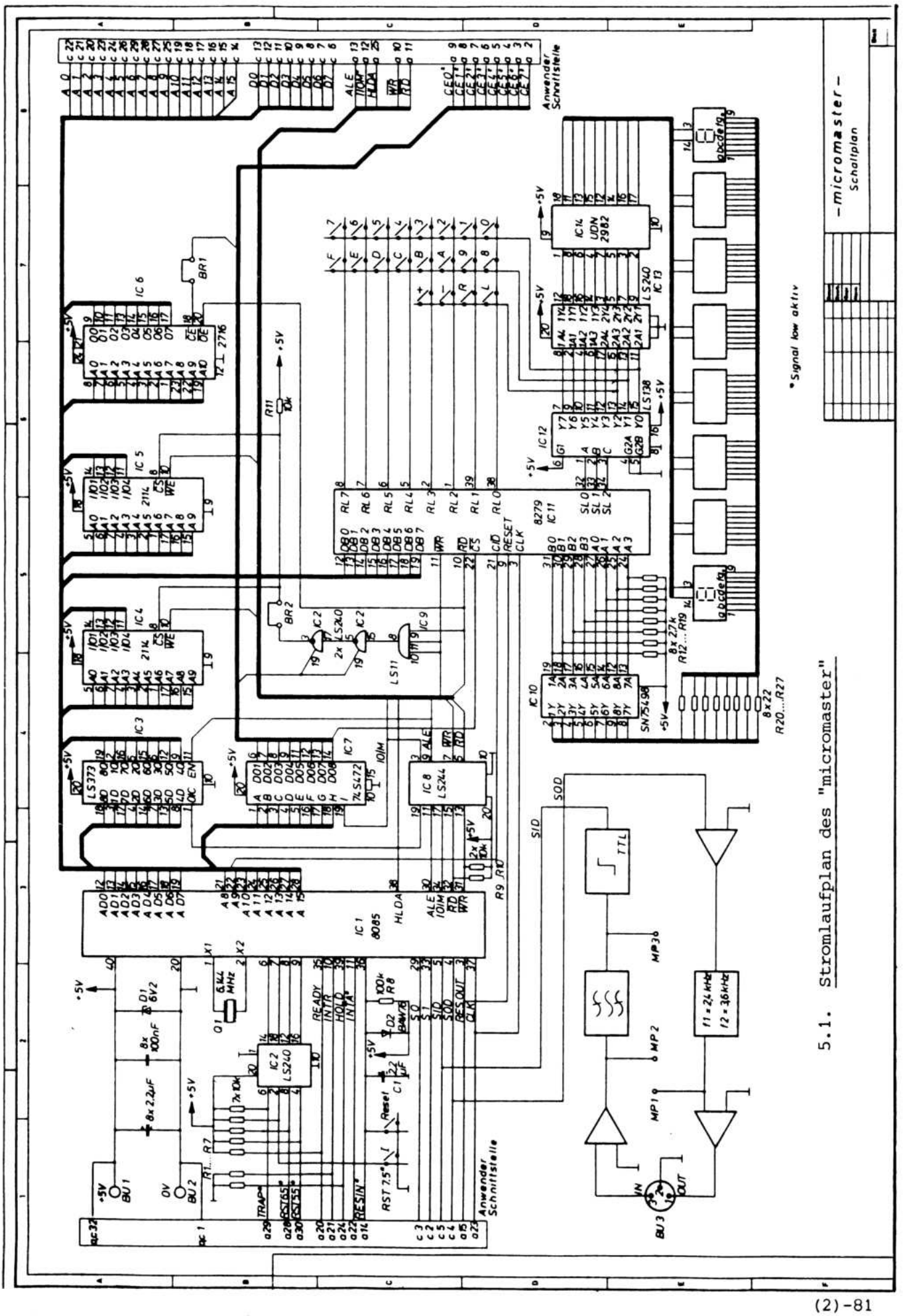
enthalten. Die 8 Anzeigestellen ermöglichen nicht nur die Anzeige von Adressen und Daten, sondern gleichzeitig auch die Ausgabe von symbolischen Abkürzungen. Eine Zusammenstellung ist in einer Liste im Anhang enthalten. Die gesamte Steuerung der Anzeige und Tastatur erfolgt über einen speziellen Schnittstellen-Baustein 8279.

Das Magnetbandgeräte-Interface ermöglicht den Anschluß normaler Kassettenrecorder oder Tonbandgeräte zur Abspeicherung auf ein externes Speichermedium mit praktisch unbegrenzter Kapazität, wobei jedoch der Anwender die Organisation dieses Mediums selbst vornehmen muß. Dazu gehört z.B. das Nummerieren der Kassetten, das Mitschreiben des Bandzählwerkes, falls mehrere Programme auf einer Kassette abgespeichert sind etc.

Im Anhang dieser Lektion befindet sich auch ein Stromlaufplan des micromaster. Es sei an dieser Stelle erwähnt, daß wir auf die Hardware von Mikroprozessoren und deren Systembausteine in der vierten Lektion detaillierter eingehen werden.



Blockschaltbild -micromaster-



5.1. Stromlaufplan des "micromaster"

5.1. Stromlaufplan des "micromaster"

5.2. Befehlsliste des 8080/8085 - alphabetisch geordnet

Assembler		Byte	HEX
ACI	d8	2	CE
ADC	A	1	8F
ADC	B	1	88
ADC	C	1	89
ADC	D	1	8A
ADC	E	1	8B
ADC	H	1	8C
ADC	L	1	8D
ADC	M	1	8E
ADD	A	1	87
ADD	B	1	80
ADD	C	1	81
ADD	D	1	82
ADD	E	1	83
ADD	H	1	84
ADD	L	1	85
ADD	M	1	86
ADI	d8	2	C6
ANA	A	1	A7
ANA	B	1	AO
ANA	C	1	A1
ANA	D	1	A2
ANA	E	1	A3
ANA	H	1	A4
ANA	L	1	A5
ANA	M	1	A6
ANI	d8	2	E6
CALL	a16	3	CD
CC	a16	3	DC
CM	a16	3	FC
CMA		1	2F
CMC		1	3F

Assembler		Byte	HEX
CMP	A	1	BF
CMP	B	1	B8
CMP	C	1	B9
CMP	D	1	BA
CMP	E	1	BB
CMP	H	1	BC
CMP	L	1	BD
CMP	M	1	BE
CNC	a16	3	D4
CNZ	a16	3	C4
CP	a16	3	F4
CPE	a16	3	EC
CPI	d8	2	FE
CPO	a16	3	E4
CZ	a16	3	CC
DAA		1	27
DAD	B	1	O9
DAD	D	1	19
DAD	H	1	29
DAD	SP	1	39
DCR	A	1	3D
DCR	B	1	O5
DCR	C	1	OD
DCR	D	1	15
DCR	E	1	1D
DCR	H	1	25
DCR	L	1	2D
DCR	M	1	35
DCX	B	1	OB
DCX	D	1	1B
DCX	H	1	2B
DCX	SP	1	3B

Assembler		Byte	HEX
DI		1	F3
EI		1	FB
HLT		1	76
IN	a8	2	DB
INR	A	1	3C
INR	B	1	04
INR	C	1	0C
INR	D	1	14
INR	E	1	1C
INR	H	1	24
INR	L	1	2C
INR	M	1	34
INX	B	1	03
INX	D	1	13
INX	H	1	23
INX	SP	1	33
JC	a16	3	DA
JM	a16	3	FA
JMP	a16	3	C3
JNC	a16	3	D2
JNZ	a16	3	C2
JP	a16	3	F2
JPE	a16	3	EA
JPO	a16	3	E2
JZ	a16	3	CA
LDA	a16	3	3A
LDAX	B	1	0A
LDAX	D	1	1A
LHLD	a16	3	2A
LXI	B,d16	3	01
LXI	D,d16	3	11
LXI	H,d16	3	21
LXI	SP,d16	3	31
MOV	A,A	1	7F

Assembler		Byte	HEX
MOV	A,B	1	78
MOV	A,C	1	79
MOV	A,D	1	7A
MOV	A,E	1	7B
MOV	A,H	1	7C
MOV	A,L	1	7D
MOV	A,M	1	7E
MOV	B,A	1	47
MOV	B,B	1	40
MOV	B,C	1	41
MOV	B,D	1	42
MOV	B,E	1	43
MOV	B,H	1	44
MOV	B,L	1	45
MOV	B,M	1	46
MOV	C,A	1	4F
MOV	C,B	1	48
MOV	C,C	1	49
MOV	C,D	1	4A
MOV	C,E	1	4B
MOV	C,H	1	4C
MOV	C,L	1	4D
MOV	C,M	1	4E
MOV	D,A	1	57
MOV	D,B	1	50
MOV	D,C	1	51
MOV	D,D	1	52
MOV	D,E	1	53
MOV	D,H	1	54
MOV	D,L	1	55
MOV	D,M	1	56
MOV	E,A	1	5F
MOV	E,B	1	58
MOV	E,C	1	59

Assembler	Byte	HEX
MOV E,D	1	5A
MOV E,E	1	5B
MOV E,H	1	5C
MOV E,L	1	5D
MOV E,M	1	5E
MOV H,A	1	67
MOV H,B	1	60
MOV H,C	1	61
MOV H,D	1	62
MOV H,E	1	63
MOV H,H	1	64
MOV H,L	1	65
MOV H,M	1	66
MOV L,A	1	6F
MOV L,B	1	68
MOV L,C	1	69
MOV L,D	1	6A
MOV L,E	1	6B
MOV L,H	1	6C
MOV L,L	1	6D
MOV L,M	1	6E
MOV M,A	1	77
MOV M,B	1	70
MOV M,C	1	71
MOV M,D	1	72
MOV M,E	1	73
MOV M,H	1	74
MOV M,L	1	75
MVI A,d8	2	3E
MVI B,d8	2	06
MVI C,d8	2	0E
MVI D,d8	2	16
MVI E,d8	2	1E

Assembler	Byte	HEX
MVI H,d8	2	26
MVI L,d8	2	2E
MVI M,d8	2	36
NOP	1	00
ORA A	1	B7
ORA B	1	B0
ORA C	1	B1
ORA D	1	B2
ORA E	1	B3
ORA H	1	B4
ORA L	1	B5
ORA M	1	B6
ORI d8	2	F6
OUT a8	2	D3
PCHL	1	E9
POP B	1	C1
POP D	1	D1
POP H	1	E1
POP PSW	1	F1
PUSH B	1	C5
PUSH D	1	D5
PUSH H	1	E5
PUSH PSW	1	F5
RAL	1	17
RAR	1	1F
RC	1	D8
RET	1	C9
RIM	1	20
RLC	1	07
RM	1	F8
RNC	1	D0
RNZ	1	C0
RP	1	F0

Assembler	Byte	HEX
RPE	1	E8
RPO	1	EO
RRC	1	OF
RST O	1	C7
RST 1	1	CF
RST 2	1	D7
RST 3	1	DF
RST 4	1	E7
RST 5	1	EF
RST 6	1	F7
RST 7	1	FF
RZ	1	C8
SBB A	1	9F
SBB B	1	98
SBB C	1	99
SBB D	1	9A
SBB E	1	9B
SBB H	1	9C
SBB L	1	9D
SBB M	1	9E
SBI d8	2	DE
SHLD a16	3	22
SIM	1	3O
SPHL	1	F9
STA a16	3	32
STAX B	1	O2
STAX D	1	12
STC	1	37
SUB A	1	97
SUB B	1	9O
SUB C	1	91
SUB D	1	92

Assembler	Byte	HEX
SUB E	1	93
SUB H	1	94
SUB L	1	95
SUB M	1	96
SUI d8	2	D6
XCHG	1	EB
XRA A	1	AF
XRA B	1	A8
XRA C	1	A9
XRA D	1	AA
XRA E	1	AB
XRA H	1	AC
XRA L	1	AD
XRA M	1	AE
XRI d8	2	EE
XTHL	1	E3

d8 = 8 Bit-Daten
d16 = 16 Bit-Daten

a8 = 8 Bit-Kanal-Adresse
a16 = 16 Bit-Speicher-Adresse

5.3. Liste der symbolischen Anzeigeabkürzungen

In der folgenden Liste sind Buchstaben teilweise klein geschrieben, d.h. den symbolischen Namen finden Sie so, wie er in der Anzeige erscheint. Der Grund für das Kleinschreiben liegt in den beschränkten Darstellungsmöglichkeiten mit 7-Segment-Anzeigen.

a. Symbolische Namen für Register-Paare

A F	Akkumulator und Flag-Register (Zustandswort)
b C	BC-Registerpaar
d E	DE-Registerpaar
H L	HL-Registerpaar
P C	Programmzähler (program counter)
S P	Stapelzeiger (stack pointer)
I S	Interrupt-Statuswort

b. Weitere symbolische Namen

A A	Anfangsadresse
A U	Anfangsadresse des umzurechnenden Bereiches
b A	Bestimmungsadresse
E A	Endadresse
E U	Endadresse des umzurechnenden Bereiches
d B	Datenbyte
H 1	Unterbrechungsadresse (Halt 1)
H 2	Unterbrechungsadresse (Halt 2)
P A	Peripherieadresse
S A	Startadresse

5.4.

MZ	Hex-Code	Adresse	Name	Op-Code	Operand	Bemerkungen	A	B	C	D	E	H

K 6. Lösungen zu den Aufgaben in dieser Lektion

Aufgabe Seite 10:

91: Falsch, weil ein Name mit einem Buchstaben oder Sonderzeichen beginnen muß.

MARKE Falsch, weil der Doppelpunkt nach dem Namen fehlt

Aufgabe Seite 24:

```
          0 1    0 0 0    0 1 0

Operationscode      Ziel          Ursprung
für MOV             Register B    Register C

also        MOV  B,C  ; Lade Register B mit In-
                      ; halt von Register C
```

A 7.

Hausaufgaben zur Lektion 2

Name:

Anschrift:

Teiln.-Nr.:

Korrektur-Rand
Nicht beschreiben

1. Wieviel Bits enthält ein 3-Byte langer Befehl?

2. Welches ist der wesentliche Unterschied zwischen einem Interpreter und einem Übersetzer?

3. Muß man bei der Programmierung eines zeitkritischen Problems eine Programmiersprache auf höherer oder niederer Ebene wählen?

4. Aus welchen Feldern kann eine Befehlszeile bei der Assembler-Programmierung bestehen?

5. Was geschieht beim Assemblieren mit einem Namen im Markenfeld?

Korrektur-Rand
Nicht beschreiben

6. Welche beiden Möglichkeiten gibt es für die Verwendung der Register im Prozessor?

7. Wie wird das erste Byte eines Befehls bezeichnet und wie heißen die evtl. ein oder zwei folgenden Bytes?

8. Nennen Sie drei wichtige Befehlstypen für den 8080/8085.

9. Wieviel mögliche MOV Register-Register-Befehle gibt es, wenn Sie alle Register in Betracht ziehen?

Korrektur-Rand
Nicht beschreiben

10. Welche mathematischen Funktionen werden zu den arithmetischen Operationen gezählt?

11. Welches Ergebnis steht im Akkumulator, wenn der Befehl SUB A ausgeführt wird?

12. Welche logischen Grundfunktionen können im 8080/8085 ausgeführt werden?

13. Erklären Sie den Unterschied zwischen einem bedingten und unbedingten Sprung.

Korrektur-Rand
Nicht beschreiben

14. Führen Sie nochmals die Übung mit Kommando 1 aus. Schreiben Sie dann eine beliebige Konstante in den Speicherbereich von OOOOH bis OOFFH. Kontrollieren Sie den Speicherbereich mit Kommando O. Was stellen Sie fest? Geben Sie eine kurze Begründung für das Ergebnis.

Korrektur-Rand
Nicht beschreiben

15. Verschieben Sie mit Kommando 2 den Speicherbereich 07E3H bis 07FFH in den Speicher ab 1800H. Starten Sie den kopierten Bereich mit Kommando 7 ab Adresse 1800H. Was passiert?

16. Nach wieviel Ausführungen des Befehls RAL wird der ursprüngliche Stand im Akkumulator wieder hergestellt?

Zensierung:

Lehrgangsleiter: