

SKRIPTUM

PROGRAMMIEREN VON MIKROCOMPUTERN

CPU 6502

G. Eisenack

Bei kleinen Systemen nicht unbedingt erforderlich, aber meist sehr nützlich, sind Externspeicher. Im einfachsten Fall wird ein handelsüblicher Kassettenrecorder verwendet.

1.2 Zentraleinheit eines Mikrocomputers

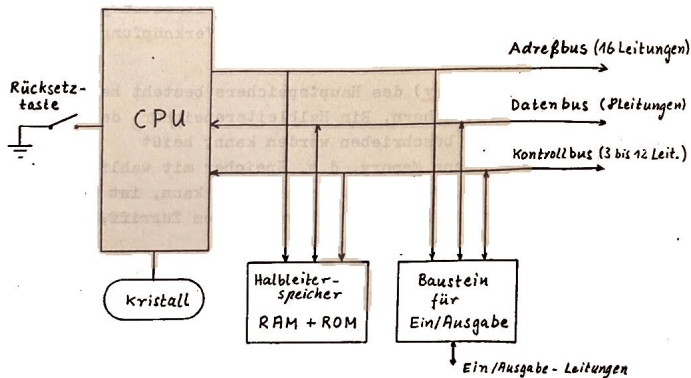
Mikroprozessoren sind Prozessoren (Prozessor = Leitwerk + Rechenwerk = CPU), die in einer - oder sehr wenigen - integrierten Schaltung(en) realisiert sind.

Mikrocomputer = Computer, dessen Prozessor ein Mikroprozessor ist.

Heute wird meistens mit Mikroprozessoren gearbeitet, die eine Wortlänge von 8 bit = 1 Byte und 16 bit-Adressen haben. Zu diesen Mikroprozessoren gehören z.B. die "großen Vier":

- 8080/85 von Intel
- 6800 von Motorola
- 6502 von MOS-Technology
- Z80 von Zilog

Die Zentraleinheit mit einem solchen Mikroprozessor sieht nun folgendermaßen aus:



Die 8 bit-CPU hat 8 Anschlüsse (Pins) für die Datenleitungen, die zusammen den sogenannten Datenbus ergeben. An diese Leitungen werden die Speicher (ROM + RAM) und Bausteine für die Ein-/Ausgabe angeschlossen.

Die CPU hat ferner 16 Anschlüsse für den "Adreßbus", an den ebenfalls die Speicher und eventuell auch die Ein-/Ausgabe-Bausteine angeschlossen werden.

Über den Kontrollbus, der bei den verschiedenen Mikroprozessoren auch recht verschiedene Kontrollinformationen überträgt, werden von der CPU einige Kontrollsignale ausgesendet und manchmal auch empfangen.

Typisches Beispiel für eine Kontrollinformation ist die Mitteilung der CPU an den Hauptspeicher, ob die CPU lesen oder schreiben möchte (R/W - Signal).

Der eingezeichnete Kristall dient zur Definition der Taktfrequenz (z.B. 1 MHz) und die Rücksetztaste ist erforderlich, um die CPU in einen definierten Anfangszustand zu versetzen.

Ganze Zentraleinheiten mit beschränktem Speicher (typisch: 2 kB ROM + 64 Byte RAM) werden heute von vielen Herstellern als Ein-Chip-Mikrocomputer angeboten.

Sofern der ROM-Bereich nicht programmierbar ist - und das ist in der Regel der Fall - wird das Programm beim Herstellungsprozeß des Ein-Chip-Mikrocomputers in den ROM-Bereich gebracht. Dies ist bei Verwendung geringer Stückzahlen ziemlich teuer.

Es gibt aber inzwischen auch Ein-Chip-Mikrocomputer bei denen das ROM durch ein EPROM ersetzt ist (z.B. Intel 8748, 1 kB EPROM + 64 byte RAM, Preis 1979: ca. 170 DM).

Und nun noch einige Bemerkungen zum Mikroprozessor-Markt. Der erste Mikroprozessor "Intel 4004" wurde 1971 auf den Markt gebracht. Er arbeitete mit einer Wortlänge von 4 bit. Es folgte der 8 bit-Mikroprozessor 8008 und 1974 der bekannte 8080 von Intel. Seit 1975 sind auch 16 bit-CPUs erhältlich.

1978 wurden bereits 14 Millionen Mikroprozessoren verkauft. 13 Millionen davon im 4 bit-Bereich und 1 Million im * 8 bit-Bereich. Die Preise für 8 bit-Mikroprozessoren liegen je nach Stückzahl zwischen 5 DM und 50 DM (1979).

1.3 Betriebssystem

Jeder Computer, in den Programme eingegeben werden sollen, benötigt bereits ein Programm, mit dem man Programme eingeben kann. Dieses Eingabeprogramm muß in einem Festspeicher stehen (heute im ROM, früher fest verdrahtet).

Nach dem Drücken der Rücksetztaste (Reset) wird die Anfangsadresse des Eingabe-Programms in den Befehlszähler geladen und der Computer beginnt zu laufen (Systemstart). Ein laufendes Programm kann jederzeit mit der Rücksetztaste unterbrochen werden; anschließend läuft das Eingabe-Programm.

Dieses Eingabe-Programm wird meist zusammen mit anderen nützlichen Programmen in einem ROM bzw. PROM gespeichert und bildet mit diesen ein kleines Betriebssystem. Dieses kleine Betriebssystem wird häufig Monitor genannt. Der Monitor gestattet es, Speicherzellen anzuzeigen und deren Inhalte zu verändern (Programmeingabe). Ferner kann der Befehlszähler verändert werden (Programmstart).

Ist ein Externspeicher (z.B. Kassettenrecorder oder Floppy disk) vorgesehen, so enthält der Monitor auch Programme zum Lesen und Abspeichern von Programmen. Bei größeren Systemen wird der Monitor durch weitere Programme, die auf einem Externspeicher stehen, zu einem umfangreicheren Betriebssystem ergänzt.

Beispiel: Beim KIM-Mikrocomputer besteht das "Betriebssystem" aus 2 kB Programmen mit folgenden Fähigkeiten:

* Quelle: Zeitschrift Byte 7/79, p.99. Danach wurden 1978 am meisten verkauft: 4 bit TMS 1000 (Ein-Chip-Mikrocomputer)
8 bit 6502

Anzeigen und Ändern von Zellen

Starten von Programmen

Ausgeben von Speicherbereichen auf Kassette oder Lochstreifen

Einlesen von Speicherbereichen von Kassette oder Lochstreifen

Einzelschrittdurchführung von Programmen

Bemerkung 1: Soll der Mikrocomputer nur für einen Zweck eingesetzt werden, wird kein Eingabe-Programm benötigt. Das erforderliche Programm steht im ROM und wird durch Drücken der Rücksetztaste gestartet (oder durch Einschalten des Geräts).

Bemerkung 2: Der Zusammenbau eines Mikrocomputers (in den Programme eingegeben werden sollen) aus seinen Einzelkomponenten ist jedenfalls für das erste Exemplar ein echtes Problem, da ja zum Arbeiten mit dem Mikrocomputer ein Eingabe-Programm erforderlich ist (das Eingabe-Programm kann also auf dem zusammengebauten System nicht getestet werden).

Deshalb bieten einige Hersteller ein ROM an, in dem ein bereits ausgetesteter Monitor abgespeichert ist. Der TIM-Monitor für den 6502 und Mikbug für den 6800 sind hierfür Beispiele.

2. Grundkonzepte der Programmierung

Wir werden uns bei der Programmierung mit dem Mikroprozessor 6502 beschäftigen, der sich besonders durch seinen sehr übersichtlichen Befehlsvorrat auszeichnet. In den Grundkonzepten sind die Mikroprozessoren aber sehr ähnlich. Bei größeren Abweichungen zu anderen Mikroprozessoren (es werden 6800, 8080, Z80 berücksichtigt) werden Anmerkungen gemacht.

2.1 Befehlsstruktur von Mikroprozessoren (8 bit)

Die Wortlänge der 8 bit-Mikroprozessoren ist - wie der Name sagt - 8 bit, also:

Wortlänge = 8 bit = 1 Byte

Für die Adressen werden üblicherweise 16 bit verwendet:

Adresse = 16 bit = 2 Bytes

Somit können mit diesen Mikroprozessoren 2^{16} Bytes = 64 kB adressiert werden (Siemens 305: 16 kWorte = 48 kB).

Den Inhalt eines Bytes gibt man sedezimal (=hexadezimal) an,

z.B. $8A = 10001010$

Auch Adressen werden sedezimal geschrieben, z.B.

OE5F

Befehle im Mikrocomputer haben keine feste Länge. Es gibt 1 Byte-, 2 Byte- und 3 Byte-Befehle (beim Z80 auch noch 4 Byte-Befehle).

Beispiele für Befehle

1. a) Lade den Akkumulator mit dem Inhalt einer Zelle (LDA)

$\underbrace{AD}_{\text{Op-Code}} \underbrace{5F \quad OE}_{\text{Adreßteil (gibt Adresse OE5F an)}} \quad 3\text{-Byte-Befehl}$

AD ist der Operationscode, also der Operationsteil des Befehls.

Die 16 bit-Adresse der Zelle wird eigenartigerweise in umgekehrter Reihenfolge im Befehl angegeben.

b) Lade den Akkumulator mit der Sedezimalzahl 6D (LDA)

$\underbrace{A9}_{\text{Op-Code}} \underbrace{6D}_{\text{Sedezimalzahl}} \quad 2 \text{ Byte-Befehl}$

2. Verschiebe Inhalt des Akkus um eine Stelle nach links (ASL)

$\underbrace{0A}_{\text{Operationscode}} \quad 1 \text{ Byte-Befehl}$

Die Operationscodes sind der Befehlstabelle für den 6502 (Anhang) zu entnehmen:

In der Zeile LDA - Abkürzung für Load Accumulator - sind die Operationscodes AD (in Spalte "absolute") und A9 (in Spalte "immediate") zu finden.

In der Zeile ASL - Abkürzung für Arithmetic Shift Left - der Operationscode OA (in Spalte "accumulator").

Die große Anzahl von Spalten ergibt sich durch die Vielzahl von Adressierungsmöglichkeiten, auf die wir später eingehen werden.

Die 8 bit-Mikroprozessoren und auch die meisten 16 bit-Mikroprozessoren sind Ein-Adreß-Maschinen, d.h. im Befehlsword wird höchstens ein Operand, der im Hauptspeicher steht, adressiert. Falls im Befehl ein zweiter Operand erforderlich ist (z.B. bei der Addition), steht dieser entweder im Akkumulator oder in einem andren Register der CPU.

Bemerkung: Die Eigenart, im Adreßteil des Befehls zunächst das niederwertige, dann das höherwertige Byte der Adresse anzugeben, teilt der 6502 mit den Mikroprozessoren 8080 und Z80. Beim 6800 wird die Adresse dagegen "normal" angegeben.

2.2 Einführende Programme (Addition)

Zur Einführung soll ein Programm geschrieben werden, welches zwei Hex-Zahlen (=Hexadezimal-Zahlen = Sedezimal-Zahlen) addiert.

Erinnern wir noch einmal an eine Dualaddition

```

z.B.      11001010
          11100101
          11010111
          ↑
          Übertrag = Carry

```

Das Carry-Bit wird bei der Addition in einen Spezialspeicher der CPU eingetragen (Carry-Bit = 1, falls Übertrag; Carry-Bit = 0, falls kein Übertrag) und bei der nächsten Addition noch zuaddiert. Soll also durch das Carry-Bit kein Fehler bei der Addition entstehen, so ist vor der Addition Carry-Bit = 0 zu setzen. Dies geschieht durch den Befehl CLC.

Aufgabe: 1. Summand in 0000 (0000 Adresse einer Speicherzelle)
 2. Summand in 0001
 Summe nach 0002

Wir wollen das Programm, beginnend bei Adresse 0200 (Angabe in Hex!) abspeichern.

Adresse	Befehl	Symbol
0200	18	CLC (Clear Carry)
0201	AD 00 00	LDA (Load Accumulator)
0204	6D 01 00	ADC (Add with Carry)
0207	8D 02 00	STA (Store Accumulator)
020A	4C 4F 1C	JMP (Jump)

Nach dem Löschen des Carry-Bits wird der 1. Summand in den Akkumulator geladen. Anschließend der 2. Summand zum Akkumulator (dual) addiert, wobei das Ergebnis im Akkumulator steht. Deshalb wird durch STA der Akkumulator-Inhalt in die Ergebnis-Zelle gebracht. Der letzte Befehl ist ein Sprung in das Eingabe-Programm des Monitors.

Das Programm wird mit Hilfe des Monitors eingegeben; ebenfalls die beiden Summanden. Anschließend wird das Programm mit Hilfe des Monitors bei Adresse 0200 gestartet..

Nach Durchlauf des Programms kann der Monitor wieder bedient werden, da ja in das Eingabe-Programm zurückgesprungen wurde.

Beim Fehlen des Sprungs in das Eingabe-Programm würden nach STA völlig unkontrollierbare Befehle ablaufen (das System "bricht zusammen") und meist ist nur mit der Rücksetztaste aus diesem Zustand herauszukommen.

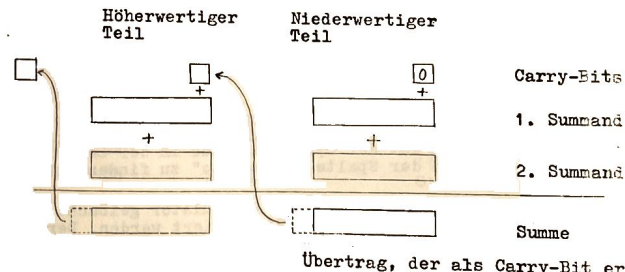
Im zweiten Programmbeispiel wollen wir Zahlen, die sich in 2 Bytes darstellen lassen (0-65535) addieren. Wie im ersten Beispiel soll ein Überschreiten des Zahlenbereichs unberücksichtigt bleiben.

- 2-Byte-Addition
1. Summand in 0010,0011
 2. Summand in 0012,0013
- Summe nach 0014,0015

Adresse	Maschinen-Befehl	Symbolischer Befehl
0300	18	CLC
0301	AD 11 00	LDA #0011
0304	6D 13 00	ADC #0013
0307	8D 15 00	STA #0015
030A	AD 10 00	LDA #0010
030D	6D 12 00	ADC #0012
0310	8D 14 00	STA #0014
0313	4C .. .	JMP #.... Sprung in Monitor

In den ersten vier Befehlen wird der niederwertige Teil der Summanden addiert und die Summe abgespeichert. Anschließend werden die höherwertigen Teile der Summanden addiert, wobei ein Übertrag berücksichtigt wird.

Folgendes Bild soll die 2 Byte-Addition verdeutlichen:



Wir geben bereits jetzt die Befehle auch in symbolischer Form an, so wie sie später von einem Assembler bearbeitet werden können. § deutet an, daß die Adresse als Hex-Zahl angegeben wird.

Häufig möchte man Dezimalzahlen und nicht Dualzahlen verarbeiten. Dazu dient die BCD-Darstellung (Binary Coded Decimal Code) von Dezimalzahlen. Jede Dezimalziffer wird als 4 Bit-Dualzahl verschlüsselt. In einem Byte lassen sich also 2-stellige Dezimalzahlen darstellen.

Beispiel: 93 wird durch 10010011, also durch die Hex-Zahl 93 dargestellt.

Um Dezimalzahlen in BCD-Darstellung addieren bzw. subtrahieren zu können, muß das Rechenwerk auf "Dezimalarithmetik" umschalten. Dies geschieht mit dem Befehl SED (Set Decimal Mode) und wird mit dem Befehl CLD (Clear Decimal Mode) rückgängig gemacht. Das Rechenwerk kann nur für die Addition und die Subtraktion auf Dezimalarithmetik umgeschaltet werden.

Dezimaladdition 1. Summand in 0000
2. Summand in 0001
Summe in 0002,0003

In diesem Beispiel soll das Überschreiten des 2-ziffrigen Zahlenbereichs berücksichtigt werden.

0200	F8	SED	(Set Decimal Mode)
0201	18	CLC	
0202	AD 00 00	LDA §0000	
0205	6D 01 00	ADC §0001	
0208	AD 03 00	STA §0003	
020E	A9 00	LDA #00	(lösche Akku)
020D	69 00	ADC #00	(Addiere 00 zum Akku)
020F	8D 02 00	STA §0002	
0212	D8	CLD	(Clear Decimal Mode)
0213	4C . . .	JMP §....	Sprung in Monitor

#00 bedeutet, daß 00 nicht als Adresse des Operanden zu deuten ist, sondern der Operand (als Hex-Zahl) selbst ist. In solchen Fällen ist der Operationscode in der Befehlsliste des 6502 in der Spalte "Immediate" zu finden.

LDA #00
ADC #00 bewirkt also, daß der Akkumulator gelöscht wird und zu diesem 00 und das Carry-Bit addiert werden. Der Akkumulator enthält somit einen eventuellen Übertrag.

Bemerkung: Bei den Mikroprozessoren 8080 und Z80 wird die Operandenadresse (z.B. bei der Addition) in der Regel nicht direkt im Befehl angegeben, sondern muß vorher in ein Registerpaar der CPU eingegeben werden.

2. Für die 2 Byte-Addition ist beim 8080 und Z80 ein Befehl vorhanden.

3. Die Dezimaladdition bei den Mikroprozessoren 8080, Z80 und 6800 wird nicht durch Umschalten des Rechenwerks erreicht, sondern nach einer Dualaddition wird durch einen Sonderbefehl (Decimal Adjust Accumulator) das Ergebnis korrigiert.

2.3 Logische Befehle, Schiebepfehle

Logische Befehle

Die logischen Grundoperationen Konjunktion (= UND = AND), Disjunktion (= ODER = OR) und Antivalenz (= exklusives ODER) sind im Befehlssatz fast aller Mikroprozessoren enthalten, denn durch Verknüpfung dieser Grundoperationen läßt sich jede logische Funktion herstellen.

Die Wahrheitstabellen für diese Grundfunktionen sind:

	UND	ODER	Exklusives ODER
	$\begin{matrix} & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{matrix}$	$\begin{matrix} & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{matrix}$	$\begin{matrix} & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{matrix}$

Nehmen wir an

Akkumulatorinhalt: 10101101
Inhalt der Zelle 3E4F: 11001001

, so ersieht man die bitweise Wirkung der Befehle AND, ORA (OR with Accumulator) und BOR (Exclusive OR) aus folgender Tabelle:

Maschinenbefehl	symb. Befehl	Ergebnis im Akkumulator
2D 4F 3E	AND §3E4F	10001001
0D 4F 3E	ORA §3E4F	11101101
4D 4F 3E	BOR §3E4F	01100100

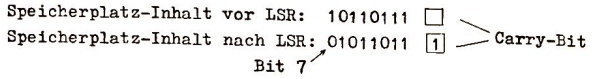
Die Negation des Akkumulatorinhalts (= 1er-Komplement = Vertauschen von Nullen und Einsen) erreicht man durch:

EOR #FF (Maschinenbefehl: 49 FF)

Schiebepfeile(Shift-Befehle), Rotationen

1. Rechtsverschiebung LSR (Logical Shift Right)

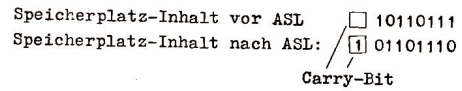
Bei der logischen Rechtsverschiebung wird der Inhalt der adressierten Speicherzelle um 1 Bit nach rechts verschoben, wobei von links eine 0 nachrückt und das herausgeschobene Bit im Carry-Bit aufgefangen wird, z.B.:



Bemerkung: a) Rechtsverschiebung bewirkt bei Dualzahlen eine Division durch 2.
 b) Im Gegensatz zur logischen Rechtsverschiebung wird bei der "arithmetischen Rechtsverschiebung" Bit 7 dem Bit 6 angeglichen. Dies entspricht der Division durch 2 bei Dualzahlen mit Vorzeichen. (Beim 6502 und 8080 nicht vorhanden).

2. Linksverschiebung ASL (Arithmetic Shift Left)

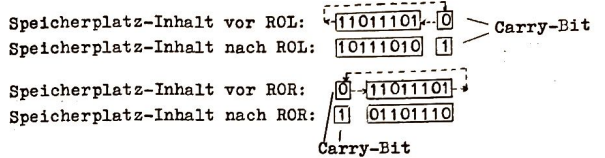
Die logische Linksverschiebung stimmt mit arithmetischen Linksverschiebung überein. Die Wirkung von ASL ist analog zur Wirkung von LSR; z.B.:



Bemerkung: Linksverschiebung bewirkt bei Dualzahlen die Multiplikation mit 2.

3. Links-Rotation ROL (Rotate Left) und Rechts-Rotation ROR

Die Wirkung der Rotationsbefehle auf den Inhalt einer Speicherstelle und das Carry-Bit erkennt man an folgenden Beispielen:



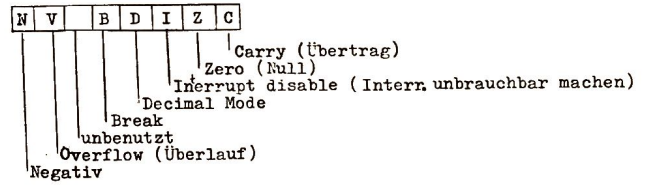
Beispiel: In den Zellen 1200,1201,1202 stehe eine Dualzahl a mit $a < 2^{23}$. a soll verdoppelt werden.

```

0200 0E 02 12      ASL  $\$1202$ 
0203 2E 01 12      ROL  $\$1201$ 
0206 2E 00 12      ROL  $\$1200$ 
0209 4C .. ..      JMP  $\$....$  Sprung in den Monitor
  
```

2.4 Flags und Statusregister

Bei zahlreichen Befehlen werden außer der Durchführung der eigentlichen Operation noch "Flags" (Fahnen) gesetzt; das sind Bits, die gewisse Zustände anzeigen sollen. Die Flags werden in dem sogenannten Status-Register der CPU geführt.



Die Bedeutung der meisten Flags kann jetzt noch nicht besprochen werden. Die Carry-Flag und die Dezimal-Flag haben wir bereits kennen gelernt. Die Dezimal-Flag wurde durch SED auf 1 und durch CLD auf 0 gesetzt. Die Carry-Flag wurde automatisch nach einem Additionsbefehl - je nach Ergebnis - auf 0 oder 1 gesetzt. Welche Flags durch einen Befehl verändert werden, ist in der Befehlsliste des 6502 zu finden.

Beispiel: Bei dem Befehl LDA (Load Accumulator) wird

- a) Z = 1 gesetzt, falls 0 in den Akkumulator gebracht wurde, sonst wird Z = 0 gesetzt.
- b) N = 1 gesetzt, falls Zahl im Akkumulator negativ (d.h. Bit 7, das höchstwertigste Bit ist 1), sonst wird N = 0 gesetzt.
- c) Alle weiteren Flags bleiben unverändert.

Es gibt auch Befehle die nur Flags verändern:

CLC	C = 0	(Clear Carry)
SEC	C = 1	(Set Carry)
CLD	D = 0	(Clear Decimal Mode)
SED	D = 1	(Set Decimal Mode)
CLI	I = 0	(Clear Interrupt Disable Flag)
SEI	I = 1	(Set Interrupt Disable Flag)
CLV	V = 0	(Clear Overflow Flag)

Zu den Befehlen, die ebenfalls nur Flags verändern, gehören noch BIT (mit dem man gewisse Bits von Speicherzellen kontrollieren kann) und die wichtigen Vergleichsbefehle.

Der Vergleichsbefehl CMP (Compare) setzt die Flags auf folgende Weise:

CMP M bewirkt:

$$\begin{cases} Z=1, & \text{falls (Akku) - (M) = 0} \\ Z=0, & \text{falls (Akku) - (M) \neq 0} \\ C=1, & \text{falls (Akku) \ge (M)} \\ C=0, & \text{falls (Akku) < (M)} \end{cases}$$

(Akku) bedeutet "Inhalt des Akkus" und (M) analog "Inhalt der Speicherzelle M".

Flags können nun abgefragt werden. Dies geschieht mit bedingten Sprüngen, die im nächsten Abschnitt behandelt werden.

Bemerkung: Das Flagkonzept wird wohl bei allen Mikroprozessoren verwendet. Unterschiede gibt es in der Art und in der Behandlung der Flags. C, Z, N sind meist vorhanden.

2.5 Sprünge und Unterprogramme

Unbedingter Sprung (Jump)

Bei diesem Sprung wird, ohne auf eine Bedingung zu achten, gesprungen (vgl. GO TO in FORTRAN). Die Sprungadresse kann direkt oder indirekt angegeben werden. Wir werden die Befehle an Beispielen erläutern.

a) Direkt (absolute)

Beispiel:

4C 03 04 Sprünge nach 0403
 symbolisch: JMP \$0403 (Jump)

b) Indirekt

Beispiel:

6C 03 04 Sprünge zur Adresse, die in
 0403,0404 steht.
 symbolisch: JMP (\$0403)

Bedingte Sprünge (Branches)

In Abhängigkeit vom Zustand gewisser Flags wird gesprungen:

BCC	Sprung, falls C = 0	(Branch on Carry Clear)
BCS	Sprung, falls C = 1	(Branch on Carry Set)
BEQ	Sprung, falls Z = 1	(Branch on Equal Zero)
BNE	Sprung, falls Z = 0	(Branch on Not Equal Zero)
BMI	Sprung, falls N = 1	(Branch on Minus)
BPL	Sprung, falls N = 0	(Branch on Plus)
BVC	Sprung, falls V = 0	(Branch on Overflow Clear)
BVS	Sprung, falls V = 1	(Branch on Overflow Set)

Diese Sprünge sind relative Sprünge, d.h. die Adresse wird nicht direkt (absolute), sondern als Distanz zum Befehlszählerstand angegeben.

