Befehlssatz des 6510

Hier finden Sie, alphabetisch geordnet, eine Auflistung aller bekannten Befehle des C64-Prozessors. Dazu gehören auch die »illegalen Opcodes«.

Zuerst ein Wort zu den illegalen Opcodes, die in Tabelle 1 enthalten sind:
Seit Erscheinen des C 64 vor ungefähr drei Jahren sind einige verschiedene Versionen des Prozessors 6510 gebaut worden. Diese sind untereinander voll kompatibel, was den normalen Befehlssatz aus Tabelle 2 anbetrifft. Die illegalen Opcodes jedoch laufen nicht auf allen Versionen der CPU 6510. Welche Befehle auf welchem Computer eine korrekte Ausführung bewirken, läßt sich nur durch Ausprobieren feststellen. Äußerst hilfreich dabei ist der SMON aus dieser Ausgabe: Er zeigt einen illegalen Opcode nicht wie die meisten Maschinensprachmonitore durch drei Fragezeichen an, sondern disassembliert den Befehl mit den in Tabelle 1 genannten Abkürzungen. Ein vorangestelltes Sternchen (*) kennzeichnet bei SMON den Befehl als illegalen Opcode (zum Beispiel *AXS).

In Tabelle 3 finden Sie eine Übersicht über die in den beiden anderen Tabellen verwendeten Abkürzungen.

(tr)

■A11 : AND register with $11
Das X- beziehungsweise Y-Register wird mit $11 AND-verknüpft und das Ergebnis X- beziehungsweise Y-indiziert abgelegt

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:
A11 OP,X	9C	ABX	3
A11 OP,Y	9E	ABY	3

■AAX : AND akku with X-Register and store akku
Entspricht befehlsfolge:
AND zwischen Akku und X-Register
STA

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:
AAX #OP	8B	IM	2
AAX OP	87	ZP	2
AAX OP,Y	97	ZPY	2
AAX OP	8F	ABS	3
AAX (OP,X)	83	(OP,X)	2

■ASR : AND with akku and shift right
Entspricht Befehlsfolge:
AND
LSR

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:
ASR #OP	6B	IM	2

■ARR : AND with akku and rotate right
Entspricht Befehlsfolge:
AND
ROR

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:
ARR #OP	4b	IM	2

■AXS : AND akku and X-register and subtract from data
Der Wert wird von dem Ergebnis der AND-Verknüpfung zwischen Akku und X-Register subtrahiert und in das X-Register geschrieben.

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:
AXS #OP	CB	IM	2

■DCP : decrement and compare with akku
Entspricht Befehlsfolge:
DEC
CMP

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:
DCP OP	C7	ZP	2
DCP OP,X	D7	ZPX	2
DCP OP	CF	ABS	3
DCP OP,X	DF	ABX	3
DCP OP,Y	DB	ABY	3
DCP (OP,X)	C3	(ZP,X)	2
DCP (OP),Y	D3	(ZP),Y	2

■DOP : double NOP
Folgende Codes wirken wie der NOP-Befehl, sind aber zwei Byte lang. Das zweite Byte wird dabei übersprungen.

Ø4, 14, 34, 44, 54, 64, 74, D4, F4, 80, 89, 93

■ISC : increment and subtract with carry
Entspricht Befehlsfolge:
INC
SBC

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:
ISC OP	E7	ZP	2
ISC OP,X	F7	ZPX	2
ISC OP	EF	ABS	3
ISC OP,X	FF	ABX	3
ISC OP,Y	FB	ABY	3
ISC (OP,X)	E3	(OP,X)	2
ISC (OP),Y	F3	(OP),Y	2

■KIL : killer codes
Folgende Codes bewirken einen Absturz des Prozessors, dem auch mit einem RUN/STOP-RESTORE nicht mehr beizukommen ist.

Ø2, 12, 22, 32, 42, 52, 62, 72, 92, B2, D2, F2

■LAR : load akku, AND with stackregister, transfer result to akku, X-register and stackregister
Entspricht Befehlsfolge:
LDA
AND
TAX
TXS

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:
LAR OB,Y	BB	ABY	3

■LAX : load to akku and X-register
Entspricht Befehlsfolge:
LDA
TAX

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:
LAX OP	A7	ZP	2
LAX OP,Y	B7	ZPY	2
LAX OP	AF	ABS	3
LAX OP,Y	BF	ABY	3
LAX (OP,X)	A3	(OP,X)	2
LAX (OP),Y	B3	(OP),Y	2

■NOP : no operation
Folgende Codes haben wie der Code $EA die NOP-Funktion:

1A, 3A, 5A, 7A, DA, FA

Tabelle 1. Die »illegalen Opcodes« des 6510-Prozessors

■ RLA : rotate left, AND with akku and store akku
Entspricht Befehlsfolge:
ROL
AND
STA

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:
RLA OP	27	ZP	2
RLA OP,X	37	ZPX	2
RLA OP	2F	ABS	3
RLA OP,X	3F	ABX	3
RLA OP,Y	3B	ABY	3
RLA (OP,X)	23	(OP,X)	2
RLA (OP),Y	33	(OP),Y	2

■ RRA : rotate right and add with carry
Entspricht Befehlsfolge:
ROR
ADC

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:
RRA OP	67	ZP	2
RRA OP,X	77	ZPX	2
RRA OP	6F	ABS	3
RRA OP,X	7F	ABX	3
RRA OP,Y	7B	ABY	3
RRA (OP,X)	63	(OP,X)	2
RRA (OP),Y	73	(OP),Y	2

■ SLO : shift left and OR with akku

Entspricht Befehlsfolge:
ASL
ORA

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:
SLO OP	07	ZP	2
SLO OP,X	17	ZPX	2
SLO OP	0F	ABS	3
SLO OP,X	1F	ABX	3
SLO OP,Y	1B	ABY	3
SLO (OP,X)	13	(OP,X)	2
SLO (OP),Y	03	(OP),Y	2

■ SRE : shift right and EOR with akku
Entspricht Befehlsfolge:
LSR
EOR

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:
SRE OP	47	ZP	2
SRE OP,X	57	ZPX	2
SRE OP	4F	ABS	3
SRE OP,X	5F	ABX	3
SRE OP,Y	5B	ABY	3
SRE (OP,X)	43	(OP,X)	2
SRE (OP),Y	53	(OP),Y	2

■ TOP : triple NOP
Folgende Codes wirken wie der NOP-Befehl, sind aber drei Byte lang. Das zweite und das dritte Byte wird dabei übersprungen.

0C, 1C, 3C, 5C, 7C, DC, FC

Tabelle 1. Die »illegalen Opcodes« des 6510-Prozessors (Schluß)

■ ADC : add with carry
addiere Adresseninhalt plus Carry-Flag zum Akkumulator

Flags: N Z C I D V
+ + + + +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
ADC #OP	69	IM	2	2
ADC OP	65	ZP	2	3
ADC OP,X	75	ZPX	2	4
ADC OP	6D	ABS	3	4
ADC OP,X	7D	ABX	3	4
ADC OP,Y	79	ABY	3	4
ADC (OP,X)	61	(ZP,X)	2	6
ADC (OP),Y	71	(ZP),Y	2	5

■ AND : AND akku
verknüpfe Speicher mit Akku durch logische UND

Flags: N Z C I D V
+ +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
AND #OP	29	IM	2	2
AND OP	25	ZP	2	3
AND OP,X	35	ZPX	2	4
AND OP	2D	ABS	3	4
AND OP,X	3D	ABX	3	4
AND OP,Y	39	ABY	3	4
AND (OP,X)	21	(ZP,X)	2	6
AND (OP),Y	31	(ZP),Y	2	5

■ ASL : arithmetic shift left
schiebe Bits eines Speichers um eine Stelle nach links

Flags: N Z C I D V
+ + +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
ASL	0A	Akku	1	2
ASL OP	06	ZP	2	5
ASL OP,X	16	ZPX	2	6
ASL OP	0E	ABS	3	6
ASL OP,X	1E	ABX	3	7

■ BCC : branch if carry clear
verzweige, falls das Übertragsbit gelöscht ist

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
BCC OP	90	REL	2	2

■ BCS : branch if carry set
verzweige, falls das Übertragsbit gesetzt ist

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
BCS OP	B0	REL	2	2

■ BEQ : branch if equal (to zero)
verzweige, falls das Ergebnis der letzten Operation gleich (Null) war

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
BEQ OP	F0	REL	2	2

■ BIT : test bits
verknüpfe Speicher und Akku durch AND, setze entsprechende Flags (Akku wird nicht verändert !)

Flags: N Z C I D V
+ + +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
BIT OP	24	ZP	2	3
BIT OP	2C	ABS	3	4

■ BMI : branch if minus
verzweige, falls das Ergebnis der letzten Operation kleiner Null war

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
BMI OP	30	REL	2	2

Tabelle 2. Die Befehle des 6510-Prozessors

■ **BNE** : branch if not equal (to zero)
verzweige, falls das Ergebnis der letzten Operation ungleich (Null) war

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
BNE OP	D0	REL	2	2

■ **BPL** : branch if plus
verzweige, falls das Ergebnis der letzten Operation größer Null war

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
BPL OP	10	REL	2	2

■ **BRK** : break
Programmstop und Sprung über Breakpointer

Flags: N Z C I D V
+

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
BRK	00	-	1	7

■ **BVC** : branch if overflow clear
verzweige, falls das Überlaufsbit gelöscht ist

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
BVC OP	50	REL	2	2

■ **BVS** : branch if overflow set
verzweige, falls das Überlaufsbit gesetzt ist

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
BVS OP	70	REL	2	2

■ **CLC** : clear carry
lösche das Übertragsbit

Flags: N Z C I D V
+

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
CLC	18	-	1	2

■ **CLD** : clear decimal mode
lösche das Bit für den Dezimalmodus

Flags: N Z C I D V
+

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
CLD	D8	-	1	2

■ **CLI** : clear interrupt flag
lösche das Interruptbit (Interrupts nun erlaubt)

Flags: N Z C I D V
+

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
CLI	58	-	1	2

■ **CLV** : clear overflow flag
lösche das Überlaufbit

Flags: N Z C I D V
+

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
CLV	B8	-	1	2

■ **CMP** : compare with akku
vergleiche Speicher mit Akkuinhalt

Flags: N Z C I D V
+ + +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
CMP #OP	C9	IM	2	2
CMP OP	C5	ZP	2	3
CMP OP,X	D5	ZPX	2	4
CMP OP	CD	ABS	3	4
CMP OP,X	DD	ABX	3	4
CMP OP,Y	D9	ABY	3	4
CMP (OP,X)	C1	(ZP,X)	2	6
CMP (OP),Y	D1	(ZP),Y	2	5

■ **CPX** : compare with X-register
vergleiche Speicherinhalt mit X-Register

Flags: N Z C I D V
+ + +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
CPX #OP	C9	IM	2	2
CPX OP	E4	ZP	2	3
CPX OP	EC	ABS	3	4

■ **CPY** : compare with Y-register
vergleiche Speicherinhalt mit Y-Register

Flags: N Z C I D V
+ + +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
CPY #OP	C0	IM	2	2
CPY OP	C4	ZP	2	3
CPY OP	CC	ABS	3	4

■ **DEC** : decrement
subtrahiere Eins von Speicherinhalt

Flags: N Z C I D V
+ +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
DEC OP	C6	ZP	2	5
DEC OP,X	D6	ZPX	2	6
DEC OP	CE	ABS	3	6
DEC OP,X	DE	ABX	3	7

■ **DEX** : decrement X-register
subtrahiere Eins vom Inhalt des X-Registers

Flags: N Z C I D V
+ +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
DEX	CA	-	1	2

■ **DEY** : decrement Y-register
subtrahiere Eins vom Inhalt des Y-Registers

Flags: N Z C I D V
+ +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
DEY	88	-	1	2

Tabelle 2. Die Befehle des 6510-Prozessors (Fortsetzung)

■ **EOR** : exclusive-or
verknüpfe Akku und Speicher durch logisches EXKLUSIV-ODER

Flags: N Z C I D V
+ +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
EOR #OP	49	IM	2	2
EOR OP	45	ZP	2	3
EOR OP,X	55	ZPX	2	4
EOR OP	4D	ABS	3	4
EOR OP,X	5D	ABX	3	4
EOR OP,Y	59	ABY	3	4
EOR (OP,X)	41	(ZP,X)	2	6
EOR (OP),Y	51	(ZP),Y	2	5

■ **INC** : increment
addiere Eins zu Speicherinhalt

Flags: N Z C I D V
+ +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
INC OP	E6	ZP	2	5
INC OP,X	F6	ZPX	2	6
INC OP	EE	ABS	3	6
INC OP,X	FE	ABX	3	7

■ **INX** : increment X-register
addiere Eins zu X-Registerinhalt

Flags: N Z C I D V
+ +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
INX	E8	-	1	2

■ **INY** : increment Y-register
addiere Eins zu Y-Registerinhalt

Flags: N Z C I D V
+ +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
INY	C8	-	1	2

■ **JMP** : jump
springe zu Adresse

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
JMP OP	4C	ABS	3	3
JMP (OP)	6C	IND	3	5

■ **JSR** : jump subroutine
Springe in Unterprogramm

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
JSR OP	20	ABS	3	6

■ **LDA** : load akku
schreibe Wert in Akku

Flags: N Z C I D V
+ +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
LDA #OP	A9	IM	2	2
LDA OP	A5	ZP	2	3
LDA OP,X	B5	ZPX	2	4
LDA OP	AD	ABS	3	4
LDA OP,X	BD	ABX	3	4
LDA OP,Y	B9	ABY	3	4
LDA (OP,X)	A1	(ZP,X)	2	6
LDA (OP),Y	B1	(ZP),Y	2	5

■ **LDX** : load X-register
schreibe Wert ins X-Register

Flags: N Z C I D V
+ +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
LDX #OP	A2	IM	2	2
LDX OP	A5	ZP	2	3
LDX OP,Y	B6	ZPY	2	4
LDX OP	AE	ABS	3	4
LDX OP,Y	BE	ABY	3	4

■ **LDY** : load Y-register
schreibe Wert ins Y-Register

Flags: N Z C I D V
+ +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
LDY #OP	A0	IM	2	2
LDY OP	A4	ZP	2	3
LDY OP,X	B4	ZPX	2	4
LDY OP	AC	ABS	3	4
LDY OP,X	BC	ABX	3	4

■ **LSR** : logical shift right
bitweises Rechtsschieben eines Speicherinhalts
(Bit 0 wird ins Carry-Flag geschoben, Bit 7 wird auf Null gesetzt)

Flags: N Z C I D V
+ + +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
LSR	4A	Akku	1	2
LSR OP	46	ZP	2	5
LSR OP,X	56	ZPX	2	6
LSR OP	4E	ABS	3	6
LSR OP,X	5E	ABX	3	7

■ **NOP** : no operation
keine Ausführung (Dummy-Befehl)

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
NOP	EA	-	1	2

■ **ORA** : OR akku
verknüpfe Speicherinhalt und Akku durch logisches ODER

Flags: N Z C I D V
+ +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
ORA #OP	09	IM	2	2
ORA OP	05	ZP	2	3
ORA OP,X	15	ZPX	2	4
ORA OP	0D	ABS	3	4
ORA OP,X	1D	ABX	3	4
ORA OP,Y	19	ABY	3	4
ORA (OP,X)	01	(ZP,X)	2	6
ORA (OP),Y	11	(ZP),Y	2	5

■ **PHA** : push akku
schiebe Akkuinhalt auf Stack

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
PHA	48	-	1	3

■ **PHP** : push processor-status
schiebe Statusregister auf Stack

Tabelle 2. Die Befehle des 6510-Prozessors (Fortsetzung)

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
PHP	08	-	1	3

■ PLA : pull akku
lade Akku mit oberstem Stackbyte

Flags: N Z C I D V
+ +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
PLA	68	-	1	3

■ PLP : pull processor-status
lade Statusregister mit oberstem Stackbyte

Flags: N Z C I D V
+ + + + + +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
PLP	28	-	1	4

■ ROL : rotate left
rotiere Speicherinhalt um ein Bit nach links (Bit 7 kommt ins Carryflag, Inhalt des Carry-Flags kommt ins Bit 0)

Flags: N Z C I D V
+ + +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
ROL	2A	Akku	1	2
ROL OP	26	ZP	2	5
ROL OP,X	36	ZPX	2	6
ROL OP	2E	ABS	3	6
ROL OP,X	3E	ABX	3	7

■ ROR : rotate right
rotiere Speicherinhalt um ein Bit nach rechts (Bit 0 kommt ins Carryflag, Inhalt des Carryflags kommt ins Bit 7)

Flags: N Z C I D V
+ + +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
ROR	6A	Akku	1	2
ROR OP	66	ZP	2	5
ROR OP,X	76	ZPX	2	6
ROR OP	6E	ABS	3	6
ROR OP,X	7E	ABX	3	7

■ RTI : return from interrupt
nach Ausführen eines Interupt normales Programm weiter abarbeiten

Flags: N Z C I D V
wie vor Ausführung des Interrupts

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
RTI	40	-	1	6

■ RTS : return from subroutine
Rücksprung aus Unterprogramm

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
RTS	60	-	1	6

■ SBC : subtract with carry
subtrahiere Speicherinhalt vom Akku unter Berücksichtigung des Vorzeichens

Flags: N Z C I D V
+ + + + +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
SBC #OP	E9	IM	2	2
SBC OP	E5	ZP	2	3
SBC OP,X	F5	ZPX	2	4
SBC OP	ED	ABS	3	4
SBC OP,X	FD	ABX	3	4
SBC OP,Y	F9	ABY	3	4
SBC (OP,X)	E1	(ZP,X)	2	6
SBC (OP),Y	F1	(ZP),Y	2	5

■ SEC : set carry
setze das Übertragsflag auf Eins

Flags: N Z C I D V
+

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
SEC	38	-	1	2

■ SED : set decimal mode
setze das Dezimal-Modus-Flag auf Eins

Flags: N Z C I D V
+

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
SED	F8	-	1	2

■ SEI : set interrupt
setze das Interruptflag auf Eins (es werden keine Interrupts mehr erlaubt)

Flags: N Z C I D V
+

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
SEI	78	-	1	2

■ STA : store akku
schreibe Akkuinhalt in Speicher

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
STA OP	85	ZP	2	3
STA OP,X	95	ZPX	2	4
STA OP	8D	ABS	3	4
STA OP,X	9D	ABX	3	5
STA OP,Y	99	ABY	3	5
STA (OP,X)	81	(ZP,X)	2	6
STA (OP),Y	91	(ZP),Y	2	6

■ STX : store X-register
schreibe X-Registerinhalt in Speicher

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
STX OP	86	ZP	2	3
STX OP,Y	96	ZPY	2	4
STX OP	8E	ABS	3	4

■ STY : store Y-register
schreibe Y-Registerinhalt in Speicher

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
STY OP	84	ZP	2	3
STY OP,X	94	ZPX	2	4
STY OP	8C	ABS	3	4

Tabelle 2. Die Befehle des 6510-Prozessors (Fortsetzung)

■ TAX : transfer akku to X-register
schreibe Akkuinhalt ins X-Register

Flags: N Z C I D V
+ +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
TAX	AA	-	1	2

■ TAY : transfer akku to Y-register
schreibe Akkuinhalt ins Y-Register

Flags: N Z C I D V
+ +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
TAY	A8	-	1	2

■ TSX : transfer stackregister to X-register
schreibe Stackregisterinhalt ins X-Register

Flags: N Z C I D V
+ +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
TSX	BA	-	1	2

■ TXA : transfer X-register to akku

Flags: N Z C I D V
+ +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
TXA	8A	-	1	2

■ TXS : transfer X-register to stackregister
schreibe X-Registerinhalt ins Stackregister

Flags: N Z C I D V
keine

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
TXS	9A	-	1	2

■ TYA : transfer Y-register to akku
schreibe Y-Registerinhalt in Akku

Flags: N Z C I D V
+ +

Addressierungsarten:

Assembler:	Hex-Code:	Abkürzung:	Byte:	Takte:
TYA	98	-	1	2

Tabelle 2. Die Befehle des 6510-Prozessors (Schluß)

■ ABS : absolute (absolut)
Der Operand ist eine vierstellige, hexadezimale Adresse.

Beispiel:
```
LDA $C000
```
Der Inhalt der Adresse $C000 wird in den Akku geladen.

■ ABX : absolut X-indiziert
Der Operand ist eine vierstellige hexadezimale Zahl. Der Inhalt des X-Registers wird zum Operanden addiert und ergibt die Arbeits-Adresse.

Beispiel:
```
LDX #$10
LDA $C000,X
```
Der Inhalt der Speicherstelle $C010 ($C000 + $0010) wird in den Akku geladen.

■ ABY : absolut Y-indiziert
Der Operand ist eine vierstellige hexadezimale Zahl. Der Inhalt des Y-Registers wird zum Operanden addiert und ergibt die Arbeits-Adresse.

Beispiel:
```
LDY #$10
LDA $C000,Y
```
Der Inhalt der Speicherstelle $C010 ($C000 + $0010) wird in den Akku geladen.

>>Byte<< : In den Tabellen 2 und 3 gibt diese Spalte die jeweilige Länge des kompletten Befehls mit Operand an.

>>Flags<< : einzelne Bits des Statusregisters

N : negative flag. Zeigt an, daß bei einer Operation einer der beiden Operanden zwischen $80 (128) und $FF (255) liegt, also das letzte Bit gesetzt ist.

Z : zero flag. Zeigt an, daß das Ergebnis einer Operation im Akku gleich Null ist.

C : carry flag. Zeigt an, daß bei einer Operation ein Übertrag entstanden ist.

I : interrupt flag. Durch Setzen dieses Bit lassen sich Interrupts unterbinden.

D : decimal flag. Durch Setzen dieses Bit wir der Prozessor in den Dezimalmodus geschaltet. Das bedeutet, daß zum Beispiel das Ergebnis der Addition von $09 und $01 nicht $0A, sondern $10 ergibt.

V : overflow flag (Überlauf). Zeigt an, daß das Ergebnis einer Operation größer $FF (=255) war.

■ IM : immediate (unmittelbar)
Die Adressierungsart >>immediate<< bedeutet, daß der Operand unmittelbar als Wert weiterverarbeitet wird.

Beispiel:
```
LDA #$00
```
Die hexadezimale Zahl $00 wird direkt in den Akku geladen.

■ OP : Operand
Je nach Adressierungsart besteht der Operand eines Befehls aus einem (Adressierung >>immediate<< und >>zeropage<<) oder zwei Byte (>>absolute<<).

■ ZP : zeropage
Der Operand besteht aus einem Byte und gibt eine Adresse in der Zeropage (Speicherbereich $0000 bis $00FF) an.

Beispiel:
```
LDA $2B
```
Der Inhalt der Speicherstelle $002B wird in den Akku geladen.

■ ZPX : Zeropage X-indiziert
Der Inhalt des X-Registers wird zum zweistelligen, hexadezimalen Operanden addiert. Das Ergebnis ist eine Adresse in der Zeropage (Speicherbereich $0000 bis $00FF).

Beispiel:
```
LDX #$05
LDA $43,X
```
Der Inhalt der Adresse $0048 ($0043 + $0005) wird in den Akku geladen.

■ ZPY : Zeropage Y-indiziert
Der Inhalt des Y-Registers wird zum zweistelligen, hexadezimalen Operanden addiert. Das Ergebnis ist eine Adresse in der Zeropage (Speicherbereich $0000 bis $00FF).

Beispiel:
```
LDY #$05
LDA $43,Y
```
Der Inhalt der Adresse $0048 ($0043 + $0005) wird in den Akku geladen.

Tabelle 3. Diese Abkürzungen werden in den Tabellen 1 und 2 verwendet

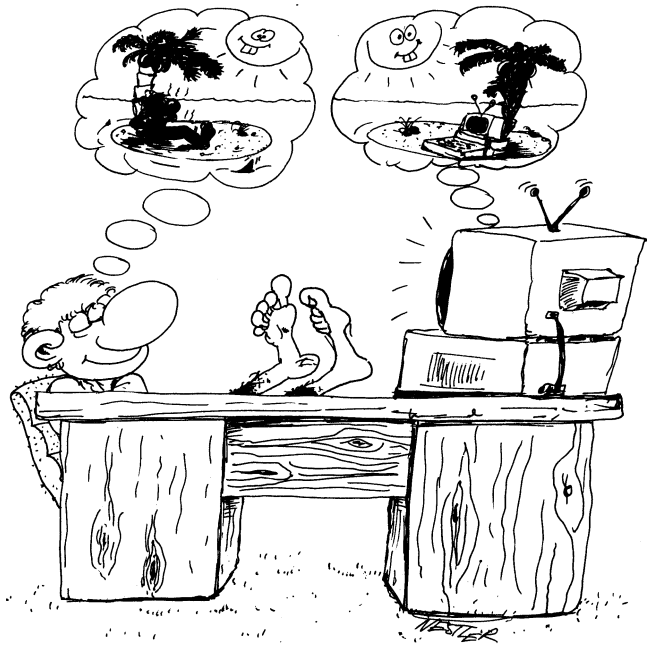
```
■ (ZP,X) : indiziert indirekt
Der Inhalt des X-Registers wird zum zweistelligen,
hexadezimalen Operanden addiert und ergibt eine Adresse
in der Zeropage (Speicherbereich $0000 bis $00FF).
Deren Inhalt und der Inhalt der darauffolgenden Adresse
ergibt in der Form Lo-Byte/Hi-Byte die Arbeitsadresse.

Beispiel:
Adresse $20 hat den Inhalt $00
Adresse $21 hat den Inhalt $C0
LDX #$0E
LDA ($12,X)
Der Inhalt der Zeropage-Adressen $0020 ($000E + $0012)
und $0021 ergibt die Arbeits-Adresse $C000. Deren
Inhalt wird in den Akku geladen.

■ (ZP),Y : indirekt indiziert
Der zweistellige, hexdezimale Operand ergibt eine
Adresse in der Zeropage (Speicherbereich $0000 bis
$00FF). Deren Inhalt und der Inhalt der darauffolgenden
Speicherstelle ergibt in der Form Lo-Byte/Hi-Byte eine
Adresse, zu der der Inhalt des Y-Registers addiert
wird. Das Ergebnis ist die Arbeitsadresse.

Beispiel:
Adresse $20 hat den Inhalt $00
Adresse $21 hat den Inhalt $C0
LDY #$10
LDA ($20),Y
Der Inhalt der Adresse $C010 ($C000 + $0010) wird in
den Akku geladen.
```

Tabelle 3. Diese Abkürzungen werden in den Tabellen 1 und 2 verwendet (Schluß)



ROM-Routinen in eigenen Programmen

Das Rad ist schon erfunden! Ähnlich verhält es sich mit verschiedenen Routinen, die ein Assembler-Programmierer immer wieder benötigt. Aber warum soll man sich die Arbeit des Programmierens machen, wenn das Betriebssystem viele ständig benötigte Routinen schon enthält und man nur noch zu wissen braucht, ab welcher Adresse sie stehen?

Angenommen, Sie möchten in Assembler einige komplexe Dinge programmieren wie beispielsweise eine neue mathematische Funktion (wie wäre es mit dem Kotangens) und diese auf dem Bildschirm ausgeben. Das ist eine große Aufgabe, zu der zunächst einmal die Übernahme des Arguments in das Maschinenprogramm, dann einige Fließkomma-Rechenoperationen und schließlich die Ausgabe auf dem Bildschirm geschrieben werden müßten, wenn da nicht schon fast alles an verborgener Stelle als fertige Programm-Module im Computer vorhanden wäre!

Sowohl im unteren (von $A000 bis $BFFF) als auch im oberen ROM-Bereich (von $E000 bis $FFFF) liegt die Firmware fest verschachtelt vor. Der untere ROM-Abschnitt wird manchmal auch Basic-Interpreter, der obere ROM-Bereich Betriebssystem genannt, wobei diese Einteilung aber den Kern der Sache nicht genau trifft, denn Interpreter, Editor und Betriebssystem führen ein gemischtes Dasein quer durch alle genannten ROM-Bereiche hindurch.

Mindestens fünf Informationen braucht ein Assembler-Programmierer, wenn er das breite Programmangebot des ROMs nutzen möchte:

1. Einsprungadresse
2. Format der Eingabeparameter
3. Adressen der Eingabeparameter
4. Adressen der Ausgabeparameter
5. Format der Ausgabeparameter

Nicht alle Routinen, die man benutzen kann, erfordern alle fünf Informationen, manche weniger, einige auch mehr und schließlich gibt es noch Programmroutinen, die noch den Aufruf einer oder sogar mehrerer anderer Routinen nötig machen.

In der beigefügten Tabelle sind – nach Anwendungen sortiert – die wichtigsten Firmware-Möglichkeiten mit den erforderlichen Ein- und Ausgabeparametern aufgeführt. Das sind natürlich beileibe nicht alle. Die Auswahl erfolgte subjektiv! Es sind einfach diejenigen, die mir bislang am häufigsten untergekommen sind. Außerdem wurde auf die Kernel-Routinen verzichtet: Man findet diese sehr gut dokumentiert bereits in einer Reihe von Büchern und im Assembler-Kurs.

Die Tabelle nennt den Label-Namen, die Einsprungadresse und gibt eine Kurzbeschreibung der Funktion. Das Ein- und auch das Ausgabeformat ist ebenso angegeben wie auch die Adressen, an denen diese Parameter übergeben werden. Die verwendeten Bezeichnungen halten sich eng an die im Assembler-Kurs kennengelernten. Sie sind allgemein üblich: