6502/6510/8500/8502 Opcode matrix:

imm = #$00
zp = $00
zpx = $00,X
zpy = $00,Y
izx = ($00,X)
izy = ($00),Y
abs = $0000
abx = $0000,X
aby = $0000,Y
ind = ($0000)
rel = $0000 (PC-relative)

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	BRK 7	ORA izx 6	KIL	SLO izx 8	NOP zp 3	ORA zp 3	ASL zp 5	SLO zp 5	PHP 3	ORA imm 2	ASL 2	ANC imm 2	NOP abs 4	ORA abs 4	ASL abs 6	SLO abs 6
1x	BPL rel 2*	ORA izy 5*	KIL	SLO izy 8	NOP zpx 4	ORA zpx 4	ASL zpx 6	SLO zpx 6	CLC 2	ORA aby 4*	NOP 2	SLO aby 7	NOP abx 4*	ORA abx 4*	ASL abx 7	SLO abx 7
2x	JSR abs 6	AND izx 6	KIL	RLA izx 8	BIT zp 3	AND zp 3	ROL zp 5	RLA zp 5	PLP 4	AND imm 2	ROL 2	ANC imm 2	BIT abs 4	AND abs 4	ROL abs 6	RLA abs 6
3x	BMI rel 2*	AND izy 5*	KIL	RLA izy 8	NOP zpx 4	AND zpx 4	ROL zpx 6	RLA zpx 6	SEC 2	AND aby 4*	NOP 2	RLA aby 7	NOP abx 4*	AND abx 4*	ROL abx 7	RLA abx 7
4x	RTI 6	EOR izx 6	KIL	SRE izx 8	NOP zp 3	EOR zp 3	LSR zp 5	SRE zp 5	PHA 3	EOR imm 2	LSR 2	ALR imm 2	JMP abs 3	EOR abs 4	LSR abs 6	SRE abs 6
5x	BVC rel 2*	EOR izy 5*	KIL	SRE izy 8	NOP zpx 4	EOR zpx 4	LSR zpx 6	SRE zpx 6	CLI 2	EOR aby 4*	NOP 2	SRE aby 7	NOP abx 4*	EOR abx 4*	LSR abx 7	SRE abx 7
6x	RTS 6	ADC izx 6	KIL	RRA izx 8	NOP zp 3	ADC zp 3	ROR zp 5	RRA zp 5	PLA 4	ADC imm 2	ROR 2	ARR imm 2	JMP ind 5	ADC abs 4	ROR abs 6	RRA abs 6
7x	BVS rel 2*	ADC izy 5*	KIL	RRA izy 8	NOP zpx 4	ADC zpx 4	ROR zpx 6	RRA zpx 6	SEI 2	ADC aby 4*	NOP 2	RRA aby 7	NOP abx 4*	ADC abx 4*	ROR abx 7	RRA abx 7
8x	NOP imm 2	STA izx 6	NOP imm 2	SAX izx 6	STY zp 3	STA zp 3	STX zp 3	SAX zp 3	DEY 2	NOP imm 2	TXA 2	*XAA imm 2*	STY abs 4	STA abs 4	STX abs 4	SAX abs 4
9x	BCC rel 2*	STA izy 6	KIL	*AHX izy 6*	STY zpx 4	STA zpx 4	STX zpy 4	SAX zpy 4	TYA 2	STA aby 5	TXS 2	*TAS aby 5*	*SHY abx 5*	STA abx 5	*SHX aby 5*	*AHX aby 5*
Ax	LDY imm 2	LDA izx 6	LDX imm 2	LAX izx 6	LDY zp 3	LDA zp 3	LDX zp 3	LAX zp 3	TAY 2	LDA imm 2	TAX 2	*LAX imm 2*	LDY abs 4	LDA abs 4	LDX abs 4	LAX abs 4
Bx	BCS rel 2*	LDA izy 5*	KIL	LAX izy 5*	LDY zpx 4	LDA zpx 4	LDX zpy 4	LAX zpy 4	CLV 2	LDA aby 4*	TSX 2	LAS aby 4*	LDY abx 4*	LDA abx 4*	LDX aby 4*	LAX aby 4*

Cx	CPY imm 2	CMP izx 6	NOP imm 2	DCP izx 8	CPY zp 3	CMP zp 3	DEC zp 5	DCP zp 5	INY 2	CMP imm 2	DEX 2	AXS imm 2	CPY abs 4	CMP abs 4	DEC abs 6	DCP abs 6
Dx	BNE rel 2*	CMP izy 5*	KIL	DCP izy 8	NOP zpx 4	CMP zpx 4	DEC zpx 6	DCP zpx 6	CLD 2	CMP aby 4*	NOP 2	DCP aby 7	NOP abx 4*	CMP abx 4*	DEC abx 7	DCP abx 7
Ex	CPX imm 2	SBC izx 6	NOP imm 2	ISC izx 8	CPX zp 3	SBC zp 3	INC zp 5	ISC zp 5	INX 2	SBC imm 2	NOP 2	SBC imm 2	CPX abs 4	SBC abs 4	INC abs 6	ISC abs 6
Fx	BEQ rel 2*	SBC izy 5*	KIL	ISC izy 8	NOP zpx 4	SBC zpx 4	INC zpx 6	ISC zpx 6	SED 2	SBC aby 4*	NOP 2	ISC aby 7	NOP abx 4*	SBC abx 4*	INC abx 7	ISC abx 7

"*" : add 1 cycle if page boundary is crossed.
add 1 cycle on branches if taken.

Logical and arithmetic commands:

Opcode	imp	imm	zp	zpx	zpy	izx	izy	abs	abx	aby	ind	rel	Function	N	V	B	D	I	Z	C
ORA		$09	$05	$15		$01	$11	$0D	$1D	$19			A:=A or {adr}	*					*	
AND		$29	$25	$35		$21	$31	$2D	$3D	$39			A:=A&{adr}	*					*	
EOR		$49	$45	$55		$41	$51	$4D	$5D	$59			A:=A exor {adr}	*					*	
ADC		$69	$65	$75		$61	$71	$6D	$7D	$79			A:=A+{adr}	*	*				*	*
SBC		$E9	$E5	$F5		$E1	$F1	$ED	$FD	$F9			A:=A-{adr}	*	*				*	*
CMP		$C9	$C5	$D5		$C1	$D1	$CD	$DD	$D9			A-{adr}	*					*	*
CPX		$E0	$E4					$EC					X-{adr}	*					*	*
CPY		$C0	$C4					$CC					Y-{adr}	*					*	*
DEC			$C6	$D6				$CE	$DE				{adr}:={adr}-1	*					*	
DEX	$CA												X:=X-1	*					*	
DEY	$88												Y:=Y-1	*					*	
INC			$E6	$F6				$EE	$FE				{adr}:={adr}+1	*					*	
INX	$E8												X:=X+1	*					*	
INY	$C8												Y:=Y+1	*					*	
ASL	$0A		$06	$16				$0E	$1E				{adr}:={adr}*2	*					*	*
ROL	$2A		$26	$36				$2E	$3E				{adr}:={adr}*2+C	*					*	*
LSR	$4A		$46	$56				$4E	$5E				{adr}:={adr}/2	*					*	*
ROR	$6A		$66	$76				$6E	$7E				{adr}:={adr}/2+C*128	*					*	*

Move commands:

Opcode	imp	imm	zp	zpx	zpy	izx	izy	abs	abx	aby	ind	rel	Function	N	V	B	D	I	Z	C
LDA		$A9	$A5	$B5		$A1	$B1	$AD	$BD	$B9			A:={adr}	*					*	
STA			$85	$95		$81	$91	$8D	$9D	$99			{adr}:=A							

LDX		$A2	$A6		$B6			$AE		$BE			X:={adr}	*					*	
STX			$86		$96			$8E					{adr}:=X							
LDY		$A0	$A4	$B4				$AC	$BC				Y:={adr}	*					*	
STY			$84	$94				$8C					{adr}:=Y							
TAX	$AA												X:=A	*					*	
TXA	$8A												A:=X	*					*	
TAY	$A8												Y:=A	*					*	
TYA	$98												A:=Y	*					*	
TSX	$BA												X:=S	*					*	
TXS	$9A												S:=X							
PLA	$68												A:=+(S)	*					*	
PHA	$48												(S)-:=A							
PLP	$28												P:=+(S)	*	*		*	*	*	*
PHP	$08												(S)-:=P							

Jump/Flag commands:

Opcode	imp	imm	zp	zpx	zpy	izx	izy	abs	abx	aby	ind	rel	Function	N	V	B	D	I	Z	C
BPL												$10	branch on N=0							
BMI												$30	branch on N=1							
BVC												$50	branch on V=0							
BVS												$70	branch on V=1							
BCC												$90	branch on C=0							
BCS												$B0	branch on C=1							
BNE												$D0	branch on Z=0							
BEQ												$F0	branch on Z=1							
BRK	$00												(S)-:=PC,P PC:=($FFFE)			1		1		
RTI	$40												P,PC:=+(S)	*	*		*	*	*	*
JSR								$20					(S)-:=PC PC:={adr}							
RTS	$60												PC:=+(S)							
JMP								$4C			$6C		PC:={adr}							
BIT			$24					$2C					N:=b7 V:=b6 Z:=A&{adr}	*	*				*	
CLC	$18												C:=0							0
SEC	$38												C:=1							1
CLD	$D8												D:=0				0			
SED	$F8												D:=1				1			
CLI	$58												I:=0					0		
SEI	$78												I:=1					1		
CLV	$B8												V:=0		0					

NOP	$EA																			

Flags of the status register:

The processor status register has 8 bits, where 7 are used as flags:

N = negative flag (1 when result is negative)
V = overflow flag (1 on signed overflow)
# = unused (always 1)
B = break flag (1 when interupt was caused by a BRK)
D = decimal flag (1 when CPU in BCD mode)
I = IRQ flag (when 1, no interupts will occur (exceptions are IRQs forced by BRK and NMIs))
Z = zero flag (1 when all bits of a result are 0)
C = carry flag (1 on unsigned overflow)

Hardware vectors:

$FFFA = NMI vector (NMI=not maskable interupts)
$FFFC = Reset vector
$FFFE = IRQ vector

Illegal opcodes:

Opcode	imp	imm	zp	zpx	zpy	izx	izy	abs	abx	aby	ind	rel	Function	N	V	B	D	I	Z	C
SLO			$07	$17		$03	$13	$0F	$1F	$1B			{adr}:={adr}*2 A:=A or {adr}	*					*	*
RLA			$27	$37		$23	$33	$2F	$3F	$3B			{adr}:={adr}rol A:=A and {adr}	*					*	*
SRE			$47	$57		$43	$53	$4F	$5F	$5B			{adr}:={adr}/2 A:=A exor {adr}	*					*	*
RRA			$67	$77		$63	$73	$6F	$7F	$7B			{adr}:={adr}ror A:=A adc {adr}	*	*				*	*
SAX			$87		$97	$83		$8F					{adr}:=A&X							
LAX			$A7		$B7	$A3	$B3	$AF		$BF			A,X:={adr}	*					*	
DCP			$C7	$D7		$C3	$D3	$CF	$DF	$DB			{adr}:={adr}-1 A-{adr}	*					*	*
ISC			$E7	$F7		$E3	$F3	$EF	$FF	$FB			{adr}:={adr}+1 A:=A-{adr}	*	*				*	*
ANC		$0B											A:=A&#{imm}	*					*	*
ANC		$2B											A:=A&#{imm}	*					*	*
ALR		$4B											A:=(A&#{imm})/2	*					*	*
ARR		$6B											A:=(A&#{imm})/2	*	*				*	*
XAA²		$8B											A:=X&#{imm}	*					*	
LAX²		$AB											A,X:=#{imm}	*					*	
AXS		$CB											X:=A&X-#{imm}	*					*	*

SBC		$EB											A:=A-#{imm}	*	*				*	*
AHX¹							$93			$9F			{adr}:=A&X&H							
SHY¹									$9C				{adr}:=Y&H							
SHX¹										$9E			{adr}:=X&H							
TAS¹										$9B			S:=A&X {adr}:=S&H							
LAS										$BB			A,X,S:={adr}&S	*					*	

```
¹ = unstable in certain matters
² = highly unstable (results are not predictable on some machines)
A = Akkumulator
X = X-Register
Y = Y-Register
S = Stack-Pointer
P = Status-Register
+(S) = Stack-Pointer relative with pre-increment
(S)- = Stack-Pointer relative with post-decrement
```

Combinations of two operations with the same addressing mode:

```
SLO {adr} = ASL {adr} + ORA {adr}
RLA {adr} = ROL {adr} + AND {adr}
SRE {adr} = LSR {adr} + EOR {adr}
RRA {adr} = ROR {adr} + ADC {adr}
SAX {adr} = store A&X into {adr}
LAX {adr} = LDA {adr} + LDX {adr}
DCP {adr} = DEC {adr} + CMP {adr}
ISC {adr} = INC {adr} + SBC {adr}
```

note to SAX: the A&X operation is a result of A and X put onto the bus at the same time.

Combinations of an immediate and an implied command:

```
ANC #{imm} = AND #{imm} + (ASL)
ANC #{imm} = AND #{imm} + (ROL)
ALR #{imm} = AND #{imm} + LSR
ARR #{imm} = AND #{imm} + ROR
XAA #{imm} = TXA + AND #{imm}
LAX #{imm} = LDA #{imm} + TAX
AXS #{imm} = A&X minus #{imm} into X
SBC #{imm} = SBC #{imm} + NOP
```

note to ANC: this command performs an AND operation only, but bit 7 is put into the carry, as if the ASL/ROL would have been executed.
note to ARR: part of this command are some ADC mechanisms. following effects appear after AND but before ROR: the V-Flag is set according to (A and #{imm})+#{imm}, bit 0 does NOT go into carry, but bit 7 is exchanged with the carry.
note to XAA: DO NOT USE!!! Highly unstable!!!
note to LAX: DO NOT USE!!! On my C128, this opcode is stable, but on my C64-II it loses bits so that the operation looks like this: ORA #? AND #{imm} TAX.
note to AXS: performs CMP and DEX at the same time, so that the MINUS sets the flag like CMP, not SBC.

Combinations of STA/STX/STY:

```
AHX {adr} = stores A&X&H into {adr}
SHX {adr} = stores X&H into {adr}
SHY {adr} = stores Y&H into {adr}
```

note: sometimes the &H drops off. Also page boundary crossing will not work as expected (the bank where the value is stored may be equal to the value stored).

Combinations of STA/TXS and LDA/TSX:

```
TAS {adr} = stores A&X into S and A&X&H into {adr}
LAS {adr} = stores {adr}&S into A, X and S
```

note to LAS: is called as "propably unreliable" in one source.

Bit configuration does not allow any operation on these ones:

```
NOP = has no effects
NOP #{imm} = fetches #{imm} but has no effects
NOP {adr} = fetches {adr} but has no effects
```

```
KIL = halts the CPU. the data bus will be set to #$FF
```

Aliases used in other illegal opcode sources:

SLO = ASO
SRE = LSE
ISC = ISB
ALR = ASR
SHX = A11 (A11 was a result of only having tested this one on adress $1000)
SHY = A11
LAS = LAR
KIL = JAM, HLT

The 6502 bugs:

Zeropage index will not leave zeropage when page boundary is crossed:

```
LDX #$01
LDA $FF,X
```

...will fetch from adress $0000 and not $0100 as indexed.

Indirect adressing modes are not able to fetch an adress which crosses the page boundary:

Four examples to illustrate this:

```
LDA ($FF),Y

LDX #$00
LDA ($FF,X)

LDX #$FF
LDA ($00,X)
```

... will all fetch the low-byte from $00FF and the high-byte from $0000

```
JMP ($12FF)
```

... will fetch the low-byte from $12FF and the high-byte from $1200

The N, V and Z flags do not work correctly in BCD mode:

N will always carry bit 7.
V will always be ((U eor N) nand (U eor V)) (while U is bit 7 of operand 1, V is bit 7 of operand 2 and N is the N flag after the ADC is performed).
please note that SBC is truly ADC with an inverted operand!
Z will be 0 when the non-BCD operation WOULD have resulted in $00, no matter what value the result of the BCD operation is.

example to Z:

```
SED
CLC
LDA #$80
ADC #$80
```

... results in A=$60, but the Z flag is 1.

BCD and non BCD values:

Since only nibble values from 0 to 9 are valid in BCD, it's interesting to see what happens when using A to F:

```
$00+$0F=$15 (an easy way to convert a hex-digit into BCD...)
$00+$1F=$25 (can be claimed as being "ok" since 10+$0F=25)
$10+$1F=$35 ("ok")
$05+$1F=$2A (a non-BCD result, still somewhat "ok" since 5+10+$0F=20+$0A)
$0F+$0A=$1F ("ok", since $0F+$0A=$0F+10)
$0F+$0B=$10 (now, this is plain bullshit!)
```

Different versions of the 6502:

In the C64/C128 series of computers, slightly modified versions of the 6502 were used. The modifications did not affect the functional part of the processor itself. Only a so-called processor port was added. This port, in combination with an external PLA, was used to map ROM and I/O areas into the 64KB RAM of the C64. Also, some bits of the port were used for the legendary Datasette.

The port can be accessed through memory adresses $0000 and $0001, while $0001 is the port itself, and $0000 is the data direction register for it.

Explanation for the bits of $0001:

7 - unused (Flash 8: 0=8MHz/1=1MHz)
6 - unused (C128: ASCII/DIN sense/switch (1=ASCII/0=DIN))
5 - Cassette motor control (0 = motor on)
4 - Cassette switch sense (0 = PLAY pressed)
3 - Cassette write line
2 - CHAREN (0=Character ROM instead of I/O area)
1 - HIRAM ($E000-$FFFF)

0 - LORAM ($A000-$BFFF)

If HIRAM or LORAM is set, the I/O area is mapped to $D000-$DFFF.

$0000 should always be set to $2F (%00101111)

Note to bit 6: This bit is used to select either the ASCII or the DIN character ROM of a C128. When data direction is set to INPUT, the charset is selected externally with the ASCII/DIN key.

CPU versions:

6502: NMOS, used in Commodore disk drives, PET, various other 8 bit computers
6502C: 6502 with additional HALT pin, used in Atari 8 bit computer range
6510: 6502 with additional processor port, used in C64
8500: CMOS version of the 6510, used in C64C and C64G
8502: 2 MHz version of the 8500, used in C128
7501: HMOS-1 version of the 6502, used in C16/C116/Plus4
8501: HMOS-2 version of the 6502, used in C16/C116/Plus4

All of these processors are the same concerning the software-side.

Some processors of the family which are not 100% compatible:

65C02: Extension of the 6502.
65SC02: Small version of the 65C02 which lost a few opcodes again.
65CE02: Extension of the 65C02, used in the C65.
65816: Extended 6502 with new opcodes and 16 bit operation modes.

Zeropage/Stack:

The first 256 bytes of adressable memory are called Zeropage. The 6502 processor family offers a wide selection of adressing modes to work with this part of the memory, which generally results in shorter and (even more important) faster code.

Following the Zeropage, the next 256 bytes (located at $0100-$01FF) are used as processor stack. The stack function of this part is defined as it is in most other CPU's: Writing to stack will automatically decrement the stack pointer, while reading from it will increment it.

back