View As Newsfeed ⇒

View By Category ⇒

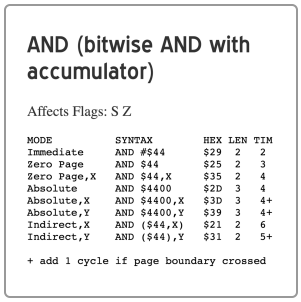
View Full Archive ⇒

NEWS, EDITORIALS, REFERENCE

August 4, 2017 *#39 Reference*

RSS Feed

6502 / 6510 Instruction Set



Every Commodore 64 programmer should have the 6502/6510 instruction set at their fingertips. Although there are many reference texts like this, I find them to be lacking.

My goal is to provide the fastest, easiest-to-use and best organized 6502/6510 instruction set reference document on the internet. It's not just for you, it's for me too. If you have ideas for how I can improve this, let me know in the comments.

Special thanks to 6502.org for the source of the raw content.

Last Updated: September 2021

Functional

Bitwise Instructions	
AND	ASL

Editorial

December 25, 2024 *Editorial*
World of Commodore '24

January 15, 2020 *Editorial*
World of Commodore '19

December 20, 2018 *Editorial*
World of Commodore '18

June 5, 2018 *Editorial*
Review: 1541 Ultimate II+

December 20, 2017 *Editorial*
World of Commodore '17

May 16, 2017 *Editorial*
Review: FREEZE64 Fanzine

December 5, 2016 *Editorial*
World of Commodore '16

Reference

September 3, 2019 *Reference*
SD2IEC User's Manual

January 11, 2019 *Reference*
C64 KERNAL ROM: Making Sense

November 21, 2018 *Reference*
Commodore Hardware Information

March 2, 2018 *Reference*
Key Map to Screen Editor

January 19, 2018 *Reference*
VIC-20 / Commodore 64 SuperChart

August 4, 2017 *Reference*
6502 / 6510 Instruction Set

August 4, 2017 *Reference*
Commodore 64 PETSCII Codes

August 3, 2017 *Reference*
Commodore 64 Screen Codes

Search Archives

Recent Posts

May 8, 2025 *Hardware*
MouSTer and OpCoders Store

December 25, 2024 *Editorial*

EOR	LSR
ORA	ROL
	ROR
Branch Instructions	
BPL	BMI
BVC	BVS
BCC	BCS
BNE	BEQ
Compare Instructions	
CMP	BIT
CPX	
CPY	
Flag Instructions	
CLC	CLD
SEC	SED
CLI	CLV
SEI	
Jump Instructions	
JMP	RTS
JSR	RTI
Math Instructions	
ADC	SBC
Memory Instructions	
LDA	STA
LDX	STX
LDY	STY
DEC	INC
Register Instructions	
TAX	TAY
TXA	TYA
DEX	DEY
INX	INY
Stack Instructions	

World of Commodore '24

November 14, 2024 *Software*
Desktop Designer

September 24, 2024 *Programming Theory*
Networking in C64 OS

May 17, 2024 *Hardware*
RAD Review

May 9, 2024 *Programming Theory*
Hybrid BASIC/ASM Programs

March 25, 2024 *Programming Theory*
Hidden Files

January 26, 2024 *Software*
Eliza for C64 OS

November 30, 2023 *Software*
Fast App Switching

August 17, 2023 *Software*
Image File Formats

May 12, 2023 *Software*
Mounting and Mouse Wheel

January 10, 2023 *Technical Deep Dive*
Gaps in Software IEC

December 20, 2022 *Software*
C64 OS v1.0 release

May 11, 2022 *Software*
Updates on C64 OS, Beta 0.8 and 0.9

December 16, 2021 *Programming Theory*
Text Rendering and MText

October 22, 2021 *Technical Deep Dive*
VIC-II and FLI Timing (2/3)

August 26, 2021 *Technical Deep Dive*

PHA	PLA
PHP	PLP
TSX	TXS
Other Instructions	
BRK	NOP

Alphabetical

A... is for Add		
ADC	AND	ASL
B... is for Branch		
BCC	BCS	BEQ
BIT	BMI	BNE
BPL	BRK	BVC
BVS		
C... is for Clear		
CLC	CLD	CLI
CLV	CMP	CPX
CPY		
D... is for Decrement		
DEC	DEX	DEY
E... is for Exclusive		
EOR		
I... is for Increment		
INC	INX	INY
J... is for Jump		
JMP	JSR	
L... is for Load		
LDA	LDX	LDY
LSR		
N... is for No		

VIC-II and FLI Timing (1/3)

July 23, 2021 *Software*
Image Search and Conversion Service

June 23, 2021 *Software*
C64 OS Subsite and Guides

April 30, 2021 *Software*
Updates on C64 OS, Beta 0.5

March 25, 2021 *Programming Theory*
Shared Libraries

February 22, 2021 *Software*
Introduction to File Manager

January 31, 2021 *Software*
GeoRAM, DigiMAX and NMI Programming

December 30, 2020 *Programming Theory*
Anatomy of a Utility

November 10, 2020 *Technical Deep Dive*
How Does the 1541 Drive Work

View full archive ⇒

Notices

NOP		
O... is for Or		
ORA		
P... is for Push		
PHA	PHP	PLA
PLP		
R... is for Roll		
ROL	ROR	RTI
RTS		
S... is for Subtract		
SBC	SEC	SED
SEI	STA	STX
STY		
T... is for Transfer		
TAX	TAY	TSX
TXA	TXS	TYA

Processor Flags

The Processor flags are the bits of the 6502's 8-bit processor status register. The behavior of many instructions is influenced by the state of one or more processor flags at the time of execution. Additionally, some instructions may change the state of one or more flags. Following the syntax table for each instruction, you will find a block of processor flags such as the one below. The highlighted flags are those which are potentially modified by the instruction.

N	*V*	*–*	*B*	*D*	*I*	**Z**	*C*

Each flag has a letter symbol for easier reference. Here are the flags and what they mean:

Flag	Sym	Bit	Description
Negative	N	b7	Set when an operation results in a negative number

Overflow	V	b6	Set when a signed addition or subtraction results in an overflow
Unused	—	b5	This bit of the processor status register is not used
Break	B	b4	Set when a BRK instruction is executed
Decimal Mode	D	b3	When set, certain instructions operate in decimal rather than binary mode
Interrupt Mask	I	b2	When set, interrupt requests are ignored
Zero	Z	b1	Set when an operation results in a zero
Carry	C	b0	Set when an unsigned addition or subtraction results in an overflow

See: Flag Instructions

The Interrupt Flag (I)

The interrupt flag is used to mask (prevent) maskable interrupts (aka IRQs) from occurring. When the flag is set (SEI) interrupts are masked. When it is clear (CLI) interrupts are allowed to occur. This flag does not signal the presence or absence of an interrupt condition.

The interrupt flag is automatically set in response to an interrupt, to prevent the interruption of an interrupt service routine in progress. The state of the interrupt flag is restored to its prior state by the return from interrupt (RTI) instruction. If you want an interrupt service routine to be able to be interrupted, you must clear the interrupt flag (CLI) within that routine.

The Decimal Flag (D)

The decimal flag controls how the 6502 adds and subtracts. If set, arithmetic is carried out in packed binary coded decimal. This flag is unchanged by interrupts and is unknown on power-up. If you are uncertain of what mode you are in, you should explicitly clear (CLD) or set (SED) decimal mode to ensure expected behavior.

The Overflow Flag (V)

The overflow flag is generally misunderstood and therefore under-utilised. After addition (ADC) or subtraction (SBC), the overflow flag is set if the two's complement result is less than -128 or greater than +127, otherwise the flag is cleared.

In twos complement, $80 through $FF represent -128 through -1, and $00 through $7F represents 0 through +127. Thus, after:

```
CLC
LDA #$7F ;  127
ADC #$01 ; +  1
```

The overflow flag is set, because in signed arithmetic (127 + 1 = -128). And after:

```
SEC
LDA #$80 ; -128
SBC #$01 ; -  1
```

The overflow flag is set, because in signed arthmetic (-128 -1 = 127).

The overflow flag is not affected by increments, decrements, shifts and logical operations. It is only affected by: ADC, SBC, BIT, CLV, PLP and RTI. Unlike the decimal flag and the carry flag which have pairs of instructions for clearing and setting (CLD/SED and CLC/SEC), there is no instruction to set the overflow flag. A workaround is to BIT any byte whose bit6 is set. A common technique is to BIT an RTS instruction.

For a more thorough explanation of the overflow flag, see: The Overflow (V) Flag Explained.

Execution Times

Op code execution times are measured in machine cycles; one machine cycle equals one clock cycle. Many instructions require one extra cycle for execution if a page boundary is crossed; these are indicated by a + following the time values shown.

Program Counter

When the 6502 is ready for the next instruction it increments the program counter before fetching the instruction. Once it has the op code, it increments the program counter by the length of the operand, if any. This must be accounted for when calculating branches or when pushing bytes to create a false return address (i.e. jump table addresses are made up of addresses-1 when it is intended to use an RTS rather than a JMP).

The program counter is loaded least significant byte first. Therefore the most significant byte must be pushed first when creating a false

return address.

When calculating branches a forward branch of 6 skips the following 6 bytes so, effectively the program counter points to the address that is 8 bytes beyond the address of the branch opcode; and a backward branch of $FA (256-6) goes to an address 4 bytes before the branch instruction.

See: Branch Instructions

Wrap-Around

Use caution with indexed zero page operations as they are subject to wrap-around. For example, if the X register holds $FF and you execute LDA $80,X you will not access $017F as you might expect; instead you access $7F i.e. $80-1. This characteristic can be used to advantage but make sure your code is well commented.

It is possible, however, to access $017F when X = $FF by using the Absolute,X addressing mode of LDA $80,X. That is, instead of:

```
LDA $80,X    ; ZeroPage,X - the resulting object code is: B5 80
```

which accesses $007F when X=$FF, use:

```
LDA $0080,X  ; Absolute,X - the resulting object code is: BD 80
```

which accesses $017F when X = $FF (a at cost of one additional byte and one additional cycle). All of the ZeroPage,X and ZeroPage,Y instructions except STX ZeroPage,Y and STY ZeroPage,X have a corresponding Absolute,X and Absolute,Y instruction. Unfortunately, a lot of 6502 assemblers don't have an easy way to force Absolute addressing, i.e. most will assemble a LDA $0080,X as B5 80. One way to overcome this is to insert the bytes using the .BYTE pseudo-op (on some 6502 assemblers this pseudo-op is called DB or DFB, consult the assembler documentation) as follows:

```
.BYTE $BD,$80,$00  ; LDA $0080,X (absolute,X addressing mode)
```

The comment is optional, but highly recommended for clarity.

In cases where you are writing code that will be relocated you must consider wrap-around when assigning dummy values for addresses that will be adjusted. Both zero and the semi-standard $FFFF should be avoided for dummy labels. The use of zero or zero page values will result in assembled code with zero page opcodes when

you wanted absolute codes. With $FFFF, the problem is in addresses+1 as you wrap around to page 0.

See: Memory Instructions

Bitwise Instructions

AND

Bitwise AND with Accumulator

Mode	Syntax	HEX	LEN	TIME
Immediate	AND #$44	$29	2	2
Zero Page	AND $44	$25	2	3
Zero Page,X	AND $44,X	$35	2	4
Absolute	AND $4400	$2D	3	4
Absolute,X	AND $4400,X	$3D	3	4+
Absolute,Y	AND $4400,Y	$39	3	4+
(Indirect,X)	AND ($44,X)	$21	2	6
(Indirect),Y	AND ($44),Y	$31	2	5+

N	*V*	*-*	*B*	*D*	*I*	**Z**	*C*

+ add 1 cycle if page boundary crossed

EOR

Bitwise Exclusive-OR with Accumulator

Mode	Syntax	HEX	LEN	TIME
Immediate	EOR #$44	$49	2	2
Zero Page	EOR $44	$45	2	3

Zero Page,X	EOR $44,X	$55	2	4
Absolute	EOR $4400	$4D	3	4
Absolute,X	EOR $4400,X	$5D	3	4+
Absolute,Y	EOR $4400,Y	$59	3	4+
(Indirect,X)	EOR ($44,X)	$41	2	6
(Indirect),Y	EOR ($44),Y	$51	2	5+

N	V	-	B	D	I	Z	C

+ add 1 cycle if page boundary crossed

ORA

Bitwise OR with Accumulator

Mode	Syntax	HEX	LEN	TIME
Immediate	ORA #$44	$09	2	2
Zero Page	ORA $44	$05	2	3
Zero Page,X	ORA $44,X	$15	2	4
Absolute	ORA $4400	$0D	3	4
Absolute,X	ORA $4400,X	$1D	3	4+
Absolute,Y	ORA $4400,Y	$19	3	4+
(Indirect,X)	ORA ($44,X)	$01	2	6
(Indirect),Y	ORA ($44),Y	$11	2	5+

N	V	-	B	D	I	Z	C

+ add 1 cycle if page boundary crossed

ASL

Arithmetic Shift Left

ASL shifts all bits left one position.
0 is shifted into bit 0 and the original bit 7 is shifted into the Carry.

Mode	Syntax	HEX	LEN	TIME
Accumulator	ASL A	$0A	1	2
Zero Page	ASL $44	$06	2	5
Zero Page,X	ASL $44,X	$16	2	6
Absolute	ASL $4400	$0E	3	6
Absolute,X	ASL $4400,X	$1E	3	7

N	V	-	B	D	I	Z	C

LSR

Logical Shift Right

LSR shifts all bits right one position.
0 is shifted into bit 7 and the original bit 0 is shifted into the Carry.

Mode	Syntax	HEX	LEN	TIME
Accumulator	LSR A	$4A	1	2
Zero Page	LSR $44	$46	2	5
Zero Page,X	LSR $44,X	$56	2	6
Absolute	LSR $4400	$4E	3	6
Absolute,X	LSR $4400,X	$5E	3	7

N	V	-	B	D	I	Z	C

ROL

Rotate Left

ROL shifts all bits left one position.
The Carry is shifted into bit 0 and the original bit 7 is shifted into the Carry.

Mode	Syntax	HEX	LEN	TIME
Accumulator	ROL A	$2A	1	2
Zero Page	ROL $44	$26	2	5

Zero Page,X	ROL $44,X	$36	2	6
Absolute	ROL $4400	$2E	3	6
Absolute,X	ROL $4400,X	$3E	3	7

N	V	-	B	D	I	Z	C

ROR

Rotate Right

ROR shifts all bits right one position.
The Carry is shifted into bit 7 and the original bit 0 is shifted into the Carry.

Mode	Syntax	HEX	LEN	TIME
Accumulator	ROR A	$6A	1	2
Zero Page	ROR $44	$66	2	5
Zero Page,X	ROR $44,X	$76	2	6
Absolute	ROR $4400	$6E	3	6
Absolute,X	ROR $4400,X	$7E	3	7

N	V	-	B	D	I	Z	C

Branch Instructions

All branches are relative mode and have a length of two bytes. Syntax is "Bxx Displacement" or (better) "Bxx Label". Branches are dependant on the status of the flag bits when the opcode is encountered.

See the notes on the Program Counter for more on displacements.

Mnem	Description	HEX	LEN	TIME
BPL	Branch on Plus	$10	2	2++
BMI	Branch on Minus	$30	2	2++

Mnem	Description	HEX	LEN	TIME
BVC	Branch on Overflow Clear	$50	2	2++
BVS	Branch on Overflow Set	$70	2	2++

Mnem	Description	HEX	LEN	TIME
BCC	Branch on Carry Clear	$90	2	2++
BCS	Branch on Carry Set	$B0	2	2++

Mnem	Description	HEX	LEN	TIME
BNE	Branch on Not Equal	$D0	2	2++
BEQ	Branch on Equal	$F0	2	2++

N	*V*	*-*	*B*	*D*	*I*	*Z*	*C*

A branch not taken requires 2 machine cycles.
A branch requires 3 cycles if taken, plus 1 cycle if the branch cross a page boundary.

Notes

There is no BRA (Branch Always) instruction but it can be easily emulated by branching on the basis of a known condition. One of the best flags to use for this purpose is the Overflow flag which is unchanged by all but the addition and subtraction operations.

A page boundary crossing occurs when the branch destination is on a different page than the instruction AFTER the branch instruction. For example:

```
SEC
BCS LABEL
NOP
```

A page boundary crossing occurs (i.e. the BCS takes 4 cycles) when (the address of) LABEL and the NOP are on different pages. This

means that

```
CLV
BVC LABEL
LABEL NOP
```

the BVC instruction will take 3 cycles no matter what address it is located at.

Compare Instructions

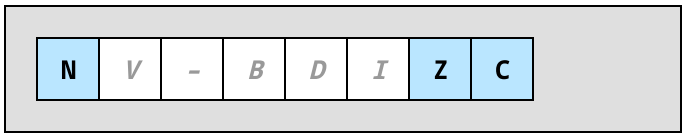
CMP

Compare Accumulator

Compare sets processor flags as if a subtraction had been carried out.

If the accumulator and the compared value are equal, the result of the subtraction is zero and the Zero (Z) flag is set. If the accumulator is equal or greater than the compared value, the Carry (C) flag is set.

Mode	Syntax	HEX	LEN	TIME
Immediate	CMP #$44	$C9	2	2
Zero Page	CMP $44	$C5	2	3
Zero Page,X	CMP $44,X	$D5	2	4
Absolute	CMP $4400	$CD	3	4
Absolute,X	CMP $4400,X	$DD	3	4+
Absolute,Y	CMP $4400,Y	$D9	3	4+
(Indirect,X)	CMP ($44,X)	$C1	2	6
(Indirect),Y	CMP ($44),Y	$D1	2	5+



+ add 1 cycle if page boundary crossed

CPX

Compare X Register

Operation and flag results are identical to equivalent mode accumulator CMP operations.

Mode	Syntax	HEX	LEN	TIME
Immediate	CPX #$44	$E0	2	2
Zero Page	CPX $44	$E4	2	3
Absolute	CPX $4400	$EC	3	4

CPY

Compare Y Register

Operation and flag results are identical to equivalent mode accumulator CMP operations.

Mode	Syntax	HEX	LEN	TIME
Immediate	CPY #$44	$C0	2	2
Zero Page	CPY $44	$C4	2	3
Absolute	CPY $4400	$CC	3	4

BIT

Test Bits

BIT sets the Z flag as though the value in the address tested were ANDed with the accumulator.

The N and V flags are set equal to bits 7 and 6 respectively of the value in the tested address.

Mode	Syntax	HEX	LEN	TIME
Zero Page	BIT $44	$24	2	3
Absolute	BIT $4400	$2C	3	4

N	V	-	B	D	I	Z	C

BIT is often used to skip one or two following bytes as in:

```
CLOSE1 LDX #$10     ; If entered here, we
       .BYTE $2C    ; effectively perform

CLOSE2 LDX #$20     ; a BIT test on $20A2,
       .BYTE $2C    ; another one on $30A2,

CLOSE3 LDX #$30     ; and end up with the X

CLOSEX LDA #12      ; register still at $10
       STA ICCOM,X  ; upon arrival here.
```

Beware: a BIT instruction used in this way as a NOP does have effects: the flags may be modified, and the read of the absolute address, if it happens to access an I/O device, may cause an unwanted action.

Flag Instructions

Instructions that change Processor Status Flags

CLC/SEC

Clear Carry, Set Carry

Mode	Syntax	HEX	LEN	TIME
Implied	CLC	$18	1	2
Implied	SEC	$38	1	2

N	V	-	B	D	I	Z	**C**

CLI/SEI

Clear Interrupt, Set Interrupt

Mode	Syntax	HEX	LEN	TIME
Implied	CLI	$58	1	2
Implied	SEI	$78	1	2

N	V	-	B	D	**I**	Z	C

CLD/SED

Clear Decimal, Set Decimal

Mode	Syntax	HEX	LEN	TIME
Implied	CLD	$D8	1	2
Implied	SED	$F8	1	2

N	V	-	B	**D**	I	Z	C

CLV

Clear Overflow

Mode	Syntax	HEX	LEN	TIME
Implied	CLV	$B8	1	2

N	V	-	B	D	I	Z	C

Jump Instructions

JMP

Jump

JMP loads the program counter with the absolute address, or the address stored at the memory location of the indirect address. Program execution proceeds from the new program counter value.

Mode	Syntax	HEX	LEN	TIME
Absolute	JMP $5597	$4C	3	3
Indirect	JMP ($5597)	$6C	3	5

N	V	-	B	D	I	Z	C

Note that there is no carry associated with the indirect jump so, an indirect JMP ***must never use a vector beginning on the last byte of a page***.

For example, JMP ($30FF) will read the vector low byte from $30FF, but it will read the vector high byte from $3000, NOT from $3100 as you might expect. Unless the vector is intentionally laid out like this, it will likely lead to a crash.

JSR

Jump Saving Return

JSR pushes the address-1 of the next operation to the stack before transferring the value of the argument to the program counter. JSR behaves just like a JMP, but saves the return address to the stack first, thus creating a subroutine.

Subroutines are normally terminated by an RTS instruction.

Mode	Syntax	HEX	LEN	TIME
Absolute	JSR $5597	$20	3	6

N	V	-	B	D	I	Z	C

RTS

Return to Saved

RTS pulls the top two bytes off the stack (low byte first) and transfers them to the program counter. The program counter is incremented by one and then execution proceeds from there.

RTS is typically used in combination with a JSR which saves the return address-1 to the stack.

Mode	Syntax	HEX	LEN	TIME
Implied	RTS	$60	1	6

N	V	-	B	D	I	Z	C

RTS can also be used to implement a jump table where addresses-1 are pushed onto the stack and accessed via RTS.

In this example, the X register is used to select which routine will be executed.

```
LDX #1
JSR EXEC
```

```
        ...
        RTS

EXEC    LDA HIBYTE,X
        PHA
        LDA LOBYTE,X
        PHA
        RTS

LOBYTE  .BYTE <ROUTINE0-1,<ROUTINE1-1
        .BYTE <ROUTINE2-1,<ROUTINE3-1

HIBYTE  .BYTE >ROUTINE0-1,>ROUTINE1-1
        .BYTE >ROUTINE2-1,>ROUTINE3-1
```

RTI

Return from Interrupt

RTI retrieves the Processor Status byte and Program Counter from the stack in that order. Interrupts push the program counter first and then the processor status.

Unlike RTS, the return address on the stack is the actual address rather than the address-1.

Mode	Syntax	HEX	LEN	TIME
Implied	RTI	$40	1	6

N	V	-	B	D	I	Z	C

Math Instructions

ADC

Add with Carry

ADC behavior depends on the state of the decimal flag. In decimal mode, the values upon which the addition is performed are interpreted as packed BCD (Binary Coded Decimal).

Mode	Syntax	HEX	LEN	TIME
Immediate	ADC #$44	$69	2	2
Zero Page	ADC $44	$65	2	3
Zero Page,X	ADC $44,X	$75	2	4
Absolute	ADC $4400	$6D	3	4
Absolute,X	ADC $4400,X	$7D	3	4+
Absolute,Y	ADC $4400,Y	$79	3	4+
(Indirect,X)	ADC ($44,X)	$61	2	6
(Indirect),Y	ADC ($44),Y	$71	2	5+

N	V	-	B	D	I	Z	C

+ add 1 cycle if page boundary crossed

There is no way to add without the carry.

SBC

Subtract with Carry

SBC behavior depends on the state of the decimal flag. In decimal mode, the values upon which the subtraction is performed are interpreted as packed BCD (Binary Coded Decimal).

Mode	Syntax	HEX	LEN	TIME
Immediate	SBC #$44	$E9	2	2
Zero Page	SBC $44	$E5	2	3
Zero Page,X	SBC $44,X	$F5	2	4
Absolute	SBC $4400	$ED	3	4
Absolute,X	SBC $4400,X	$FD	3	4+
Absolute,Y	SBC $4400,Y	$F9	3	4+
(Indirect,X)	SBC ($44,X)	$E1	2	6

(Indirect),Y	SBC ($44),Y	$F1	2	5+

N	V	-	B	D	I	Z	C

+ add 1 cycle if page boundary crossed

There is no way to subtract without the carry.

The carry works as an inverse borrow. i.e., to subtract, first set the carry. If the carry is cleared by the operation, a borrow has occurred.

Memory Instructions

LDA

Load Accumulator

Mode	Syntax	HEX	LEN	TIME
Immediate	LDA #$44	$A9	2	2
Zero Page	LDA $44	$A5	2	3
Zero Page,X	LDA $44,X	$B5	2	4
Absolute	LDA $4400	$AD	3	4
Absolute,X	LDA $4400,X	$BD	3	4+
Absolute,Y	LDA $4400,Y	$B9	3	4+
(Indirect,X)	LDA ($44,X)	$A1	2	6
(Indirect),Y	LDA ($44),Y	$B1	2	5+

N	V	-	B	D	I	Z	C

+ add 1 cycle if page boundary crossed

LDX

Load X Register

Mode	Syntax	HEX	LEN	TIME
Immediate	LDX #$44	$A2	2	2
Zero Page	LDX $44	$A6	2	3
Zero Page,Y	LDX $44,Y	$B6	2	4
Absolute	LDX $4400	$AE	3	4
Absolute,Y	LDX $4400,Y	$BE	3	4+

N	V	-	B	D	I	Z	C

LDY

Load Y Register

Mode	Syntax	HEX	LEN	TIME
Immediate	LDY #$44	$A0	2	2
Zero Page	LDY $44	$A4	2	3
Zero Page,X	LDY $44,X	$B4	2	4
Absolute	LDY $4400	$AC	3	4
Absolute,X	LDY $4400,X	$BC	3	4+

N	V	-	B	D	I	Z	C

STA

Store Accumulator

Mode	Syntax	HEX	LEN	TIME

Zero Page	STA $44	$85	2	3
Zero Page,X	STA $44,X	$95	2	4
Absolute	STA $4400	$8D	3	4
Absolute,X	STA $4400,X	$9D	3	5
Absolute,Y	STA $4400,Y	$99	3	5
(Indirect,X)	STA ($44,X)	$81	2	6
(Indirect),Y	STA ($44),Y	$91	2	6

N	V	-	B	D	I	Z	C

STX

Store X Register

Mode	Syntax	HEX	LEN	TIME
Zero Page	STX $44	$86	2	3
Zero Page,Y	STX $44,Y	$96	2	4
Absolute	STX $4400	$8E	3	4

N	V	-	B	D	I	Z	C

STY

Store Y Register

Mode	Syntax	HEX	LEN	TIME
Zero Page	STY $44	$84	2	3
Zero Page,X	STY $44,X	$94	2	4
Absolute	STY $4400	$8C	3	4

N	V	-	B	D	I	Z	C

DEC

Decrement Memory

Mode	Syntax	HEX	LEN	TIME
Zero Page	DEC $44	$C6	2	5
Zero Page,X	DEC $44,X	$D6	2	6
Absolute	DEC $4400	$CE	3	6
Absolute,X	DEC $4400,X	$DE	3	7

N	V	-	B	D	I	Z	C

INC

Increment Memory

Mode	Syntax	HEX	LEN	TIME
Zero Page	INC $44	$E6	2	5
Zero Page,X	INC $44,X	$F6	2	6
Absolute	INC $4400	$EE	3	6
Absolute,X	INC $4400,X	$FE	3	7

N	V	-	B	D	I	Z	C

Register Instructions

These instructions are implied mode, have a length of one byte and require two machine cycles.

Mnem	Description	HEX	LEN	TIME
TAX	Transfer A to X	$AA	1	2
TAY	Transfer A to Y	$A8	1	2
TXA	Tranfer X to A	$8A	1	2
TYA	Transfer Y to A	$98	1	2

N	V	-	B	D	I	Z	C

Mnem	Description	HEX	LEN	TIME
DEX	Decrement X	$CA	1	2
DEY	Decrement Y	$88	1	2
INX	Increment X	$E8	1	2
INY	Increment Y	$C8	1	2

N	V	-	B	D	I	Z	C

Stack Instructions

These instructions are implied mode, have a length of one byte and require machine cycles as indicated. The "PULL" operations are known as "POP" on most other microprocessors.

With the 6502, the stack is always on page $01 ($0100-$01FF) and works top down. i.e., when you push to the stack the stack pointer is decremented. When you pull from the stack the stack pointer is incremented.

Mnem	Description	HEX	LEN	TIME
PHA	Push Accumulator	$48	1	3
PHP	Push Processor Status	$08	1	3

TXS	Transfer X to Stack Pointer	$9A	1	2

N	*V*	*-*	*B*	*D*	*I*	*Z*	*C*

Mnem	Description	HEX	LEN	TIME
PLA	Pull Accumulator	$68	1	4
TSX	Transfer Stack Pointer to X	$BA	1	2

N	*V*	*-*	*B*	*D*	*I*	**Z**	*C*

Mnem	Description	HEX	LEN	TIME
PLP	Pull Processor Status	$28	1	4

N	**V**	*-*	**B**	**D**	**I**	**Z**	**C**

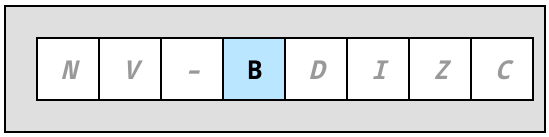
Other Instructions

BRK

Break

BRK sets the B flag, and then generates a forced interrupt. The Interrupt flag is ignored and the CPU goes through the normal interrupt process. In the interrupt service routine, the state of the B flag can be used to distinguish a BRK from a standard interrupt.

Mode	Syntax	HEX	LEN	TIME
Implied	BRK	$00	1	7



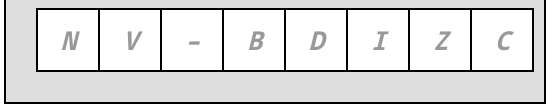
BRK causes a non-maskable interrupt and increments the program counter by one. Therefore an RTI will go to the address of the BRK +2 so that BRK may be used to replace a two-byte instruction for debugging and the subsequent RTI will be correct.

NOP

No Operation

A NOP takes 2 machine cycles to execute, but it has no effect on any register, memory location, or processor flag. Thus, it takes up time and space but performs no operation.

Mode	Syntax	HEX	LEN	TIME
Implied	NOP	$EA	1	2



NOP can be used to reserve space for future modifications or to remove existing code without changing the memory locations of code that follows it.

NOP can also be used in tightly timed code, to idly take up 2 cycles without having any other side effects.

The original source for the above content is: http://6502.org/tutorials/6502opcodes.html

In case there are any errors in my rendition, or in the 6502.org source, here is the relevant section from the official C64 Programmer's Reference Guide: Chapter 5: Basic to Machine Language (PDF)

Do you like what you see?

You've just read one of my high-quality, long-form, weblog posts, for free! First, thank you for your interest, it makes producing this content feel worthwhile. I love to hear your input and feedback in the forums below. And I do my best to answer every question.

I'm creating C64 OS and documenting my progress along the way, to give something to you and contribute to the Commodore community. Please consider purchasing one of the items I am currently offering or making a small donation, to help me continue to bring you updates, in-depth technical discussions and programming reference. Your generous support is greatly appreciated.

Greg Naçu — C64OS.com

Want to support my hard work? Here's how!

Content

Welcome Page
Order C64 OS
C64 OS User's Guide
C64 OS System Updates
Weblog Full Archive
Commodore 8-Bit Buyer's Guide

Search c64os.com
Search

Legal

Terms & Conditions
Privacy Policy
Cookies Policy
C64 OS License Agreement
Thanks and Credits

About

OpCoders Inc.
Gregory Nacu
Contact Us

Media Resources