illNES Documentation

Release 0.1

Jonathan Couldridge

Oct 08, 2017

Contents:

1 Rockwell 6502 Programmers Reference **3**
- 1.1 Quick Reference . . . 4
- 1.2 Instruction set Lookup Table . . . 4
- 1.3 Detailed Instruction Reference . . . 4
- 1.4 Instructions by Purpose . . . 7
- 1.5 Processor Status Word (Flags) . . . 7
- 1.6 Binary Coded Decimal (BCD) . . . 7
- 1.7 Addressing Modes . . . 7
- 1.8 6502 Memory Map . . . 7
- 1.9 The Stack . . . 7
- 1.10 Program Flow Control . . . 7
- 1.11 Interrupts and Interrupt ReQuests (IRQ's) . . . 7
- 1.12 System Startup sequence (boot-strap) . . . 7
- 1.13 Bibliography . . . 7

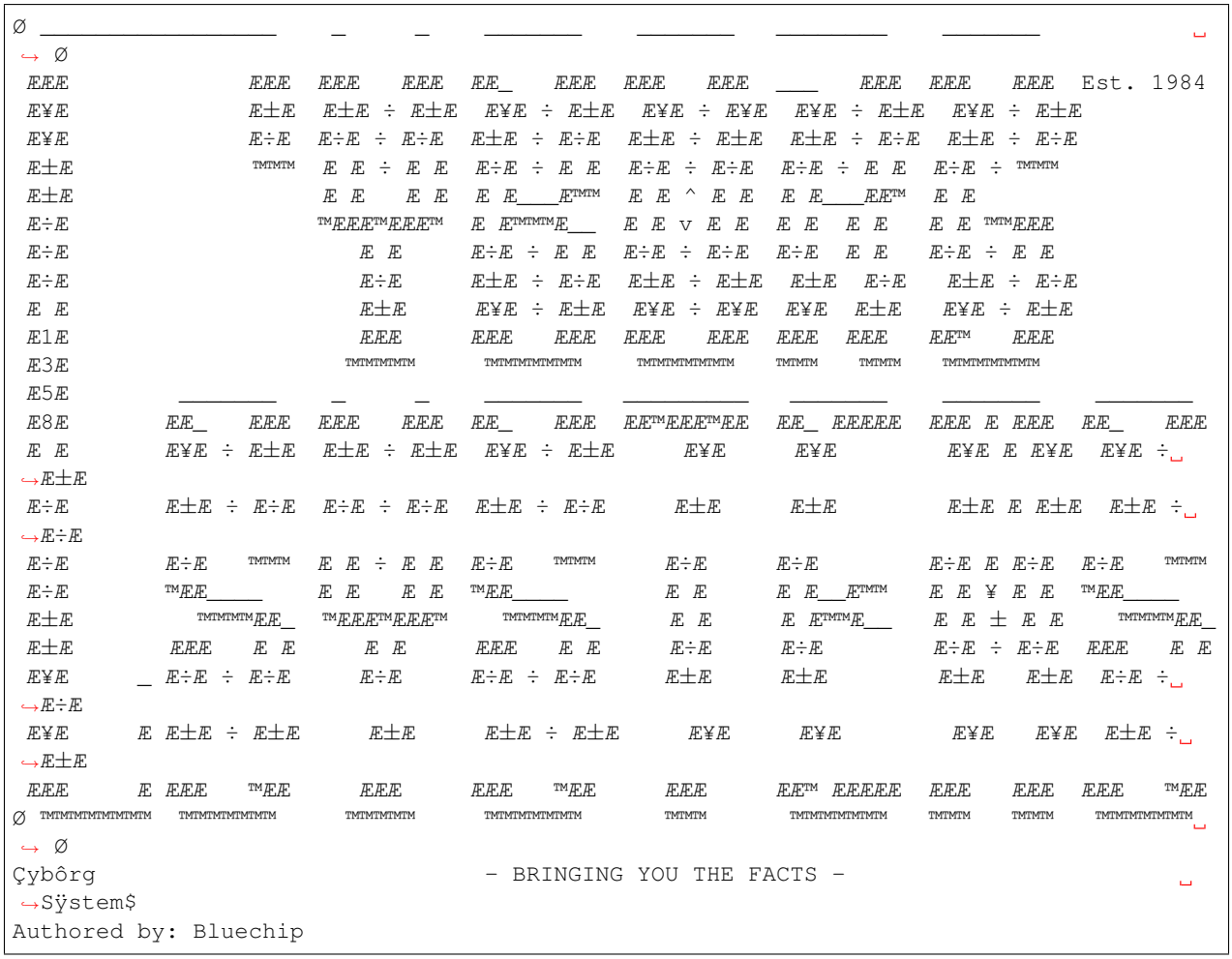
2 Indices and tables **9**

Note: This is a conversion of the Cyborg Systems 6502 documentation.

original source url: *http://homepage.ntlworld.com/cyborgsystems/CS_Main/6502/6502.htm*

CHAPTER 1

Rockwell 6502 Programmers Reference



Quick Reference

Instruction set Lookup Table

I-Len	T-Cnt
Mnemonic	
Address Mode	

I-Len Instruction length. The number of bytes of memory required to store the instruction.

T-Cnt Timing Cycle (T-State) count. The number of clock cycles required to execute the instruction.

Mnemonic The abbreviated "human readable" identity of the instruction.

Address Mode The *memory addressing mode* used by the instruction.

The opcode for any instruction may be composed by reading the row and column it is in:

- `ASL` in **Absolute** addressing mode is in row `1-`, column `-E`
- Therefore it's opcode (hexadecimal representation) is `1E` (or `$1E` or `0x1E`)

Some instructions (or more specifically *Addressing Modes*) require either one or two bytes of data as a parameter.

Under these circumstances, this data immediately follows the instruction itself.

Mnemonic Code	Assembled Code
`PLA`	`$68`
`BEQ $03`	`$F0 $9F`
`JMP $A5B6`	`$4C $B6 $A5`

Key to Addressing modes:

Implied	**Accumulator**	**Immediate**	**Zero Page**
-	A	#	0
Absolute	**Indirect**	**X-indexed**	**Y-indexed**
/	>	X	Y

Note: The Cyborg Systems docs featured an incomplete table, so this is currently ommitted from these docs until a nice way to fit them into readthedocs is figured out.

The wikipedia table can be used in the meantime.

Detailed Instruction Reference

Note:

- Uppercase flags are affected by the instruction.
- `M` refers to the value retrieved from memory, subject to the addressing mode.
- `.N` where `N` is `0-7` refers to a single bit of a byte value. e.g. `A.7` is bit 7 of the Accumulator register.
- lower case letters are local variables e.g. `t`.

- Since these docs are for a codebase that uses them, the actual CPU code may be more helpful than the pseudo-code presented here. The original logic is here in the name of preservation.

ADC

ADC	Add Memory to A with Carry							
Flags	N	V	-	b	d	i	Z	C

Address Mode	Syntax	OpCode	I-Len	T-Cnt
Immediate	`ADC #$A5`	`$69`	2	2
Zero Page	`ADC $A5`	`$65`	2	3
Zero Page,X	`ADC $A5,X`	`$75`	2	4
Absolute	`ADC $A5B6`	`$6D`	3	4
Absolute,X	`ADC $A5B6,X`	`$7D`	3	4+
Absolute,Y	`ADC $A5B6,Y`	`$79`	3	4+
(Indirect,X)	`ADC ($A5,X)`	`$61`	2	6
(Indirect),Y	`ADC ($A5),Y`	`$71`	2	5+
	+ Add 1 (one) T-State if a Page Boundary is crossed			

```
t = A + M + P.C
P.V = (A.7!=t.7) ? 1:0
P.N = A.7
P.Z = (t==0) ? 1:0
IF (P.D)
    t = bcd(A) + bcd(M) + P.C
    P.C = (t>99) ? 1:0
ELSE
    P.C = (t>255) ? 1:0
A = t & 0xFF
```

- There is no add-without-carry instruction!
- As the result of ADC depends on the contents of the *Carry Flag (P.C)*
- When performing "single precision" (or "8 bit") arithmetic it is often necessary to ensure that the *Carry Flag (P.C)* is CLEARed with a *CLC* command before the ADC is executed ...to ensure that the Carry Flag (P.C) has no bearing on the result.
- CLEARing the *Carry Flag (P.C)* with a *CLC* command is normally necessary before the first stage of a "multiple precision" (or "more than 8 bit") addition.
- The action of ADC is dependant on the setting of *Decimal Flag (P.D)*

AND

AND	Bitwise-AND A with Memory							
Flags	N	v	-	b	d	i	Z	c

Address Mode	Syntax	OpCode	I-Len	T-Cnt
Immediate	`AND #$A5`	`$29`	2	2
Zero Page	`AND $A5`	`$25`	2	2
Zero Page,X	`AND $A5,X`	`$35`	2	3
Absolute	`AND $A5B6`	`$2D`	3	4
Absolute,X	`AND $A5B6,X`	`$3D`	3	4+
Absolute,Y	`AND $A5B6,Y`	`$39`	3	4+
(Indirect,X)	`AND ($A5,X)`	`$21`	2	6
(Indirect),Y	`AND ($A5),Y`	`$31`	2	5+
	+ Add 1 (one) T-State if a Page Boundary is crossed			

```
A = A & M
P.N = A.7
P.Z = (A==0) ? 1:0
```

AND	0	1		1 0 1 0 AND
0	0	0		1 1 0 0 =
1	0	1		1 0 0 0

ASL

ASL	Arithmetic Shift Left							
Flags	N	v	-	b	d	i	Z	C

Address Mode	Syntax	OpCode	I-Len	T-Cnt
Accumulator	`ASL A`	`$0A`	1	2
Zero Page	`ASL $A5`	`$06`	2	5
Zero Page,X	`ASL $A5,X`	`$16`	2	6
Absolute	`ASL $A5B6`	`$0E`	3	6
Absolute,X	`ASL $A5B6,X`	`$1E`	3	7

```
P.C = B.7
B = (B << 1) & $FE
P.N = B.7
P.Z = (B==0) ? 1:0
```

`P.C <- b.7 <- b.6 <- b.5 <- b.4 <- b.3 <- b.2 <- b.1 <- b.0 <- 0`

Before:	P.C = ?	B = 1 1 1 0 1 1 1 0
After:	P.C = 0	B = 1 1 0 1 1 1 0 0

- **In my experience, this is NOT an "arithmetic" shift**
  - An Arithmetic shift *normally* preserves the Most Significant Bit (MSb) or "Sign bit" of the source value
  - ASL does NOT do this on the 6502.
  - The 6502 places a copy of the sign from the result of a *Logical Shift Left* into the *sigN Flag (P.N)*
- This instruction would be better named as `SLS` (logical Shift Left and update Sign)

CLC

Instructions by Purpose

Processor Status Word (Flags)

Note: Until this document is complete, the titles are here to ensure cross-referencing from other documents works.

C

D

N

Binary Coded Decimal (BCD)

Addressing Modes

6502 Memory Map

The Stack

Program Flow Control

Interrupts and Interrupt ReQuests (IRQ's)

System Startup sequence (boot-strap)

Bibliography

Please ask questions or report any errors you may find here

CHAPTER 2

Indices and tables

- genindex
- modindex