R6502 IP Core Specification

Author: Jens Gutschmidt
opencores@vivare-services.com

Rev. 0.7
September 15, 2018

This page has been intentionally left blank.

Revision History

Rev.	Date	Author	Description
0.1	12/18/06	Jens Gutschmidt	First Draft
0.2	01/02/07	Jens Gutschmidt	Pictures of FSM‘s
0.3	08/20/08	Jens Gutschmidt	- Tables and timing diagrams - Deleting pictures of FSM
0.4	10/01/08	Jens Gutschmidt	- New ideas for timing diagrams
0.5	01/02/09	Jens Gutschmidt	- Textual changes / spell checking - Insert R6502_TC block diagram
0.6	02/01/09	Jens Gutschmidt	- Work on Timing Diagrams
0.7	11/09/18	Jens Gutschmidt	- Work on Timing Diagrams - Corrected all branch descriptions (page crossing computed wrong) - PHP writes always E&B = ‘1’to stack - Adding Interrupt section - Adding ADC / SBC Decimal Mode

Contents

.

LIST OF FIGURES

LIST OF TABLES

1 Introduction

The Central Processing Unit (CPU) 6502 was introduced at 1976 by Commodore Computers. It is an 8 Bit processor which is well known worldwide. The also most known computer system in the 70s/80s was the APPLE based on this famous 6502.

In this century building of little computer systems based on an 8 Bit architecture is easier than ever before. Many CAD/CAM tool are existing on the market to give you help to do this job.

In the last decade the technology give us more and more possibilities to reach our dreams – higher, faster, wider. FPGA's (Field Programmable Gate Arrays) and CPLD's (Complex Programmable Logic Devices) are coming up. They shortening the time of development, simulation and verification of such systems dramatically. On the other hand stands the rising complexity of designs and well knowing of desired description languages – VHDL (Very high scale integration Hardware Description Language) or Verilog.

Many operations and functions of computer tasks aren't need really the power of "modern" processors today like Intel's Pentium. Nevertheless it is not usable – some times also impossible - to build simple systems with discrete IC's. The necessity of flexibility compel us the using of high tech parts and tools...to build a "simple" system.

The using of standard IP's (Intellectual Properties) is the key to reach that goal.

Now talking about the specification of the **6502 True Cycle Core IP**. Written in VHDL but developed with Mentor's HDL Designer. The 6502 is very easy to handle for people were have experience with other cpu's. Also suitable for beginners who will learning to build her own first cpu system.

2 Architecture

The following figure shows the internal architecture of the Commodore's 6502.

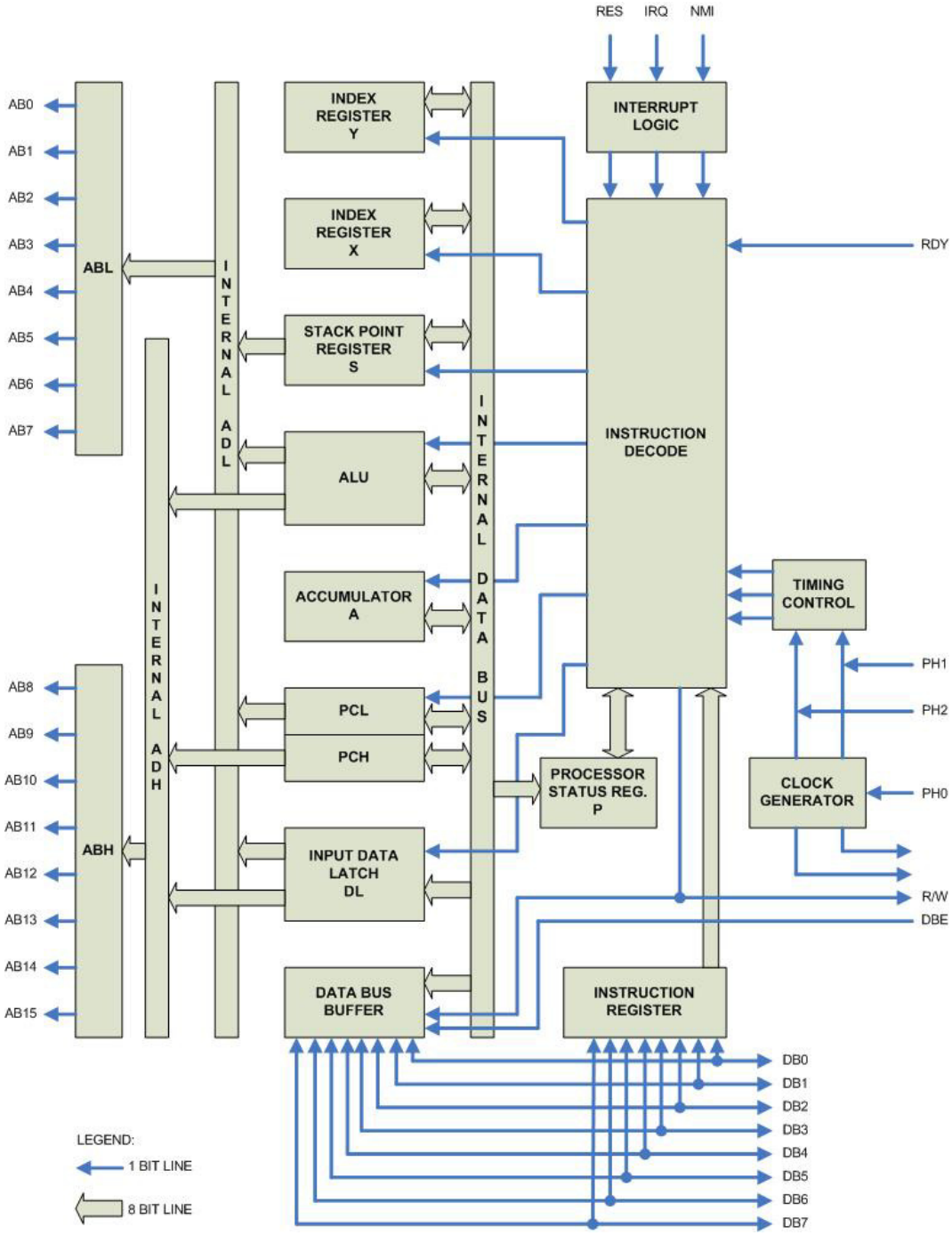


Figure (1): 6502 architecture

First, for more clearance and better understanding of the **6502 True Cycle Core IP** let me summarize the most important quality of Commodore's 6502.

This architecture based on asynch logic. The two phase clock is a special attribute of that. The Data Bus is only active while clock phase PH1 is "1" for reading and writing (PH0 is "0"). All transfers of data are occur at active phase of PH1.

The whole 6502 is fully statically, so all registers holding her values if the clocking will be going to D.C. Internally all operation of 16 Bit are split into two 8 Bit busses. This is very interesting and causes one more clock cycle in some operations (see "PCL", "PCH" -> Program Counter Low/High) which can operate over page boundaries.

Address lines are 16 Bit wide. 65.536 addresses for byte access are possible. There are no differences exist between memory and I/O cycles. All I/O devices must be connected via memory mapped I/O. The mnemonics are also all the same for both. The usage of address space is normally organized form top to the button "ROM – I/O – RAM" (0xFFFF – 0x0000).

Only one pin R/$\overline{W}$ controls the read and write operations.

The 6502 can hold in every cycle – except write cycles - by applying RDY. So wait states are generate able for any read cycle.

There are two interrupts $\overline{IRQ}$ and $\overline{NMI}$. $\overline{IRQ}$ is mask able, $\overline{NMI}$ is not. Note that $\overline{RES}$ is internally handled as an interrupt, but is resetting the 6502 to the starting point of operation at vector address 0xFFFF. Every interrupt has its own vector which can point elsewhere to the 16 Bit address space. The vectors are located from 0xFFFF to 0xFFFA.

The Stack Pointer is 256 Bytes deep and is hard-wired to 0x01FF (internally 0x0200 –0x01FF going to the address bus) for the first memory location and grows downward to 0x0100. Decrementing again will produce a wrap around back to 0x01FF.

Arithmetical operations like ADC and SBC can handle binary values from 0-255 and decimal values from 0-99 without using other op codes. Only one user changeable bit into the Status Register determine the exact arithmetic mode of that operations.

The 6502 generally operates into a pipeline mode. That means that a finish of an OP code falls every time into a fetch cycle of the next OP. This save one clock cycle

Every operation consume between two and seven clock cycles.

To build an useful und backward software and hardware compatible VHDL Core for the 6502 many unavoidable changes may be forced to simulation and verification before any real core will be made.

The requirements:

- Vendor in depend for implementation in any FPGA
- True Cycle for all operations as described in original publications
- No “Mixed Mode” – Only 6502, not 65C02 -> for performance and area
- No fantastic or useful extensions – as like the original 6502 as is meaningful
- Implement the “undocumented OP’s” all as NOP
- Only one clock
- Full synch design
- Operating speed up to 60 MHz – or higher
- Easily to change for future requirement – building variants
- The activity on data and address busses since the execution of OP codes is not forced to be like the original – may be or may be not.

The design based on finite state machines (fsm). Every OP code has its own fsm. Some registers are for internally use only and stores temporally values. These registers are not accessible by the programmer.

Some fsm own registers for internally use and don’t share this registers with other fsm. This help to decrease fan-in and increase performance.

The next picture shows the hierarchical structure of the R6502 IP core.

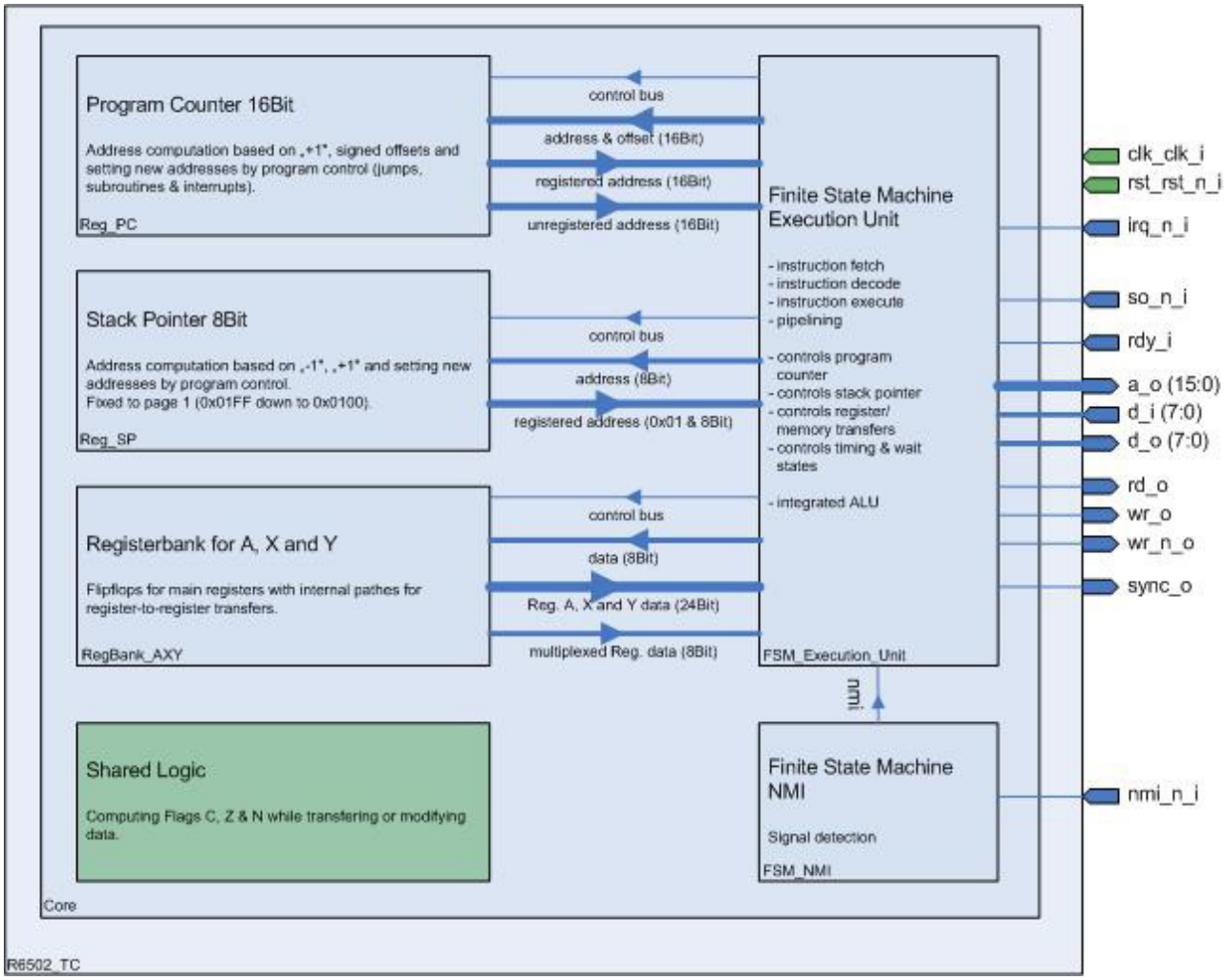


Figure (2): R6502_TC IP core architecture

3 Operation

ADC

Operation: Add memory or immediate value to accumulator A with carry (A + M + C → A, C)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Immediate	ADC # Oper	69	2	2	√	√	-	-	-	-	√	√
Zero Page	ADC Oper	65	2	3	√	√	-	-	-	-	√	√
Zero Page, X	ADC Oper, X	75	2	4	√	√	-	-	-	-	√	√
Absolute	ADC Oper	6D	3	4	√	√	-	-	-	-	√	√
Absolute, X	ADC Oper, X	7D	3	4*	√	√	-	-	-	-	√	√
Absolute, Y	ADC Oper, Y	79	3	4*	√	√	-	-	-	-	√	√
(Indirect, X)	ADC (Oper, X)	61	2	6	√	√	-	-	-	-	√	√
(Indirect), Y	ADC (Oper), Y	71	2	5*	√	√	-	-	-	-	√	√

* => Add 1 if page boundary is crossed

Table (1): ADC – Short Reference

Example Immediate (assumed A=$FE, C=0):

Address	Bytes	Mnemonic	
$9000	69 03	ADC #$03	;
$9002	…	*next OP*	; A is now $01, C=1, N=0, Z=0, V=0

```
        1111 1110 ($FE)
ADC     0000 0011 ($03)
        0000 0001 ($01), C=1, N=0, Z=0, V=0
```

AND

Operation: "AND" memory or immediate value with accumulator A (A ∩ M → A)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Immediate	AND # Oper	29	2	2	√	-	-	-	-	-	√	-
Zero Page	AND Oper	25	2	3	√	-	-	-	-	-	√	-
Zero Page, X	AND Oper, X	35	2	4	√	-	-	-	-	-	√	-
Absolute	AND Oper	2D	3	4	√	-	-	-	-	-	√	-
Absolute, X	AND Oper, X	3D	3	4*	√	-	-	-	-	-	√	-
Absolute, Y	AND Oper, Y	39	3	4*	√	-	-	-	-	-	√	-
(Indirect, X)	AND (Oper, X)	21	2	6	√	-	-	-	-	-	√	-
(Indirect), Y	AND (Oper), Y	31	2	5*	√	-	-	-	-	-	√	-

* => Add 1 if page boundary is crossed

Table (2): AND – Short Reference

Example Immediate (assumed A is $C5):

Address	Bytes	Mnemonic	
$9000	29 71	AND #$71	;
$9002	…	*next OP*	; A is now $41, N=0, Z=0

```
        1100 0101 ($C5)
AND     0111 0001 ($71)
        0100 0001 ($41), N=0, Z=0
```

ASL

Operation: Shift Left One Bit (Memory or Accumulator) (C ← 7 6 5 4 3 2 1 0 ← 0)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Accumulator	ASL A	0A	1	2	√	-	-	-	-	-	√	√
Zero Page	ASL Oper	06	2	5	√	-	-	-	-	-	√	√
Zero Page, X	ASL Oper, X	16	2	6	√	-	-	-	-	-	√	√
Absolute	ASL Oper	0E	3	6	√	-	-	-	-	-	√	√
Absolute, X	ASL Oper, X	1E	3	7	√	-	-	-	-	-	√	√

Table (3): ASL – Short Reference

Example Accumulator (assumed A=$55):

Address	Bytes	Mnemonic	
$9000	0A	ASL A	;
$9001	…	*next OP*	; A is now $AA, N=1, Z=0

ASL 0101 0101 ($55)
1010 1010 ($AA), C=0, N=1, Z=0

BCC

Operation: Branch on Carry Clear (Branch on C = 0)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Relative	BCC Oper	90	2	2*	-	-	-	-	-	-	-	-

* => Add 1 if branch occurs to same page

* => Add 2 if branch occurs to different page

Table (4): BCC – Short Reference

Example Relative (assumed C=1, no branch):

```
Address     Bytes     Mnemonic
$9000       90 10     BCC $10     ; BCC to same page, - NO BRANCH - jumps to $9002
$9002       …         next OP     ; C=1 => no branch, 2 cycles
```

Example Relative (assumed C=1, no branch):

```
Address     Bytes     Mnemonic
$90FF       90 10     BCC $10     ; BCC to same page, - NO BRANCH - jumps to
                                  ; $9101
$9101       …         next OP     ; C=1 => no branch, 2 cycles
```

Example Relative (assumed C=0, branch to same page):

```
Address     Bytes     Mnemonic
$9000       90 10     BCC $10     ; BCC to same page, - BRANCH - jumps to $9002 + $10
  …
$9012       …         next OP     ; C=0 => branch, 3 cycles
```

Example Relative (assumed C=0, branch to different page):

```
Address     Bytes     Mnemonic
$90FD       90 10     BCC $10     ; BCC to different page, - BRANCH - jumps to
  …                               ; $9109
$9109       …         next OP     ; C=0 => branch, 4 cycles
```

BCS

Operation: Branch on Carry Set (Branch on C = 1)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Relative	BCS Oper	B0	2	2*	-	-	-	-	-	-	-	-

* => Add 1 if branch occurs to same page

* => Add 2 if branch occurs to different page

Table (5): BCS – Short Reference

Example Relative (assumed C=0, same page):

```
Address     Bytes     Mnemonic
$9000       B0 10     BCS $10     ; BCS to same page, - NO BRANCH - jumps to $9002
$9002       …         next OP     ; C=0 => no branch, 2 cycles
```

Example Relative (assumed C=0, different page):

```
Address     Bytes     Mnemonic
$90FF       B0 10     BCS $10     ; BCS to different page, - NO BRANCH - jumps to
                                  ; $9101
$9101       …         next OP     ; C=0 => no branch, 2 cycles (!!!)
```

Example Relative (assumed C=1, same page):

```
Address     Bytes     Mnemonic
$9000       B0 10     BCS $10     ; BCS to same page, - BRANCH - jumps to $9002 + $10
  …
$9012       …         next OP     ; C=1 => branch, 3 cycles
```

Example Relative (assumed C=1, different page):

```
Address     Bytes     Mnemonic
$90FD       B0 10     BCS $10     ; BCS to different page, - BRANCH - jumps to
  …                               ; $910F
$910F       …         next OP     ; C=1 => branch, 4 cycles
```

BEQ

Operation: Branch on result zero (Branch on Z = 1)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Relative	BEQ Oper	F0	2	2*	-	-	-	-	-	-	-	-

* => Add 1 if branch occurs to same page

* => Add 2 if branch occurs to different page

Table (6): BEQ – Short Reference

Example Relative (assumed Z=0, same page):

```
Address     Bytes     Mnemonic
$9000       F0 10     BEQ $10     ; BEQ to same page, - NO BRANCH - jumps to $9002
$9002       …         next OP     ; Z=0 => no branch, 2 cycles
```

Example Relative (assumed Z=0, different page):

```
Address     Bytes     Mnemonic
$90FF       F0 10     BEQ $10     ; BEQ to different page, - NO BRANCH - jumps to
                                  ; $9101
$9101       …         next OP     ; Z=0 => no branch, 2 cycles (!!!)
```

Example Relative (assumed Z=1, same page):

```
Address     Bytes     Mnemonic
$9000       F0 10     BEQ $10     ; BEQ to same page, - BRANCH - jumps to $9002 + $10
  …
$9012       …         next OP     ; Z=1 => branch, 3 cycles
```

Example Relative (assumed Z=1, different page):

```
Address     Bytes     Mnemonic
$90FD       F0 10     BEQ $10     ; BEQ to different page, - BRANCH - jumps to
  …                               ; $910F
$910F       …         next OP     ; Z=1 => branch, 4 cycles
```

BIT

Operation: Bit 6 and 7 are transferred to the status register. If the result of A ∩ M then Z = 1, otherwise Z = 0 (A ∩ M → A, M7 → N, M6 → V)

Addressing Mode	Assembly Language Form		OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Zero Page	BIT	Oper	24	2	3	7	6	-	-	-	-	√	-
Absolute	BIT	Oper	2C	3	4	7	6	-	-	-	-	√	-

Table (7): BIT – Short Reference

BMI

Operation: Branch on result minus (Branch on N = 1)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Relative	BMI Oper	30	2	2*	-	-	-	-	-	-	-	-

* => Add 1 if branch occurs to same page

* => Add 2 if branch occurs to different page

Table (8): BMI – Short Reference

Example Relative (assumed Z=0, same page):

```
Address     Bytes     Mnemonic
$9000       30 10     BMI $10     ; BMI to same page, - NO BRANCH - jumps to $9002
$9002       …         next OP     ; N=0 => no branch, 2 cycles
```

Example Relative (assumed Z=0, different page):

```
Address     Bytes     Mnemonic
$90FF       30 10     BMI $10     ; BMI to different page, - NO BRANCH - jumps to
                                  ; $9101
$9101       …         next OP     ; N=0 => no branch, 2 cycles (!!!)
```

Example Relative (assumed Z=1, same page):

```
Address     Bytes     Mnemonic
$9000       30 10     BMI $10     ; BMI to same page, - BRANCH - jumps to $9002 + $10
  …
$9012       …         next OP     ; N=1 => branch, 3 cycles
```

Example Relative (assumed Z=1, different page):

```
Address     Bytes     Mnemonic
$90FD       30 10     BMI $10     ; BMI to different page, - BRANCH - jumps to
  …                               ; $910F
$910F       …         next OP     ; N=1 => branch, 4 cycles
```

BNE

Operation: Branch on result not zero (Branch on Z = 0)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Relative	BNE Oper	D0	2	2*	-	-	-	-	-	-	-	-

* => Add 1 if branch occurs to same page

* => Add 2 if branch occurs to different page

Table (9): BNE – Short Reference

Example Relative (assumed Z=1, same page):

```
Address     Bytes     Mnemonic
$9000       D0 10     BNE $10     ; BNE to same page, - NO BRANCH - jumps to $9002
$9002       …         next OP     ; Z=1 => no branch, 2 cycles
```

Example Relative (assumed Z=1, different page):

```
Address     Bytes     Mnemonic
$90FF       D0 10     BNE $10     ; BNE to different page, - NO BRANCH - jumps to
                                  ; $9101
$9101       …         next OP     ; Z=1 => no branch, 2 cycles (!!!)
```

Example Relative (assumed Z=0, same page):

```
Address     Bytes     Mnemonic
$9000       D0 10     BNE $10     ; BNE to same page, - BRANCH - jumps to $9002 + $10
  …
$9012       …         next OP     ; Z=0 => branch, 3 cycles
```

Example Relative (assumed Z=0, different page):

```
Address     Bytes     Mnemonic
$90FD       D0 10     BNE $10     ; BNE to different page, - BRANCH - jumps to
  …                               ; $910F
$910F       …         next OP     ; Z=0 => branch, 4 cycles
```

BPL

Operation: Branch on result plus (Branch on N = 0)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Relative	BPL Oper	10	2	2*	-	-	-	-	-	-	-	-

* => Add 1 if branch occurs to same page

* => Add 2 if branch occurs to different page

Table (10): BPL – Short Reference

Example Relative (assumed N=1, same page):

```
Address     Bytes      Mnemonic
$9000       10 10      BPL $10     ; BPL to same page, - NO BRANCH - jumps to $9002
$9002       …          next OP     ; N=1 => no branch, 2 cycles
```

Example Relative (assumed N=1, different page):

```
Address     Bytes      Mnemonic
$90FF       10 10      BPL $10     ; BPL to different page, - NO BRANCH - jumps to
                                   ; $9101
$9101       …          next OP     ; N=1 => no branch, 2 cycles (!!!)
```

Example Relative (assumed N=0, same page):

```
Address     Bytes      Mnemonic
$9000       10 10      BPL $10     ; BPL to same page, - BRANCH - jumps to $9002 + $10
  …
$9012       …          next OP     ; N=0 => branch, 3 cycles
```

Example Relative (assumed N=0, different page):

```
Address     Bytes      Mnemonic
$90FD       10 10      BPL $10     ; BPL to different page, - BRANCH - jumps to
  …                                ; $910F
$910F       …          next OP     ; N=0 => branch, 4 cycles
```

BRK

Operation: Forced Interrupt (PC + 2 ↓, P ↓ NV11DIZC, E & B are written as 1 to stack)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Implied	BRK	00	1	7	-	-	-	-	-	1	-	-

Table (11): BRK – Short Reference

BVC

Operation: Branch on no overflow (Branch on V = 0)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Relative	BVC Oper	50	2	2*	-	-	-	-	-	-	-	-

* => Add 1 if branch occurs to same page

* => Add 2 if branch occurs to different page

Table (12): BVC – Short Reference

Example Relative (assumed V=1, same page):

```
Address     Bytes     Mnemonic
$9000       50 10     BVC $10     ; BVC to same page, - NO BRANCH - jumps to $9002
$9002       …         next OP     ; V=1 => no branch, 2 cycles
```

Example Relative (assumed V=1, different page):

```
Address     Bytes     Mnemonic
$90FF       50 10     BVC $10     ; BVC to different page, - NO BRANCH - jumps to
                                  ; $9101
$9101       …         next OP     ; V=1 => no branch, 2 cycles (!!!)
```

Example Relative (assumed V=0, same page):

```
Address     Bytes     Mnemonic
$9000       50 10     BVC $10     ; BVC to same page, - BRANCH - jumps to $9002 + $10
  …
$9012       …         next OP     ; V=0 => branch, 3 cycles
```

Example Relative (assumed V=0, different page):

```
Address     Bytes     Mnemonic
$90FD       50 10     BVC $10     ; BVC to different page, - BRANCH - jumps to
  …                               ; $910F
$910F       …         next OP     ; V=0 => branch, 4 cycles
```

BVS

Operation: Branch on overflow (Branch on V = 1)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Relative	BVS Oper	70	2	2*	-	-	-	-	-	-	-	-

* => Add 1 if branch occurs to same page

* => Add 2 if branch occurs to different page

Table (13): BVS – Short Reference

Example Relative (assumed V=0, same page):

```
Address     Bytes       Mnemonic
$9000       70 10       BVS $10     ; BVS to same page, - NO BRANCH - jumps to $9002
$9002       …           next OP     ; V=0 => no branch, 2 cycles
```

Example Relative (assumed V=0, different page):

```
Address     Bytes       Mnemonic
$90FF       70 10       BVS $10     ; BVS to different page, - NO BRANCH - jumps to
                                    ; $9101
$9101       …           next OP     ; V=0 => no branch, 2 cycles (!!!)
```

Example Relative (assumed V=1, same page):

```
Address     Bytes       Mnemonic
$9000       70 10       BVS $10     ; BVS to same page, - BRANCH - jumps to $9002 + $10
  …
$9012       …           next OP     ; V=1 => branch, 3 cycles
```

Example Relative (assumed V=1, different page):

```
Address     Bytes       Mnemonic
$90FD       70 10       BVS $10     ; BVS to different page, - BRANCH - jumps to
  …                                 ; $910F
$910F       …           next OP     ; V=1 => branch, 4 cycles
```

CLC

Operation: Clear Carry flag (0 → C)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Implied	CLC	18	1	2	-	-	-	-	-	-	-	0

Table (14): CLC – Short Reference

Example:

```
Address     Bytes     Mnemonic
$9000       18        CLC        ;
$9001       …         next OP    ; C is now 0
```

CLD

Operation: Clear decimal flag (0 → D)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Implied	CLD	D8	1	2	-	-	-	-	0	-	-	-

Table (15): CLD – Short Reference

Example:

```
Address     Bytes     Mnemonic
$9000       D8        CLD        ;
$9001       …         next OP    ; D is now 0
```

CLI

Operation: Clear interrupt disable flag (0 → I)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Implied	CLI	58	1	2	-	-	-	-	-	0	-	-

Table (16): CLI – Short Reference

Example:

Address	Bytes	Mnemonic	
$9000	58	CLI	;
$9001	…	*next OP*	; I is now 0

CLV

Operation: Clear overflow flag (0 → O)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Implied	CLV	B8	1	2	-	0	-	-	-	-	-	-

Table (17): CLV – Short Reference

Example:

Address	Bytes	Mnemonic	
$9000	B8	CLV	;
$9001	…	*next OP*	; V is now 0

CMP

Operation: Compare Memory and Accumulator (A - M)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Immediate	CMP # Oper	C9	2	2	√	-	-	-	-	-	√	√
Zero Page	CMP Oper	C5	2	3	√	-	-	-	-	-	√	√
Zero Page, X	CMP Oper, X	D5	2	4	√	-	-	-	-	-	√	√
Absolute	CMP Oper	CD	3	4	√	-	-	-	-	-	√	√
Absolute, X	CMP Oper, X	DD	3	4*	√	-	-	-	-	-	√	√
Absolute, Y	CMP Oper, Y	D9	3	4*	√	-	-	-	-	-	√	√
(Indirect, X)	CMP (Oper, X)	C1	2	6	√	-	-	-	-	-	√	√
(Indirect), Y	CMP (Oper), Y	D1	2	5*	√	-	-	-	-	-	√	√

* => Add 1 if page boundary is crossed

Table (18): CMP – Short Reference

CPX

Operation: Compare Memory and Index X (X - M)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Immediate	CPX # Oper	E0	2	2	√	-	-	-	-	-	√	√
Zero Page	CPX Oper	E4	2	3	√	-	-	-	-	-	√	√
Absolute	CPX Oper	EC	3	4	√	-	-	-	-	-	√	√

Table (19): CPX – Short Reference

CPY

Operation: Compare Memory and Index Y (Y - M)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Immediate	CPY # Oper	C0	2	2	√	-	-	-	-	-	√	√
Zero Page	CPY Oper	C4	2	3	√	-	-	-	-	-	√	√
Absolute	CPY Oper	CC	3	4	√	-	-	-	-	-	√	√

Table (20): CPY – Short Reference

DEC

Operation: Decrement Memory by one (M – 1 → M)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Zero Page	DEC Oper	C6	2	5	√	-	-	-	-	-	√	-
Zero Page, X	DEC Oper, X	D6	2	6	√	-	-	-	-	-	√	-
Absolute	DEC Oper	CE	3	6	√	-	-	-	-	-	√	-
Absolute, X	DEC Oper, X	DE	3	7	√	-	-	-	-	-	√	-

Table (21): DEC – Short Reference

DEX

Operation: Decrement index X by one (X - 1 → X)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Implied	DEX	CA	1	2	√	-	-	-	-	-	√	-

Table (22): DEX – Short Reference

Example Implied (assume X=$01):

```
Address     Bytes     Mnemonic
$9000       CA        DEX        ;
$9001       …         next OP    ; X is now $00, N=0, Z=1

DEX   0000 0001 ($01)
      0000 0000 ($00), N=0, Z=1
```

DEY

Operation: Decrement index Y by one (Y - 1 → Y)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Implied	DEY	88	1	2	√	-	-	-	-	-	√	-

Table (23): DEX – Short Reference

Example Implied (assume Y=$00):

```
Address     Bytes     Mnemonic
$9000       88        DEY        ;
$9001       …         next OP    ; Y is now $FF, N=1, Z=0

DEY   0000 0000 ($00)
      1111 1111 ($FF), N=1, Z=0
```

EOR

Operation: "Exclusive-Or" memory or immediate value with accumulator A (A ⩛ M → A)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Immediate	EOR # Oper	49	2	2	√	-	-	-	-	-	√	-
Zero Page	EOR Oper	45	2	3	√	-	-	-	-	-	√	-
Zero Page, X	EOR Oper, X	55	2	4	√	-	-	-	-	-	√	-
Absolute	EOR Oper	4D	3	4	√	-	-	-	-	-	√	-
Absolute, X	EOR Oper, X	5D	3	4*	√	-	-	-	-	-	√	-
Absolute, Y	EOR Oper, Y	59	3	4*	√	-	-	-	-	-	√	-
(Indirect, X)	EOR (Oper, X)	41	2	6	√	-	-	-	-	-	√	-
(Indirect), Y	EOR (Oper), Y	51	2	5*	√	-	-	-	-	-	√	-

* => Add 1 if page boundary is crossed

Table (24): EOR – Short Reference

Example Immediate (assumed A is $23):

Address	Bytes	Mnemonic	
$9000	49 63	EOR #$63	;
$9002	…	*next OP*	; A is now $40, N=0, Z=0

```
        0110 0011 ($63)
EOR     0010 0011 ($23)
        0100 0000 ($40), N=0, Z=0
```

INC

Operation: Increment Memory by one (M + 1 → M)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Zero Page	INC Oper	E6	2	5	√	-	-	-	-	-	√	-
Zero Page, X	INC Oper, X	F6	2	6	√	-	-	-	-	-	√	-
Absolute	INC Oper	EE	3	6	√	-	-	-	-	-	√	-
Absolute, X	INC Oper, X	FE	3	7	√	-	-	-	-	-	√	-

Table (25): INC – Short Reference

INX

Operation: Increment index X by one (X + 1 → X)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Implied	INX	E8	1	2	√	-	-	-	-	-	√	-

Table (26): INX – Short Reference

Example Implied (assume X=$0C):

Address	Bytes	Mnemonic	
$9000	E8	INX	;
$9001	…	*next OP*	; X is now $0D, N=0, Z=0

INX 0000 1100 ($0C)
0000 1101 ($0D), N=0, Z=0

INY

Operation: Increment index Y by one (Y + 1 → Y)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Implied	INY	C8	1	2	√	-	-	-	-	-	√	-

Table (27): INY – Short Reference

Example Implied (assume Y=$FF):

Address	Bytes	Mnemonic
$9000	C8	INY
$9001	…	*next OP*

INY 1111 1111 ($FF)
0000 0000 ($00), N=0, Z=1

JMP

Operation: Jump to new location ((PC + 1) → PCL, (PC + 2) → PCH)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Absolute	JMP Oper	4C	3	3	-	-	-	-	-	-	-	-
Indirect	JMP (Oper)	6C	3	5	-	-	-	-	-	-	-	-

Table (28): JMP – Short Reference

In mode “Indirect” JMP fetches the high-order byte of the effective address from the first byte of the current page, if the indirect address location spans a page boundary.

6502: JMP ($02FF) gets ADL from $02FF, ADH from $0200 (5 cycles)

JSR

Operation: Jump to subroutine saving return address (PC + 2 ↓, (PC + 1) → PCH, (PC + 2) → PCL)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Absolute	JSR	20	3	6	-	-	-	-	-	-	-	-

Table (29): JSR – Short Reference

Example:

Address	Bytes	Mnemonic	
$9000	20 72 F0	JSR $F072	;
F072	…	*next OP*	

LDA

Operation: Load accumulator A with memory or immediate value (M → A)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Immediate	LDA # Oper	A9	2	2	√	-	-	-	-	-	√	-
Zero Page	LDA Oper	A5	2	3	√	-	-	-	-	-	√	-
Zero Page, X	LDA Oper, X	B5	2	4	√	-	-	-	-	-	√	-
Absolute	LDA Oper	AD	3	4	√	-	-	-	-	-	√	-
Absolute, X	LDA Oper, X	BD	3	4*	√	-	-	-	-	-	√	-
Absolute, Y	LDA Oper, Y	B9	3	4*	√	-	-	-	-	-	√	-
(Indirect, X)	LDA (Oper, X)	A1	2	6	√	-	-	-	-	-	√	-
(Indirect), Y	LDA (Oper), Y	B1	2	5*	√	-	-	-	-	-	√	-

* => Add 1 if page boundary is crossed

Table (30): LDA – Short Reference

Example Immediate:

Address	Bytes	Mnemonic	
$9000	A9 00	LDA #$00	;
$9002	…	*next OP*	; A is now $00, N=0, Z=1

LDX

Operation: Load index X with memory or immediate value (M → X)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Immediate	LDX # Oper	A2	2	2	√	-	-	-	-	-	√	-
Zero Page	LDX Oper	A6	2	3	√	-	-	-	-	-	√	-
Zero Page, Y	LDX Oper, Y	B6	2	4	√	-	-	-	-	-	√	-
Absolute	LDX Oper	AE	3	4	√	-	-	-	-	-	√	-
Absolute, Y	LDX Oper, Y	BE	3	4*	√	-	-	-	-	-	√	-

* => Add 1 if page boundary is crossed

Table (31): LDX – Short Reference

Example Immediate:

Address	Bytes	Mnemonic	
$9000	A2 8F	LDX #$8F	;
$9002	…	*next OP*	; X is now $8F, N=1, Z=0

LDY

Operation: Load index Y with memory or immediate value (M → Y)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Immediate	LDY # Oper	A0	2	2	√	-	-	-	-	-	√	-
Zero Page	LDY Oper	A4	2	3	√	-	-	-	-	-	√	-
Zero Page, X	LDY Oper, X	B4	2	4	√	-	-	-	-	-	√	-
Absolute	LDY Oper	AC	3	4	√	-	-	-	-	-	√	-
Absolute, X	LDY Oper, X	BC	3	4*	√	-	-	-	-	-	√	-

* => Add 1 if page boundary is crossed

Table (32): LDY – Short Reference

Example Immediate:

Address	Bytes	Mnemonic	
$9000	A0 02	LDY #$02	;
$9002	…	*next OP*	; Y is now $02, N=0, Z=0

LSR

Operation: Shift Right One Bit (Memory or Accumulator) (0 → 7 6 5 4 3 2 1 0 → C)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Accumulator	LSR A	4A	1	2	0	-	-	-	-	-	√	√
Zero Page	LSR Oper	46	2	5	0	-	-	-	-	-	√	√
Zero Page, X	LSR Oper, X	56	2	6	0	-	-	-	-	-	√	√
Absolute	LSR Oper	4E	3	6	0	-	-	-	-	-	√	√
Absolute, X	LSR Oper, X	5E	3	7	0	-	-	-	-	-	√	√

Table (33): LSR – Short Reference

NOP

Operation: No Operation

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Implied	NOP	EA	1	2	-	-	-	-	-	-	-	-

Table (34): NOP – Short Reference

Example:

Address	Bytes	Mnemonic	
$9000	EA	NOP	;
$9001	…	*next OP*	;

All unknown instructios are decoded as NOP.

ORA

Operation: "Or" memory or immediate value with accumulator A (A V M → A)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Immediate	OR # Oper	09	2	2	√	-	-	-	-	-	√	-
Zero Page	OR Oper	05	2	3	√	-	-	-	-	-	√	-
Zero Page, X	OR Oper, X	15	2	4	√	-	-	-	-	-	√	-
Absolute	OR Oper	0D	3	4	√	-	-	-	-	-	√	-
Absolute, X	OR Oper, X	1D	3	4*	√	-	-	-	-	-	√	-
Absolute, Y	OR Oper, Y	19	3	4*	√	-	-	-	-	-	√	-
(Indirect, X)	OR (Oper, X)	01	2	6	√	-	-	-	-	-	√	-
(Indirect), Y	OR (Oper), Y	11	2	5*	√	-	-	-	-	-	√	-

* => Add 1 if page boundary is crossed

Table (35): OR – Short Reference

Example Immediate (assumed A is $8A):

Address	Bytes	Mnemonic	
$9000	09 11	OR #$11	;
$9002	…	*next OP*	; A is now $9B, N=1, Z=0

```
        1000 1010 ($8A)
OR      0001 0001 ($11)
        1001 1011 ($9B), N=1, Z=0
```

PHA

Operation: Push accumulator on stack (A ↓)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Implied	PHA	48	1	3	-	-	-	-	-	-	-	-

Table (36): PHA – Short Reference

PHP

Operation: Push processor status on stack (P ↓ NV11DIZC, E & B are written as 1 to stack)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Implied	PHP	08	1	3	-	-	-	-	-	-	-	-

Table (37): PHP – Short Reference

PLA

Operation: Pull accumulator from stack (A ↑)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Implied	PLA	68	1	3	√	-	-	-	-	-	√	-

Table (38): PLA – Short Reference

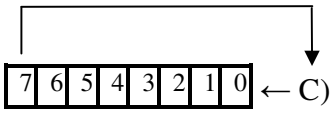
PLP

Operation: Pull processor status from stack (P ↑)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Implied	PLP	28	1	3	From Stack (E&B unchanged)							

Table (39): PLP – Short Reference

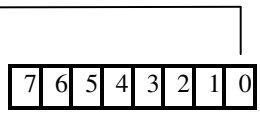
ROL

Operation: Rotate one bit left (memory or accumulator) ( ← C)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Accumulator	ROL A	2A	1	2	√	-	-	-	-	-	√	√
Zero Page	ROL Oper	26	2	5	√	-	-	-	-	-	√	√
Zero Page, X	ROL Oper, X	36	2	6	√	-	-	-	-	-	√	√
Absolute	ROL Oper	2E	3	6	√	-	-	-	-	-	√	√
Absolute, X	ROL Oper, X	3E	3	7	√	-	-	-	-	-	√	√

Table (40): ROL – Short Reference

ROR

Operation: Rotate one bit right (memory or accumulator) (C → )

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Accumulator	ROR A	6A	1	2	√	-	-	-	-	-	√	√
Zero Page	ROR Oper	66	2	5	√	-	-	-	-	-	√	√
Zero Page, X	ROR Oper, X	76	2	6	√	-	-	-	-	-	√	√
Absolute	ROR Oper	6E	3	6	√	-	-	-	-	-	√	√
Absolute, X	ROR Oper, X	7E	3	7	√	-	-	-	-	-	√	√

Table (41): ROR – Short Reference

RTI

Operation: Return from interrupt (P ↑, PC ↑)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Implied	RTI	40	1	6	From Stack							

Table (42): RTI – Short Reference

RTS

Operation: Return from subroutine (PC ↑, PC + 1 → PC)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Implied	RTS	60	1	6	-	-	-	-	-	-	-	-

Table (43): RTS – Short Reference

SBC

Operation: Substract memory or immediate value from accumulator with borrow (A - M - $\overline{C}$ → A, C)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Immediate	SBC # Oper	E9	2	2	√	√	-	-	-	-	√	√
Zero Page	SBC Oper	E5	2	3	√	√	-	-	-	-	√	√
Zero Page, X	SBC Oper, X	F5	2	4	√	√	-	-	-	-	√	√
Absolute	SBC Oper	ED	3	4	√	√	-	-	-	-	√	√
Absolute, X	SBC Oper, X	FD	3	4*	√	√	-	-	-	-	√	√
Absolute, Y	SBC Oper, Y	F9	3	4*	√	√	-	-	-	-	√	√
(Indirect, X)	SBC (Oper, X)	E1	2	6	√	√	-	-	-	-	√	√
(Indirect), Y	SBC (Oper), Y	F1	2	5*	√	√	-	-	-	-	√	√

* => Add 1 if page boundary is crossed

Table (44): SBC – Short Reference

SEC

Operation: Set carry flag (1 → C)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Implied	SEC	38	1	2	-	-	-	-	-	-	-	1

Table (45): SEC – Short Reference

Example:

```
Address     Bytes     Mnemonic
$9000       38        SEC        ;
$9001       …         next OP    ; C in now 1
```

SED

Operation: Set decimal flag (1 → D)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Implied	SED	F8	1	2	-	-	-	-	1	-	-	-

Table (46): CLD – Short Reference

Example:

```
Address     Bytes     Mnemonic
$9000       F8        SED        ;
$9001       …         next OP    ; D in now 1
```

SEI

Operation: Set interrupt disable flag (1 → I)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Implied	SEI	78	1	2	-	-	-	-	-	1	-	-

Table (47): SEI – Short Reference

Example:

Address	Bytes	Mnemonic	
$9000	78	SEI	;
$9001	…	*next OP*	; I in now 1

STA

Operation: Store accumulator in memory (A → M)

Addressing Mode	Assembly Language Form		OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Zero Page	STA	Oper	85	2	3	-	-	-	-	-	-	-	-
Zero Page, X	STA	Oper, X	95	2	4	-	-	-	-	-	-	-	-
Absolute	STA	Oper	8D	3	4	-	-	-	-	-	-	-	-
Absolute, X	STA	Oper, X	9D	3	5	-	-	-	-	-	-	-	-
Absolute, Y	STA	Oper, Y	99	3	5	-	-	-	-	-	-	-	-
(Indirect, X)	STA	(Oper, X)	81	2	6	-	-	-	-	-	-	-	-
(Indirect), Y	STA	(Oper), Y	91	2	6	-	-	-	-	-	-	-	-

Table (48): STA – Short Reference

STX

Operation: Store index X in memory (X → M)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Zero Page	STX Oper	86	2	3	-	-	-	-	-	-	-	-
Zero Page, Y	STX Oper, Y	96	2	4	-	-	-	-	-	-	-	-
Absolute	STX Oper	8E	3	4	-	-	-	-	-	-	-	-

Table (49): STX – Short Reference

STY

Operation: Store index Y in memory (Y → M)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Zero Page	STY Oper	84	2	3	-	-	-	-	-	-	-	-
Zero Page, X	STY Oper, X	94	2	4	-	-	-	-	-	-	-	-
Absolute	STY Oper	8C	3	4	-	-	-	-	-	-	-	-

Table (50): STY – Short Reference

TAX

Operation: Transfer accumulator A to index X (A → X)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Implied	TAX	AA	1	2	√	-	-	-	-	-	√	-

Table (51): TAX – Short Reference

Example Implied (assume A=$5D):

```
Address     Bytes     Mnemonic
$9000       AA        TAX        ;
$9001       …         next OP    ; X is now $5D, N=0, Z=0
```

TAY

Operation: Transfer accumulator A to index Y (A → Y)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Implied	TAY	A8	1	2	√	-	-	-	-	-	√	-

Table (52): TAY – Short Reference

Example Implied (assume A=$89):

```
Address     Bytes     Mnemonic
$9000       A8        TAY        ;
$9001       …         next OP    ; Y is now $89, N=1, Z=0
```

TSX

Operation: Transfer stack pointer S to index X (S → X)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Implied	TSX	BA	1	2	√	-	-	-	-	-	√	-

Table (53): TSX – Short Reference

Example Implied (assume S=$F2):

```
Address     Bytes     Mnemonic
$9000       BA        TSX
$9001       …         next OP    ; X is now $F2, N=1, Z=0
```

TXA

Operation: Transfer index X to accumulator A (X → A)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Implied	TXA	8A	1	2	√	-	-	-	-	-	√	-

Table (54): TXA – Short Reference

Example Implied (assume X=$00):

Address	Bytes	Mnemonic	
$9000	8A	TXA	
$9001	…	*next OP*	; A is now $00, N=0, Z=1

TXS

Operation: Transfer index X to stack pointer S (X → S)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Implied	TXS	9A	1	2	-	-	-	-	-	-	-	-

Table (55): TXS – Short Reference

Example Implied (assume X=$E0):

Address	Bytes	Mnemonic	
$9000	9A	TXS	
$9001	…	*next OP*	; S is now $E0 (all flags are uneffected

TYA

Operation: Transfer index Y to accumulator A (Y → A)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles	N	V	E	B	D	I	Z	C
Implied	TYA	98	1	2	√	-	-	-	-	-	√	-

Table (56): TYA – Short Reference

Example Implied (assume Y=$14):

```
Address     Bytes     Mnemonic
$9000       98        TYA        ;
$9001       …         next OP    ; A is now $14, N=0, Z=0
```

4 Registers

List of Registers

Name	Address	Width	Access	Description
A	-	8	Instruction	Accumulator
X	-	8	Instruction	X Register
Y	-	8	Instruction	Y Register
S	-	8	Instruction	Stack
PSW	-	8	Instruction	Processor Status Word
PC	-	16	Instruction	Program Counter

Table 1: List of registers

PSW – Description

Bit #	Access	Description
7		N-Flag, negative result (1 -> negative)
6		V-Flag, overflow (1 -> overflow)
5		*RESERVED (always read as '1')*
4		B-Flag, BRK instruction
3		D-Flag, decimal mode (1 -> decimal)
2		I-Flag, interrupt disable (1 -> disabled)
1		Z-Flag, zero result (1 -> zero)
0		C-Flag, carry

Table 2: PSW Description

Reset Value:24h

5 Clocks

Name	Source	Rates (MHz)			Remarks	Description
		Max	Min	Resolution		
clk_clk_i	-	60	0	-	Achievable rates depend on synthesis and place&route.	Master Clock

Table 3: List of clocks

6 Interrupts

If an NMI occurred after a BRK instruction was fetched and before the first byte of the vector of BRK could be loaded into the program counter the vector of NMI will be loaded instead.

The priority of interrupts is as followed if all of three interrupts occur at the same time:

(BRK ignored) – NMI – IRQ

NMI

The interrupt input “nmi_n_i” is edge triggered at a falling edge ‘1’ -> ‘0’. It is not mask-able.

There is a need of one cycle ‘1’ following with one cycle ‘0’ to detect an NMI interrupt as “valid”. “nmi_n_i” can switch back to ‘1’ or held on ‘0’ within cycle 3.

After the internal IRQ sequence of 7 cycles has finished, the “valid” flag re-set and allow another incoming NMI to detect. The maximum frequency of falling edges on “nmi_n_i” must be carefully defined to avoid a stack overflow while a currently processing interrupt is interrupted by another one.

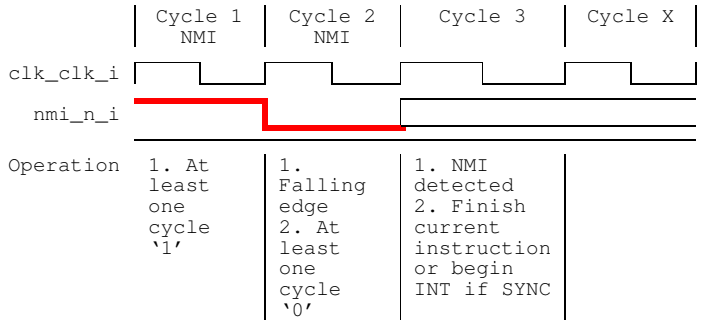


Figure (3): Interrupt NMI – Timing Diagram

IRQ

The interrupt input “irq_n_i” is level triggered at a level of ‘0’. It is mask-able by using the I-Flag (I=’1’ to disable IRQ).

There is a need of level ‘0’ at a rising edge of “clk_clk_i” and “SYNC=’1’” to detect an IRQ interrupt as “valid”. “irq_n_i” can switch back to ‘1’ or held on ‘0’ within cycle 3.

After the internal IRQ sequence of 7 cycles has finished, the I-Flag is set to ‘1’ automatically to disallow another incoming IRQ to detect. The starting ISR (Interrupt Service Routine) is responsible to manage and service the devices which was generating the interrupt and enable IRQ again (I-Flag=’0’).

Devices must switch back “irq_n_i” to ‘1’ after the interrupt service is finished.

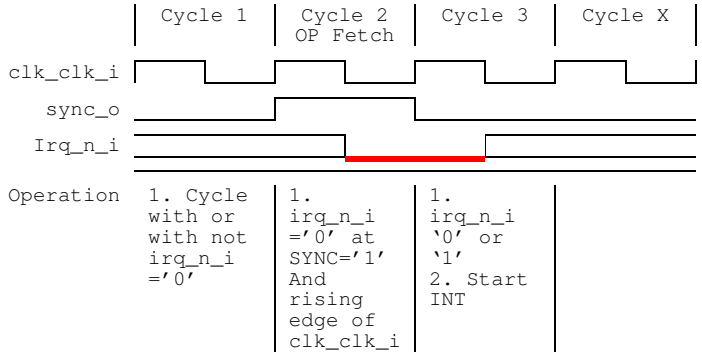


Figure (4): Interrupt IRQ – Timing Diagram

BRK

“BRK” is an instruction to force an interrupt by software. Internally it is handled like a real hardware interrupt and is mostly used by debug software.

“BRK” transfers the B-Flag = 1 onto the stack. The B-Flag itself leaves untouched and the I-Flag after “BRK” is 1.

Appendix A

ADC / SBC Decimal Mode

The R6502 allows handling of Binary Coded Decimals (BCD) directly if the D-Flag is set to 1.

BCD defines a representation of decimal numbers 0-9 within an underlying hexadecimal number system.

For example:

99d => 63h

But in BCD the representation is

99d => 99h (BCD)

The R6502 has the ability to convert the result value into BCD automatically if the nibbles exceed the decimal range of 0-9 with some limitations. Input values are not corrected in any way.

Some examples with ADC IMM (DECIMAL):

ACCU	IMM	Carry in	Real World Result HEX	Real World Result BCD	R6502 Result BCD	Low Nibble (LN) High Nibble (HN)
01h	09h	0	0Ah	10h	10h	LN > 9 (Ah), Carry to HN (=10h)
00h	09h	1	0Ah	10h	10h	LN > 9 (Ah), Carry to HN (=10h)
01h	0Fh	0	10h	16h	00h	LN < 9 (0h), NO Carry to HN (=00h)
0Fh	0Fh	0	1Eh	30h	14h	LN > 9 (Eh), Carry to HN (=1h new), HNnew = HNa + HNimm + 1
F0h	F0h	0	(1)E0h	(4)80h	40h	LN < 9 (0h), NO Carry to HN (=Eh new), HNnew =HNa + HNimm + 0

FFh	FFh	0	(1)FEh	(5)10h	54h	LN > 9 (Eh), Carry to HN (=Fh new), HNnew =HNa + HNimm + 1
FFh	FFh	1	(1)FFh	(5)11h	55h	LN > 9 (Fh), Carry to HN (=Fh new), HNnew =HNa + HNimm + 1

Table 4: Some examples of ADC

In general (NOT valid for 6502), there are four levels of decoding and corrections for hexadecimal values to convert to BCD:

Level	Result pure HEX		Corrected Result	System	Tasks
L1	04h	=>	04h	(BCD)	no correction needed
L2	0Ch	=>	12h	(BCD)	correction of low nibble, add carry to the high nibble
L3	12h	=>	18h	(BCD)	correction of low nibble from high nibble (backward correction) if carry occur from low to high nibble.
L4	1Bh	=>	27h	(BCD)	correction of low nibble from high nibble , add carry to the high nibble (backward correction)

Table 5: Levels of BCD decoding and correction

Even the R6502 correct the nibbles if a nibble value is greater than 9, the handling and generation of carry bits from one nibble to the other covers not all possible decoding levels L1 to L4. Incorrect results may occur if using invalid BCD numbers containing the hex parts A-F. The backward correction to the low nibble and new generation of the carry to correct the high nibble (L4) was not implemented into the R6502 chip and the r6502_tc IP core also.

Timing Diagrams

ADC, SBC

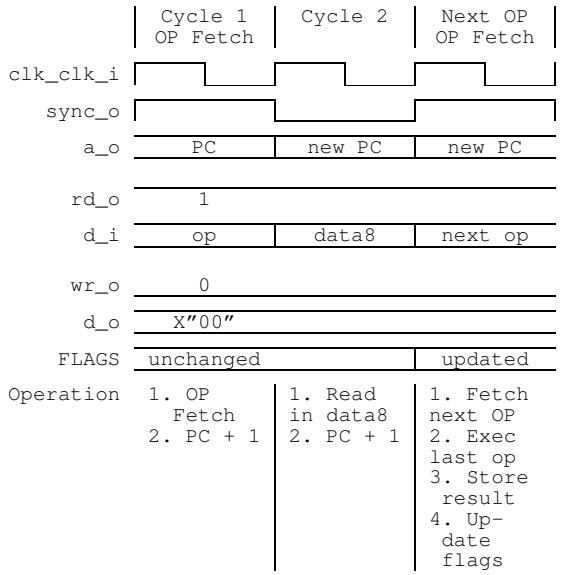


Figure (5): ADC, SBC – Timing Diagram "Immediate"

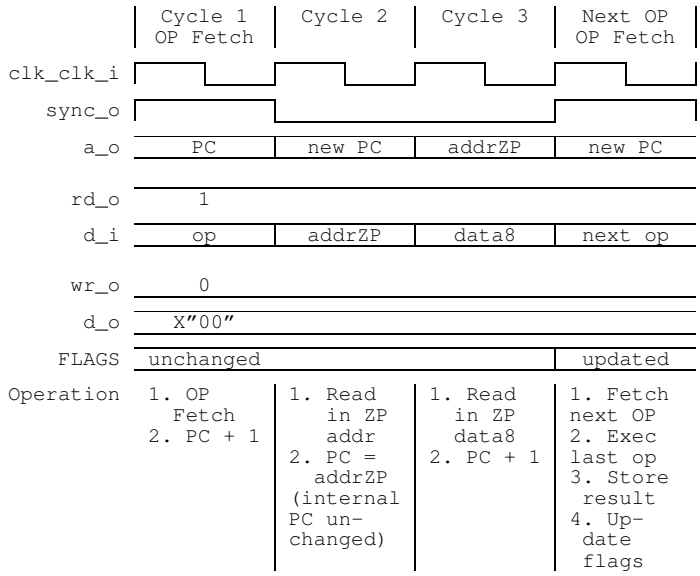


Figure (6): ADC, SBC – Timing Diagram "Zero Page"

ADC, SBC (cont.)

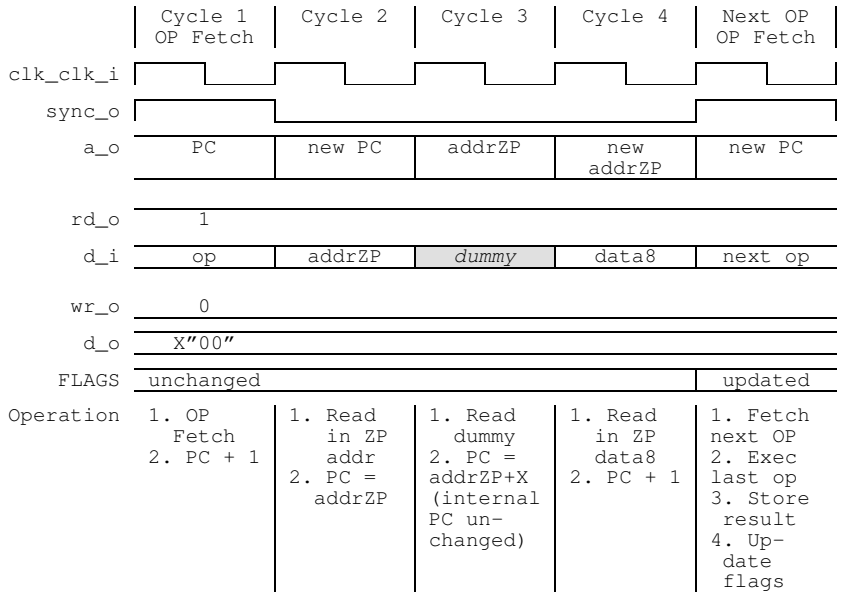


Figure (7): ADC, SBC – Timing Diagram "Zero Page, X"

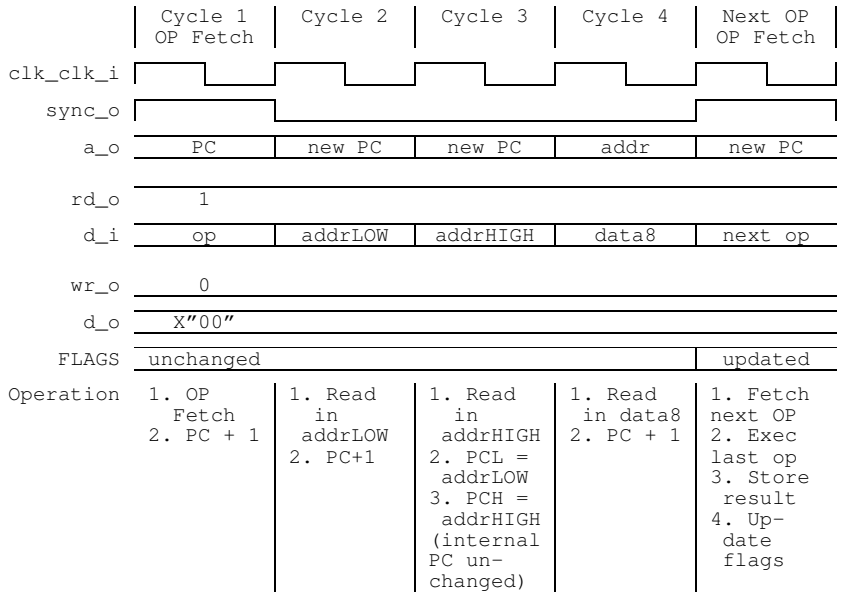


Figure (8): ADC, SBC – Timing Diagram "Absolute"

ADC, SBC (cont.)

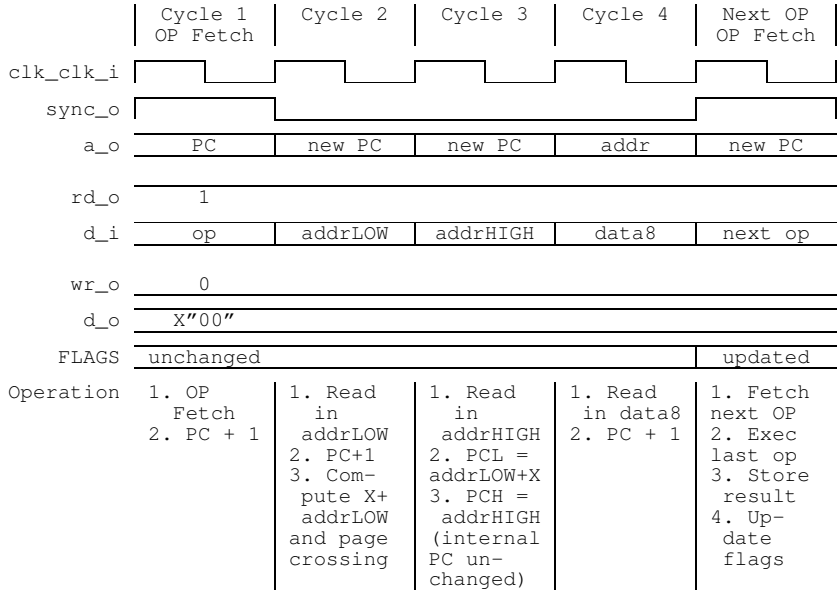


Figure (9): ADC, SBC – Timing Diagram "Absolute, X" – no page crossing

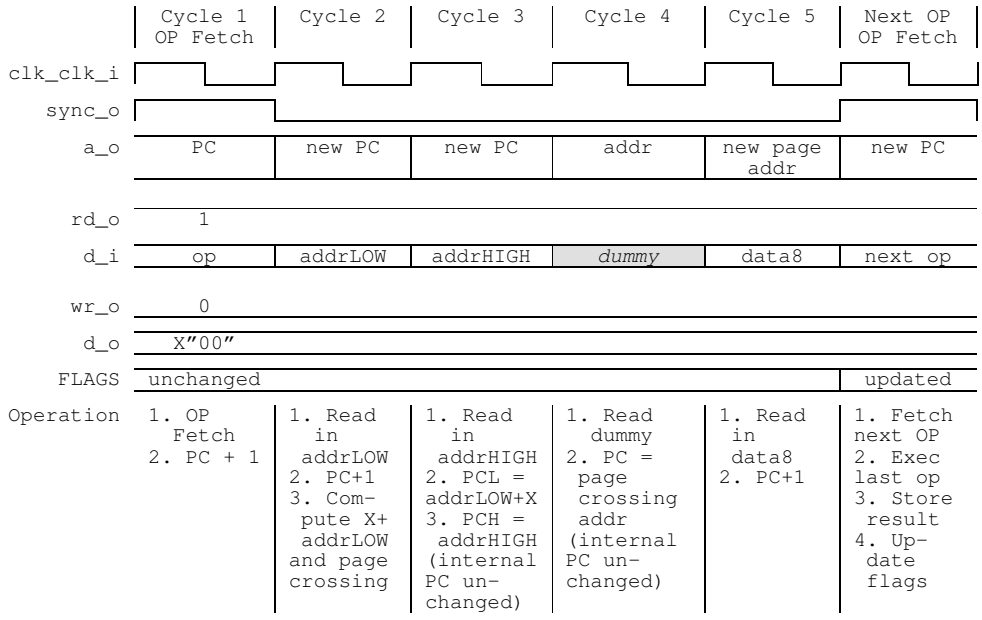


Figure (10): ADC, SBC – Timing Diagram "Absolute, X" – page crossing

ADC, SBC (cont.)

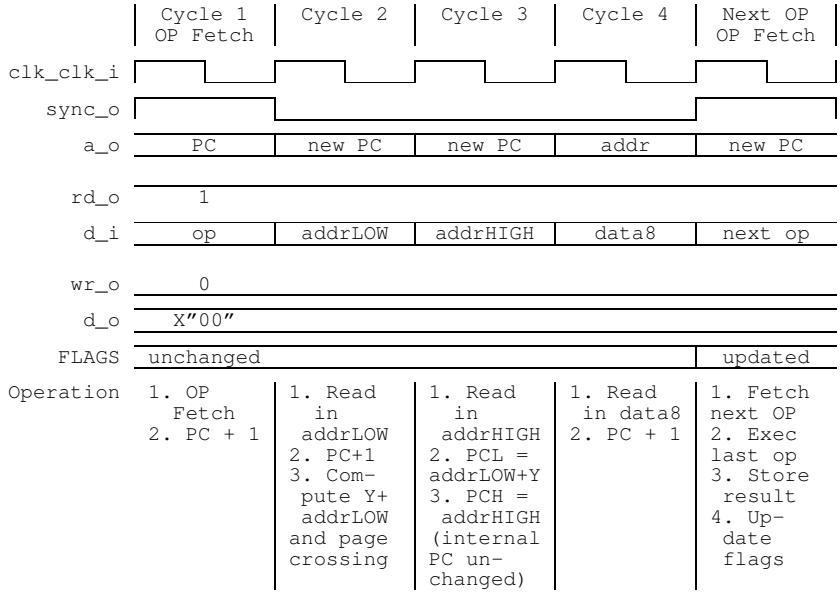


Figure (11): ADC, SBC – Timing Diagram "Absolute, Y" – no page crossing

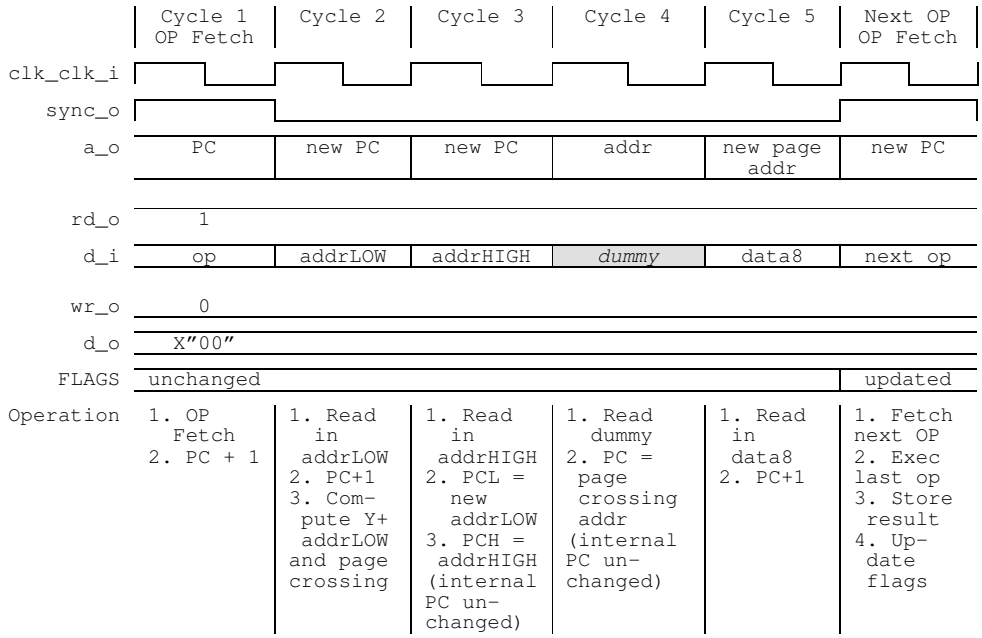


Figure (12): ADC, SBC – Timing Diagram "Absolute, Y" – page crossing

ADC, SBC (cont.)

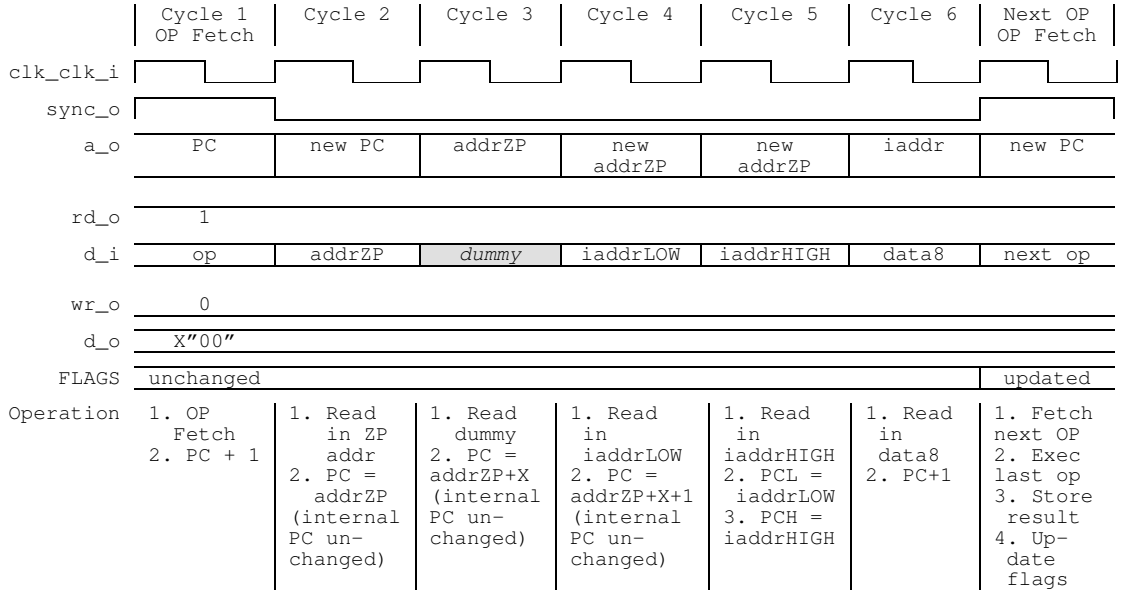


Figure (13): ADC, SBC – Timing Diagram "(Indirect, X)"

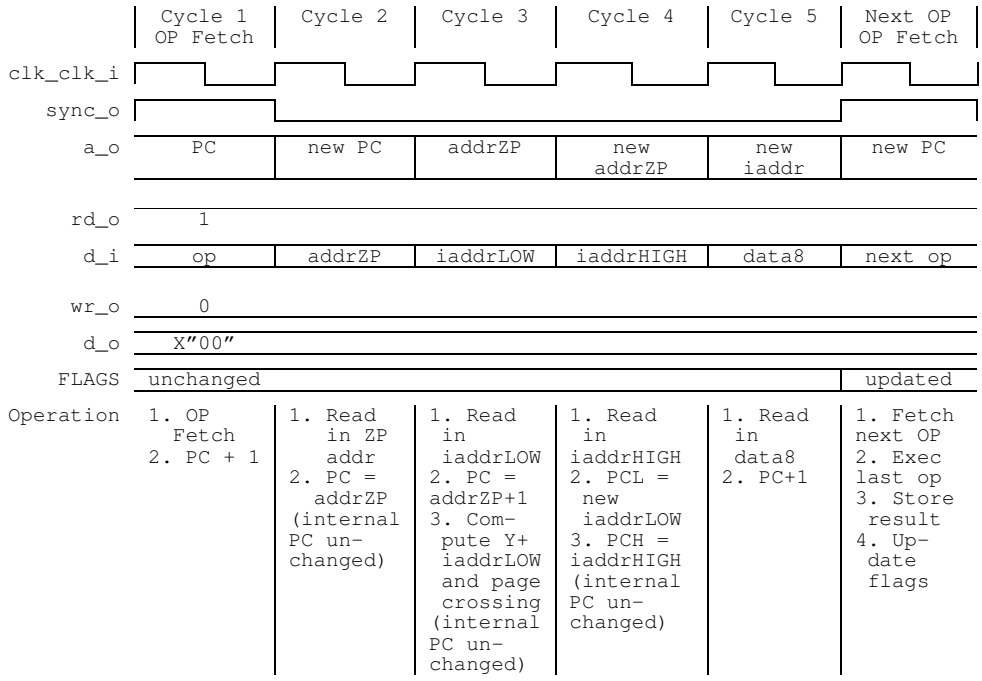


Figure (14): ADC, SBC – Timing Diagram "(Indirect), Y" – no page crossing

ADC, SBC (cont.)

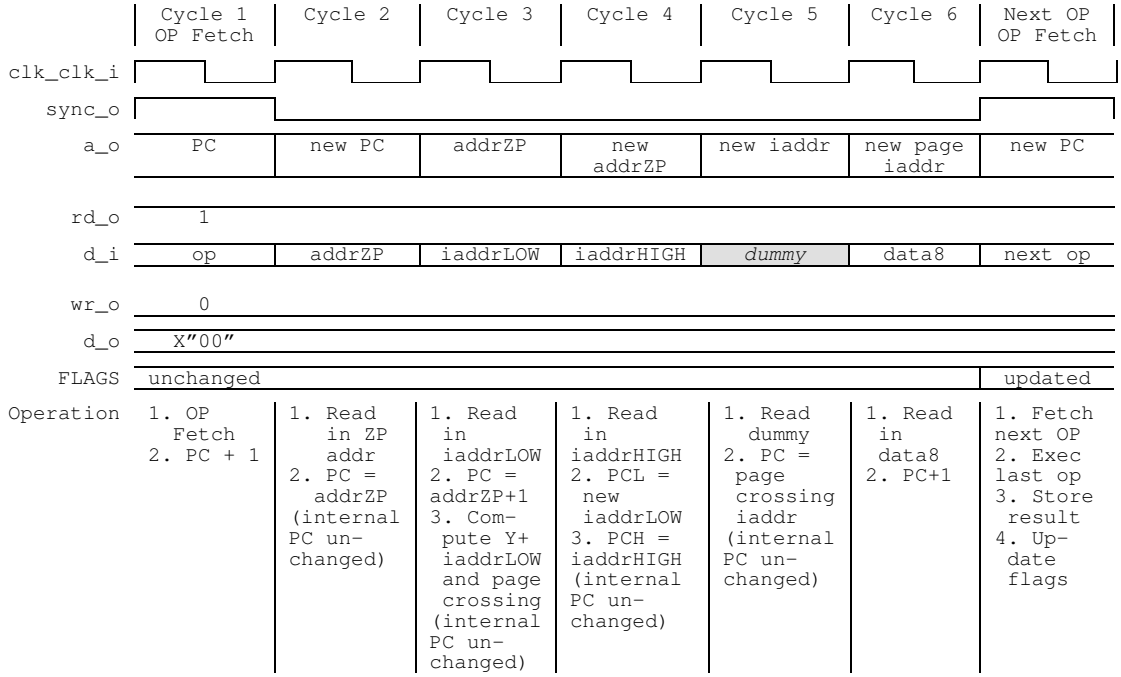


Figure (15): ADC, SBC – Timing Diagram "(Indirect), Y" – page crossing

AND, BIT, CMP, CPX, CPY, EOR, LDA, LDX, LDY, ORA

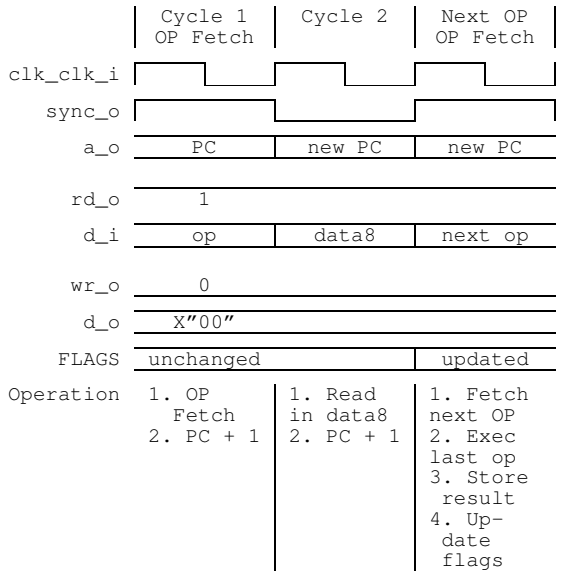


Figure (16): AND, CMP, CPX, CPY, EOR, LDA, LDX, LDY, ORA – Timing Diagram "Immediate"

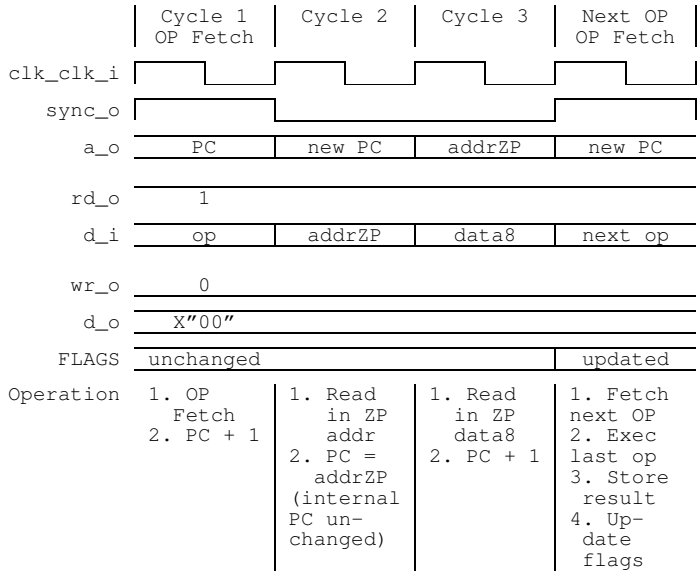


Figure (17): AND, BIT, CMP, CPX, CPY, EOR, LDA, LDX, LDY, ORA – Timing Diagram "Zero Page"

AND, BIT, CMP, CPX, CPY, EOR, LDA, LDX, LDY, ORA (cont.)

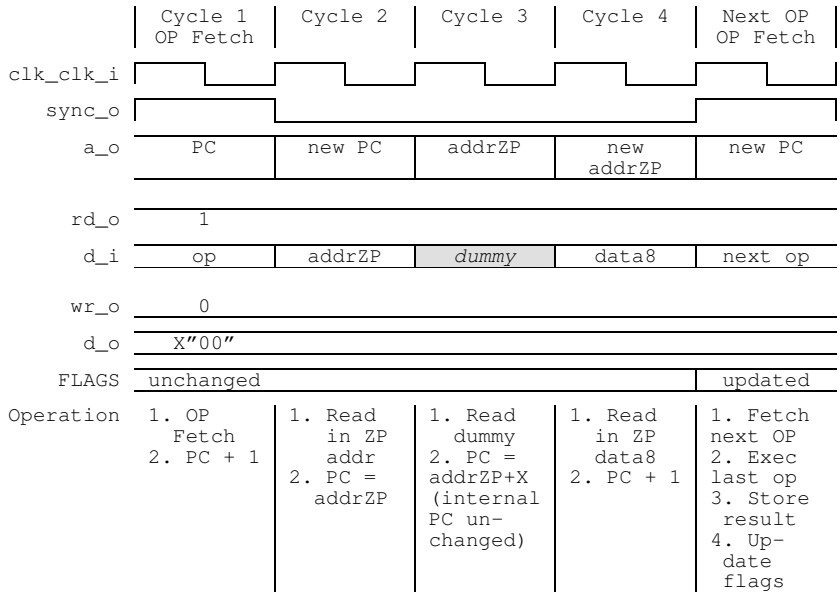


Figure (18): AND, CMP, EOR, LDA, LDY, ORA – Timing Diagram "Zero Page, X"

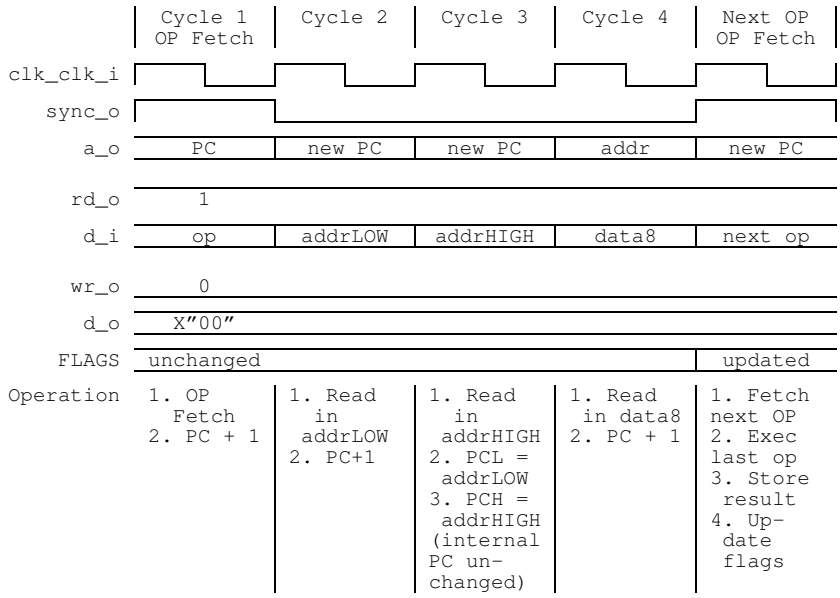


Figure (19): AND, BIT, CMP, CPX, CPY, EOR, LDA, LDX, LDY, ORA – Timing Diagram "Absolute"

AND, BIT, CMP, CPX, CPY, EOR, LDA, LDX, LDY, ORA (cont.)

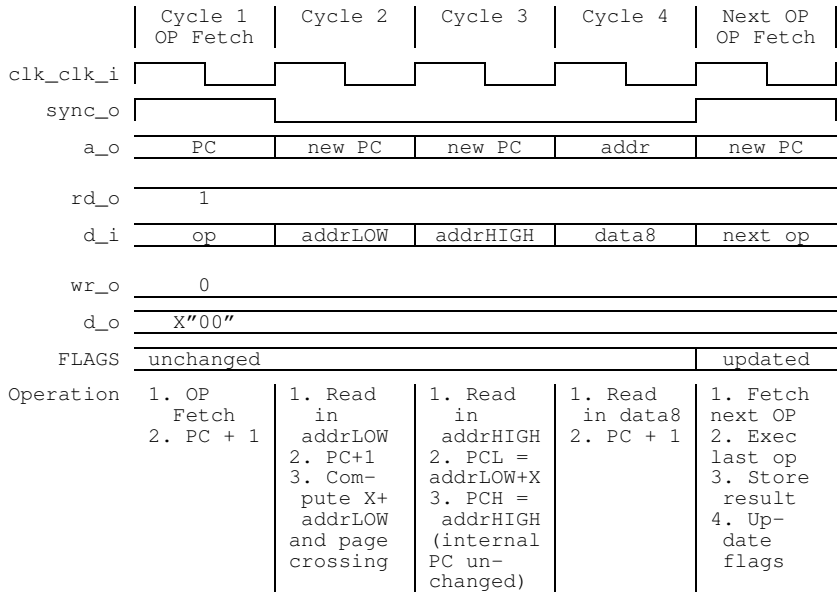


Figure (20): AND, CMP, EOR, LDA, LDY, ORA – Timing Diagram "Absolute, X" – no page crossing

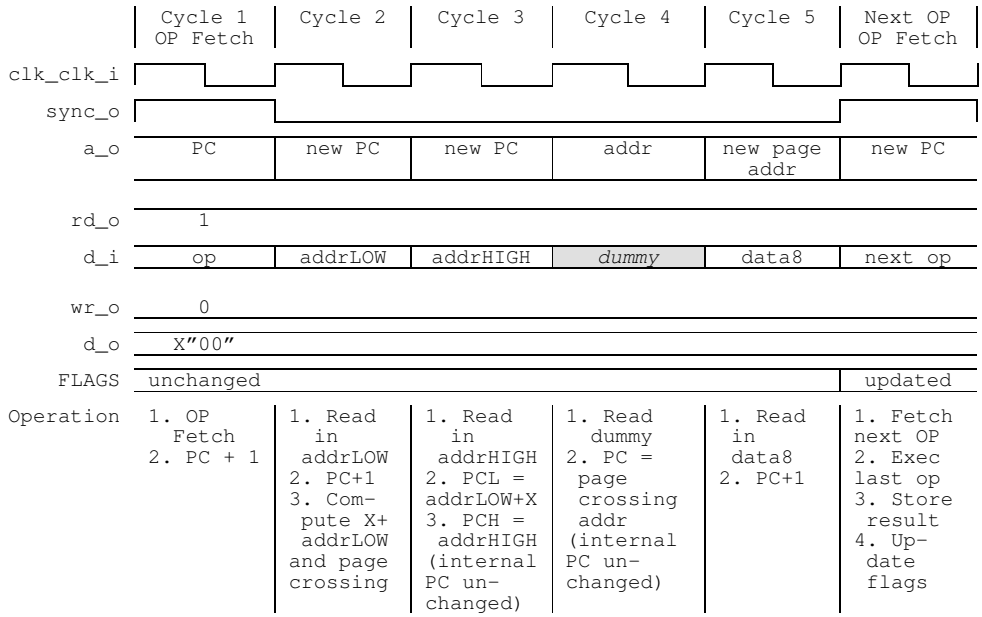


Figure (21): AND, CMP, EOR, LDA, LDY, ORA – Timing Diagram "Absolute, X" – page crossing

AND, BIT, CMP, CPX, CPY, EOR, LDA, LDX, LDY, ORA (cont.)

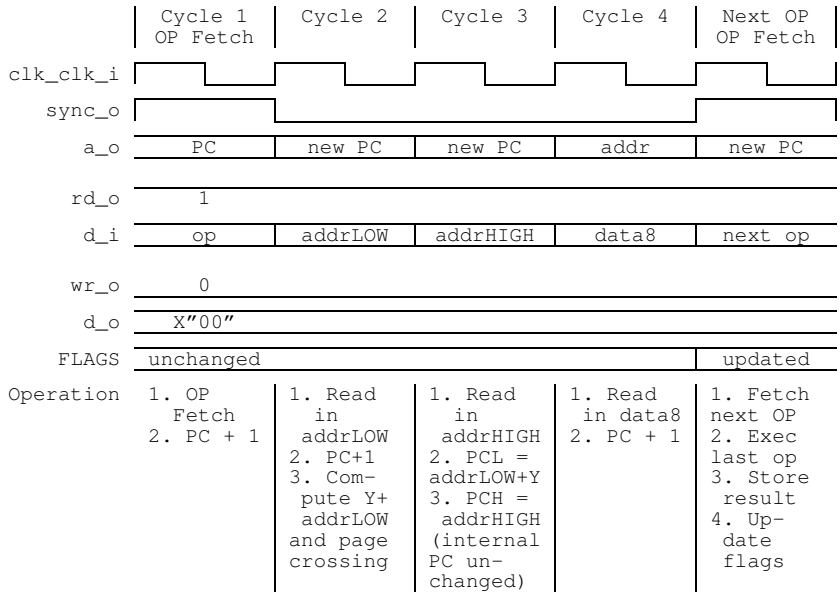


Figure (22): AND, CMP, EOR, LDA, LDX, ORA – Timing Diagram "Absolute, Y" – no page crossing

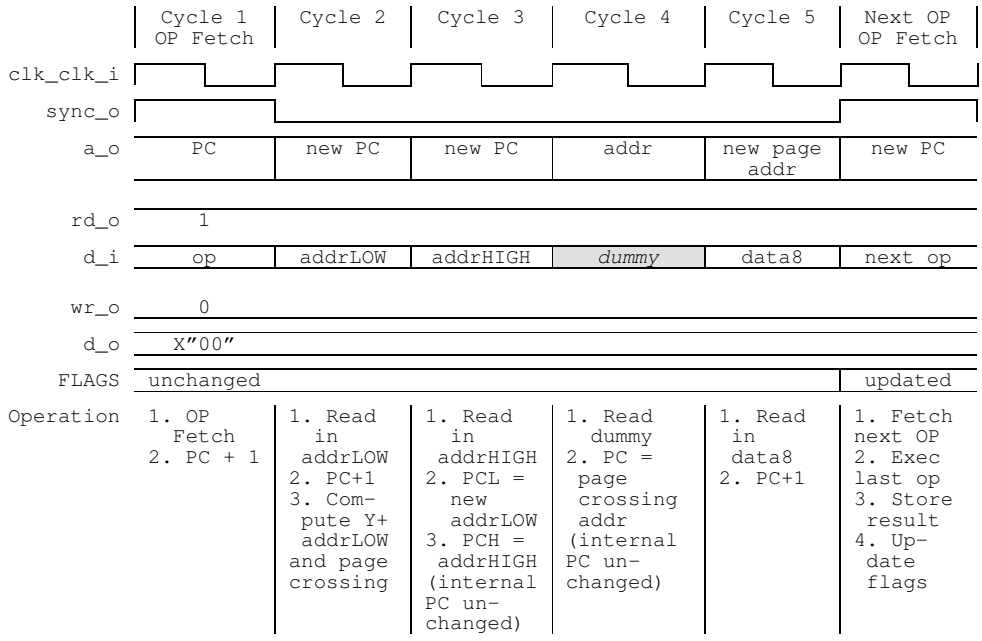


Figure (23): AND, CMP, EOR, LDA, LDX, ORA – Timing Diagram "Absolute, Y" – page crossing

AND, BIT, CMP, CPX, CPY, EOR, LDA, LDX, LDY, ORA (cont.)

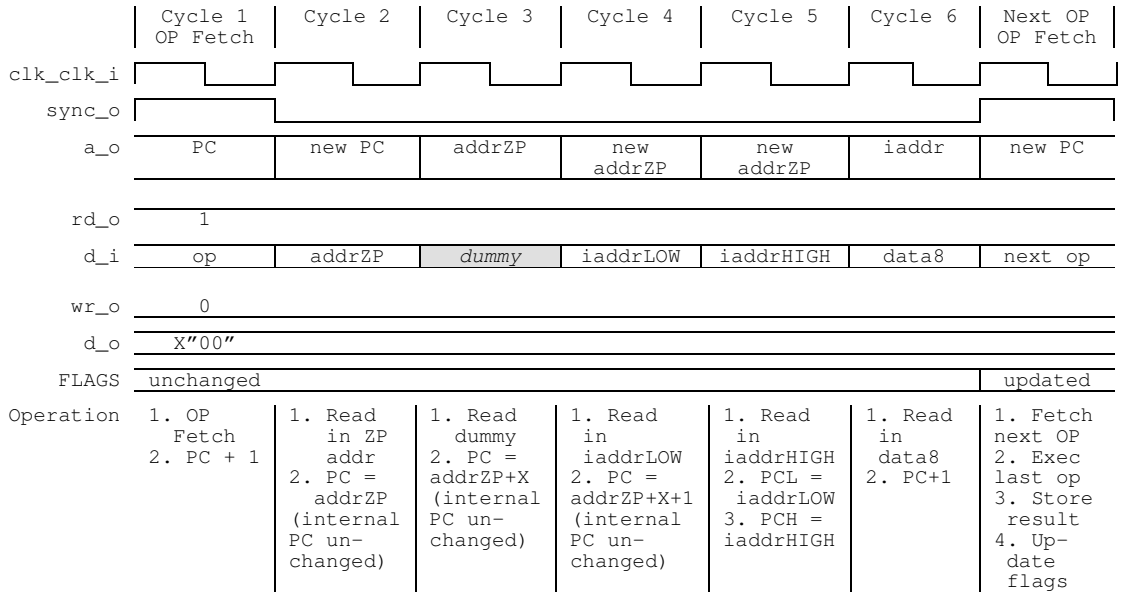


Figure (24): AND, CMP, EOR, LDA, ORA – Timing Diagram "(Indirect, X)"

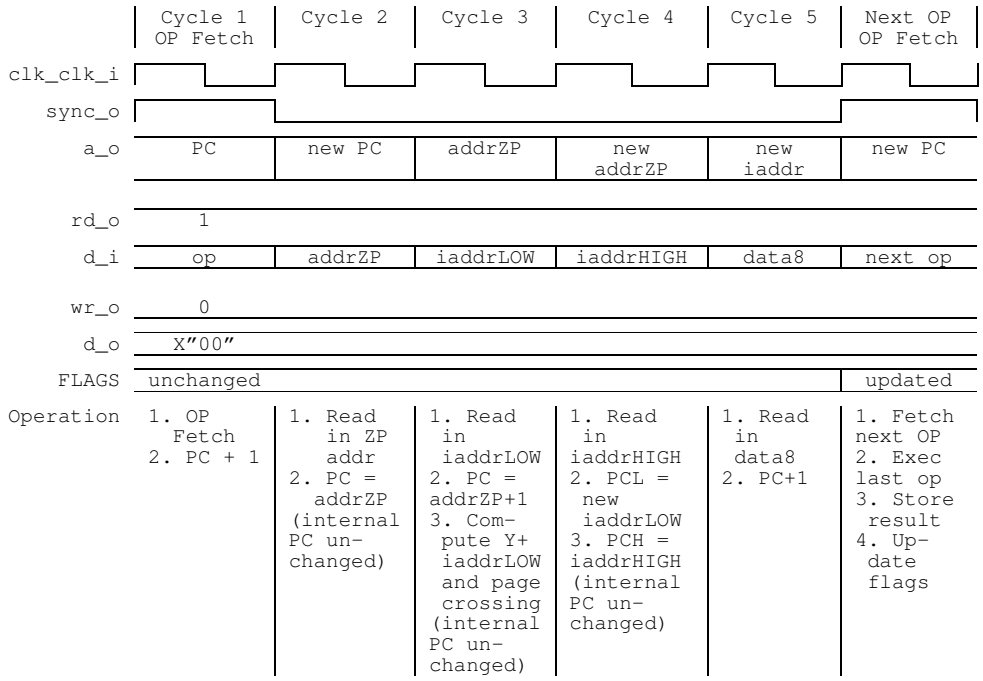


Figure (25): AND, CMP, EOR, LDA, ORA – Timing Diagram "(Indirect), Y" – no page crossing

AND, BIT, CMP, CPX, CPY, EOR, LDA, LDX, LDY, ORA (cont.)

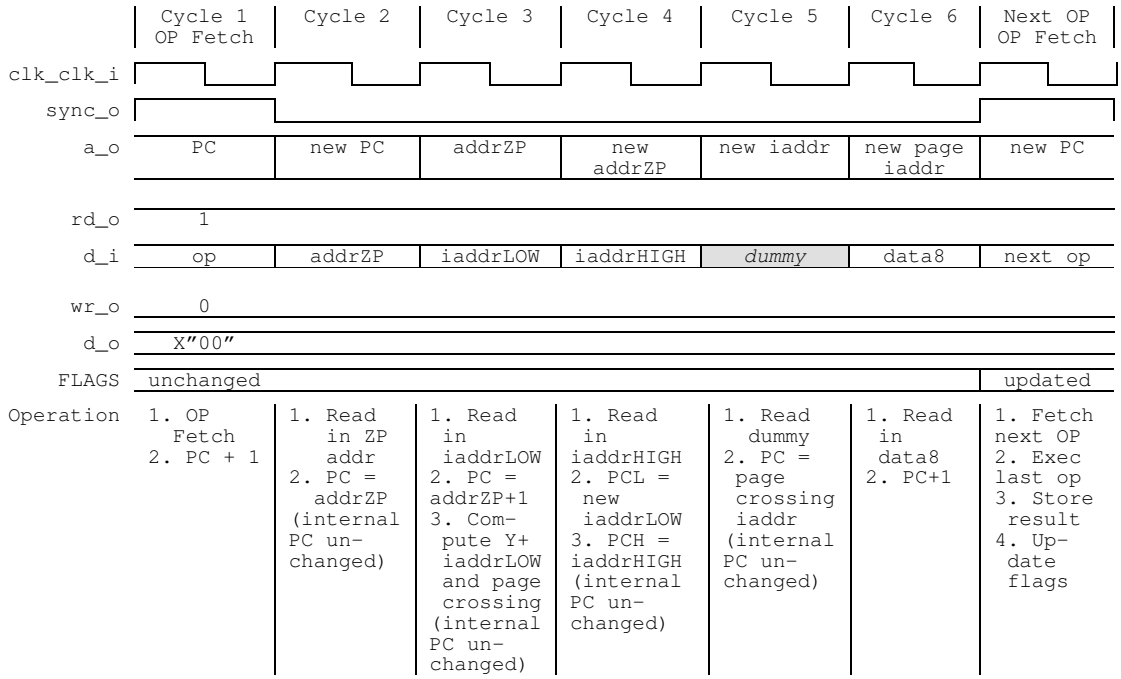


Figure (26): AND, CMP, EOR, LDA, ORA – Timing Diagram "(Indirect), Y" – page crossing

BCC, BCS, BVC, BVS, BPL, BMI, BEQ, BNE

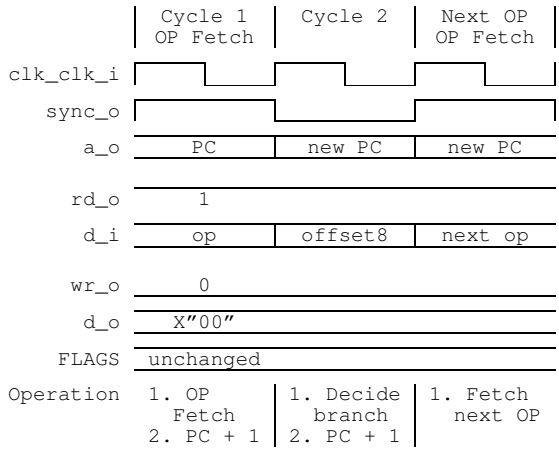


Figure (27): BCC, BCS, BVC, BVS, BPL, BMI, BEQ, BNE – Timing Diagram "no branch, same page"

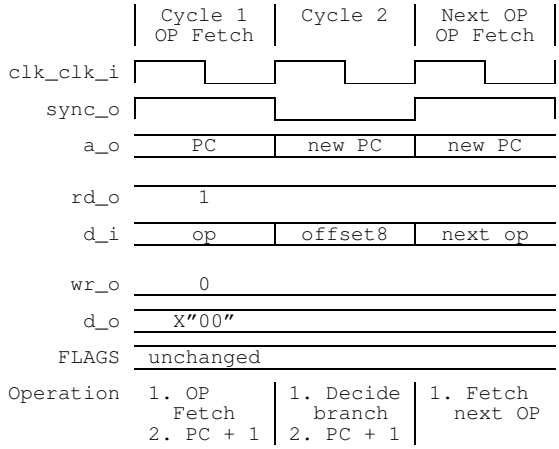


Figure (28): BCC, BCS, BVC, BVS, BPL, BMI, BEQ, BNE – Timing Diagram "no branch, different page"

BCC, BCS, BVC, BVS, BPL, BMI, BEQ, BNE (cont.)

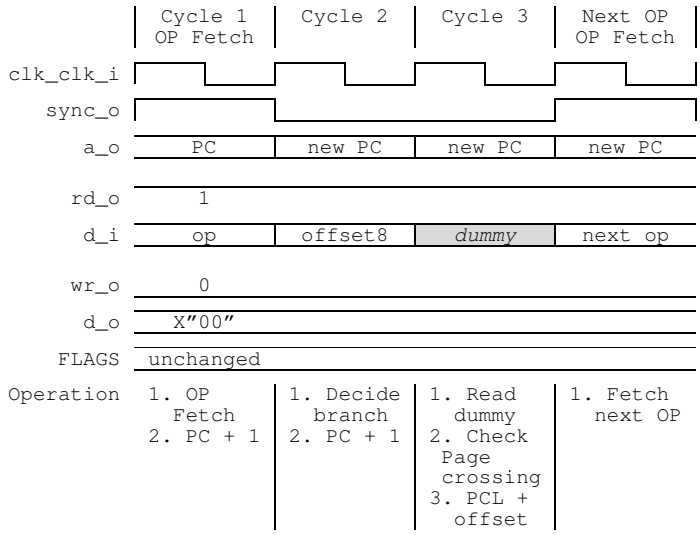


Figure (29): BCC, BCS, BVC, BVS, BPL, BMI, BEQ, BNE – Timing Diagram "branch, same page"

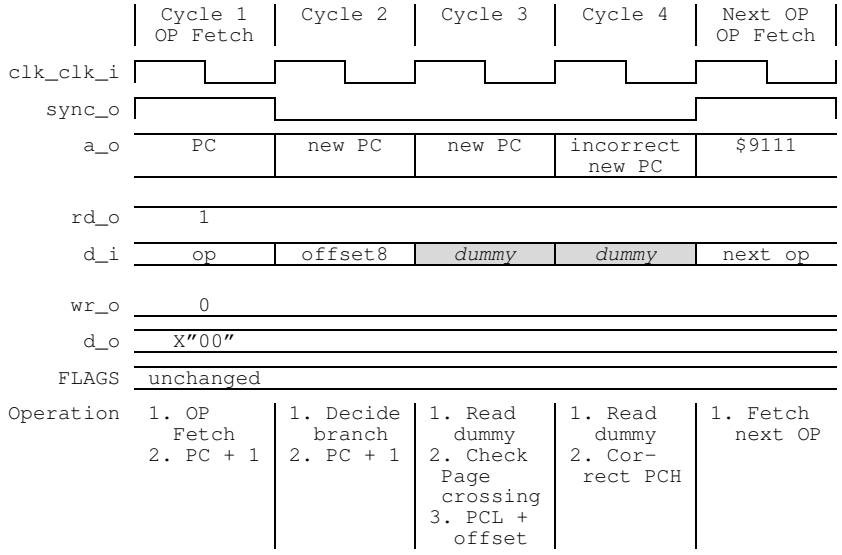


Figure (30): BCC, BCS, BVC, BVS, BPL, BMI, BEQ, BNE – Timing Diagram "branch, different page"

BRK

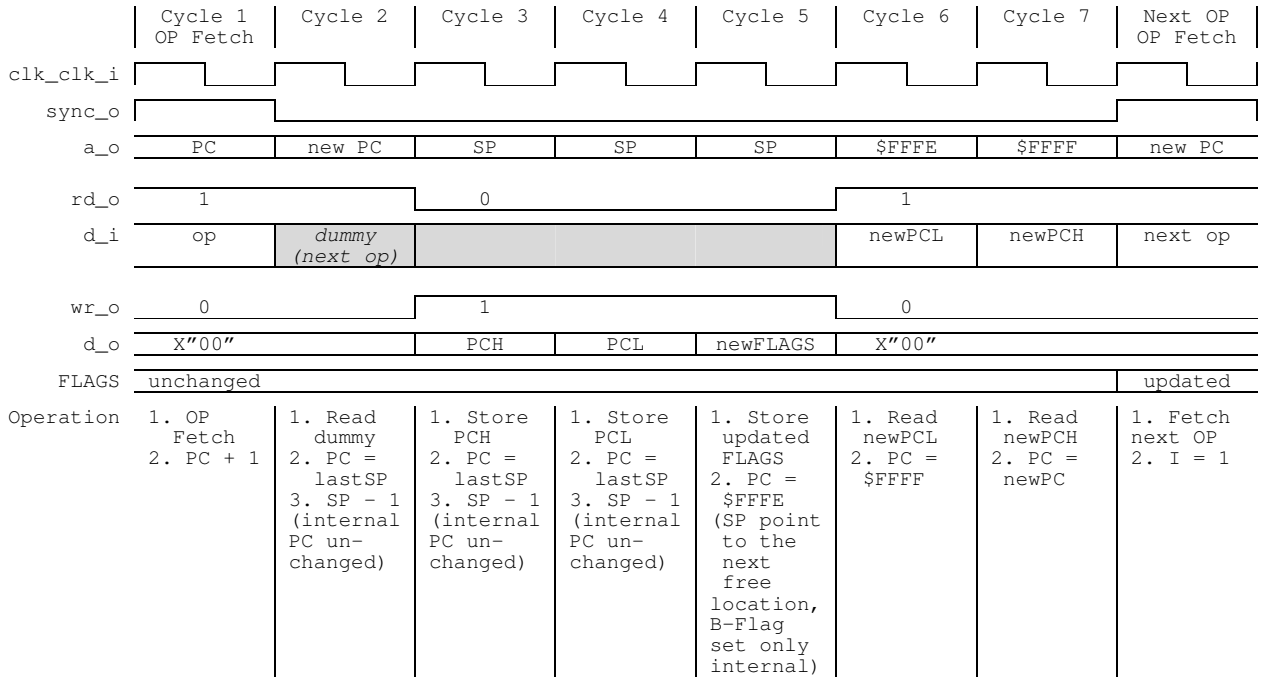


Figure (31): BRK – Timing Diagram "Implied"

ASL, LSR, ROL, ROR, DEC, INC

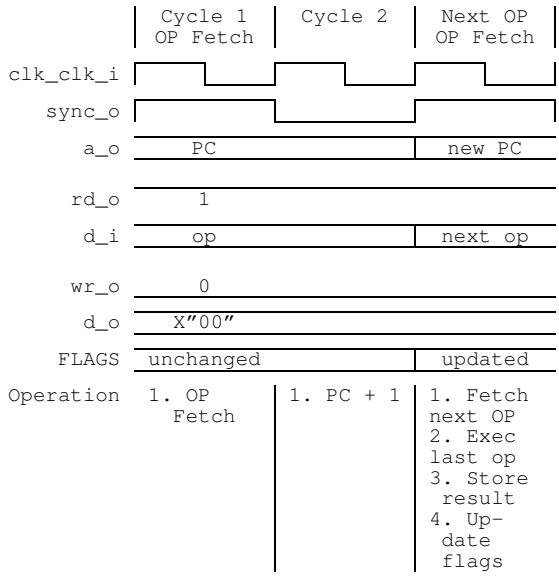


Figure (32): ASL A, LSR A, ROL A, ROR A – Timing Diagram "Immediate"

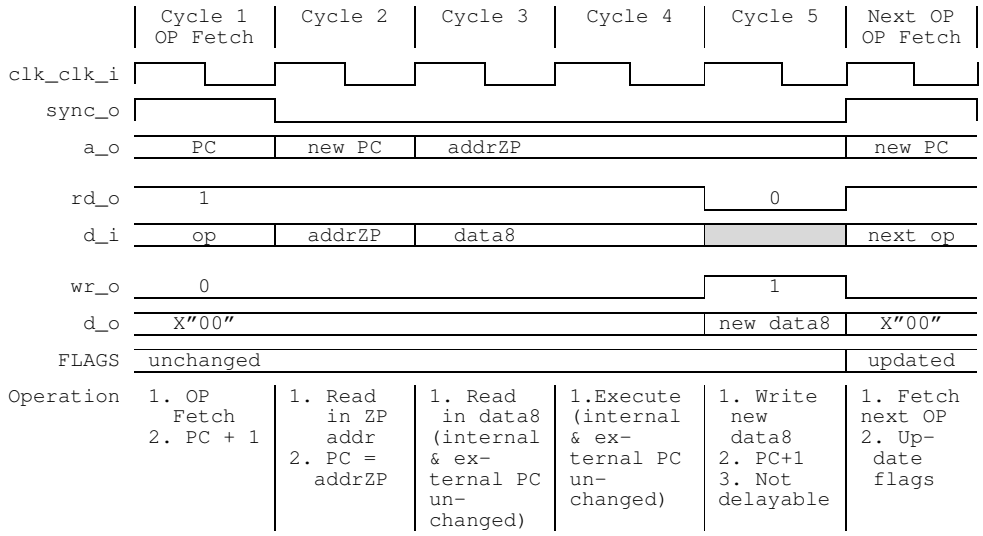


Figure (33): ASL, LSR, ROL, ROR, DEC, INC – Timing Diagram "Zero Page"

ASL, LSR, ROL, ROR, DEC, INC (cont.)

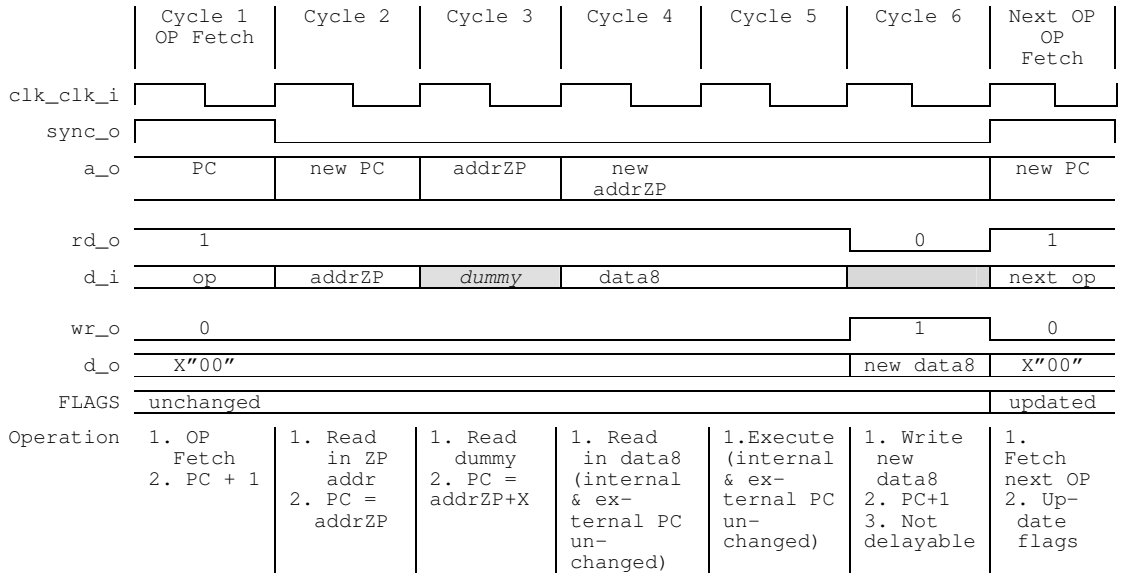


Figure (34): ASL, LSR, ROL, ROR, DEC, INC – Timing Diagram "Zero Page, X"

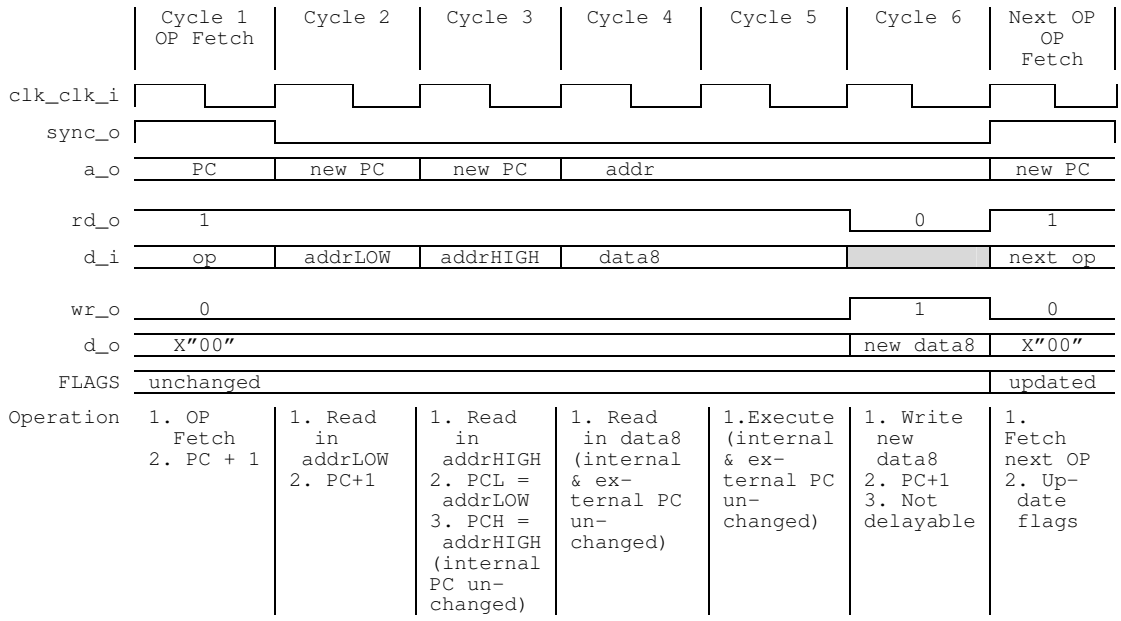


Figure (35): ASL, LSR, ROL, ROR, DEC, INC – Timing Diagram "Absolute"

ASL, LSR, ROL, ROR, DEC, INC (cont.)

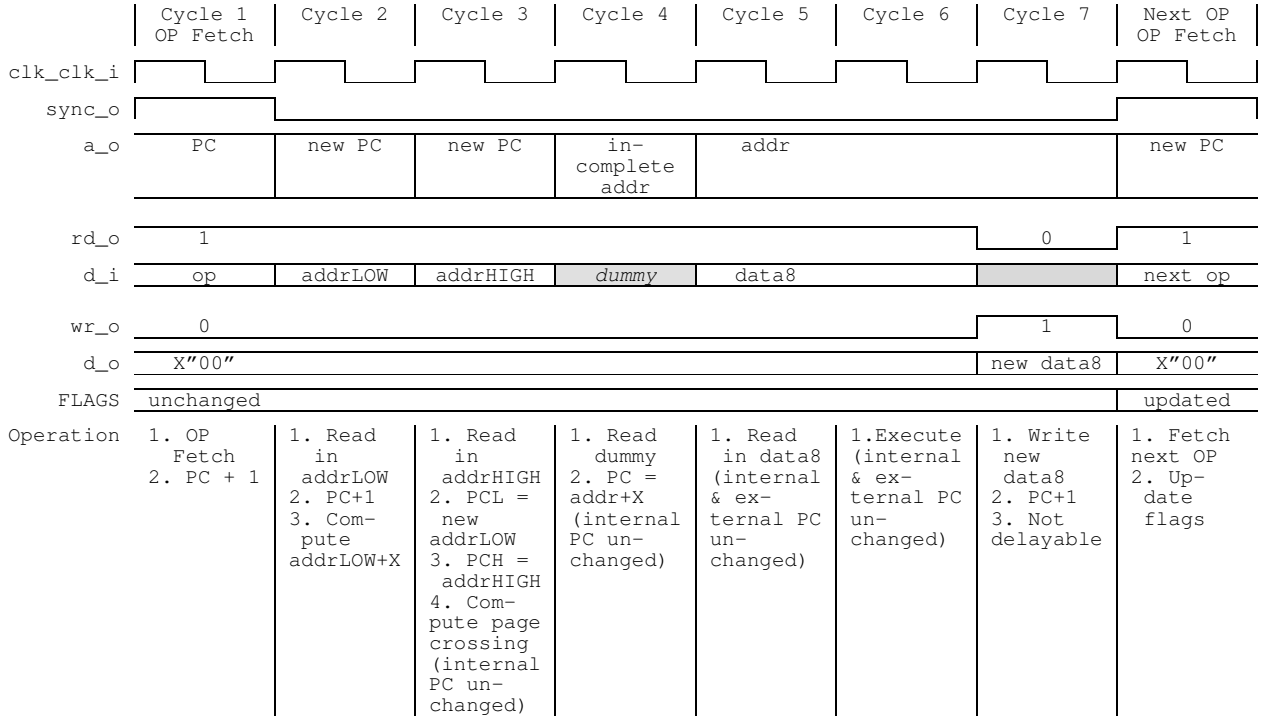


Figure (36): ASL, LSR, ROL, ROR, DEC, INC – Timing Diagram "Absolute, X"

CLC, CLD, CLI, CLV, DEX, DEY, INX, INY, SEC, SED, SEI, TAX, TAY, TSX, TXA, TYA

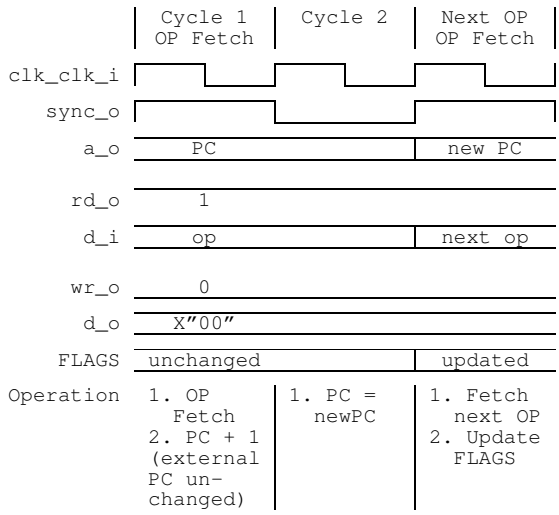


Figure (37): CLC, CLD, CLI, CLV, DEX, DEY, INX, INY, SEC, SED, SEI, TAX, TAY, TSX, TXA, TYA – Timing Diagram

JMP

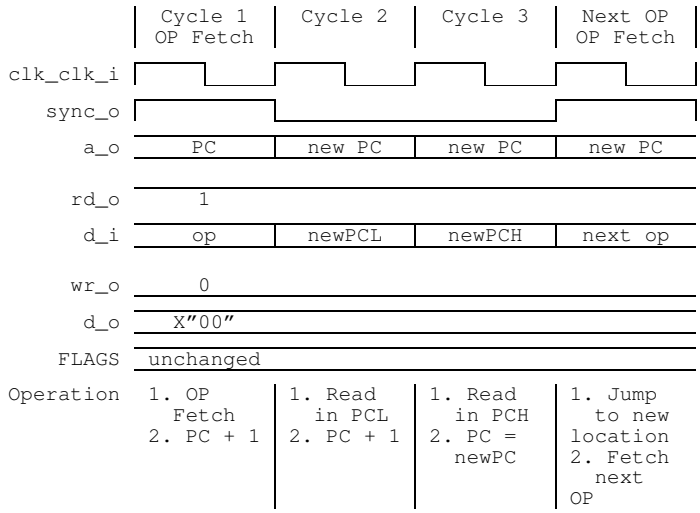


Figure (38): JMP – Timing Diagram "Absolute"

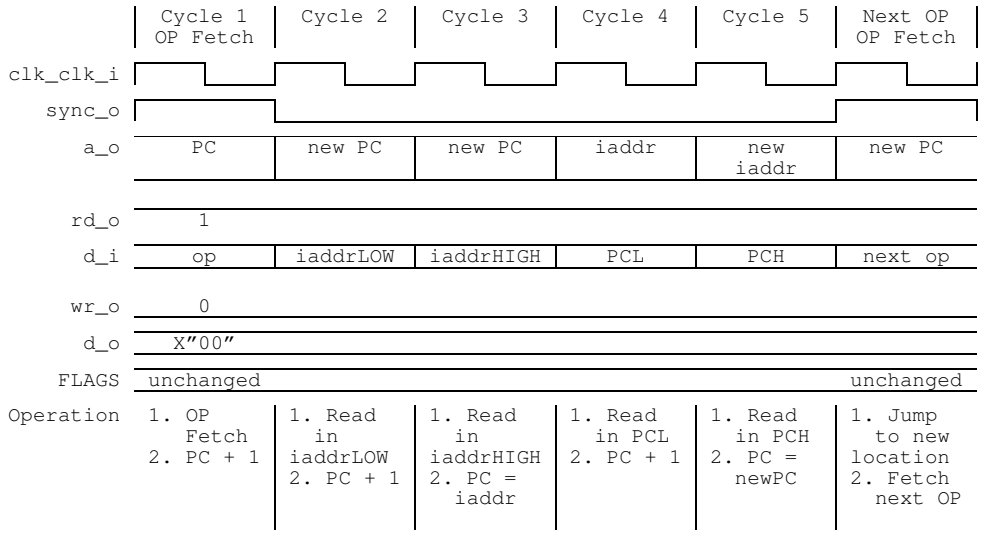


Figure (39): JMP – Timing Diagram "Indirect"

JSR

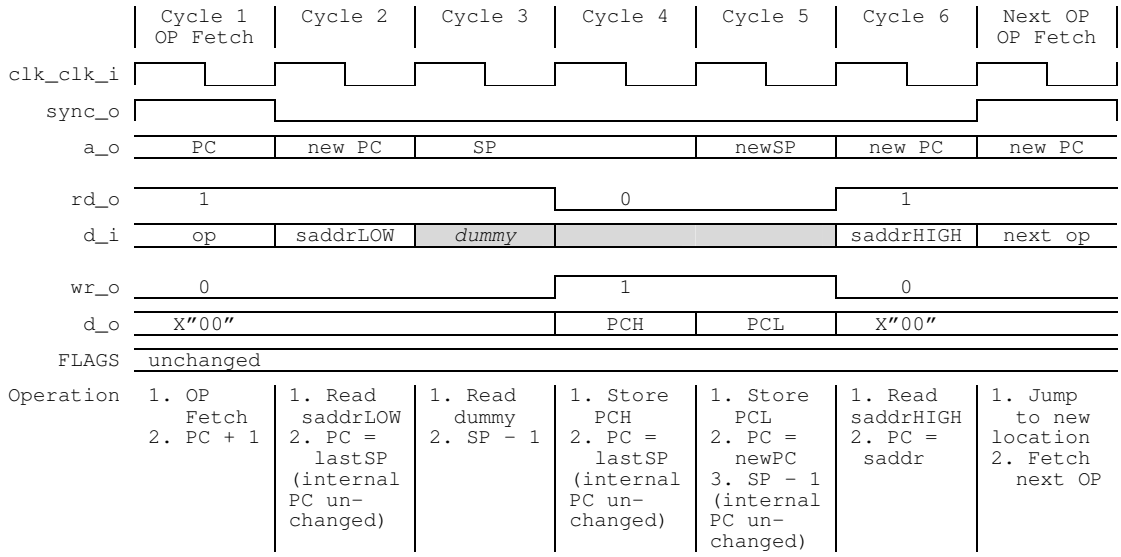


Figure (40): JSR – Timing Diagram

NOP, TXS

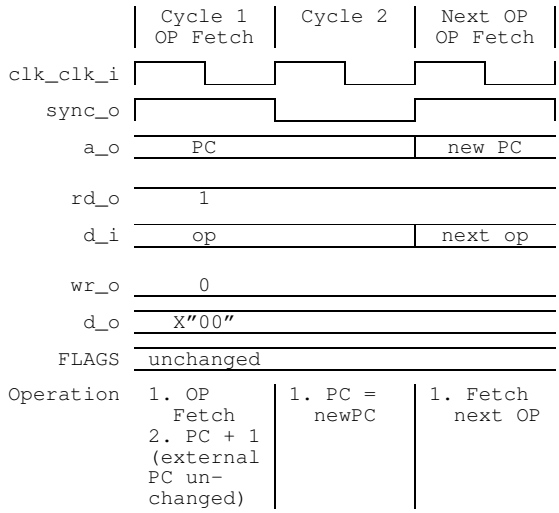


Figure (41): NOP, TXS – Timing Diagram

PLA, PLP

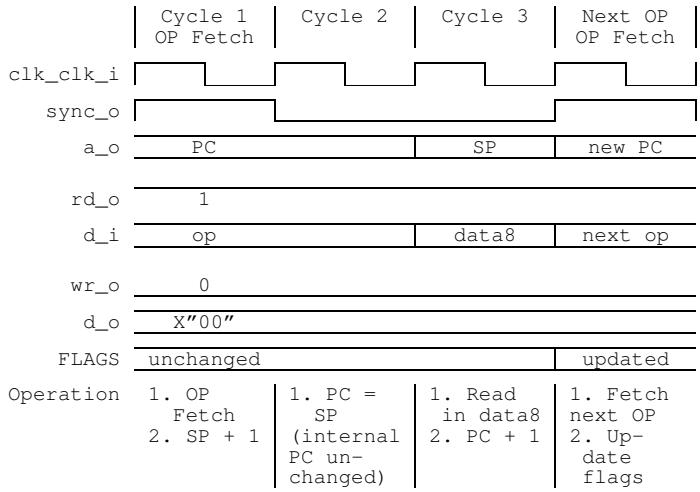


Figure (42): PLA, PLP – Timing Diagram "Implied"

PHA, PHP

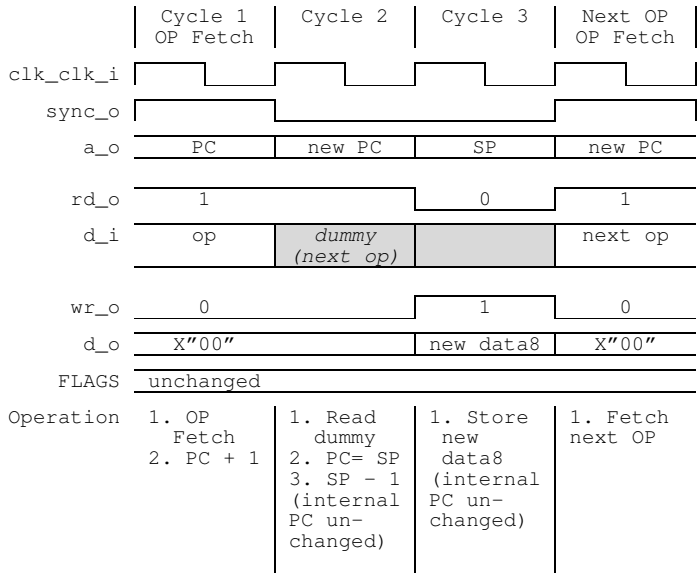


Figure (43): PHA, PHP – Timing Diagram "Implied"

RTI

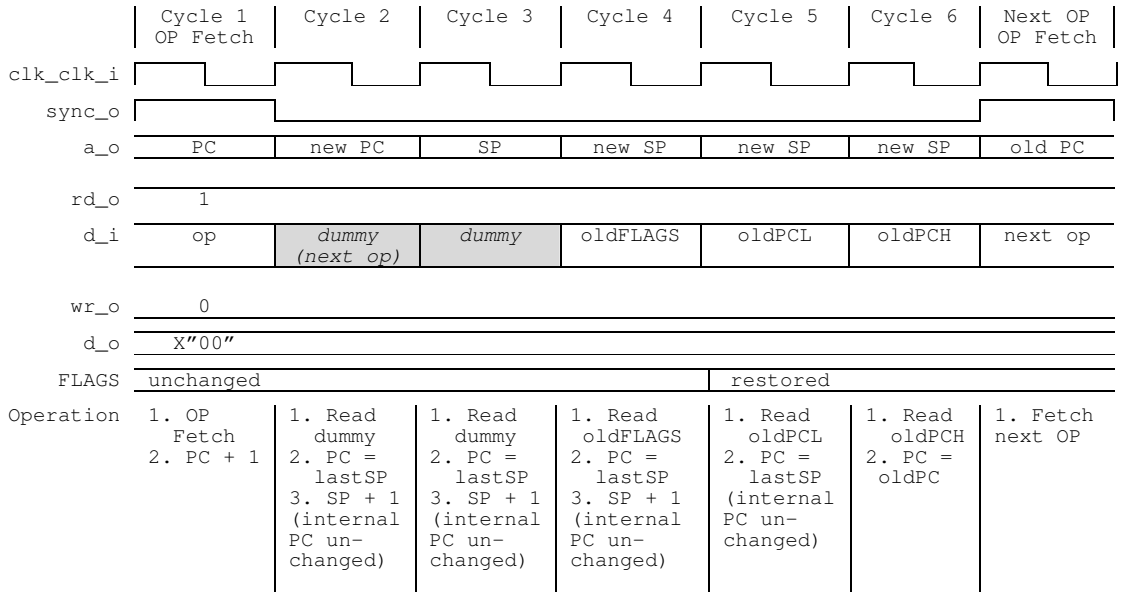


Figure (44): RTI – Timing Diagram "Implied"

RTS

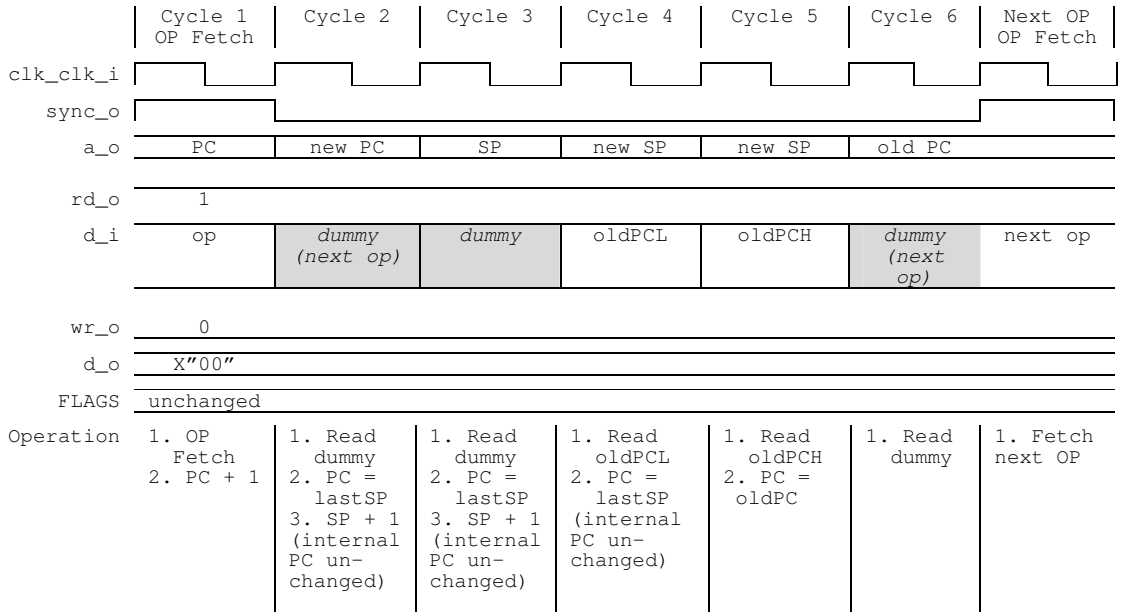


Figure (45): RTS – Timing Diagram "Implied"

STA, STX, STY

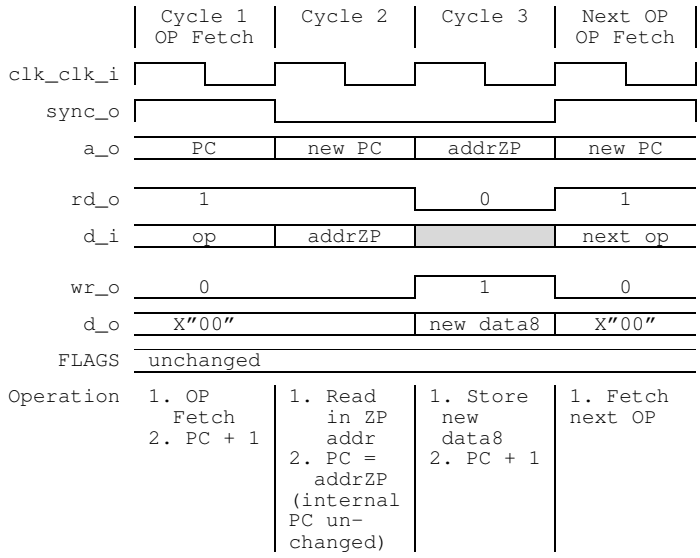


Figure (46): STA, STX, STY – Timing Diagram "Zero Page"

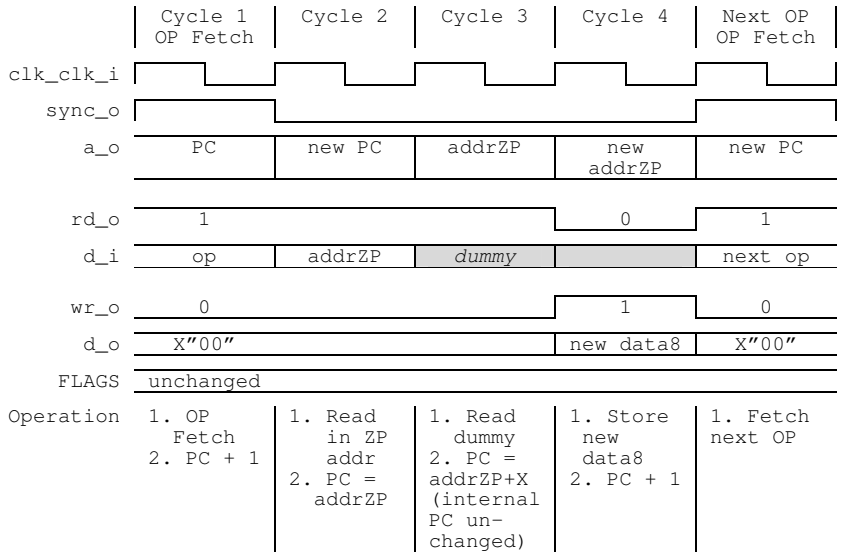


Figure (47): STA, STY – Timing Diagram "Zero Page, X"

STA, STX, STY (cont.)

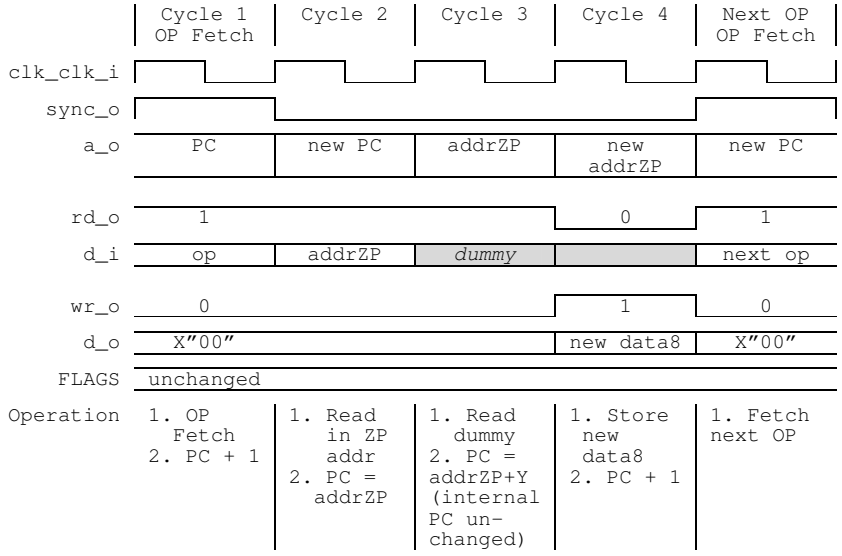


Figure (48): STX – Timing Diagram "Zero Page, Y"

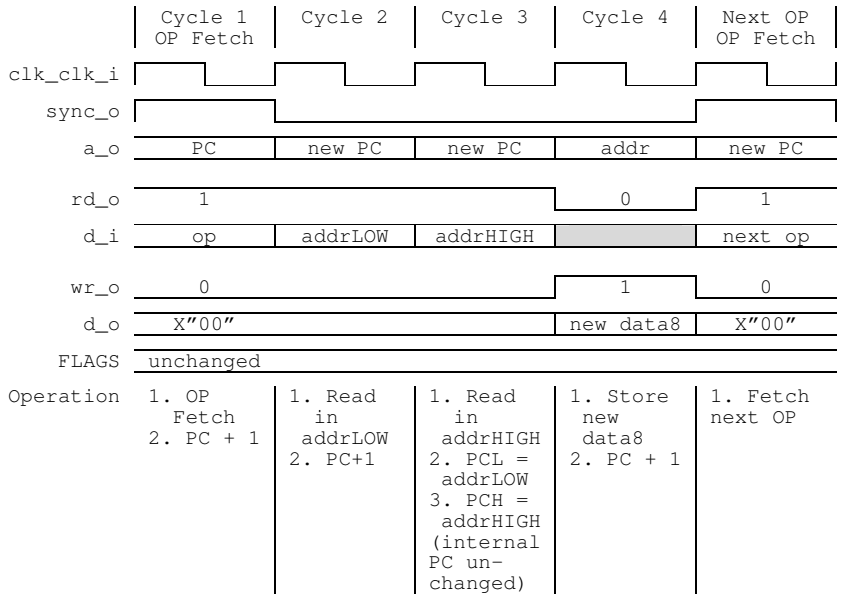


Figure (49): STA, STX, STY – Timing Diagram "Absolute"

STA, STX, STY (cont.)

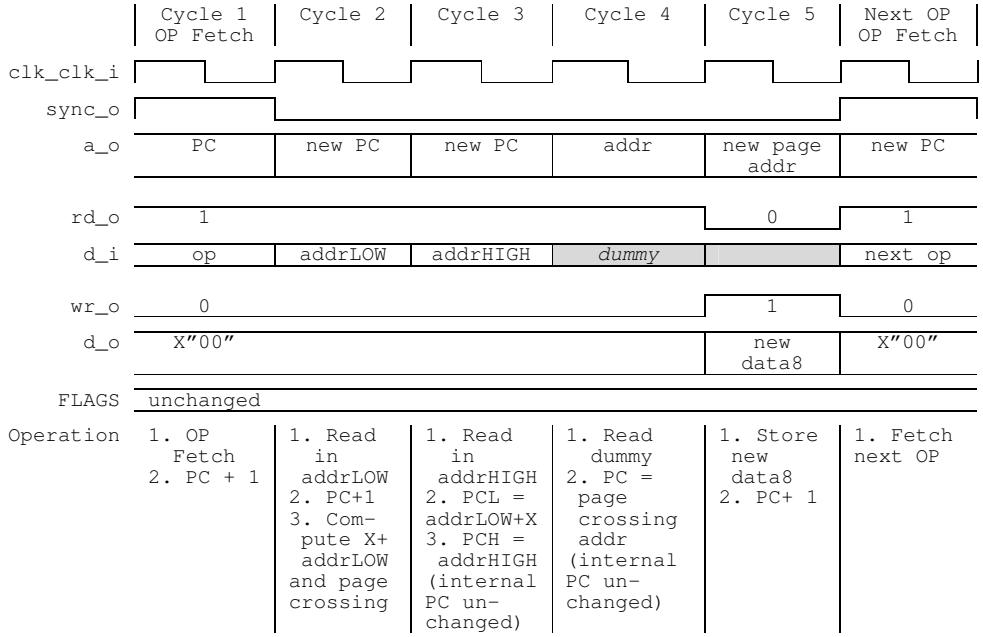


Figure (50): STA – Timing Diagram "Absolute, X"

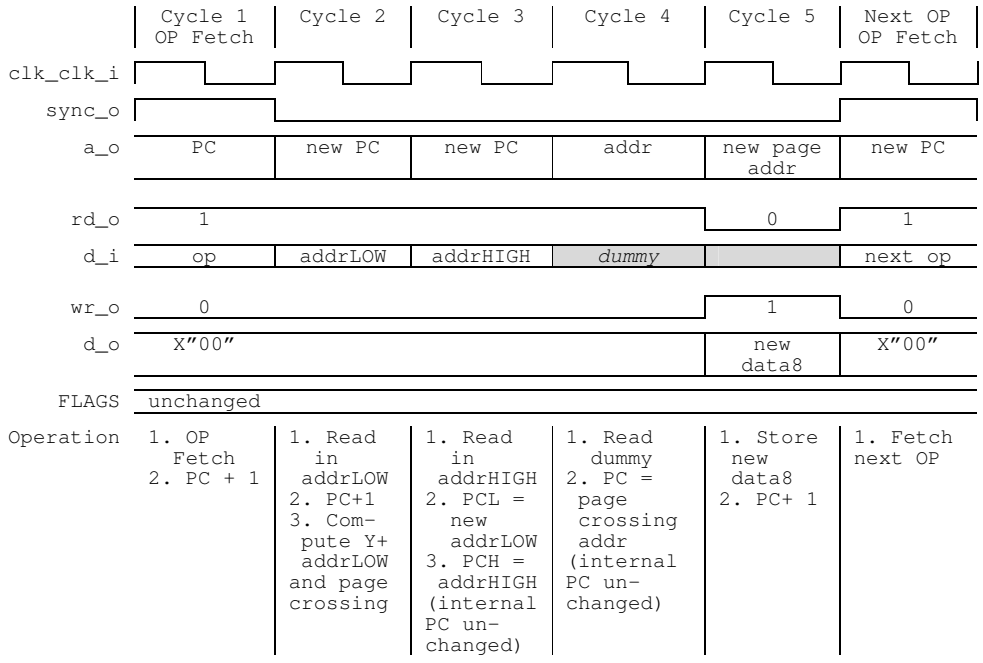


Figure (51): STA – Timing Diagram "Absolute, Y"

STA, STX, STY (cont.)

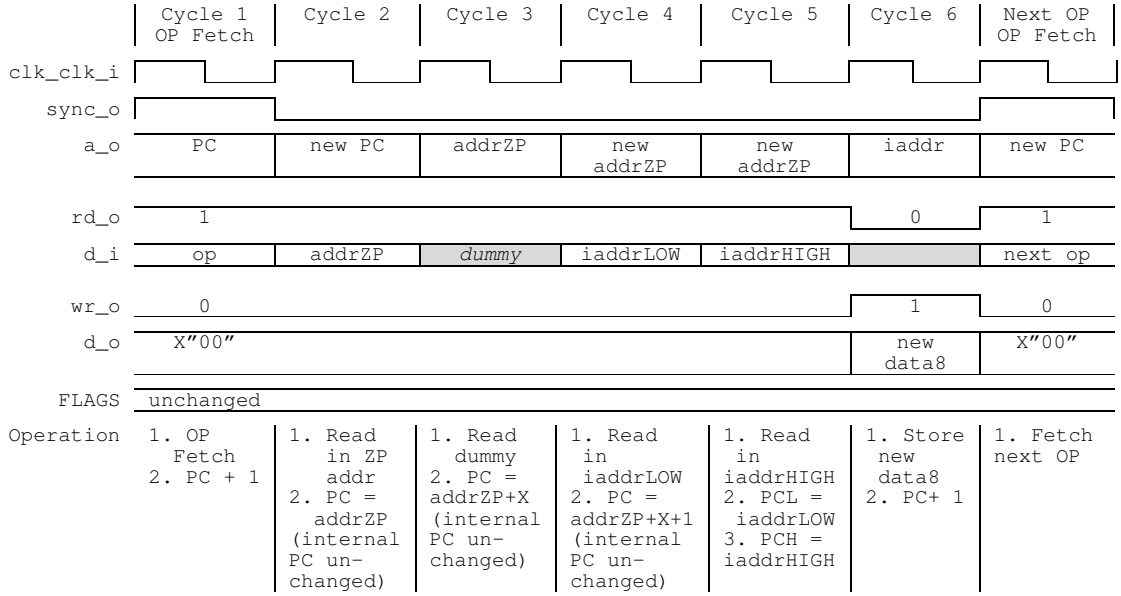


Figure (52): STA – Timing Diagram "(Indirect, X)"

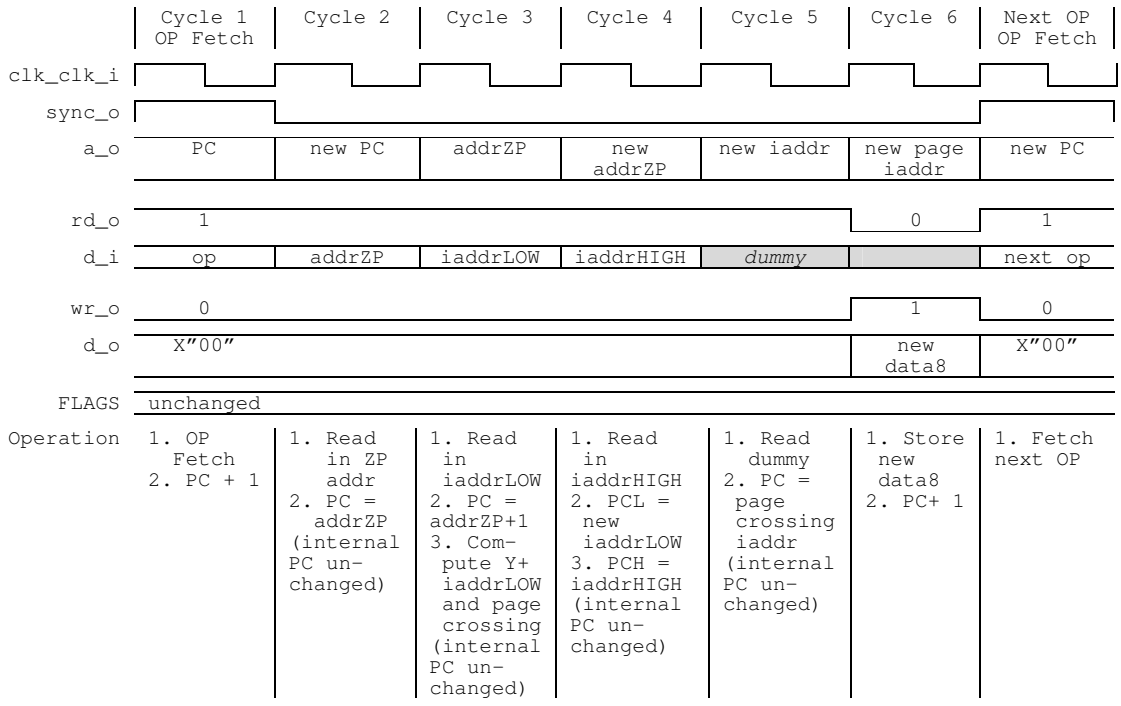


Figure (53): STA – Timing Diagram "(Indirect), Y"

Instruction Table

MSB \ LSB	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	BRK Implied 1 7	ORA (IND,X) 2 6				ORA ZP 2 3	ASL ZP 2 5		PHP Implied 1 3	ORA IMM 2 2	ASL Accum 1 2			ORA ABS 3 4	ASL ABS 3 6	
1	BPL Relative 2 2**	ORA (IND),Y 2 5*				ORA ZP,X 2 4	ASL ZP,X 2 6		CLC Implied 1 3	ORA ABS,Y 3 4*				ORA ABS,X 3 4*	ASL ABS,X 3 7	
2	JSR ABS 3 6	AND (IND,X) 2 6			BIT ZP 2 3	AND ZP 2 3	ROL ZP 2 5		PLP Implied 1 3	AND IMM 2 2	ROL Accum 1 2		BIT ABS 3 4	AND ABS 3 4	ROL ABS 3 6	
3	BMI Relative 2 2**	AND (IND),Y 2 5*				AND ZP,X 2 4	ROL ZP,X 2 6		SEC Implied 1 3	AND ABS,Y 2 4*				AND ABS,X 3 4*	ROL ABS,X 3 7	
4	RTI Implied 1 6	EOR (IND,X) 2 6				EOR ZP 2 3	LSR ZP 2 5		PHA Implied 1 3	EOR IMM 2 2	LSR Accum 1 2		JMP ABS 3 3	EOR ABS 3 4	LSR ABS 3 6	
5	BVC Relative 2 2**	EOR (IND),Y 2 5*				EOR ZP,X 2 4	LSR ZP,X 2 6		CLI Implied 1 3	EOR ABS,Y 3 4*				EOR ABS,X 3 4*	LSR ABS,X 3 7	
6	RTS Implied 1 6	ADC (IND,X) 2 6				ADC ZP 2 3+	ROR ZP 2 5		PLA Implied 1 3	ADC IMM 2 2+	ROR Accum 1 2		JMP (ABS) 3 5	ADC ABS 3 4+	ROR ABS 3 6	
7	BVS Relative 2 2**	ADC (IND),Y 2 5*+				ADC ZP,X 2 4+	ROR ZP,X 2 6		SEI Implied 1 3	ADC ABS,Y 3 4*+				ADC ABS,X 3 4*+	ROR ABS,X 3 7	
8		STA (IND,X) 2 6			STY ZP 2 3	STA ZP 2 3	STX ZP 2 3		DEY Implied 1 3		TXA Implied 1 2		STY ABS 3 4	STA ABS 3 4	STX ABS 3 4	
9	BCC Relative 2 2**	STA (IND),Y 2 6			STY ZP,X 2 4	STA ZP,X 2 4	STX ZP,Y 2 4		TYA Implied 1 3	STA ABS,Y 3 5	TXS Implied 1 2			STA ABS,X 3 5		
A	LDY IMM 2 2	LDA (IND,X) 2 6			LDY ZP 2 3	LDA ZP 2 3	LDX ZP 2 3		TAY Implied 1 3	LDA IMM 2 2	TAX Implied 1 2		LDY ABS 3 4	LDA ABS 3 4	LDX ABS 3 4	
B	BCS Relative 2 2**	LDA (IND),Y 2 5*			LDY ZP,X 2 4	LDA ZP,X 2 4	LDX ZP,Y 2 4		CLV Implied 1 3	LDA ABS,Y 3 4*	TSX Implied 1 2		LDY ABS,X 3 4*	LDA ABS,X 3 4*	LDX ABS,Y 3 4*	
C	CPY IMM 2 2	CMP (IND,X) 2 6			CPY ZP 2 3	CMP ZP 2 3	DEC ZP 2 5		INY Implied 1 3	CMP IMM 2 2	DEX Implied 1 2		CPY ABS 3 4	CMP ABS 3 4	DEC ABS 3 6	
D	BNE Relative 2 2**	CMP (IND),Y 2 5*				CMP ZP,X 2 4	DEC ZP,X 2 6		CLD Implied 1 3	CMP ABS,Y 3 4*				CMP ABS,X 3 4*	DEC ABS,X 3 7	
E	CPX IMM 2 2	SBC (IND,X) 2 6+			CPX ZP 2 3	SBC ZP 2 3+	INC ZP 2 5		INX Implied 1 3	SBC IMM 2 2+	NOP Implied 1 2		CPX ABX 3 4	SBC ABS 3 4+	INC ABS 3 6	
F	BEQ Relative 2 2**	SBC (IND),Y 2 5*+				SBC ZP,X 2 4+	INC ZP,X 2 6		SED Implied 1 3	SBC ABS,Y 3 4*+				SBC ABS,X 3 4*+	INC ABS,X 3 7	

\+ Add 1 to N if in decimal mode, * ADD 1 to N if page boundary is crossed
** Add 1 to N if branch occurs to same page – Add 2 to N if branch occurs to different page

Index

This section contains an alphabetical list of helpful document entries with their corresponding page numbers.