6502 Microprocessor

DavidSolid (LateX document), Unknown (Original)

December 2, 2018

Contents

1 Introduction

Most of the following information has been taking out of the "Commodore 64 Programmers Reference Manual" simply because it was available in electronic form and there appears to be no difference between this documentation and the 6502 documentation, they are both from the 6500 family after all. I've made changes and additions where appropriate.

In theory you should be able to use any code you can find for emulating the 6510 (the C64 processor).

2 Registers

Almost all calculations are done in the microprocessor. Registers are special pieces of memory in the processor which are used to carry out, and store information about calculations. The 6502 has the following registers:

2.1 Accumulator (A)

This is **the most important register** in the microprocessor. Various machine language instructions allow you to copy the contents of a memory location into the accumulator, copy the contents of the accumulator into a memory location, modify the contents of the accumulator or some other register directly, without affecting any memory. And the accumulator is the only register that has instructions for performing math.

2.2 X Index Register

This is a very important register. There are instructions for nearly all of the transformations you can make to the accumulator. But there are other instructions for things that only the X register can do. Various machine language instructions allow you to copy the contents of a memory location into the X register, copy the contents of the X register into a memory location, and modify the contents of the X, or some other register directly.

2.3 Y Index Register

This is a very important register. There are instructions for nearly all of the transformations you can make to the accumulator, and the X register. But there are other instructions for things that only the Y register can do. Various machine language instructions allow you to copy the contents of a memory location into the Y register, copy the contents of the Y register into a memory location, and modify the contents of the Y, or some other register directly.

2.4 Status register

This register consists of eight "flags" (a flag = something that indi- cates whether something has, or has not occurred). Bits of this register are altered depending on the result of arithmetic and logical operations. These bits are described below:

Bit No.	7	6	5	4	3	2	1	0
Simbols	S	V		B	D	I	Z	C

Table 1: Status register layout

- Bit 0 - C - Carry flag: this holds the carry out of the most significant bit in any arithmetic operation. In subtraction operations however, this flag is cleared - set to 0 - if a borrow is required, set to 1 - if no borrow is required. The carry flag is also used in shift and rotate logical operations.
- Bit 1 - Z - Zero flag: this is set to 1 when any arithmetic or logical operation produces a zero result, and is set to 0 if the result is non-zero.
- Bit 2 - I: this is an interrupt enable/disable flag. If it is set, interrupts are disabled. If it is cleared, interrupts are enabled.
- Bit 3 - D: this is the decimal mode status flag. When set, and an Add with Carry or Subtract with Carry instruction is executed, the source values are treated as valid BCD (Binary Coded Decimal, eg. 0x00-0x99 = 0-99) numbers. The result generated is also a BCD number.
- Bit 4 - B: this is set when a software interrupt (BRK instruction) is executed.
- Bit 5: not used. Supposed to be logical 1 at all times.
- Bit 6 - V - Overflow flag: when an arithmetic operation produces a result too large to be represented in a byte, V is set.
- Bit 7 - S - Sign flag: this is set if the result of an operation is negative, cleared if positive.

The most commonly used flags are C, Z, V, S.

2.5 Program Counter

This contains the address of the current machine language instruction being executed. Since the operating system is always "RUN"ning in the Commodore VIC-20 (or, for that matter, any computer), the program counter is always changing. It could only be stopped by halting the microprocessor in some way.

2.6 Stack pointer

This register contains the location of the first empty place on the stack. The stack is used for temporary storage by machine language pro- grams, and by the computer.

3 Addressing Modes

Instructions need operands to work on. There are various ways of indicating where the processor is to get these operands. The different methods used to do this are called addressing modes. The 6502 offers 11 modes, as described below.

3.1 Immediate

In this mode the operand's value is given in the instruction itself. In assembly language this is indicated by "#" before the operand. eg. LDA #$0A - means "load the accumulator with the hex value 0A" In machine code different modes are indicated by different codes. So LDA would be translated into different codes depending on the addressing mode. In this mode, it is: $A9 $0A

3.2 Absolute and Zero-page Absolute

In these modes the operands address is given. eg. LDA $31F6 - (assembler) $AD $31F6 - (machine code) If the address is on zero page - i.e. any address where the high byte is 00 - only 1 byte is needed for the address. The processor automatically fills the 00 high byte. eg. LDA $F4 $A5 $F4 Note the different instruction codes for the different modes. Note also that for 2 byte addresses, the low byte is store first, eg. LDA $31F6 is stored as three bytes in memory, $AD $F6 $31. Zero-page absolute is usually just called zero-page.

3.3 Implied

No operand addresses are required for this mode. They are implied by the instruction. eg. TAX - (transfer accumulator contents to X-register) $AA

3.4 Accumulator

In this mode the instruction operates on data in the accumulator, so no operands are needed. eg. LSR - logical bit shift right $4A

3.5 Indexed and Zero-page Indexed

In these modes the address given is added to the value in either the X or Y index register to give the actual address of the operand. eg. LDA $31F6, Y $D9 $31F6 LDA $31F6, X $DD $31F6 Note that the different operation codes determine the index register used. In the zero-page version, you should note

that the X and Y registers are not interchangeable. Most instructions which can be used with zero-page indexing do so with X only. eg. LDA $20, X $B5 $20

3.6 Indirect

This mode applies only to the JMP instruction - JuMP to new location. It is indicated by parenthesis around the operand. The operand is the address of the bytes whose value is the new location. eg. JMP ($215F) Assume the following -

byte	value
$215F	$76
$2160	$30

This instruction takes the value of bytes $215F, $2160 and uses that as the address to jump to - i.e. $3076 (remember that addresses are stored with low byte first).

3.7 Pre-indexed indirect

In this mode a zer0-page address is added to the contents of the X-register to give the address of the bytes holding the address of the operand. The indirection is indicated by parenthesis in assembly language. eg. LDA ($3E, X) $A1 $3E Assume the following -

byte	value
X-reg.	$05
$0043	$15
$0044	$24
$2415	$6E

Then the instruction is executed by:

1. adding $3E and $05 = $0043
2. getting address contained in bytes $0043, $0044 = $2415
3. loading contents of $2415 - i.e. $6E - into accumulator
4. Note
   (a) When adding the 1-byte address and the X-register, wrap around addition is used - i.e. the sum is always a zero-page address. eg. FF + 2 = 0001 not 0101 as you might expect. DON'T FORGET THIS WHEN EMULATING THIS MODE.
   (b) Only the X register is used in this mode.

3.8 Post-indexed indirect

In this mode the contents of a zero-page address (and the following byte) give the indirect addressm which is added to the contents of the Y-register to yield the actual address of the operand. Again, inassembly language, the instruction is indicated by parenthesis. eg. LDA ($4C), Y Note that the parenthesis are only around the 2nd byte of the instruction since it is the part that does the indirection. Assume the following -

byte	value
$004C	$00
$004D	$21
Y-reg.	$05
$2105	$6D

Then the instruction above executes by:

1. getting the address in bytes $4C, $4D = $2100
2. adding the contents of the Y-register = $2105
3. loading the contents of the byte $2105 - i.e. $6D into the accumulator.

Note: only the Y-register is used in this mode.

3.9 Relative

This mode is used with Branch-on-Condition instructions. It is probably the mode you will use most often. A 1 byte value is added to the program counter, and the program continues execution from that address. The 1 byte number is treated as a signed number - i.e. if bit 7 is 1, the number given byt bits 0-6 is negative; if bit 7 is 0, the number is positive. This enables a branch displacement of up to 127 bytes in either direction.

Table 2: Example

bit no.	7	6	5	4	3	2	1	0	signed value	unsigned value
value	1	0	1	0	0	1	1	1	-39	$A7
value	0	0	1	0	0	1	1	1	+39	$27

Instruction example: BEQ $A7 $F0 $A7 This instruction will check the zero status bit. If it is set, 39 decimal will be subtracted from the program counter and execution continues from that address. If the zero status bit is not set, execution continues from the following instruction. Notes:

1. The program counter points to the start of the instruction after the branch instruction before the branch displacement is added. Remember to take this into account when calculating displacements.

2. Branch-on-condition instructions work by checking the relevant status bits in the status register. Make sure that they have been set or unset as you want them. This is often done using a CMP instruction.

3. If you find you need to branch further than 127 bytes, use the opposite branch-on-condition and a JMP.

4 Instruction Set

MCS6502 MICROPROCESSOR INSTRUCTION SET - ALPHABETIC SEQUENCE	
Mnemonic	Description
ADC	Add Memory to Accumulator with Carry
AND	"AND" Memory with Accumulator
ASL	Shift Left One Bit (Memory or Accumulator)
BCC	Branch on Carry Clear
BCS	Branch on Carry Set
BEQ	Branch on Result Zero
BIT	Test Bits in Memory with Accumulator
BMI	Branch on Result Minus
BNE	Branch on Result not Zero
BPL	Branch on Result Plus
BRK	Force Break
BVC	Branch on Overflow Clear
BVS	Branch on Overflow Set
CLC	Clear Carry Flag
CLD	Clear Decimal Mode
CLI	Clear interrupt Disable Bit
CLV	Clear Overflow Flag
CMP	Compare Memory and Accumulator
CPX	Compare Memory and Index X
CPY	Compare Memory and Index Y
DEC	Decrement Memory by One
DEX	Decrement Index X by One
DEY	Decrement Index Y by One
EOR	"Exclusive-Or" Memory with Accumulator
INC	Increment Memory by One
INX	Increment Index X by One
INY	Increment Index Y by One
JMP	Jump to New Location
Continues in next page	

JSR	Jump to New Location Saving Return Address
LDA	Load Accumulator with Memory
LDX	Load Index X with Memory
LDY	Load Index Y with Memory
LSR	Shift Right One Bit (Memory or Accumulator)
NOP	No Operation
ORA	"OR" Memory with Accumulator
PHA	Push Accumulator on Stack
PHP	Push Processor Status on Stack
PLA	Pull Accumulator from Stack
PLP	Pull Processor Status from Stack
ROL	Rotate One Bit Left (Memory or Accumulator)
ROR	Rotate One Bit Right (Memory or Accumulator)
RTI	Return from Interrupt
RTS	Return from Subroutine
SBC	Subtract Memory from Accumulator with Borrow
SEC	Set Carry Flag
SED	Set Decimal Mode
SEI	Set Interrupt Disable Status
STA	Store Accumulator in Memory
STX	Store Index X in Memory
STY	Store Index Y in Memory
TAX	Transfer Accumulator to Index X
TAY	Transfer Accumulator to Index Y
TSX	Transfer Stack Pointer to Index X
TXA	Transfer Index X to Accumulator
TXS	Transfer Index X to Stack Pointer
TYA	Transfer Index Y to Accumulator

Table 3: List of documented opcodes

The following notation applies to this summary:

Note: At the top of each table is located in parentheses a reference number (Ref: XX) which directs the user to that Section in the MCS6500 Microcomputer Family Programming Manual in which the instruction is defined and discussed.

Symbol	Meaning	Symbol	Meaning
A	Accumulator	EOR	Logical Exclusive Or
X, Y	Index Registers	fromS	Transfer from Stack
M	Memory	toS	Transfer to Stack
P	Processor Status Register	$\Rightarrow$	Transfer to
S	Stack Pointer	$\Leftarrow$	Transfer from
/	Change	$\vee$	Logical OR
-	No Change	PC	Program Counter
+	Add	PCH	Program Counter High
$\wedge$	Logical AND	PCL	Program Counter Low
-	Subtract	OPER	OPERAND
		#	IMMEDIATE ADDRESSING MODE

Table 4: Notation table

4.1 ADC (Add memory to accumulator with carry)

Operation: $A + M + C \implies A, C$

N	Z	C	I	D	V
/	/	/	-	-	/

(Ref: 2.2.1)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
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	60	3	4
Absolute,X	ADC Oper,X	70	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.

4.2 AND ("AND" memory with accumulator)

Operation: $A \wedge M \implies A$

N	Z	C	I	D	V
/	/	-	-	-	-

(Ref: 2.2.3.0)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
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.

4.3 ASL (ASL Shift Left One Bit (Memory or Accumulator))

Operation: $C \Leftarrow$ | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | $\Leftarrow 0$

N	Z	C	I	D	V
/	/	/	-	-	-

(Ref: 10.2)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
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

4.4 BCC (Branch on Carry Clear)

Operation: Branch on C = 0

N	Z	C	I	D	V
-	-	-	-	-	-

(Ref: 4.1.1.3)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BCC Oper	90	2	2*

* Add 1 if branch occurs to same page. * Add 2 if branch occurs to different page.

4.5 BCS (BCS Branch on carry set)

Operation: Branch on C = 1

N	Z	C	I	D	V
-	-	-	-	-	-

(Ref: 4.1.1.4)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BCS Oper	B0	2	2*

* Add 1 if branch occurs to same page. * Add 2 if branch occurs to next page.

4.6 BEQ (BEQ Branch on result zero)

Operation: Branch on Z = 1

N	Z	C	I	D	V
-	-	-	-	-	-

(Ref: 4.1.1.5)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BEQ Oper	F0	2	2*

* Add 1 if branch occurs to same page. * Add 2 if branch occurs to next page.

4.7 BIT (BIT Test bits in memory with accumulator)

Operation: $A \wedge M, M7 \implies N, M6 \implies V$

Bit 6 and 7 are transferred to the status register. If the result of $A \wedge M$ is zero then Z = 1, otherwise Z = 0

N	Z	C	I	D	V
M7	/	_	_	_	M6

(Ref: 4.2.1.1)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Zero Page	BIT Oper	24	2	3
Absolute	BIT Oper	2C	3	4

4.8 BMI (BMI Branch on result minus)

Operation: Branch on N = 1

N	Z	C	I	D	V
-	-	-	-	-	-

(Ref: 4.1.1.1)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BMI Oper	30	2	2*

* Add 1 if branch occurs to same page. * Add 1 if branch occurs to different page.

4.9 BNE (BNE Branch on result not zero)

Operation: Branch on Z = 0

N	Z	C	I	D	V
-	-	-	-	-	-

(Ref: 4.1.1.6)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BMI Oper	D0	2	2*

* Add 1 if branch occurs to same page. * Add 2 if branch occurs to different page.

4.10 BPL (BPL Branch on result plus)

Operation: Branch on N = 0

N	Z	C	I	D	V
-	-	-	-	-	-

(Ref: 4.1.1.2)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BPL Oper	10	2	2*

* Add 1 if branch occurs to same page. * Add 2 if branch occurs to different page.

4.11 BRK (BRK Force Break)

Operation: Forced Interrupt PC + 2 toS P toS

N	Z	C	I	D	V
-	-	-	1	-	-

(Ref: 9.11)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	BRK	00	1	7

1. A BRK command cannot be masked by setting I.

4.12 BVC (BVC Branch on overflow clear)

Operation: Branch on V = 0

N	Z	C	I	D	V
-	-	-	-	-	-

(Ref: 4.1.1.8)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BVC Oper	50	2	2*

* Add 1 if branch occurs to same page. * Add 2 if branch occurs to different page.

4.13 BVS (BVS Branch on overflow set)

Operation: Branch on V = 1

N	Z	C	I	D	V
-	-	-	-	-	-

(Ref: 4.1.1.7)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BVS Oper	70	2	2*

* Add 1 if branch occurs to same page. * Add 2 if branch occurs to different page.

4.14 CLC (CLC Clear carry flag)

Operation: 0 -¿ C

N	Z	C	I	D	V
-	-	0	-	-	-

(Ref: 3.0.2)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	CLC	18	1	2

4.15 CLD (CLD Clear decimal mode)

Operation: 0 -¿ D

N	A	C	I	D	V
-	-	-	-	0	-

(Ref: 3.3.2)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	CLD	D8	1	2

4.16 CLI (CLI Clear interrupt disable bit)

Operation: 0 -¿ I

N	Z	C	I	D	V
-	-	-	0	-	-

(Ref: 3.2.2)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	CLI	58	1	2