6502 Microprocessor

Revision 1.02 by _Bnu.

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).

THE REGISTERS INSIDE THE 6502 MICROPROCESSOR

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:

THE ACCUMULATOR

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.

THE 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.

THE 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.

THE STATUS REGISTER

This register consists of eight "flags" (a flag = something that indicates 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
	S	V		B	D	I	Z	C

Bit0 - 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.

Bit1 - 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.

THE 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.

THE 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 programs, and by the computer.

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.

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

2 & 3) 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.

4) Implied
No operand addresses are required for this mode. They are implied by the instruction.

```
eg.  TAX - (transfer accumulator contents to X-register)
     $AA
```

5) Accumulator
In this mode the instruction operates on data in the accumulator, so no operands are needed.

```
eg.  LSR - logical bit shift right
     $4A
```

6 & 7) 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
```

8) 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).

9) 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:
(i)   adding $3E and $05 = $0043
(ii) getting address contained in bytes $0043, $0044 = $2415
(iii) loading contents of $2415 - i.e. $6E - into accumulator

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.

10) 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:
(i)   getting the address in bytes $4C, $4D = $2100
(ii) adding the contents of the Y-register = $2105
(111) loading the contents of the byte $2105 - i.e. $6D into the
      accumulator.
Note: only the Y-register is used in this mode.

11) 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.

eg	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: a) 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.
        b) 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.
        c) If you find you need to branch further than 127 bytes, use the
opposite branch-on-condition and a JMP.

```
+------------------------------------------------------------------------
|
|      MCS6502 MICROPROCESSOR INSTRUCTION SET - ALPHABETIC SEQUENCE
|
+------------------------------------------------------------------------
|
|     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
|
+------------------------------------------------------------------------


------------------------------------------------------------------------+
                                                                        |
       MCS6502 MICROPROCESSOR INSTRUCTION SET - ALPHABETIC SEQUENCE     |
                                                                        |
------------------------------------------------------------------------+
                                                                        |
      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

The following notation applies to this summary:

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

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.

ADC Add memory to accumulator with carry ADC

```
Operation:  A + M + C -> 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	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.

```
AND                  "AND" memory with accumulator                    AND

Operation:  A /\ M -> 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.

```
ASL          ASL Shift Left One Bit (Memory or Accumulator)           ASL
                 +-+-+-+-+-+-+-+-+
Operation:  C <- |7|6|5|4|3|2|1|0| <- 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

```
BCC                     BCC Branch on Carry Clear                     BCC
                                                           N Z C I D V
Operation:  Branch on C = 0                                _ _ _ _ _ _
                             (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.

```
BCS                      BCS Branch on carry set                      BCS

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.

```
BEQ                    BEQ Branch on result zero                      BEQ
                                                      N Z C I D V
Operation:  Branch on Z = 1                           _ _ _ _ _ _
                             (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.

```
BIT             BIT Test bits in memory with accumulator              BIT

Operation:  A /\ M, M7 -> N, M6 -> V

Bit 6 and 7 are transferred to the status register.   N Z C I D V
If the result of A /\ M is zero then Z = 1, otherwise M7/ _ _ _ M6
Z = 0
                             (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

```
BMI                    BMI Branch on result minus                     BMI

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.

```
BNE                   BNE Branch on result not zero                   BNE

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.

```
BPL                     BPL Branch on result plus                     BPL

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.

```
BRK                            BRK Force Break                         BRK

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.

```
BVC                     BVC Branch on overflow clear                   BVC

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.

```
BVS                      BVS Branch on overflow set                    BVS

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.

```
CLC                          CLC Clear carry flag                      CLC

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

```
CLD                         CLD Clear decimal mode                     CLD
```

```
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

```
CLI                  CLI Clear interrupt disable bit                CLI

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

```
CLV                      CLV Clear overflow flag                    CLV

Operation: 0 -> V                                   N Z C I D V
                                                    _ _ _ _ _ 0
                              (Ref: 3.6.1)
```

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	CLV	B8	1	2

```
CMP                CMP Compare memory and accumulator               CMP

Operation:  A - M                                   N Z C I D V
                                                    / / / _ _ _
                              (Ref: 4.2.1)
```

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

```
CPX                  CPX Compare Memory and Index X                 CPX
                                                    N Z C I D V
Operation:  X - M                                   / / / _ _ _
                               (Ref: 7.8)
```

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	CPX *Oper	E0	2	2
Zero Page	CPX Oper	E4	2	3
Absolute	CPX Oper	EC	3	4

```
CPY                  CPY Compare memory and index Y                  CPY
                                                      N Z C I D V
Operation:  Y - M                                     / / / _ _ _
                               (Ref: 7.9)
```

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	CPY *Oper	C0	2	2
Zero Page	CPY Oper	C4	2	3
Absolute	CPY Oper	CC	3	4

```
DEC                   DEC Decrement memory by one                    DEC

Operation:  M - 1 -> M                                N Z C I D V
                                                      / / _ _ _ _
                               (Ref: 10.7)
```

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

```
DEX                   DEX Decrement index X by one                   DEX

Operation:  X - 1 -> X                                N Z C I D V
                                                      / / _ _ _ _
                               (Ref: 7.6)
```

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	DEX	CA	1	2

```
DEY                   DEY Decrement index Y by one                   DEY

Operation:  X - 1 -> Y                                N Z C I D V
                                                      / / _ _ _ _
                               (Ref: 7.7)
```

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	DEY	88	1	2

```
EOR            EOR "Exclusive-Or" memory with accumulator            EOR

Operation:  A EOR M -> A                              N Z C I D V
                                                      / / _ _ _ _
                             (Ref: 2.2.3.2)
```

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
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	40	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.

```
INC                    INC Increment memory by one                     INC
                                                                N Z C I D V
Operation:  M + 1 -> M                                          / / _ _ _ _
                               (Ref: 10.6)
```

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

```
INX                    INX Increment Index X by one                    INX
                                                                N Z C I D V
Operation:  X + 1 -> X                                          / / _ _ _ _
                               (Ref: 7.4)
```

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	INX	E8	1	2

```
INY                    INY Increment Index Y by one                    INY

Operation:  X + 1 -> X                                          N Z C I D V
                                                                / / _ _ _ _
                               (Ref: 7.5)
```

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	INY	C8	1	2

```
JMP                     JMP Jump to new location                       JMP

Operation:  (PC + 1) -> PCL                                     N Z C I D V
            (PC + 2) -> PCH   (Ref: 4.0.2)                      _ _ _ _ _ _
                              (Ref: 9.8.1)
```

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Absolute	JMP Oper	4C	3	3
Indirect	JMP (Oper)	6C	3	5

```
JSR          JSR Jump to new location saving return address          JSR

Operation:  PC + 2 toS, (PC + 1) -> PCL                         N Z C I D V
                        (PC + 2) -> PCH                         _ _ _ _ _ _
                               (Ref: 8.1)
```

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Absolute	JSR Oper	20	3	6

```
LDA                  LDA Load accumulator with memory                 LDA

Operation:  M -> A                                 N Z C I D V
                                                   / / _ _ _ _
                              (Ref: 2.1.1)
```

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

```
LDX                   LDX Load index X with memory                    LDX

Operation:  M -> X                                 N Z C I D V
                                                   / / _ _ _ _
                               (Ref: 7.0)
```

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
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 when page boundary is crossed.

```
LDY                   LDY Load index Y with memory                    LDY
                                                   N Z C I D V
Operation:  M -> Y                                 / / _ _ _ _
                               (Ref: 7.1)
```

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
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 when page boundary is crossed.

```
LSR          LSR Shift right one bit (memory or accumulator)          LSR


                 +-+-+-+-+-+-+-+-+
Operation:  0 -> |7|6|5|4|3|2|1|0| -> C             N Z C I D V
                 +-+-+-+-+-+-+-+-+                  0 / / _ _ _
                               (Ref: 10.1)
```

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

Accumulator	LSR A	4A	1	2
Zero Page	LSR Oper	46	2	5
Zero Page,X	LSR Oper,X	56	2	6
Absolute	LSR Oper	4E	3	6
Absolute,X	LSR Oper,X	5E	3	7

```
NOP                          NOP No operation                          NOP
                                                      N Z C I D V
Operation:  No Operation (2 cycles)                   _ _ _ _ _ _
```

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	NOP	EA	1	2

```
ORA                  ORA "OR" memory with accumulator                  ORA

Operation: A V M -> A                                 N Z C I D V
                                                      / / _ _ _ _
                             (Ref: 2.2.3.1)
```

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	ORA #Oper	09	2	2
Zero Page	ORA Oper	05	2	3
Zero Page,X	ORA Oper,X	15	2	4
Absolute	ORA Oper	0D	3	4
Absolute,X	ORA Oper,X	10	3	4*
Absolute,Y	ORA Oper,Y	19	3	4*
(Indirect,X)	ORA (Oper,X)	01	2	6
(Indirect),Y	ORA (Oper),Y	11	2	5

* Add 1 on page crossing

```
PHA                    PHA Push accumulator on stack                   PHA

Operation:  A toS                                     N Z C I D V
                                                      _ _ _ _ _ _
                               (Ref: 8.5)
```

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	PHA	48	1	3

```
PHP                  PHP Push processor status on stack                PHP

Operation:  P toS                                     N Z C I D V
                                                      _ _ _ _ _ _
                               (Ref: 8.11)
```

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	PHP	08	1	3

```
PLA                  PLA Pull accumulator from stack                   PLA
```

```
Operation:  A fromS                                  N Z C I D V
                                                     _ _ _ _ _ _
                               (Ref: 8.6)
```

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	PLA	68	1	4

```
PLP               PLP Pull processor status from stack              PLA

Operation:  P fromS                                  N Z C I D V
                                                      From Stack
                               (Ref: 8.12)
```

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	PLP	28	1	4

```
ROL          ROL Rotate one bit left (memory or accumulator)         ROL

             +------------------------------+
             |         M or A               |
             |   +-+-+-+-+-+-+-+-+    +-+   |
Operation:   +-< |7|6|5|4|3|2|1|0| <- |C| <-+         N Z C I D V
                 +-+-+-+-+-+-+-+-+    +-+             / / / _ _ _
                               (Ref: 10.3)
```

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

```
ROR          ROR Rotate one bit right (memory or accumulator)        ROR

             +------------------------------+
             |                              |
             |   +-+    +-+-+-+-+-+-+-+-+   |
Operation:   +-> |C| -> |7|6|5|4|3|2|1|0| >-+         N Z C I D V
                 +-+    +-+-+-+-+-+-+-+-+             / / / _ _ _
                               (Ref: 10.4)
```

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

```
  Note: ROR instruction is available on MCS650X microprocessors after
        June, 1976.


RTI                    RTI Return from interrupt                     RTI
                                                     N Z C I D V
```

```
Operation:  P fromS PC fromS                           From Stack
                               (Ref: 9.6)
```

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	RTI	4D	1	6

```
RTS                    RTS Return from subroutine                    RTS
                                                      N Z C I D V
Operation:  PC fromS, PC + 1 -> PC                    _ _ _ _ _ _
                               (Ref: 8.2)
```

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	RTS	60	1	6

```
SBC          SBC Subtract memory from accumulator with borrow        SBC
                    -
Operation:  A - M - C -> A                            N Z C I D V
       -                                              / / / _ _ /
  Note:C = Borrow             (Ref: 2.2.2)
```

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
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 when page boundary is crossed.

```
SEC                         SEC Set carry flag                       SEC

Operation:  1 -> C                                    N Z C I D V
                                                      _ _ 1 _ _ _
                              (Ref: 3.0.1)
```

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	SEC	38	1	2

```
SED                        SED Set decimal mode                      SED
                                                      N Z C I D V
Operation:  1 -> D                                    _ _ _ _ 1 _
                              (Ref: 3.3.1)
```

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	SED	F8	1	2

```
SEI                   SEI Set interrupt disable status               SED
                                                      N Z C I D V
```

```
Operation:  1 -> I                                      _ _ _ 1 _ _
                              (Ref: 3.2.1)
```

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	SEI	78	1	2

```
STA                    STA Store accumulator in memory                 STA

Operation:  A -> M                                      N Z C I D V
                                                        _ _ _ _ _ _
                              (Ref: 2.1.2)
```

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

```
STX                      STX Store index X in memory                   STX

Operation: X -> M                                       N Z C I D V
                                                        _ _ _ _ _ _
                               (Ref: 7.2)
```

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Zero Page	STX Oper	86	2	3
Zero Page,Y	STX Oper,Y	96	2	4
Absolute	STX Oper	8E	3	4

```
STY                      STY Store index Y in memory                   STY

Operation: Y -> M                                       N Z C I D V
                                                        _ _ _ _ _ _
                               (Ref: 7.3)
```

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Zero Page	STY Oper	84	2	3
Zero Page,X	STY Oper,X	94	2	4
Absolute	STY Oper	8C	3	4

```
TAX                  TAX Transfer accumulator to index X               TAX

Operation:  A -> X                                      N Z C I D V
                                                        / / _ _ _ _
                               (Ref: 7.11)
```

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	TAX	AA	1	2

```
TAY                TAY Transfer accumulator to index Y               TAY

Operation:  A -> Y                                     N Z C I D V
                                                       / / _ _ _ _
                               (Ref: 7.13)
```

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	TAY	A8	1	2

```
TSX              TSX Transfer stack pointer to index X               TSX

Operation:  S -> X                                     N Z C I D V
                                                       / / _ _ _ _
                               (Ref: 8.9)
```

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	TSX	BA	1	2

```
TXA                TXA Transfer index X to accumulator               TXA
                                                       N Z C I D V
Operation:  X -> A                                     / / _ _ _ _
                               (Ref: 7.12)
```

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	TXA	8A	1	2

```
TXS              TXS Transfer index X to stack pointer               TXS
                                                       N Z C I D V
Operation:  X -> S                                     _ _ _ _ _ _
                               (Ref: 8.8)
```

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	TXS	9A	1	2

```
TYA                TYA Transfer index Y to accumulator               TYA

Operation:  Y -> A                                     N Z C I D V
                                                       / / _ _ _ _
                               (Ref: 7.14)
```

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	TYA	98	1	2

```
+------------------------------------------------------------------------
| INSTRUCTION ADDRESSING MODES AND RELATED EXECUTION TIMES
| (in clock cycles)
+------------------------------------------------------------------------

                A   A   A   B   B   B   B   B   B   B   B   B   B   C
                D   N   S   C   C   E   I   M   N   P   R   V   V   L
```

	C	D	L	C	S	Q	T	I	E	L	K	C	S	C
Accumulator	.	.	2	.	.	.	.	.	.	.	.	.	.	.
Immediate	2	2		.	.	.	.	.	.	.	.	.	.	.
Zero Page	3	3	5	.	.	.	3	.	.	.	.	.	.	.
Zero Page,X	4	4	6	.	.	.	.	.	.	.	.	.	.	.
Zero Page,Y	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Absolute	4	4	6	.	.	.	4	.	.	.	.	.	.	.
Absolute,X	4*	4*	7	.	.	.	.	.	.	.	.	.	.	.
Absolute,Y	4*	4*	.	.	.	.	.	.	.	.	.	.	.	.
Implied	.	.	.	.	.	.	.	.	.	.	.	.	.	2
Relative	.	.	.	2**	2**	2**	.	2**	2**	2**	7	2**	2**	.
(Indirect,X)	6	6	.	.	.	.	.	.	.	.	.	.	.	.
(Indirect),Y	5*	5*	.	.	.	.	.	.	.	.	.	.	.	.
Abs. Indirect	.	.	.	.	.	.	.	.	.	.	.	.	.	.

	C	C	C	C	C	C	D	D	D	E	I	I	I	J
	L	L	L	M	P	P	E	E	E	O	N	N	N	M
	D	I	V	P	X	Y	C	X	Y	R	C	X	Y	P
Accumulator	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Immediate	.	.	.	2	2	2	.	.	.	2	.	.	.	.
Zero Page	.	.	.	3	3	3	5	.	.	3	5	.	.	.
Zero Page,X	.	.	.	4	.	.	6	.	.	4	6	.	.	.
Zero Page,Y	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Absolute	.	.	.	4	4	4	6	.	.	4	6	.	.	3
Absolute,X	.	.	.	4*	.	.	7	.	.	4*	7	.	.	.
Absolute,Y	.	.	.	4*	.	.	.	.	.	4*	.	.	.	.
Implied	2	2	2	.	.	.	.	2	2	.	.	2	2	.
Relative	.	.	.	.	.	.	.	.	.	.	.	.	.	.
(Indirect,X)	.	.	.	6	.	.	.	.	.	6	.	.	.	.
(Indirect),Y	.	.	.	5*	.	.	.	.	.	5*	.	.	.	.
Abs. Indirect	.	.	.	.	.	.	.	.	.	.	.	.	.	5

* Add one cycle if indexing across page boundary
** Add one cycle if branch is taken, Add one additional if branching operation crosses page boundary

INSTRUCTION ADDRESSING MODES AND RELATED EXECUTION TIMES
(in clock cycles)

	J	L	L	L	L	N	O	P	P	P	P	R	R	R
	S	D	D	D	S	O	R	H	H	L	L	O	O	T
	R	A	X	Y	R	P	A	A	P	A	P	L	R	I
Accumulator	.	.	.	.	2	.	.	.	.	.	.	2	2	.
Immediate	.	2	2	2	.	.	2	.	.	.	.	.	.	.
Zero Page	.	3	3	3	5	.	3	.	.	.	.	5	5	.
Zero Page,X	.	4	.	4	6	.	4	.	.	.	.	6	6	.
Zero Page,Y	.	.	4	.	.	.	.	.	.	.	.	.	.	.
Absolute	6	4	4	4	6	.	4	.	.	.	.	6	6	.
Absolute,X	.	4*	.	4*	7	.	4*	.	.	.	.	7	7	.
Absolute,Y	.	4*	4*	.	.	.	4*	.	.	.	.	.	.	.
Implied	.	.	.	.	.	2	.	3	3	4	4	.	.	6
Relative	.	.	.	.	.	.	.	.	.	.	.	.	.	.
(Indirect,X)	.	6	.	.	.	.	6	.	.	.	.	.	.	.
(Indirect),Y	.	5*	.	.	.	.	5*	.	.	.	.	.	.	.
Abs. Indirect	.	.	.	.	.	.	.	.	.	.	.	.	.	.

	R	S	S	S	S	S	S	S	T	T	T	T	T	T
	T	B	E	E	E	T	T	T	A	A	S	X	X	Y
	S	C	C	D	I	A	X	Y	X	Y	X	A	S	A
Accumulator	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Immediate	.	2	.	.	.	.	.	.	.	.	.	.	.	.
Zero Page	.	3	.	.	.	3	3	3	.	.	.	.	.	.

Zero Page,X	.	4	.	.	.	4	.	4	.	.	.	.	.	.
Zero Page,Y	.	.	.	.	.	.	4	.	.	.	.	.	.	.
Absolute	.	4	.	.	.	4	4	4	.	.	.	.	.	.
Absolute,X	.	4*	.	.	.	5	.	.	.	.	.	.	.	.
Absolute,Y	.	4*	.	.	.	5	.	.	.	.	.	.	.	.
Implied	6	.	2	2	2	.	.	.	2	2	2	2	2	2
Relative	.	.	.	.	.	.	.	.	.	.	.	.	.	.
(Indirect,X)	.	6	.	.	.	6	.	.	.	.	.	.	.	.
(Indirect),Y	.	5*	.	.	.	6	.	.	.	.	.	.	.	.
Abs. Indirect	.	.	.	.	.	.	.	.	.	.	.	.	.	.

* Add one cycle if indexing across page boundary
** Add one cycle if branch is taken, Add one additional if branching operation crosses page boundary

```
00 - BRK                        20 - JSR
01 - ORA - (Indirect,X)         21 - AND - (Indirect,X)
02 - Future Expansion           22 - Future Expansion
03 - Future Expansion           23 - Future Expansion
04 - Future Expansion           24 - BIT - Zero Page
05 - ORA - Zero Page            25 - AND - Zero Page
06 - ASL - Zero Page            26 - ROL - Zero Page
07 - Future Expansion           27 - Future Expansion
08 - PHP                        28 - PLP
09 - ORA - Immediate            29 - AND - Immediate
0A - ASL - Accumulator          2A - ROL - Accumulator
0B - Future Expansion           2B - Future Expansion
0C - Future Expansion           2C - BIT - Absolute
0D - ORA - Absolute             2D - AND - Absolute
0E - ASL - Absolute             2E - ROL - Absolute
0F - Future Expansion           2F - Future Expansion
10 - BPL                        30 - BMI
11 - ORA - (Indirect),Y         31 - AND - (Indirect),Y
12 - Future Expansion           32 - Future Expansion
13 - Future Expansion           33 - Future Expansion
14 - Future Expansion           34 - Future Expansion
15 - ORA - Zero Page,X          35 - AND - Zero Page,X
16 - ASL - Zero Page,X          36 - ROL - Zero Page,X
17 - Future Expansion           37 - Future Expansion
18 - CLC                        38 - SEC
19 - ORA - Absolute,Y           39 - AND - Absolute,Y
1A - Future Expansion           3A - Future Expansion
1B - Future Expansion           3B - Future Expansion
1C - Future Expansion           3C - Future Expansion
1D - ORA - Absolute,X           3D - AND - Absolute,X
1E - ASL - Absolute,X           3E - ROL - Absolute,X
1F - Future Expansion           3F - Future Expansion

40 - RTI                        60 - RTS
41 - EOR - (Indirect,X)         61 - ADC - (Indirect,X)
42 - Future Expansion           62 - Future Expansion
43 - Future Expansion           63 - Future Expansion
44 - Future Expansion           64 - Future Expansion
45 - EOR - Zero Page            65 - ADC - Zero Page
46 - LSR - Zero Page            66 - ROR - Zero Page
47 - Future Expansion           67 - Future Expansion
48 - PHA                        68 - PLA
49 - EOR - Immediate            69 - ADC - Immediate
4A - LSR - Accumulator          6A - ROR - Accumulator
4B - Future Expansion           6B - Future Expansion
4C - JMP - Absolute             6C - JMP - Indirect
4D - EOR - Absolute             6D - ADC - Absolute
4E - LSR - Absolute             6E - ROR - Absolute
```

```
4F - Future Expansion          6F - Future Expansion
50 - BVC                       70 - BVS
51 - EOR - (Indirect),Y        71 - ADC - (Indirect),Y
52 - Future Expansion          72 - Future Expansion
53 - Future Expansion          73 - Future Expansion
54 - Future Expansion          74 - Future Expansion
55 - EOR - Zero Page,X         75 - ADC - Zero Page,X
56 - LSR - Zero Page,X         76 - ROR - Zero Page,X
57 - Future Expansion          77 - Future Expansion
58 - CLI                       78 - SEI
59 - EOR - Absolute,Y          79 - ADC - Absolute,Y
5A - Future Expansion          7A - Future Expansion
5B - Future Expansion          7B - Future Expansion
5C - Future Expansion          7C - Future Expansion
5D - EOR - Absolute,X          7D - ADC - Absolute,X
5E - LSR - Absolute,X          7E - ROR - Absolute,X
5F - Future Expansion          7F - Future Expansion

80 - Future Expansion          A0 - LDY - Immediate
81 - STA - (Indirect,X)        A1 - LDA - (Indirect,X)
82 - Future Expansion          A2 - LDX - Immediate
83 - Future Expansion          A3 - Future Expansion
84 - STY - Zero Page           A4 - LDY - Zero Page
85 - STA - Zero Page           A5 - LDA - Zero Page
86 - STX - Zero Page           A6 - LDX - Zero Page
87 - Future Expansion          A7 - Future Expansion
88 - DEY                       A8 - TAY
89 - Future Expansion          A9 - LDA - Immediate
8A - TXA                       AA - TAX
8B - Future Expansion          AB - Future Expansion
8C - STY - Absolute            AC - LDY - Absolute
8D - STA - Absolute            AD - LDA - Absolute
8E - STX - Absolute            AE - LDX - Absolute
8F - Future Expansion          AF - Future Expansion
90 - BCC                       B0 - BCS
91 - STA - (Indirect),Y        B1 - LDA - (Indirect),Y
92 - Future Expansion          B2 - Future Expansion
93 - Future Expansion          B3 - Future Expansion
94 - STY - Zero Page,X         B4 - LDY - Zero Page,X
95 - STA - Zero Page,X         B5 - LDA - Zero Page,X
96 - STX - Zero Page,Y         B6 - LDX - Zero Page,Y
97 - Future Expansion          B7 - Future Expansion
98 - TYA                       B8 - CLV
99 - STA - Absolute,Y          B9 - LDA - Absolute,Y
9A - TXS                       BA - TSX
9B - Future Expansion          BB - Future Expansion
9C - Future Expansion          BC - LDY - Absolute,X
9D - STA - Absolute,X          BD - LDA - Absolute,X
9E - Future Expansion          BE - LDX - Absolute,Y
9F - Future Expansion          BF - Future Expansion

C0 - Cpy - Immediate           E0 - CPX - Immediate
C1 - CMP - (Indirect,X)        E1 - SBC - (Indirect,X)
C2 - Future Expansion          E2 - Future Expansion
C3 - Future Expansion          E3 - Future Expansion
C4 - CPY - Zero Page           E4 - CPX - Zero Page
C5 - CMP - Zero Page           E5 - SBC - Zero Page
C6 - DEC - Zero Page           E6 - INC - Zero Page
C7 - Future Expansion          E7 - Future Expansion
C8 - INY                       E8 - INX
C9 - CMP - Immediate           E9 - SBC - Immediate
CA - DEX                       EA - NOP
CB - Future Expansion          EB - Future Expansion
CC - CPY - Absolute            EC - CPX - Absolute
CD - CMP - Absolute            ED - SBC - Absolute
```

```
        CE - DEC - Absolute             EE - INC - Absolute
        CF - Future Expansion           EF - Future Expansion
        D0 - BNE                        F0 - BEQ
        D1 - CMP   (Indirect@,Y         F1 - SBC - (Indirect),Y
        D2 - Future Expansion           F2 - Future Expansion
        D3 - Future Expansion           F3 - Future Expansion
        D4 - Future Expansion           F4 - Future Expansion
        D5 - CMP - Zero Page,X          F5 - SBC - Zero Page,X
        D6 - DEC - Zero Page,X          F6 - INC - Zero Page,X
        D7 - Future Expansion           F7 - Future Expansion
        D8 - CLD                        F8 - SED
        D9 - CMP - Absolute,Y           F9 - SBC - Absolute,Y
        DA - Future Expansion           FA - Future Expansion
        DB - Future Expansion           FB - Future Expansion
        DC - Future Expansion           FC - Future Expansion
        DD - CMP - Absolute,X           FD - SBC - Absolute,X
        DE - DEC - Absolute,X           FE - INC - Absolute,X
        DF - Future Expansion           FF - Future Expansion
```

INSTRUCTION OPERATION

The following code has been taken from VICE for the purposes of showing how each instruction operates. No particular addressing mode is used since we only wish to see the operation of the instruction itself.

```
     src : the byte of data that is being addressed.
     SET_SIGN : sets\resets the sign flag depending on bit 7.
     SET_ZERO : sets\resets the zero flag depending on whether the result
                is zero or not.
     SET_CARRY(condition) : if the condition has a non-zero value then the
                carry flag is set, else it is reset.
     SET_OVERFLOW(condition) : if the condition is true then the overflow
                flag is set, else it is reset.
     SET_INTERRUPT :  }
     SET_BREAK :      }  As for SET_CARRY and SET_OVERFLOW.
     SET_DECIMAL :    }
     REL_ADDR(PC, src) : returns the relative address obtained by adding
                the displacement src to the PC.
     SET_SR : set the Program Status Register to the value given.
     GET_SR : get the value of the Program Status Register.
     PULL : Pull a byte off the stack.
     PUSH : Push a byte onto the stack.
     LOAD : Get a byte from the memory address.
     STORE : Store a byte in a memory address.
     IF_CARRY, IF_OVERFLOW, IF_SIGN, IF_ZERO etc : Returns true if the
                relevant flag is set, otherwise returns false.
     clk : the number of cycles an instruction takes. This is shown below
                in situations where the number of cycles changes depending
                on the result of the instruction (eg. Branching instructions).

     AC = Accumulator
     XR = X register
     YR = Y register
     PC = Program Counter
     SP = Stack Pointer
```

```
/* ADC */
    unsigned int temp = src + AC + (IF_CARRY() ? 1 : 0);
    SET_ZERO(temp & 0xff);      /* This is not valid in decimal mode */
    if (IF_DECIMAL()) {
        if (((AC & 0xf) + (src & 0xf) + (IF_CARRY() ? 1 : 0)) > 9) temp += 6;
        SET_SIGN(temp);
        SET_OVERFLOW(!((AC ^ src) & 0x80) && ((AC ^ temp) & 0x80));
        if (temp > 0x99) temp += 96;
        SET_CARRY(temp > 0x99);
    } else {
        SET_SIGN(temp);
        SET_OVERFLOW(!((AC ^ src) & 0x80) && ((AC ^ temp) & 0x80));
        SET_CARRY(temp > 0xff);
    }
    AC = ((BYTE) temp);

/* AND */
    src &= AC;
    SET_SIGN(src);
    SET_ZERO(src);
    AC = src;

/* ASL */
    SET_CARRY(src & 0x80);
    src <<= 1;
    src &= 0xff;
    SET_SIGN(src);
    SET_ZERO(src);
    STORE src in memory or accumulator depending on addressing mode.

/* BCC */
    if (!IF_CARRY()) {
        clk += ((PC & 0xFF00) != (REL_ADDR(PC, src) & 0xFF00) ? 2 : 1);
        PC = REL_ADDR(PC, src);
    }

/* BCS */
    if (IF_CARRY()) {
        clk += ((PC & 0xFF00) != (REL_ADDR(PC, src) & 0xFF00) ? 2 : 1);
        PC = REL_ADDR(PC, src);
    }

/* BEQ */
    if (IF_ZERO()) {
        clk += ((PC & 0xFF00) != (REL_ADDR(PC, src) & 0xFF00) ? 2 : 1);
        PC = REL_ADDR(PC, src);
    }

/* BIT */
    SET_SIGN(src);
    SET_OVERFLOW(0x40 & src);   /* Copy bit 6 to OVERFLOW flag. */
    SET_ZERO(src & AC);

/* BMI */
    if (IF_SIGN()) {
        clk += ((PC & 0xFF00) != (REL_ADDR(PC, src) & 0xFF00) ? 2 : 1);
        PC = REL_ADDR(PC, src);
    }

/* BNE */
    if (!IF_ZERO()) {
        clk += ((PC & 0xFF00) != (REL_ADDR(PC, src) & 0xFF00) ? 2 : 1);
        PC = REL_ADDR(PC, src);
    }

/* BPL */
    if (!IF_SIGN()) {
        clk += ((PC & 0xFF00) != (REL_ADDR(PC, src) & 0xFF00) ? 2 : 1);
        PC = REL_ADDR(PC, src);
    }

/* BRK */
```

```
    PC++;
    PUSH((PC >> 8) & 0xff);     /* Push return address onto the stack. */
    PUSH(PC & 0xff);
    SET_BREAK((1));             /* Set BFlag before pushing */
    PUSH(SR);
    SET_INTERRUPT((1));
    PC = (LOAD(0xFFFE) | (LOAD(0xFFFF) << 8));

/* BVC */
    if (!IF_OVERFLOW()) {
        clk += ((PC & 0xFF00) != (REL_ADDR(PC, src) & 0xFF00) ? 2 : 1);
        PC = REL_ADDR(PC, src);
    }

/* BVS */
    if (IF_OVERFLOW()) {
        clk += ((PC & 0xFF00) != (REL_ADDR(PC, src) & 0xFF00) ? 2 : 1);
        PC = REL_ADDR(PC, src);
    }

/* CLC */
    SET_CARRY((0));

/* CLD */
    SET_DECIMAL((0));

/* CLI */
    SET_INTERRUPT((0));

/* CLV */
    SET_OVERFLOW((0));

/* CMP */
    src = AC - src;
    SET_CARRY(src < 0x100);
    SET_SIGN(src);
    SET_ZERO(src &= 0xff);

/* CPX */
    src = XR - src;
    SET_CARRY(src < 0x100);
    SET_SIGN(src);
    SET_ZERO(src &= 0xff);

/* CPY */
    src = YR - src;
    SET_CARRY(src < 0x100);
    SET_SIGN(src);
    SET_ZERO(src &= 0xff);

/* DEC */
    src = (src - 1) & 0xff;
    SET_SIGN(src);
    SET_ZERO(src);
    STORE(address, (src));

/* DEX */
    unsigned src = XR;
    src = (src - 1) & 0xff;
    SET_SIGN(src);
    SET_ZERO(src);
    XR = (src);

/* DEY */
    unsigned src = YR;
    src = (src - 1) & 0xff;
    SET_SIGN(src);
    SET_ZERO(src);
    YR = (src);

/* EOR */
    src ^= AC;
    SET_SIGN(src);
    SET_ZERO(src);
    AC = src;

/* INC */
    src = (src + 1) & 0xff;
    SET_SIGN(src);
    SET_ZERO(src);
    STORE(address, (src));

/* INX */
    unsigned src = XR;
    src = (src + 1) & 0xff;
    SET_SIGN(src);
    SET_ZERO(src);
    XR = (src);

/* INY */
    unsigned src = YR;
    src = (src + 1) & 0xff;
    SET_SIGN(src);
    SET_ZERO(src);
    YR = (src);

/* JMP */
    PC = (src);

/* JSR */
    PC--;
    PUSH((PC >> 8) & 0xff);     /* Push return address onto the stack. */
    PUSH(PC & 0xff);
    PC = (src);

/* LDA */
    SET_SIGN(src);
    SET_ZERO(src);
    AC = (src);

/* LDX */
    SET_SIGN(src);
    SET_ZERO(src);
    XR = (src);

/* LDY */
    SET_SIGN(src);
    SET_ZERO(src);
    YR = (src);

/* LSR */
    SET_CARRY(src & 0x01);
    src >>= 1;
    SET_SIGN(src);
    SET_ZERO(src);
    STORE src in memory or accumulator depending on addressing mode.

/* NOP */
    Nothing.
```

```
/* ORA */
    src |= AC;
    SET_SIGN(src);
    SET_ZERO(src);
    AC = src;

/* PHA */
    src = AC;
    PUSH(src);

/* PHP */
    src = GET_SR;
    PUSH(src);

/* PLA */
    src = PULL();
    SET_SIGN(src);      /* Change sign and zero flag accordingly. */
    SET_ZERO(src);

/* PLP */
    src = PULL();
    SET_SR((src));

/* ROL */
    src <<= 1;
    if (IF_CARRY()) src |= 0x1;
    SET_CARRY(src > 0xff);
    src &= 0xff;
    SET_SIGN(src);
    SET_ZERO(src);
    STORE src in memory or accumulator depending on addressing mode.

/* ROR */
    if (IF_CARRY()) src |= 0x100;
    SET_CARRY(src & 0x01);
    src >>= 1;
    SET_SIGN(src);
    SET_ZERO(src);
    STORE src in memory or accumulator depending on addressing mode.

/* RTI */
    src = PULL();
    SET_SR(src);
    src = PULL();
    src |= (PULL() << 8);       /* Load return address from stack. */
    PC = (src);

/* RTS */
    src = PULL();
    src += ((PULL()) << 8) + 1; /* Load return address from stack and add 1. */
    PC = (src);

/* SBC */
    unsigned int temp = AC - src - (IF_CARRY() ? 0 : 1);
    SET_SIGN(temp);
    SET_ZERO(temp & 0xff);      /* Sign and Zero are invalid in decimal mode */
    SET_OVERFLOW(((AC ^ temp) & 0x80) && ((AC ^ src) & 0x80));
    if (IF_DECIMAL()) {
        if ( ((AC & 0xf) - (IF_CARRY() ? 0 : 1)) < (src & 0xf)) /* EP */ temp -= 6;
        if (temp > 0x99) temp -= 0x60;
    }
    SET_CARRY(temp < 0x100);
    AC = (temp & 0xff);

/* SEC */
```

```
    SET_CARRY((1));

/* SED */
    SET_DECIMAL((1));

/* SEI */
    SET_INTERRUPT((1));

/* STA */
    STORE(address, (src));

/* STX */
    STORE(address, (src));

/* STY */
    STORE(address, (src));

/* TAX */
    unsigned src = AC;
    SET_SIGN(src);
    SET_ZERO(src);
    XR = (src);

/* TAY */
    unsigned src = AC;
    SET_SIGN(src);
    SET_ZERO(src);
    YR = (src);

/* TSX */
    unsigned src = SP;
    SET_SIGN(src);
    SET_ZERO(src);
    XR = (src);

/* TXA */
    unsigned src = XR;
    SET_SIGN(src);
    SET_ZERO(src);
    AC = (src);

/* TXS */
    unsigned src = XR;
    SP = (src);

/* TYA */
    unsigned src = YR;
    SET_SIGN(src);
    SET_ZERO(src);
    AC = (src);
```