==========
From *Microprocessor programming and applications for scientists and engineers:*
by Richard R. Smardzewski
==========

CHAPTER 4

The 6502 MPU
---

* Architecture
* Execution
* Mnemonics
* Registers
* Paging
* Instructions
* Addressing Modes

4.1 ARCHITECTURE

The architecture of a microcomputer is depicted schematically in Fig. 4-1 for a "typical" 6502 system. The microprocessor (MPU) communicates with external devices, chips via three busses. The data bus is 8-bits wide and carries data (bytes) from the MPU to external devices and vice-versa. It is bidirectional in nature. The address bus is 16-bits wide and carries addresses generated in the MPU to the various external devices. These 16-bit addresses specify where (in memory) the data will go to or come from (on the data bus). A control bus carries the various synchronization signals such as R/W (i.e. is this a read or write operation?). ROM is Read Only Memory and contains such information as the operating program or Monitor for the system. It is non-volatile and remains on when the power is removed. RAM (Random Access Memory) is the Read/Write Memory which, as its name implies, is really a scratchpad (for bytes) which can be written into, written over or read from. It is volatile in nature and its contents are lost upon removal of power. Finally, the Input/Output (I/O) devices are those chips which enable the MPU to communicate with the outside world (i.e. keyboards, CRT's, switches, voltages, currents ....). They are also connected to the MPU via the three busses. Other additional chips such as buffers, decoders, drivers, etc. are necessary to construct a real system. For more information on the design and interfacing of an actual system, the reader is referred to the many available hardware texts on the subject.

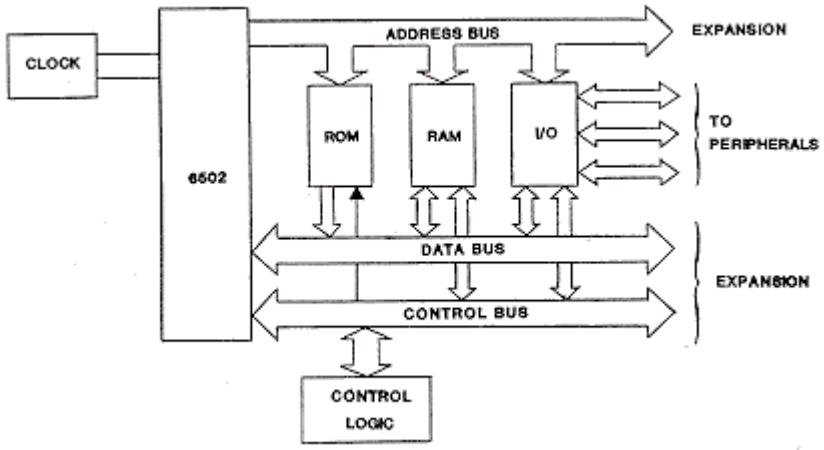


Fig. 4-1. A "Typical" 6502 System.

4.2 EXECUTION

The 6502 MPU executes programs by fetching an instruction from memory, executing it, and then fetching the next instruction. Instructions are 1-3 bytes long with the first byte representing the Operation Code (op-code) of the instruction, usually expressed in hexadecimal notation. For example, $AA = TAX = Transfer Accumulator to X Index. The remaining 1-2 bytes following the instruction can represent either data or an address. The 6502 distinguishes instruction bytes from address bytes or data bytes by noting their relative sequence in a program. In a typical 6502 MPU operating at 1MHz, instruction times vary between 2-7 microseconds depending upon the particular type of instruction.

4.3 MNEMONICS

Mnemonics represent a shorthand notation or Assembly Language which describes the instructions of the MPU in a more understandable way (to humans) than hexadecimal op-codes or 8-bit groups of 0's and 1's. It is much easier to remember that the TAX instruction means Transfer Accumulator to X Index than the hex op-code $AA or the 8-bit number 1010 1010. Microprocessors, on the other hand, operate only on binary voltages represented by logical 0's and 1's (Machine Language). Consequently, Mnemonics must be translated into such 0's and 1's before the MPU can proceed. This is accomplished by a program called an Assembler which is usually (but not always) available in most microcomputing systems. For instruction purposes, the Mnemonic format will be used to illustrate sample programs throughout the text. Keep in mind however, that this is only for the sake of human understanding.

4.4 REGISTERS

The 6502 microprocessor uses seven Registers (6 internal, 1 Temporary) in data manipulation/handling.

PC Register

The Program Counter (PC) Register is a 16-bit wide register which determines which memory location will be accessed next. It is automatically incremented after each memory access and contains the address of the next instruction to be executed. Being 16-bits wide, it can address any one of 2^{16} = 65,536 locations. It is physically implemented in the 6502 as two 8-bit registers, PCL and PCH. L and H respectively refer to the lower (L) and higher (H) order bytes of the Program Counter. If the PC Register is loaded with a specific 16-bit number (2 bytes) during the execution of a program, control will be transferred to that particular address in memory and execution will resume from that point forward.

A, X, Y Registers

The Accumulator (A), X-Register (X) and Y-Register (Y) are three 8-bit wide general purpose registers which are used for a variety of operations. Arithmetic/Logical operations are performed in the Accumulator while the X and Y registers are employed as index registers or counters and are capable of being incremented/decremented in a controlled fashion.

P Register - N, V, B, D, I, Z, C

The Processor Status (P) Register contains seven usable bits called Status Flags. Five of these are affected by the outcome of a preceding instruction. During program execution these particular bits are constantly changing to reflect the status of the previous operation. The two remaining bits are control bits and can be set (=1) or cleared (=0) under program control.

N is the Negative Flag which is set (=1) whenever the result of the previous operation produces a negative value. Otherwise it is cleared (=0).

V is the Overflow Flag which is set (=1) if the result of a signed arithmetic opeartion produces a result greater than +127 or less that -128.

B it the Break Command Flag and is set (=1) by a "break" instruction.

D is the Decimal Mode bit which when set (=1) enables the 6502 to treat data as binary-coded-decimal (BCD) numbers.

I is the IRQ Disable bit which when set (=1) prevents the 6502 from being externally interrupted (by the IRQ line).

Z is the Zero Flag which is set (=1) if the result of the preceding operation is zero.

C is the Carry Flag and is used extensively during arithmetic operations. It can be viewed as a ninth bit which is set (=1) whenever the addition of two 8-bit number produces a result which exceeds 8-bits.

Stack

The Stack is an 8-bit wide temporary working register which is primarily used for transitory storage of program variables. It is implemented in the 256 locations at addressed $0100-01FF of the 6502's address space and characterized as a push-down top memory location ($01FF) downwards in memory and unloaded in reverse fashion. It can be thought of as a first-in last-out (FILO) memory.

Stack Pointer

The Stack Pointer (S) is an 8-bit wide internal register which is an abbreviated address register for the Stack. It represents the low order address byte of the next available location in the Stack. As such, it points to the next empty Stack location.

4.5 PAGING

The concept of paging is introduced as a helpful way of organizing memory in a microcomputer. The 6502 microprocessor is a capable of accessing 2^{16} = 65,536 or 64K (1K = 1024) locations in memory. In hexadecimal notation this incluedes all 16-bit addresses from $0000 to $FFFF. If we define a Page as a block of 256 memory locations, the 64K location can be represented as 256 pages, each containing 256 locations. The higher order bits (nos. 8-15) of the 16-bit address represnet the Page Number while the lower order bits (nos. 0-7) represent the locations within a Page. For example, Page 3 represents all addresses in the range $0300-$03FF, while Page FF contains the address $FF00-$FFFF. Using this notation, Page 1 is normally reserved for the Stack area.

4.6 INSTRUCTIONS AND ADDRESSING MODES

The 6502 Microprocessor has 56 different instructions and can operate in 13 addressing modes. The various allowed combinations of instruction types and addressing modes permit a total of 151 different executable instructions. Each of these instructions is examined by way of discussion and example in the following pages. Mnemonic and hexadecimal ($) notations are used throughout the programming examples unless otherwise stated. The reader is encouraged to work through each programming example since each demonstrates the application of a particular instruction and/or addressing mode.

6502 INSTRUCTION SET

Operation	Mnemonic
Load and Store Instructions	LDA,STA,LDX,LDY,STX,STY
Register Transfer Instructions	TAX,TAY,TXA,TYA
Break Instruction	BRK
No Operation Instruction	NOP
Jump Operation Instruction	JMP
Increment/Decrement Instructions	INC,INX,INY,DEC,DEX,DEY
Logical Instructions	AND,ORA,EOR
Flag Instructions	CLC,SEC,CLD,SED,CLV
Arithmetic Instructions	ADC,SBC
Branch Instructions	BCC,BCS,BEQ,BNE,BMI,BPL BVC,BVS
Compare Instructions	CMP,CPX,CPY
Bit Test Instruction	BIT

Shift and Rotate Instructions	ASL,LSR,ROL,ROR
Subroutine Insturctions	JSR,RTS
Stack Instructions	PHA,PLA,PHP,PLP,TXS,TSX
Interrupt Instructions	RTI,SEI,CLI

Fig. 4-2. The 56 Instructions of the 6502 MPU.

6502 ADDRESSING MODES

Absolute
Zero Page
Immediate
Implied
Relative
Accumulator
Absolute Indexed, X
Absolute Indexed, Y
Zero Page, X
Zero Page, Y
Indirect Absolute
Indirect Indexed (Post-Indexed Indirect)
Indexed Indirect (Pre-Indexed Indirect)

Fig 4-3. The 13 Addressing Modes of the 6502 MPU.

MONITOR COMMANDS

In every microcomputing system or programming langauge there exists a prescribed set of functional command which are input to the computer from a particular device (keyboard, tape, disk, ...). Such commands are specific to that computer's operating system (Monitor Program) or programming language. For example, in the BASIC language (a type of program), the RUN command, when entered, begins program execution from the lowest numbered statement.

In the AIN 65 Microcomputing System, those Monitor Command which are most useful and necessary are the <M>, </>, <*>, <G>, <K> and <R> Commands. They are entered from the keyboard and perform the following functions:

<M> = Displays the contents (in Hex) of the various memory locations in groups of four.
</> = Changes the contents of the various memory locations to the entered Hex values.
<*> = Sets/Re-Sets the Program Counter (PC) to the entered Hex value.
<G> = Begins program execution from that address currently contained the Program Counter (PC).
<K> = Disassembles (i.e. converts machine --> assembly language) the contents (valid instruction op codes) of specified memory location for display/printout)
<R> = Displays the contents (in Hex) of the six internal registers of the 6502.
**** = Program Counter (PC)
PS = Processor Status Register (P)
AA = Accumulator (A)
XX = X-Register (X)
YY = Y-Register (Y)
SS = Stack Pointer (S)

These Monitor Commands will be used from this point foward to enter, alter and display machine/assembly language programs in the instructional

experiments that follow. With few exceptions, these machine-language programs can be used with any 6502-based system with the single provison that they reside in or access those user-available portions of the system's RAM

LDA - Load Accumulator with Memory
STA - Store Accumulator in Memory

The LDA and STA instructions can operate in three non-indexed addressing modes. They are the Absolute, Zero Page, and Immediate Addressing Modes.

ABSOLUTE ADDRESSING

In the Absolute Addressing Mode, the second and third bytes of the instruction are the address of the location where the data is.

Exp.1 - Absolute Addressing

1. Store $AA in location $0211 using the Monitor <M> and </> Commands. Use the same commands to store $FF in location $0300.
2. Key in the following program in Hex format beginning at location $0100.

	Comments
$0100 AD LDA@	Load the Accumulator with the
1 11	contents of $0211
2 02	
3 8D STA@	Store the Accumulator at
4 00	location $0300
5 03	
6 00 BRK	Return to Monitor
$0107 EA NOP	No Operation

3. After keying in the program, examine the contents of location $0300.
4. Next, run the program by keying in <*> = 0100 followed by RETURN then <G> / and RETURN. The display should read <0107 EA NOP.
5. Examine the contents of location $0300. Has it changed? Answer: Yes, $FF is $AA.
6. AIM Printout:

```
<M>=0211 AA 3D 40 35
<M>=0300 FF 41 2C C3
<*>=0100
<G>/
 0107 EA NOP

<M>=0300 AA 41 2C C3

<K>*=0100
/04
 0100 AD LDA 0211
 0103 BD STA 0300
 0106 00 BRK
 0107 EA NOP
```

ZERO PAGE ADDRESSING

In the Zero Page addressing mode the second byte of the instruction is the low order address of the Zero Page memory location containing the data.

Exp.2 - Zero Page Addressing

1. Store $EE in location $0020 using the Monitor <M> and </> commands.
2. Store $FF in locations $0030 and 0350.
3. Key in the following program:

	Comments
$0100 A5 LDA	Load the accumulator with the
1 20	contents of $0020
2 8D STA@	Store the accumulator at
3 50	location $0350
4 03	
5 AD LDA@	Load the accumulator with the
6 50	contents of $0350
7 03	
8 85 STA_Z	Store the accumulator at
9 30	location $0030
A 00 BRK	Return to Monitor
$010B EA NOP	No Operation

4. After keying in the program, examine the contents of locations $0030 and 0350.
5. Next, run the program by keying in <*> = 0100, RETURN, <G>/, RETURN. The display should read <010B EA NOP.
6. What are the current contents of locations $0030 and $0350? Answer: $EE, EE
7. AIM Printout:

Zero Page Addressing

```
<M>=0020 45 9D F4 21
</> 0020 EE 00 00 00
<M>=0030 87 E9 B1 45
</> 0030 FF 00 00 00
<M>=0350 07 39 0D B5
</> 0350 FF 00 00 00

<M>=0020 EE 00 00 00
<M>=0030 FF 00 00 00
<M>=0350 FF 00 00 00

<*>=0100
<G>/
 010B EA NOP

<M>=0030 EE 00 00 00
<M>=0350 EE 00 00 00

<K>*=0100
/06
 0100 A5 LDA 20
 0102 8D STA 0350
 0105 AD LDA 0350
 0108 85 STA 30
 010A 00 BRK
 010B EA NOP
```

IMMEDIATE ADDRESSING

In the Immediate Addressing Mode the second byte of the instruction is the data.

Exp. 3 - Immediate Addressing

1. Store $EE in location $0021 and $FF in locations $0030 and $0350.
2. Key in the following program:

	Comments
$0100 A9 LDA#	Load the accumulator with $BC
1 BC	
2 85 STA_z	Store the contents of the
3 20	accumulator at location $0020
4 8D STA@	Store also at location $0350
5 50	
6 03	
7 85 STA_z	And at $0030
8 30	
9 00 BRK	Return to Monitor
$010A EA NOP	No Operation

3. Run the program
4. What is now contained in locations $0020, 0030 and 0350? Answer: $BC
5. What is contained in the accumulator? Answer: $BC
6. Can you use the STA instruction in the Immediate Mode? Answer: NO. You can't store Data into Data only into Memory locations.
7. AIM Printout.

Immediate Addressing

```
<M>=0020 37 CC 36 C2
</> 0020 EE 00 00 00
<M>=0030 D8 C7 F9 C3
</> 0030 FF 00 00 00
<M>=0350 34 E8 3D CC
</> 0350 FF 00 00 00

<*>=0100
<G>/
 0100 EA NOP

<M>=0020 BC 00 00 00
<M>=0030 BC 00 00 00
<M>=0350 BC 00 00 00

<K>*=100
/06
 0100 A9 LDA #BC
 0102 85 STA 20
 0104 8D STA 0350
 0107 85 STA 30
 0109 00 BRK
 010A EA NOP
```

LDX - Load X Register with Memory
LDY - Load Y Register with Memory
STX - Store X Register in Memory
STY - Store Y Register in Memory

As with the LDA and STA instructions, the LDX, LDY, STX, and STY instructions can also be used inn the Absolute, Zero Page and (for LDX, LDY) the Immediate Addressing Mode.

Exp. 4 - LDX, LDY, STX, STY Operations

1. Store $44 in locations $0066, 0067 and 0068
2. Key in the following program:

	Comments
$0100 A0 LDY#	Load the Y Register with $55
1 55	
2 A6 LDX_z	Load the X Register with the
3 66	contents of $0066
4 84 STY_z	Store the Y Register with at Zero
5 67	Page Location $0067
6 86 STX_z	Store the X Register at Zero
7 68	Page Location $0068
8 AC LDY@	Load the Y Register with the
9 10	contents of $0110
A 01	
B 8C STY@	Store the Y Register at Zero
C 66	Page Location $0066
D 00	
E 00 BRK	Return to Monitor
F EA NOP	No Operation
$0110 66	

3. Run the program
4. What are the contents of locations $0066, 0067, and 0068? Answer: $66, 55, 44
5. What is unusual about the instuction at location $010B?
6. Do not alter the program or the contents of $0066, 0067, or 0068. They are used in Exp. 5.
7. AIM Printout.

LDX, LDY, STX, STY Operations

```
<M>=0066 33 C8 1A C7
</> 0066 44 44 44 00

<*>=0100
<G>/
 010F EA NOP

<M>=0066 66 55 44 00

<K>*=0100
/09
 0100 A0 LDY #55
 0102 A6 LDX 66
 0104 84 STY 67
```

```
0106 86 STX 68
0108 AC LDY 0110
010B 8C STY 0066
010E 00 BRK
010F EA NOP
0110 66 ROR 00
```

TAX - Transfer Contents of Accumulator to X Register
TAY - Transfer Contents of Accumulator to Y Register
TXA - Transfer Contents of X Register to Accumulator
TYA - Transfer Contents of Y Register to Accumulator

The Transfer instructions (TAX, TAY, TXA, TYA) are single byte insturctions which use a special mode of addressing called Implied Addressing. Data is transferred to or from the accumulator and the X or Y Registers. No R/W Memory locations are accessed.

Exp. 5 - TAX, TAY, TXA, TYA Operations

1. Modify the program in Exp. 4 by keying in the following instructions beginning at location $0108.

	Comments
$0108 98 TYA	Transfer contents of Y to A
9 AA TXA	Transfer contents of X to A
A 86 STA	Store contents of X at
B 66	location $0066
C 00 BRK	Return to Monitor
$010D EA NOP	No Operation

2. Run the program starting at $0100.
3. What are the contents of $0066, 0067, and 0068? Answer: $55, 55, 66
4. Run the program again starting at $0100. What are the contents of $0066, 0067, and 0068? Answer: $55, 55, 55
5. Explanation:

$0100 A0 LDY#	Load the Y Register with $55
1 55	
2 A6 LDX_z	Load the X Register with the
3 66	contents of $0066[$66]
4 84 STY_z	Store the contents of Y [$55] at
5 67	$0067 --> $0067[$55]
6 86 STX_z	Store the contents of X [$66] at
7 68	$0068 --> $0068[$66]
8 98 TYA	Transfer contents of Y[$55] to A
9 AA TXA	Transfer contents of X [$55] to A
A 86 STA	Store contents of X at
B 66	$0066 --> $0066[$55]
C 00 BRK	Return to Monitor
$010D EA NOP	No Operation

6. AIM Printout.

TAX, TAY, TXA, TYA Operations

```
<M>=0066 66 55 44 00
```

```
<*>=0100
<G>/
 010D EA NOP

<M>=0066 55 55 66 00

<*>=0100
<G>/
 010D EA NOP

<M>=0066 55 55 55 00

<K>*=0100
/09
 0100 A0 LDY #55
 0102 A6 LDX 66
 0104 84 STY 67
 0106 86 STX 68
 0108 98 TYA
 0109 AA TAX
 010A 86 STX 66
 010C 00 BRK
 010D EA NOP
```

BRK - Jump to Interrupt Routine

The BRK instruction forces a halt to the program execution sequence and returns control to an Interrupt Routine whose starting address is contained in two consecutive memory locations designated IRQL and IRQH. Upon power-up or hitting thr RESET button, IRQL and IRQH are loaded with the starting address of the Monitor Program (or Master Control Program). All or the important registers in the 6502 are saved during exection of the a BRK instruction. This makes it a very useful tool in program debugging since specific sections of a large program, separated by BK instructions, can by run and sequentially stopped for Register (A, X, Y, S, P, PC) examination and checking. The specific interrupt sequences of the BRK instruction (which is really a software-forced interrupt) are discussed in greater detail in that Section dealing with Interrupts.

NOP - No Operation

The NOP instruction is really a one-byte pseudo-instruction which does nothing but occupy time and memory. As such, it is especially useful in developing programs since a group of NOP's can be used to reserve program space for later addition of instructions. It can also be used in those routines involving the generations of programmed time delays.

JMP - Jump to New Location

When a JMP instruction is encountered during program execution, program control is transferred to that location specified by the JMP instruction and the program sequence continues from that point on. It, in effect, puts a new 2-byte value in the Program Counter (PCH-PCL) where PCH = higher order byte of PC and PCL = lower order byte of PC. The JMP instruction is a 3-byte long instruction with 2 Addressing Modes: Absolute and Indirect Absolute.

In the Absolute Mode the two bytes following the instruction are the low and high order bytes of the address in memory (PCL, PCH) where program control is transferred.

In the Indirect Absolute Mode, the two bytes following the instruction are the lower and higher order bytes of the address in memory (ADL, ADH) which contains the PCL byte of memory address where program control is transferred. The PCH byte of the destination address is contained in the next memory location (i.e. ADH,ADL + 1). This process is depicted in Fig. 4-4.

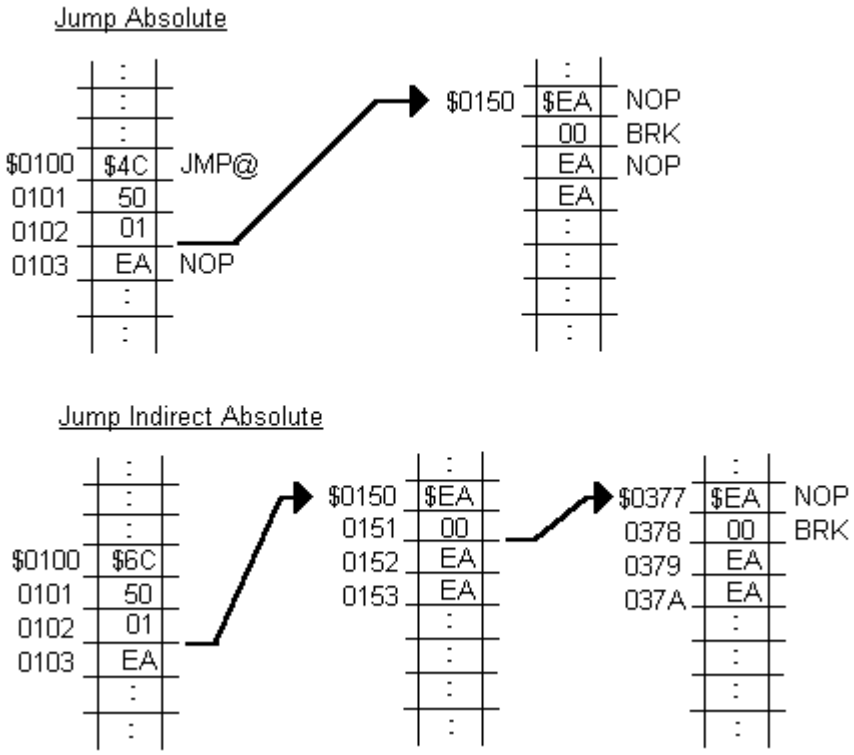


Fig. 4-4. Absolute and Indirect Absolute JMP Operations.

Exp. 6 - JMP Opertations

AIM Printout:

Jump Absolute

```
<K>*=0100
/04
 0100 4C JMP 0150
 0103 00 BRK
 0104 EA NOP
 0105 EA NOP
<K>*=150
/04
 0150 EA NOP
 0151 00 BRK
 0152 EA NOP
 0153 EA NOP

<*>=0100
<R>
 **** PS AA XX YY SS
```

Jump Indirect Absolute

```
<K>*=0100
/04
 0100 6C JMP (0150)
 0103 00 BRK
 0104 EA NOP
 0105 EA NOP
<K>*=150
/04
 0150 77 ???
 0151 03 ???
 0152 EA NOP
 0153 EA NOP
<K>*=0377
/04
 0377 EA NOP
 0378 00 BRK
```

```
 0100 00 00 00 00 FF
<G>/
 0152 EA NOP
<R>
 **** PS AA XX YY SS
 0152 30 00 00 00 FF
```

```
 0379 EA NOP
 037A EA NOP

<*>=0100
<R>
 **** PS AA XX YY SS
 0100 00 00 00 00 FF
<G>/
 0379 EA NOP
<R>
 **** PS AA XX YY SS
 0379 30 00 00 00 FF
```

INC - Increment Memory by One
INX - Increment X Register by One
INY - Increment Y Register by One
DEC - Decrement Memory by One
DEX - Decrement X Register by One
DEY - Decrement Y Register by One

The INC and DEC instructions modify the contents of locations in memory and operate in either the Absolute or Zero Page Addressing Mode. They are either 2 or 3 bytes long. The INX, INY, DEX, and DEY instructions modify the X and Y Registers only, are 1 byte long, and operate in the Implied Addressing Mode.

Exp. 7 - INC and DEC Operations

1. Load $AF and $D0 in locations $0200 and $0201 respectively.
2. Key in the following program and run it.

	Comments
$0100 EE INC@	Increment the contents of
1 00	location $0200
2 02	
3 CE DEC@	Decrement the contents of
4 01	location $0201
5 02	
6 00	Return to Monitor
$0107 EA	No Operation

3. What are the contents of location $0200 and $0201? Answer: $B0, CF
4. Run the program again starting from location $0100. What are the new contents of location $0200 and $0201? Answer: $B1, CE
5. Binary Solution:

INC

	Hex
1010 1111	$AF
+ 0000 0001	+ 01
1011 0000	B0

INC

	Hex	
1101 0000	$D0	0000 0001 = 1
+ 1111 1111	- 01	1111 1110 = 1 (one's
1100 1111	CF	+ 1 complement)
		1111 1111 = -1

5. AIM Printout:

```
<K>*=100
/04
 0100 EE INC 0200
 0103 CE DEC 0201
 0106 00 BRK
 0107 EA NOP

<M>=0200 AF D0 00 00
<*>=0100
<G>/
 0107 EA NOP
<M>=0200 B0 CF 00 00
<*>=0100
<G>/
 0107 EA NOP
<M>=0200 B1 CE 00 00
```

INSTRUCTION MNEMONIC ENTRY MODE

In the AIM 65, there are two primary ways of entering a machine langauge program into memory. The first uses the Monitor <M> and </> Command to enter the various hexadecimal op-codes into specified memory locations. The second method involves the Instruction Mnemonic Entry Mode and employs the <I> Command. This is by far the preferred choice since it is much easier to remember and correlate instuction mnemonics than hexadecimal op-codes. Futthermore, the relative address offsets (used in Branch instructions) are computed by the Mnemonic Assembler from a simple entry of the destination location in the branch instructions (still 2-bytes long). Before program execution, however, the assembler must be exited by using the ESC key whereby the Monitor is automatically re-entered.

Two other Monitor Commands which are very useful in program development are the <R> and <K> Commands. The <R> Command is used to display the current contents of the six Internal Registers of the 6502 while the <K> Command is used to dissassemble op-codes in memory for display/printout in Mnemonic format. For these tutorial reasons, instruction mnemonics rather than hexadecimal op-codes will be used from this point forward to demonstrate the various machine-language programs.

AND - AND Memory with Accumulator
ORA - OR Memory with Accumulator
EOR - Exclusive-OR Memory with Accumulator

The AND, ORA, and EOR instructions perform logical operationson the current contents of the accumulator and the contents of the memory location accessed. The result is stored in the Accumulator. Put another way, the modify the contents of the accumulator by using the contents

of a memory location. They operate in the Immediate, Zero Page, Absolute, and Indirect Addressing Modes. The outcome of their individual operations is as follows:

The result of an AND operation = 1 only if both bits = 1.
The result of an ORA operation = 0 only if both bits = 0.
The result of an EOR operation = 0 only if both bits = SAME.

[Contents]			Result [in Accumulator]		
Memory	Accumulator		AND	ORA	EOR
1	1	⇨	1	1	0
1	0		0	1	1
0	1		0	1	1
0	0		0	0	0

Using the AND, ORA, and EOR instructions with a suitable bit MASK, one or more bits can be cleared, set or complemented.

1. Clearing a Bit to 0 with AND

```
    1100 1010
AND 0111 1111   <-- MASK
    0100 1010
```

2. Setting a Bit to 1 with ORA

```
    1001 1010
ORA 0100 0000   <-- MASK
    1101 1010
```

3. Complementing (Toggling) a Bit with EOR

```
    1010 0011
EOR 0010 0000   <-- MASK
    1000 0011
EOR 0010 0000   <-- MASK
    1010 0011
EOR 0010 0000   <--MASK
    1000 0011
```

AND - clears bits
ORA - sets bits
EOR - complements (toggles) bits

Exp. 8 - AND, ORA, and EOR Operations

1. Store $AA, BB, CC and 44 in locations $0200-0203.
2. Key in the following program:

```
                                  Comments
$0300  LDA $0200  Load A with the contents of $0200
     3 AND $0230  AND A with contents of $0203
     6 STA $0200          Store A with contents of  $0200
```

```
    9  LDA $0201          Load A with contents of $0201
    C  ORA $0203          OR A with contents of $0203
$0312  LDA $0202Load A with the contents of $0202
    5  EOR $0203XOR A with contents of $0203
    B  BRK                Return to Monitor
$031C  NOP                No operation
```

3. Run the program beginning from location $0300.
4. What are the new contents of $0200-0203? Answer: $00, FF, 88, 44.
5. Run the program again beginning at $0300. What are the contents of locations $0200-0203 now? Answer: $00, FF, CC, 44.
6. AIM Printout.

```
<M>=0200 AA BB CC 44
<*>=0300
<G>/
 031C EA NOP
<M>=0200 00 FF 88 44
<*>=0300
<G>/
 031C EA NOP
<M>=0200 00 FF CC 44
```

```
<K>*=0300
/11
 0300 AD LDA 0200
 0303 2D AND 0203
 0306 8D STA 0200
 0309 AD LDA 0201
 030C 0D 0RA 0203
 030F 8D STA 0201
 0312 AD LDA 0202
 0315 4D EOR 0203
 0318 8D STA 0202
 031B 00 BRK
 031C EA NOP
```

CLC - Clear Carry Flag
SEC - Set Carry Flag

The CLC and SEC instructions respectively clear (=0) and set (=1) the Carry Flag (C) in the Processor Status (P) Register. The clearing and setting of this flag is an important step when doing binary or decimal (BCD) addition and subtraction. It can be thought of as a flag bit distinct from the accumulator itself but directly affected by accumulator operations as though it were a Ninth Bit in the accumulator. CLC and SEC are both one-byte instructions operating in the Implied Addressing Mode.

CLD - Clear Decimal Mode
SED - Set Decimal Mode

CLD and SED are one-byte insturctions which also operate in the Implied Addressing Mode and respectively clear (=0) and set (=1) the Decimal Flag (D) in the Processor Status Register. Setting the Decimal Mode (SED) allows the 6502 to carry out arithmetic operations on binary-coded-decimals (BCD) data and story/display the results in BCD format. Otherwise, all operations are carried out in straight binary format. For example, in the Binary Mode (D=0), 15_{10} = 0000 1111 whereas in the Decimal Mode (D=1), 15_{10} = 0001 0101.

ADC - Add to Accumulator with Carry
SBC - Subtract from Accumulator with Barrow

The ADC instuction adds the value of memory and the carry flag from the previous operation to the contents of the accumulator and stores the result in the accumulator. If the result of a binary add exceeds 255 or a decimal add exceeds 99, the carry flag is set (=1) otherwise it is cleared (=0). If the result is zero, the zero flag (Z) is set otherwise it is not. If the result exceeds +127 or -128, the overflow flag (V) is set.

SBC subtracts the value of memory and borrow from the value of the of the accumulator using two's complement arithmetic and stores the result in the accumulator. Borrow is defined as carry complemented (C). The carry flag is set (borrow cleared) if the result >= 0. The carry flag is cleared (borrow set) if the result < 0. The N, Z, and V flags operate in the same manner as with ADC.

Prior to first using an ADC instruction, the carry flag should be cleared (CLC) to indicate a no-carry condtion. Similary, befor using the SBC instruction, the carry flag should be set (SBC) to indicate a no-borrow condition.

* Clear the carry flag (CLC) prior to add (ADC)
* Set the carry flag (SEC) prior to subtract (SBC)

CLV - Clear Overflow Flag

The Overlow Flag (V) is set (=1) if the addition (ADC) or subtraction (SBC) of two signed binary numbers produces a result (in the Accumulator) which is > +127 or < -128. It can be rest (=0) by the one-byte long CLV instruction. It is also automatically reset (=0) at the beginning of the next ADC or SBC instruction.

Exp. 9 - Addition of two Binary Numbers

GOAL: Add the contents of location $0203 to locations $0200-0202 and store the results in $0100-0102.

1. Load $05, 06, 07, and 04 into locations $0200-0203.
2. Key in the following program beginning at $0300.

```
                                Comments
$0300 CLC           Clear the carry flag
    1 CLD           Clear the decimal mode
    2 LDA $0200     Load A with the contents of $0200
    3 ADC $0203 Add the contents of $0203
    8 STA $0100     Store the result in $0100
    B LDA $0201 Load A with the contents of $0201
    E ADC $0203 Add the contents of $0203
$0311 STA $0101 Store the result in $0101
    4 LDA $0202     Load A with the contents of $0202
    7 ADC $0203 Add the contents of $0203
    A STA $0102 Store the result in $0102
    D BRK           Return to Monitor
$031E NOP           No operation
```

3. Run the program
4. Examine the contents of locations $0100, 0101, 0102. What are they? Answer: $09, 0A, 0B
5. Replace the CLD instruction in location $0301 with the SED instruction.
6. Repeat steps 1 and 3 again and examine the contents of $0100-0102. What are they?

Answer: $09, 10, 11.

7. Why is the CLC instruction included in both programs? Answer: To indicate an initial no-carry condition.
8: AIM Printout

Binary Addition

```
<M>=0200 05 06 07 04
<*>=0300
<G>/
 031E EA NOP
<M>=0100 09 0A 0B 00

<K>*=0300
/13
 0300 18 CLC
 0301 D8 CLD
 0302 AD LDA 0200
 0305 6D ADC 0203
 0308 8D STA 0100
 030B AD LDA 0201
 030E 6D ADC 0203
 0311 8D STA 0101
 0314 AD LDA 0202
 0317 6D ADC 0203
 031A 8D STA 0102
 031D 00 BRK
 031E EA NOP
```

Decimal Addition

```
<M>=0200 05 06 07 04
<*>=0300
<G>/
 031E EA NOP
<M>=0100 09 10 11 00

<K>*=0300
/13
 0300 18 CLC
 0301 D8 SED
 0302 AD LDA 0200
 0305 6D ADC 0203
 0308 8D STA 0100
 030B AD LDA 0201
 030E 6D ADC 0203
 0311 8D STA 0101
 0314 AD LDA 0202
 0317 6D ADC 0203
 031A 8D STA 0102
 031D BRK
 031E EA NOP
```

Exp. 10 - Subtraction of 2 Binary Numbers

GOAL: Subtract $81 from $E9 twice. $E9 is stored in location $0300, $81 in location $0301 and the results in locations $0302, 0303.

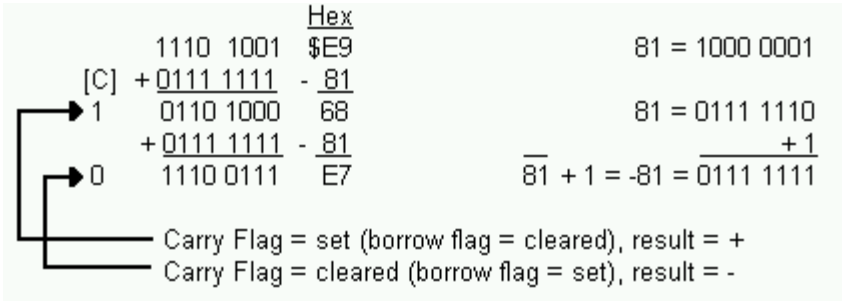
1. Load $E9 and $81 into locations $0300 and $0301.
   (XX = dosen't matter)

$0300	0301	0302	0303
$E9	81	XX	XX

2. Key in the following program beginning at $0200.

	Comments
$0200 CLD	Clear the decimal mode
1 SEC	Set the carry flag
2 LDA $0300	Load A with the contents of $0300
5 SBC $0301	Subtact the contents of $0301
8 STA $0302	Store the result in $0302
B SBC $0301	Subtract contents of $0301 again
E STA $0303	Store the result in $0303
$0211 BRK	Return to Monitor
$0212 NOP	No operation

3. Run the program from $0200.
4. What are the contents of $0302, 0303? Answer: $68, E7.
5. What is the value of the carry flag? Answer: 0.
6. Binary Solution.



Exp. 11 - Subtraction of 2 Number in the Decimal Mode

GOAL: Subtract 81 from 89 twice. 89 and 81 are stored in locations $0300, 0301 and the first and second results of the seccessive subtractions are stored in locations $0302, 0303.

1. Load $89 and $81 into locations $0300 and $0301.
   (XX = dosen't matter)

$0300	0301	0302	0303
$89	81	XX	XX

2. Key in the following program beginning at $0200.

	Comments
$0200 SED	Set the decimal mode
1 SEC	Set the carry flag (Clear borrow)
2 LDA $0300	Load A with the contents of $0300
5 SBC $0301	Subtact the contents of $0301
8 STA $0302	Store the result in $0302
B SBC $0301	Subtract contents of $0301 again
E STA $0303	Store the result in $0303
$0211 BRK	Return to Monitor
$0212 NOP	No operation

3. Run the program starting from location $0200.
4. What are the contents of $0302, 0303? Answer: $08, 27.
5. What is the state of the carry flag? Answer: 0.
6. When performing BCD subtraction, a negative result is indicated by a cleared (=0) Carry Flag (set Borrow Flag). This is similar to Binary subtraction except that the answer is in ten's complement form not in two's complement form. In order to convert the result of the second subtraction (27) into a meaningful (positive) number with a (separate) negative sign, it must be ten's complemented.

$$\overline{27}_{10} = 27 - 100 = -73$$

7. AIM Printout

Binary Subrtaction

```
<M>=0300 E9 81 00 00
<*>=2000
<G>/
 0212 EA NOP
<M>=0300 E9 81 68 E7

<K>*=0200
```

Decimal Subtraction

```
<M>=0300 89 81 00 00
<*>=2000
<G>/
 0212 EA NOP
<M>=0300 89 81 08 27

<K>*=0200
```

```
/09                              /09
 0200 D8 CLD                      0200 F8 SED
 0201 38 SEC                      0201 38 SEC
 0202 AD LDA 0300                 0202 AD LDA 0300
 0205 ED SBC 0301                 0205 ED SBC 0301
 0208 8D STA 0302                 0208 8D STA 0302
 020B ED SBC 0301                 020B ED SBC 0301
 020E BD STA 0303                 020E BD STA 0303
 0211 00 BRK                      0211 00 BRK
 0212 EA NOP                      0212 EA NOP
```

BCC - Branch on Carry Clear (C = 0)
BCS - Branch on Carry Set (C = 1)
BEQ - Branch on Result Zero (Z = 1)
BNE - Branch on Result not Zero (Z = 0)
BMI - Branch on Negative Result (N = 1)
BPL - Branch on Positive Result (N = 0)
BVC - Branch on Overflow Clear (V = 0)
BVS - Branch on Overflow Set (V = 1)

Each branch instruction (BCC...) interrogates (tests) a specific bit (C, Z, N, V) in the Processor Status (P) Register. Depending upon the state (0 or 1) of the bit, a branch to another instruction may occur. If no branch occurs, the program continues along as if the branch instruction weren't there. Branch instuctions employ the Relative Addressing Mode which means that the destination location of the branch is relative to the location following the branch instruction. It can be forwards or backwards from that point by a maximum of +127 ($7F) or -128 ($80) locations and may cross a page boundary (Ex. --> going from location $02FF to location $0300 crosses the Page 2->3 boundary). All branch instructions are two bytes long; the relative displacement. In most assemblers, including the primitive AIM instruction mnemonic entry mode, this displacement is computed from the 16-bit destination address entry following the branch instruction.

CMP - Compare Memory and the Accumulator
CPX - Compare Memory and the X Register
CPY - Compare Memory and the Y Register

The CMP, CPX, and CPY instructions compare the contents of memory locations to the current contents of the Accumulator (A), X-Register (X), Y-Register (Y), respectively to determine if the difference between the two is positive, negative or zero. The specified contents of a particular memory location are subtracted from the current contents of A, X, or Y. No memory locations are altered and only the Processor Status (P) Register is changed. The Z flag (of the P register) is set (=1) by an equality, otherwise it is reset (=0) by the status of the sign bit (Bit No. 7) while the Carry Flag is set/reset if the contents of the accessed memory location are less/greater than the current contents of the register (A, X, Y) under examination. These comparision instructions are almost always used in conjunction with branch instructions and operate in the Absolute, Zero Page and Immediate Addressing Modes. Depending upon the results of the comparision, a branch to another part of the program can be taken.

Exp. 12 - Branch and Comparision Operations

GOAL: Branch to different routines depending upon the value of the number written to location $0200 begin equal to, greater, or less than $75. If = $75 store in location $0201. If > $75, store in $0202. If < $75, store in location $0203.

1. Load $75 in location $0200 and $00 in $0201-$0203.

$0200	0201	0202	0203
$75	00	00	00

2. Key in the following program.

	Comments
$0300 CLC	Clear the carry flag
1 LDA $0200	Load A with the contents of $0200
4 CMP #75	Compare $75 with the contents of A
6 BEQ $030C	If equal (i.e. Z=1) go to $030C
8 BCS $0311	If greater (i.e. C=1) go to $0311
A BCC $0316	If less than (i.e. C=0) go to $0316
C STA $0201	Store equal result in $0201
F BRK	Return to Monitor
$0310 NOP	
1 STA $0202	Store greater result in $0202
4 BRK	Return to Monitor
5 NOP	
6 STA $0203	Store lesser result in $0203
9 BRK	Return to Monitor
$031A NOP	No operation

3. Run the program. What does the display read? Answer: < 0310 EA NOP
4. What are the contents of locations $0200-0203? Answer: $75, 75, 00, 00.
5. Run the program again after loading $74, 00, 00, 00 in locations $0200-0203.
6. What are the new contents of $0200-0203? Answer: $74, 00, 00, 74.
7. Run the program again after loading $76, 00, 00, 00 in locations $0200-0203.
8. What are the new contents of $0200-0203? Answer: $76, 00, 76, 00.
9. AIM Printout.

```
<M>=0200 75 00 00 00
<*>=0300
<G>/
 0310 EA NOP
<M>=0200 75 75 00 00
</> 0200 74 00 00 00
<*>=0300
<G>/
 031A EA NOP
<M>=0200 74 00 00 74
</> 0200 76 00 00 00
<*>=0300
<G>/
 0315 EA NOP
<M>=0200 76 00 76 00

<K>*=0300
/15
 0300 18 CLC
 0301 AD LDA 0200
 0304 C9 CMP #75
 0306 F0 BEQ 030C
 0308 B0 BCS 0311
 030A 90 BCC 0316
 030C 8D STA 0201
```

```
030F 00 BRK
0310 EA NOP
0311 8D STA 0202
0314 00 BRK
0315 EA NOP
0316 8D STA 0203
0319 00 BRK
031A EA NOP
```