65816

16 Bit Card

Software Developer's Guide

APPLIED ENGINEERING

Special Applied Engineering (Beta)16 Bit Card Software Developer's Package

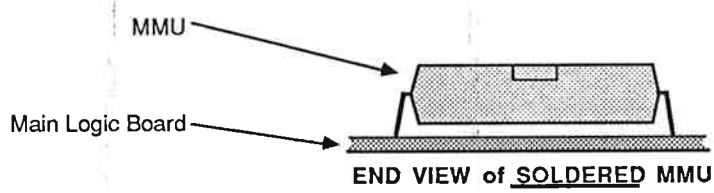
The version of the 16 Bit Card that is being sent to software developers is "only" capable of addressing up to 8 Meg of memory. The version that will be shipped to customers will be capable of addressing up to 16 Meg of memory, the full capability of the 65816 processor. This Beta version of the 16 Bit Card is provided with only one ribbon cable to connect it to a RamWorks II memory expansion card. Ordinarilly it would have another shorter ribbon cable to connect the 16 Bit Card (P2) to a 2 Meg. RamWorks memory expander piggy-back card. This "2 Meg." cable is not required when using the 512K version of the RamWorks memory expander piggy-back card.

Applied Engineering Technical Support

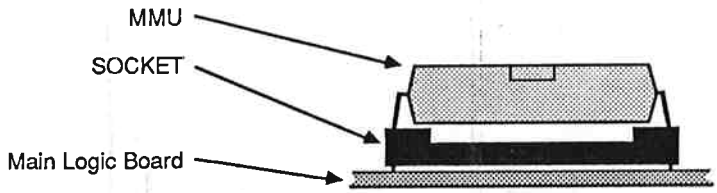
Applied Engineering has a staff of technicians dedicated to answering specific questions about Applied Engineering products and software. If your question cannot be resolved by the technician, he will refer the question to the appropriate engineer. The technical support representatives are available Monday through Friday, between the hours of 9 AM to 5 PM (Central). The technical support telephone number is (214)241-6069. Please have as much information as possible available about your problem if you call.

Soldered MMU chip on the //e main logic board

Important!: Some (very few) Apple //e's were manufactured with the MMU chip soldered in. If your //e does not have a socket for the MMU, the MMU will have to be desoldered and a socket installed. This is very tricky and should only done by a professional with the proper tools. Apple Computer, Inc has assured us that //e's are now assembled with socketed MMU chips.



END VIEW of SOLDERED MMU

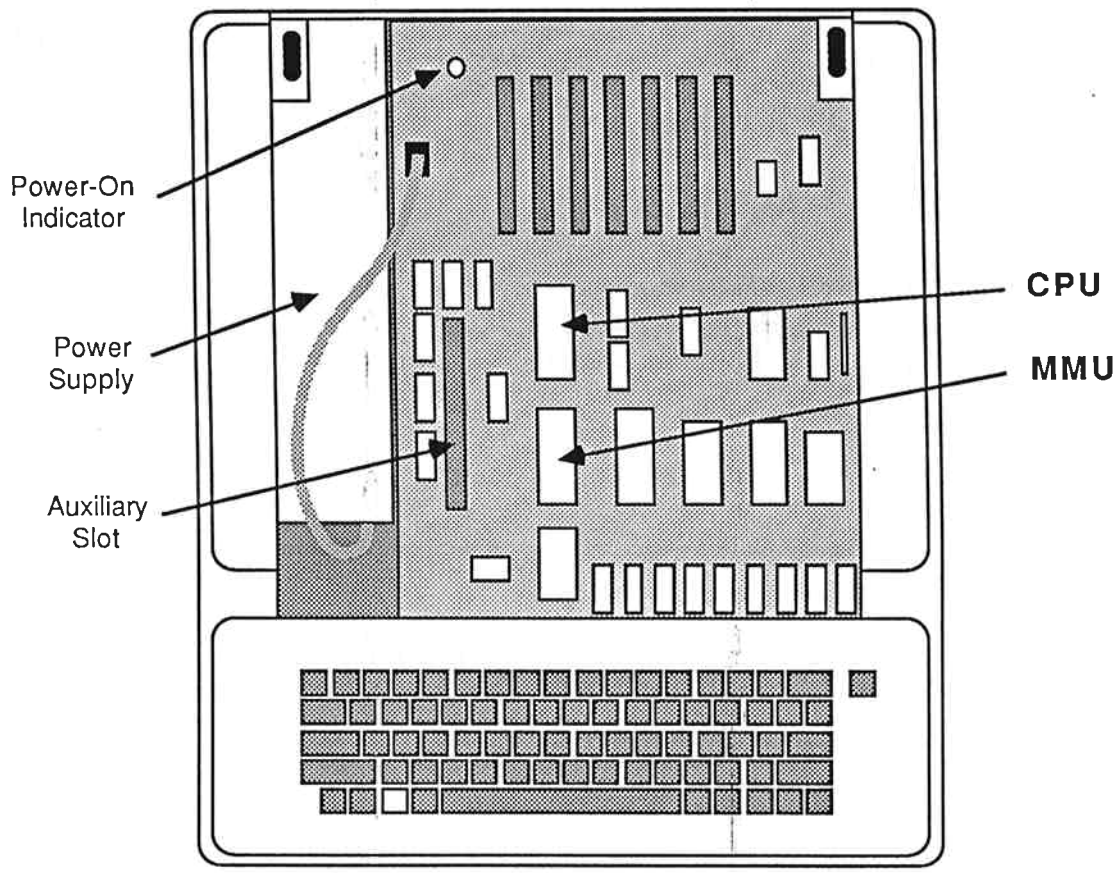


END VIEW of SOCKETED MMU

Installing the 16 Bit Card

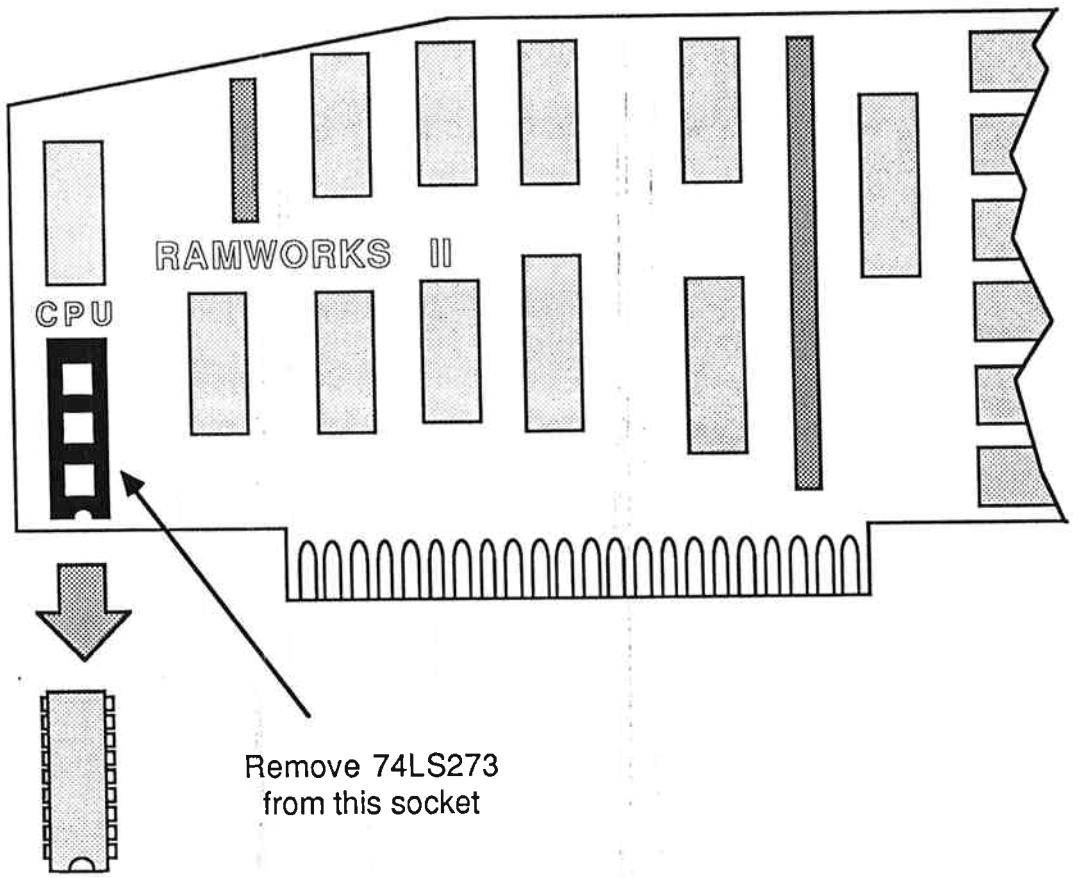
- Turn the //e power switch to the OFF position, but leave the computer plugged in.
- Remove the //e top lid.
- Make sure the power-on indicator light inside the computer is OFF. (See Illustration 1.)

Illustration 1:



- If installed, remove the RamWorks II card from the //e auxiliary slot.
- Remove the 74LS273 chip from the RamWorks II socket marked "CPU." (Refer to Illustration 2.) Carefully set the RamWorks II aside and store the 74LS273 in a safe place.

Illustration 2:



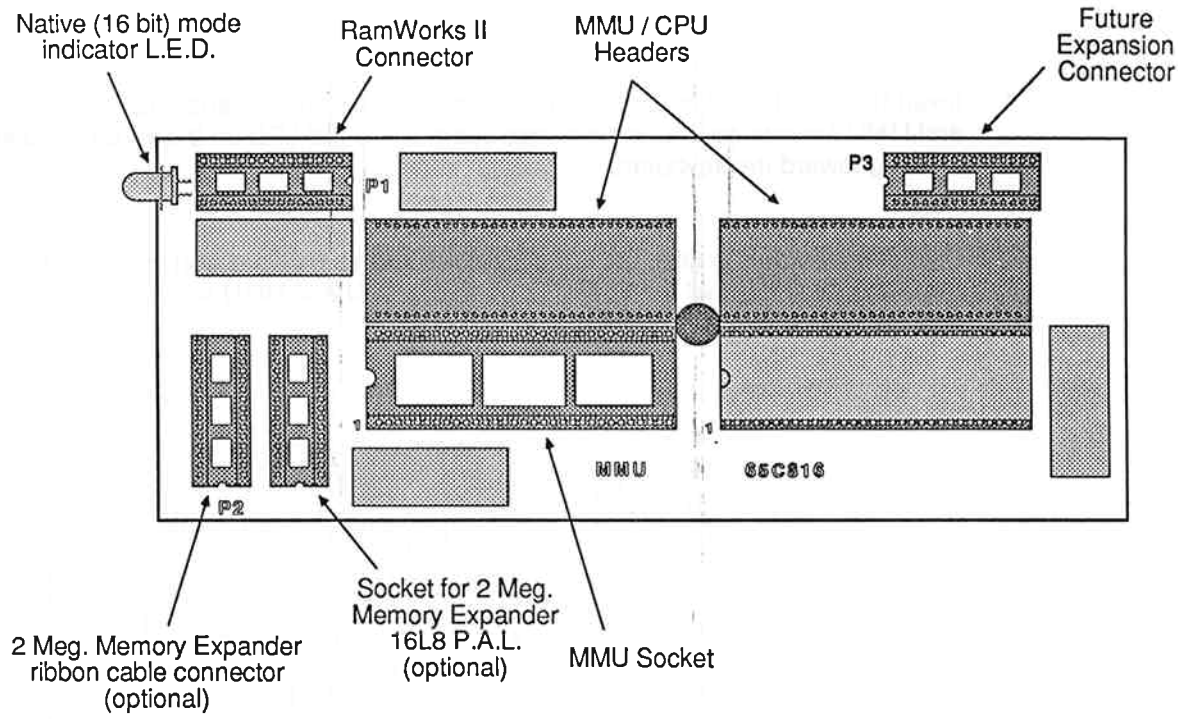
- Locate the CPU chip and the MMU chip on the //e main logic board. (Refer to Illustration 1.)

- Remove the MMU chip from the //e main logic board. Use a small flatblade screwdriver to gently lift alternate ends of the chip until it is free from its socket. Carefully set the MMU chip aside.

- Remove the CPU chip in the same manner. The //e's CPU chip is not required with the 16 Bit Card installed. Store it in a safe place.

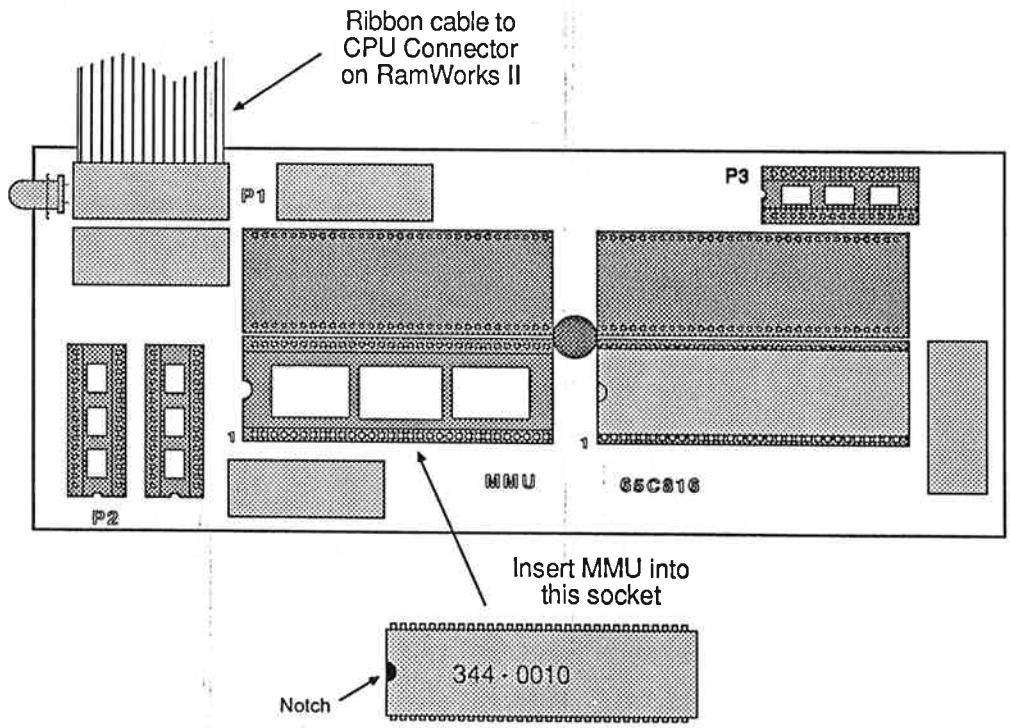
- Verify that all pins on the 16 Bit Card CPU and MMU header connectors are straight. (Refer to Illustration 3.)

Illustration 3:



- Install the MMU chip on the 16 Bit Card, as shown in Illustration 4. Be sure the notch is oriented as indictated in the illustration.

Illustration 4:

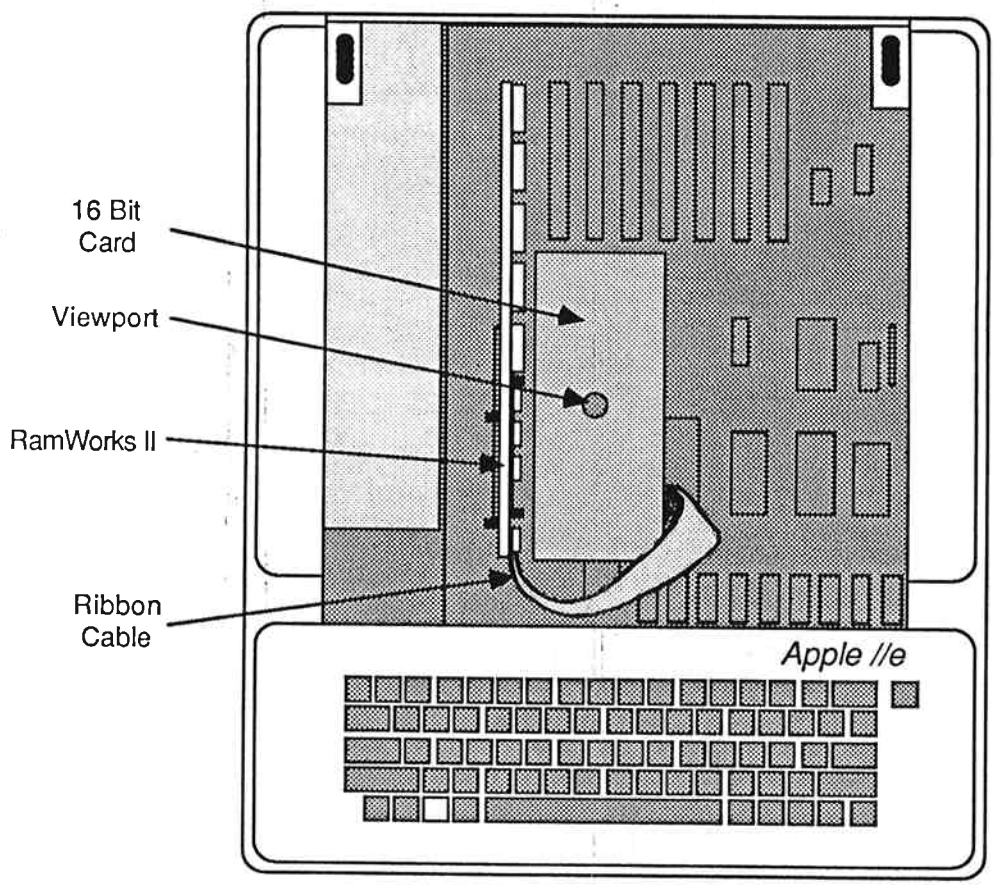


- Plug one of the ribbon cable header connectors (both ends are the same) into the 16 Bit Card socket marked "P1" exactly as shown in Illustration 4.

- Invert the 16 Bit Card (solder side up; component side down) and position it above the CPU and MMU sockets on the //e main logic board . The red LED on the 16 Bit Card should be pointing toward the keyboard.

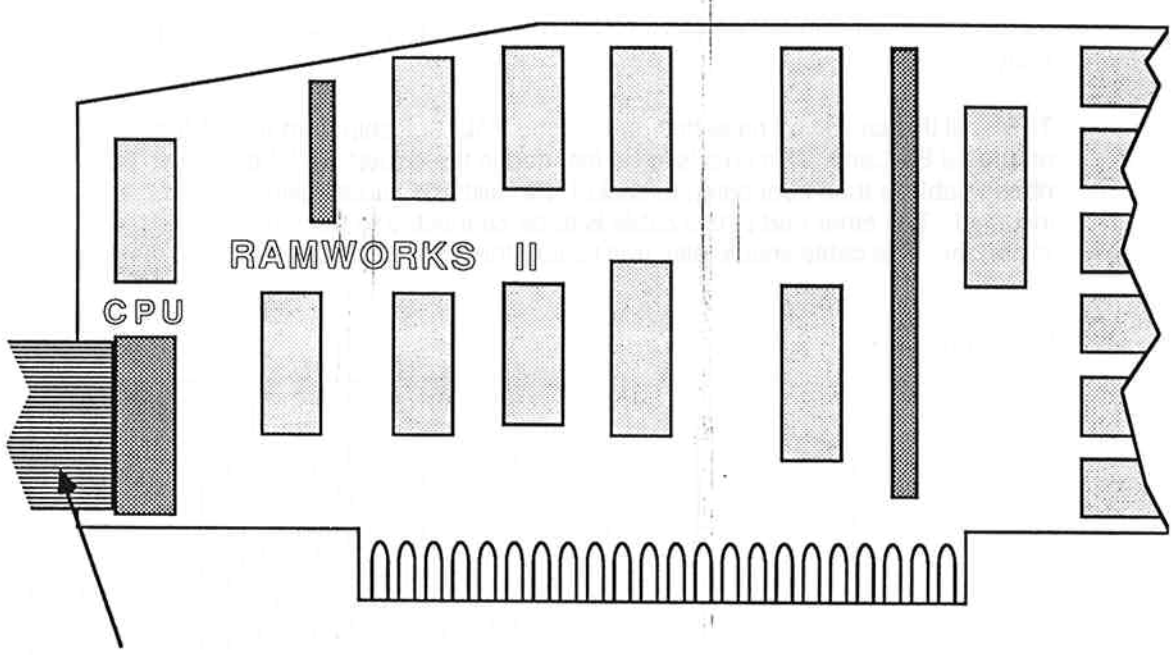
- Using the viewport to align the header pins on the 16 Bit Card with the socket holes on the //e main logic board, install the 16 Bit Card into the CPU and MMU sockets. Press gently but firmly until the card is securely seated.

Illustration 5:



- Position the keyboard end of the RamWorks II card near the installed 16 Bit Card. Install the free end of the 16 Bit Card ribbon cable to the RamWorks II socket marked "CPU." Verify that all header connector pins are fully seated in the socket and that the cable is installed as shown in Illustration 6.

Illustration 6:



- Install the RamWorks II card into the //e's auxiliary slot.

- Replace the //e's top lid. Installation is complete.

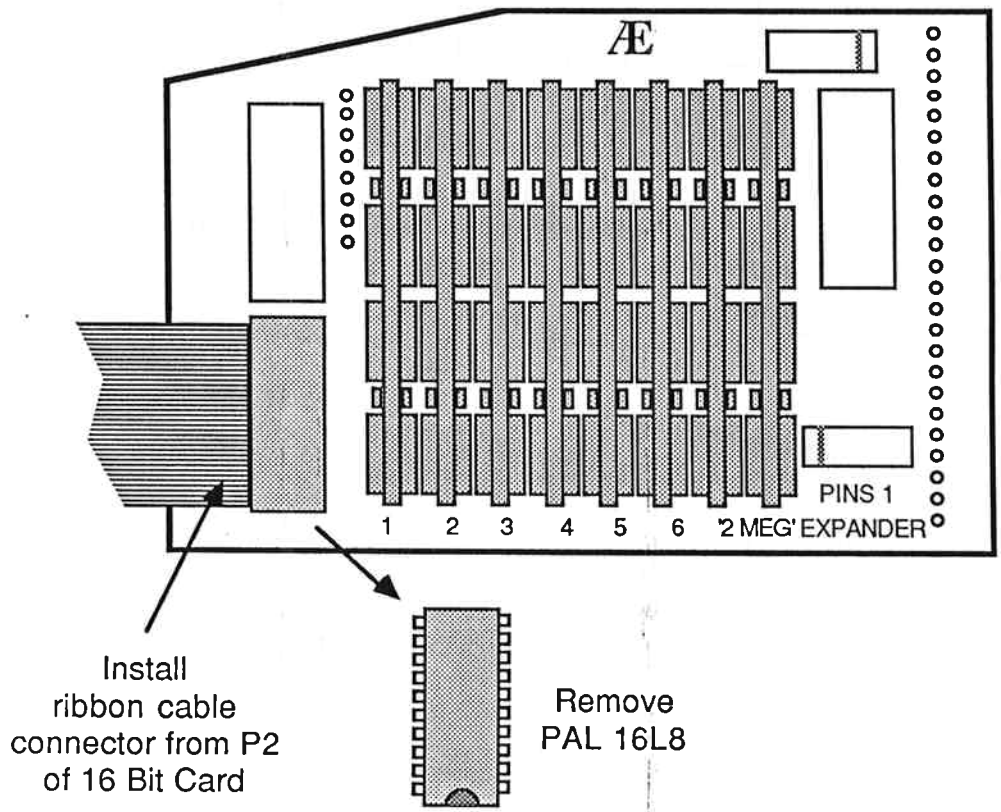
- Boot the disk labeled "Æ 16 Bit Developer's Disk" and run the program "TEST816."

  If the computer will not boot or fails the test program, check to see that all chips, cables, and connectors are securely seated in their sockets. Also check for bent pins on the MMU chip and on the ribbon cable and CPU / MMU headers.

For developers with the 2 Meg. RamWorks memory expansion piggy-back card, a special ribbon cable is required to connect the 16 Bit Card to the 2 Meg. expander card. This cable is available from Applied Engineering.

To install this cable you must first remove the PAL16L8 chip from the 2 Meg. expander and install it on the 16 Bit Card. This chip is to be inserted in the socket *NEXT* to socket "P2." One end of the ribbon cable is then connected to socket "P2" with the cable trailing toward the keyboard when installed. The other end of this cable is to be connected to the empty 16L8 socket on the 2 Meg. expander. The cable should also trail toward the keyboard end of the card when installed.

Illustration 7:



The 16 Bit Card will allow you to address up to 16 Meg linearly, using the 65816 processor's native mode of operation. In 65C02 emulation mode, the memory on the Ramworks II card will look and act exactly like the memory on a Ramworks II without the 16 Bit Card installed, with one exception: with the 16 Bit Card installed, hitting CONTROL-RESET will always put you back in BANK 0; on a Ramworks II without the 16 Bit Card, CONTROL-RESET has no effect on the bank register.

If you have a 1 Meg Ramworks II, you will get banks 00 thru 0F, whether you are in 65C02 emulation mode or in the 65816 native mode. If you have a 1 Meg Ramworks II with a 1/2 Meg (512 K) piggy back, you will get banks 00-17, whether you are in 65C02 emulation mode or in 65815 native mode.

If you have worked with the Applied Engineering 2 Meg piggy back board before, you probably know of its unique memory mapping scheme. Banks are arranged in the order 00 through 0F (on Ramworks II), then from 10-17,30-37,50-57,70-77 (on the 2 Meg piggy back). This is done to maintain compatibility with other piggy back cards from Applied Engineering, and with the original Ramworks. In 65C02 emulation mode, the banks retain this partially non linear mapping; however, in 65816 native mode, the banks become linearized, from 00 thru 2F.

In an Apple IIe equipped with a Ramworks II but not a 16 Bit Card, the memory on the Ramworks II is accessed as alternate banks of auxiliary memory. The 64K of memory on the Apple IIe motherboard is accessed when the MMU's softswitches are set one way (MAIN memory) and the memory on the Ramworks II card is accessed when the MMU's softswitches are set the other way (AUXILIARY memory). One unique bank of 64K of memory is chosen from the available banks on the Ramworks II card by the BANK SELECT REGISTER, which is in the IIe's memory map at location $C073. Bank 0 on the Ramworks II card is where the video generator circuits in the Apple IIe look for the 80 column video and Double High Resolution graphics information. No matter what 64K bank the BANK SELECT REGISTER is pointing to, all video access goes to bank 0. (This feature is patented by Applied Engineering.)

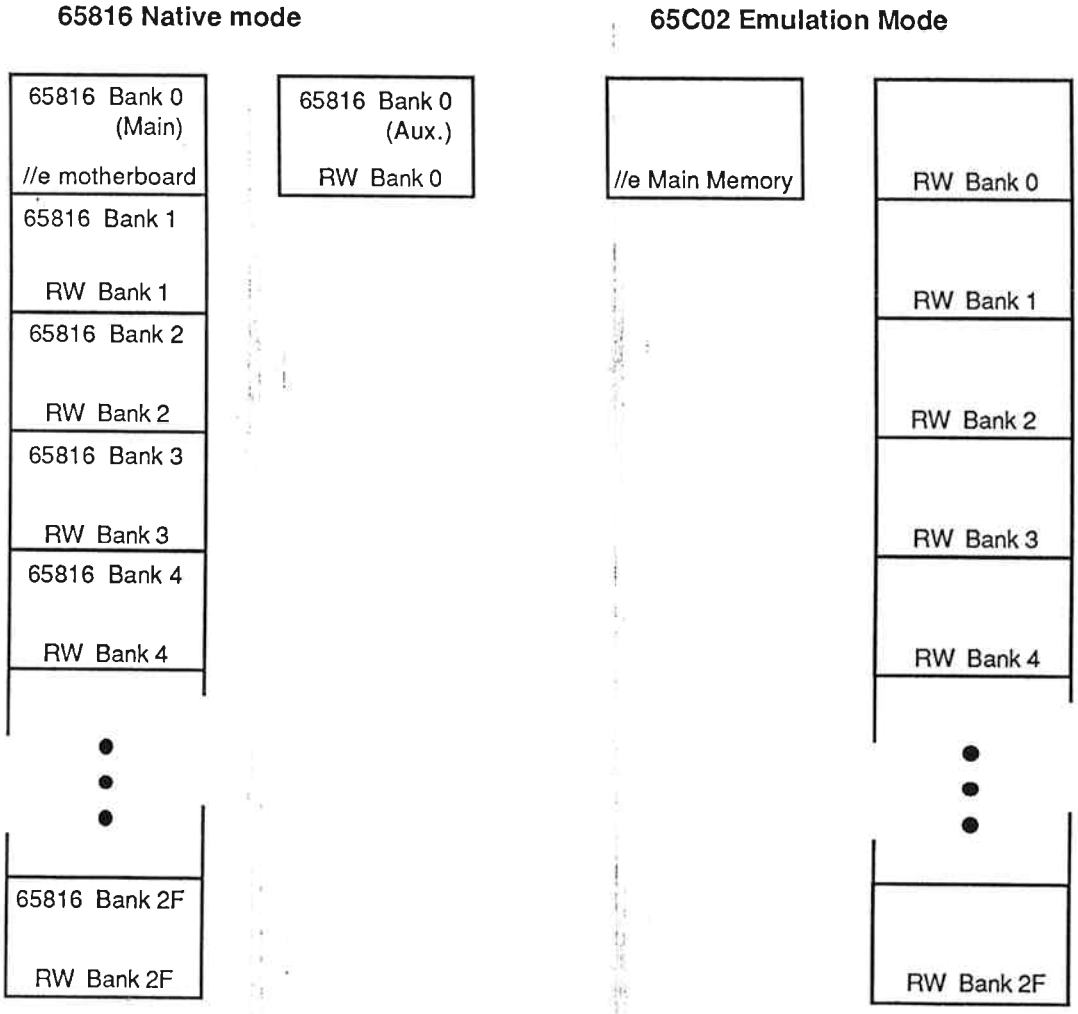
All hardware locations, including the MMU's softswitches, are located in the $C000 to $CFFF range of memory (hereafter referred to as $CXXX), which is called the HARDWARE PAGE. With a Ramworks II installed, access to $CXXX range of memory IN ANY BANK will access the hardware page. In other words, the $CXXX range of ANY BANK is mapped into the HARDWARE PAGE.

When the 16 Bit Card is installed and running in the 65C02 emulation mode, the softswitches still work exactly as they do without the 16 Bit Card. However, when the processor is in the 65816 native mode accesses to the hardware page can only be accomplished from 65816 BANK 0. Any bank other than 65816 BANK 0 will not allow you to access the hardware page. If you are in a 65816 bank other than BANK 0, and you access the $CXXX range, you will be accessing RAM MEMORY, NOT the hardware page. When you are in 65816 BANK 0, the Apple IIe softswitches, which are in the hardware page, will allow you to flip back and forth between main memory or auxilllary memory. If you are in a bank other than BANK 0, the softswitches will have no effect. That is, even if you go into 65816 BANK 0 and flip MMU softswitches so that you are looking at AUX memory, when you go into a 65816 bank other than BANK 0, the softswitches will have no effect. This is because there is no auxiliary memory associated with 65816 banks other than BANK 0. In 65816 native mode, BANK 0 main memory is the 64K on the Apple IIe motherboard, and BANK 0 auxiliary memory is the first 64K on the Ramworks card. This allows you to use the softswitches to flip between main memory and aux memory (as long as you are in BANK 0); this makes using the 80 column video and double high resolution graphics easier. If the 65816 is in a bank other than BANK 0, it will map into a corresponding bank on the Ramworks II or a piggy back card.

The softswitches that control access to the LANGUAGE CARD area of memory that overlays the motherboard ROM space can only be accessed from 65816 BANK 0. Further, they only have an effect in 65816 BANK 0. Because the 65816 looks for its interrupt vectors in BANK 0 at locations $FFF4 through $FFFF, you must use the language card RAM space to store these vectors.

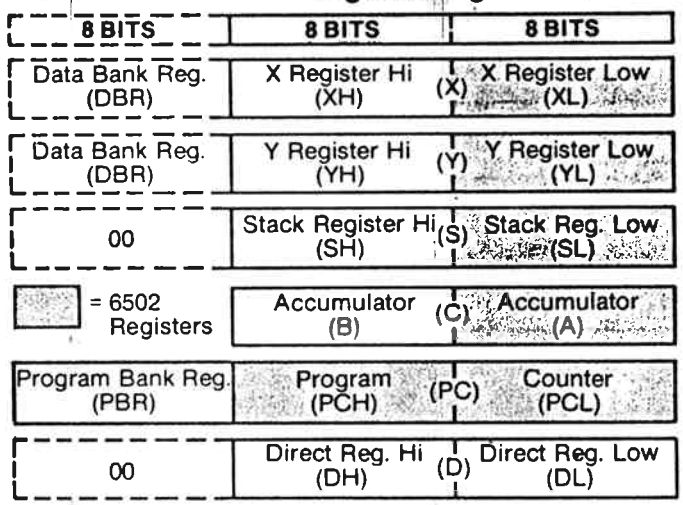
One further note on using softswitches: The 65816 can have 8-bit wide registers or 16-bit wide registers. In the 65C02 emulation mode all registers (except the PC) are 8-bits wide, but in the native mode you can set the width of the X and Y registers with the X bit in the Processor Status Register (P). If X=0 the X and Y registers are 16-bits wide, and if X=1 then X and Y are 8-bits wide. The M bit in the P register controls the width of the Accumulator. If M=0 then the Accumulator is 16-bits wide, and if M=1 then the accumulator is 8-bits wide. You should only access the hardware page if M=1 and X=1. This will prevent unwanted problems because of writes to two successive addresses.

16 Bit Memory Maps

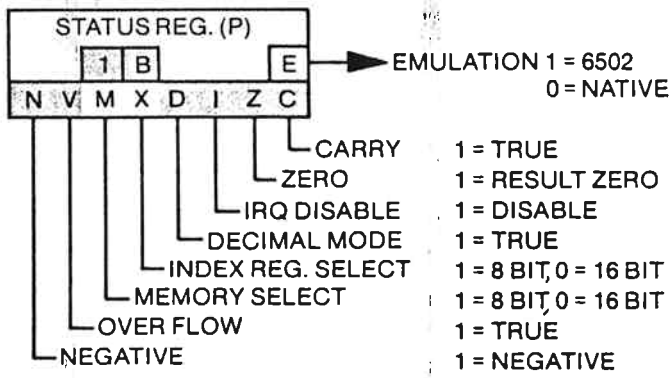


The following pages have been excerpted from the W65C816 Data Sheet and are reprinted with permission from Western Design Center, Inc.

W65C816 Processor Programming Model



Status Register Coding



Functional Description

The W65C802 offers the design engineer the opportunity to utilize both existing software programs and hardware configurations, while also achieving the added advantages of increased register lengths and faster execution times. The W65C802's "ease of use" design and implementation features provide the designer with increased flexibility and reduced implementation costs. In the Emulation mode, the W65C802 not only offers software compatibility, but is also hardware (pin-to-pin) compatible with 6502 designs...plus it provides the advantages of 16-bit internal operation in 6502-compatible applications. The W65C802 is an excellent direct replacement microprocessor for 6502 designs.

The W65C816 provides the design engineer with upward mobility and software compatibility in applications where a 16-bit system configuration is desired. The W65C816's 16-bit hardware configuration, coupled with current software allows a wide selection of system applications. In the Emulation mode, the W65C816 offers many advantages, including full software compatibility with 6502 coding. In addition, the W65C816's powerful instruction set and addressing modes make it an excellent choice for new 16-bit designs.

Internal organization of the W65C802 and W65C816 can be divided into two parts: 1) The Register Section, and 2) The Control Section. Instructions (or opcodes) obtained from program memory are executed by implementing a series of data transfers within the Register Section. Signals that cause data transfers to be executed are generated within the Control Section. Both the W65C802 and the W65C816 have a 16-bit internal architecture with an 8-bit external data bus.

Instruction Register and Decode

An opcode enters the processor on the Data Bus, and is latched into the Instruction Register during the instruction fetch cycle. This instruction is then decoded, along with timing and interrupt signals, to generate the various Instruction Register control signals.

Timing Control Unit (TCU)

The Timing Control Unit keeps track of each instruction cycle as it is executed. The TCU is set to zero each time an instruction fetch is executed, and is advanced at the beginning of each cycle for as many cycles as is required to complete the instruction. Each data transfer between registers depends upon decoding the contents of both the Instruction Register and the Timing Control Unit.

Arithmetic and Logic Unit (ALU)

All arithmetic and logic operations take place within the 16-bit ALU. In addition to data operations, the ALU also calculates the effective address for relative and indexed addressing modes. The result of a data operation is stored in either memory or an internal register. Carry, Negative, Overflow and Zero flags may be updated following the ALU data operation.

Internal Registers (Refer to Programming Model)

Accumulators (A, B, C)

The Accumulator is a general purpose register which stores one of the operands, or the result of most arithmetic and logical operations. In the Native mode (E=0), when the Accumulator Select Bit (M) equals zero, the Accumulator is established as 16 bits wide (A + B = C). When the Accumulator Select Bit (M) equals one, the Accumulator is 8 bits wide (A). In this case, the upper 8 bits (B) may be used for temporary storage in conjunction with the Exchange Accumulator (XBA) instruction.

Data Bank Register (DBR)

During modes of operation, the 8-bit Data Bank Register holds the default bank address for memory transfers. The 24-bit address is composed of the 16-bit instruction effective address and the 8-bit Data Bank address. The register value is multiplexed with the data value and is present on the Data/Address lines during the first half of a data transfer memory cycle for the W65C816. The Data Bank Register is initialized to zero during Reset.

Direct (D)

The 16-bit Direct Register provides an address offset for all instructions using direct addressing. The effective bank zero address is formed by adding the 8-bit instruction operand address to the Direct Register. The Direct Register is initialized to zero during Reset.

Index (X and Y)

There are two Index Registers (X and Y) which may be used as general purpose registers or to provide an index value for calculation of the effective address. When executing an instruction with indexed addressing, the microprocessor fetches the opcode and the base address, and then modifies the address by adding the Index Register contents to the address prior to performing the desired operation. Pre-indexing or post-indexing of indirect addresses may be selected. In the Native mode (E=0), both Index Registers are 16 bits wide (providing the Index Select Bit (X) equals zero). If the Index Select Bit (X) equals one, both registers will be 8 bits wide, and the high byte is forced to zero.

Processor Status (P)

The 8-bit Processor Status Register contains status flags and mode select bits. The Carry (C), Negative (N), Overflow (V), and Zero (Z) status flags serve to report the status of most ALU operations. These status flags are tested by use of Conditional Branch instructions. The Decimal (D), IRQ Disable (I), Memory/Accumulator (M), and Index (X) bits are used as mode select flags. These flags are set by the program to change microprocessor operations.

The Emulation (E) select and the Break (B) flags are accessible only through the Processor Status Register. The Emulation mode select flag is selected by the Exchange Carry and Emulation Bits (XCE) instruction. Table 1, W65C802 and W65C816 Mode Comparison, illustrates the features of the Native (E=0) and Emulation (E=1) modes. The M and X flags are always equal to one in the Emulation mode. When an interrupt occurs during the Emulation mode, the Break flag is written to stack memory as bit 4 of the Processor Status Register.

Program Bank Register (PBR)

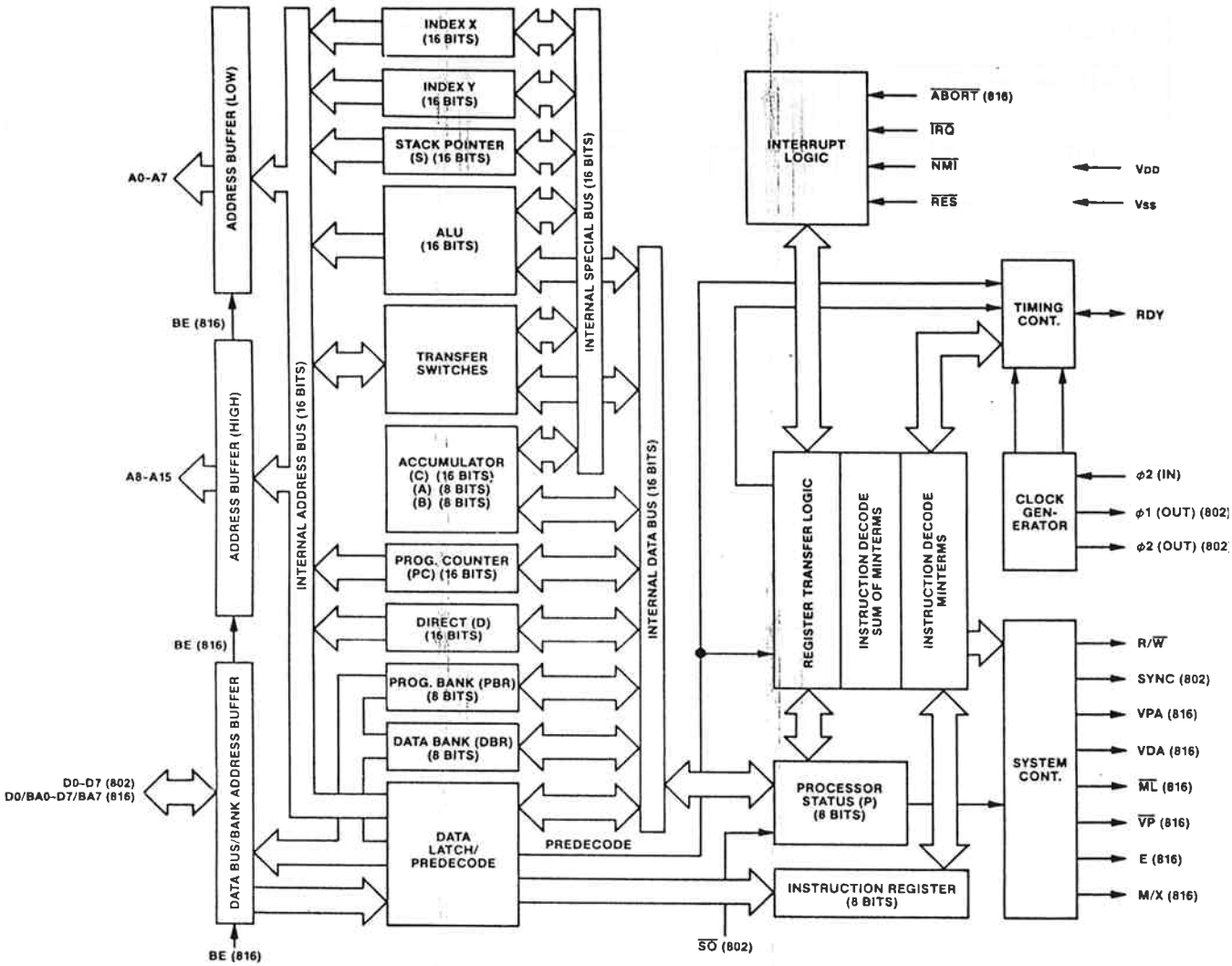
The 8-bit Program Bank Register holds the bank address for all instruction fetches. The 24-bit address consists of the 16-bit instruction effective address and the 8-bit Program Bank address. The register value is multiplexed with the data value and presented on the Data/Address lines during the first half of a program memory read cycle. The Program Bank Register is initialized to zero during Reset. The PHK instruction pushes the PBR register onto the Stack.

Program Counter (PC)

The 16-bit Program Counter Register provides the addresses which are used to step the microprocessor through sequential program instructions. The register is incremented each time an instruction or operand is fetched from program memory.

Stack Pointer (S)

The Stack Pointer is a 16-bit register which is used to indicate the next available location in the stack memory area. It serves as the effective address in stack addressing modes as well as subroutine and interrupt processing. The Stack Pointer allows simple implementation of nested subroutines and multiple-level interrupts. During the Emulation mode, the Stack Pointer high-order byte (SH) is always equal to one. The bank address for all stack operations is Bank zero.



Block Diagram — Internal Architecture

W65C816 Compatibility Issues

	W65C816/802	W65C02	NMOS 6502
1. S (Stack)	Always page 1 (E = 1), 8 bits 16 bits when (E = 0).	Always page 1, 8 bits	Always page 1, 8 bits
2. X (X Index Register)	Indexed page zero always in page 0 (E = 1), Cross page (E = 0).	Always page 0	Always page 0
3. Y (Y Index Register)	Indexed page zero always in page 0 (E = 1), Cross page (E = 0).	Always page 0	Always page 0
4. A (Accumulator)	8 bits (M = 1), 16 bits (M = 0)	8 bits	8 bits
5. P (Flag Registor)	N, V, and Z flags valid in decimal mode. D = 0 after reset or interrupt.	N, V, and Z flags valid in decimal mode. D = 0 after reset and interrupt.	N, V, and Z flags invalid in decimal mode. D = unknown after reset. D not modified after interrupt.
6. Timing A. ABS, X ASL, LSR, ROL, ROR With No Page Crossing	7 cycles	6 cycles	7 cycles
B. Jump Indirect Operand = XXFF	5 cycles	6 cycles	5 cycles and invalid page crossing
C. Branch Across Page	4 cycles (E = 1) 3 cycles (E = 0)	4 cycles	4 cycles
D. Decimal Mode	No additional cycle	Add 1 cycle	No additional cycle
7. BRK Vector	00FFFE,F (E = 1) BRK bit = 0 on stack if $\overline{IRQ}$, $\overline{NMI}$, $\overline{ABORT}$. 00FFE6, 7 (E = 0) X = X on Stack always.	FFFE,F BRK bit = 0 on stack if $\overline{IRQ}$, $\overline{NMI}$.	FFFE,F BRK bit = 0 on stack if $\overline{IRQ}$, $\overline{NMI}$.
8. Interrupt or Break Bank Address	PBR not pushed (E = 1) RTI PBR not pulled (E = 1) PBR pushed (E = 0) RTI PBR pulled (E = 0)	Not available	Not available
9. Memory Lock ($\overline{ML}$)	$\overline{ML}$ = 0 during Read, Modify and Write cycles.	$\overline{ML}$ = 0 during Modify and Write.	Not available
10. Indexed Across Page Boundary (d),y; a,x; a,y	Extra read of invalid address. (Note 1)	Extra read of last instruction fetch.	Extra read of invalid address.
11. RDY Pulled During Write Cycle.	Ignored (E = 1) for W65C802 only. Processor stops (E = 0).	Processor stops	Ignored
12. WAI and STP Instructions.	Available	Available	Not available
13. Unused OP Codes	One reserved OP Code specified as WDM will be used in future systems. The W65C816 performs a no-operation.	No operation	Unknown and some "hang up" processor.
14. Bank Address Handling	PBR = 00 after reset or interrupts.	Not available	Not available
15. R/$\overline{W}$ During Read-Modify-Write Instructions	E = 1, R/$\overline{W}$ = 0 during Modify and Write cycles. E = 0, R/$\overline{W}$ = 0 only during Write cycle.	R/$\overline{W}$ = 0 only during Write cycle	R/$\overline{W}$ = 0 during Modify and Write cycles.
16. Pin 7	W65C802 = SYNC. W65C816 = VPA	SYNC	SYNC
17. COP Instruction Signatures 00-7F user defined Signatures 80-FF reserved	Available	Not available	Not available

Note 1. See Caveat section for additional information.

W65C802 and W65C816 Microprocessor Addressing Modes

The W65C816 is capable of directly addressing 16 MBytes of memory. This address space has special significance within certain addressing modes, as follows:

Reset and Interrupt Vectors

The Reset and Interrupt vectors use the majority of the fixed addresses between 00FFE0 and 00FFFF.

Stack

The Stack may use memory from 000000 to 00FFFF. The effective address of Stack and Stack Relative addressing modes will always be within this range.

Direct

The Direct addressing modes are usually used to store memory registers and pointers. The effective address generated by Direct, Direct,X and Direct,Y addressing modes is always in Bank 0 (000000-00FFFF).

Program Address Space

The Program Bank register is not affected by the Relative, Relative Long, Absolute, Absolute Indirect, and Absolute Indexed Indirect addressing modes or by incrementing the Program Counter from FFFF. The only instructions that affect the Program Bank register are: RTI, RTL, JML, JSL, and JMP Absolute Long. Program code may exceed 64K bytes although code segments may not span bank boundaries.

Data Address Space

The data address space is contiguous throughout the 16 MByte address space. Words, arrays, records, or any data structures may span 64 KByte bank boundaries with no compromise in code efficiency. The following addressing modes generate 24-bit effective addresses:

- Direct Indexed Indirect (d,x)
- Direct Indirect Indexed (d),y
- Direct Indirect (d)
- Direct Indirect Long [d]
- Direct Indirect Long Indexed [d],y
- Absolute a
- Absolute a,x
- Absolute a,y
- Absolute Long al
- Absolute Long Indexed al,x
- Stack Relative Indirect Indexed (d,s),y

The following addressing mode descriptions provide additional detail as to how effective addresses are calculated.

Twenty-four addressing modes are available for use with the W65C802 and W65C816 microprocessors. The "long" addressing modes may be used with the W65C802; however, the high byte of the address is not available to the hardware. Detailed descriptions of the 24 addressing modes are as follows:

1. Immediate Addressing—#

The operand is the second byte (second and third bytes when in the 16-bit mode) of the instruction.

2. Absolute—a

With Absolute addressing the second and third bytes of the instruction form the low-order 16 bits of the effective address. The Data Bank Register contains the high-order 8 bits of the operand address.

Instruction:	opcode	addrl	addrh
Operand Address:	DBR	addrh	addrl

3. Absolute Long—al

The second, third, and fourth byte of the instruction form the 24-bit effective address.

Instruction:	opcode	addrl	addrh	baddr
Operand Address:	baddr	addrh	addrl	

4. Direct—d

The second byte of the instruction is added to the Direct Register (D) to form the effective address. An additional cycle is required when the Direct Register is not page aligned (DL not equal 0). The Bank register is always 0.

Instruction:	opcode	offset
		Direct Register
	+	offset
Operand Address:	00	effective address

5. Accumulator—A

This form of addressing always uses a single byte instruction. The operand is the Accumulator.

6. Implied—i

Implied addressing uses a single byte instruction. The operand is implicitly defined by the instruction.

7. Direct Indirect Indexed—(d),y

This address mode is often referred to as Indirect,Y. The second byte of the instruction is added to the Direct Register (D). The 16-bit contents of this memory location is then combined with the Data Bank register to form a 24-bit base address. The Y Index Register is added to the base address to form the effective address.

Instruction:	opcode	offset
		Direct Register
	+	offset
	00	direct address
then:	00	(direct address)
	+ DBR	
		base address
	+	Y Reg
Operand Address:		effective address

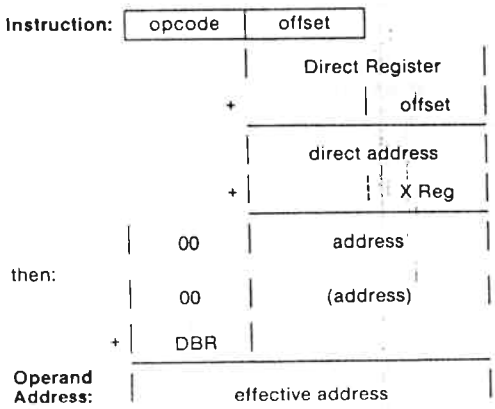
8. Direct Indirect Long Indexed—[d],y

With this addressing mode, the 24-bit base address is pointed to by the sum of the second byte of the instruction and the Direct Register. The effective address is this 24-bit base address plus the Y Index Register.

Instruction:	opcode	offset
		Direct Register
	+	offset
	00	direct address
then:		(direct address)
	+	Y Reg
Operand Address:		effective address

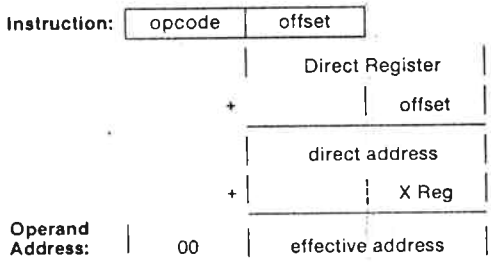
9. Direct Indexed Indirect—(d,x)

This address mode is often referred to as Indirect,X. The second byte of the instruction is added to the sum of the Direct Register and the X Index Register. The result points to the low-order 16 bits of the effective address. The Data Bank Register contains the high-order 8 bits of the effective address.



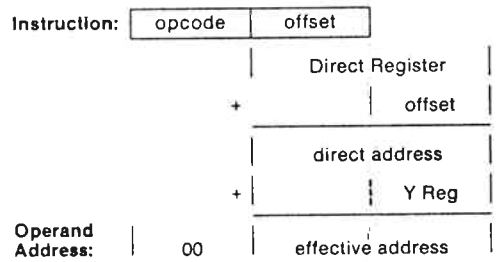
10. Direct Indexed With X—d,x

The second byte of the instruction is added to the sum of the Direct Register and the X Index Register to form the 16-bit effective address. The operand is always in Bank 0.



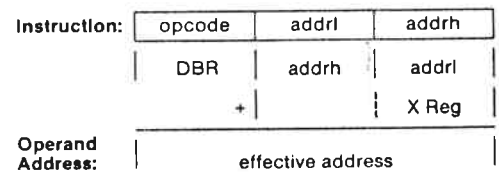
11. Direct Indexed With Y—d,y

The second byte of the instruction is added to the sum of the Direct Register and the Y Index Register to form the 16-bit effective address. The operand is always in Bank 0.



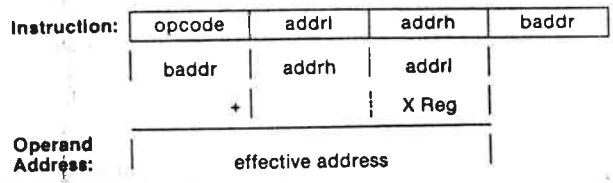
12. Absolute Indexed With X—a,x

The second and third bytes of the instruction are added to the X Index Register to form the low-order 16 bits of the effective address. The Data Bank Register contains the high-order 8 bits of the effective address.



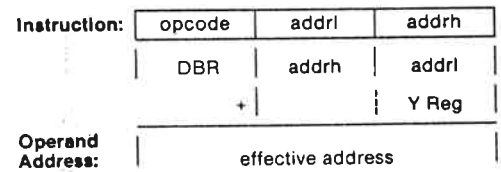
13. Absolute Long Indexed With X—al,x

The second, third and fourth bytes of the instruction form a 24-bit base address. The effective address is the sum of this 24-bit address and the X Index Register.



14. Absolute Indexed With Y—a,y

The second and third bytes of the instruction are added to the Y Index Register to form the low-order 16 bits of the effective address. The Data Bank Register contains the high-order 8 bits of the effective address.



15. Program Counter Relative—r

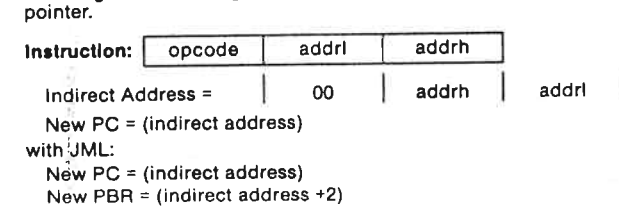
This address mode, referred to as Relative Addressing, is used only with the Branch instructions. If the condition being tested is met, the second byte of the instruction is added to the Program Counter, which has been updated to point to the opcode of the next instruction. The offset is a signed 8-bit quantity in the range from -128 to 127. The Program Bank Register is not affected.

16. Program Counter Relative Long—rl

This address mode, referred to as Relative Long Addressing, is used only with the Unconditional Branch Long instruction (BRL) and the Push Effective Relative instruction (PER). The second and third bytes of the instruction are added to the Program Counter, which has been updated to point to the opcode of the next instruction. With the branch instruction, the Program Counter is loaded with the result. With the Push Effective Relative instruction, the result is stored on the stack. The offset is a signed 16-bit quantity in the range from -32768 to 32767. The Program Bank Register is not affected.

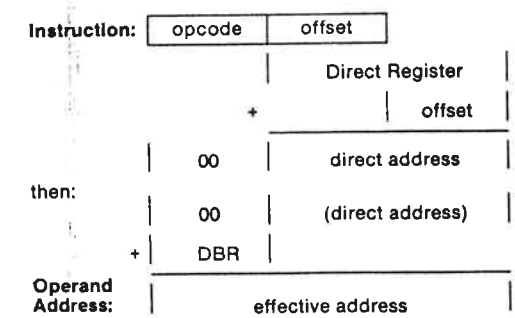
17. Absolute Indirect—(a)

The second and third bytes of the instruction form an address to a pointer in Bank 0. The Program Counter is loaded with the first and second bytes at this pointer. With the Jump Long (JML) instruction, the Program Bank Register is loaded with the third byte of the pointer.



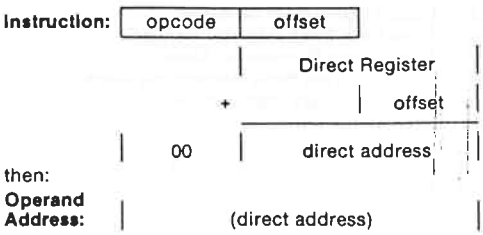
18. Direct Indirect—(d)

The second byte of the instruction is added to the Direct Register to form a pointer to the low-order 16 bits of the effective address. The Data Bank Register contains the high-order 8 bits of the effective address.



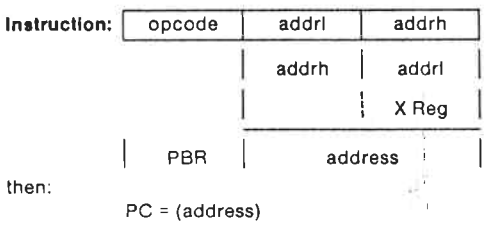
19. Direct Indirect Long—[d]

The second byte of the instruction is added to the Direct Register to form a pointer to the 24-bit effective address.



20. Absolute Indexed Indirect—(a,x)

The second and third bytes of the instruction are added to the X Index Register to form a 16-bit pointer in Bank 0. The contents of this pointer are loaded in the Program Counter. The Program Bank Register is not changed.

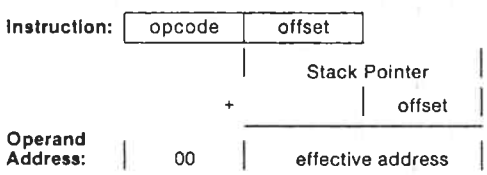


21. Stack—s

Stack addressing refers to all instructions that push or pull data from the stack, such as Push, Pull, Jump to Subroutine, Return from Subroutine, Interrupts, and Return from Interrupt. The bank address is always 0. Interrupt Vectors are always fetched from Bank 0.

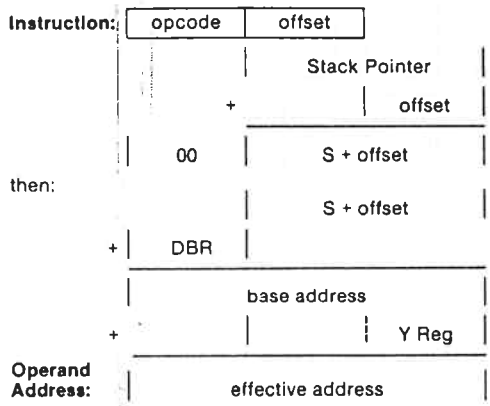
22. Stack Relative—d,s

The low-order 16 bits of the effective address is formed from the sum of the second byte of the instruction and the Stack Pointer. The high-order 8 bits of the effective address is always zero. The relative offset is an unsigned 8-bit quantity in the range of 0 to 255.



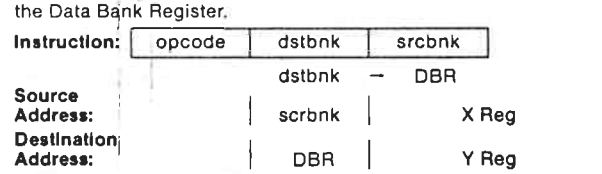
23. Stack Relative Indirect Indexed—(d,s),y

The second byte of the instruction is added to the Stack Pointer to form a pointer to the low-order 16-bit base address in Bank 0. The Data Bank Register contains the high-order 8 bits of the base address. The effective address is the sum of the 24-bit base address and the Y Index Register.



24. Block Source Bank, Destination Bank—xyc

This addressing mode is used by the Block Move instructions. The second byte of the instruction contains the high-order 8 bits of the destination address. The Y index Register contains the low-order 16 bits of the destination address. The third byte of the instruction contains the high-order 8 bits of the source address. The X Index Register contains the low-order 16 bits of the source address. The C Accumulator contains one less than the number of bytes to move. The second byte of the block move instructions is also loaded into the Data Bank Register.



Increment (MVN) or decrement (MVP) X and Y.
Decrement C (if greater than zero), then PC+3 → PC.

W65C802 and W65C816 Instruction Set—Alphabetical Sequence

ADC	Add Memory to Accumulator with Carry
AND	"AND" Memory with Accumulator
ASL	Shift One Bit Left, Memory or Accumulator
BCC	Branch on Carry Clear (Pc = 0)
BCS	Branch on Carry Set (Pc = 1)
BEQ	Branch if Equal (Pz = 1)
BIT	Bit Test
BMI	Branch if Result Minus (PN = 1)
BNE	Branch if Not Equal (Pz = 0)
BPL	Branch if Result Plus (PN = 0)
BRA	Branch Always
BRK	Force Break
BRL	Branch Always Long
BVC	Branch on Overflow Clear (Pv = 0)
BVS	Branch on Overflow Set (Pv = 1)
CLC	Clear Carry Flag
CLD	Clear Decimal Mode
CLI	Clear Interrupt Disable Bit
CLV	Clear Overflow Flag
CMP	Compare Memory and Accumulator
COP	Coprocessor
CPX	Compare Memory and Index X
CPY	Compare Memory and Index Y
DEC	Decrement Memory or Accumulator by One
DEX	Decrement Index X by One
DEY	Decrement Index Y by One
EOR	"Exclusive OR" Memory with Accumulator
INC	Increment Memory or Accumulator by One
INX	Increment Index X by One
INY	Increment Index Y by One
JML	Jump Long
JMP	Jump to New Location
JSL	Jump Subroutine Long
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 One Bit Right (Memory or Accumulator)
MVN	Block Move Negative
MVP	Block Move Positive
NOP	No Operation
ORA	"OR" Memory with Accumulator
PEA	Push Effective Absolute Address on Stack (or Push Immediate Data on Stack)
PEI	Push Effective Indirect Address on Stack (or Push Direct Data on Stack)
PER	Push Effective Program Counter Relative Address on Stack
PHA	Push Accumulator on Stack
PHB	Push Data Bank Register on Stack
PHD	Push Direct Register on Stack
PHK	Push Program Bank Register on Stack
PHP	Push Processor Status on Stack
PHX	Push Index X on Stack
PHY	Push Index Y on Stack
PLA	Pull Accumulator from Stack
PLB	Pull Data Bank Register from Stack
PLD	Pull Direct Register from Stack
PLP	Pull Processor Status from Stack
PLX	Pull Index X from Stack
PLY	Pull Index Y form Stack
REP	Reset Status Bits
ROL	Rotate One Bit Left (Memory or Accumulator)
ROR	Rotate One Bit Right (Memory or Accumulator)
RTI	Return from Interrupt
RTL	Return from Subroutine Long
RTS	Return from Subroutine
SBC	Subtract Memory from Accumulator with Borrow
SEC	Set Carry Flag
SED	Set Decimal Mode
SEI	Set Interrupt Disable Status
SEP	Set Processor Status Bite
STA	Store Accumulator in Memory
STP	Stop the Clock
STX	Store Index X in Memory
STY	Store Index Y in Memory
STZ	Store Zero in Memory
TAX	Transfer Accumulator to Index X
TAY	Transfer Accumulator to Index Y
TCD	Transfer C Accumulator to Direct Register
TCS	Transfer C Accumulator to Stack Pointer Register
TDC	Transfer Direct Register to C Accumulator
TRB	Test and Reset Bit
TSB	Test and Set Bit
TSC	Transfer Stack Pointer Register to C Accumulator
TSX	Transfer Stack Pointer Register to Index X
TXA	Transfer Index X to Accumulator
TXS	Transfer Index X to Stack Pointer Register
TXY	Transfer Index X to Index Y
TYA	Transfer Index Y to Accumulator
TYX	Transfer Index Y to Index X
WAI	Wait for Interrupt
WDM	Reserved for Future Use
XBA	Exchange B and A Accumulator
XCE	Exchange Carry and Emulation Bits

For alternate mnemonics, see Table 7.

Vector Locations

E = 1		
OOFFFE,F	—$\overline{\text{IRQ}}$/BRK	Hardware/Software
OOFFFC,D	—$\overline{\text{RESET}}$	Hardware
OOFFFA,B	—$\overline{\text{NMI}}$	Hardware
OOFFF8,9	—$\overline{\text{ABORT}}$	Hardware
OOFFF6,7	—(Reserved)	
OOFFF4,5	—COP	Software

E = 0		
OOFFEE,F	—$\overline{\text{IRQ}}$	Hardware
OOFFEC,D	—(Reserved)	
OOFFEA,B	—$\overline{\text{NMI}}$	Hardware
OOFFE8,9	—$\overline{\text{ABORT}}$	Hardware
OOFFE6,7	—BRK	Software
OOFFE4,5	—COP	Software

The VP output is low during the two cycles used for vector location access.
When an interrupt is executed, D = 0 and I = 1 in Status Register P.

Opcode Matrix

MSD	LSD 0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	MSD
0	BRK s 2 8	ORA (d,x) 2 6	COP s 2★8	ORA d,s 2★4	TSB d 2●5	ORA d 2 3	ASL d 2 5	ORA [d] 2★6	PHP s 1 3	ORA # 2 2	ASL A 1 2	PHD s 1★4	TSB a 3●6	ORA a 3 4	ASL a 3 6	ORA al 4★5	0
1	BPL r 2 2	ORA (d),y 2 5	ORA (d) 2●5	ORA (d,s),y 2★7	TRB d 2●5	ORA d,x 2 4	ASL d,x 2 6	ORA [d],y 2★6	CLC i 1 2	ORA a,y 3 4	INC A 1●2	TCS i 1★2	TRB a 3●6	ORA a,x 3 4	ASL a,x 3 7	ORA al,x 4★5	1
2	JSR a 3 6	AND (d,x) 2 6	JSL al 4★8	AND d,s 2★4	BIT d 2 3	AND d 2 3	ROL d 2 5	AND [d] 2★6	PLP s 1 4	AND # 2 2	ROL A 1 2	PLD s 1★5	BIT a 3 4	AND a 3 4	ROL a 3 6	AND al 4★5	2
3	BMI r 2 2	AND (d),y 2 5	AND (d) 2●5	AND (d,s),y 2★7	BIT d,x 2●4	AND d,x 2 4	ROL d,x 2 6	AND [d],y 2★6	SEC i 1 2	AND a,y 3 4	DEC A 1●2	TSC i 1★2	BIT a,x 3●4	AND a,x 3 4	ROL a,x 3 7	AND al,x 4★5	3
4	RTI s 1 7	EOR (d,x) 2 6	WDM 2★2	EOR d,s 2★4	MVP xyc 3★7	EOR d 2 3	LSR d 2 5	EOR [d] 2★6	PHA s 1 3	EOR # 2 2	LSR A 1 2	PHK s 1★3	JMP a 3 3	EOR a 3 4	LSR a 3 6	EOR al 4★5	4
5	BVC r 2 2	EOR (d),y 2 5	EOR (d) 2●5	EOR (d,s),y 2★7	MVN xyc 3★7	EOR d,x 2 4	LSR d,x 2 6	EOR [d],y 2★6	CLI i 1 2	EOR a,y 3 4	PHY s 1●3	TCD i 1★2	JMP al 4★4	EOR a,x 3 4	LSR a,x 3 7	EOR al,x 4★5	5
6	RTS s 1 6	ADC (d,x) 2 6	PER s 3★6	ADC d,s 2★4	STZ d 2●3	ADC d 2 3	ROR d 2 5	ADC [d] 2★6	PLA s 1 4	ADC # 2 2	ROR A 1 2	RTL s 1★6	JMP (a) 3 5	ADC a 3 4	ROR a 3 6	ADC al 4★5	6
7	BVS r 2 2	ADC (d),y 2 5	ADC (d) 2●5	ADC (d,s),y 2★7	STZ d,x 2●4	ADC d,x 2 4	ROR d,x 2 6	ADC [d],y 2★6	SEI i 1 2	ADC a,y 3 4	PLY s 1●4	TDC i 1★2	JMP (a,x) 3●6	ADC a,x 3 4	ROR a,x 3 7	ADC al,x 4★5	7
8	BRA r 2●2	STA (d,x) 2 6	BRL rl 3★3	STA d,s 2★4	STY d 2 3	STA d 2 3	STX d 2 3	STA [d] 2★6	DEY i 1 2	BIT # 2●2	TXA i 1 2	PHB s 1★3	STY a 3 4	STA a 3 4	STX a 3 4	STA al 4★5	8
9	BCC r 2 2	STA (d),y 2 6	STA (d) 2●5	STA (d,s),y 2★7	STY d,x 2 4	STA d,x 2 4	STX d,y 2 4	STA [d],y 2★6	TYA i 1 2	STA a,y 3 5	TXS i 1 2	TXY i 1★2	STZ a 3●4	STA a,x 3 5	STZ a,x 3●5	STA al,x 4★5	9
A	LDY # 2 2	LDA (d,x) 2 6	LDX # 2 2	LDA d,s 2★4	LDY d 2 3	LDA d 2 3	LDX d 2 3	LDA [d] 2★6	TAY i 1 2	LDA # 2 2	TAX i 1 2	PLB s 1★4	LDY a 3 4	LDA a 3 4	LDX a 3 4	LDA al 4★5	A
B	BCS r 2 2	LDA (d),y 2 5	LDA (d) 2●5	LDA (d,s),y 2★7	LDY d,x 2 4	LDA d,x 2 4	LDX d,y 2 4	LDA [d],y 2★6	CLV i 1 2	LDA a,y 3 4	TSX i 1 2	TYX i 1★2	LDY a,x 3 4	LDA a,x 3 4	LDX a,y 3 4	LDA al,x 4★5	B
C	CPY # 2 2	CMP (d,x) 2 6	REP # 2★3	CMP d,s 2★4	CPY d 2 3	CMP d 2 3	DEC d 2 5	CMP [d] 2★6	INY i 1 2	CMP # 2 2	DEX i 1 2	WAI i 1●3	CPY a 3 4	CMP a 3 4	DEC a 3 6	CMP al 4★5	C
D	BNE r 2 2	CMP (d),y 2 5	CMP (d) 2●5	CMP (d,s),y 2★7	PEI s 2★6	CMP d,x 2 4	DEC d,x 2 6	CMP [d],y 2★6	CLD i 1 2	CMP a,y 3 4	PHX s 1●3	STP i 1●3	JML (a) 3★6	CMP a,x 3 4	DEC a,x 3 7	CMP al,x 4★5	D
E	CPX # 2 2	SBC (d,x) 2 6	SEP # 2★3	SBC d,s 2★4	CPX d 2 3	SBC d 2 3	INC d 2 5	SBC [d] 2★6	INX i 1 2	SBC # 2 2	NOP i 1 2	XBA i 1★3	CPX a 3 4	SBC a 3 4	INC a 3 6	SBC al 4★5	E
F	BEQ r 2 2	SBC (d),y 2 5	SBC (d) 2●5	SBC (d,s),y 2★7	PEA s 3★5	SBC d,x 2 4	INC d,x 2 6	SBC [d],y 2★6	SED i 1 2	SBC a,y 3 4	PLX s 1●4	XCE i 1★2	JSR (a,x) 3★6	SBC a,x 3 4	INC a,x 3 7	SBC al,x 4★5	F
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

symbol	addressing mode	symbol	addressing mode
#	immediate	[d]	direct indirect long
A	accumulator	[d],y	direct indirect long indexed
r	program counter relative	a	absolute
rl	program counter relative long	a,x	absolute indexed (with x)
i	implied	a,y	absolute indexed (with y)
s	stack	al	absolute long
d	direct	al,x	absolute long indexed
d,x	direct indexed (with x)	d,s	stack relative
d,y	direct indexed (with y)	(d,s),y	stack relative indirect indexed
(d)	direct indirect	(a)	absolute indirect
(d,x)	direct indexed indirect	(a,x)	absolute indexed indirect
(d),y	direct indirect indexed	xyc	block move

Op Code Matrix Legend

INSTRUCTION MNEMONIC

BASE NO. BYTES

★ = New W65C816/802 Opcodes
● = New W65C02 Opcodes
Blank = NMOS 6502 Opcodes

ADDRESSING MODE

BASE NO. CYCLES

Operation, Operation Codes, and Status Register

MNE-MONIC	OPERATION	#	a	al	d	A	l	(d),y	[d],y	(d,x)	d,x	d,y	a,x	al,x	a,y	r	rl	(a)	(d)	[d]	(a,x)	s	d,s	(d,s),y	xyc	PROCESSOR STATUS CODE 7 6 5 4 3 2 1 0	MNE-MONIC
																										N V M X D I Z C E= 0	
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	N V 1 B D I Z C E= 1	
ADC	A + M + C → A	69	6D	6F	65			71	77	61	75		7D	7F	79				72	67			63	73		N V . . . . Z C	ADC
AND	A∧M → A	29	2D	2F	25			31	37	21	35		3D	3F	39				32	27			23	33		N . . . . . Z .	AND
ASL	C ← [15/7 0] ← 0		0E		06	0A					16		1E													N . . . . . Z C	ASL
BCC	BRANCH IF C = 0															90										. . . .	BCC
BCS	BRANCH IF C = 1															B0										. . . .	BCS
BEQ	BRANCH IF Z = 1															F0										. . . .	BEQ
BIT	A∧M (NOTE 1)	89	2C		24						34		3C													M_7 M_6 . . . . Z .	BIT
BMI	BRANCH IF N = 1															30										. . . .	BMI
BNE	BRANCH IF Z = 0															D0										. . . .	BNE
BPL	BRANCH IF N = 0															10										. . . .	BPL
BRA	BRANCH ALWAYS															80										. . . .	• BRA
BRK	BREAK (NOTE 2)																					00				. . . • 0 I . .	★ BRK
BRL	BRANCH LONG ALWAYS																82									. . . .	BRL
BVC	BRANCH IF V = 0															50										. . . .	BVC
BVS	BRANCH IF V = 1															70										. . . .	BVS
CLC	0 → C						18																			 0	CLC
CLD	0 → D						D8																			. . . . 0 . . .	CLD
CLI	0 → I						58																			. . . . . 0 . .	CLI
CLV	0 → V						B8																			. . . .	CLV
CMP	A-M	C9	CD	CF	C5			D1	D7	C1	D5		DD	DF	D9				D2	C7			C3	D3		N . . . . . Z C	CMP
COP	CO-PROCESSOR																					02				. . . . 0 I . .	★ COP
CPX	X-M	E0	EC		E4																					N . . . . . Z C	CPX
CPY	Y-M	C0	CC		C4						D6		DE													N . . . . . Z C	CPY
DEC	DECREMENT		CE		C6	3A																				N . . . . . Z .	DEC
DEX	X - 1 → X						CA																			N . . . . . Z .	DEX
DEY	Y - 1 → Y						88																			N . . . . . Z .	DEY
EOR	AVM → A	49	4D	4F	45	1A		51	57	41	55		5D	5F	59				52	47			43	53		N . . . . . Z .	EOR
INC	INCREMENTS		EE		E6						F6		FE													N . . . . . Z .	INC
INX	X + 1 → X						E8																			N . . . . . Z .	INX
INY	Y + 1 → Y						C8																			N . . . . . Z .	INY
JML	JUMP LONG TO NEW LOC.																	DC								. . . .	★ JML
JMP	JUMP TO NEW LOC.		4C	5C														6C			7C					. . . .	JMP
JSL	JUMP LONG TO SUB.			22																						. . . .	JSL
JSR	JUMP TO SUB.		20																		FC					. . . .	JSR
LDA	M → A	A9	AD	AF	A5			B1	B7	A1	B5		BD	BF	B9				B2	A7		A3	B3			N . . . . . Z .	LDA
LDX	M → X	A2	AE		A6							B6			BE											N . . . . . Z .	LDX
LDY	M → Y	A0	AC		A4						B4		BC													N . . . . . Z .	LDY
LSR	0 → [15/7 0] → C		4E		46	4A					56		5E													0 . . . . . Z C	LSR
MVN	M → M BACKWARD																								54	. . . .	★ MVN
MVP	M → M FORWARD																								44	. . . .	★ MVP
NOP	NO OPERATION						EA																			. . . .	NOP
ORA	AVM → A	09	0D	0F	05			11	17	01	15		1D	1F	19				12	07			03	13		N . . . . . Z .	ORA
PEA	Mpc + 1, Mpc + 2 → Ms - 1, Ms S - 2 → S																					F4				. . . .	★ PEA
PEI	M(d), M(d + 1) → Ms - 1, Ms S - 2 → S																					D4				. . . .	★ PEI
PER	Mpc + rl, Mpc + rl + 1 → Ms - 1, Ms S - 2 → S																					62				. . . .	★ PER
PHA	A → Ms, S - 1 → S																					48				. . . .	PHA
PHB	DBR → Ms, S - 1 → S																					8B				. . . .	PHB
PHD	D → Ms, Ms - 1, S - 2 → S																					0B				. . . .	★ PHD
PHK	PBR → Ms, S - 1 → S																					4B				. . . .	★ PHK
PHP	P → Ms, S - 1 → S																					08				. . . .	PHP
PHX	X → Ms, S - 1 → S																					DA				. . . .	• PHX
PHY	Y → Ms, S - 1 → S																					5A				. . . .	• PHY
PLA	S + 1 → S, Ms → A																					68				N . . . . . Z .	PLA
PLB	S + 1 → S, Ms → DBR																					AB				N . . . . . Z .	★ PLB
PLD	S + 2 → S, Ms - 1, Ms → D																					2B				N . . . . . Z .	★ PLD
PLP	S + 1 → S, Ms → P																					28				N V M X D I Z C	PLP
PLX	S + 1 → S, Ms → X																					FA				N . . . . . Z .	PLX
PLY	S + 1 → S, Ms → Y																					7A				N . . . . . Z .	PLY
REP	MAP → P	C2																								N V M X D I Z C	★ REP
ROL	[15/7 0] ← C		2E		26	2A					36		3E													N . . . . . Z C	ROL
ROR	C → [15/7 0]		6E		66	6A					76		7E													N . . . . . Z C	ROR
RTI	RTRN FROM INT.																					40				N V M X D I Z C	RTI
RTL	RTRN FROM SUB. LONG																					6B				. . . .	★ RTL
RTS	RTRN SUBROUTINE																					60				. . . .	RTS
SBC	A - M - C̄ → A	E9	ED	EF	E5			F1	F7	E1	F5		FD	FF	F9				F2	E7			E3	F3		N V . . . . Z C	SBC
SEC	1 → C						38																			 1	SEC
SED	1 → D						F8																			. . . . 1 . . .	SED
SEI	1 → I						78																			. . . . . 1 . .	SEI
SEP	MVP → P	E2																								N V M X D I Z C	★ SEP
STA	A → M		8D	8F	85			91	97	81	95		9D	9F	99				92	87			83	93		. . . .	STA
STP	STOP (1 → φ2)						DB																			. . . .	• STP
STX	X → M		8E		86							96														. . . .	STX
STY	Y → M		8C		84						94															. . . .	STY
STZ	00 → M		9C		64						74		9E													. . . .	• STZ
TAX	A → X						AA																			N . . . . . Z .	TAX
TAY	A → Y						A8																			N . . . . . Z .	TAY
TCD	C → D						5B																			N . . . . . Z .	★ TCD
TCS	C → S						1B																			. . . .	★ TCS
TDC	D → C						7B																			N . . . . . Z .	★ TDC
TRB			1C		14																					. . . . . Z .	• TRB
TSB	AVM → M		0C		04																					. . . . . Z .	• TSB
TSC	S → C						3B																			N . . . . . Z .	★ TSC
TSX	S → X						BA																			N . . . . . Z .	TSX
TXA	X → A						8A																			N . . . . . Z .	TXA
TXS	X → S						9A																			. . . .	TXS
TXY	X → Y						9B																			N . . . . . Z .	★ TXY
TYA	Y → A						98																			N . . . . . Z .	TYA
TYX	Y → X						BB																			N . . . . . Z .	★ TYX
WAI	0 → RDY						CB																			. . . .	• WAI
WDM	NO OPERATION (RESERVED)						42																			. . . .	★ WDM
XBA	B ⟷ A						EB																			N . . . . . Z .	★ XBA
XCE	C ⟷ E						FB																			. . . . . . E	★ XCE

Notes:
1. Bit immediate N and V flags not affected. When M = 0, M_{15} → N and M_{14} → V.
2. Break Bit (B) in Status register indicates hardware or software break.
3. ★ = New W65C816/802 Instructions
   • = New W65C02 Instructions
   Blank = NMOS 6502

\+ Add
\- Subtract
∧ AND
V OR
⊻ Exclusive OR

Detailed Instruction Operation

ADDRESS MODE		CYCLE	VP	ML	VDA	VPA	ADDRESS BUS	DATA BUS	R/W
1. Immediate # (LDY,CPY,CPX,LDX,ORA, AND,EOR,ADC,BIT,LDA, CMP,SBC REP,SEP) (14 Op Codes) (2 and 3 bytes) (2 and 3 cycles)	(1)(8)	1.	1	1	1	1	PBR,PC	Op Code	1
		2.	1	1	0	1	PBR,PC+1	IDL	1
		2a.	1	1	0	1	PBR,PC+2	IDH	1
2a. Absolute a (BIT,STY,STZ,LDY, CPY,CPX,STX,LDX, ORA,AND,EOR,ADC, STA,LDA,CMP,SBC) (18 Op Codes) (3 bytes) (4 and 5 cycles)		1.	1	1	1	1	PBR,PC	Op Code	1
		2.	1	1	0	1	PBR,PC+1	AAL	1
		3.	1	1	0	1	PBR,PC+2	AAH	1
		4.	1	1	1	0	DBR,AA	Data Low	1/0
	(1)	4a.	1	1	1	0	DBR,AA+1	Data High	1/0
2b. Absolute (R-M-W) a (ASL,ROL,LSR,ROR DEC,INC,TSB,TRB) (6 Op Codes) (3 bytes) (6 and 8 cycles)		1.	1	1	1	1	PBR,PC	Op Code	1
		2.	1	1	0	1	PBR,PC+1	AAL	1
		3.	1	1	0	1	PBR,PC+2	AAH	1
		4.	1	0	1	0	DBR,AA	Data Low	1
	(1)	4a.	1	0	1	0	DBR,AA+1	Data High	1
	(3)	5.	1	0	0	0	DBR,AA+1	IO	1
	(1)	6a.	1	0	1	0	DBR,AA+1	Data High	0
		6.	1	0	1	0	DBR,AA	Data Low	0
2c. Absolute (JUMP) a (JMP)(4C) (1 Op Code) (3 bytes) (3 cycles)		1.	1	1	1	1	PBR,PC	Op Code	1
		2.	1	1	0	1	PBR,PC+1	NEW PCL	1
		3.	1	1	0	1	PBR,PC+2	NEW PCH	1
		1.	1	1	1	1	PBR, NEW PC	Op Code	1
2d. Absolute (Jump to subroutine) a (JSR) (1 Op Code) (3 bytes) (6 cycles) (different order from N6502)		1.	1	1	1	1	PBR,PC	Op Code	1
		2.	1	1	0	1	PBR,PC+1	NEW PCL	1
		3.	1	1	0	1	PBR,PC+2	NEW PCH	1
		4.	1	1	0	0	PBR,PC+2	IO	1
		5.	1	1	1	0	0,S	PCH	0
		6.	1	1	1	0	0,S-1	PCL	0
		1.	1	1	1	1	PBR,NEW PC	Next Op Code	1
*3a. Absolute Long al (ORA,AND,EOR,ADC STA,LDA,CMP,SBC) (8 Op Codes) (4 bytes) (5 and 6 cycles)		1.	1	1	1	1	PBR,PC	Op Code	1
		2.	1	1	0	1	PBR,PC+1	AAL	1
		3.	1	1	0	1	PBR,PC+2	AAH	1
		4.	1	1	0	1	PBR,PC+3	AAB	1
		5.	1	1	1	0	AAB,AA	Data Low	1/0
	(1)	5a.	1	1	1	0	AAB,AA+1	Data High	1/0
*3b. Absolute Long (JUMP) al (JMP) (1 Op Code) (4 bytes) (4 cycles)		1.	1	1	1	1	PBR,PC	Op Code	1
		2.	1	1	0	1	PBR,PC+1	NEW PCL	1
		3.	1	1	0	1	PBR,PC+2	NEW PCH	1
		4.	1	1	0	1	PBR,PC+3	NEW BR	1
		1.	1	1	1	1	NEW PBR,PC	Op Code	1
*3c. Absolute Long (Jump to Subroutine Long) al (JSL) (1 Op Code) (4 bytes) (7 cycles)		1.	1	1	1	1	PBR,PC	Op Code	1
		2.	1	1	0	1	PBR,PC+1	NEW PCL	1
		3.	1	1	0	1	PBR,PC+2	NEW PCH	1
		4.	1	1	1	0	0,S	PBR	0
		5.	1	1	0	0	0,S	IO	1
		6.	1	1	0	1	PBR,PC+3	NEW PBR	1
		7.	1	1	1	0	0,S-1	PCH	0
		8.	1	1	1	0	0,S-2	PCL	0
		1.	1	1	1	1	NEW PBR,PC	Next Op Code	1
4a. Direct d (BIT,STZ,STY,LDY, CPY,CPX,STX,LDX, ORA,AND,EOR,ADC, STA,LDA,CMP,SBC) (18 Op Codes) (2 bytes) (3,4 and 5 cycles)		1.	1	1	1	1	PBR,PC	Op Code	1
		2.	1	1	0	1	PBR,PC+1	DO	1
	(2)	2a.	1	1	0	0	PBR,PC+1	IO	1
		3.	1	1	1	0	0,D+DO	Data Low	1/0
	(1)	3a.	1	1	1	0	0,D+DO+1	Data High	1/0
4b. Direct (R-M-W) d (ASL,ROL,LSR,ROR DEC,INC,TSB,TRB) (6 Op Codes) (2 bytes) (5,6,7 and 8 cycles)		1.	1	1	1	1	PBR,PC	Op Code	1
		2.	1	1	0	1	PBR,PC+1	DO	1
	(2)	2a.	1	1	0	0	PBR,PC+1	IO	1
		3.	1	0	1	0	0,D+DO	Data Low	1
	(1)	3a.	1	0	1	0	0,D+DO+1	Data High	1
	(3)	4.	1	0	0	0	0,D+DO+1	IO	1
	(1)	5a.	1	0	1	0	0,D+DO+1	Data High	0
		5.	1	0	1	0	0,D+DO	Data Low	0
5. Accumulator A (ASL,INC,ROL,DEC,LSR,ROR) (6 Op Codes) (1 byte) (2 cycles)		1.	1	1	1	1	PBR,PC	Op Code	1
		2.	1	1	0	0	PBR,PC+1	IO	1
6a. Implied i (DEY, INY, INX, DEX, NOP, XCE, TYA, TAY,TXA, TXS, TAX,TSX,TCS,TSC,TCD, TDC,TXY,TYX,CLC,SEC, CLI,SEI,CLV,CLD,SED) (25 Op Codes) (1 byte) (2 cycles)		1.	1	1	1	1	PBR,PC	Op Code	1
		2.	1	1	0	0	PBR,PC+1	IO	1
*6b. Implied i (XBA) (1 Op Code) (1 byte) (3 cycles)		1.	1	1	1	1	PBR,PC	Op Code	1
		2.	1	1	0	0	PBR,PC+1	IO	1
		3.	1	1	0	0	PBR,PC+1	IO	1
							RDY		
• 6c. Wait For Interrupt (WAI) (1 Op Code) (1 byte) (3 cycles)		1.	1	1	1	1	1	PBR,PC	Op Code	1
	(9)	2.	1	1	0	0	1	PBR,PC+1	IO	1
		3.	1	1	0	0	0	PBR,PC+1	IO	1
	IRQ,NMI	1.	1	1	1	1	1	PBR,PC+1	IRQ(BRK)	1
• 6d. Stop-The-Clock (STP) (1 Op Code) (1 byte) (3 cycles)		1.	1	1	1	1	1	PBR,PC	Op Code	1
		2.	1	1	0	0	1	PBR,PC+1	IO	1
	RES=1	3.	1	1	0	0	1	PBR,PC+1	IO	1
	RES=0	1c.	1	1	0	0	1	PBR,PC+1	RES(BRK)	1
	RES=0	1b.	1	1	0	0	1	PBR,PC+1	RES(BRK)	1
	RES=1	1a.	1	1	0	0	1	PBR,PC+1	RES(BRK)	1
See 21a Stack (Hardware stack)		1.	1	1	1	1	1	PBR,PC+1	BEGIN	1

ADDRESS MODE		CYCLE	VP	ML	VDA	VPA	ADDRESS BUS	DATA BUS	R/W
7. Direct Indirect Indexed (d),y (ORA,AND,EOR,ADC, STA,LDA,CMP,SBC) (8 Op Codes) (2 bytes) (5,6,7 and 8 cycles)		1.	1	1	1	1	PBR,PC	Op Code	1
		2.	1	1	0	1	PBR,PC+1	DO	1
	(2)	2a.	1	1	0	0	PBR,PC+1	IO	1
		3.	1	1	1	0	0,D+DO	AAL	1
		4.	1	1	1	0	0,D+DO+1	AAH	1
	(4)	4a.	1	1	0	0	DBR,AAH,AAL+Y	L IO	1
		5.	1	1	1	0	DBR,AA+Y	Data Low	1/0
	(1)	5a.	1	1	1	0	DBR,AA+Y+1	Data High	1/0
8. Direct Indirect Indexed Long [d],y (ORA,AND,EOR,ADC, STA,LDA,CMP,SBC) (8 Op Codes) (2 bytes) (6,7 and 8 cycles)		1.	1	1	1	1	PBR,PC	Op Code	1
		2.	1	1	0	1	PBR,PC+1	DO	1
	(2)	2a.	1	1	0	0	PBR,PC+1	IO	1
		3.	1	1	1	0	0,D+DO	AAL	1
		4.	1	1	1	0	0,D+DO+1	AAH	1
		5.	1	1	1	0	0,D+DO+2	AAB	1
		6.	1	1	1	0	AAB,AA+Y	Data Low	1/0
	(1)	6a.	1	1	1	0	AAB,AA+Y+1	Data High	1/0
9. Direct Indexed Indirect (d,x) (ORA,AND,EOR,ADC, STA,LDA,CMP,SBC) (8 Op Codes) (2 bytes) (6,7 and 8 cycles)		1.	1	1	1	1	PBR,PC	Op Code	1
		2.	1	1	0	1	PBR,PC+1	DO	1
	(2)	2a.	1	1	0	0	PBR,PC+1	IO	1
		3.	1	1	0	0	PBR,PC+1	IO	1
		4.	1	1	1	0	0,D+DO+X	AAL	1
		5.	1	1	1	0	0,D+DO+X+1	AAH	1
		6.	1	1	1	0	DBR,AA	Data Low	1/0
	(1)	6a.	1	1	1	0	DBR,AA+1	Data High	1/0
10a. Direct,X d,x (BIT,STZ,STY,LDY, ORA,AND,EOR,ADC, STA,LDA,CMP,SBC) (11 Op Codes) (2 bytes) (4,5 and 6 cycles)		1.	1	1	1	1	PBR,PC	Op Code	1
		2.	1	1	0	1	PBR,PC+1	DO	1
	(2)	2a.	1	1	0	0	PBR,PC+1	IO	1
		3.	1	1	0	0	PBR,PC+1	IO	1
		4.	1	1	1	0	0,D+DO+X	Data Low	1/0
	(1)	4a.	1	1	1	0	0,D+DO+X+1	Data High	1/0
10b. Direct,X(R-M-W) d,x (ASL,ROL,LSR,ROR, DEC,INC) (6 Op Codes) (2 bytes) (6,7,8 and 9 cycles)		1.	1	1	1	1	PBR,PC	Op Code	1
		2.	1	1	0	1	PBR,PC+1	DO	1
	(2)	2a.	1	1	0	0	PBR,PC+1	IO	1
		3.	1	1	0	0	PBR,PC+1	IO	1
		4.	1	0	1	0	0,D+DO+X	Data Low	1
	(1)	4a.	1	0	1	0	0,D+DO+X+1	Data High	1
	(3)	5.	1	0	0	0	0,D+DO+X+1	IO	1
	(1)	6a.	1	0	1	0	0,D+DO+X+1	Data High	0
		6.	1	0	1	0	0,D+DO+X	Data Low	0
11. Direct,Y d,y (STX,LDX) (2 Op Codes) (2 bytes) (4,5 and 6 cycles)		1.	1	1	1	1	PBR,PC	Op Code	1
		2.	1	1	0	1	PBR,PC+1	DO	1
	(2)	2a.	1	1	0	0	PBR,PC+1	IO	1
		3.	1	1	0	0	PBR,PC+1	IO	1
		4.	1	1	1	0	0,D+DO+Y	Data Low	1/0
	(1)	4a.	1	1	1	0	0,D+DO+Y+1	Data High	1/0
12a. Absolute,X a,x (BIT,LDY,STZ, ORA,AND,EOR,ADC, STA,LDA,CMP,SBC) (11 Op Codes) (3 bytes) (4,5 and 6 cycles)		1.	1	1	1	1	PBR,PC	Op Code	1
		2.	1	1	0	1	PBR,PC+1	AAL	1
		3.	1	1	0	1	PBR,PC+2	AAH	1
	(4)	3a.	1	1	0	0	DBR,AAH,AAL+X	L IO	1
		4.	1	1	1	0	DBR,AA+X	Data Low	1/0
	(1)	4a.	1	1	1	0	DBR,AA+X+1	Data High	1/0
12b. Absolute,X(R-M-W) a,x (ASL,ROL,LSR,ROR, DEC,INC) (6 Op Codes) (3 bytes) (7 and 9 cycles)		1.	1	1	1	1	PBR,PC	Op Code	1
		2.	1	1	0	1	PBR,PC+1	AAL	1
		3.	1	1	0	1	PBR,PC+2	AAH	1
		4.	1	1	0	0	DBR,AAH,AAL+X	L IO	1
		5.	1	0	1	0	DBR,AA+X	Data Low	1
	(1)	5a.	1	0	1	0	DBR,AA+X+1	Data High	1
	(3)	6.	1	0	0	0	DBR,AA+X+1	IO	1
	(1)	7a.	1	0	1	0	DBR,AA+X+1	Data High	0
		7.	1	0	1	0	DBR,AA+X	Data Low	0
*13. Absolute Long,X al,x (ORA,AND,EOR,ADC, STA,LDA,CMP,SBC) (8 Op Codes) (4 bytes) (5 and 6 cycles)		1.	1	1	1	1	PBR,PC	Op Code	1
		2.	1	1	0	1	PBR,PC+1	AAL	1
		3.	1	1	0	1	PBR,PC+2	AAH	1
		4.	1	1	0	1	PBR,PC+3	AAB	1
		5.	1	1	1	0	AAB,AA+X	Data Low	1/0
	(1)	5a.	1	1	1	0	AAB,AA+X+1	Data High	1/0
14. Absolute,Y a,y (LDX,ORA,AND,EOR,ADC, STA,LDA,CMP,SBC) (9 Op Codes) (3 bytes) (4,5 and 6 cycles)		1.	1	1	1	1	PBR,PC	Op Code	1
		2.	1	1	0	1	PBR,PC+1	AAL	1
		3.	1	1	0	1	PBR,PC+2	AAH	1
	(4)	3a.	1	1	0	0	DBR,AAH,AAL+Y	L IO	1
		4.	1	1	1	0	DBR,AA+Y	Data Low	1/0
	(1)	4a.	1	1	1	0	DBR,AA+Y+1	Data High	1/0
15. Relative r (BPL,BMI,BVC,BVS,BCC, BCS,BNE,BEQ,BRA) (9 Op Codes) (2 bytes) (2,3 and 4 cycles)		1.	1	1	1	1	PBR,PC	Op Code	1
		2.	1	1	0	1	PBR,PC+1	Offset	1
	(5)	2a.	1	1	0	0	PBR,PC+1	IO	1
	(6)	2b.	1	1	0	0	PBR,PC+1	IO	1
		1.	1	1	1	1	PBR,PC+Offset	Op Code	1
*16. Relative Long rl (BRL) (1 Op Code) (3 bytes) (4 cycles)		1.	1	1	1	1	PBR,PC	Op Code	1
		2.	1	1	0	1	PBR,PC+1	Offset Low	1
		3.	1	1	0	1	PBR,PC+2	Offset High	1
		4.	1	1	0	0	PBR,PC+2	IO	1
		1.	1	1	1	1	PBR,PC+Offset	Op Code	1
17a. Absolute Indirect (a) (JMP) (1 Op Code) (3 bytes) (5 cycles)		1.	1	1	1	1	PBR,PC	Op Code	1
		2.	1	1	0	1	PBR,PC+1	AAL	1
		3.	1	1	0	1	PBR,PC+2	AAH	1
		4.	1	1	1	0	0,AA	NEW PCL	1
		5.	1	1	1	0	0,AA+1	NEW PCH	1
		1.	1	1	1	1	PBR,NEW PC	Op Code	1
*17b. Absolute Indirect (a) (JML) (1 Op Code) (3 bytes) (6 cycles)		1.	1	1	1	1	PBR,PC	Op Code	1
		2.	1	1	0	1	PBR,PC+1	AAL	1
		3.	1	1	0	1	PBR,PC+2	AAH	1
		4.	1	1	1	0	0,AA	NEW PCL	1
		5.	1	1	1	0	0,AA+1	NEW PCH	1
		6.	1	1	1	0	0,AA+2	NEW PBR	1
		1.	1	1	1	1	NEW PBR,PC	Op Code	1
• 18. Direct Indirect (d) (ORA,AND,EOR,ADC, STA,LDA,CMP,SBC) (8 Op Codes) (2 bytes) (5,6 and 7 cycles)		1.	1	1	1	1	PBR,PC	Op Code	1
		2.	1	1	0	1	PBR,PC+1	DO	1
	(2)	2a.	1	1	0	0	PBR,PC+1	IO	1
		3.	1	1	1	0	0,D+DO	AAL	1
		4.	1	1	1	0	0,D+DO+1	AAH	1
		5.	1	1	1	0	DBR,AA	Data Low	1/
	(1)	5a.	1	1	1	0	DBR,AA+1	Data Low	

Detailed Instruction Operation (continued)

	ADDRESS MODE		CYCLE	VP	ML	VDA	VPA	ADDRESS BUS	DATA BUS	R/W
*19	Direct Indirect Long [d]		1.	1	1	1	1	PBR,PC	Op Code	1
	(ORA,AND,EOR,ADC		2.	1	1	0	1	PBR,PC+1	DO	1
	STA,LDA,CMP,SBC)	(2)	2a.	1	1	0	0	PBR,PC+1	IO	1
	(8 Op Codes)		3.	1	1	1	0	0,D+DO	AAL	1
	(2 bytes)		4.	1	1	1	0	0,D+DO+1	AAH	1
	(6,7 and 8 cycles)		5.	1	1	1	0	0,D+DO+2	AAB	1
			6.	1	1	1	0	AAB,AA	Data Low	1/0
		(1)	6a.	1	1	1	0	AAB,AA+1	Data High	1/0
20a.	Absolute Indexed Indirect (a,x)		1.	1	1	1	1	PBR,PC	Op Code	1
	(JMP)		2.	1	1	0	1	PBR,PC+1	AAL	1
	(1 Op Code)		3.	1	1	0	1	PBR,PC+2	AAH	1
	(3 bytes)		4.	1	1	0	0	PBR,PC+2	IO	1
	(6 cycles)		5.	1	1	0	1	PBR,AA+X	NEW PCL	1
			6.	1	1	0	1	PBR,AA+X+1	NEW PCH	1
			1.	1	1	1	1	PBR, NEW PC	Op Code	1
*20b	Absolute Indexed Indirect		1.	1	1	1	1	PBR,PC	Op Code	1
	(Jump to Subroutine Indexed		2.	1	1	0	1	PBR,PC+1	AAL	1
	Indirect) (a,x)		3.	1	1	1	0	0,S	PCH	0
	(JSR)		4.	1	1	1	0	0,S-1	PCL	0
	(1 Op Code)		5.	1	1	0	1	PBR,PC+2	AAH	1
	(3 bytes)		6.	1	1	0	0	PBR,PC+2	IO	1
	(8 cycles)		7.	1	1	0	1	PBR,AA+X	NEW PCL	1
			8.	1	1	0	1	PBR,AA+X+1	NEW PCH	1
			1.	1	1	1	1	PBR,NEW PC	Next Op Code	1
21a.	Stack (Hardware		1.	1	1	1	1	PBR,PC	IO	1
	Interrupts) s	(3)	2.	1	1	0	0	PBR,PC	IO	1
	(IRQ,NMI,ABORT,RES)	(7)	3.	1	1	1	0	0,S	PBR	0
	(4 hardware interrupts)		4.	1	1	1	0	0,S-1	PCH	0
	(0 bytes)		5.	1	1	1	0	0,S-2	PCL	0
	(7 and 8 cycles)		6.	1	1	1	0	0,S-3	P	0
			7.	0	1	1	0	0,VA	AAVL	1
			8.	0	1	1	0	0,VA+1	AAVH	1
			1.	1	1	1	1	0,AAV	Next Op Code	1
21b.	Stack (Software		1.	1	1	1	1	PBR,PC	Op Code	1
	Interrupts) s	(3)	2.	1	1	0	1	PBR,PC+1	Signature	1
	(BRK,COP)	(7)	3.	1	1	1	0	0,S	PBR	0
	(2 Op Codes)		4.	1	1	1	0	0,S-1	PCH	0
	(2 bytes)		5.	1	1	1	0	0,S-2	PCL	0
	(7 and 8 cycles)		6.	1	1	1	0	0,S-3 (COP Latches)	P	0
			7.	0	1	1	0	0,VA	AAVL	1
			8.	0	1	1	0	0,VA+1	AAVH	1
			1.	1	1	1	1	0,AAV	Next Op Code	1
21c.	Stack (Return from		1.	1	1	1	1	PBR,PC	Op Code	1
	Interrupt) s		2.	1	1	0	0	PBR,PC+1	IO	1
	(RTI)	(3)	3.	1	1	0	0	PBR,PC+1	IO	1
	(1 Op Code)		4.	1	1	1	0	0,S+1	P	1
	(1 byte)		5.	1	1	1	0	0,S+2	PCL	1
	(6 and 7 cycles)		6.	1	1	1	0	0,S+3	PCH	1
	(different order from N6502)	(7)	7.	1	1	1	0	0,S+4	PBR	1
			1.	1	1	1	1	PBR,PC	New Op Code	1
21d.	Stack (Return from		1.	1	1	1	1	PBR,PC	Op Code	1
	Subroutine) s		2.	1	1	0	0	PBR,PC+1	IO	1
	(RTS)		3.	1	1	0	0	PBR,PC+1	IO	1
	(1 Op Code)		4.	1	1	1	0	0,S+1	PCL	1
	(1 byte)		5.	1	1	1	0	0,S+2	PCH	1
	(6 cycles)		6.	1	1	0	0	0,S+2	IO	1
			1.	1	1	1	1	PBR,PC	Op Code	1
*21e.	Stack (Return from		1.	1	1	1	1	PBR,PC	Op Code	1
	Subroutine Long) s		2.	1	1	0	0	PBR,PC+1	IO	1
	(RTL)		3.	1	1	0	0	PBR,PC+1	IO	1
	(1 Op Code)		4.	1	1	1	0	0,S+1	NEW PCL	1
	(1 byte)		5.	1	1	1	0	0,S+2	NEW PCH	1
	(6 cycles)		6.	1	1	1	0	0,S+3	NEW PBR	1
			1.	1	1	1	1	NEW PBR,PC	Next Op Code	1
21f.	Stack (Push) s		1.	1	1	1	1	PBR,PC	Op Code	1
	(PHP,PHA,PHY,PHX,		2.	1	1	0	0	PBR,PC+1	IO	1
	PHD,PHK,PHB)	(1)	3a.	1	1	1	0	0,S	Register High	1
	(7 Op Codes)		3.	1	1	1	0	0,S-1	Register Low	1
	(1 byte)									
	(3 and 4 cycles)									
21g.	Stack (Pull) s		1.	1	1	1	1	PBR,PC	Op Code	1
	(PLP,PLA,PLY,PLX,PLD,PLB)		2.	1	1	0	0	PBR,PC+1	IO	1
	(Different than N6502)		3.	1	1	0	0	PBR,PC+1	IO	1
	(6 Op Codes)		4.	1	1	1	0	0,S+1	Register Low	1
	(1 byte)	(1)	4a.	1	1	1	0	0,S+2	Register High	1
	(4 and 5 cycles)									
*21h.	Stack (Push Effective		1.	1	1	1	1	PBR,PC	Op Code	1
	Indirect Address) s		2.	1	1	0	1	PBR,PC+1	DO	1
	(PEI)	(2)	2a.	1	1	0	0	PBR,PC+1	IO	1
	(1 Op Code)		3.	1	1	1	0	0,D+DO	AAL	1
	(2 bytes)		4.	1	1	1	0	0,D+DO+1	AAH	1
	(6 and 7 cycles)		5	1	1	1	0	0,S	AAH	0
			6.	1	1	1	0	0,S-1	AAL	0
*21i.	Stack (Push Effective		1.	1	1	1	1	PBR,PC	Op Code	1
	Absolute Address) s		2.	1	1	0	1	PBR,PC+1	AAL	1
	(PEA)		3.	1	1	0	1	PBR,PC+2	AAH	1
	(1 Op Code)		4.	1	1	1	0	0,S	AAH	0
	(3 bytes)		5.	1	1	1	0	0,S-1	AAL	0
	(5 cycles)									
*21j.	Stack (Push Effective		1.	1	1	1	1	PBR,PC	Op Code	1
	Program Counter Relative		2.	1	1	0	1	PBR,PC+1	Offset Low	1
	Address) s		3.	1	1	0	1	PBR,PC+2	Offset High	1
	(PER)		4.	1	1	0	0	PBR,PC+2	IO	1
	(1 Op Code)		5.	1	1	1	0	0,S	PCH+OFF+ CARRY	0
	(3 bytes)									
	(6 cycles)		6.	1	1	1	0	0,S-1	PCL+OFFSET	0
*22.	Stack Relative d,s		1.	1	1	1	1	PBR,PC	Op Code	1
	(ORA,AND,EOR,ADL,		2.	1	1	0	1	PBR,PC+1	SO	1
	STA,LDA,CMP,SDC)		3.	1	1	0	0	PBR,PC+1	IO	1
	(8 Op Codes)		4.	1	1	1	0	0,S+SO	Data Low	1/0
	(2 bytes)	(1)	4a.	1	1	1	0	0,S+SO+1	Data High	1/0
	(4 and 5 cycles)									

	ADDRESS MODE		CYCLE	VP	ML	VDA	VPA	ADDRESS BUS	DATA BUS	R/W
*23.	Stack Relative Indirect		1.	1	1	1	1	PBR,PC	Op Code	1
	Indexed (d,s),y		2.	1	1	0	1	PBR,PC+1	SO	1
	(ORA,AND,EOR,ADC,		3.	1	1	0	0	PBR+PC+1	IO	1
	STA,LDA,CMP,SDC)		4.	1	1	1	0	0,S+SO	AAL	1
	(8 Op Codes)		5.	1	1	1	0	0,S+SO+1	AAH	1
	(2 bytes)		6.	1	1	0	0	0,S+SO+1	IO	1
	(7 and 8 Cycles)		7.	1	1	1	0	DBR,AA+Y	Data Low	1/0
		(1)	7a.	1	1	1	0	DBR,AA+Y+1	Data High	1/0
*24a.	Block Move Positive (forward) xyc		1.	1	1	1	1	PBR,PC	Op Code	1
	(MVP)		2.	1	1	0	1	PBR,PC+1	DBA	1
	(1 Op Code)	N-2	3.	1	1	0	1	PBR,PC+2	SBA	1
	(3 bytes)	Byte	4.	1	1	1	0	SBA,X	Source Data	1
	(7 cycles)	C=2	5.	1	1	1	0	DBA,Y	Dest. Data	0
	x = Source Address		6.	1	1	0	0	DBA,Y	IO	1
	y = Destination		7.	1	1	0	0	DBA,Y	IO	1
	c =Number of Bytes to Move -1		1.	1	1	1	1	PBR,PC	Op Code	1
	x,y Decrement		2.	1	1	0	1	PBR,PC+1	DBA	1
	MVP is used when the		3.	1	1	0	1	PBR,PC+2	SBA	1
	destination start address	N-1	4.	1	1	1	0	SBA,X-1	Source Data	1
	is higher (more positive)	Byte	5.	1	1	1	0	DBA,Y-1	Dest. Data	0
	than the source start address.	C=1	6.	1	1	0	0	DBA,Y-1	IO	1
			7.	1	1	0	0	DBA,Y-1	IO	1
	FFFFFF Dest. Start Source Start Dest. End Source End 000000	N Byte Last C=0	1.	1	1	1	1	PBR,PC	Op Code	1
			2.	1	1	0	1	PBR,PC+1	DBA	1
			3.	1	1	0	1	PBR,PC+2	SBA	1
			4.	1	1	1	0	SBA,X-2	Source Data	1
			5.	1	1	1	0	DBA,Y-2	Dest. Data	0
			6.	1	1	0	0	DBA,Y-2	IO	1
			7.	1	1	0	0	DBA,Y-2	IO	1
			1.	1	1	1	1	PBR,PC+3	Next Op Code	1
*24b.	Block Move Negative (backward) xyc		1.	1	1	1	1	PBR,PC	Op Code	1
	(MVN)		2.	1	1	0	1	PBR,PC+1	DBA	1
	(1 Op Code)	N-2	3.	1	1	0	1	PBR,PC+2	SBA	1
	(3 bytes)	Byte	4.	1	1	1	0	SBA,X	Source Data	1
	(7 cycles)	C=2	5.	1	1	1	0	DBA,Y	Dest. Data	0
	x = Source Address		6.	1	1	0	0	DBA,Y	IO	1
	y = Destination		7.	1	1	0	0	DBA,Y	IO	1
	c = Number of Bytes to Move -1		1.	1	1	1	1	PBR,PC	Op Code	1
	x,y Increment		2.	1	1	0	1	PBR,PC+1	DBA	1
		N-1	3.	1	1	0	1	PBR,PC+2	SBA	1
	FFFFFF	Byte	4.	1	1	1	0	SBA,X+1	Source Data	1
	Source End	C=1	5.	1	1	1	0	DBA,Y+1	Dest. Data	0
	Dest.End		6.	1	1	0	0	DBA,Y+1	IO	1
	Source Start		7.	1	1	0	0	DBA,Y+1	IO	1
	Dest. Start		1.	1	1	1	1	PBR,PC	Op Code	1
			2.	1	1	0	1	PBR,PC+1	DBA	1
		N Byte	3.	1	1	0	1	PBR,PC+2	SBA	1
	000000	C=0	4.	1	1	1	0	SBA,X+2	Source Data	1
	MVN is used when the		5.	1	1	1	0	DBA,Y+2	Dest. Data	0
	destination start address		6.	1	1	0	0	DBA,Y+2	IO	1
	is lower (more negative)		7.	1	1	0	0	DBA,Y+2	IO	1
	than the source start address.		1.	1	1	1	1	PBR,PC+3	Next Op Code	1

Notes:

(1) Add 1 byte (for immediate only) for M=0 or X=0 (i.e. 16 bit data), add 1 cycle for M=0 or X=0.

(2) Add 1 cycle for direct register low (DL) not equal 0.

(3) Special case for aborting instruction. This is the last cycle which may be aborted or the Status, PBR or DBR registers will be updated.

(4) Add 1 cycle for indexing across page boundaries, or write, or X=0. When X=1 or in the emulation mode, this cycle contains invalid addresses.

(5) Add 1 cycle if branch is taken.

(6) Add 1 cycle if branch is taken across page boundaries in 6502 emulation mode (E=1).

(7) Subtract 1 cycle for 6502 emulation mode (E=1).

(8) Add 1 cycle for REP,SEP.

(9) Wait at cycle 2 for 2 cycles after $\overline{NMI}$ or $\overline{IRQ}$ active input.

Abbreviations:

AAB Absolute Address Bank
AAH Absolute Address High
AAL Absolute Address Low
AAVH Absolute Address Vector High
AAVL Absolute Address Vector Low
C Accumulator
D Direct Register
DBA Destination Bank Address
DBR Data Bank Register
DO Direct Offset
IDH Immediate Data High
IDL Immediate Data Low
IO Internal Operation
P Status Register
PBR Program Bank Register
PC Program Counter
R-M-W Read-Modify-Write
S Stack Address
SBA Source Bank Address
SO Stack Offset
VA Vector Address
x,y Index Registers
* = New W65C816/802 Addressing Modes
● = New W65C02 Addressing Modes
Blank = NMOS 6502 Addressing Modes

Recommended W65C816 and W65C802 Assembler Syntax Standards

Directives

Assembler directives are those parts of the assembly language source program which give directions to the assembler; this includes the definition of data area and constants within a program. This standard excludes any definitions of assembler directives.

Comments

An assembler should provide a way to use any line of the source program as a comment. The recommended way of doing this is to treat any blank line, or any line that starts with a semi-colon or an asterisk as a comment. Other special characters may be used as well.

The Source Line

Any line which causes the generation of a single W65C816 or W65C802 machine language instruction should be divided into four fields: a label field, the operation code, the operand, and the comment field.

The Label Field—The label field begins in column one of the line. A label must start with an alphabetic character, and may be followed by zero or more alphanumeric characters. An assembler may define an upper limit on the number of characters that can be in a label, so long as that upper limit is greater than or equal to six characters. An assembler may limit the alphabetic characters to upper-case characters if desired. If lower-case characters are allowed, they should be treated as identical to their upper-case equivalents. Other characters may be allowed in the label, so long as their use does not conflict with the coding of operand fields.

The Operation Code Field—The operation code shall consist of a three character sequence (mnemonic) from Table 3. It shall start no sooner than column 2 of the line, or one space after the label if a label is coded.

Many of the operation codes in Table 3 have duplicate mnemonics; when two or more machine language instructions have the same mnemonic, the assembler resolves the difference based on the operand.

If an assembler allows lower-case letters in labels, it must also allow lower-case letters in the mnemonic. When lower-case letters are used in the mnemonic, they shall be treated as equivalent to the upper-case counterpart. Thus, the mnemonics LDA, lda, and LdA must all be recognized, and are equivalent.

In addition to the mnemonics shown in Table 3, an assembler may provide the alternate mnemonics shown in Table 6.

Alternate Mnemonics

Standard	Alias
BCC	BLT
BCS	BGE
CMP A	CMA
DEC A	DEA
INC A	INA
JSL	JSR
JML	JMP
TCD	TAD
TCS	TAS
TDC	TDA
TSC	TSA
XBA	SWA

JSL should be recognized as equivalent to JSR when it is specified with a long absolute address. JML is equivalent to JMP with long addressing forced.

The Operand Field—The operand field may start no sooner than one space after the operation code field. The assembler must be capable of at least twenty-four bit address calculations. The assembler should be capable of specifying addresses as labels, integer constants, and hexadecimal constants. The assembler must allow addition and subtraction in the operand field. Labels shall be recognized by the fact that they start alphabetic characters. Decimal numbers shall be recognized as containing only the decimal digits 0...9. Hexadecimal constants shall be recognized by prefixing the constant with a "$" character, followed by zero or more of either the decimal digits or the hexadecimal digits "A"..."F". If lower-case letters are allowed in the label field, then they shall also be allowed as hexadecimal digits.

All constants, no matter what their format, shall provide at least enough precision to specify all values that can be represented by a twenty-four bit signed or unsigned integer represented in two's complement notation.

Table 8 shows the operand formats which shall be recognized by the assembler. The symbol **d** is a label or value which the assembler can recognize as being less than $100. The symbol **a** is a label or value which the assembler can recognize as greater the $FF but less than $10000; the symbol **al** is a label or value that the assembler can recognize as being greater than $FFFF. The symbol EXT is a label which cannot be located by the assembler at the time the instruction is assembled. Unless instructed otherwise, an assembler shall assume that EXT labels are two bytes long. The symbols **r** and **rl** are 8 and 16 bit signed displacements calculated by the assembler.

Note that the operand does not determine whether or not immediate addressing loads one or two bytes; this is determined by the setting of the status register. This forces the requirement for a directive or directives that tell the assembler to generate one or two bytes of space for immediate loads. The directives provided shall allow separate settings for the accumulator and index registers.

The assembler shall use the <, >, and ^ characters after the # character in immediate address to specify which byte or bytes will be selected from the value of the operand. Any calculations in the operand must be performed before the byte selection takes place. Table 7 defines the action taken by each operand by showing the effect of the operator on an address. The column that shows a two byte immediate value show the bytes in the order in which they appear in memory. The coding of the operand is for an assembler which uses 32 bit address calculations, showing the way that the address should be reduced to a 24 bit value.

Byte Selection Operator

Operand	One Byte Result	Two Byte Result	
#$01020304	04	04	03
#<$01020304	04	04	03
#>$01020304	03	03	02
#^$01020304	02	02	01

In any location in an operand where an address, or expression resulting in an address, can be coded, the assembler shall recognize the prefix characters <, |, and >, which force one byte (direct page), two byte (absolute) or three byte (long absolute) addressing. In cases where the addressing mode is not forced, the assembler shall assume that the address is two bytes unless the assembler is able to determine the type of addressing required by context, in which case that addressing mode will be used. Addresses shall be truncated without error if an addressing mode is forced which does not require the entire value of the address. For example,

```
LDA  $0203      LDA  |$010203
```

are completely equivalent. If the addressing mode is not forced, and the type of addressing cannot be determined from context, the assembler shall assume that a two byte address is to be used. If an instruction does not have a short addressing mode (as in LDA, which has no direct page indexed by Y) and a short address is used in the operand, the assembler shall automatically extend the address by padding the most significant bytes with zeroes in order to extend the address to the length needed. As with immediate addressing, any expression evaluation shall take place before the address is selected; thus, the address selection character is only used once, before the address of expression.

The ! (exclamation point) character should be supported as an alternative to the | (vertical bar).

A long indirect address is indicated in the operand field of an instruction by surrounding the direct page address where the indirect address is found by square brackets; direct page addresses which contain sixteen-bit addresses are indicated by being surrounded by parentheses.

The operands of a block move instruction are specified as source bank, destination bank—the opposite order of the object bytes generated.

Comment Field—The comment field may start no sooner than one space after the operation code field or operand field depending on instruction type.

Address Mode Formats

Addressing Mode	Format
Immediate	#d
	#a
	#al
	#EXT
	#<d
	#<a
	#<al
	#<EXT
	#>d
	#>a
	#>al
	#>EXT
	#^d
	#^a
	#^al
	#^EXT
Absolute	!d
	!a
	a
	!al
	!EXT
	EXT
Absolute Long	>d
	>a
	>al
	al
	>EXT
Direct Page	d
	<d
	<a
	<al
	<EXT
Accumulator	A
Implied Addressing	(no operand)
Direct Indirect Indexed	(d),y
	(<d),y
	(<a),y
	(<al),y
	(<EXT),y
Direct Indirect Indexed Long	[d],y
	[<d],y
	[<a],y
	[<al],y
	[<EXT],y
Direct Indexed Indirect	(d,x)
	(<d,x)
	(<a,x)
	(<al,x)
	(<EXT,x)
Direct Indexed by X	d,x
	<d,x
	<a,x
	<al,x
	<EXT,x
Direct Indexed by Y	d,y
	<d,y
	<a,y
	<al,y
	<EXT,y
Absolute Indexed by X	d,x
	!d,x
	a,x
	!a,x
	!al,x
	!EXT,x
	EXT,x

Addressing Mode	Format	
Absolute Indexed by Y	!d,y	
	d,y	
	a,y	
	!a,y	
	!al,y	
	!EXT,y	
	EXT,y	
Absolute Long Indexed by X	>d,x	
	>a,x	
	>al,x	
	al,x	
	>EXT,x	
Program Counter Relative and Program Counter Relative Long	d	(the assembler calculates **r** and **rl**)
	a	
	al	
	EXT	
Absolute Indirect	(d)	
	(!d)	
	(a)	
	(!a)	
	(!al)	
	(EXT)	
Direct Indirect	(d)	
	(<a)	
	(<al)	
	(<EXT)	
Direct Indirect Long	[d]	
	[<a]	
	[<al]	
	[<EXT]	
Absolute Indexed	(d,x)	
	(!d,x)	
	(a,x)	
	(!a,x)	
	(!al,x)	
	(EXT,x)	
	(!EXT,x)	
Stack Addressing	(no operand)	
Stack Relative Indirect Indexed	(d,s),y	
	(<d,s),y	
	(<a,s),y	
	(<al,s),y	
	(<EXT,s),y	
Block Move	d,d	
	d,a	
	d,al	
	d,EXT	
	a,d	
	a,a	
	a,al	
	a,EXT	
	al,d	
	al,a	
	al,al	
	al,EXT	
	EXT,d	
	EXT,a	
	EXT,al	
	EXT,EXT	

Note: The alternate ! (exclamation point) is used in place of the | (vertical bar).

Addressing Mode Summary

Address Mode	Instruction Times In Memory Cycles		Memory Utilization In Number of Program Sequence Bytes	
	Original 8 Bit NMOS 6502	New W65C816	Original 8 Bit NMOS 6502	New W65C816
1. Immediate	2	2(3)	2	2(3)
2. Absolute	4(5)	4(3,5)	3	3
3. Absolute Long	—	5(3)	—	4
4. Direct	3(5)	3(3,4,5)	2	2
5. Accumulator	2	2	1	1
6. Implied	2	2	1	1
7. Direct Indirect Indexed (d),y	5(1)	5(1,3,4)	2	2
8. Direct Indirect Indexed Long [d], y	—	6(3,4)	—	2
9. Direct Indexed Indirect (d,x)	6	6(3,4)	2	2
10. Direct, X	4(5)	4(3,4,5)	2	2
11. Direct, Y	4	4(3,4)	2	2
12. Absolute, X	4(1,5)	4(1,3,5)	3	3
13. Absolute Long, X	—	5(3)	—	4
14. Absolute, Y	4(1)	4(1,3)	3	3
15. Relative	2(1,2)	2(2)	2	2
16. Relative Long	—	3(2)	—	3
17. Absolute Indirect (Jump)	5	5	3	3
18. Direct Indirect	—	5(3,4)	—	2
19. Direct Indirect Long	—	6(3,4)	—	2
20. Absolute Indexed Indirect (Jump)	—	6	—	3
21. Stack	3-7	3-8	1-3	1-4
22. Stack Relative	—	4(3)	—	2
23. Stack Relative Indirect Indexed	—	7(3)	—	2
24. Block Move X, Y, C (Source, Destination, Block Length)	—	7	—	3

NOTES:
1. Page boundary, add 1 cycle if page boundary is crossed when forming address.
2. Branch taken, add 1 cycle if branch is taken.
3. M = 0 or X = 0, 16 bit operation, add 1 cycle, add 1 byte for immediate.
4. Direct register low (DL) not equal zero, add 1 cycle.
5. Read-Modify-Write, add 2 cycles for M = 1, add 3 cycles for M = 0.

Caveats and Application Information

Stack Addressing

When in the Native mode, the Stack may use memory locations 000000 to 00FFFFF. The effective address of Stack, Stack Relative, and Stack Relative Indirect Indexed addressing modes will always be within this range. In the Emulation mode, the Stack address range is 000100 to 0001FF. The following opcodes and addressing modes will increment or decrement beyond this range when accessing two or three bytes:

JSL; JSR(a,x); PEA; PEI; PER; PHD; PLD; RTL; d,s; (d,s),y

Direct Addressing

The Direct Addressing modes are often used to access memory registers and pointers. The effective address generated by Direct; Direct,X and Direct,Y addressing modes will always be in the Native mode range 000000 to 00FFFF. When in the Emulation mode, the direct addressing range is 000000 to 0000FF, except for [Direct] and [Direct],Y addressing modes and the PEI instruction which will increment from 0000FE or 0000FF into the Stack area.

When in the Emulation mode and DH is not equal to zero, the direct addressing range is 00DH00 to 00DHFF, except for [Direct] and [Direct],Y addressing modes and the PEI instruction which will increment from 00DHFE or 00DHFF into the next higher page.

When in the Emulation mode and DL in not equal to zero, the direct addressing range is 000000 to 00FFFF.

Absolute Indexed Addressing (W65C816 Only)

The Absolute Indexed addressing modes are used to address data outside the direct addressing range. The W65C02 and W65C802 addressing range is 0000 to FFFF. Indexing from page FFXX may result in a 00YY data fetch when using the W65C02 or W65C802. In contrast, indexing from page ZZFFXX may result in ZZ+1,00YY when using the W65C816.

Future Microprocessors (i.e., W65C832)

Future WDC microprocessors will support all current W65C816 operating modes for both index and offset address generation.

$\overline{\text{ABORT}}$ Input (W65C816 Only)

$\overline{\text{ABORT}}$ should be held low for a period not to exceed one cycle. Also, if $\overline{\text{ABORT}}$ is held low during the Abort Interrupt sequence, the Abort Interrupt will be aborted. It is not recommended to abort the Abort Interrupt. The $\overline{\text{ABORT}}$ internal latch is cleared during the second cycle of the Abort Interrupt. Asserting the $\overline{\text{ABORT}}$ input after the following instruction cycles will cause registers to be modified:

- **Read-Modify-Write:** Processor status modified if $\overline{\text{ABORT}}$ is asserted after a modify cycle.
- **RTI:** Processor status will be modified if $\overline{\text{ABORT}}$ is asserted after cycle 3.
- **$\overline{\text{IRQ}}$, $\overline{\text{NMI}}$, $\overline{\text{ABORT}}$ BRK, COP:** When $\overline{\text{ABORT}}$ is asserted after cycle 2, PBR and DBR will become 00 (Emulation mode) or PBR will become 00 (Native mode).

The Abort Interrupt has been designed for virtual memory systems. For this reason, asynchronous $\overline{\text{ABORT}}$'s may cause undesirable results due to the above conditions.

VDA and VPA Valid Memory Address Output Signals (W65C816 Only)

When VDA or VPA are high and during all write cycles, the Address Bus is always valid. VDA and VPA should be used to qualify all memory cycles. Note that when VDA and VPA are both low, invalid addresses may be generated. The Page and Bank addresses could also be invalid. This will be due to low byte addition only. The cycle when only low byte addition occurs is an optional cycle for instructions which read memory when the Index Register consists of 8 bits. This optional cycle becomes a standard cycle for the Store instruction, all instructions using the 16-bit Index Register mode, and the Read-Modify-Write instruction when using 8- or 16-bit Index Register modes.

Apple II, IIe, IIc and II+ Disk Systems (W65C816 Only)

VDA and VPA should not be used to qualify addresses during disk operation on Apple systems. Consult your Apple representative for hardware/software configurations.

DB/BA Operation when RDY is Pulled Low (W65C816 Only)

When RDY is low, the Data Bus is held in the data transfer state (i.e., $\phi 2$ high). The Bank address external transparent latch should be latched when the $\phi 2$ clock or RDY is low.

M/X Output (W65C816 Only)

The M/X output reflects the value of the M and X bits of the processor Status Register. The REP, SEP and PLP instructions may change the state of the M and X bits. Note that the M/X output is invalid during the instruction cycle following REP, SEP and PLP instruction execution. This cycle is used as the opcode fetch cycle of the next instruction.

All Opcodes Function in All Modes of Operation

It should be noted that all opcodes function in all modes of operation. However, some instructions and addressing modes are intended for W65C816 24-bit addressing and are therefore less useful for the W65C802. The following is a list of instructions and addressing modes which are primarily intended for W65C816 use:

JSL; RTL; [d]; [d],y; JMP al; JML; al; al,x

The following Instructions may be used with the W65C802 even though a Bank Address is not multiplexed on the Data Bus:

PHK; PHB; PLB

The following instructions have "limited" use in the Emulation mode:

- The REP and SEP instructions cannot modify the M and X bits when in the Emulation mode. In this mode the M and X bits will always be high (logic 1).
- When in the Emulation mode, the MVP and MVN instructions use the X and Y Index Registers for the memory address. Also, the MVP and MVN instructions can only move data within the memory range 0000 (Source Bank) to 00FF (Destination Bank) for the W65C816, and 0000 to 00FF for the W65C802.

Indirect Jumps

The JMP (a) and JML (a) instructions use the direct Bank for indirect addressing, while JMP (a,x) and JSR (a,x) use the Program Bank for indirect address tables.

Switching Modes

When switching from the Native mode to the Emulation mode, the X and M bits of the Status Register are set high (logic 1), the high byte of the Stack is set to 01, and the high bytes of the X and Y Index Registers are set to 00. To save previous values, these bytes must always be stored before changing modes. Note that the low byte of the S, X and Y Registers and the low and high byte of the Accumulator (A and B) are not affected by a mode change.

How Hardware Interrupts, BRK, and COP Instructions Affect the Program Bank and the Data Bank Registers

When in the Native mode, the Program Bank register (PBR) is cleared to 00 when a hardware interrupt, BRK or COP is executed. In the Native mode, previous PBR contents is automatically saved on Stack.

In the Emulation mode, the PBR and DBR registers are cleared to 00 when a hardware interrupt, BRK or COP is executed. In this case, previous contents of the PBR are not automatically saved.

Note that a Return from Interrupt (RTI) should always be executed from the same "mode" which originally generated the interrupt.

Binary Mode

The Binary mode is set whenever a hardware or software interrupt is executed. The D flag within the Status Register is cleared to zero.

WAI Instruction

The WAI instruction pulls RDY low and places the processor in the WAI "low power" mode. $\overline{\text{NMI}}$, $\overline{\text{IRQ}}$ or RESET will terminate the WAI condition and transfer control to the interrupt handler routine. Note that an $\overline{\text{ABORT}}$ input will abort the WAI instruction, but will not restart the processor. When the Status Register I flag is set ($\overline{\text{IRQ}}$ disabled), the $\overline{\text{IRQ}}$ interrupt will cause the next instruction (following the WAI instruction) to be executed without going to the $\overline{\text{IRQ}}$ interrupt handler. This method results in the highest speed response to an $\overline{\text{IRQ}}$ input. When an interrupt

is received after an $\overline{ABORT}$ which occurs during the WAI instruction, the processor will return to the WAI instruction. Other than $\overline{RES}$ (highest priority), $\overline{ABORT}$ is the next highest priority, followed by $\overline{NMI}$ or $\overline{IRQ}$ interrupts.

STP Instruction

The STP instruction disables the $\phi 2$ clock to all circuitry. When disabled, the $\phi 2$ clock is held in the high state. In this case, the Data Bus will remain in the data transfer state and the Bank address will not be multiplexed onto the Data Bus. Upon executing the STP instruction, the $\overline{RES}$ signal is the only input which can restart the processor. The processor is restarted by enabling the $\phi 2$ clock, which occurs on the falling edge of the $\overline{RES}$ input. Note that the external oscillator must be stable and operating properly before $\overline{RES}$ goes high.

COP Signatures

Signatures 00-7F may be user defined, while signatures 80-FF are reserved for instructions on future microprocessors (i.e., W65C832). Contact WDC for software emulation of future microprocessor hardware functions.

WDM Opcode Use

The WDM opcode will be used on future microprocessors. For example, the new W65C832 uses this opcode to provide 32-bit floating-point and other 32-bit math and data operations. Note that the W65C832 will be a plug-to-plug replacement for the W65C816, and can be used where high-speed, 32-bit math processing is required. The W65C832 will be available in the near future.

RDY Pulled During Write

The NMOS 6502 does not stop during a write operation. In contrast, both the W65C02 and the W65C816 do stop during write operations. The W65C802 stops during a write when in the Native mode, but does not stop when in the Emulation mode.

MVN and MVP Affects on the Data Bank Register

The MVN and MVP instructions change the Data Bank Register to the value of the second byte of the instruction (destination bank address).

Interrupt Priorities

The following interrupt priorities will be in effect should more than one interrupt occur at the same time:

$\overline{RES}$	Highest Priority
$\overline{ABORT}$	
$\overline{NMI}$	
$\overline{IRQ}$	Lowest Priority

Transfers from 8-Bit to 16-Bit, or 16-Bit to 8-Bit Registers

All transfers from one register to another will result in a full 16-bit output from the source register. The destination register size will determine the number of bits actually stored in the destination register and the values stored in the processor Status Register. The following are always 16-bit transfers, regardless of the accumulator size:

TCS; TSC; TCD; TDC

Stack Transfers

When in the Emulation mode, a 01 is forced into SH. In this case, the B Accumulator will not be loaded into SH during a TCS instruction. When in the Native mode, the B Accumulator is transferred to SH. Note that in both the Emulation and Native modes, the full 16 bits of the Stack Register are transferred to the A, B and C Accumulators, regardless of the state of the M bit in the Status Register.