65816

16 Bit Card

Software
Developer's
Guide

Æ 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.
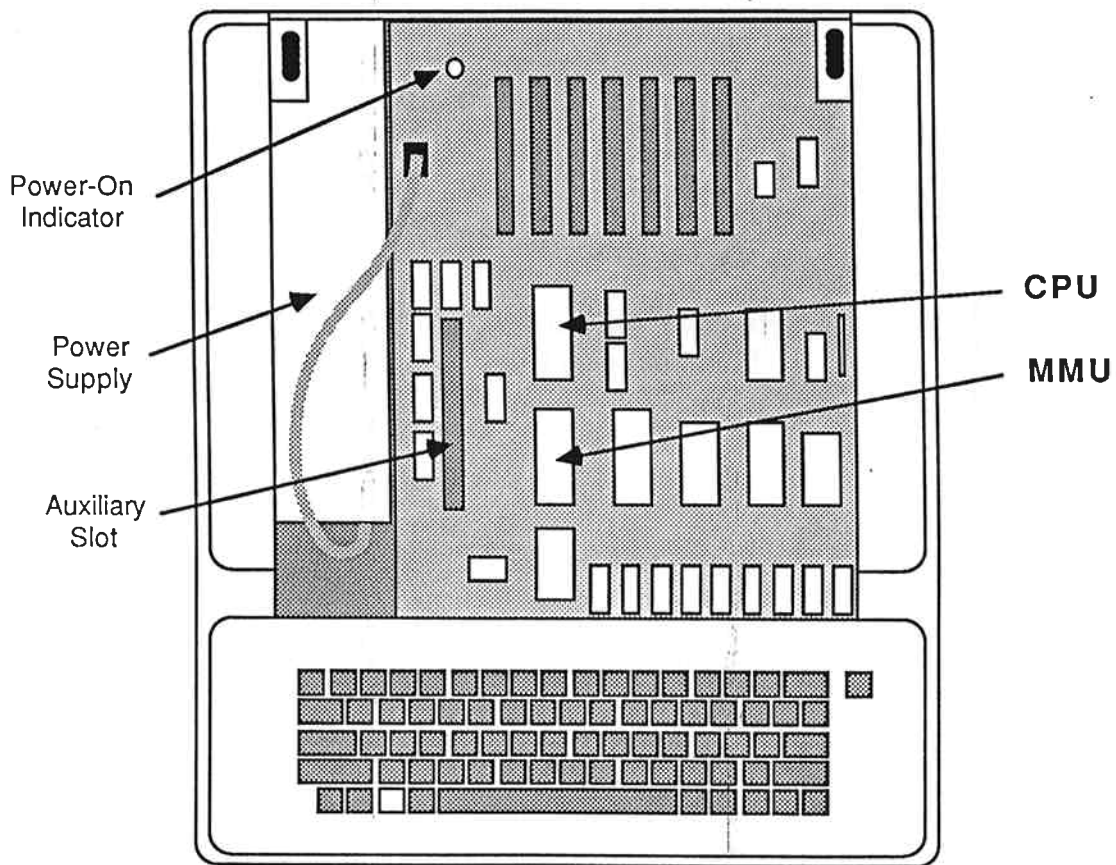
MMU ⟶

Main Logic Board ⟶

END VIEW of SOLDERED MMU

MMU ⟶

SOCKET ⟶

Main Logic Board ⟶

END VIEW of SOCKETED MMU
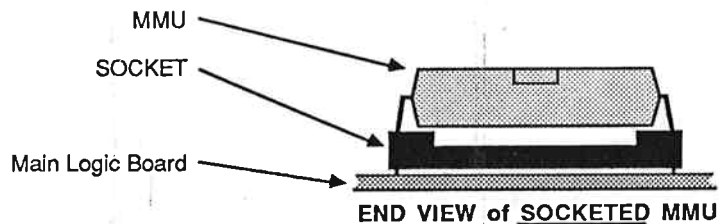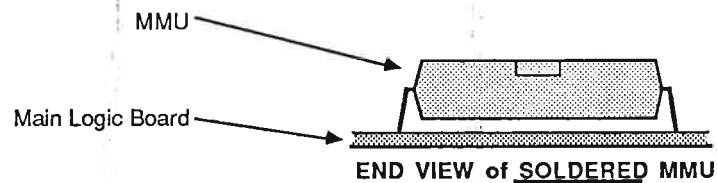
Installing the 16 Bit Card

- 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.)
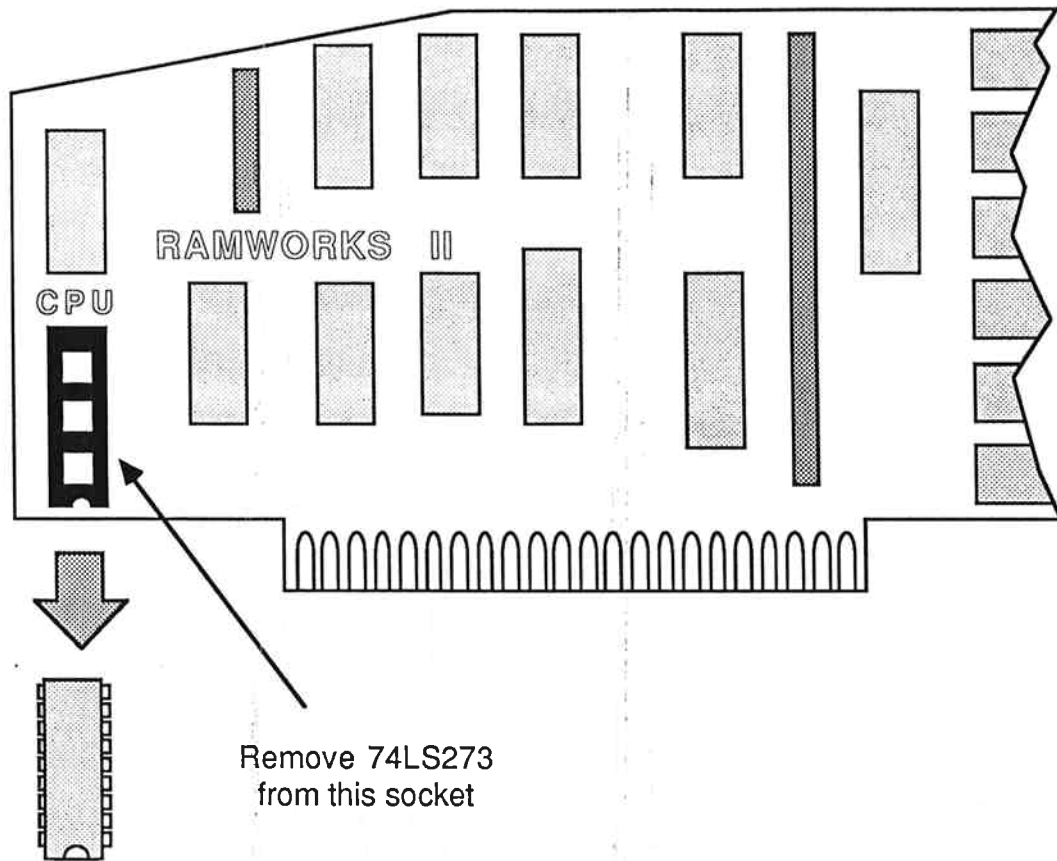
Illustration 1:

- 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:

RAMWORKS II

CPU

Remove 74LS273
from this socket

- 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:

Native (16 bit) mode indicator L.E.D.
RamWorks II Connector
MMU / CPU Headers
Future Expansion Connector

2 Meg. Memory Expander ribbon cable connector (optional)
Socket for 2 Meg. Memory Expander 16L8 P.A.L. (optional)
MMU Socket
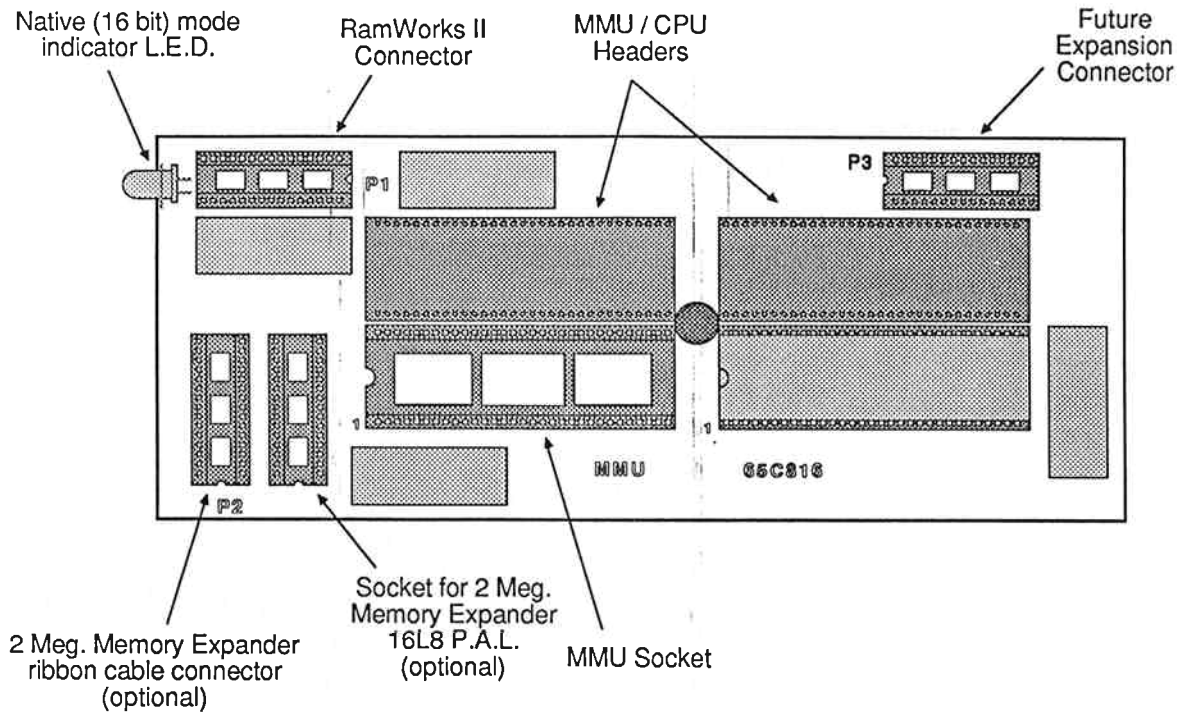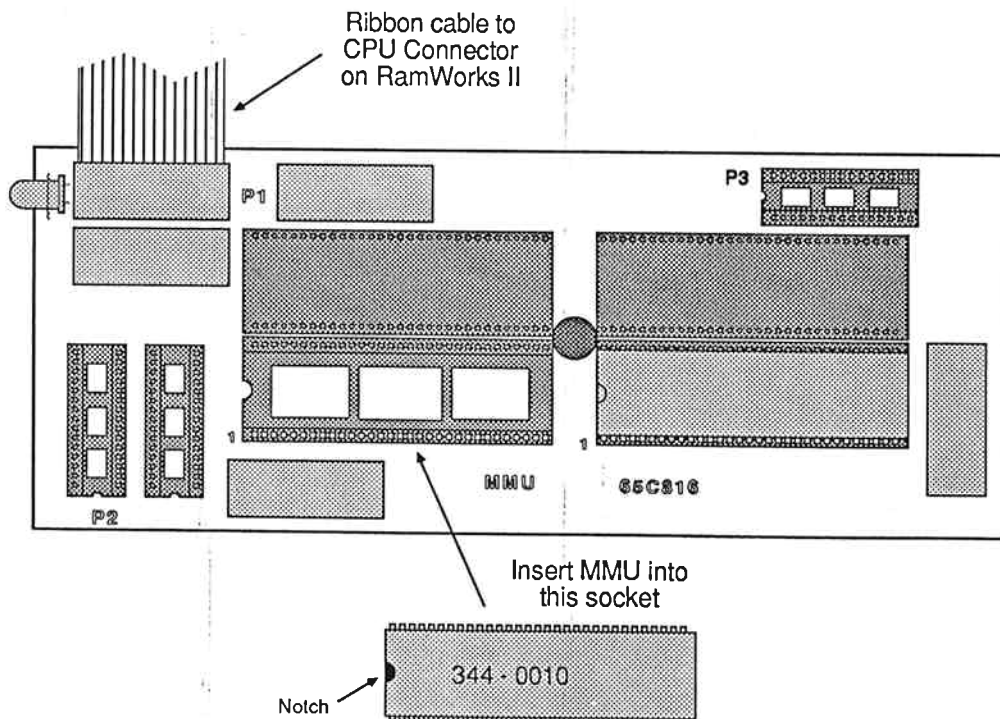
- Install the MMU chip on the 16 Bit Card, as shown in Illustration 4. Be sure the notch is oriented as indictated in the illustration.

Illustration 4:

Ribbon cable to CPU Connector on RamWorks II

Insert MMU into this socket

344 - 0010

Notch

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:

16 Bit
Card

Viewport

RamWorks II

Ribbon
Cable

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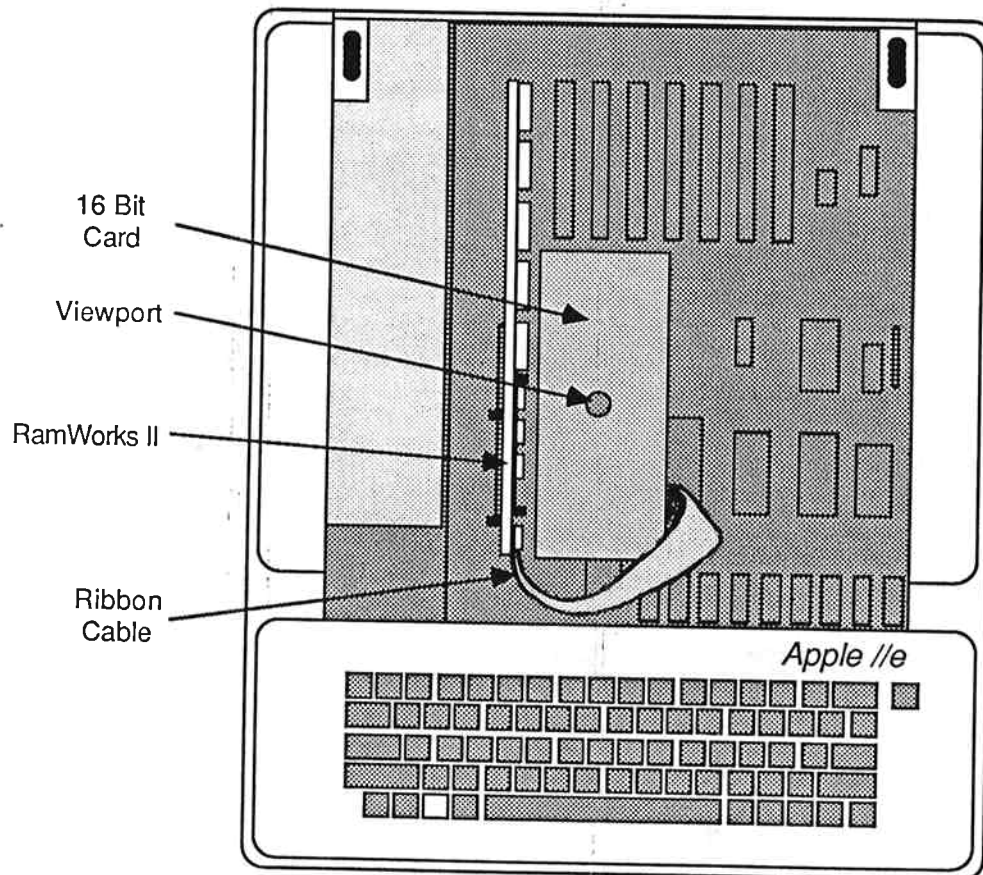
- Position the keyboard end of the RamWorks II card near the installed 16 Bit Card. Install the free end of the 16 Bit Card ribbon cable to the RamWorks II socket marked "CPU." Verify that all header connector pins are fully seated in the socket and that the cable is installed as shown in Illustration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

Illustration 6:
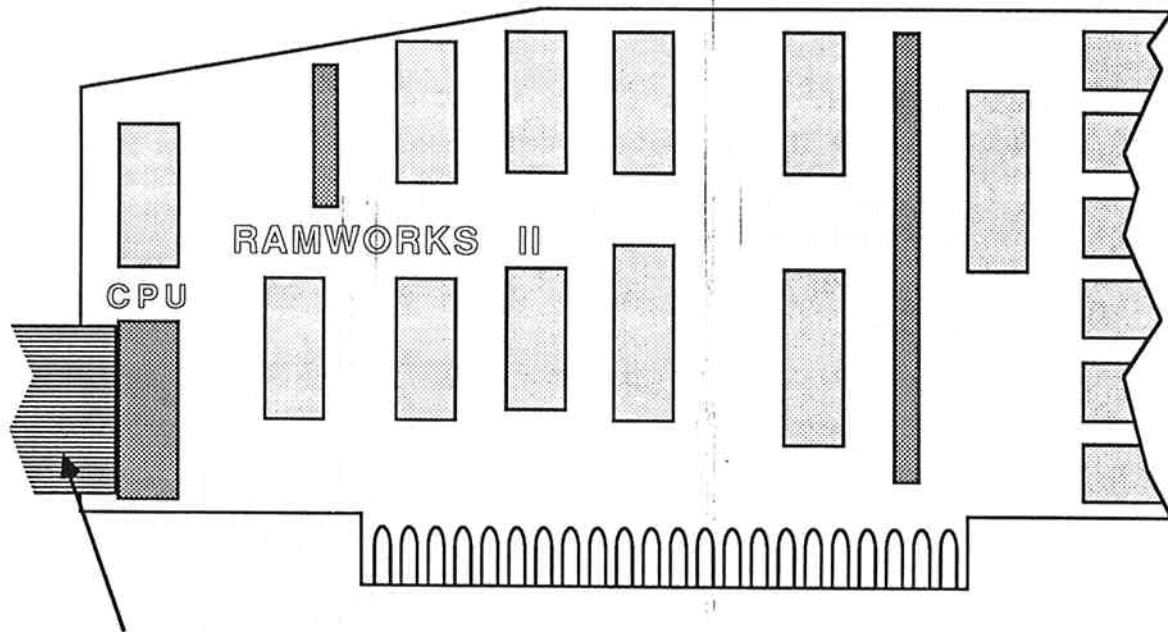
RAMWORKS II
CPU

Ribbon cable from
P1 of 16 Bit Card.

- 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:

Install
ribbon cable
connector from P2
of 16 Bit Card

Remove
PAL 16L8
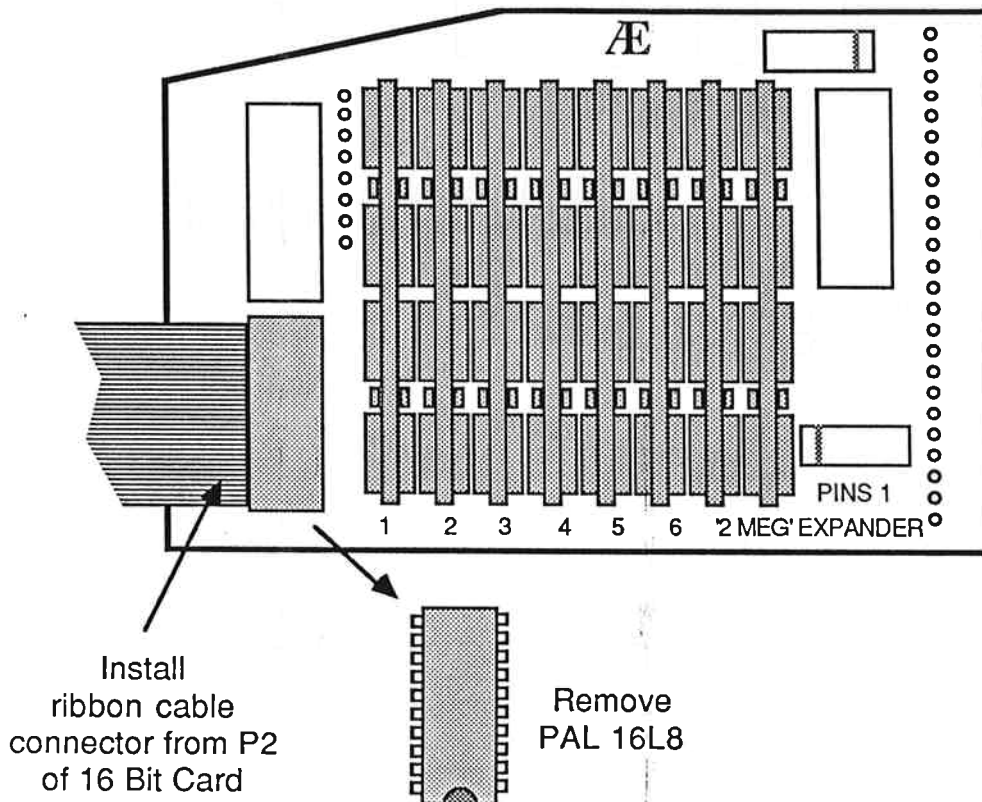
The 16 Bit Card will allow you to address up to 16 Meg linearly, using the 65816 processor's native mode of operation. In 65C02 emulation mode, the memory on the Ramworks II card will look and act exactly like the memory on a Ramworks II without the 16 Bit Card installed, with one exception: with the 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.
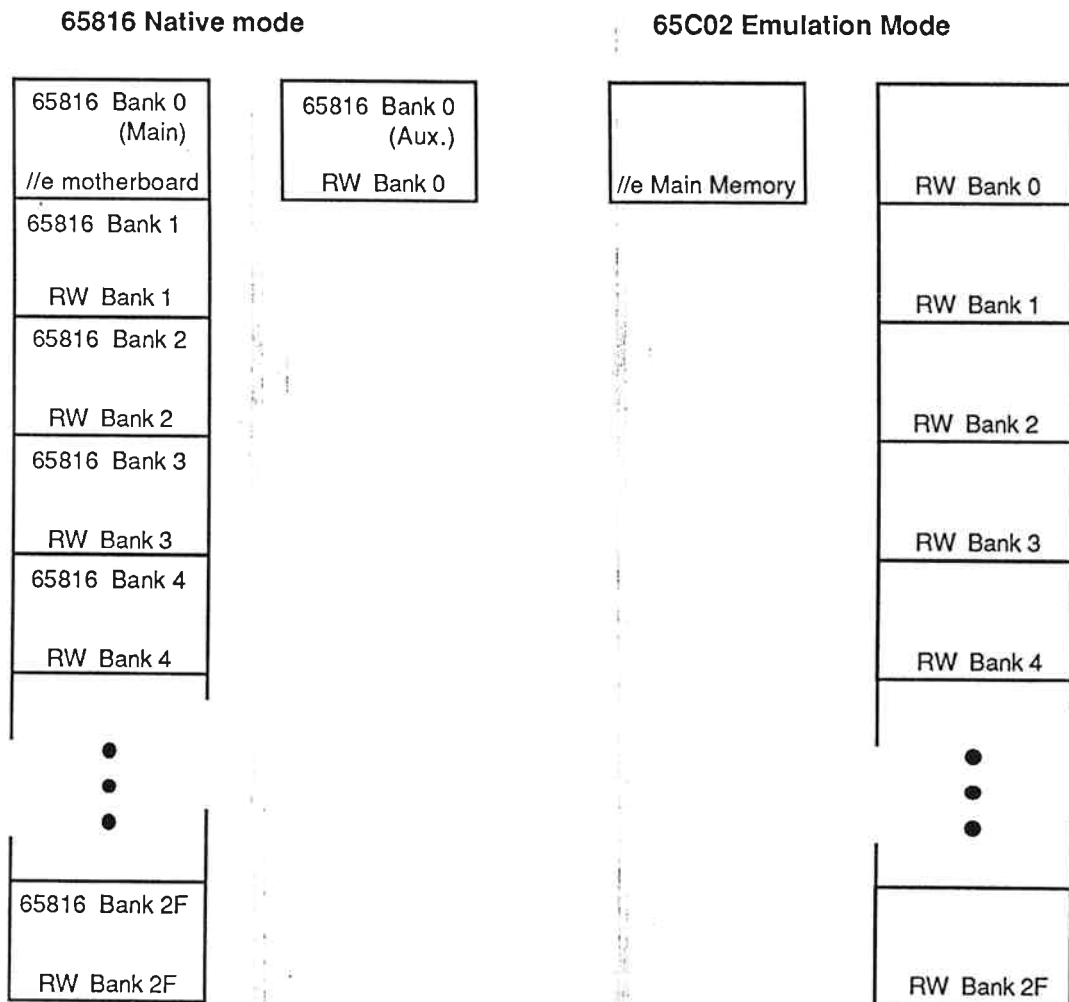
When the 16 Bit Card is installed and running in the 65C02 emulation mode, the softswitches still work exactly as they do without the 16 Bit Card. However, when the processor is in the 65816 native mode accesses to the hardware page can only be accomplished from 65816 BANK 0. Any bank other than 65816 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 auxillary 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

65816 Native mode

| 65816 Bank 0 (Main) |
| //e motherboard |
| 65816 Bank 1 |
| RW Bank 1 |
| 65816 Bank 2 |
| RW Bank 2 |
| 65816 Bank 3 |
| RW Bank 3 |
| 65816 Bank 4 |
| RW Bank 4 |
| ⋮ |
| 65816 Bank 2F |
| RW Bank 2F |

| 65816 Bank 0 (Aux.) |
| RW Bank 0 |

65C02 Emulation Mode

| //e Main Memory |

| RW Bank 0 |
| RW Bank 1 |
| RW Bank 2 |
| RW Bank 3 |
| RW Bank 4 |
| ⋮ |
| RW Bank 2F |

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

8 BITS	8 BITS		8 BITS
Data Bank Reg. (DBR)	X Register Hi (XH)	(X)	X Register Low (XL)
Data Bank Reg. (DBR)	Y Register Hi (YH)	(Y)	Y Register Low (YL)
00	Stack Register Hi (SH)	(S)	Stack Reg. Low (SL)
= 6502 Registers	Accumulator (B)	(C)	Accumulator (A)
Program Bank Reg. (PBR)	Program (PCH)	(PC)	Counter (PCL)
00	Direct Reg. Hi (DH)	(D)	Direct Reg. Low (DL)

Status Register Coding

```
        STATUS REG. (P)
        | 1 | B |     | E |  ──▶ EMULATION 1 = 6502
        N V M X D I Z C              0 = NATIVE
                    │ └─CARRY        1 = TRUE
                    └──ZERO          1 = RESULT ZERO
                  └────IRQ DISABLE   1 = DISABLE
                └──────DECIMAL MODE  1 = TRUE
              └────────INDEX REG. SELECT   1 = 8 BIT, 0 = 16 BIT
            └──────────MEMORY SELECT       1 = 8 BIT, 0 = 16 BIT
          └────────────OVER FLOW    1 = TRUE
        └──────────────NEGATIVE     1 = NEGATIVE
```
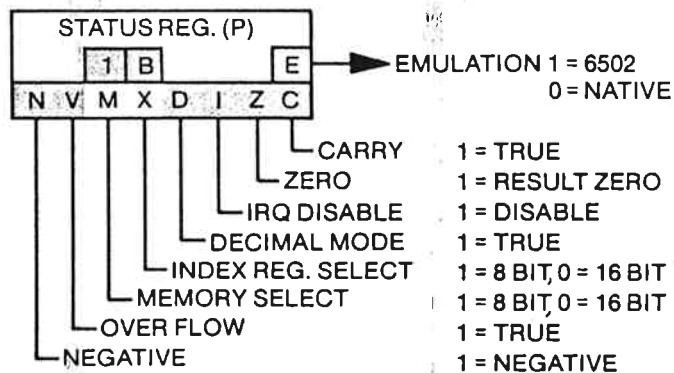
Functional Description

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.
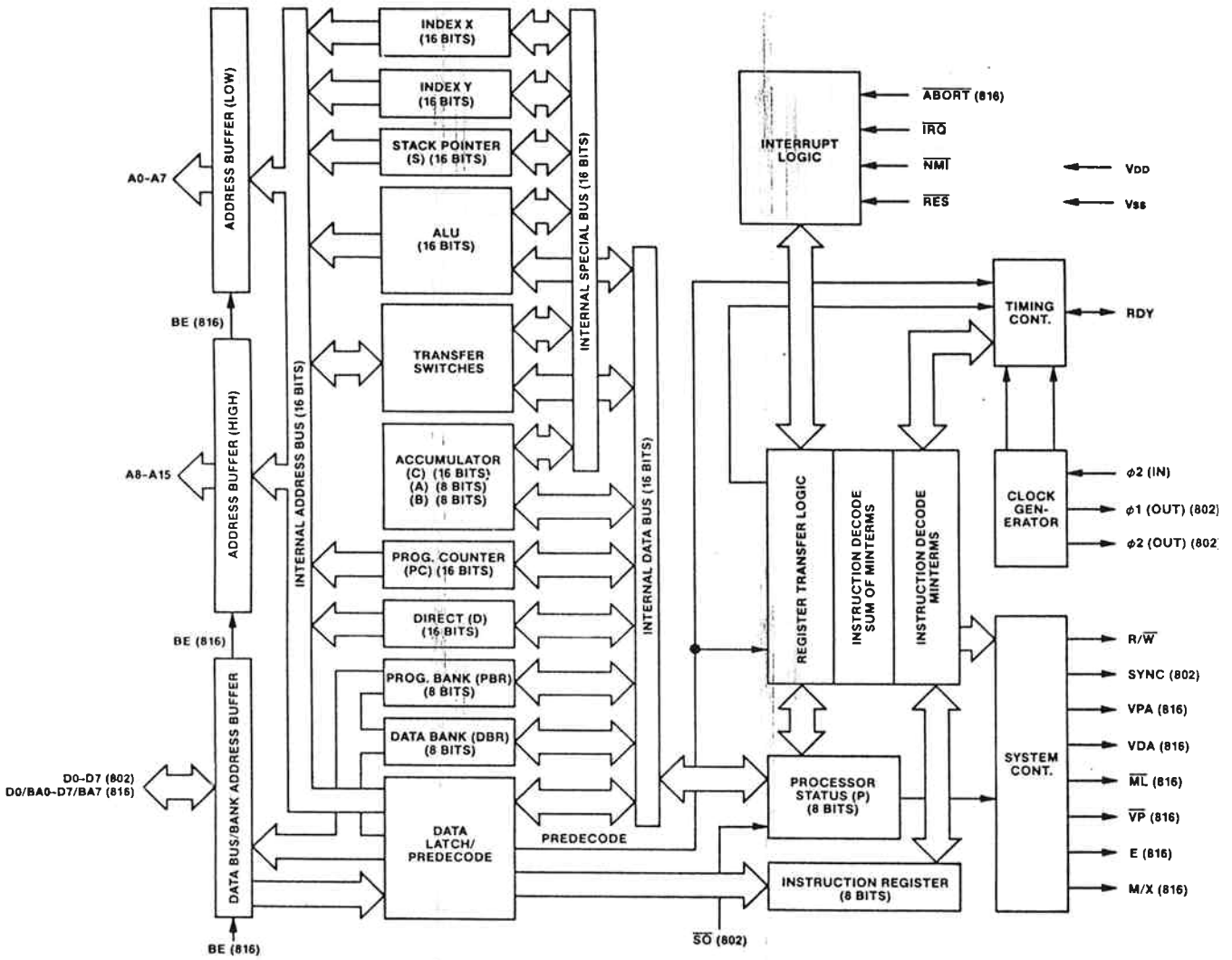
Block Diagram — Internal Architecture

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 Register)	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 IRQ, NMI, ABORT. 00FFE6, 7 (E = 0) X = X on Stack always.	FFFE,F BRK bit = 0 on stack if IRQ, NMI.	FFFE,F BRK bit = 0 on stack if IRQ, 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 (ML)	ML = 0 during Read, Modify and Write cycles.	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/W During Read-Modify-Write Instructions	E = 1, R/W = 0 during Modify and Write cycles. E = 0, R/W = 0 only during Write cycle.	R/W = 0 only during Write cycle	R/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
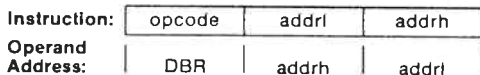
3. Absolute Long—al
The second, third, and fourth byte of the instruction form the 24-bit effective address.

Instruction:	opcode	addrl	addrh	baddr
Operand Address:	baddr	addrh	addrl	
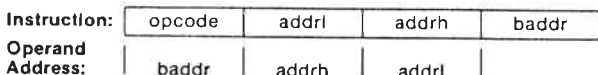
4. Direct—d
The second byte of the instruction is added to the Direct Register (D) to form the effective address. An additional cycle is required when the Direct Register is not page aligned (DL not equal 0). The Bank register is always 0.

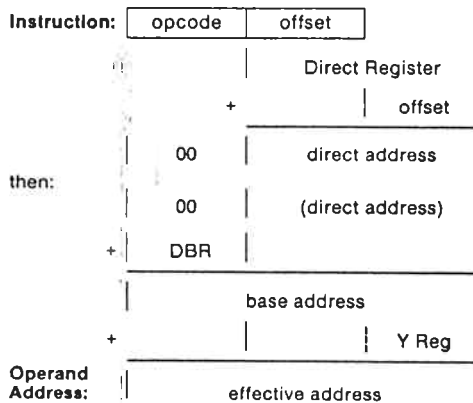| Instruction:	opcode	offset
		Direct Register
	+	offset
Operand Address:	00	effective address
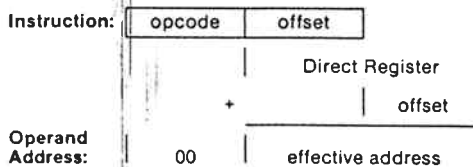
5. Accumulator—A
This form of addressing always uses a single byte instruction. The operand is the Accumulator.

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

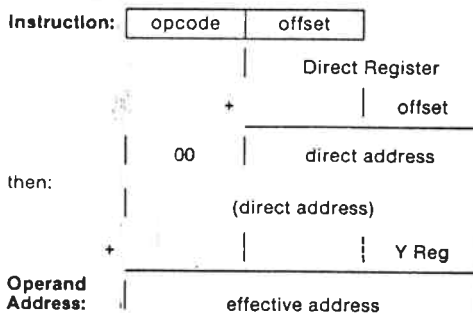### 9. Direct Indexed Indirect—(d,x)
This address mode is often referred to as Indirect,X. The second byte of the instruction is added to the sum of the Direct Register and the X Index Register. The result points to the low-order 16 bits of the effective address. The Data Bank Register contains the high-order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

Instruction: | opcode | offset |

    Direct Register
  + | offset |

         | direct address |
  + | X Reg |

then:
  | 00 | address |
  | 00 | (address) |
  + | DBR |

Operand Address: | effective address |
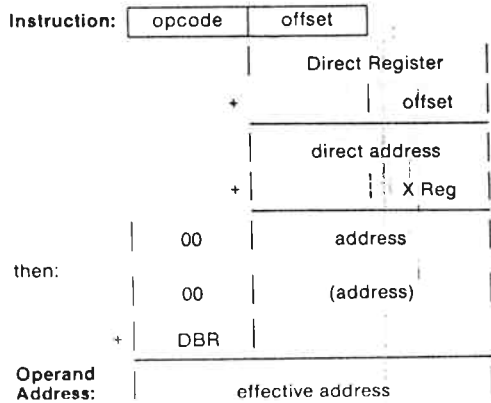
10. Direct Indexed With X—d,x

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.

Instruction: | opcode | offset |

         Direct Register
  + | offset |

         | direct address |
  + | X Reg |

Operand Address: | 00 | effective address |

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.

Instruction: | opcode | offset |

         Direct Register
  + | offset |

         | direct address |
  + | Y Reg |

Operand Address: | 00 | effective address |
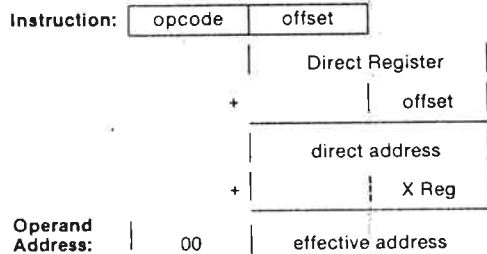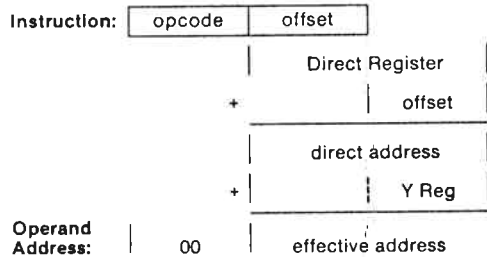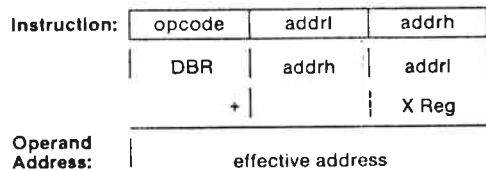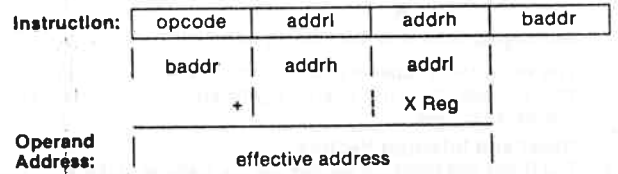
12. Absolute Indexed With X—a,x

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.

Instruction: | opcode | addrl | addrh |

  | DBR | addrh | addrl |
  + | X Reg |

Operand Address: | 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.

Instruction: | opcode | addrl | addrh | baddr |

  | baddr | addrh | addrl |
  + | X Reg |

Operand Address: | effective address |

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.

Instruction: | opcode | addrl | addrh |

  | DBR | addrh | addrl |
  + | Y Reg |

Operand Address: | effective address |
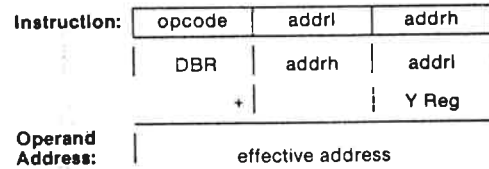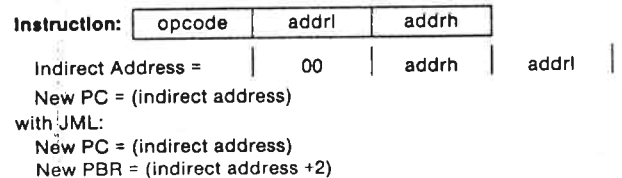
15. Program Counter Relative—r

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.

Instruction: | opcode | addrl | addrh |

Indirect Address = | 00 | addrh | addrl |

New PC = (indirect address)
with JML:
New PC = (indirect address)
New PBR = (indirect address +2)

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.

Instruction: | opcode | offset |

         Direct Register
  + | offset |

  | 00 | direct address |

then:
  | 00 | (direct address) |
  + | DBR |

Operand Address: | effective address |
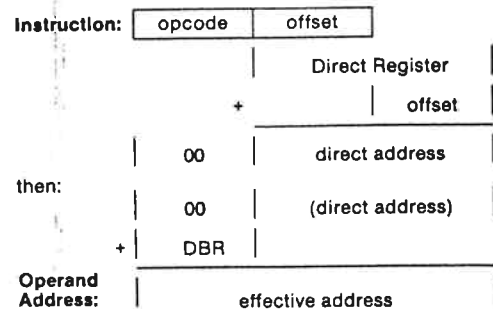
19. Direct Indirect Long—[d]

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

```
Instruction:  | opcode | offset |

                      |       Direct Register       |
             +        |            offset           |
             |   00   |        direct address       |
then:

Operand
Address:      |          (direct address)          |
```
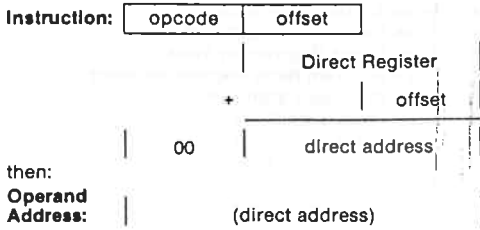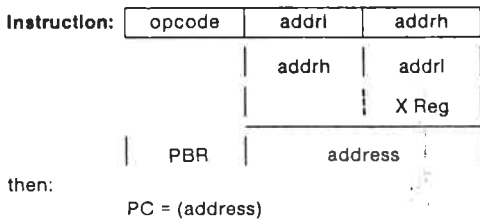
20. Absolute Indexed Indirect—(a,x)

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.

```
Instruction:  | opcode | addrl | addrh |

                      | addrh  | addrl |
                      |         X Reg   |
             |  PBR   |      address    |
then:

                PC = (address)
```
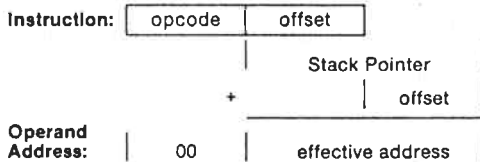
21. Stack—s

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.
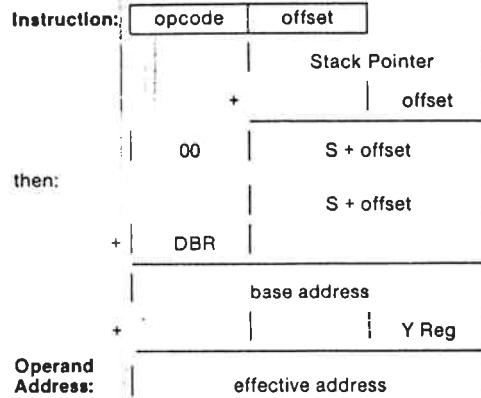
22. Stack Relative—d,s

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.

```
Instruction:  | opcode | offset |

                      |       Stack Pointer       |
             +        |          offset           |
Operand
Address:      |  00   |     effective address     |
```

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.

```
Instruction:  | opcode | offset |

                      |       Stack Pointer       |
             +        |          offset           |
             |  00   |        S + offset          |
then:

                      |        S + offset          |
          + |   DBR   |

                      |         base address       |
          +           |              Y Reg         |

Operand
Address:      |        effective address          |
```

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.

```
Instruction:  | opcode | dstbnk | srcbnk |

                       dstbnk  →  DBR
Source
Address:               | srcbnk |    X Reg    |
Destination
Address:               |  DBR   |    Y Reg    |
```
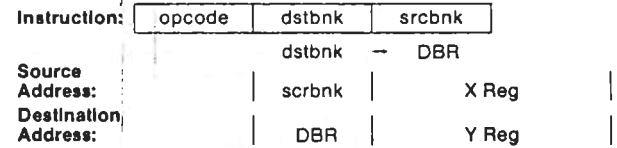
Increment (MVN) or decrement (MVP) X and Y.
Decrement C (if greater than zero), then PC+3 → PC.

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			E = 0		
OOFFFE,F —$\overline{\text{IRQ}}$/BRK	Hardware/Software		OOFFEE,F —$\overline{\text{IRQ}}$	Hardware	
OOFFFC,D—$\overline{\text{RESET}}$	Hardware		OOFFEC,D—(Reserved)		
OOFFFA,B —$\overline{\text{NMI}}$	Hardware		OOFFEA,B —$\overline{\text{NMI}}$	Hardware	
OOFFF8,9 —$\overline{\text{ABORT}}$	Hardware		OOFFE8,9 —$\overline{\text{ABORT}}$	Hardware	
OOFFF6,7 —(Reserved)			OOFFE6,7 —BRK	Software	
OOFFF4,5 —COP	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.

17

Opcode Matrix

LSD

MSD	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		ADDRESSING MODE
	★ = New W65C816/802 Opcodes	
	● = New W65C02 Opcodes	
BASE NO. BYTES	Blank = NMOS 6502 Opcodes	BASE NO. CYCLES