CHAPTER 2
SOFTWARE DESCRIPTION

2.1 INTRODUCTION

During the early 1970's, microprocessors (MPU) and microcomputers (MCU) helped ease the shortage of hardware designers by providing the hardware with more intelligence. However, because the power of any MPU or MCU is the result of the software programs, a shortage of software engineers was created. Thus, as MPUs and MCUs reduced hardware costs, software development costs rose. As a result, the system designer of today must carefully weigh the software and support costs of his system. Processors such as those of the M6805 HMOS/M146805 CMOS Family, which are designed to include the programming features inherited from minicomputers, require less effort from the programmer and make system design much more efficient. The importance of "user-friendly" software in mini and mainframe computers is a widely accepted fact. Easy-to-use software is the key to writing and maintaining efficient programs.

The M6805 HMOS/M146805 CMOS Family architecture is based upon the Von Neumann model which places all data, program, and I/O spaces into a single address map. Thus, since only a single address map must be supported, very few special purpose instructions are necessary in the M6805 HMOS/M146805 CMOS Family instruction set. The overall result of this is a small, very regular, and easy-to-remember instruction set.

A regular instruction set is symmetrical in that, for most instructions, there is a complement instruction. Some of these instructions (plus complements) are listed below.

LDA	—	STA	Load and Store
INC	—	DEC	Increment and Decrement
BEQ	—	BNE	Branch If Equal and Branch If Not Equal
ADD	—	SUB	Add and Subtract
AND	—	ORA	Logic AND and Logic OR
BCLR	—	BSET	Bit Clear and Bit Set
ROR	—	ROL	Rotate Right and Rotate Left
JSR	—	RTS	Jump-To-Subroutine and Return-From-Subroutine

The symmetry provided by the M6805 HMOS/M146805 CMOS Family instruction set means that the programmer need only remember about 30 to 40 separate instructions to know the entire instruction set. The M6805 HMOS Family has 59 instructions in its instruction set; whereas, the M146805 CMOS Family has 61. The two additional instructions for the M146805 CMOS Family are the STOP and WAIT instructions which enable one of the CMOS low-power standby modes.

The instruction set is expanded by the use of a variety of versatile addressing modes. The addressing modes, which are part of the minicomputer heritage of M6805 HMOS/ M146805 CMOS Family, expand the instruction set by allowing the programmer to specify how the data for a particular instruction is to be fetched. As illustrated in the Op-code Map of Appendix I, The 59/61 separate instructions, enhanced by the seven address-ing modes, expand into 207/209 opcodes; however, the programmer need only remember 66/68 items (59/61 instructions plus seven addressing modes) instead of 207/209.

2.2 REGISTER SET

Each M6805 HMOS/M146805 CMOS Family member contains five registers as shown in Figure 2-1. The accumulator (A) and index register (X) are used as working program registers. The condition code register (CC) is used to indicate the current status of the processor program. The program counter (PC) contains the memory address of the next instruction that the processor is to execute. The stack pointer (SP) register contains the address of the next free stack location. For more information concerning each register, see the section below describing that register.

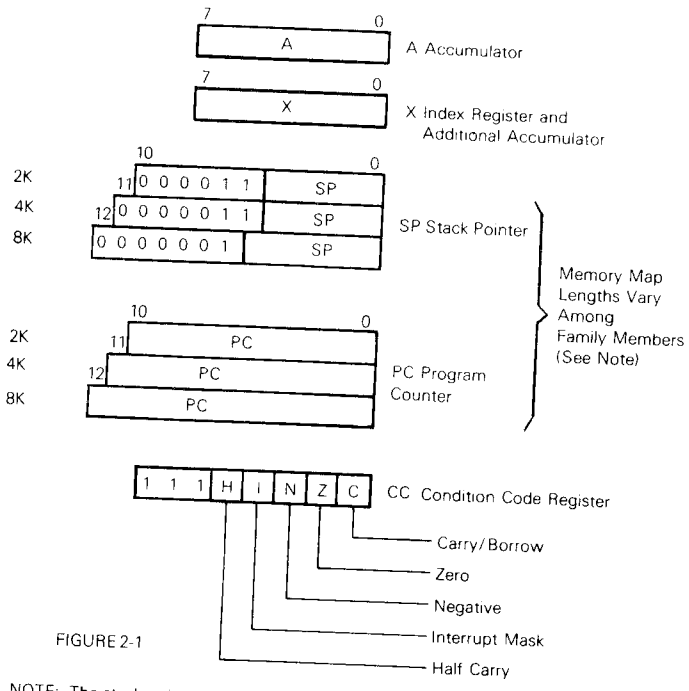


FIGURE 2-1

NOTE: The stack pointer and program counter size is determined by the memory size that the family member device can access; e.g., an 8K memory map requires a 13-bit stack pointer and program counter.

Figure 2-1. M6805 HMOS/M146805 CMOS Family Register Architecture

2.2.1 Accumulator (A)

The A register is a general purpose 8-bit register that is used by the program for arithmetic calculations and data manipulations. The full set of read/modify/write instructions operates on the A register. The accumulator is used in the register/memory instructions for data manipulation and arithmetic calculation. Refer to the Instruction Set Summary discussion later in this chapter for information about the read/modify/write and register/memory instruction. An example using the accumulator to add the contents of two memory locations is shown below.

```
B6  50          LDA  $50     Load accumulator with contents of memory location $50.
BB  87          ADD  $87     Add the contents of memory location $87 to the accumulator.
B7  3C          STA  $3C     Store the accumulator contents in memory location $3C.
```

2.2.2 Index Register (X)

The index register is used in the indexed modes of addressing or used as an auxiliary accumulator. It is an 8-bit register and can be loaded either directly or from memory, have its contents stored in memory, or its contents can be compared to memory.

In indexed instructions, the X register provides an 8-bit value, that is added to an instruction-provided value, to create an effective address. The indexed addressing mode is further described in the Addressing Modes paragraph of this chapter.

The X register is also used in the M6805 HMOS/M146805 CMOS Family for limited calculations and data manipulation. The full set of read/modify/write instructions operates on the X register as well as the accumulator. Instruction sequences which do not use the X register for indexed addressing may use X as a temporary storage cell, or accumulator.

The following example shows a typical use of the index register in one of the indexed addressing modes. The example performs a block move that is BCNT in length.

```
        LDX  #BCNT       GET LENGTH
REPEAT  LDA  SOURCE,X    GET DATA
        STA  DESTIN,X    STORE IT
        DECX             NEXT
        BNE  REPEAT      REPEAT IF MORE.
```

The X register is also useful in counting events since it can be incremented or decremented. The INCX or DECX instructions can be used to control the count. By either decrementing or incrementing the X register, starting at a known value, and then comparing the X register contents to the contents of a memory location (or a specific number) a loop can be ended or a branch taken after a certain number of events.

The following routine uses the index register as a counter for a keypad debounce routine of CNT X 6, CMOS (or CNT X 8, HMOS).

```
AE   FF   DBNCE   LDX   #CNT      CNT = 255 in this example
5A        AGAIN   DECX
26   FD           BNE   AGAIN
```

2.2.3 Program Counter (PC)

The PC contains the memory address of the next instruction that is to be fetched and executed. Normally, the PC points to the next sequential instruction; however, the PC may be altered by interrupts or certain instructions. During a valid interrupt, the PC is loaded with the appropriate interrupt vector. The jump and branch instructions modify the PC so that the next instruction to be executed is not necessarily the next instruction in physical memory. The actual size of the PC depends upon the size of the address space of the individual family members and currently ranges from 11 to 13 bits.

2.2.4 Stack Pointer (SP)

The stack array (stack) is an area of memory in RAM used for the temporary storage of important information. It is a sequence of registers (memory locations) used in a last-in-first-out (LIFO) fashion. A stack pointer is used to specify where the last-in entry is located or where the next-in entry will go. Since the stack must be written to, as well as read, it must be located in RAM.

Interrupts and subroutines make use of the stack to temporarily save important data. The SP is used to automatically store the return address (two bytes of the PC) on subroutine calls and to automatically store all registers (five bytes; A, X, PC and CC) during interrupts. The saved registers may be interleaved on the stack (nested), thus allowing for: (1) nesting of subroutines and interrupts, (2) subroutines to be interrupted, and (3) interrupts to call subroutines. The nesting of subroutines and interrupts can only occur to some maximum amount, which is described below.

Since the M6805 HMOS/M146805 CMOS is a family of devices, the actual size of the stack pointer may vary with memory size of the particular family member (see appropriate data sheets). But from the programmer's perspective, the stack pointers all appear similar on the different members. Both the hardware $\overline{\text{RESET}}$ pin and the reset stack pointer (RSP) instruction reset the stack pointer to its maximum value ($7F on all current members). The stack pointer on the M6805 HMOS/M146805 CMOS Family always points to the next free location on the stack. Each "push" decrements the SP while each "pull" increments it ("push" and "pull" are not available as user instructions in the M6805 HMOS/M146805 CMOS Family).

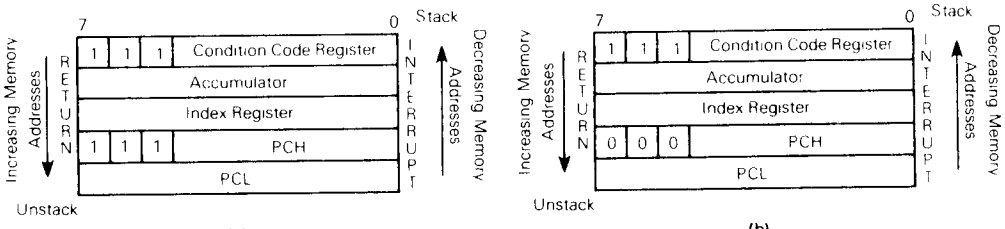
Nested subroutine calls and interrupts must not underflow the SP. The usable stack length will vary between devices as well as between the M6805 HMOS and M146805 CMOS Families. In the M6805 HMOS Family, the usable stack length is $2^n - 1$ (where n = the number of bits in the stack pointer); however, in the M146805 CMOS Family the

usable stack length is 2^n (where n = number of bits in the stack pointer). When the allowable stack length is exceeded, the SP will wrap around to the top of stack. This condition of stack underflow should be avoided since the previously stacked data will be lost. An example of calculating the usable stack length for an M6805 HMOS Family device with a 5-bit stack pointer is: $2^5 - 1$ or 31 bytes maximum. However, for an M146805 CMOS Family device, with a 6-bit stack pointer, the calculation is: 2^6 or 64 bytes maximum.

A 5-bit M6805 HMOS Family device SP accommodates up to 15 nested subroutine calls (30 bytes), six interrupts (30 bytes), or a mixture of both. The programmer must exercise care when approaching the underflow threshold. When the SP underflows it will wrap around, and the contents more than likely are lost. The stack limit in the 5-bit M6805 HMOS Family example above is thus stated to be 31, not 32, bytes. The stack limit is well beyond the needs required by most programs. A maximum subroutine nesting of five levels (10 bytes) coupled with one interrupt level (five bytes) occupies only 15 bytes of stack space. The allowed stack length is typically traded off against the needed data RAM space.

In the M6805 HMOS/M146805 CMOS Family, the stack builds in the direction of decreasing address; therefore, the SP always points to the next empty location on the stack. The SP is decremented each time a data byte is pushed onto the stack and it is incremented each time a data type is pulled from the stack. The SP is only changed during certain operations and, except for the RSP instruction, it is not under direct software control. During external or power-on reset, and during a reset pointer (RSP) instruction, the SP is set to its upper limit ($7F).

The order in which bytes are stored onto and retrieved from the stack is shown in Figure 2-2. Note that the PC has a number of fixed and variable bits. The number of variable bits depends upon the size of the memory available in a particular family member (see Figure 2-1 for this relationship).



NOTES:

1. Since, in all family devices, the stack pointer decrements during pushes, the PCL is stacked first, followed by the PCH, etc. Pulling from the stack is in the reverse order.
2. Fixed bits in the M6805 HMOS Family PC are always set, whereas, the M146805 CMOS Family PC fixed bits are always clear.

Figure 2-2. Stacking Order

2.2.5 Condition Code Register (CC)

The M6805 HMOS/M146805 CMOS Family uses five condition code flag bits, labeled H, I, N, Z, and C, which reside in the CC register. The three MSBs of the CC register are all ones which fill the register to eight bits.

The function of the condition codes is to retain information concerning the results of the last executed data reference instruction. The effect of an instruction on each condition code is shown, together with the instruction, in Appendix D. Any bit or a combination of bits, except the I bit, are testable using the conditional branch instructions. See the Addressing Modes paragraph for more information.

2.2.5.1 CARRY (C). The C bit is set if a carry or borrow out of the 8-bit ALU occurred during the last arithmetic operation. It is also set during shift, rotate, and bit test instructions.

The C bit is mainly set in one of six ways.

1. It is set during an add instruction if the result of the additions produces a carry out of the 8-bit ALU (arithmetic logic unit).
2. For subtraction and comparison instructions, it is set when the absolute value of the subtrahend is larger than the absolute value of the minuend. This generally implies a borrow.
3. It is changed during shift and rotate instructions. For these instructions the bit shifted out of the accumulator becomes the C bit.
4. It is set when an SEC instruction is executed.
5. It is set when a COM instruction is executed.
6. It is set if a bit test and branch bit is set.

Two instructions, add with carry (ADC) and subtract with carry (SBC), use the carry bit as part of the instruction. This simplifies the addition or subtraction of numbers that are longer than eight bits. The carry bit may be tested with various conditional branch instructions.

2.2.5.2 ZERO (Z). The Z bit is set if the result of the last data manipulation, arithmetic, or logical operation was zero. The Z bit is set only if all eight bits of the result are zero; otherwise, it is cleared.

The Z bit can be used to cause a branch with the BHI, BLS, BNE, or BEQ instructions. When the BHI instruction is used, both the C bit and Z bit are used for the branch.

The Z bit can be used to initiate a branch after the A or X contents equal the contents of a memory location. For example, the accumulator can be compared to the contents of a memory location and when the eight resultant bits are all zeros (Z bit set), a branch would result with the BEQ instruction. Conversely, if the same comparison were made and a BNE instruction were used, a branch would result after each compare until the eight resultant bits were all zeros (Z bit set).

2.2.5.3 NEGATIVE (N). The N bit is set when bit seven of the result of the last data manipulation, arithmetic, or logical operation is set. This indicates that the result of the operation was negative. The N bit is cleared by the CLR and LSR instructions. In all other instructions affecting the N bit, its condition is determined by bit 7 of the result.

The N bit can be used to cause a branch if it is set by using the BMI instruction. Likewise, the N bit can be used for a branch if it cleared by using the BPL instruction. In one case it is tested for a negative result and in the other it is tested for a positive result.

The N bit can be used to initiate a branch after a comparison of two numbers. For example, the contents of the X register could be compared to the contents of memory location M and a branch taken if N = 1. In using the CPX instruction, the N bit would remain clear and no branch is taken, as long as the X register contents were greater than or equal to the contents of M; however, if the X register contents become less than the contents of M, the N bit becomes 1 and a branch could be initiated (using BMI instruction).

2.2.5.4 HALF CARRY (H). The H bit is set when a carry occurs between bits 3 and 4 during an ADD or ADC instruction. The half-carry flag may be used in BCD addition subroutines since each binary-coded-decimal digit is contained either in the 0-3 (least significant) or 4-7 bits. Thus, when the sum of the two least significant BCDs results in a carry out of bit position 3 into bit position 4, the H bit is set. Chapter 3 describes a routine which uses the H bit to emulate the MC6800 DAA (decimal adjust) instruction.

2.2.5.5 INTERRUPT MASK (I). When the I bit is set, the external interrupt and timer interrupt are masked (disabled). Clearing the I bit allows interrupts to be enabled. If an interrupt occurs while the I bit is set, the interrupt is latched internally and held until the I bit is cleared. The interrupt vector is then serviced normally.

Except for when an external interrupt ($\overline{INT}$ or $\overline{IRQ}$) is applied, the I bit is controlled by program instructions. Some program instructions change the I bit only as a result of the instruction, whereas, others cause it to change as a part of the instruction. For example, CLI clears the I bit and SEI sets the I bit; however, SWI automatically sets the I bit as part of the interrupt instruction. The STOP and WAIT instructions in M146805 CMOS Family parts also automatically set the I bit as part of instruction. See the Interrupts section of Chapter 4 for more information.

NOTE

The SWI instruction and $\overline{RESET}$ are the only non-maskable interrupts in the M6805 HMOS/M146805 CMOS Families.

2.3 ADDRESSING MODES

The power of any computer lies in its ability to access memory. The addressing modes of the processor provide that capability. The M6805 HMOS/M146805 CMOS Family has a set of addressing modes that meets these criteria extremely well.

The addressing modes define the manner in which an instruction is to obtain the data required for its execution. An instruction, because of different addressing modes, may access its operand in one of up-to-five different addressing modes. In this manner, the

addressing modes expand the basic 59 M6805 HMOS Family instructions (61 for M146805 CMOS Family) into 207 separate operations (209 for M146805 CMOS Family). Some addressing modes require that the 8-bit opcode be accompanied by one or two additional bytes. These bytes either contain the data for the operations, the address for the data, or both.

In the addressing mode descriptions which follow, the term effective address (EA) is used. The EA is the address in memory from which the argument for an instruction is fetched or stored. In two-operand instructions, such as add to accumulator (ADD), one of the effective operands (the accumulator) is inherent and not considered an addressing mode per se.

Descriptions and examples of the various modes of addressing the M6805 HMOS/ M146805 CMOS Family are provided in the paragraphs which follow. Several program assembly examples are shown for each mode, and one of the examples is described in detail (ORG, EQU, and FCB are assembler directives and not part of the instruction set). Parentheses are used in these descriptions/examples of the various addressing modes to indicate "the contents of" the location or register referred to; e.g., (PC) indicates the contents of the location pointed to by the PC. The colon symbol (:) indicates a concatenation of bytes. In the following examples, the program counter (PC) is initially assumed to be pointing to the location of the first opcode byte. The first PC + 1 is the first incremental result and shows that the PC is pointing to the location immediately following the first opcode byte.

The information provided in the program assembly examples uses several symbols to identify the various types of numbers that occur in a program. These symbols include:

1. A blank or no symbol indicates a decimal number.
2. A $ immediately preceding a number indicates it is a hexadecimal number; e.g., $24 is 24 in hexadecimal or the equivalent of 36 in decimal.
3. A # indicates immediate operand and the number is found in the location following the opcode.

There are seven different addressing modes used in the M6805 HMOS/M146805 CMOS Family, namely: inherent, immediate, direct, extended, indexed, relative, and bit manipulation. The indexed and bit manipulation addressing modes contain additional subdivisions to increase their flexibility; i.e., three additional for the indexed mode and two for bit manipulation. Each of these programming modes is discussed in the paragraphs which follow. The cycle-by-cycle description of each instruction in all possible addressing modes is included in Appendix G. This allows the processor bus activity and instruction operation relationship to be studied.

2.3.1 Inherent Addressing Mode

In this addressing mode there is no EA (effective address). Inherent address instructions are used when all information required for the instruction is already within the CPU, and no external operands, from memory or the program, are needed. Since all the information necessary to carry out the instruction is contained in the opcode, and no external

operands are needed, inherent instructions only require one byte. These one-byte instructions are shown in Appendix E as part of control and read/modify/write instruction tables.

The following is an example of a subroutine that clears all registers (accumulator and index) plus the C bit and then returns. Figure 2-3 shows an example of the steps required to perform the TAX instruction in the subroutine.

```
05B9  4F        CLEAR   CLRA    Clear Accumulator
05BA  97                TAX     Transfer Accumulator
                                Contents to Index Register
05BB  98                CLC     Clear the Carry Bit
05BC  81                RTS     Return from Subroutine
```

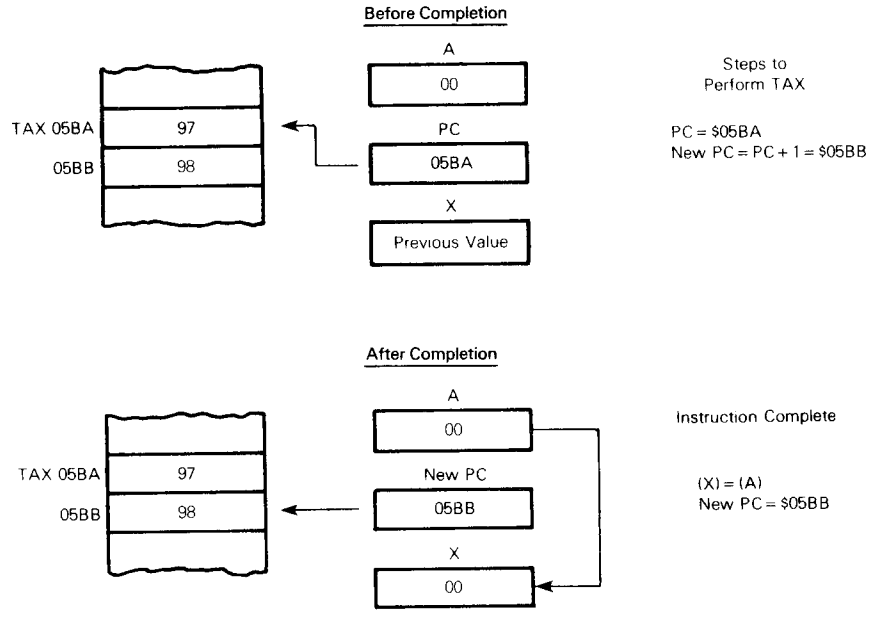


Figure 2-3. Inherent Addressing Mode Example

2.3.2 Immediate Addressing Mode

The EA of an immediate mode instruction is the location following the opcode. This mode is used to hold a value or constant which is known at the time the program is written, and which is not changed during program execution. These are two-byte instructions, one for the opcode and one for the immediate data byte. Immediate addressing may be used by any register/memory instructions as shown in Appendix E.

PC + 1 → PC
EA = PC
PC + 1 → PC

The following is an example which subtracts 5 from the contents of the accumulator and compares the results to 10. Figure 2-4 shows an example of the steps required to perform the SUB instruction.

```
05BC  B6   4B        LDA   $4B      Load Accumulator from RAM
05BE  A0   05        SUB   #5       Subtract 5 from Accumulator
05C0  A1   0A        CMP   #10      Compare Accumulator to 10
```

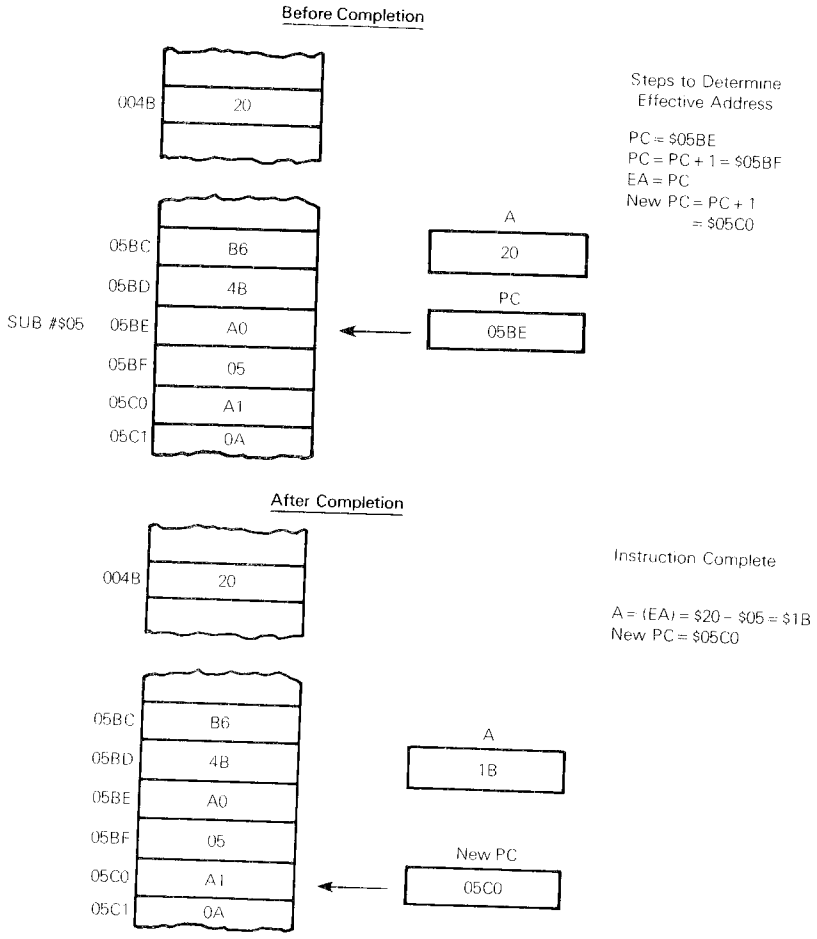


Figure 2-4. Immediate Addressing Mode Example

2.3.3 Extended Addressing Mode

The EA of an extended mode instruction is contained in the two bytes following the opcode. Extended addressing references any location in the M6805 HMOS/M146805 CMOS Family memory space, I/O, RAM, and ROM. The extended addressing mode allows an instruction to access all of memory. Also, since the two bytes following the opcode contain 16 bits, the addressing range of the M6805 HMOS/M146805 CMOS Family may be

extended in the future without affecting the instruction set or addressing modes. Extended addressing mode instructions are three bytes long, the one-byte opcode plus a two-byte address. All register/memory instructions, as shown in Appendix E, can use extended addressing.

PC + 1 → PC
EA = (PC): (PC + 1)
PC + 2 → PC

The following example loads the contents of a memory location (labeled COUNT) into the index register and then jumps to a subroutine to provide a delay. Figure 2-5 shows an example of the steps required to determine the EA from which to load the index register.

```
                0800  COUNT   EQU   $800
                1200  DELAY   EQU   $1200
0409  CE        0800          LDX   COUNT    Load Index Register with
                                               Contents of Location
                                               $800
040C  CD        1200          JSR   DELAY    Jump to Subroutine
                                               Located at $1200
```

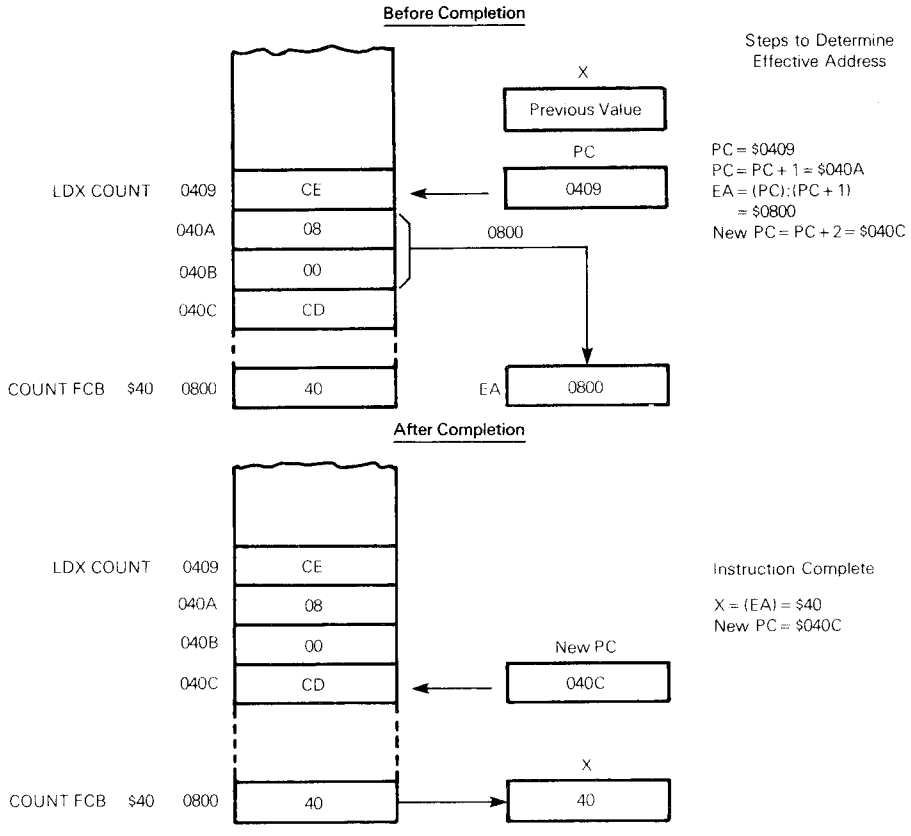


Figure 2-5. Extended Addressing Mode Example

2.3.4 Direct Addressing Mode

The direct addressing mode is similar to the extended addressing mode except only one byte is used to form the EA. Direct addressing allows an instruction to only access any location in page 0 (locations $00-$FF) with a two-byte instruction; therefore, the upper address bits are set to $00. Direct addressing may be used with any read-modify-write, or register/memory and bit manipulation instruction.

The following example adds two 16-bit numbers. The result is then placed in the location of the first number; however, if the result exceeds 16-bits the C bit will be set. Figure 2-6 illustrates the steps required to determine the EA from which to load the accumulator with the contents of NUM1 (first number).

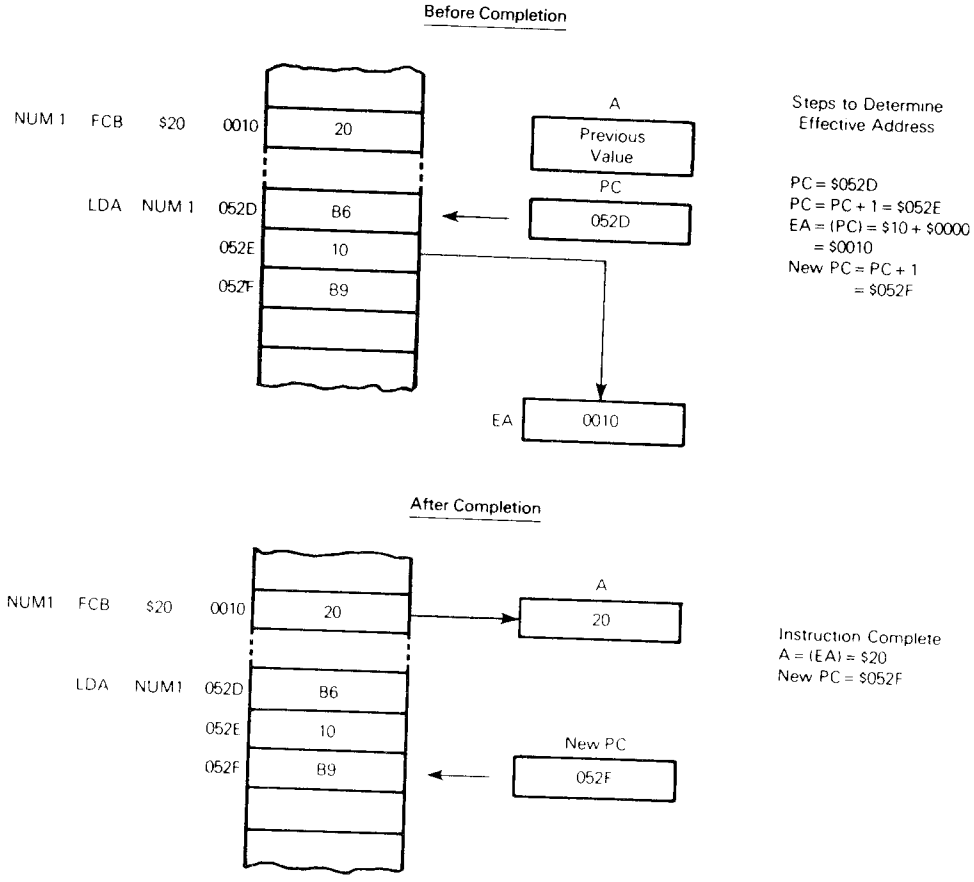


Figure 2-6. Direct Addressing Mode Example

```
                        ORG   $10
          NUM1          RMB   2
          NUM2          RMB   2
0527  B6  11            LDA   NUM1+1    Load Accumulator with Contents of Location $0011
0529  BB  13            ADD   NUM2+1    Add Contents of Location $0013 to Accumulator
052B  B7  11            STA   NUM1+1    Save Result in Location $0011
052D  B6  10            LDA   NUM1      Load Accumulator with Contents of Location $0010
052F  B9  12            ADC   NUM2      Add Contents of Location $0012 and C Bit to Accumulator
0531  B7  10            STA   NUM1      Save Result in Location $0010
```

2.3.5 Indexed Addressing Mode

In the indexed addressing mode, the EA is variable and depends upon two factors: (1) the current contents of the index (X) register and (2) the offset contained in the byte(s) following the opcode. Three types of indexed addressing exist in the M6805 HMOS/M146805 CMOS Family: no offset, 8-bit offset, and 16-bit offset. A good assembler should use the indexed addressing mode which requires the least offset. Either the no-offset or 8-bit offset indexed addressing mode may be used with any read-modify-write or register/memory instruction. The 16-bit offset indexed addressing is used only with register/memory instructions.

2.3.5.1 INDEXED — NO OFFSET. In this mode the contents of the X register is the EA; therefore, it is a one-byte instruction. This mode is used to create an EA which is pointing to data in the lowest 256 bytes of the address space, including: I/O, RAM, and part of ROM. It may be used to move a pointer through a table, point to a frequently referenced location (e.g., an I/O location), or hold the address of a piece of data that is calculated by a program. Indexed, no-offset instructions use only one byte: the opcode.

EA = X + $0000
PC + 1 → PC

In the following example, locations $45 to $50 are to be initialized with blanks (ASCII $20). Figure 2-7 illustrates the steps necessary to determine the EA from which to store the accumulator contents into a memory location pointed to by the index register.

```
05F0  AE  45            LDX   #$45      Initialize Index Register with $45
05F2  A6  20            LDA   #$20      Load Accumulator with $20
05F4  F7        REPEAT  STA   ,X        Store Accumulator Contents in Location Pointed to by Index Register
05F5  5C                INCX            Next Location
05F6  A3  51            CPX   #$51      Finished
05F8  26  FC            BNE   REPEAT    Repeat if More
```

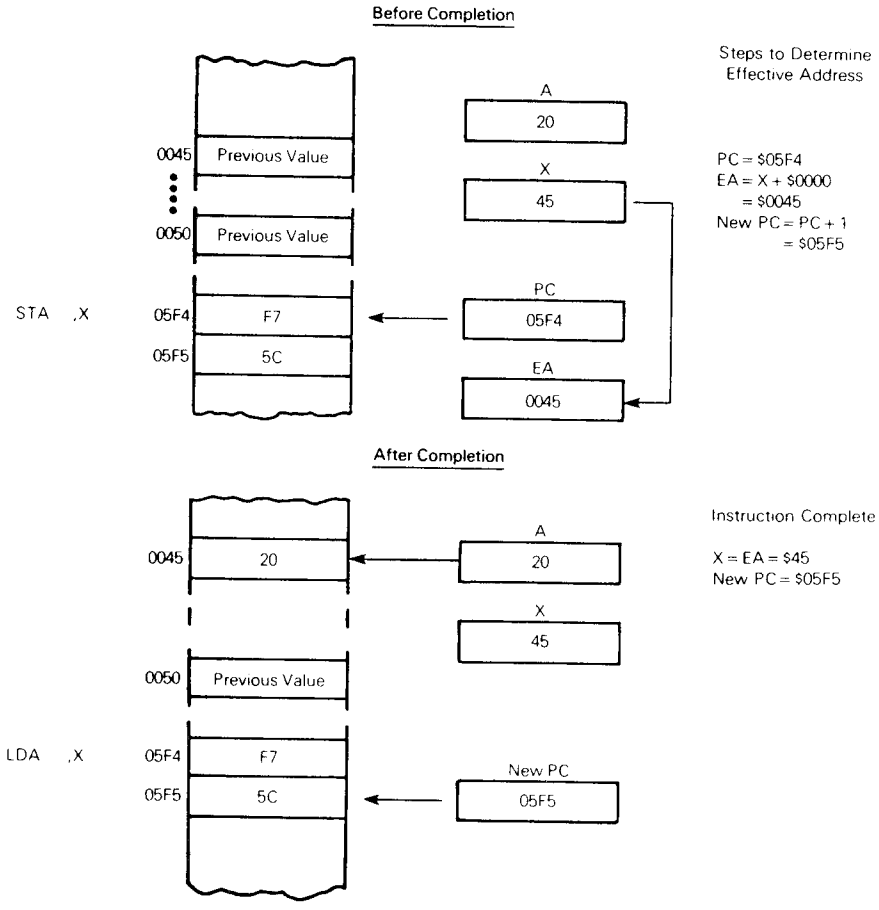


Figure 2-7. Indexed Addressing Mode, No Offset Example

2.3.5.2 INDEXED — 8-BIT OFFSET. To determine the EA in this addressing mode, the contents of the X register is added to the contents of the byte following the opcode. This addressing mode is useful in selecting the kth element of an n element table. To use this mode the table must begin in the lowest 256 memory locations, and may extend through the first 511 memory locations (1FE is the last location at which the instruction may begin) of the M6805 HMOS/M146805 CMOS Family. All indexed 8-bit offset addressing can be used for ROM, RAM, or I/O. This is a two-byte instruction with the offset contained in the byte following the opcode. Efficient use of ROM encourages the inclusion of as many tables as possible in page zero and page one.

PC + 1 → PC
EA = (PC) + X + $0000
PC + 1 → PC

The following subroutine searches a list, which contains 256 separate items, for the first occurrence of a value contained in the accumulator. The search starts at $80 and continues through $180 unless the accumulator contents matches one of the list items. Figure 2-8 shows the steps required to determine the EA of the next item to be compared.

```
                        LIST    EQU   $80
                                ORG   $075A
075A  5F                FIND    CLRX                Clear Index Register
075B  E1    80          REPEAT  CMP   LIST,X        Compare Accumulator to
                                                    Contents of Location $80 + X
075D  27    03                  BEQ   RETURN        Return if Match Found
075F  5C                        INCX                Else Next Item
0760  26    F9                  BNE   REPEAT        If 256 Items Checked then
                                                    Finish Else Repeat
0672  81                RETURN  RTS
```

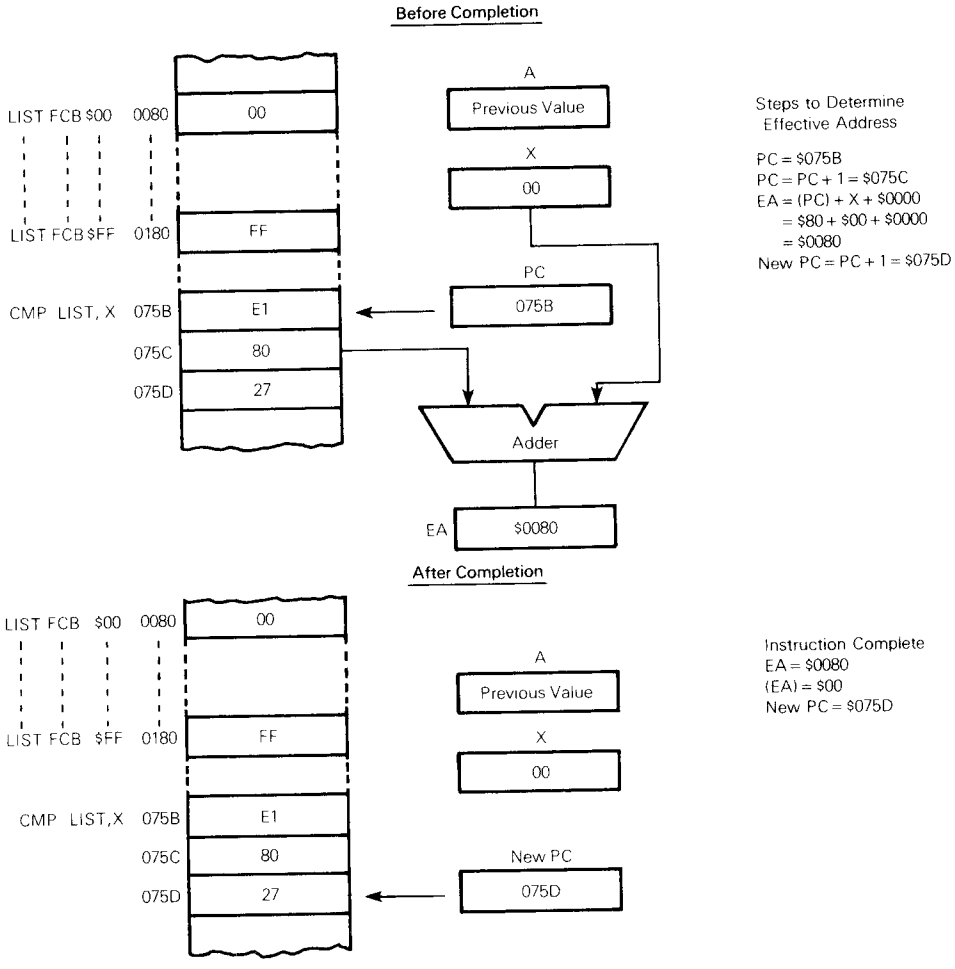


Figure 2-8. Indexed Addressing Mode, 8-Bit Offset Example

2.3.5.3 INDEXED — 16-BIT OFFSET. The EA for this two-byte offset addressing mode is calculated by adding the concatenated contents of the next two bytes following the op-code to the contents of the X register. This addressing mode is used in a manner similar to the indexed with 8-bit offset; except that since the offset is 16 bits, the tables being referenced can be anywhere in the M6805 HMOS/M146805 CMOS Family address space. For more details refer to the Indexing Compatibility paragraph below. This addressing mode is a three-byte instruction: one for the opcode and two for the offset value.

PC + 1 → PC
EA = (PC): (PC + 1) + X
PC + 2 → PC

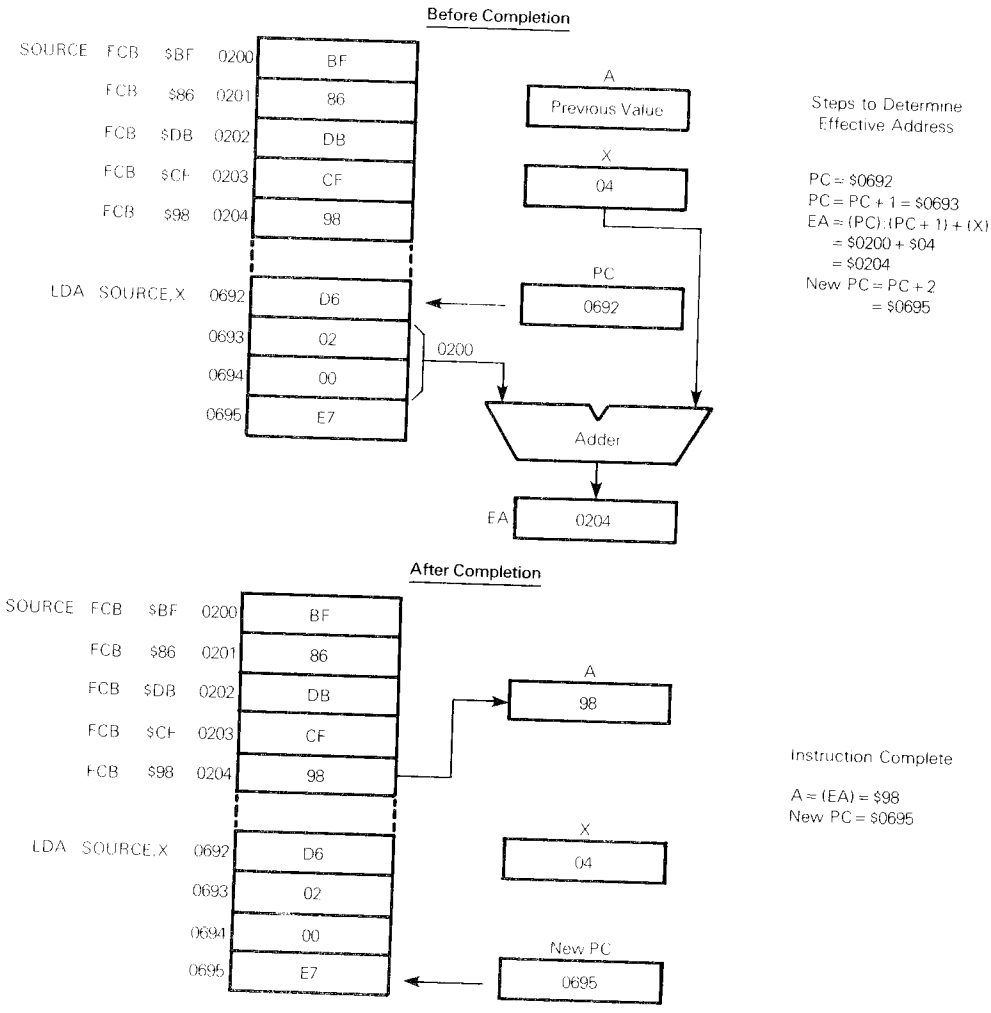


Figure 2-9. Indexed Addressing Mode, 16-Bit Offset Example

In the following example, a block of data is moved from a source table to a destination table. The index register contains the block length. Figure 2-9 illustrates the steps required to determine the EA from which to store the memory address contents into the accumulator.

```
                        SOURCE  EQU  $200
                        DESTIN  EQU  $40
0690  AE   04                    LDX  #$04
0692  D6   0200       BLKMOV     LDA  SOURCE,X   Load the Accumulator with
                                                 Contents of Location
                                                 SOURCE + X
0695  E7   40                    STA  DESTIN,X   Store the Contents of the
                                                 Accumulator in Location
                                                 DESTIN + X
0698  5A                        DECX            Next Location
0699  2A   0692                  BPL  BLKMOV     Repeat if More
```

2.3.5.4 INDEXING COMPATIBILITY Since the index register in the M6805 HMOS/M146805 CMOS Family is only eight bits long, and the offset values are zero, eight, or 16 bits, the MC6800 user may thus find that the X register on the M6805 HMOS/M146805 CMOS Family is best utilized "backwards" from the MC6800. That is, the offset will contain the address of the table and the index register contains the displacement into the table.

2.3.6 Relative Addressing Modes

Relative addressing is used only for branch instructions and specifies a location relative to the current value of PC. The EA is formed by adding the contents of the byte following the opcode to the value of the PC. Since the PC will always point to the next statement in line while the addition is being performed, a zero relative offset byte results in no branch. The resultant EA is used if, and only if, a relative branch is taken. Note that by the time the byte following the opcode is added to the contents of the PC, it is already pointing to the next instruction while the addition is being performed. Branch instructions always contain two bytes of machine code: one for the opcode and one for the relative offset byte. Because it is desirable to branch in either direction, the offset byte is sign extended with a range of -128 to $+127$ bytes. The effective range however, must be computed with respect to the address of the next instruction in line. Relative branch instructions consist of two bytes; therefore, the effective range of a branch instruction from the beginning of the branch instruction is defined as (where R is defined as the address of the branch instruction):

$$(PC+2)-128 \leq R \leq (PC+2) + 127$$

or

$$PC-126 \leq R \leq PC + 129 \text{ (for conditional branch only)}$$

A jump (JMP) or jump-to-subroutine (JSR) should be used if the branch range is exceeded.

$PC + 1 \rightarrow PC$
$(PC) \rightarrow TEMP$
$PC + 1 \rightarrow PC$
$EA = PC + TEMP$ iff branch is taken

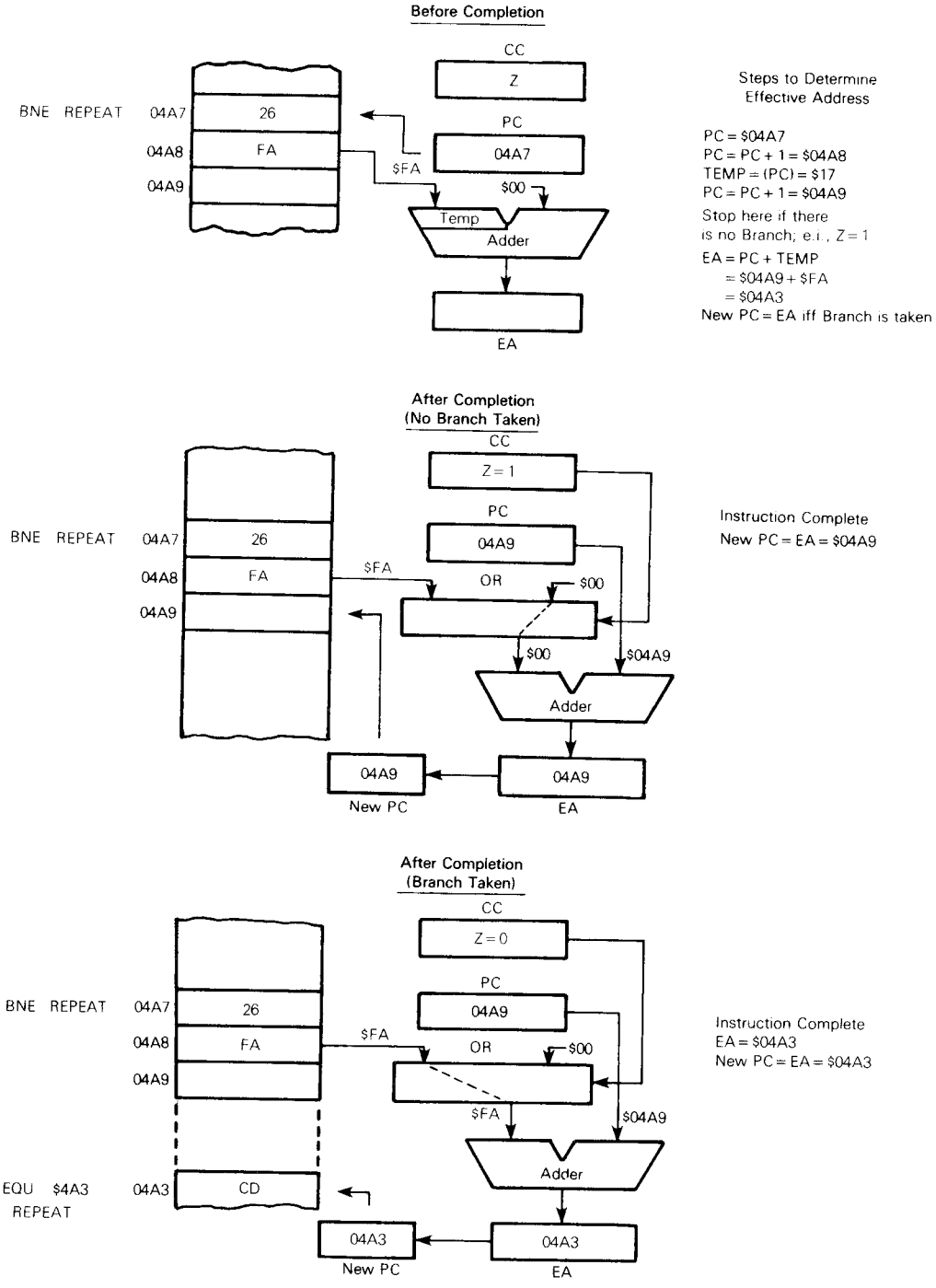


Figure 2-10. Relative Addressing Mode Example

In the following example, the routine uses the index register as a counter for executing the subroutine WORK 50 times. The conditional branch, BNE, tests the Z bit which is set if the result of the DECX instruction clears the index register. The line of code shown in Figure 2-10, contains an instruction to branch to REPEAT, if the condition code register Z bit has not been set by the previous program step (DECX). Note in Figure 2-10 that the Z bit controls which number is added to the PC contents. If the branch is taken, the relative offset byte ($FA) is added; however, if the branch is not taken, nothing is added which leaves the EA at PC + 2. Note in this case the relative offset byte $FA indicates a backward branch since the most significant bit is a 1.

Assembly Examples:

```
04A1  AE   50          LDX    #50
04A3  CD   04C0  REPEAT JSR    WORK
04A6  5A                DECX
04A7  26   FA    04A3   BNE    REPEAT (See Example Description)
```

2.3.7 Bit Manipulation

Bit manipulation consists of two different addressing modes: bit set/clear and bit test and branch. The bit set/clear mode allows individual memory and I/O bits to be set or cleared under program control. The bit test and branch mode allows any bit in memory to be tested and a branch to be executed as a result. Each of these addressing modes is described below.

2.3.7.1 BIT SET/CLEAR ADDRESSING MODE. Direct byte addressing and bit addressing are combined in instructions which set and clear individual memory and I/O bits. In the bit set and bit clear instructions, the memory address location (containing the bit to be modified) is specified as direct address in the location following the opcode. As in direct addressing, the first 256 memory locations can be addressed. The actual bit to be modified, within the byte, is specified within the low nibble of the opcode. The bit set and clear instructions are two-byte instructions: one for the opcode (including the bit number) and the other to address the byte which contains the bit of interest.

CAUTION

On some M6805 Family HMOS devices, the data direction registers are write-only registers and will read as $FF. Therefore, the bit set/clear instructions (or read/modify/write instructions) shall not be used to manipulate the data direction register.

PC + 1 → PC
EA = (PC) + $0000
PC + 1 → PC

The following example compares the true bit manipulation of the M6805 HMOS/ M146805 CMOS Family to the conventional method of bit manipulation. This example uses the bit manipulation instruction to turn off a LED using bit 2 of port B and three conventional instructions to turn the LED on. The example polls the timer control register interrupt request bit (TCR, bit 7) to determine when the LED should turn on.

Assembly Example:

```
                0001        PORTB   EQU     $01         Define Port B
                                                        Address
                0009        TIMER   EQU     $09         Define TCR
                                                        Address

                        BIT MANIPULATION INSTRUCTIONS

058F  15  01                       BCLR    2,PORTB     Turn Off LED
0591  0F  09  FC      REPT          BRCLR   7,TIMER,REPT Check Timer
                                                        Status Repeat if
                                                        Not Timed Out
0594  14  01                       BSET    2,PORTB     Turn on LED if
                                                        Timer Times Out

                        CONVENTIONAL INSTRUCTIONS

                      AGAIN LDA     TIMER       Get Timer Status
                            BIT     #$80        Mask Out Proper
                                                Bit
                            BNE     AGAIN       Test-Turn On if
                                                Timer Times Out
                            LDA     PORTB       Get Port B Data
                            AND     #$FB        Clear Proper Bit
                            STA     PORTB       Save Modified
                                                Data to Turn Off
                                                LED
                            BRA     REPT
```

Figure 2-11 shows an example of the bit set/clear addressing mode. In this example, the assembly example above contains an instruction to clear bit 2 PORTB. (PORTB in this case is equal to the contents of memory location $001, which is the result of adding the byte following the opcode to $0000.)

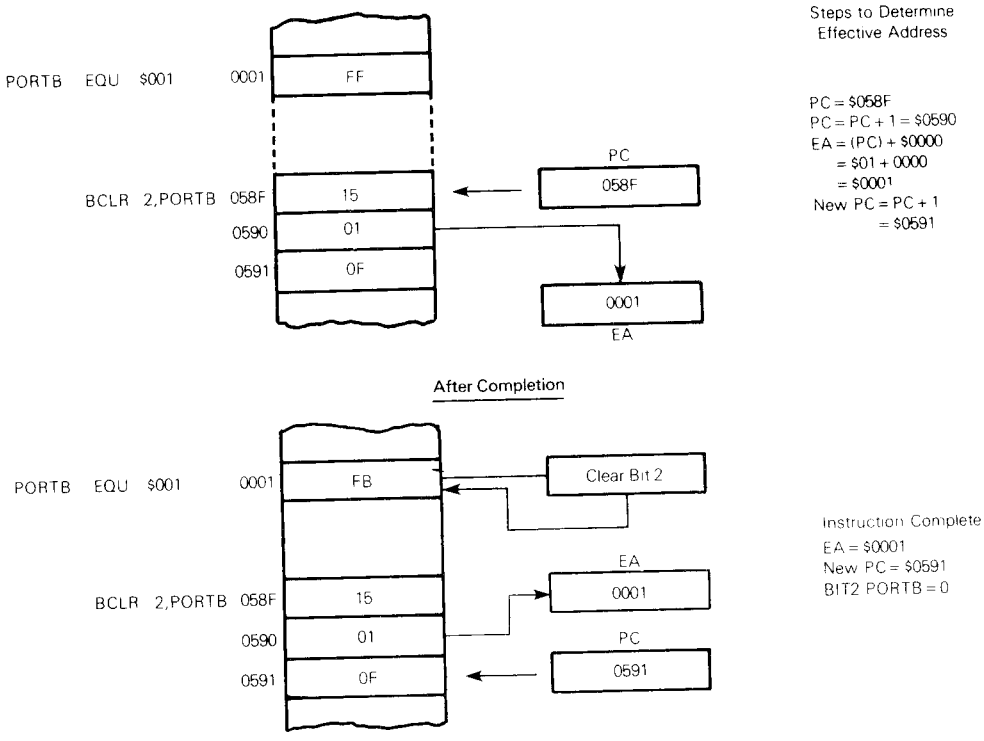


Figure 2-11. Bit Set/Clear Addressing Mode Example

2.3.7.2 BIT TEST AND BRANCH ADDRESSING MODE. This mode is a combination of direct, relative, and bit set/clear addressing. The data byte to be tested is located via a direct address in the location following the opcode. The actual bit to be tested, within the byte, is specified within the low order nibble of the opcode. The relative address for branching is in the byte following the direct address (second byte following the opcode). Thus, the bit test and branch instructions are three-byte instructions (opcode byte, direct byte, and relative byte). A bit test and branch has a relative addressing range of $PC - 125 \leq R \leq PC + 130$ from the beginning of the instruction.

The bit manipulation routine shown in the previous paragraph uses a bit test and branch instruction to poll the timer; i.e., REPT BRCL 7,TIMER, REPT. This instruction causes timer bit 7 to be tested until it is cleared, at which time it falls through to turn on a LED. Figure 2-12 illustrates this loop by showing both the branch and no branch status. Note that if timer bit 7 is clear (timer not timed out), a backward branch is taken as long as the C bit is cleared ($FD is added to $0594 and its sign bit is negative). When the timer times out, timer bit 7 is set (C bit is also set) and the program falls through to $0594. Notice in the same routine example, that conventional bit test and branch instructions require three separate instructions to perform the same function.

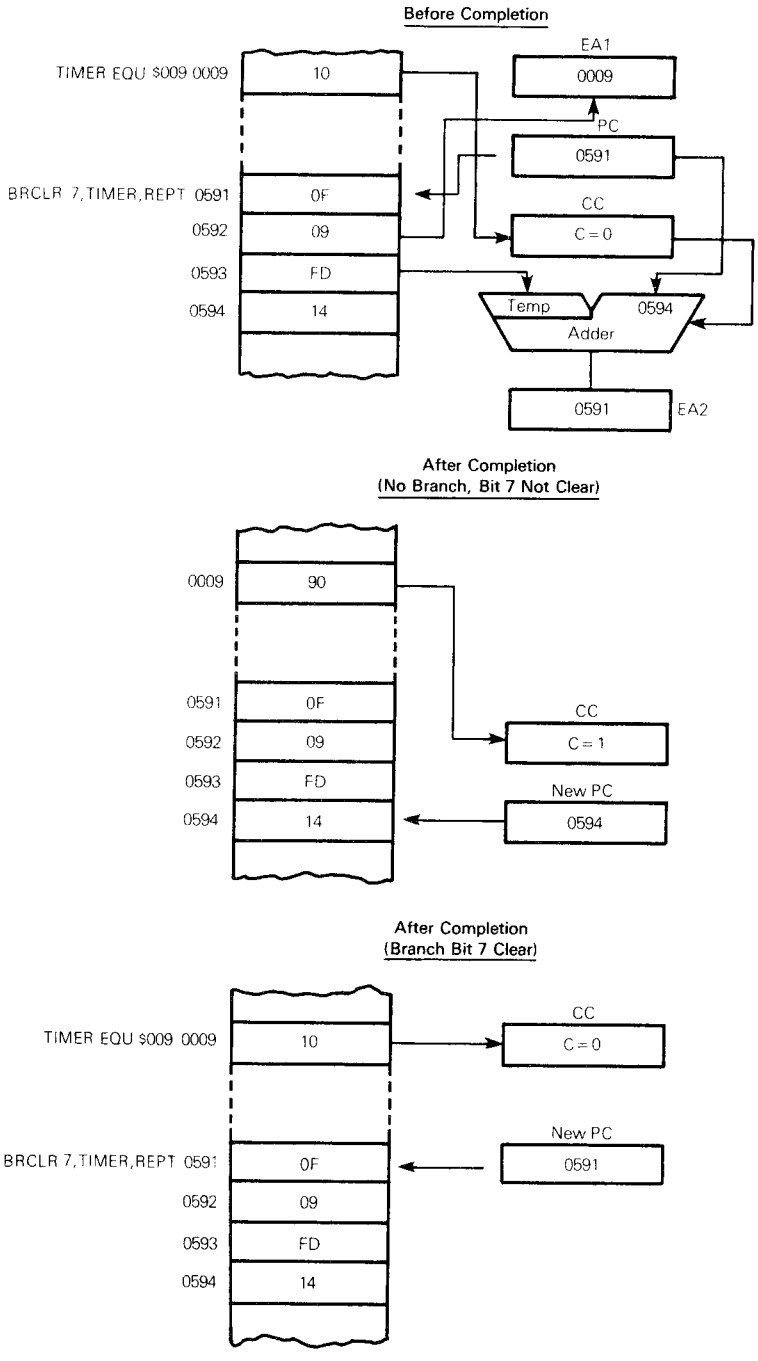


Figure 2-12. Bit Test and Branch Addressing Mode Example

2.4 INSTRUCTION SET OVERVIEW

2.4.1 Introduction

It is convenient to view the M6805 HMOS/M146805 CMOS Family as having five different instruction types rather than one set of instructions. These include: register/memory, read/modify/write, branch, control, and bit manipulation. Appendix C contains a detailed definition of the instruction set used with the M6805 HMOS/M146805 CMOS Family; Appendix D contains an alphabetical listing of the instruction set; Appendix E provides a tabular functional listing of the instruction set; Appendix F contains a numerical listing which shows the mnemonic, addressing mode, cycles, and byte of the instruction set; Appendix G provides a cycle-by-cycle summary of the instruction set; and Appendix I contains an instruction set opcode map.

2.4.2 Register/Memory Instructions

Most of these instructions contain two operands. One operand is inherently defined as either the accumulator or the index register; whereas, the other operand is fetched from memory via one of the addressing modes. The addressing modes which are applicable to the register/memory instructions are given below.

Immediate
Direct
Extended
Indexed — No Offset
Indexed — 8-Bit (One Byte) Offset
Indexed — 16-Bit (Two Byte) Offset

Immediate addressing is not usable with store and jump instructions (STA, STX, JMP, and JSR). An alphabetical listing of the register/memory instruction is provided below.

ADC Add Memory and Carry to Accumulator
ADD Add Memory to Accumulator
AND AND Memory with Accumulator
BIT Bit Test Memory with Accumulator (Logical Compare)
CMP Compare Accumulator with Memory (Arithmetic Compare)
CPX Compare Index Register with Memory (Arithmetic Compare)
EOR Exclusive OR Memory with Accumulator
JMP Jump
JSR Jump to Subroutine
LDA Load Accumulator from Memory
LDX Load Index Register from Memory
ORA OR Memory with Accumulator
SBC Subtract Memory and Borrow from Accumulator
STA Store Accumulator in Memory
STX Store Index Register in Memory
SUB Subtract Memory for Accumulator

2.4.3 Read/Modify/Write Instructions

These instructions read a memory location or register, modify or test the contents, and then write the modified value back into the memory or the register. The available addressing modes for these instructions are given below. Note that all read/modify/write instruction memory accesses are limited to the first 511 locations.

Direct
Inherent
Indexed — No Offset
Indexed — 1 Byte Offset

The read/modify/write instructions are listed below.

ASL	Arithmetic Shift Left (Same as LSL)
ASR	Arithmetic Shift Right
CLR	Clear
COM	Complement
DEC	Decrement
INC	Increment
LSL	Logical Shift Left (Same as ASL)
LSR	Logical Shift Right
NEG	Negate (Twos Complement)
ROL	Rotate Left thru Carry
ROR	Rotate Right thru Carry
TST	Test for Negative or Zero

2.4.4 Control Instructions

Instructions in this group have inherent addressing, thus, only contain one byte. These instructions manipulate condition code bits, control stack and interrupt operations, transfer data between the accumulator and index register, and do nothing (NOP). The control instructions are listed below.

CLC	Clear Carry Bit
CLI	Clear Interrupt Mask Bit
NOP	No Operation
RSP	Reset Stack Pointer
RTI	Return from Interrupt
RTS	Return from Subroutine
SEC	Set Carry Bit
SEI	Set Interrupt Mask Bit
SWI	Software Interrupt
TAX	Transfer Accumulator to Index Register
TXA	Transfer Index Register to Accumulator

2.4.5 Bit Manipulation Instructions

There are two basic types of bit manipulation instructions. One group either sets or clears any single bit in a memory byte. This instruction group uses the bit set/clear addressing mode which is similar to direct addressing. The bit number (0-7) is part of the opcode. The other group tests the state of any single bit in a memory location and branches if the bit is set or clear. These instructions have "test and branch" addressing. The bit manipulation instructions are shown below (the term iff is an abbreviation for "if-and-only-if").

BCLR n	Clear Bit n in Memory
BRCLR n	Branch iff Bit n in Memory is Clear
BRSET n	Branch iff Bit n in Memory is Set
BSET n	Set Bit n in Memory (n = 0. . .7)

2.4.6 Branch Instruction

In this set of instructions the program branches to a different routine when a particular condition is met. When the specified condition is not met, execution continues with the next instruction. Most of the branch instructions test the state of one or more of the condition code bits. Relative is the only legal addressing mode applicable to the branch instructions. A list of the branch instructions is provided below (the term iff is an abbreviation for "if-and-only-iff").

BCC Branch iff Carry is Clear (Same as BHS)
BCS Branch iff Carry is Set (Same as BLO)
BEQ Branch iff Equal to Zero
BHCC Branch iff Half Carry is Clear
BHCS Branch iff Half Carry is Set
BHI Branch iff Higher than Zero
BHS Branch iff Higher or Same as Zero (Same as BCC)
BIH Branch iff Interrupt Line is High
BIL Branch iff Interrupt Line is Low
BLO Branch iff Lower than Zero (Same as BCS)
BLS Branch iff Lower or Same as Zero
BMC Branch iff Interrupt Mask is Clear
BMI Branch iff Minus
BMS Branch iff Interrupt Mask is Set
BNE Branch iff Not Equal to Zero
BPL Branch iff Plus
BRA Branch Always
BRN Branch Never
BSR Branch to Subroutine

Note that the BIH and BIL instructions permit an external interrupt pin (INT or IRQ) to be easily tested.