Providing users with concise, up-to-date information on Motorola's M6805 HMOS/M146805 CMOS family, this manual shows how various family members can be used for design of control and instrumentation as well as other diverse applications. Basic design differences between 15 of the family devices are listed in this easy-to-use manual:

MC6805P2	MC68705P3
MC6805P4	MC68705R3
MC6805P6	MC68705U3
MC6805R2	MC1468705G2
MC6805R3	MC146805E2
MC6805T2	MC146805F2
MC6805U2	MC146805G2
MC6805U3	

This versatile family of microcomputers and a microprocessor offers many versions for diverse applications with the latest features including EPROM for easy prototype system design, low power versions, low cost, and powerful architecture.

Detailed information on software (descriptions and applications) and hardware (features and applications) is provided in the main chapters. Also included are chapters on the EPROM programmer and on the self-test, the on-chip firmware test capability. Appendices are included to provide designers with the latest M6805 HMOS/M146805 CMOS family programming information.

MOTOROLA

M6805 HMOS
M146805 CMOS
FAMILY
MICROCOMPUTER/MICROPROCESSOR
USER'S MANUAL

SECOND EDITION

CHAPTER 1
GENERAL DESCRIPTION

1.1 INTRODUCTION TO THE M6805 HMOS/M146805 CMOS FAMILY

The continuing technological evolution in microprocessors and microcomputers has resulted in larger, more complex, and more powerful devices which contain characteristics of both mini and mainframe computers. The technological evolution of the MC6800 to the M6809 Family and the 16-bit MC68000 is a clear example of devices which evolved upward from the mini and mainframe computer architecture. The experience gained during this upward evaluation has greatly enhanced the expertise needed to design more powerful low- and mid-range devices. By using the architectural characteristics of the mini and mainframe computers, the microprocessor/microcomputer hardware and software becomes regular and versatile, yet simple.

The demanding requirements of the mid-range control-oriented microprocessor market (low cost) can be met with the M6805 HMOS/M146805 CMOS Families of microcomputer (MCU) and microprocessor (MPU). The M6805 HMOS/M146805 CMOS Families are the first to provide the software and hardware capabilities of more advanced computers to the controller market. Previously, designers and manufacturers were required to choose between "no processor at all" or a processor that functioned more like a calculator than a computer.

Control-oriented microprocessors have evolved from two different bases: calculator based and computer based. The calculator-based design was at first considered as a natural building block for controllers since, most often, a controller was required to be a complete self-contained unit. However, calculator based control-oriented microprocessors use a split memory architecture containing separate data paths between the CPU and peripherals (memory or I/O or registers). In addition, calculator-based I/O, display, and keypad were separated from program and data storage memory. Because of this, separate address maps were required which forced the inclusion of many special purpose instructions and resulted in an irregular architecture. As a result, these calculator based devices required that hardware and software designers remember and consider many special cases in order to perform any task. Thus, the software and hardware became very random, irregular, and difficult to update.

The computer-based design led to another group of processors, like the MC6800, which contained many features of large computers. These devices contain a single data bus which allows access to a single address map, eliminating the need for split memory architecture. In this one-address map design, all I/O, program, and data may be accessed with the same instruction; therefore, there are fewer instructions to remember. The

actual number of unique instructions is increased by a variety of addressing modes which define how an instruction accesses any data required for the operation. For example, depending upon which addressing mode is used, the accumulator may be loaded (LDA instruction) with data in six different ways. This effectively provides the programmer with more tools to work with but fewer things to remember. Thus, because of regularity of the architecture, the hardware is regular and can be implemented more efficiently.

All members of the M6805 HMOS/M146805 CMOS Family of MCUs and MPUs are designed around a common core which consists of CPU, timer, oscillator, ROM (EPROM, masked, or none), control section (for interrupts and reset), and varying amounts of bidirectional I/O lines. In addition to this common core, additional items can be added such as: additional memory, A/D converter, phase-lock-loop, and additional I/O lines. As of the printing of this manual in late 1982, this versatile common core design has already provided 11 different M6805 HMOS Family devices and four different M146805 CMOS Family devices. These 15 different family members allow the user to choose the device best suited for his particular application. The increased number of devices could preclude paying for a supplied feature that is not needed or paying extra to externally add a needed feature that is not included.

Information describing I/O options and general operation of the M6805 HMOS/M146805 CMOS Family members is included in this chapter. Detailed information concerning device operation is included in the following chapters as well as appendices. Chapters discussing hardware and software applications are also included to illustrate some of the family features and provide a useful tool for the user.

The M6805 HMOS/M146805 CMOS Family architecture and instruction set are very similar to that of Motorola's MC6800. Any programmer who has worked with the MC6800 can attain equivalent proficiency with the M6805 HMOS/M146805 CMOS Family in a relatively short time. In some respects the M6805 HMOS/M146805 CMOS Family is more powerful than the MC6800 (depending upon the application) as a result of architecture optimization. Appendix A summarizes the architectural and instruction set differences between the M6805 HMOS/M146805 CMOS and M6800 Families.

1.2 OPTIMIZED FOR CONTROLLER APPLICATIONS

The M6805 HMOS/M146805 CMOS Family architecture has been optimized for controller applications, rather than general purpose data processing operations. Several features contribute to this optimization.

1.2.1 Instruction Set

The instruction set, used with the M6805 HMOS/M146805 CMOS Family, is specifically designed for byte-efficient program storage. Byte efficiency permits a maximum amount of program function to be implemented within a finite amount of on-chip ROM. Improved ROM efficiency allows the M6805 HMOS/M146805 CMOS Family to be used in applications where other processors might not perform the task in the available ROM space.

More features may be included in applications where ROM space is more than adequate. In some cases the user might wish to include programs for more than one application. In such cases the appropriate program could be selected by the power-up initialization program. The ability to nest subroutines, the addition of true bit test and bit manipulation instructions, the multi-function instructions, and the versatile addressing modes all contribute to byte efficiency.

Superficial comparisons of the number of bytes per instruction for the M6805 HMOS/M146805 CMOS Family, when compared to other machines in this class, can be very misleading. A single M6805 HMOS/M146805 Family instruction occupying 2 or 3 bytes accomplishes as much real programming work as several single byte instructions, or a subroutine, would accomplish in many other processors.

The bit test and bit manipulation instructions permit the program to:

branch on bit set
branch on bit clear
set bit
clear bit.

These instructions operate on any individual bit in the first 256 address spaces (page zero). As such, the bit manipulations access I/O pins, RAM bits, and ROM bits.

In the M6805 HMOS/M146805 CMOS Family, a page consists of 256 consecutive memory locations. Page zero includes the lowest-numbered 256 memory addresses ($00 through $FF), page one the next 256 memory addresses ($100 through $1FF), etc. The first 128 bytes of page zero memory locations ($00 through $7F) are used primarily for I/O function registers, timer, PLL, RAM, and the stack. The next 128 bytes of page zero ($80 through $FF) contain ROM which is available for the user program. An efficient use of pages zero and one would be for storage of tables since these two pages are easily accessed by the indexed addressing mode.

1.2.2 Addressing Modes

One of the chief measures of the effectiveness of a computer architecture is its ability to access data. The M6805 HMOS/M146805 CMOS Family has several major memory addressing modes. They include immediate, direct, and extended, plus three distinct indexed modes. The programmer is thus given the opportunity to optimize the code to the task. The indexed addressing modes permit conversion tables, jump tables, and data tables to be located anywhere in the address space. The use of tables is an important tool in controller type applications.

Efficient addressing methods are coupled with instructions which manipulate memory without disturbing the program registers. Thus, RAM may be used for the same functions that other processors use general purpose registers (increment, decrement, clear, complement, test, etc.). The M6805 HMOS/M146805 CMOS Family members have a very versatile, efficient, and easy-to-use I/O structure. All microcomputer I/O function registers are memory mapped into the first 16 processor addresses. Advantage is thus taken of the efficient addressing modes, the many memory reference instructions, and the use of

RAM (or I/O registers) as general purpose registers. As an example, there are 64 unique instructions which permit the programmer to modify an I/O port. The programmer's problem is not so much how to accomplish a given I/O task, but rather to choose the most effective method from the many methods available. In addition, as with other M6800 Family I/O devices, most M6805 HMOS/M146805 CMOS Family I/O pins are individually programmed as inputs or outputs under software control.

1.3 CHOICE OF TECHNOLOGIES

The first option to be selected by the system designer is the choice between HMOS or CMOS as a processor technology.

1.3.1 HMOS Features

The NMOS (N-Channel Metal Oxide on Silicon) technology has been the mainstay of the M6800 Family. The current state of the continual shrinking of NMOS is called HMOS (High-Density NMOS).

The prime consideration in choosing an M6805 HMOS Family microcomputer is its lower price. Motorola's highly-efficient fabrication process results in a greater yield than other processes. The decreased production costs ultimately result in lower selling prices. The economics of large scale production also contribute to a low selling price.

The high speed of Motorola's HMOS, when compared to PMOS or other NMOS processors, produces a very high performance/price ratio.

A low voltage inhibit (LVI) feature may be selected on HMOS versions. The LVI option forces a reset when the supply voltage drops below a threshold which guarantees correct operation. The CMOS Family members offer wide operating voltage and clock speed ranges, which preclude establishing an LVI threshold.

1.3.2 CMOS Features

An emerging microcomputer technology is CMOS (Complementary MOS, both P- and N-Channel devices). The unique properties of CMOS are increasingly attractive. Some applications are simply not feasible with PMOS, NMOS, or HMOS microcomputers.

Maximum power consumption of CMOS parts ranges from 1/15 to 1/200 of that of an equivalent HMOS part. Low power consumption is important in several classes of applications; thus, CMOS microcomputers are desirable.

(a) Portable Equipment — Hand-held and other portable units operated from self-contained batteries. Battery drain is frequently important in such applications.

(b) Battery Back-Up — CMOS is appropriate in ac powered applications when some or all system functions must continue during a power outage. A small, rechargeable battery keeps a CMOS MCU operable.

(c) Storage Batteries — Automotive and telephone equipment operate from larger batteries. Automobile battery drain must be low when the engine is not running. Telephones must operate independently of ac power.

(d) Heat Dissipation — Packaging constraints sometimes preclude dissipating electronics-generated heat, or the heat is costly to dissipate. In addition, dissipation of heat directly effects device reliability.

(e) Power Costs — The cost of electricity to power the equipment becomes a significant factor in calculating the total life cycle cost of equipment which operates continuously.

The CMOS technology inherently operates over a wide range of supply voltages. Thus, CMOS is used where the supply voltage fluctuates, such as in battery powered equipment; or if line power is available, a lower-cost, loosely regulated supply may be used.

An additional advantage of CMOS is that circuitry is fully static. CMOS microcomputers may be operated at any clock rate less than the guaranteed maximum. This feature may be used to conserve power, since power consumption increases with higher clock frequencies. Static operation may also be advantageous during product developments.

1.4 HARDWARE

Every M6805 HMOS/M146805 CMOS Family microcomputer or microprocessor contains hardware common to all versions, plus a combination of options unique to a particular version. There are also several differences among family members of which potential users should be aware.

1.4.1 Hardware Common To All Devices

Figure 1-1 details the hardware functional blocks common to all M6805 HMOS/M146805 CMOS Family devices.

The central processor unit (CPU) contains the 8-bit arithmetic logic unit, accumulator, program counter, index register, stack pointer, condition code register, instruction decoder, and timing and control logic. These elements resemble the M6800 Family of microprocessors which reflect the M6805 HMOS/M146805 CMOS Family heritage.

The M6805 HMOS/M146805 CMOS Family has on-chip RAM, permitting the microcomputer versions to operate without external memory. The addressing modes and register-like memory operations use this RAM to the fullest extent possible.

Parallel I/O capability, with pins programmable as input or output, is built into every unit.

The external interrupt input, and the capability for multiple nesting of subroutine and interrupts, are features usually found on much more powerful architectures. They permit an M6805 HMOS/M146805 CMOS Family MCU to be used in projects usually considered too complex for microcomputers.

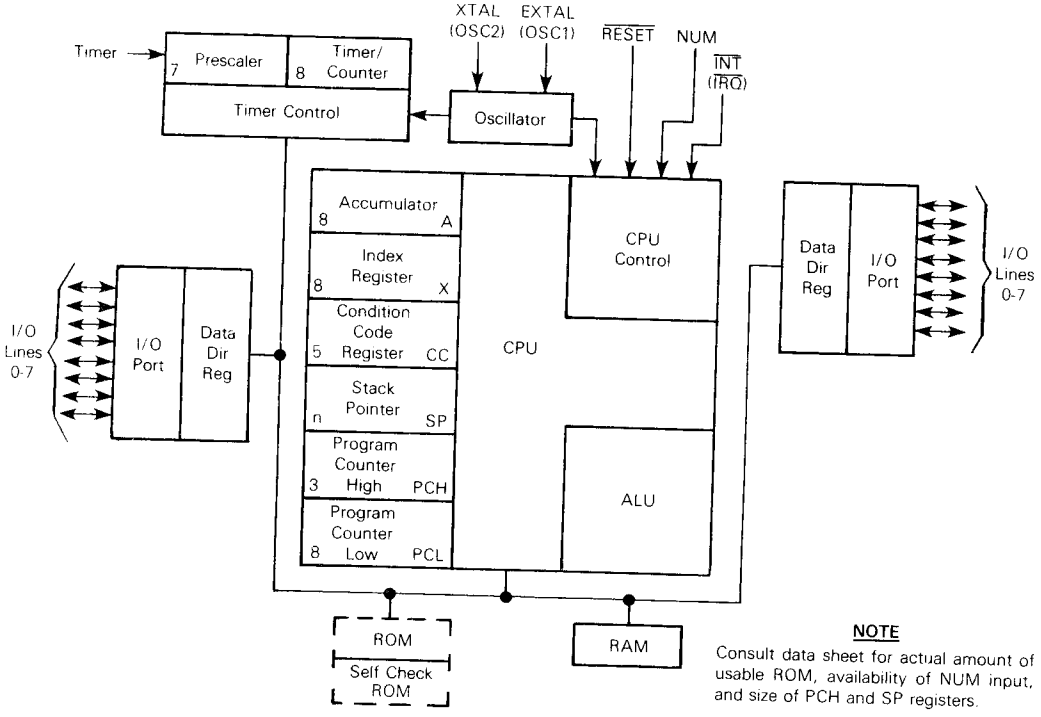


Figure 1-1. M6805 HMOS/M146805 CMOS Family Basic Microcomputer Block Diagram

A feature which greatly simplifies software development and extends the capability of a microcomputer is an on-chip timer/counter. This 8-bit counter and its prescaler can be programmed for innumerable functions. It can generate an interrupt at software selected intervals. It can also be used as an event counter to generate an interrupt after some software selected number of external events. The timer/counter can also be used for timekeeping, measuring and generating pulses, and counting external events. In the case of the M146805 CMOS Family devices, the timer can be set to "wake-up" the processor from the power-saving WAIT mode.

The external interrupt and timer/counter interrupt are vectored to different service routine addresses. This greatly simplifies interrupt programming. It also speeds execution of interrupt routines, by eliminating software interrupt polling, for determining the source of the interrupt.

The first 16 processor addresses are reserved for memory mapped I/O registers. The programmer of the M6805 HMOS/M146805 CMOS Family may take full advantage of the versatile addressing modes and the register-like RAM operations of the Family.

1.4.2 Family Options

In addition to the common hardware described previously, users can make selections from among devices having a combination of hardware options. Potential users should consult their local Motorola sales representative or the most recent data brochures to determine which versions have reached production. Table 1-1 provides a listing of the features included in each family member. Figure 1-2 illustrates the part number nomenclature of three different members of the M6805 HMOS/M146805 CMOS Family.

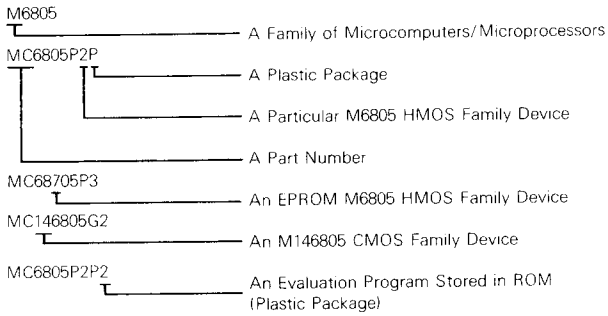


Figure 1-2. M6805 HMOS/M146805 CMOS Family Nomenclature Example

The first option to be selected by the system designer is the choice of technology. In general, the HMOS units would be selected unless the application specifically requires one of the unique characteristics of CMOS.

User ROM sizes range from none, for the microprocessor, to 4K and larger. Future versions will have additional ROM sizes. When self-check ROM is a part of the device, the ROM area used in the self-check operation is not included in the published ROM sizes. The entire ROM space is available to the user for his program.

A portion of the ROM is located in page zero (the direct page) to facilitate more efficient access to look up tables using all available addressing modes. This ROM can, of course, be used for program storage as well as look-up tables.

The initial M6805 HMOS/M146805 CMOS Family devices contain either 64 or 112 bytes of on-chip RAM which is located in page zero. Future devices may accommodate additional or differing amounts of RAM.

Package size options permit as many as four, full 8-bit bidirectional I/O ports. Each pin is defined under software control as an input or output by loading a data direction register.

Table 1-1. M6805 HMOS/M146805 CMOS Family List of Features

M6805 HMOS Family MCUs

Features	MC6805P2	MC6805P4	MC6805P6	MC6805R2	MC6805R3	MC6805T2	MC6805U2	MC6805U3
Technology	HMOS	HMOS	HMOS	HMOS	HMOS	HMOS	HMOS	HMOS
Number of Pins	28	28	28	40	40	28	40	40
On-Chip RAM (Bytes)	64	112*	64	64	112	64	64	112
On-Chip User ROM (Bytes)	1.1K	1.1K	1.8K	2K	3.8K	2.5K	2K	3.8K
External Bus	None	None	None	None	None	None	None	None
Bidirectional I/O Lines	20	20	20	24	24	19	24	24
Unidirectional I/O Lines	None	None	None	6 Inputs	6 Inputs	None	8 Inputs	8 Inputs
Other I/O Features	Timer	Timer	Timer	Timer, A/D	Timer, A/D	Timer, PLL	Timer	Timer
External Interrupt Inputs	1	1	1	2	2	2	2	2
EPROM Version	MC68705P3	MC68705P3	MC68705P3	MC68705R3	MC68705R3	None	MC68705U3	MC68705U3
STOP and WAIT	No	No	No	No	No	No	No	No

*Indicates standby RAM

M6805 HMOS/M146805 CMOS Family EPROM MCUs

Features	MC68705P3	MC68705R3	MC68705U3	MC1468705G2
Technology	HMOS	HMOS	HMOS	CMOS
Number of Pins	28	40	40	40
On-Chip RAM (Bytes)	112	112	112	112
On-Chip User ROM (Bytes)	1.8K EPROM	3.8K EPROM	3.8K EPROM	2K EPROM
External Bus	None	None	None	None
Bidirectional I/O Lines	20	24	24	32
Unidirectional I/O Lines	None	6 Inputs	8 Inputs	None
Other I/O Features	Timer	Timer, A/D	Timer	Timer
External Interrupt Inputs	1	2	2	1
EPROM Version	–	–	–	–
STOP and WAIT	No	No	No	Yes

M146805 CMOS Family MPU/MCUs

Features	MC146805E2	MC146805F2	MC146805G2
Technology	CMOS	CMOS	CMOS
Number of Pins	40	28	40
On-Chip RAM (Bytes)	112	64	112
On-Chip User ROM (Bytes)	None	1K	2K
External Bus	Yes	None	None
Bidirectional I/O Lines	16	16	32
Unidirectional I/O Lines	None	4 Inputs	None
Other I/O Features	Timer	Timer	Timer
External Interrupt Inputs	1	1	1
EPROM Version	None	None	MC1468705G2
STOP and WAIT	Yes	Yes	Yes

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.

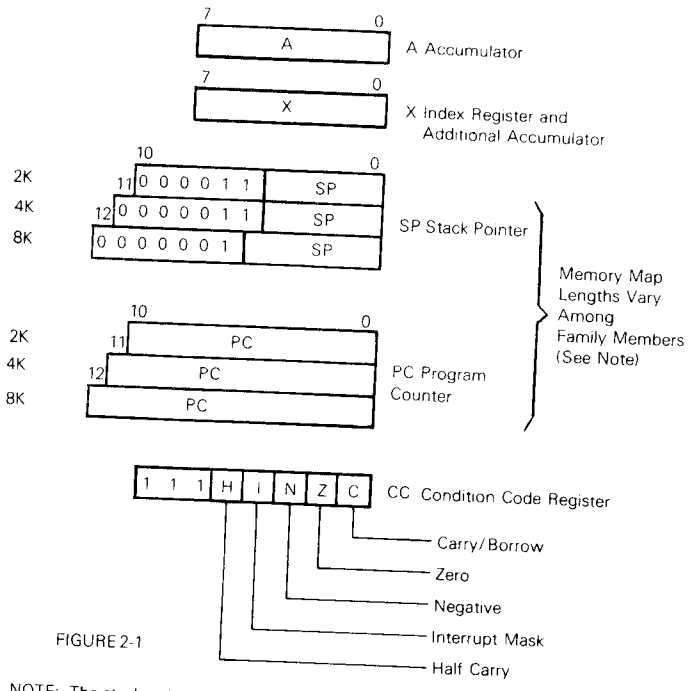


FIGURE 2-1

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).

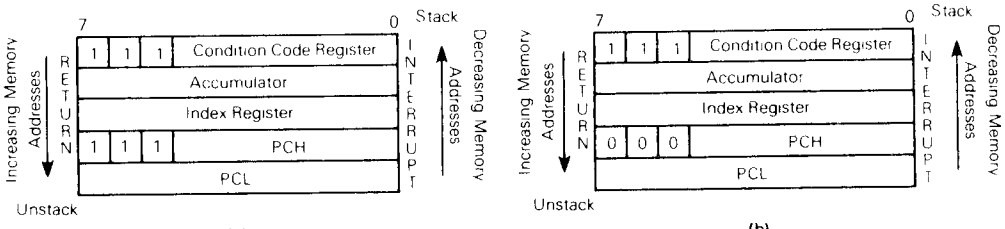
Nested subroutine calls and interrupts must not underflow the SP. The usable stack length will vary between devices as well as between the M6805 HMOS and M146805 CMOS Families. In the M6805 HMOS Family, the usable stack length is $2^n - 1$ (where n = the number of bits in the stack pointer); however,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

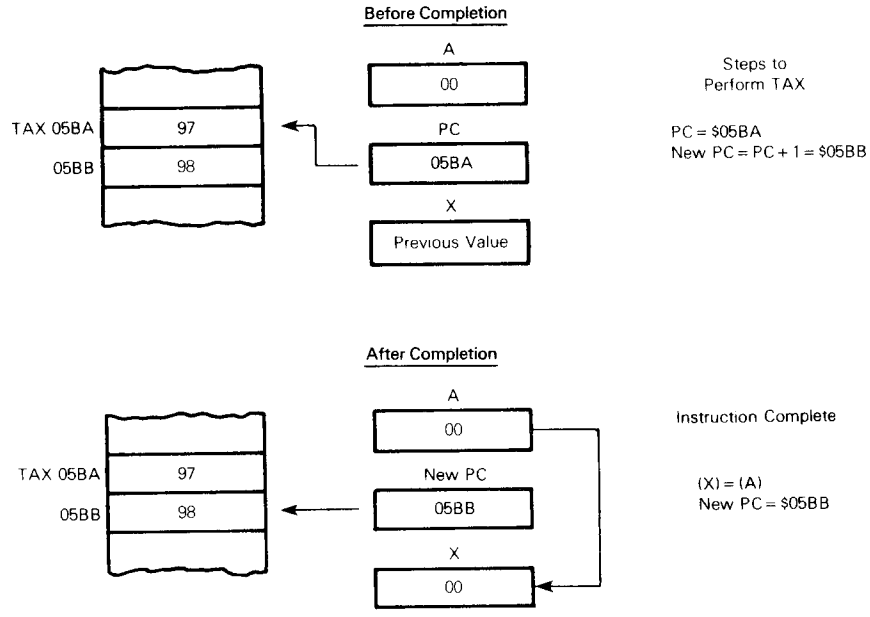


Figure 2-3. Inherent Addressing Mode Example

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
```

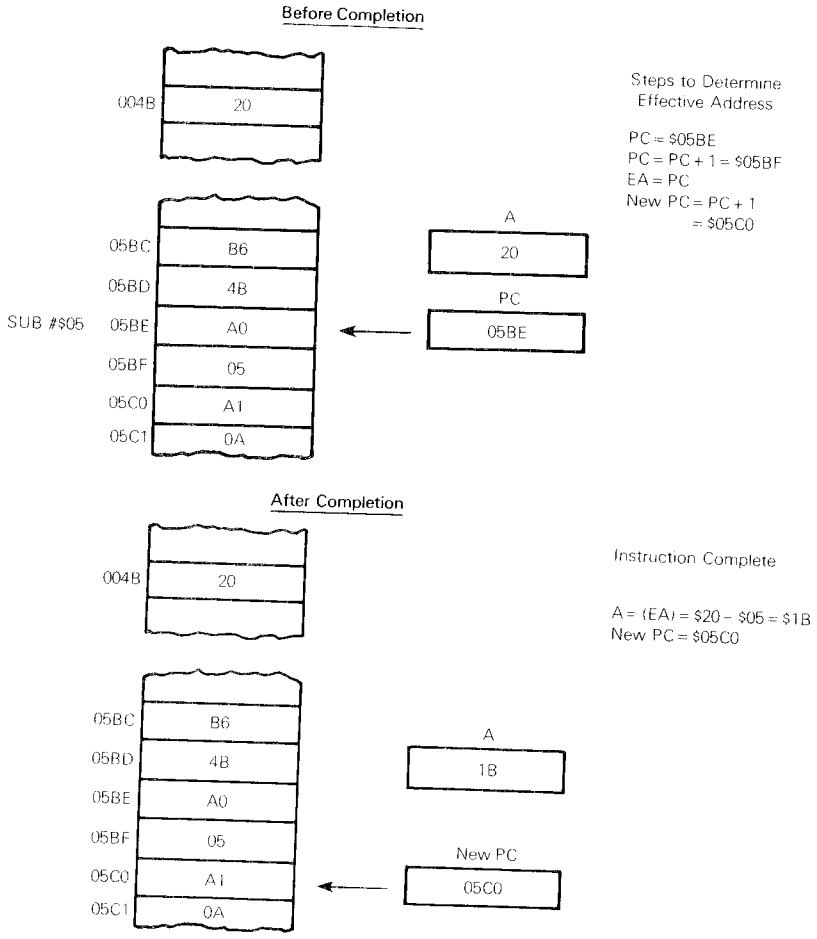


Figure 2-4. Immediate Addressing Mode Example

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
```

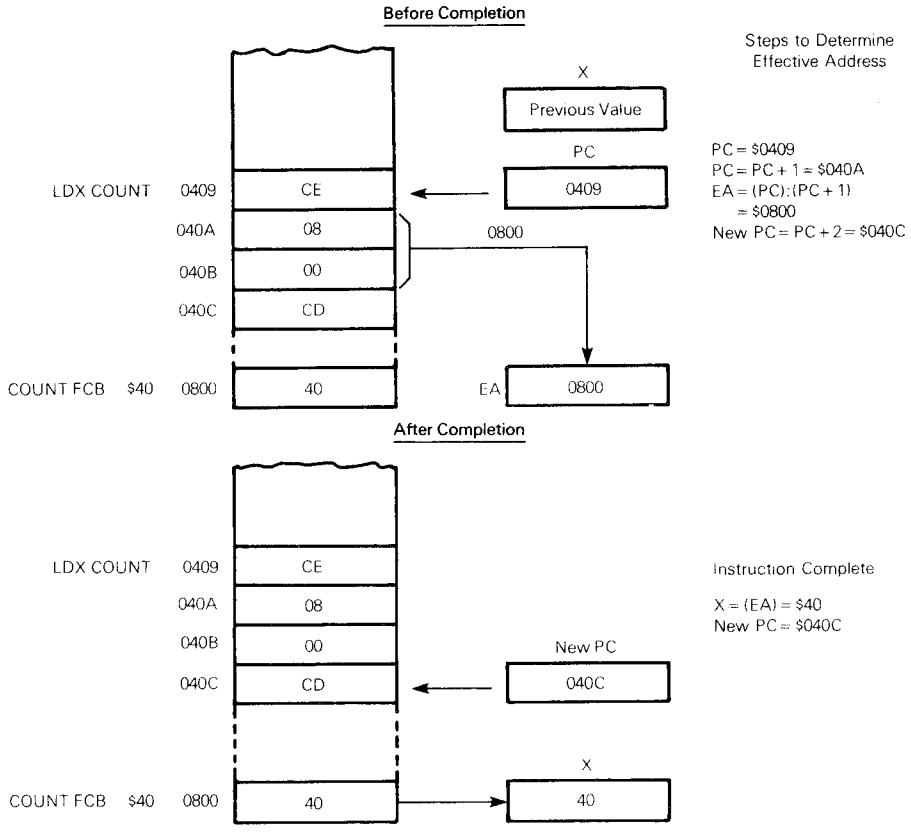


Figure 2-5. Extended Addressing Mode Example

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).

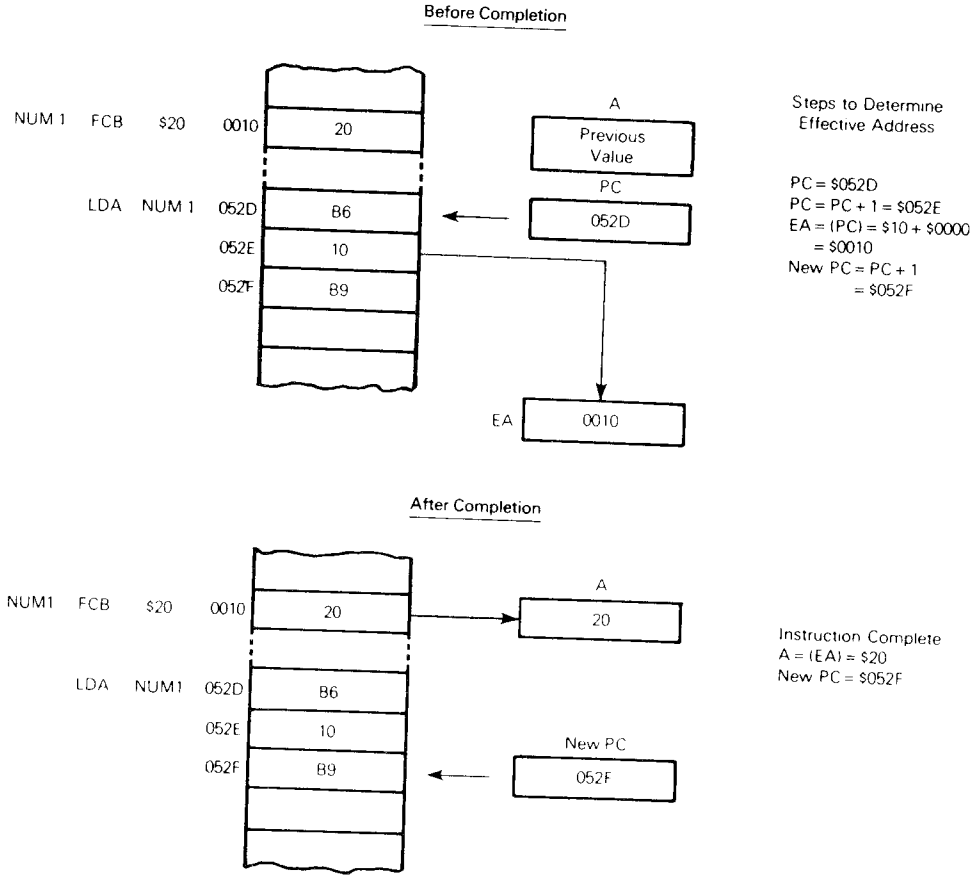


Figure 2-6. Direct Addressing Mode Example

```
                        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
```

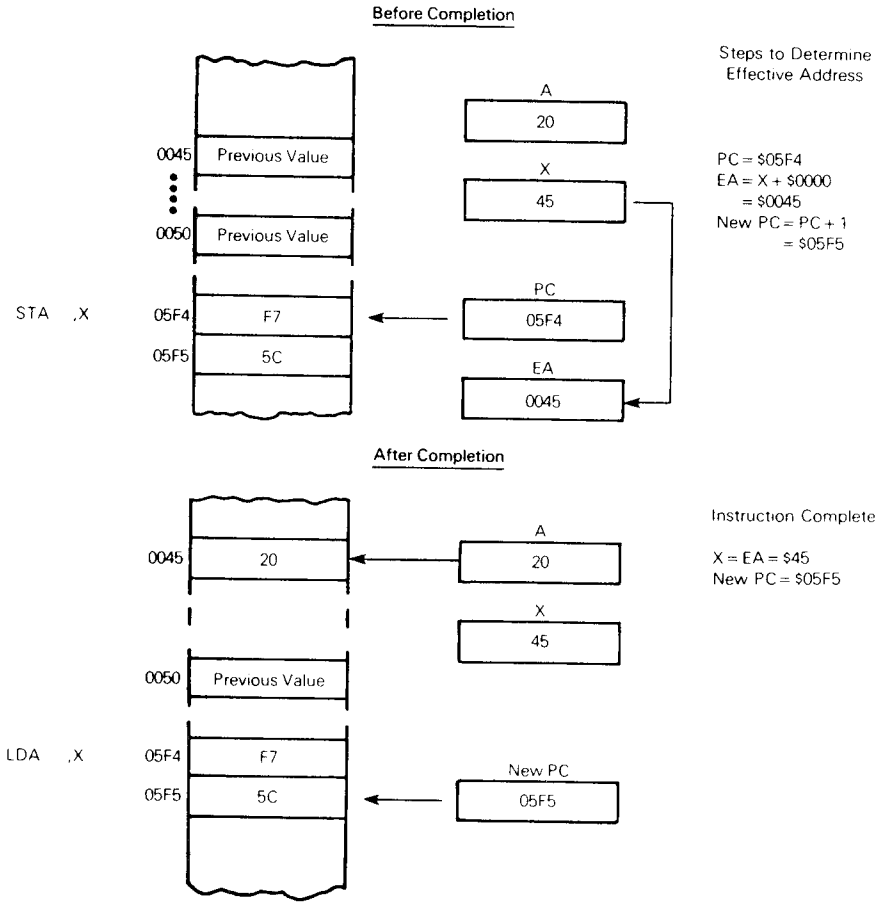


Figure 2-7. Indexed Addressing Mode, No Offset Example

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
```

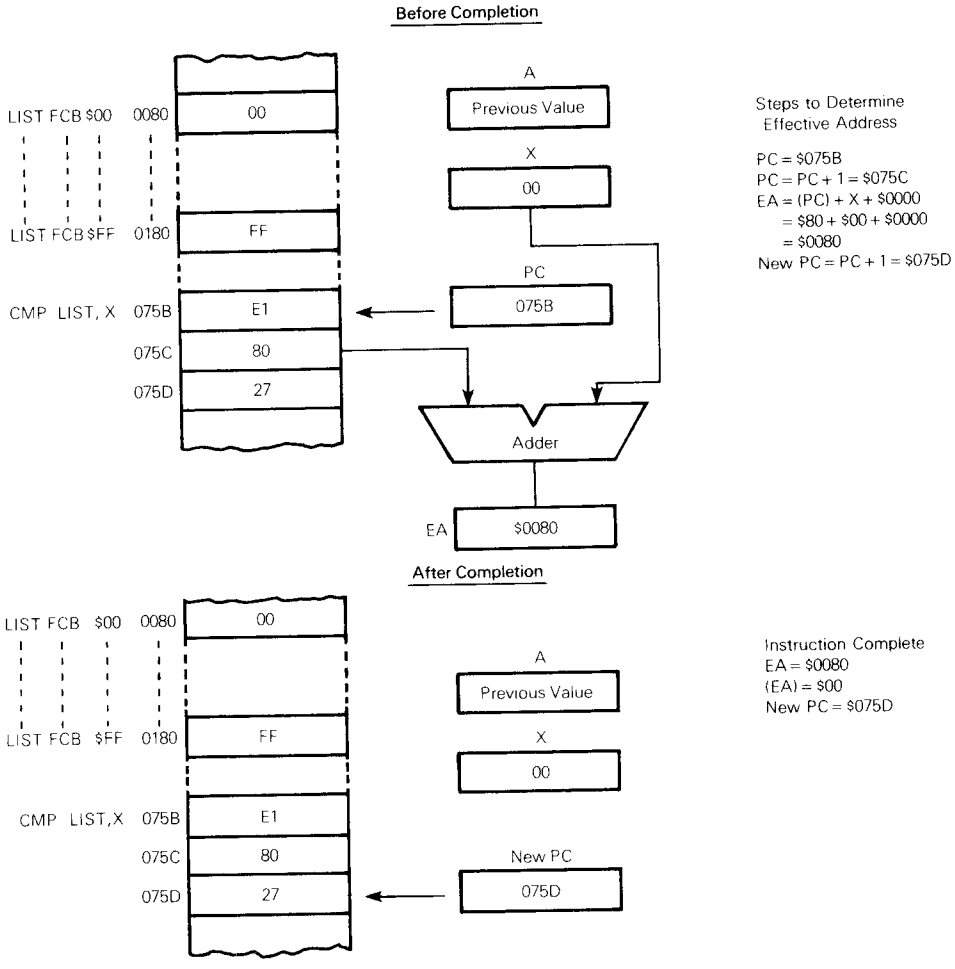


Figure 2-8. Indexed Addressing Mode, 8-Bit Offset Example

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

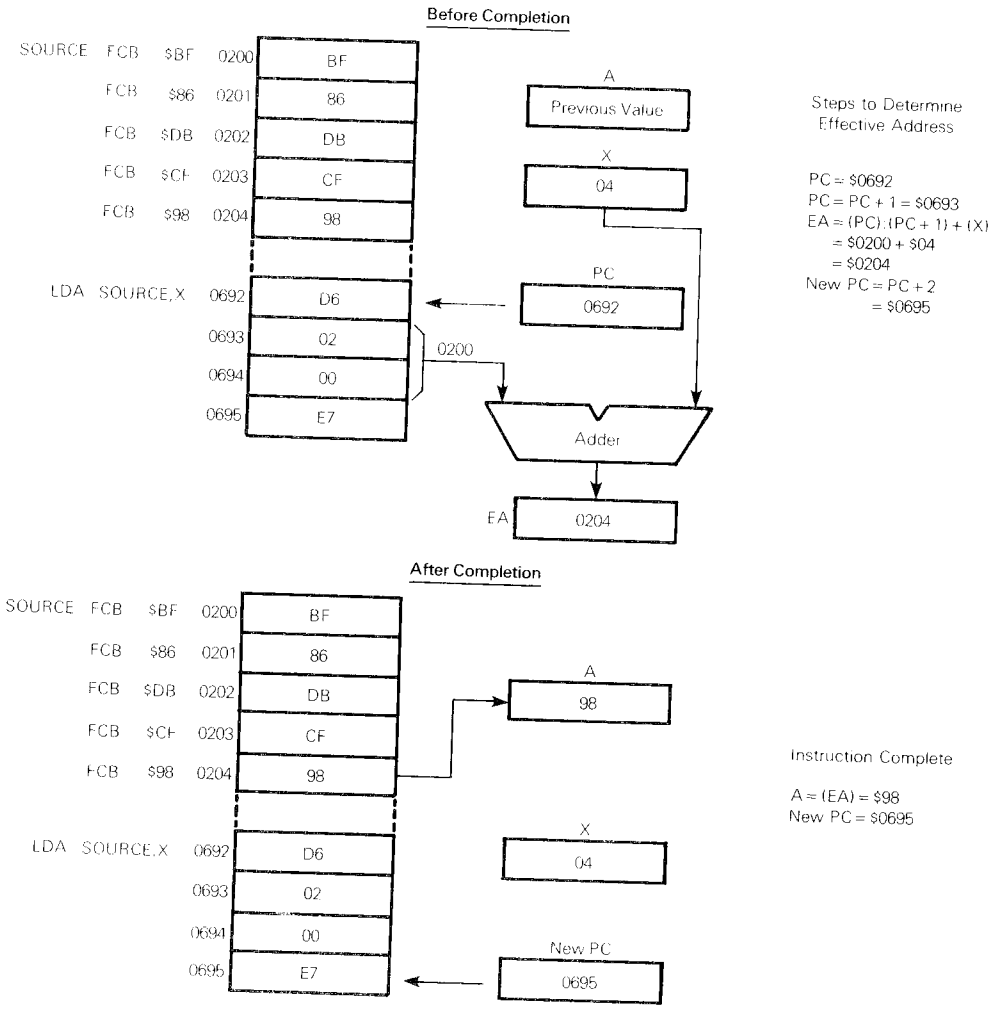


Figure 2-9. Indexed Addressing Mode, 16-Bit Offset Example

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

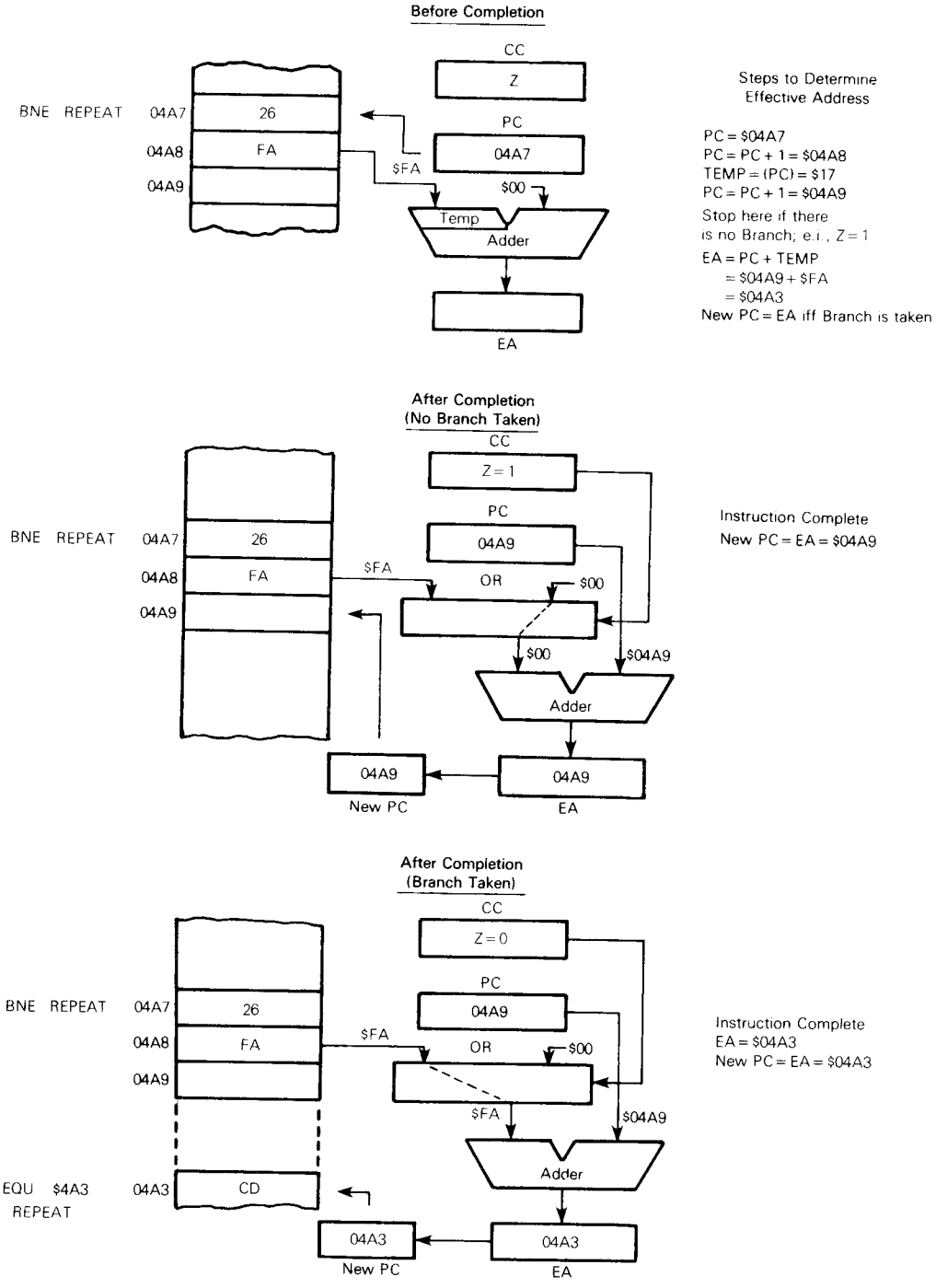


Figure 2-10. Relative Addressing Mode Example

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.)

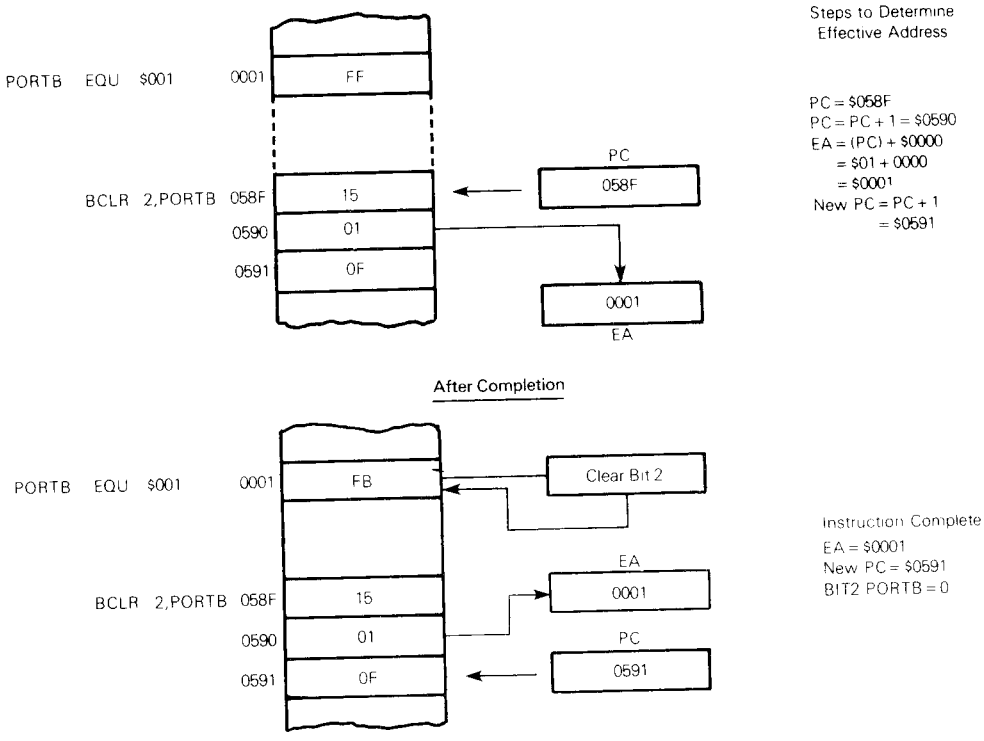


Figure 2-11. Bit Set/Clear Addressing Mode Example

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.

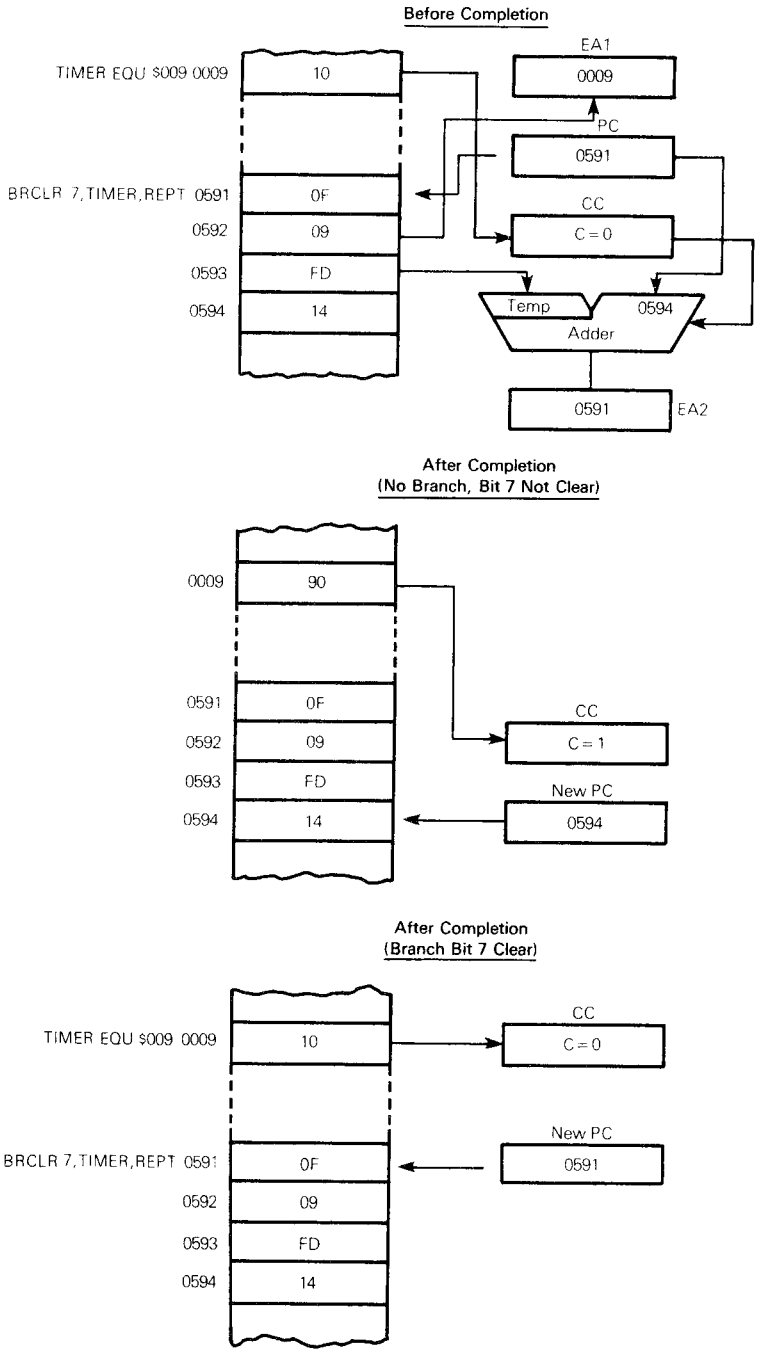


Figure 2-12. Bit Test and Branch Addressing Mode Example

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.

CHAPTER 3
SOFTWARE APPLICATIONS

3.1 INTRODUCTION

The term "software" is generally used to define computer programs and, in its broadest sense, it refers to an entire set of programs, procedures, and all related documentation associated with a system. In this manual, software refers to programs or routines. The writing of software is best learned by the experience of writing your own programs; however, a few good examples can certainly speed the learning experience. The examples provided in this chapter illustrate various M6805 HMOS/M146805 CMOS Family software features and include some commonly used routines. Included at the end of this chapter is a small debug monitor program named ASSIST05. The ASSIST05 debug monitor includes many features and routines which are useful for product evaluation and development. The routines described in the following paragraphs are not necessarily the most efficient; however, each may be used as a learning tool.

3.2 SERIAL I/O ROUTINES

Although serial I/O hardware is to be included on future M6805 HMOS/M146805 CMOS Family members, none exists on the current 14 family members. However, serial I/O can be implemented on any device, provided a small amount of software and port I/O overhead can be spared.

Three different serial I/O routine examples are discussed in this chapter. The first example generates the serial data and clock inputs for the MC145000 multiplexed LCD driver. The second example generates serial data in an NRZ format, for use with an RS-232 interface, at speeds up to 9600 baud. The third example generates serial data in an NRZ format for use in a serial loop interface.

3.2.1 MC145000 Serial I/O Software

The MC145000 (Master) LCD driver is designed to drive liquid crystal displays in a multiplexed-by-four configuration. It can drive up to 48 LCD segments or six 7-segment plus decimal point characters. The required hardware connections are shown in Figure 3-1. Data for each character must be translated into a format that produces the desired display. Table 3-1 provides a listing of the display format (hexadecimal code) for each displayed character. After the format translation is completed, data can be clocked serially into the MC145000 LCD driver. Each segment of 7-segment character plus

decimal point is represented by one bit of an 8-bit byte. As shown in Figure 3-2, a logic "1" in any bit will activate the corresponding segment of the character, plus decimal points.

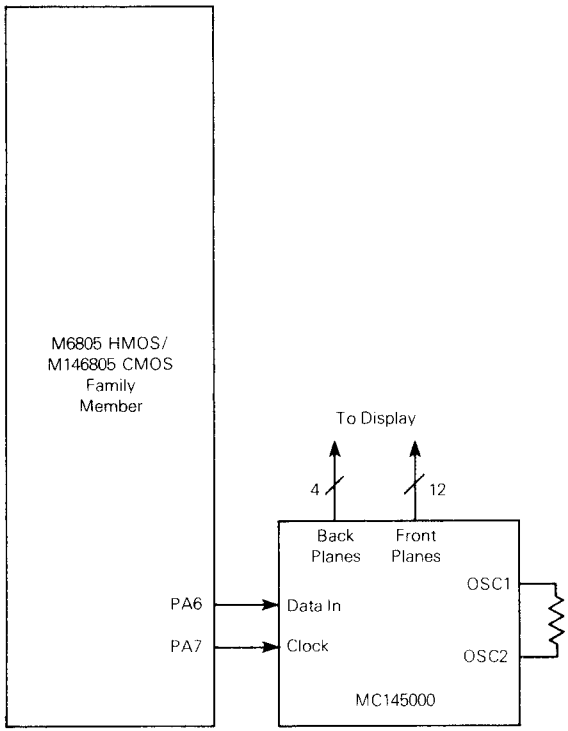


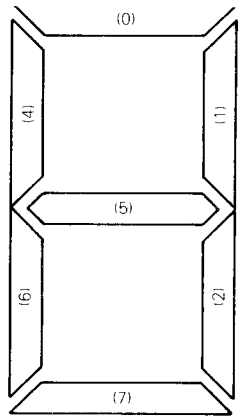
Figure 3-1. MC145000 LCD Driver Interface Schematic Diagram

Table 3-1. Display Format Conversions

Displayed Character	Display Format Hex Code
0	D7
1	06
2	E3
3	A7
4	36
5	B5
6	F5
7	07
8	F7
9	B7
A	77
b	F4
C	D1

Displayed Character	Display Format Hex Code
d	E6
E	F1
F	71
P	73
Y	B6
H	76
U	D6
L	D0
blank	00
- (dash)	20
= (equal)	A0
n	64
r	60
° (degrees)	33

NOTE: A Decimal point can be added to all but the right-most display digit by setting b3 [segment (3)] to a 1.



NOTE: Digits in parentheses represent equivalent bits of the activating byte.

Figure 3-2. 7-Segment Display Format

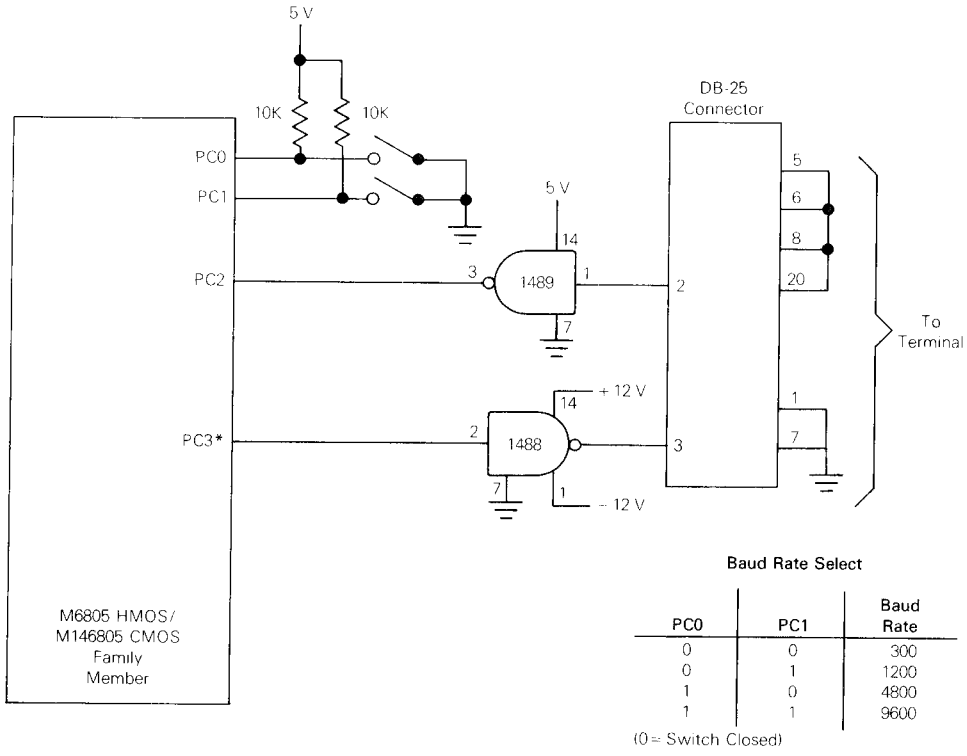
Figure 3-3 contains two software subroutine examples: DISPLY and DISTAB. The DISPLY subroutine clocks data from the accumulator into the MC145000. The DISTAB subroutine loads an eight character table into the MC145000. Note that in the DISPLY subroutine the use of bit manipulation (BSET and BCLR) helps keep the subroutine short and relatively simple. In this case, port A bits 6 and 7 are used for the data and clock lines; however, any port lines could have been stipulated in the program.

```
                    ****************************************
                    *                                      *
                    *       DISPLAY TABLE CONTENTS         *
                    *                                      *
                    *       A,X REGISTERS DESTROYED        *
                    *                                      *
                    ****************************************
                    *
AE Ø5         A DISTAB LDX     #5
E6 49         A DISCHR LDA     DTABL,X  LOAD DISPLAY
AD Ø9      1EØC        BSR     DISPLY   TABLE INTO
5A                     DECX             145ØØØ
2A F9      1DFF        BPL     DISCHR
81                     RTS
                    ****************************************
                    *                                      *
                    *       SHIFT ONE CHARACTER INTO       *
                    *             DISPLAY                  *
                    *                                      *
                    *       A REGISTER DESTROYED           *
                    *                                      *
                    ****************************************
                    *
BF 5Ø         A DISPLY STX     WORK1    SAVE INDEX
1D ØØ         A        BCLR    6,PORTA  CLEAR DATA
AE Ø8         A        LDX     #8
48              DIS1   LSLA             SET UP
24 Ø2      1E17        BCC     DIS2     BIT OF
1C ØØ         A        BSET    6,PORTA  ACCUMULATOR
1E ØØ         A DIS2   BSET    7,PORTA  CLOCK
1F ØØ         A        BCLR    7,PORTA  IT
1D ØØ         A        BCLR    6,PORTA  CLEAR DATA
5A                     DECX             COMPLETE?
26 F2      1E12        BNE     DIS1     NO
BE 5Ø         A        LDX     WORK1    RESTORE INDEX
81                     RTS
```

Figure 3-3. Serial I/O Display Subroutine Examples

3.2.2 Serial I/O Software for RS-232

The example discussed here uses two I/O port lines as the serial input and output lines. Figure 3-4 contains a schematic diagram of an RS-232 interface for serial I/O. Included as part of Figure 3-4 is the baud rate selection table showing baud rates of 300, 1200, 4800, and 9600. The example subroutine is illustrated in Figure 3-5. In this example, PC2 is used as the input line and PC3 is used as the output line. Software loops are used to generate the desired baud rates; therefore, the crystal frequency (f_{OSC}) is critical (3.579545 MHz). The subroutine example shown in Figure 3-5 is taken from the MC146805G2( )1 evaluation monitor. The same subroutine is essentially used in all M6805 HMOS/M146805 CMOS Family evaluation programs; however, in the example, the instructions followed by the comment "CMOS DITTO" or "CMOS EQUALIZATION" cannot be used with HMOS versions of the evaluation program. These extra instructions are necessary in the CMOS version to "make-up" for the generally fewer cycles-per-instruction of the M146805 CMOS Family members.



Baud Rate Select

PC0	PC1	Baud Rate
0	0	300
0	1	1200
1	0	4800
1	1	9600

(0 = Switch Closed)

* For devices which have port C as input-only, use PB7

Figure 3-4. RS-232 Interface for Serial I/O via I/O Port Lines Schematic Diagram

```
                        *
                        *       S E R I A L   I / O   R O U T I N E S
                        *
                        *       THESE SUBROUTINES ARE MODIFICATIONS OF THE ORIGINAL NMOS
                        *       VERSION.  DIFFERENCES ARE DUE TO THE VARIATION IN CYCLE
                        *       TIME OF CMOS INSTRUCTIONS VS. NMOS.
                        *
                        *       SINCE THE INT AND TIMER INTERRUPT VECTORS ARE USED IN THE
                        *       BICYCLE ODOMETER, THE I-BIT SHOULD  ALWAYS  BE  SET  WHEN
                        *       RUNNING  THE  MONITOR.  HENCE, THE CODE THAT FIDDLES WITH
                        *       THE I-BIT HAS BEEN ELIMINATED.
                        *
                        *
                        *       DEFINITION OF SERIAL I/O LINES
                        *
                        *       NOTE: CHANGING °IN' OR °OUT' WILL NECESSITATE CHANGING THE
                        *       WAY °PUT' IS SETUP DURING RESET.
                        *
07C3 00 02              PUT     EQU     PORTC   SERIAL I/O PORT
07C3 00 02              IN      EQU     2       SERIAL INPUT LINE#
07C3 00 03              OUT     EQU     3       SERIAL OUTPUT LINE#
                        *
                        *       GETC --- GET A CHARACTER FROM THE TERMINAL
                        *
                        *       A GETS THE CHARACTER TYPED, X IS UNCHANGED.
                        *
07C3 BF 15              GETC    STX     XTEMP   SAVE X
07C5 A6 08                      LDA     #8      NUMBER OF BITS TO READ
07C7 B7 17                      STA     COUNT
07C9 04 02 FD           GETC4   BRSET   IN,PUT,GETC4 WAIT FOR HILO TRANSITION
                        *
                        *       DELAY 1/2 BIT TIME
                        *
07CC B6 02                      LDA     PUT
07CE A4 03                      AND     #%11    GET CURRENT BAUD RATE
07D0 97                         TAX
07D1 DE 08 4B                   LDX     DELAYS,X GET LOOP CONSTANT
07D4 A6 04              GETC3   LDA     #4
07D6 9D                 GETC2   NOP
07D7 4A                         DECA
07D8 26 FC                      BNE     GETC2
07DA 5D                         TSTX            LOOP PADDING
07DB 14 02                      BSET    IN,PUT  DITTO
07DD 14 02                      BSET    IN,PUT  CMOS DITTO
07DF 5A                         DECX
07E0 26 F2                      BNE     GETC3   MAJOR LOOP TEST
                        *
                        *       NOW WE SHOULD BE IN THE MIDDLE OF THE START BIT
                        *
07E2 04 02 E4                   BRSET   IN,PUT,GETC4 FALSE START BIT TEST
07E5 7D                         TST     ,X      MORE TIMING DELAYS
07E6 7D                         TST     ,X
07E7 7D                         TST     ,X
                        *
                        *       MAIN LOOP FOR GETC
                        *
07E8 AD 46              GETC7   BSR     DELAY   (6) COMMON DELAY ROUTINE
07EA 05 02 00                   BRCLR   IN,PUT,GETC6 (5)   TEST INPUT AND SET C-BIT
07ED 7D                 GETC6   TST     ,X      (4) TIMING EQUALIZER
07EE 9D                         NOP             (2) CMOS EQUALIZATION
07EF 9D                         NOP             (2) CMOS EQUALIZATION
07F0 9D                         NOP             (2) CMOS EQUALIZATION
07F1 9D                         NOP             (2) CMOS EQUALIZATION
07F2 9D                         NOP             (2) CMOS EQUALIZATION
07F3 9D                         NOP             (2) CMOS EQUALIZATION
07F4 36 16                      ROR     CHAR    (5) ADD THIS BIT TO THE BYTE
07F6 3A 17                      DEC     COUNT   (5)
07F8 26 EE                      BNE     GETC7   (3) STILL MORE BITS TO GET(SEE?)
                        *
07FA AD 34                      BSR     DELAY   WAIT OUT THE 9TH BIT
07FC B6 16                      LDA     CHAR    GET ASSEMBLED BYTE
07FE BE 15                      LDX     XTEMP   RESTORE X
                        *
```

Figure 3-5. Serial I/O Software Subroutine Example

```
0800 81                         RTS             AND RETURN
                        *
                        *       PUTC --- PRINT A ON THE TERMINAL
                        *
                        *       X AND A UNCHANGED
                        *
0801 B7 16              PUTC    STA     CHAR
0803 B7 14                      STA     ATEMP   SAVE IT IN BOTH PLACES
0805 BF 15                      STX     XTEMP   DON'T FORGET ABOUT X
0807 A6 09                      LDA     #9      GOING TO PUT OUT
0809 B7 17                      STA     COUNT   9 BITS THIS TIME
080B 5F                         CLRX            FOR VERY OBSCURE REASONS
080C 98                         CLC             THIS IS THE START BIT
080D 20 02                      BRA     PUTC2   JUMP IN THE MIDDLE OF THINGS
                        *
                        *       MAIN LOOP FOR PUTC
                        *
080F 36 16              PUTC5   ROR     CHAR    (5) GET NEXT BIT FROM MEMORY
0811 24 04              PUTC2   BCC     PUTC3   (3) NOW SET OR CLEAR PORT BIT
0813 16 02                      BSET    OUT,PUT
0815 20 04                      BRA     PUTC4
0817 17 02              PUTC3   BCLR    OUT,PUT (5)
0819 20 00                      BRA     PUTC4   (3) EQUALIZE TIMING AGAIN
081B DD 08 30           PUTC4   JSR     DELAY,X (7) MUST BE 2-BYTE INDEXED JSR
                        *                           THIS IS WHY X MUST BE ZERO
081E 43                         COMA            (3) CMOS EQUALIZATION
081F 43                         COMA            (3) CMOS EQUALIZATION
0820 43                         COMA            (3) CMOS EQUALIZATION
0821 3A 17                      DEC     COUNT   (5)
0823 26 EA                      BNE     PUTC5   (3) STILL MORE BITS
                        *
0825 14 02                      BSET    IN,PUT   7 CYCLE DELAY
0827 16 02                      BSET    OUT,PUT    SEND STOP BIT
                        *
0829 AD 05                      BSR     DELAY   DELAY FOR THE STOP BIT
082B BE 15                      LDX     XTEMP   RESTORE X AND
082D B6 14                      LDA     ATEMP   OF COURSE A
082F 81                         RTS
                        *
                        *       DELAY --- PRECISE DELAY FOR GETC/PUTC
                        *
0830 B6 02              DELAY   LDA     PUT     FIRST, FIND OUT
0832 A4 03                      AND     #%11    WHAT THE BAUD RATE IS
0834 97                         TAX
0835 DE 08 4B                   LDX     DELAYS,X LOOP CONSTANT FROM TABLE
0838 A6 F8                      LDA     #$F8    FUNNY ADJUSTMENT FOR SUBROUTINE OVERHEAD
083A AB 09              DEL3    ADD     #$09
083C                    DEL2
083C 9D                         NOP             CMOS EQUALIZATION
083D 4A                         DECA
083E 26 FC                      BNE     DEL2
0840 5D                         TSTX            LOOP PADDING
0841 14 02                      BSET    IN,PUT  DITTO
0843 14 02                      BSET    IN,PUT  CMOS DITTO
0845 5A                         DECX
0846 26 F2                      BNE     DEL3    MAIN LOOP
0848 9D                         NOP             CMOS EQUALIZATION
0849 9D                         NOP             CMOS EQUALIZATION
084A 81                         RTS             WITH X STILL EQUAL TO ZERO
                        *
                        *       DELAYS FOR BAUD RATE CALCULATION
                        *
                        *       THIS TABLE MUST NOT BE PUT ON PAGE ZERO SINCE
                        *       THE ACCESSING MUST TAKE 6 CYCLES.
                        *
084B 20                 DELAYS  FCB     32       300 BAUD
084C 08                         FCB     8       1200 BAUD
084D 02                         FCB     2       4800 BAUD
084E 01                         FCB     1       9600 BAUD
```

Figure 3-5. Serial I/O Software Subroutine Example (Continued)

3.2.3 Software Serial Loop

The M6805 HMOS/M146805 CMOS Family may be used in either distributed or network processing. Those family devices may also be used as intelligent peripherals, and offer a variety of special features including: A/D, PLL, timer, and I/O lines. In addition, the devices could be interfaced with one of several serial interfaces; e.g., point-to-point, multidrop, or loop.

The MC6805R2( )1 is an evaluation device which contains the serial routines required to setup a serial loop. In this configuration, one device in the loop serves as the loop master and transmits user commands to the loop. Each slave device in the loop examines the transmitted message and determines if it must execute the command. The user commands enter the loop via the terminal connected to the loop master as shown in Figure 3-6. The programmed configuration of port C lines PC0-PC3 determine which device is the master, which devices are slaves, and the loop baud rate.

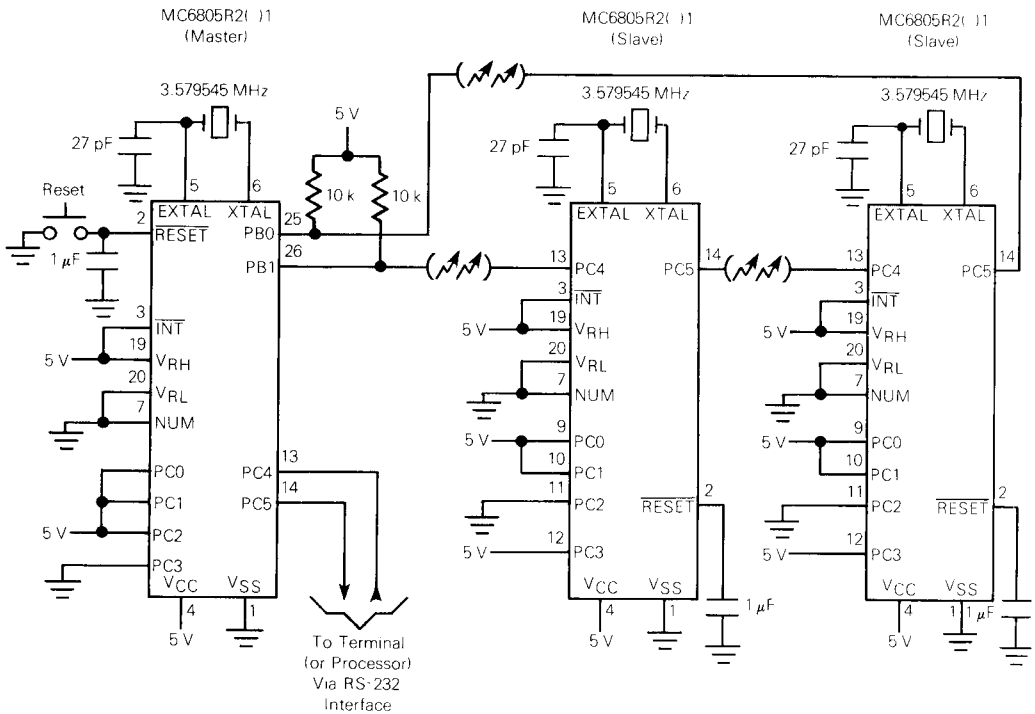


Figure 3-6. Software Serial Loop Hardware Connections Schematic Diagram

Each transmitted message contains either two or three bytes. They include:

1. destination address (node)
2. address of byte being accessed within destination plus a read/write bit
3. data, provided read/write bit in byte two indicates a write

Each device in the loop (master and slaves) examines the node address. If the node address is zero, the slave device processes the message. If the node address is non-zero, the slave device decrements the node address and passes the message to the next node in the loop.

When a slave device processes a message, data is read from or written to the address specified in the first seven bits of byte two. Only seven bits of address are necessary since the MC6805R2( )1 RAM and registers are located in the first 128 bytes. If the read/write bit in byte two indicates a write is requested, the data contained in byte three is written to the specified address; however, if the bit indicates a read is requested, the slave performs the read (from its own on-chip memory) and forms a new message that includes the maximum node address and the data just read. The maximum node address guarantees that the data is received by the loop master since it includes all devices in the loop. The loop master can then display the data in response to the user command input.

Some improvements could be made to the loop system discussed above. Improvements could include replacing the operator controlled terminal with a microprocessor (MPU), microcomputer (MCU), or an intelligent peripheral controller (IPC). The new device can submit commands to the loop in the same format as the terminal; however, when it is not providing input to the loop, it can be processing other functions not necessarily related to the loop.

The features of the slave devices could be used by the controlling MPU, MCU, or IPC. The M6805 HMOS/M146805 CMOS Family devices can be used as intelligent peripherals to provide improvements in system throughput. Serial links allow long-distance communications with minimum line costs. The example discussed above provides a simple but powerful system that can be used as a basis for a more sophisticated system.

3.3 BLOCK MOVE

One of the more commonly used routines is one in which a block of data, located in memory, is copied or moved to another memory location. The indexed addressing modes of the M6805 HMOS/M146805 CMOS Family makes the block move relatively simple.

An example of this routine is shown in Figure 3-7. In this example, the location of the first table entry is used as the offset for the indexed instruction. The index register is used to step through the table; therefore, the table may be up to 256 bytes long. This example uses a table length of 64 bytes ($40). Note that in the example of Figure 3-7, the source table and the destination are located in page zero. The difference between the two indexed instructions is the number of bytes and cycles required for execution.

```
                    SOURCE    EQU    $F0
                    DESTIN    EQU    $40
AE    20                      LDX    #$20      Load Index Register W/ Table Length
E6    F0            REPEAT    LDA    SOURCE,X  Get Table Entry
E7    40                      STA    DESTIN,X  Store Entry Table
5A                            DECX             Next Entry
26    F8                      BNE    REPEAT    REPEAT If More
```

Figure 3-7. Block Move Routine Example

3.4 STACK EMULATION

By proper use of the stack, the versatility of a program can be increased. This can be done by allowing registers or values to be stored temporarily in RAM and then later retrieved. Variables which are stored in the stack are always positioned relative to the top of the stack. Stacks operate in a last-in-first-out (LIFO) fashion; that is, the last byte in is the first byte that can be retrieved. Because of this LIFO characteristic, the stack is useful for passing subroutine variables as well as other valuable programming tools.

The M6805 HMOS/M146805 CMOS Family stack is reserved for subroutine return addresses and for saving register contents during interrupts. This is sufficient for most control-oriented applications; however, the routine shown in Figure 3-8 can provide the MC146805E2 MPU with additional stack capability for temporary variable storage. In this routine, a temporary location called POINTER serves to hold the relative address of the next free stack location. When the routine is entered, the contents of POINTER are transferred to the index register. The two-byte indexed addressing mode is used to allow the stack to be located in any part of RAM. Since the index register is used to provide a relative address, the stack wraps around if more than 256 locations are pushed onto the stack. The stacking routine shown in Figure 3-8 uses two fixed temporary locations: one (called POINTER) is used to save the stack pointer and the other (called TEMPX) is used as a temporary storage for the index register. However, if the index register can be dedicated to the stack, both temporary locations can be deleted. In this example, two subroutines, PUSH and PULL are used to manipulate data. Subroutine PUSH is used by first loading the accumulator with the data to be saved and then performing a subroutine call to PUSH. Subroutine PULL is used by calling the subroutine PULL after which the data retrieved is contained in the accumulator.

NOTE

If a single-chip MCU is used instead of the MC146805E2, the stack must be located in RAM and a routine must check that the boundaries are not exceeded.

```
            ORG     $10
TEMP        RMB     1
POINTR      RMB     1
STACK       EQU     $3FF
            ORG     $1000
PUSH        STX     TEMPX       Save Index Reg Contents
            LDX     POINTR      Get Pointer
            STA     STACK,X     Save Byte at Stack and Pointer
            DEC     POINTR      Adjust Pointer
            LDX     TEMPX       Retrieve Index Reg Contents
            RTS

PULL        STX     TEMPX
            INC     POINTR
            LDX     POINTR
            LDA     STACK,X
            LDX     TEMPX
            RTS
```

Figure 3-8. Stack Emulation Routine

3.5 KEYPAD SCAN ROUTINE

A common task for control-oriented microprocessors is to scan a 4 × 4 keypad, such as the one illustrated in the example of Figure 3-9. The example shown uses port A lines 4-7 as scanning outputs and port A lines 0-3 as sensing inputs. The routine example shown in Figure 3-10 is intended for use with CMOS microprocessors; however, it could be modified as discussed below for HMOS microprocessors. It is often desirable to place M146805 CMOS Family members in a low-power mode; therefore, the STOP instruction is incorporated in the routine shown in Figure 3-10.

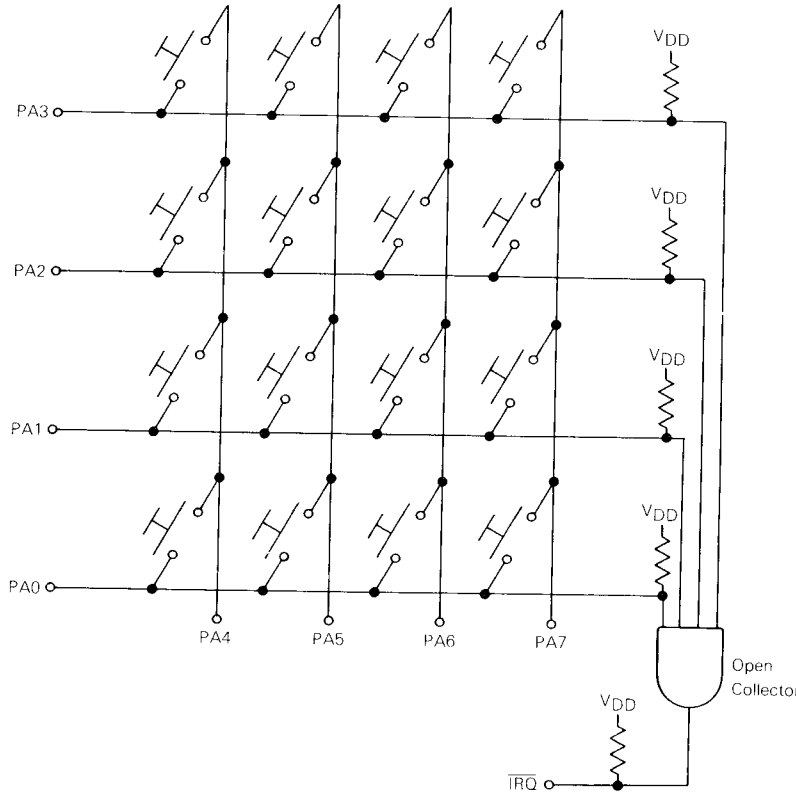


Figure 3-9. 4 x 4 Keypad and Closure Detection Circuit Schematic Diagram

The example shown in Figure 3-10 uses an interrupt driven routine and supports either the STOP or a normal wait for interrupt (see below). If one of the keypad switches is depressed while in the STOP mode, the $\overline{IRQ}$ line goes low. (This is the result of the port A scanning lines being low in the STOP mode.) When IRQ goes low the KEYSCN vector is selected and calls the KEYSCN interrupt service routine. The interrupt service routine first causes IRQ to go high and then scans each column (PA4-PA7) individually to determine which keypad switch was depressed. Once the closed keypad switch is detected, the information is stored and a debounce subroutine is called to verify the closure. The

debounce consists of checking for a keypad switch closure after a 1536 bus cycle (2040 for HMOS) delay to assure that the interrupt was not a result of noise. If a keypad switch closure still exists after the debounce is completed, the routine waits for the switch to be released before forcing all scanning lines low for detection of the next closure. (A Schmitt trigger input on the $\overline{IRQ}$ line further reduces the effects of noise.) Once the key closure is verified, a decode routine is used to determine which keypad was switch closed. If after the debounce subroutine is completed, no keypad switch is detected as being closed, the closure is considered invalid and the processor again enters the STOP mode.

```
PAGE  001  KEYSCN  .SA:0

00001                              OPT    CMOS
00002                          *
00003           0000     A PORTA   EQU    0
00004           0004     A DDRA    EQU    4
00005           0180     A DECODE  EQU    $180
00006                          *
00007A 0100                        ORG    $100
00008                          *
00009A 0100 3F 00        A RESET   CLR    PORTA     PREPARE SCANNING LINES
00010A 0102 A6 F0        A         LDA    #$F0      PA4-PA7 AS OUTPUTS
00011A 0104 B7 04        A         STA    DDRA      WHICH OUTPUT LOWS
00012A 0106 8E                 STOP  STOP           ENTER LOW PWR MODE - WAIT FOR INT
00013A 0107 20 FD     0106         BRA    STOP
00014                          *
00015A 0109 A6 EF        A KEYSCN  LDA    #$EF      CHECK 1ST COLUMN WITH A LOW
00016A 010B B7 00        A         STA    PORTA     AND OTHERS HIGH
00017A 010D 2E 05     0114 REPEAT  BIL    GOTIT     IF IRQ LINE LOW, COLUMN FOUND
00018A 010F 38 00        A         LSL    PORTA     ELSE TRY NEXT COLUMN
00019A 0111 25 FA     010D         BCS    REPEAT    REPEAT IF MORE COLUMNS, ELSE
00020A 0113 80                 RETURN RTI           WAIT FOR VALID CLOSURE
00021                          *
00022A 0114 B6 00        A GOTIT   LDA    PORTA     SAVE KEY IN ACCA
00023A 0116 AD 0C     0124         BSR    DBOUNC    WAIT 1.5K BUS CYCLES (2K FOR HMOS)
00024A 0118 2F F9     0113         BIH    RETURN    IF IRQ LINE HIGH, INVALID CLOSURE
00025A 011A 2E FE     011A RELEAS  BIL    RELEAS    WAIT FOR KEY RELEASE
00026A 011C AD 06     0124         BSR    DBOUNC    PAUSE
00027A 011E 2E FA     011A         BIL    RELEAS    IF IRQ LINE LOW, KEY NOT RELEASED
00028A 0120 3F 00        A         CLR    PORTA     PREPARE SCAN LINES FOR STOP MODE
00029A 0122 20 5C     0180         BRA    DECODE    GO TO USER KEY DECODE ROUTINE
00030                          *
00031A 0124 AE FF        A DBOUNC  LDX    #$FF
00032A 0126 5A                 AGAIN DECX           LOOPS 1536 TIMES FOR CMOS
00033A 0127 26 FD     0126         BNE    AGAIN     OR 2040 FOR HMOS
00034A 0129 81                     RTS
00035                          *
00036A 012A 80                 TWIRQ RTI
00037A 012B 80                 TIRQ  RTI
00038A 012C 80                 SWI   RTI
00039                          *
00040A 07F6                        ORG    $7F6
00041                          *
00042A 07F6     012A     A         FDB    TWIRQ     TIMER WAIT VECTOR
00043A 07F8     012B     A         FDB    TIRQ      TIMER INTERNAL VECTOR
00044A 07FA     0109     A         FDB    KEYSCN    EXTERNAL INTERRUPT VECTOR
00045A 07FC     012C     A         FDB    SWI       SOFTWARE INTERRUPT VECTOR
00046A 07FE     0100     A         FDB    RESET     RESET VECTOR
00047                          *
00048                              END
TOTAL ERRORS 00000--00000
```

Figure 3-10. KEYSCN Routine Example

A value which represents the position of the closed keypad switch is passed, via the accumulator, to a routine which decodes the position either into a number or a pointer for other routines. All routines which require that the keypad be scanned, can enter the routine either by using the STOP mode (as discussed above for CMOS) or by enabling the external interrupt with a CLI instruction. The CLI instruction then requires a BRA instruction to wait for a keypad switch closure to generate an interrupt.

3.6 DAA (DECIMAL ADJUST ACCUMULATOR)

Although the M6805 HMOS/M146805 CMOS Family is primarily a controller, it is occasionally required to perform arithmetic operations on BCD numbers. Since the ADD instruction operates on binary data, the result of the ADD instruction must be adjusted in these cases. A DAA subroutine example is shown in Figure 3-11. The DAA subroutine should be called immediately after the binary ADD instruction.

```
PAGE  001  DAA      .SA:1

00001                          *
00002                          *  DAA --- DECIMAL ADJUST ACCUMULATOR
00003                          *
00004                          *  THIS INSTRUCTION SIMULATES THE DAA INSTRUCTION
00005                          *  AVAILABLE ON 6800, 6801, AND 6809 PROCESSORS.
00006                          *
00007                          *  THE SUBROUTINE SHOULD BE CALLED IMMEDIATELY AFTER
00008                          *  AN 'ADC' OR 'ADD' INSTRUCTION WHEN PERFORMING BCD
00009                          *  ARITHMETIC.
00010                          *
00011                          *  EXAMPLE:
00012                          *            LDA ARG1
00013                          *            ADD ARG2
00014                          *            BSR DAA
00015                          *
00016                          *  AT ENTRY:
00017                          *             A -- RESULT OF PREVIOUS ADD OR ADC
00018                          *            CC -- RESULT OF PREVIOUS ADD OR ADC
00019                          *
00020                          *  AT EXIT:
00021                          *             A -- CORRECTED BCD NUMBERS
00022                          *            CC -- CARRY BIT SET OR CLEARED FOR
00023                          *                  MULTI-PRECISION ARITHMETIC.
00024                          *
00025                          *  NO WORK AREA IS NEEDED; AND, THE INDEX REGISTER IS
00026                          *  UNAFFECTED.  TWO OR FOUR STACK LOCATIONS MAY BE USED
00027                          *  FOR SUBROUTINE RETURN ADDRESSES.
00028                          *
00029A 0080                          ORG    $80
00030                          *
00031A 0080 25 04    0086 DAA      BCS    DAAHAI    IF CARRY THEN ADJUST HIGH DIGIT
00032A 0082 A1 99      A           CMP    #$99      DOUBLE OVERFLOW? (>99?)
00033A 0084 23 08    008E          BLS    DAALOW    NO, CHECK LOW DIGIT
00034A 0086 40             DAAHAI NEGA              AVOID CLOBBERING H-BIT BY
00035A 0087 A0 60      A           SUB    #$60      A + $60 = - (- A - $60)
00036A 0089 40                     NEGA
00037                          *  THE ABOVE ADJUST MEANS WE MUST RETURN WITH CARRY SET
00038A 008A AD 02    008E          BSR    DAALOW    CHECK LOW DIGIT
00039A 008C 99                     SEC              SET CARRY BIT
00040A 008D 81                     RTS              RETURN WITH CARRY SET
00041                          *  CHECK LOW DIGIT FOR OVERFLOW
00042A 008E 28 03    0093 DAALOW BHCC   DAANOO    NO OVERFLOW DETECTED
00043A 0090 AB 06      A           ADD    #6        ADJUST FOR KNOWN OVERFLOW
00044A 0092 81                     RTS              RETURN WITH CARRY CLEAR
00045A 0093 AB 06      A DAANOO    ADD    #6        LOW DIGIT A-F?
00046A 0095 29 02    0099          BHCS   DAARTS    BRANCH ADJUSTED IF
00047A 0097 A0 06      A           SUB    #6        CORRECT ASSUMPTION
00048A 0099 81             DAARTS RTS              RETURN WITH CARRY CLEAR
00049                              END
```

Figure 3-11. DAA Subroutine Example

3.7 MULTIPLY

Multiply subroutines for either 16-bit × 16-bit or 8-bit × 8-bit multiplications can be written using less than 30 bytes. Examples of both cases are illustrated in Figures 3-12 and 3-13. The 16-bit × 16-bit routine is from an example in the User's Group Library. The 8-bit × 8-bit routine of Figure 3-13 is also included in the MC146805G2( )1 evaluation program.

```
PAGE  001  DPMUL05 .SA:0

00001                          *
00002                          *  REF HISPDMUP PGM IN USERS GROUP LIBRARY
00003                          *  LOAD MULTIPLIER INTO (QH,QL)
00004                          *  LOAD MULTIPLICAND INTO (PH,PL)
00005                          *  (PH,PL) * (QH,QL) ---> (TEMPA,TEMPB,QH,QL)
00006                          *
00007                          *  RESULTS ARE:
00008                          *  TEMPA    MOST SIGNIFICANT BYTE
00009                          *  TEMPB    SECOND SIGNIFICANT BYTE
00010                          *  QH       THIRD SIGNIFICANT BYTE
00011                          *  QL       LEAST SIGNIFICANT BYTE
00012                          *
00013A 0064                              ORG     $64
00014                          *
00015A 0064     0001       A PH       RMB     1
00016A 0065     0001       A PL       RMB     1
00017A 0066     0001       A TEMPA    RMB     1
00018A 0067     0001       A TEMPB    RMB     1
00019A 0068     0001       A QH       RMB     1
00020A 0069     0001       A QL       RMB     1
00021                          *
00022A 0080                              ORG     $80
00023A 0080 AE 10          A STRT     LDX     #16
00024A 0082 3F 66          A          CLR     TEMPA
00025A 0084 3F 67          A          CLR     TEMPB
00026A 0086 36 68          A          ROR     QH
00027A 0088 36 69          A          ROR     QL
00028A 008A 24 0C       0098 NXT      BCC     ROTAT
00029A 008C B6 67          A          LDA     TEMPB
00030A 008E BB 65          A          ADD     PL
00031A 0090 B7 67          A          STA     TEMPB
00032A 0092 B6 66          A          LDA     TEMPA
00033A 0094 B9 64          A          ADC     PH
00034A 0096 B7 66          A          STA     TEMPA
00035A 0098 36 66          A ROTAT    ROR     TEMPA
00036A 009A 36 67          A          ROR     TEMPB
00037A 009C 36 68          A          ROR     QH
00038A 009E 36 69          A          ROR     QL
00039A 00A0 5A                           DECX
00040A 00A1 26 E7       008A          BNE     NXT
00041A 00A3 81                           RTS
00042                          *
00043                                    END
```

Figure 3-12. 16-bit × 16-bit Multiplication Subroutine Example

```
                        ********************************************************************
                        *
                        *  MULTIPLY
                        *
                        *       8 BIT BY 8 BIT UNSIGNED MULTIPLY
                        *       OPERANDS IN A AND X ON ENTRY
                        *       16 BIT RESULT IN X:A ON EXIT; X HAS MSB.
                        *
                        *       AVERAGE EXECUTION = 323 CYCLES
                        *       WORST CASE = 425 CYCLES
                        *
                        *
                        *************************
                        *
                        * MULTIPLY / DIVIDE VARIABLES
                        *
                        *************************
                        *
                        * DIVISOR, MTOTAL
                        *
                        *       DIVISOR FOR 16 BIT / 16 BIT DIVIDE ROUTINE
                        *       ALSO USED AS A TEMP IN MULTIPLY
                        *
        005D      A MTOTAL EQU     *
005D    0002      A DIVSR  RMB     2
                        *
                        * DIVIDEND
                        *
                        *       DIVIDEND FOR 16 BIT / 16 BIT DIVIDE ROUTINE
                        *
005F    0002      A DIVDND RMB     2
                        *
                        * TEMPORARY BYTE
                        *
0061    0001      A TEMP   RMB     1
                        *
                        * SAVEX, MCOUNT
                        *
                        *       TEMPORARY STORAGE FOR X REGISTER IN DIVIDE
                        *       ALSO USED FOR COUNTER IN MULTIPLY
                        *
        0062      A MCOUNT EQU     *
0062    0001      A SAVEX  RMB     1
                        *
        012C      A MULST  EQU     *
012C 3F 5D        A        CLR     MTOTAL    INITIALIZE RESULT TEMP
012E 3F 5E        A        CLR     MTOTAL+1
0130 B7 61        A        STA     TEMP      SAVE ONE ARGUMENT
0132 A6 09        A        LDA     #9
0134 B7 62        A        STA     MCOUNT    BYTE LENGTH = 8
                        *
                        * THE ALGORITHM IS A PLAIN SHIFT AND ADD
                        *
        0136      A BIGLOP EQU     *
0136 B6 61        A        LDA     TEMP      GET BACK ARGUMENT
        0138      A SMLOOP EQU     *
0138 3A 62        A        DEC     MCOUNT    WHILE COUNT IS NOT ZERO
013A 27 13     014F        BEQ     DONE
013C 38 5E        A        LSL     MTOTAL+1 SHIFT TOTAL LEFT BY 1
013E 39 5D        A        ROL     MTOTAL
0140 58                    LSLX             GET NEXT BIT FROM X
0141 24 F5     0138        BCC     SMLOOP    NO ADD IF C=0
                        *
                        * C=1; ADD A TO TOTAL
                        *
0143 B7 61        A        STA     TEMP
0145 BB 5E        A        ADD     MTOTAL+1
0147 24 02     014B        BCC     NOCARY
0149 3C 5D        A        INC     MTOTAL
014B B7 5E        A NOCARY STA     MTOTAL+1
014D 20 E7     0136        BRA     BIGLOP
                        *
                        * HERE TO EXIT
                        *
        014F      A DONE   EQU     *
014F BE 5D        A        LDX     MTOTAL
0151 B6 5E        A        LDA     MTOTAL+1 RETURN RESULT IN A:X
0153 81                    RTS
```

Figure 3-13. 8-bit × 8-bit Multiplication Subroutine Example

3.8 DIVIDE

Two examples of subroutines which can be used for performing division of two numbers are illustrated in Figures 3-14 and 3-15. One subroutine performs a 16-bit ÷ 16-bit with an 8-bit result and the other performs a 16-bit ÷ 16-bit with a 16-bit result. The subroutine of Figure 3-14 is included as part of the MC146805G2( )1 evaluation program. The subroutine of Figure 3-15 is from the User's Group Library. Notice that neither subroutine requires more than 50 bytes.

```
                    ********************************************************************
                    *
                    *       D I V I D E   R O U T I N E
                    *
                    ********************************************************************
                    *
                    *
                    *       16 BIT / 16 BIT --> 8 BIT RESULT        DIVIDE
                    *
                    *       ON ENTRY:
                    *          DIVSR CONTAINS THE DIVISOR
                    *          DIVDND CONTAINS THE DIVIDEND
                    *
                    *       ON EXIT:
                    *          A CONTAINS THE ROUNDED QUOTIENT
                    *          DIVSR AND DIVDND ARE DESTROYED
                    *          TEMP IS DESTROYED
                    *
                    *          IF DIVISION BY ZERO, 255 IS RETURNED.
                    *
                    *       ADAPTED FROM A 6801 DIVIDE ALGORITHM FOUND IN THE 6801
                    *       USER'S MANUAL WRITTEN BY BILL BRUCE.
                    *
                    *       AVERAGE EXECUTION SPEED = 644 CYCLES
                    *       WORST CASE SPEED = 1376 CYCLES
                    *
                    *
                    ********************************************************************

                    ************************
                    *
                    * MULTIPLY / DIVIDE VARIABLES
                    *
                    ************************
                    *
                    * DIVISOR, MTOTAL
                    *
                    *       DIVISOR FOR 16 BIT / 16 BIT DIVIDE ROUTINE
                    *       ALSO USED AS A TEMP IN MULTIPLY
                    *
    005D    005D    A MTOTAL EQU    *
005D        0002    A DIVSR  RMB    2
                    *
                    * DIVIDEND
                    *
                    *       DIVIDEND FOR 16 BIT / 16 BIT DIVIDE ROUTINE
                    *
005F        0002    A DIVDND RMB    2
                    *
                    * TEMPORARY BYTE
                    *
0061        0001    A TEMP   RMB    1
                    *
                    * SAVEX, MCOUNT
                    *
                    *       TEMPORARY STORAGE FOR X REGISTER IN DIVIDE
                    *       ALSO USED FOR COUNTER IN MULTIPLY
                    *
    0062    0062    A MCOUNT EQU    *
0062        0001    A SAVEX  RMB    1
                    *
```

Figure 3-14. 16-bit ÷ 16-bit With 8-Bit Result Subroutine Example

```
          00E5      A DIV     EQU     *
00E5 A6 02          A         LDA     #2        SET SHIFT COUNT TO GENERATE 9 BITS
00E7 B7 61          A         STA     TEMP      8 FOR RESULT PLUS 1 TO ROUND
00E9 B6 5D          A         LDA     DIVSR
00EB BE 5E          A         LDX     DIVSR+1
00ED 26 07     00F6           BNE     NOZERO    CHECK FOR DIVISION BY ZERO
00EF 4D                       TSTA
00F0 26 05     00F7           BNE     DIV01     BRANCH IF NOT ZERO
00F2 A6 FF          A         LDA     #255      DIVISION BY ZERO
00F4 20 35     012B           BRA     DIVOUT    GO EXIT
                         *
          00F6      A NOZERO  EQU     *
00F6 4D                       TSTA              SHIFT DIVISOR LEFT UNTIL SIGN BIT =1
00F7 2B 06     00FF DIV01     BMI     OUTD
00F9 3C 61          A LOOP2   INC     TEMP
00FB 58                       LSLX
00FC 49                       ROLA              INCR SHIFT COUNT
00FD 2A FA     00F9           BPL     LOOP2
00FF B7 5D          A OUTD    STA     DIVSR     RESTORE DIVISOR
0101 BF 5E          A         STX     DIVSR+1
0103 5F                       CLRX              CLEAR PLACE FOR QUOTENT
                         * MAIN LOOP
0104 B6 60          A LOOP    LDA     DIVDND+1  DIVIDEND-DIVISOR --> DIVIDEND
0106 B0 5E          A         SUB     DIVSR+1
0108 B7 60          A         STA     DIVDND+1
010A B6 5F          A         LDA     DIVDND
010C B2 5D          A         SBC     DIVSR
010E 24 09     0119           BCC     ZOT       BRANCH IF CARRY SET (BEFORE SAVE OF DIVDND)
0110 B6 60          A         LDA     DIVDND+1  ADD IT BACK
0112 BB 5E          A         ADD     DIVSR+1   NOTE, MSB WAS NEVER STORED SO WE ONLY
0114 B7 60          A         STA     DIVDND+1  HAVE TO ADD TO THE LS BYTE
0116 58                       LSLX              SHIFT IN ZERO
0117 20 04     011D           BRA     OVER
0119 B7 5F          A ZOT     STA     DIVDND    SAVE MS BYTE OF NEW DIVIDEND
011B 99                       SEC
011C 59                       ROLX              SHIFT 1 BIT INTO QUOTIENT
011D 49                  OVER ROLA              CARRY INTO QUOTIENT
011E 34 5D          A         LSR     DIVSR     SHIFT DIVISOR RIGHT BY 1
0120 36 5E          A         ROR     DIVSR+1
0122 3A 61          A         DEC     TEMP      DONE
0124 26 DE     0104           BNE     LOOP      BRANCH IF NOT
0126 44                       LSRA              GET CORRECT QUOTIENT
0127 56                       RORX              C = ROUND BIT
0128 9F                       TXA
0129 A9 00          A         ADC     #0        AND ROUND
          012B      A DIVOUT  EQU     *
012B 81                       RTS               EXIT
```

Figure 3-14. 16-bit ÷ 16-bit With 8-bit Result Subroutine Example (Continued)

```
          DIVIDE  .SA:1

00001                     ********************************************************************************
00002                     *
00003                     *     DIVIDE:  16 Bit / 16 Bit Divide Routine with 16 Bit Result
00004                     *              Reference DIV16 program in the 6800 User's Group Library.
00005                     *              16 Bit Dividend in DVDND and DVDND+1
00006                     *              16 Bit Divisor in DVSOR and DVSOR+1
00007                     *              16 Bit Result in DVDND and DVDND+1
00008                     *
00009                     *===============================================================================
00010                     *
00011A 0040                         ORG     $40
00012                     *
00013A 0040     0001      A COUNT   RMB     1
00014A 0041     0002      A DVSOR   RMB     2
00015A 0043     0002      A DVDND   RMB     2
00016A 0045     0001      A TEMPA   RMB     1
00017                     *
00018                     *
00019                     *===============================================================================
00020                     *
00021A 0100                         ORG     $100
00022                     *
00023A 0100 A6 01         A DIVIDE  LDA     #1
00024A 0102 3D 41         A         TST     DVSOR
00025A 0104 2B 0B      0111         BMI     DIV153
00026A 0106 4C            DIV151    INCA
00027A 0107 38 42         A         ASL     DVSOR+1
00028A 0109 39 41         A         ROL     DVSOR
00029A 010B 2B 04      0111         BMI     DIV153
00030A 010D A1 11         A         CMP     #17
00031A 010F 26 F5      0106         BNE     DIV151
00032A 0111 B7 40         A DIV153  STA     COUNT     Save the Counter
00033A 0113 B6 43         A         LDA     DVDND
00034A 0115 BE 44         A         LDX     DVDND+1
00035A 0117 3F 43         A         CLR     DVDND
00036A 0119 3F 44         A         CLR     DVDND+1
00037A 011B B7 45         A DIV163  STA     TEMPA
00038A 011D 9F                      TXA
00039A 011E B0 42         A         SUB     DVSOR+1
00040A 0120 97                      TAX
00041A 0121 B6 45         A         LDA     TEMPA
00042A 0123 B2 41         A         SBC     DVSOR
00043A 0125 24 0D      0134         BCC     DIV165    Divisor still OK
00044A 0127 B7 45         A         STA     TEMPA     Divisor too large
00045A 0129 9F                      TXA
00046A 012A BB 42         A         ADD     DVSOR+1
00047A 012C 97                      TAX
00048A 012D B6 45         A         LDA     TEMPA
00049A 012F B9 41         A         ADC     DVSOR
00050A 0131 98                      CLC
00051A 0132 20 01      0135         BRA     DIV167
00052A 0134 99            DIV165    SEC
00053A 0135 39 44         A DIV167  ROL     DVDND+1
00054A 0137 39 43         A         ROL     DVDND
00055A 0139 34 41         A         LSR     DVSOR     Adjust Divisor
00056A 013B 36 42         A         ROR     DVSOR+1
00057A 013D 3A 40         A         DEC     COUNT
00058A 013F 26 DA      011B         BNE     DIV163
00059A 0141 81                      RTS
00060                     *
00061                     *
00062                     ********************************************************************************
00063                               END
```

Figure 3-15. 16-bit ÷ 16-Bit With 16-Bit Result Subroutine Example

3.9 ASSIST05 DEBUG MONITOR

Debug monitor ASSIST05 is a monitor which is intended for use with the MC146805E2 Microprocessor Unit (MPU). The ASSIST05 monitor uses an RS-232 interface to allow users to quickly perform hardware and software development and evaluation. Figure 3-16 contains a schematic diagram of one possible circuit that could be used to implement ASSIST05. The program listing for ASSIST05 is provided in Figure 3-17.

The serial interface shown in Figure 3-16 is provided by an MC6850 ACIA. However, this serial hardware, and the CHRIN and CHROUT subroutines of ASSIST05, could be replaced by hardware shown in Figure 3-4 and the GETC and PUTC subroutines of Figure 3-5. All M6805 HMOS/M146805 CMOS Family MCU evaluation devices include debug monitors which can be used with an RS-232 interface as discussed in the Serial I/O Software For RS-232 paragraph. If, in the case of an MC146805E2 MPU, a debug monitor that does not require an RS-232 interface is desired, Motorola Application Note AN-823 or AN-823A can be used. This application note describes a debug monitor for the MC146805E2 which uses a keypad and LCD for the user interface.

The ASSIST05 program includes commands which allow memory and register examine/change, breakpoint set/point/display, single or multiple trace, and tape punch/load. In the paragraphs which follow, each of the commands is described in greater detail, and some of the routines in ASSIST05, which might be useful in other programs, are also discussed.

3.9.1 ASSIST05 Command Description

The ASSIST05 program is initialized by either a power-on or manual reset to the MC146805E2. After a reset, "ASSIST05 1.0" is printed and the prompt character ">" is displayed to indicate that commands may be entered.

Table 3-2 summarizes the commands which may be entered. Commands are entered by typing the command, as shown in Table 3-2, followed by a carriage return.

Table 3-2. ASSIST05 Valid Display Commands

Command	Usage
R	Display all Register Contents
A	Display/Change User Accumulator Contents
X	Display/Change User Index Register Contents
C	Display/Change User Condition Code Register Contents
P XXXX	Change User Program Counter Contents
W XXXX YYYY	Write Memory to Tape
B	Display Breakpoints
B N XXXX	Set Breakpoint #N
B N O	Clear Breakpoint #N
T	Trace One Instruction
T XXXX	Trace XXXX Instruction
M XXXX	Display/Change Memory
G	Continue Program Executive at Current Program Counter
GXXXX	Execute Program at Address XXXX

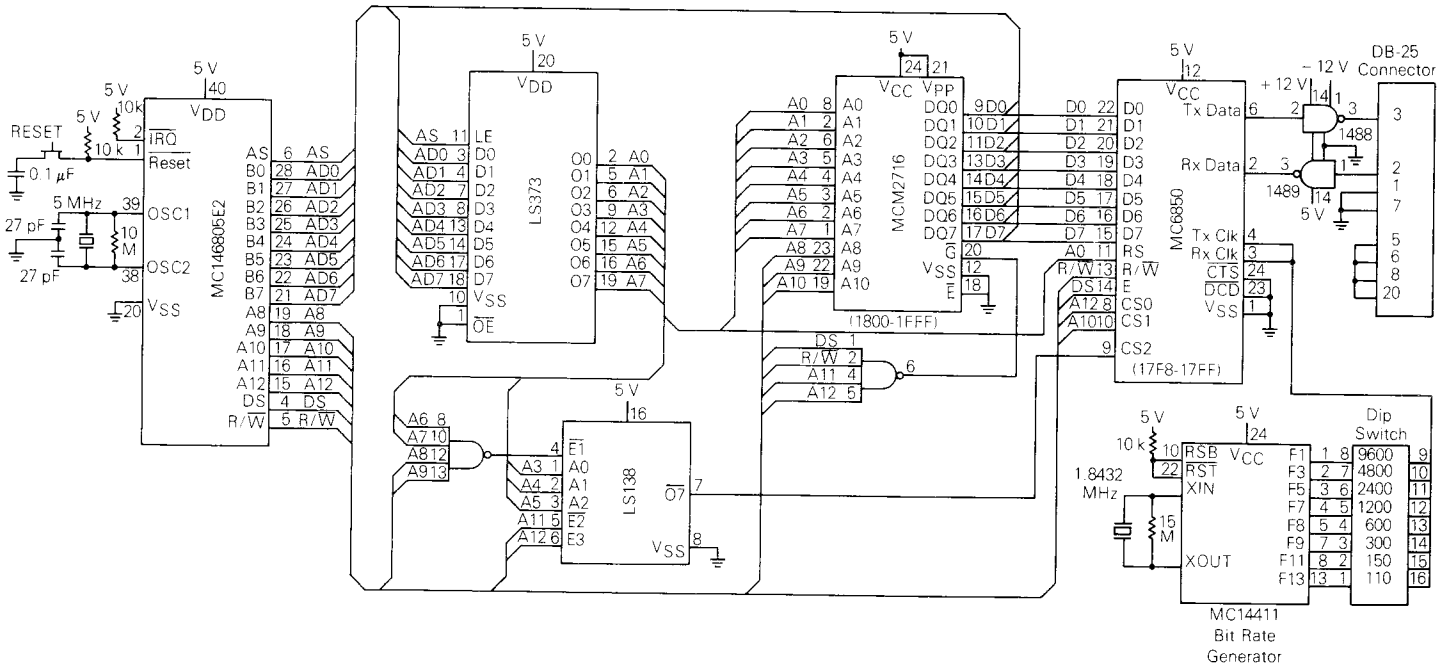


Figure 3-16. ASSIST05 Interface Schematic Diagram

3.9.2 Detailed Command Description

Register Examine/Change
The current user register contents may be displayed all at once or individually (except SP). The SP may not be directly modified by the user. The PC may be modified with either P or G commands.

R — Display Registers
Current user registers are displayed in the following format: PC A X C SP. After the registers have been displayed, the prompt character is returned.

A — Display/Change the Accumulator
This command begins by printing the current contents of the accumulator in hexadecimal. The user may then enter a new value in hexadecimal or a carriage return to terminate the command.

X — Display/Change the Index Register
This procedure is the same as the A command, but affects the index register instead.

C — Display/Change the Condition Code Register
This procedure is the same as the A command, but affects the condition code register instead.

Tape Punch/Load
This allows the user to load a tape, via the RS-232 interface, in the Motorola S1-S9 format. Memory is then loaded with the data which is contained in the file on the tape. Files in S1-S9 format have the destination addresses contained within the file. The format of this command is: W XXXX YYYY. The memory contents of addresses XXXX to YYYY are output to the RS-232 port. Data is then stored onto tape in the Motorola S1-S9 format.

Breakpoints
Up to three breakpoints may be used to allow debugging of user programs. The program execution can be halted at specified addresses so that the current user registers and memory may be examined and evaluated. Whenever program execution reaches a breakpoint address, program execution ceases, the current user registers are displayed, and the prompt character is returned. Following this, the applicable breakpoint command can then be entered. Breakpoints may only be entered from valid RAM addresses. The current program counter is not displayed; however, it may be examined by using the R command.

P — Display the Program Counter
This procedure is similar to the A command, but affects the program counter instead. Note that this is a two-byte register.

NOTE

When the user program execution is initiated via the G command, instructions up to and including the instruction at the breakpoint are executed. If the user program execution is halted with a reset, all enabled breakpoint address locations will contain $83 and should be reloaded.

B — Display Breakpoints
This command allows all breakpoint addresses to be displayed and the prompt character is returned.

B N XXXX — Set Breakpoint #N
This command enables breakpoint N, where N is a number 0-2 at address XXXX, and where XXXX is the address of the last instruction to be executed before returning to ASSIST05.

B N 0 — Clear Breakpoint #N
This command disables breakpoint N, where N is a number 0-2.

Instruction Trace
This command is used to execute one or more instructions, and is generally used after a breakpoint is reached. Tracing may also be used to step through ROM-based programs; however, unlike breakpoints, tracing is not done in real-time. To use the trace command on ROM-based programs, the user must put a jump-to-the ROM entry address in RAM. The user then sets a breakpoint at the jump instruction address. Once the breakpoint address is encountered, the jump is executed and control is returned to ASSIST05. The current user PC then points to the ROM entry address and tracing may then be used.

T — Trace One Instruction
With this command, a single instruction, located at the user PC, is executed and the registers are then displayed. Control is then returned to ASSIST05.

T XXXX — Trace XXXX Instructions
With this command, XXXX instructions are executed, beginning at the current user PC. After the specified number of instructions are executed, the registers are displayed and control is returned to ASSIST05. The instructions executed during the trace instruction are not executed in real-time. The P instruction may be used prior to tracing to point to the first instruction to be executed.

Memory Examine/Change
This command allows memory, at the specified address XXXX, to be examined. Then, if desired, the contents of that location may be changed, or the previous location or next location may be examined.

M XXXX — Display/Change Memory
With this command, memory locations XXXX, XXXX – 1, or XXXX + 1 may be acted upon. To do this or to return to ASSIST05, one of four terminal keys need be depressed. These include:

↑—to examine previous location (XXXX – 1)
LF—to examine next location (XXXX + 1)
HH—to change contents of specified location XXXX
CR—to exit memory examine/change command and return to ASSIST05.

Execute User Program

Two execute commands are used to allow real-time execution of the user program. Execution can continue either from the current user PC or begin at a specified address.

G — Continue Program Execution at Current PC

This command allows the user program to continue execution from the current user PC. This command is usually used after a breakpoint has been executed or if the user has previously altered the PC with the P instruction.

G XXXX — Execute Program at Address XXXX

This command results in the user PC being loaded with address XXXX. The user program then starts execution from the new (current) user PC.

3.9.3 ASSIST05 Routines

The ASSIST05 program contains many useful routines and subroutines which might be used in other programs. Some of the more unusual includes a routine that can find the current SP (LOCSTK) and the trace routine which allows ROM-based code to be debugged. For more information consult the complete listing which is in Figure 3-17.

```
PAGE  001  ASSIST05.SA:1  ASSIST

00001                              NAM    ASSIST05



00003                    ***************************************************
00004                    *    MONITOR FOR THE AUSTIN 6805 EVALUATION MODULE*
00005                    *          (C) COPYRIGHT 1979 MOTOROLA INC.       *
00006                    ***************************************************


00008                    ***************************************************
00009                    *
00010                    *      THE MONITOR HAS THE FOLLOWING COMMANDS:
00011                    *
00012                    *      R              --  PRINT REGISTERS
00013                    *
00014                    *      A              --  DISPLAY/CHANGE A REGISTER
00015                    *
00016                    *      X              --  DISPLAY/CHANGE X REGISTER
00017                    *
00018                    *      C              --  DISPLAY/CHANGE CONDITION CODE
00019                    *
00020                    *      P              --  DISPLAY/CHANGE PROGRAM COUNTER
00021                    *
00022                    *      L              --  LOAD TAPE FILE INTO MEMORY
00023                    *
00024                    *      W XXXX YYYY --  WRITE MEMORY TO TAPE FILE
00025                    *
00026                    *      B              --  DISPLAY BREAKPOINTS
00027                    *      B N XXXX       --  SET BREAKPOINT NUMBER N
00028                    *      B N 0          --  CLEAR BREAKPOINT NUMBER N
00029                    *
00030                    *      T              --  TRACE ONE INSTRUCTION
00031                    *      T XXXX         --  TRACE XXX INSTRUCTIONS
00032                    *
00033                    *      M XXXX         --  MEMORY EXAMINE/CHANGE.
00034                    *            TYPE:     ↑ -- TO EXAMINE PREVIOUS
00035                    *                     LF -- TO EXAMINE NEXT
00036                    *                     HH -- CHANGE TO HEX DATA
00037                    *                     CR -- TERMINATE COMMAND
00038                    *
00039                    *      G              --  CONTINUE PROGRAM EXECUTION FROM
00040                    *                         CURRENT PROGRAM COUNTER.
00041                    *      G XXXX         --  GO EXECUTE PROGRAM AT SPECIFIED
00042                    *                         ADDRESS.
00043                    *
00044                    ***************************************************


00046                    ***************************************************
00047                    *      MC146805E2 GLOBAL PARAMETERS                *
00048                    ***************************************************
00049  1800      A MONSTR EQU    $1800     START OF MONITOR
```

Figure 3-17. ASSIST05 Program Listing

```
PAGE  002  ASSIST05.SA:1  ASSIST

00050          001F     A PCMASK EQU    $1F       MASK OFF FOR 8K ADDRESS SPACE (E2)
00051          0003     A NUMBKP EQU    3         NUMBER OF BREAKPOINTS
00052          17F8     A ACIA   EQU    $17F8     ACIA ADDRESS
00053          003E     A PROMPT EQU    '>        PROMPT CHARACTER
00054          0008     A TIMER  EQU    8         TIMER DATA REGISTER
00055          0009     A TIMEC  EQU    9         TIMER CONTROL REGISTER



00057                     ************************************************
00058                     *       EQUATES                                *
00059                     ************************************************
00060          0004     A EOT    EQU    $04       END OF TEXT
00061          000D     A CR     EQU    $0D       CARRIAGE RETURN
00062          000A     A LF     EQU    $0A       LINE FEED
00063          0011     A DC1    EQU    $11       READER ON CONTROL FUNCTION
00064          0012     A DC2    EQU    $12       PUNCH ON CONTROL FUNCTION
00065          0013     A DC3    EQU    $13       X-OFF CONTROL FUNCTION
00066          0014     A DC4    EQU    $14       STOP CONTROL FUNCTION
00067          0020     A SP     EQU    $20       SPACE
00068          0007     A BELL   EQU    $07       CONTROL-G (BELL)
00069          0083     A SWIOP  EQU    $83       SOFTWARE INTERRUPT OPCODE
00070          00CC     A JMPOP  EQU    $CC       EXTENDED JUMP OPCODE



00072                     ************************************************
00073                     * MONITOR WORK AREA AT STACK BOTTOM
00074                     ************************************************
00075A 0041                        ORG    $41       BOTTOM OF STACK
00076          0038     A BKPTBL EQU    *-3*NUMBKP BKPT TABLE UNDER STACK BOTTOM
00077A 0041    0001     A SWIFLG RMB    1         SWI FUNCTION FLAG
00078A 0042    0001     A WORK1  RMB    1         CHRIN/LOAD/STORE/PUTBYT
00079A 0043    0001     A WORK2  RMB    1         LOAD/STORE/PUTBYT
00080A 0044    0001     A ADDRH  RMB    1         HIGH ADDRESS BYTE
00081A 0045    0001     A ADDRL  RMB    1         LOW ADDRESS BYTE
00082A 0046    0001     A WORK3  RMB    1         LOAD/STORE/PUNCH
00083A 0047    0001     A WORK4  RMB    1         STORE/PUNCH
00084A 0048    0001     A WORK5  RMB    1         TRACE
00085A 0049    0001     A WORK6  RMB    1         TRACE
00086A 004A    0001     A WORK7  RMB    1         TRACE
00087A 004B    0001     A PNCNT  RMB    1         PUNCH BREAKPOINT
00088A 004C    0002     A PNRCNT RMB    2         PUNCH
00089A 004E    0001     A CHKSUM RMB    1         PUNCH
00090A 004F    000C     A VECRAM RMB    12        VECTORS



00092A 1800                        ORG    MONSTR    START OF MONITOR
```

Figure 3-17. ASSIST05 Program Listing (Continued)

```
PAGE  003  ASSIST05.SA:1  ASSIST

00094                          **************************************************
00095                          * MONITOR BASE STRING/TABLE PAGE
00096                          * (MUST BE AT THE BEGINNING OF A PAGE)
00097                          **************************************************
00098          1800      A MBASE  EQU     *         START OF WORK PAGE IN ROM
00099                          * MSGUP MUST BE FIRST IN PAGE
00100A 1800    41        A MSGUP  FCC     /ASSIST05 1.1/FIREUP MESSAGE
00101A 180C    04        A MSGNUL FCB     EOT       END OF STRING
00102A 180D    3F        A MSGERR FCC     /? ERROR ?/
00103A 1816    04        A        FCB     EOT
00104A 1817    53        A MSGS1  FCB     'S,'1,EOT S1 START RECORD TEXT
00105A 181A    53        A MSGS9  FCC     /S9030000FC/
00106A 1824    0D        A        FCB     CR        S9 RECORD TEXT
00107A 1825    14        A MSGMOF FCB     DC4,DC3,EOT MOTORS OFF TEXT
00108A 1828    49        A MSGWAS FCC     /IS OPCODE/
00109A 1831    04        A        FCB     EOT
00110A 1832    CC        A VECTAB FCB     JMPOP
00111A 1833    1C85      A        FDB     TIRQ
00112A 1835    CC        A        FCB     JMPOP
00113A 1836    1C85      A        FDB     TIRQ
00114A 1838    CC        A        FCB     JMPOP
00115A 1839    1CA1      A        FDB     IRQ
00116A 183B    CC        A        FCB     JMPOP
00117A 183C    1A94      A        FDB     SWI


00119                          **************************************************
00120                          *       GO --- START EXECUTION
00121                          **************************************************
00122A 183E CD 19A9       A CMDG   JSR     GETADR    OBTAIN INPUT ADDRESS
00123A 1841 24 14      1857        BCC     NEXT      DO CONTINUE IF NONE
00124A 1843 B6 44         A        LDA     ADDRH     CHECK ADDRESS BOUNDRIES
00125A 1845 A1 20         A        CMP     #$20      FOR OVERRUN
00126A 1847 25 03      184C        BLO     GADDR
00127A 1849 CC 1A74       A        JMP     CMDERR    ERROR IF $2000 OR LARGER
00128A 184C CD 1B23       A GADDR  JSR     LOCSTK    OBTAIN CURRENT STACK ADDRESS-3
00129A 184F B6 44         A        LDA     ADDRH     LOAD PC HIGH
00130A 1851 E7 07         A        STA     7,X       INTO STACK
00131A 1853 B6 45         A        LDA     ADDRL     LOAD PC LOW
00132A 1855 E7 08         A        STA     8,X       INTO STACK
00133A 1857 0E 47 03 185D NEXT     BRSET   7,WORK4,CONT
00134A 185A CC 1A26       A        JMP     CMD
00135A 185D CD 1CC3       A CONT   JSR     SCNBKP    INIT BREAKPOINT SCAN PARMS
00136A 1860 F6                     GOINSB  LDA     ,X        LOAD HI BYTE
00137A 1861 2B 10      1873        BMI     GONOB     BRA EMPTY
00138A 1863 B7 44         A        STA     ADDRH     STORE HI ADDRESS
00139A 1865 E6 01         A        LDA     1,X       LOAD LOW
00140A 1867 B7 45         A        STA     ADDRL     STORE LOW
00141A 1869 CD 1943       A        JSR     LOAD      LOAD OPCODE
00142A 186C E7 02         A        STA     2,X       STORE INTO TABLE
00143A 186E A6 83         A        LDA     #SWIOP    REPLACE WITH OPCODE
00144A 1870 CD 1952       A        JSR     STORE     STORE IN PLACE
00145A 1873 5C                     GONOB   INCX              TO
00146A 1874 5C                             INCX              NEXT
00147A 1875 5C                             INCX              BREAKPOINT
00148A 1876 3A 4B         A        DEC     PNCNT     COUNT DOWN
00149A 1878 26 E6      1860        BNE     GOINSB    LOOP IF MORE
```

Figure 3-17. ASSIST05 Program Listing (Continued)

```
00150A 187A 33 41        A          COM     SWIFLG   FLAG BREAKPOINTS ARE IN
00151                               ****RESET USERS TIMER ENVIRONMENT******
00152A 187C 80                            RTI              RESTART PROGRAM


00154                               ******************************************
00155                               *  CLBYTE - LOAD SUBROUTINE TO READ NEXT  *
00156                               *           BYTE, ADJUST CHECKSUM,        *
00157                               *           DECREMENT COUNT.              *
00158                               * OUTPUT: A=BYTE                         *
00159                               *         CC=REFLECTS COUNT DECREMENT     *
00160                               ******************************************
00161A 187D AD 0C     188B CLBYTE BSR     GETBYT   OBTAIN NEXT BYTE
00162A 187F 24 57     18D8        BCC     CMDMIN   ERROR IF NONE
00163A 1881 B7 43        A        STA     WORK2    SAVE VALUE
00164A 1883 BB 4E        A        ADD     CHKSUM   ADD TO CHECKSUM
00165A 1885 B7 4E        A        STA     CHKSUM   REPLACE
00166A 1887 B6 43        A        LDA     WORK2    RELOAD BYTE VALUE
00167A 1889 5A                    DECX             COUNT DOWN
00168A 188A 81                    RTS              RETURN TO CALLER


00170                               ******************************************
00171                               *  GETBYT - READ BYTE IN HEX SUBROUTINE   *
00172                               * OUTPUT: C=0, Z=1 NO NUMBER              *
00173                               *         C=0, Z=0 INVALID NUMBER         *
00174                               *         C=1, Z=1, A=BINARY BYTE VALUE   *
00175                               ******************************************
00176A 188B CD 198E      A GETBYT JSR     GETNYB   GET HEX DIGIT
00177A 188E 24 0E     189E        BCC     GETBRZ   RETURN NO NUMBER
00178A 1890 48           GETBY2 ASLA              SHIFT
00179A 1891 48                  ASLA              OVER
00180A 1892 48                  ASLA              BY
00181A 1893 48                  ASLA              FOUR
00182A 1894 B7 43        A      STA     WORK2    SAVE HIGH HEX DIGIT
00183A 1896 CD 198E      A      JSR     GETNYB   GET LOW HEX
00184A 1899 4D                  TSTA             FORCE Z=0 (DELIMITER IF INVALID)
00185A 189A 24 04     18A0      BCC     GETBRT   RETURN IF INVALID NUMBER
00186A 189C BA 43        A      ORA     WORK2    COMBINE HEX DIGITS
00187A 189E 3F 43        A GETBRZ CLR     WORK2    SET Z=1
00188A 18A0 81           GETBRT RTS               RETURN TO CALLER


00190                               ******************************************
00191                               *  L -- LOAD FILE INTO MEMORY COMMAND     *
00192                               ******************************************
00193A 18A1 CD 19D0      A CMDL   JSR     CHRIN    READ CARRIAGE RETURN
00194A 18A4 A6 11        A        LDA     #DC1     TURN ON READER
00195A 18A6 CD 19EA      A        JSR     CHROUT   WITH DC1 CONTROL CODE
00196                             * SEARCH FOR AN 'S'
00197A 18A9 CD 19D0      A CMDLT  JSR     CHRIN    READ A CHARACTER
00198A 18AC A1 53        A CMDLSS CMP     #'S      ? 'S'
00199A 18AE 26 F9     18A9        BNE     CMDLT    LOOP IF NOT
00200A 18B0 CD 19D0      A        JSR     CHRIN    READ SECOND CHARACTER
00201A 18B3 A1 39        A        CMP     #'9      ? 'S9' RECORD
```

Figure 3-17. ASSIST05 Program Listing (Continued)

```
00202A 18B5 27 24    18DB          BEQ     CLEOF     BRANCH END OF FILE
00203A 18B7 A1 31       A          CMP     #'1       ? 'S1' RECORD
00204A 18B9 26 F1    18AC          BNE     CMDLSS    NO, TRY 'S' AGAIN
00205                           * READ ADDRESS AND COUNT
00206A 18BB 3F 4E       A          CLR     CHKSUM    ZERO CHKSUM
00207A 18BD AD BE    187D          BSR     CLBYTE    OBTAIN SIZE OF RECORD
00208A 18BF 97                     TAX               START COUNTDOWN IN X REGISTER
00209A 18C0 AD BB    187D          BSR     CLBYTE    OBTAIN START OF ADDRESS
00210A 18C2 B7 44       A          STA     ADDRH     STORE IT
00211A 18C4 AD B7    187D          BSR     CLBYTE    OBTAIN LOW ADDRESS
00212A 18C6 B7 45       A          STA     ADDRL     STORE IT
00213                           * NOW LOAD TEXT
00214A 18C8 AD B3    187D CLLOAD   BSR     CLBYTE    NEXT CHARACTER
00215A 18CA 27 08    18D4          BEQ     CLEOR     BRANCH IF COUNT DONE
00216A 18CC CD 1952     A          JSR     STORE     STORE CHARACTER
00217A 18CF CD 1962     A          JSR     PTRUP1    UP ADDRESS POINTER
00218A 18D2 20 F4    18C8          BRA     CLLOAD    LOOP UNTIL COUNT DEPLETED
00219                           * END OF RECORD
00220A 18D4 3C 4E       A CLEOR    INC     CHKSUM    TEST VALID CHECKSUM
00221A 18D6 27 C9    18A1          BEQ     CMDL      CONTINUE IF SO
00222A 18D8 CC 1A74     A CMDMIN   JMP     CMDERR    ERROR IF INVALID
00223                           * END OF FILE
00224A 18DB AD A0    187D CLEOF    BSR     CLBYTE    READ S9 LENGTH
00225A 18DD 97                     TAX               PREPARE S9 FLUSH COUNT
00226A 18DE AD 9D    187D CLEOFL   BSR     CLBYTE    SKIP HEX PAIR
00227A 18E0 26 FC    18DE          BNE     CLEOFL    BRANCH MORE
00228A 18E2 AE 25       A          LDX     #MSGMOF-MBASE TURN MOTORS OUT
00229A 18E4 CC 1A23     A          JMP     CMDPDT    SEND AND END COMMAND


00231                           ******************************************
00232                           *  M -- EXAMINE/CHANGE MEMORY            *
00233                           *  MCHNGE -- REGISTER CHANGE ENTRY POINT  *
00234                           ******************************************
00235A 18E7 CD 19A9     A CMDM     JSR     GETADR    OBTAIN ADDRESS VALUE
00236A 18EA 24 EC    18D8          BCC     CMDMIN    INVALID IF NO ADDRESS
00237A 18EC B6 44       A          LDA     ADDRH     CHECK ADDRESS
00238A 18EE A1 20       A          CMP     #$20      FOR OVERRUN
00239A 18F0 25 03    18F5          BLO     CMDMLP
00240A 18F2 CC 1A74     A          JMP     CMDERR    ERROR IF $2000 OF LARGER
00241A 18F5 CD 1B16     A CMDMLP   JSR     PRTADR    PRINT OUT ADDRESS AND SPACE
00242A 18F8 AD 49    1943 MCHNGE   BSR     LOAD      LOAD BYTE INTO A REGISTER
00243A 18FA CD 1B1D     A          JSR     CRBYTS    PRINT WITH SPACE
00244A 18FD CD 198E     A          JSR     GETNYB    SEE IF CHANGE WANTED
00245A 1900 24 0D    190F          BCC     CMDMDL    BRANCH NO
00246A 1902 AD 8C    1890          BSR     GETBY2    OBTAIN FULL BYTE
00247A 1904 26 D2    18D8          BNE     CMDMIN    TERMINATE IF INVALID HEX
00248A 1906 24 07    190F          BCC     CMDMDL    BRANCH IF OTHER DELIMITER
00249A 1908 AD 48    1952          BSR     STORE     STORE NEW VALUE
00250A 190A 25 CC    18D8          BCS     CMDMIN    BRANCH IF STORE FAILS
00251A 190C CD 19D0     A          JSR     CHRIN     OBTAIN DELIMITER
00252                           * CHECK OUT DELIMITERS
00253A 190F A1 0A       A CMDMDL   CMP     #LF       ? TO NEXT BYTE
00254A 1911 27 1D    1930          BEQ     CMDMLF    BRANCH IF SO
00255A 1913 A1 5E       A          CMP     #'↑       ? TO PREVIOUS BYTE
00256A 1915 27 03    191A          BEQ     CMDMBK    BRANCH YES
00257A 1917 CC 1A29     A          JMP     CMDNNL    ENTER COMMAND HANDLER
```

Figure 3-17. ASSIST05 Program Listing (Continued)

```
PAGE  006  ASSIST05.SA:1  ASSIST

00258A 191A 3D 45       A CMDMBK TST     ADDRL    ? LOW BYTE ZERO
00259A 191C 26 0C    192A        BNE     CMDMB2   NO, JUST DOWN IT
00260A 191E 3A 44       A        DEC     ADDRH    DOWN HIGH FOR CARRY
00261A 1920 B6 44       A        LDA     ADDRH    CHECK ADDRESS
00262A 1922 A1 FF       A        CMP     #$FF     FOR UNDERFLOW
00263A 1924 26 04    192A        BNE     CMDMB2
00264A 1926 A6 1F       A        LDA     #$1F     CLEAR ADDRESS ON UNDERFLOW
00265A 1928 B7 44       A        STA     ADDRH
00266A 192A 3A 45       A CMDMB2 DEC     ADDRL    DOWN LOW BYTE
00267A 192C AD 51    197F        BSR     PCRLF    TO NEXT LINE
00268A 192E 20 C5    18F5        BRA     CMDMLP   TO NEXT BYTE
00269A 1930 A6 0D       A CMDMLF LDA     #CR      SEND JUST CARRIAGE RETURN
00270A 1932 CD 19F2     A        JSR     CHROU2   OUTPUT IT
00271A 1935 AD 2B    1962        BSR     PTRUP1   UP POINTER BY ONE
00272A 1937 B6 44       A        LDA     ADDRH    CHECK ADDRESS
00273A 1939 A1 20       A        CMP     #$20     FOR OVERRUN
00274A 193B 25 B8    18F5        BLO     CMDMLP
00275A 193D 3F 44       A        CLR     ADDRH    IF LARGER CLEAR
00276A 193F 3F 45       A        CLR     ADDRL    ADDRESS
00277A 1941 20 B2    18F5        BRA     CMDMLP   TO NEXT BYTE



00279                            ********************************************
00280                            *       LOAD - LOAD INTO A FROM ADDRESS IN   *
00281                            *              POINTER ADDRH/ADDRL           *
00282                            * INPUT: ADDRH/ADDRL=ADDRESS                 *
00283                            * OUTPUT: A=BYTE FROM POINTED LOCATION       *
00284                            * X IS TRANSPARENT                           *
00285                            * WORK1,WORK2,WORK3 USED                     *
00286                            ********************************************
00287A 1943 BF 42       A LOAD   STX     WORK1    SAVE X
00288A 1945 AE C6       A        LDX     #$C6     C6=LDA 2-BYTE EXTENDED

00290A 1947 BF 43       A LDSTCM STX     WORK2    PUT OPCODE IN PLACE
00291A 1949 AE 81       A        LDX     #$81     81=RTS
00292A 194B BF 46       A        STX     WORK3    NOW THE RETURN
00293A 194D BD 43       A        JSR     WORK2    EXECUTE BUILT ROUTINE
00294A 194F BE 42       A        LDX     WORK1    RESTORE X
00295A 1951 81                   RTS              AND EXIT



00297                            ********************************************
00298                            *       STORE - STORE A AT ADDRESS IN POINTER*
00299                            *                ADDRH/ADDRL                 *
00300                            * INPUT: A=BYTE TO STORE                     *
00301                            *        ADDRH/ADDRL=ADDRESS                 *
00302                            * OUTPUT: C=0 STORE WENT OK                  *
00303                            *         C=1 STORE DID NOT TAKE (NOT RAM)   *
00304                            * REGISTERS TRANSPARENT                      *
00305                            * (A NOT TRANSPARENT ON INVALID STORE)       *
00306                            * WORK1,WORK2,WORK3,WORK4 USED               *
00307                            ********************************************
00308A 1952 BF 42       A STORE  STX     WORK1    SAVE X
00309A 1954 AE C7       A        LDX     #$C7     C7=STA 2-EXTENDED
```

Figure 3-17. ASSIST05 Program Listing (Continued)

```
00310A 1956 AD EF    1947          BSR     LDSTCM    CALL STORE ROUTINE
00311A 1958 B7 47       A          STA     WORK4     SAVE VALUE STORED
00312A 195A AD E7    1943          BSR     LOAD      ATTEMPT LOAD
00313A 195C B1 47       A          CMP     WORK4     ? VALID STORE
00314A 195E 27 01    1961          BEQ     STRTS     BRANCH IF VALID
00315A 1960 99                     SEC               SHOW INVALID STORE
00316A 1961 81               STRTS RTS               RETURN



00318                              ****************************************
00319                              *       PTRUP1 - INCREMENT MEMORY POINTER  *
00320                              ****************************************
00321A 1962 3C 45       A PTRUP1 INC     ADDRL     INCREMENT LOW BYTE
00322A 1964 26 02    1968          BNE     PRTRTS    NON-ZERO MEANS  NO CARRY
00323A 1966 3C 44       A          INC     ADDRH     INCREMENT HIGH BYTE
00324A 1968 81               PRTRTS RTS               RETURN TO CALLER



00326                              ****************************************
00327                              *       PUTBYT --- PRINT A IN HEX          *
00328                              *  X TRANSPARENT                           *
00329                              *  WORK1 USED                              *
00330                              ****************************************
00331A 1969 B7 42       A PUTBYT STA     WORK1     SAVE A
00332A 196B 44                     LSRA              SHIFT TO
00333A 196C 44                     LSRA              LEFT HEX
00334A 196D 44                     LSRA              DIGIT
00335A 196E 44                     LSRA              SHIFT HIGH NYBBLE DOWN
00336A 196F AD 02    1973          BSR     PUTNYB    PRINT IT
00337A 1971 B6 42       A          LDA     WORK1
00338                              * FALL INTO PUTNYB



00340                              ************************************************
00341                              *       PUTNYB --- PRINT LOWER NYBBLE OF A IN HEX*
00342                              *  A,X TRANSPARENT                               *
00343                              ************************************************
00344A 1973 A4 0F       A PUTNYB AND     #$F       MASK OFF HIGH NYBBLE
00345A 1975 AB 30       A          ADD     #'0       ADD ASCII ZERO
00346A 1977 A1 39       A          CMP     #'9       CHECK FOR A-F
00347A 1979 23 6F    19EA          BLS     CHROUT    OK, SEND OUT
00348A 197B AB 07       A          ADD     #'A-'9-1  ADJUSTMENT FOR HEX A-F
00349A 197D 20 6B    19EA          BRA     CHROUT    NOW SEND OUT



00351                              ************************************************
```

Figure 3-17. ASSIST05 Program Listing (Continued)

```
PAGE  008  ASSIST05.SA:1  ASSIST

00352                          *  PDATA  - PRINT MONITOR STRING AFTER CR/LF
00353                          *  PDATA1 - PRINT MONITOR STRING
00354                          *  PCRLF  - PRINT CARRIAGE RETURN AND LINE FEED
00355                          * INPUT: X=OFFSET TO STRING IN BASE PAGE (UNUSED FOR PCRL
00356                          ************************************************
00357A 197F AE 0C          A PCRLF  LDX    #MSGNUL-MBASE LOAD NULL STRING ADDRESS
00358A 1981 A6 0D          A PDATA  LDA    #CR       PREPARE CARRIAGE RETURN
00359A 1983 AD 65      19EA PDLOOP BSR    CHROUT    SEND NEXT CHARACTER
00360A 1985 D6 1800        A PDATA1 LDA    MBASE,X   LOAD NEXT CHARACTER
00361A 1988 5C                      INCX             BUMP POINTER UP ONE
00362A 1989 A1 04          A        CMP    #EOT      ? END OF STRING
00363A 198B 26 F6      1983         BNE    PDLOOP    BRANCH NO
00364A 198D 81                      RTS              RETURN DONE


00366                          ****************************************
00367                          * GETNYB - OBTAIN NEXT HEX CHARACTER     *
00368                          * OUTPUT: C=0 NOT HEX INPUT, A=DELIMITER *
00369                          *         C=1 HEX INPUT, A=BINARY VALUE  *
00370                          *  X TRANSPARENT                         *
00371                          *  WORK1 IN USE                          *
00372                          ****************************************
00373A 198E CD 19D0        A GETNYB JSR    CHRIN     OBTAIN CHARACTER
00374A 1991 A1 30          A        CMP    #'0       ? LOWER THAN ZERO
00375A 1993 25 12      19A7         BLO    GETNCH    BRANCH NOT HEX
00376A 1995 A1 39          A        CMP    #'9       ? HIGHER THAN NINE
00377A 1997 23 0A      19A3         BLS    GETNHX    BRANCH IF 0 THRU 9
00378A 1999 A1 41          A        CMPA   #'A       ? LOWER THAN AN "A"
00379A 199B 25 0A      19A7         BLO    GETNCH    BRANCH NOT HEX
00380A 199D A1 46          A        CMPA   #'F       ? HIGHER THAN AN "F"
00381A 199F 22 06      19A7         BHI    GETNCH    BRANCH NOT HEX
00382A 19A1 A0 07          A        SUB    #7        ADJUST TO $A OFFSET
00383A 19A3 A4 0F          A GETNHX AND    #$0F      CLEAR ASCII BITS
00384A 19A5 99                      SEC              SET CARRY
00385A 19A6 81                      RTS              RETURN
00386A 19A7 98                GETNCH CLC             CLEAR CARRY FOR NO HEX
00387A 19A8 81                      RTS              RETURN


00389                          ****************************************
00390                          *     GETADR - BUILD ANY SIZE BINARY     *
00391                          *              NUMBER FROM INPUT.        *
00392                          *              LEADING BLANKS SKIPPED.   *
00393                          * OUTPUT: CC=0 NO NUMBER ENTERED         *
00394                          *         CC=1 ADDRH/ADDRL HAS NUMBER    *
00395                          *         A=DELIMITER                    *
00396                          * A,X VOLATILE                           *
00397                          * WORK1 IN USE
00398                          ****************************************
00399A 19A9 CD 19E8        A GETADR JSR    PUTSP
00400A 19AC 3F 44          A        CLR    ADDRH     CLEAR HIGH BYTE
00401A 19AE AD DE      198E         BSR    GETNYB    OBTAIN FIRST HEX VALUE
00402A 19B0 25 06      19B8         BCS    GETGTD    BRANCH IF GOT IT
00403A 19B2 A1 20          A        CMP    #'        ? SPACE
00404A 19B4 27 F3      19A9         BEQ    GETADR    LOOP IF SO
00405A 19B6 98                      CLC              RETURN NO NUMBER
```

Figure 3-17. ASSIST05 Program Listing (Continued)

```
00406A 19B7 81                      RTS             RETURN
00407A 19B8 B7 45       A GETGTD    STA     ADDRL   INITIALIZE LOW VALUE
00408A 19BA AD D2    198E GETALP    BSR     GETNYB  OBTAIN NEXT HEX
00409A 19BC 24 10    19CE           BCC     GETARG  BRANCH IF NONE
00410A 19BE 48                      ASLA            OVER
00411A 19BF 48                      ASLA            FOUR
00412A 19C0 48                      ASLA            BITS
00413A 19C1 48                      ASLA            FOR SHIFT
00414A 19C2 AE 04       A           LDX     #4      LOOP FOUR TIMES
00415A 19C4 48                GETASF ASLA            SHIFT NEXT BIT
00416A 19C5 39 45       A           ROL     ADDRL   INTO LOW BYTE
00417A 19C7 39 44       A           ROL     ADDRH   INTO HIGH BYTE
00418A 19C9 5A                      DECX            COUNT DOWN
00419A 19CA 26 F8    19C4          BNE     GETASF  LOOP UNTIL DONE
00420A 19CC 20 EC    19BA          BRA     GETALP  NOW DO NEXT HEX
00421A 19CE 99                GETARG SEC            SHOW NUMBER OBTAINED
00422A 19CF 81                      RTS             RETURN TO CALLER


00424                       ****************************************
00425                       * CHRIN  -  OBTAIN NEXT INPUT CHARACTER  *
00426                       * OUTPUT: A=CHARACTER RECIEVED           *
00427                       * X IS TRANSPARENT                       *
00428                       * NULLS AND RUBOUTS IGNORED              *
00429                       * ALL CHARACTERS ECHOED OUT              *
00430                       * WORK1 USED                             *
00431                       ****************************************
00432A 19D0 C6 17F8     A CHRIN   LDA     ACIA    LOAD STATUS REGISTER
00433A 19D3 44                    LSRA            CHECK FOR INPUT
00434A 19D4 24 FA    19D0         BCC     CHRIN   LOOP UNTIL SOME
00435A 19D6 C6 17F9     A         LDA     ACIA+1  LOAD CHARACTER
00436A 19D9 A4 7F       A         AND     #$7F    AND OFF PARITY
00437A 19DB 27 F3    19D0         BEQ     CHRIN   IGNORE NULLS
00438A 19DD A1 7F       A         CMP     #$7F    ? DEL
00439A 19DF 27 EF    19D0         BEQ     CHRIN   IGNORE DELETES
00440A 19E1 B7 42       A         STA     WORK1   SAVE CHARACTER
00441A 19E3 AD 05    19EA         BSR     CHROUT  ECHO CHARACTER
00442A 19E5 B6 42       A         LDA     WORK1   RESTORE CHARACTER
00443A 19E7 81                    RTS             RETURN TO CALLER


00445                       ************************************************
00446                       *      PUTS --- PRINT A BLANK (SPACE)          *
00447                       *  X UNCHANGED                                 *
00448                       ************************************************
00449A 19E8 A6 20       A PUTSP   LDA     #SP     LOAD SPACE
00450                       * FALL INTO CHROUT


00452                       ****************************************
00453                       * CHROUT  -  SEND CHARACTER TO TERMINAL. *
00454                       *            A CARRIAGE RETURN HAS AN    *
00455                       *            ADDED LINE FEED.            *
```

Figure 3-17. ASSIST05 Program Listing (Continued)

```
PAGE  010  ASSIST05.SA:1  ASSIST

00456                          * INPUT: A=ASCII CHARACTER TO SEND       *
00457                          * A NOT TRANSPARENT                      *
00458                          ******************************************
00459A 19EA A1 0D        A CHROUT CMP     #CR       ? CARRIAGE RETURN
00460A 19EC 26 04     19F2        BNE     CHROU2    BRANCH NOT
00461A 19EE AD 02     19F2        BSR     CHROU2    RECURSIVE CALL FOR CR
00462A 19F0 A6 0A        A        LDA     #LF       NOW SEND LINE FEED
00463A 19F2 C7 17F9      A CHROU2 STA     ACIA+1    STORE CHARACTER INTO PIC
00464A 19F5 C6 17F8      A CHROLP LDA     ACIA      LOAD STATUS REGISTER
00465A 19F8 A5 02        A        BIT     #$02      ? READY FOR NEXT
00466A 19FA 27 F9     19F5        BEQ     CHROLP    LOOP UNTIL READY
00467A 19FC 81                    RTS               AND RETURN




00469                          ******************************************
00470                          *       RESET --- POWER ON RESET ROUTINE
00471                          *
00472                          *  INITIALIZE ACIA, PUT OUT STARTUP MESSAGE
00473                          ******************************************
00474A 19FD AE 0B        A RESET  LDX     #11       MOVE VECTOR TABLE
00475A 19FF D6 1832      A RST    LDA     VECTAB,X  TO RAM USING A
00476A 1A02 E7 4F        A        STA     VECRAM,X  BLOCK MOVE ROUTINE
00477A 1A04 5A                    DECX              TO ALLOW CHANGES
00478A 1A05 2A F8     19FF        BPL     RST       ON THE FLY
00479A 1A07 A6 03        A        LDA     #3        RESET ACIA
00480A 1A09 C7 17F8      A        STA     ACIA      TO INITIALIZE
00481A 1A0C A6 51        A        LDA     #$51      8 BITS-NO PARITY-2 STOP BITS
00482A 1A0E C7 17F8      A        STA     ACIA      SETUP ACIA PARAMETERS
00483A 1A11 CD 1CC3      A        JSR     SCNBKP    CLEAR BREAKPOINTS
00484A 1A14 A6 FF        A        LDA     #$FF      TURN HIGH BIT ON
00485A 1A16 F7              REBCLR STA     ,X        SHOW SLOT EMPTY
00486A 1A17 5C                    INCX              TO
00487A 1A18 5C                    INCX              NEXT
00488A 1A19 5C                    INCX              SLOT
00489A 1A1A 3A 4B        A        DEC     PNCNT     COUNT DOWN
00490A 1A1C 26 F8     1A16        BNE     REBCLR    CLEAR NEXT
00491A 1A1E 3F 41        A RESREN CLR     SWIFLG    SETUP MONITOR ENTRANCE VALUE
00492A 1A20 83                    SWI               ENTER MONITOR
00493A 1A21 20 FB     1A1E        BRA     RESREN    REENTER IF "G"



00495                          ******************************************
00496                          * COMMAND HANDLER                         *
00497                          ******************************************
00498A 1A23 CD 1981      A CMDPDT JSR     PDATA     SEND MESSAGE OUT

00500A 1A26 CD 197F      A CMD    JSR     PCRLF     TO NEW LINE
00501A 1A29 A6 3E        A CMDNNL LDA     #PROMPT   READY PROMPT CHR
00502A 1A2B AD BD     19EA        BSR     CHROUT    SEND IT OUT
00503A 1A2D CD 1CA7      A        JSR     REMBKP    REMOVE BREAKPOINTS IF IN
00504A 1A30 AD 9E     19D0        BSR     CHRIN     GET NEXT CHARACTER
00505A 1A32 5F                    CLRX              ZERO FOR SOME COMMANDS
00506A 1A33 A1 43        A        CMPA    #'C       ? DISPLAY/CHANGE C REGISTER
00507A 1A35 27 49     1A80        BEQ     CMDC      BRANCH IF SO
```

Figure 3-17. ASSIST05 Program Listing (Continued)

```
00508A 1A37 A1 58       A          CMPA   #'X      ? DISPLAY/CHANGE X REGISTER
00509A 1A39 27 43    1A7E          BEQ    CMDX     BRANCH IF SO
00510A 1A3B A1 41       A          CMPA   #'A      ? DISPLAY/CHANGE A REGISTER
00511A 1A3D 27 40    1A7F          BEQ    CMDA     BRANCH IF SO
00512A 1A3F A1 52       A          CMP    #'R      ? REGISTER DISPLAY
00513A 1A41 27 35    1A78          BEQ    REGR     BRANCH YES
00514A 1A43 A1 4C       A          CMP    #'L      ? LOAD FILE
00515A 1A45 26 03    1A4A          BNE    NOTL     NOPE
00516A 1A47 CC 18A1     A          JMP    CMDL     BRANCH YES
00517A 1A4A A1 47       A NOTL     CMPA   #'G      ? GO COMMAND
00518A 1A4C 26 05    1A53          BNE    NOTG     BRANCH NOT
00519A 1A4E 1E 47       A          BSET   7,WORK4
00520A 1A50 CC 183E     A ISP      JMP    CMDG     GO TO IT
00521A 1A53 A1 4D       A NOTG     CMP    #'M      ? MEMORY COMMAND
00522A 1A55 26 03    1A5A          BNE    NOTM     BRANCH NOT
00523A 1A57 CC 18E7     A          JMP    CMDM     GO TO MEMORY DISPLAY/CHANGE
00524A 1A5A A1 54       A NOTM     CMP    #'T      ? TRACE
00525A 1A5C 26 03    1A61          BNE    NOTT     ERROR IF NOT
00526A 1A5E CC 1C23     A          JMP    CMDT     GO TO IT
00527A 1A61 A1 57       A NOTT     CMP    #'W      ? WRITE MEMORY
00528A 1A63 26 03    1A68          BNE    NOTW     BRANCH NO
00529A 1A65 CC 1B97     A          JMP    CMDW     GO TO IT
00530A 1A68 A1 42       A NOTW     CMP    #'B      ? BREAKPOINT COMMAND
00531A 1A6A 27 0F    1A7B          BEQ    BPNT     YES
00532A 1A6C A1 50       A          CMP    #'P      ? PC COMMAND
00533A 1A6E 26 04    1A74          BNE    CMDERR
00534A 1A70 1F 47       A          BCLR   7,WORK4
00535A 1A72 20 DC    1A50          BRA    ISP
00536A 1A74 AE 0D       A CMDERR   LDX    #MSGERR-MBASE LOAD ERROR STRING
00537A 1A76 20 AB    1A23 TOCPDT   BRA    CMDPDT   AND SEND IT OUT
00538A 1A78 CC 1AEE     A REGR     JMP    CMDR
00539A 1A7B CC 1B35     A BPNT     JMP    CMDB


00541                        ************************************************
00542                        *  X -- DISPLAY/CHANGE X REGISTER              *
00543                        ************************************************
00544A 1A7E 5C               CMDX   INCX           INCREMENT INDEX
00545                        * FALL THROUGH


00547                        ************************************************
00548                        *  A -- DISPLAY/CHANGE A REGISTER
00549                        ************************************************
00550A 1A7F 5C               CMDA   INCX           INCREMENT INDEX
00551                        * FALL THROUGH


00553                        ************************************************
00554                        *  C -- DISPLAY/CHANGE CONDITION CODE REGISTER  *
00555                        ************************************************
00556A 1A80 CD 19E8      A CMDC   JSR    PUTSP    SPACE BEFORE VALUE
00557A 1A83 BF 42        A        STX    WORK1    SAVE INDEX VALUE
```

Figure 3-17. ASSIST05 Program Listing (Continued)

```
00558A 1A85 CD 1B23      A          JSR     LOCSTK    LOCATE STACK ADDRESS
00559A 1A88 9F                      TXA               STACK-2 TO A
00560A 1A89 BB 42        A          ADD     WORK1     ADD PROPER OFFSET
00561A 1A8B AB 04        A          ADD     #4        MAKE UP FOR ADDRESS RETURN DIFFERE
00562A 1A8D 3F 44        A          CLR     ADDRH     SETUP ZERO HIGH BYTE
00563A 1A8F B7 45        A          STA     ADDRL     AND SET IN LOW
00564A 1A91 CC 18F8      A TOMCHG   JMP     MCHNGE    NOW ENTER MEMORY CHANGE COMMAND


00566                               ****************************************
00567                               *           S W I  HANDLER             *
00568                               * DETERMINE PROCESSING SWIFLG VALUE      *
00569                               ****************************************
00570A 1A94 5F                      SWI     CLRX              DEFAULT TO STARTUP MESSAGE
00571A 1A95 3D 41        A          TST     SWIFLG    IS THIS RESET
00572A 1A97 26 04     1A9D          BNE     SWICHK    IF NOT REMOVE BREAKPOINTS
00573A 1A99 3C 41        A          INC     SWIFLG    SHOW WE ARE NOW INITIALIZED
00574A 1A9B 20 86     1A23          BRA     CMDPDT    TO COMMAND HANDLER
00575A 1A9D CD 1CC3      A SWICHK   JSR     SCNBKP
00576A 1AA0 F6                      SWIREP  LDA     ,X        RESTORE OPCODES
00577A 1AA1 2B 0B     1AAE          BMI     SWINOB
00578A 1AA3 B7 44        A          STA     ADDRH
00579A 1AA5 E6 01        A          LDA     1,X
00580A 1AA7 B7 45        A          STA     ADDRL
00581A 1AA9 E6 02        A          LDA     2,X
00582A 1AAB CD 1952      A          JSR     STORE
00583A 1AAE 5C                      SWINOB  INCX
00584A 1AAF 5C                      INCX
00585A 1AB0 5C                      INCX
00586A 1AB1 3A 4B        A          DEC     PNCNT
00587A 1AB3 26 EB     1AA0          BNE     SWIREP
00588                               * TRACE ONE INSTRUCTION IF PC AT A BREAKPOINT
00589A 1AB5 CD 1B23      A          JSR     LOCSTK    FIND STACK
00590A 1AB8 E6 08        A          LDA     8,X       GET PC AND ADJUST
00591A 1ABA A0 01        A          SUB     #1
00592A 1ABC B7 47        A          STA     WORK4     SAVE PCL
00593A 1ABE E6 07        A          LDA     7,X
00594A 1AC0 A2 00        A          SBC     #0
00595A 1AC2 B7 46        A          STA     WORK3     SAVE PCH
00596A 1AC4 BF 48        A          STX     WORK5     SAVE SP
00597A 1AC6 CD 1CC3      A          JSR     SCNBKP
00598A 1AC9 F6                      SWITRY  LDA     0,X
00599A 1ACA 2B 1B     1AE7          BMI     SWICMP
00600A 1ACC B1 46        A          CMP     WORK3
00601A 1ACE 26 17     1AE7          BNE     SWICMP
00602A 1AD0 E6 01        A          LDA     1,X
00603A 1AD2 B1 47        A          CMP     WORK4
00604A 1AD4 26 11     1AE7          BNE     SWICMP
00605A 1AD6 BE 48        A          LDX     WORK5
00606A 1AD8 E7 08        A          STA     8,X
00607A 1ADA B6 46        A          LDA     WORK3
00608A 1ADC E7 07        A          STA     7,X
00609A 1ADE 3F 4A        A          CLR     WORK7
00610A 1AE0 A6 01        A          LDA     #1
00611A 1AE2 B7 49        A          STA     WORK6
00612A 1AE4 CC 1C32      A          JMP     TRACE
00613A 1AE7 5C                      SWICMP  INCX
```

Figure 3-17. ASSIST05 Program Listing (Continued)

```
00614A 1AE8 5C                    INCX
00615A 1AE9 5C                    INCX
00616A 1AEA 3A 4B       A         DEC     PNCNT
00617A 1AEC 26 DB     1AC9        BNE     SWITRY
00618                           * FALL INTO REGISTER DISPLAY FOR BREAKPOINT



00620                           ***********************************************
00621                           * R -- PRINT REGISTERS                        *
00622                           ***********************************************
00623A 1AEE CD 19E8     A CMDR    JSR     PUTSP     SPACE BEFORE DISPLAY
00624A 1AF1 AD 30     1B23        BSR     LOCSTK    LOCATE STACK-4
00625A 1AF3 E6 07       A         LDA     7,X       OFFSET FOR PC HIGH
00626A 1AF5 E7 07       A         STA     7,X       RESTORE INTO STACK
00627A 1AF7 CD 1969     A         JSR     PUTBYT    PLACE BYTE OUT
00628A 1AFA E6 08       A         LDA     8,X       OFFSET TO PC LOW
00629A 1AFC AD 1F     1B1D        BSR     CRBYTS    TO HEX AND SPACE
00630A 1AFE E6 05       A         LDA     5,X       NOW TO A REGISTER
00631A 1B00 AD 1B     1B1D        BSR     CRBYTS    TO HEX AND SPACE
00632A 1B02 E6 06       A         LDA     6,X       NOW X
00633A 1B04 AD 17     1B1D        BSR     CRBYTS    HEX AND SPACE
00634A 1B06 E6 04       A         LDA     4,X       NOW CONDITION CODE
00635A 1B08 AA E0       A         ORA     #$E0      SET ON UNUSED BITS
00636A 1B0A E7 04       A         STA     4,X       RESTORE
00637A 1B0C AD 0F     1B1D        BSR     CRBYTS    HEX AND SPACE
00638A 1B0E 9F                    TXA               STACK POINTER-3
00639A 1B0F AB 08       A         ADD     #8        TO USERS STACK POINTER
00640A 1B11 AD 0A     1B1D        BSR     CRBYTS    TO HEX AND SPACE
00641A 1B13 CC 1A26     A GTOCMD  JMP     CMD       BACK TO COMMAND HANDLER
00642                           * PRINT ADDRESS SUBROUTINE (X UNCHANGED)
00643A 1B16 B6 44       A PRTADR  LDA     ADDRH     LOAD HIGH BYTE
00644A 1B18 CD 1969     A         JSR     PUTBYT    SEND OUT AS HEX
00645A 1B1B B6 45       A         LDA     ADDRL     LOAD LOW BYTE
00646A 1B1D CD 1969     A CRBYTS  JSR     PUTBYT    PUT OUT IN HEX
00647A 1B20 CC 19E8     A         JMP     PUTSP     FOLLOW WITH A SPACE



00649                           ***********************************************
00650                           *  LOCSTK - LOCATE CALLERS STACK POINTER      *
00651                           * RETURNS X=STACK POINTER-3                   *
00652                           * A VOLATILE                                  *
00653                           ***********************************************
00654A 1B23 AD 01     1B26 LOCSTK BSR     LOCST2    LEAVE ADDRESS ON STACK
00655           001B      A STKHI  EQU     */256     HI BYTE ON STACK
00656           0025      A STKLOW EQU     *-(*/256)*256 LOW BYTE ON STACK
00657A 1B25 81                    RTS               RETURN WITH RESULT
00658A 1B26 AE 7F       A LOCST2  LDX     #$7F      LOAD HIGH STACK WORD ADDRESS
00659A 1B28 A6 1B       A LOCLOP  LDA     #STKHI    HIGH BYTE FOR COMPARE
00660A 1B2A 5A                    LOCDWN  DECX      TO NEXT LOWER BYTE IN STACK
00661A 1B2B F1                    CMP     ,X        ? THIS THE SAME
00662A 1B2C 26 FC     1B2A        BNE     LOCDWN    IF NOT TRY NEXT LOWER
00663A 1B2E A6 25       A         LDA     #STKLOW   COMPARE WITH LOW ADDRESS BYTE
00664A 1B30 E1 01       A         CMP     1,X       ? FOUND RETURN ADDRESS
00665A 1B32 26 F4     1B28        BNE     LOCLOP    LOOP IF NOT
00666A 1B34 81                    RTS               RETURN WITH X SET
```

Figure 3-17. ASSIST05 Program Listing (Continued)

```
PAGE  014  ASSIST05.SA:1  ASSIST

00668                          ********************************************
00669                          *  B -- BREAKPOINT CLEAR, SET, OR DISPLAY    *
00670                          ********************************************
00671A 1B35 CD 19D0      A CMDB   JSR     CHRIN     READ NEXT CHARACTER
00672A 1B38 A1 20        A        CMP     #'        ? DISPLAY ONLY
00673A 1B3A 26 38     1B74        BNE     BDSPLY    BRANCH IF SO
00674A 1B3C AD 4F     1B8D        BSR     PGTADR    OBTAIN BREAKPOINT NUMBER
00675A 1B3E 5D                    TSTX              ? ANY HIGH BYTE VALUE
00676A 1B3F 26 49     1B8A        BNE     BKERR     ERROR IF SO
00677A 1B41 4A                    DECA              DOWN COUNT BY ONE
00678A 1B42 A1 03        A        CMP     #NUMBKP   ? TO HIGH
00679A 1B44 24 44     1B8A        BHS     BKERR     ERROR IF SO
00680A 1B46 48                    ASLA              TIMES TWO
00681A 1B47 BB 45        A        ADD     ADDRL     PLUS ONE FOR THREE TIMES
00682A 1B49 AB 38        A        ADD     #BKPTBL   FIND TABLE ADDRESS
00683A 1B4B 4A                    DECA
00684A 1B4C B7 43        A        STA     WORK2     SAVE ADDRESS
00685A 1B4E AD 3D     1B8D        BSR     PGTADR    OBTAIN ADDRESS
00686A 1B50 A3 20        A        CPX     #$20
00687A 1B52 24 36     1B8A        BHS     BKERR
00688A 1B54 BE 43        A        LDX     WORK2     RELOAD ENTRY POINTER
00689A 1B56 E7 01        A        STA     1,X       SAVE LOW ADDRESS
00690A 1B58 26 08     1B62        BNE     BKNOCL    BRANCH IF NOT ZERO
00691A 1B5A B6 44        A        LDA     ADDRH     LOAD HIGH ADDRESS
00692A 1B5C 26 06     1B64        BNE     BKNCR     BRANCH NOT NULL
00693A 1B5E 4A                    DECA              CREATE NEGATIVE VALUE
00694A 1B5F F7                    STA     ,X        STORE AS HIGH BYTE
00695A 1B60 20 B1     1B13        BRA     GTOCMD    END COMMAND
00696A 1B62 B6 44        A BKNOCL LDA     ADDRH     LOAD HIGH ADDRESS
00697A 1B64 F7           BKNCR    STA     ,X        STORE HIGH BYTE
00698A 1B65 CD 1943      A        JSR     LOAD      LOAD BYTE AT THE ADDRESS
00699A 1B68 43                    COMA              INVERT IT
00700A 1B69 CD 1952      A        JSR     STORE     ATTEMPT STORE
00701A 1B6C 25 1C     1B8A        BCS     BKERR     ERROR IF DID NOT STORE
00702A 1B6E 43                    COMA              RESTORE PROPER VALUE
00703A 1B6F CD 1952      A        JSR     STORE     STORE IT BACK
00704A 1B72 20 9F     1B13        BRA     GTOCMD    END COMMAND

00706                          * DISPLAY BREAKPOINTS
00707A 1B74 CD 1CC3      A BDSPLY JSR     SCNBKP    PREPARE SCAN OF TABLE
00708A 1B77 F6           BDSPLP   LDA     ,X        OBTAIN HIGH BYTE
00709A 1B78 2B 07     1B81        BMI     BDSKP     SKIP IF UNUSED SLOT
00710A 1B7A CD 1969      A        JSR     PUTBYT    PRINT OUT HIGHT BYTE
00711A 1B7D E6 01        A        LDA     1,X       LOAD LOW BYTE
00712A 1B7F AD 9C     1B1D        BSR     CRBYTS    PRINT IT OUT WITH A SPACE
00713A 1B81 5C           BDSKP    INCX              TO
00714A 1B82 5C                    INCX              NEXT
00715A 1B83 5C                    INCX              ENTRY
00716A 1B84 3A 4B        A        DEC     PNCNT     COUNT DOWN
00717A 1B86 26 EF     1B77        BNE     BDSPLP    LOOP IF MORE
00718A 1B88 20 89     1B13        BRA     GTOCMD    END COMMAND

00720A 1B8A CC 1A74      A BKERR  JMP     CMDERR    GIVE ERROR RESPONSE



00722                          ********************************************
00723                          *  W -- WRITE MEMORY TO TAPE FILE S1/S9      *
```

Figure 3-17. ASSIST05 Program Listing (Continued)

```
00724                    ***********************************************
00725A 1B8D CD 19A9    A PGTADR JSR     GETADR   OBTAIN INPUT ADDRESS
00726A 1B90 24 F8    1B8A       BCC     BKERR    ABORT IF NONE
00727A 1B92 BE 44      A        LDX     ADDRH    READY HIGH BYTE
00728A 1B94 B6 45      A        LDA     ADDRL    READY LOW BYTE
00729A 1B96 81                  RTS              BACK TO PUNCH COMMAND

00731A 1B97 AD F4    1B8D CMDW  BSR     PGTADR   GET STARTING ADDRESS
00732A 1B99 A3 20      A        CPX     #$20
00733A 1B9B 24 ED    1B8A       BHS     BKERR
00734A 1B9D B7 47      A        STA     WORK4    INTO WORK4
00735A 1B9F BF 46      A        STX     WORK3    AND WORK3
00736A 1BA1 AD EA    1B8D       BSR     PGTADR   GET ENDING ADDRESS
00737A 1BA3 A3 20      A        CPX     #$20
00738A 1BA5 24 E3    1B8A       BHS     BKERR
00739A 1BA7 4C                  INCA             ADD ONE TO INCLUDE TOP BYTE
00740A 1BA8 26 01    1BAB       BNE     PUPH     BRANCH NO CARRY
00741A 1BAA 5C                  INCX             UP HIGH BYTE AS WELL
00742A 1BAB B0 47      A PUPH   SUB     WORK4    COMPUTE SIZE
00743A 1BAD B7 4D      A        STA     PNRCNT+1 AND SAVE
00744A 1BAF 9F                  TXA              NOW
00745A 1BB0 B2 46      A        SBC     WORK3    SIZE HIGH BYTE
00746A 1BB2 B7 4C      A        STA     PNRCNT   AND SAVE
00747A 1BB4 B6 47      A        LDA     WORK4    MOVE
00748A 1BB6 B7 45      A        STA     ADDRL    TO
00749A 1BB8 B6 46      A        LDA     WORK3    MEMORY
00750A 1BBA B7 44      A        STA     ADDRH    POINTER
00751                    * ADDR->MEMORY START,  PNRCNT=BYTE COUNT OF AREA
00752                    * NOW TURN ON THE PUNCH
00753A 1BBC A6 12      A        LDA     #DC2     PUNCH ON CONTROL
00754A 1BBE CD 19EA    A        JSR     CHROUT   SEND OUT
00755                    * NOW SEND CR FOLLOWED BY 24 NULLS AND 'S1'
00756A 1BC1 AD 53    1C16 PREC  BSR     PNCRNL   SEND CR/LF AND NULLS
00757A 1BC3 AE 17      A        LDX     #MSGS1-MBASE POINT TO STRING
00758A 1BC5 CD 1985    A        JSR     PDATA1   SEND 'S1' OUT
00759                    * NOW SEND NEXT 24 BYTES OR LESS IF TO THE END
00760A 1BC8 B6 4D      A        LDA     PNRCNT+1 LOW COUNT LEFT
00761A 1BCA A0 18      A        SUB     #24      MINUS 24
00762A 1BCC B7 4D      A        STA     PNRCNT+1 STORE RESULT
00763A 1BCE 24 08    1BD8       BCC     PALL24   IF NO CARRY THEN OK
00764A 1BD0 3A 4C      A        DEC     PNRCNT   DOWN HIGH BYTE
00765A 1BD2 2A 04    1BD8       BPL     PALL24   ALL 24 OK
00766A 1BD4 AB 18      A        ADD     #24      WAS LESS SO BACK UP TO ORIGINAL
00767A 1BD6 20 02    1BDA       BRA     PGOTC    GO USE COUNT HERE
00768A 1BD8 A6 18      A PALL24 LDA     #24      USE ALL 24
00769A 1BDA B7 4B      A PGOTC  STA     PNCNT    COUNT FOR THIS RECORD
00770                    * SEND THE FRAME COUNT AND START CHECKSUMMING
00771A 1BDC 3F 4E      A        CLR     CHKSUM
00772A 1BDE AB 03      A        ADD     #3       ADJUST FOR COUNT AND ADDRESS
00773A 1BE0 AD 2B    1C0D       BSR     PUNBYT   SEND FRAME COUNT
00774                    * SEND ADDRESS
00775A 1BE2 B6 44      A        LDA     ADDRH    HI BYTE
00776A 1BE4 AD 27    1C0D       BSR     PUNBYT   SEND IT
00777A 1BE6 B6 45      A        LDA     ADDRL    LOW BYTE
00778A 1BE8 AD 23    1C0D       BSR     PUNBYT   SEND IT
00779                    * NOW SEND DATA
00780A 1BEA CD 1943    A PUNLOP JSR     LOAD     LOAD NEXT BYTE
00781A 1BED AD 1E    1C0D       BSR     PUNBYT   SEND IT OUT
```

Figure 3-17. ASSIST05 Program Listing (Continued)

```
00782A 1BEF CD 1962     A          JSR     PTRUP1    UP ADDRESS BY ONE
00783A 1BF2 3A 4B       A          DEC     PNCNT     COUNT DOWN
00784A 1BF4 26 F4    1BEA          BNE     PUNLOP    LOOP UNTIL ZERO
00785                              * SEND OUT THE CHECKSUM
00786A 1BF6 B6 4E       A          LDA     CHKSUM    LOAD CHECKSUM
00787A 1BF8 43                     COMA              COMPLEMENT IT
00788A 1BF9 AD 12    1C0D          BSR     PUNBYT    SEND IT OUT
00789                              * LOOP OR SEND S9
00790A 1BFB B6 4C       A          LDA     PNRCNT    ? MINUS
00791A 1BFD 2B 04    1C03          BMI     PNEND     YES QUIT
00792A 1BFF BB 4D       A          ADD     PNRCNT+1  ? ZERO
00793A 1C01 26 BE    1BC1          BNE     PREC      NO, DO NEXT RECORD
00794A 1C03 AD 11    1C16 PNEND    BSR     PNCRNL    SEND CR AND NULLS
00795A 1C05 AE 1A       A          LDX     #MSGS9-MBASE LOAD S9 TEXT
00796A 1C07 CD 1985     A          JSR     PDATA1    SEND AND TO COMMAND HANDLER
00797A 1C0A CC 1A26     A          JMP     CMD       TO COMMAND HANDLER

00799                              * SUB TO SEND BYTE IN HEX AND ADJUST CHECKSUM
00800A 1C0D 97                     PUNBYT  TAX               SAVE BYTE
00801A 1C0E BB 4E       A          ADD     CHKSUM    ADD TO CHECKSUM
00802A 1C10 B7 4E       A          STA     CHKSUM    STORE BACK
00803A 1C12 9F                     TXA               RESTORE BYTE
00804A 1C13 CC 1969     A          JMP     PUTBYT    SEND OUT IN HEX

00806                              * SUB TO SEND CR/LF AND 24 NULLS
00807A 1C16 CD 197F     A PNCRNL   JSR     PCRLF     SEND CR/LF
00808A 1C19 AE 18       A          LDX     #24       COUNT NULLS
00809A 1C1B 4F                     PNULLS  CLRA              CREATE NULL
00810A 1C1C CD 19EA     A          JSR     CHROUT    SEND OUT
00811A 1C1F 5A                     DECX              COUNT DOWN
00812A 1C20 26 F9    1C1B          BNE     PNULLS    LOOP UNTIL DONE
00813A 1C22 81                     RTS               RETURN


00815                              ********************************************
00816                              *  T -- TRACE COMMAND                      *
00817                              ********************************************
00818A 1C23 A6 01       A CMDT     LDA     #1        DEFAULT COUNT
00819A 1C25 B7 45       A          STA     ADDRL     TO ONE *GETADR CLEARS ADDRH*
00820A 1C27 CD 19A9     A          JSR     GETADR    BUILD ADDRESS IF ANY
00821A 1C2A B6 44       A          LDA     ADDRH     SAVE VALUE IN TEMPORARY
00822A 1C2C B7 4A       A          STA     WORK7     LOCATIONS FOR LATER
00823A 1C2E B6 45       A          LDA     ADDRL     USE
00824A 1C30 B7 49       A          STA     WORK6
00825                              * SETUP TIMER TO TRIGGER INTERRUPT
00826          1C32     A TRACE    EQU     *
00827A 1C32 CD 1B23     A          JSR     LOCSTK
00828A 1C35 E6 04       A          LDA     4,X       GET CURRENT USER I-MASK
00829A 1C37 A4 08       A          AND     #8
00830A 1C39 B7 48       A          STA     WORK5     SAVE IT
00831A 1C3B E6 07       A          LDA     7,X       GET CURRENT USER PC
00832A 1C3D B7 44       A          STA     ADDRH
00833A 1C3F E6 08       A          LDA     8,X
00834A 1C41 B7 45       A          STA     ADDRL
00835A 1C43 CD 1943     A          JSR     LOAD      GET OPCODE
00836A 1C46 A1 83       A          CMP     #$83      SWI?
00837A 1C48 26 0E    1C58          BNE     TRACE3    IF YES THEN
```

Figure 3-17. ASSIST05 Program Listing (Continued)

```
00838A 1C4A B6 45        A          LDA     ADDRL     INC USER PC
00839A 1C4C AB 01        A          ADD     #1
00840A 1C4E E7 08        A          STA     8,X
00841A 1C50 B6 44        A          LDA     ADDRH
00842A 1C52 A9 00        A          ADC     #0
00843A 1C54 E7 07        A          STA     7,X
00844A 1C56 20 2D     1C85          BRA     TIRQ      CONTINUE TO TRACE
00845A 1C58 A1 9B        A TRACE3    CMP     #$9B      SEI?
00846A 1C5A 26 14     1C70          BNE     TRACE2    IF YES
00847A 1C5C E6 04        A          LDA     4,X       THEN SET IT IN THE STACK
00848A 1C5E AA 08        A          ORA     #8
00849A 1C60 E7 04        A          STA     4,X
00850A 1C62 B6 45        A          LDA     ADDRL     THEN INC USER PC
00851A 1C64 AB 01        A          ADD     #1
00852A 1C66 E7 08        A          STA     8,X
00853A 1C68 B6 44        A          LDA     ADDRH
00854A 1C6A A9 00        A          ADC     #0
00855A 1C6C E7 07        A          STA     7,X
00856A 1C6E 20 15     1C85          BRA     TIRQ      CONTINUE TO TRACE
00857A 1C70 A1 9A        A TRACE2    CMP     #$9A      CLI?
00858A 1C72 26 02     1C76          BNE     TRACE1    IF YES THEN
00859A 1C74 3F 48        A          CLR     WORK5
00860A 1C76 E6 04        A TRACE1    LDA     4,X
00861A 1C78 A4 F7        A          AND     #$F7
00862A 1C7A E7 04        A          STA     4,X
00863A 1C7C A6 10        A          LDA     #16       THEN SET UP TIMER
00864A 1C7E B7 08        A          STA     TIMER
00865A 1C80 A6 08        A          LDA     #8
00866A 1C82 B7 09        A          STA     TIMEC
00867A 1C84 80                      RTI               EXECUTE ONE INSTRUCTION


00869                               ********************************************
00870                               *  TIRQ -- TIMER INTERRUPT ROUTINE         *
00871                               ********************************************
00872          1C85      A TIRQ    EQU     *
00873                               *  RESTORE I-MASK TO PROPER STATE
00874A 1C85 A6 40        A          LDA     #$40
00875A 1C87 B7 09        A          STA     TIMEC
00876A 1C89 CD 1B23      A          JSR     LOCSTK
00877A 1C8C E6 04        A          LDA     4,X
00878A 1C8E BA 48        A          ORA     WORK5
00879A 1C90 E7 04        A          STA     4,X
00880                               *  SEE IF MORE TRACING IS DESIRED
00881A 1C92 3A 49        A          DEC     WORK6
00882A 1C94 26 9C     1C32          BNE     TRACE
00883A 1C96 3D 4A        A          TST     WORK7
00884A 1C98 27 04     1C9E          BEQ     DISREG
00885A 1C9A 3A 4A        A          DEC     WORK7
00886A 1C9C 20 94     1C32          BRA     TRACE
00887A 1C9E CC 1AEE      A DISREG    JMP     CMDR


00889                               ********************************************
00890                               *  INT -- INTERRUPT ROUTINE                *
00891                               ********************************************
```

Figure 3-17. ASSIST05 Program Listing (Continued)

```
PAGE  018  ASSIST05.SA:1  ASSIST

00892           1CA1      A IRQ     EQU     *
00893A 1CA1 CC 1A74       A         JMP     CMDERR    HARDWARE INTERRUPT UNUSED


00895                               ********************************************
00896                               * TWIRQ - TIMER INTERRUPT ROUTINE (WAIT MODE)*
00897                               ********************************************
00898           1CA4      A TWIRQ   EQU     *
00899A 1CA4 CC 1A74       A         JMP     CMDERR    TIMER WAIT INTERRUPT UNUSED


00901                               ********************************************
00902                               *  DELBKP - DELETE BREAKPOINT SUBROUTINE     *
00903                               ********************************************
00904A 1CA7 AD 1A     1CC3 REMBKP BSR     SCNBKP    SETUP PARAMETERS
00905A 1CA9 2A 17     1CC2        BPL     REMRTS    RETURN IF NOT IN
00906A 1CAB F6                    REMLOP LDA      ,X        LOAD HIGH ADDRESS
00907A 1CAC 2B 0B     1CB9        BMI     REMNOB    SKIP IF NULL
00908A 1CAE B7 44       A         STA     ADDRH     STORE HIGH ADDRESS
00909A 1CB0 E6 01       A         LDA     1,X       LOAD LOW ADDRESS
00910A 1CB2 B7 45       A         STA     ADDRL     STORE IT
00911A 1CB4 E6 02       A         LDA     2,X       LOAD OPCODE
00912A 1CB6 CD 1952     A         JSR     STORE     STORE IT BACK OVER 'SWI'
00913A 1CB9 5C                    REMNOB INCX             TO
00914A 1CBA 5C                           INCX             NEXT
00915A 1CBB 5C                           INCX             ENTRY
00916A 1CBC 3A 4B       A         DEC     PNCNT     COUNT DOWN
00917A 1CBE 26 EB     1CAB        BNE     REMLOP    LOOP IF MORE
00918A 1CC0 33 41       A         COM     SWIFLG    MAKE POSITIVE TO SHOW REMOVED
00919A 1CC2 81                    REMRTS RTS              RETURN

00921                             * SETUP FOR BREAKPOINT TABLE SCAN
00922A 1CC3 A6 03       A SCNBKP LDA      #NUMBKP   LOAD NUMBER OF BREAKPOINTS
00923A 1CC5 B7 4B       A         STA     PNCNT     SETUP FOR COUNTDOWN
00924A 1CC7 AE 38       A         LDX     #BKPTBL   LOAD TABLE ADDRESS
00925A 1CC9 3D 41       A         TST     SWIFLG    TEST IF BREAKPOINTS IN ALREADY
00926A 1CCB 81                    RTS               RETURN


00928                               ********************************************
00929                               *       INTERRUPT VECTORS                  *
00930                               ********************************************
00931A 1FF6                               ORG     MONSTR+$800-$A START OF VECTORS
00932A 1FF6     004F      A         FDB     VECRAM    TIMER INTERRUPT HANDLER (WAIT MODE
00933A 1FF8     0052      A         FDB     VECRAM+3  TIMER INTERRUPT HANDLER
00934A 1FFA     0055      A         FDB     VECRAM+6  INTERRUPT HANDLER
00935A 1FFC     0058      A         FDB     VECRAM+9  SWI HANDLER
00936A 1FFE     19FD      A         FDB     RESET     POWER ON VECTOR


00938                               END
```

Figure 3-17. ASSIST05 Program Listing (Concluded)

CHAPTER 4
HARDWARE FEATURES

4.1 INTRODUCTION

Each member of the M6805 HMOS/M146805 CMOS Family (except for the MC146805E2) contains, on-chip, nearly all of the support hardware necessary for a complete processor system. The block diagram of Figure 4-1 shows a Central Processing Unit (CPU) which is identical for all members of the family, including the MC146805E2. There is one main difference in various family members and that is the size of the stack pointer and program counter registers. Since the size of these two registers is determined by the amount of device memory, they vary from 11 bits to 13 bits. Each family member contains an on-chip oscillator which provides the processor timing, plus reset, and interrupt logic. Peripheral I/O such as a timer, some bidirectional I/O lines, RAM, and ROM (except for the MC146805E2) are included on-chip in all family members. The peripherals and memory are located in similar locations throughout the family; therefore, once the user is familiar with any family device, he is familiar with all. In addition, new devices can be incorporated in the family by adding to and/or subtracting from the peripheral blocks associated with the CPU. These peripheral blocks could include additional I/O lines, more RAM, EPROM, A/D converter, phase-lock-loop, or an external bus. The choice of using inexpensive HMOS or low-power, static CMOS is also available.

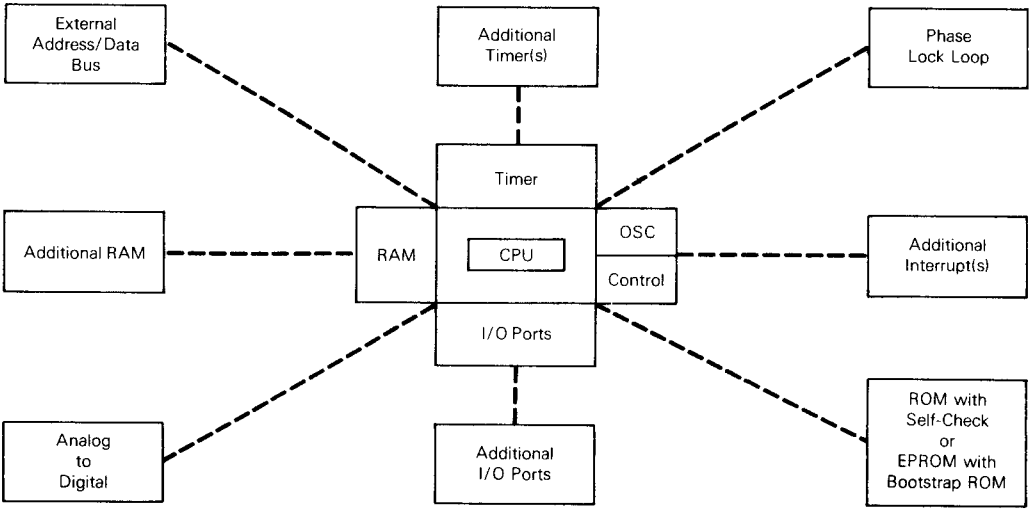


Figure 4-1. M6805 HMOS/M146805 CMOS Family Block Diagram

The M6805 HMOS/M146805 CMOS Family of MCU/MPU devices are implemented using a single address map, memory mapped I/O, and Von Neumann architecture. Peripheral I/O devices (A/D, timer, PLL, etc.) are accessed by the CPU via the peripheral control and/or data registers which are located in the address map. Data is transferred to the peripheral I/O devices with the same instructions that are used to access memory. The key to using the M6805 HMOS/M146805 CMOS Family I/O features is in learning how the peripheral registers effect the device operation. Since a second address map is not used, there is no need for the system designer to learn a second set of specialized I/O instructions.

4.2 PROCESSING TECHNOLOGY

As stated above, system designers have the option of using either HMOS (M6805) or CMOS (M146805) technology. Since each technology has its advantages, there are applications which will favor one over the other. Table 4-1 provides a comparison of representative features between HMOS and CMOS.

Table 4-1. Comparison of Features Between HMOS and CMOS

HMOS	CMOS
Inexpensive Due to Smaller Die Size	Low Power Consumption
Fast	Silicon-Gate Devices are as Fast as HMOS Devices Completely Static Operation Wider Voltage Range (3-6 V) Increased Noise Immunity
Consumes Ten Times More Power than CMOS	More Expensive Since CMOS Cell is Larger
Dynamic Operation (Requires Continuous Clock) Limited Voltage Range	Sensitive to SCR Latchup

4.3 TEMPORARY STORAGE (RAM)

Random Access Memory (RAM) is used as temporary storage by the CPU. The RAM is temporary in that it is volatile and its contents are lost if power is removed. However, since RAM can be read from or written to, it is best used for storing variables. All on-chip RAM is contained in the first 128 memory locations and the top of RAM is presently used by the processor as a program control stack. The stack is used to store return addresses for subroutine calls and the machine state for interrupts. The stack pointer register is used to keep track of (point to) the next free stack address location. The stack operates in a LIFO (last-in-first-out) mode so that operations may be nested. The actual stack size varies between the different family members; however, in all cases, exceeding the stack limit should be avoided. If the stack limit is exceeded, the stack pointer wraps around to the top of the stack ($7F) and more than likely stack data is lost. Each interrupt requires five bytes of stack space and each subroutine requires two bytes. If, at worst case, a program requires five levels of subroutine nesting and one level of interrupt, then 15 bytes of stack space should be reserved. Any unreserved stack RAM may be used for other purposes.

Low-power standby RAM for HMOS is available on the MC6805P4. Although the processor is dynamic, the RAM is static and may be powered from a separate standby supply voltage which does not power any other part of the device, thus, lowering standby supply current requirements. The amount of standby RAM implemented is a mask option and is determined by the minimum necessary for the particular application.

4.4 PERMANENT STORAGE (ROM OR EPROM)

All M6805 HMOS/M146805 CMOS Family devices, except the MC146805E2, contain some form of permanent, non-volatile memory. It may be either mask programmed ROM or UV-light erasable EPROM; however, the M68705 HMOS EPROM versions contain EPROM as the main storage and a small mask ROM which is used to store the bootstrap programming routines. Non-volatile memory is generally used to store the user programs as well as tables and constants. The mask ROM versions are the most economical for large quantities while the EPROM versions are best suited for limited quantities used for production or prototyping. Currently three EPROM versions exist. Each has slightly more storage and versatility than the current mask ROM versions; however, the EPROM versions can emulate the functions of more than one of the current mask ROM versions and could be used for future mask ROM versions.

4.5 OSCILLATOR

This on-chip oscillator contained on every M6805 HMOS/M146805 CMOS Family device essentially generates the timing used by the device. The oscillator can be used in a number of different modes as shown in Figure 4-2. Each mode has its advantages and the basic trade-off is between economy and accuracy.

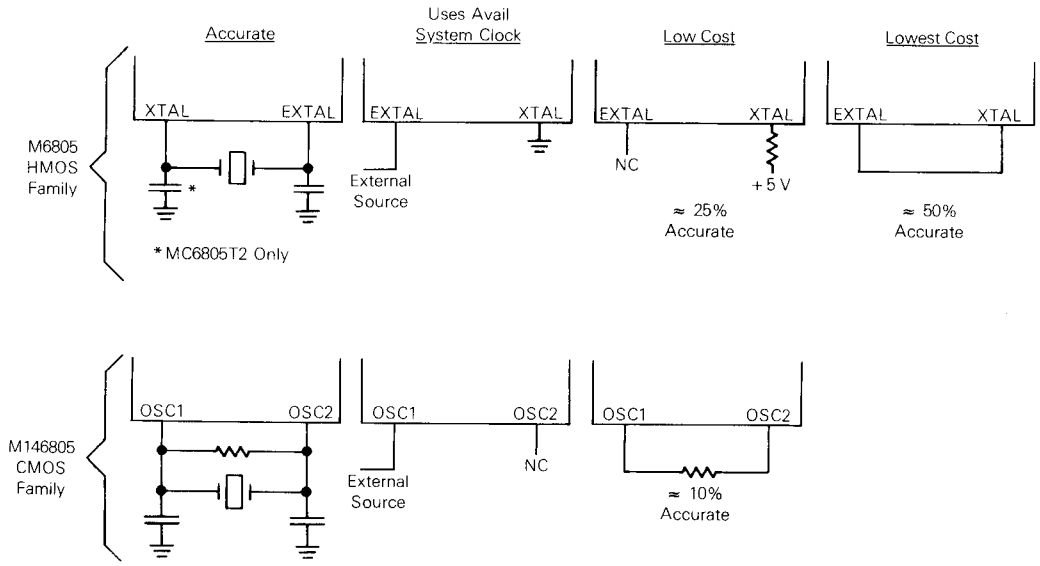


Figure 4-2. M6805 HMOS/M146805 CMOS Family Oscillator Modes

Except for the EPROM members of M6805 HMOS Family, a manufacturing mask option is required to select either the crystal oscillator or the resistor oscillator circuit. The oscillator frequency is internally divided by four to produce the internal system clocks. The EPROM devices of the M6805 HMOS Family utilize the mask option register (MOR) to select the crystal or resistor oscillator circuit.

The M146805 CMOS Family devices also use a manufacturing mask option to select either the crystal or resistor circuit. However, a second manufacturing mask option provides either a divide-by-two or divide-by-four circuit to produce the internal system clock. The EPROM devices of the M146805 CMOS Family also utilize the mask option register (MOR) to select the crystal or resistor oscillator circuit.

4.6 RESETS

The M6805 HMOS/M146805 CMOS Family processor can be reset in two ways: either by the initial power-up or by the external reset input pin ($\overline{\text{RESET}}$). Additionally, a low voltage inhibit (LVI) circuit is included on some HMOS masked ROM versions to force a reset if V_{CC} falls to V_{LVI}. Any of the reset methods allow an orderly start-up; additionally, the $\overline{\text{RESET}}$ input can be used to exit the CMOS STOP and WAIT modes of program execution. Both the LVI and external $\overline{\text{RESET}}$ inputs allow the processor to recover from otherwise catastrophic errors. External reset ($\overline{\text{RESET}}$) is implemented with a Schmitt trigger input for improved noise immunity. Figure 4-3 illustrates the required timing and logic levels for devices implemented with LVI. All M6805 HMOS Family members have the equivalent of an internal pullup resistor as shown in Figure 4-4 so that the $\overline{\text{RESET}}$ pin will reflect the drop in V_{CC}.

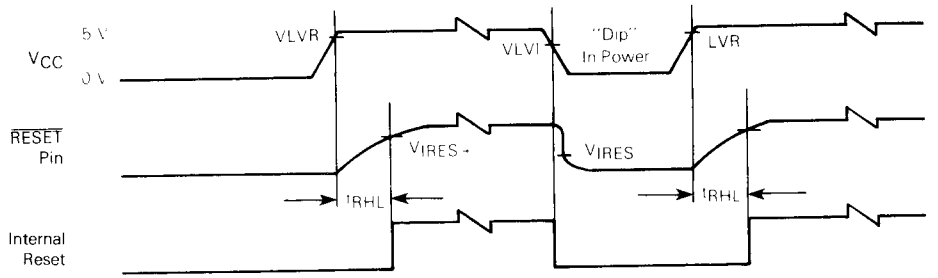


Figure 4-3. Power and Reset Timing

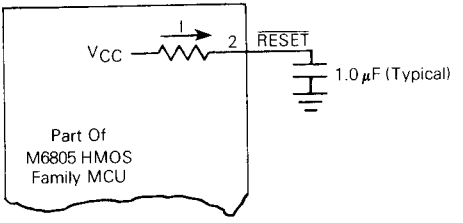


Figure 4-4. Power up Reset Delay Circuit

HMOS power-on reset circuitry includes the equivalent of an internal pullup resistor, so that only a capacitor is required externally (see Figure 4-4). The power-on reset occurs when a positive transition is detected on V_{CC}. The power-on reset is used strictly for power turn-on conditions and should not be used to detect any drops in the power supply voltage. There is no provision for a power-down reset. For CMOS devices, the power-on circuitry provides for a 1920 t_{cyc} delay from the time of the first oscillator operation. If the external $\overline{RESET}$ pin is low at the end of the 1920 t_{cyc} time out, the processor remains in the reset condition until the $\overline{RESET}$ pin goes high.

Any reset causes the following to occur:

1. All interrupt requests are cleared to "0".
2. All interrupt masks are set to "1".
3. All data direction registers are cleared to "0" (input).
4. The stack pointer is reset to $7F (top of stack).
5. The STOP and WAIT latches (M146805 CMOS only) are reset.
6. The reset vector is fetched and placed in the program counter. (The reset vector contains the address of the reset routine.)

4.7 INTERRUPTS

4.7.1 General

The M6805 HMOS/M146805 CMOS Family program execution may be interrupted in the following ways:

1. Externally via the $\overline{IRQ}$ (CMOS) or $\overline{INT}$ (HMOS) pins. Additionally, some M6805 HMOS members include a second external interrupt ($\overline{INT2}$). External interrupts are maskable.
2. Internally with the on-chip timer. The timer interrupt is maskable.
3. Internally by executing the software interrupt instruction (SWI). The SWI is non-maskable.

When an external or timer interrupt occurs, the interrupt is not immediately serviced until the current instruction being executed is completed. Until the completion of the current instruction, the interrupt is considered pending. After the current instruction execution is completed, unmasked interrupts may be serviced. If both an external and a timer interrupt are pending, the external interrupt is serviced first; however, the timer interrupt request remains pending unless it is cleared during the external interrupt service routine. The software interrupt is executed in much the same manner as any other instruction. The external interrupt pin ($\overline{IRQ}$ or $\overline{INT}$) may be tested with the BIL or BIH conditional branch instructions. These instructions may be used to allow the external interrupt pins (except $\overline{INT2}$) to be used as an additional input pin regardless of the state of the interrupt mask in the condition code register.

4.7.2 Timer Interrupt

If the timer mask bit (TCR6) is cleared, then each time the timer decrements to zero (transitions from $01 to $00) an interrupt request is generated. The actual processor interrupt is generated only if the interrupt mask bit of the condition code register is also cleared. When the interrupt is recognized, the current state of the machine is pushed onto the stack and the I bit in the condition code register is set, masking further interrupts until the present one is serviced. The contents of the timer interrupt vector, containing the location of the timer interrupt service routine, is then loaded into the program counter.

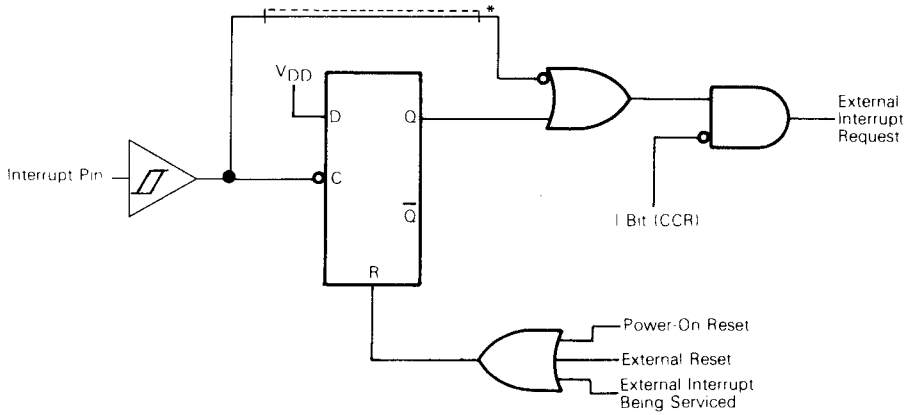
If the CMOS WAIT mode is enabled (M146805 CMOS Family only), the timer may be used to exit the low-power mode and the timer WAIT vector is used instead of the normal timer interrupt vector. Software must be used to clear the timer interrupt request bit (TCR7). At the end of the timer interrupt service routine, the software normally executes an RTI instruction which restores the machine state and starts executing the interrupted program. Note that if an external hardware interrupt is used to exit the WAIT mode, the timer interrupt will vector to the normal timer vector instead of the timer WAIT vector.

4.7.3 External Interrupts

All external interrupts are maskable. If the interrupt mask bit (I bit) of the condition code register is set, all interrupts are disabled. Clearing the I bit enables the external interrupts. Additionally, $\overline{INT2}$ requires that bit 6 of the miscellaneous register also be cleared. The external interrupts recognize both level- and edge-sensitive trigger interrupts for the M146805 CMOS Family as shown in Figure 4-5. The M6805 HMOS Family requires negative edge-sensitive trigger interrupts only. The level-sensitive line is mask optional on the MC146805G2 and MC146805F2 (see Figure 4-5). The level-sensitive triggered interrupts are generally used for multiple "wire-ORed" interrupt sources as shown in Figure 4-5b. Edge-sensitive interrupts may be used for periodic interrupts; however, since the interrupt request is latched by the processor, interrupt sources may return to other tasks. Periodic interrupt requests require that the interrupt request line be held low for at least one t_{cyc} and not be repeated until the end of the service routine and the stacking operations are complete. This ensures that all requests are recognized. The interrupt line must also be released high to allow the interrupt latch to be reset.

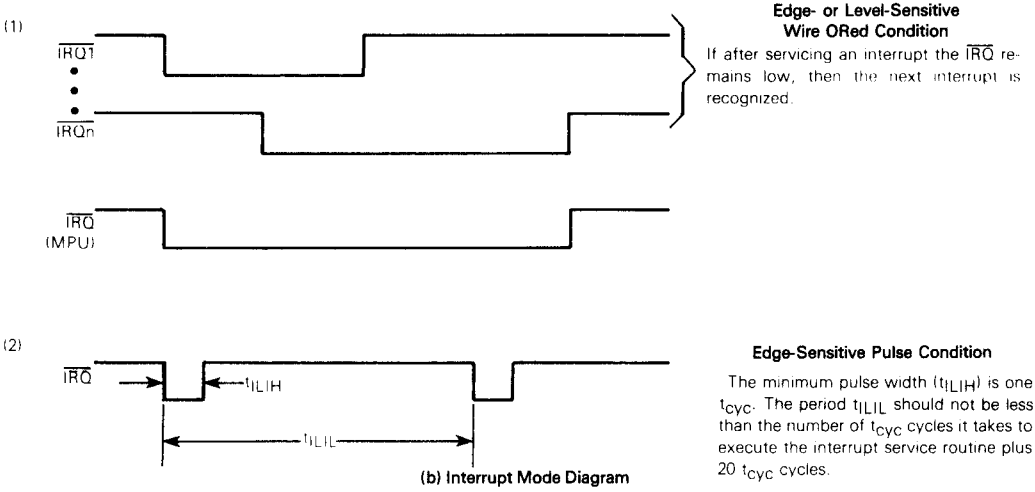
Upon servicing a pending interrupt request, the processor executes the following sequences:

1. Mask all interrupts (set I bit).
2. Stack all CPU registers.
3. Load the program counter with the appropriate vector location contents ($\overline{INT2}$ uses the same vector location as does the timer).
4. Execute service routine.



* Mask optional for M146805 CMOS Family

(a) Interrupt Functional Diagram



Edge- or Level-Sensitive Wire ORed Condition

If after servicing an interrupt the $\overline{IRQ}$ remains low, then the next interrupt is recognized.

Edge-Sensitive Pulse Condition

The minimum pulse width (t_{ILIH}) is one t_{cyc}. The period t_{ILIL} should not be less than the number of t_{cyc} cycles it takes to execute the interrupt service routine plus 20 t_{cyc} cycles.

Figure 4-5. External Interrupt

4.7.4 Software Interrupt (SWI)

The software interrupt is executed the same as any other instructon and as such will take precedence over hardware interrupts only if the I bit is set (interrupts masked). The SWI instruction is executed similar to the hardware interrupts in that the I bit is set, CPU registers are stacked, etc. The SWI is executed regardless of the state of the interrupt mask in the condition code register; however, when the I bit is clear and an external or internal hardware interrupt is pending, the SWI instruction (or any other instruction) is not fetched until after the hardware interrupts have been serviced. The SWI uses its own unique vector location.

4.8 I/O PORTS

At least 16 individually programmable, bidirectional I/O lines are included on each member of the M6805 HMOS/M146805 CMOS Family; however, more than this exists on most family members. Each line is individually programmable as either an input or an output via its corresponding data direction register (DDR) bit as shown in Figure 4-6. Table 4-2 provides a description of the effects of port data register operation. Data is written into the port output data latch regardless of the state of the DDR; therefore, initial output data should be written to the output data latch before programming the DDR. After a port line has been configured as an output, the data on that line reflects the corresponding bit of the output data latch. A read of the port data register reflects the last value written to the port output data latch for output lines and the current status of the input pins. Note that the DDRs in the M6805 HMOS Family are write-only registers and should not be used with any of the read-modify-write (RMW) instructions such as the bit manipulation instructions. The M146805 CMOS Family DDRs are read/write registers and may be used with RMW instructions.

Some devices include a number of input-only lines. These lines have no DDR and have read-only data registers.

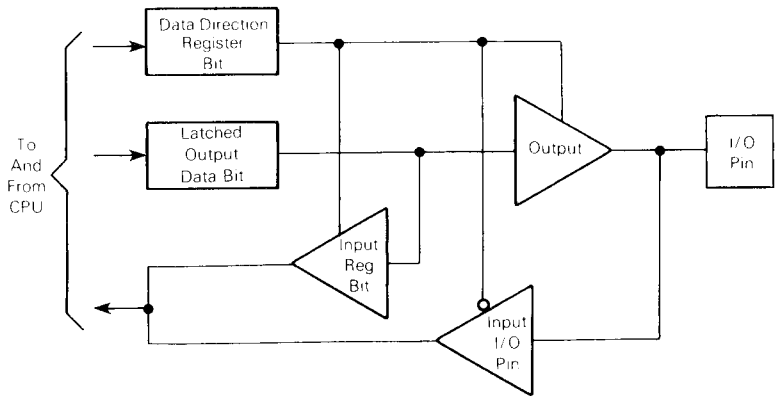


Figure 4-6. Typical Port I/O Circuitry

Table 4-2. Port Data Register Accesses

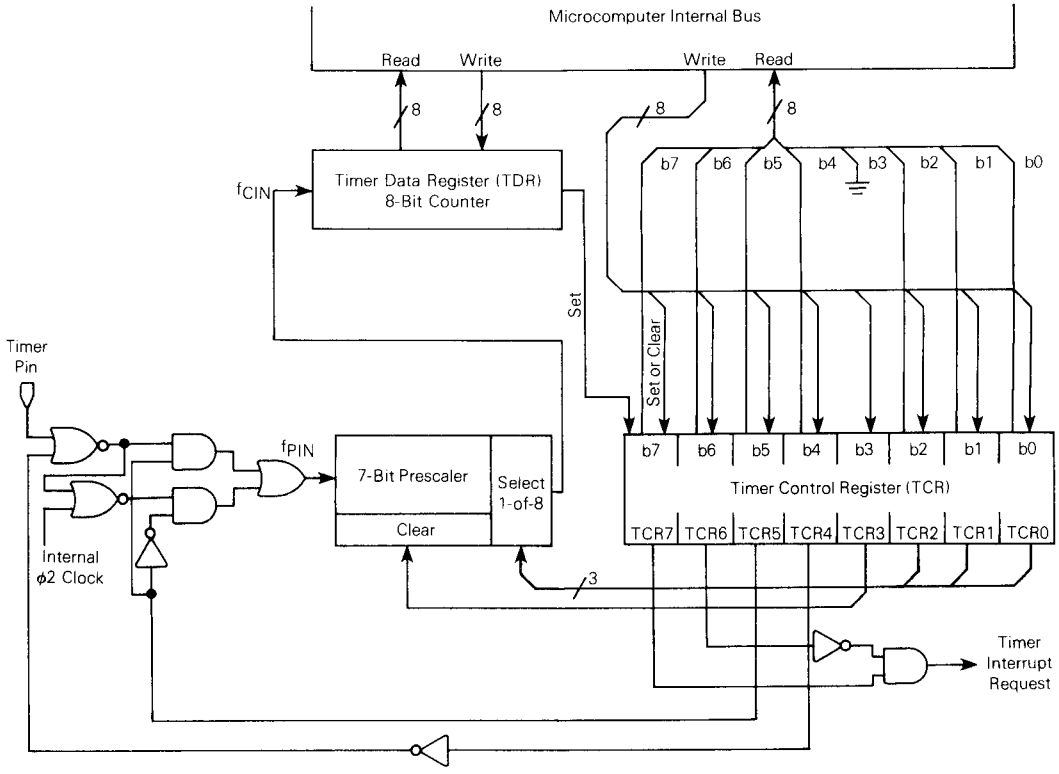
R/$\overline{W}$	DDR Bit	Results
0	0	The I/O pin is in input mode. Data is written into the output data latch.
0	1	Data is written into the output data latch and output to the I/O pin.
1	0	The state of the I/O pin is read.
1	1	The I/O pin is in an output mode. The output data latch is read.

R/$\overline{W}$ is an internal line.

4.9 TIMER DESCRIPTION

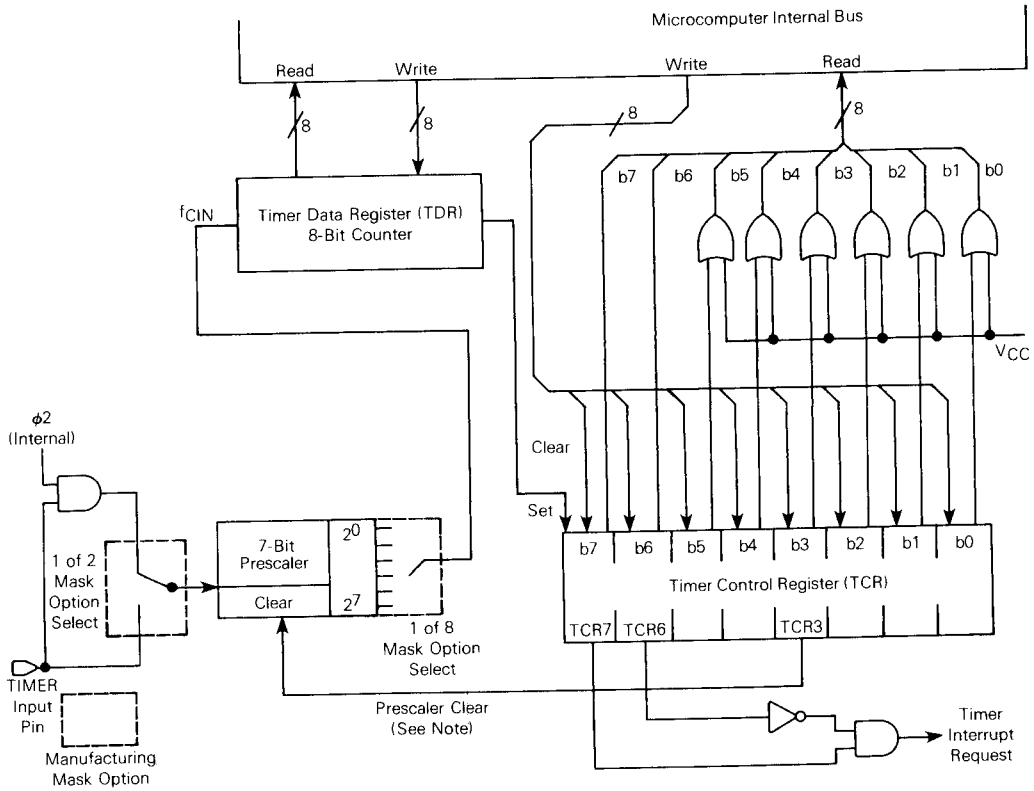
4.9.1 General

All M6805 HMOS/M146805 CMOS Family devices contain at least one timer on chip. The timer is basically composed of a 7-bit prescaler, an 8-bit counter, and interrupt logic. The M6805 HMOS and M146805 CMOS devices differ slightly in two areas. First, the input to the timer, as shown in Figures 4-7 and 4-8, is programmed differently. In the M146805 CMOS Family, the input is selected by programming bits 4 and 5 of the timer control register (TCR). In the M6805 HMOS Family they are mask programmable (except for the MC6805R3 and MC6805U3). The second difference is the prescaler which is software programmable in the M146805 CMOS Family and mask programmable in the M6805 HMOS Family.



NOTE: TCR3 always reads as a logical 0.

Figure 4-7. M146805 CMOS Family Timer Block Diagram



NOTE: The TCR3 prescaler clear bit is not available in the MC6805P2, MC6805P4, and MC6805T2; however, it is used as shown in all other M6805 HMOS Family MCUs. The TCR3 bit always reads as a logical 0.

Figure 4-8. M6805 HMOS Family Timer Block Diagram

The timer interrupt operates similarly to the external interrupts; however, users must clear the interrupt request bit (TCR7) to prevent a second timer interrrupt service from occurring.

Descriptions of the HMOS and CMOS timers follow in more detail. The EPROM versions allow either CMOS or HMOS timer operations via the programmable mask option register (MOR).

4.9.2 M146805 CMOS Family Timer

4.9.2.1 GENERAL. The MCU timer contains an 8-bit software programmable counter with 7-bit software selectable prescaler as shown in Figure 4-7. The counter may be preloaded under program control and decrements toward zero. When the counter decrements to zero, the timer interrupt request bit, i.e., bit 7 of the timer control register (TCR), is set. Then, if the timer interrupt is not masked, i.e., bit 6 of the TCR and the I bit in

the condition code register are both cleared, the processor receives an interrupt. After completion of the current instruction, the processor proceeds to store the appropriate registers on the stack, and then fetches the timer vector address in order to begin servicing.

The counter continues to count after it reaches zero, allowing the software to determine the number of internal or external input clocks since the timer interrupt request bit was set. The counter may be read at any time by the processor without disturbing the count. The contents of the counter become stable prior to the read portion of a cycle and do not change during the read. The timer interrupt request bit remains set until cleared by the software. If a clear (write TCR7 = 0) occurs before the timer interrupt is serviced, the interrupt is lost. The TCR7 bit may also be used as a scanned status bit in a non-interrupt mode of operation (TCR6 = 1).

The prescaler is a 7-bit divider which is used to extend the maximum length of the timer. Bit 0, bit 1, and bit 2 of the TCR are programmed to choose the appropriate prescaler output which is used as the counter input. The processor cannot write into or read from the prescaler; however, its contents are cleared to all "0s" by the write operation into TCR when bit 3 of the written data equals 1. This allows for truncation-free counting.

The timer input can be configured in one of three different operating modes, plus a disable mode, depending on the value written to the TCR4 and TCR5 control bits. Refer to the Timer Control Register paragraph.

4.9.2.2 TIMER INPUT MODE 1. If TCR4 and TCR5 are both programmed to a "0", the input to the timer is from an internal clock and the TIMER input pin is disabled. The internal clock mode can be used for periodic interrupt generation, as well as a reference in frequency and event measurement. The internal clock is the instruction cycle clock. During a WAIT instruction, the internal clock to the timer continues to run at its normal rate.

4.9.2.3 TIMER INPUT MODE 2. With TCR4 = 1 and TCR5 = 0, the internal clock and the TIMER input pin are ANDed to form the timer input signal. This mode can be used to measure external pulse widths. The external pulse simply turns on the internal clock for the duration of the pulse. The resolution of the measurement in this mode is ±1 clock.

4.9.2.4 TIMER INPUT MODE 3. If TCR4 = 0 and TCR5 = 1, then all inputs to the timer are disabled.

4.9.2.5 TIMER INPUT MODE 4. If TCR4 = 1 and TCR5 = 1, the internal clock input to the timer is disabled and the TIMER input pin becomes the input to the timer. In this mode, the timer can be used to count external events as well as external frequencies for generating periodic interrupts. The counter is clocked by the falling edge of the external signal.

Figure 4-7 shows a block diagram of the timer subsystem. Power-on reset and the STOP instruction cause the counter to be set to $F0.

4.9.2.6 TIMER CONTROL REGISTER (TCR). The eight bits in the TCR are used to control various functions such as configuring the operation mode, setting the division ratio of the prescaler, and generating the timer interrupt request signal. A description of each TCR bit function is provided below. All bits in this register except bit 3 are read/write bits.

7	6	5	4	3	2	1	0
TCR7	TCR6	TCR5	TCR4	TCR3	TCR2	TCR1	TCR0

TCR7 — Timer interrupt request bit: bit used to indicate the timer interrupt when it is logic "1".
1 — Set whenever the counter decrements to zero, or under program control.
0 — Cleared on external reset, power-on reset, STOP instruction, or program control.

TCR6 — Timer interrupt mask bit: when this bit is a logic "1" it inhibits the timer interrupt to the processor.
1 — Set on external reset, power-on reset, STOP instruction, or program control.
0 — Cleared under program control.

TCR5 — External or internal bit: selects the input clock source to be either the external TIMER pin or the internal clock. (Unaffected by reset.)
1 — Select external clock source.
0 — Select internal clock source.

TCR4 — External enable bit: control bit used to enable the external timer pin (Unaffected by reset.)
1 — Enable external timer pin.
0 — Disable external timer pin.

Summary of Timer Clock Source Options

TCR5	TCR4	Option
0	0	Internal Clock to Timer
0	1	AND of Internal Clock and TIMER Pin to Timer
1	0	Inputs to Timer Disabled
1	1	TIMER Pin to Timer

Refer to Figure 4-7 for logic representation.

TCR3 — Timer prescaler Reset bit: writing a "1" to this bit resets the prescaler to zero. A read of this location always indicates "0". (Unaffected by reset.)

TCR2, TCR1, TCR0 — Prescaler select bits: decoded to select one of eight taps on the prescaler. (Unaffected by reset.)

Prescaler

TCR2	TCR1	TCR0	Result
0	0	0	÷1
0	0	1	÷2
0	1	0	÷4
0	1	1	÷8

TCR2	TCR1	TCR0	Result
1	0	0	÷16
1	0	1	÷32
1	1	0	÷64
1	1	1	÷128

4.9.3 M6805 HMOS Family Timer

The timer block diagram for these family members is shown in Figure 4-8. This timer consists of an 8-bit software programmable counter (timer data register, TDR) which is decremented towards zero by a clock input from a prescaler. The prescaler clock input is received either from the TIMER pin via an external source or from the internal $\phi 2$ of the MCU. The actual clock input to the prescaler is determined by a mask option when the MCU is manufactured.

The mask option allows the prescaler to be triggered either directly from the external TIMER pin or from a gated $\phi 2$ internal clock. When $\phi 2$ signal is used as a clock source, it is only applied whenever the TIMER pin is a logical high. This allows the user to perform a pulse width measurement of the TIMER pin input pulse. In order to provide a continuous $\phi 2$ input to the prescaler in this configuration, it is only necessary to connect the TIMER pin to V_{CC}.

The prescaler divide ratio is selected by a mask option which is determined when the MCU is manufactured. This option allows the TDR to be triggered by every clock input to the prescaler (2^0), by the 128th clock input to the prescaler (2^7), or by any other power of two in between.

The TDR (8-bit counter) may be loaded under program control and is decremented towards zero by each output from the prescaler. Once the TDR has decremented to zero, it sets bit 7 of the timer control register (TCR) to generate a timer interrupt request. Bit 6 of the TCR can be software set to inhibit the timer interrupt request, or software cleared to pass the interrupt request to the processor, provided the I bit is cleared. Since the 8-bit counter (TDR) continues to count (decrement) after falling through $FF to zero, it can be read any time by the processor without disturbing the count. This allows a program to determine the length of time since a timer interrupt has occurred without disturbing the counting process. Once the processor receives the timer interrupt, the MCU responds by saving the present CPU state on the stack, fetching the timer vector, and executing the interrupt routine. The processor is sensitive to the level of the timer interrupt request line; therefore, if the interrupt is masked (I bit set), bit 7 of the TCR may be cleared by the timer interrupt service routine without generating an interrupt. When servicing a timer interrupt, bit 7 of the TCR must be cleared by the timer interrupt service routine in order to clear the timer interrupt request.

At power up or reset, the prescaler and TDR (8-bit counter) are initialized with all logical ones, TCR bit 7 is cleared, and TCR bit 6 is set.

NOTE

The above description does not fully apply to EPROM members of the M6805 HMOS/MC146805 CMOS Family (or the MC6805R3 and MC6805U3). This is because EPROM MCUs use TCR bits 0-5 to select prescaler output divide ratio, determine clocking source, and clear the prescaler. EPROM versions may also be programmed, via the MOR, to allow the prescaler to be software programmed.

4.10 ANALOG-TO-DIGITAL (A/D) CONVERTER

The MC6805R2 MCU and MC68705R3 EPROM MCU both have an 8-bit A/D converter implemented on-chip. This A/D converter uses a successive approximation technique, as shown in Figure 4-9. Up to four external analog inputs, via port D, may be connected to the A/D converter through a multiplexer. Four internal analog channels may be selected for calibration purposes (V_{RH}, V_{RL}, $V_{RH/2}$, and $V_{RH/4}$). The accuracy of these internal channels will not necessarily meet the accuracy specifications of the external channels.

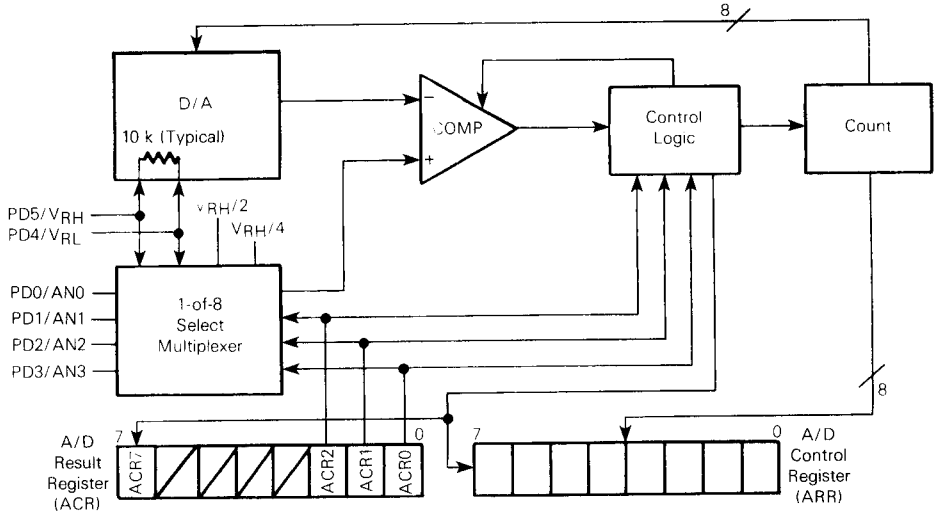


Figure 4-9. A/D Block Diagram

The multiplexer selection is controlled by the A/D control register (ACR) bits 0, 1, and 2; see Table 4-3. This register is cleared during any reset condition.

Table 4-3. A/D Input Multiplexer Selection

A/D Control Register			Input Selected	A/D Output (Hex)		
ACR2	ACR1	ACR0		Min	Typ	Max
0	0	0	AN0			
0	0	1	AN1			
0	1	0	AN2			
0	1	1	AN3			
1	0	0	V_{RH}*	FE	FF	FF
1	0	1	V_{RL}*	00	00	01
1	1	0	$V_{RH/4}$*	3F	40	41
1	1	1	$V_{RH/2}$*	7F	80	81

*Internal (Calibration) levels

Whenever the ACR is written, the conversion in progress is aborted, the conversion complete flag (ACR bit 7) is cleared, and the selected input is sampled and held internally.

The converter operates continuously using 30 machine cycles (including a 5-cycle sample time) to complete a conversion of the sampled analog input. When conversion is complete, the digitized sample or digital value is placed in the A/D result register (ARR), the conversion complete flag is set, the selected input is sampled again, and a new con-

version is started. Conversion data is updated during the part of the internal cycle that is not used for a read. This ensures that valid, stable data is continuously available after initial conversion.

NOTE

Negative transients on any analog lines during conversion will result in an erroneous reading.

The A/D is ratiometric. Two reference voltages (V_{RH} and V_{RL}) are supplied to the converter via port D pins. An input voltage greater than V_{RH} converts to $FF and no overflow indication is provided. For ratiometric conversions, the source of each analog input should use V_{RH} as the supply voltage and be referenced by V_{RL}.

4.11 PHASE-LOCK-LOOP (PLL)

4.11.1 General

The MC6805T2 MCU contains (in addition to the normal ROM, RAM, timer, and I/O functions) a phase-lock-loop (PLL). This feature, not normally found in an MCU, may be used in applications ranging from television tuner control to public service scanner radios.

By providing a PLL which is part of the on-chip MCU circuitry, the functions of frequency control and front panel indication are easily attained. The MC6805T2 contains sufficient ROM and RAM for a program allowing for controlling all the necessary television channels currently used, plus a display showing the channel number.

Figure 4-10 contains a block diagram of a PLL system in an rf synthesizer and Figure 4-11 shows the on-chip MC6805T2 components. As shown, the system components internal to

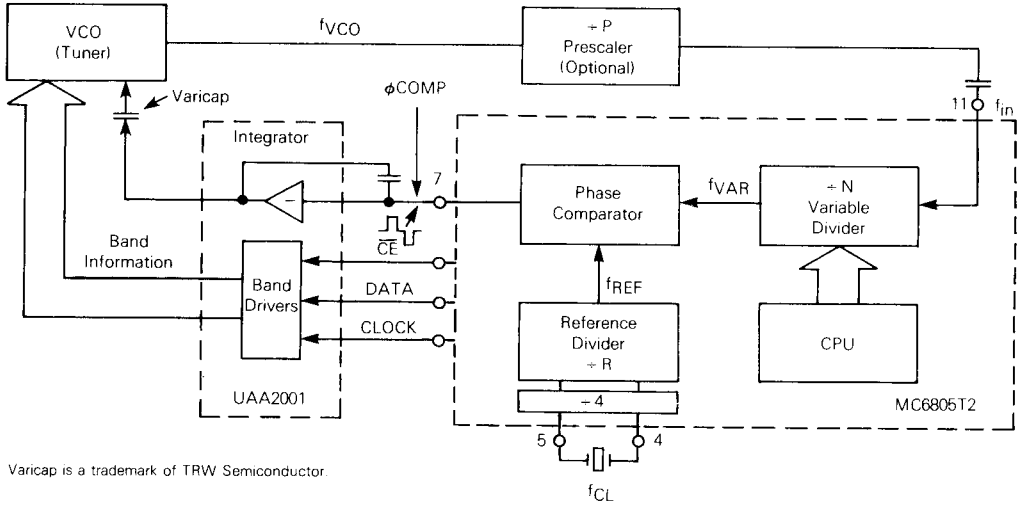


Figure 4-10. Phase-Lock-Loop in an rf Synthesizer

the MC6805T2 MCU contain: a 14-bit binary variable divider (÷ N), a fixed 10-stage reference divider (÷ R), a digital phase and frequency comparator with a three-state output, and circuitry to avoid "back-lash" effects in phase lock condition. External to the MCU, a suitable high-frequency prescaler (÷ P) and an active integrator loop filter plus a VCO rounds out the system.

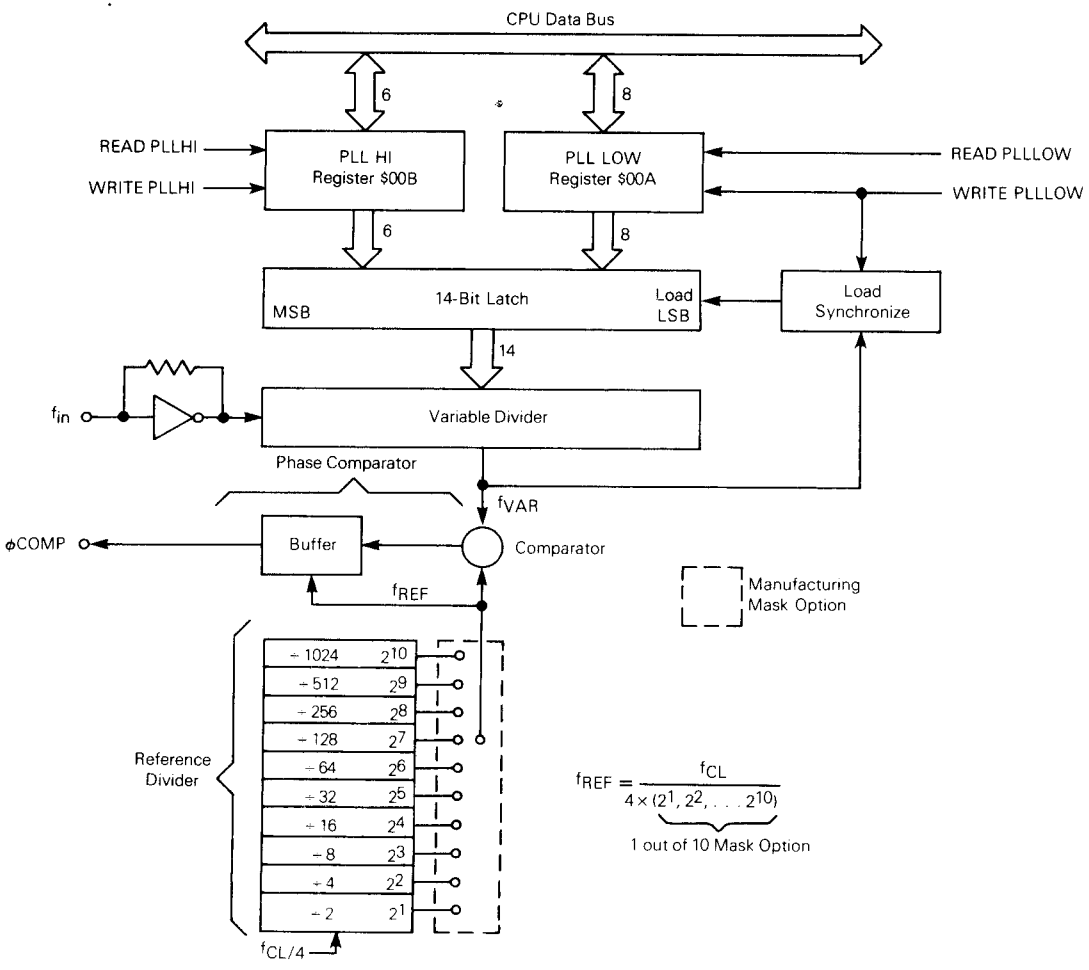


Figure 4-11. MC6805T2 PLL Block Diagram

4.11.2 Reference Divider

Refer to Figure 4-11. This 10-stage binary counter generates a reference frequency which is applied as a constant reference frequency to a phase comparator circuit. The reference divider is mask programmable, thus, allowing the user a choice of reference frequency at the time of manufacture.

4.11.3 Variable Divider

The variable divider (shown in Figure 4-11) is a 14-bit binary down counter which communicates with the CPU via two read/write registers located at address $00A, for the LS byte, and $00B, for the MS byte. The upper two bits in register $00B, always read as logical "1s". When the variable divider count has reached zero, a preset pulse, f_{VAR}, is generated. The f_{VAR} is applied to the phase comparator circuit together with the constant frequency f_{REF} signal. The phase/frequency difference between the two signals results in an error signal output (ϕ COMP, pin 7) which is used to control the VCO frequency. In addition, the f_{VAR} signal is also used to reload the 14-bit divider latch as shown in Figure 4-11.

Data transfers from registers $00A and $00B to the latch occur outside the preset time and only during a write operation performed on register $00A. For example, a 6-bit data transfer to register $00B is only transferred to the variable divider if followed by a write operation to register $00A. Figure 4-12 shows a typical error free manipulation of the 14-bit data in the fine tuning operations.

```
FTUP    LDA     PLLLOW
        INCA                    Check if LS Byte = $FF (Reg $00A)
        BNE     TT1             If Not Increment Only LS Byte
        INC     PLLLOW          Increment MSB (Reg $00B) Before LSB
TT1     INC     PLLLOW
        •
        •
        •
FTDWN   TST     PLLLOW          Check if LS Byte = $00
        BNE     TT2             If Not Decrement Only LS Byte
        DEC     PLLHI           Decrement MSB Before LSB
TT2     DEC     PLLLOW
        •
        •
        •
```

Figure 4-12. Typical Fine Tune Software Example

The use of the 14-bit latch synchronizes the data transfer between two asynchronous systems, namely, the CPU and the variable divider.

At power-up reset both the variable divider and the contents of the PLL registers are set to logical "1s".

The variable frequency input pin, f_{in}, is self biased requiring an ac coupled signal of about 0.5 V. The input frequency range of f_{in} allows the device, together with a suitable prescaler, to cover the entire TV frequency spectrum.

4.11.4 Phase Comparator

The phase comparator compares the frequency and phase of f_{VAR} and f_{REF}, and according to the phase relationship generates a three-level output (1, 0, or Hi-Z), ϕCOMP, as shown in Figure 4-13. The output waveform is then integrated, amplified, and the resultant dc voltage is applied to the voltage controlled oscillator.

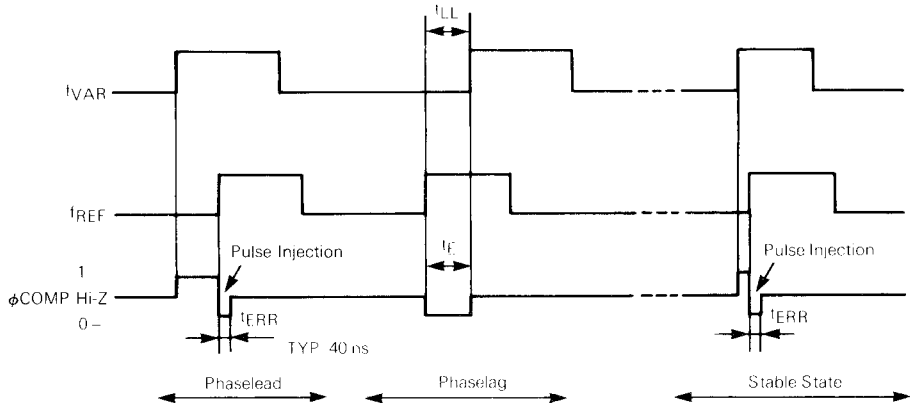


Figure 4-13. Phase Comparator Output Waveform

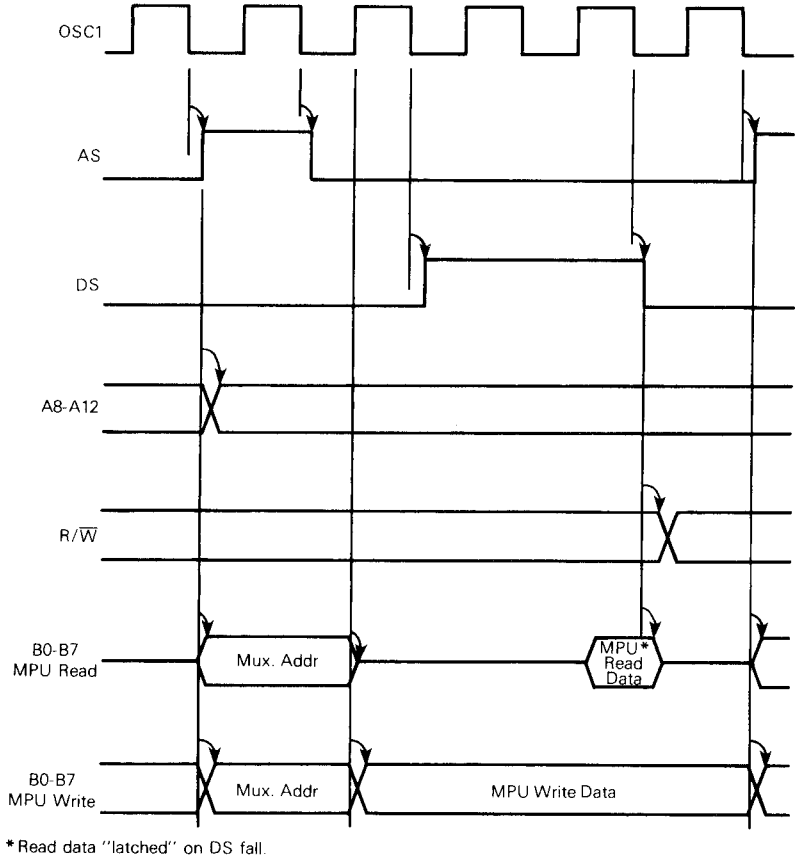
In practice, a linear characteristic around the steady-state region can not be achieved due to internal propagation delays. Thus, phase comparators exhibit non-linear characteristics and for systems which lock in phase, this results in a "backlash" effect—creating sidebands and FM distortion. To avoid this effect, a very short pulse is injected periodically into the system. The loop, in turn, attempts to cancel this interference and in so doing brings the phase comparator to its linear zone.

4.12 MC146805E2 MICROPROCESSOR (MPU) EXTERNAL BUS DESCRIPTION

The MC146805E2 CMOS MPU does not contain on-chip non-volatile memory; however, by using the external multiplexed address-then-data bus, additional memory and peripherals may be added. In order to conserve pins, the MC146805E2 multiplexes the data bus with the eight lower address bits. The lower address bits appear on the bus first and are valid prior to the falling edge of address strobe (AS). Data is then transferred during data strobe (DS) high. The MC146805E2 latches read data (R/$\overline{W}$ is high) on the falling edge of DS.

The MC146805E2 bus timing is generated from the waveform at the OSC1 input. Figure 4-14 shows the relationship of the MC146805E2 bus timing to the OSC1 input. Because the MC146805E2 is a completely static device, it may be operated at any frequency below its maximum (1 MHz bus) rate. Since generating the timing specifications for all of the possible frequencies is impossible, Figure 4-14 can be used to estimate the effects on bus timing for the oscillator frequency (f_{OSC}). For instance, decreasing f_{OSC} increases

the multiplexed address hold time since the multiplexed bus does not switch until a half OSC1 cycle after AS goes low. On the other hand, the required read data hold time is not a function of f_{osc}.



*Read data "latched" on DS fall.

Figure 4-14. OSC1 to Bus Transitions

CHAPTER 5
HARDWARE APPLICATIONS

5.1 INTRODUCTION

When the initial microprocessors appeared in the marketplace, the actual on-chip circuitry was extremely limited. This required the use of a large number of devices just to support the actual processor. However, as technology progressed much of the support hardware was included on-chip with the processor. The M6805 HMOS/M146805 CMOS Family now includes standard on-chip features such as: an oscillator, ROM, RAM, timer, and a wide variety of I/O devices. Combining these standard features with other features such as analog-to-digital conversion, phase-lock-loop, etc. onto a single chip simplifies system design efforts while reducing production costs.

This chapter contains discussions and examples of applications which describe how some of these on-chip hardware features may be used and enhanced. The first paragraphs provide discussions of some of the features, whereas, the latter paragraphs describe application examples which perform real tasks.

The evaluation ROM devices for each member of the M6805 HMOS/M146805 CMOS Family contain evaluation examples which can be used to better understand the device. Many of the evaluation examples have been used to perform real tasks, and many of these are described in various Motorola application notes.

One paragraph of this chapter is dedicated to CMOS design considerations. This discussion highlights the somewhat different design considerations required when designing a system using CMOS.

5.2 I/O EXPANSION

The M6805 HMOS/M146805 CMOS Family devices may require interfacing with other peripherals. Several representative descriptions are provided in this paragraph, all of which are in general terms except for the MC146805E2 MPU.

5.2.1 MC146805E2 Microprocessor Unit (MPU)

The MC146805E2 MPU is the only member of the M6805 HMOS/M146805 CMOS Family that has no on-chip ROM; however, it does use a multiplexed address/data bus to interface with external memory or peripherals. Multiplexed bus memory peripheral interfacing techniques are discussed below. In addition, the MC146805E2 can also be interfaced with non-multiplexed bus memory peripherals, and this technique is also discussed

below. In some applications it is necessary to interface the MC146805E2 with peripherals which require longer access times ("slow memory"). A discussion of this technique is also included as part of this paragraph.

5.2.1.1 INTERFACING MULTIPLEXED BUS MEMORY WITH PERIPHERALS. A multiplexed bus device is characterized by an address latch and an output enable signal. The address latch captures the lower eight bits of the address from the multiplexed bus, and the output enable signal is used to determine when data can be safely transferred. The circuit in Figure 5-1 illustrates a typical multiplexed bus interface. This figure provides a detailed representation of the minimum circuit required to use the CBUG05 debug monitor which is contained in the MCM65516 2K x 8 CMOS ROM. A complete description of the CBUG05 can be found in Motorola Application Note AN-823.

The circuit shown in Figure 5-1 consists entirely of CMOS devices. The system could be expanded easily by adding CMOS RAM, ROM, EPROM, or peripherals such as MC146818 real time clock or the MC146823 CMOS peripheral interface.

The MCM65516 ROM uses the AS signal from the MC146805E2 to latch the multiplexed address and the DS signal to transfer data. The data transfer direction is controlled by the R/$\overline{W}$ signal. Since the MCM65516 ROM is a read-only device, R/$\overline{W}$ is used together with A11 and A12 to provide the chip select and enable lines to ensure that an inadvertant write does not cause a bus conflict. The chip enable and select lines on the MCM65516 are mask programmable as either active high or active low; therefore, no external address decoding is necessary in this example. A second example is discussed below which uses an MC146823 CMOS peripheral interface to emulate the C and D ports of the MC146805G2.

5.2.1.2 INTERFACING NON-MULTIPLEXED BUS MEMORY WITH PERIPHERALS. Since the majority of existing memory and peripheral devices use a non-multiplexed bus, an interface with the MC146805E2 can be relatively simple. The main difference between multiplexed and non-multiplexed memory and peripheral devices is the absence of an address latch in non-multiplexed bus devices. The non-multiplexed bus devices require that all address lines be valid for the entire cycle. In order to provide this valid address to a non-multiplexed memory or peripheral device, the MC146805E2 multiplexed bus can be demultiplexed merely by adding an external address latch. This is illustrated in Figure 5-2 which uses an MC74HC373 to demultiplex the bus for the non-multiplexed MCM27C16 EPROMs. The multiplexed address lines (B0-B7 of the MC146805E2) are latched in the MC74HC373 by the falling edge of address strobe (AS). They remain latched until AS goes high. The emulator shown in Figure 5-2 is further discussed in the Emulating The MC146805G2 MCU paragraph.

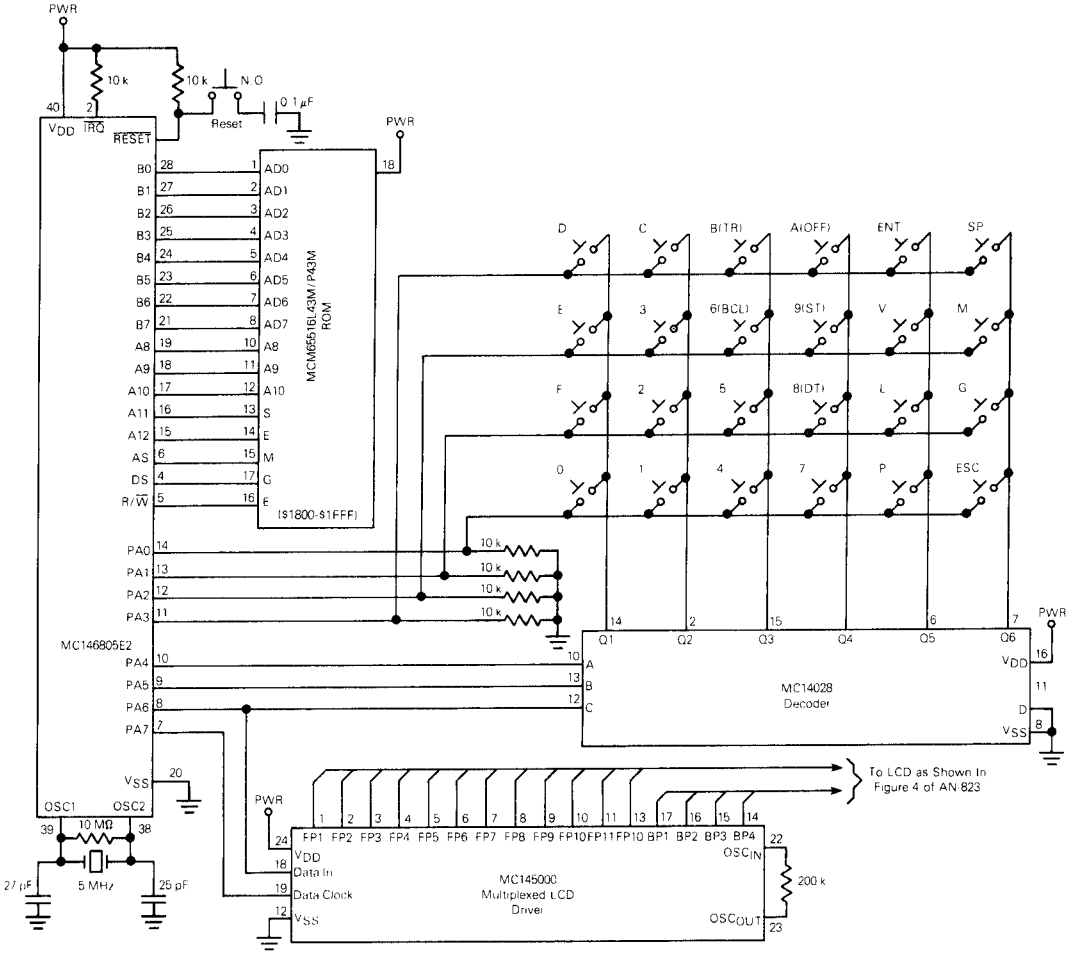
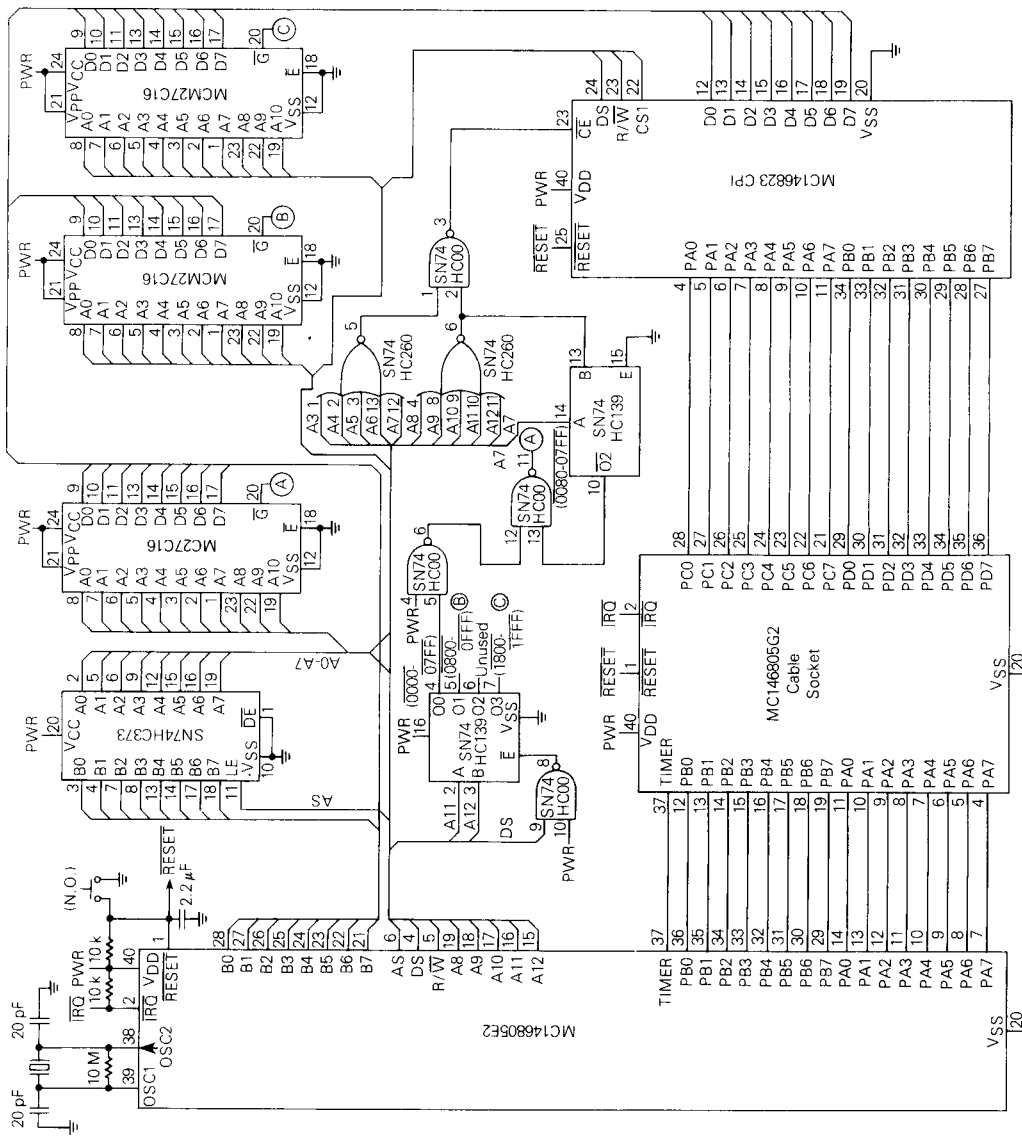


Figure 5-1. CBUG05 Debug Monitor Minimum CMOS Only System, Schematic Diagram



5-2. MC146805G2 Emulator Schematic Diagram

5.2.1.3 INTERFACING WITH SLOW MEMORY AND PERIPHERAL DEVICES. At times, it is desirable to use memory or peripheral devices which require both chip enable and output enable. In these devices, the access time is calculated from when chip enable is valid, whereas, output enable simply opens the gates to the external bus.

The emulator circuit of Figure 5-2 shows an interface with an MC146805E2 and an MCM27C16; however, slow, single-supply SC682716 EPROMs could be used. Note that the chip enable ($\overline{E}$) of the MCM27C16 EPROM is continuously held low. This allows the address to be gated by using the MC146805E2 DS signal to generate the output enable ($\overline{G}$). The DS signal is actually used in decoder SN74HC139 to generate three $\overline{G}$ inputs, one for each EPROM. In this type of interface, the output enable time is the limiting factor and it is typically much shorter than the access time. On most devices power consumption increases when this type of interface is used.

5.2.1.4 EMULATING THE MC146805G2 MCU. The circuit shown in Figure 5-2 illustrates the use of each of the three interfacing techniques to allow the MC146805E2 MCU to provide real-time emulation for the MC146805G2 Microcomputer (MCU). In the circuit of Figure 5-2 all devices are CMOS; however, the actual MC146805G2 power consumption will be approximately 20% of that consumed by the emulator. More information concerning MC146805G2 emulation, as well as other MCUs, is contained in Motorola Application Note AN-853.

5.2.2 Single-Chip Microcomputer (MCUs)

The increased circuit density of single-chip MCUs greatly reduces the need for additional hardware external to the MCU itself. By combining as much I/O as needed on a single integrated circuit device, the cost is lowered and reliability increased. Since the probability of system failure increases with each added system component, system reliability increases as a result of more system components being designed into a single-chip MCU.

The single-chip MCUs which are part of the M6805 HMOS/M146805 CMOS Family continue to grow in order to fill the diversity of I/O needs in the controller market; however, some applications may require I/O functions that are not yet included, or are unsuitable for inclusion, as part of the single-chip MCU. Whatever the reason, the MCU must provide a means for using external devices in a system.

All M6805 HMOS/M146805 CMOS MCUs contain programmable bidirectional I/O lines; however, the actual number of these I/O lines may vary between specific MCUs. In all cases an external interface may be simulated by properly manipulating these lines. The MC145000 LCD driver interface described in Chapter 3 is an example of such an interface.

More complicated interfaces may be simulated as shown in Figure 5-3. This example combines the MC146818 real-time clock with an M6805 HMOS/M146805 CMOS Family MCU. Eleven I/O lines are used in this interface to provide the multiplexed bus required by the MC146818. If an interface requiring more lines were used, a peripheral interface adapter (PIA), latch, or input buffer could be added to increase the effective number of I/O lines.

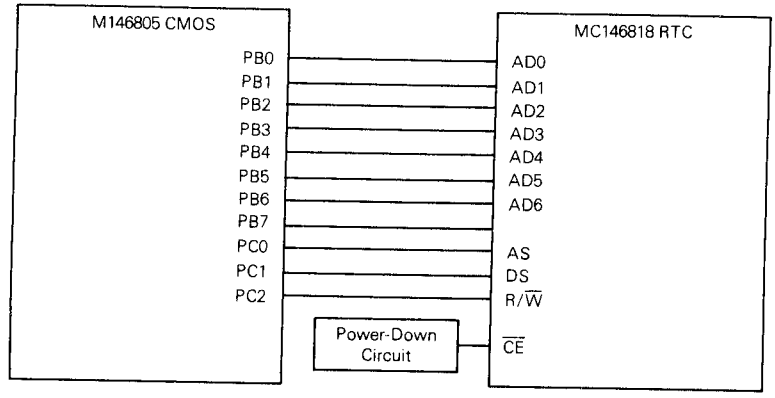


Figure 5-3. MCU Interface With Multiplexed Bus Peripheral

All MCU interfaces require some amount of software overhead. The software requirement for the MC146818 interface is illustrated in Figure 5-4. The MC146818 multiplexed bus requires that signals and transitions occur in specific order. The software of Figure 5-4 guarantees that these timing requirements are met.

```
*     INITIALIZATION OF I/O LINES
PORTB   EQU   1
PORTC   EQU   2
DDRB    EQU   5
DDRC    EQU   6
AS      EQU   0
DS      EQU   1
RW      EQU   0
ADDR    RMB   1          ADDRESS OF BYTE TO BE ACCESSED
INIT    CLR   PORTC      CLEAR PORT C OUTPUT DATA LATCH
        CLR   PORTB      CLEAR PORT B OUTPUT DATA LATCH
        LDA   #$FF       CONFIGURE PB0-PB7 AS OUTPUT
        STA   DDRB
        LDA   #7         CONFIGURE PC0-PC2 AS OUTPUTS
        STA   DDRC
        RTS              RETURN
*       READ MC146818
READ    BSET  RW,PORTC   FORCE R/W AND AS HIGH
        BSET  AS, PORTC
        LDA   ADDR       PRESENT MUXED ADDRESS
        STA   PORTB
        BCLR  AS,PORTC   FORCE AS LOW TO LATCH MUXED ADDRESS
        CLR   DDRB       CONFIGURE PORT B AS INPUTS
        BSET  DS,PORTC   READ DS HIGH
        LDA   PORTB      READ DATA
        STA   DATA       STORE DATA IN RAM FOR LATER USE
        BRA   INIT       REINITIALIZE LINES AND RETURN
*       WRITE MC146818
WRITE   BSET  AS,PORTC   FORCE AS HIGH
        LDA   ADDR       PRESENT MUXED ADDRESS
        STA   PORTB
        BCLR  AS,PORTC   FORCE AS LOW TO LATCH MUXED ADDRESS
        LDA   DATA       PRESENT WRITE DATA
        STA   PORTB
        BSET  DS,PORTC   TOGGLE DS TO LATCH DATA INTO MC146818
        BCLR  DS,PORTC
        BRA   INIT       REINITIALIZE LINES AND RETURNS
```

Figure 5-4. MCU to MC146818 Interface Software

5.3 PERIODIC WAKE-UP FROM WAIT MODE

The timer may be used to generate a signal which causes a member of the M146805 CMOS Family to exit from the WAIT mode. The WAIT instruction (like the STOP instruction) places the MPU or MCU into a low-power mode which may be exited by using either a reset or an external interrupt; however, unlike the STOP mode, the WAIT mode does not disable the timer. In the WAIT mode, the timer interrupt can also cause the processor to exit the WAIT mode and begin execution of the program pointed to by the timer wait interrupt vector. This feature of using the timer interrupt to periodically "wake-up" the processor, is extremely useful in systems that require the lowest possible power consumption and at the same time require infrequent processor control. In these systems, the processor is "put to sleep" and periodically "awakened" by the timer interrupt.

The example shown in Figure 5-5 is similar to the keyscan example described in Chapter 3. The main difference between the examples is that the keyscan routine uses the external interrupt to exit the STOP mode; whereas, the example of Figure 5-5 uses the internal bus frequency (clock) to exit the WAIT mode. Power comsumption using the WAIT mode is slightly higher than the STOP mode. This is because the timer is active and consumes power during the WAIT mode. Also, in the example of Figure 5-5, the data at ports A and B are compared (using the SUB instruction) and the difference is outputted at port C every 298 internal clock cycles. No external hardware, except as necessary to guard against possible CMOS latch-up, is necessary.

```
PORTA   EQU   0
PORTB   EQU   1
PORTC   EQU   2
DDRC    EQU   6
TDR     EQU   8
TCR     EQU   9
RESET   CLR   PORTC   INITIALIZE PORT C OUTPUT DATA LATCH
        LDA   #$FF    CONFIGURE PORT C AS OUTPUT PINS
        STA   DDRC
        LDA   #$20    DISABLE TIMER
        STA   TCR
PAUSE   LDA   #$F9    LOAD TIMER DATA REGISTER
        STA   TDR
        LDA   #$00    ENABLE TIMER INTERRUPT AND FOR INTERNAL INPUT
        STA   TCR
        WAIT          ENTER LOW POWER WAIT MODE
        BRA   PAUSE   RETURN HERE AFTER INTERRUPT IS SERVICED

TWIRQ   BCLR  7,TCR   CLEAR TIMER INTERRUPT
        LDA   PORTA   READ PORT A REGISTER INPUT DATA
        SUB   PORTB   FIND DIFFERENCE
        STA   PORTC   OUTPUT DIFFERENCE TO PORT C
        RTI           RETURN FROM INTERRUPT
```

Figure 5-5. Timer Wait Mode Exit Software

5.4 INTERRUPTS

The $\overline{IRQ}$ or $\overline{INT}$ pins on the M6805 HMOS/M146805 CMOS Family may be used in many different interrupt-type applications. An interrupt is used either as a request by a peripheral for MPU/MCU service or as a flag to the MPU/MCU which indicates the occurrence of some event. The following paragraphs provide descriptions in which the interrupt line is used as a flag for the processor.

5.4.1 Exiting From STOP Mode

The STOP instruction is used in the M146805 CMOS Family to enter a low-power operating mode. In most MPU/MCU applications, there are intervals in which no processing, except to wait for an event to occur, is required. The example described in Chapter 3 of the 4 × 4 keypad interface is a typical example. One of the features of this keypad interface is that the processor enters the low-power STOP mode and remains there until a valid keypad switch closure occurs. When a key is depressed, the $\overline{IRQ}$ line is pulled low. This causes the processor to exit the STOP mode and enter the interrupt service routine. The interrupt service routine polls (scans) the keypad rows and columns to determine which key was depressed. After the depressed key location is verified, the interrupt service routine is exited. Processing then continues at the instruction that follows the STOP instruction that was last executed.

The location of the depressed keypad key may be used in conjunction with a jump table to initiate the execution of any one of a number of routines, or a conversion table to translate the key location into a value or a character. When keypad input is required, all that need be done to accept it is to execute the STOP instruction. The interrupt mask is automatically cleared by the STOP instruction, the depressed keypad key causes an interrupt, and the depressed key location is returned in the accumulator.

5.4.2 60 Hz Interrupt For Time of Day Clock

By attenuating the 60 Hz standard 110 Vac power line and inputting this signal into the M6805 HMOS/M146805 CMOS Family MCU as shown in Figure 5-6, a time of day clock can be controlled. Since the 60 Hz line voltage is constantly monitored and regularly corrected by the power company, its average frequency is maintained as close to 60 Hz as possible. This accurate frequency and ready availability make the standard power line ideal for accurate timekeeping. The circuit shown in Figure 5-6 first attenuates the line voltage to a level that meets the maximum input voltage specification of the $\overline{INT}$ pin. The capacitor serves to eliminate dc from the $\overline{INT}$ pin input and the diodes limit the peak-to-peak voltage. A Schmitt trigger, which is internal to the MPU/MCU, ensures that noise does not generate false interrupts. The diodes clamp the input ac voltage to ensure that it does not exceed the rated peak-to-peak input, while, at the same time, providing 60 falling edges per second. Thus, the MPU/MCU enters the interrupt service routine 60 times per second.

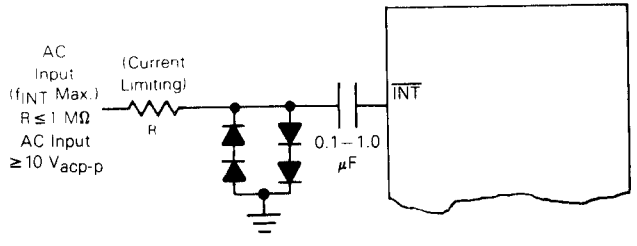


Figure 5-6. Typical Zero Crossing Interrupt Circuit Schematic Diagram

The software illustrated in Figure 5-7 is necessary to count the number of interrupts and convert that number to seconds, minutes, and hours. Also included in the software of Figure 5-7 is a procedure that initializes the clock.

```
PAGE  001  CLOCK   .SA:0

00001                          *
00002                          *  RAM VARIABLES
00003                          *
00004A 0064                           ORG     $64       ON CHIP RAM
00005                          *
00006A 0064     0004      A CLOCK     RMB     4         TIME-OF-DAY CLOCK
00007           0064      A HRS       EQU     CLOCK     HOURS IN CLOCK
00008           0065      A MINS      EQU     CLOCK+1   MINUTES IN CLOCK
00009           0066      A SECS      EQU     CLOCK+2   SECONDS IN CLOCK
00010                          *
00011A 0080                           ORG     $80
00012                          *
00013                          *  MAXIMUM COUNTER VALUES FOR TIME OF DAY
00014                          *  CHANGE 13 TO 24 FOR 24 HOUR CLOCK
00015                          *  CHANGE JIFFIES FROM 60 TO 50 FOR 50 Hz OPERATION.
00016                          *
00017A 0080     0D        A MODULO FCB     13        HOURS
00018A 0081     3C        A        FCB     60        MINUTES
00019A 0082     3C        A        FCB     60        SECONDS
00020A 0083     3C        A        FCB     60        JIFFIES  (SIXTIETH)
00021                          *
00022                          *  TICK --- 60 HZ INTERRUPT FOR TIME OF DAY CLOCK
00023                          *
00024A 0084 AE 03         A TICK    LDX     #3        BEGIN AT LSB OF CLOCK
00025A 0086 6C 64         A TICK2   INC     CLOCK,X   BUMP THIS DIGIT
00026A 0088 E6 64         A         LDA     CLOCK,X   SEE IF IT OVERFLOWED
00027A 008A E1 80         A         CMP     MODULO,X  COMPARE WITH MODULO VALUES
00028A 008C 25 07      0095         BLO     CLKOUT    DONE IF LOWER THAN MODULO
00029A 008E 6F 64         A         CLR     CLOCK,X   RESET THIS COUNTER AND
00030A 0090 5A                      DECX              GO TO NEXT WITH OVERFLOW
00031A 0091 2A F3      0086         BPL     TICK2     WHILE NOT AT HOURS COUNTER
00032A 0093 3C 64         A         INC     HRS       REMOVE FOR 24 HOUR CLOCK
00033A 0095 80            CLKOUT RTI
00034                          *
00035                          *  INTERRUPT VECTOR
00036                          *
00037A 07FA                           ORG     $7FA
00038                          *
00039A 07FA     0084      A           FDB     TICK      12/24 HOUR CLOCK VECTOR
00040                          *
00041                                 END
```

Figure 5-7. Time of Day Clock Software Listing

5.5 CMOS DESIGN CONSIDERATIONS

Digital devices may be implemented in any number of processing technologies, and, as shown in Table 5-1, each processing technology has its advantages and disadvantages. For applications requiring low power consumption, CMOS has been the dominant technology; however, until recently, CMOS could not match the speeds found in other processes. With the advent of silicon-gate CMOS and the emerging high-density CMOS (HCMOS) processes, CMOS technology is not only replacing NMOS in many designs, but is responsible for a whole new field of products.

The M146805 CMOS Family of MPU/MCUs, and the M74HCXX Family of CMOS interface logic, combine the CMOS low power consumption with the speeds of NMOS/HMOS and TTL. However, since CMOS requires a larger silicon area than NMOS or HMOS, it is more expensive.

Table 5-1. Comparison of Processing Techniques

Process	Advantages	Disadvantages	Comments
TTL	Fast with high drive capability.	Consumes more power than NMOS/HMOS, CMOS or HCMOS.	TTL has a high drive capability compared to NMOS and is used in interface devices.
NMOS/HMOS	Fast, high density, inexpensive, consumes less power than TTL.	Restrictive voltage supply requirements for portable applications.	HMOS is high-speed, high-density version of NMOS.
CMOS	Consumes very little power, uses a wider voltage range than NMOS, Si-gate versions are as fast as NMOS.	More expensive than NMOS; metal-gate versions are slow.	CMOS densities are approaching those of NMOS.
HCMOS	Fast, dense, inexpensive, low-power.	Consumes more power than CMOS but less than NMOS.	Currently available in 74HCXX series of CMOS interface logic. Combines NMOS and CMOS devices to get the best of both processes.

The following two paragraphs provide a design criteria discussion that should be considered in order to use CMOS effectively and reliably. These discussions include: (1) factors that contribute to CMOS power consumption and how the effects of these factors can be reduced, and (2) the phenomenon of CMOS latch-up and how it can be avoided.

5.5.1 Power Consumption

The two factors which greatly affect CMOS power consumption are supply voltage and operating frequency. Reducing the supply voltage (V_{DD}) proportionally reduces power consumption since the EI product is lower. A "side effect" of lowering the supply voltage is a reduction in the maximum operating frequency of the device. This is the result of reduced internal drive caused by lowered V_{DD}.

The power consumption of a CMOS device is primarily affected by capacitive loading rather than resistive loading as for HMOS or NMOS. Each CMOS cell is basically composed of two complementary transistors (a P channel and an N channel), and, in the steady state, only one transistor is turned on. The active P-channel transistor sources current when the output is a logic high, and presents a high impedance when the output is a logic low. Thus, the overall result is extremely low power consumption because there is no power loss through the active P-channel transistor. Since only one transistor is turned on during the steady state, power consumption is determined by leakage currents.

During a transition, both transistors pass through the active regions of their operating characteristics. The actual time spent simultaneously in these active regions directly affects the power consumption. The higher the operating frequency, the more time is spent in these simultaneous active regions, thus, higher power consumption. By reducing the number of transitions within the CMOS device, power consumption can be reduced. Also, since power consumption depends upon the time spent in transition, the rise and fall times of the signals should be as fast as possible. This can be accomplished by minimizing capacitive loading. It is important to note that although slower operating frequencies have longer rise and fall times, the effects of this additional time are generally negligible when compared to effects of reducing operating frequency.

5.5.2 CMOS Latch-Up

Due to the required layout of CMOS devices, a virtual semiconductor controlled rectifier (SCR) may be formed when an input exceeds the supply voltage. The SCR that is formed by this high input causes the device to become "latched" in a mode that may result in excessive current drain and eventual destruction of the device. Although the M146805 CMOS Family is implemented with input protection diodes, care should be exercised to ensure that the maximum input voltage specification is not exceeded. Some systems may require that the CMOS circuitry be isolated from voltage transients; others may require no additional circuitry.

5.6 32-kHz OSCILLATOR

The M146805 CMOS Family can operate at frequencies down to dc; however, the on-chip oscillator cannot be used with series type crystals. Because low frequency crystals are series type, additional circuitry is necessary. To generate this clock input, an external oscillator similar to that shown in Figure 5-8 is required. The MC14069 CMOS hex inverter was chosen for this circuit because of its low power consumption, and its ability to operate in the linear region with reasonable stability. Only two resistors are required for the oscillator: bias resistor R1 ensures linear operation and R2 provides current limiting protection for the crystal. Two load capacitors (C1 and C2) ensure proper loading plus correct start-up frequency. Variable capacitor C1 also allows limited tuning of the output frequency.

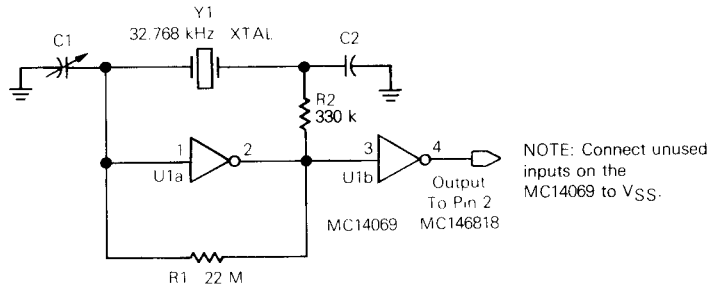
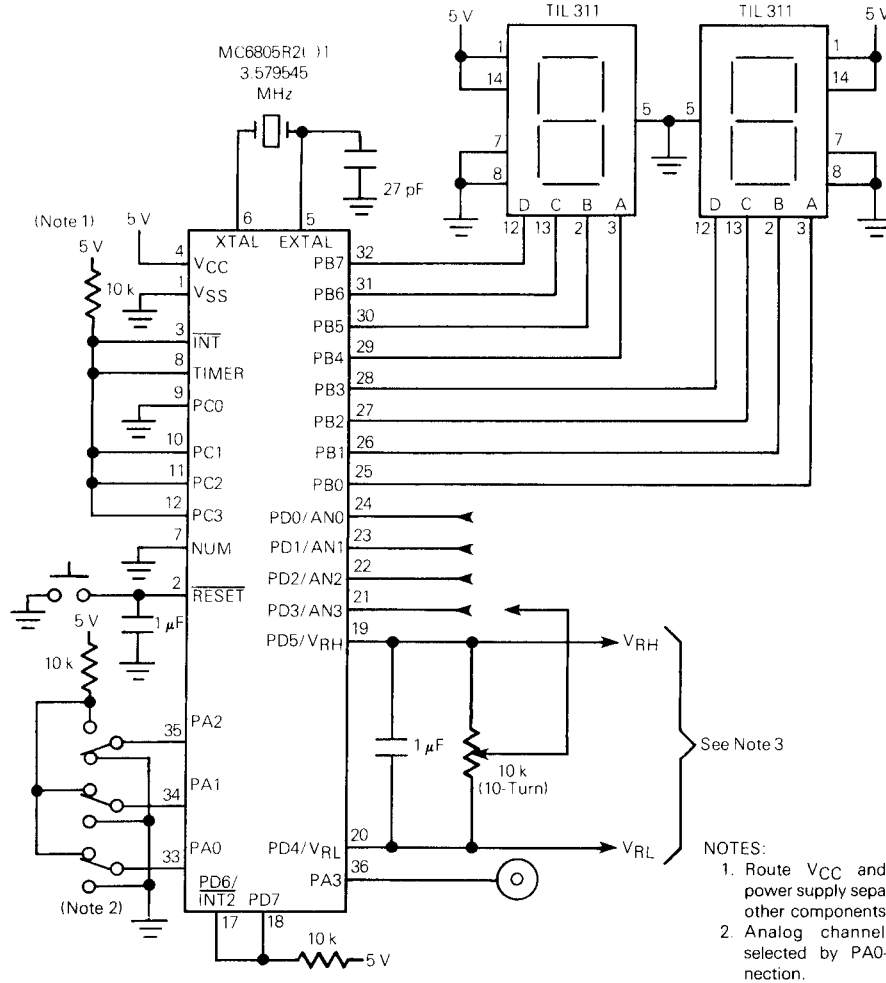


Figure 5-8. 32.768 kHz Square Wave Oscillator Schematic Diagram

The 32-kHz oscillator described above functions properly and exhibits relatively good frequency stability over ambient temperature ranges. However, there is a possibility of minor frequency variations resulting from voltage fluctuation. The 32 kHz oscillator circuit, shown in Figure 5-8, will react only slightly to a decrease in V_{DD} from 5 V down to 3.9 V. The actual change in frequency over this 1.1 V range would be about 0.1 Hz and could result in an error of about 7.9 seconds per month.

With R2 as a 330 k resistor, the oscillator is very sensitive to the values of capacitors C1 and C2, and at times the R2 value must be chosen to match the crystal. With R2 removed or decreased in value to 2 k, the oscillator is less sensitive to C1 and C2; however, it is considerably more prone to frequency changes resulting from voltage variations. For example, with R2 at a value of 2 k, a 1.1 volt change in V_{DD} could result in a 1.2 Hz frequency change. This amounts to an error of 87 seconds in a month.

In many cases, either of the above discussed errors (7.9 or 87 seconds) is not acceptable. In these cases, there are two suggestions which might be helpful. The first, and possibly the easiest to implement, is to keep the battery backup voltage equal to V_{DD}. This would require a slightly higher power consumption from the backup battery but the frequency drift would be lessened or eliminated. A second method would be to use a voltage and temperature compensated oscillator to provide a stable 32.768 kHz source. The latter method is more complex and requires more power; however, in systems where accurate time is a requirement, a simple oscillator is not adequate. Higher frequency crystals exhibit similar voltage-frequency drift characteristics even when attached directly to an M146805 CMOS Family device.



NOTES:

1. Route V_{CC} and V_{SS} to power supply separately from other components.
2. Analog channel AN3 is selected by PA0-PA2 connection.
3. V_{RH} and V_{RL} should be furnished by a separate power source (not V_{CC}). The sum of V_{RH} and V_{RL} must not exceed V_{CC}.

Figure 5-9. Stand-Alone A/D Converter Schematic Diagram

5.7 STAND-ALONE ANALOG-TO-DIGITAL CONVERTER (AN-869)

The stand-alone A/D converter shown in Figure 5-9 is configured using an MC6805R2( )1 MCU. The circuit uses three SPDT switches to control the multiplexer selection (a 1-of-8 on-chip select multiplexer which is controlled via the A/D control register from inputs at PA0-PA2). Table 5-2 lists the inputs to the A/D control register which select either the AN0-AN3 inputs or an internal calibration level. The eight bit result of the A/D conversion is output on port B (PB0-PB7). The output on PB3 may be used to indicate that the port B data is valid.

Table 5-2. A/D Control Inputs For Selecting AN0-AN3 and Calibration Channels

Channel	PC2	PC1	PC0
AN0	0	0	0
AN1	0	0	1
AN2	0	1	0
AN3	0	1	1
V_{RH} (Calibration)	1	0	0
V_{RL} (Calibration)	1	0	1
$V_{RH/4}$ (Calibration)	1	1	0
$V_{RH/2}$ (Calibration)	1	1	1

As shown in Figure 5-9, the output of the 10 k 10-turn potentiometer can be used to select a voltage value between V_{RH} and V_{RL}. An input voltage to the selected AN0-AN3 input, which is equal to V_{RH}, converts to FF (full scale) on the LCD. Conversely an input which is equal to V_{RL} converts to 00 on the LCD. Input levels between V_{RH} and V_{RL} provide corresponding indications on the LCD. Figure 5-10 contains the program listing for the A/D conversion routine which is used in the MC6805R2( )1.

5.8 FREQUENCY SYNTHESIZER USING THE MC6805T2L1 (AN-871)

The MC6805T2L1 Microcomputer Unit (MCU) contains seven distinct program modules, one of which is referred to as the frequency synthesis program (synthesizer [PLL05] mode). The synthesizer mode allows the MC6805T2L1 to function in a phase-locked loop (PLL), which controls the output frequency of a variable frequency oscillator (VFO). The program is written in a way that it can be used to synthesize a complete set of TV channels for either Europe, Japan, or USA. The firmware program is located in masked ROM, and automatically takes into account differences in intermediate frequencies, first and last channel numbers, channel spacing, etc., as they exist between the systems used in those countries/lands. The desired mode option configuration is entered by selecting the PLL-( ) country code shown in Table 5-3.

Figure 5-11 provides a schematic diagram of the MC6805T2L1 used in a synthesizer mode configuration (USA selected). All peripheral devices are shown, except for the VCO (tuner) and prescaler (if used).

```
                        *       SAD --- STAND ALONE ANALOG TO DIGITAL CONVERTER
                        *
                        *       IN THIS MODE, THE 68Ø5R2 OPERATES AS AN 8 CHANNEL
                        *       MULTIPLEXED ANALOG TO DIGITAL CONVERTER. INPUT TO THE
                        *       CONVERTER IS VIA THREE CHANNEL SELECT LINES ON PORT A.
                        *       THE DESIRED CHANNEL IS TRANSFERRED TO THE A TO D CONTROL
                        *       REGISTER AND THE PROGRAM WAITS FOR A CONVERSION TO
                        *       COMPLETE. AFTER COMPLETION, THE RESULT REGISTER IS
                        *       TRANSFERRED TO PORT B. AN EXTERNAL END OF CONVERSION BIT
                        *       IS ALSO PROVIDED TO SYNCHRONIZE THE RESULTS ON PORT B.
                        *       THIS BIT (PORT A BIT 3) IS LOW WHEN THE OUTPUT OF PORT B
                        *       IS CHANGING AND HIGH OTHERWISE.
                        *       I/O REGISTER ADDRESSES
                        *
          ØØØØ      A PORTA   EQU     $ØØØ      I/O PORT Ø
          ØØØ1      A PORTB   EQU     $ØØ1      I/O PORT 1
          ØØØ2      A PORTC   EQU     $ØØ2      I/O PORT 2
          ØØØ3      A PORTD   EQU     $ØØ3      I/O PORT 3 (ALSO A/D INPUT STUFF)
          ØØØ4      A DDR     EQU     4         DATA DIRECTION REGISTER OFFSET
          ØØØ8      A TIMER   EQU     $ØØ8      8-BIT TIMER REGISTER
          ØØØ9      A TCR     EQU     $ØØ9      TIMER CONTROL REGISTER
          ØØØA      A MISC    EQU     $ØØA      MISCELLANEOUS REGISTER (INT2 FLAGS)
          ØØØE      A ADCSR   EQU     $ØØE      A TO D CONTROL/STATUS REGISTER
          ØØØF      A RESULT  EQU     $ØØF      A TO D RESULT REGISTER
          Ø04Ø      A RAM     EQU     $Ø4Ø      START OF ON-CHIP RAM AREA
          Ø08Ø      A ZROM    EQU     $Ø8Ø      START OF PAGE ZERO ROM
          Ø7CØ      A ROM     EQU     $7CØ      START OF MAIN ROM AREA
          1ØØØ      A MEMSIZ  EQU     $1ØØØ     MEMORY ADDRESS SPACE SIZE
                        *
                        *       BITS IN VARIOUS CONTROL REGISTERS
                        *
          ØØØ7      A EOC     EQU     7         END OF CONVERSION BIT IN ADCSR
          ØØØ7      A INT2F   EQU     7         INT2 FLAG BIT IN MISC
          ØØØ6      A INT2E   EQU     6         INT2 ENABLE BIT IN MISC
                        *
ØDC5 17 ØØ        A SAD     BCLR    3,PORTA   INITIALIZE OUTPUT INVALID
ØDC7 A6 Ø8        A         LDA     #%1ØØØ    MAKE BIT 3 OUTPUT, 2-Ø INPUTS
ØDC9 B7 Ø4        A         STA     PORTA+DDR
ØDCB A6 FF        A         LDA     #$FF      AND MAKE B ALL OUTPUTS
ØDCD B7 Ø5        A         STA     PORTB+DDR
ØDCF B6 ØØ        A         LDA     PORTA     PICKUP CHANNEL #
ØDD1 A4 Ø7        A         AND     #%111     CLEAR GARBAGE
ØDD3 B7 ØE        A         STA     ADCSR     START CONVERSION
ØDD5 ØF ØE FD ØDD5         BRCLR   EOC,ADCSR,* WAIT FOR IT TO FINISH
          ØDD8      A SADLP   EQU     *
ØDD8 B6 ØØ        A         LDA     PORTA     GET NEXT CHANNEL
ØDDA A4 Ø7        A         AND     #%111     MASK GARBAGE
ØDDC BE ØF        A         LDX     RESULT    PICKUP LATEST RESULT
ØDDE B7 ØE        A         STA     ADCSR     START NEXT CONVERSION
ØDEØ 17 ØØ        A         BCLR    3,PORTA   DATA ABOUT TO CHANGE
ØDE2 BF Ø1        A         STX     PORTB     DATA CHANGING
ØDE4 16 ØØ        A         BSET    3,PORTA   DATA NOW VALID
ØDE6 2Ø FØ     ØDD8         BRA     SADLP
```

Figure 5-10. A/D Conversion Routine Software

Table 5-3. PLL Country/Land Selection Configuration

MC6805T2L1 Pins					Mode Option
PC0	PC1	PC2	PB0	PB1	
X	1	1	1	0	PLL-Europe
X	1	1	0	1	PLL-USA
X	1	1	1	1	PLL-Japan

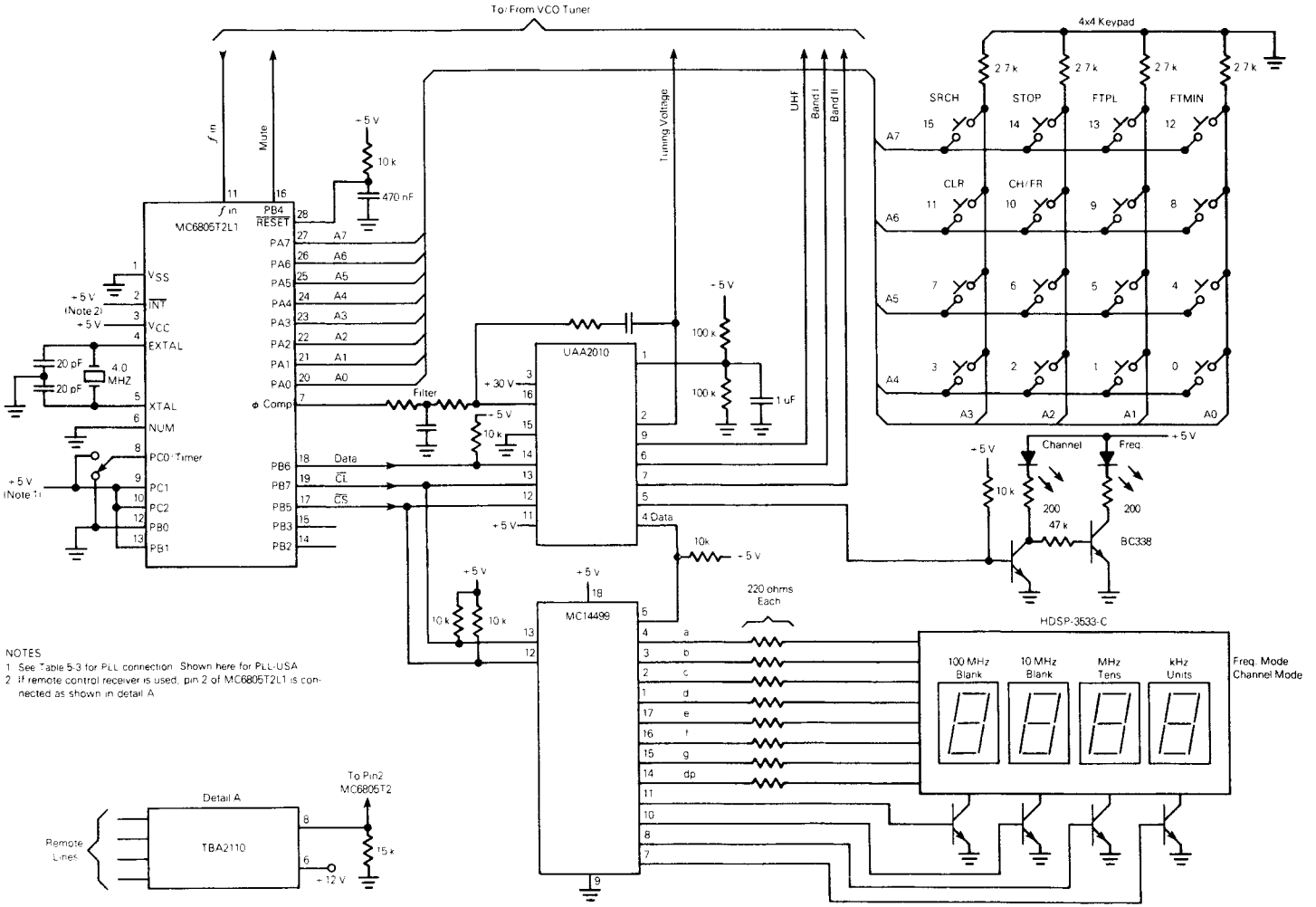


Figure 5-11. Synthesizer Mode Configuration Schematic Diagram

By using a keypad and display, any channel from 00-99 may be selected and displayed. The program calculates the frequency code (divide ratio) and stores it in the PLL registers (Hi and Lo). The contents of the PLL register are then loaded into a variable divider where it controls the f_{in} division. The variable divider output frequency is then compared to a reference divider frequency and the result of this comparison (ϕ COMP) is used to control the synthesizer frequency. Actually, the VCO frequency is divided in a prescaler in order to develop the f_{in} input at pin 11. A complete list of all synthesized channels is shown in Tables 5-4, 5-5, and 5-6.

In the synthesizer mode of operation configuration shown in Figure 5-11, the MC6805T2L1 program provides the following functions in conjunction with the 4 × 4 keypad.

1. channel select (keys 0-9)
2. channel or frequency display (CH/FR, key 10)
3. clear after one digit selected (CLEAR, key 11)
4. decrement frequency in 62.5 kHz steps (FTMIN, key 12)
5. increment frequency in 62.5 kHz steps (FTPL, key 13)
6. stop search (STOP, key 14)
7. channel search incrementing by one repeatedly (SRCH, key 15)

NOTE

In the description which follows, a European system is assumed. For USA and Japan differences refer to Tables 5-5 and 5-6.

At reset, the system of Figure 5-11 synthesizes and displays channel number 00 (lowest channel) and the channel indicator lights. Depressing the CH/FR key (10) causes the corresponding channel frequency (331.1) in MHz to appear on the 4-digit display, and the frequency indicator lights. The value shown can now be incremented in 62.5 kHz steps by depressing the FTMIN key (12). Only hundreds of kHz are displayed (331.125 displays as 331.1); therefore, the display might not change with each 62.5 kHz step.

A channel search operation can be activated by depressing the SRCH key (15). Depressing the SRCH key starts the search at the current channel and increments the channel every 350 milliseconds. When channel 99 (USA 83, Japan 62) is reached, the search cycle is repeated from channel 00 (USA 02, Japan 01). While the channel number is advanced, the corresponding frequency is also synthesized successively. Since the CH/FR key remains active, either the channel or resultant frequency is visible. To stop the search, it is only necessary to depress the STOP key (14).

For the USA and Japan PLL configurations, all channel numbers are not used; therefore, entering a non-existant channel (for example, 98 or 00) is interpreted by the program as a reset. After reset, the USA configuration display is 02 and the Japan configuration display is 01. A channel search operation is only made on existing channels.

The CLR key (11) allows the first selected digit (tens) of the channel number to be cleared in case of error during entry.

Table 5-4. Channel Characteristics for Europe

Channel No.	Band I	Band III	Band UHF	Divide Ratio Hex	Divide Ratio Decimal	Osc. Frequency MHz
00	0	1	0	14D2	5298	331.125
01	1	0	0	0552	1362	85.125
02	1	0	0	572	1394	87.125
03	1	0	0	5E2	1506	94.125
04	1	0	0	652	1618	101.125
05	0	1	0	D62	3426	214.125
06	0	1	0	DD2	3538	221.125
07	0	1	0	E42	3650	228.125
08	0	1	0	EB2	3762	235.125
09	0	1	0	F22	3874	242.125
10	0	1	0	F92	3986	249.125
11	0	1	0	1002	4098	256.125
12	0	1	0	1072	4210	263.125
13	1	0	0	5CA	1482	92.625
14	1	0	0	652	1618	101.125
15	1	0	0	792	1938	121.125
16	0	1	0	D62	3426	214.125
17	0	1	0	DEA	3554	222.125
18	0	1	0	E72	3698	231.125
19	0	1	0	F02	3842	240.125
20	0	1	0	F92	3986	249.125
21	0	0	1	1FE2	8162	510.125
22	0	0	1	2062	8290	518.125
23	0	0	1	20E2	8418	526.125
24	0	0	1	2162	8546	534.125
25	0	0	1	21E2	8674	542.125
26	0	0	1	2262	8802	550.125
27	0	0	1	22E2	8930	558.125
28	0	0	1	2362	9058	566.125
29	0	0	1	23E2	9186	574.125
30	0	0	1	2462	9314	582.125
31	0	0	1	24E2	9442	590.125
32	0	0	1	2562	9570	598.125
33	0	0	1	25E2	9698	606.125
34	0	0	1	2662	9826	614.125
35	0	0	1	26E2	9954	622.125
36	0	0	1	2762	10082	630.125
37	0	0	1	27E2	10210	638.125
38	0	0	1	2862	10338	646.125
39	0	0	1	28E2	10466	654.125
40	0	0	1	2962	10594	662.125
41	0	0	1	29E2	10722	670.125
42	0	0	1	2A62	10850	678.125
43	0	0	1	2AE2	10978	686.125
44	0	0	1	2BC2	11106	694.125
45	0	0	1	2BE2	11234	702.125
46	0	0	1	2C62	11362	710.125
47	0	0	1	2CE2	11490	718.125
48	0	0	1	2D62	11618	726.125
49	0	0	1	2DE2	11746	734.125
50	0	0	1	2E62	11874	742.125

Table 5-4. Channel Characteristics for Europe (Continued)

Channel No.	Band			Divide Ratio		Osc. Frequency MHz
	I	III	UHF	Hex	Decimal	
51	0	0	1	2EE2	12002	750.125
52	0	0	1	2F62	12130	758.125
53	0	0	1	2FE2	12258	766.125
54	0	0	1	3062	13386	774.125
55	0	0	1	30E2	12514	782.125
56	0	0	1	3162	12642	790.125
57	0	0	1	31E2	12770	798.125
58	0	0	1	3262	12898	806.125
59	0	0	1	32E2	13026	814.125
60	0	0	1	3362	13154	822.125
61	0	0	1	33E2	13282	830.125
62	0	0	1	3462	13410	838.125
63	0	0	1	34E2	13538	846.125
64	0	0	1	3562	13666	854.125
65	0	0	1	35E2	13794	862.125
66	0	0	1	3662	13922	870.125
67	0	0	1	36E2	14050	878.125
68	0	0	1	3762	14178	886.125
69	0	0	1	37E2	14306	894.125
70	1	0	0	602	1538	96.125
71	1	0	0	672	1650	103.125
72	1	0	0	7D2	2002	125.125
73	1	0	0	862	2146	134.125
74	1	0	0	8D2	2258	141.125
75	0	1	0	B12	2834	177.125
76	0	1	0	F82	3970	248.125
77	0	1	0	FF2	4082	255.125
78	1	0	0	6C2	1730	108.125
79	1	0	0	732	1842	115.125
80	1	0	0	7A2	1954	122.125
81	1	0	0	902	2306	144.125
82	0	1	0	972	2418	151.125
83	0	1	0	9E2	2530	158.125
84	0	1	0	A52	2642	165.125
85	0	1	0	AC2	2754	172.125
86	0	1	0	B32	2866	179.125
87	0	1	0	BA2	2978	186.125
88	0	1	0	C12	3090	193.125
89	0	1	0	C82	3202	200.125
90	0	1	0	CF2	3314	207.125
91	0	1	0	10E2	4322	270.125
92	0	1	0	1152	4434	277.125
93	0	1	0	11C2	4546	284.125
94	0	1	0	1232	4658	291.125
95	0	1	0	12A2	4770	298.125
96	0	1	0	1312	4882	305.125
97	0	1	0	1382	4994	312.125
98	0	1	0	13F2	5106	319.125
99	0	1	0	1462	5218	326.125

Table 5-5. Channel Characteristics for USA

Channel No.	Band			Divide Ratio		Osc. Frequency MHz
	I	III	UHF	Hex	Decimal	
02	1	0	0	650	1616	101.000
03	1	0	0	6B0	1712	107.000
04	1	0	0	710	1808	113.000
05	1	0	0	7B0	1968	123.000
06	1	0	0	810	2064	129.000
07	0	1	0	DD0	3536	221.000
08	0	1	0	E30	3632	227.000
09	0	1	0	E90	3728	233.000
10	0	1	0	EF0	3824	239.000
11	0	1	0	F50	3920	245.000
12	0	1	0	FB0	4016	251.000
13	0	1	0	1010	4112	257.000
14	0	0	1	2050	8272	517.000
15	0	0	1	20B0	8368	523.000
16	0	0	1	2110	8464	529.000
17	0	0	1	2170	8560	535.000
18	0	0	1	21D0	8656	541.000
19	0	0	1	2230	8752	547.000
20	0	0	1	2290	8848	553.000
21	0	0	1	22F0	8944	559.000
22	0	0	1	2350	9040	565.000
23	0	0	1	23B0	9136	571.000
24	0	0	1	2410	9232	577.000
25	0	0	1	2470	9328	583.000
26	0	0	1	24D0	9424	589.000
27	0	0	1	2530	9520	595.000
28	0	0	1	2590	9616	601.000
29	0	0	1	25F0	9712	607.000
30	0	0	1	2650	9808	613.000
31	0	0	1	26B0	9904	619.000
32	0	0	1	2710	10000	625.000
33	0	0	1	2770	10096	631.000
34	0	0	1	27D0	10192	637.000
35	0	0	1	2830	10288	643.000
36	0	0	1	2890	10384	649.000
37	0	0	1	28F0	10480	655.000
38	0	0	1	2950	10576	661.000
39	0	0	1	29B0	10672	667.000
40	0	0	1	2A10	10768	673.000
41	0	0	1	2A70	10864	679.000
42	0	0	1	2AD0	10960	685.000
43	0	0	1	2B30	11056	691.000
44	0	0	1	2B90	11152	697.000
45	0	0	1	2BF0	11248	703.000
46	0	0	1	2C50	11344	709.000
47	0	0	1	2CB0	11440	715.000
48	0	0	1	2D10	11536	721.000
49	0	0	1	2D70	11632	727.000
50	0	0	1	2DD0	11728	733.000
51	0	0	1	2E30	11824	739.000
52	0	0	1	2E90	11920	745.000
53	0	0	1	2EF0	12016	751.000
54	0	0	1	2F50	12112	757.000
55	0	0	1	2FB0	12208	763.000
56	0	0	1	3010	12304	769.000
57	0	0	1	3070	12400	775.000
58	0	0	1	30D0	12496	781.000
59	0	0	1	3130	12592	787.000
60	0	0	1	3190	12688	793.000

Table 5-5. Channel Characteristics for USA (Continued)

Channel No.	Band			Divide Ratio		Osc. Frequency MHz
	I	III	UHF	Hex.	Decimal	
61	0	0	1	31F0	12784	799.000
62	0	0	1	3250	12880	805.000
63	0	0	1	32B0	12976	811.000
64	0	0	1	3310	13072	817.000
65	0	0	1	3370	13168	823.000
66	0	0	1	33D0	13264	829.000
67	0	0	1	3430	13360	835.000
68	0	0	1	3490	13456	841.000
69	0	0	1	34F0	13552	847.000
70	0	0	1	3550	13648	853.000
71	0	0	1	35B0	13744	859.000
72	0	0	1	3610	13840	865.000
73	0	0	1	3670	13936	871.000
74	0	0	1	36D0	14032	877.000
75	0	0	1	3730	14128	883.000
76	0	0	1	3790	14224	889.000
77	0	0	1	37F0	14320	895.000
78	0	0	1	3850	14416	901.000
79	0	0	1	38B0	14512	907.000
80	0	0	1	3910	14608	913.000
81	0	0	1	3970	14704	919.000
82	0	0	1	39D0	14800	925.000
83	0	0	1	3A30	14896	931.000

Table 5-6. Channel Characteristics for Japan

Channel No.	Band I	Band III	Band UHF	Divide Ratio Hex.	Divide Ratio Decimal	Osc. Frequency MHz
01	1	0	0	960	2400	150.000
02	1	0	0	9C0	2496	156.000
03	1	0	0	A20	2592	162.000
04	0	1	0	E60	3680	230.000
05	0	1	0	EC0	3776	236.000
06	0	1	0	F20	3872	242.000
07	0	1	0	F80	3968	248.000
08	0	1	0	FC0	4032	252.000
09	0	1	0	1020	4128	258.000
10	0	1	0	1080	4224	264.000
11	0	1	0	10E0	4320	270.000
12	0	1	0	1140	4416	276.000
13	0	0	1	2120	8480	530.000
14	0	0	1	2180	8576	536.000
15	0	0	1	21E0	8672	542.000
16	0	0	1	2240	8768	548.000
17	0	0	1	22A0	8864	554.000
18	0	0	1	2300	8960	560.000
19	0	0	1	2360	9056	566.000
20	0	0	1	23C0	9152	572.000
21	0	0	1	2420	9248	578.000
22	0	0	1	2480	9344	584.000
23	0	0	1	24E0	9440	590.000
24	0	0	1	2540	9536	598.000
25	0	0	1	25A0	9632	602.000
26	0	0	1	2600	9728	608.000
27	0	0	1	2660	9824	614.000
28	0	0	1	26C0	9920	620.000
29	0	0	1	2720	10016	626.000
30	0	0	1	2780	10112	632.000
31	0	0	1	27E0	10208	638.000
32	0	0	1	2840	10304	644.000
33	0	0	1	28A0	10400	650.000
34	0	0	1	2900	10496	656.000
35	0	0	1	2960	10592	662.000
36	0	0	1	29C0	10688	668.000
37	0	0	1	2A20	10784	674.000
38	0	0	1	2A80	10880	680.000
39	0	0	1	2AE0	10976	686.000
40	0	0	1	2B40	11072	692.000
41	0	0	1	2BA0	11168	698.000
42	0	0	1	2C00	11264	704.000
43	0	0	1	2C60	11360	710.000
44	0	0	1	2CC0	11456	716.000
45	0	0	1	2D20	11552	722.000
46	0	0	1	2D80	11648	728.000
47	0	0	1	2DE0	11744	734.000
48	0	0	1	2E40	11840	740.000
49	0	0	1	2EA0	11936	746.000
50	0	0	1	2F00	12032	752.000

Table 5-6. Channel Characteristics for Japan (Continued)

Channel No.	Band I	Band III	Band UHF	Divide Ratio Hex.	Divide Ratio Decimal	Osc. Frequency MHz
51	0	0	1	2F60	12128	758.000
52	0	0	1	2FC0	12224	764.000
53	0	0	1	3020	12320	770.000
54	0	0	1	3080	12416	776.000
55	0	0	1	30E0	12512	782.000
56	0	0	1	3140	12608	788.000
57	0	0	1	31A0	12704	794.000
58	0	0	1	3200	12800	800.000
59	0	0	1	3260	12896	806.000
60	0	0	1	32C0	12992	812.000
61	0	0	1	3320	13088	818.000
62	0	0	1	3380	13184	824.000

A flowchart showing the main synthesizer (PLL05) mode routine is shown in Figure 5-12 and selection of the various programs in the MC6805T2L1 is shown in Figure 5-13. Note that in Figure 5-13 the reset routine will exit to one of the eight modes depending upon the configuration of PB0, PB1, PC0, PC1, and PC2 per Table 1.

5.9 KEYLESS ENTRY SYSTEM USING THE MC146805F2( )1 (AN-863)

5.9.1 Introduction

The keyless entry system (also referred to as a digital lock) is a dedicated MC146805F2( )1 Microcomputer Unit (MCU), executing a program, that can control a larger configuration to form a security system. Figure 5-14 contains a schematic diagram of the digital lock complete with keypad and liquid crystal display.

NOTE

The keyless entry system using the MC146805F2( )1 8-Bit Microcomputer Unit is not intended to be used by itself in a secure entry system. It is intended to be used only as an aid in better understanding the MC146805F2 MCU and how it can fit into a secure entry system.

The digital lock accepts inputs from the 3 × 4 keypad, and, if the inputs are in the correctly coded sequence, generates an output which indicates the lock is open. The digital lock MCU has a feature which protects against "trial-and-error" attempts to gain entry. If two incorrect code combinations are entered, an alarm output is generated (PB2 goes high). The alarm condition remains active until the combination is entered or power is disconnected.

The user interfaces with the digital lock MCU through a 3 × 4 keypad and a "wake-up" pushbutton. This allows multiple users to gain access to a secure area without the necessity of carrying a key. The LCD displays a dash for each keypad entry. This ensures that the user knows how many of the required keypad entries have been made. Once the correct combination has been entered via the keypad, the LCD spells out the word OPEN. From this time, the user has eight seconds to open the door or other locked device.

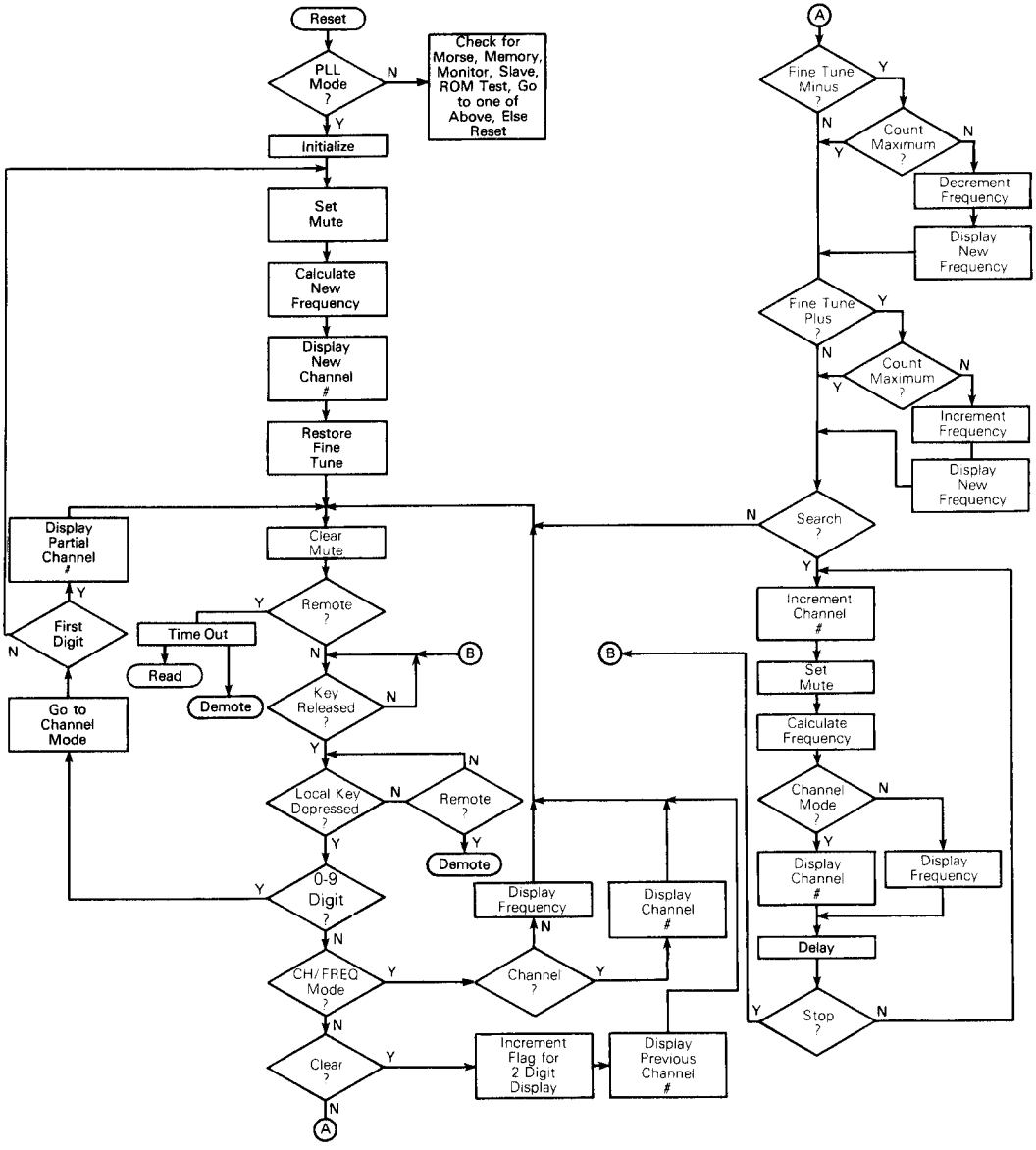


Figure 5-12. Main Synthesizer (PLL05) Routine Flowchart

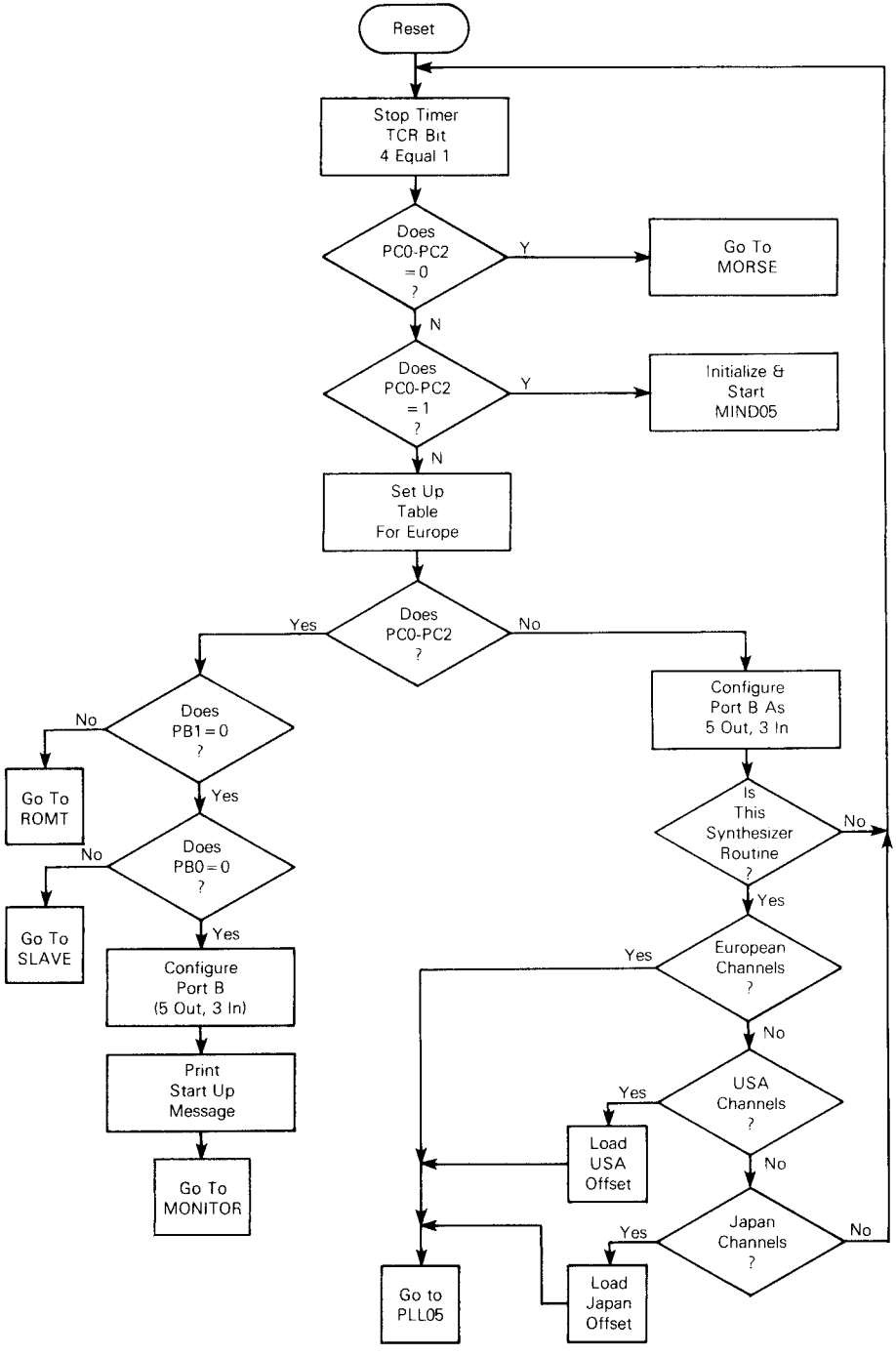


Figure 5-13. Reset Routine Flowchart

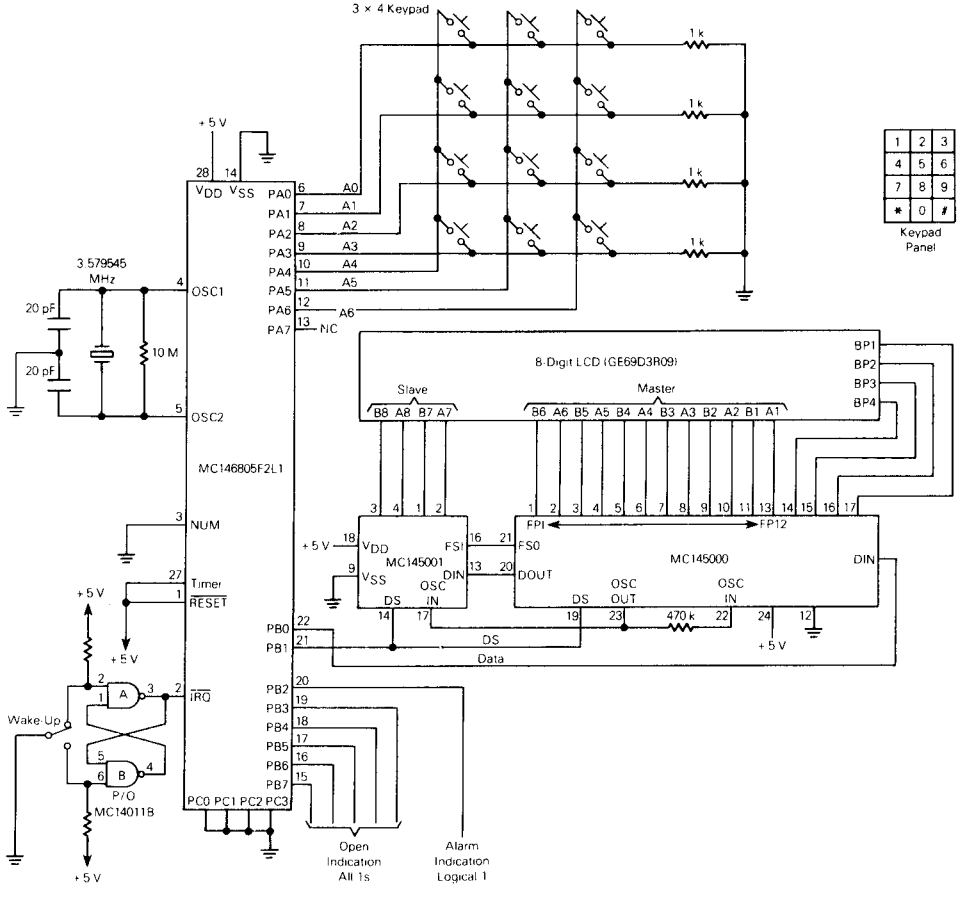


Figure 5-14. Digital Lock System Schematic Diagram

5.9.2 Initialization

When power is initially applied or if power is lost and then reapplied, the 8-digit combination code is lost in RAM. It now becomes necessary to enter a new 8-digit combination. This can be done by performing the procedure outlined in the Changing The Coded Sequence paragraph.

5.9.3 Operation

Two operating modes are described below. One is the normal user procedure to open the digital lock and the other describes a method to change the coded sequence combination.

5.9.3.1 OPENING THE DIGITAL LOCK. To open the digital lock proceed as follows:

1. Press the "wake-up" pushbutton and check that the LCD is clear.
2. Use the keypad to enter the 8-digit combination code. Note that each time a keypad switch is depressed a dash will appear, on the LCD, to indicate that a digit is entered. The total number of digits entered is equal to the total number of dashes.
3. Once the correct 8-digit combination code is entered, the LCD displays the word "OPEN". The open signal is then active for approximately eight seconds. If the user fails to mechanically open the door (or other entry device) during the 8-second time period, the above procedure must be repeated to again gain entry.

NOTE

If an incorrect code is entered for the second time, the alarm signal becomes active. The alarm will stay active until the correct code is entered, as described above, or power is removed.

5.9.3.2 CHANGING THE CODED SEQUENCE. To change the digital lock coded sequence (combination), proceed as follows:

1. Press the "wake-up" pushbutton and check that the LCD is clear.
2. Use the keypad to enter the 8-digit "change combination code" number 14680502. Note that each time a keypad switch is depressed, a dash will appear, on the LCD, to indicate that a digit is entered. Once all eight digits are entered, the LCD goes blank.
3. Use the keypad to enter the new 8-digit combination code. As before, a dash appears each time a keypad switch is depressed.
4. Once the eight new digits are entered, the word "VERIFY" appears on the LCD. This is a prompt for the user to enter the same 8-digit combination code as in 3 above. If the second 8-digit entry is not exactly the same as the first, the word "ERROR" is displayed on the LCD. In this case, the user must repeat the procedure from 3 above.

NOTE

Changing the combination coded sequence does not open the lock. Once the new code has been verified, the LCD goes blank. The lock can then be opened as described above in the Opening The Digital Lock paragraph.

5.10 BICYCLE COMPUTER USING THE MC146805G2( )1 (AN-858)

5.10.1 Introduction

In the configuration shown in Figure 5-15, the MC146805G2( )1 is used as a bicycle computer. Features provided by the bicycle computer include: (1) instantaneous speed, (2) average speed, (3) resettable trip odometer, (4) resettable long distance odometer, (5) cadence (pedal crank revolutions per minute), (6) selection of English or metric units, and (7) calibration for wheel size.

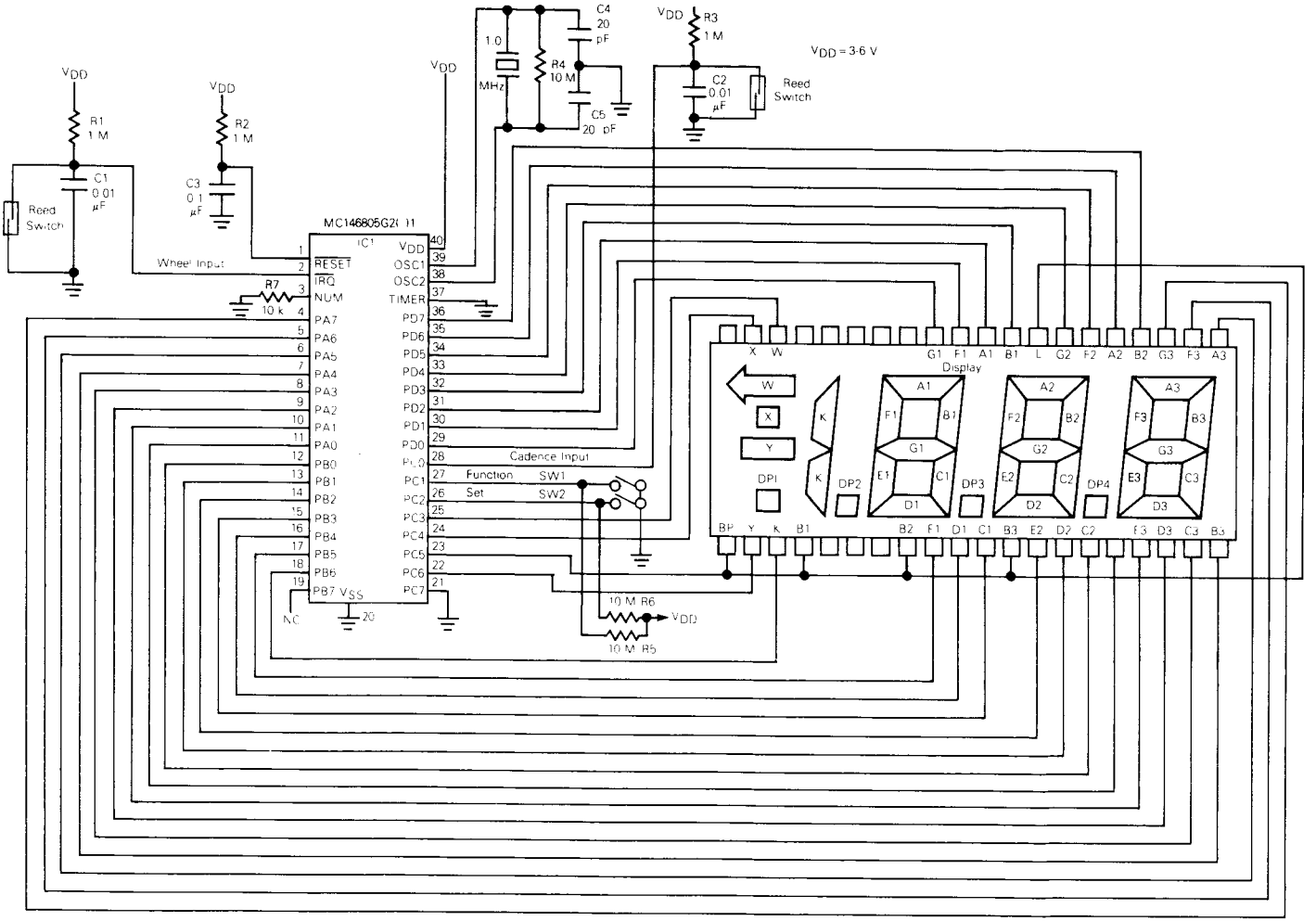
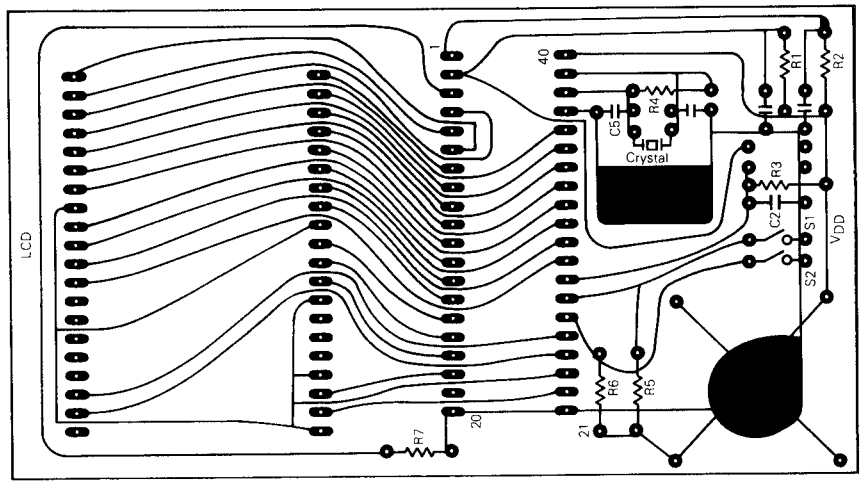
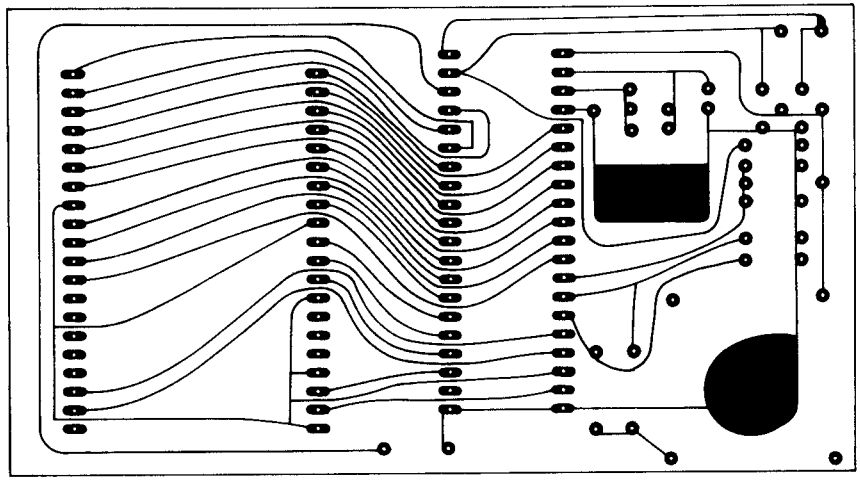


Figure 5-15. Bicycle Computer Schematic Diagram



a. Parts Location



b. Circuit Board Art
(Actual Size)

Figure 5-16. Bicycle Computer Circuit Board

5.10.2 Hardware Configuration

A schematic diagram for the bicycle computer is shown in Figure 5-15 and Figure 5-16 shows a parts layout diagram plus circuit board art. As shown on the schematic diagram, the MC146805G2( )1 and the liquid crystal display (LCD) are the only major components required for the bicycle computer. All necessary drive signals for the LCD are contained in firmware. Two pushbutton switches (S1 and S2, function and set) are required to furnish two momentary ground inputs, and two sensor inputs (one from the wheel and one

from the pedal crank) are required as an interrupt and to pulse certain counters. Each sensor is a normally-open switch which is activated by a magnet mounted on the wheel and pedal crank.

Figure 5-15 shows the layout of a PCB that may be used when assembling the bicycle computer. The printed circuit board (PCB) is designed to fit in a Wonder-Lite case. The Wonder-Lite is designed to mount on a bicycle and provide nighttime illumination. Dimensions for this bord are 4.5" × 2.5" and could require some tailoring before fitting into the mounting case. However, an equivalent size wire-wrap type board using the wire-wrap connections and mounting sockets could be used with an equivalently sized case.

5.10.3 Bicycle Computer Function

When power is initially applied to the circuit or when the MC146805G2( )1 is reset, the bicycle computer program is selected and the bicycle computer displays the current instantaneous speed on the display (Function 1). Each time the "function" button (S1) is pushed, the bicycle computer will step to the next function. The functions are:

1. instantaneous speed
2. average speed
3. resettable trip odometer
4. resettable long distance odometer
5. cadence
6. English or metric units selection
7. wheel size calibration

Each time the function switch is pushed, the program steps to the next function; however, after function 7 it returns to function 1. Some functions may require resetting. For example, at the beginning of each bicycle trip it may be desirable to reset the trip odometer to zero for miles or kilometers. The "set" pushbutton (S2) is provided to perform this task. If the set button is pushed while in function 3, the trip odometer is reset to zero. However, it is not desirable to have the "set" button enabled at all times. For example, if the "set" button were accidentally pushed during a trip, the trip odometer would be reset to zero. Therefore, the "set" button is only enabled for the first five seconds after a new function is selected. Pushing the "set" button after five seconds will not affect the function. During the five seconds that the "set" button is enabled, the bicycle computer displays a fixed function identification display. For example, the trip odometer will display " odo " during this function 3 time. After five seconds the selected function value is displayed and remains displayed until the "function" button is again pushed, stepping to the next function. Complete descriptions for these functions are provided in Motorola Application Note AN-858.

Wonder-Lite is a trademark of Wonder Corp. of America

5.11 AVAILABLE APPLICATION NOTES

Several application notes for the M6805 HMOS/M146805 CMOS Family are (or will be) available as of the printing of this users manual. A list of these application notes is provided in Table 5-7.

Table 5-7. List of Available Application Notes

AN#	Title
823	CBUG05 Monitor Program for MC146805E2 Microprocessor Unit
852	Monitor for the MC146805G2L1 Microcomputer
853	M146805 CMOS Family Emulators
855	Versatile Thermostat using CMOS MC146805E2 MPU
857	MC68705P3/R3/U3 EPROM Microcomputer Programming Module
858	Bicycle Computer using the MC146805G2L1 Microcomputer
863	Keyless Entry System using an MC146805F2( )1 8-Bit Microcomputer Unit
869	Application Summary for the MC6805R2( )1 Single-Chip Microcomputer With A/D Converter
871	AN Applications Summary for the MC6805T2L1 Single-Chip Microcomputer With Phase-Lock-Loop
883	A Radio Set Phase-Lock-Loop (PLL) using an MC6805T2( )2 Single-Chip Microcomputer

In addition, the TWX, TELEX, DITEL, and telephone numbers plus the mailing address of the Literature Distribution Center (where the application notes are available) are listed below.

The Literature Distribution Center, located in Phoenix, Arizona, offers a method by which a sales office or customer can order the application notes listed in Table 5-7. A listing of various methods to communicate with the Literature Distribution Center is shown below.

Phone:	Literature Distribution Center	602-994-6561
TWX:	(MOT SEMI PHX)	901-951-1334 (LDC)
DITEL:	(Motorola Facilities)	234-6561
Mail Drop:	Broadway Bldg. (BB100)	
Address:	Motorola Semiconductors Products, Inc. Literature Distribution Center P. O. Box 20924 Phoenix, Az 85036	

CHAPTER 6
EPROM PROGRAMMING

6.1 INTRODUCTION

6.1.1 General

The M6805 HMOS/M146805 CMOS Family of MCUs uses either on-chip masked ROM or on-chip EPROMs for program storage. Erasable Programmable Read Only Memory (EPROM) devices allow programs to be written into memory and, if desired, later erased with ultraviolet light and revised. These features give the user an alterable, non-volatile memory. Each EPROM in this family includes a bootstrap routine in masked ROM, which makes programming relatively easy. Currently, four EPROM devices exist, three of which are implemented in the M6805 HMOS Family with the fourth being implemented in the M146805 CMOS Family.* These devices may be used to emulate various masked ROM versions of other members of the family. The EPROM devices have more capabilities than do the masked ROM versions, thus allowing some EPROM devices to emulate more than one masked ROM version.

Each EPROM includes a Mask Option Register (MOR) which is implemented in EPROM. The MOR is located at address $784 in the MC68705P3, $F38 in the MC68705R3, $F38 in the MC68705U3, and $1FF5 in the MC1468705G2. The M6805 HMOS Family MOR is used to determine which of the timer options are to be used and to select the clock oscillator circuit (crystal or RC); whereas, the M146805 CMOS Family MOR is used to select the clock oscillator circuit, divide ratio of the clock oscillator, and type of interrupt trigger input. The MOR, like all EPROM locations, contains all zeros after erasing. Table 6-1 gives a description of the function of each MOR bit used in the M6805 HMOS Family and Table 6-2 provides equivalent MOR information for the MC1468705G2.

6.1.2 M6805 HMOS Family Bootstrap

Each member of the M6805 HMOS Family of EPROM devices contains a bootstrap program which is implemented in on-chip masked ROM. The bootstrap program clocks an external counter which is used to generate an address. The address is then used to read a location in an external memory. The data from the external memory is presented to the EPROM via an I/O port. After data from that location is loaded into the EPROM, the

*At the initial printing of this manual, four different M6805 HMOS/M146805 CMOS Family EPROM types are available; however, others are scheduled to follow.

bootstrap routine clocks the counter to increment the address and read the next location. After the data from all locations are loaded into the EPROM, its contents are compared to those in external memory. The programming status is indicated by two LEDs (see Table 6-3 and Figure 6-1).

Table 6-1. M6805 HMOS Family Mask Option Register

b7	b6	b5	b4	b3	b2	b1	b0	Mask Option Register
CLK	TOPT	CLS			P2	P1	P0	

b7, CLK	Clock Oscillator Type 1 = RC 0 = Crystal **NOTE** V_{IHTP} on the TIMER/BOOT pin (8) forces the crystal mode.
b6, TOPT	Timer Option 1 = M6805 HMOS Family type timer/prescaler. All bits, except 3, 6, and 7, of the timer control register (TCR) are invisible to the user. Bits 5, 2, 1, and 0 of the mask option register determine the equivalent M6805 HMOS Family mask options. 0 = All TCR bits are implemented as a software programmable timer. The state of MOR bits 5, 4, 2, 1, and 0 sets the initial values of their respective TCR bits (TCR is then software controlled after initialization).
b5, CLS	Timer/Clock Source 1 = External TIMER pin 0 = Internal $\phi 2$
b4	Not used if MOR TOPT = 1. Sets initial value of TCR TIE if MOR TOPT = 0.
b3	Not used.
b2, P2 b1, P1 b0, P0	Prescaler Option – the logical levels of these bits, when decoded, select one of eight taps on the timer prescaler. The division resulting from decoding combinations of these three bits is shown here.

P2	P1	P0	Prescaler Division
0	0	0	1 (Bypass Prescaler)
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Table 6-2. M1468705G2 Mask Option Register

b7	b6	b5	b4	b3	b2	b1	b0	Mask Option Register
CLK	DIV		INT					

b7, CLK	Clock Oscillator Type 1 = RC 0 = Crystal
b6, DIV	Determines Division of Clock Oscillator 1 = Divide-by-2 oscillator clock 0 = Divide-by-4 oscillator clock
b5,	Not used.
b4, INT	Determines type of Interrupt Trigger Input 1 = Both Edge-sensitive and level-sensitive triggered interrupt 0 = Edge-sensitive triggered interrupt only
b0, b1, b2, b3	Not used.

Table 6-3. M6805 HMOS EPROM LED Results

LED	Function
DS1 (PB1)	Turned on (when PB1 goes low) to indicate EPROM device is programmed.
DS2 (PB2)	Turned on (when PB2 goes low) to indicate EPROM contents are successfully verified (approximately two seconds after DS1 is turned on). Programming and verification are now complete.

Two examples for programming the M6805 HMOS Family MOR are discussed below.

Example 1 When emulating an MC6805P2 (using an MC68705P3) to verify your program with an RC oscillator and an event counter input for the timer with no prescaling, the MOR should be programmed to '11111000". To write the MOR, it is simply programmed as any other EPROM byte. (The same criteria is applicable when using the MC68705R3 to emulate the MC6805R2 or the MC68705U3 to emulate the MC6805U2.)

Example 2 Suppose you wish to use the EPROM programmable prescaler functions, and you wish the initial condition of the prescaler to be a divide-by-64, with the input disabled and an internal clock source. If the clock oscillator is to be in the crystal mode, the MOR would be programmed to "00001110".

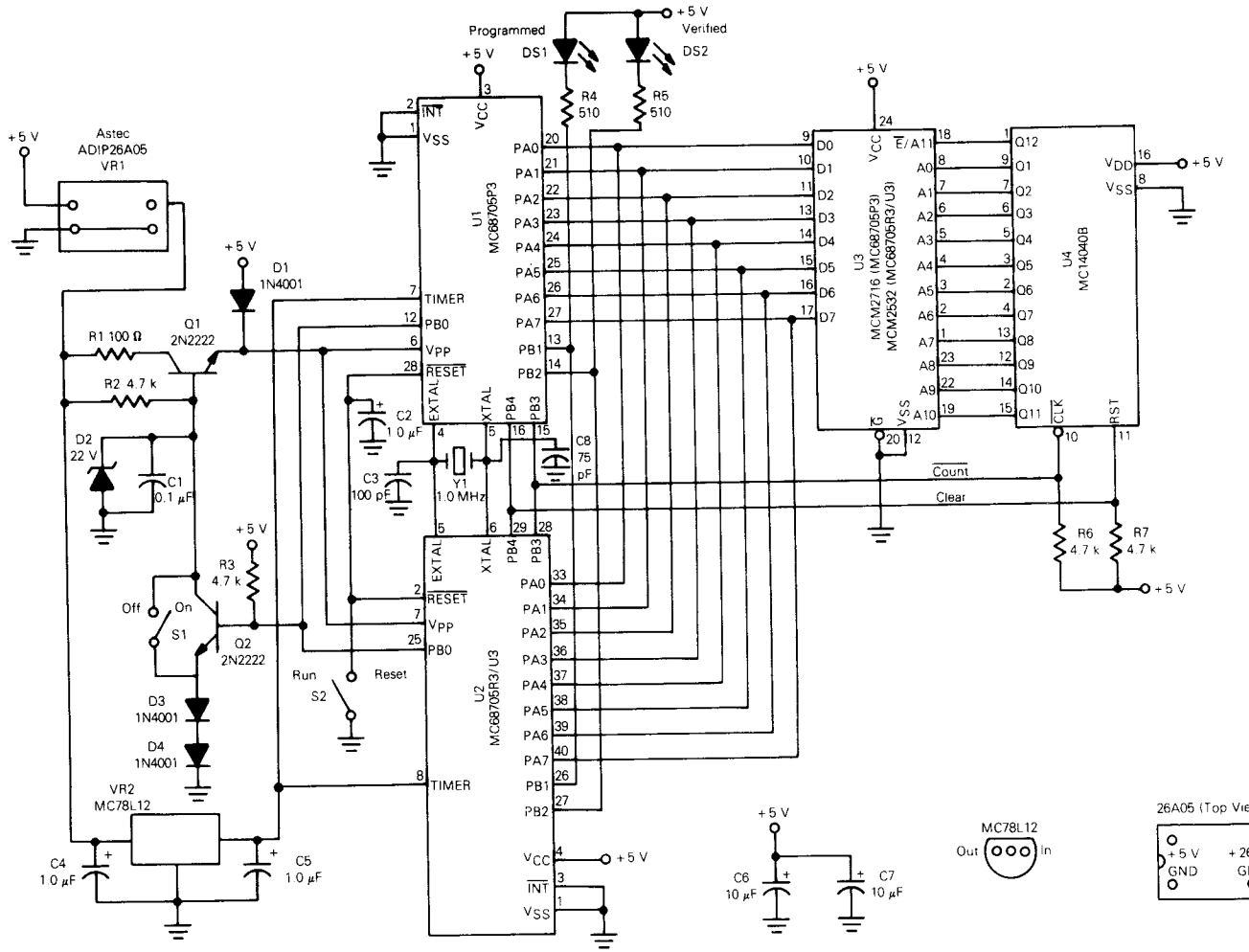


Figure 6-1. MC68705P3/R3/U3 Programming Module Schematic Diagram

6.1.3 M146805 CMOS Family Bootstrap

The MC1468705G2 MCU EPROM device also contains a bootstrap program which is implemented in on-chip masked ROM. However, in this program no external counter is required to generate the address. Instead, the address is generated internally and applied via port A and port D lines to read the location in external memory. As with the M6805 HMOS Family, the data from external memory is presented to an I/O port. After data from that location is loaded into the EPROM, the bootstrap routine increments the output address and reads the next location. Two LEDs provide an indication of the programming status (see Table 6-4 and Figure 6-2).

Table 6-4. MC1468705G2 EPROM LED Results

LED	Function
DS2 (PD6)	Turned on (when PD6 goes low) to indicate EPROM device is being programmed.
DS1 (PD5)	Turned on (when PD5 goes low) to indicate EPROM contents are successfully verified. Programming and verification are now complete.

An example for programming the MC1468705G2 EPROM MOR is as follows: when emulating an MC146805G2 (using an MC1468705G2) to verify your program with a crystal oscillator, a divide-by-4 oscillator clock, and both edge-sensitive and level-sensitive triggered inputs, the MOR should be programmed to "00010000".

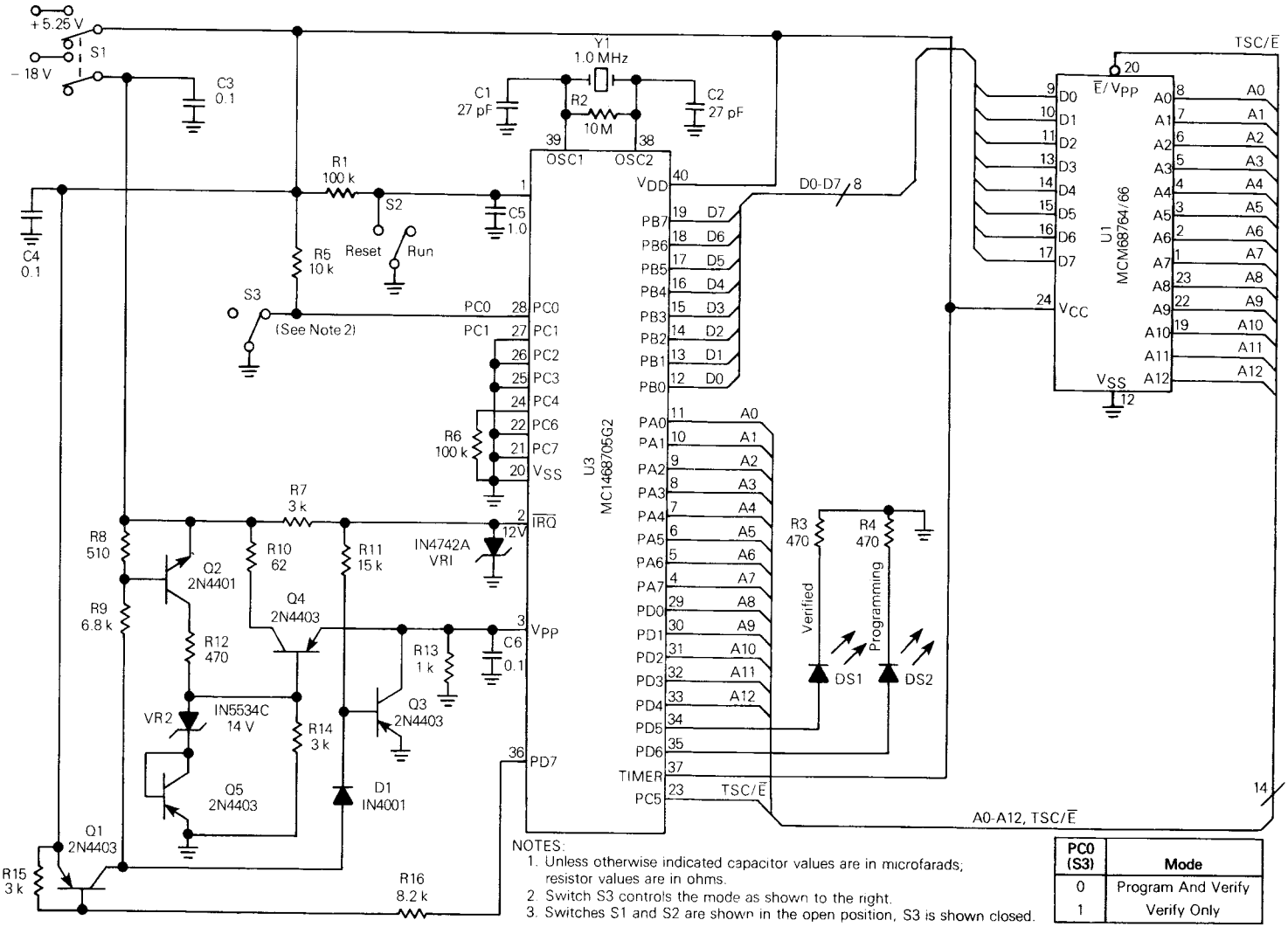
6.2 PROGRAMMING

6.2.1 M6805 HMOS Family

Figure 6-1 contains a schematic diagram of a circuit which can be used to program the MC68705P3, MC68705R3, and MC68705U3 EPROM Microcomputer Unit devices. Since the routine required to program the EPROM MCU is actually located within the device, only a small number of parts are required to build the circuit for programming the EPROM MCU. Figure 6-3 shows a parts layout of the printed circuit board and Table 6-5 provides a parts list.

Except for the socket used for mounting the EPROM MCU device the use of either a 2K (MCM2716) or 4K (MCM2532) EPROM for U2, programming either of the EPROM MCUs is basically the same. Because of this similarity, the procedure for programming the MC68705P3 is described first, followed by the MC68705R3/U3 procedure.

6.2.1.1 MC68705P3 Programming. Prior to programming the MC68705P3 EPROM, it should be erased by exposing it to a high-intensity ultraviolet (UV) light with a wavelength of 2537 angstroms. The recommended dose (UV intensity x exposure time) is 15 Ws/cm^2. The UV lamps should be used without shortwave filters and the MC68705P3 should be positioned about one inch from the UV tubes. Be sure the EPROM window is shielded from light except when erasing.


+5.25 V
S1
−18 V
C3
0.1
C4
0.1
Y1
1.0 MHz
C1
27 pF
R2
10 M
C2
27 pF
OSC1
OSC2
R1
100 k
S2
Reset
Run
C5
1.0
R5
10 k
S3
(See Note 2)
R6
100 k
R7
3 k
R8
510
R9
6.8 k
Q2
2N4401
R10
62
Q4
2N4403
R11
15 k
IN4742A
VRI
12V
R12
470
R13
1 k
C6
0.1
VR2
IN5534C
14 V
R14
3 k
Q3
2N4403
Q5
2N4403
D1
IN4001
Q1
2N4403
R15
3 k
R16
8.2 k
U3
MC1468705G2
VDD
VSS
IRQ
VPP
PD7
TIMER
PC5
TSC/E
D0-D7
8
R3
470
R4
470
Verified
Programming
DS1
DS2
U1
MCM68764/66
E/VPP
VCC
VSS
A0-A12, TSC/E
14
NOTES:
1. Unless otherwise indicated capacitor values are in microfarads; resistor values are in ohms.
2. Switch S3 controls the mode as shown to the right.
3. Switches S1 and S2 are shown in the open position, S3 is shown closed.
PC0 (S3)
Mode
0
Program And Verify
1
Verify Only
0 = Switch closed 1 = Switch open

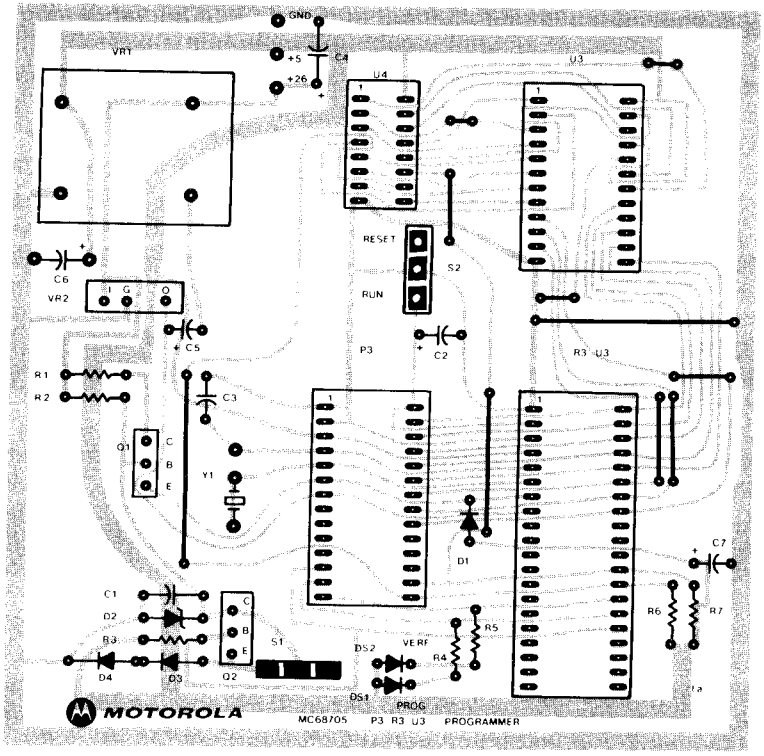


Figure 6-3. MC68705P3/R3/U3 Programming Module Parts Layout

Table 6-5. MC68705P3/R3/U3 Programming Module Parts List

R1	100 Ω
R2	4.7 kΩ
R3	4.7 kΩ
R4	510 Ω
R5	510 Ω
R6	4.7 kΩ
R7	4.7 kΩ
C1	0.1 μF
C2	1.0 μF
C3	100 pF
C4	1.0 μF
C5	1.0 μF
C6	10 μF
C7	10 μF
D1	1N4001
D2	22V Zener-1N4748A or Equiv.
D3	1N4001
D4	1N4001

Q1	2N2222 or Equiv.
Q2	2N2222 or Equiv.
Y1	1 MHz (AT-Cut Parallel Resonance, 100 Ω Max.)
U1	MC68705P3 } Only Use One
U2	MC68705R3/U3 } Only Use One
U3	MCM2716 or MCM2532
U4	MC14040B
VR1	ASTEC Voltage Converter 26A05
VR2	MC78L12
DS1	Red LED
DS2	Green LED
PCB1	Printed Circuit Board
Misc:	1 – 40 Pin Low Insertion Force Socket
	1 – 28 Pin Low Insertion Force Socket
	1 – 24 Pin Low Insertion Force Socket
	1 – 16 Pin Solder Tail Socket
	2 – SPDT Switches

The MCM2716 UV EPROM is used for U3 when programming the MC68705P3. Before the MC68705P3 can be programmed, the MCM2716 UV EPROM must first be programmed with an exact duplicate of the information that is to be transferred to the MC68705P3.

NOTE

The first 128 bytes of EPROM (MCM2716) are ignored; location $80 of the EPROM is placed in location $80 of the MC68705P3.

Step 1—Close switches S1 and S2 and be sure that voltage (+5 V in this case) is not applied to the circuit board.

Step 2—Insert the MCM2716 into the socket for U3 and insert the MC68705P3 into the U1 socket.

Step 3—Apply +5 V to the circuit board.

Step 4—Open switch S1 to apply V_{PP} to the MCU and then open switch S2 to remove reset.

NOTE

Once the MCU comes out of reset, the CLEAR output control line (PB4) goes high and then low, then the MC14040B counter is clocked by the PB3 output (COUNT). The counter selects the MCM2716 EPROM byte which is to load the equivalent MC68705P3 EPROM byte selected by the MCU bootstrap program. Once data is programmed, COUNT increments the counter to the next location. This continues until the MCU is completely programmed.

Step 5—Check that the programmed LED indicator is lit followed by lighting of the verified indicator LED. This signals that the EPROM MPU has been correctly programmed.

Step 6—Close switch S1 to remove V_{PP} and V_{IHTP}. Close switch S2 to reset the MCU.

Step 7—Disconnect (or turn off) the +5 V input to the circuit board and then remove the newly programmed EPROM MCU from its socket.

Step 8—Remove the U3 EPROM from its socket if no further programming is required.

6.2.1.2 MC68705R3/MC68705U3 Programming. Programming either of these MCU EPROMs is similar to that described above for the MC68705P3 with three minor exceptions. These three exceptions are:

1. The MCM2532 UV EPROM is used for U3 when programming either the MC68705R3 or MC68705U3 EPROM MCU. This UV EPROM must be programmed with an exact duplicate of the information being transferred to MC68705R3 or MC68705U3.

2. In step 2 the MCM2532 is inserted into the U3 socket and the MC68705R3 or MC68705U3 is inserted into the U2 socket.
3. In the note under step 4, operation of the MCM2532 and MC68705R3/U3 is identical to that described for the MCM2716 and MC68705P3.

6.2.1.3 Printed Circuit Board. The PCB is a double-sided board with plated through holes. However, a single-sided board requiring only 10 wire jumpers could be used. The wire jumpers would be in place of the wiring shown in the component section of Figure 6.4. Component tolerances are generally not critical. The 5-to-26 volt converter (VR1) is manufactured by ASTEC International under part number ADIP26A05; however, if this part is not available, + 26 Vdc may be applied to the soldering feed through which is adjacent to the C4 + soldering feed through (PCB ground must be connected for this supply).

Figure 6-3 is a parts layout detail as shown from the component side of the board. Figure 6-4 contains the circuit board art (both component side and circuit side) detail. These are actual sizes and can be used for developing a double-sided board.

6.2.2 MC1468705G2 Programming

Figure 6-2 contains a schematic diagram of a circuit which can be used to program the MC1468705G2 EPROM Microcomputer Unit, and Table 6-6 contains a parts list. Since the routine required to program the EPROM MCU plus the address to select the data is actually located in the device, only a small number of parts are required to build the circuit for programming the EPROM MCU. The procedure for programming the MC1468705G2 is described below.

Table 6-6. MC1468705G2 MCU EPROM Programming Circuit Parts List

R1	100 kΩ
R2	10 MΩ
R3	470 Ω
R4	470 Ω
R5	10 kΩ
R6	100 kΩ
R7	3 kΩ
R8	510 Ω
R9	6.8 kΩ
R10	62 Ω
R11	15 kΩ
R12	470 Ω
R13	1 kΩ
R14	3 kΩ
R15	3 kΩ
R16	8.2 kΩ
C1	27 pF
C2	27 pF
C3	0.1 μF
C4	0.1 μF
C5	1 μF
C6	0.1 μF

D1	1N4001
Q1	2N4403
Q2	2N4401
Q3	2N4403
Q4	2N4403
Q5	2N4403
Y1	1 MHz (AT-Cut Parallel Resonance, 100 ohms Max.)
U1	MCM68764 or MCM68766
U2	MC1468705G2
VR1	1N4742A
VR2	1N5534C
DS1	LED
DS2	LED
PB1	Printed Circuit Board
Misc.	1 – 40 Pin Low Insertion Force Socket 1 – 24 Pin Low Insertion Force Socket 2 – SPDT Switches 1 – DPDT Switch

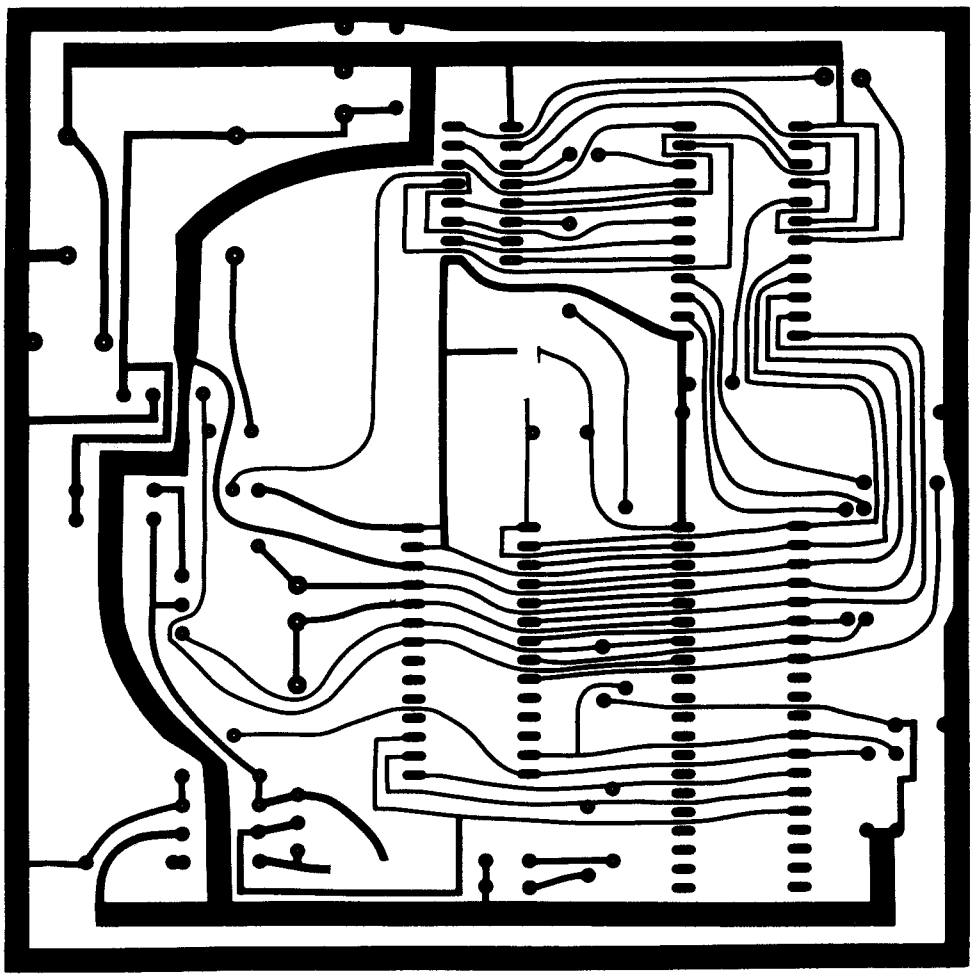


Figure 6-4a. MC68705P3/R3/U3 Programming Module Circuit Board Art

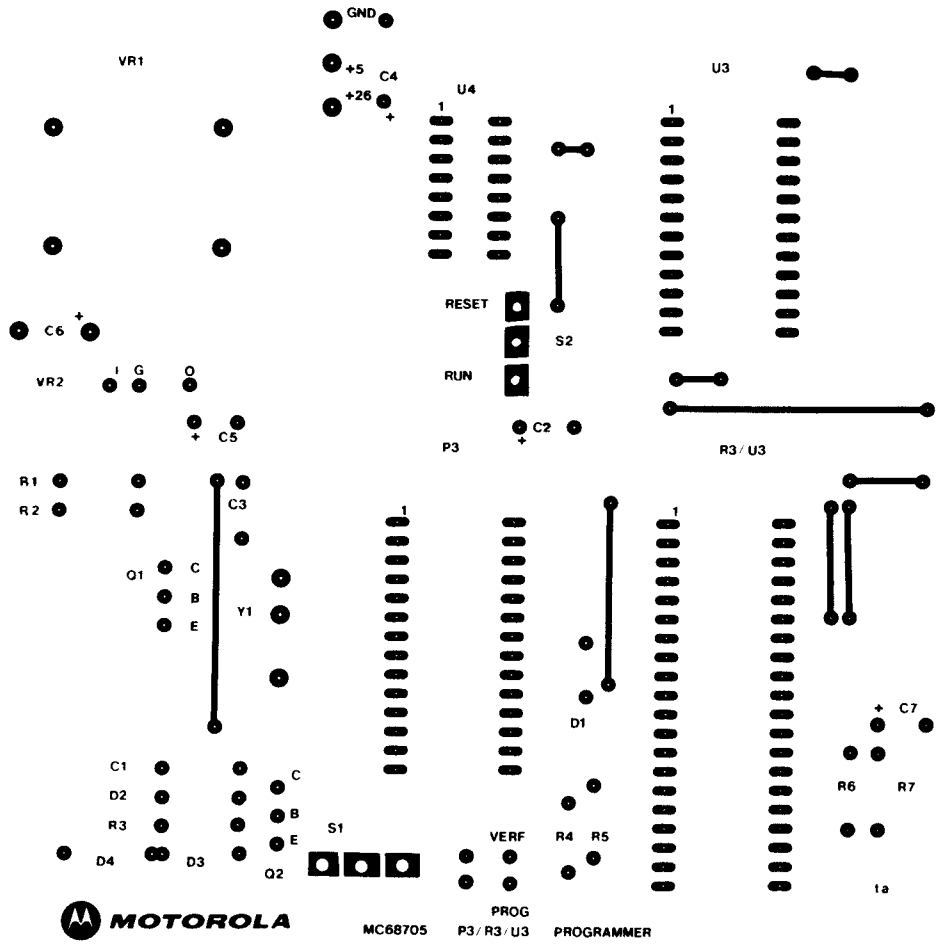


Figure 6-4b. MC6870P3/R3/U3 Programming Module Circuit Board Art (Continued)

The schematic diagram of Figure 6-2 provides connections for using an MCM68764 or an MCM68766 (8K X 8) EPROM. Since each of these EPROM devices are 24-pin devices, the 24-pin low insertion force socket connector for U1 can be used.

Since the actual EPROM memory used in the MC1468705G2 is 2106 bytes, the EPROM device(s) used in programming only needs 4K bytes of memory location. Figure 6-5 shows the MCM68764 or MCM68766 memory locations in which the MC1468705G2 program should be stored.

Prior to programming the MC1468705G2 EPROM, it should be erased by exposure to high-intensity ultraviolet (UV) light with a wavelength of 2537 angstroms. The recommended integrated dose (UV intensity × exposure time) is 40 Ws/cm2. The UV lamps should be used without shortwave filters and the MC14648705G2 should be positioned about one inch from the UV tubes. Be sure the EPROM window is shielded from light with an **OPAQUE** cover at all times except when erasing. This protects both the EPROM and light-sensitive nodes.

CAUTION

Be sure that S1 is open, and S2 is closed when inserting the MC1468705G2 and/or MCM68764 EPROM(s) into their respective sockets. This ensures that $\overline{\text{RESET}}$ is held low and power is not applied when inserting the device(s).

Note that the MCM68764 (or MCM68766) memory locations which correspond to RAM locations or unused EPROM or ROM locations in the MC1468705G2, may be programmed as either $00 or $FF (don't care).

To program the MC1468705G2 proceed as follows:

Step 1 — Open S3 to select the programming and verify mode. Close S1 (to apply the proper voltages for the V_{DD}, TIMER, and $\overline{\text{IRQ}}$ pins).

Step 2 — Open S2 to remove reset.

NOTE

Once the MCU comes out of reset, the V_{PP} control line (PD7) goes low and the V_{PP} voltage is applied to pin 3. With V_{PP}, applied, the EPROM is programmed one byte at a time with the corresponding data in the MCM68764 or MCM68766 EPROM. The MC1468705G2 bootstrap provides the address and enable TSC/$\overline{\text{E}}$ signals to permit complete self programming.

Step 3 — Check that the programming LED is turned on and remains on through the programming sequence. After completion of the programming sequence, this LED turns off.

NOTE

Transfer of the entire contents of MCM68764 or MCM68766 EPROM requires approximately 200 seconds. The internal timer is then cleared and the loop is repeated to verify that the programmed data is precisely the same as the incoming data from the EPROM; if so, the verified LED is turned on.

Step 4 — If the verified LED is not turned on, the exact program has not been loaded from the EPROM to the MC1468705G2, indicating a possible defect.

Step 5 — Close S2 (to reset the MC1468705G2) and open S1 prior to removing any device (MCU or EPROM) from its socket.

CAUTION

Once the MC1468705G2 is programmed and connected for normal operation, be sure that V_{PP} (pin 3) is connected directly to V_{SS}.

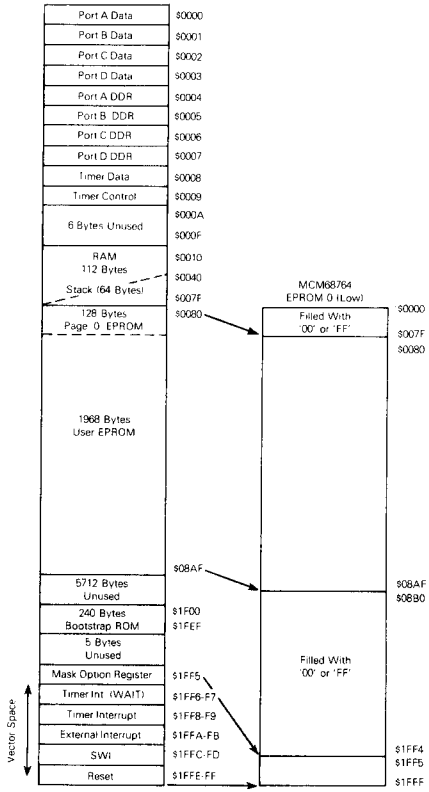


Figure 6-5. MC1468705G2 Program Memory Location In MCM68764

APPENDIX B
RASM05 MACRO ASSEMBLER SYNTAX AND DIRECTIVES

B.1 ASSEMBLY LANGUAGE SYNTAX AND ASSEMBLER DIRECTIVES

This appendix provides information concerning the assembly language syntax and assembler directive for the M6805 HMOS/M146805 CMOS Family. This information is more thoroughly discussed in *Macro Assemblers Reference Manual* M68MASR(D2) for M6800, 6801, 6805, and 6809; Motorola Literature Distribution Center, Phoenix, Az.

M6805 Family assembly language source statements follow the same format as M6800 source statements. See *Macro Assembler Reference Manual* M68MASR(D2) for detailed MC6805 HMOS/M146805 CMOS Family syntax. Highlights of syntax and assembler directives are discussed in the following paragraphs.

B.2 OPERATION FIELD SYNTAX

All instruction mnemonics for the M6805 HMOS/M146805 CMOS Family are three, four, or five characters long. Examples are:

LDA
JSR
INC
BHCC
BRSET

If the accumulator or index register is used as the operand of read/modify/write instructions, then the register is appended to the operation field. For example:

NEGA
RORX
INCX
DECA
TSTA

B.3 OPERAND FIELD SYNTAX

B.3.1 Inherent

Inherent instructions are the only type which do not include information in the operand field. All information necessary is incorporated in the operation field. Some examples are listed below. Note that an "A" or an "X" is added to the opcode for the register reference inherent instructions.

```
RTS
CLC
INCA
RORA
INCX
RORX
```

B.3.2 Immediate

The immediate value appears in the operand field preceded by a "#". Example:

```
LDA     #30
LDX     #$49
CPX     #$FF
LDA     #ADDR
```

B.3.3 Direct Addressing

The direct address appears in the operand field. If, on any pass through the source program, the assembler finds an unresolved (undefined) forward reference, the longer extended adressing mode is chosen instead of the direct addressing mode even if the address is subsequently found to be on page zero. To ensure direct addressing for direct variables, always define the variable before using it. In read/modify/write instructions all addresses are assumed to be direct since extended addressing is illegal with this mode. Examples:

```
LDA     CAT
STA     $30
CPX     DOG
ROL     $01
```

Where CAT and DOG have addresses <$100.

B.3.4 Extended Addressing

The extended address appears in the operand field. This mode is only legal when executing register/memory instructions. Examples:

```
LDA    BIG
LDA    $325
STA    COW
```

Where BIG and COW have addresses >$100.

B.3.5 Indexed — No Offset

The characters comma and X appear in the operand field. For example:

```
LDA    ,X
COM    ,X
STA    ,X
INC    ,X
TST    ,X
```

B.3.6 Indexed — One Byte Offset

The offset appears followed by a comma and "X". The offset must have a value <$100. Examples:

```
LDA    3, X
LDA    TABLE, X
INC    50, X
```

Where TABLE <$100.

B.3.7 Indexed — Two Byte Offset

The offset appears followed by a comma and "X". The offset would normally have a value >$100. Examples:

```
LDA    300, X
LDA    ZOT, X
COM    500, X
```

Where ZOT >$100.

B.3.8 Bit Set/Clear

The bit set and clear instructions contain the bit number followed by a comma and the address. Examples:

```
BSET   3, CAT
BCLR   4, $30
BCLR   5, DOG
```

Where CAT and DOG are <$100.

B.3.9 Bit Test and Branch

The bit test and branch instructions contain the bit number, a comma, the address to be tested, a comma, and the location to branch to if the test was successful. Examples:

```
PIG    BRSET   3, CAT, DOG
DOG    BRCLR   4, CAT, PIG
```

Where CAT <$100, DOG and PIG are relative addresses similar to those explained in the next paragraph.

B.3.10 Relative Addressing

The operand field contains the label of the address to be loaded into the program counter if the branch is taken. The branch address must be in the range −126 to +129. Examples:

```
BEQ    CAT
BNE    DOG
BRA    PIG
```

B.4 ASSEMBLER DIRECTIVE SUMMARY

The assembler directives are instructions to the assembler rather than instructions which are directly translated into object code. Detailed descriptions are provided in the M68MASR(D2) reference manual.

B.4.1 Assembly Control Directives

END	Program end
FAIL	Programmer generated errors
NAM	Assign program name
ORG	Origin program counter

B.4.6. Listing Control Directives

OPT ABS	Select absolute MDOS-loadable object output
OPT CL	Print conditional assembly directives
OPT NOCL	Don't print conditional assembly directives
OPT CMO	Allow CMOS instructions STOP and WAIT (M146805 CMOS only)
OPT NOCMO	Don't allow CMOS instructions STOP and WAIT (M146805 CMOS only)
OPT CRE	Print cross reference table
OPT G	Print generated lines of FCB, FCC, and FDB directives
OPT NOG	Don't print generated lines of FCB, FCC, and FDB directives
OPT L	Print source listing from this point
OPT NOL	Inhibit printing of source listing from this point
OPT LLE = n	Change line length
OPT LOAD	Select absolute EXORciser-loadable object output
OPT M	Creat object output in memory
OPT MC	Print macro calls
OPT NOMC	Don't print macro calls
OPT MD	Print macro definitions
OPT NOMD	Don't print macro definitions
OPT MEX	Print macro expansions
OPT NOMEX	Don't print macro expansions
OPT O	Create object output file
OPT NOO	Do not create object output file
OPT P = n	Change page length
OPT NOP	Inhibit paging and printing of headings
OPT REL	Select relocatable object output
OPT S	Print symbol table
OPT SE	Print user-supplied sequence numbers
OPT U	Print unassembled code from conditional directives
OPT NOU	Don't print unassembled code from conditional directives
PAGE	Print subsequent statements on top of next page
SPC	Skip lines
TTL	Initialize heading for source listing

APPENDIX C
INSTRUCTION SET
DETAILED DEFINITION

C.1 INTRODUCTION

In the pages that follow this section, the various accumulator and memory operations, together with the respective mnemonic, provides a heading for each of the executable instructions. The STOP and WAIT instructions apply only to the M146805 CMOS Family. The pages are arranged in alphabetical order of the mnemonic. A brief description of the operation is provided along with other applicable pertinent information, including: condition code status, Boolean formula, source forms, usable addressing modes, number of execution cycles (for both HMOS and CMOS), number of bytes required, and the opcode for each usable addressing mode. Paragraph C.2 contains a listing of the various nomenclature (abbreviations and signs) used in the operations.

C.2 NOMENCLATURE

The following nomenclature is used in the executable instructions which follow this paragraph.

(a) Operators:

( )	indirection, i.e., (SP) means the value pointed to by SP
←	is loaded with (read: "gets")
•	boolean AND
v	boolean (inclusive) OR
⊕	boolean EXCLUSIVE OR
~	boolean NOT
–	negation (twos complement)

(b) Registers in the MPU:

ACCA	Accumulator (shown as A in Boolean formula for condition codes and source forms)
CC	Condition Code Register
X	Index Register
PC	Program Counter
PCH	Program Counter High Byte
PCL	Program Counter Low Byte
SP	Stack Pointer

(c) Memory and Addressing:

M	Contents of any memory location (one byte)
Rel	Relative address (i.e., the twos complement number stored in the second byte of machine code in a branch instruction)

(d) Bits in the Condition Code Register:

C	Carry/Borrow, Bit 0
Z	Zero Indicator, Bit 1
N	Negative Indicator, Bit 2
I	Interrupt Mask, Bit 3
H	Half Carry Indicator, Bit 4

(e) Status of Individual Bits BEFORE Execution of an Instruction

An	Bit n of ACCA (n = 7, 6, 5, 4, 3, 2, 1, 0)
Xn	Bit n of X (n = 7, 6, 5, 4, 3, 2, 1, 0)
Mn	Bit n of M (n = 7, 6, 5, 4, 3, 2, 1, 0). In read/modify/write instructions, Mn is used to represent bit n of M, A or X.

(f) Status of Individual Bits AFTER Execution of an Instruction:

Rn	Bit n of the result (n = 7, 6, 5, 4, 3, 2, 1, 0)

(g) Source Forms:

P	Operands with IMMediate, DIRect, EXTended and INDexed (0, 1, 2 byte offset) addressing modes
Q	Operands with DIRect, INDexed (0 and 1 byte offset) addressing modes
dd	Relative operands
DR	Operands with DIRect addressing mode only.

(h) iff abbreviation for if-and-only-if.

ADC

Add with Carry

ADC

Operation: ACCA ← ACCA + M + C

Description: Adds the contents of the C bit to the sum of the contents of ACCA and M, and places the result in ACCA.

Condition Codes:

- H: Set if there was a carry from bit 3; cleared otherwise.
- I: Not affected.
- N: Set if the most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- C: Set if there was a carry from the most significant bit of the result; cleared otherwise.

Boolean Formulae for Condition Codes:

$H = A3 \cdot M3 \vee M3 \cdot R3 \vee R3 \cdot A3$

$N = R7$

$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

$C = A7 \cdot M7 \vee M7 \cdot \overline{R7} \vee \overline{R7} \cdot A7$

Source Form(s): ADC P

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate	2	2	2	A9
Direct	4	3	2	B9
Extended	5	4	3	C9
Indexed 0 Offset	4	3	1	F9
Indexed 1-Byte	5	4	2	E9
Indexed 2-Byte	6	5	3	D9

ADD

Add

ADD

Operation: ACCA ← ACCA + M

Description: Adds the contents of ACCA and the contents of M and places the result in ACCA.

Condition Codes:

- H: Set if there was a carry from bit 3; cleared otherwise.
- I: Not affected.
- N: Set if the most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- C: Set if there was a carry from the most significant bit of the result; cleared otherwise.

Boolean Formulae for Condition Codes:

$H = A3 \cdot M3 \vee M3 \cdot R3 \vee R3 \cdot A3$
$N = R7$
$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$
$C = A7 \cdot M7 \vee M7 \cdot \overline{R7} \vee \overline{R7} \cdot A7$

Source Form(s): ADD P

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate	2	2	2	AB
Direct	4	3	2	BB
Extended	5	4	3	CB
Indexed 0 Offset	4	3	1	FB
Indexed 1-Byte	5	4	2	EB
Indexed 2-Byte	6	5	3	DB

AND

Logical AND

AND

Operation: ACCA ← ACCA • M

Description: Performs logical AND between the contents of ACCA and the contents of M and places the result in ACCA. Each bit of ACCA after the operation will be the logical AND result of the corresponding bits of M and of ACCA before the operation.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- C: Not affected.

Boolean Formulae for Condition Codes:

$N = \overline{R7}$

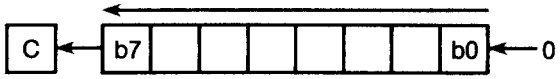
$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

Source Form(s): AND P

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate	2	2	2	A4
Direct	4	3	2	B4
Extended	5	4	3	C4
Indexed 0 Offset	4	3	1	F4
Indexed 1-Byte	5	4	2	E4
Indexed 2-Byte	6	5	3	D4

ASL — Arithmetic Shift Left — ASL

Operation:



Description: Shifts all bits of ACCA, X, or M one place to the left. Bit 0 is loaded with a zero. The C bit is loaded from the most significant bit of ACCA, X or M.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- C: Set if, before the operation, the most significant bit of ACCA, X or M was set; cleared otherwise.

Boolean Formulae for Condition Codes:

$N = \overline{R7}$

$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

C = b7 (before operation)

Comments: Same opcode as LSL

Source Form(s): ASL Q, ASLA, ASLX

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator	4	3	1	48
Index Register	4	3	1	58
Immediate				
Direct	6	5	2	38
Extended				
Indexed 0 Offset	6	5	1	78
Indexed 1-Byte	7	6	2	68
Indexed 2-Byte				

ASR — Arithmetic Shift Right — ASR

Operation:

Description: Shifts all bits of ACCA, X or M one place to the right. Bit 7 is held constant. Bit 0 is loaded into the C bit.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- C: Set if, before the operation, the least significant bit of ACCA, X or M was set; cleared otherwise.

Boolean Formulae for Condition Codes:

$N = \overline{R7}$

$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

C = b0 (before operation)

Source Form(s): ASR Q, ASRA, ASRX

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator	4	3	1	47
Index Register	4	3	1	57
Immediate				
Direct	6	5	2	37
Extended				
Indexed 0 Offset	6	5	1	77
Indexed 1-Byte	7	6	2	67
Indexed 2-Byte				

BCC

Branch If Carry Clear

BCC

Operation: PC ← PC + 0002 + Rel iff C = 0

Description: Tests the state of the C bit and causes a branch iff C is clear. See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Comments: Same opcode as BHS

Source Form(s): BCC dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	24
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BCLR n

Clear Bit In Memory

BCLR n

Operation: Mn ← 0

Description: Clear bit n (n = 0, 7) in location M. All other bits in M are unaffected.

Condition Codes: Not affected.

Source Form(s): BCLR n, DR

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate				
Direct	7	5	2	11 + 2•n
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BCS — Branch if Carry Set — BCS

Operation: PC ← PC + 0002 + Rel iff C = 1

Description: Tests the state of the C bit and causes a branch iff C is set. See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Comments: Same opcode as BLO

Source Form(s): BCS dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	25
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BEQ

Branch if Equal

BEQ

Operation: PC ← PC + 0002 + Rel iff Z = 1

Description: Tests the state of the Z bit and causes a branch iff Z is set. Following a compare or subtract instruction BEQ will cause a branch if the arguments were equal. See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Source Form(s): BEQ dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	27
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BHCC Branch if Half Carry Clear BHCC

Operation: PC ← PC + 0002 + Rel iff H = 0

Description: Tests the state of the H bit and causes a branch iff H is clear. See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Source Form(s): BHCC dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	28
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BHCS Branch if Half Carry Set BHCS

Operation: PC ← PC + 0002 + Rel iff H = 1

Description: Tests the state of the H bit and causes a branch iff H is set. See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Source Form(s): BHCS dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	29
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BHI

Branch If Higher

BHI

Operation: PC ← PC + 0002 + Rel iff (C v Z) = 0
i.e., if ACCA > M (unsigned binary numbers)

Description: Causes a branch iff both C and Z are zero. If the BHI instruction is executed immediately after execution of either of the CMP or SUB instructions, the branch will occur if and only if the unsigned binary number represented by the minuend (i.e., ACCA) was greater than the unsigned binary number represented by the subtrahend (i.e., M). See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Source Form(s): BHI dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	22
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BHS

Branch If Higher or Same

BHS

Operation: PC ← PC + 0002 + Rel iff C = 0

Description: Following an unsigned compare or subtract, BHS will cause a branch iff the register was higher than or the same as the location in memory. See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Comments: Same opcode as BCC

Source Form(s): BHS dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	24
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BIH

Branch if Interrupt Line is High

BIH

Operation: PC ← PC + 0002 + Rel iff $\overline{INT}$ = 1

Description: Tests the state of the external interrupt pin and branches iff it is high. See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Comments: In systems not using interrupts, this instruction and BIL can be used to create an extra I/O input bit. This instruction does NOT test the state of the interrupt mask bit nor does it indicate whether an interrupt is pending. All it does is indicate whether the $\overline{INT}$ line is high.

Source Form(s): BIH dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	2F
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BIL

Branch if Interrupt Line is Low

BIL

Operation: PC ← PC + 0002 + Rel iff $\overline{INT}$ = 0

Description: Tests the state of the external interrupt pin and branches iff it is low. See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Comments: In systems not using interrupts, this instruction and BIH can be used to create an extra I/O input bit. This instruction does NOT test the state of the interrupt mask bit nor does it indicate whether an interrupt is pending. All it does is indicate whether the $\overline{INT}$ line is Low.

Source Form(s): BIL dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	2E
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BIT — Bit Test Memory with Accumulator — BIT

Operation: ACCA • M

Description: Performs the logical AND comparison of the contents of ACCA and the contents of M and modifies the condition codes accordingly. The contents of ACCA and M are unchanged.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the result of the AND is set; cleared otherwise.
- Z: Set if all bits of the result of the AND are cleared; cleared otherwise.
- C: Not affected.

Boolean Formulae for Condition Codes:

$N = R7$

$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

Source Form(s): BIT P

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate	2	2	2	A5
Direct	4	3	2	B5
Extended	5	4	3	C5
Indexed 0 Offset	4	3	1	F5
Indexed 1-Byte	5	4	2	E5
Indexed 2-Byte	6	5	3	D5

BLO

Branch If Lower

BLO

Operation: PC ← PC + 0002 + Rel iff C = 1

Description: Following a compare, BLO will branch iff the register was lower than the memory location. See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Comments: Same opcode as BCS

Source Form(s): BLO dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	25
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BLS

Branch if Lower or Same

BLS

Operation: PC ← PC + 0002 + Rel iff (C v Z) = 1
i.e., if ACCA ≤ M (unsigned binary numbers)

Description: Causes a branch if (C is set) OR (Z is set). If the BLS instruction is executed immediately after execution of either of the instructions CMP or SUB, the branch will occur if and only if the unsigned binary number represented by the minuend (i.e., ACCA) was less than or equal to the unsigned binary number represented by the subtrahend (i.e., M). See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Source Form(s): BLS dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	23
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BMC

Branch if Interrupt Mask is Clear

BMC

Operation: PC ← PC + 0002 + Rel iff I = 0

Description: Tests the state of the I bit and causes a branch iff I is clear. See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Comments: This instruction does NOT branch on the condition of the external interrupt line. The test is performed only on the interrupt mask bit.

Source Form(s): BMC dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	2C
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BMI

Branch if Minus

BMI

Operation: PC ← PC + 0002 + Rel iff N = 1

Description: Tests the state of the N bit and causes a branch iff N is set. See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Source Form(s) BMI dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	2B
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BMS — Branch if Interrupt Mask Bit is Set — BMS

Operation: PC ← PC + 0002 + Rel iff I = 1

Description: Tests the state of the I bit and causes a branch iff I is set. See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Comments: This instruction does NOT branch on the condition of the external interrupt line. The test is performed only on the interrupt mask bit.

Source Form(s): BMS dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	2D
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BNE

Branch if Not Equal

BNE

Operation: PC ← PC + 0002 + Rel iff Z = 0

Description: Tests the state of the Z bit and causes a branch iff Z is clear. Following a compare or subtract instruction BNE will cause a branch if the arguments were different. See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Source Form(s): BNE dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	26
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BPL

Branch if Plus

BPL

Operation: PC ← PC + 0002 + Rel iff N = 0

Description: Tests the state of the N bit and causes a branch iff N is clear. See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Source Form(s): BPL dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	2A
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BRA

Branch Always

BRA

Operation: PC ← PC + 0002 + Rel

Description: Unconditional branch to the address given by the foregoing formula, in which Rel is the relative address stored as a twos complement number in the second byte of machine code corresponding to the branch instruction.

NOTE: The source program specifies the destination of any branch instruction by its absolute address, either as a numerical value or as a symbol or expression which can be evaluated by the assembler. The assembler obtains the relative address Rel from the absolute address and the current value of the program counter.

Condition Codes: Not affected.

Source Form(s): BRA dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	20
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BRCLR n Branch if Bit n is Clear BRCLR n

Operation: PC ← PC + 0003 + Rel iff bit n of M is zero

Description: Tests bit n (n = 0, 7) of location M and branches iff the bit is clear.

Condition Codes:

H: Not affected.
I: Not affected.
N: Not affected.
Z: Not affected.
C: Set if Mn = 1; cleared otherwise.

Boolean Formulae for Condition Codes:

C = Mn

Comments: The C bit is set to the state of the bit tested. Used with an appropriate rotate instruction, this instruction is an easy way to do serial to parallel conversions.

Source Form(s): BRCLR n, DR, dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	10	5	3	01+2•n
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BRN — Branch Never — BRN

Description: Never branches. Branch never is a 2 byte 4 cycle NOP.

Condition Codes: Not affected.

Comments: BRN is included here to demonstrate the nature of branches on the M6805 HMOS/M146805 CMOS Family. Each branch is matched with an inverse that varies only in the least significant bit of the opcode. BRN is the inverse of BRA. This instruction may have some use during program debugging.

Source Form(s): BRN dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	21
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BRSET n Branch if Bit n is Set BRSET n

Operation: PC ← PC + 0003 + Rel iff Bit n of M is not zero

Description: Tests bit n (n = 0, 7) of location M and branches iff the bit is set.

Condition Codes:

H: Not affected.
I: Not affected.
N: Not affected.
Z: Not affected.
C: Set if Mn = 1; cleared otherwise.

Boolean Formulae for Condition Codes:

C = Mn

Comments: The C bit is set to the state of the bit tested. Used with an appropriate rotate instruction, this instruction is an easy way to provide serial to parallel conversions.

Source Form(s): BRSET n, DR, dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	10	5	3	2•n
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BSET n — Set Bit in Memory — BSET n

Operation: Mn ← 1

Description: Set bit n (n = 0, 7) in location M. All other bits in M are unaffected.

Condition Codes: Not affected.

Source Form(s): BSET n, DR

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate				
Direct	7	5	2	10 + 2•n
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BSR

Branch to Subroutine

BSR

Operation: PC ← PC + 0002
(SP) ← PCL; SP ← SP − 0001
(SP) ← PCH; SP ← SP − 0001
PC ← PC + Rel

Description: The program counter is incremented by 2. The least (low) significant byte of the program counter contents is pushed onto the stack. The stack pointer is then decremented (by one). The most (high) significant byte of the program counter contents is then pushed onto the stack. Unused bits in the program counter high byte are stored as 1s on the stack. The stack pointer is again decremented (by one). A branch then occurs to the location specified by the relative offset. See the BRA instruction for details of the branch execution.

Condition Codes: Not affected.

Source Form(s): BSR dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	8	6	2	AD
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

CLC

Clear Carry Bit

CLC

Operation: C bit ← 0

Description: Clears the carry bit in the processor condition code register.

Condition Codes:

H: Not affected.
I: Not affected.
N: Not affected.
Z: Not affected.
C: Cleared.

Boolean Formulae for Condition Codes:

C = 0

Source Form(s): CLC

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent	2	2	1	98
Relative				
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

CLI

Clear Interrupt Mask Bit

CLI

Operation: I bit ← 0

Description: Clears the interrupt mask bit in the processor condition code register. This enables the microprocessor to service interrupts. Interrupts that were pending while the I bit was set will now begin to have effect.

Condition Codes:

H: Not affected.
I: Cleared
N: Not affected.
Z: Not affected.
C: Not affected.

Boolean Formulae for Condition Codes:

I = 0

Source Form(s): CLI

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent	2	2	1	9A
Relative				
Accumulator				
Index Registers				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

CLR — Clear — CLR

Operation: X ← 00 or,
ACCA ← 00 or,
M ← 00

Description: The contents of ACCA, X, or M are replaced with zeroes.

Condition Codes:

H:	Not affected.
I:	Not affected.
N:	Cleared.
Z:	Set.
C:	Not affected.

Boolean Formulae for Condition Codes:

N = 0
Z = 1

Source Form(s): CLR Q, CLRA, CLRX

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator	4	3	1	4F
Index Register	4	3	1	5F
Immediate				
Direct	6	5	2	3F
Extended				
Indexed 0 Offset	6	5	1	7F
Indexed 1-Byte	7	6	2	6F
Indexed 2-Byte				

CMP — Compare Accumulator with Memory — CMP

Operation: ACCA − M

Description: Compares the contents of ACCA and the contents of M and sets the condition codes, which may then be used for controlling the conditional branches. Both operands are unaffected.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the result of the subtraction is set; cleared otherwise.
- Z: Set if all bits of the result of the subtraction are cleared; cleared otherwise.
- C: Set if the absolute value of the contents of memory is larger than the absolute value of the accumulator; cleared otherwise.

Boolean Formulae for Condition Codes:

$N = R7$

$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

$C = \overline{A7} \cdot M7 \vee M7 \cdot R7 \vee R7 \cdot \overline{A7}$

Source Form(s): CMP P

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate	2	2	2	A1
Direct	4	3	2	B1
Extended	5	4	3	C1
Indexed 0 Offset	4	3	1	F1
Indexed 1-Byte	5	4	2	E1
Indexed 2-Byte	6	5	3	D1

COM — Complement — COM

Operation: X ← ~X = $FF − X or,
ACCA ← ~ACCA = $FF − ACCA or,
M ← ~M = $FF − M

Description: Replaces the contents of ACCA, X, or M with the ones complement. Each bit of the operand is replaced with the complement of that bit.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- C: Set.

Boolean Formulae for Condition Codes:

$N = R7$
$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$
$C = 1$

Source Form(s): COM Q, COMA, COMX

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator	4	3	1	43
Index Register	4	3	1	53
Immediate				
Direct	6	5	2	33
Extended				
Indexed 0 Offset	6	5	1	73
Indexed 1-Byte	7	6	2	63
Indexed 2-Byte				

CPX — Compare Index Register with Memory — CPX

Operation: X − M

Description: Compares the contents of X to the contents of M and sets the condition codes, which may then be used for controlling the conditional branches. Both operands are unaffected.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the result of the subtraction is set; cleared otherwise.
- Z: Set if all bits of the result of the subtraction are cleared; cleared otherwise.
- C: Set if the absolute value of the contents of memory is larger than the absolute value of the index register; cleared otherwise.

Boolean Formulae for Condition Codes:

$N = R7$

$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

$C = \overline{X7} \cdot M7 \vee M7 \cdot R7 \vee R7 \cdot \overline{X7}$

Source Form(s): CPX P

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate	2	2	2	A3
Direct	4	3	2	B3
Extended	5	4	3	C3
Indexed 0 Offset	4	3	1	F3
Indexed 1-Byte	5	4	2	E3
Indexed 2-Byte	6	5	3	D3

DEC — Decrement — DEC

Operation: X ← X − 01 or,
ACCA ← ACCA − 01 or,
M ← M − 01

Description: Subtract one from the contents of ACCA, X, or M. The N and Z bits are set or reset according to the result of this operation. The C bit is not affected by this operation.

Condition Codes:

H: Not affected.
I: Not affected.
N: Set if the most significant bit of the result is set; cleared otherwise.
Z: Set if all bits of the result are cleared; cleared otherwise.
C: Not affected.

Boolean Formulae for Condition Codes:

$N = \overline{R7}$
$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

Source Form(s): DEC Q, DECA, DECX, (DEX is recognized by the Assembler as DECX)

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator	4	3	1	4A
Index Register	4	3	1	5A
Immediate				
Direct	6	5	2	3A
Extended				
Indexed 0 Offset	6	5	1	7A
Indexed 1-Byte	7	6	2	6A
Indexed 2-Byte				

EOR — Exclusive Or Memory with Accumulator — EOR

Operation: ACCA ← ACCA ⊕ M

Description: Performs the logical EXCLUSIVE OR between the contents of ACCA and the contents of M, and places the result in ACCA. Each bit of ACCA after the operation will be the logical EXCLUSIVE OR of the corresponding bit of M and ACCA before the operation.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- C: Not affected.

Boolean Formulae for Condition Codes:

$N = R7$

$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

Source Form(s): EOR P

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate	2	2	2	A8
Direct	4	3	2	B8
Extended	5	4	3	C8
Indexed 0 Offset	4	3	1	F8
Indexed 1-Byte	5	4	2	E8
Indexed 2-Byte	6	5	3	D8

Operation: X ← X + 01 or,
ACCA ← ACCA + 01 or,
M ← M + 01

Description: Add one to the contents of ACCA, X, or M. The N and Z bits are set or reset according to the result of this operation. The C bit is not affected by this operation.

Condition Codes:

H: Not affected.
I: Not affected.
N: Set if the most significant bit of the result is set; cleared otherwise.
Z: Set if all bits of the result are cleared; cleared otherwise.
C: Not affected.

Boolean Formulae for Condition Codes:

$N = R7$
$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

Source Form(s): INC Q, INCA, INCX (INX is recognized by the Assembler as INCX)

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator	4	3	1	4C
Index Register	4	3	1	5C
Immediate				
Direct	6	5	2	3C
Extended				
Indexed 0 Offset	6	5	1	7C
Indexed 1-Byte	7	6	2	6C
Indexed 2-Byte				

JMP

Jump

JMP

Operation: PC ← effective address

Description: A jump occurs to the instruction stored at the effective address. The effective address is obtained according to the rules for EXTended, DIRect or INDexed addressing.

Condition Codes: Not affected.

Source Form(s): JMP P

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate				
Direct	3	2	2	BC
Extended	4	3	3	CC
Indexed 0 Offset	3	2	1	FC
Indexed 1-Byte	4	3	2	EC
Indexed 2-Byte	5	4	3	DC

JSR — Jump to Subroutine — JSR

Operation: PC ← PC + N
(SP) ← PCL; SP ← SP − 0001
(SP) ← PCH ; SP ← SP − 0001
PC ← effective address

Description: The program counter is incremented by N (N = 1, 2, or 3 depending on the addressing mode), and is then pushed onto the stack (least significant byte first). Unused bits in the program counter high byte are stored as 1s on the stack. The stack pointer points to the next empty location on the stack. A jump occurs to the instruction stored at the effective address. The effective address is obtained according to the rules for EXTended, DIRect, or INDexed addressing.

Condition Codes: Not affected.

Source Form(s): JSR P

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate				
Direct	7	5	2	BD
Extended	8	6	3	CD
Indexed 0 Offset	7	5	1	FD
Indexed 1-Byte	8	6	2	ED
Indexed 2-Byte	9	7	3	DD

LDA — Load Accumulator from Memory — LDA

Operation: ACCA ← M

Description: Loads the contents of memory into the accumulator. The condition codes are set according to the data.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the accumulator is set; cleared otherwise.
- Z: Set if all bits of the accumulator are cleared; cleared otherwise.
- C: Not affected.

Boolean Formulae for Condition Codes:

$N = R7$

$Z = \overline{R7} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

Source Form(s): LDA P

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate	2	2	2	A6
Direct	4	3	2	B6
Extended	5	4	3	C6
Indexed 0 Offset	4	3	1	F6
Indexed 1-Byte	5	4	2	E6
Indexed 2-Byte	6	5	3	D6

LDX — Load Index Register from Memory — LDX

Operation: X ← M

Description: Loads the contents of memory into the index register. The condition codes are set according to the data.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the index register is set; cleared otherwise.
- Z: Set if all bits of the index register are cleared; cleared otherwise.
- C: Not affected.

Boolean Formulae for Condition Codes:

$N = R7$

$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

Source Form(s): LDX P

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate	2	2	2	AE
Direct	4	3	2	BE
Extended	5	4	3	CE
Indexed 0 Offset	4	3	1	FE
Indexed 1-Byte	5	4	2	EE
Indexed 2-Byte	6	5	3	DE

LSL — Logical Shift Left — LSL

Operation:

Description: Shifts all bits of the ACCA, X or M one place to the left. Bit 0 is loaded with a zero. The C bit is loaded from the most significant bit of ACCA, X or M.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- C: Set if, before the operation, the most significant bit of ACCA, X or M was set; cleared otherwise.

Boolean Formulae for Condition Codes:

$N = R7$

$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

C = b7 (before operation)

Comments: Same as ASL

Source Form(s): LSL Q, LSLA, LSLX

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator	4	3	1	48
Index Register	4	3	1	58
Immediate				
Direct	6	5	2	38
Extended				
Indexed 0 Offset	6	5	1	78
Indexed 1-Byte	7	6	2	68
Indexed 2-Byte				

LSR — Logical Shift Right — LSR

Operation:

Description: Shifts all bits of ACCA, X or M one place to the right. Bit 7 is loaded with a zero. Bit 0 is loaded into the C bit.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Cleared.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- C: Set if, before the operation, the least significant bit of ACCA, X or M was set; cleared otherwise.

Boolean Formulae for Condition Codes:

$N = 0$

$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

C = b0 (before operation)

Source Form(s): LSR Q, LSRA, LSRX

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator	4	3	1	44
Index Register	4	3	1	54
Immediate				
Direct	6	5	2	34
Extended				
Indexed 0 Offset	6	5	1	74
Indexed 1-Byte	7	6	2	64
Indexed 2-Byte				

NEG — Negate — NEG

Operation: X ← − X (i.e., 00 − X) or,
ACCA ← − ACCA (i.e., 00 − ACCA) or,
M ← − M (i.e., 00 − M)

Description: Replaces the contents of ACCA, X or M with its twos complement. Note that $80 is left unchanged.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- C: Set if there would be a borrow in the implied subtraction from zero; the C bit will be set in all cases except when the contents of ACCA, X or M before the NEG is 00.

Boolean Formulae for Condition Codes:

N = R7
Z = $\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}$
C = R7vR6vR5vR4vR3vR2vR1vR0

Source Form(s): NEG Q, NEGA, NEGX

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator	4	3	1	40
Index Register	4	3	1	50
Immediate				
Direct	6	5	2	30
Extended				
Indexed 0 Offset	6	5	1	70
Indexed 1-Byte	7	6	2	60
Indexed 2-Byte				

NOP — No Operation — NOP

Description: This is a single-byte instruction which causes only the program counter to be incremented. No other registers are changed.

Condition Codes: Not affected.

Source Form(s): NOP

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent	2	2	1	9D
Relative				
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

ORA — Inclusive OR — ORA

Operation: ACCA ← ACCA v M

Description: Performs logical OR between the contents of ACCA and the contents of M and places the result in ACCA. Each bit of ACCA after the operation will be the logical (inclusive) OR result of the corresponding bits of M and ACCA before the operation.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- C: Not affected.

Boolean Formulae for Condition Codes:

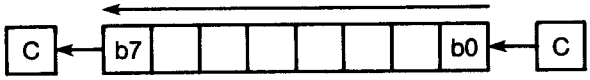
$$N = \overline{R7}$$
$$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$$

Source Form(s): ORA P

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate	2	2	2	AA
Direct	4	3	2	BA
Extended	5	4	3	CA
Indexed 0 Offset	4	3	1	FA
Indexed 1-Byte	5	4	2	EA
Indexed 2-Byte	6	5	3	DA

ROL — Rotate Left thru Carry — ROL

Operation:



Description: Shifts all bits of the ACCA, X, or M one place to the left. Bit 0 is loaded from the C bit. The C bit is loaded from the most significant bit of ACCA, X, or M.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- C: Set if, before the operation, the most significant bit of ACCA, X or M was set; cleared otherwise.

Boolean Formulae for Condition Codes:

$N = R7$

$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

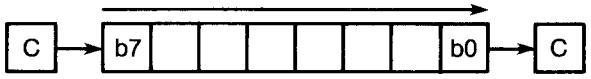
C = b7 (before operation)

Source Form(s): ROL Q, ROLA, ROLX

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator	4	3	1	49
Index Register	4	3	1	59
Immediate				
Direct	6	5	2	39
Extended				
Indexed 0 Offset	6	5	1	79
Indexed 1-Byte	7	6	2	69
Indexed 2-Byte				

ROR — Rotate Right Thru Carry — ROR

Operation:



Description: Shifts all bits of ACCA, X, or M one place to the right. Bit 7 is loaded from the C bit. Bit 0 is loaded into the C bit.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- C: Set if, before the operation, the least significant bit of ACCA, X or M was set; cleared otherwise.

Boolean Formulae for Condition Codes:

$N = \overline{R7}$

$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

C = b0 (before operation)

Source Form(s): ROR Q, RORA, RORX

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator	4	3	1	46
Index Register	4	3	1	56
Immediate				
Direct	6	5	2	36
Extended				
Indexed 0 Offset	6	5	1	76
Indexed 1-Byte	7	6	2	66
Indexed 2-Byte				

RSP

Reset Stack Pointer

RSP

Operation: SP ← $7F

Description: Resets the stack pointer to the top of the stack.

Condition Codes: Not affected.

Source Form(s): RSP

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent	2	2	1	9C
Relative				
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

RTI

Return from Interrupt

RTI

Operation:
SP ← SP + 0001 ; CC ← (SP)
SP ← SP + 0001 ; ACCA ← (SP)
SP ← SP + 0001 ; X ← (SP)
SP ← SP + 0001 ; PCH ← (SP)
SP ← SP + 0001 ; PCL ← (SP)

Description: The condition codes, accumulator, index register, and the program counter are restored according to the state previously saved on the stack. Note that the interrupt mask bit (I bit) will be reset if and only if the corresponding bit stored on the stack is zero.

Condition Codes: Set or cleared according to the first byte pulled from the stack.

Source Form(s): RTI

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent	9	9	1	80
Relative				
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

RTS

Return from Subroutine

RTS

Operation: SP ← SP + 0001 ; PCH ← (SP)
SP ← SP + 0001 ; PCL ← (SP)

Description: The stack pointer is incremented (by one). The contents of the byte of memory, pointed to by the stack pointer, are loaded into the high byte of the program counter. The stack pointer is again incremented (by one). The byte pointed to by the stack pointer is loaded into the low byte of the program counter.

Condition Codes: Not affected.

Source Form(s): RTS

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent	6	6	1	81
Relative				
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

SBC

Subtract with Carry

SBC

Operation: ACCA ← ACCA − M − C

Description: Subtracts the contents of M and C from the contents of ACCA, and places the result in ACCA.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- C: Set if the absolute value of the contents of memory plus the previous carry is larger than the absolute value of the accumulator; cleared otherwise.

Boolean Formulae for Condition Codes:

$N = \overline{R7}$

$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

$C = \overline{A7} \cdot M7 \vee M7 \cdot R7 \vee R7 \cdot \overline{A7}$

Source Form(s): SBC P

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate	2	2	2	A2
Direct	4	3	2	B2
Extended	5	4	3	C2
Indexed 0 Offset	4	3	1	F2
Indexed 1-Byte	5	4	2	E2
Indexed 2-Byte	6	5	3	D2

SEC

Set Carry Bit

SEC

Operation: C bit ← 1

Description: Sets the carry bit in the processor condition code register.

Condition Codes:

H: Not affected.
I: Not affected.
N: Not affected.
Z: Not affected.
C: Set.

Boolean Formulae for Condition Codes:

C = 1

Source Form(s): SEC

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent	2	2	1	99
Relative				
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

SEI

Set Interrupt Mask Bit

SEI

Operation: I bit ← 1

Description: Sets the interrupt mask bit in the processor condition code register. The microprocessor is inhibited from servicing interrupts, and will continue with execution of the instructions of the program until the interrupt mask bit is cleared.

Condition Codes:

H: Not affected.
I: Set
N: Not affected.
Z: Not affected.
C: Not affected.

Boolean Formulae for Condition Codes:

I = 1

Source Form(s): SEI

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent	2	2	1	9B
Relative				
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

STA — Store Accumulator in Memory — STA

Operation: M ← ACCA

Description: Stores the contents of ACCA in memory. The contents of ACCA remain the same.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the accumulator is set; cleared otherwise.
- Z: Set if all bits of the accumulator are clear; cleared otherwise.
- C: Not affected.

Boolean Formulae for Condition Codes:

$$N = \overline{A7}$$

$$Z = \overline{A7} \cdot \overline{A6} \cdot \overline{A5} \cdot \overline{A4} \cdot \overline{A3} \cdot \overline{A2} \cdot \overline{A1} \cdot \overline{A0}$$

Source Form(s): STA P

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate				
Direct	5	4	2	B7
Extended	6	5	3	C7
Indexed 0 Offset	5	4	1	F7
Indexed 1-Byte	6	5	2	E7
Indexed 2-Byte	7	6	3	D7

STOP — Enable $\overline{IRQ}$, Stop Oscillator — STOP

Description: Reduces power consumption by eliminating all dynamic power dissipation. Results in: (1) timer prescaler to clear, (2) disabling of timer interrupts, (3) timer interrupt flag bit to clear, (4) external interrupt request enabling, and (5) inhibiting of oscillator.

When $\overline{RESET}$ or $\overline{IRQ}$ input goes low: (1) oscillator is enabled, (2) a delay of 1920 instruction cycles allows oscillator to stabilize, (3) the interrupt request vector is fetched, and (4) service routine is executed.

External interrupts are enabled following the RTI command.

Condition Codes:

H: Not affected.
I: Cleared.
N: Not affected.
Z: Not affected.
C: Not affected.

Source Form(s): STOP

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent	—	2	1	8E
Relative				
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

STX — Store Index Register in Memory — STX

Operation: M ← X

Description: Stores the contents of X in memory. The contents of X remain the same.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the index register is set; cleared otherwise.
- Z: Set if all bits of the index register are clear; cleared otherwise.
- C: Not affected.

Boolean Formulae for Condition Codes:

$$N = X7$$
$$Z = \overline{X7} \cdot \overline{X6} \cdot \overline{X5} \cdot \overline{X4} \cdot \overline{X3} \cdot \overline{X2} \cdot \overline{X1} \cdot \overline{X0}$$

Source Form(s): STX P

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate				
Direct	5	4	2	BF
Extended	6	5	3	CF
Indexed 0 Offset	5	4	1	FF
Indexed 1-Byte	6	5	2	EF
Indexed 2-Byte	7	6	3	DF

SUB — Subtract — SUB

Operation: ACCA ← ACCA − M

Description: Subtracts the contents of M from the contents of ACCA and places the result in ACCA.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the results are cleared; cleared otherwise.
- C: Set if the absolute value of the contents of memory are larger than the absolute value of the accumulator; cleared otherwise.

Boolean Formulae for Condition Codes:

$N = R7$

$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

$C = \overline{A7} \cdot M7 \vee R7 \vee R7 \vee \overline{A7}$

Source Form(s): SUB P

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate	2	2	2	A0
Direct	4	3	2	B0
Extended	5	4	3	C0
Indexed 0 Offset	4	3	1	F0
Indexed 1-Byte	5	4	2	E0
Indexed 2-Byte	6	5	3	D0

Operation:
PC ← PC + 0001
(SP) ← PCL ; SP ← SP − 0001
(SP) ← PCH ; SP ← SP − 0001
(SP) ← X ; SP ← SP − 0001
(SP) ← ACCA ; SP ← SP − 0001
(SP) ← CC ; SP ← SP − 0001
I bit ← 1
PCH ← n − 0003
PCL ← n − 0002

Description: The program counter is incremented (by one). The program counter, index register and accumulator are pushed onto the stack. The condition code register bits are then pushed onto the stack with bits H, I, N, Z, and C going into bit positions 4 through 0 with the top three bits (7, 6 and 5) containing ones. The stack pointer is decremented by one after each byte is stored on the stack.

The interrupt mask bit is then set. The program counter is then loaded with the address stored in the software interrupt vector located at memory locations n − 0002 and n − 0003, where n is the address corresponding to a high state on all lines of the address bus.

Condition Codes:

H:	Not affected.
I:	Set.
N:	Not affected.
Z:	Not affected.
C:	Not affected.

Boolean Formulae for Condition Codes:
I = 1

Caution: This instruction is used by Motorola in some of its software products and may be unavailable for general use.

Source Form(s): SWI

Software Interrupt (Continued)

SWI

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent	11	10	1	83
Relative				
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

TAX — Transfer Accumulator to Index Register — TAX

Operation: X ← ACCA

Description: Loads the index register with the contents of the accumulator. The contents of the accumulator are unchanged.

Condition Codes: Not affected.

Source Form(s): TAX

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent	2	2	1	97
Relative				
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

TST

Test for Negative or Zero

TST

Operation: X – 00 or,
ACCA – 00 or,
M – 0

Description: Sets the condition codes N and Z according to the contents of ACCA, X, or M.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the contents of ACCA, X, or M is set; cleared otherwise.
- Z: Set if all bits of ACCA, X, or M are clear; cleared otherwise.
- C: Not affected.

Boolean Formulae for Condition Codes:

$N = M7$

$Z = \overline{M7} \cdot \overline{M6} \cdot \overline{M5} \cdot \overline{M4} \cdot \overline{M3} \cdot \overline{M2} \cdot \overline{M1} \cdot \overline{M0}$

Source Form(s): TST Q, TSTA, TSTX

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator	4	3	1	4D
Index Register	4	3	1	5D
Immediate				
Direct	6	4	2	3D
Extended				
Indexed 0 Offset	6	4	1	7D
Indexed 1-Byte	7	5	2	6D
Indexed 2-Byte				

TXA Transfer Index Register to Accumulator TXA

Operation: ACCA ← X

Description: Loads the accumulator with the contents of the index register. The contents of the index register are unchanged.

Condition Codes: Not affected.

Source Form(s): TXA

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent	2	2	1	9F
Relative				
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

WAIT — Enable Interrupt, Stop Processor — WAIT

Description: Reduces power consumption by eliminating dynamic power dissipation in all circuits except the timer and timer prescaler. Causes enabling of external interrupts and stops clocking or processor circuits.

Timer interrupts may be enabled or disabled by programmer prior to execution of WAIT.

When $\overline{\text{RESET}}$ or $\overline{\text{IRQ}}$ input goes low, or timer counter reaches zero with counter interrupt enabled: (1) processor clocks are enabled, and (2) interrupt request, or reset, or timer interrupt vector is fetched.

Interrupts are enabled following the RTI command.

Condition Codes:

H: Not affected.
I: Cleared.
N: Not affected.
Z: Not affected.
C: Not affected.

Source Form(s): WAIT

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent	—	2	1	8F
Relative				
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

APPENDIX D
INSTRUCTION SET
ALPHABETICAL LISTING

This appendix provides an alphabetical listing of the mnemonic instruction set, together with addressing modes used and the effects on the condition code register.

	Addressing Modes										Condition Codes				
Mnemonic	Inherent	Immediate	Direct	Extended	Relative	Indexed (No Offset)	Indexed (8 Bits)	Indexed (16 Bits)	Bit Set/ Clear	Bit Test & Branch	H	I	N	Z	C
ADC		X	X	X		X	X	X			Λ	•	Λ	Λ	Λ
ADD		X	X	X		X	X	X			Λ	•	Λ	Λ	Λ
AND		X	X	X		X	X	X			•	•	Λ	Λ	•
ASL	X		X			X	X				•	•	Λ	Λ	Λ
ASR	X		X			X	X				•	•	Λ	Λ	Λ
BCC					X						•	•	•	•	•
BCLR									X		•	•	•	•	•
BCS					X						•	•	•	•	•
BEQ					X						•	•	•	•	•
BHCC					X						•	•	•	•	•
BHCS					X						•	•	•	•	•
BHI					X						•	•	•	•	•
BHS					X						•	•	•	•	•
BIH					X						•	•	•	•	•
BIL					X						•	•	•	•	•
BIT		X	X	X		X	X	X			•	•	Λ	Λ	•
BLO					X						•	•	•	•	•
BLS					X						•	•	•	•	•
BMC					X						•	•	•	•	•
BMI					X						•	•	•	•	•
BMS					X						•	•	•	•	•
BNE					X						•	•	•	•	•
BPL					X						•	•	•	•	•
BRA					X						•	•	•	•	•
BRN					X						•	•	•	•	•
BRCLR										X	•	•	•	•	Λ
BRSET										X	•	•	•	•	Λ
BSET									X		•	•	•	•	•
BSR					X						•	•	•	•	•
CLC	X										•	•	•	•	0
CLI	X										•	0	•	•	•
CLR	X		X			X	X				•	•	0	1	•
CMP		X	X	X		X	X	X			•	•	Λ	Λ	Λ
COM	X		X			X	X				•	•	Λ	Λ	1
CPX		X	X	X		X	X	X			•	•	Λ	Λ	Λ

Mnemonic	Addressing Modes										Condition Codes				
	Inherent	Immediate	Direct	Extended	Relative	Indexed (No Offset)	Indexed (8 Bits)	Indexed (16 Bits)	Bit Set/ Clear	Bit Test & Branch	H	I	N	Z	C
DEC	X		X			X	X				•	•	Λ	Λ	•
EOR		X	X	X		X	X	X			•	•	Λ	Λ	•
INC	X		X			X	X				•	•	Λ	Λ	•
JMP			X	X		X	X	X			•	•	•	•	•
JSR			X	X		X	X	X			•	•	•	•	•
LDA		X	X	X		X	X	X			•	•	Λ	Λ	•
LDX		X	X	X		X	X	X			•	•	Λ	Λ	•
LSL	X		X			X	X				•	•	Λ	Λ	Λ
LSR	X		X			X	X				•	•	0	Λ	Λ
NEQ	X		X			X	X				•	•	Λ	Λ	Λ
NOP	X										•	•	•	•	•
ORA		X	X	X		X	X	X			•	•	Λ	Λ	•
ROL	X		X			X	X				•	•	Λ	Λ	Λ
RSP	X										•	•	•	•	•
RTI	X										?	?	?	?	?
RTS	X										•	•	•	•	•
SBC		X	X	X		X	X	X			•	•	Λ	Λ	Λ
SEC	X										•	•	•	•	1
SEI	X										•	1	•	•	•
STA			X	X		X	X	X			•	•	Λ	Λ	•
STX			X	X		X	X	X			•	•	Λ	Λ	•
STOP	X										•	1	•	•	•
SUB		X	X	X		X	X	X			•	•	Λ	Λ	Λ
SWI	X										•	1	•	•	•
TAX	X										•	•	•	•	•
TST	X		X			X	X				•	•	Λ	Λ	•
TXA	X										•	•	•	•	•
WAIT	X										•	1	•	•	•

Condition Code Symbols

H Half Carry (From Bit 3)
I Interrupt Mask
N Negative (Sign Bit)
Z Zero
C Carry/Borrow
Λ Test and Set if True, Cleared Otherwise
• Not Affected
? Load CC Register From Stack
1 Set
0 Clear

APPENDIX E
INSTRUCTION SET
FUNCTIONAL LISTING

This instruction set contains a list of functions which are categorized as to the type of instruction. It provides five different categories of instructions and provides the following information for each function: (1) corresponding mnemonic, (2) addressing mode, (3) op code, (4) number of bytes, and (5) number of cycles.

Branch Instructions

Function	Mnemonic	Relative Addressing Mode		
		Op Code	# Bytes	HMOS/CMOS # Of Cycles
Branch Always	BRA	20	2	4/3
Branch Never	BRN	21	2	4/3
Branch IFF Higher	BHI	22	2	4/3
Branch IFF Lower or Same	BLS	23	2	4/3
Branch IFF Carry Clear	BCC	24	2	4/3
(Branch IFF Higher or Same)	(BHS)	24	2	4/3
Branch IFF Carry Set	BCS	25	2	4/3
(Branch IFF Lower)	(BLO)	25	2	4/3
Branch IFF Not Equal	BNE	26	2	4/3
Branch IFF Equal	BEQ	27	2	4/3
Branch IFF Half Carry Clear	BHCC	28	2	4/3
Branch IFF Half Carry Set	BHCS	29	2	4/3
Branch IFF Plus	BPL	2A	2	4/3
Branch IFF Minus	BMI	2B	2	4/3
Branch IFF Interrupt Mask Bit is Clear	BMC	2C	2	4/3
Branch IFF Interrupt Mask Bit is Set	BMS	2D	2	4/3
Branch IFF Interrupt Line is Low	BIL	2E	2	4/3
Branch IFF Interrupt Line is High	BIH	2F	2	4/3
Branch to Subroutine	BSR	AD	2	8/6

Bit Manipulation Instructions

Function	Mnemonic	Addressing Modes					
		Bit Set/Clear			Bit Test and Branch		
		Op Code	# Bytes	HMOS/CMOS # of Cycles	Op Code	# Bytes	HMOS/CMOS # of Cycles
Branch IFF Bit n is set	BRSET n (n = 0....7)	–	–	–	2 • n	3	10/5
Branch IFF Bit n is clear	BRCLR n (n = 0....7)	–	–	–	01 + 2 • n	3	10/5
Set Bit n	BSET n (n = 0....7)	10 + 2 • n	2	7/5	–	–	–
Clear bit n	BCLR n (n = 0....7)	11 + 2 • n	2	7/5	–	–	–

Control Instructions

Function	Mnemonic	Inherent		
		Op Code	# Bytes	HMOS/CMOS # of Cycles
Transfer A to X	TAX	97	1	2/2
Transfer X to A	TXA	9F	1	2/2
Set Carry Bit	SEC	99	1	2/2
Clear Carry Bit	CLC	98	1	2/2
Set Interrupt Mask Bit	SEI	9B	1	2/2
Clear Interrupt Mask Bit	CLI	9A	1	2/2
Software Interrupt	SWI	83	1	11/10
Return from Subroutine	RTS	81	1	6/6
Return from Interrupt	RTI	80	1	9/9
Reset Stack Pointer	RSP	9C	1	2/2
No-Operation	NOP	9D	1	2/2
Enable IRQ, Stop Oscillator	STOP	8E	1	-/2
Enable Interrupt, Stop Processor	WAIT	8F	1	-/2

Read/Modify/Write Instructions

Function	Mnem.	Addressing Modes														
		Inherent (A)			Inherent (X)			Direct			Indexed (No Offset)			Indexed (8-Bit Offset)		
		Op Code	# Bytes	Cycles (see note)	Op Code	# Bytes	Cycles (see note)	Op Code	# Bytes	Cycles (see note)	Op Code	# Bytes	Cycles (see note)	Op Code	# Bytes	Cycles (see note)
Increment	INC	4C	1	4/3	5C	1	4/3	3C	2	6/5	7C	1	6/5	6C	2	7/6
Decrement	DEC	4A	1	4/3	5A	1	4/3	3A	2	6/5	7A	1	6/5	6A	2	7/6
Clear	CLR	4F	1	4/3	5F	1	4/3	3F	2	6/5	7F	1	6/5	6F	2	7/6
Complement	COM	43	1	4/3	53	1	4/3	33	2	6/5	73	1	6/5	63	2	7/6
Negate (2's complement)	NEG	40	1	4/3	50	1	4/3	30	2	6/5	70	1	6/5	60	2	7/6
Rotate Left Thru Carry	ROL	49	1	4/3	59	1	4/3	39	2	6/5	79	1	6/5	69	2	7/6
Rotate Right Thru Carry	ROR	46	1	4/3	56	1	4/3	36	2	6/5	76	1	6/5	66	2	7/6
Logical Shift Left	LSL	48	1	4/3	58	1	4/3	38	2	6/5	78	1	6/5	68	2	7/6
Logical Shift Right	LSR	44	1	4/3	54	1	4/3	34	2	6/5	74	1	6/5	64	2	7/6
Arithmetic Shift Right	ASR	47	1	4/3	57	1	4/3	37	2	6/5	77	1	6/5	67	2	7/6
Test for Negative or Zero	TST	4D	1	4/3	5D	1	4/3	3D	2	6/4	7D	1	6/4	6D	2	7/5

NOTE: The cycles column actually shows the number of HMOS/CMOS cycles (e.g., 4/3 indicates 4 HMOS cycles or 3 CMOS cycles).

Register/Memory Instructions

Function	Mnem.	Addressing Modes																	
		Immediate			Direct			Extended			Indexed (No Offset)			Indexed (8-Bit Offset)			Indexed (16-Bit Offset)		
		Op Code	# Bytes	Cycles (see note)	Op Code	# Bytes	Cycles (see note)	Op Code	# Bytes	Cycles (see note)	Op Code	# Bytes	Cycles (see note)	Op Code	# Bytes	Cycles (see note)	Op Code	# Bytes	Cycles (see note)
Load A from Memory	LDA	A6	2	2/2	B6	2	4/3	C6	3	5/4	F6	1	4/3	E6	2	5/4	D6	3	6/5
Load X from Memory	LDX	AE	2	2/2	BE	2	4/3	CE	3	5/4	FE	1	4/3	EE	2	5/4	DE	3	6/5
Store A in Memory	STA	–	–	–	B7	2	5/4	C7	3	6/5	F7	1	5/4	E7	2	6/5	D7	3	7/6
Store X in Memory	STX	–	–	–	BF	2	5/4	CF	3	6/5	FF	1	5/4	EF	2	6/5	DF	3	7/6
Add Memory to A	ADD	AB	2	2/2	BB	2	4/3	CB	3	5/4	FB	1	4/3	EB	2	5/4	DB	3	6/5
Add Memory and Carry to A	ADC	A9	2	2/2	B9	2	4/3	C9	3	5/4	F9	1	4/3	E9	2	5/4	D9	3	6/5
Subtract Memory	SUB	A0	2	2/2	B0	2	4/3	C0	3	5/4	F0	1	4/3	E0	2	5/4	D0	3	6/5
Subtract Memory from A with Borrow	SBC	A2	2	2/2	B2	2	4/3	C2	3	5/4	F2	1	4/3	E2	2	5/4	D2	3	6/5
AND Memory to A	AND	A4	2	2/2	B4	2	4/3	C4	3	5/4	F4	1	4/3	E4	2	5/4	D4	3	6/5
OR Memory with A	ORA	AA	2	2/2	BA	2	4/3	CA	3	5/4	FA	1	4/3	EA	2	5/4	DA	3	6/5
Exclusive OR Memory with A	EOR	A8	2	2/2	B8	2	4/3	C8	3	5/4	F8	1	4/3	E8	2	5/4	D8	3	6/5
Arithmetic Compare A with Memory	CMP	A1	2	2/2	B1	2	4/3	C1	3	5/4	F1	1	4/3	E1	2	5/4	D1	3	6/5
Arithmetic Compare X with Memory	CPX	A3	2	2/2	B3	2	4/3	C3	3	5/4	F3	1	4/3	E3	2	5/4	D3	3	6/5
Bit Test Memory with A (Logical Compare)	BIT	A5	2	2/2	B5	2	4/3	C5	3	5/4	F5	1	4/3	E5	2	5/4	D5	3	6/5
Jump Unconditional	JMP	–	–	–	BC	2	3/2	CC	3	4/3	FC	1	3/2	EC	2	4/3	DC	3	5/4
Jump to Subroutine	JSR	–	–	–	BD	2	7/5	CD	3	8/6	FD	1	7/5	ED	2	8/6	DD	3	9/7

NOTE: The cycles column actually shows the number of HMOS/CMOS cycles (e.g., 4/3 indicates 4 HMOS cycles or 3 CMOS cycles).

APPENDIX F
INSTRUCTION SET
NUMERICAL LISTING

This appendix provides a numerical listing of the operation codes used with the M6805 HMOS/M146805 CMOS Family. In addition, the corresponding mnemonic, mode, number of MCU/MPU cycles required to complete the instruction, and the number of bytes contained in the instruction are also included. Symbols and abbreviations used in the appendix are listed below.

Miscellaneous Symbols

OP	Operations Code (Hexadecimal)
~	Number of MPU Cycles
#	Number of Program Bytes
MNEM	Mnemonic Abbreviation

Abbreviations for Address Modes

INH	Inherent
A	Accumulator
X	Index Register
IMM	Immediate
DIR	Direct
REL	Relative
BSC	Bit Set/Clear
BTB	Bit Test and Branch
IX	Indexed (No Offset)
IX1	Indexed, 1-Byte (8-Bit) Offset
IX2	Indexed, 2-Byte (16-Bit) Offset
EXT	Extended

INSTRUCTION SET NUMERICAL LISTING

OP	MNEM	MODE	~ HMOS	~ CMOS	#
00	BRSET0	BTB	10	5	3
01	BRCLR0	BTB	10	5	3
02	BRSET1	BTB	10	5	3
03	BRCLR1	BTB	10	5	3
04	BRSET2	BTB	10	5	3
05	BRCLR2	BTB	10	5	3
06	BRSET3	BTB	10	5	3
07	BRCLR3	BTB	10	5	3
08	BRSET4	BTB	10	5	3
09	BRCLR4	BTB	10	5	3
0A	BRSET5	BTB	10	5	3
0B	BRCLR5	BTB	10	5	3
0C	BRSET6	BTB	10	5	3
0D	BRCLR6	BTB	10	5	3
0E	BRSET7	BTB	10	5	3
0F	BRCLR7	BTB	10	5	3
10	BSET0	BSC	7	5	2
11	BCLR0	BSC	7	5	2
12	BSET1	BSC	7	5	2
13	BCLR1	BSC	7	5	2
14	BSET2	BSC	7	5	2
15	BCLR2	BSC	7	5	2
16	BSET3	BSC	7	5	2
17	BCLR3	BSC	7	5	2
18	BSET4	BSC	7	5	2
19	BCLR4	BSC	7	5	2
1A	BSET5	BSC	7	5	2
1B	BCLR5	BSC	7	5	2
1C	BSET6	BSC	7	5	2
1D	BCLR6	BSC	7	5	2
1E	BSET7	BSC	7	5	2
1F	BCLR7	BSC	7	5	2
20	BRA	REL	4	3	2
21	BRN	REL	4	3	2
22	BHI	REL	4	3	2
23	BLS	REL	4	3	2
24	BCC	REL	4	3	2
25	BCS	REL	4	3	2
26	BNE	REL	4	3	2
27	BEQ	REL	4	3	2
28	BHCC	REL	4	3	2
29	BHCS	REL	4	3	2
2A	BPL	REL	4	3	2
2B	BMI	REL	4	3	2
2C	BMC	REL	4	3	2

INSTRUCTION SET NUMERICAL LISTING (CONTINUED)

OP	MNEM	MODE	~ HMOS	~ CMOS	#
2D	BMS	REL	4	3	2
2E	BIL	REL	4	3	2
2F	BIH	REL	4	3	2
30	NEG	DIR	6	5	2
33	COM	DIR	6	5	2
34	LSR	DIR	6	5	2
36	ROR	DIR	6	5	2
37	ASR	DIR	6	5	2
38	LSL	DIR	6	5	2
39	ROL	DIR	6	5	2
3A	DEC	DIR	6	5	2
3C	INC	DIR	6	5	2
3D	TST	DIR	6	4	2
3F	CLR	DIR	6	5	2
40	NEGA	INH	4	3	1
43	COMA	INH	4	3	1
44	LSRA	INH	4	3	1
46	RORA	INH	4	3	1
47	ASRA	INH	4	3	1
48	LSLA	INH	4	3	1
49	ROLA	INH	4	3	1
4A	DECA	INH	4	3	1
4C	INCA	INH	4	3	1
4D	TSTA	INH	4	3	1
4F	CLRA	INH	4	3	1
50	NEGX	INH	4	3	1
53	COMX	INH	4	3	1
54	LSRX	INH	4	3	1
56	RORX	INH	4	3	1
57	ASRX	INH	4	3	1
58	LSLX	INH	4	3	1
59	ROLX	INH	4	3	1
5A	DECX	INH	4	3	1
5C	INCX	INH	4	3	1
5D	TSTX	INH	4	3	1
5F	CLRX	INH	4	3	1
60	NEG	IX1	7	6	2
63	COM	IX1	7	6	2
64	LSR	IX1	7	6	2
66	ROR	IX1	7	6	2
67	ASR	IX1	7	6	2
68	LSL	IX1	7	6	2
69	ROL	IX1	7	6	2
6A	DEC	IX1	7	6	2
6C	INC	IX1	7	6	2

INSTRUCTION SET NUMERICAL LISTING (CONTINUED)

OP	MNEM	MODE	~ HMOS	~ CMOS	#
6D	TST	IX1	7	5	2
6F	CLR	IX1	7	6	2
70	NEG	IX	6	5	1
73	COM	IX	6	5	1
74	LSR	IX	6	5	1
76	ROR	IX	6	5	1
77	ASR	IX	6	5	1
78	LSL	IX	6	5	1
79	ROL	IX	6	5	1
7A	DEC	IX	6	5	1
7C	INC	IX	6	5	1
7D	TST	IX	6	4	1
7F	CLR	IX	6	5	1
80	RTI	INH	9	9	1
81	RTS	INH	6	6	1
83	SWI	INH	11	10	1
8E	STOP	INH	—	2	1
8F	WAIT	INH	—	2	1
97	TAX	INH	2	2	1
98	CLC	INH	2	2	1
99	SEC	INH	2	2	1
9A	CLI	INH	2	2	1
9B	SEI	INH	2	2	1
9C	RSF	INH	2	2	1
9D	NOP	INH	2	2	1
9F	TXA	INH	2	2	1
A0	SUB	IMM	2	2	2
A1	CMP	IMM	2	2	2
A2	SBC	IMM	2	2	2
A3	CPX	IMM	2	2	2
A4	AND	IMM	2	2	2
A5	BIT	IMM	2	2	2
A6	LDA	IMM	2	2	2
A8	EOR	IMM	2	2	2
A9	ADC	IMM	2	2	2
AA	ORA	IMM	2	2	2
AB	ADD	IMM	2	2	2
AD	BSR	IMM	8	6	2
AE	LDX	IMM	2	2	2
B0	SUB	DIR	4	3	2
B1	CMP	DIR	4	3	2
B2	SBC	DIR	4	3	2
B3	CPX	DIR	4	3	2
B4	AND	DIR	4	3	2
B5	BIT	DIR	4	3	2
B6	LDA	DIR	4	3	2

INSTRUCTION SET NUMERICAL LISTING (CONTINUED)

OP	MNEM	MODE	~ HMOS	~ CMOS	#
B7	STA	DIR	4	4	2
B8	EOR	DIR	4	3	2
B9	ADC	DIR	4	3	2
BA	ORA	DIR	4	3	2
BB	ADD	DIR	4	3	2
BC	JMP	DIR	3	3	2
BD	JSR	DIR	7	5	2
BE	LDX	DIR	4	3	2
BF	STX	DIR	5	4	2
C0	SUB	EXT	5	4	3
C1	CMP	EXT	5	4	3
C2	SBC	EXT	5	4	3
C3	CPX	EXT	5	4	3
C4	AND	EXT	5	4	3
C5	BIT	EXT	5	4	3
C6	LDA	EXT	5	4	3
C7	STA	EXT	6	5	3
C8	EOR	EXT	5	4	3
C9	ADC	EXT	5	4	3
CA	ORA	EXT	5	4	3
CB	ADD	EXT	5	4	3
CC	JMP	EXT	4	4	3
CD	JSR	EXT	8	6	3
CE	LDX	EXT	4	4	3
CF	STX	EXT	5	5	3
D0	SUB	IX2	6	5	3
D1	CMP	IX2	6	5	3
D2	SBC	IX2	6	5	3
D3	CPX	IX2	6	5	3
D4	AND	IX2	6	5	3
D5	BIT	IX2	6	5	3
D6	LDA	IX2	6	5	3
D7	STA	IX2	7	6	3
D8	EOR	IX2	6	5	3
D9	ADC	IX2	6	5	3
DA	ORA	IX2	6	5	3
DB	ADD	IX2	6	5	3
DC	JMP	IX2	5	5	3
DD	JSR	IX2	9	7	3
DE	LDX	IX2	6	5	3
DF	STX	IX2	7	6	3
E0	SUB	IX1	5	4	2
E1	CMP	IX1	5	4	2
E2	SBC	IX1	5	4	2
E3	CPX	IX1	5	4	2

INSTRUCTION SET NUMERICAL LISTING (CONCLUDED)

OP	MNEM	MODE	~ HMOS	~ CMOS	#
E4	AND	IX1	5	4	2
E5	BIT	IX1	5	4	2
E6	LDA	IX1	5	4	2
E7	STA	IX1	6	5	2
E8	EOR	IX1	5	4	2
E9	ADC	IX1	5	4	2
EA	ORA	IX1	5	4	2
EB	ADD	IX1	5	4	2
EC	JMP	IX1	4	4	2
ED	JSR	IX1	8	6	2
EE	LDX	IX1	5	4	2
EF	STX	IX1	6	5	2
F0	SUB	IX	4	3	1
F1	CMP	IX	4	3	1
F2	SBC	IX	4	3	1
F3	CPX	IX	4	3	1
F4	AND	IX	4	3	1
F5	BIT	IX	4	3	1
F6	LDA	IX	4	3	1
F7	STA	IX	5	4	1
F8	EOR	IX	4	3	1
F9	ADC	IX	4	3	1
FA	ORA	IX	4	3	1
FB	ADD	IX	4	3	1
FC	JMP	IX	3	3	1
FD	JSR	IX	7	5	1
FE	LDX	IX	4	3	1
FF	STX	IX	5	4	1

APPENDIX G
INSTRUCTION SET CYCLE-BY-CYCLE OPERATION SUMMARY

This appendix provides a detailed description of the cycle-by-cycle operation for each instruction. The information is contained in two tables: one for the M146805 CMOS Family and the other for the M6805 HMOS Family. Each table contains information which includes the total number of cycles required to execute the instruction, plus a step-by-step breakdown of each cycle. All of the M6805 HMOS Family and, except for the MC146805E2 Microprocessor Unit (MPU), all of the M146805 CMOS Family are Microcomputer Units (MCUs). This means that only the MC146805E2 has an external address bus, R/$\overline{W}$ pin, and data bus. In all others, these are internal to the MCU and are not connected to any external pin(s).

The information contained in these two tables is useful in comparing actual with expected results, while debugging both software and hardware, during control program execution. The information is categorized in groups according to the addressing mode and number of cycles per instructions.

Table G1. M146805 CMOS Family Summary of Cycle-by-Cycle Operation

Instructions	Cycles	Cycle #	Address Bus*	R/$\overline{W}$	Data Bus*
INHERENT					
ASL ASR CLR COM		1	Opcode Address	1	Opcode
DEC INC LSL LSR	3	2	Opcode Address + 1	1	Opcode Next Instruction
NEG ROL ROR TST		3	Opcode Address + 1	1	Opcode Next Instruction
CLC CLI NOP RSP SEC	2	1	Opcode Address	1	Opcode
SEI STOP TAX TXA WAIT		2	Opcode Address + 1	1	Opcode Next Instruction
		1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Opcode Next Instruction
RTS	6	3	Stack Pointer	1	Return Address (HI Byte)***
		4	Stack Pointer + 1	1	Return Address (LO Byte)***
		5	Stack Pointer + 2	1	Irrelevant Data
		6	New Opcode Address	1	New Opcode
		1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Opcode Next Instruction
		3	Stack Pointer	0	Return Address (LO Byte)
		4	Stack Pointer − 1	0	Return Address (HI Byte)
		5	Stack Pointer − 2	0	Contents of Index Register
SWI	10	6	Stack Pointer − 3	0	Contents of Accumulator
		7	Stack Pointer − 4	0	Contents of CC Register
		8	Vector Address $1FFC**	1	Address of Interrupt Routine (HI Byte)
		9	Vector Address $1FFD**	1	Address of Interrupt Routine (LO Byte)
		10	Interrupt Routine Starting Address	1	Interrupt Routine First Opcode
		1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Opcode Next Instruction
		3	Stack Pointer	1	Irrelevant Data
		4	Stack Pointer + 1	1	Contents of CC Register***
RTI	9	5	Stack Pointer + 2	1	Contents of Accumulator***
		6	Stack Pointer + 3	1	Contents of Index Register***
		7	Stack Pointer + 4	1	Return Address (HI Byte)***
		8	Stack Pointer + 5	1	Return Address (LO Byte)***
		9	New Opcode Address	1	New Opcode
IMMEDIATE					
ADC ADD AND BIT					
CMP CPX EOR LDA	2	1	Opcode Address	1	Opcode
LDX ORA SBC SUB		2	Opcode Address + 1	1	Operand Data
BIT SET/CLEAR					
		1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Address of Operand
BSET n	5	3	Address of Operand	1	Operand Data
BCLR n		4	Address of Operand	1	Operand Data
		5	Address of Operand	0	Manipulated Data
BIT TEST AND BRANCH					
		1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Address of Operand
BRSET n	5	3	Address of Operand	1	Operand Data
BRCLR n		4	Opcode Address + 2	1	Branch Offset
		5	Opcode Address + 2	1	Branch Offset

Table G1. M146805 CMOS Family Summary of Cycle-by-Cycle Operation (Continued)

Instructions	Cycles	Cycle #	Address Bus*	R/$\overline{W}$	Data Bus
RELATIVE					
BCC (BHS) BCS (BLO) BEQ BHCC BHCS BHI BIH BIL BLS BMC BMI BMS BNE BPL BRA BRN	3	1 2 3	Opcode Address Opcode Address + 1 Opcode Address + 1	1 1 1	Opcode Branch Offset Branch Offset
BSR	6	1 2 3 4 5 6	Opcode Address Opcode Address + 1 Opcode Address + 1 Subroutine Starting Address Stack Pointer Stack Pointer – 1	1 1 1 1 0 0	Opcode Branch Offset Branch Offset 1st Subroutine Opcode Return Address (LO Byte) Return Address (HI Byte)
DIRECT					
JMP	2	1 2	Opcode Address Opcode Address + 1	1 1	Opcode Jump Address
ADC ADD AND BIT CMP CPX EOR LDA LDX ORA SBC SUB	3	1 2 3	Opcode Address Opcode Address + 1 Address of Operand	1 1 1	Opcode Address of Operand Operand Data
TST	4	1 2 3 4	Opcode Address Opcode Address + 1 Address of Operand Opcode Address + 2	1 1 1 1	Opcode Address of Operand Operand Data Opcode Next Instruction
STA STX	4	1 2 3 4	Opcode Address Opcode Address + 1 Opcode Address + 1 Address of Operand	1 1 1 0	Opcode Address of Operand Address of Operand Operand Data
ASL ASR CLR COM DEC INC LSL LSR NEG ROL ROR	5	1 2 3 4 5	Opcode Address Opcode Address + 1 Address of Operand Address of Operand Address of Operand	1 1 1 1 0	Opcode Address of Operand Current Operand Data Current Operand Data New Operand Data
JSR	5	1 2 3 4 5	Opcode Address Opcode Address + 1 Subroutine Starting Address Stack Pointer Stack Pointer – 1	1 1 1 0 0	Opcode Subroutine Address (LO Byte) 1st Subroutine Opcode Return Address (LO Byte) Return Address (HI Byte)
EXTENDED					
JMP	3	1 2 3	Opcode Address Opcode Address + 1 Opcode Address + 2	1 1 1	Opcode Jump Address (HI Byte) Jump Address (LO Byte)**
ADC ADD AND BIT CMP CPX EOR LDA LDX ORA SBC SUB	4	1 2 3 4	Opcode Address Opcode Address + 1 Opcode Address + 2 Address of Operand	1 1 1 1	Opcode Address of Operand (HI Byte) Address of Operand (LO Byte) Operand Data
STA STX	5	1 2 3 4 5	Opcode Address Opcode Address + 1 Opcode Address + 2 Opcode Address + 2 Address of Operand	1 1 1 1 0	Opcode Address of Operand (HI Byte) Address of Operand (LO Byte) Address of Operand (LO Byte) Operand Data
JSR	6	1 2 3 4 5 6	Opcode Address Opcode Address + 1 Opcode Address + 2 Subroutine Starting Address Stack Pointer Stack Pointer – 1	1 1 1 1 0 0	Opcode Addr of Subroutine (HI Byte) Addr of Subroutine (LO Byte) 1st Subroutine Opcode Return Address (LO Byte) Return Address (HI Byte)**

Table G1. M146805 CMOS Family Summary of Cycle-by-Cycle Operation (Continued)

Instructions	Cycles	Cycle #	Address Bus*	R/$\overline{W}$	Data Bus
INDEXED, NO-OFFSET					
JMP	2	1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Opcode Next Instruction
ADC ADD AND BIT CMP CPX EOR LDA LDX ORA SBC SUB	3	1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Opcode Next Instruction
		3	Index Register	1	Operand Data
TST	4	1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Opcode Next Instruction
		3	Index Register	1	Operand Data
		4	Opcode Address + 1	1	Opcode Next Instruction
STA STX	4	1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Opcode Next Instruction
		3	Opcode Address + 1	1	Opcode Next Instruction
		4	Index Register	0	Operand Data
ASL ASR CLR COM DEC INC LSL LSR NEG ROL ROR	5	1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Opcode Next Instruction
		3	Index Register	1	Current Operand Data
		4	Index Register	1	Current Operand Data
		5	Index Register	0	New Operand Data
JSR	5	1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Opcode Next Instruction
		3	Index Register	1	1st Subroutine Opcode
		4	Stack Pointer	0	Return Address (LO Byte)
		5	Stack Pointer − 1	0	Return Address (HI Byte)
INDEXED, 8-BIT OFFSET					
JMP	3	1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Offset
		3	Opcode Address + 1	1	Offset
ADC ADD AND BIT CMP CPX EOR LDA LDX ORA SBC SUB	4	1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Offset
		3	Opcode Address + 1	1	Offset
		4	Index Register + Offset	1	Operand Data
STA STX	5	1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Offset
		3	Opcode Address + 1	1	Offset
		4	Opcode Address + 1	1	Offset
		5	Index Register + Offset	0	Operand Data
TST	5	1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Offset
		3	Opcode Address + 1	1	Offset
		4	Index Register + Offset	1	Operand Data
		5	Opcode Address + 2	1	Opcode Next Instruction
ASL ASR CLR COM DEC INC LSL LSR NEG ROL ROR	6	1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Offset
		3	Opcode Address + 1	1	Offset
		4	Index Register + Offset	1	Current Operand Data
		5	Index Register + Offset	1	Current Operand Data
		6	Index Register + Offset	0	New Operand Data
JSR	6	1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Offset
		3	Opcode Address + 1	1	Offset
		4	Index Register + Offset	1	1st Subroutine Opcode
		5	Stack Pointer	0	Return Address (LO Byte)
		6	Stack Pointer − 1	0	Return Address (HI Byte)**

Table G1. M146805 CMOS Family Summary of Cycle-by-Cycle Operation (Continued)

Instructions	Cycles	Cycle #	Address Bus*	R/$\overline{W}$	Data Bus
INDEXED, 16-BIT OFFSET					
JMP	4	1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Offset (HI Byte)
		3	Opcode Address + 2	1	Offset (LO Byte)
		4	Opcode Address + 2	1	Offset (LO Byte)
ADC ADD AND BIT CMP CPX EOR LDA LDX ORA SBC SUB	5	1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Offset (HI Byte)
		3	Opcode Address + 2	1	Offset (LO Byte)
		4	Opcode Address + 2	1	Offset (LO Byte)
		5	Index Register + Offset	1	Operand Data
STA STX	6	1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Offset (HI Byte)
		3	Opcode Address + 2	1	Offset (LO Byte)
		4	Opcode Address + 2	1	Offset (LO Byte)
		5	Opcode Address + 2	1	Offset (LO Byte)
		6	Index Register + Offset	0	New Operand Data
JSR	7	1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Offset (HI Byte)
		3	Opcode Address + 2	1	Offset (LO Byte)
		4	Opcode Address + 2	1	Offset (LO Byte)
		5	Index Register + Offset	1	1st Subroutine Opcode
		6	Stack Pointer	0	Return Address (LO Byte)
		7	Stack Pointer – 1	0	Return Address (HI Byte)**

Table G1. M146805 CMOS Family Summary of Cycle-by-Cycle Operation (Concluded)

RESET AND INTERRUPT						
Instructions	**Cycles**	**Cycle #**	**Address Bus***	**Reset**	**R/$\overline{W}$**	**Data Bus***
Hardware Reset	5		$1FFE**	0	1	Irrelevant Data
			$1FFE**	0	1	Irrelevant Data
		1	$1FFE**	1	1	Irrelevant Data
		2	$1FFE**	1	1	Irrelevant Data
		3	$1FFE**	1	1	Vector (HI Byte)
		4	$1FFF**	1	1	Vector (LO Byte)
		5	Reset Vector	1	1	Opcode
Power on Reset	1922	1	$1FFE	1	1	Irrelevant Data
		•	•	•	•	•
		•	•	•	•	•
		•	•	•	•	•
		1919	$1FFE**	1	1	Irrelevant Data
		1920	$1FFE**	1	1	Vector (HI Byte)
		1921	$1FFF**	1	1	Vector (LO Byte)
		1922	Reset Vector	1	1	Opcode

HARDWARE INTERRUPTS						
Instructions	**Cycles**	**Cycle #**	**Address Bus***	**$\overline{IRQ}$**	**R/$\overline{W}$**	**Data Bus***
$\overline{IRQ}$ Interrupt (Vector HI: $1FFA,** Vector LO: $1FFB**) Timer Interrupt (Vector HI: $1FF9**, Vector LO: $1FF8**)	10		Last Cycle of Previous Instruction	0	X	X
		1	Next Opcode Address	0	1	Irrelevant Data
		2	Next Opcode Address	X	1	Irrelevant Data
		3	Stack Pointer	X	0	Return Addr. (LO Byte)
		4	Stack Pointer − 1	X	0	Return Addr. (HI Byte)
		5	Stack Pointer − 2	X	0	Contents Index Reg
		6	Stack Pointer − 3	X	0	Contents Accumulator
		7	Stack Pointer − 4	X	0	Contents CC Register
		8	$1FFA**	X	1	Vector (HI Byte)
		9	$1FFB**	X	1	Vector (LO Byte)
		10	$\overline{IRQ}$ Vector	X	1	Interrupt Routine First

*Except for the MC146805E2 MPU, the address bus, R/$\overline{W}$, and data bus are internal to the device.

**All values given are for devices with 13-bit program counters (e.g., MC146805E2 and MC146805G2). For devices with 11-bit program counters (MC146805F2), the HI byte is "07" instead of "1F"

X Indicates don't care.

***On the MC146805E2 the data bus is external and, since the stack is on-chip, data on the external bus is ignored during the RTI and RTS instructions.

Table G2. M6805 HMOS Family Summary of Cycle-by-Cycle Operation

Instructions	Cycles	Cycle #	Address Bus	R/$\overline{W}$	Data Bus
INHERENT					
ASL ASR CLR COM DEC INC LSL LSR NEG ROL ROR TST	4	1 2 3 4	Opcode Address Opcode Address + 1 Opcode Address + 2 Opcode Address + 2	1 1 1 1	Opcode Opcode Next Instruction Byte Following Next Opcode Byte Following Next Opcode
CLC CLI NOP RSP SEC SEI TAX TXA	2	1 2	Opcode Address Opcode Address + 1	1 1	Opcode Opcode Next Instruction
RTS	6	1 2 3 4 5 6	Opcode Address Opcode Address + 1 Stack Pointer Stack Pointer + 1 Stack Pointer + 2 Stack Pointer + 3	1 1 1 1 1 1	Opcode Opcode Next Instruction Irrelevant Data Return Address (HI Byte) Return Address (LO Byte) Irrelevant Data
SWI	11	1 2 3 4 5 6 7 8 9 10 11	Opcode Address Opcode Address + 1 Stack Pointer Stack Pointer − 1 Stack Pointer − 2 Stack Pointer − 3 Stack Pointer − 4 Stack Pointer − 5 Vector Address $7FC* Vector Address $7FD* Interrupt Routine Starting Address	1 1 0 0 0 0 0 1 1 1 1	Opcode Opcode Next Instruction Return Address (LO Byte) Return Address (HI Byte) Contents of Index Register Contents of Accumulator Contents of CC Register Irrelevant Data Addr. of Int. Routine (HI Byte) Addr. of Int. Routine (LO Byte) Interrupt Routine First Opcode
RTI	9	1 2 3 4 5 6 7 8 9	Opcode Address Opcode Address + 1 Stack Pointer Stack Pointer + 1 Stack Pointer + 2 Stack Pointer + 3 Stack Pointer + 4 Stack Pointer + 5 Stack Pointer + 6	1 1 1 1 1 1 1 1 1	Opcode Opcode Next Instruction Irrelevant Data Contents of CC Register Contents of Accumulator Contents of Index Register Return Address (HI Byte) Return Address (LO Byte) Irrelevant Data
IMMEDIATE					
ADC ADD AND BIT CMP CPX EOR LDA LDX ORA SBC SUB	2	1 2	Opcode Address Opcode Address + 1	1 1	Opcode Operand Data
BIT SET/CLEAR					
BSET n BCLR n	7	1 2 3 4 5 6 7	Opcode Address Opcode Address + 1 $07F Address of Operand Address of Operand Address of Operand Address of Operand	1 1 1 1 1 1 0	Opcode Address of Operand Data at $07F (Unused) Operand Data Operand Data Operand Data Manipulated Data
BIT TEST AND BRANCH					
BRSET n BRCLR n	10	1 2 3 4 5 6 7 8 9 10	Opcode Address Opcode Address + 1 $07F Address of Operand Address of Operand Address of Operand Address of Operand Opcode Address + 2 Opcode Address + 3 Opcode Address + 3	1 1 1 1 1 1 1 1 1 1	Opcode Address of Operand Data at $07F (Unused) Operand Data Operand Data Operand Data Operand Data Branch Offset Opcode Next Instruction Opcode Next Instruction

Table G2. M6805 HMOS Family Summary of Cycle-by-Cycle Operation (Continued)

Instructions	Cycles	Cycle #	Address Bus	R/$\overline{W}$	Data Bus
RELATIVE					
BCC (BHS) BCS (BLO) BEQ BHCC BHCS BHI BIH BIL BLS BMC BMI BMS BNE BPL BRA BRN	4	1 2 3 4	Opcode Address Opcode Address + 1 Opcode Address + 2 Opcode Address + 2	1 1 1 1	Opcode Branch Offset Opcode Next Instruction Opcode Next Instruction
BSR	8	1 2 3 4 5 6 7 8	Opcode Address Opcode Address + 1 Opcode Address + 2 Opcode Address + 2 Subroutine Starting Address Stack Pointer Stack Pointer – 1 Stack Pointer – 2	1 1 1 1 1 0 0 1	Opcode Branch Offset Opcode Next Instruction Opcode Next Instruction 1st Subroutine Opcode Return Address (LO Byte) Return Address (HI Byte) Irrelevant Data
DIRECT					
JMP	3	1 2 3	Opcode Address Opcode Address + 1 $07F	1 1 1	Opcode Jump Address Data at $07F (Unused)
ADC ADD AND BIT CMP CPX EOR LDA LDX ORA SBC SUB	4	1 2 3 4	Opcode Address Opcode Address + 1 $07F Address of Operand	1 1 1 1	Opcode Address of Operand Data at $07F (Unused) Operand Data
TST	6	1 2 3 4 5 6	Opcode Address Opcode Address + 1 $07F Address of Operand Address of Operand Address of Operand	1 1 1 1 1 1	Opcode Address of Operand Data at $07F (Unused) Operand Data Operand Data Operand Data
STA STX	5	1 2 3 4 5	Opcode Address Opcode Address + 1 $07F Address of Operand Address of Operand	1 1 1 1 0	Opcode Address of Operand Data at $07F (Unused) Current Operand Data New Operand Data
ASL ASR CLR COM DEC INC LSL LSR NEG ROL ROR	6	1 2 3 4 5 6	Opcode Address Opcode Address + 1 $07F Address of Operand Address of Operand Address of Operand	1 1 1 1 1 0	Opcode Address of Operand Data at $07F (Unused) Current Operand Data Current Operand Data New Operand Data
JSR	7	1 2 3 4 5 6 7	Opcode Address Opcode Address + 1 $07F Subroutine Starting Address Stack Pointer Stack Pointer – 1 Stack Pointer – 2	1 1 1 1 0 0 1	Opcode Subroutine Address (LO Byte) Data at $07F (Unused) 1st Subroutine Opcode Return Address (LO Byte) Return Address (Hi Byte)** Irrelevant Data

Table G2. M6805 HMOS Family Summary of Cycle-by-Cycle Operation (Continued)

Instructions	Cycles	Cycle #	Address Bus*	R/$\overline{W}$	Data Bus
EXTENDED					
JMP	4	1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Jump Address (HI Byte)
		3	Opcode Address + 2	1	Jump Address (LO Byte)
		4	$XFF	1	Data at $XFF (Unused)
ADC BIT ORA ADD CMP LDX AND EOR SBC CPX LDA SUB	5	1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Address of Operand (HI Byte)
		3	Opcode Address + 2	1	Address of Operand (LO Byte)
		4	$XFF	1	Data at $XFF
		5	Address of Operand	1	Operand Data
STA STX	6	1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Address of Operand (HI Byte)
		3	Opcode Address + 2	1	Address of Operand (LO Byte)
		4	XFF	1	Data at $XFF (Unused)
		5	Address of Operand	1	Current Operand Data
		6	Address of Operand	0	New Operand Data
JSR	8	1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Addr of Subroutine (HI Byte)
		3	Opcode Address + 2	1	Addr of Subroutine (LO Byte)
		4	$XFF	1	Data at $XFF (Unused)
		5	Subroutine Starting Address	1	1st Subroutine Opcode
		6	Stack Pointer	0	Return Address (LO Byte)
		7	Stack Pointer − 1	0	Return Address (HI Byte)**
		8	Stack Pointer − 2	1	Irrelevant Data
INDEXED NO OFFSET					
JMP	3	1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Opcode Next Instruction
		3	$07F	1	Data at $07F (Unused)
ADC ADD AND BIT CMP CPX EOR LDA LDX ORA SBC SUB	4	1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Opcode Next Instruction
		3	$07F	1	Data at $07F (Unused)
		4	Index Register	1	Operand Data
TST	6	1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Opcode Next Instruction
		3	$07F	1	Data at $07F (Unused)
		4	Index Register	1	Operand Data
		5	Index Register	1	Operand Data
		6	Index Register	1	Operand Data
STA STX	5	1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Opcode Next Instruction
		3	$07F	1	Data at $07F (Unused)
		4	Index Register	1	Current Operand Data
		5	Index Register	0	New Operand Data
ASL ASR CLR COM DEC INC LSL LSR NEG ROL ROR	6	1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Opcode Next Instruction
		3	$07F	1	Data at $07F (Unused)
		4	Index Register	1	Current Operand Data
		5	Index Register	1	Current Operand Data
		6	Index Register	0	New Operand Data
JSR	7	1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Opcode Next Instruction
		3	$07F	1	Data at $07F (Unused)
		4	Index Register	1	1st Subroutine Opcode
		5	Stack Pointer	0	Return Address (LO Byte)
		6	Stack Pointer − 1	0	Return Address (HI Byte)**
		7	Stack Pointer − 2	1	Irrelevant Data

Table G2. M6805 HMOS Family Summary of Cycle-by-Cycle Operation (Continued)

Instructions	Cycles	Cycle #	Address Bus	R/$\overline{W}$	Data Bus
INDEXED 8-BIT OFFSET					
JMP	4	1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Offset
		3	$07F	1	Data at $07F (Unused)
		4	$07F	1	Data at $07F (Unused)
ADC ADD AND BIT CMP CPX EOR LDA LDX ORA SBC SUB	5	1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Offset
		3	$07F	1	Data at $07F (Unused)
		4	$07F	1	Data at $07F (Unused)
		5	Index Register + Offset	1	Operand Data
STA STX	6	1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Offset
		3	$07F	1	Data at $07F (Unused)
		4	$07F	1	Data at $07F (Unused)
		5	Index Register + Offset	1	Current Operand Data
		6	Index Register + Offset	0	New Operand Data
TST	7	1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Offset
		3	$07F	1	Data at $07F (Unused)
		4	$07F	1	Data at $07F (Unused)
		5	Index Register + Offset	1	Operand Data
		6	Index Register + Offset	1	Operand Data
		7	Index Register + Offset	1	Operand Data
ASL ASR CLR COM DEC INC LSL LSR NEG ROL ROR	7	1	Opcode Address	1	Opcode
		2	Opcode Address+ 1	1	Offset
		3	$07F	1	Data at $07F (Unused)
		4	$07F	1	Current Operand Data
		5	Index Register + Offset	1	Current Operand Data
		6	Index Register + Offset	1	Current Operand Data
		7	Index Register + Offset	0	New Operand Data
JSR	8	1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Offset
		3	$07F	1	Data at $07F (Unused)
		4	$07F	1	Data at $07F (Unused)
		5	Index Register + Offset	1	1st Subroutine Opcode
		6	Stack Pointer	0	Return Address (LO Byte)
		7	Stack Pointer − 1	0	Return Address (HI Byte)**
		8	Stack Pointer − 2	1	Irrelevant Data
INDEXED 16-BIT OFFSET					
JMP	5	1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Offset (HI Byte)
		3	Opcode Address + 2	1	Offset (LO Byte)
		4	$XFF	1	Data at $XFF (Unused)
		5	$XFF	1	Data at $XFF (Unused)
ADC ADD AND BIT CMP CPX EOR LDA LDX ORA SBC SUB	6	1	Opcode Address	1	Opcode
		2	Opcode Address + 1	1	Offset (HI Byte)
		3	Opcode Address + 2	1	Offset (LO Byte)
		4	$XFF	1	Data at $XFF (Unused)
		5	$XFF	1	Data at $XFF (Unused)
		6	Index Register + Offset	1	Operand Data

Table G2. M6805 HMOS Family Summary of Cycle-by-Cycle Operation (Concluded)

RESET FUNCTION						
Instructions	**Cycles**	**Cycle #**	**Address Bus**	**Reset**	**R/$\overline{W}$**	**Data Bus**
			Irrelevant Address	0	X	Irrelevant Data
			Irrelevant Address	0	X	Irrelevant Data
		1	$7FE*	1	1	Reset Vector (HI Byte)
Hardware Reset	8	2	$7FE*	1	1	Reset Vector (HI Byte)
Power-on Reset		3	$7FE*	1	1	Reset Vector (HI Byte)
		4	$7FE*	1	1	Reset Vector (HI Byte)
		5	$7FE*	1	1	Reset Vector (HI Byte)
		6	$7FE*	1	1	Reset Vector (HI Byte)
		7	$7FF*	1	1	Reset Vector (LO Byte)
		8	$000	1	1	Data at $000 (Unusable)

HARDWARE INTERRUPTS						
Instructions	**Cycles**	**Cycle #**	**Address Bus**	**$\overline{IRQ}$**	**R/$\overline{W}$**	**Data Bus**
			Last Cycle of Previous Instruction	0	X	Data from Last Cycle
		1	Next Opcode Address	0	1	Next Opcode
		2	Next Opcode Address	X	1	Next Opcode
		3	Stack Pointer	X	0	Return Addr. (LO Byte)
$\overline{IRQ}$ Interrupt	11	4	Stack Pointer − 1	X	0	Return Addr. (HI Byte)**
(Vector HI: $7FA*,		5	Stack Pointer − 2	X	0	Contents of Index Register
Vector LO: $7FB*)		6	Stack Pointer − 3	X	0	Contents of Accumulator
Timer Interrupt		7	Stack Pointer − 4	X	0	Contents of CC Register
(Vector HI: $7F8*,		8	Stack Pointer − 5	X	1	Data at Stack Pointer − 5 (Unused)
Vector LO: $7F9*)		9	Vector Address HI	X	1	Vector (HI Byte)
		10	Vector Address LO	X	1	Vector (LO Byte)
		11	Vector Address LO + 1	X	1	Data at Vector LO (Unusable)

NOTES:

* All values given are for devices with 11-bit program counters (e.g., MC6805P2, MC6805P4, MC68705P3, etc.). For devices with 12-bit program counters (e.g., MC6805R2, MC6805U2, MC68705R3, MC68705U3, etc.) the HI byte is "0F" instead of "07"

** For storing the HI byte of the PC on the stack, unused bits are stored as 1s; e.g., a PCH of "03" is stored as "FB" on devices with 11-bit PCs and a PCH of "03" is stored as F3 on devices with a 12-bit PC.

X Indicates don't care.

APPENDIX H
ASCII HEXADECIMAL CODE CONVERSION CHART

This appendix shows the equivalent alphanumeric characters for the equivalent ASCII hexadecimal code.

Hex	ASCII	Hex	ASCII	Hex	ASCII	Hex	ASCII	Hex	ASCII
00	nul	1C	fs	38	8	54	T	70	p
01	soh	1D	gs	39	9	55	U	71	q
02	stx	1E	rs	3A	:	56	V	72	r
03	etx	1F	us	3B	;	57	W	73	s
04	eot	20	sp	3C	<	58	X	74	t
05	enq	21	!	3D	=	59	Y	75	u
06	ack	22	"	3E	>	5A	Z	76	v
07	bel	23	#	3F	?	5B	[	77	w
08	bs	24	$	40	@	5C	\	78	x
09	ht	25	%	41	A	5D	]	79	y
0A	nl	26	&	42	B	5E	^	7A	z
0B	vt	27	'	43	C	5F	—	7B	{
0C	ff	28	(	44	D	60	'	7C	|
0D	cr	29	)	45	E	61	a	7D	}
0E	so	2A	*	46	F	62	b	7E	~
0F	si	2B	+	47	G	63	c	7F	del
10	dle	2C	,	48	H	64	d		
11	dc1	2D	—	49	I	65	e		
12	dc2	2E	.	4A	J	66	f		
13	dc3	2F	/	4B	K	67	g		
14	dc4	30	0	4C	L	68	h		
15	nak	31	1	4D	M	69	i		
16	syn	32	2	4E	N	6A	j		
17	etb	33	3	4F	O	6B	k		
18	can	34	4	50	P	6C	l		
19	em	35	5	51	Q	6D	m		
1A	sub	36	6	52	R	6E	n		
1B	esc	37	7	53	S	6F	o		

APPENDIX I
INSTRUCTION SET
OPCODE MAP

The opcode map contains a summary of opcodes used with the M6805 HMOS and M146805 CMOS Family. The map is outlined by two sets (0-F) of hexadecimal numbers: one horizontal and one vertical. The horizontal set represents the MSD and the vertical set represents the LSD. For example, a 25 opcode represents a BCS (located at the 2 and 5 coordinates) used in the relative mode. There are five different opcodes for COM, each in a different addressing mode (direct; accumulator; indexed; indexed, one-byte offset; and indexed, two-byte offset). A legend is provided, as part of the map, to show the information contained in each coordinate square. The legend represents the coordinates for opcode F0 (SUB). Included in the legend is the opcode binary equivalent, the number of execution cycles required for both the M6805 HMOS and M146805 CMOS Family, the required number of bytes, the address mode, and the mnemonic.

	Bit Manipulation		Branch	Read/Modify/Write				
	BTB	BSC	REL	DIR	A	X	IX1	IX
Hi / Low	0 0000	1 0001	2 0010	3 0011	4 0100	5 0101	6 0110	7 0111
0 0000	10 5 BRSET0 3 BTB	7 5 BSET0 2 BSC	4 3 BRA 2 REL	6 5 NEG 2 DIR	4 3 NEG 1 A	4 3 NEG 1 X	7 6 NEG 2 IX1	6 5 NEG 1 IX
1 0001	10 5 BRCLR0 3 BTB	7 5 BCLR0 2 BSC	4 3 BRN 2 REL					
2 0010	10 5 BRSET1 3 BTB	7 5 BSET1 2 BSC	4 3 BHI 2 REL					
3 0011	10 5 BRCLR1 3 BTB	7 5 BCLR1 2 BSC	4 3 BLS 2 REL	6 5 COM 2 DIR	4 3 COM 1 A	4 3 COM 1 X	7 6 COM 2 IX1	6 5 COM 1 IX
4 0100	10 5 BRSET2 3 BTB	7 5 BSET2 2 BSC	4 3 BCC 2 REL	6 5 LSR 2 DIR	4 3 LSR 1 A	4 3 LSR 1 X	7 6 LSR 2 IX1	6 5 LSR 1 IX
5 0101	10 5 BRCLR2 3 BTB	7 5 BCLR2 2 BSC	4 3 BCS 2 REL					
6 0110	10 5 BRSET3 3 BTB	7 5 BSET3 2 BSC	4 3 BNE 2 REL	6 5 ROR 2 DIR	4 3 ROR 1 A	4 3 ROR 1 X	7 6 ROR 2 IX1	6 5 ROR 1 IX
7 0111	10 5 BRCLR3 3 BTB	7 5 BCLR3 2 BSC	4 3 BEQ 2 REL	6 5 ASR 2 DIR	4 3 ASR 1 A	4 3 ASR 1 X	7 6 ASR 2 IX1	6 5 ASR 1 IX
8 1000	10 5 BRSET4 3 BTB	7 5 BSET4 2 BSC	4 3 BHCC 2 REL	6 5 LSL 2 DIR	4 3 LSL 1 A	4 3 LSL 1 X	7 6 LSL 2 IX1	6 5 LSL 1 IX
9 1001	10 5 BRCLR4 3 BTB	7 5 BCLR4 2 BSC	4 3 BHCS 2 REL	6 5 ROL 2 DIR	4 3 ROL 1 A	4 3 ROL 1 X	7 6 ROL 2 IX1	6 5 ROL 1 IX
A 1010	10 5 BRSET5 3 BTB	7 5 BSET5 2 BSC	4 3 BPL 2 REL	6 5 DEC 2 DIR	4 3 DEC 1 A	4 3 DEC 1 X	7 6 DEC 2 IX1	6 5 DEC 1 IX
B 1011	10 5 BRCLR5 3 BTB	7 5 BCLR5 2 BSC	4 3 BMI 2 REL					
C 1100	10 5 BRSET6 3 BTB	7 5 BSET6 2 BSC	4 3 BMC 2 REL	6 5 INC 2 DIR	4 3 INC 1 A	4 3 INC 1 X	7 6 INC 2 IX1	6 5 INC 1 IX
D 1101	10 5 BRCLR6 3 BTB	7 5 BCLR6 2 BSC	4 3 BMS 2 REL	6 4 TST 2 DIR	4 3 TST 1 A	4 3 TST 1 X	7 5 TST 2 IX1	6 4 TST 1 IX
E 1110	10 5 BRSET7 3 BTB	7 5 BSET7 2 BSC	4 3 BIL 2 REL					
F 1111	10 5 BRCLR7 3 BTB	7 5 BCLR7 2 BSC	4 3 BIH 2 REL	6 5 CLR 2 DIR	4 3 CLR 1 A	4 3 CLR 1 X	7 6 CLR 2 IX1	6 5 CLR 1 IX

Abbreviations for Address Modes

INH	Inherent	EXT	Extended	IX	Indexed (No Offset)
A	Accumulator	REL	Relative	IX1	Indexed, 1 Byte (8-Bit) Offset
X	Index Register	BSC	Bit Set/Clear	IX2	Indexed, 2 Byte (16-Bit) Offset
IMM	Immediate	BTB	Bit Test and Branch	*	M146805 CMOS Family Only
DIR	Direct				

Family Instruction Set Opcode Map

	Control		Register/Memory						
	INH	INH	IMM	DIR	EXT	IX2	IX1	IX	
	8 1000	9 1001	A 1010	B 1011	C 1100	D 1101	E 1110	F 1111	Hi / Low
	9 9 RTI 1 INH		2 2 SUB 2 IMM	4 3 SUB 2 DIR	5 4 SUB 3 EXT	6 5 SUB 3 IX2	5 4 SUB 2 IX1	4 3 SUB 1 IX	0 0000
	6 6 RTS 1 INH		2 2 CMP 2 IMM	4 3 CMP 2 DIR	5 4 CMP 3 EXT	6 5 CMP 3 IX2	5 4 CMP 2 IX1	4 3 CMP 1 IX	1 0001
			2 2 SBC 2 IMM	4 3 SBC 2 DIR	5 4 SBC 3 EXT	6 5 SBC 3 IX2	5 4 SBC 2 IX1	4 3 SBC 1 IX	2 0010
	11 10 SWI 1 INH		2 2 CPX 2 IMM	4 3 CPX 2 DIR	5 4 CPX 3 EXT	6 5 CPX 3 IX2	5 4 CPX 2 IX1	4 3 CPX 1 IX	3 0011
			2 2 AND 2 IMM	4 3 AND 2 DIR	5 4 AND 3 EXT	6 5 AND 3 IX2	5 4 AND 2 IX1	4 3 AND 1 IX	4 0100
			2 2 BIT 2 IMM	4 3 BIT 2 DIR	5 4 BIT 3 EXT	6 5 BIT 3 IX2	5 4 BIT 2 IX1	4 3 BIT 1 IX	5 0101
			2 2 LDA 2 IMM	4 3 LDA 2 Dir	5 4 LDA 3 EXT	6 5 LDA 3 IX2	5 4 LDA 2 IX1	4 3 LDA 1 IX	6 0110
		2 2 TAX 1 INH		5 4 STA 2 DIR	6 5 STA 3 EXT	7 6 STA 3 IX2	6 5 STA 2 IX1	5 4 STA 1 IX	7 0111
		2 2 CLC 1 INH	2 2 EOR 2 IMM	4 3 EOR 2 DIR	5 4 EOR 3 EXT	6 5 EOR 3 IX2	5 4 EOR 2 IX1	4 3 EOR 1 IX	8 1000
		2 2 SEC 1 INH	2 2 ADC 2 IMM	4 3 ADC 2 DIR	5 4 ADC 3 EXT	6 5 ADC 3 IX2	5 4 ADC 2 IX1	4 3 ADC 1 IX	9 1001
		2 2 CLI 1 INH	2 2 ORA 2 IMM	4 3 ORA 2 DIR	5 4 ORA 3 EXT	6 5 ORA 3 IX2	5 4 ORA 2 IX1	4 3 ORA 1 IX	A 1010
		2 2 SEI 1 INH	2 2 ADD 2 IMM	4 3 ADD 2 DIR	5 4 ADD 3 EXT	6 5 ADD 3 IX2	5 4 ADD 2 IX1	4 3 ADD 1 IX	B 1011
		2 2 RSP 1 INH		3 2 JMP 2 DIR	4 3 JMP 3 EXT	5 4 JMP 3 IX2	4 3 JMP 2 IX1	3 2 JMP 1 IX	C 1100
		2 2 NOP 1 INH	8 6 BSR 2 REL	7 5 JSR 2 DIR	8 6 JSR 3 EXT	9 7 JSR 3 IX2	8 6 JSR 2 IX1	7 5 JSR 1 IX	D 1101
	* 2 STOP 1 INH		2 2 LDX 2 IMM	4 3 LDX 2 DIR	5 4 LDX 3 EXT	6 5 LDX 3 IX2	5 4 LDX 2 IX1	4 3 LDX 1 IX	E 1110
	* 2 WAIT 1 INH	2 2 TXA 1 INH		5 4 STX 2 DIR	6 5 STX 3 EXT	7 6 STX 3 IX2	6 5 STX 2 IX1	5 4 STX 1 IX	F 1111

LEGEND